Vertex-Tracing – Interaktives Ray-Tracing durch adaptiv progressives Refinement im Objektraum

Dissertation

Zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)
vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von
Thomas Ullmann

Ilmenau
03. Mai 2003

Tag des Kolloquiums:	26.11.2003
Erstgutachter:	Prof. Dr. sc. techn. Beat Brüderlin
Zweitgutachter:	Prof. Dr. rer. nat. Bernd Fröhlich
Drittgutachter:	Prof. Dr. rer. nat. Heinrich Müller

Zusammenfassung

Die vorliegende Dissertation beschreibt ein Verfahren zur interaktiv physikalisch exakten Simulation spekularer Reflexionen sowie spekularer Brechungen in virtuellen Umgebungen. Unter dem Begriff *Vertex-Tracing* wird in dieser Arbeit ein Ansatz vorgestellt, der es durch hybrides Rendering erlaubt, traditionelles Hardware-Rendering mit globalen Beleuchtungsphänomenen zu ergänzen.

Kern des Verfahrens Vertex-Tracing bildet ein *adaptiv progressives Ray-Tracing*. Im Gegensatz zum Standard-Ray-Tracing besteht das Ziel darin, vorhandene Bildkohärenzen auszunutzen, indem nur diejenigen Pixel (Samples) berechnet werden, die für die Rekonstruktion des Finalbildes erforderlich sind. Die schrittweise Annäherung an das gewünschte Finalbild erfolgt durch Verfeinerung (Refinement) der Geometrie ausgewählter Polyeder. Diese sind Szenenobjekte, die aufgrund ihrer visuellen Charakteristik in Form spekularer Reflexionen oder Brechungen einem Vertex-Tracing unterzogen werden sollen. Ausgangspunkt der Berechnung dieser Objekte stellen ihre Objekt-Vertices dar. Sie bilden jeweils den Aufpunkt eines geschossenen Primärstrahles vom Betrachter und sind zugleich Startpunkt für eine weitere rekursive Strahlenverfolgung im Sinne des klassischen Ray-Tracing. Je nach Bedarf erfolgt das Einfügen neuer Vertices, dass eine schrittweise Verfeinerung der Objektgeometrie nach sich zieht.

Die Rekonstruktion des Finalbildes erfolgt durch bilineare Interpolation mit Hilfe von Graphik-Hardware. Ihre Nutzung gestattet nicht nur ein kombiniertes Rendering mit herkömmlichen OpenGL-Objekten, sondern erlaubt darüber hinaus eine effiziente Behandlung von Texturen im Vertex-Tracing. In diesem Zusammenhang wird ein verzögerter Textur-Lookup vorgestellt. Er verhindert ein vollständiges Sampling von Texturen, das vor allem bei hochfrequenten Texturen einen erheblichen Mehraufwand bedeuten würde.

Im Hinblick auf die Beschleunigung des Verfahrens werden ferner Aspekte einer verteilt, parallelen Berechnung untersucht beziehungsweise umgesetzt. Im Vordergrund steht dabei die Verteilung des Vertex-Tracings im Rechner-Cluster sowie eine Parallelisierung des Algorithmus auf Shared-Memory-Maschinen. Beide Ansätze zeigen, dass trotz des adaptiv progressiven Charakters des Verfahrens Vertex-Tracing ein signifikanter Speed-Up erzielbar ist.

Abstract

This dissertation presents an approach for interactive, physically exact simulation of specular reflections and specular refractions in virtual environments. The introduced approach is called *Vertex Tracing* and allows a hybrid rendering to add global illumination effects in traditional hardware rendering systems.

The core of the Vertex Tracing is an adaptive progressive ray tracing. In contrast to standard ray tracing we use image coherence to compute only pixels (samples) that are essential for the final image reconstruction. The step by step adaption towards the final image is performed by geometry refinement of chosen polyhedra. These are scene objects with visual characteristics as specular reflections or refractions and should be handled particularly for Vertex Tracing. The object vertices are the starting point of computation. First, primary rays are shot from the eye point to the object vertices and after that, like in classical ray tracing approaches, a recursive ray shooting is performed from each vertex. If needed new vertices are inserted and consequently a step by step refinement of the object geometry is done.

The reconstruction of the final image is performed by bilinear interpolation via graphics hardware. Beside the possibility of a combined rendering with OpenGL-objects, the use of graphics hardware additionally allows an efficient handling of textures. In this context, we introduce a defered texture lookup to prevent a costly sampling of high frequent textures.

In addition, this thesis considers aspects of a distributed and parallel computation to speed up Vertex Tracing. In detail we implemented a distributed Vertex Tracing for a heterogenous network as well as a parallel approach for shared memory machines. Despite the adaptive progressive characteristic of the Vertex Tracing both techniques show that a significant speed-up can be achieved.

Vorwort

Diese Arbeit entstand während meiner Tätigkeit im Geschäftsfeld Nutzfahrzeuge der Daimler-Chrysler AG in enger Zusammenarbeit mit der Daimler-Chrysler Forschung in Ulm.

Mein besonderer Dank gilt Herrn Prof. Dr. B. Brüderlin. Sein Zuspruch zum Thema und sein unermüdliches Engagement bezüglich der inhaltlichen Fragestellungen ermöglichten nicht nur eine zielstrebige Anfertigung, sondern verhalfen darüber hinaus, dem wissenschaftlichen Charakter der Arbeit Rechnung zu tragen. Hervorheben möchte ich das sehr konstruktive und vor allem persönliche Arbeitsverhältnis mit Prof. Brüderlin. Seine Aufgeschlossenheit und Offenheit war nicht zuletzt die Grundlage für viele, äußerst anregende Diskussionen, die das Thema der Arbeit stets lebendig und bedeutend wirken ließen.

Außerdem möchte ich mich recht herzlich bei Daniel Beier und Alexander Schmidt bedanken. Ihre Ideen, Ausarbeitungen und Implementationen schafften zum Teil die Basis der hier vorliegenden Arbeit. Die konzentrierte sowie zielgerichtete Arbeitsweise der Beiden trug maßgeblich dazu bei, in relativ kurzer Zeit einen funktionsfähigen Prototypen zu erstellen, welcher in Verbindung mit dem dahinter stehenden Konzept eine solide Grundlage für weiterführende Arbeiten darstellte.

Ein großer Dank ist des Weiteren an Thomas Preidel gerichtet. Mit seiner Hilfe erfuhr diese Arbeit bedeutende Erweiterungen, die dem Verfahren einen deutlichen Mehrwert vermittelten. Hervorzuheben ist an dieser Stelle insbesondere seine äußerst kollegiale Art, die trotz vieler kritischer Diskussionen zu einem sehr angenehmen sowie konstruktiven Arbeitsprozess beitrug und die Arbeit zügig zu dem jetzigen Stand brachte.

Ein herzliches „Danke schön!“ geht auch an alle Mitarbeiter im Virtual Reality Competence Center der DaimlerChrysler AG in Ulm, die mir über viele Jahr stets das Gefühl gaben, ein Teil der Abteilung zu sein. Einen besonderen Dank möchte ich an dieser Stelle Herrn Dr. K. Grebner aussprechen, der mir stets in politisch und ökonomisch schwierigen Situationen den Rücken stärkte und somit nicht unwesentlich die Umsetzung dieser Arbeit sicherstellte.

Mein Dank richtet sich außerdem an die Herren Dr. T. W. Hellmuth und B. Ott,

des Geschäftsfeldes Nutzfahrzeuge der DaimlerChrysler AG. Sie sorgten im Wesentlichen dafür, dass ich mich in der Zeit als Doktorand im Konzern vorwiegend dieser Arbeit widmen konnte. Im Alltag des Geschäftslebens wusste ich sie stets als Schutzschild sowie als Vermittler auf meiner Seite.

Sehr dankbar bin ich darüber hinaus Herrn Dr. A. Rößler, der mir genügend Freiraum für die Fertigstellung der Arbeit bei IC:IDO zur Verfügung stellte. Äußerst hilfreich waren zudem seine kritischen Anmerkungen bei der Durchsicht dieser Arbeit, die für den nötigen Feinschliff sorgten.

Ein ganz besonders herzlicher Dank geht an meine liebe Freundin Corinna Saupe, die mir die lange und zum Teil beschwerliche Zeit der Bearbeitung stets mit den angenehmen Seiten des Lebens versüßte. Sie war und ist für mich stets die Insel im Meer, vor allem in den Situation, wenn sich die Arbeit im Chaos verliert.

Besonders hervorheben möchte ich schließlich meine Eltern, denen ich meinen Entwicklungsweg zu verdanken haben. Meiner Mutter danke ich vor allem für ihre ununterbrochene Fürsorge, die mir stets das Gefühl gib, etwas besonderes zu sein. Meinem Vater, der unter großem Bedauern während der Bearbeitung verstarb, werde ich ewig für seine mitgegebenen Lebenserfahrungen verbunden sein. Ihm habe ich einen Großteil meiner Motivation für diese Arbeit zu verdanken und bin mir sicher, dass er ein wenig Stolz auf mich wäre.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

„All unser Wissen gründet sich auf Wahrnehmung.“

Leonardo da Vinci

Die Verschmelzung von Illusion und Realität ist und bleibt eine der dominierenden Herausforderungen in der Computergraphik. Dennoch, synthetisch generierte Darstellungen erfuhren vor allem in jüngster Vergangenheit einen noch nicht erreichten Realismus, der sich schon bald einem direkten Vergleich mit realen Szenen zu stellen vermag. Unter dem Begriff *High-End Visualisierung* bekannt, finden immer mehr physikalische Komponenten Einfluss auf das Erscheinungsbild einer erzeugten, virtuellen Welt und lassen sie Schritt für Schritt „lebendiger“ wirken.

Ein Siegeszug, der in diesem Zusammenhang seines Gleichen sucht, fand in den letzten Jahren insbesondere auf dem Gebiet der *interaktiven* Computergraphik statt. Der Einsatz modernster Hardware verleiht hier der Computergraphik nicht nur intuitive Interaktionsmöglichkeiten, sondern verbindet diese gleichsam mit verblüffend, realistisch wirkenden Visualisierungen. Getrieben durch multimediale Anwendungen wie in der Computerspieleindustrie, ist vor allem im Bereich der visuellen Qualität, eine rasante Fortentwicklung zu verzeichnen, die nach und nach sogar zum Motor für professionelle Anwendungen mutieren könnte.

Doch gerade in der Bezeichnung „realistisch wirkende Visualisierung“ steckt der eigentliche Knackpunkt. Darstellungen wie man sie in der Spieleindustrie findet, stellen oft eine elegante jedoch lediglich eine *plausible* Simulation von Lichtinteraktionen dar und werden mit Hilfe einer Vielzahl zum Teil stark physikalischer Approximationen erzielt. Approximationen, die beispielsweise zur Visualisierung von Spieleszenarios einen hinreichenden Realitätseindruck erzeugen und im Sinne eines Immersionsgewinns ihren Zweck erfüllen.

Die Evaluierung von industriellen Prototypen durch *Virtual Reality* (VR) verlangt dagegen oft den Einsatz von exakt physikalischen Modellen. Hier steht vielmehr die Simulation eines Produktes im Vordergrund, die das tatsächliche Verhalten in der Realität nachempfindet und dadurch eine Aussage über Charakteristiken eines Prototypen zulässt.

Virtual Reality gilt darüber hinaus als Synonym für Interaktion und besitzt den Anspruch eines intuitiven Agierens in der virtuellen Welt. Somit stellt die Implementierung von physikalisch exakten Modellen unter der Gewährleistung von Interaktion stets eine Gradwanderung dar, die vor allem in VR verstärktes Feingefühl erfordert beziehungsweise einen engeren Fokus auf das Anwendungsfeld verlangt.

Die physikalisch exakte Berechnung *spekularer Reflexionen*[1] beziehungsweise *spekularer Refraktionen*[2] [Gla95, WW92, FvDFH90] ist Inhalt dieser Arbeit, wobei nicht die Simulation als solches, sondern insbesondere die Prämisse der Interaktion den Schwerpunkt bildet und eine Herausforderung auf dem Gebiet der Computergraphik darstellt.

1.1 Visualisierung und Anwendungsaspekte

Das Generieren von synthetischen Abbildungen erfolgt durch sogenannte *Rendering*-Algorithmen, die die Simulation von Licht und seiner Interaktion mit der Umgebung beschreiben. Die Umgebung wiederum wird durch ihre virtuellen Objekte, Lichtquellen und der Kamera charakterisiert. Ziel ist es beim klassischen Rendering, die Verteilung des Lichtes so realistisch wie möglich nachzuempfinden und dabei den Echtzeitanforderungen gerecht zu werden. Die Entwicklung derartiger Rendering-Algorithmen ist dabei stark von der gezielten Anwendung abhängig und führt auch hier oftmals zum Spagat zwischen Qualität und Quantität.

Um jeder Anforderung zu genügen, existiert in der Computergraphik ein breites Spektrum an Rendering-Verfahren, welche von stark approximativ bis physikalisch korrekt reichen. Renderer mit hohen Interaktionsraten basieren in der Regel auf standardisierten Graphikbibliotheken und schöpfen ein hohes Potential an Hardware aus. Die hohen *Update-Raten* (*Frame-Raten*) werden jedoch durch starke Approximationen erkauft. So basieren die Hardware-Renderer meist auf *lokalen Beleuchtungsmodellen*, in denen jedes Objekt als alleiniges in der Szenen betrachtet wird und somit der Beleuchtungseindruck ausschließlich von der Lichtquelle, der Objektoberfläche und der Betrachtungsrichtung abhängig ist. Effekte wie Schatten oder Reflexionen, die aus der Interaktion zwischen den Szenenobjekten resultieren, sind in diesem Modell nicht impliziert, sondern erfordern eine zusätzliche Behandlung.

Algorithmen, die sich stärker an physikalisch korrekten Modellen anlehnen, bilden die Welt der *globalen Beleuchtungsmodelle*. Unter dem Begriff *Global Illumination* existiert eine Vielzahl von Verfahren, die unter Betrachtung von Lichtinteraktionen zwischen den Szenenobjekten die Schaffung von höchstmöglichem Realismus (Photorealismus) verfolgen, jedoch ihren Tribut durch hohen Kostenaufwand zahlen. Als typische Vertreter, wenn auch nur durch ihre partiellen Lösungen gekennzeichnet, können *Ray-Tracing* [Whi80, Gla89, Shi00] und *Radiosity* [WW92] angesehen werden. Während Ray-Tracing auf der Verfolgung spekularer Reflexionen basiert, bietet Radiosity ein Modell

[1] Auch als perfekte Reflexion (Spiegelreflexion) ohne Streuung bezeichnet.
[2] Auch als perfekte Brechung ohne Streuung bezeichnet.

zur Simulation *perfekt diffuser*[3] Reflexionen [Gla95].

Der Anwendungsschwerpunkt dieser Arbeit liegt in der Simulation spiegelnder, sprich spekularer Reflexionen. Die Ursache dafür ergibt sich aus den Anforderungen der Automobilindustrie, Spiegelungen im Interieur von Fahrzeugen mit Hilfe von Virtual Reality zu evaluieren. Hier spielt der Aspekt der Ergonomie eine wesentliche Rolle. Insbesondere auf Instrumentenverglasungen oder Fahrzeugscheiben treten Blendeffekte auf, die störenden Einfluss auf die Sicherheit des Fahrzeugführers nehmen können.

Die Blendwirkung bezüglich des Fahrers hängt im Wesentlichen von zwei Faktoren ab:

1. Von der Strahlungsdichte (*Radiance*) des reflektierten Lichtes entsprechend seiner Interaktion mit der Objektoberfläche und

2. von der geometrischen Form des spiegelnden Objektes, dem Reflektor.

Während der erste Teil durch komplexere Zusammenhängen aus dem Bereich der klassischen Elektrodynamik beziehungsweise Photometrie[4] [BW89] resultiert, kann der zweite anhand einfacherer Gesetze aus der geometrischen Optik[5] beschrieben werden. Unter Umständen könnte man somit, allein durch die geeignete Wahl der Reflektorgeometrie, ein nahezu blendfreies Ablesen von Cockpitinstrumenten erzielen, ohne dabei das photometrische Reflexionsverhalten der Materialoberfläche, im Speziellen zu betrachten.

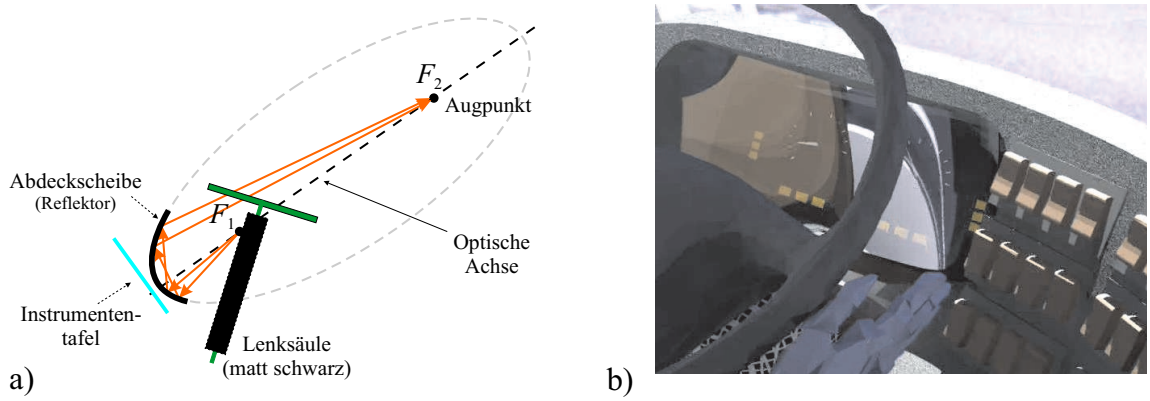


Abbildung 1.1: *a) Anwendung der geometrischen Optik am Beispiel einer Ellipse (Ellipsoid im Raum) im LKW; b) Reflexionsberechnung der Abdeckscheibe mit* RADIANCE *[War94], Quelle [AB99]*

Wie beispielsweise von den optischen Eigenschaften einer Ellipse bekannt, verlaufen alle Strahlen, die durch einen Brennpunkt F_1 gelangen nach ihrer Reflexion durch den

[3] Eintreffende Strahlung wird in alle Richtungen gleichmäßig reflektiert.

[4] Die Photometrie stellt die Spezialisierung der Radiometrie bezüglich des menschlichen Sehsinnes dar [Gla95].

[5] Anwendung findet hier unter anderem das Reflexionsgesetz, Einfallswinkel θ_i ist gleich Ausfallswinkel θ_r ($\theta_i = \theta_r$) beziehungsweise das Gesetz über die Reflexion an konkaven und konvexen Spiegeln.

Brennpunkt F_2. Es sei angenommen, der Reflektor (Abdeckscheibe eines Instrumentes) ist ein Ausschnitt aus der Mantelfläche eines Ellipsoiden, F_1 liegt auf einem schwach diffus reflektierenden Objekt (zum Beispiel Lenksäule) und F_2 befindet sich im Augpunkt des Fahrers [AB99] (Abbildung 1.1a). In diesem Fall gelangt nur der Anteil an Strahlungsenergie in das Auge des Fahrers, der vom diffus reflektierenden Objekt ausgeht und nach Abschwächung der spekularen Reflexion übrig bleibt. Durch die Fokussierung der Strahlung erreicht man hier ein „kontrollierbares“ Reflexionsverhalten des Reflektors, das den Einfluss des Reflexionskoeffizienten der Materialoberfläche an Relevanz verlieren lässt.

Der Flächenausschnitt eines Ellipsoiden oder verwandter Flächen zweiter Ordnung (Paraboloid, Hyperboloid) stellen jedoch nur eine idealisierte Form der Reflektorgeometrie dar, da der Augpunkt des Fahrers in seiner Position variiert. Somit finden heute modifizierte Modelle vor allem im PKW ihren Einsatz, wobei die Anpassung an Designrichtlinien ein zusätzliches Kriterium darstellt. Im Bezug auf den LKW treten hier jedoch verstärkt Probleme auf. Aufgrund des ungünstigeren Neigungswinkels des Kombiinstrumentes, resultierend aus der erhöhten Sitzposition des Fahrers, gestaltet sich die Suche nach einer akzeptablen Lösung weitaus schwieriger.

Eine Untersuchung im Vorfeld einer physikalischen Prototypenentwicklung mit Hilfe des Werkzeuges VR liegt dabei nahe. Im Gegensatz zu einer statischen Visualisierung von spekularen Reflexionen (Abbildung 1.1b), bietet sich hier die Möglichkeit einer *on-the-fly* Parametermodifikation und einer anschließenden Evaluierung in Echtzeit.

Der Schwerpunkt dieser Arbeit liegt nicht auf der physikalisch korrekten Simulation von Strahlungsphänomenen im Sinne der Radiometrie oder Photometrie. Vielmehr steht die Schaffung einer Methode zur Lösung der dargestellten Problematik im Vordergrund, die insbesondere auf der geometrischen Optik basiert. Das heißt, die *Real-Time*-Simulation spekularer Reflexionen bildet den Schwerpunkt dieser Arbeit. Folgerichtig wurde als *Basisalgorithmus*, für das in dieser Arbeit vorgestellte Verfahren, das Ray-Tracing gewählt. Als Vertreter der Global-Illumination Verfahren ist es prädestiniert für die Berechnung spekularer Reflexionen sowie Refraktionen und bietet durch seinen schlanken Algorithmus eine gute Ausgangsposition weiterführender Arbeiten. Betrachtungen, die darüber hinaus zur Wahl des Ray-Tracings führten, werden in Abschnitt 1.2 beziehungsweise Kapitel 2 näher erläutert.

1.2 Motivation

Wie in Abschnitt 1.1 bereits herausgestellt, liegt der Kern dieser Arbeit in der interaktiven Simulation spekularer Reflexionen beziehungsweise Refraktionen. Als Quasi-Standard für derartige Berechnungen etablierte sich in der Computergraphik das Ray-Tracing, das auch in dieser Arbeit die Basis für ein Sampling zur Verfügung stellt. Die Vorgehensweise beim Ray-Tracing besteht darin, durch jedes Pixel der Bildebene einen Strahl vom Augpunkt des Betrachters in den Objektraum zu entsenden und den Anteil an Strahlung aufzusammeln, der in Umkehrrichtung dazu in das Auge des Betrachters fällt. Die entsandten Strahlen werden dabei mit den Objekten der virtuellen Szene auf Kollision getestet. Jeder

Schnittpunkt liefert entsprechend der vorliegenden Objekteigenschaft einen Beitrag durch die von ihm ausgehende Strahlung.

Ray-Tracing ist vor allem durch seine Fähigkeit der Generierung photorealistischer Bilder bekannt, die zum Teil auch als *super-real*[6] bezeichnet werden. Gleichzeitig verbindet man mit dem Begriff jedoch auch lange Rendering-Zyklen, die aus den hohen Kosten durch „Millionen“ von Strahl-Schnittpunkttests resultieren. Seit der Vorstellung des Verfahrens vor etwa 20 Jahren durch WHITTED [Whi80] gilt Ray-Tracing in der Regel noch immer als *Off-Line* Rendering-Methode.

Die Tatsache, dass etwa 95% der Kosten auf den Strahl-Schnittpunkttest entfallen [Whi80] und die Suche eines Schnittpunktes durch ein klassisches Suchproblem auf Baumstrukturen [OW93] optimierbar ist, führte schon in der Vergangenheit zu der Schlussfolgerung, dass Ray-Tracing irgendwann Scan-line-basiertes Hardware-Rendering an Geschwindigkeit übertreffen könnte. Gegeben durch die durchschnittlich *sub-lineare* Zeitkomplexität eines Schnittpunkttestes[7] und des in der Regel *linearen* Charakters des Hardware-Renderings muss es offenbar einen sogenannten *Break-Even-Point* bezüglich der Szenenkomplexität gegeben. Abbildung 1.2 illustriert diesen Sachverhalt.

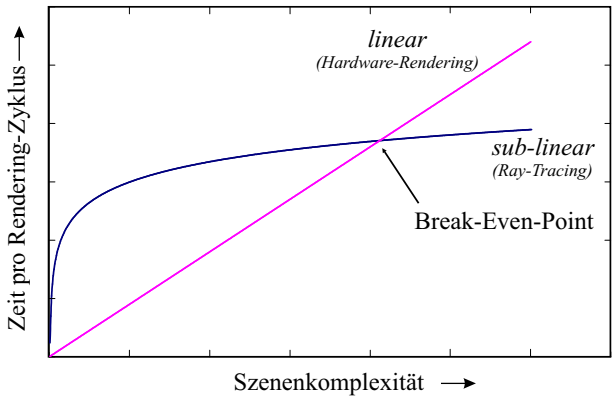


Abbildung 1.2: *Schematische Darstellung des Break-Even-Points zwischen Ray-Tracing und polygon-basiertem Hardware-Rendering*

Das Erreichen des Break-Even-Points stellt eines der Schwerpunkte heutiger, high-end Ray-Tracer dar. So wurden beispielsweise in den Arbeiten von [PMSS99] oder [WBWS01] bereits interaktive Frame-Raten erreicht, indem insbesondere hoch optimierte Kollisionstests in paralleler Umgebung ihren Einsatz fanden. Die Überwindung des Break-Even-Points könnte also nicht nur das Ray-Tracing für interaktives Rendering stärker etablieren, sondern herkömmliches Hardware-Rendering sogar um ein vielfaches an Leistung bei entsprechend hohem Datenvolumen übertreffen. Zutreffend ist diese Tatsache jedoch nur,

[6] Der Begriff 'super-real' resultiert aus der Generierung unbegrenzt scharfer, rekursiver Reflexionen.

[7] Die Zeitkomplexität eines Strahlen-Schnittpunkttestes liegt bei Verfahren, die auf den *Worst-Case* optimiert sind, bei $O(\log n)$ und bei heuristischen Verfahren (*Average Case* Verfahren) zwischen $O(1)$ und $O(n)$. Nähere Erläuterungen dazu findet man unter anderem in [SKM98, Hav00].

solange man von einer Datenaufbereitung beim Hardware-Rendering absieht. Auch das Hardware-Rendering kann bei gezielter Datenvorverarbeitung in sub-linearer Zeitkomplexität erfolgen, wie es beispielsweise die Arbeiten von [WFP+01] belegen.

Die Optimierung des Strahl-Schnittpunkttestes beim Ray-Tracing kann neben einer Parallelisierung vor allem durch Kohärenzeigenschaften der Szene (*Object-Space Kohärenz*) erzielt werden. Das heißt, die Kohärenz, gegeben durch Nachbarschaftsbeziehungen von Objekten oder innerhalb eines Objektes, wird mittels geeigneter Datenstrukturen erfasst und ermöglicht somit die Ausnutzung dergleichen. Es entstehen hierarchische Datenstrukturen, die die sub-lineare Zeitkomplexität des Algorithmus begründen [GP95].

Neben der eigentlichen Zeitkomplexität trägt außerdem ein konstanter Faktor k nicht unwesentlich zum Laufzeitverhalten des Ray-Tracing bei. Große k verschieben den Break-Even-Point nicht nur deutlich nach rechts sondern schrauben die Schwelle des Real-time Ray-Tracings auch bei kleiner Szenenkomplexität empfindlich nach oben. Ein interaktives Rendering auf Single-Prozessor-Systemen ist in diesem Fall kaum realisierbar.

Doch neben Hardware-spezifischen Optimierungsmethoden[8] bietet die Ausnutzung weiterer Kohärenzeigenschaften ein hohes Potential zur Minimierung von k und der entsprechenden Beschleunigung. Kohärenzen, die über die bereits erwähnte Object-Space Kohärenz hinausreichen, sind im Wesentlichen[9]:

- Die Kohärenz zwischen benachbarten Pixeln, die sogenannte *Image-Space* Kohärenz. Die Beleuchtungswerte nahe beieinander liegender Pixel, die sich aus dem Rendering der Szene und ihrer Projektion in den Bildraum ergeben, variieren überwiegend langsam.

- Die Kohärenz zwischen aufeinander folgender Frames, die sogenannte *Temporale* Kohärenz oder *Frame-To-Frame* Kohärenz. Bei geringer Differenz der Betrachterposition beziehungsweise -richtung zwischen aufeinander folgenden Frames können die bereits berechneten Bildinformationen zur Erstellung des neuen Frames herangezogen werden.

Am Beispiel der Image-Space Kohärenz kann man sich nun folgenden Sachverhalt verdeutlichen: Die Zeitkomplexität des Ray-Tracings skaliert linear mit der Menge der berechneten Pixel (*Samples*) im Bildraum, sie bestimmt unter anderem die Größe von k. Die Reduzierung der Pixelanzahl hat also einen expliziten Einfluss auf k und damit auf das Laufzeitverhalten des Verfahrens. Eine dramatische Reduzierung der Abtastung (Sampling) lässt sich hier durch die Ausnutzung der Image-Space Kohärenz erreichen. Aufgrund einer nur *allmählich* variierenden Radiance-Funktion über dem Bildraum, kann die Eigenschaft eines Pixels zur Bestimmung seines Nachbarn herangezogen werden.

[8] Denkbar sind hier die Ausnutzung Superskalarer Prozessorarchitekturen, der Einsatz von SIMD (Single Instruction Multiply Data) Einheiten oder optimierter Caching- bzw. Pre-Fetching-Verfahren.

[9] Zu bemerken ist hier, dass es zwischen den Kohärenzeigenschaften keine klare Trennung gibt. Es existieren zum Teil konkretere Bezeichnungen wie beispielsweise *Ray*-Kohärenz oder *Depth*-Kohärenz [GP95], die sich jedoch im Allgemeinen auf die Basisklassen zurückführen lassen. Ihre Klassifikation ist zudem abhängig von der Sichtweise ihrer Beurteilung.

Folgerichtig ist das Primärziel dieser Arbeit die Ausnutzung von Image-Space Kohärenz im Hinblick auf eine Beschleunigung des Ray-Tracings. Es soll gezeigt werden, dass sich durch ein *adaptiv progressives Sampling* (↗ Kapitel 3) die Menge der erforderlichen Samples zur Generierung eines Bildes je nach Szenencharakteristik bis zu Faktor 10 und mehr reduzieren lässt. Außerdem soll aufgrund der Tatsache, dass eine virtuelle Szene im Verhältnis nur wenige Objekte mit wahrnehmbar spekularen Reflexionseigenschaften enthält [HG83], eine weitere Reduzierung des Samplings erfolgen. Die Einführung eines *Hybrid-Renderings*, gekennzeichnet durch ein selektives Ray-Tracing spekularer Objekte in Kombination eines Hardware-Renderings aller übrigen Szenenobjekte, ermöglicht Laufzeitvorteile insbesondere bei Szenen mittlerer Komplexität und zielt speziell auf interaktive Anwendungen in VR.

1.3 Beitrag der Arbeit und Zielbeschreibung

Die Ausnutzung von Kohärenzeigenschaften ist zweifellos der Schlüssel zur Beschleunigung des Ray-Tracings. Auch das hier vorgestellte Verfahren bedient sich ihrer durch verschiedenste Methoden, mit der Schaffung eines echtzeitfähigen Ray-Tracings zum Ziel. Wie in Abschnitt 1.2 bereits herausgearbeitet wurde, liegt dabei der Schwerpunkt auf der Ausnutzung von Image-Space Kohärenz, welche durch eine im Allgemein langsam variierende Radiance-Funktion im Bildraum resultiert.

Der Kern dieser Arbeit basiert auf einem gezielten Sampling der Radiance-Funktion und verfolgt eine signifikante Reduktion von Strahl-Schnittpunkttests, die zur Generierung des Bildes erforderlich sind. Anhand der ermittelten Samples beziehungsweise Stützstellen erfolgt im Anschluss der Abtastung eine Rekonstruktion der Radiance-Funktion

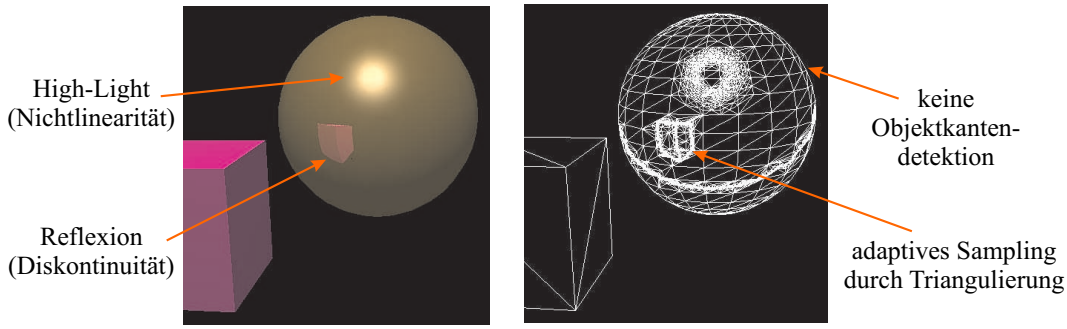


Abbildung 1.3: *Adaptiv progressives Refinement im Objektraum - Prinzip des Vertex-Tracings*

mit Hilfe bilinearer Interpolation (Gouraud-Shading). Das Sampling arbeitet dabei *adaptiv progressiv*. Das heißt, enthaltene *Nichtlinearitäten* beziehungsweise *Diskontinuitäten*[10]

[10]Nichtlinearitäten der Radiance-Funktion treten insbesondere in Glanzlichtern, den sogenannten High-Lights, auf Objektoberflächen auf. Diskontinuitäten hingegen findet man beispielsweise an Reflexions-

der Radiance-Funktion werden Schritt für Schritt angenähert und entsprechend genauer approximiert. Abbildung 1.3 verdeutlicht dieses Prinzip. Am Beispiel der Kugel ist ein adaptives Sampling der Radiance-Werte an den Stellen zu erkennen, an denen eine hohe Varianz der Radiance-Funktion auftritt (Reflexion, Glanzlicht, Schattenkanten).

Im Gegensatz zu den Arbeiten von [PLS97, NG97, ELPZ94] erfolgt jedoch die Generierung der Stützgeometrie für das Sampling im Objektraum mit anschließender Projektion in den Bildraum. Hinzu kommt, dass im Gegensatz zu [Bal99] die Stützgeometrie direkt auf der vorhandenen Geometrie der triangulierten Szenenobjekten aufsetzt und eine weitere, adaptive Triangulierung der gleichen stattfindet. Dieser Ansatz beschreibt nach aktuellem Kenntnisstand einen neuen Weg eines adaptiv progressiven Ray-Tracings und kann im Wesentlichen mit folgenden Vorteilen, die unter anderen in den Kapiteln 2 und 3 näher erläutert werden, aufwarten:

- Ein Kollisionstest aller Primärstrahlen entfällt, da die Geometrie der Szenenobjekte und damit ihrer Stützpunkte (*Vertices*) als Grundlage einer weiteren Strahlenverfolgung (Ray-Tracing) dient.
- Es bedarf keiner zusätzlichen Objektkantendetektion, da als Ausgangsbasis einer weiteren Triangulierung die primäre Objektgeometrie herangezogen wird.
- Der qualitative Eindruck der generierten Szene ist auch bei wenigen verfügbaren Samples relativ hoch, da die Objekte in ihrer ursprünglichen Geometrie erhalten bleiben und unmittelbar darstellbar sind. Die Darstellung eines *„schnellen Previews“* wird dadurch garantiert.
- Eine Kombination von Hardware-basiertem Rendering und Ray-Tracing wird direkt möglich, da die Darstellung von Objekten, die mittels Ray-Tracing generiert wurden, ebenfalls durch Hardware-Rendering erfolgt.
- Das Laufzeitverhalten skaliert gegenüber dem Ray-Tracing sub-linear bezüglich der Anzahl der Pixel in der Bildebene. Das Ray-Tracing besitzt hier linearen Charakter.

Erstere Eigenschaft verleiht unterdessen dem Verfahren seinen Namen: Die Vertices eines Objektes stellen die Ausgangsbasis jeder weiteren Berechnung dar. Das heißt, sie dienen als Startpunkt für eine Strahlenverfolgung im Sinne des Ray-Tracings und ermöglichen einen Verzicht auf die Schnittpunktbestimmung von Primärstrahlen. Die Objekt-Vertices sind zentraler Bestandteil des Verfahrens und gaben ihm den Namen *Vertex-Tracing*.

Doch das Prinzip Vertex-Tracing soll nicht allein zur Beschleunigung des Ray-Tracings beitragen. Vielmehr soll auch hier dem Potential eines verteilten, parallelen Ray-Tracings Rechnung getragen werden, dass durch die Möglichkeit eines voneinander unabhängigen Samplings besticht. Die adaptiv progressive Charakteristik des Vertex-Tracings bringt jedoch zusätzliche Abhängigkeiten im Sampling-Prozess mit sich, dass eine Parallelisierung erheblich erschwert. Kapitel 4 beschreibt verschiedene Wege zur Lösung dieses Problems, sei es durch verteiltes Rechnen im *Cluster* oder mittels *Multi-Threading*.

sowie Schattenkanten, an denen ein Sprung (Unstetigkeit) der Strahlungsdichte auftritt. Eine detaillierte Betrachtung findet unter anderem in den Abschnitten 6.1.1 und 6.1.2 statt.

1.4 Strukturierung der Arbeit

Der Fokus dieser Arbeit liegt auf der ausführlichen Beschreibung des hier vorgestellten Vertex-Tracings. Das heißt, alle Ausführungen insbesondere im Vorfeld der eigentlichen Bearbeitung des Kernthemas beabsichtigen eine sukzessive Heranführung an die Beschreibung des Verfahrens. Dabei werden neben den Aspekten der Motivation und des Anwendungshintergrundes vor allem bereits existierende Arbeiten mit themenverwandten Inhalten vorgestellt beziehungsweise untersucht und mit dem eigenen Verfahren in Beziehung gesetzt.

Ziel soll es nicht sein, eine vollständige Aufstellung der Grundlagen der Computergraphik insbesondere der Global-Illumination oder des Ray-Tracings durchzuführen. Diesbezüglich wird auf angegebene Referenzen im Text verwiesen, auf die der Leser bei Bedarf zurückgreifen kann. An dieser Stelle soll gesondert auf die Literatur von [FvDFH90, WW92, Gla95] referenziert werden, da sie einen umfassenden Überblick der hier tangierten Themen bietet. Die Gliederung der Arbeit gestaltet sich im Einzelnen wie folgt:

Kapitel 1 gibt eine zielgerichtete Einführung in die Thematik, indem aus den Anwendungshintergründen die Anforderungen des Verfahrens abgeleitet werden. Ferner erfolgt für den Leser eine Motivation des hier beschrittenen Weges zur Problemlösung mit anschließender Ausarbeitung des Mehrwertes und der Zielstellung dieser Arbeit.

Kapitel 2 beschreibt eine grobe Einordnung der Arbeit in die Welt des Renderings. Darüber hinaus wird anhand existierender, themenverwandter Verfahren beziehungsweise Lösungensvarianten der hier vorliegende Ansatz diskutiert. Es erfolgt eine umfangreiche *State-of-the-Art* Übersicht.

Kapitel 3 beginnt mit der eigentlichen Beschreibung des Verfahrens Vertex-Tracing. Hierzu liegt der Schwerpunkt auf der Darstellung des adaptiven Samplings, des Refinements[11] und der anschließenden Rekonstruktion der Radiance-Funktion.

Kapitel 4 setzt die Betrachtung des Vertex-Tracings fort, erläutert das Verfahren hier jedoch unter dem Gesichtspunkt einer verteilten, parallelen Berechnung von Sample-Werten. Das Thema verteiltes, paralleles Vertex-Tracing wird dabei unterschieden zwischen einer verteilten Berechnung mittels PVM[12] im Rechner-Cluster und einer Parallelisierung mit Hilfe von Multi-Threading auf Shared-Memory Rechnerarchitekturen.

Kapitel 5 evaluiert das Verhalten des Vertex-Tracings in verschiedenen Testszenarien und gibt eine Einschätzung bezüglich seiner Praxisrelevanz.

Kapitel 6 gibt eine Bewertung des Systems in Bezug auf seine Korrektheit und Laufzeitverhalten gegenüber dem Ray-Tracing wieder. Diese Bewertung erfolgt anhand von Analysen, die verschiedene Eigenschaften des Vertex-Tracings näher untersuchen.

In *Kapitel 7* wird die Arbeit mit einer Zusammenfassung und dem Ausblick auf weiterführende Arbeiten abgeschlossen.

[11] Sich an eine Schranke annähernder Verfeinerungsprozess.

[12] PVM steht für *Parallel Virtual Machine*, ein API zur Programmierung verteilter Anwendungen.

Kapitel 2

Themenverwandte, bisherige Arbeiten

Inhalt des vorliegenden Kapitels ist die Darstellung existierender Verfahren, die das Thema dieser Arbeit mehr oder weniger tangieren. Der Schwerpunkt liegt dabei auf Rendering-Methoden, die sich explizit mit der Simulation spekularer Reflexionen befassen oder zumindest einen plausiblen Eindruck derartiger Effekte vermitteln. Alle aufgeführten Verfahren dienen im weiteren Sinne der Beschleunigung des Renderings und verdeutlichen insbesondere die jeweils unternommene Gradwanderung zwischen Qualität und Quantität.

2.1 Rendering-Verfahren

Ziel des Rendering-Prozesses ist es im Allgemeinen, unter der Mitwirkung von Vorstellungskraft, Signalverarbeitung und Physik realistisch wirkende Bilder zu generieren. Der Prozess betrachtet dabei das Licht, die Kamera und eine Menge an 3D-Objekten in ihrer Interaktion zueinander und bildet damit die Charakteristiken der realen Welt in geeigneter Form nach.

In der Literatur existiert eine große Anzahl verschiedener Rendering-Verfahren [WW92], die, jedes auf seine Weise, den Anforderungen der praktischen Anwendung individuell gerecht werden. Die theoretische Grundlage aller Verfahren setzte Kajiya mit der *Rendering-Equation* [Kaj86]. Die von einem Punkt $\vec{x}$ ausgehende Radiance L in Richtung ω^{out} ist definiert mit:

$$L(\vec{x}, \omega^{out}) \;=\; L^e(\vec{x}, \omega^{out}) + \int_{\Omega^{in}} f_r(\vec{x}, \omega^{in} \rightarrow \omega^{out}) \cdot L(h(\vec{x}, -\omega^{in}), \omega^{in}) \cdot cos\theta d\omega^{in} \quad (2.1)$$

wobei L^e die emittierte Radiance von $\vec{x}$ in Richtung ω^{out}, f_r die *Bidirectional Reflectance/Distribution Function* (BRDF) der Objektoberfläche, $h(\vec{x}, -\omega^{in})$ die Sichtbarkeit von $\vec{x}$ in Richtung $-\omega^{in}$, $L(h(\vec{x}, -\omega^{in}), \omega^{in})$ die einfallende Strahlung aus Richtung ω^{in}, $cos\theta$ den Winkel zwischen ω^{in} und der Normalen in $\vec{x}$, $d\omega^{in}$ den Raumwinkel (Solid Angle) sowie Ω^{in} die Einheitskugel um $\vec{x}$ darstellt. Die Abhängigkeit der ausfallenden von der einfallenden

Radiance gibt der Gleichung einen impliziten Charakter, was eine vollständige Lösung der Radiance-Equation erheblich erschwert.

Alle Rendering-Verfahren können im Allgemeinen auf die Rendering-Equation zurückgeführt werden, sie stellen verschiedene Stufen einer Approximation dergleichen dar. Lokale Beleuchtungsmodelle sind gekennzeichnet durch eine hohe Approximation. Sie repräsentieren das Integral von 2.1 meist durch vorgegebene Werte, die lediglich bezüglich des Einfallswinkels variieren. Globale Beleuchtungsmodelle lösen hingegen die implizite Gleichung, indem die einfallende Radiance $L(h(\vec{x}, -\omega^{in}), \omega^{in})$ in Betracht gezogen wird. Sie behandeln zudem das Sichtbarkeitsproblem $h(\vec{x}, -\omega^{in})$. Zur Lösung des Integrals über der Domäne Ω^{in} werden wiederum verschiedene Ansätze benutzt. Eine Möglichkeit liegt in der Nutzung der *Finiten Elemente* Methode, wie beispielsweise beim Radiosity der Fall [CW93]. Verfahren wie *Monte-Carlo* Ray-Tracing oder *Path-Tracing* nutzen dagegen numerische Quadratur durch diskretes Sampling [Shi00]. Das Sampling erfolgt dabei durch *Stratifikation*[1] sowie durch *Importance Sampling* [Gla95], um eine effiziente Abtastung der Eingangsfunktion zu provozieren. Wird die Domäne Ω^{in} aus 2.1 durch Stratifikation in Regionen R_i mit

$$\Omega^{in} = \bigcup_{i=1}^{n} R_i \tag{2.2}$$

unterteilt, folgt nach Gleichung 2.1

$$L(\vec{x}, \omega^{out}) = L^e(\vec{x}, \omega^{out}) + \sum_{i=1}^{n} \int_{R_i} f_r(\vec{x}, \omega^{in} \rightarrow \omega^{out}) \cdot L(h(\vec{x}, -\omega^{in}), \omega^{in}) \cdot cos\theta d\omega^{in}, \tag{2.3}$$

so dass nur noch für jedes Stratum R_i die Lösung des Integrals erfolgen muss. Das Standard-Ray-Tracing, das im Rahmen dieser Arbeit als Sampling-Grundlage dient, leitet sich nun aus 2.3 durch Reduzierung eines Stratum R_i auf Punktgröße ab. Ein Lösen des Integrals über R_i entfällt. Hinzu kommt, dass nur die spekulare Gegenrichtung von ω^{out} als einziges Sample dient, um lediglich den spekularen Reflexions- als auch Refraktionsanteil zu bestimmen. Das Standard-Ray-Tracing kann damit nur direkt gerichtete Radiance verarbeiten. Alle diffusen Anteile bleiben unbehandelt.

Die Eliminierung des Integrals im Falle des Standard-Ray-Tracings bringt hinsichtlich der Lösung der impliziten Rendering-Equation enorme Laufzeitvorteile mit sich. Dennoch, die Klärung des Sichtbarkeitsproblems bleibt. Hier werden circa 95 % der Gesamtlaufzeit eines Standard-Ray-Tracers verbraucht. Das Ray-Tracing hat sich dennoch als Standardalgorithmus für spekulare Lichtinteraktionen etabliert. Im Gegensatz zu anderen Rendering-Verfahren, die auf Grundlage lokaler Beleuchtungsmodelle derartige Effekte generieren, überzeugt das Ray-Tracing durch seine Korrektheit sowie einfachen und vor allem effizienten Algorithmus bezüglich hoher Szenenkomplexitäten. Inwieweit andere Rendering-Verfahren Potential zur Abbildung spekularer Reflexionen beziehungsweise Refraktionen besitzen, soll in den nachfolgenden Abschnitten näher untersucht werden.

[1] Die Domäne wird in nicht-überlappende Regionen aufgeteilt, in denen eine bessere Auswertung der Funktion erfolgt kann.

2.2 Hardware-basiertes Rendering spekularer Reflexionen

Die Zunahme an Realismus beim heutigen interaktiven Rendering begründet sich insbesondere durch die extensive Ausnutzung von Hardware. Mit der Integration weiterer Stufen der traditionellen Rendering-Pipeline und der Einbettung neuer, vollständig programmierbarer Funktionseinheiten wie dem Vertex-Shader [LKM01] beziehungsweise Pixel-Shader [Spi00a, Spi00b] kann den physikalischen Gegebenheiten der realen Welt mehr und mehr Rechnung getragen werden.

Im Weiteren soll nun ein Überblick über verschiedene State-of-the-Art Verfahren erfolgen, die interaktives Rendering spekularer Reflexionen zum Teil durch die Ausnutzung von Hardware erlauben.

2.2.1 Environment-Mapping

Aufbauend auf dem Textur-Mapping gilt das Environment-Mapping [WW92] als eine spezialisierte Methode, die den Realismus von interaktivem Rendering weiter erhöhen soll. Traditionelles Environment-Mapping, erstmals beschrieben von Blinn und Newell [BN76], verfolgt das Ziel, Reflexionen einer perfekt spiegelnden Objektoberfläche schnell zu erfassen und auf dem Objekt entsprechend darzustellen. Die Idee dahinter gestaltet sich wie folgt: Im Falle, dass ein reflektierendes Objekt klein bezüglich seiner Distanz zur Umgebung ist, kann die einfallende Radiance allein mit der Richtung des reflektierten Strahls in Abhängigkeit gesetzt werden. Das heißt, der Aufpunkt des reflektierten Strahls auf der Objektoberfläche, sein Ursprung, ist vernachlässigbar. In der zwei-dimensionalen Environment-Map erfolgt die Speicherung der einfallenden Radiance bezüglich dem reflektierenden Objekt, indem ein Sampling der Umgebung in einem Pre-Compute-Prozess stattfindet.

Zur Parametrisierung der Mapping-Funktion finden heute vorwiegend *Sphere-Maps* [WNDS99] beziehungsweise *Cube-Maps* [Gre86, nC99] ihren Einsatz. Das *Spherical-Environment-Mapping* (Sphere-Maps) zählt zu den blickpunktabhängigen (view-dependent) Verfahren, da in der Regel die Environment-Map nach Veränderung des Blickpunktes reproduziert werden muss. Eine Cube-Map ist dagegen blickpunktunabhängig (view-independent). Ähnlich zum *Parabolic-Environment-Mapping* [HS98] wird hier das Sampling der Umgebung einmalig in einem Pre-Prozess vorgenommen und kann bei Blickrichtungsänderung direkt wiederverwendet werden. Zutreffend ist die Möglichkeit einer wiederverwendbaren Environment-Map jedoch nur bei statischen Szenen. Im Falle von Dynamik muss hier zumindest ein Teil der Environment-Map on-the-fly neu generiert werden, dass entsprechende Laufzeiteinbußen verursacht

Ferner wird das Environment-Mapping durch signifikante Schwächen hinsichtlich der physikalischen Korrektheit berechneter spekularer Reflexionen charakterisiert, welche das Verfahren für die Anforderungen dieser Arbeit ausscheiden ließen. Diese fundamentale Restriktion ergibt sich aus der Abbildung vom $\mathbb{R}^3$ in den $\mathbb{R}^2$, indem eine Projektion der Orientierung des reflektierten Strahls in eine zwei-dimensionale Environment-Map erfolgt.

Ergebnis dieser Projektion sind nur noch zwei verbleibende Freiheitsgrade zur Adressierung der Environment-Map, die vollkommen unabhängig vom Aufpunkt des reflektierenden Strahls ist. Es resultiert eine Positionsunabhängigkeit bezüglich des reflektierenden Objektes, so dass zwei Punkte eines Objektes, mit identischer Orientierung des reflektierten Strahls, das gleiche Environment reflektieren. Hinzu kommt, dass die Generierung der Environment-Map von einem bestimmten Punkt der Szene erfolgt und dementsprechend auch nur für diesen Punkt eine exakte, spekulare Reflektion aufweist. Die Darstellung von Interreflektionen, wie sie beispielsweise auf einem Torus zu finden sind, kann in diesem Zusammenhang ebenfalls nicht abgebildet werden.

2.2.2 Image-Based Rendering

Unter Image-Based Rendering (IBR) versteht man interaktives Rendering durch intelligentes Ausnutzen vordefinierter Bildsequenzen. Ziel ist es dabei, eine Approximation der *Plenoptischen* Funktion[2] [AB91] aus einer gegebenen Menge von Samples (Images) zu generieren und somit von einer beliebigen Position eine neue Ansicht der Szene zu rekonstruieren. IBR kann als Lösung der Rendering-Equation [Kaj86] durch Warping von existierenden Bildern gesehen werden. Es wird die Eliminierung traditioneller Objektrepresentationen (zum Beispiel Polyeder) verfolgt.

Eine Vielzahl von IBR-Verfahren basieren auf Environment-Maps, welche zur Speicherung einfallender Radiance dienen. Die Generierung der Maps erfolgt von fest vorgegebenen Betrachtungspunkten, zwischen denen sich im Anschluss der Betrachter nur diskret bewegen dabei allerdings in seiner Blickrichtung variieren kann (*Apple QuickTime VR* [Che95]). Begegnet wird dem Problem „fixer Viewpoints“ mittels Viewpoint-Interpolation durch Morphing oder durch Reprojektion anhand von Z-Werten. So erfolgt beispielsweise in [WC93] die Rekonstruktion neuer Viewpoints durch die Verwendung von Tiefeninformationen. Ähnlich dazu wird in [MB95] ein Verfahren, basierend auf einem Paar zylindrischer Projektionen mit definierter Kamerapositionen, zur Suche nach korrespondierenden Punkten zwischen verschiedenen Bildern beschrieben.

Eine Erweiterung des auf Environment-Maps basierenden IBRs zeigen [CON99] mit der Vorstellung der *Radiance-Environment-Maps*. Diese synthetischen Maps enthalten abgetastete BRDFs und ermöglichen ein interaktives Rendering komplexer Materialeigenschaften, geben jedoch nur in einem Punkt korrekte Reflexionscharakteristiken wieder. Der Grund liegt in der Speicherung von blickpunktabhängigen Effekten wie beispielsweise dem *Fresnel-Term*.

Andere IBR-Systeme umgehen die Benutzung von Environment-Maps und bedienen sich dem sogenannten *Light-Field* [LH96] beziehungsweise *Lumigraph* [GGSC96, HKSS98]. Beide zueinander ähnliche Verfahren beschreiben die Plenoptische Funktion mit Hilfe einer 4-dimensionalen Repräsentation, der *Light-Slap-Repräsentation*, unter der Voraussetzung eines leeren Raums ohne Hindernis, dem *Empty Space*. Die Vorteile dieser Methode sind,

[2]Die Plenoptische Funktion beschreibt alle Bildinformationen, die aus einer bestimmten Betrachterposition sichtbar sind. Sie wird definiert als $p = P(\Theta, \Phi, \lambda, x, y, z, t)$ mit dem Raumwinkel Θ, Φ, der Wellenlänge λ, der Position x, y, z im Raum und der Zeit t.

sowohl in ihre Unabhängigkeit von Tiefeninformationen der einbezogenen Bilder als auch in ihrer größeren Flexibilität während der Bildrekonstruktion zu sehen. Ein Nachteil besteht unter anderem darin, dass bis dato die Methode nur unter Annahme einer Szene mit ideal diffusen Objekten operiert. Diesbezüglich wurde in [LR98] die Nutzung von *Layered-Depth-Images* (LDI) [GHC97] vorgeschlagen. Unter Einbeziehung von geometrischen Informationen wie Tiefe und Normalen konnten somit auch non-diffuse Reflektoren einer synthetischen Szene dargestellt werden. Erweitert wurde diese Methode von [HLCS99] durch eine entkoppelte Speicherung von Geometrie- und Radiance-Informationen in separate Light-Fields. Das Geometrie-Light-Field unternimmt dabei eine Farbkodierung der ausgehenden beziehungsweise eingehenden Strahlenrichtung des Objektes, die als Lookup in ein zweites Light-Field oder eine Environment-Map dienen. Es ermöglicht ein Hardware-basiertes Rendering.

Die rasante Entwicklung des IBRs in den letzten Jahren spricht zweifellos für das hohe Potential des Verfahrens, unabhängig der Szenenkomplexität photorealistisches, interaktives Rendering zu ermöglichen. Dennoch, die Qualität des Renderings ist vor allem von der Menge vordefinierter Radiance-Samples abhängig und lässt das IBR trotz modernster Kompressionsverfahren schnell Gigabytes an Speicher fressen. Hinzu kommt, dass das Aufzeichnen von Bildsequenzen in der Regel dem Pre-Prozess vorbehalten ist, da hier enorm hohe Laufzeitkosten anfallen. Ein interaktives Rendering dynamischer Szenen ist dadurch nahezu unmöglich und charakterisieren das IBR zur Lösung der eigenen Problemstellung als zu unflexibel.

2.2.3 Explizite Geometriespiegelung

Eine weitere Gruppe von Algorithmen, die speziell auf die Darstellung von Spiegelreflexionen (perfekt spekulare Reflexion) abzielt, bedient sich der expliziten Transformation von Geometrie im Objektraum.

In diesem Zusammenhang wurde in [Die96, MH99] eine mittlerweile populäre Technik für interaktives Rendering von Spiegelreflexionen an planaren Flächen vorgestellt. Hier erfolgt eine Spiegelung der Szene mit Hilfe einer einfachen, affinen Abbildung (Transformation) am planaren Reflektor. Die gespiegelte Szene ist dabei nur in den Pixeln sichtbar, die vom Reflektor in der aktuellen Kameraperspektive überdeckt werden. Das dafür notwendige *Multi-Pass* Rendering für die Darstellung *eines* planaren Reflektors geschieht in folgenden Schritten:

1. Der Reflektor wird separat in den Stencil-Buffer gerendert, um die vom Reflektor überdeckten Pixel zu ermitteln.

2. Die Szene wird an der Ebene des Reflektors gespiegelt und unter Berücksichtigung des Stencil-Buffers in den Frame-Buffer gerendert. Nur die Pixel werden geschrieben, die vom Stencil-Buffer markiert sind.

3. Die ungespiegelte Szene wird über den bereits vorhandenen Frame-Buffer Inhalt geblendet und ergibt die fertige Abbildung.

Wie in [Die96] beschrieben, ist auf diese Weise auch die Generierung von Mehrfachreflexionen möglich, indem dieser Vorgang leicht modifiziert für jeden weiteren Reflektor wiederholt wird.

Ein anderes Verfahren, das sich dem erwähnten Multi-Pass Rendering bedient, jedoch interaktive Reflexionen an gekrümmten Flächen (*Curved-Surfaces*) ermöglicht, wurde in [OR98] vorgeschlagen. Ofek und Rappoport erzeugen entsprechend der gekrümmten Reflektorgeometrie *virtuelle Objekte*, indem jeder Vertex des sich reflektierenden Objektes separat am Reflektor gespiegelt wird. Um die Spiegelung auf eine affine Transformation zurückzuführen, finden *Reflection-Subdivisions* in Verbindung mit einer approximierenden Beschleunigungsmethode, den sogenannten *Explosion-Maps*, ihren Einsatz. Die Darstellung der resultierenden, virtuellen Objekte kann schließlich im Schritt zwei des oben skizzierten Multi-Pass Verfahrens mittels traditionellem Hardware-Rendering ausgeübt werden.

Trotz Ausdehnung des Stencil-Buffer-Verfahrens auf Curved-Surfaces birgt die Methode aus [OR98] zum Teil fundamentale Probleme in sich. Im Falle von konkaven Flächen verursacht jede Reflexion die Entstehung eines Brennvolumens, das darin enthaltende Objekte mitunter als chaotische Abbildungen auf dem Reflektor entstehen lässt. Auf die Spiegelung eines einzelnen Vertex bezogen, bedeutet dies die Entstehung unendlich vieler virtueller Vertices. Die Darstellung eines eindeutigen, virtuellen Polygons ist nicht definiert. Es gibt weder eindeutige Topologieinformationen noch ist die Anzahl der zu berechnenden, virtuellen Vertices deterministisch. Da jede konkave Fläche ein Brennvolumen ausbildet, kann das Verfahren von Ofek und Rappoport im engeren Sinne nur auf konvexe Objekte angewendet werden und stellt damit eine massive Limitierung dar. Im Falle konvexer Objekte erzeugt das Verfahren jedoch erstaunlich gute Approximationen, die dem visuellen Eindruck eines Ray-Tracers sehr nahe kommen und darüber hinaus in Echtzeit realisierbar sind.

2.3 Ray-Tracing

Seit der eigentlichen Vorstellung des Ray-Tracings durch Whitted in [Whi80] wurde in den letzten zwei Dekaden eine schier endlose Liste an Forschungsarbeit geleistet, die vor allem auf dem Charme des Verfahrens zurückzuführen ist. Es verbindet durch seinen rekursiven Kern einen relativ einfachen Algorithmus, der leicht zu implementieren ist und nicht zuletzt „schöne Bilder“ generiert.

Trotz der relativ starken Vereinfachungen, die im Falle des Standard-Ray-Tracings an der Rendering-Equation vorgenommen wurden (↗ Abschnitt 2.1), liegt der Hauptnachteil des Ray-Tracings immer noch in seinem hohen Berechnungsaufwand. Dieser lässt selbst bei kleinen Szenen ein interaktives Rendering zur Herausforderung werden. Viele Forschungsprojekte widmeten sich daher diesem Thema und verfolgten primär die Beschleunigung des Verfahrens. Themenfelder wie beispielsweise

- effiziente Schnittpunktbestimmung (Kollisionserkennung) [MH99, AC, Bad90],

- Optimierung von Datenstrukturen [AW87, RSH00, HB00],
- Parallelisierung [PMSS99, PPL+99, PSL+98, RJ97, WBWS01, KK96],
- adaptiv progressives Ray-Tracing (Approximation) [Bal99, TBD96] oder
- Hardware-Optimierung [WBWS01, PKGH97]

bewirkten die Verbesserung des Basisalgorithmus und erlauben heute zum Teil interaktives Ray-Tracing. Einen guten Überblick über verschiedene Beschleunigungsverfahren und Erweiterungen des Basis-Ray-Tracers findet man vor allem in [Gla89, WW92, FvDFH90, Rau93, Shi00, MH99, WG84].

2.3.1 Adaptiv progressives Ray-Tracing

Progressives Ray-Tracing: Progressives Ray-Tracing bedeutet, dass die Approximation des zu generierenden Bildes in einem oft sichtbaren Prozess solange verbessert wird, bis die gewünschte Qualität erreicht ist. Einige frei verfügbare Ray-Tracer wie RADIANCE [War94] oder POV-Ray [PR] bedienen sich einer einfachen Technik [JvW83, Whi80], um bereits in einer frühen Berechnungsphase einen ersten Eindruck (Preview) des Bildes zuzulassen. Hier werden quadratische Pixelblöcke mit konstanter Farbe durch jeweils *ein* Sample-Wert repräsentiert. Die Blöcke sind dabei im Initialzustand groß und werden progressiv bis mindestens auf Pixelgröße verfeinert.

Eine andere Methodik eines progressiven Ray-Tracers beschrieb HANRAHAN in [Han86]. Sein Ray-Tracer bevorzugt eine *breadth-first* Bearbeitung (anstatt der traditionellen *depth-first*), um die Kohärenz zwischen Strahlen der gleichen Rekursionstiefe besser auszunutzen. Das berechnete Bild zeigt sich erst ohne Reflexion beziehungsweise Brechung und fügt diese schließlich Schritt für Schritt entsprechend der Rekursionstiefe hinzu.

Adaptiv progressives Ray-Tracing: Standard-Ray-Tracer basieren auf *uniformen* Sampling, bei dem das zu berechnende Bild Pixel für Pixel abgetastet und der resultierende Farbwert je Pixel bestimmt wird. Adaptive Ray-Tracer arbeiten dagegen *non-uniform*. Sie variieren in der Verteilung der ermittelten Samples über der Bildebene, indem ein adaptives Sampling des Definitionsbereiches (des Bildes) angepasst an die Varianz der Funktionswerte (der Radiance-Funktion) erfolgt. Das adaptive Sampling erfolgt dabei in der Regel progressiv, da sich die Wahl der neu zu bestimmenden Samples auf der Auswertung des vorherigen Schrittes stützt.

Eine Form adaptiven Ray-Tracings findet man beim *Supersampling* [Coo86, Dip85] zur Reduktion von *Aliasing*-Effekten[3]. Hier wird entsprechend der Radiance-Funktion mindestens ein Primärstrahl pro Pixel ausgesandt und die endgültige Farbe des Pixels aus dem Durchschnitt der ermittelten Samples pro Pixel berechnet. Das Bild ist nicht komplett, solange jedes Pixel nicht mindestens einmal abgetastet wurde.

[3] Treppeneffekte, die durch die Rastercharakteristik (Diskretisierung) des Rechnerbildes entstehen.

Eine Rekonstruktion der Radiance-Funktion mit weniger Samples (*Undersampling*) als vorhandener Pixel zu erreichen, ist das Ziel einer anderen Gruppe adaptiver Ray-Tracer, zu denen auch das in dieser Arbeit vorgestellte Vertex-Tracing zählt. In diesem Zusammenhang schlugen erstmals PAINTER und SLOAN [PS89] die Behandlung des Bildes als kontinuierliche Region ohne Pixelgrenzen vor. Sie beabsichtigten die Rekonstruktion eines schnellen low-quality Images sowie eines *anti-aliased* high-quality Images durch eine optimale Sample-Verteilung im Bildraum. Hierzu bedienten sie sich eines Samplings, das auf der Verfeinerung des Bildes durch einen zweidimensionalen BSP-Tree basiert und erst bei Erreichung einer gewünschten Qualitätsstufe abbricht. Die generierten Samples wurden durch die sogenannte *Delaunay Triangulierung* [ELPZ94] im Bildraum interpoliert.

Erweitert wurde die Arbeit aus [PS89] durch PIGHIN *et al.* in [PLS97]. Ihr Verfahren basiert auf der Nutzung von *Constrained-Edges*, welche vorhandene Diskontinuitäten in der Bildebene repräsentieren. Die Constrained-Edges resultieren in einem *Discontinuity Mesh* wie sie auch im Bereich von Radiosity Verwendung finden [Hec92, LTG92] und bewirken eine Verbesserung der Delaunay Triangulierung. Diese orientiert sich nun direkt an den vorgegebenen Kanten und erzeugt dadurch bereits in einer frühen Berechnungsphase gute Approximationen mit scharf ausgeprägten Diskontinuitätskanten. Der Nachteil dieser Methode liegt jedoch in der Bereitstellung eines derartigen Discontinuity Meshes. PIGHIN *et al.* schlug dazu Hardware-basierte Methoden zur Bestimmung von Objekt-, Schatten- sowie Reflexionskanten vor, bei denen jedoch insbesondere die Methoden zur Schatten- oder Reflexionsberechnung auch mit Hardware-Unterstützung zu Laufzeitengpässen führen können (↗ Abschnitt 2.2). Die Behandlung von Nichtlinearitäten bleibt darüber hinaus unberücksichtigt.

RAIDL und BARTH entwickelten einen Algorithmus, der ein rekursives unterteilen der Bildebene in Form eines Quad-Trees vorsieht [RB96]. Ziel ist es, mit Hilfe *priorisierter* Regionen und einem sogenannten *Neighbour-Influence* Mechanismus eine progressive Verfeinerung bis zur fehlerfreien Darstellung zu realisieren und dabei dem Problem des Aliasing während des Verfeinerungsprozesses zu begegnen. Dabei begünstigt das strukturierte Sampling an den Eckpunkten der Regionen zwar das Anti-Aliasing, lässt jedoch Diskontinuitätskanten durch bilineare Interpolation verwaschen. Darüber hinaus kann die Generierung eines fehlerfreien Finalbildes (siehe [Bal99]) nicht garantiert werden.

In [Guo98] wurde von GUO ein modifiziertes Verfahren von adaptiv progressivem Sampling vorgestellt und in [SSPS] bezüglich Texturen erweitert. GUO nutzt sogenannte *Directional Coherence Maps* (DCM) zur effizienten Behandlung von Radiance-Diskontinuitäten. Mit Hilfe der DCMs werden Diskontinuitäten des Bildes im Divide-and-Conquer Prinzip solange verfeinert, bis nur noch einfach gerichtete Diskontinuitätskanten durch *orientierte Finite Elemente* approximativ repräsentiert und schließlich interpoliert werden können. Das Sampling ist dabei nur im weiteren Sinne adaptiv, da lediglich entlang der Grenzen von Basisblöcken ein gezieltes Sampling zur Kantendetektion erfolgt. Mit dieser Methode kann GUO äußerst effektiv Diskontinuitäten detektieren. Das DCM-Prinzip wirft jedoch Probleme bei der Erkennung von Nichtlinearitäten auf.

Während die Verfahren aus [AF84, AMS91, Guo98, JvW83, RB96, SSPS, PS89] sich auf ein Sampling im Bildraum stützen, arbeitet die Methode von [Bal99, TBD96] im

Objektraum. Bala *et al.* benutzen sogenannte *Interpolanten*, die um die Szenenobjekte konstruiert werden und ihre Radiance repräsentieren. Je nach Bedarf ist es möglich, die Interpolanten weiter zu verfeinern, um sowohl Diskontinuitäten als auch Nichtlinearitäten besser zu approximieren. Typisch für dieses Objektraum-basierte Verfahren ist die Möglichkeit zur Ausnutzung von Object-Space Kohärenz sowie temporaler Kohärenz. Bala *et al.* beschreiben, wie die Einführung von *hierarchischen Linetrees* schnelle Updates bei Szenenmodifikationen oder Viewpoint-Änderungen gestatten. Außerdem wird ein *Error-Bound System* unterstützt, dass unter Vorgabe eines Darstellungsfehlers die entsprechende visuelle Qualität des Bildes garantiert. Nachteil des Verfahrens besteht jedoch darin, dass die Konstruktion von Interpolanten nur um konvexe Szenenobjekte erfolgen kann.

Wie leicht zu erkennen, gibt es eine Fülle von Algorithmen, die sich dem intelligenten Sampling beim Ray-Tracings verschrieben haben. Es ist aber auch die logische Konsequenz bei der näheren Betrachtung von Bildern, ihre Charakteristik in Form von Kohärenzeigenschaften sinnvoll auszunutzen. Auffällig ist nur, dass die Mehrheit der Sampler ein adaptiv progressives Refinement im Bildraum durchführen. Gründe dafür liegen sicherlich in der einfacheren Berechnung im $\mathbb{R}^2$ und darüber hinaus im relativ geringem Speicherbedarf. Wie bereits erwähnt, ist jedoch ein Refinement im Objektraum prädestiniert, die Eigenschaften der temporalen Kohärenz beziehungsweise Szenenkohärenz auszunutzen. Einmal aufgebaute Datenstrukturen bieten ein mächtiges Werkzeug, geeignete Szenencharakteristiken abzubilden und können somit unter anderem elegant zur schnellen Rekonstruktion von Radiance-Information im nachfolgenden Frame wiederverwendet werden.

Auch das hier vorgestellte Vertex-Tracing [USBB01] kann zur Gruppe der Sampler mit Objektraum-basiertem Refinement hinzugezählt werden und nutzt die sich daraus ergebenen Vorteile bezüglich der Kohärenz. Im Gegensatz zu allen anderen bekannten Verfahren wird hier jedoch keine Detektion von Diskontinuitätskanten, die aus Objektgrenzen resultieren, notwendig, da sich die Datenstruktur des Refinements direkt an der Geometrie *beliebig tesselierter* Objekte orientiert. Ferner beeinflusst dieser Ansatz die Behandlung von Texturen begünstigend und begegnet damit einem weiteren Problem beim adaptiv progressiven Ray-Tracing. Durch den Einsatz modernster Graphikhardware können effiziente Mischformen zwischen Rasterisierungsgraphik und Ray-Tracing erstellt werden.

2.3.2 Parallelisierung progressiver Ray-Tracer

Im Gegensatz zur Parallelisierung traditioneller Ray-Tracer [KK96, RJ97, PSL+98, PMSS99, PPL+99, WBWS01], in denen das Bild meist in rechteckige Bereiche aufgeteilt und eine unabhängige Berechnung auf jedem Prozessor vorgenommen wird, existiert nur spärlich Literatur bezüglich einer verteilten Berechnung adaptiv progressiver Ray-Tracer. Ihre Parallelisierung beziehungsweise Verteilung gestaltet sich gegenüber dem konventionellen Ray-Tracing weitaus schwieriger. Aufgrund der inhärent sequentiellen Natur des adaptiv progressiven Ray-Tracings entstehen während der sukzessiven Sample-Generierung zusätzliche Abhängigkeiten zwischen den Verfeinerungsebenen. Die Position eines neu zu bestimmenden Samples resultiert ausschließlich aus den Ergebnissen des vorangegange-

nen Samplings. Die Entscheidung über neue Sample-Positionen kann ein Prozessor nicht unabhängig von anderen vornehmen. Es könnten sonst neue, ungewollte *Diskontinuitäten* an den Bereichsgrenzen auftreten, die sich als Artefakten darstellen.

Eine weiterer Hürde beim Parallelisieren eines adaptiv progressiven Ray-Tracers ist, wie bei den meisten parallelen Algorithmen, die intelligente Verteilung der anfallenden Arbeit, das *Load-Balancing*. Je nach Prozessorleistung, Kommunikationsvermögen und Komplexität der Teilaufgabe gilt es, die Last dynamisch oder statisch gleichmäßig zu wichten, um eine optimale Auslastung des Systems zu gewährleisten [Hub97, KGGK94].

Notkin und Gotsman beschreiben in [NG95] ein statisches beziehungsweise dynamisches Load-Balancing für paralleles, adaptiv progressives Ray-Tracing. Das System basiert auf einer Master-Slave Architektur, indem die Bildebene vom Master in rechteckige Bereiche unterteilt und jedem Slave eine Anzahl an Bereichen zugewiesen wird. In einem Pre-Prozess erfolgt zunächst ein statisches Load-Balancing. Hierzu wird die Komplexität jedes Bereiches grob bestimmt, um ein gleichmäßiges Verteilen auf die Prozessoren zuzulassen. Die zugewiesenen Bereiche können dabei in verschiedenen Regionen des Bildes liegen. Ein Prozessor bearbeitet exklusiv seine Bereiche, die jedoch nicht stringent zusammenliegen. Da der Pre-Prozess nur eine grobe Approximation darstellt, die auf Stichproben basiert, ist die Qualität des Load-Balancings relativ schlecht. Abhilfe schafft hier ein zusätzliches, dynamisches Load-Balancing der Slave-Prozesse. Hierzu wird zyklisch jedem Slave eine bestimmte Anzahl an Samples zugewiesen, die sich an die Last eines jeden Prozessors dynamisch anpasst.

Auch in [NG95] liegt die Schwierigkeit in der sequentiellen Abhängigkeit des progressiven Ray-Tracings. Notkin und Gotsman reduzieren dieses Problem, indem zusätzliche Samples an den Bereichsgrenzen geschossen werden, was jedoch eine erhöhte Redundanz im Sampling hervorruft.

Eine Erweiterung des Verfahrens hinsichtlich Interaktion und Gewährleistung vordefinierter Update-Raten findet man in [RGS97]. Reisman *et al.* implementierten zusätzlich eine *Interprocess*-Kommunikation zwischen den Slaves, um ein verbessertes Sampling an den Bereichsgrenzen zu erlauben. Samples, die aufgrund des Sampling-Generators der Delaunay-Triangulierung in Nachbarregionen hineinfallen, werden dem Nachbarprozess mitgeteilt. Der zusätzliche Kommunikationsaufwand wird dabei durch konvexe Bereiche mit möglichst kurzen Grenzen reduziert.

Auch die Parallelisierung des hier vorgestellten Vertex-Tracings [UBSB01] soll die Erlangung interaktiver Update-Raten positiv beeinflussen. Im Gegensatz zu [NG95] und [RGS97] basiert die Aufteilung jedoch nicht auf rechteckigen Regionen der Bildebene, sondern vielmehr auf einer Distribution von Primärdreiecken des Objektraumes (↗ Kapitel 4). Ferner findet zur Vermeidung zusätzlicher Kommunikation kein Informationsaustausch zwischen den Slave-Prozessen wie in [RGS97] statt. Auftretende Abhängigkeiten während des Samplings, die über Bereichsgrenzen hinauslaufen, können als sequentieller Anteil im Anschluss der verteilten Berechnung vom Master abgearbeitet werden. Hervorzuheben ist, dass dabei das Prinzip des Vertex-Tracings derartige Abhängigkeiten stark reduziert.

Implementiert wurde ein dynamisches Load-Balancing, welches zentralisiert vom Master vorgenommen wird und auf einer apriori Aufwandsabschätzung basiert. Zielplattfor-

men sind sowohl heterogene Rechnernetzwerke als auch Shared-Memory-Systeme. Neu in diesem Umfeld ist jedoch die hier vorgestellte Parallelisierung des adaptiv progressiven Ray-Tracers mittels *Multi-Threading*. Dieser Ansatz impliziert nicht nur die Verwendung von Shared-Memory-Systemen, sondern erlaubt darüber hinaus einen parallelen Sampling-Prozess, der sich äquivalent zum sequentiellen Vorbild verhält.

2.4 Diskussion

Die Anforderung der vorliegenden Arbeit besteht in der Simulation spekularer Reflexionen in Echtzeit. Die Lösung dieser Problemstellung führte schließlich zur Entwicklung eines neuartigen Ansatzes in Form eines adaptiv progressiven Ray-Tracers und dessen Parallelisierung. Andere Rendering-Verfahren mit der Möglichkeit zur spekularen Reflexionsberechnung wie das Environment-Mapping, das Image-Based Rendering oder Verfahren zur expliziten Geometriespiegelung zeigten sich dabei als mehr oder weniger ungeeignet. Unzureichende physikalische Korrektheit, Nichtdeterminismus, inakzeptable Pre-Compute-Zeiten, hoher Speicherbedarf oder ungenügende Flexibilität während des Renderings ließen diese Algorithmen durch ihre Grundeigenschaften ausscheiden.

Das Ray-Tracing liefert hier den Basisalgorithmus eines adaptiv progressiven Samplings. Der vorliegende Ansatz differenziert sich jedoch von bereits veröffentlichten Verfahren und beschreibt insofern einen weiteren Weg in diesem Forschungsgebiet. Ausgangsbasis zur Sample-Generierung ist hier, im Gegensatz zu allen bekannten Verfahren, die unmittelbare, polygonale Objektgeometrie im Objektraum. Die Detektion von Objektsilhouetten, als Diskontinuitätskanten im Bildraum, wird erstmals hinfällig und verursacht dadurch eine signifikante Reduzierung der Sample-Anzahl. Ferner unterstützt die durch das Sampling generierte Datenstruktur direkt die Integration von Texturen.

Hinsichtlich der Parallelisierung des Vertex-Tracers wird mit dieser Arbeit eine Lastverteilung vorgestellt, die, konträr zu anderen Arbeiten, ausschließlich auf Information des Objektraumes basiert. Die Folge ist eine erhebliche Reduktion von Nachbarschaftsabhängigkeiten, die bei einem verteilten Ansatz zusätzlich zwischen den Slaves entstehen können. Hinzu kommt, das durch einen Multi-Threading Ansatz vollständig der sequenzielle Anteil des progressiven Sampling „aufgelöst“ wird und insofern höhere *Speed-Up* Werte begünstigt.

Kapitel 3

Vertex-Tracing

Das Verfahren *Vertex-Tracing* ist eine Modifikation sowie Erweiterung des Basisalgorithmus Ray-Tracing. Es bezweckt mittels Refinement im Objektraum ein optimiertes Sampling der Radiance-Funktion im Bildraum, um mit einer möglichst geringen Anzahl von Abtastwerten (Samples) eine hinreichend qualitative Rekonstruktion der Radiance-Funktion zuzulassen.

Das vorliegende Kapitel gibt einen umfassenden Einblick über die Mechanismen des Vertex-Tracings. Basierend auf der Beschreibung der zugrunde liegenden Ideen wird zunächst ein Überblick über die grundsätzlichen Prozesse des Verfahrens in Abschnitt 3.1 gegeben. Aufgrund der Charakteristik des Verfahrens entspricht das weitere Vorgehen den Phasen der *Signalverarbeitung*. Hierzu findet zuerst eine Beschreibung des adaptiv progressiven Samplings der Radiance-Funktion in Abschnitt 3.2 statt, bevor näher auf den Prozess des Refinements in Abschnitt 3.3 eingegangen wird. Schwerpunkte beim Refinement bilden unter anderem Strategien und Datenstrukturen zur effizienten Verfeinerung sowie Methoden, die eine sichere Konvergenz des adaptiv progressiven Funktions-Samplings erzwingen. Der mittlere Teil des Kapitels (Abschnitt 3.4) beschäftigt sich weiterführend mit der Optimierung und Beschleunigung des Samplings, das wiederum Einfluss auf den Refinement-Prozess ausübt. Methoden zur Rekonstruktion der Radiance-Funktion werden im Vordergrund des Abschnitts 3.5 stehen. In diesem Zusammenhang wird insbesondere auf das Thema Rendering und Shading sowie die Integration von Texturen zur qualitativen Aufwertung des Systems eingegangen.

3.1 Idee und Überblick

Der Basisalgorithmus eines Standard-Ray-Tracers generiert für jedes Pixel der Bildebene einen Strahl, dessen Startpunkt sich im Augpunkt des Betrachters befindet und dessen Richtung durch die Position des zu durchlaufenden Pixels definiert ist (in Abbildung 3.1, blauer Strahlengang). Dieser Primärstrahl $\vec{r}_p$ wird auf Kollision mit allen Szenenobjekten geprüft. Im Falle einer Kollision entstehen am Schnittpunkt des Objektes Sekundärstrahlen für Reflexion ($\vec{r}_r$), Brechung ($\vec{r}_t$) und Beleuchtung ($\vec{r}_s$), wobei für die Strahlen $\vec{r}_r$ und $\vec{r}_t$ unter Umständen eine weitere rekursive Betrachtung erfolgt [Whi80].

Zur effizienten Bestimmung eines Schnittpunktes zwischen Strahl und Szenenobjekt gibt es eine Vielzahl von Verfahren in der Literatur (↗ Abschnitt 2.3), die sich unabhängig der Strahlenart einsetzen lassen. Für die schnelle Berechnung von Schnittpunkten bei Primärstrahlen existieren jedoch auch Methoden, die eine Sonderbehandlung darstellen. Eine solche Methode, auch als *First-hit Speedup* bezeichnet, wurde von WEGHORST in [WG84] vorgestellt. Unter Zuhilfenahme eines *Hidden-Surface-Removal* (HSR) Algorithmus, hier ein modifiziertes Z-Buffer Verfahren, wird für jedes einzelne Pixel das sichtbare Szenenobjekt ermittelt. Für jeden Primärstrahl eines Pixels erfolgt schließlich eine äußerst effektive Schnittpunktberechnung mit dem referenzierten Objekt oder besser noch dem referenzierten Polygon. Diese Methode impliziert, dass der HSR eine effizientere Lösung als ein ordinärer Schnittpunkttest aufweist.

Eine noch stärkere Modifikation der Primärstrahlberechnung führt das Vertex-Tracing ein. Abbildung 3.1 zeigt die Idee des Verfahrens (roter Strahlengang). Ohne Berücksichtigung des diskreten Charakters der Bildebene, ergibt sich der Primärstrahl $\vec{r}_p$ direkt aus:

$$\vec{r}_p \quad = \quad \vec{v}_i - \vec{p}_a \tag{3.1}$$

mit $\vec{p}_a$ dem Augpunkt und jeweils einem Vertex $\vec{v}_i$ eines Szenenobjektes, dem *Primär-Vertex*. Der Schnittpunkttest von Primärstrahlen entfällt damit vollständig, da jeder

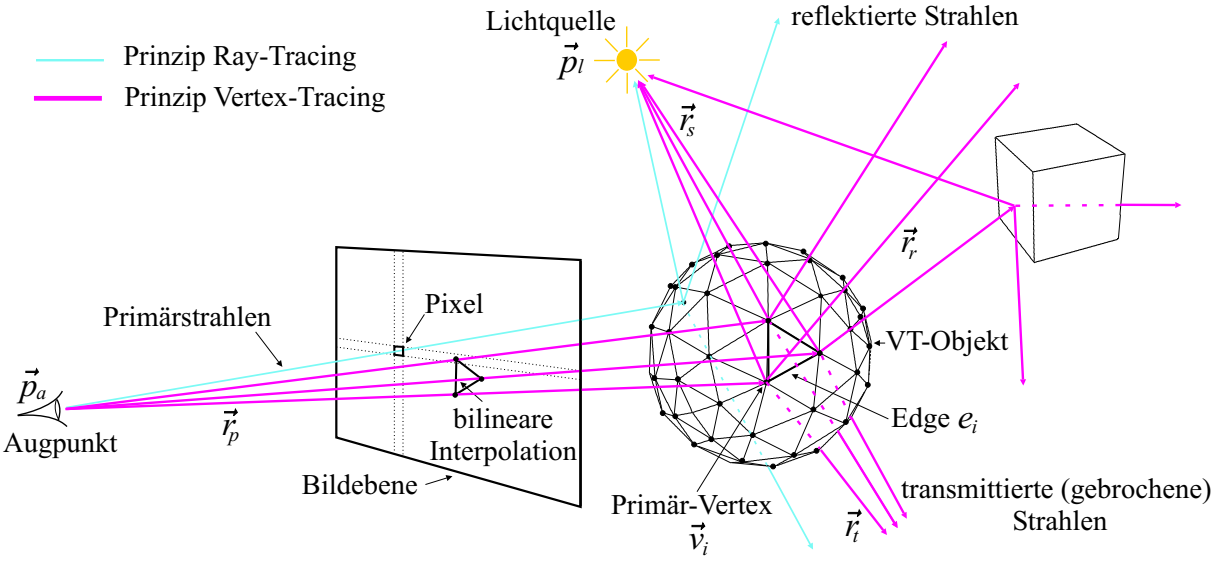


Abbildung 3.1: *Grundidee des Vertex-Tracings. Unterschiede zum traditionellen Ray-Tracing liegen vor allem im direkten Beschuss von Objekt-Vertices durch Primärstrahlen und der adaptiven Verfeinerung der Objektgeometrie.*

Primär-Vertex des Objektes den Schnittpunkt von $\vec{r}_p$ unmittelbar impliziert. Jedes betrachtete Szenenobjekt (VT-Objekt) bildet mit seiner Geometrie die Basis eines adaptiven

Samplings. Die beliebige Anordnung der Vertices entspricht dabei einer willkürlichen Varianz der Primär-Samples im Bildraum, die als Grundlage einer adaptiven Verfeinerung des Polygonnetzes dienen. Die Vertices stellen somit die Ausgangsbasis jeder weiteren Berechnung dar und gaben dem Verfahren den Namen 'Vertex-Tracing'.

Die direkte Zuweisung der Primärstrahlen auf die Vertices $\vec{v}_i$ einer Primär-Face hat die Bildung eines sogenannten *Beams* wie in [HH84] zur Folge, der zueinander kohärente Strahlen repräsentiert. Im Gegensatz zu den Arbeiten in [HH84] beziehungsweise dem verwandten Cone-Tracing aus [Ama84] muss jedoch bei dem hier vorgestellten Ansatz keine Kollisionserkennung des Beams mit der Szene erfolgen. Der wesentlichste Unterschied liegt jedoch vor allem in der fortführenden Strahlenverfolgung. Während in [Ama84] und [HH84] weitere rekursive Beams erzeugt werden, erfolgt beim Vertex-Tracing eine rekursive Betrachtung durch Strahlen.

Da in der Regel nicht alle Vertices aus der Perspektive des Betrachters sichtbar sind, muss eine Bestimmung der Sichtbarkeit (Visibility) für jeden Vertex beziehungsweise der verbindenden Kanten (Edges) erfolgen. Dieser Visibility-Test arbeitet als Pre-Prozess mittels Hardware-Rendering. Er besitzt insofern Ähnlichkeit zu dem oben skizzierten Vorgehen aus [WG84] (↗ Abschnitt 3.4.1).

Für jeden Vertex einer sichtbaren Edge findet im Weiteren eine identische Betrachtung zum Ray-Tracing statt. Es wird ein Schattenfühler $\vec{r}_s$ mit:

$$\vec{r}_s \;=\; \vec{p}_l - \vec{v}_i \tag{3.2}$$

und bedingt ein reflektierter Strahl $\vec{r}_r$ beziehungsweise ein gebrochener Strahl $\vec{r}_t$ mit:

$$\vec{r}_r \;=\; \vec{r}_p \,-\, 2(\vec{r}_p \cdot \vec{n}_i)\,\vec{n}_i, \tag{3.3}$$

$$\vec{r}_t \;=\; \frac{\eta_1}{\eta_2}\,\vec{r}_p \,+\, \left[-\frac{\eta_1}{\eta_2}\,\vec{r}_p \,\cdot\, \vec{n}_i \,-\, \sqrt{1 \,+\, ((\vec{r}_p \,\cdot\, \vec{n}_i)^2 \,-\, 1)\left(\frac{\eta_1}{\eta_2}\right)^2}\right] \vec{n}_i \tag{3.4}$$

erzeugt[1] und im traditionellen Stil des Ray-Tracings in den Raum mit entsprechender Rekursionstiefe verfolgt. Die akkumulierten Intensitäten der Sekundärstrahlen werden auf jedem Vertex zusammengefasst und entscheiden über die Unterteilung der Kante sowie ihrer adjazenten Polygone.

Mit den neu entstandenen, aus dem Refinement-Prozess resultierenden Vertices wird analog zu den Primär-Vertices verfahren. Der Refinement-Prozess selbst bricht ab, wenn ein entsprechendes Qualitätskriterium erfüllt wurde, das heißt, die Interpolation entlang einer Kante zulässig ist. Abbildung 3.2 zeigt noch einmal schematisch den Basisalgorithmus des Vertex-Tracers. Ausgehend von einer tesselierten polygonalen Szene wird jede Edge e_i eines VT-Objektes auf Sichtbarkeit geprüft. Im Falle der Sichtbarkeit beginnt das Sampling. Für jeden Vertex wird ein Primärstrahl geschossen und entsprechend der Objekteigenschaften eine rekursive Strahlenverfolgung durchgeführt, die wiederum

[1] η_1, η_2 bestimmen den Brechungsindex der beteiligten Medien und $\vec{n}_i$ gibt die Normale auf dem Objekt-Vertex $\vec{v}_i$ an. Die ausführliche Herleitung der Gleichungen 3.3 beziehungsweise 3.4 findet man zum Beispiel in [Rau93]. Die Gleichungen basieren auf den Gesetzen der geometrischen Optik.

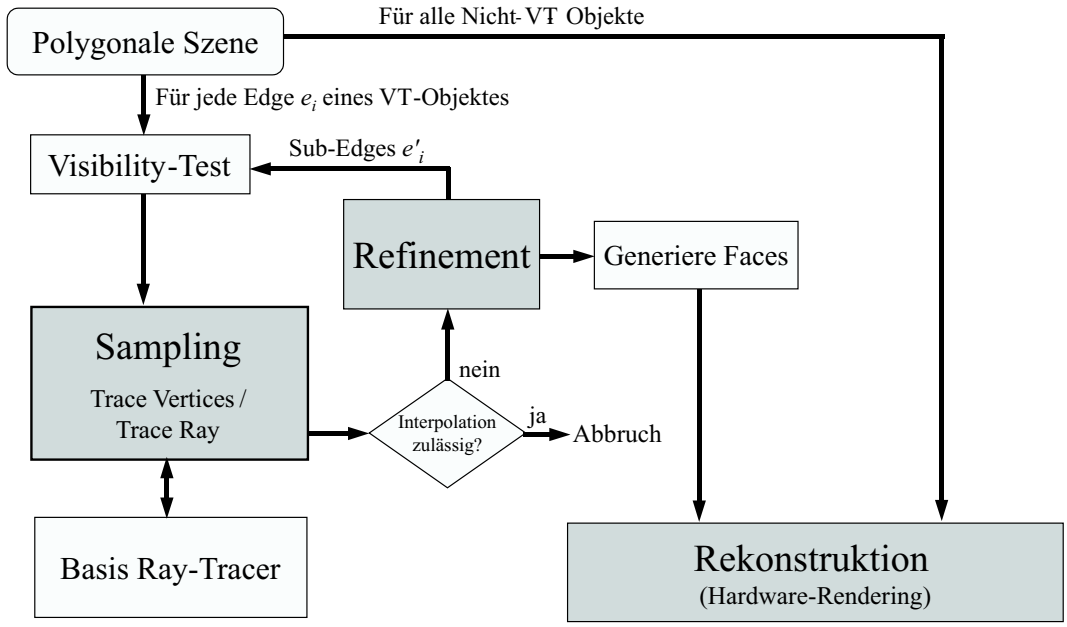


Abbildung 3.2: *Basisalgorithmus des Vertex-Tracings*

auf dem Basis-Ray-Tracer aufsetzt. Anhand der akkumulierten Radiance-Werte auf den Edge-Vertices folgt die Entscheidung über eine notwendige Unterteilung der Edge. Ist die Interpolation der Radiance-Werte nicht zulässig, muss die Edge unterteilt werden. Durch das Refinement entstehen neue Sub-Edges beziehungsweise Sub-Faces. Letztere werden in jedem Verfeinerungsschritt dem Hardware-Rendering übergeben, um ein progressives Verfahren zu unterstützen. Nach einer perspektivischen Projektion der Faces erfolgt der Rekonstruktionsprozess durch bilineare Interpolation der Radiance mittels *Gouraud-Shading*. Die generierten Sub-Edges e'_i reihen sich unterdessen in die Liste der zu testenden Edges ein und beginnen den Zyklus von Neuem.

Das Rendering aller Nicht-VT-Objekte geschieht unabhängig zum eigentlichen Vertex-Tracing-Prozess. Ein hybrides Rendering von Objekten mit globalen Beleuchtungscharakteristiken und Vertretern lokaler Beleuchtungsmodelle, wie sie typischerweise durch Hardware-Rendering erzeugt werden, ist in der Form realisierbar.

3.2 Adaptiv progressives Sampling

Eine Vielzahl von Verfahren aus der digitalen Bildsynthese basiert auf dem Sampling und der anschließenden Rekonstruktion der Radiance-Funktion. Insbesondere das Ray-Tracing kann dabei als typischer Vertreter derartiger Sampling-Methoden angesehen und

im Speziellen zur Gruppe von *Punkt-Samplern*[2] hinzugezählt werden.

Jeder infinitesimal dünne Strahl des Ray-Tracing-Prozesses stellt eine punktuelle Probe der Radiance in einer gegebenen Richtung dar. Je nach Sichtweise erfolgt das Sampling dabei im $\mathbb{R}^3$ dem Objektraum oder im $\mathbb{R}^2$ dem Bildraum. Typische Vertreter des $\mathbb{R}^3$-Samplings sind das *Path-Tracing* beziehungsweise das *Monte-Carlo Ray-Tracing* [Shi00]. Hier wird beispielsweise mittels stochastischem Sampling das Integral der einfallenden Radiance über einem infinitesimal kleinen Flächenstück im Objektraum approximiert (Illumination-Sphere Sampling).

Das klassische Ray-Tracing hingegen kann im weiteren Sinne als Sampling im $\mathbb{R}^2$ angesehen werden. Ziel ist es hier, eine kontinuierliche Radiance-Funktion über dem Bildraum (↗ Abbildung 3.3a), mit Hilfe von Stichproben zu rekonstruieren. Die resultierende Approximation der Radiance-Funktion ist dabei vor allem von der Art des Samplings abhängig, bei dem im Allgemeinen zwischen uniform und non-uniform unterschieden wird.

Das Sampling beim Standard-Ray-Tracing ist in der Regel *uniform*, das heißt, die Wahl der Sample-Positionen erfolgt mittels regelmäßigem Sample-Gitter (↗ Abbildung 3.3b). Standard-Ray-Tracer nutzen hierfür das diskrete Pixelraster der Bildebene, indem jedes Pixel eine Stichprobe der Radiance-Funktion enthält. Der Abstand zweier Pixel entspricht dabei dem Sample-Intervall s, das zu einer maximalen Abtastfrequenz von $f_a = 1/s$ führt. Nach dem *Nyquist*-Kriterium mit $f_a > 2f_{max}$ sollte jedoch die Abtastfrequenz mindestens das Doppelte der maximalen Ausgangsfrequenz f_{max} betragen. Da nun in der Bildsynthese keine obere Schranke für vorkommende Frequenzen existiert, sondern diese sogar unendlich hoch sein können (beispielsweise beim Übergang von Schwarz nach Weiß auf einem Schachbrett), sind Unterabtastungen vorprogrammiert, die sich negativ in Aliasing-Artefakten darstellen. Eine Erhöhung der Sample-Dichte ist aufgrund des vorgegebenen Pixelrasters jedoch kaum möglich[3], würde es auch nur an wenigen hochfrequenten Bildbereichen eine bessere Approximation bewirken. Hier liegt der entscheidende Nachteil des uniformen Samplings. Eine *Überabtastung* von Regionen mit niedrigen Frequenzen beziehungsweise hoher Image-Kohärenz, ist in der Regel unvermeidbar. Die Generierung „redundanter“ Samples führt zu hohen Laufzeitverlusten. Abhilfe schafft ein adaptives Vorgehen des uniformen Samplings wie unter anderem in [AMS91, BTB91, JvW83, Whi80] beschrieben.

Im Gegensatz zu seinem uniformen Vertreter findet beim adaptiv *non-uniforme* Sampling (↗ Abbildung 3.3c und 3.3e) eine kontinuierliche Anpassung an die abzutastende Frequenz statt, indem sich die Sample-Positionen beliebig der Funktion anpassen können. Beginnend mit einem non-uniformen *Initial-Sample-Pattern*, das beispielsweise stochastisch ermittelt werden kann, startet ein adaptiver Verfeinerungsprozess, der sich Schritt für Schritt der gewünschten Approximation der Radiance-Funktion annähert. Der Vorteil dieser Sample-Strategie beruht dabei im Wesentlichen auf zwei Faktoren:

[2]Andere Methoden wie das Cone-Tracing [Ama84] oder das Beam-Tracing [HH84] bedienen sich einem Bereichs-Sampling, bei dem über ein bestimmtes Intervall (Sample-Bereich) des Definitionsbereiches integriert wird.

[3]Durch die Einführung von Sub-Pixels kann ein Super-Sampling erzielt werden. Formen des Super-Samplings finden vor allem bei Anti-Aliasing Algorithmen Anwendung.

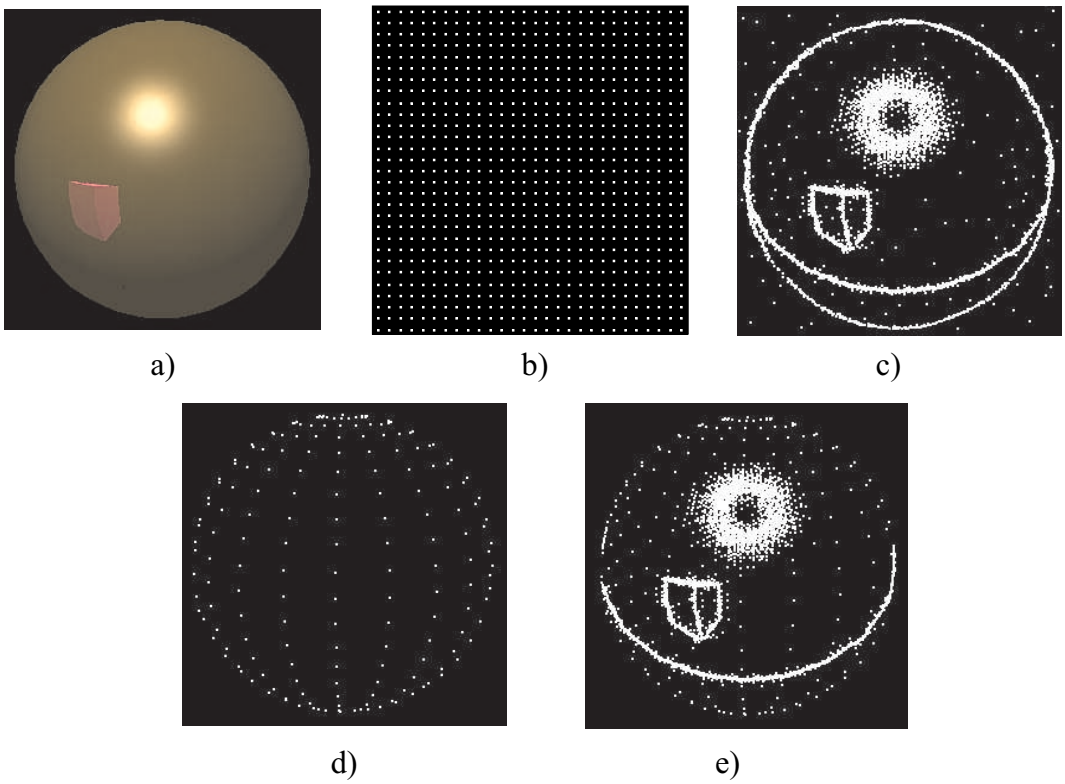


Abbildung 3.3: *Varianten des Bild-Samplings. a) Radiance-Funktion, die einem Sampling unterzogen werden soll. b) Traditionelles uniformes Sampling beim Ray-Tracing anhand des Pixelrasters. c) Non-uniformes adaptives Sampling mit Objektkantendetektion. d) Initial Sample-Pattern beim Vertex-Tracing gegeben durch die Objekt-Vertices. e) Non-uniformes adaptives Sampling ohne Objektkantendetektion beim Vertex-Tracing.*

1. Die subjektive Wahrnehmung von Aliasing-Artefakten ist deutlich gegenüber dem uniformen Sampling reduziert. Hier werden unstrukturierte Aliasing-Artefakte (Rauschen) generiert, die das visuelle System des Menschen als weniger störend empfindet.

2. Es findet keine Überabtastung niederfrequenter Bereiche statt, sondern das Sampling konzentriert sich auf Regionen höherer Frequenz. Die Auswahl relevanter Samples kann gezielter und damit effizienter erfolgen.

Auch das Vertex-Tracing verfährt nach diesem Prinzip. Wie bereits weiter oben erwähnt, stellt hier die Objektgeometrie mit ihren Vertices $\vec{v}_i$ die Ausgangsbasis des Sample-Prozesses und bildet insofern das Initial-Sample-Pattern. Abbildung 3.3d und 3.3e verdeutlichen diesen Zusammenhang. Die sichtbaren Vertices der Szenenobjekte (hier die Kugel) dienen als Initial-Samples (Abbildung 3.3d), von denen ausgehend ein adaptives Sampling bis zum gewünschten Ergebnis stattfindet (Abbildung 3.3e).

Die Verteilung der Initial-Samples über der Bildebene ist dabei abhängig von der Geometrie der Szenenobjekte sowie ihrer Lage bezüglich der Bildebene. Unabhängigkeit herrscht dagegen bezüglich der Tiefenkomplexität der Szene, da verdeckte Vertices durch den Visibility-Test herausgefiltert werden (siehe Abschnitt 3.4.1).

Bemerkenswert ist zudem die Tatsache, dass die Initial-Samples im Bildraum eine Abbildung der Primär-Vertices aus dem Objektraum repräsentieren. Da die Datenstruktur für das Refinement sowie zur bilinearen Interpolation der Samples ebenfalls im Objektraum definiert ist und mit den Primär-Vertices startet, ist die geometrische Gestalt der Szenenobjekte auch im Bildraum bereits impliziert. Es bedarf keiner Detektion von Objektkanten beziehungsweise -silhouetten wie es jedoch in äquivalenten Sampling-Verfahren [TBD96, Guo98] (Abbildung 3.3c) vorgenommen werden muss.

Zusammenfassend sei an dieser Stelle herausgestellt, dass die Datenstruktur für das Refinement im $\mathbb{R}^3$ erzeugt wird, jedoch die Kriterien des adaptiven Refinements anhand von Informationen aus dem $\mathbb{R}^2$ abgeleitet werden. Eine ausführliche Betrachtung des Refinement-Prozesses und seiner Randbedingungen folgt in den anschließenden Abschnitten.

3.3 Refinement im Objektraum

Im Anschluss eines jeden Sampling-Prozesses erfolgt die Rekonstruktion des Signals, das mit Hilfe des Samplings approximiert werden soll. Ein Signal $f(x, y)$ über der Bildebene wird durch n Samples $s_i = f(\vec{p_i})$ an den Positionen $\vec{p_i}$ abgetastet. Ergebnis dieser Abtastung ist die Funktion $f'(x, y)$ als Approximation des Originalsignals, der Radiance-Funktion. Genügt $f'(x, y)$ nicht den Kriterien der Rekonstruktion erfolgt ein *Resampling* des Signals an den neuen Sample-Positionen $\vec{p'}_i$ mit den Werten $s'_i = f'(\vec{p'}_i)$. Der Prozess des adaptiven Refinements stellt sich demnach wie folgt dar [Gla95]:

$$f(x,y) \overset{Sampling}{\longrightarrow} s_i = f(\vec{p_i}) \overset{Rekonstruktion}{\longrightarrow} f'(x,y) \overset{Resampling}{\longrightarrow} s'_i = f'(\vec{p'}_i). \tag{3.5}$$

Der Vorgang wird solange wiederholt, bis der Fehler δ_{error} mit

$$\delta_{error} = |f(x,y) - f'(x,y)| \tag{3.6}$$

unter einen vorgegebenen Schwellwert ϵ sinkt oder eine maximale Rekursionstiefe erreicht.

Die Position $\vec{p_i}$ der Initial-Samples s_i ergibt sich beim Vertex-Tracing aus der Projektion $\mathbf{M}_P$ der Objekt-Vertices $\vec{v_i}$ in die Bildebene mit $\vec{p_i} = \mathbf{M}_P \cdot \vec{v_i}$. Die Bildebene wird dabei als kontinuierliche Region betrachtet, so dass keine Diskretisierung im Bezug auf das Pixelraster vorgenommen wird. Für jedes $\vec{p_i}$ folgt die Bestimmung des Sample-Wertes s_i mit $s_i = f(\vec{p_i})$. Die Funktion f entspricht dem traditionellen rekursiven Ray-Tracings nach [Whi80], indem alle Intensitäten der geschossenen Strahlen entsprechend ihres Beitrags zum endgültigen Sample-Wert s_i gewichtet akkumuliert werden.

Mit der Projektion der Objekt-Vertices $\vec{v_i}$ bleiben die Topologieinformation des polygonalen VT-Objektes auch im $\mathbb{R}^2$ erhalten, so dass diese Informationen für ein weiteres

Refinement genutzt werden können. Infolge dessen existiert zwischen zwei beliebigen $\vec{p}_i$ und $\vec{p}_j$ die Objektkante e_i im $\mathbb{R}^3$. Das Kriterium einer Unterteilung von e_i ergibt sich aus der Differenz δ zwischen den Samples s_i und s_j mit $\delta = |s_i - s_j|$. Im Falle von $\delta > \epsilon$ erfolgt eine Unterteilung, indem ein neuer Vertex $\vec{v_i'}$ mit

$$\vec{v_i'} = \frac{\vec{v}_i + \vec{v}_j}{2} \tag{3.7}$$

im $\mathbb{R}^3$ eingefügt wird. Entsprechend resultiert der Normalenvektor $\vec{n_i'}$ auf $\vec{v_i'}$ in normierter Form mit

$$\vec{n_i'} = \frac{\vec{n}_i + \vec{n}_j}{\|\vec{n}_i + \vec{n}_j\|}. \tag{3.8}$$

Aus der Projektion von $\vec{v_i'}$ in den Bildraum folgt $\vec{p_i'}$. Durch ein Resampling der Radiance-Funktion an der Stelle $\vec{p_i'}$ entsteht ein neuer Sample-Wert s_i' mit $s_i' = f'(\vec{p'}_i)$, der wiederum zur Entscheidung einer weiteren Unterteilung herangezogen werden kann.

Um eine schnelle Konvergenz des adaptiven Refinement-Prozesses zu erreichen und darüber hinaus ein betrachtungsabhängiges (View-Dependent) Refinement zu gewährleisten, werden alle Edges e_i eines VT-Objektes entsprechend ihrer projizierten Länge l mit $l = |\vec{p_i} - \vec{p_j}|$ in eine sogenannte *Edge-List* einsortiert (↗ Abbildung 3.4). Beginnend mit der Längsten, wird jede Edge auf eine notwendige Unterteilung getestet. Muss eine Edge unterteilt werden, entstehen neue Sub-Edges e_i', die sich ebenfalls in die Edge-List

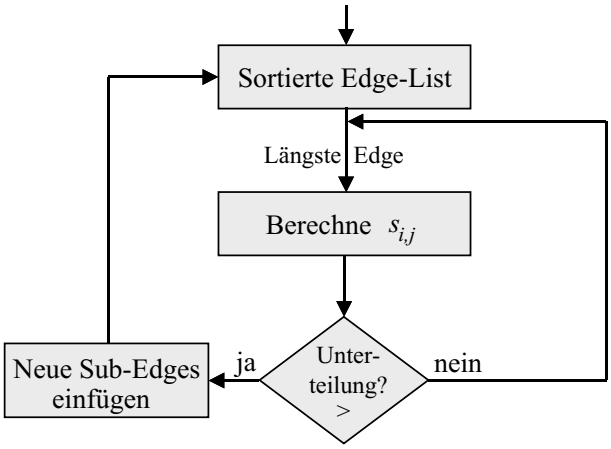


Abbildung 3.4: *Der iterative Prozess der Kantenunterteilung basiert auf einer sortierten Kantenliste, der Edge-List.*

einsortieren und einem erneuten Unterteilungstest unterziehen, sofern sie sichtbar sind. Der Vorgang verläuft iterativ, bis die Edge-List keine Elemente mehr besitzt. Initialisiert wird die Edge-List anhand der Primär-Edges, indem eine Sortierung in durchschnittlich $O(n \log(n))$ mit n der Anzahl der Primär-Edges erfolgt. Die Einordnung jeder neu entstanden Edge (*Sekundär-Edge*) geschieht dagegen on-the-fly in einen *Binärbaum*, so dass

jede Sub-Edge im Falle eines balancierten Baumes[4] in $O(\log(n))$ mit n der Anzahl der Listenelemente einsortiert werden kann.

Die neue topologische Struktur, die bei der Aufspaltung einer Edge e_i durch einen Vertex $\vec{v'}_i$ entsteht, zeigt Abbildung 3.5. Das Einfügen von $\vec{v'}_1$ in e_1 bewirkt die Entstehung der Sub-Edges e_{sub_1} und e_{sub_2}. Außerdem werden die adjazenten Faces f_L und f_R durch die neuen Edges e_L und e_R unterteilt. Es entstehen vier neuen Faces f_{L_1}, f_{L_2}, f_{R_1} und f_{R_2}.

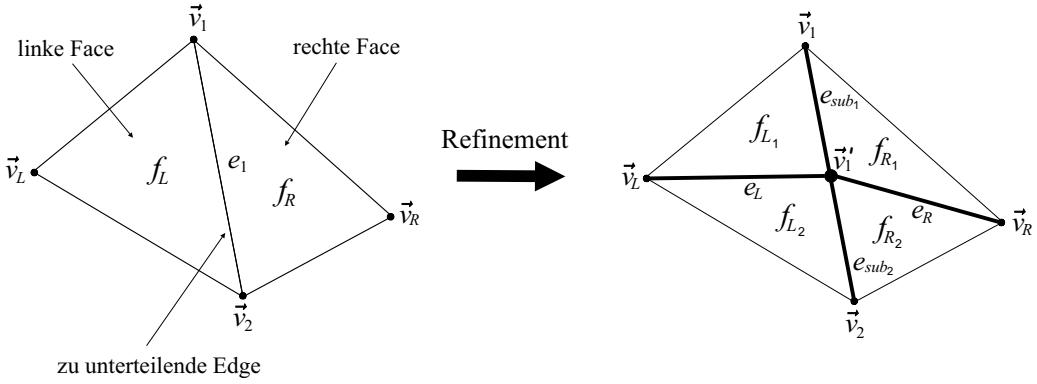


Abbildung 3.5: *Edge-basierte Unterteilung adjazenter Faces.*

Der Vorteil dieser Edge-basierten Methode liegt vor allem darin, dass im Gegensatz zu [SL91] pro Iterationszyklus lediglich *ein* neuer Vertex $\vec{v'}_i$ erzeugt wird und sein Sample-Wert berechnet werden muss. Vorausgesetzt die aktuell betrachtete Edge besitzt zwei adjazente Faces (links und rechts), entstehen bei einem einzigen Refinement-Schritt zugleich *vier* neue Sub-Faces. Nachteilig kann hier jedoch die exklusive Behandlung von Dreiecken angesehen werden. Aufgrund der Tatsache, dass die Sub-Edges e_L und e_R stets durch die gegenüberliegenden Vertices $\vec{v}_L$ beziehungsweise $\vec{v}_R$ verknüpft werden, muss eine Dreiecksrepräsentation vorliegen, um Eindeutigkeit zu wahren.

Abbildung 3.6 demonstriert das schrittweise Refinement eines Primärdreiecks entlang einer Diskontinuitätskante. Zu bemerken ist hier, dass die Verfeinerungsschritte *eins* und *zwei* lediglich eine neue Edge und damit nur zwei neue Teildreiecke entstehen lassen. Der Grund dafür liegt in den Außenkanten. Sie besitzen nur ein angrenzendes Dreieck, nämlich das Primärdreieck selbst. Im weiteren Refinement-Prozess (ab Stufe drei in Abbildung 3.6) besitzt jede Kante in der Regel zwei adjazente Dreiecke, so dass eine Verfeinerung analog zu Abbildung 3.5 erfolgen kann. Abbildung 3.6 zeigt auch, dass sich das Edge-basierte Refinement selbst mit einer geringen Anzahl an Samples relative schnell an der Diskontinuitätskante annähert, also ein gutes Konvergenzverhalten aufweist. Fälle, in denen das

[4]Entsteht ein stark unbalancierter Binärbaum (beispielsweise durch eine zum Teil vorsortierte Menge), ergibt sich ein Aufwand für die Einsortierung von $O(1)$ (Best-Case) bis zu linearer Zeitkomplexität $O(n)$ (Worst-Case).

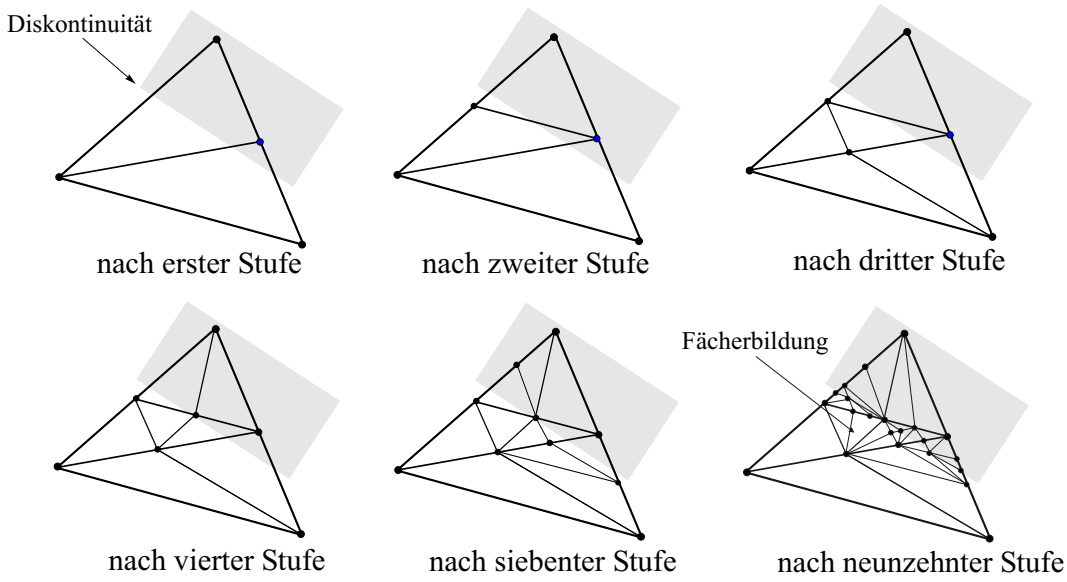


Abbildung 3.6: *Schrittweises Refinement entlang einer Diskontinuität.*

Refinement zusätzliche Parameter beziehungsweise Strategien im Hinblick auf Robustheit und Konvergenztreue benötigt, werden detaillierter in Abschnitt 3.3.2 untersucht.

Typisch für diese Refinement-Methode ist darüber hinaus die sogenannte *Fächerbildung* während der Verfeinerung (↗ Abbildung 3.6). Der Fächer orientiert sich dabei annähernd orthogonal zu den Diskontinuitäten und erzeugt entsprechend lang gezogene Strukturen. Ihre Generierung kann als kontraproduktiv im Hinblick auf eine gerichtete Verfeinerungsstruktur zur Approximation der Radiance-Funktion angesehen werden. Es entstehen lange, bezüglich der Diskontinuitäten annähernd orthogonale Interpolationskanten, die einen hohen Interpolationsfehler hervorrufen. Vor allem während der ersten Verfeinerungsschritte werden diesbezüglich deutliche Approximationsartefakte generiert, die jedoch in höheren Stufen zu regelmäßigen Strukturen mutieren. Begegnen könnte man diesem Problem beispielsweise mit einer intelligenteren Edge-Unterteilung, als in Gleichung 3.7 der Fall. Methoden zur Näherung der lokalen Diskontinuitäten werden weiter unten beschrieben.

3.3.1 Kriterien des Refinements

Die Entscheidung, ob ein neuer Sample-Wert s'_i zwischen s_i und s_j ermittelt werden muss, hängt maßgeblich davon ab, welche Kriterien beim Vergleich der Sample-Werte herangezogen werden. Wie oben bereits herausgearbeitet, stellt den Ausgangspunkt dabei die Differenz δ mit

$$\delta = |s_i - s_j| \tag{3.9}$$

dar, die gegen einen Schwellwert ϵ verglichen wird oder direkt als Entscheidungskriterium dient. δ steht im engeren Sinne für den Fehler zwischen der abzutastenden Funktion und ihrer Approximation mittels Sampling (↗ Gleichung 3.6). Die Fehlertoleranz ϵ kontrolliert dagegen direkt das Verhältnis zwischen der erreichten Rekonstruktionsqualität und der Rendering-Geschwindigkeit des gesamten Verfahrens.

Kriterien, die zur Bestimmung von δ herangezogen werden können, sind vor allem die *Intensität*, der *Kontrast* oder der *Ray-Tree* eines Samples s_i [Gla95]. Im Hinblick auf das Vertex-Tracing stehen die Intensität beziehungsweise der Ray-Tree als Kriterium im Vordergrund. Der Vergleich anhand des Kontrastes C wird in dieser Arbeit nicht näher betrachtet. C ist eine eindimensionale Größe mit $C \in \mathbb{R}^1$ und damit prädestiniert für das Refinement von Grauwertbildern.

Im Falle von n-dimensionalen Quantities eines Sample-Wertes spricht man vom Vergleich der Intensität I. Erfolgt die Darstellung des Bildes im Rechner durch die Farbkomponenten RGB folgt $n = 3$ mit $I \in \mathbb{R}^3$. Der Vergleich des Intensitätsunterschiedes zweier RGB-Farbwerte A und B kann schließlich mittels folgender *Vektornormen* mathematisch interpretiert werden:

$$L^{\infty}(A,B) = max(|A_R - B_R|, |A_G - B_G|, |A_B - B_B|), \tag{3.10}$$

$$L^{1}(A,B) = |A_R - B_R| + |A_G - B_G| + |A_B - B_B|, \tag{3.11}$$

$$L^{2}(A,B) = \sqrt{(A_R - B_R)^2 + (A_G - B_G)^2 + (A_B - B_B)^2}. \tag{3.12}$$

Gleichung 3.10 wird als *Maximumnorm*, 3.11 als *Betragssummennorm* und 3.12 als *Euklidische Norm* bezeichnet. Jede Gleichung steht für eine Interpretation der Distanz der Farbvektoren im Farbraum. Trotz des exakteren Ergebnisses der L^2-Norm wird aufgrund des akzeptableren Berechnungsaufwandes der L^1-Norm diese für den Vergleich zweier Farbwerte A und B der Samples s_i und s_j beim Vertex-Tracing herangezogen. Die Entscheidung, ob das Einfügen eines neuen Samples s'_i auf der Edge e_i notwendig ist, kann schließlich durch die Bedingung:

IF $((|A_R - B_R| + |A_G - B_G| + |A_B - B_B|) > \epsilon)$
THEN *Yes*
ELSE *No*

mit ϵ als Schwellwert getroffen werden (↗ Abbildung 3.7, Intensitätswerte). Die Anlehnung des RGB-Farbmodells an die technischen Grundlagen der Bildsynthese vereinfachen die Verarbeitung der einzelnen Farbkomponenten R, G und B durch die Möglichkeit ihres expliziten Zugriffs. Im Gegensatz zu anderen Farbmodellen wie dem HSV-Modell [Fel92] ist das RGB-Farbmodell jedoch ungeeignet, zieht man die visuelle Wahrnehmung des Menschen in Betracht. Das HSV-Modell orientiert sich vor allem an der menschlichen Farbperzeption, indem es die Komponenten *Hue* (Farbwinkel), *Saturation* (Sättigung) sowie *Value* (Helligkeit) definiert. Damit wird ein unabhängiger Vergleich von *drei* Schwellwerten realisierbar, dass eine individuelle Justierung entsprechend der visuellen Wahrnehmung erlaubt. Die Bedingung zum Einfügen eines neuen Samples s'_i auf der Edge e_i ergibt sich daraus wie folgt:

$$\begin{array}{l}\textsc{If}\ \ ((|A_H - B_H| > \epsilon_H) \vee (|A_S - B_S| > \epsilon_S) \vee (|A_V - B_V| > \epsilon_V) \\ \quad \textsc{Then}\ \textit{Yes} \\ \quad \textsc{Else}\ \textit{No}\end{array}$$

mit dem Schwellwert ϵ_H des Farbwinkels, ϵ_S der Sättigung und ϵ_V der Helligkeit. Nachteilig in der Verwendung des HSV-Modells ist allerdings die Notwendigkeit einer Konvertierung aus und in das RGB-Modell aufgrund der technischen Gegebenheiten.

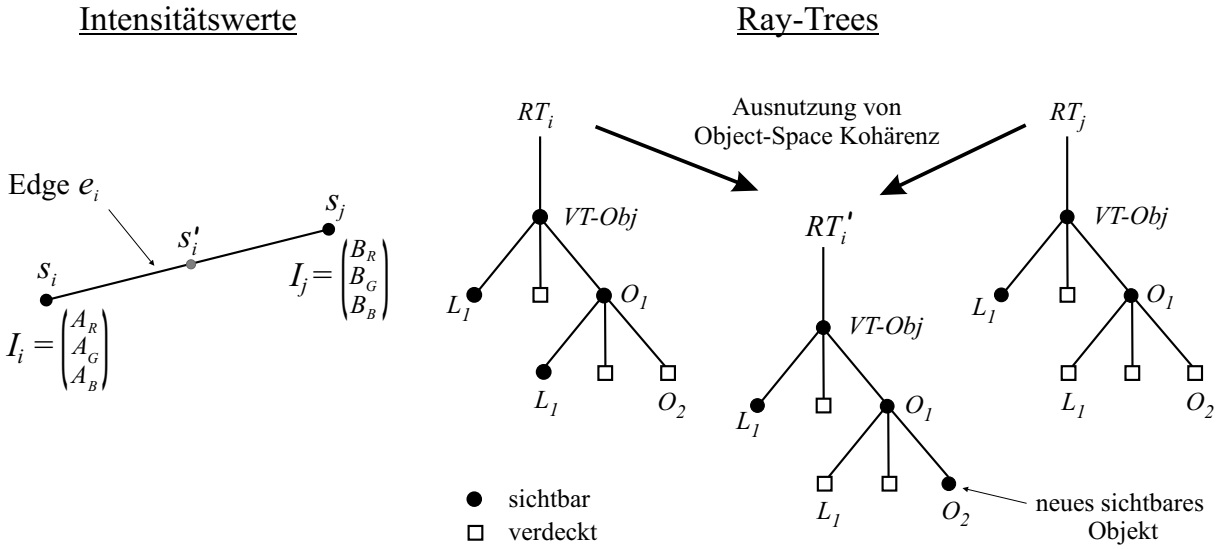


Abbildung 3.7: *Die Intensität und der Ray-Tree als Refinement-Kriterium. Der Ray-Tree kann gleichzeitig zur Ausnutzung von Object-Space Kohärenz genutzt werden.*

Die Intensität als Kriterium für ein Resampling eignet sich insbesondere zur Detektion von Nichtlinearitäten wie bei Glanzlichtern auf Objektoberflächen zu beobachten. Schwächen zeigt der Intensitätsvergleich jedoch bei der Erkennung von Diskontinuitäten wie zum Beispiel Schattenkanten. Eine bessere Lösung schafft hier die Einbeziehung von Objekt-basierten Refinement-Tests. Die Grundidee [Rot82] besteht darin, zwischen zwei Samples s_i und s_j die jeweils beteiligten Objekte „hinter“ den Samples zu untersuchen. Sind die Objekte verschieden, existieren Diskontinuitäten zwischen den Samples, so dass verfeinert werden muss. Beim Ray-Tracing eignet sich hierzu insbesondere die Verwendung des Ray-Trees [AMS91, JvW83], der eine Historie der Strahl-Objekt-Kollisionen abbildet. Der Vergleich an sich erfolgt rekursiv, beginnend mit der Wurzel des Ray-Trees RT bis zu den einzelnen Blattknoten. Knoten können dabei sichtbare Objekte beziehungsweise Lichtquellen bezüglich des Strahlenursprungs enthalten. Die Bedingung für die Unterteilung einer Edge e_i anhand des Ray-Tree-Vergleiches ergibt sich beim Vertex-Tracing demnach aus:

$$\begin{array}{l}\textsc{If}\ \ (RT_i \neq RT_j) \\ \quad \textsc{Then}\ \textit{Yes} \\ \quad \textsc{Else}\ \textit{No}.\end{array}$$

Abbildung 3.7 zeigt den Aufbau und die Charakteristik des Ray-Trees im Einzelnen. Typisch beim Vertex-Tracings sind die identischen Wurzelknoten des Baumes, da sich jeder Primärstrahl direkt aus dem entsprechende VT-Objekt definiert. Knotendifferenzen können erst durch Sekundärstrahlen auftreten und stehen als Indikatoren für eventuell vorhandene Diskontinuitäten aufgrund von Schatten-, Reflexions- oder Brechungskanten. Im Beispiel der Abbildung 3.7 unterscheiden sich RT_i und RT_j lediglich hinsichtlich der Beleuchtung des reflektierten Objektes O_1. Während O_1 in RT_j im Schatten liegt, wird es in RT_i durch L_1 beleuchtet. Die Differenz zwischen RT_i und RT_j erzwingt eine Unterteilung von e_i.

Neben der Detektion von Diskontinuitäten durch den Einsatz des Ray-Trees, kann dieser darüber hinaus auch gleichzeitig zur Ausnutzung von Object-Space Kohärenz eingesetzt werden. Die Unterteilung von e_i durch das Einfügen von s'_i in Abbildung 3.7 ruft die Bildung eines neuen Ray-Trees RT'_i hervor. Aufgrund von Kohärenz wird sein Aufbau mit hoher Wahrscheinlichkeit die Schnittmenge aus RT_i und RT_j bilden, so dass während der Berechnung von s'_i ein Zugriff auf bereits vorhandene Kollisionsinformationen erfolgen kann. Die Schnittmenge von RT_i und RT_j ist jedoch kein Garant für den Strahlengang in s'_i, wie die „unerwartete“ Sichtbarkeit des Objektes O_2 in RT'_i beweist. Sie kann lediglich als Indiz bei der Berechnung von RT'_i herangezogen werden und unterstützt insofern die Beschleunigung des Kollisionstests.

Vorausgesetzt die Ray-Trees zweier adjazenter Vertices einer Edge unterscheiden sich, ist der Einsatz des Ray-Trees ein mächtiges Werkzeug zur Detektion von Diskontinuitäten entlang einer Edge. Dennoch, zur Erkennung von Nichtlinearitäten eignet sich ausschließlich der Intensitätsvergleich, da in diesem Fall identische Ray-Trees vorliegen. Vielversprechend ist also eine Kombination beider Kriterien, um sowohl eine robuste Detektion von Nichtlinearitäten als auch Diskontinuitäten zu erzielen. Beispiele, die die Charakteristiken der einzelnen Kriterien und ihre Kombination veranschaulichen, sind Gegenstand von Abschnitt 5.1.1.

3.3.2 Konvergenz

Das adaptiv progressive Sampling der Radiance-Funktion basiert beim Vertex-Tracing auf dem Refinement eines non-uniformen Initial-Sample-Pattern, das insbesondere durch die Charakteristik der Szenengeometrie bestimmt wird. Die Ausprägung des Refinements orientiert sich dabei an der jeweils lokalen Frequenz des Ausgangssignals, mit dem Ziel, eine möglichst gute Approximation der Radiance-Funktion zu erreichen.

Gesteuert wird dieser Prozess unter Zuhilfenahme der Fehlerschranke ϵ, die jeweils mit dem Fehler δ aus Gleichung 3.9 in Beziehung gesetzt wird. Der Vergleich führt entsprechend zur Fortsetzung oder zum Abbruch des lokalen adaptiven Refinements. Das Problem bei dieser Vorgehensweise besteht jedoch darin, dass aufgrund des vereinzelten Samplings lokale Maxima beziehungsweise Minima der Radiance-Funktion, wie sie bei Nichtlinearitäten oder Diskontinuitäten auftreten, unerfasst bleiben können.

Abbildung 3.8 verdeutlicht dies an einem Beispiel. Die Radiance-Funktion $f(x)$ wird an den Stellen p_1 und p_2 abgetastet. Aus der Differenz der Funktionswerte s_1 und s_2 ergibt

sich δ als Fehlermaß für ein notwendiges Resampling von $f(x)$ im Intervall $[p_1, p_2]$. Trotz des hohen Approximationsfehlers ε_A mit

$$\varepsilon_A = max\,|f(x) - f'(x)|\,, \quad x \in [p_1, p_2]$$

folgt kein Resampling der Radiance-Funktion, da δ den Schwellwert ϵ nicht überschreitet. Problematisch ist in diesem Zusammenhang vor allem die Tatsache, dass die Größe von ε_A nicht abschätzbar ist. Ein definierter Approximationsfehler, verursacht durch das Sampling des Ausgangssignals, kann insofern nicht garantiert werden. Methoden, die sich

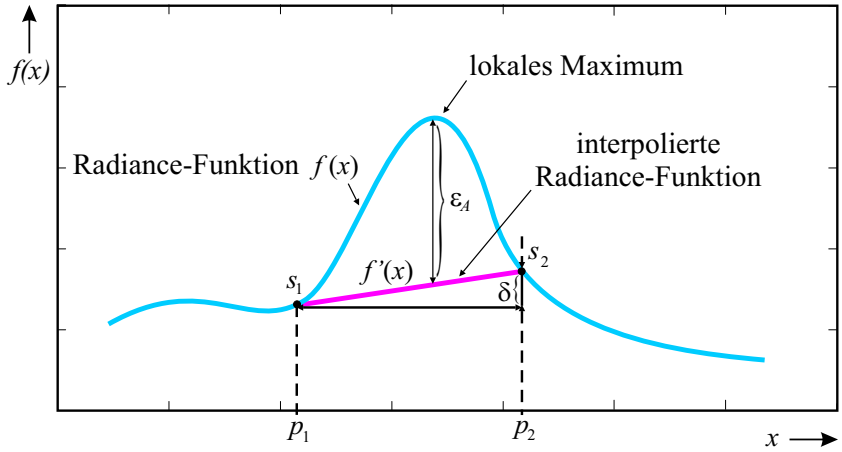


Abbildung 3.8: *Fall einer fehlerhaften Detektion des lokalen Maximums beziehungsweise Minimums beim Sampling-Prozess.*

ausschließlich mit der Lösung dieses Problems beschäftigen [Bal99], ermöglichen die Kontrolle von ε_A, sind jedoch mit zum Teil erheblichen Laufzeitkosten verbunden. Sie werden ferner in Kapitel 6 diskutiert.

Vernachlässigt man aus Kostengründen eine genauere Untersuchung hinsichtlich lokaler Maxima und Minima, hat dies Einfluss auf das Konvergenzverhalten des Refinement-Prozesses. Abbildung 3.9 demonstriert das Refinement eines Dreiecks aufgrund durchlaufender Diskontinuitäten. Betrachtet man jeweils das markierte Dreieck während der Verfeinerungsschritte, fällt auf, dass nur zwei Edges des Dreiecks weiter unterteilt werden. Die dritte Edge (die linke Edge) bleibt, trotz der vermeintlich enthaltenen Diskontinuitäten, in ihrer Länge bestehen. Sie wird nicht unterteilt, da ihre Endpunkte p_1, p_2 kein genügend großes δ aufweisen. Das lokale Maximum zwischen p_1 und p_2 bleibt folglich Abbildung 3.8 unentdeckt. Die Edges des markierten Dreiecks nähern sich schließlich nach mehreren Verfeinerungsschritten einander an, wobei ihre Längen gegen einen Wert konvergieren, der weit über dem festgelegten Minimalwert liegen kann. Das Verfahren bricht in diesem Fall nicht ab. Nochmals bemerkt sei an dieser Stelle, dass das beschriebene Konvergenzproblem nicht existieren würde, wenn es eine *zuverlässige* Detektion von Diskontinuitäten beziehungsweise Nichtlinearitäten entlang einer Edge gäbe.

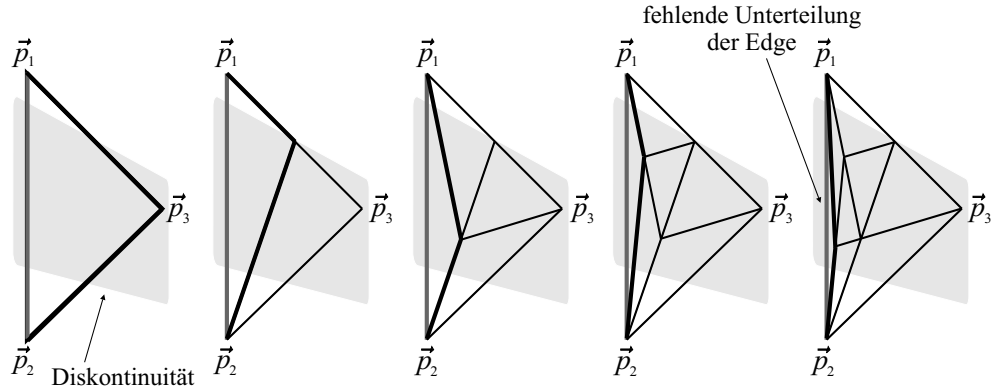


Abbildung 3.9: *Die Verfeinerung des markierten Dreiecks bricht nicht ab, da die Längen seiner Edges über dem Minimalwert bleiben. Die fehlende Unterteilung der linken Edge bewirkt dieses Konvergenzproblem.*

Begegnen kann man diesem Konvergenzproblem durch eine *Zwangsverfeinerung*, wie in Abbildung 3.10 dargestellt. Als Kriterium dient der Stauchungsfaktor ξ eines beliebigen Dreiecks, das auf eine notwendige Unterteilung untersucht werden soll. ξ errechnet sich aus dem Verhältnis der längsten Seite c eines Dreiecks, der Grundseite und der Summe der beiden übrigen Seiten a und b mit $\xi = \frac{c}{a+b}$.

Bezüglich Abbildung 3.10a sei zunächst angenommen, dass Edge e_1 aufgrund einer Diskontinuität unterteilt werden muss. Dabei entstehen gemäß Abbildung 3.10b die Sub-

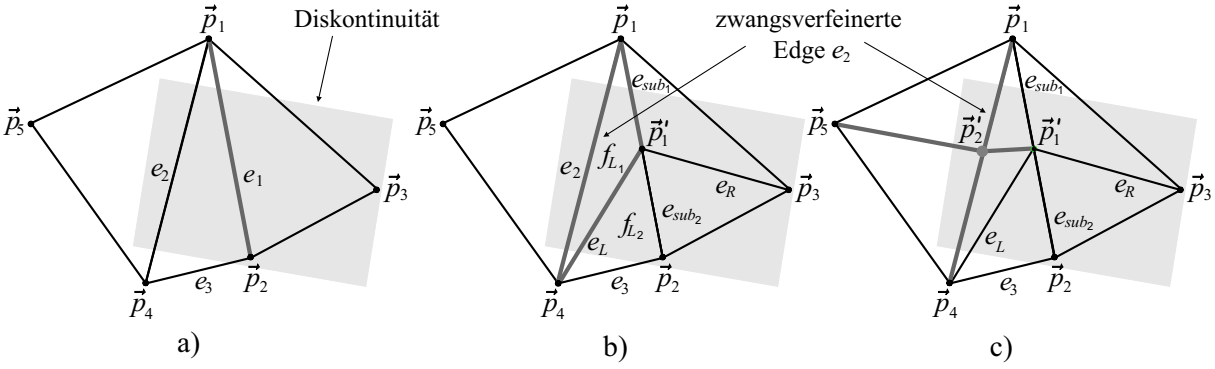


Abbildung 3.10: *Lösung des Konvergenzproblems durch Zwangsverfeinerung*

Edges $e_{sub_1}, e_{sub_2}, e_L$ und e_R. Im Anschluss erfolgt der Test auf Zwangsverfeinerung für jede neue Sub-Face, indem stets die gegenüberliegende Seite von $\vec{p_1'}$ als Grundseite angesehen wird. Für f_{L_1} ergibt sich demnach der Stauchungsfaktor ξ mit

$$\xi = \frac{e_2}{e_{sub_1} + e_L}.$$

Im Falle von $\xi > \epsilon_Z$, mit ϵ_Z dem Schwellwert der Zwangsverfeinerung, muss e_2 in jedem Fall unterteilt werden. Eine Verfeinerung entsprechend Abbildung 3.10c durch das Einfügen von $\vec{p_2'}$ resultiert aus dieser Vorgehensweise.

Außer zur Verhinderung von Konvergenzproblemen, die sich durch nicht detektierte Maxima oder Minima der Radiance-Funktion ergeben, kann eine Analyse des Stauchungsfaktors auch bei Abbruchproblemen aufgrund von numerischer Ungenauigkeit erfolgreich eingesetzt werden. Betrachtet man dazu die zu unterteilende Edge e_1 am Beispiel der Abbildung 3.10a, so besitzt sie jeweils eine rechte und linke adjazente Face. Bei einem Flächeninhalt A einer beliebigen Face mit $A \to 0$ führt das Einfügen neuer Edges nicht zum Abbruch, da die Minimallänge einer Edge aufgrund von Rundungsfehlern nicht erreicht werden kann. Das Verfahren verliert sich in einer endlosen Verfeinerungsrekursion.

Vermeidbar ist dieses Problem durch die Analyse von A mit $A > 0$ unter Einbeziehung des Stauchungsfaktors. Hierzu muss am Beispiel des linken Dreiecks der Abbildung 3.10a zunächst die längste Seite aus e_1, e_2 und e_3 bestimmt werden. Mit e_2 als die Längste ergibt sich der Stauchungsfaktor

$$\xi = \frac{e_2}{e_1 + e_3}.$$

Für den Fall $\xi = 1$ würde kein Flächeninhalt des betrachteten Dreiecks existieren. Eine Verfeinerung dieses Dreiecks wäre nicht notwendig, das Verfahren würde an dieser Stelle abbrechen.

Das Konvergenzverhalten des Edge-basierten Refinement-Prozesses kann durch die Methode der Zwangsverfeinerung und durch Abbruch bei zu kleinem Flächeninhalt als äußerst robust eingestuft werden. Darüber hinaus hilft jedoch insbesondere die Zwangsverfeinerung, auch nicht detektierte Diskontinuitäten beziehungsweise Nichtlinearitäten zu finden. Entstehen während des Refinement-Prozesses überproportional stark gestauchte Dreiecke ist das ein Hinweis auf nicht detektierte Features. Die Zwangsverfeinerung unterstützt also zum Großteil die konsistente Rekonstruktion der Radiance-Funktion, was sich hinsichtlich der Bildqualität positiv widerspiegelt. Ein Beispiel dafür demonstriert Abbildung 5.3 aus Abschnitt 5.1.

3.4 Optimierung und Beschleunigung des Samplings

Den Kern des Vertex-Tracers bildet ein adaptiv progressiver Ray-Tracer, dessen Ziel es ist, möglichst wenige Samples zur Rekonstruktion der Radiance-Funktion abzutasten. Je nach Charakteristik des abzutastenden Bildes kann damit die Anzahl der erforderlichen Samples bis auf 5 % und weniger im Vergleich zum traditionellen Ray-Tracing reduziert werden. Die Menge an Samples und die anfallenden Kosten zur Bestimmung eines jeden Einzelnen stellen die Schlüsselkomponenten hinsichtlich Effizienz und Laufzeitverhalten des Vertex-Tracings dar.

Diesbezüglich soll nun im Weiteren näher auf Konzepte und Methoden eingegangen werden, die insbesondere zur Beschleunigung des Verfahrens dienen. Ein Schwerpunkt liegt dabei auf der intelligenten Wahl tatsächlich relevanter Samples zur Bildrekonstruktion. Da das Refinement beim Vertex-Tracing im Objektraum geschieht und jeder Sample

im $\mathbb{R}^2$ von einem Vertex im $\mathbb{R}^3$ resultiert (↗ Abschnitt 3.3), impliziert das Verfahren im Gegensatz zum Standard-Ray-Tracer keinen Visibility-Test. Ein Test auf Sichtbarkeit für jeden verwendeten Vertex sollte also erfolgen, um unnötige Berechnungen zu vermeiden. Zu diesem Zweck werden im Vertex-Tracing verschiedene Methoden von Visibility-Tests angewendet, die in unterschiedlichen Stufen des Rendering-Prozesses eingreifen. Eine Betrachtung der einzelnen Tests findet in diesem Abschnitt statt, wobei zwischen

- dem Visibility-Test anhand eines ID-Buffers im Pre-Prozess,
- dem Visibility-Test anhand des Z-Buffers während des Refinements und
- dem Visibility-Test bezüglich des Viewports ebenfalls während des Refinements

differenziert wird.

Wurde die Anzahl der Samples schließlich auf ein Minimum reduziert, ist das Laufzeitverhalten des Vertex-Tracers stark von den Kosten zur Bestimmung *eines* Samples abhängig. Die Bedeutung eines effizienten Strahl-Schnittpunkttests sowie die Möglichkeit, temporale Kohärenzen auszunutzen, wird Gegenstand der Betrachtungen im zweiten Teil dieses Abschnittes sein.

3.4.1 Visibility-Test mittels ID-Buffer

Die Anzahl der zu bestimmenden Samples ist zum einen abhängig von der Dichte des Initial-Sample-Pattern und zum anderen von der Summe der Samples, die nachträglich während des Refinement-Prozesses erzeugt werden. Betrachtet man lediglich das Initial-Sample-Pattern ergibt sich die Menge der Initial-Samples aus der Menge aller sichtbaren Primär-Vertices von allen sichtbaren VT-Objekten. Wie oben bereits erwähnt, existiert jedoch beim Vertex-Tracing kein implizierter Visibility-Test. Während beim traditionellen Ray-Tracing dieser automatisch erfolgt, indem jeder Primärstrahl auf Kollision mit einem Szenenobjekt getestet wird, findet beim Vertex-Tracing eine direkte Zuweisung zwischen Primärstrahl und Vertex statt (↗ Abschnitt 3.1). Es wird kein Visibility-Test durchgeführt.

Trotz der Tatsache, dass das Vertex-Tracing auch absolut ohne Visibility-Test auskommen würde, ist jedoch vor allem bei höherer Szenenkomplexität ein derartiger Test äußerst sinnvoll, um unnötiges Sampling zu vermeiden. Die Praxis zeigt sogar, dass gerade der Visibility-Test als integraler Bestandteil des Vertex-Tracing-Algorithmus angesehen werden muss, um ein effizientes Rendering zu erlauben.

Der in diesem Abschnitt beschriebene Visibility-Test, der sogenannte ID-Buffer-Test, ist der Erste von drei Tests. Er findet nach jeder Kameraänderung als Pre-Prozess des eigentlichen Vertex-Tracings statt und zielt auf die Selektion ausschließlich sichtbarer Primär-Vertices. Die Dichte des Initial-Sample-Pattern wird durch ihn bestimmt. Der Test benutzt das Prinzip des ID-Buffers [WG84] in einer modifizierten Form, wie in Abbildung 3.11 schematisch dargestellt. Jede Face f_i eines VT-Objektes wird mit einer eindeutigen Farb-ID versehen und in den ID-Buffer gerendert. Objekte, die keine VT-Objekte sind (alle übrigen OpenGL-Objekte), erhalten die Farb-ID 0, ebenso der Hintergrund des Buffers.

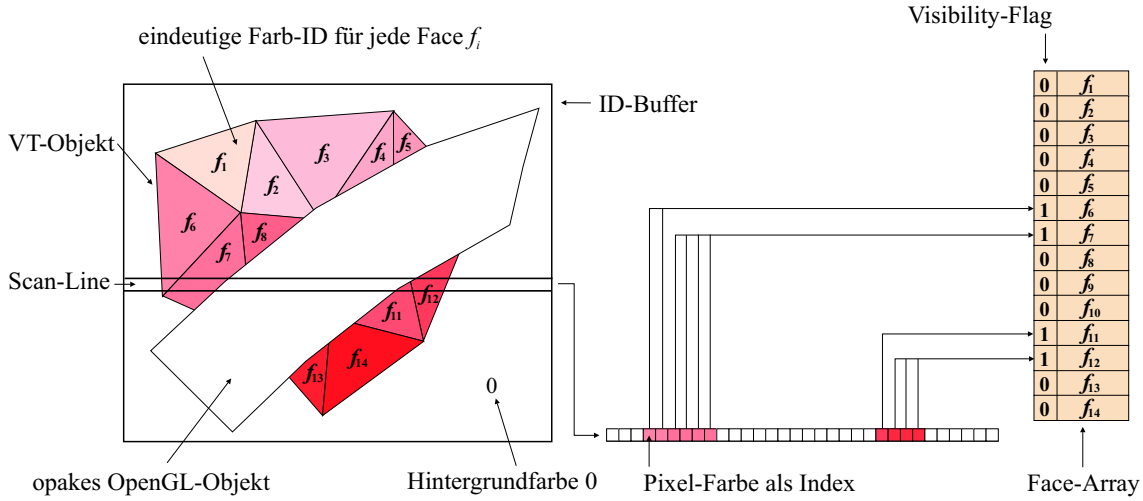


Abbildung 3.11: *ID-Buffer-Test. Die Farb-ID eines Pixels einer Scan-Line dient der Indizierung in einem Face-Array zum Setzen des Visibility-Flags.*

Nach dem Einlesen des ID-Buffers in den Hauptspeicher fungiert jedes Pixel als Index für die Indexierung auf ein zusammenhängendes *Array* von Faces aller VT-Objekte mit auf 0 gesetztem Visibility-Flag. Scan-Line für Scan-Line und Pixel für Pixel wird nun der ID-Buffer abgearbeitet und das Visibility-Flag der Face, auf das der Index zeigt, auf 1 gesetzt. Alle Faces, die mindestens durch ein Pixel sichtbar sind, werden schließlich als *visible* markiert.

Nachdem die Sichtbarkeit aller Faces bestimmt wurde, muss nun noch Rückschluss auf die Sichtbarkeit der einzelnen Primär-Vertices beziehungsweise Primär-Edges gezogen werden. Die Vorgehensweise dazu veranschaulicht der Algorithmus in Abbildung 3.12. Im

Visibility-Test(ID-Buffer)

```
1  Visibility-Test_ID-Buffer(ID-Buffer)
2  For all VT-Objekte O_i
3      For all Edges e_i
4          If (Visible(f_R^{e_i}) OR Visible(f_L^{e_i}))
5              If (!Raytraced(v_1^{e_i})) ⇒ RayTrace(v_1^{e_i})
6              If (!Raytraced(v_2^{e_i})) ⇒ RayTrace(v_2^{e_i})
7              EdgeList ← SortEdge(e_i)
```

Abbildung 3.12: *Vollständiger Algorithmus des Visibility-Tests mittels ID-Buffer. Der Test erfolgt als Pre-Prozess nach jeder Kameraänderung.*

Anschluss an den ID-Buffer-Test (Zeile 1) gemäß Abbildung 3.11 wird für jede Edge e_i eines jeden VT-Objektes O_i geprüft, welche der beiden adjazenten Faces $f_R^{e_i}$ und $f_L^{e_i}$ (linke und rechte Face) sichtbar sind (Zeile 4). Im Falle, dass mindestens eine der beide Faces sichtbar ist, muss auch e_i sichtbar sein. Die Vertices $\vec{v}_1^{e_i}$ und $\vec{v}_2^{e_i}$ werden dem Ray-Tracing unterzogen, sollte dies noch nicht bereits geschehen sein. Mit dem Einfügen der gefundenen Edge e_i anhand ihrer Länge in eine sortierte Edge-List (Zeile 7) wird die Bearbeitung der Edge abgeschlossen und mit der Nächsten fortgefahren.

Die resultierende Edge-List dient als Ausgangsbasis des gesamten Refinement-Prozesses, indem, beginnend mit der längsten Edge (die Oberste der Edge-List), Edge für Edge auf eine notwendige Verfeinerung getestet wird. In der Edge-List sind dabei alle Edges enthalten, die eine sichtbare Face begrenzen, auch wenn die zugehörigen Edges der Face selbst verdeckt bleiben. Die nötige Robustheit des Testes ist damit gegeben.

Hinsichtlich des Laufzeitverhaltens sei erwähnt, dass das Auslesen des ID-Buffers die wesentlichste Komponente darstellt. Bei moderner Graphik-Hardware liegen die Lesegeschwindigkeiten der Graphik-Buffer bei einer Auflösung von 1024x768 und 32 Bit Farbtiefe im Bereich von 20-40 ms [MHE01], so dass hier ein brauchbarer Wert existiert. Hinzu kommt ein eher vernachlässigbarer Wert für das Setzen der Visibility-Flags und für das Rendering der Szene in den ID-Buffer. Letzteres fällt erst bei höherer Szenenkomplexität stärker ins Gewicht, da der Visibility-Test schließlich ein Zwei-Pass-Rendering erfordert (↗ Abschnitt 3.6). Eine detaillierte Darstellung, wie sich der ID-Buffer-Test in den Rendering-Prozess des Vertex-Tracings integriert und welche einzelnen Schritte für den Test durchzuführen sind, erfolgt in Abschnitt 3.5 anhand der Abbildung 3.17.

3.4.2 Einsatz des Z-Buffers

Der Visibility-Test unter Nutzung des Z-Buffers findet während des Refinement-Prozesses statt. Analog zum oben beschriebenen ID-Buffer-Test soll für jede in die Edge-List neu einzufügende Edge e_i bestimmt werden, ob e_i sichtbar ist oder nicht. Im Gegensatz zum ID-Buffer-Test, bei dem unter Analyse der Faces auf die Sichtbarkeit der Edges rückgeschlossen wurde, erfolgt bei der Verwendung des Z-Buffers eine direkte Untersuchung der zugehörigen Vertices. Aus ihrer Sichtbarkeit kann, wenn auch nicht eindeutig, die Sichtbarkeit der Edge bestimmt werden. Ist mindestens ein Vertex sichtbar, so ist die Edge e_i wenigstens partiell sichtbar. Sind beide Vertices verdeckt, bedeutet dies jedoch keine vollständige Verdeckung der Edge, wie im Detail weiter unten beschrieben wird.

Das Prinzip des Z-Buffer-Tests zeigt Abbildung 3.13. Identisch zum ID-Buffer-Test entsteht der Z-Buffer im ersten Rendering-Pass, der als Pre-Prozess nach jeder Kameraänderung ausgeführt wird. Für jede neue, aus dem Refinement hervorgegangene Edge e_i folgt der Test der Vertices $\vec{v}_1^{e_i}$ und $\vec{v}_2^{e_i}$ auf Sichtbarkeit. Die hierzu nötige z-Komponente ergibt sich dabei aus der Projektion $\mathbf{M}_P$ des Vertex $\vec{v}_i^{e_i}$ (↗ Abschnitt 3.3) und wird mit dem entsprechenden Eintrag im Z-Buffer verglichen. Ist der Z-Wert im Buffer größer, so ist der Vertex sichtbar[5]. Ist mindestens ein Vertex sichtbar, wird e_i in die Edge-List einsortiert und steht für das weitere Refinement zur Verfügung (↗ Abbildung 3.14). Im Falle,

[5]Vorausgesetzt, die Werte im Z-Buffer steigen mit der Entfernung zur Betrachterposition.

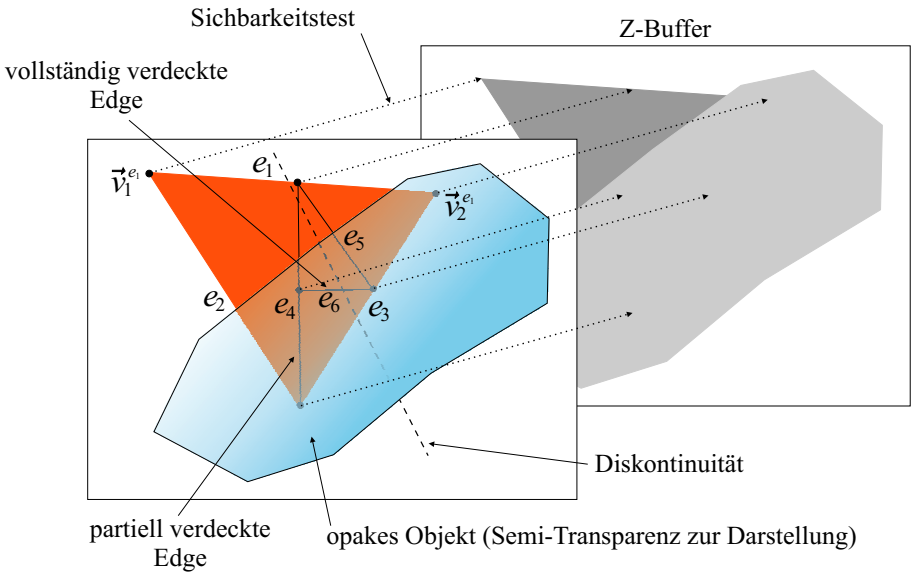


Abbildung 3.13: *Z-Buffer-Test. Der Test erfolgt während des Refinement-Prozesses für jede neu erzeugte Edge e_i.*

dass $\vec{v}_1^{e_i}$ und $\vec{v}_2^{e_i}$ verdeckt sind, folgt die Verwerfung der Edge e_i, so dass keine weitere Verfeinerung an dieser Stelle stattfinden kann.

Der Z-Buffer-Test erlaubt im Anschluss an die Vorauswahl durch den ID-Buffer-Test eine weitere Reduzierung an Samples in Form von Edges. Edges, die während des Refinement-Prozesses neu entstehen und sich als nicht sichtbar erweisen, werden für nachfolgende Verfeinerungsschritte nicht weiter in Betracht gezogen. Vor allem bei Faces, die

Visibility-Test_Z-Buffer(Z-Buffer, e_i)

1 If (Visible($v_1^{e_i}$) OR Visible($v_2^{e_i}$))
2 EdgeList $\leftarrow$ SortEdge(e_i)

Abbildung 3.14: *Algorithmus des Visibility-Tests mittels Z-Buffer. Eine Edge ist sichtbar und wird in die Edge-List einsortiert, solange mindestens einer ihrer Vertices sichtbar ist.*

trotz großer Verdeckungen den ID-Buffer-Test aufgrund weniger sichtbarer Pixel überstanden haben, macht sich die Selektion durch den Z-Buffer-Test unter Umständen besonders stark bemerkbar. Befinden sich im verdeckten Bereich der Faces Features wie Diskontinuitäten, würden diese ohne Test bis zum Abbruch verfeinert werden. In diesem Fall trägt der Z-Buffer-Test signifikant zur Effizienzverbesserung des Vertex-Tracings bei (↗

Kapitel 5).

Problematisch am Z-Buffer-Test ist jedoch, wie bereits erwähnt, die eindeutige Bestimmung der Sichtbarkeit einer Edge. Abbildung 3.15 zeigt anhand der Edge e_4, dass aus der Analyse ihrer Vertices $\vec{v}_2$ und $\vec{v}_4$ keine zuverlässige Aussage über die Sichtbarkeit von e_4 getroffen werden kann. Während $\vec{v}_2$ und $\vec{v}_4$ selbst verdeckt sind, ist e_4 partiell sichtbar. Der Verlauf der Diskontinuität bleibt unerkannt, da für e_4 keine weitere Verfeinerung erfolgt. Nach Algorithmus (Abbildung) 3.14 wird e_4 bereits unmittelbar nach ihrer Entstehung als nicht sichtbar eingestuft und dementsprechend auch nicht in die Edge-List eingefügt. Eine Ausnahme bilden die Edges e_1, e_2 und e_3. Sie sind Primär-Edges, so dass

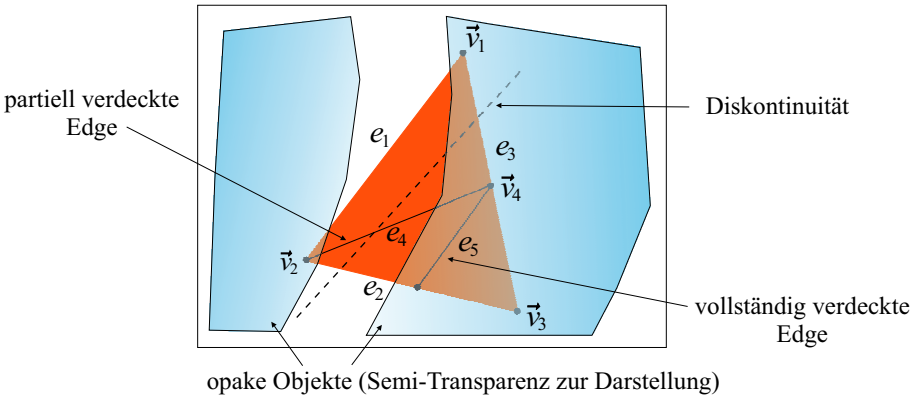


Abbildung 3.15: *Problemfall beim Z-Buffer-Test. Aus der Analyse der Vertices einer Edge kann nicht eindeutig auf die Sichtbarkeit der Edge rückgeschlossen werden.*

ihr Visibility-Test mittels ID-Buffer erfolgt. Weil dabei nur die Sichtbarkeit der Face zählt, werden e_1, e_2 und e_3 ebenfalls als sichtbar eingestuft und für ein weiteres Refinement in Betracht gezogen.

Die Lösung der eindeutigen Sichtbarkeitsbestimmung könnte durch ein erneutes Rendering der generierten Edge beziehungsweise Sub-Faces erfolgen. Während ein Sichtbarkeitstest der Sub-Faces in Anlehnung zum ID-Buffer denkbar wäre, würde sich die Analyse der Edge jedoch weitaus schwieriger gestalten. Hier müsste ein Pixel-für-Pixel-Test entlang der Edge durchgeführt werden, um eine eindeutige Aussage über die Sichtbarkeit der Edge zu erhalten. Diese Vorgehensweise würde jedoch ein mehrfaches Einlesen der Graphik-Buffer bedeuten und ist insofern nur ineffizient realisierbar. Da darüber hinaus der Z-Buffer-Test nur als Ergänzung zum ID-Buffer-Test fungiert, soll an dieser Stelle der marginale Qualitätsverlust zum Vorteil des Laufzeitverhaltens in Kauf genommen werden. Eine weitere Betrachtung des Themas ist damit in Kapitel 7.1 einzuordnen.

3.4.3 Visibility-Test durch Clipping

Der dritte Visibility-Test, der im Rahmen des Vertex-Tracings durchgeführt wird, untersucht Edges auf Sichtbarkeit bezüglich des *Viewports*. An der Seite des Z-Buffer-Tests

entscheidet der *Viewport-Test* ebenfalls während des Refinement-Prozesses, ob eine neue Sekundär-Edge e_i im Bildausschnitt sichtbar ist oder nicht. Besteht e_i den Test, erfolgt die Einsortierung der Edge in Edge-List (↗ Abbildung 3.16), so dass sie weiteren Betrachtungen unterzogen werden kann. Liegt e_i jedoch völlig außerhalb des Viewports wird sie verworfen. Der Refinement-Prozess beschränkt sich damit auf das Innere des sichtbaren Bildausschnittes.

Viewport-Test(Viewport, e_i **)**

```
1       visible = CS_Clipping(e_i)
2       If (visible)
3           EdgeList ← SortEdge(e_i)
```

Abbildung 3.16: *Viewport-Test. Die Bestimmung der Sichtbarkeit einer Edge geschieht mittels leicht modifiziertem Clipping-Algorithmus von* Cohen *und* Sutherland.

Vorgenommen wird der Viewport-Test durch *Clipping* der Edge e_i an den Grenzen des Viewport-Windows, um auf eine tatsächliche Sichtbarkeit von e_i rückschließen zu können. Der Einsatz des Clipping-Algorithmus von Cohen und Sutherland [BG89, FvDFH90] ermöglicht in diesem Zusammenhang eine effiziente Bestimmung. Er entscheidet, ob eine Edge völlig innerhalb oder außerhalb des Viewports liegt oder die Seiten des Viewports schneidet und ein entsprechendes Clipping erfordert. Der Clipping-Algorithmus wurde für den Visibility-Test leicht modifiziert. Der Abbruch erfolgt unmittelbar nach der Bestimmung der Sichtbarkeit von e_i (siehe Zeile 1 in Abbildung 3.16), ohne ein Clipping der Edge an den Viewport-Grenzen abzuschließen, wie es der Basisalgorithmus eigentlich vorsieht.

3.4.4 Kollisionsdetektion

Das Thema der Kollisionsdetektion spielt nicht nur beim Standard-Ray-Tracing eine entscheidende Rolle. Auch die Effizienz des Vertex-Tracings wird letztendlich signifikant vom Laufzeitverhalten eines jeden einzelnen Schnittpunkttests bestimmt. Trotz erheblicher Strahlenreduktion, die das Verfahren Vertex-Tracing charakterisiert, verbleiben noch immer eine Vielzahl durchzuführender Strahlen-Schnittpunkttests, deren Kosten es zu minimieren gilt.

Wie im Abschnitt 1.2 bereits erwähnt, ist die Zeitkomplexität des Basis-Ray-Tracings sub-linear. Sie liegt bei heuristischen Strahlen-Schnittpunkt-Verfahren zwischen $O(1)$ und $O(n)$ [SKM98]. Bestimmt wird die durchschnittliche Zeitkomplexität vor allem durch die Art der verwendeten Datenstruktur. Nur im Falle von nahezu statischen Datenstrukturen, wie sie bei klassischen Raumpartitionierungsverfahren entstehen, ist im *Average-Case* ein Suche im sub-linearen Bereich möglich. Je höher der Anteil an Dynamik in der Szene wird,

desto mehr muss auf dynamische Strukturen zurückgegriffen werden. Ein zyklisches Update der Datenstruktur wird notwendig, das je nach Grad der Dynamik das Suchproblem bis auf eine *sub-quadratische* Zeitkomplexität anheben kann. Die Wahl der Beschleunigungsstruktur korreliert insofern direkt mit dem Laufzeitverhalten des Verfahrens und richtet sich nach den Anforderungen aus der Praxis.

Das Laufzeitverhalten des Vertex-Tracings wird im Wesentlichen bestimmt durch:

- das Refinement,
- die Anzahl der zu berechnenden Strahlen und
- die Effizienz eines einzelnen Strahlen-Schnittpunkttestes.

Geht man davon aus, das sich der Aufwand des Refinements durch die erfolgreiche Reduzierung der Strahlenanzahl mehr als rentiert, verbleibt die Effizienz des Schnittpunkttests als signifikante Größe. Ihre Optimierung entscheidet nicht unwesentlich über das gesamte Laufzeitverhalten des Vertex-Tracings. Eine Implementierung effizienter Datenstrukturen unter Berücksichtigung modernster Hardware-Architekturen können hier enorme Laufzeitverbesserungen hervorrufen.

Die Betrachtung der Thematik Kollisionsdetektion wurde jedoch an dieser Stelle nicht weiter forciert, da der Schwerpunkt der hier vorgestellten Arbeit auf einem Verfahren zur Strahlenreduktion liegt. Das Vertex-Tracing wurde vorerst mit einer OOB-basierten Kollisionsdetektion ausgestattet, die sich für Virtual Reality Anwendungen eignet. Eine effizientere Implementierung ist in diesem Zusammenhang denkbar und wird Schwerpunkt des Abschnitts 6.2.2 sein.

3.4.5 Temporale Kohärenz

Einen weiteren Punkt im Hinblick auf Optimierung und Beschleunigung des Vertex-Tracings stellt die Ausnutzung temporaler Kohärenz dar. Gemeint ist damit, dass ein Teil der erfolgten Berechnung von den darauf folgenden Frames wiederverwendet wird. Im Falle des Vertex-Tracings bieten sich dafür insbesondere die Primär-Vertices an. Aufgrund des geometriebasierten Samplings kann in ihnen relativ einfach das Ergebnis der bereits auf ihrer Basis geschossenen Schattenfühler abgespeichert werden. Wurde die Beleuchtung für einen beliebigen Primär-Vertex einmal bestimmt, ist ein Zugriff auf diese Information im nachfolgenden Frame möglich, solange sich die entsprechenden Lichtquellen in ihrer Position nicht verändern.

Positive Auswirkungen besitzt diese Methode vor allem auf das Laufzeitverhalten während der Preview-Berechnung. Eine Anwendung des Verfahrens auch auf Sekundär-Vertices wurde jedoch hier nicht weiter verfolgt. Die Einführung spezieller Datenstrukturen zur Speicherung direkter Beleuchtungsinformationen innerhalb einer Face wäre dafür notwendig.

3.5 Rekonstruktion durch Hardware-Rendering

Nachdem in den vorangegangenen Abschnitten die Themen Sampling und Refinement sowie Verfahren zur Optimierung stärker im Vordergrund standen, soll nun die Rekonstruktion der Radiance-Funktion in Form des eigentlichen Rendering-Prozesses näher beleuchtet werden. Hierzu zählen vor allem Fragestellungen wie: Welche Methoden und Konzepte sind notwendig, um das Verfahren Vertex-Tracing als hybrides Rendering optimal abzubilden? Aber auch Fragen, in Bezug auf Shading, verwendete Shading-Modelle und damit eng verbunden der Einsatz von Texturen sind in diesem Kontext zu beantworten.

Wie bereits erwähnt, dient zur Rekonstruktion das Hardware-basierte Gouraud-Shading, das für eine lineare Interpolation zwischen den Sample-Werten dient. Die vom Refinement generierten Faces werden dem Hardware-Rendering übergeben und entsprechend dargestellt. Das Vertex-Tracing zeichnet sich darüber hinaus durch seine Eigenschaft eines hybriden Renderings aus. Das heißt, sowohl ein Rendering auf Grundlage lokaler Beleuchtungsmodelle in Form von OpenGL-Hardware-Rendering als auch auf Grundlage eines globalen Beleuchtungsmodells in Form des Vertex-Tracings sind synchron realisierbar. Kerngedanke in diesem Zusammenhang ist die Differenzierung der Szenenobjekte. Spekular reflektierende Objekte werden als VT-Objekte (Vertex-Tracing-Objekte) definiert und dem Vertex-Tracing unterzogen. Objekte, die vorwiegend diffus reflektieren bleiben jedoch „normale“ Szenenobjekte (Nicht-VT-Objekte)[6] und werden direkt dem herkömmlichen OpenGL-Rendering zugewiesen.

Im engeren Sinne müsste an dieser Stelle von einem *hybriden Shading* gesprochen werden, da letztlich die Darstellung sowohl für VT-Objekte als auch für Nicht-VT-Objekte gleichsam durch Rasterisierungs-Hardware erfolgt (↗ Abschnitt 3.1). Die eigentliche Unterscheidung findet in der Bestimmung der Farbintensität, dem Shading, auf dem Polygon-Vertex statt. Hier wird das entsprechende Beleuchtungsmodell herangezogen: das lokale beim OpenGL-Hardware-Rendering, das globale beim Vertex-Tracing. Da das Shading jedoch als integraler Bestandteil des Renderings betrachtet werden kann, wird in dieser Arbeit die Bezeichnung *hybrides Rendering* forciert.

Abbildung 3.17 zeigt den gesamten Rendering-Prozess des Vertex-Tracings, der als Zwei-Pass-Rendering fungiert. Ein neuer Rendering-Zyklus beginnt mit jeder Kameraänderung und veranlasst im ersten Rendering-Pass zuerst das Löschen aller Buffer. Im Anschluss daran findet der Visibility-Test mittels ID-Buffer statt. Im Rahmen dieses Tests erfolgt nach der Initialisierung aller Faces aller VT-Objekte zunächst ein Rendering der gesamten Szene abzüglich aller VT-Objekte in den ID-Buffer. Die Farb-ID eines jeden Nicht-VT-Objektes wird dabei auf „0“ gesetzt, so dass lediglich der Z-Buffer effektiv beschrieben wird. Mit einer eindeutigen Farb-ID > 0 für jede Face eines VT-Objektes folgt anschließend das Rendering der VT-Objekte. Der ID-Buffer sowie der Z-Buffer werden gelesen und ermöglichen ein Setzen der Visibility-Flags (↗ Abschnitt 3.4.1). Das Löschen des ID-Buffers (Frame-Buffers) und des Z-Buffers beendet den Visibility-Test und schafft

[6] Natürlich können auch diffuse Objekte als VT-Objekte definiert werden. Dies ist jedoch nur sinnvoll, wenn andere globale Beleuchtungsphänomene wie beispielsweise Schatten auf den Objekten dargestellt werden sollen.

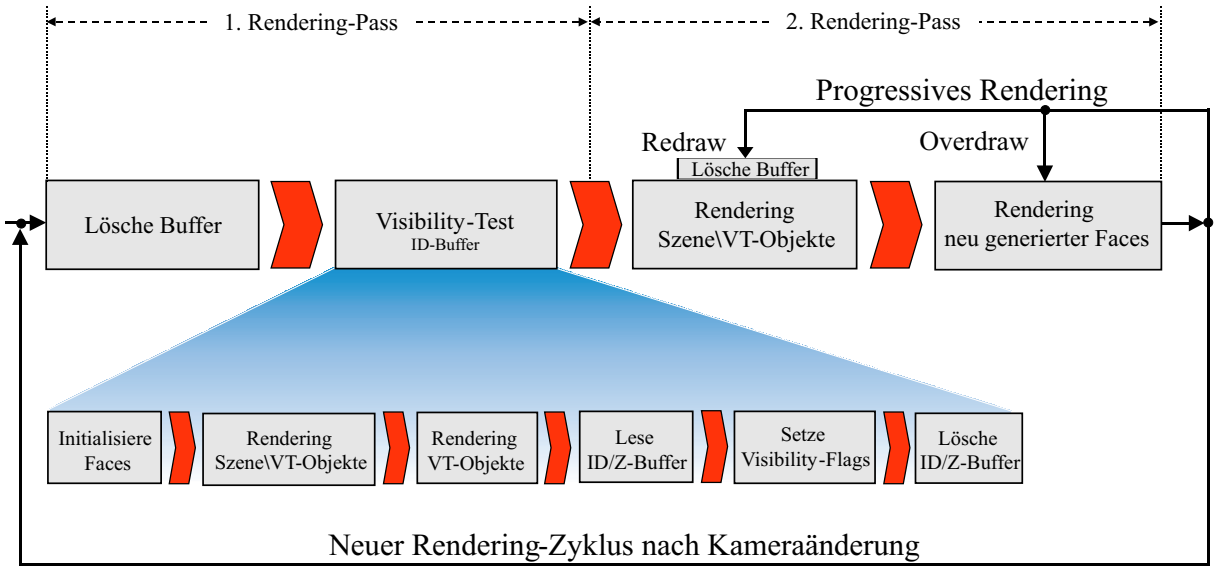


Abbildung 3.17: *Rendering-Prozess des Vertex-Tracings.*

die Voraussetzung für den zweiten Rendering-Pass.

Der zweite Pass beginnt mit einem erneuten Rendering aller Nicht-VT-Objekte in den Frame-Buffer, jedoch jetzt unverändert für eine finale Darstellung der Szene. Im Anschluss daran startet der Refinement-Prozess des Vertex-Tracings. Je nach Rekursionstiefe des Refinements entstehen hier eine Vielzahl neu generierter Faces. Ihr Rendering geschieht in der letzten Stufe des zweiten Rendering-Pass, indem jede Face der OpenGL-Hardware übergeben wird. Die Darstellung aller generierten Faces komplettiert letztendlich die abzubildende Szene des aktuellen Rendering-Zyklus.

3.5.1 Progressives Rendering

Kern des Vertex-Tracings ist ein adaptiv progressives Refinement zur Approximation der Radiance-Funktion (↗ Abschnitt 3.2). Progressiv bedeutet in diesem Zusammenhang, dass die Annäherung der Funktion Schritt für Schritt geschieht, bis das gewünschte Ergebnis erreicht wird. Ziel ist es, ein beliebiges Zwischenergebnis nach einer bestimmten Anzahl von Refinement-Stufen darzustellen. Mit Hilfe visueller Updates soll ein sich kontinuierlich verbessernder Eindruck des zu generierenden Finalbildes hervorgerufen werden.

Die Einführung eines progressiven Renderings während des zweiten Rendering-Pass erlaubt eine derart sukzessive Darstellung. Hierzu werden nach einer beliebigen Anzahl k von Refinement-Stufen die neu generierten Faces aus dem Refinement-Prozess dem Hardware-Rendering übergeben. Unterschieden wird dabei zwischen einem sogenannten *Overdraw* und *Redraw*, wie in Abbildung 3.17 dargestellt. Beim Redraw erfolgt während eines Updates ein kompletter Neuaufbau des Frame-Buffers. Dazu muss zuerst der Frame-Buffer gelöscht werden, bevor ein erneutes Rendering aller Nicht-VT-Objekten durch-

geführt wird. Daran anschließend erfolgt die Darstellung der genierten Faces. Wie in Abbildung 3.18 demonstriert, werden alle unterteilten Parent-Faces durch ihre Child-Faces substituiert. Nur die Child-Faces werden gerendert. Zuzüglich müssen jedoch alle nicht unterteilten Parent-Faces ebenfalls bei jedem Update dem Rendering übergeben werden, um eine kontinuierliche Darstellung des entsprechenden VT-Objektes zu gewährleisten.

Beim Overdraw verhält sich die Vorgehensweise anders. Hier werden bei jedem Update lediglich die neu hinzugekommenen Faces gerendert. Der Inhalt des Frame-Buffers sowie des Z-Buffers bleibt vollständig bestehen, so dass ein „Overdraw“ (Überschreiben) der neu generierten Faces entsteht. Problem dabei ist, dass die Tiefe der Child-Faces gleich der Tiefe der Parent-Faces ist, da die Child-Faces die Parent-Faces lediglich substituieren. Es entsteht ein sogenanntes *Z-Fighting*[7] mit unerwünschten Artefakten im Resultat.

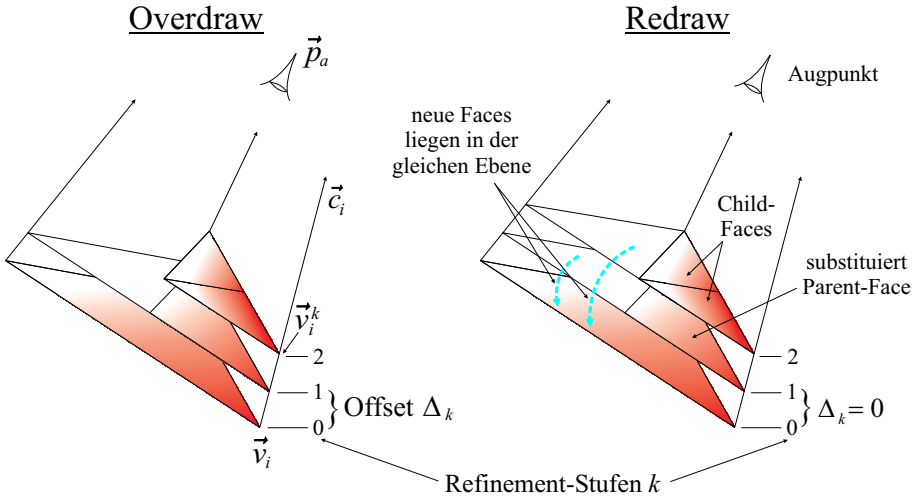


Abbildung 3.18: *Progressives Rendering mittels Overdraw und Redraw.*

Die Lösung zeigt Abbildung 3.18 mit der Einführung des Offsets Δ_k zwischen den einzelnen Refinement-Stufen k im $\mathbb{R}^3$. Damit sich Δ_k relativ zur Kameraposition $\vec{p}_a$ verhält, soll die Distanz d vom Ausgangs-Vertex $\vec{v}_i$ zur Kamera mit

$$d \;=\; \|\vec{v}_i - \vec{p}_a\| \tag{3.13}$$

einbezogen werden. Unter Berücksichtigung eines konstanten Faktors f ermittelt sich Δ_k aus

$$\Delta_k \;=\; d \cdot f \cdot k. \tag{3.14}$$

Die neue Position des Vertex $\vec{v}_i^k$ der k-ten Refinement-Stufe lässt sich nun mit

$$\vec{v}_i^k \;=\; \vec{v}_i - (\vec{c}_i \cdot \Delta_k) \tag{3.15}$$

[7] Als Z-Fighting bezeichnet man die nicht eindeutige Verdeckungsberechnung aufgrund der begrenzten Z-Buffer-Genauigkeit. Derartige Phänomene treten vor allem bei eng aufeinander liegenden Polygonen auf.

ermitteln. Dabei gibt $\vec{c}_i$ den normierten Richtungsvektor zur Kamera an. Setzt man schließlich 3.13 in 3.14 und danach 3.14 in 3.15 ein, erhält man unter Berücksichtigung der Definition von $\vec{c}_i$:

$$\vec{v}_i^{\,k} \;=\; \vec{v}_i - \left(\frac{\vec{v}_i - \vec{p}_a}{\|\vec{v}_i - \vec{p}_a\|} \cdot \|\vec{v}_i - \vec{p}_a\| \cdot f \cdot k \right).$$

Nach entsprechender Kürzung ergibt sich die endgültige Bestimmung $\vec{v}_i^{\,k}$ mit:

$$\vec{v}_i^{\,k} \;=\; \vec{v}_i - ((\vec{v}_i - \vec{p}_a) \cdot f \cdot k)\,. \tag{3.16}$$

f wurde in diesem Zusammenhang experimentell bestimmt und ist im Wesentlichen von der Genauigkeit des Z-Buffers abhängig. Mit $f = 0.0001$ wurden bei 24 Bit Z-Buffer-Tiefe sehr gute Ergebnisse erzielt, so dass Z-Fighting nur noch in extremen Konstellationen[8] auftritt. Beispiele für die Auswirkung verschiedener f sind in Abschnitt 5.1.2 dargestellt.

Auch wenn ein Z-Fighting nicht vollständig vermieden werden kann, stellt das progressive Rendering mittels Overdraw eine vor allem laufzeiteffiziente Methode dar. Im Gegensatz zum Redraw müssen nach jedem Refinement-Schritt nur die neu hinzugekommenen Faces dem Rendering übergeben werden. Darüber hinaus entfällt ein ständiges Rendering aller Nicht-VT-Objekte, das bei komplexeren Szenen durchaus stark ins Gewicht fallen kann. Die Methode Overdraw ist wie so oft eine Gratwanderung zwischen Qualität und Quantität.

3.5.2 Shading

Als *Shading* bezeichnet man die Bestimmung des Anteils an Licht, das einen Punkt auf der Oberfläche eines Objektes durch Emission, Reflexion oder Transmission verlässt. Die verwendete Funktion zur Berechnung wird als *Shading-Modell* bezeichnet. Das eingesetzte Shading-Modell ist maßgeblich für den Grad der realitätsnahen Abbildung des berechneten Objektes verantwortlich, es bestimmt jedoch auch empfindlich des Laufzeitverhalten des gesamten Renderings.

Das hier verwendete Shading-Modell ist wie bei einer Vielzahl von Ray-Tracern das Phong-Modell [Pho75] mit geringfügiger Modifikation [WW92]. Unter Ergänzung des Terms $k_t \, cos^p \phi_t$, mit ϕ_t dem Winkel zwischen transmittiertem Strahl und Blickrichtung, p einer materialabhängigen Konstante und k_t dem materialabhängigen Transmissionskoeffizienten, berechnet sich die lokale Intensität I_{local} eines Punktes, der von den Lichtquellen $L_1, ..., L_k$ beleuchtet wird aus:

$$I_{local} \;=\; I_a k_a \;+\; \sum_{i=1}^{k} I_{source,i} \left[k_d \, (\vec{n} \cdot \vec{L}_i) \;+\; k_r \, cos^m \phi_s \;+\; k_t \, cos^p \phi_t \right]. \tag{3.17}$$

Dabei ist k_d der diffuse, k_s der spekulare und k_a der umgebungsabhängige Reflexionskoeffizient sowie I_a die Intensität des Umgebungslichtes und $I_{source,i}$ die der Lichtquelle

[8] Solche Konstellationen entstehen beispielsweise bei einer großen Tiefendiskrepanz zwischen den Vertices eines Dreiecks.

i. Ferner gibt ϕ_s den Winkel zwischen Betrachter und reflektierten Strahl und m eine materialabhängige Konstante an. $\vec{n} \cdot \vec{L_i}$ bestimmt den Winkel zwischen Lichteinfall und Normale zur Berechnung der diffusen Reflexionsintensität. Im Falle, dass nur eine Lichtquelle existiert, liefert entweder nur die spekulare Komponente oder nur die transparente Komponente einen Beitrag. Eine Lichtquelle kann bezüglich des Betrachters nur hinter oder vor dem Objekt liegen.

Zur lokalen Komponente I_{local} werden nun noch zwei globale Komponenten I_{refl} und I_{trans} hinzu addiert. Die Gesamtintensität $I(\vec{v}_i)$ eines Vertex $\vec{v}_i$ ergibt sich schließlich aus

$$I(\vec{v}_i) \;=\; I_{local}(\vec{v}_i) \;+\; k_r I_{refl}(\vec{p}_r) \;+\; k_t I_{trans}(\vec{p}_t). \tag{3.18}$$

$I_{refl}(\vec{p}_r)$ repräsentiert die Lichtintensität, die aufgrund von Reflexion von einem Punkt $\vec{p}_r$ in $\vec{v}_i$ unter Berücksichtigung des Abschwächungsfaktors k_r ankommt (siehe Abbildung 3.19). Entsprechendes gilt für die Intensität $I_{trans}(\vec{p}_t)$ und dem Abschwächungsfaktor k_t

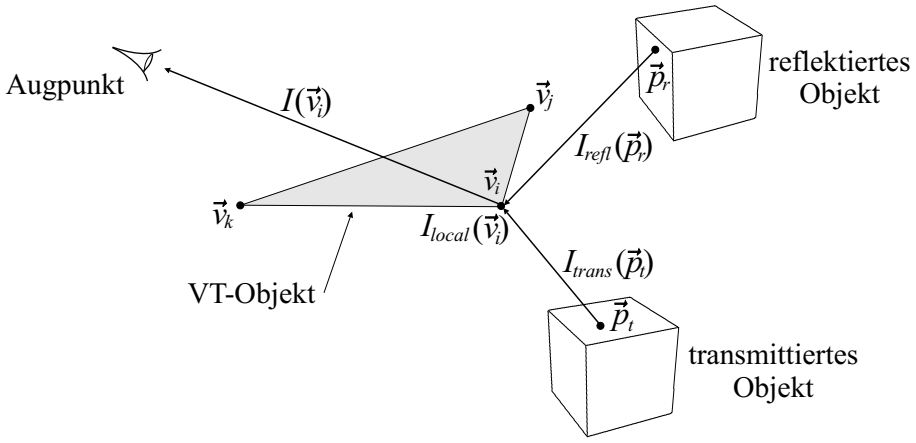


Abbildung 3.19: *Shading-Prinzip beim Vertex-Tracing.*

für die Transmission. Da die Strahlenverfolgung rekursiv erfolgt, muss sich $I(\vec{v}_i)$ auch entsprechend rekursiv ermitteln. Nimmt man Gleichung 3.18 als Grundlage, so ergibt sich am Beispiel der Reflexionskomponente $I(\vec{p}_r)$ die Rekursion

$$I(\vec{p}_r^{\,j}) \;=\; I_{local}(\vec{p}_r^{\,j}) \;+\; k_r^{\,j+1}\, I_{refl}(\vec{p}_r^{\,j+1}) \;+\; k_t^{\,j+1}\, I_{trans}(\vec{p}_t^{\,j+1}), \quad j = 0, 1, ..., n \tag{3.19}$$

mit j der Anzahl der Rekursionsstufen. Die Abschwächungsfaktoren $k_r^{\,j+1}$ und $k_t^{\,j+1}$ entscheiden über den Abbruch der Rekursion. Sinkt der Beitrag der globalen Komponenten unter einen bestimmten Schwellwert, bricht die Rekursion ab, sofern nicht bereits eine maximale, vordefinierte Rekursionstiefe erreicht wurde.

3.5.3 Integration von Texturen

Der Einsatz von Texturen zielt auf eine qualitative Aufwertung des Renderings. Mit ihrer Hilfe lassen sich elegant komplexe Oberflächencharakteristiken approximieren, wie sie bei-

spielsweise durch die Zusammenfassung verschiedener Materialeigenschaften in einem Objekt (Schachbrett) entstehen. Ein Rendering derart hochfrequenter Radiance-Information ist aufgrund von Texturen erst effizient durchführbar.

Was sich beim traditionellen Hardware-Rendering beziehungsweise Ray-Tracing als leistungsfähig herausstellt, entpuppt sich jedoch beim adaptiv progressiven Ray-Tracing als destruktiv. Die zum Teil hochfrequenten Informationen von Texturen verursachen eine vermehrte Sample-Generierung, da sich die nutzbare Image-Kohärenzen entscheidend reduziert. Aus dem kontinuierlichen Radiance-Verlauf einer Objektdarstellung kann nach einer Texturierung eine stark variierende Radiance-Funktion resultieren. Der Vorteil des adaptiven Samplings, die Reduzierung von Abtastwerten, wird drastisch dezimiert. Methoden, wie in [vWvNJ91] vorgeschlagen, versuchen dem Problem zu begegnen. Hier werden Messungen im Textur-Raum durchgeführt, die weitere Kriterien für ein effektives Sampling heranziehen.

Eine jedoch weitaus effizientere Methode als in [vWvNJ91] ist die zusätzliche Ausnutzung von Objektrauminformationen. Berechnet man für jeden Sample nur die zugehörigen Textur-Koordinaten und verzichtet auf einen sofortigen Textur-Lockup, kann das adaptive Refinement auch ohne Berücksichtigung der in der Textur gespeicherten Radiance-Werte durchgeführt werden. Der eigentliche Textur-Lockup geschieht im Anschluss an das Refinement mittels Hardware-Rendering, indem eine Interpolation der Textur-Koordinaten zwischen den Samples erfolgt.

Die prinzipielle Vorgehensweise bezüglich des Einsatzes von Texturen beim Vertex-Tracing zeigt Abbildung 3.20. Abhängig vom Refinement-Prozess entstehen neue Faces f_i. Die Steuerung des Refinements erfolgt dabei durch Kriterien, wie in Abschnitt 3.3.1 dargestellt, jedoch ohne die Einbindung von Radiance-Werten aus Texturen. Für jeden Vertex $\vec{v}_i$ einer Face f_i werden am Beispiel der Abbildung 3.20 lediglich die zugehörigen Textur-Koordinaten u_i^r, v_i^r (falls vorhanden) des reflektierten Objektes ermittelt und entsprechend abgespeichert. Erst beim eigentlichen Rendering-Prozess erfolgt der Textur-Lockup. Jede Face f_i wird nun mit ihrer Primär-Textur in den Frame-Buffer gerendert. Existiert ebenfalls eine reflektierte Textur, findet darüber hinaus auch ein Rendering von f_i mit dieser Textur statt. Eine geeignete Blend-Operation mischt schließlich die Texturen im Frame-Buffer.

Analog zum reflektierten erfolgt die Zuweisung von Textur-Koordinaten bei einem transmittierten Objekt. Gibt es sowohl eine reflektierte als auch transmittierte Textur müssen inklusive der Primär-Textur bis zu drei Texturen im Frame-Buffer übereinander geblendet werden. Mit dieser Methode entfällt vollständig ein Sampling zum Teil hochfrequenter Texturen für die 1. Rekursionsstufe. Für jede weitere Stufe wäre dieser Ansatz ebenfalls denkbar, mündet aber aufgrund des hohen Speicherbedarfs der Textur-Koordinaten in Ineffizienz. Für Rekursionsstufen > 1 erfolgt deshalb ein vollständiges Sampling von Texturen basierend auf ihren Radiance-Werten.

Zeigt jeder Vertex einer Face f_i auf die gleiche reflektierte oder transmittierte Textur, wird von einer sogenannten *Defined Face* gesprochen. Treten Textur-Koordinaten unterschiedlicher Texturen auf oder besitzen sogar einzelne Vertices keine Textur-Koordinaten, entstehen sogenannte *Undefined Faces*. Wie in Abbildung 3.21 verdeutlicht, wird in die-

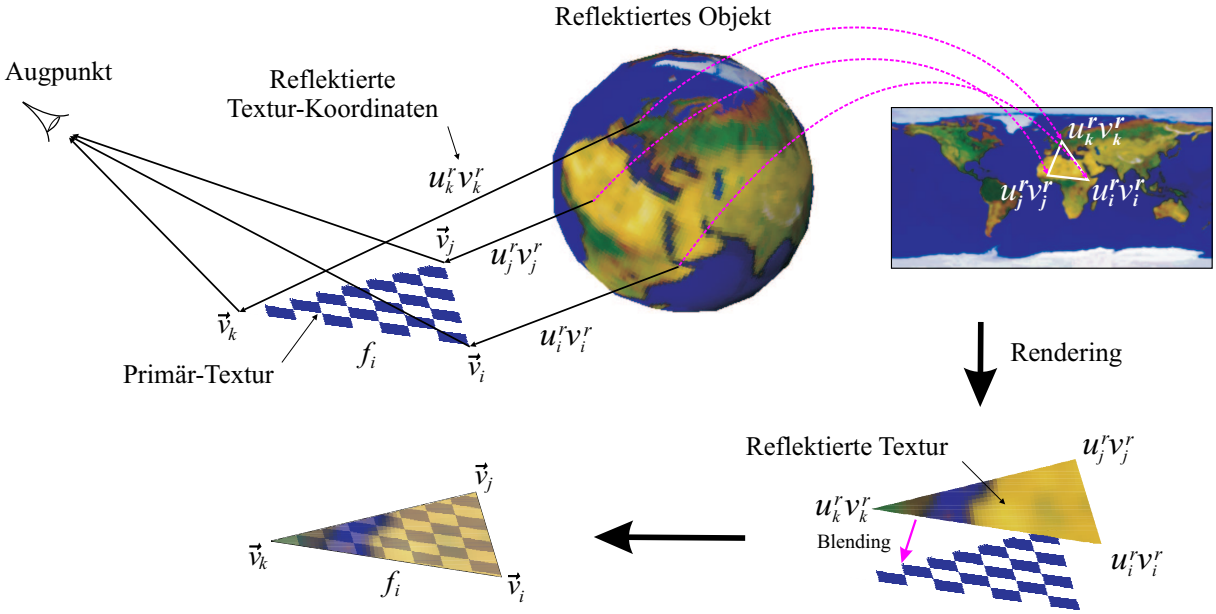


Abbildung 3.20: *Prinzip der Integration von Texturen beim Vertex-Tracing. Zeigen alle drei Vertices einer Face auf ein und die selbe Textur, wird von einer einer sogenannten 'Defined Face' gesprochen.*

sem Fall unmittelbar der Radiance-Wert aus der Textur ausgelesen. Nach Gleichung 3.18 dient der ermittelte Farbwert in Verbindung mit der Objektgrundfarbe schließlich zur Bestimmung der Intensität auf dem Vertex $\vec{v}_i$.

3.5.4 Multi-Pass Texturing

Wurden schließlich auf jedem Vertex einer Defined Face f_i die Textur-Koordinaten der reflektierten sowie transmittierten Objekte bestimmt, folgt das Rendering von f_i. Basierend auf Gleichung 3.18 ermittelt sich die endgültige Fragment-Color C_{frag} im Frame-Buffer aus

$$C_{frag} = C_{prim}CT_{prim} + \alpha_r(C_{refl}CT_{refl} + C_{r,comb}) + \alpha_t(C_{trans}CT_{trans} + C_{t,comb}) \quad (3.20)$$

mit $C_{prim}, C_{refl}, C_{trans}$ der Objektgrundfarbe, $CT_{prim}, CT_{refl}, CT_{trans}$ der Textur-Farbe und α_r, α_t dem Alphawert als Abschwächungsfaktor der Reflexion beziehungsweise Transmission. Angenommen sei dabei, dass die Objektfarbe aus der Modulation (GL_MODULATE) von Grundfarbe und Textur-Farbe resultiert. C_{comb} repräsentiert die kombinierte Farbe ab der 2. Rekursionsstufe, die aus einer weiteren Reflexion beziehungsweise Transmission resultiert. Es gilt

$$C_{comb} = k_r C_{refl} + k_t C_{trans}. \quad (3.21)$$

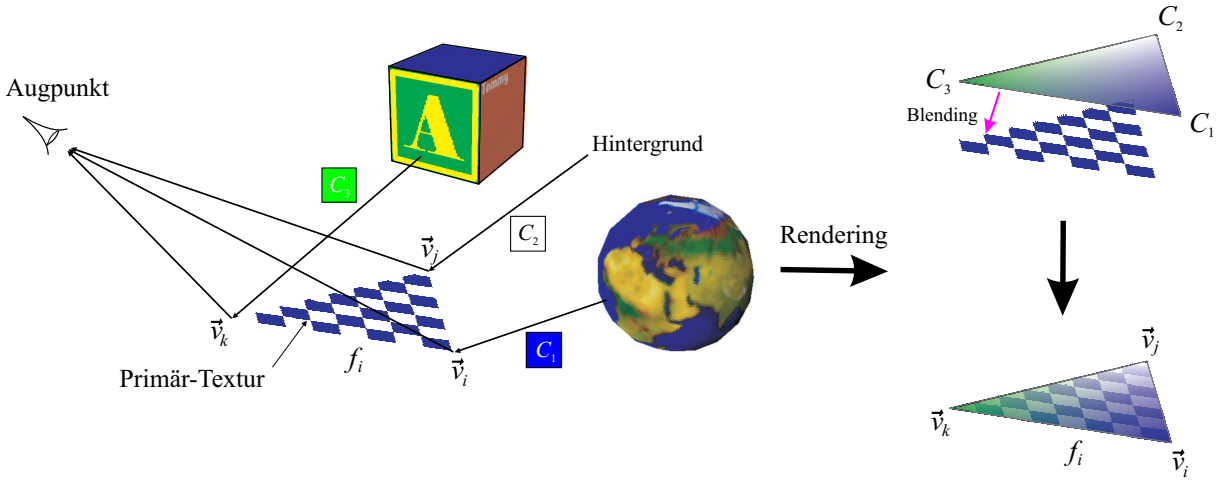


Abbildung 3.21: *Bei nicht identischen Texturen auf den Vertices einer Face wird jeweils der Farbwert aus der Textur ausgelesen und zur Interpolation herangezogen. Solche Faces werden als 'Undefined Faces' bezeichnet.*

Durch das gleichzeitige Rendering von bis zu drei Texturen auf einer Face liegt die Anwendung eines Multi-Texturing [WNDS99] nahe. Problematisch dabei ist jedoch die feste Verdrahtung der Textur-Units, wie in Abbildung 3.22 dargestellt. Bis OpenGL-Version 1.2 ist eine frei programmierbare Kombination der Textur-Units weitestgehend ausgeschlossen. Die ausführbaren Fragment-Operation sind vordefiniert und ermöglichen keine Abbildung der Gleichung 3.20 (siehe dazu [WNDS99]). Hinzu kommt, dass lediglich die Primär-Farbe C_{prim} geeignet mit TE_0 verknüpft werden kann. Andere Farb-Eingänge, wie die GL_TEXTURE_ENV_COLOR sind nicht flexible genug. Zur Erfüllung von Gleichung 3.20 fehlen also mindestens zwei weitere vollwertige Farb-Eingänge[9].

Ohne eine flexiblere Programmierung der Textur-Units, wie es aktuelle OpenGL-Extensions bereits unterstützen und in Abschnitt 3.5.5 angewandt wird, bleibt nur die Alternative eines *Multi-Pass Texturing*. Dabei müsste das Rendering nach Gleichung 3.20 in bis zu fünf Durchgängen (Pass) erfolgen. Um jedoch diesen enorm hohen Rendering-Aufwand zu vermeiden, soll Gleichung 3.20 leicht modifiziert werden. Fasst man $C_{r,comb}$ und C_{refl} beziehungsweise $C_{t,comb}$ und C_{trans} schon vor der Textur-Modulation zusammen, ergibt sich

$$C_{frag} = C_{prim}CT_{prim} + \alpha_r((C_{refl} + C_{r,comb}) \cdot CT_{refl}) + \alpha_t((C_{trans} + C_{t,comb}) \cdot CT_{trans}). \quad (3.22)$$

Sicherlich stellt Gleichung 3.22 nur eine Näherung zu 3.20 dar. Die geänderte Farbkombination ermöglicht jedoch eine Addition vor dem eigentlichen Rendering-Prozess, aufgrund dessen sich das Multi-Pass Texturing auf drei Durchgänge reduzieren lässt. Für jeden

[9]OpenGL bietet auch die Möglichkeit einer sogenannten *Specular Color*. Diese Farbe kann jedoch nicht mit einer Textur direkt kombiniert werden. Eine Verknüpfung ist bis OpenGL-Version 1.2 lediglich mit der finalen Fragment-Farbe durchführbar.

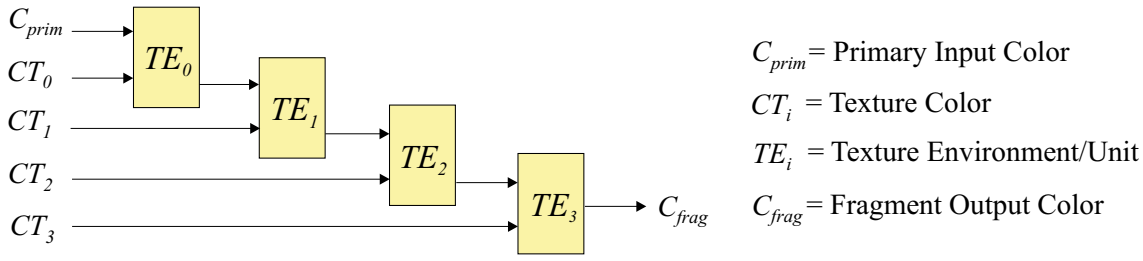


Abbildung 3.22: *Funktionsweise des nicht-programmierbaren* Multi-Texturing *(OpenGL-Version 1.2). Der Ausgang einer Textur-Unit stellt den Eingang der nachfolgenden Einheit dar. Es kann lediglich eine Primär-Farbe C_{prim} mit den Texturen kombiniert werden. Quelle [MB99].*

Durchgang wird jeweils die Objektgrundfarbe mit der Textur-Farbe in einer Textur-Unit moduliert und in den Frame-Buffer geschrieben. Ab dem zweiten Zyklus erfolgt das Schreiben unter Berücksichtigung eines α-*Blendings*, wie in Abbildung 3.20 bereits verdeutlicht. Die Blend-Operation fungiert in diesem Zusammenhang zum einen als Abschwächungsfaktor α_r beziehungsweise α_t und zum anderen als Addition nach Gleichung 3.22.

Trotz der Tatsache, dass das Multi-Pass Texturing noch immer bis zu drei Render-Zyklen für eine Face f_i durchläuft, zeigt die Praxis, dass diese Methode auf älteren Systemen durchaus vertretbare Laufzeiten liefert. Vor allem bei Szenen mit einer geringen Anzahl spekularer Objekte fällt das Multi-Pass Texturing kaum ins Gewicht. Treten verstärkt spekulare Effekte auf, ist sicherlich ein echtes Multi-Texturing in *einem* Durchgang die effizientere Lösung, wie im folgenden Abschnitt beschrieben.

3.5.5 Multi-Texturing

Erst auf modernen Graphik-Subsystemen[10] kann bezüglich des Vertex-Tracings ein echtes Multi-Texturing in nur einem Pass durchgeführt werden. Insbesondere mit der Integration des sogenannten *Pixel-Shaders* [Spi00a] wurde erstmalig eine äußerst flexible Programmierung der Textur-Units realisiert. Hier lassen sich nicht nur die Ein- und Ausgänge der Units beliebig kombinieren, sondern darüber hinaus auch eine zyklische Rückkopplung zwischen den Textur-Units realisieren. Derartige Iterationen der Textur-Verarbeitung können in einem festgelegten Rahmen solange durchgeführt werden, bis das gewünschte Fragment-Ergebnis erreicht wurde.

Die Abbildung der Textur-Shading-Funktion nach Gleichung 3.22 ist jedoch selbst mit dem Funktionsumfang eines Pixel-Shaders nicht explizit durchführbar. Den Knackpunkt stellt die Anzahl der verfügbaren Farb-Input-Register dar. Es können lediglich zwei Farben, die *Primary-Color* und die *Secondary-Color*, für eine Pixel-Shader-Operation definiert werden, wobei letztere zusätzlich ohne Alphawert auskommen muss. Bei einer bei-

[10] In diesem Zusammenhang ist vor allem an Graphikkarten wie NVIDIA ab Geforce3, ATI ab Radeon 8500 oder MATROX ab Parhelia gedacht.

spielsweise gegebenen Anzahl von vier Textur-Units und zwei Farb-Eingängen bedeutet dies, dass weiterhin noch eine Farbkomponente zur Abbildung der Shading-Funktion fehlt.

Abhilfe schafft in diesem Zusammenhang die Möglichkeit des sogenannten *Pass-Through* [DS00]. Diese Funktion einer Textur-Unit erlaubt es, den Wert einer Textur-Koordinate direkte auf die auszugebende Fragment-Color abzubilden. Das heißt, die Textur-Koordinaten s, t, r, q repräsentieren unmittelbar die Farbkomponenten R, G, B, A. Nach Gleichung 3.22 werden mit CT_{refl}, CT_{trans} und CT_{prim} drei Textur-Units beansprucht. Besitzt das Graphik-Subsystem mindestens vier Textur-Units in Summe, kann also mittels Pass-Through die dritte Farbkomponente aus 3.22 dargestellt werden. Die genaue Vorgehensweise dazu zeigt Abbildung 3.23. Soll die Shading-Funktion in *einem* Rendering-Pass ausgeführt werden, ist die Zusammenschaltung von zwei *General-* und einem *Final-Combiner* [Spi00b] notwendig. Beginnend mit dem General-Combiner 0 erfolgt zunächst das Laden der Texturen CT_{refl}, CT_{trans} sowie die Zuweisung der Register durch die Farben C_{refl} und C_{trans}. Letztere wird mittels Pass-Through über Textur-Unit 3 eingesteuert. Nach der jeweiligen Verknüpfung der Register durch Multiplikation dient das Ergebnis als Eingang für den nachfolgenden General-Combiner 1. Mit dem Laden von

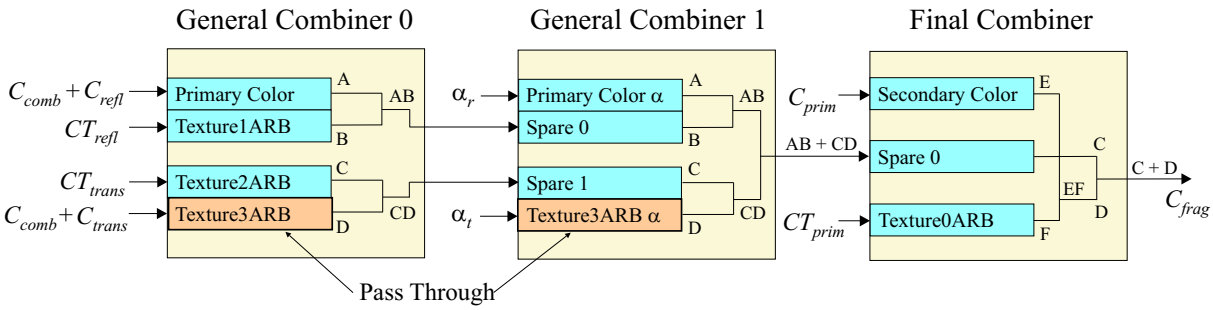


Abbildung 3.23: *Prinzipschaltung eines 1-Pass Multi-Texturing. Die Eingabe einer dritten Farbe erfolgt mittels Pass-Through-Funktion am Beispiel der Textur-Unit 3.*

α_r und α_t als Alphawerte erfolgt hier die Abschwächung der Ergebnisse aus Combiner 0 bevor diese nach einer additiven Verknüpfung in den Final-Combiner überführt werden. Der Final-Combiner übernimmt die Multiplikation der Primär-Komponenten C_{prim} und CT_{prim}. Aus einer letzten Addition resultiert schließlich die Fragment-Farbe C_{frag}, die unmittelbar in den Frame-Buffer geschrieben wird.

Besitzt das Graphik-Subsystem sechs oder mehr Textur-Units ist aufgrund der Pass-Through-Funktion ohne weiteres eine Abbildung von Gleichung 3.20 in einem Pass denkbar. Im Falle, dass weniger als vier Textur-Units existieren, müssen jedoch mindestens zwei Rendering-Pass ausgeführt werden. Die Addition zwischen den einzelnen Durchläufen kann dabei wieder durch Blend-Operationen erfolgen (↗ Abschnitt 3.5.4).

3.6 Diskussion

Das in diesem Kapitel beschriebene Verfahren Vertex-Tracing zielt auf ein interaktives Rendering spekularer, physikalisch korrekter Reflexionen beziehungsweise Transmissionen. Erreicht wird dies durch eine massive Reduzierung des Samplings der Radiance-Funktion im Bildraum. Kern des Verfahrens bildet hierzu ein adaptiv progressives Refinement im Objektraum, welches im Gegensatz zu anderen Verfahren direkt auf der Geometrie polygonaler Szenenobjekte aufsetzt und diese als Initial-Sample-Pattern nutzt.

Die Vorteile dieser Methode liegen vor allem darin, dass weder ein Kollisionstest von Primärstrahlen noch die Detektion von Objektsilhouetten während des adaptiven Samplings notwendig ist. Es entsteht nicht nur eine äußerst hohe Sample-Dezimierung sondern zugleich eine Kostenreduktion bezüglich der Bestimmung eines einzelnen Sample-Wertes. Nachteilig bei einer derart inhärenten Primärstrahlberechnung ist jedoch das Fehlen eines Visibility-Tests. Im Gegensatz zum traditionellen Ray-Tracing, bei dem der Visibility-Test im Verfahren unmittelbar impliziert ist, muss beim Vertex-Tracing eine separate Sichtbarkeitsuntersuchung (↗ Abschnitt 3.4.1) durchgeführt werden. Mit der Verwendung von Rasterisierungs-Hardware für den Visibility-Test entsteht jedoch eine lineare Zeitkomplexität, die den sub-linearen Charakter des Ray-Tracings relativiert. Das Verfahren Vertex-Tracing nähert sich insofern bezüglich seiner Gesamtzeitkomplexität einer linearen Funktion an. Was sich im Hinblick auf überdurchschnittlich hohe Szenenkomplexitäten nachteilig verhält, wirkt sich jedoch auf kleine und mittlere Szenen äußerst effizient aus. Die hohe Einstiegshürde des Ray-Tracings, die aufgrund seines sub-linearen Verhaltens hervorgerufen wird, kann hiermit stark zugunsten kleinerer und mittlerer Szenenkomplexitäten nach unter verschoben werden.

Relativ unabhängig von der Szenenkomplexität ist hingegen die Anzahl der generierten Vertices sowie Faces während des Refinement-Prozesses. Mehr noch: im Gegensatz zum Standard-Ray-Tracing, bei dem die Anzahl der Samples (Pixels) mit der Fläche des Viewports linear skaliert, existiert beim Vertex-Tracing eine sub-lineare Abhängigkeit gegenüber der Auflösung. Bei vierfacher Fläche des Viewports werden in etwa nur doppelt so viele Vertices beziehungsweise Faces erzeugt (↗ Abschnitt 5.2). Die Anzahl der generierten Faces einer fixen Viewport-Größe ist wiederum von den Kriterien des Refinements abhängig (↗ Abschnitt 3.3.1). Wurde die minimale Kantenlänge sehr klein gewählt ($l < 2$), entstehen entsprechend kleine Faces, die nur wenige Pixels überdecken. Damit sich das Verhältnis zwischen Rendering-Aufwand einer Face zu ihrer interpolierten Fläche nicht negativ verhält, drängt sich eine andere Behandlung von derartiger Faces auf. Hier wäre ein konventionelles Ray-Tracing jedes einzelnen überdeckten Pixels sicherlich effizienter, da der Mehraufwand für das Refinement und das eigentliche Rendering der Face entfällt.

Ferner kann die Anzahl der generierten Vertices sowie Faces durch ein verzögertes Refinement weiter reduziert werden. Eine alternative Refinement-Strategie wäre beispielsweise in der Form denkbar, dass Diskontinuitäten entlang einer Edge durch gezieltes Sampling erst näher bestimmt werden, bevor ein neuer Vertex eingefügt wird. Dies würde nicht nur die Anzahl der generierten Faces verringern, sondern hätte außerdem einen positiven

Einfluss auf die visuelle Qualität der ersten Refinement-Stufen. Eine bessere Orientierung der Faces in Bezug auf den Verlauf von Diskontinuitäten wäre die Folge.

Kapitel 4

Verteiltes, paralleles Vertex-Tracing

Nachdem im vorangegangenen Kapitel 3 die Basisalgorithmen des Vertex-Tracings ausführlich dargelegt wurden, soll nun die Betrachtung des Verfahrens unter dem Aspekt der Parallelisierung [Hub97, KGGK94, Mul93] im Vordergrund stehen. Hierzu gilt es zunächst, die Teilmengen des Gesamtalgorithmus ausfindig zu machen, die sich für eine parallele Ausführung eignen oder auf eine sequentielle Abarbeitung angewiesen sind.

Darauf basierend sollen im Weiteren grundlegende Konzepte vorgestellt werden, die eine Parallelisierung des Verfahrens ermöglichen. Differenziert wird dabei zwischen einem *Verteilten* und *Parallelen* Ansatz[1]. Der Verteilte Ansatz zielt auf eine Parallelisierung im Rechner-Cluster[2]. Hier sollen Konzepte beschrieben werden, die mittels *Message Passing* als Kommunikationsform im Rechnernetz eine Aufgabenverteilung vornehmen und die Ergebnisse der Berechnung zusammenführen. Der Parallele Ansatz richtet sich hingegen auf die Verwendung von *Shared-Memory* Systemen[3]. Mit Hilfe von Kommunikationsformen des Betriebssystems steht hier eine Parallelisierung durch *Multi-Threading* im Mittelpunkt.

Die Bewertung der verschiedenen Vorgehensweisen erfolgt jeweils unter den Aspekten des Kommunikationsbedarfs sowie der gleichmäßigen Lastverteilung, dem *Load-Balancing*[4]. Die daraus resultierenden Ergebnisse entscheiden letztlich über die Effizienz des parallelen Konzeptes.

[1] In der Literatur wird diesbezüglich auch oft zwischen *Distributed Computing* und *Parallel Computing* unterschieden.

[2] Rechner-Cluster gehören zu der Gruppe von Parallelrechnern mit NORMA-Architektur (No Remote Memory Access).

[3] Shared-Memory Systeme zählen zu Parallelrechnern mit UMA- beziehungsweise NUMA-Architektur ((Non-) Uniform Memory Access).

[4] In der Literatur wird auch oft zwischen der eigentlichen Lastverteilung (*Load-Distribution*) und dem Lastausgleich (*Load-Balancing*) unterschieden. Da in dieser Arbeit die Lastverteilung stets eng mit einer Gleichverteilung verbunden ist, wird in diesem Zusammenhang unmittelbar von einem Load-Balancing gesprochen.

4.1 Aspekte paralleler Ray-Tracing Systeme

Der Charme des Ray-Tracings liegt nicht nur in der Einfachheit seines Algorithmus, sondern besticht erst recht im Hinblick auf eine Parallelisierung. Die Möglichkeit, zwei benachbarte Pixel völlig unabhängig voneinander berechnen zu können, garantiert ein außergewöhnlich hohes Potential bezüglich einer kooperativen Ausführung.

Trotz dieser Voraussetzungen existieren jedoch auch bei der Parallelisierung des Ray-Tracings verschiedene Faktoren, die die typischen Leistungsmaße wie beispielsweise *Speed-Up, Effizienz* oder *Redundanz* [HK96] des parallelen Algorithmus gegenüber seines sequentiellen Pendants signifikant beeinflussen. Zwei der wesentlichsten Faktoren, die maßgeblich die Wahl des parallelen Systemdesigns bestimmen, sind

- das *Computation-Model* sowie
- das *Load-Balancing*.

Das Computation-Model entscheidet, welche Komponenten des Algorithmus auf die verfügbaren Knoten verteilt werden. Unterschieden wird dabei zwischen den Verteilungsarten

- *Demand-Driven* und
- *Data-Driven*.

Im Fall eines Demand-Driven Systems wird jedem Prozessor p_i eine konkrete Aufgabe zugeteilt (Aufgabenverteilung). p_i ist für *alle* Berechnungen zuständig, die zur Lösung dieser Aufgabe erforderlich sind. Beim Demand-Driven Ray-Tracing kann diese Aufgabe aus der vollständigen Berechnung eines zugewiesenen Bildausschnittes bestehen. Der Prozessor muss von der Erzeugung des Primärstrahles, über die Kollisionserkennung bis hin zur Bestimmung der Pixel-Farbe (Shading) sämtliche Aufgaben erfüllen.

Bei Ray-Tracing Systemen, die auf dem Prinzip des Data-Driven basieren, erfolgt die Arbeitsverteilung aufgrund der Datenbasis (Datenverteilung). Jeder Prozessor besitzt einen zugewiesenen Teilbereich des gesamten Datenvolumens. Fällt die Aufgabe des Gesamtsystems bezüglich der Daten in den Zuständigkeitsbereich des Prozessors, ist er für die Bearbeitung der Aufgabe zuständig. Nur er besitzt die exklusiven Zugriffsrechte. Im Data-Driven Ray-Tracing wird dazu meist der gesamte Objektraum (Szene) in Segmente zerlegt und unter den Prozessoren verteilt. Durchdringt ein verfolgter Strahl den Raum des entsprechenden Prozessors, prüft dieser den Strahl auf Kollision mit den Objekten, die in seinem Bereich angesiedelt sind.

Der Vorteil des Data-Driven Ray-Tracings liegt vor allem in der effizienten Nutzung der Ressource Speicher. Hier kann insbesondere in parallelen Systemen mit wenig lokalem Speicher, eine Berechnung von komplexen Szenen vorgenommen werden. Nachteilig wirkt sich jedoch der erhöhte Kommunikationsaufwand aus. Aufgrund der Strahlenverfolgung durch den Raum entsteht ein häufiger Wechsel zwischen den Zuständigkeitsbereichen im gesamten Datensegment. Die Folge ist ein hoher Kommunikationsbedarf, der sich als *Overhead* im parallelen System auswirkt.

Die Wahl des Computation-Models übt unter anderem starken Einfluss auf die Strategie des Load-Balancings. Das Load-Balancing garantiert, dass jeder beteiligte Prozessor des parallelen Systems den Anteil an *Load* erhält, der seiner Leistungsfähigkeit entspricht. Ziel ist es, dass jeder Prozessor unabhängig der übertragenen Aufgaben in etwa die gleiche Berechnungszeit benötigt. Das Load-Balancing ist ein wesentlicher Faktor zur Erreichung des maximalen Speed-Ups durch Parallelisierung. Das Problem einer guten Realisierung von Load-Balancing liegt in der genauen Vorhersage, wie sich die Last der Berechnung verteilen wird. So ist es schwer apriori abzuschätzen, inwieweit die Lastverteilung gewichtet werden muss, um eine Gleichverteilung auf verfügbare Prozessoren zu erreichen. Vor diesem Hintergrund unterscheidet man hauptsächlich zwischen zwei Arten von Load-Balancing:

- *statisches Load-Balancing* und
- *dynamisches Load-Balancing.*

Beim statischen Load-Balancing, das vorwiegend im Pre-Prozess stattfindet, lassen sich nur schwer Aussagen über den Berechnungsaufwand des jeweiligen Teilprozesses treffen. So kann beim Ray-Tracing nur bedingt der Aufwand zur Verfolgung eines definierten Strahls abgeschätzt werden. Statisches Load-Balancing wird beim Ray-Tracing in der Regel durch eine fixe Aufteilung der Bildebene in Regionen praktiziert. Die Größen der Regionen können dabei durch stark approximierte Vorberechnungen entsprechend bestimmt werden, bleiben jedoch über den gesamten Berechnungsverlauf konstant.

Im Gegensatz dazu basiert das dynamische Load-Balancing auf einer kontinuierlichen Anpassung des Loads während der Berechnungsphase. Beim Ray-Tracing existieren dazu zwei prinzipielle Konzepte. Zum einen erfolgt nach jeder Kameraänderung eine dynamische Bestimmung der Regionsgrößen und eine erneute Zuweisung auf die vorhandenen Prozessoren. Zum anderen wird der Bildbereich abhängig von der Menge p an Prozessoren in eine bestimmte Anzahl n mit $n \gg p$ von vordefinierten Regionen (*Mosaic-Tiles*) aufgeteilt. Jedem Prozessor wird immer dann eine neue Region zugewiesen, wenn dieser die Bearbeitung der vorherigen beendet hat und noch zu berechnende Regionen vorhanden sind. Dieses auch als *Task-Queue* bezeichnete Load-Balancing findet in heutigen parallelen Ray-Tracing Systemen vor allem durch seine einfache Implementierung starke Verbreitung.

Inwieweit die vorhandenen Konzepte paralleler Ray-Tracing-Systeme in Bezug auf eine Parallelisierung des Vertex-Tracings Anwendung finden beziehungsweise angepasst werden müssen, zeigen die nachfolgenden Abschnitte.

4.2 Parallele und sequentielle Komponenten des Vertex-Tracings

Nach Amdahl's Gesetz [Amd67] kann ein Algorithmus nur maximal so viel Speed-Up bei einer beliebigen Anzahl von Prozessoren erreichen, wie sein sequentieller Anteil im

Algorithmus zulässt. Bestimmt $T(1)$ mit $T(1) = 1$ die totale Berechnungszeit eines Algorithmus mit einem Prozessor und $T(p)$ die totale Berechnungszeit mit p Prozessoren, dann gilt für den Speed-Up S:

$$\begin{aligned} S &= \frac{T(1)}{T(p)} \\ &= \frac{1}{\beta + \frac{1-\beta}{p}} \end{aligned} \tag{4.1}$$

mit β dem sequentiellen Anteil des Algorithmus. Läuft die Anzahl der Prozessoren gegen unendlich ($p \to \infty$), so ergibt sich aus 4.1 ein maximaler Speed-Up S_{max} von

$$S_{max} = \frac{1}{\beta}, \tag{4.2}$$

der das indirekt proportionale Verhalten von β gegenüber S verdeutlicht. Auf das Vertex-Tracing bezogen, bedeutet dies, dass auch hier ein maximal erreichbarer Speed-Up stark vom sequentiellen Anteil des Algorithmus abhängig ist.

Aufschluss über die parallelen sowie sequentiellen Komponenten des Vertex-Tracings erhält man durch eine funktionale *Dekomposition* des Algorithmus. Mögliche Varianten einer Dekomposition zeigen die Abbildungen 4.1 und 4.2. Die Parallelisierung des Kerns des Vertex-Tracings ist vor allem von der gewählten Datenebene abhängig. Setzt man die zu bestimmenden Vertices als Grundlage einer Parallelisierung, entsteht eine Dekomposition nach Abbildung 4.1. Die einzig parallelisierbare Komponente ist das Sampling

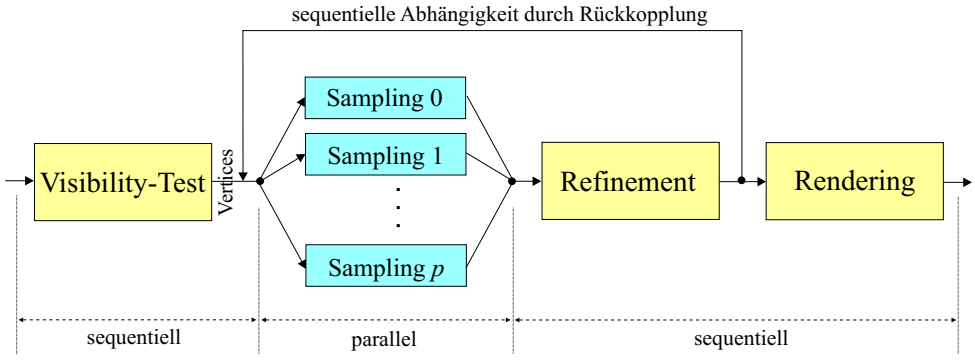


Abbildung 4.1: *Dekomposition des Algorithmus Vertex-Tracing in seine sequentiellen und parallelen Anteile basierend auf der parallelen Verarbeitung von Vertices.*

der Vertices. Ihre Berechnung ist voneinander unabhängig und kann insofern nebenläufig vorgenommen werden. Da das Verfahren adaptiv progressiv arbeitet, existiert jedoch nur eine bestimmte Menge an Vertices, die sich innerhalb einer Refinement-Stufe bearbeiten lassen. Jede weitere Stufe setzt die Auswertung der bereits bestimmten Samples durch

das Refinement voraus. Erst nach diesem Schritt entstehen neue Vertices, die wiederum dem Sampling zugeführt werden. Es resultiert eine sequentielle Abhängigkeit in Form

$$Sampling_k \longrightarrow Refinement_k \longrightarrow Sampling_{k+1} \longrightarrow Refinement_{k+1} \longrightarrow ...$$

mit k der Anzahl an Refinement-Stufen.

Diese Eigenschaften entstehen aufgrund der allmählichen Annährung an die Radiance-Funktion und sind typisch für ein adaptiv progressives Ray-Tracing. Eine vollkommen unabhängige Berechnung beliebiger Samples wie beim Standard-Ray-Tracing muss auch beim Vertex-Tracing aufgegeben werden und erschwert insofern die Parallelisierung des Verfahrens. Der maximal erreichbare Speed-Up wird durch den zusätzlichen sequentiellen Anteil des Refinements dezimiert.

Eine alternative Vorgehensweise zeigt Abbildung 4.2. Hier erfolgt die Dekomposition des Verfahrens auf Grundlage einer höheren Datenebene, der Parallelisierung anhand von Faces. Wird jede Face eines VT-Objektes als unabhängige Datenstruktur betrachtet, wirkt das Refinement lokal auf jede Face und kann insofern ebenfalls parallel ausgeführt werden. Der Vorteil dieser Methode liegt vor allem darin, dass die sequentielle Abhängig-

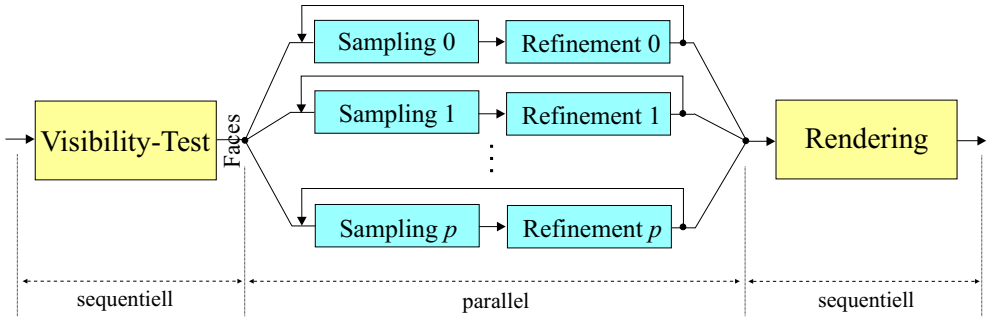


Abbildung 4.2: *Dekomposition des Algorithmus Vertex-Tracing in seine sequentiellen und parallelen Anteile basierend auf der parallelen Verarbeitung von Faces.*

keit zwischen dem Sampling und Refinement bezüglich einer Parallelisierung umgangen wird. Eine Kommunikation zwischen den Prozessoren ist bis zum Abbruch des gesamten Refinement-Prozesses in diesem Kontext nicht mehr notwendig.

Erscheint die Parallelisierung anhand von Faces geeigneter als die der Vertex-Methode (↗ Abbildung 4.1), stellt sie dennoch nur einen Kompromiss in Bezug auf die Visualisierungsqualität dar. Mit der Zwangsverfeinerung von Edges, als integraler Bestandteil einer konsistenten Rekonstruktion der Radiance-Funktion (↗ Abschnitt 3.3.2), entstehen Face-übergreifende Abhängigkeiten, die hier nicht berücksichtigt werden. Eine Parallelisierung nach Abbildung 4.2 kann zur Nichtdetektion von Features führen, die sich sonst aus einer Zwangsverfeinerung ergeben hätten. Denkbar wäre eine Kommunikation zwischen den einzelnen Prozessoren während des Refinement-Prozesses. Muss eine Edge, deren adjazente Face einem Nachbar-Prozessor gehört, zwangsverfeinert werden, erfolgt ein Informationsaustausch. Da der Bedarf an Zwangsverfeinerungen beliebig hoch sein kann, führt jedoch

diese Form der Kommunikation unter Umständen zur starken Belastung des parallelen Systems.

Ein ähnlicher Ansatz wie in Abbildung 4.2 entsteht bei der Verteilung von Objekten anstelle von Faces. Gegenüber der Face-Methode hat diese noch höhere Datenebene den Vorteil, dass das Problem der Zwangsverfeinerung nicht auftreten kann. Zwischen den VT-Objekten existieren keinerlei Abhängigkeiten. Problematisch hierbei wird jedoch die verfügbare Granularität, die für eine effiziente Parallelisierung im Hinblick auf ein gutes Load-Balancing notwendig ist. Gibt es nicht genügend VT-Objekte beziehungsweise sind nicht ausreichend von ihnen sichtbar, kann das parallele System den anfallenden Load nicht gleichmäßig genug verteilen. Das System würde zur starken *Disbalance* neigen.

Die vorgestellten Konzepte eines parallelen Vertex-Tracings zeigen, dass die Wahl der Datenabstraktionsebene maßgeblich für die Parallelisierbarkeit des Verfahrens verantwortlich ist. Eine Verteilung auf Vertex-Ebene bedeutet zwar einen hohen Kommunikationsbedarf (aufgrund der sequentiellen Abhängigkeit zwischen Sampling und Refinement), besticht jedoch durch eine konsistente Bearbeitung sowie einem leicht implementierbaren Load-Balancing. Besonders gut geeignet, ist diese Methode demnach für Systeme mit Multi-Threading wie in Abschnitt 4.4 näher beschrieben. Eine verteilte Berechnung auf der Basis von Faces ist dagegen eher für Master-Slave-Konzepte im Cluster prädestiniert. Hier genügt das langsamere Message-Passing dem geringen Kommunikationsbedarf. Eine ausführliche Beschreibung eines solchen Systems folgt im Abschnitt 4.3.

4.3 Message-Passing im heterogenen Rechnernetz

Ein mächtiges Programmierwerkzeug zur Implementierung paralleler Algorithmen ist *Message-Passing* [KGGK94]. Es ermöglicht eine explizite Kontrolle über die Tätigkeit eingebundener Prozessoren im parallelen System und bietet Mechanismen für einen direkten Datenaustausch zwischen verschiedenen Prozessen. Hauptvorteil ist jedoch, dass eine Implementierung mittels Message-Passing nicht zwingend an ein Hardware-System gebunden ist. Das Rechnersystem, auf dem das parallele Programm läuft, wird sekundär.

Vor dem Hintergrund, ein verteiltes Vertex-Tracing sowohl in einem Rechnernetz als auch auf Shared-Memory-Maschinen zu implementieren, wurde das Konzept Message-Passing verfolgt [Bei01]. Die verbreitetsten Plattformen für Message-Passing sind wohl *PVM*[5] und *MPI*[6]. Während sich MPI durch eine bessere Kommunikationsgeschwindigkeit aufgrund der exklusiven Unterstützung homogener Systeme auszeichnet, bietet PVM die Möglichkeit, gleichzeitig heterogene Rechnerarchitekturen in einer parallelen virtuellen Maschine zu vereinen[7]. Diese Eigenschaft von PVM wurde im Sinne eines verteilten Vertex-Tracing im heterogenen Netzwerk genutzt. Darüber hinaus vermag PVM Client-Prozesse selbsttätig zu administrieren, indem diese zentral erzeugt beziehungsweise zerstört werden können. Ferner existiert mit der Erweiterung *CPPvm* [Goe01] eine

[5]PVM: Parallel Virtual Machine

[6]MPI: Message Passing Interface

[7]Ein detaillierte Beschreibung der Vor- und Nachteile von PVM und MPI findet man in [GKP96].

Schnittstelle zum objektorientierten Paradigma.

In den nachfolgenden Abschnitten soll nun ein Konzept vorgestellt werden, das basierend auf PVM ein verteiltes Vertex-Tracing realisiert. Der Schwerpunkt liegt dabei auf der Wahl eines geeigneten Load-Balancings sowie einer effizienten Datenverteilung. Da starke Engpässe in der Bandbreite heterogener Rechnernetze auftreten können, entscheiden diese Komponenten signifikant über die Gesamteffizienz des verteilten Systems.

4.3.1 Master-Slave-Konzept

Wie bereits weiter oben erwähnt, besteht die Anforderung darin, ein verteiltes Vertex-Tracing in einem heterogenen System abzubilden. Die daraus resultierenden Kommunikationseinschränkungen führten zur Wahl des Demand-Driven Ansatzes (↗ Abschnitt 4.1). Ähnlich eines Demand-Driven Ray-Tracings, bei dem ein Master die Verteilung der Gesamtaufgabe an p Slaves übernimmt, wird auch hier ein Master-Slave-Konzept (↗ Abbildung 4.3) umgesetzt. Jeder Slave erhält beim verteilten Vertex-Tracing eine vollständige Kopie der Szene und ist für die Berechnung eines zugewiesenen Teils (*Task*) der Gesamtaufgabe zuständig. Die Dekomposition des Algorithmus erfolgt dabei nach Abbildung 4.2. Aufgabe des Masters ist es, nach der Durchführung des Visibility-Tests, jedem Sla-

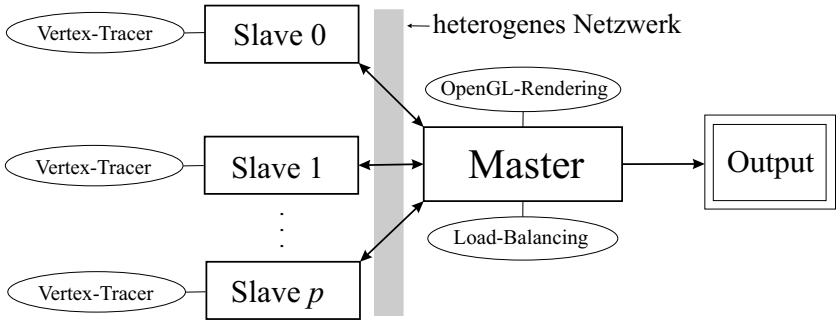


Abbildung 4.3: *Master-Slave Konzept für ein verteiltes Vertex-Tracing nach dem Demand-Driven Prinzip.*

ve eine bestimmte Anzahl an sichtbaren Faces zu übergeben (↗ Abschnitt 4.3.3). Der Slave vollführt nun das Vertex-Tracing im Sinne eines Samplings und Refinements bis zum gewünschten Ergebnis. Die dabei generierten Faces werden abzüglich der substituierten Faces (↗ Abschnitt 3.5.1) mit Shading-Informationen versehen und zum Master zurückgesendet. Dieser übernimmt schließlich das eigentliche Rendering. Alle Nicht-VT-Objekte und alle neu generierten Faces (VT-Objekte) werden mittels OpenGL-Hardware dargestellt.

Mit der Zentralisierung des Hardware-Renderings im Master, kann theoretisch jeder Knoten eines heterogenen Rechnernetzes als Slave fungieren. Die Leistung eines Slaves ist einzig und allein von der Prozessor- und Speicherausstattung abhängig. Begünstigend ist diese Eigenschaft im Hinblick auf eine Ausführung auf Shared-Memory-Systemen.

Graphik-Subsysteme, deren Anzahl auf solchen Systemen in der Regel stark begrenzt ist, können exklusiv vom Master verwendet werden. Der Shared-Memory dient hier vor allem einer schnelleren Kommunikation in Form des Message-Passing.

Eine wesentliche Rolle übernimmt der Master im Hinblick auf ein effizientes Load-Balancing. Dieses ist ebenfalls zentralisiert angeordnet und damit Bestandteil des Master-Prozesses. Das Load-Balancing entscheidet nicht nur über die vollständige Auslastung eines jeden Slaves, sondern sorgt darüber hinaus, dass jeder zugewiesene Task in der gleichen Berechnungszeit ausgeführt werden kann.

4.3.2 Kommunikation und Synchronisation

Im Hinblick auf die Anwendung in einem heterogenen Netzwerk, ist insbesondere das zu übertragende Datenvolumen, aber auch die Anzahl eines notwendigen Verbindungsaufbaus für die Leistungsfähigkeit eines verteilten Vertex-Tracings maßgebend. Jeder Verbindungsaufbau besitzt eine bestimmte *Latenzzeit*, die sich negativ auf den erreichbaren Speed-Up auswirken kann. Die Anzahl an Verbindungen, die während eines zu berechnenden Frames aufgebaut werden, hängt dabei im Wesentlichen vom gewählten Kommunikationsmechanismus zum Datenaustausch und zur Synchronisation ab.

Den Kommunikationsverlauf in Abhängigkeit von der Zeit zeigt im Einzelnen Abbildung 4.4. Mit dem Senden der aktuellen Kameraparameter (blaue Linien) beginnt der Master gegenüber dem Slave mit der Anfrage seiner Berechnungsanforderung (Task). Ist der Slave empfangsbereit, signalisierte er dieses durch Senden einer Quittierung. Der Master startet daraufhin mit der Übertragung der Primär-Faces, die für diesen Slave vorgesehen sind. Nach Erhalt aller Primär-Faces kann der Slave mit dem Vertex-Tracing beginnen. Das Ergebnis, die generierten Faces, wird schließlich zum Master zurückgesendet und per Rendering dargestellt. Erfolgt während der Berechnungsphase des Slaves eine Kameraänderung, bricht dieser durch Erhalt neuer Kameraparameter mit dem Vertex-Tracing ab (rote Linien). Das Quittierungssignal verhindert dabei, dass der Master womöglich ununterbrochen neue Kameraparameter verschickt. Erst nach Erhalt der Quittierung ist es dem Master erlaubt, neue Parameter zu verschicken.

Die vom Master empfangenen Faces werden auf ihre Aktualität geprüft, indem die momentane Kamera-ID des Masters mit der Kamera-ID der empfangenen Faces verglichen wird. Bei Ungleichheit erfolgt ein Verwerfen der generierten Faces. Die neu gesendeten Kameraparameter (grüne Linien) werden unterdessen vom Slave verarbeitetet. Erst nach Empfang der generierten Faces durch den Master endet bei Übereinstimmung der Kamera-ID die Kommunikation dieses Tasks.

Die Übertragung der Primär-Faces in Richtung Slave erfolgt durch die eindeutige Zuordnung jeder Face zu einem Index. Das heißt, bei n Primär-Faces werden n Indizes benötigt, um dem Slave die zu bearbeitenden Primär-Faces mitzuteilen. Bei einer Indexgröße von 32 Bit entspricht die zu übertragene Gesamtgröße $D = n \cdot 4$ Bytes.

Hinsichtlich des Datenvolumens der zu übertragenden generierten Faces vom Slave zum Master spielt neben der Anzahl auch die Eigenschaft jeder einzelnen Face beziehungsweise ihrer Vertices eine entscheidende Rolle. Ziel ist es, nur die tatsächlich benötigten

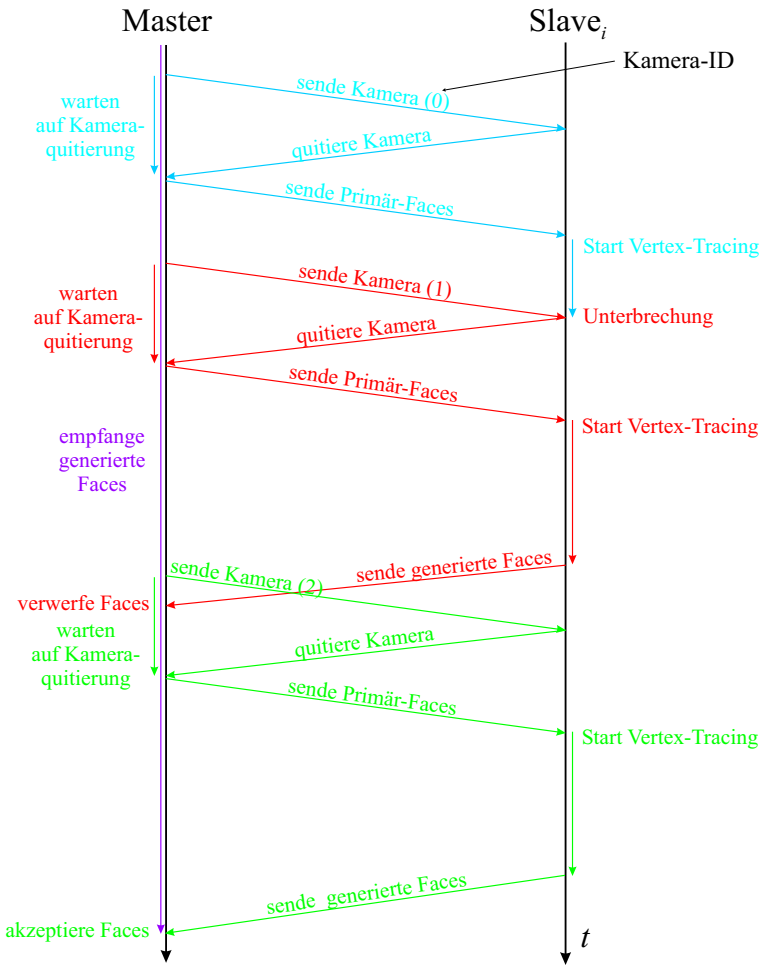


Abbildung 4.4: *Kommunikationsmechanismus zwischen Master und Slave. Die generierten Faces werden erst vom Master akzeptiert, wenn ihre Berechnung auf Basis der aktuellen Kamera erfolgte.*

Information, die die Eigenschaft eines Vertex spezifizieren, zu übertragen. Eine Auflistung möglicher Vertex-Parameter zeigt Abbildung 4.5 in Anlehnung an Gleichung 3.20. Die in Gleichung 3.20 enthaltene Textur-Farbe CT wird hier mittels Textur-Index TX und der entsprechenden Textur-Koordinaten T definiert. Der eigentliche Textur-Lookup zur Bestimmung von CT erfolgt erst durch den Master während des Rendering-Prozesses. Jedes TX ist einer Textur eindeutig zugeordnet. Geht man von 256 Texturen als maximale Anzahl aus, benötigt die Übertragung des Textur-Index jeweils 1 Byte.

Neben den Vertex-Koordinaten $(x, y, z)^T$, dessen Übertragung in jedem Fall erfolgen muss, sind verschiedene Übertragungskombinationen der übrigen Parameter denkbar. Existieren keine Texturen, bedarf es lediglich der Übermittlung von C_{frag} als resultierende Farbe aus der Rekursion des Ray-Tracings. Die minimale Datenmenge D_V^{min}, die pro Vertex übertragen werden muss, beträgt damit 16 Bytes. Im Falle von texturier-

Parameter pro Vertex $\vec{v}$	Größe [Byte]	Minimal	Maximal
$\vec{v} = (x, y, z)^T$	12	x	x
C_{frag}	4	x	
C_{prim}	4		x
$T_{prim} = (u, v)^T$	8		x
C_{refl}	4		x
$C_{r,comb}$	4		x
TX_{refl}	1		x
$T_{refl} = (u^r, v^r)^T$	8		x
C_{trans}	4		x
$C_{t,comb}$	4		x
TX_{trans}	1		x
$T_{trans} = (u^t, v^t)^T$	8		x
Gesamtgröße D_V pro Vertex [Byte]		16	58

Abbildung 4.5: *Mögliche Vertex-Parameter einer generierten Face.*

ten Objekten entsteht jedoch ein weitaus größeres Datenaufkommen. Neben den Textur-Koordinaten T und den entsprechenden Textur-Indizes TX muss zusätzlich die Grundfarbe $(C_{prim}, C_{refl}, C_{trans})$ des Objektes separat übermittelt werden. Hinzu kommt $C_{r,comb}$ beziehungsweise $C_{t,comb}$ als resultierende Farbe aus der weiteren Rekursion des Ray-Tracings. Gespart werden kann am Textur-Index der Primär-Textur. Hier erfolgt die Bestimmung der Primär-Textur über die Zugehörigkeit der generierten Faces zum VT-Objekt.

Die maximale Datenmenge D_V^{max}, die zur Übertragung eines Vertex notwendig ist, kann bei entsprechender Vertex-Eigenschaft 58 Bytes betragen (↗ Abbildung 4.5). Das heißt, sowohl eine Primär-Textur, eine reflektierte als auch transmittierte Textur müssen in diesem Fall dargestellt werden. Die resultierende mittlere Datenmenge $\overline{D}_V$ liegt bei etwas 37 Byte/Vertex. Hinzu kommen die Daten D_F, die die Topologieinformationen der generierten Faces enthalten. Jede Face benötigt 3 Indizes, um ihre zugehörigen Vertices zu referenzieren. Bei einer Indexgenauigkeit von 32 Bit entspricht das 12 Byte/Face. Das durchschnittliche Gesamtdatenvolumen D_{ges} der generierten Faces, das vom Slave zum Master übertragen werden muss, ergibt sich schließlich aus

$$\begin{aligned} D_{ges} &= m \cdot \overline{D}_V + n \cdot D_F \\ &= m \cdot 37 + n \cdot 12 \end{aligned} \tag{4.3}$$

mit m der Menge an Vertices und n der Menge an Faces. Nicht betrachtet, wurde an dieser Stelle die Möglichkeit eines komprimierten Datenaustauschs. Mit Hilfe einer geeigneten Datenkompression sind hier weitere Einsparungen denkbar.

4.3.3 Load-Balancing

Wie in Abschnitt 4.3.1 bereits erwähnt, erfolgt die Verteilung des anfallenden Berechnungsaufwandes in Form des Load-Balancings *zentralisiert* durch den Master. Der Einsatz des verteilten Vertex-Tracings in einem heterogenen Rechnernetz verlangt aufgrund geringer verfügbarer Bandbreiten ein Minimum an Kommunikationsaufkommen. Ein *dezentralisiertes* Load-Balancing wäre hier aufgrund des intensiven Kommunikationsbedarfs zwischen den Slaves kaum realisierbar.

Der Einsatz in einem heterogenen System kann außerdem starke Schwankungen an verfügbaren Ressourcen hervorrufen. Typisch für ein Unternehmensnetzwerk ist der sporadische Zugriff auf verfügbare Rechenleistung durch seine Anwender. Aufgabe des Load-Balancing ist es deshalb, nicht nur auf variierenden Load aufgrund der eigentlichen Berechnung zu reagieren, sondern darüber hinaus auch Schwankungen verfügbarer Ressourcen auszugleichen. Gerecht wird diesen Anforderungen die Implementierung eines *dynamisches* Load-Balancings. Hier wird nach jedem berechneten Frame der aktuelle Zustand des Gesamtsystems berücksichtigt, um eine hinreichende Systembalance zu schaffen. Schwankungen bezüglich der Rechenleistung, Kommunikationsbandbreite oder des Berechnungsaufwands lassen sich somit wirkungsvoll ausgleichen.

In Bezug auf ein verteiltes Vertex-Tracing wurden zwei voneinander verschiedene Konzepte für ein zentralisiertes dynamisches Load-Balancing implementiert. Sowohl das *Pre-Compute* Load-Balancing als auch das *Queue* Load-Balancing basiert auf der Verteilung von Primär-Faces, die es zu verfeinern gilt. Inwieweit sich darüber hinaus beide Konzepte voneinander unterscheiden und welche Vor- und Nachteile sie bezüglich Qualität und Quantität aufweisen, soll nun im Weiteren detailliert beschrieben werden.

Pre-Compute Load-Balancing

Beim Pre-Compute Load-Balancing erfolgt die Verteilung des Loads ähnlich wie in [NG95] anhand einer apriori Aufwandsabschätzung. Das heißt, vor dem Verteilen des Loads muss zunächst eine Berechnung durchgeführt werden, die eine grobe Aussage über die Verteilung des gesamten Berechnungsaufwands (Gesamt-Load) bezüglich der Szene zulässt. Hierfür wird jede Primär-Face f_i eines VT-Objektes auf ihren potentiellen Berechnungsaufwand b_{f_i} untersucht. Die Verteilung jeder f_i geschieht dann auf Basis von b_{f_i}. Der Gesamt-Load eines Slaves ergibt sich schließlich aus der ihm zugewiesenen Anzahl von Primär-Faces f_i und dem daraus resultierenden potentiellen Berechnungsaufwand.

Die Bestimmung von b_{f_i} einer Primär-Face setzt zunächst die Bestimmung von b_{e_i} ihrer zugehörigen Primär-Edges e_i voraus (↗ Abbildung 4.6). Hierzu erfolgt zuerst ein Sampling der projizierten Vertices $\vec{p}_i, \vec{p}_j$ von e_i, um im Anschluss die Sample-Differenz δ_i bestimmen zu können. δ_i kann nun zur Abschätzung des Berechnungsaufwandes b_{e_i} der Primär-Edge verwendet werden. Es gilt im Allgemeinen: Je größer die Sample-Differenz, desto größer werden die Kosten zur Verfeinerung der Edge.

Unterschieden werden muss dabei zwischen der Differenz δ_i, die aus einer Diskontinuität und einer Nichtlinearität entsteht. Im Falle eines kombinierten Ray-Tree-Farb-Vergleichs resultiert die Diskontinuität aus voneinander abweichenden Ray-Trees (↗ Ab-

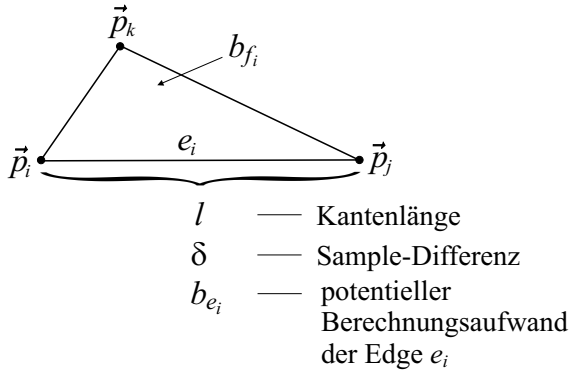


Abbildung 4.6: *Definitionen zur Bestimmung des potentiellen Berechnungsaufwandes.*

schnitt 3.3.1). Es gilt für den Ray-Tree-Vergleich:

$$\delta_{RT} = \begin{cases} 1 & : \quad RT_i \neq RT_j \\ 0 & : \quad sonst \end{cases}$$

Die Differenz, die sich aus der Nichtlinearität ergibt, wird dagegen durch den Farbvergleich bestimmt. Bei normierten Farbwerten gilt für den RGB- sowie HSV-Farbvergleich

$$\begin{aligned} \delta_{RGB} &= \frac{1}{3}(\delta_R + \delta_G + \delta_B), \qquad \delta_{RGB} \in [0,1], \\ \delta_{HSV} &= \frac{1}{3}(\delta_H + \delta_S + \delta_V), \qquad \delta_{HSV} \in [0,1]. \end{aligned}$$

Um eine gute Aussage über den potentiellen Berechnungsaufwand b_{e_i} einer Edge e_i treffen zu können, muss nun noch ihre Länge l_i ins Verhältnis gesetzt werden. Je größer l_i, desto größer ist der potentielle Berechnungsaufwand bei gleichem δ_i. Für den Fall des Ray-Tree-Vergleichs ergibt sich:

$$b_{e_i} = \delta_{RT} \frac{l_i}{l_{max}}, \qquad b_{e_i} \in [0,1] \tag{4.4}$$

mit l_{max} der maximal möglichen Kantenlänge, die aus der Diagonalen des Viewports resultiert und zur Normierung herangezogen wird. Entsprechendes gilt für den potentiellen Berechnungsaufwand beim Farb-Vergleich:

$$b_{e_i} = \delta_{RGB,HSV} \frac{l_i}{l_{max}}, \qquad b_{e_i} \in [0,1]. \tag{4.5}$$

Wesentlich dabei ist, dass im Falle verschiedener Ray-Trees nur Gleichung 4.4 zur Berechnung von b_{e_i} herangezogen wird. Gleichung 4.5 tritt erst in Kraft, wenn keine Sample-Differenz aufgrund des Ray-Tree-Vergleich gefunden oder dieser von vornherein gar nicht zur Bestimmung von δ_i einbezogen wurde. Die Bestimmung von b_{f_i} einer Primär-Face f_i ergibt sich schließlich aus

$$b_{f_i} = \frac{1}{3} \sum_{i=0}^{3} b_{e_i}.$$

Wurde für jede Primär-Face die Berechnung des potentiellen Berechnungsaufwands abgeschlossen, kann im Nachfolgenden zur Verteilung der Primär-Faces übergegangen werden. Jedem Slave soll pro Frame einmalig die Menge an Load zugeteilt werden, die im Verhältnis zu seiner Leistungsfähigkeit c_{Slave} steht. Ermittelt wird c_{Slave}, indem für jeden Slave die Anzahl an generierten Faces pro Zeiteinheit gemessen und dem Master mitgeteilt wird. Die Messung geschieht dabei für jeden Frame von Neuem. Schwankungen, wie sie durch verstärkte Ressourcennutzung im heterogenen Rechnersystem auftreten, können somit ausgeglichen werden.

Zur Bestimmung der Menge an Load b_{Slave_i}, dem jeden Slave abhängig von seiner Leistungsfähigkeit c_{Slave_i} zusteht, muss zunächst noch der Gesamt-Load b_{total} beziehungsweise die gesamte Leistungsfähigkeit c_{total} des verteilten Systems ermittelt werden. Es gilt:

$$b_{total} = \sum_{i=0}^{n} b_{f_i}, \quad c_{total} = \sum_{i=0}^{p} c_{Slave_i},$$

$$b_{Slave_i} = \frac{c_{Slave_i}}{c_{total}} \cdot b_{total},$$

mit n der Anzahl an Primär-Faces und p der Anzahl an Prozessoren. Abbildung 4.7 zeigt die Verteilung des Loads schematisch. Die Größe der Load-Pakete ist variabel, in Abhängigkeit von der aktuellen Leistungsfähigkeit eines Slaves. Lediglich im ersten Berechnungsdurchgang (Frame) werden alle Slaves mit identischer Leistungsfähigkeit initialisiert, so dass in diesem Falle auch die Paketgrößen gleichen Umfang aufweisen.

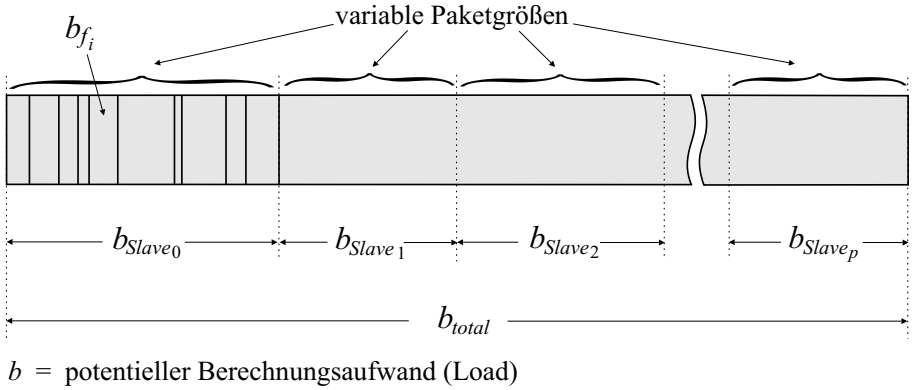


Abbildung 4.7: *Pre-Compute Load-Balancing. Zusammensetzung der Paketgrößen aus dem Berechnungsaufwand einzelner Faces.*

Vorteil dieses Konzeptes eines Load-Balancings ist insbesondere die einmalige Zustellung an Load pro Frame. Der Kommunikationsaufwand, der zur Verteilung des Gesamt-Loads notwendig ist, wird damit auf ein Minimum reduziert. Er findet lediglich einmalig zu Beginn und zum Ende der Berechnung eines Frames statt. Außerdem wirkt sich die Vorberechnung der Primär-Faces günstig auf eine zügige Darstellung aus. Der Pre-Compute-Schritt ermöglicht ein sogenanntes Preview, das einen ersten Eindruck des Finalbildes

unmittelbar zulässt. Nachteilig des Pre-Compute-Schrittes ist jedoch seine sequentielle Charakteristik. Erst nach dessen Beendigung kann mit der Verteilung des Loads, sprich mit der Parallelisierung des Algorithmus, begonnen werden.

Schwächen des Pre-Compute Load-Balancing liegen darüber hinaus in einem relativ trägen Reaktionsvermögen gegenüber sich stark verändernden Systembedingungen. Ändert sich die Leistungsverteilung des Systems auch innerhalb eines Berechnungsdurchganges, kann darauf nicht reagiert werden. Vor allem im Hinblick auf die Kommunikationsleistung kann diese Tatsache stark ins Gewicht fallen. Abrupte Schwankungen in der Übertragungsleistung sind in einem heterogenen Unternehmensnetzwerk keine Seltenheit. Selbst die Einbeziehung der Kommunikationszeit zwischen Master und Slave in die Berechnung der Paketgröße würde hier nur bedingt zur Verbesserung eines Load-Balancings führen.

Queue Load-Balancing

Das Queue Load-Balancing arbeitet, analog zu den Implementierungen in [PMSS99, WBWS01], mit einer gleichmäßigen Aufteilung des Gesamt-Loads. Es entstehen gleich große Arbeitspakete, die in einer Queue (Warteschlange) im Master aneinander gereiht werden (↗ Abbildung 4.8). Die Menge der Pakete p_n, sprich die Granularität des Gesamt-Loads, sollte dabei signifikant größer als die verfügbare Anzahl an Slaves s_p ($p_n \gg s_p$) sein. Ausgehend von der Menge aller zu berechnenden Primär-Faces wird jedem Paket p_i die gleiche Anzahl an Faces zugeteilt[8]. In Abhängigkeit der zu berechnenden Primär-Faces kann jedoch dabei die Anzahl der Faces pro Paket von Frame zu Frame variieren.

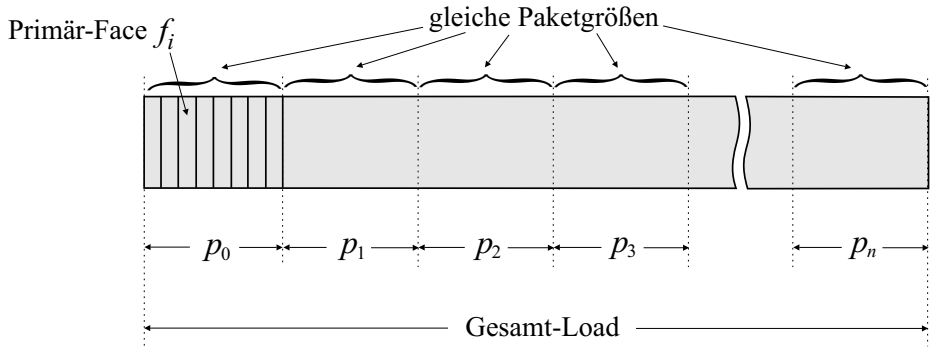


Abbildung 4.8: *Queue Load-Balancing. Pakete gleicher Größe werden an die Slaves verteilt.*

Wurden zu Beginn eines neuen Frames die zu berechnenden Primär-Faces auf die Pakete neu aufgeteilt, startet die Verteilung an die Slaves. Jedem Slave wird, beginnend mit dem ersten Paket der Queue, nach und nach ein Arbeitspaket zugewiesen. Sind alle Slaves bedient, wartet der Master auf die entsprechenden Berechnungsergebnisse. Der Slave, der

[8] Aufgrund der Division in gleiche Teilmengen kann es zu einem Rest kommen, so dass *ein* Paket eine abweichende Anzahl an Faces erhält.

seine generierten Faces an den Master zurück sendete, erhält ein neues Arbeitspaket. Diese Vorgehensweise wird solange wiederholt, bis alle Pakete der Queue abgearbeitet sind. Die Verteilung des Gesamt-Loads ist damit beendet. Hat der letzte Slave sein letztes Paket abgearbeitet und an den Master zurückgesendet, erfolgt schließlich die Darstellung der Szene. Die Berechnung des aktuellen Frames ist damit abgeschlossen.

Der Vorteil dieser Methode des Load-Balancings liegt in einer relativ guten Ausbalancierung aller etwaigen Ressourcenengpässe, die während der Berechnungsphase auftreten können. Insbesondere kurze Schwankungen sowohl in der Systemleistung einzelner Slaves als auch in der Kommunikationsleistung des Netzwerkes werden mit diesem Konzept automatisch berücksichtigt. Gleichzeitig findet darüber hinaus eine äußerst zuverlässige Ausbalancierung des anfallenden Berechnungsaufwandes statt, der sich aus dem Vertex-Tracing ergibt.

Problematisch wird jedoch die Anwendung des Queue Load-Balancing in Rechnernetzen mit geringen Datenübertragungsraten. Hier fällt der höhere Kommunikationsbedarf gegenüber dem Pre-Compute Konzept stark ins Gewicht. Der Gewinn, der aufgrund einer besseren Balancierung des System erzielt wird, relativiert sich möglicherweise durch längere Kommunikationszeiten.

Zusammenfassung

Welches Konzept eines Load-Balancings zum Einsatz kommt, hängt im Wesentlichen von der gegebenen Infrastruktur des Rechnersystems ab. Bietet das Netzwerk eine hohe Kommunikationsleistung, kann sicherlich das Queue Load-Balancing als bevorzugt eingestuft werden. In Systemen moderater Bandbreiten ist dagegen das Pre-Compute Load-Balancing geeigneter. Hier wirkt sich die auf ein Minimum reduzierte Kommunikation zwischen Master und Slave jeweils zum Anfang und Ende eines Berechnungsdurchgangs positiv aus.

Ein weiterer Vorteil des Pre-Compute Load-Balancings liegt in der gröberen Granularität des zu verteilenden Loads. Das heißt, bei gleicher Anzahl an Prozessoren wird der anfallende Load beim Pre-Compute in größere Regionen als beim Queue Load-Balancing gruppiert. Dies wirkt sich vor allem positiv auf die Qualität des verteilten Vertex-Tracings aus. Sind aufgrund des Refinement-Prozesses Zwangsverfeinerungen (↗ Abschnitt 3.3.2, 4.2) zwischen den Regionen einzelner Slaves notwendig, können diese nicht durchgeführt werden, da unter den Slaves keinerlei Kommunikation existiert. Je größer die zusammenhängenden Regionen von Primär-Faces sind, desto eher können mögliche Zwangsverfeinerungen erfolgen und damit weitere Features detektiert werden. Die Verteilung auf Basis des Pre-Compute Load-Balancing bietet insofern eine höhere Rendering-Qualität. Vorgehensweisen, die auf eine Lösung dieses Problems zielen, werden in Abschnitt 6.1.3 näher diskutiert.

4.4 Multi-Threading

Neben einer verteilten Berechnung des Vertex-Tracings mittels Message-Passing im heterogenen Netzwerk (Abschnitt 4.3), soll nun ein paralleler Ansatz auf Shared-Memory-Systemen anhand von Multi-Threading [Wal99, HH97] verfolgt werden. Ziel ist es, einen alternativen Ansatz bezüglich der Parallelisierung des Basisalgorithmus umzusetzen, der von den deutlich kürzeren Latenz- sowie Kommunikationszeiten des Multi-Threadings profitiert.

Grundlage für den hier verfolgten Multi-Treading-Ansatz ist die funktionale Dekomposition des Basisalgorithmus nach Abbildung 4.1. Die parallele Abarbeitung geschieht ausschließlich im Bereich des Samplings. Das heißt, die Verteilung der zu bearbeitenden Daten erfolgt auf der Ebene der Vertices. Jedem frei verfügbaren Thread wird ein Vertex zugewiesen, wobei Vertices aus den Primär-Vertices oder den neu generierten Vertices hervorgehen können. Das Sampling eines Vertex führt der verantwortliche Thread mittels Ray-Tracing durch.

Wie in Abschnitt 4.2 bereits herausgearbeitet, erlaubt diese Form der Parallelisierung eine hundertprozentige Abbildung der Funktionalität des sequentiellen Algorithmus. Anders als beim verteilten Konzept wird hier das Refinement weiterhin als sequentieller Teil-Prozess des Verfahrens betrachtet. Die adaptive Verfeinerung der Geometrie erfolgt damit global, über alle Primär-Faces hinweg. Es existieren keine Bereichsgrenzen zwischen einzelnen Primär-Faces, wie beim verteilten Ansatz der Fall. Auftretende Zwangsverfeinerungen zwischen den Primär-Faces können ohne weiteres berücksichtigt werden.

Nachteilig bei der hier vorliegenden Dekomposition ist die sequentielle Abhängigkeit zwischen den Refinement-Stufen. Vor dem Sampling neuer Vertices muss zunächst der Refinement-Prozess durchgeführt werden, um genau diese Vertices zu generieren. Es können Engpässe in Bezug auf die Verfügbarkeit von Vertices auftreten, wodurch eine effiziente parallele Ausführung gefährdet ist. Inwieweit diese Abhängigkeiten mit dem hier verfolgten Lösungsansatz minimiert werden können, soll in den weiteren Abschnitten näher beschrieben werden.

4.4.1 Konzept

Wie bereits weiter oben erwähnt, ist die Basis für das hier verfolgte Konzept eines Multi-Threadings die Verteilung des Samplings im Sinne einer Verteilung der zu berechnenden Vertices. Als Voraussetzung für eine effiziente Bereitstellung aller Vertices werden diese in eine Liste (Queue) einsortiert, auf die jeder Thread direkten Zugriff hat. Differenziert wird dabei zwischen den Primär-Vertices und den neu generierten Sekundär-Vertices.

Im Falle der Primär-Vertices werden diese ohne jegliche Sortierung geradewegs zu einer einfach verketteten Liste zusammengefügt (↗ Abbildung 4.9). Beginnend mit dem ersten Listenelement bedient sich jeder Thread mit einem noch nicht bearbeiteten Primär-Vertex und sorgt für dessen Sampling. Das Sampling erfolgt mittels Ray-Tracing. Wurde das letzte Element der Liste abgearbeitet, ist das Sampling aller Primär-Vertices abgeschlossen. Daran anschließend kann mit der ersten Refinement-Stufe begonnen werden.

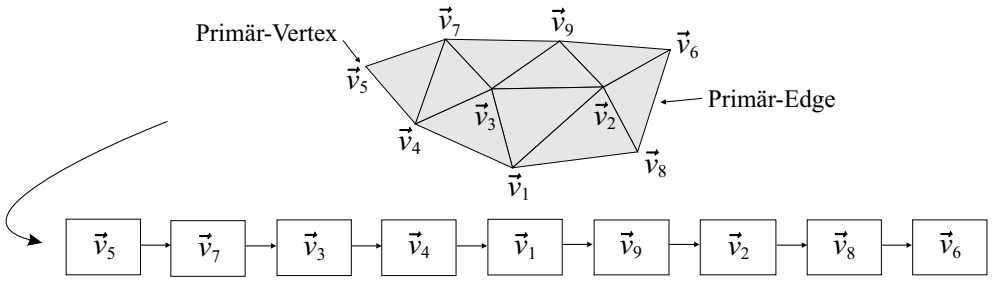


Abbildung 4.9: *Verknüpfung der Primär-Vertices zu einer einfach verketteten Liste.*

Bezüglich der parallelen Verarbeitung von Sekundär-Vertices muss ein leicht differenzierter Weg beschrieben werden. Jeder neue Sekundär-Vertex geht unmittelbar aus der Generierung einer neuen Sekundär-Edge hervor, so dass eine Abhängigkeit entsteht, die ein paralleles Sampling der Sekundär-Vertices erschwert. Laut Refinement-Prozess (↗ Abschnitt 3.3) wird jede Sekundär-Edge in die sogenannte Edge-List einsortiert, um eine geordnete Verarbeitung durch das Refinement zu garantieren[9]. Eine stringente Einhaltung der Sortierungsreihenfolge muss insofern auch bei einer Parallelisierung berücksichtigt werden. Neue Sekundär-Edges, die sich aufgrund ihrer Länge relativ weit vorn in die Edge-List einsortieren, müssen eine höhere Priorität bezüglich ihrer Bearbeitung erhalten. Erst wenn der zugehörige Sekundär-Vertex berechnet wurde, kann ein planmäßiges Refinement der Edge durchgeführt und diese aus der Edge-List entfernt werden.

Da die Edge-List als Binärbaum vorliegt, bedarf es der Generierung einer *linearen* Edge-List, um einen schnellen Zugriff auf die Listenelemente durch die Threads zu erhalten. Abbildung 4.10 zeigt die entsprechende Vorgehensweise dazu. Jede neue Sekundär-Edge wird anhand ihrer Länge l in den Baum einsortiert und findet somit an der entsprechenden Stelle Platz. Gleichzeitig erfolgt die Verknüpfung des Elements mit seinem Vorgänger und Nachfolger in Bezug auf l. Es entsteht eine lineare List, wie im unteren Teil der Abbildung 4.10 dargestellt. Im Falle eines maximal unbalancierten Binärbaumes kann dabei der Aufwand zur Einordnung des Elementes $n-1$ Schritte, mit n der Anzahl der Knoten, betragen. Entsteht ein idealer Binärbaum beträgt die Schrittweite 1. Da die einzuordnenden Edges in der Regel stark in ihrer Länge variieren, kann man durchaus von einem relativ gut balancierten Binärbaum sprechen. Der Aufwand zur Erstellung der Liste kann damit vernachlässigt werden.

Die Existenz der linearen Edge-List schafft die Voraussetzung für einen effizienten Zugriff beim Multi-Threading. Ähnlich zur Vorgehensweise bei den Primär-Vertices kann nun Element für Element der linearen Edge-List abgearbeitet werden, indem jeder enthaltene Sekundär-Vertex einem Sampling unterzogen wird. Eine Sonderstellung nimmt hier jedoch der Prozess des Refinements ein. Das Refinement darf nur auf dem ersten Element (der längsten Sekundär-Edge) angewendet werden, da nach dessen Abschluss die Edge

[9]Die Abarbeitung der Edge-List, beginnend mit der längsten Edge, ist eine Voraussetzung für ein schnelles Konvergenzverhalten des Refinement-Algorithmus (↗ Abschnitt 3.3).

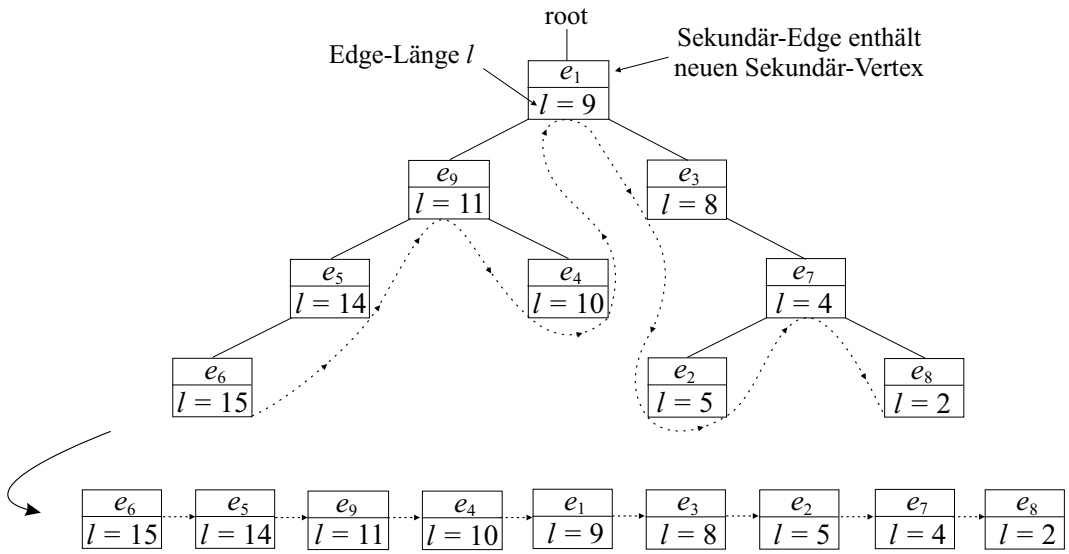


Abbildung 4.10: *Sortierung der neu generierten Edges (Sekundär-Edges) in Edge-List anhand der Länge l. Die Verknüpfung zu einer linearen Liste erfolgt simultan, um einen effizienten Zugriff durch Multi-Threading zu erlauben.*

aus der Liste entfernt wird. Inwieweit im Sinne eines Load-Balancings die Prozesse des Sampling und des Refinements effizient zusammenspielen können, beschreibt im Detail der folgenden Abschnitt.

4.4.2 Load-Balancing

Im weitesten Sinne wird beim Multi-Threading das Prinzip des Queue Load-Balancing verfolgt. Ziel ist es, jeden Thread optimal auszulasten, um einen möglichst maximalen Speed-Up des parallelen Systems zu erreichen. Grundlage für das Queue Load-Balancing bilden die im vorangegangenen Abschnitt beschrieben linearen Listen, die sowohl für den Fall der Primär-Vertices als auch der Sekundär-Vertices erzeugt wurden.

Beim Multi-Threading der Primär-Vertices (↗ Abbildung 4.11) erhält jeder frei verfügbare Thread das nächste in der Liste zu bearbeitende Element und sorgt für dessen Sampling. Zur Synchronisation der parallelen Abarbeitung dienen die Flags *InProgress* sowie *Sampled*. Findet aktuell das Sampling eines Vertex statt, signalisiert das Setzen des Flags *InProgress* die exklusive Behandlung dieses Vertex durch einen Thread. Der Zugriff ist für weitere Threads nicht gestattet. Wurde das Sampling des Vertex abgeschlossen, deutet *Sampled* auf die bereits abgeschlossene Berechnung des Vertex hin. Das Sampling aller Primär-Vertices ist beendet, nachdem *Sampled* bei jedem Element gesetzt wurde.

Abbildung 4.12 zeigt die kooperative Abarbeitung der Sekundär-Vertex-Liste. Im Gegensatz zur Bearbeitung der Primär-Vertex-Liste wird hier in Bezug auf die Verteilung des Loads grundsätzlich zwischen genau *einem* Primär-Thread und n Sekundär-Threads unterschieden. Während letztere sich lediglich für das Sampling eines Vertex verantwort-

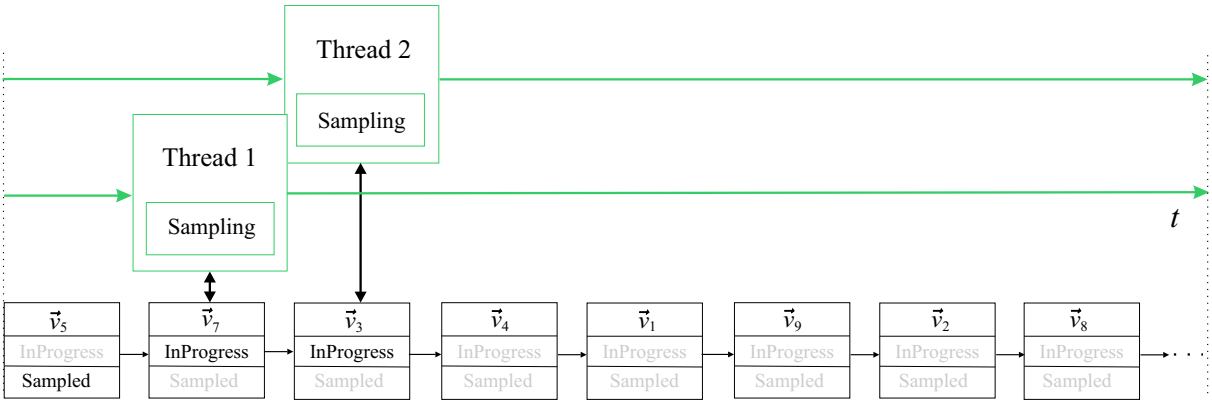


Abbildung 4.11: *Queue Load-Balancing bei der kooperativen Abarbeitung der Primär-Vertices. Jeder Thread durchläuft die Liste genau ein Mal. .*

lich zeichnen, übt der Primär-Thread zusätzlich noch die Aufgaben des Refinements aus. Der Grund für die Unterscheidung liegt in dem ausgeübten Spagat zwischen der zwingend sukzessiven Abarbeitung der sortierten Liste und der kontinuierlichen Einsortierung neuer Listenelemente, wie in Abschnitt 4.4.1 bereits erläutert. Jeder Sekundär-Thread durchläuft zyklisch die lineare Edge-List und sorgt für ein Sampling von noch nicht bearbeiteten Vertices. Zyklisch bedeutet in diesem Zusammenhang, dass, solange die Liste keine Elemente mehr besitzt, jeder Sekundär-Thread nach Erreichen des Listenendes wieder mit dem aktuell ersten Element der Liste beginnt und diese von Neuem durchläuft. Aufgabe des Primär-Threads ist es stattdessen, sich hauptsächlich um den Refinement-Prozess sowie die Bearbeitung der Liste zu kümmern. Er durchläuft die Liste genau ein Mal. Beginnend mit dem ersten Element, prüft der Primär-Thread auf ein bereits durchgeführtes Sampling des in der Sekundär-Edge enthaltenen Sekundär-Vertex. Erfolgte noch kein Sampling, übernimmt dieses der Primär-Thread, bevor die Edge dem Refinement übergeben und damit aus der linearen Edge-List gelöscht wird.

Neue Sekundär-Edges, die während des Refinement-Prozesses entstehen, werden vom Primär-Thread in die Liste einsortiert. Ein Beispiel dazu zeigt Abbildung 4.12 anhand der Edge e_9. Die Einordnung von e_9 erfolgt hier vor der aktuellen Bearbeitungsposition beider Sekundär-Threads. Haben beide Threads ihren Durchlauf der Liste noch nicht beendet, um von Neuem zu beginnen, fällt das Sampling von e_9 auf den Primär-Thread zurück. Der Vorteil dieser Methode ist eine verzögerungsfreie Abarbeitung der Liste selbst wenn neue Edges im oberen Teil der Liste einsortiert werden. Muss jedoch der Primär-Thread bei zu vielen Edges neben dem Refinement auch noch ein Sampling durchführen, können auf diese Weise ungleiche Lastverteilungen auftreten. Vor allem bei einer größeren Anzahl an Sekundär-Treads würde hier eine starke Disbalance entstehen, da sich die Laufzeit des Primär-Threads im Verhältnis zu den Sekundär-Threads entscheidend erhöht. Alternativ dazu könnten die Sekundär-Threads auch sofort mit einem neuen Durchlauf der Liste beginnen, sobald eine neue Edge eingefügt wird (↗ Abbildung 4.12). Sie würden so den

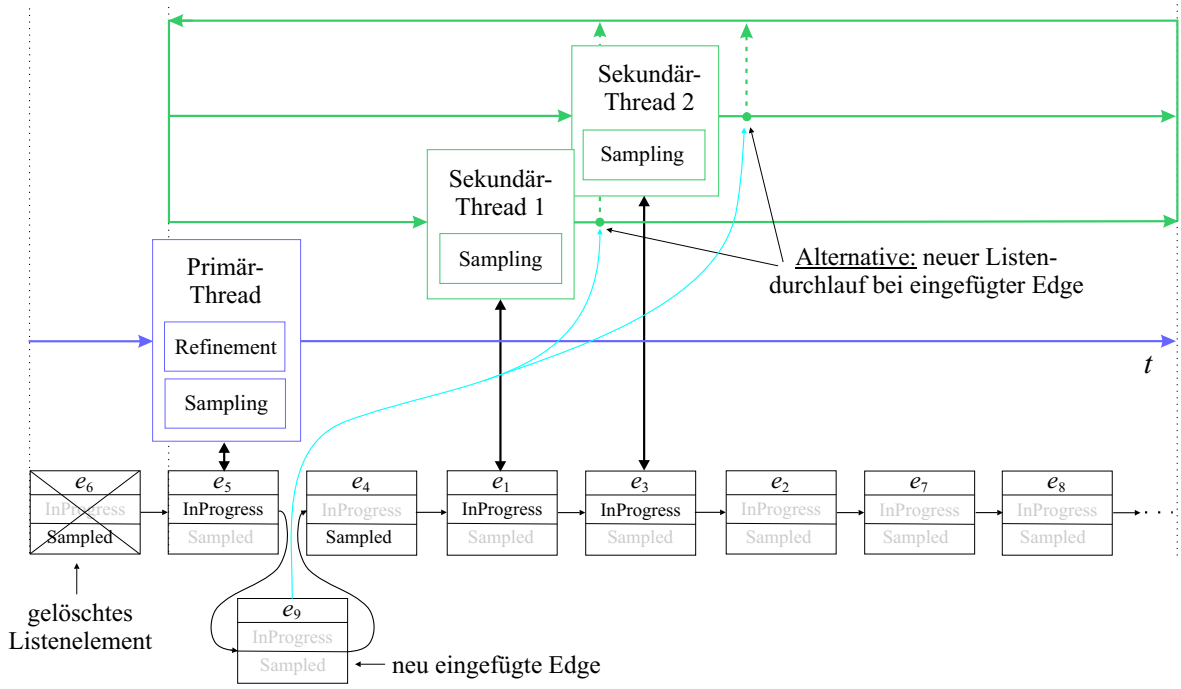


Abbildung 4.12: *Prinzip des Queue Load-Balancing bei der parallelen Abarbeitung der Sekundär-Edge-Liste. Aufgrund der einzuhaltenden Sortierungsreihenfolge wird hier zwischen Primär-Thread und Sekundär-Thread differenziert.*

Primär-Thread beim Sampling entlasten und für ein besseres Load-Balancing sorgen. Ein Nachteil dabei entsteht jedoch im erhöhten Kommunikationsaufwand der Sekundär-Threads. Diese müssen nun unter Umständen eine Vielzahl bereits berechneter Edges durchlaufen, bevor sie auf unbestimmte Edges in der Liste stoßen.

Welche Methode effizienter arbeitet, hängt im Wesentlichen von der Anzahl der verwendeten Prozessoren ab. Gibt es im Falle eines Dual-Prozessor-Systems nur einen Sekundär- und einen Primär-Thread, erzielt ein vollständiges Durchlaufen der Edge-List einen besseren Speed-Up, da sich hier der Kommunikationsaufwand minimiert. Im Falle mehrerer Sekundär-Threads ist jedoch die alternative Vorgehensweise aufgrund des verbesserten Load-Balancings im Vorteil. Sicher ist aber auch hier, dass der maximale Speed-Up mit dem Aufwand des sequentiellen Refinements begrenzt wird.

Die Gesamtberechnungszeit $T(p)$ für p Prozessoren, einem Primär-Thread und $p-1$ Sekundär-Threads beträgt schließlich

$$T(p) \;=\; \max_{1 \le i \le p} \left\{ T_R^i \;+\; T_K^i \;+\; T_L^i \right\} \tag{4.6}$$

mit T_R der eigentlichen Berechnungszeit, T_K der Kommunikationszeit und T_L der Latenz-

zeit. T_R setzt sich dabei je nach Art des Threads aus:

$$T_R = \begin{cases} T_{Refinement} + T_{Sampling} & : \text{ Primär-Thread} \\ T_{Sampling} & : \text{ Sekundär-Thread} \end{cases} \tag{4.7}$$

zusammen. Die Zeiten T_K und T_L sind von der Zugriffsgeschwindigkeit auf die lineare Liste abhängig. Vor allem bei größerem p kann der Zugriff auf den nächsten verfügbaren Vertex durch mehrfaches Abfragen bereits berechneter oder gerade in Berechnung befindlicher Vertices längere Zeit in Anspruch nehmen.

4.5 Kombination aus verteilt paralleler Berechnung

Eine logische Konsequenz aus den oben beschriebenen Konzepten eines verteilten sowie parallelen Vertex-Tracings ist die Kombination beider Verfahren. Insbesondere vor dem Hintergrund, dass der Einsatz von Rechner-Cluster an Charme gewinnt[10], bietet es sich an, sowohl eine Verteilung mittels Message-Passing im Rechnernetz als auch eine Parallelisierung mittels Multi-Threading im Shared-Memory zu verfolgen.

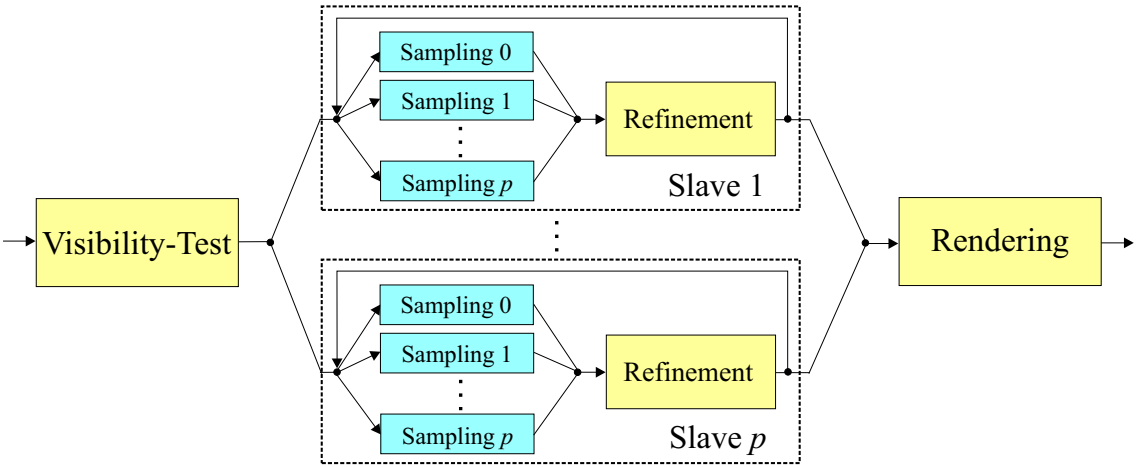


Abbildung 4.13: *Funktionales Prinzip eines verteilt parallelen Vertex-Tracings.*

Unter der Annahme, dass jeder Knoten des Rechner-Clusters mindestens zwei Prozessoren besitzt, ergibt sich die Dekomposition des Algorithmus in einem verteilt parallelen System entsprechend Abbildung 4.13. Analog zum beschriebenen Konzept aus Abschnitt 4.3 übernimmt auch hier zunächst der Master die Verteilung der Primär-Faces auf die beteiligten Slaves. Jeder Slave beginnt mit dem Vertex-Tracing seiner zugewiesen Faces. Im Gegensatz zu Abschnitt 4.3 erfolgt jedoch nun das Sampling der Vertices kooperativ mittels Multi-Threading. Die Anzahl der Threads je Slave ist dabei von der Anzahl

[10]Der verstärkte Einsatz von Rechner-Cluster lässt sich vor allem auf das gute Preis-Leistungsverhältnis von PC-Systemen zurückführen. In einem Netzwerk vereint, erreichen solche PC-Systeme zum Teil die Leistungsfähigkeit von Massiv-Parallelrechnern.

der lokalen Prozessoren abhängig. Die Gesamtskalierung des Systems entspricht somit der Summe der verfügbaren Prozessoren im Cluster. Nach Abschluss des Refinement-Prozesses werden die generierten Sekundär-Faces schließlich zum Master zurückgesendet, der für das Rendering derselben zuständig ist.

Es liegt auf der Hand, dass eine Kombination aus einem verteilt parallelen Vertex-Tracings eine effizientere Skalierung des Verfahrens im Rechner-Cluster erlaubt, als ein rein, mittels Message-Passing verfolgter Ansatz. Insbesondere in Netzwerken mit mehreren Shared-Memory-System kann hier von der weitaus geringeren Latenz- und Übertragungszeit des Multi-Threading profitiert werden. Das heißt, der Speed-Up des Gesamtsystems wächst bei gleicher Prozessoranzahl gegenüber dem reinen Message-Passing durch die Reduzierung des Kommunikationsaufwands.

Ferner begünstigt der verteilt parallele Ansatz die Darstellungsqualität des Vertex-Tracings. Geht man wiederum von einer fixen Anzahl an Prozessoren im System aus, kann die Segmentierung der Primär-Faces mit diesem Ansatz gröber gehalten werden. Die Anzahl der effektiven Slaves ist aufgrund der expliziten Nutzung der Shared-Memory-Systeme geringer, als bei einem ausschließlich verteilten Ansatz. Die gröbere Datensegmentierung reduziert das Problem einer bereichsübergreifenden Zwangsverfeinerung wie in Abschnitt 4.3.3 beschrieben.

4.6 Diskussion

Inwieweit sich die vorgestellten Konzepte eines verteilten beziehungsweise parallelen Vertex-Tracings in der Praxis bewähren, soll im Detail in Abschnitt 5.3 ausgewertet werden. Soviel kann aber an dieser Stelle bereits gesagt werden: Eine parallel kooperative Berechnung des Verfahrens bringt gegenüber der rein sequentiellen Variante einen signifikanten Laufzeitvorteil mit sich.

Trotz dieser Tatsache ist jedoch der inhärent sequentielle Charakter des Vertex-Tracings, in Bezug auf seine Skalierbarkeit deutlich zu spüren. Im Gegensatz zum Standard-Ray-Tracing verursacht insbesondere das adaptive Refinement neue sequentielle Abhängigkeiten, die eine Parallelisierung des Verfahrens nicht nur erheblich erschweren, sondern zugleich eine obere Schranke des erreichbaren Speed-Ups hervorrufen. Hinzu kommt, dass neben den sequentiellen Abhängigkeiten, die zwischen den einzelnen Stufen des Refinements existieren, auch noch Abhängigkeiten innerhalb einer Stufe auftreten können. Die Rede ist von Zwangsverfeinerungen, die zwischen adjazenten Faces entstehen. Ohne zusätzlichen Kommunikationsaufwand verursacht diese Eigenschaft vor allem beim verteilten Ansatz mittels Message-Passing die beschriebenen Probleme der Darstellungsqualität. Hier müsste ein regelmäßiger Austausch zwischen den Slaves erfolgen, um den notwendigen Zwangsverfeinerungen gerecht zu werden.

Während die Abhängigkeit, die durch Zwangsverfeinerung auftritt, lediglich einen höheren Kommunikationsbedarf hervorruft und insofern bei einer theoretisch unendlichen Kommunikationsgeschwindigkeit vernachlässigbar wäre, bleibt die sequentielle Abhängigkeit zwischen den Refinement-Stufen in jedem Fall gegenwärtig. So ist beim paralle-

len Ansatz mittels Multi-Threading die Anzahl der zu verteilenden Vertices durch die Refinement-Stufen begrenzt. Selbst bei vernachlässigtem Kommunikationsbedarf und hoher Prozessoranzahl muss vor dem Sampling jeweils ein geordnetes Refinement (Edge-List) erfolgen, so dass Idle-Zeiten der Prozessoren auftreten können.

Ein anderer Schwachpunkt, der ausschließlich beim verteilten Vertex-Tracing auftritt und hier nicht unerwähnt bleiben soll, ist die eingeschränkte Nutzung der Visibility-Tests. Zwar erfolgt der primäre Visibility-Test mittels ID-Buffer (↗ Abschnitt 3.4.1) durch den Master vor der Verteilung der Faces, jedoch kann der Test anhand des Z-Buffers (↗ Abschnitt 3.4.2) während des Refinement-Prozesses nicht durchgeführt werden. Dazu müsste jeder Slave pro Frame eine Kopie des Z-Buffers vom Master erhalten. Bei einer Auflösung von 1280x1024 Bildpunkte und einer Z-Buffer-Tiefe von 32 Bit entspräche dies 5 MB an zusätzlichen Daten, die jedem Slave jeweils zugesandt werden müssten. Alternativ könnte hier eine eigene Bestimmung der Visibility-Buffer durch den Slave in Betracht gezogen werden. Dies setzt jedoch die Existenz von Graphik-Hardware voraus. Im Hinblick auf die Verwendung von Multi-Prozessor-Maschinen ohne entsprechende Hardware wurde diese Alternative zunächst nicht weiter verfolgt. Sie stellt allerdings ein Thema weiterführender Arbeiten dar.

Kapitel 5

Testergebnisse und Praxisrelevanz

Das vorliegende Kapitel untersucht die qualitativen sowie quantitativen Eigenschaften des Vertex-Tracings anhand von Beispielszenarien. Als Referenz dazu dient ein Standard-Ray-Tracer, der parallel zu den Messungen herangezogen werden soll, um einerseits eine allgemein gültige Aussagen über die Leistungsfähigkeit des Vertex-Tracings treffen zu können und anderseits einen Vergleich mit anderen Ray-Tracing-Systemen zuzulassen. Eine wesentliche Voraussetzung dabei ist die Tatsache, dass der Vertex-Tracer den gleichen Basis-Ray-Tracer für das Sampling der Radiance-Funktion nutzt wie der Standard-Ray-Tracer. Die Kosten zur Bestimmung eines Sample-Wertes korrelieren also zwischen Vertex-Tracing und Standard-Ray-Tracing. Die Differenzierung beider Verfahren erfolgt hier vielmehr über die Anzahl der ermittelten Samples und deren Charakteristik.

Zunächst steht in Abschnitt 5.1 die Untersuchung der Rendering-Charakteristik und -Qualität des Vertex-Tracings im Vordergrund. Hier gilt es unter anderem die Zuverlässigkeit bezüglich der Detektion von Nichtlinearitäten und Diskontinuitäten anhand verschiedener Kriterien des Refinements zu bestimmen. Eine Analyse des daraus resultierenden Laufzeitverhaltens des Gesamtsystems folgt im Anschluss in Abschnitt 5.2, bevor in Abschnitt 5.3 typische Eigenschaften wie Speed-Up oder Load-Disbalance des verteilten beziehungsweise parallelen Ansatzes untersucht werden.

5.1 Rendering-Charakteristiken und -Qualität

Zur Analyse des Vertex-Tracings in Bezug auf Rendering-Qualität und typische Eigenschaften des Samplings soll die Tischszene aus Abbildung 5.1a dienen. Die Szene besteht aus 18 Objekten mit einer Gesamtsumme von circa 20'000 Polygonen. Unter den Objekten sind 3 Objekte als VT-Objekte definiert. Darunter zählen die beiden Tassen und die Tischoberfläche. Weiterhin wurde *eine* Punktlichtquelle definiert, die sich innerhalb des Schirmes der Tischlampe befindet.

Abbildung 5.1c zeigt die Szene aus einer anderen Kameraperspektive mittels Vertex-Tracing. Als Kriterium für das adaptive Sampling und Refinement dient der Ray-Tree eines ermittelten Samples in Verbindung seines resultierenden RGB-Wertes. Die Anwendung eines derart kombinierten Ray-Tree/RGB Vergleichs erlaubt schon eine relativ zu-

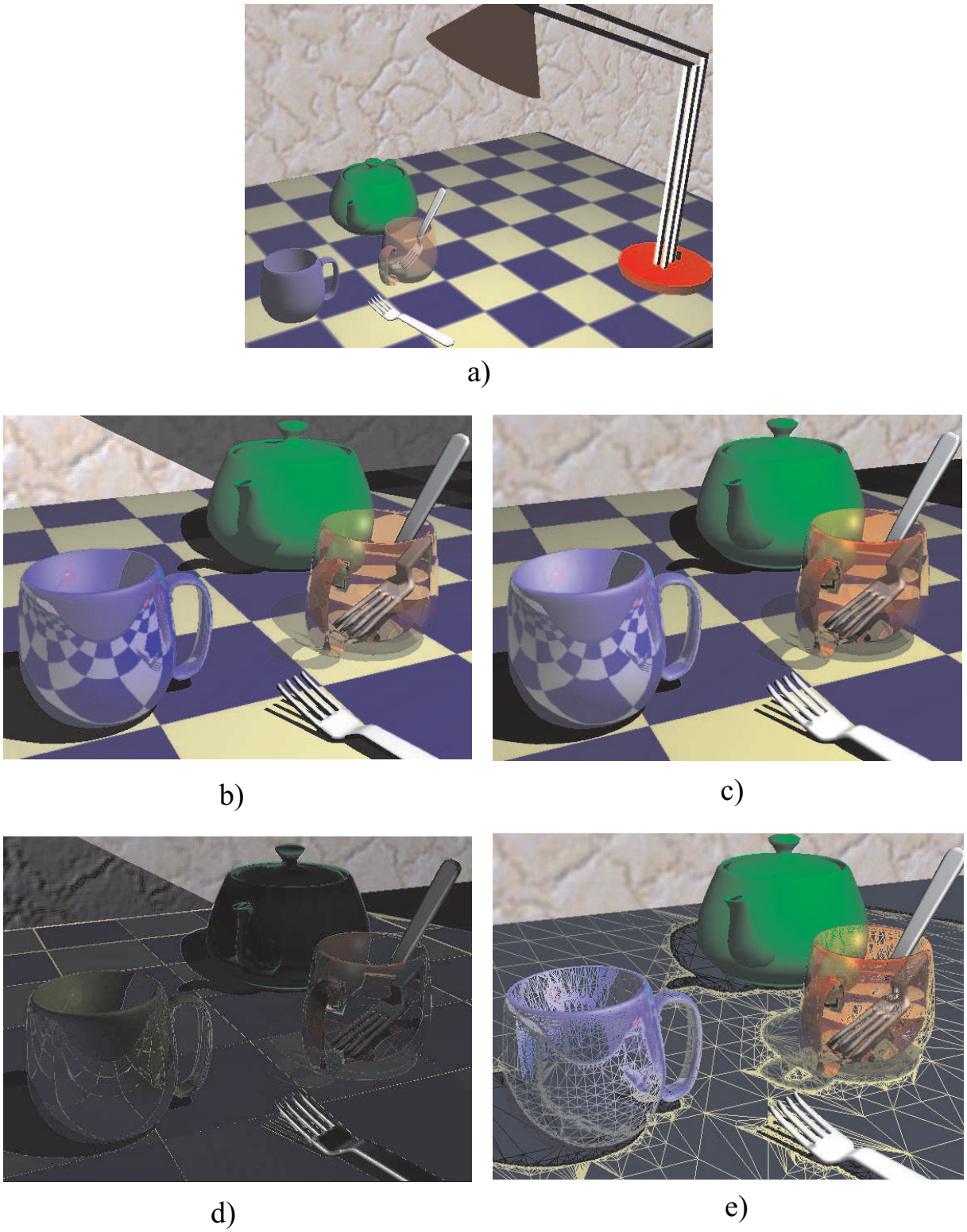


Abbildung 5.1: *Vertex-Tracing im Vergleich zu OpenGL und Standard-Ray-Tracing anhand der Tischszene. a) Rendering in OpenGL aus der Totalen. b) Ray-Tracing. c) Vertex-Tracing. d) Differenzbild (Differenzbetrag) von b und c. d) Visualisierung der generierten Faces beim Vertex-Tracing.*

verlässige Detektion von Diskontinuitäten und Nichtlinearitäten (↗ Abschnitt 5.1.1). Das Abbruchkriterium der minimalen Edge-Länge wurde auf 1 gestellt, so dass das Vertex-Tracer wie der Standard-Ray-Tracer eine Pixel-genaue Darstellung erreicht.

Als Referenz zu Abbildung 5.1c fungiert Abbildung 5.1b, die mit Hilfe eines Standard-Ray-Tracers erzeugt wurde. Sowohl b) als auch c) nutzen dabei den gleichen Basis-Ray-Tracer, so dass die Bestimmung der Samples inklusive ihres Shadings identisch erfolgt. Eine gute Vergleichsmöglichkeit zwischen dem Vertex-Tracing und Standard-Ray-Tracing bietet Abbildung 5.1d in Form des Differenzbetrags zwischen beiden Bildern. Ein Großteil der erkennbaren Kanten resultiert hier aus einer Ungenauigkeit, die durch das eigentliche Rendering hervorgerufen wird, wie am Beispiel des Schachbrettmusters erkennbar ist. Während die Erstellung von Abbildung b) komplett in Software erfolgte, wurden die generierten Faces des Vertex-Tracings in c) vollständig durch Rasterisierungs-Hardware dargestellt. Unterschiede im Shading leiten sich zum Teil aus der Modifikation von Gleichung 3.20 in Gleichung 3.22 ab. Differenzen, die aufgrund von unentdeckten Diskontinuitäten beziehungsweise Nichtlinearitäten resultieren, sind hingegen kaum auszumachen. Einzig auffällig ist die inverse Schattierung der hinteren Wand. Diese ist in 5.1c kein VT-Objekt und wird demzufolge durch übliches OpenGL-Rendering dargestellt. Betrachtet man jedoch lediglich die qualitative Darstellung der VT-Objekte kann letztendlich von einer zuverlässigen Detektion in diesem Beispiel gesprochen werden.

Abbildung 5.1e zeigt die während des Vertex-Tracings generierten Faces in Wireframe-Darstellung. Auffällig ist hier vor allem die Dichte der Vernetzung jeweils im Bereich der Tassen. Inwiefern in diesem Fall noch ein effizientes Vertex-Tracing vorliegt, soll in Abschnitt 5.2 diskutiert werden.

Weitere typische Charakteristiken des Vertex-Tracings in Bezug auf das Sampling der Radiance-Funktion präsentiert Abbildung 5.2. Beginnend mit der oberen Reihe (5.2a, 5.2b) wird hier die Wirkung des Visibility-Tests demonstriert. Das Rendering und Shading des Utah-Teapots erfolgt vollständig in OpenGL. Die Tischplatte hingegen ist ein VT-Objekt, dessen Radiance-Funktion adaptiv abgetastet wird. Deutlich zu erkennen, ist das dichtere Sampling im Bereich der Schattenkanten, jedoch nur dort, wo diese auch sichtbar sind. Ein weiteres Sampling hinter dem Teapot entfällt aufgrund des Visibility-Tests aus Abschnitt 3.4.1 und 3.4.2.

Eine differenzierte Visualisierung der detektierten Nichtlinearitäten und Diskontinuitäten ist in Abbildungen 5.2c und 5.2d (mittlere Reihe) zu finden. Rot dargestellte Punkte repräsentieren generierte Samples aufgrund einer Diskontinuität und gelbe Punkte aufgrund einer Nichtlinearität. Bei ausgeschalteter Reflexion und Transmission kann man besser eine Differenzierung beobachten. Nichtlinearitäten treten überall dort auf, wo High-Lights entstehen (oberer Tassenrand oder auf der Tasseninnenseite). Diskontinuitäten sind dagegen ausschließlich bei Schattenkanten anzutreffen. Auch ist in Abbildung 5.2d gut die fehlende Abtastung der eigentlichen Objektesilhouette zu erkennen. Trotz der Tatsache, dass diese ebenfalls Diskontinuitäten der Radiance-Funktion darstellen, geschieht hier keine weitere Betrachtung, da die Objektgeometrie implizit durch die Verfolgung von Vertices beim Vertex-Tracing gegeben ist (↗ Abschnitt 3.1). Die untere Reihe in Abbildung 5.2 zielt auf das Verhalten des Vertex-Tracings gegenüber von Tex-

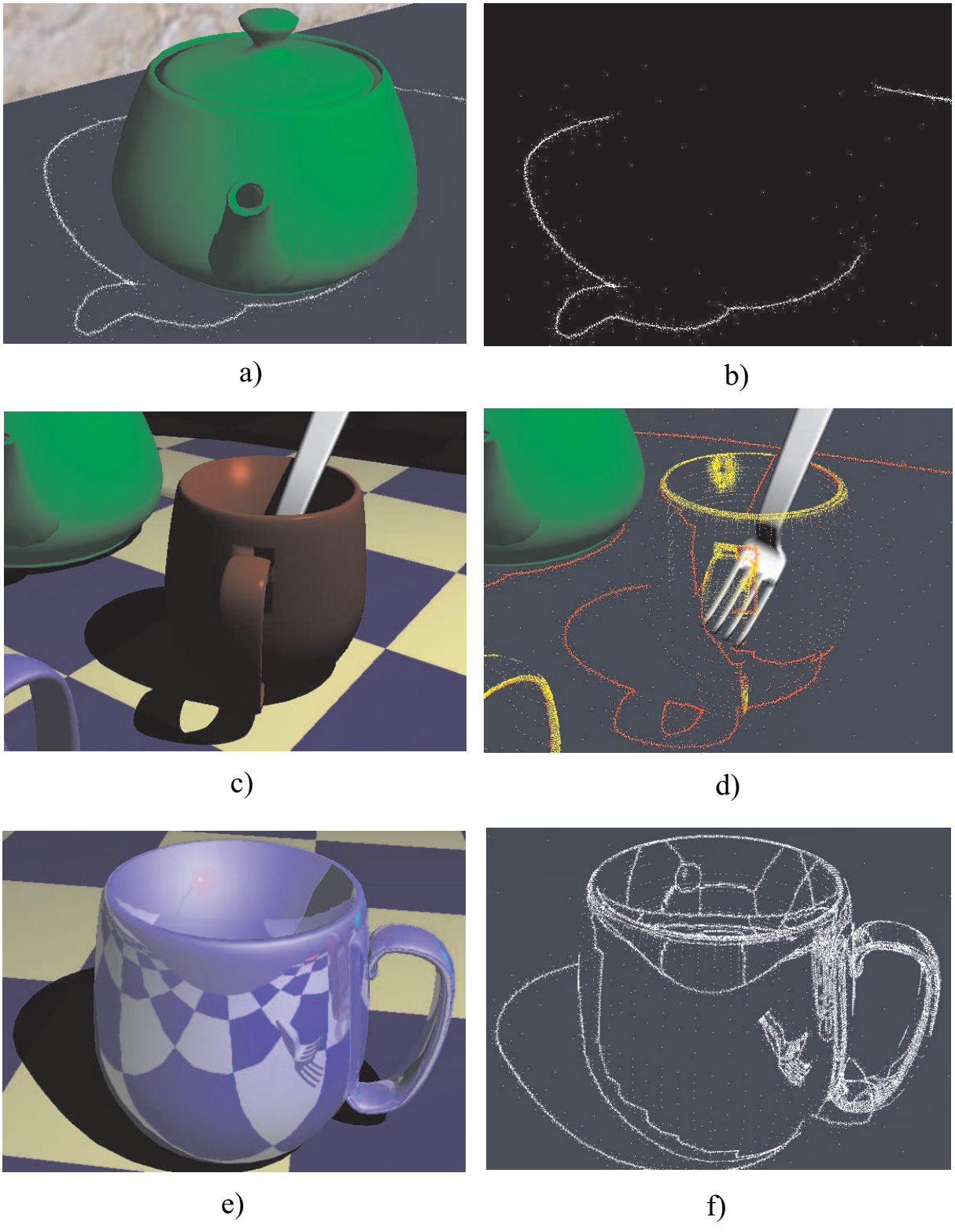


Abbildung 5.2: *Charakteristiken des Vertex-Tracings. a) und b) Sampling erfolgt nur im sichtbaren Bereich. c) und d) Darstellung detektierter Nichtlinearitäten (gelbe Punkte) und Diskontinuitäten (rote Punkte). e) und f) Es erfolgt kein Sampling der Radiance-Varianz von Texturen. Die Spiegelung des Schachbrettmusters in der Tasse benötigt keine zusätzlichen Samples.*

turen. Trotz auftretender Radiance-Varianz in der Schachbrett-Textur (Diskontinuitäten an den Grenzen zwischen hellen und dunklen Feldern) findet kein dichteres Sampling in der Textur-Reflexion der Tasse statt. Der Bereich der sich spiegelnden Textur ist in 5.2f nur spärlich mit Samples belegt.

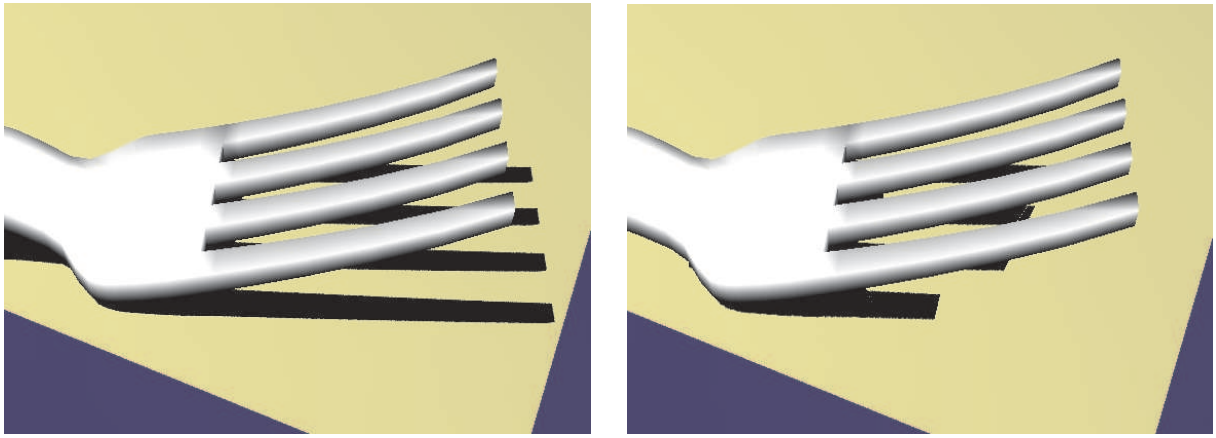


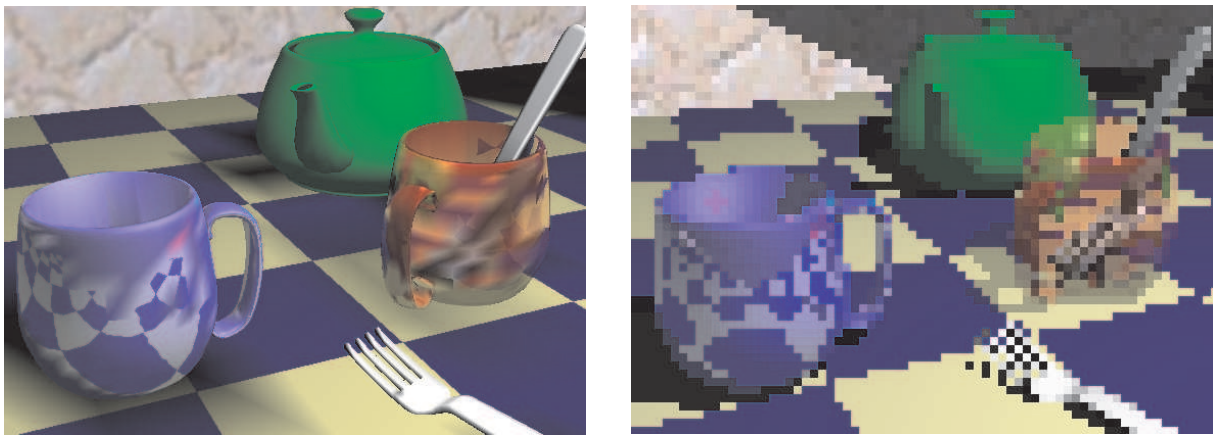
Abbildung 5.3: *Wirkungsweise der Zwangsverfeinerung: a) Mit Zwangsverfeinerung. b) Ohne Zwangsverfeinerung.*

Abbildung 5.3 demonstriert eine weitere bedeutende Eigenschaft des Verfahrens. Wie in Abschnitt 3.3.2 bereits diskutiert, trägt die sogenannte Zwangsverfeinerung von Edges entscheidend für eine robuste Detektion von Diskontinuitäten und Nichtlinearitäten bei. Während in Abbildung 5.3a mit aktivierter Zwangsverfeinerung der Schatten der Gabel vollständig erkannt wurde, treten in 5.3b deutliche Defekte auf. Der Schatten wird lediglich an den Stellen dargestellt, an denen er explizit eine Differenz zwischen zwei Vertices einer Edge verursacht.

Qualitative Vorteile des Vertex-Tracings gegenüber dem Standard-Ray-Tracing treten vor allem bei einer Preview-Erstellung auf, sollten beide Verfahren progressiv vorgehen. Wie in Abbildung 5.4a deutlich erkennbar, werden durch die inhärente Eigenschaft des Vertex-Tracings alle Objektsilhouetten korrekt dargestellt. Im Falle des Previews beim Ray-Tracing (Abbildung 5.4b) treten starke Aliasing-Artefakte auf, die aus einem groben uniformen Sampling im Bildraum resultieren. Das Vertex-Tracing bietet in diesem Fall eine bessere visuelle Qualität, da sich das Sampling nur auf das Innere der VT-Objekte beschränkt und zudem non-uniform erfolgt.

5.1.1 Ray-Tree, RGB, HSV

Qualitätsunterschiede, wie sie bei der Verwendung verschiedener Refinement-Kriterien (↗ Abschnitt 3.3.1) entstehen, verdeutlichen die nachfolgenden Abbildungen 5.5 und 5.6. Anhand des Refinement-Kriteriums wird entschieden, wann eine Edge unterteilt werden



a) b)

Abbildung 5.4: *Qualitative Unterschiede im Preview: a) Vertex-Tracing. b) Ray-Tracing.*

muss. Differenziert wird dabei zwischen dem Vergleich von Intensitäten in Form des RGB- beziehungsweise HSV-Tests und dem Vergleich des Ray-Trees. Während sich letzterer ausschließlich für die Detektion von Diskontinuitäten eignet, ist der Intensitätsvergleich vor allem auch für Nichtlinearitäten verwendbar.

Die Untersuchung der Kriterien soll anhand der Darstellungsqualität der blauen Tasse aus der Tischszene erfolgen. Das Augenmerk liegt dabei insbesondere auf der Reflexion der Gabel sowie der Tischlampe in Verbindung ihres umgebenen High-Lights im inneren der Tasse. Beginnend mit Abbildung 5.5a erfolgt zunächst ein reiner Ray-Tree-Vergleich. Deutlich erkennbar ist hier die inkorrekte Darstellung des High-Lights links neben der reflektierenden Tischlampe. In diesem Bereich besitzt jeder Vertex den gleichen Ray-Tree, so dass kein weiteres Refinement erfolgt und Interpolationsfehler auftreten. Im Gegensatz dazu wurde jedoch die Gabel korrekt dargestellt. Hier ermöglicht der Ray-Tree eine sicher Detektion der Diskontinuität in Form der sich reflektierenden Gabel.

Die Anwendung des RGB-Vergleichs in 5.5b zeigt dagegen deutliche Verwaschungen sowohl bei der Darstellung der Gabel als auch der Tischlampe. Beide Diskontinuitäten konnten aufgrund ihres marginalen Farbunterschieds nicht vollständig detektiert werden. Die Darstellung des High-Lights ist hingegen korrekt. Hier genügt der Intensitätsunterschied, um ein ausreichendes Refinement zu gewährleisten. Ähnlich zum RGB-Vergleich verhält sich der HSV-Vergleich aus 5.5c. Einziger Unterschied ist hier die bessere Detektion der reflektierenden Lampe. Die Bewertung der Komponenten des HSV-Modells in Form des Farbwinkels, der Sättigung und der Helligkeit ermöglicht eine bessere Analyse der Intensität. Obwohl weniger Vertices als beim RGB-Vergleich generiert wurden, ist die Darstellung qualitativ hochwertiger.

Eine Kombination aus verschiedenen Kriterien zeigt Abbildung 5.6. Im Falle eines gemeinsamen Ray-Tree/RGB-Vergleichs (a) wurde bereits eine gute Detektion aller Nichtlinearitäten und Diskontinuitäten erreicht. Sowohl die Gabel, die Tischlampe als auch das

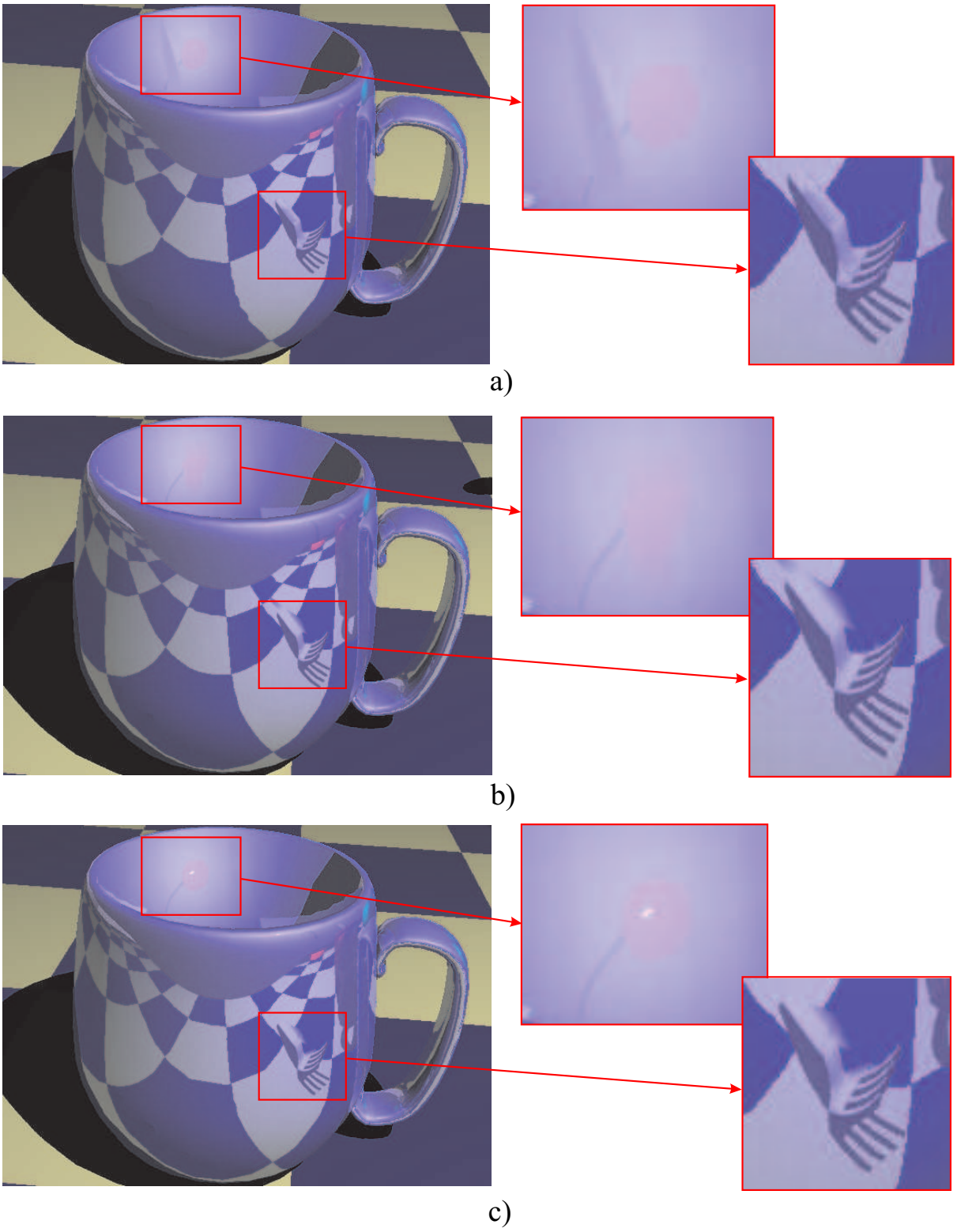


Abbildung 5.5: *Gegenüberstellung der Refinement-Kriterien. a) Ray-Tree, 22470 generierte Vertices, b) RGB, 23220 generierte Vertices und c) HSV, 23058 generierte Vertices*

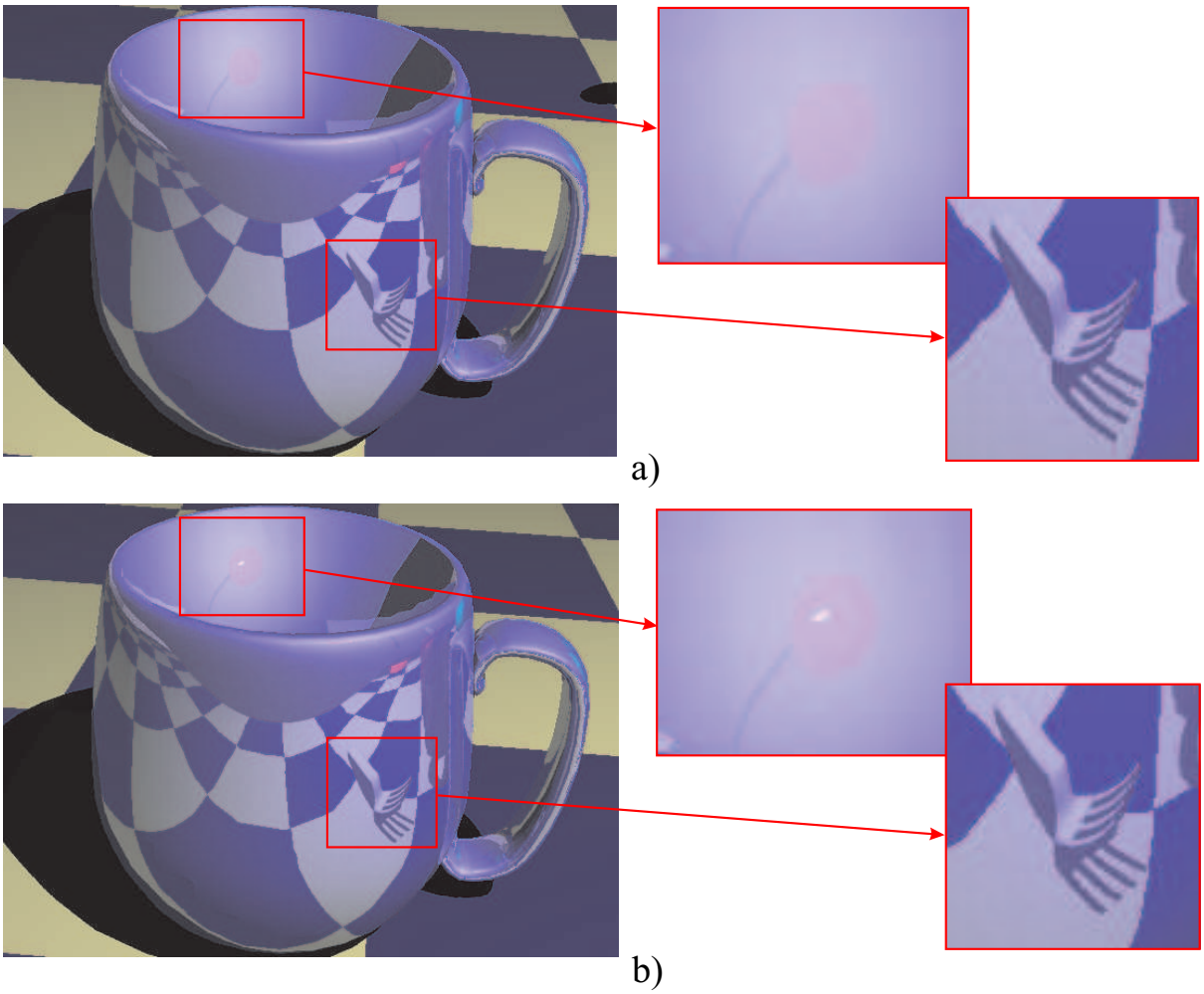


Abbildung 5.6: *Kombination von Refinement-Kriterien. a) Ray-Tree/RGB, 28535 generierte Vertices, b) Ray-Tree/HSV, 28037 generierte Vertices*

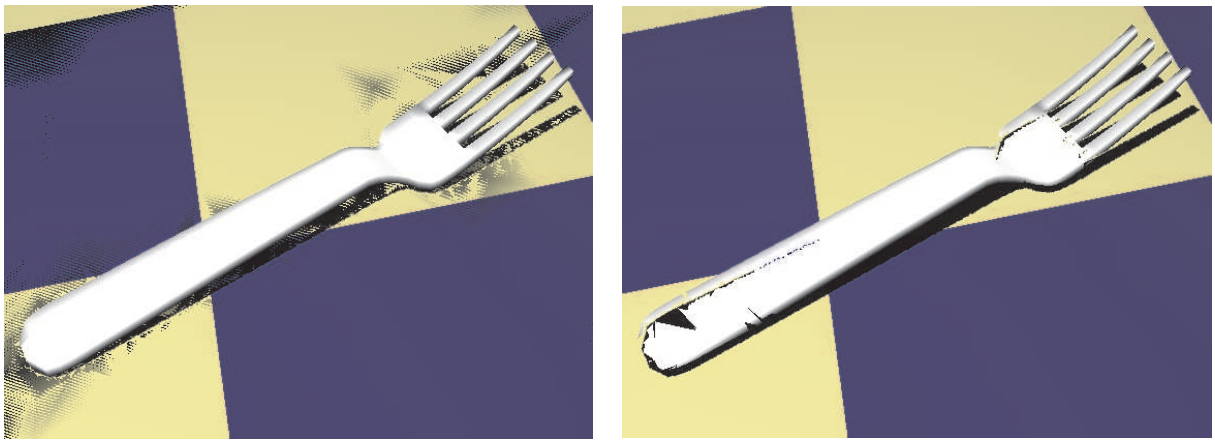
umgebene High-Light sind weitestgehend korrekt dargestellt. Hochwertiger ist allerdings ein kombinierter Ray-Tree/HSV-Vergleich, wie es Abbildung 5.6b demonstriert.

5.1.2 Rasterisierungsfehler

Darstellungsfehler, die aufgrund der finalen Rasterisierung (Scan-Line-Verfahren) der generierten Faces mittels OpenGL-Hardware auftreten, sind Gegenstand der folgenden Betrachtungen. Zunächst soll in diesem Zusammenhang die Qualität des progressiven Rendering-Konzeptes aus Abschnitt 3.5.1 untersucht werden.

Die Einführung der Methode *Overdraw* ermöglicht einen sukzessiven Aufbau der Szene bis zum finalen Bild, ohne dabei den Frame-Buffer nach jedem Refinement-Schritt vollständig von Neuem (wie beim *Redraw* der Fall) aufbauen zu müssen. Die Überlage-

rung der neuen Faces wird dabei durch die Bildungsvorschrift 3.16 erzielt, die sich über den konstanten Faktor f und der Anzahl der Refinement-Stufen k steuern lässt. Abbildung 5.7 zeigt beispielhaft die Auswirkung verschiedener f auf die Rendering-Qualität bei einer fixen Anzahl von 5 Refinement-Stufen. Wurde f wie in 5.7a zu klein gewählt, entsteht

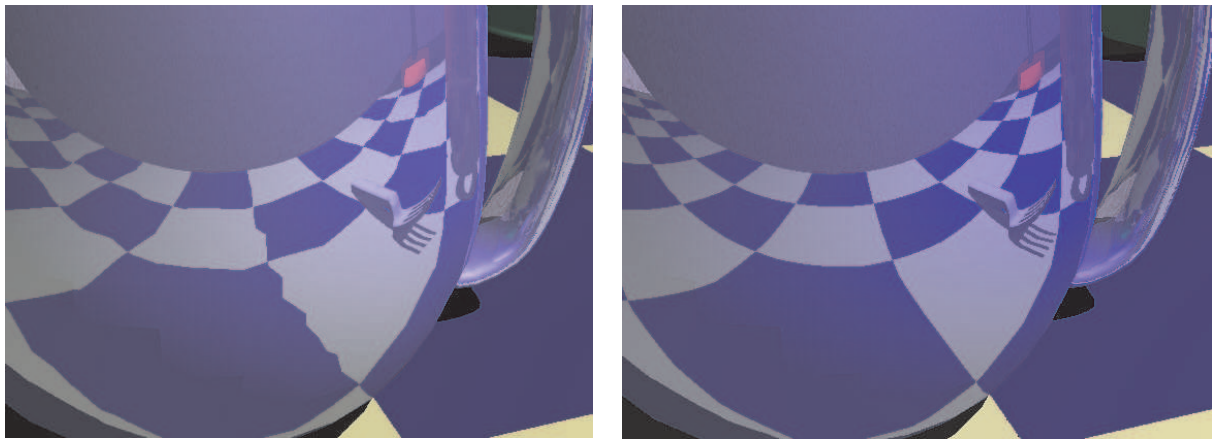


a) $f = 0.00001, k = 5$ b) $f = 0.01, k = 5$

Abbildung 5.7: *Rasterisierungsfehler beim progressiven Rendering mittels Overdraw in Abhängigkeit des Faktors f und k.*

das typische Z-Fighting zwischen zueinander konkurrierenden Flächen. Ein zu großes f verursacht hingegen ein zu hohes „übereinander Stapeln“ der Faces, so dass Defekte in darüber liegenden Objekten auftreten können. In 5.7b kommt es daher zum Überzeichnen der Gabel durch das Rendering der generierten Schatten-Polygone. In diesem Fall spielt jedoch auch die Anzahl der visualisierten Refinement-Stufen eine wesentliche Rolle. Selbst bei kleinem f kann der Effekt des Überzeichnens ohne Weiteres auftreten, wenn nur k genügend wächst. Das heißt, sowohl k als auch f müssen stets an die Gegebenheiten der Szene sowie der Hardware individuell angepasst werden, um eine maximale Rendering-Qualität im Overdraw-Modus zu erzielen. Im Beispiel der Abbildung 5.7 erwies sich $f = 0.0001$ und $k \leq 20$ als akzeptabel.

Ein anderer Effekt, der aufgrund der Rasterisierung hervortritt, ist in Abbildung 5.8 dargestellt. Im Speziellen zeigt Abbildung 5.8a ein zum Teil perspektivisch inkorrektes Abbild der sich spiegelnden Tisch-Textur im Gegensatz zur korrekten Darstellung mittels Ray-Tracing in 5.8b. Insbesondere im unteren Bereich der Tasse, indem die Streckung der Textur am größten ist, resultiert auch die größte Verzerrung. Verstärkt tritt dieser Effekt auf, wenn eine sich reflektierende oder transmittierende Textur durch nur wenige Samples dargestellt wird (hier durch den alleinigen Einsatz des Ray-Tree-Vergleichs hervorgerufen). Ursache für die Verzerrungen ist die falsche perspektivische Korrektur der Hardware im Falle der reflektierenden oder transmittierenden Textur. Wie diesem Problem begegnet werden kann, wurde im Rahmen dieser Arbeit nicht näher untersucht und stellt insofern ein Thema für weitere Arbeiten dar.



a) b)

Abbildung 5.8: *a) Inkorrekte perspektivische Darstellung von reflektierten beziehungsweise transmittierten Texturen beim Vertex-Tracing. b) Korrekte Darstellung beim Ray-Tracing.*

5.2 Laufzeitcharakteristiken

Zur Analyse des Laufzeitverhaltens beim Vertex-Tracing wurde wie im vorangegangenen Abschnitt die in Abbildung 5.1 vorgestellte Tischszene in einer Auflösung von 1024x1024 herangezogen. Die eingestellte Kameraperspektive der nachfolgenden Messungen entspricht dabei der Abbildung 5.1b, so dass sich die vordefinierten VT-Objekte (Tassen, Tischplatte) im Vordergrund befinden. In Bezug auf die Charakteristik der Tischszene kann bei dieser Perspektive von einem Worst-Case für das Vertex-Tracing gesprochen werden. Durch den spekularen Charakter der Tassen beziehungsweise der zusätzlichen Transparenz der roten Tasse befindet sich ein hoher Anteil an potentiellen Diskontinuitäten und Nichtlinearitäten im Erfassungsbereich der Kamera, so dass ein hoher Prozentsatz der Bildebene abgetastet werden muss.

Eine erste Analyse zielt auf das Laufzeitverhältnis zwischen den im Vertex-Tracing enthaltenen Komponenten, die sich aus dem Sampling, dem Refinement und der Rekonstruktion (↗ Kapitel 3) definieren. Abbildung 5.9 zeigt dazu eine Gegenüberstellung verschiedener Shading-Einstellungen des Vertex-Tracings sowie die typische Charakteristik eines Standard-Ray-Tracers. Während sich letzterer fast ausschließlich aus dem Sampling zusammensetzt, verteilt sich die Laufzeit beim Vertex-Tracing zum Teil auch auf die Komponenten Refinement und Rekonstruktion. Unterschiede zwischen den einzelnen Varianten beim Vertex-Tracing resultieren insbesondere aus den verschieden Kosten zur Bestimmung der Samples, die sich wiederum aus den verschiedenen Shading-Einstellungen ergeben. Die Anzahl der zu bestimmenden Samples hat dabei kaum Einfluss auf die Verschiebung des Verhältnisses. Im Falle von +S +R +T (Berechnung von Schattenfühlern, Reflexionsstrahlen und Transmissionsstrahlen) erhöht sich die Anzahl der Samples auf-

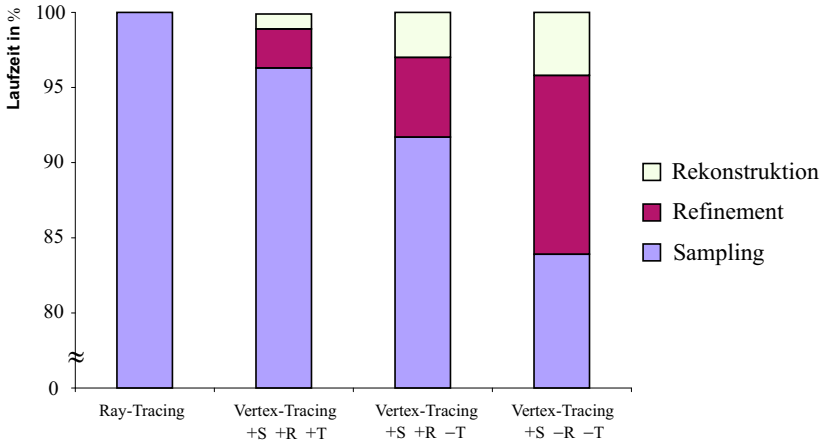


Abbildung 5.9: *Laufzeitverhältnisse zwischen den Komponenten Sampling, Refinement und Rekonstruktion beim Vertex-Tracing am Beispiel der Tischszene. +/− steht für einbeziehungsweise ausgeschaltete Schattenfühler S, Reflexionsstrahlen R und Transmissionsstrahlen T.*

grund der Zunahme von Diskontinuitäten und Nichtlinearitäten im Vergleich zu +S −R −T (nur Schattenfühler). Der Aufwand des Refinements als auch der Rekonstruktion steigt proportional, so dass das Laufzeitverhältnis zwischen +S −R −T und +S +R +T identisch bleibt. Da jedoch die Kosten zur Bestimmung der Samples im Fall von +S +R +T überproportional wachsen (rekursives Ray-Tracing), überwiegt zunehmend das Sampling. Die Kosten für Refinement und Rekonstruktion bewegen sich zusammen nur noch unter 4 Prozent.

Abbildung 5.10 demonstriert die Abhängigkeit des Vertex-Tracings von den Kosten des Samplings in einem ähnlichen Kontext. Da die Anzahl der geschossenen Strahlen den Gesamtkosten des Samplings entspricht, kann man hier die angesprochene Abhängigkeit des Vertex-Tracings beobachten. Verdoppelt sich die Anzahl der Strahlen (zwischen +S +R −T und +S +R +T), verdoppelt sich gleichermaßen die Laufzeit des Verfahrens, unabhängig davon, wie viele Samples zusätzlich bestimmt werden mussten. Das Vertex-Tracing weist damit in diesem Zusammenhang eine ähnliche Charakteristik wie das Standard-Ray-Tracing auf. Es skaliert linear mit der Anzahl der geschossen Strahlen.

Eine andere Vergleichsmöglichkeit bietet Abbildung 5.10 in Bezug auf die Gesamtlaufzeit zwischen Vertex-Tracing und Ray-Tracing. Der Faktor bewegt sich hier zwischen 6.8 und 86 zugunsten des Vertex-Tracings. Die deutliche Abschwächung des Faktors bei +S +R −T und +S +R +T ist wiederum auf das Verhältnis der geschossenen Strahlen zurückzuführen. Während beim Standard-Ray-Tracing lediglich bei einigen vorhandenen Samples Strahlen hinzukommen, entstehen beim Vertex-Tracing Samples völlig neu und zwar dort, wo neue Diskontinuitäten sowie Nichtlinearitäten entstehen. Die unterschiedlichen Faktoren zwischen Laufzeit und Strahlenanzahl (6.8 : 9.2 bei +S +R +T) resultieren darüber hinaus aus der besseren Speicherkohärenz beim Ray-Tracing, die ein adaptives Sampling zum Teil zerstört.

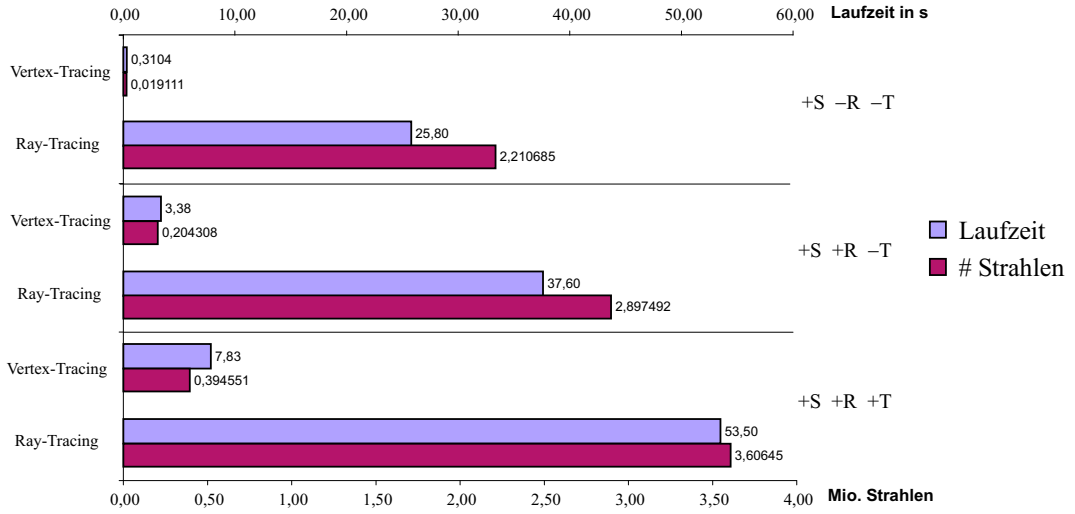


Abbildung 5.10: *Vergleich Ray-Tracing versus Vertex-Tracing in Bezug auf Laufzeit und Anzahl geschossener Strahlen.*

Die Menge berechneter Samples (Pixel) in Abhängigkeit von der gewählten Auflösung demonstrieren die Graphen aus Abbildung 5.11 im Modus +S +R −T. 5.11a zeigt im Einzelnen, dass der Prozentsatz der berechneten Pixel bei zunehmender Auflösung sinkt. Zurückzuführen ist diese Tatsache auf das adaptive Verhalten des Vertex-Tracings. Auch bei einer Verdopplung der Auflösung sowohl in X- als auch in Y-Richtung (vierfache Pixel-Anzahl) wird sich beispielsweise die Länge einer Diskontinuitätskante nur verdoppeln, so dass auch nur in etwa doppelt so viele Samples bestimmt werden müssen. Hinzu kommen Samples, die aus der Suche der Kante resultieren und deren Anzahl sich sub-linear zur Auflösung verhält. Lediglich bei der im Durchschnitt kleinen Anzahl an Nichtlinearitäten nimmt die Zahl der Samples proportional zur Fläche zu. Der Graphenverlauf in Abbildung 5.11b verhält sich entsprechend umgekehrt zu 5.11b. Die Menge der Samples wächst langsamer als die Pixel-Anzahl der Bildebene.

Die Auswirkung der Auflösung auf das Laufzeitverhalten des Vertex-Tracings zeigen die Graphen in Abbildung 5.12. Im Falle von 5.12a wurde die Laufzeit bis zum Erreichen vollständiger Konvergenz gemessen. Die Charakteristik der Funktion ist dabei analog zu der in 5.11b. Eine Verdopplung der Auflösung (vierfache Pixel-Anzahl) bedeutet nur in etwa eine Verdopplung der Laufzeit. Im Gegensatz zum Standard-Ray-Tracing reagiert damit das Vertex-Tracing weniger empfindlich auf die Vergrößerung der Bildebene und kann insofern einen weiteren Vorteil für sich verbuchen. Seine Laufzeit skaliert sub-linear zur Bildfläche. Noch deutlicher wird diese Eigenschaft bei der Darstellung der Preview-Laufzeit in 5.12b. Das Preview-Bild (↗ Abbildung 5.4) kann in nahezu konstanter Zeit berechnet werden. Die Erstellung wird geringfügig von der Lesezeit des ID- sowie Z-Buffers aus der Graphik-Hardware beeinflusst. Sie ist weitestgehend unabhängig von der Größe der Auflösung.

Das Auslesen des ID- beziehungsweise Z-Buffers dient als Grundlage zur Ausführung

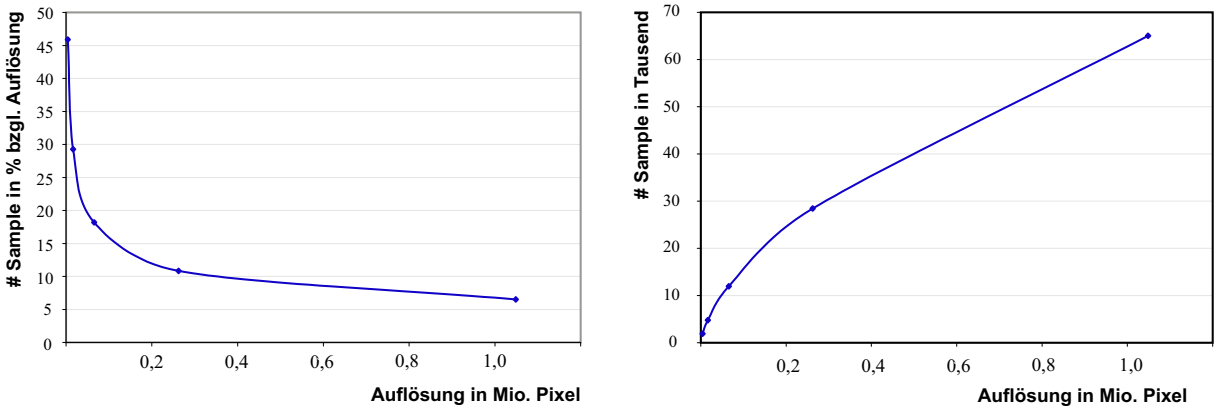


Abbildung 5.11: *Menge berechneter Samples in Abhängigkeit der Auflösung. a) Prozentual. b) Sample-Anzahl.*

der Visibility-Tests nach Abschnitt 3.4.1 und 3.4.2. Welche Auswirkung die verschiedenen Visibility-Tests auf die Laufzeit des Vertex-Tracings üben, demonstriert Abbildung 5.13. Deutlich erkennbar ist der enorme Laufzeitgewinn bei eingeschaltetem ID-Buffer-Test. Er verhindert unnötiges Sampling von VT-Objekten, die außerhalb des Erfassungsbereiches der Kamera liegen beziehungsweise von anderen Objekten verdeckt werden. Der Einsatz des Z-Buffers sowie des Viewport-Clippings (↗ Abschnitt 3.4.3) erzielen weitere Laufzeiteinsparungen. Der Z-Buffer-Test wirkt vor allem bei stark konkaven VT-Objekten, die selbst einen Großteil ihrer eigenen Faces verdecken. Bei Nahaufnahmen kommt der Clipping-Test zum Tragen. Er verhindert das Sampling von Face-Bereichen, die vom Viewport abgeschnitten werden, jedoch den ID-Buffer-Test passierten.

Trotz des deutlichen Laufzeitgewinns durch den Einsatz des ID-Buffer-Tests, stellt gerade dieser bei höherer Szenenkomplexität den „Flaschenhals“ des Vertex-Tracings dar. Die Bestimmung des ID-Buffers erfolgt in einem Pre-Compute Rendering-Pass mittels OpenGL-Hardware-Rendering der gesamten Szene (↗ Abschnitt 3.5). Ohne die Anwendung etwaiger Occlusion-Culling-Mechanismen bedeutet dies jedoch lineare Zeitkomplexität mit der Anzahl an Polygonen. Der Vorteil des Ray-Tracings, dessen Laufzeit sublinear mit der Szenenkomplexität skaliert (↗ Abbildung 1.2), wird damit relativiert. Das Vertex-Tracing skaliert linear mit der Szenenkomplexität. Es stellt insofern einen Kompromiss zwischen purem Ray-Tracing und purem Hardware-Rendering dar und ist prädestiniert für kleinere und mittlere Szenenkomplexitäten.

5.3 Parallelisierung

Die Untersuchung des parallelen Vertex-Tracings gliedert sich analog zu Kapitel 4 in das verteilte Konzept mittels Message-Passing sowie in das Shared-Memory Konzept mittels Multi-Threading. Zum Test stand wiederum die Tischszene aus Abbildung 5.1 in einer

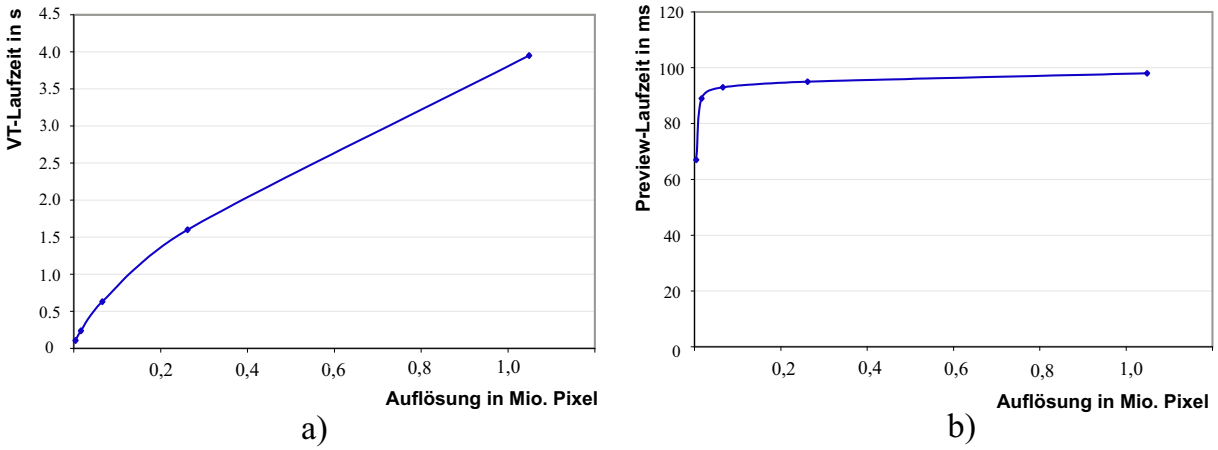


Abbildung 5.12: *Laufzeitverhalten in Abhängigkeit der Auflösung. a) Bei vollständiger Konvergenz. b) Preview.*

Auflösung von 1024x1024 jedoch hier im Modus +S +R +T zur Verfügung. Die Kameraperspektive entspricht dabei Abbildung 5.1b, so dass sich auch bei diesem Test ein hoher Anteil an Nichtlinearitäten beziehungsweise Diskontinuitäten im Bildausschnitt befindet.

5.3.1 Message-Passing

Die verteilte Berechnung des Vertex-Tracing erfolgt in einem heterogenen Rechnernetzwerk auf der Basis von Message-Passing. Grundlage dafür bildet das API PVM (↗ Abschnitt 4.3), das eine Einbeziehung unterschiedlichster Rechnerplattformen ermöglicht. Das für diesen Test benutzte Rechnernetzwerk entspricht einem klassischen Unternehmensnetzwerk mit einer Datenübertragungsrate von 100 MBit. Die einzelnen Rechnerknoten sind dabei alle Intel-basierte PCs mit Pentium4 zwischen 1.6 und 2.0 GHz. Auf ihnen wurde sowohl das OS Linux als auch Windows2000 installiert. Der Master-Knoten ist ebenfalls ein handelsüblicher PC jedoch mit Dual AMD-Athlon 1.4 GHz im Single-Mode, GeForce4 Ti 4600 und Betriebssystem Linux.

Im Vordergrund des Testes steht der erreichbare Speed-Up des verteilten Systems. Unterschieden wird dabei zwischen dem Pre-Compute Load-Balancings und dem Queue Load-Balancing folglich Abschnitt 4.3.3. Die Tischszene wurde dafür jeweils mit einer unterschiedlichen Anzahl von Knoten berechnet, um die Skalierbarkeit des verteilten Systems nachvollziehen zu können.

Die Graphen aus Abbildung 5.14 zeigen die Resultate der Messung. Das verteilte System skaliert entsprechend 5.14a recht gut bis etwa 8 Knoten, wobei das Queue Load-Balancing in dieser Größenordnung noch einen besseren Speed-Up als das Pre-Compute aufweisen kann. Zurückzuführen ist diese Tatsache vor allem auf die bessere *Load-Disbalance*[1] des Queue Load-Balancing. Wie in Abbildung 5.14b erkennbar, bewegt

[1]Die Load-Disbalance ermittelt sich aus der Differenz der jeweils geschossen Strahlen zwischen den

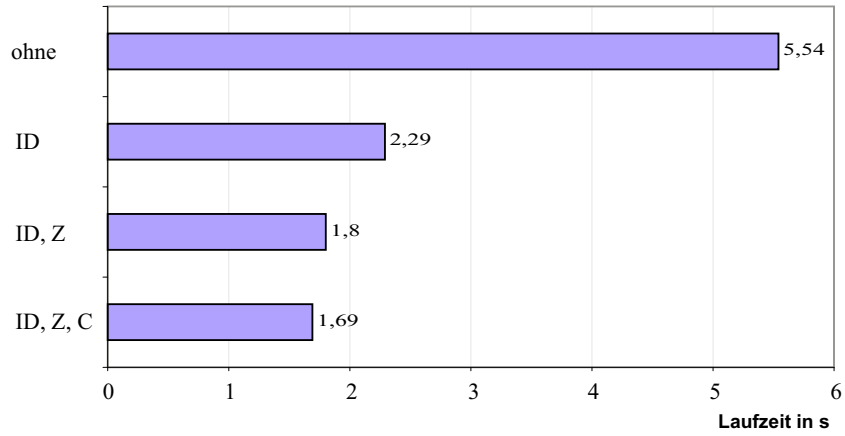


Abbildung 5.13: *Einfluss verschiedener Stufen des Visibility-Tests auf das Laufzeitverhalten des Vertex-Tracings. ID entspricht dem Visibility-Test mittels ID-Buffer, Z mittels Z-Buffer und C mittels Viewport-Clipping.*

sich dieser Wert deutlich unterhalb der Disbalance vom Pre-Compute Load-Balancing und ist zudem in seiner Gesamtheit konstanter. Im Falle der Disbalance beim Pre-Compute treten hingegen häufiger Schwankungen in der Load-Verteilung auf, was auf ein schlechteres Reaktionsvermögen dieses Konzeptes hinsichtlich sporadischer Systemschwankungen (Leistung, Kommunikation) hinweist. Das Queue-Konzept kann hier besser reagieren. Durch die Verteilung mehrerer kleinerer Pakete über den gesamten Zyklus eines Frames können derartige Schwankungen besser ausgeglichen werden.

Erstaunlich ist jedoch, dass das Pre-Compute Load-Balancing bei einer geringeren Anzahl von Knoten bessere Speed-Up-Werte erreicht, obwohl die Disbalance zum Teil doppelt so hoch ist als beim Queue Load-Balancing. Die Gründe dafür liegen insbesondere im Kommunikationsbedarf. Während beim Queue über den gesamte Frame-Zyklus Daten gesendet und empfangen werden müssen, erfolgt beim Pre-Compute Load-Balancing nur am Anfang und am Ende eines Zyklus der Datenaustausch. Provoziert das Datenvolumen bei einer kleineren Anzahl von Knoten noch keinen Engpass, ist der Datenaustausch beim Pre-Compute Load-Balancing effizienter, da die größeren Pakete hier weniger Overhead verursachen. Es werden höhere Speed-Up-Werte erzielt. Erst bei einer größeren Anzahl von Knoten kommt es zum Datenstau, da alle Slaves nahezu gleichzeitig ihre Daten zum Master zurück senden. An dieser Stelle erweist sich wiederum die Verteilung des Datenaustausches über den gesamten Frame-Zyklus als Vorteilhaft. Das Queue Load-Balancing kann an dieser Stelle besser skalieren.

Abbildung 5.14c zeigt die resultierende Laufzeit des verteilten Systems. Deutlich wird hier, dass die Frame-Rate nicht unter 1 Sekunde sinkt. Zurückzuführen ist dies auf die herrschende Latenzzeit des Systems in Verbindung mit der hohen zu übertragenden Da-

Knoten. Da das Sampling den wesentlichen Berechnungsaufwand darstellt, kann insofern eine gute Abschätzung über die Güte der Verteilung des Loads getroffen werden.

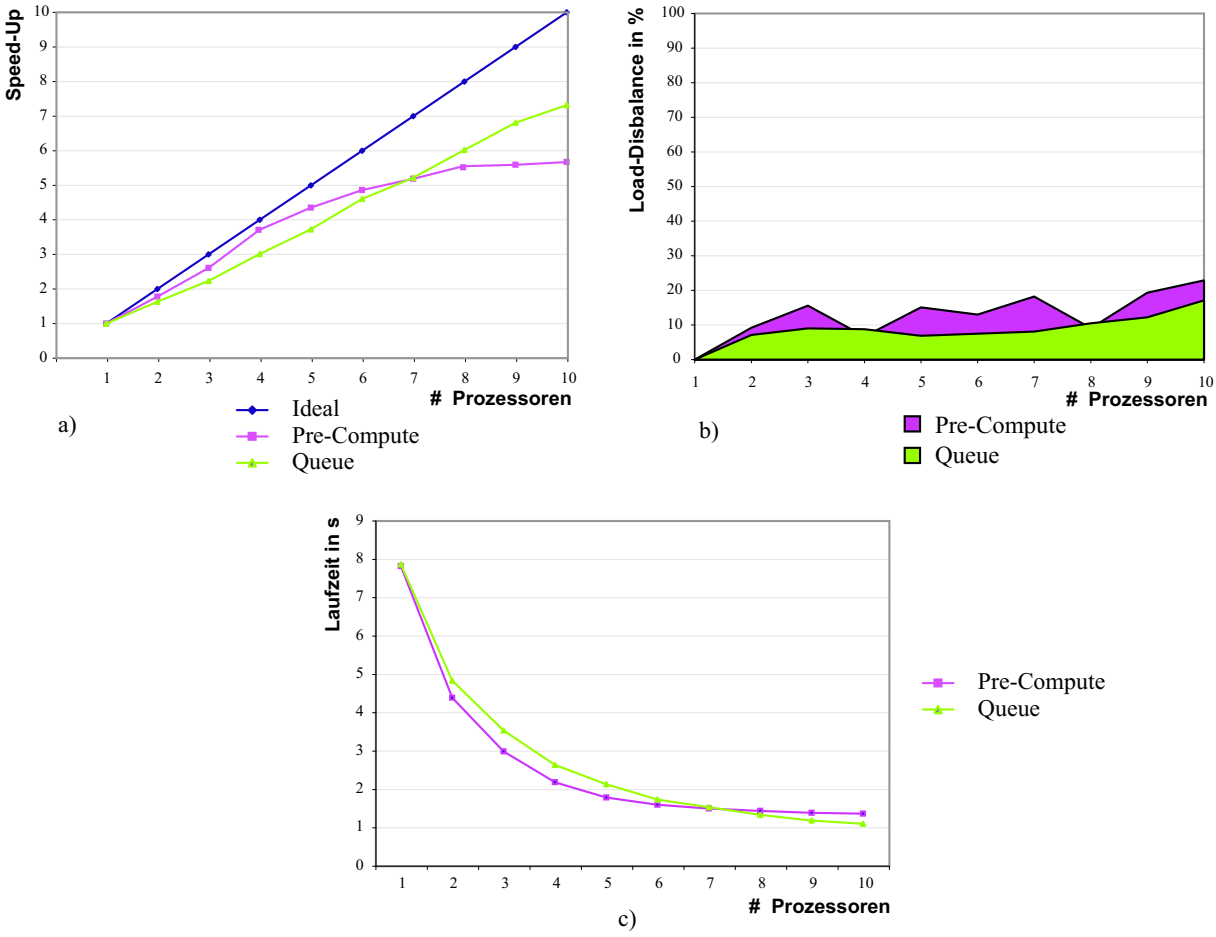


Abbildung 5.14: *Messungen verteiltes Vertex-Tracing. a) Speed-Up. b) Load-Disbalance. c) Gesamtlaufzeit.*

tenmenge. Je höher das Datenaufkommen, desto länger ist die Antwortzeit von PVM beziehungsweise CPPVM, die sich für das Packen und Entpacken der Daten verantwortlich zeichnen. Lösungen hierzu werden unter anderem in Abschnitt 6.2.3 angerissen.

5.3.2 Multi-Threading

Eine vollständige Untersuchung der Skalierbarkeit des Multi-Threading Konzeptes (↗ Abschnitt 4.4) konnte leider nicht vorgenommen werden, da zum Zeitpunkt der Messung kein Shared-Memory-System mit mehr als zwei Prozessoren zur Verfügung stand. Testplattform war demzufolge ein Dual-Athlon PC mit jeweils 1.4 GHz, 2 GByte Hauptspeicher und Graphikkarte GeForce4 Ti 4600.

Die erzielten Messergebnisse der Tischszene zeigt Abbildung 5.15. Aus den Laufzeitverhältnissen von Abbildung 5.15a lässt sich Rückschluss auf den erreichten Speed-Up ziehen. Dieser liegt in etwa bei 1.6 und trifft damit nicht die Erwartungen an das Multi-

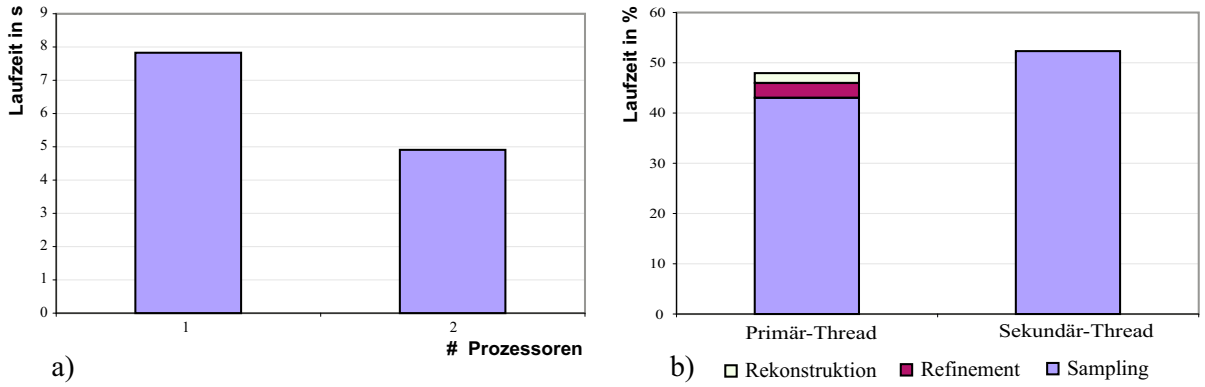


Abbildung 5.15: *Multi-Threading auf einem Dual-PC. a) Laufzeitvergleich zwischen ein und zwei Prozessoren. b) Load-Verteilung zwischen Primär- und Sekundär-Thread im Dual-Betrieb.*

Threading-Konzept. Die Ursache dafür liegt jedoch nicht in der Load-Verteilung, wie es Abbildung 5.15b mit Hilfe eines Profiler-Laufs deutlich demonstriert. Sowohl der Primär- als auch der Sekundär-Thread besitzen eine nahezu identische prozentuale Auslastung. Die Ursache für den mäßigen Speed-Up liegt vielmehr in den Implementationsdetails der Kollisionserkennung. Diese arbeitet stark inkohärent bezüglich des Speichermanagements. Das heißt, es finden oft beliebige Zugriffe im Hauptspeicher statt, so dass nahezu keine Cache-Kohärenz erzielt werden kann. Aufgrund des gemeinsamen Speicherinterfaces der Prozessoren entstehen konkurrierende Speicherzugriffe, die das Speicherinterface letztendlich zum Bottleneck der parallelen Abarbeitung werden lassen. Lösungen dafür liegen in der Anwendung einer möglichst Cache-kohärenten Kollisionserkennung, wie unter anderem in [PKGH97] oder [WBWS01] vorgestellt.

5.4 Anwendungsszenario

Die folgenden Abbildungen 5.16 - 5.21 demonstrieren ein Anwendungsszenario, indem gerade das Verfahren Vertex-Tracing seine Stärken ausspielen kann. Hier gilt es, die Ablesbarkeit des Kombiinstrumentes im KFZ-Interieur in einer VR-Umgebung zu evaluieren. Lediglich die Abdeckscheibe des Instrumentes wird dem Vertex-Tracing unterzogen. Die Darstellung aller restlichen Objekte erfolgt mit traditioneller OpenGL-Hardware.

Die Laufzeit bis zum vollständigen Refinement (finales Bild) der Abdeckscheibe beträgt aus der Perspektive des Fahrers circa 2.5 Sekunden. Die Gesamtszene besitzt eine Komplexität von etwa 340'000 Polygone und wurde in einer Auflösung von 1280x1024 dargestellt. Als Rechner diente auch hier ein Dual-Athlon 1.4 GHz System mit GeForce4 Ti 4600. Die Darstellung des Previews benötigt nur circa 0.2 Sekunden und ist damit zur Echtzeitvisualisierung in VR einsetzbar.

Abbildung 5.16: *Evaluierung der Ablesbarkeit eines KFZ-Kombiinstrumentes.*

Abbildung 5.17: *Preview-Darstellung nach 0.2 Sekunden.*

Abbildung 5.18: *Lateral versetzter Blick auf das Kombiinstrument.*

Abbildung 5.19: *Ohne Reflexion der Abdeckscheibe.*

Abbildung 5.20: *Ohne Transmission der Abdeckscheibe.*

Abbildung 5.21: *Zoom auf das Kombiinstrument*

Kapitel 6

Systemanalyse und Erweiterbarkeit

Während in den Kapiteln 3 und 4 das Verfahren Vertex-Tracing im Detail hinsichtlich seines Algorithmus und im Kapitel 5 hinsichtlich der praktischen Resultate beschrieben wurde, zielt dieses Kapitel nun auf mögliche Erweiterungen beziehungsweise Verbesserungen des Systems. Im Vordergrund stehen hierbei die Themen Korrektheit und Laufzeitverhalten. Die Diskussion bezüglich der Korrektheit bezieht sich in diesem Zusammenhang insbesondere auf den Vergleich zur Darstellungsqualität des traditionellen Ray-Tracings. Bedingt durch die unterschiedlichen Sampling-Methoden vom Vertex-Tracing und Ray-Tracing müssen hier Aussagen über die Rekonstruktionsfehler der Radiance-Funktion getroffen werden.

In Bezug auf das Laufzeitverhalten steht vor allem die Beschleunigung des Strahl-Schnittpunkttestes im Vordergrund. Aspekte wie Kohärenz, Datenstrukturen und Hardware, die zu einer Beschleunigung des Kollisionstests beitragen, sollen hier näher beleuchtet werden. Im unmittelbaren Zusammenhang steht dabei auch das Thema Parallelisierung. Hier gilt es, Potentiale und Grenzen hinsichtlich einer verbesserten kooperativen Abarbeitung des adaptiv progressiven Vertex-Tracings aufzuzeigen.

6.1 Korrektheit

Adaptives Sampling der Radiance-Funktion und ihre anschließende Rekonstruktion durch lineare Interpolation bilden den zentralen Algorithmus des Vertex-Tracings (↗ Abschnitt 3.2). Im Gegensatz zum traditionellen Ray-Tracing, bei dem ein uniformes Sampling der Bildebene als *straight-forward* Ansatz durchgeführt wird, zielt das adaptiv non-uniforme Sampling auf eine dichtere Abtastung hochfrequenter Definitionsbereiche, während in Bereichen geringerer Frequenz nur vereinzelte Stichproben genommen werden.

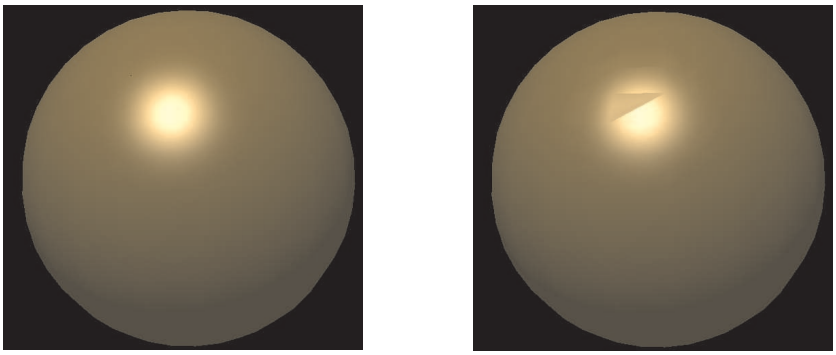
Der Fehler, der durch die Diskretisierung der Radiance-Funktion entsteht, ist dabei von der minimalen diskreten Intervallgröße abhängig und insofern beim Ray-Tracing als auch beim Vertex-Tracing durch die Auflösung des Pixel-Rasters vorgegeben. Der Vorteil des adaptiven Samplings beim Vertex-Tracing besteht nun darin, dass der Fehler zwischen zwei Sample-Werten über den gesamten Definitionsbereich in etwa gleich bleibt. Das Sampling passt sich dem Funktionsverlauf an und erfolgt demzufolge effizienter.

Nachteilig bei einem derart adaptiven Ansatz ist der zwangsläufig progressive Charakter. Das bedeutet, ein adaptives Sampling muss stets mit einem groben initialen Sample-Pattern beginnen, um sich dann progressiv (schrittweise) der Funktion anzupassen. Durch die im Sample-Pattern existierenden zum Teil großen Sample-Intervalle können nun jedoch lokale Extrema unentdeckt bleiben, wie in Abbildung 3.8 bereits dargestellt wurde. Die Gefahr einer Nichtdetektion solcher lokalen Extrema, wie sie durch Nichtlinearitäten und Diskontinuitäten in der Radiance-Funktion auftreten können, stellen die wesentliche Diskrepanz zwischen Ray-Tracing und Vertex-Tracing in Bezug auf Korrektheit dar.

Wie solche Fehldetektionen vermieden oder sogar ausgeschlossen werden können, ist ein wesentlicher Bestandteil weiterer Diskussionen in diesem Abschnitt. Darüber hinaus sollen aber auch Fragen bezüglich von Fehlern erörtert werden, die bei der Texturierung sowie bei der Parallelisierung des Verfahrens auftreten und charakteristisch für den gewählten Ansatz des Vertex-Tracings sind.

6.1.1 Nichtlinearitäten

Nichtlinearitäten wie beispielsweise in spekularen High-Lights anzutreffen (↗ Abbildung 6.1a), sind typische Merkmale der Radiance-Funktion und treten überall dort auf, wo diffuse oder spekulare Reflexionen existieren. Der nichtlineare Charakter der Radiance-Funktion ergibt sich dabei aus dem verwendeten lokalen Shading-Modell. Wie im Ab-



a) b)

Abbildung 6.1: *Beispiel einer fehlerhaft detektierten Nichtlinearität, die durch ein spekulares High-Light entsteht.*

schnitt 3.19 dargestellt, wurde im Falle des Vertex-Tracings das Phong-Modell [Pho75] verwendet. Die Nichtlinearität resultiert hier jeweils aus dem Cosinus des Winkels θ für die diffuse Reflexion und des Winkels ϕ für die spekulare Reflexion mit

$$cos(\theta) \;=\; \vec{l} \cdot \vec{n}, \quad cos(\phi) \;=\; \vec{r} \cdot \vec{a}, \tag{6.1}$$

[1] Vorausgesetzt die Vektoren $\vec{l}, \vec{n}, \vec{r}$ und $\vec{a}$ sind normalisiert.

$\vec{l}$ dem Vektor zur Lichtquelle, $\vec{n}$ der Flächennormalen, $\vec{r}$ dem Richtungsvektor der direkten Reflexion und $\vec{a}$ der Blickrichtung (↗ Abbildung 6.2).

Ein Beispiel für den nichtlinearen Charakter der Radiance-Funktion und der damit verbundenen Gefahr einer Nichtdetektion lokaler Extrema zeigt Abbildung 3.8. Das lokale Maximum der Funktion fällt hier zwischen die Punkte p_1 und p_2. Die Differenz δ der Funktionswerte s_1 und s_2 tritt nicht über einen vorgegebenen Schwellwert. Obwohl der tatsächliche Approximationsfehler ε_A signifikant größer als δ ist, kommt es an dieser Stelle nicht zur Unterteilung des Sample-Intervalls. Die Folge sind starke Interpolationsfehler, die sich als Artefakte wie in Abbildung 6.1b darstellen.

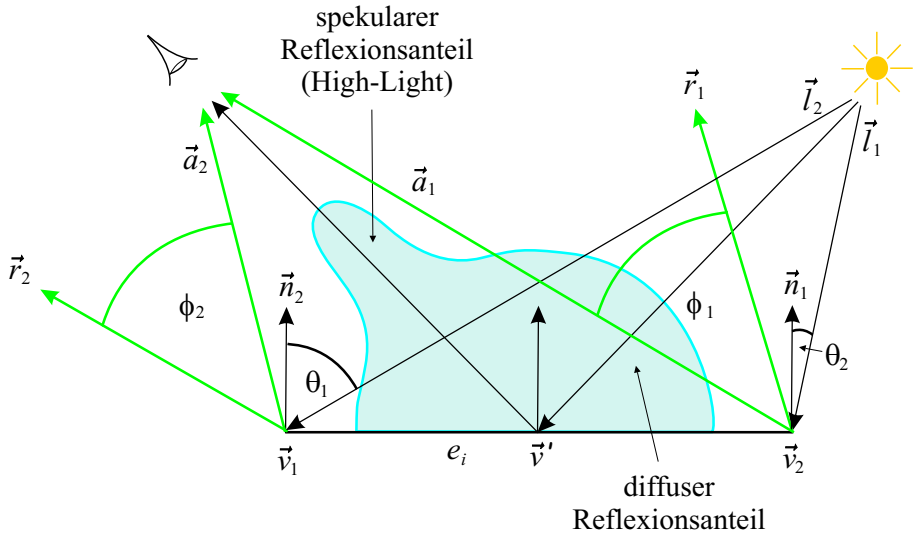


Abbildung 6.2: *Auftreten einer Nichtlinearität innerhalb einer Edge e_i zwischen ihren Vertices $\vec{v}_1$ und $\vec{v}_2$.*

Eine Möglichkeit, eine derart fehlerhafte Detektion zu vermeiden, besteht in der genaueren Untersuchung der Radiance-Funktion an den Stellen p_1 und p_2. Betrachtet man in Abbildung 3.8 die Funktion in s_1 und s_2, so fällt auf, dass sich der Anstieg in diesen Punkten zueinander konträr verhält. Die Bestimmung des Anstiegs ergibt sich aus den Gleichungen des diffusen Anteils I_d sowie spekularen Anteils I_s mit

$$\begin{aligned} I_d &= I_{source} \cdot k_d \cdot cos(\theta), \\ I_s &= I_{source} \cdot k_s \cdot cos^m(\phi) \end{aligned} \tag{6.2}$$

durch ihre jeweilige Differentiation mit

$$\begin{aligned} \frac{\partial I_d}{\partial \theta} &= I_{source} \cdot k_d \frac{d\, cos(\theta)}{d\, \theta} = -I_{source} \cdot k_d \cdot sin(\theta) \text{ und} \\ \frac{\partial I_s}{\partial \phi} &= I_{source} \cdot k_s \frac{d\, cos^m(\phi)}{d\, \phi} = -I_{source} \cdot k_s \frac{cos^m(\phi) \cdot m\, sin(\phi)}{cos(\phi)}. \end{aligned} \tag{6.3}$$

Ein lokales Extremum liegt demzufolge zwischen p_1 und p_2, wenn $\frac{\partial I_d}{\partial \theta_1}$ und $\frac{\partial I_d}{\partial \theta_2}$ für den diffusen Teil beziehungsweise $\frac{\partial I_s}{\partial \phi_1}$ und $\frac{\partial I_s}{\partial \phi_2}$ für den spekularen Teil unterschiedliche Vorzeichen besitzen. Die Unterteilung des Sample-Intervalls beziehungsweise die Unterteilung e_i muss an dieser Stelle vorgenommen werden, um eine bessere Approximation der Radiance-Funktion zu garantieren.

Alternativ zu der oben beschriebenen Methode durch Differentiation der Shading-Funktion kann auch eine rein geometrische Bestimmung mit Laufzeitvorteilen durchgeführt werden. Betrachtet man sich hierzu beispielhaft die Winkel ϕ_1 und ϕ_2 der spekularen Reflexion, so ist zu erkennen, dass die Winkel jeweils auf der gegenüberliegenden Seite von $\vec{a}_{1,2}$ liegen (↗ Abbildung 6.2). Um diese Eigenschaft auszuwerten, muss zunächst jeweils $\vec{a}$ um 90 Grad in der Ebene $\vec{a}\vec{r}$-Ebene gedreht werden. Der dabei resultierende Vektor $\vec{g}$ ergibt sich dabei aus der Linearkombination von $\vec{a}$ und $\vec{r}$ mit $\vec{g} = s\,\vec{a} + t\,\vec{r}$, wie in

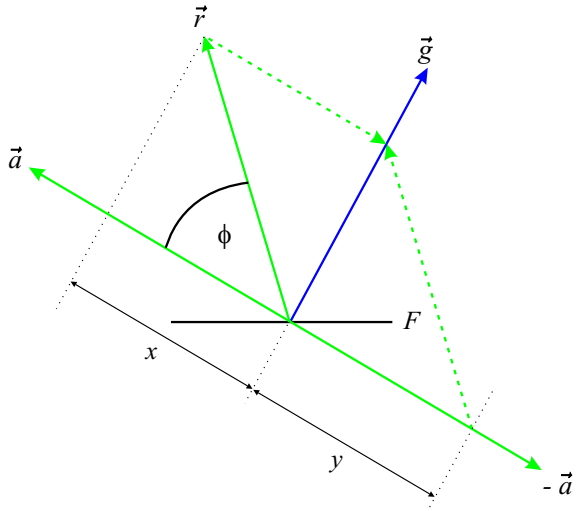


Abbildung 6.3: *Berechnung von $\vec{g}$ aus der Linearkombination von $\vec{a}$ und $\vec{r}$.*

Abbildung 6.3 dargestellt. Unter der Annahme, dass $\vec{a}$ und $\vec{r}$ normalisiert sind und $x = y$ ist, gilt:

$$\begin{aligned} \vec{g} &= s\,\vec{a} + t\,\vec{r} \\ \vec{g} &= -y\,\vec{a} + \vec{r} \\ \vec{g} &= -(\vec{a} \cdot \vec{r})\vec{a} + \vec{r} \\ \vec{g} &= -cos(\phi)\vec{a} + \vec{r}. \end{aligned} \tag{6.4}$$

Die Bestimmung von $\vec{g}$ beziehungsweise $\vec{g_1}$ und $\vec{g_2}$ auf den Vertices $\vec{v}_1$ und $\vec{v}_2$ lässt nun einen Rückschluss auf ein lokales Extremum beziehungsweise auf eine notwendige Unterteilung des Intervalls zu. Mit $cos(\gamma) = \vec{g}_1 \cdot \vec{g}_2$, dem Cosinus des Winkels zwischen $\vec{g}_1$ und $\vec{g}_2$, gilt schließlich:

$$cos(\gamma) = \begin{cases} \leq 0 & : \text{ unterteile } e_i \\ > 0 & : \text{ keine Unterteilung.} \end{cases} \tag{6.5}$$

Die Untersuchung muss dabei für jede Lichtquelle durchgeführt werden.

Eine weitere Möglichkeit, die nichtlineare Radiance-Funktion zuverlässig mit einem vorgegebenen Fehler linear zu approximieren und somit die Detektion lokaler Extrema sicherzustellen, kann mittels *Intervallarithmetik* [Han75, CS93, Bal99] beschrieben werden. Im Falle der einfachsten Form, der Standard-Intervallarithmetik, wird jede mathematische Größe einer Funktion durch ein vorgegebenes Intervall substituiert. Die Berechnung der dabei entstandenen Intervallfunktion erfolgt anhand elementarer Intervalloperationen[2]. Der resultierende Funktionsbereich der Intervallfunktion garantiert, dass der Funktionsbereichs der unbekannten Ausgangsfunktion darin enthalten ist.

Die Anwendung der Standard-Intervallarithmetik zur Abschätzung des tatsächlichen Interpolationsfehlers ε_A demonstriert Abbildung 6.4. Der Fehler ε_A, der zwischen der Radiance-Funktion $f(x)$ und der interpolierten Funktion $f'(x)$ mit $\varepsilon_A = max\,|f(x) - f'(x)|$ im Intervall $IA \in [p_1, p_2]$ resultiert, wird durch ε_{IA} eingeschlossen. ε_{IA} wiederum geht aus der Berechnung von $f(x)$ mittels Intervallarithmetik hervor und

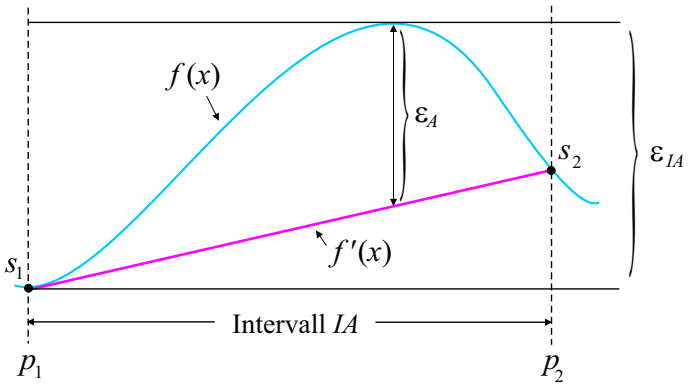


Abbildung 6.4: *Abschätzung des Interpolationsfehlers ε_A mittels Standard-Intervallarithmetik.*

schließt den Funktionsbereich von $f(x)$ in IA ein. Wird nun ε_{IA} als Entscheidungskriterium zur Unterteilung des Intervalls herangezogen, kann damit eine definierte Aussage über den maximalen Interpolationsfehler ε_A mit $\varepsilon_A \leq \varepsilon_{IA}$ getroffen werden.

Die Fehlerabschätzung mittels Standard-Intervallarithmetik ist jedoch stark konservativ. Das heißt, die resultierende Intervallgröße von ε_{IA} kann eine signifikante Diskrepanz zum tatsächlichen Funktionsverlauf aufweisen. Bessere Ergebnisse erzielen in diesem Zusammenhang erweiterte Formen der Intervallarithmetik wie lineare Intervallarithmetik [Bal99] oder affine Intervallarithmetik [CS93].

6.1.2 Diskontinuitäten

Unstetigkeiten in der Radiance-Funktion werden in diesem Zusammenhang als Diskontinuitäten bezeichnet. Der Grund für ihr Auftreten liegt in der Komposition der Szene

[2]Die Addition zweier Intervalle ist beispielsweise definiert durch $[a, b] + [c, d] = [a + c, b + d]$.

aus einer Vielzahl von Objekten, die meist in ihren Grenzen eindeutig beschrieben sind und damit bereits eine Diskontinuität im Objektraum verursachen. Die Entstehung von Diskontinuitäten kann durch folgende Fälle klassifiziert werden:

1. Diskontinuitäten aufgrund von Schatten. Beschattet ein Objekt A (*Occluder*) teilweise ein anderes Objekt B, so entstehen Schattenkanten auf B, die sich als Diskontinuitäten darstellen. Darunter zählen auch Eigenbeschattungen, die ein Objekt vor allem auf seiner lichtabgewandten Seite selbst hervorruft.

2. Diskontinuitäten aufgrund von Reflexion oder Brechung. Ein sich reflektierendes oder brechendes Objekt A in B erzeugt einen unstetigen Verlauf der Radiance-Funktion auf der Objektoberfläche von B.

3. Diskontinuitäten aufgrund der Objektgeometrie. Die Interpolation eines sich reflektierenden/brechenden Objektes A in B kann zu Fehlern führen, wenn auf der Oberfläche von A ein diskontinuierlicher Wechsel der Normalenrichtung (zum Beispiel Würfel) auftritt.

4. Diskontinuitäten durch interne Totalreflexion. Wird ein Objekt A in B gebrochen, kann aufgrund der Geometrie von B partielle Totalreflexion in B auftreten, was in einem diskontinuierlichen Verlauf der Radiance-Funktion resultiert.

5. Diskontinuitäten durch Spots. Lichtquellen in Form von Spotlights erzeugen unstetige Radiance-Verläufe an der Grenze ihres Lichtkegels.

Ein Maß für die Korrektheit des Vertex-Tracings ist die zuverlässige Detektion der oben beschriebenen Fälle von Diskontinuitäten in einem vorgegebenen Sample-Intervall. Methoden, die eine Vielzahl von Diskontinuitäten lokalisieren, wurden bereits in Abschnitt

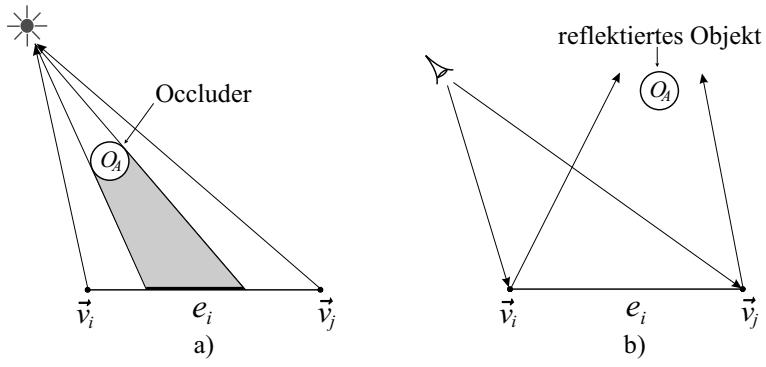


Abbildung 6.5: *Sonderfälle, bei denen die Detektion der Diskontinuität nach Abschnitt 3.3.1 fehl schlägt.*

3.3.1 näher erläutert. Zwei Sonderfälle, bei denen jedoch andere Maßnahmen zur sicheren Detektion ergriffen werden müssen, werden in Abbildung 6.5 skizziert. Sowohl im Fall der

Schattenfühler (6.5a) als auch bei reflektierenden/transmitterenden Strahlen (6.5b) wird Objekt O_A nicht erkannt. Trotz des auftretenden Schattens beziehungsweise der Reflexion entlang der Edge e_i existieren jeweils keine Schnittpunkte der von $\vec{v}_i$ und $\vec{v}_j$ entsandten Strahlen mit O_A. Es erfolgt keine weitere Unterteilung von e_i, so dass Objekt O_A für dieses Sample-Intervall unerkannt bleibt.

Eine sichere Detektion beliebiger Occluder und Objekte kann mit Hilfe des sogenannten *Shaft-Culling* [HW91, TBD96] durchgeführt werden. Basierend auf der Methode von [HW91] wird hierzu zunächst eine achsenparallele Bounding-Box (AABB) um jede Face f_i gelegt (↗ Abbildung 6.6). Im zweiten Schritt erfolgt die Bildung des Shaft-Volumens. Im Falle der Schattenfühler setzt sich das Volumen aus der Bounding-Box und der Lichtquelle zusammen (↗ Abbildung 6.6a). Für die Bildung des Shaft-Volumens bei der Reflexion/Transmission muss zusätzlich die Face f_i in Richtung der reflektierenden/transmittierenden Strahlen um die Länge d verschoben werden. Die resultierende

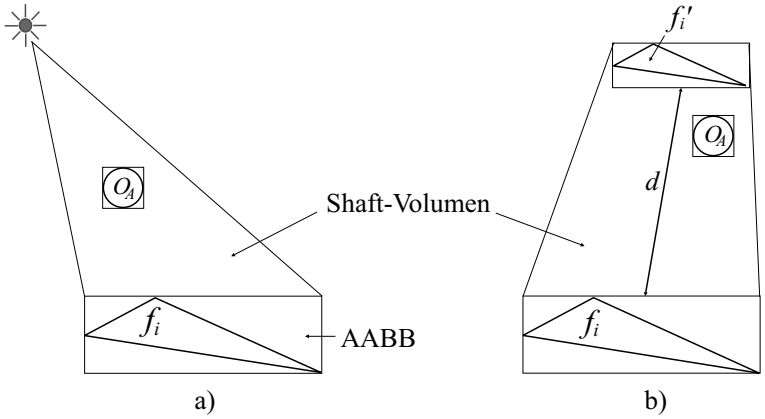


Abbildung 6.6: *Sichere Detektion von Schatten (a) und Reflexion/Transmission (b) durch Shaft-Culling, basierend auf der Methode aus [HW91].*

Face f_i' dient zur Generierung einer weitere Bounding-Box (↗ Abbildung 6.6b), die das aufgespannte Reflexions-/Transmissionsvolumen von f_i mit der Distanz d repräsentiert. d entspricht dabei der maximalen Ausdehnung der Szene im Objektraum, um die Erfassung aller Szenenobjekte sicherzustellen.

Schritt Drei beinhaltet schließlich den eigentlich Test. Jedes Shaft-Volumen einer Face f_i wird hierzu gegen alle Szenenobjekte[3] auf Schnitt getestet. Befindet sich mindestens *ein* Objekt partiell oder vollständig im Shaft-Volumen von f_i, muss jede zu f_i gehörige Edge e_i unterteilt werden. Existiert dagegen in den Shaft-Volumen der zwei adjazenten Faces von e_i kein Szenenobjekt, kann zumindest eine sichere Aussage darüber getroffen werden, dass es an dieser Stelle keine Diskontinuität aufgrund von Schatten, Reflexion oder Transmission gibt.

Andere Fälle von Diskontinuitäten, wie beispielsweise durch ein Spotlight hervorgerufen, können durch das Shaft-Culling nicht detektiert werden. Eine detaillierte

[3]Eine Repräsentation der Szenenobjekte durch AABBs beschleunigt dabei den Test erheblich.

Beschreibung von Methoden, die auch derartige Diskontinuitäten untersuchen, findet man unter anderem in [Bal99].

Eine allgemein gültige Methode, die sowohl zur besseren Detektion von Nichtlinearitäten als auch Diskontinuitäten führt, liegt im automatischen Unterteilen von zu langen Edges. Edges, deren projizierte Längen über einem bestimmten Maximum liegen, werden ohne Rücksicht auf die Sample-Werte ihrer Vertices unterteilt. In Analogie zu *Multi-Resolution-Verfahren* werden damit zumeist Polygone des Vordergrunds stärker verfeinert, so dass hier ein dichteres Sampling erfolgen kann.

6.1.3 Zwangsverfeinerung im verteilten Vertex-Tracing

Neben unentdeckten Nichtlinearitäten und Diskontinuitäten, hervorgerufen durch das adaptive Sampling der Radiance-Funktion (↗ Abschnitt 6.1.1, 6.1.2), kann darüber hinaus das verteilte Vertex-Tracing zusätzliche Fehldetektionen verursachen. Der Grund dafür liegt in der Ausführung von Zwangsverfeinerungen, wie in Abschnitt 3.3.2 beschrieben. Kommt es zu einer notwendigen Zwangsverfeinerung an der Bereichsgrenze zweier Slaves (↗ Abbildung 6.7), kann diese aufgrund der unterschiedlichen Prozessorzugehörigkeit der beteiligten Faces nicht ausgeführt werden. Für Slave 0 sowie Slave 1 stellt die zu unterteilende Edge eine Rand-Edge dar, die lediglich *eine* adjazente Face besitzt. Die dabei

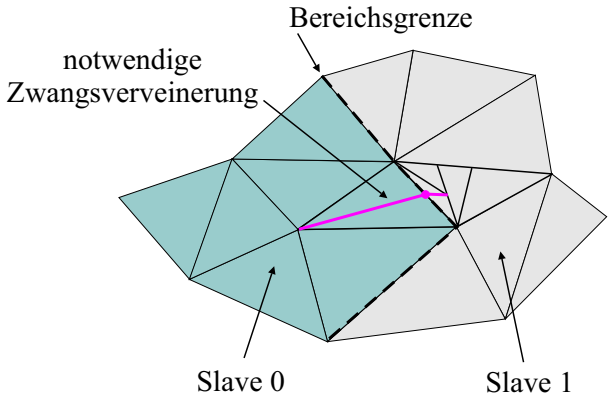


Abbildung 6.7: *Abhängigkeiten zwischen zwei Slave-Bereichen durch anfallende Zwangsverfeinerungen*

versäumte Zwangsverfeinerung kann nun erheblichen Einfluss auf die Rekonstruktionsqualität der Radiance-Funktion besitzen. Wie in Abschnitt 3.3.2 bereits erläutert, können Zwangsverfeinerungen ein Indiz für noch unentdeckte Nichtlinearitäten beziehungsweise Diskontinuitäten sein. Sie tragen insofern maßgeblich zu einer zuverlässigeren Detektion bei.

Da die Zwangsverfeinerung einen integralen Bestandteil des Vertex-Tracings darstellt, liegt es nahe, dass eine Lösung des skizzierten Problems nicht ohne signifikantem Mehraufwand erzielbar ist. Eine Möglichkeit besteht darin, eine direkte Kommunikation zwischen

den Slaves zuzulassen, um eine notwendige Zwangsverfeinerung über die eigene Bereichsgrenze hinaus durchzusetzen. Der resultierende Kommunikationsaufwand wäre jedoch beträchtlich. Vor allem bei einer hohen Anzahl an Slaves steigt die Anzahl vorhandener Bereichsgrenzen, an denen verstärkt Zwangsverfeinerungen auftreten können.

Alternativ dazu wäre auch eine iterative Behandlung von Zwangsverfeinerungen denkbar. Das heißt, notwendige Zwangsverfeinerungen an der Bereichsgrenze eines Slaves werden zunächst nur registriert und mit den generierten Faces zum Master zurückgesendet. Der Master wertet diese aus und veranlasst die betreffenden Slaves, die notwendigen Zwangsverfeinerungen vorzunehmen. Mit erneut auftretenden Zwangsverfeinerung wird analog verfahren. Der Vorgang bricht ab, sobald die maximale Refinement-Stufe erreicht ist beziehungsweise keine weiteren Zwangsverfeinerungen durchzuführen sind.

Eine implizite Lösung des Problems, würde eine hundertprozentige Detektion von Nichtlinearitäten und Diskontinuitäten mit sich führen. Finden Verfahren, wie in den Abschnitten 6.1.1, 6.1.2 beschrieben, ihren Einsatz, entfallen die Zwangsverfeinerungen, da diese ausschließlich aus Fehldetektionen resultieren. Jede Edge würde genau dann unterteilt werden, wenn ein Feature zwischen ihren Vertices tatsächlich existiert. Am Beispiel von Abbildung 6.7 würde die betrachtete Rand-Edge von jedem Slave selbstständig unterteilt werden.

6.2 Laufzeitverhalten

Eine weitere wesentliche Rolle im Hinblick auf die Erweiterbarkeit des Vertex-Tracings spielt neben der Korrektheit das Thema Laufzeitverhalten. Primärziel des Verfahrens ist es, die Anzahl an Samples durch Ausnutzung von Image-Kohärenz auf ein Minimum zu reduzieren. Im Gegensatz zum traditionellen Ray-Tracing konnte somit ein effizienteres Sampling der Radiance-Funktion erreicht werden, was sich deutlich positiv auf das Laufzeitverhalten auswirkt.

Dennoch, die Bestimmung der verbleibenden Samples zur Rekonstruktion der Radiance-Funktion stellt weiterhin das Gros an Aufwand beim Vertex-Tracing dar (↗ Abschnitt 5.2). Eine Optimierung bezüglich des Laufzeitverhaltens würde sich demnach vor allem durch ein verbessertes Sampling ergeben. Potentiale hierzu liegen zum einen in einer weiteren Reduzierung und zum anderen in einer effizienteren Bestimmung der benötigten Samples. Welche Ansätze im Detail für jeweils beide Wege denkbar sind, soll in den folgenden Abschnitten näher beleuchtet werden.

6.2.1 Sample-Reduktion

Die Gesamtsumme der zu bestimmenden Samples eines Frames hängt zum einen von der Größe des Initial-Sample-Pattern und zum anderen von der Anzahl der generierten Samples während des Refinement-Prozesses ab. Die Größe des Initial-Sample-Pattern wiederum bestimmt der Visibility-Test (↗ Abschnitt 3.4.1), der als Pre-Prozess vor dem eigentlichen Vertex-Tracing durchgeführt wird. Er entscheidet über die Anzahl der Initial-Samples und

nimmt insofern unmittelbar Einfluss auf das Laufzeitverhalten der Preview-Erstellung.

Ein Ansatz, der die Anzahl der Initial-Samples reduziert, liegt in der Zusammenfassung von Primär-Vertices vor dem eigentlichen Sampling-Prozess. Die Zusammenfassung geschieht dabei nicht auf Geometrieebene, sondern vielmehr auf Basis des Samplings. Zwei Primär-Vertices $\vec{v}_1, \vec{v}_2$ einer Primär-Edge e_i werden genau dann als *ein* Sample betrachtet, wenn die projizierte Länge l von e_i unter einen bestimmten Schwellwert ϵ sinkt. Ist dies der Fall, wird nur einer der Primär-Vertices dem Sampling (Vertex-Tracing) unterzogen. Der zweite Vertex erbt die gewonnenen Sample-Werte des anderen, so dass ein Sampling für ihn entfällt. Ein guter Schwellwert liegt bei $\epsilon \leq 1$ Pixel. In diesem Fall kann so oder so nur ein Primär-Vertex von e_i visualisiert werden.

Insbesondere bei Szenen mit hoher Tiefenkomplexität, kann man mit diesem Ansatz einen signifikanten Laufzeitgewinn bei der Preview-Erstellung erwarten. Analog zu dynamischen Multi-Resolution-Verfahren werden Objekte im Hintergrund mit geringerer Qualität behandelt.

Eine Reduzierung der Samples, die während des adaptiven Refinement-Prozesses entstehen, kann vor allem durch eine intelligentere Unterteilung der jeweils betrachteten Edge erreicht werden. Ersetzt man die gleichseitige Halbierung der Edge e_1 (↗ Abbildung 3.5) durch eine geeignetere Wahl des einzufügenden Vertex $\vec{v}'_1$, kann unter Umständen weiter die Anzahl an Samples reduziert werden. Je näher sich $\vec{v}'_1$ am enthaltenen Feature der Edge befindet, desto schneller konvergiert der adaptive Refinement-Prozess. Information, die für eine bessere Platzierung von $\vec{v}'_1$ notwendig sind, können unterdessen aus den Untersuchungen herangezogen werden, die in den Abschnitten 6.1.1 und 6.1.2 vorgeschlagen wurden.

6.2.2 Effiziente Kollisionsdetektion

Wurde die Anzahl an Samples auf ein Minimum reduziert, bestimmen maßgeblich die Kosten zur Berechnung der Sample-Werte das Laufzeitverhalten des Samplings. Gerade beim Vertex-Tracing, bei dem vor allem in den hochfrequenten Regionen der Radiance-Funktion abgetastet wird, liegen die durchschnittlichen Kosten zur Bestimmung eines Samples verhältnismäßig hoch. Im Vergleich zum Standard-Ray-Tracing werden beim Vertex-Tracing zwar signifikant weniger Samples bestimmt, jedoch dafür vorwiegend die „Teuersten“ der Bildebene.

Da die Berechnung eines Samples zu 95 % durch die eigentliche Kollisionsdetektion des Strahls mit der Szene bestimmt wird [Rau93], stellt die Optimierung dessen einen wesentlichen Faktor in Bezug auf das Laufzeitverhalten des Gesamtsystems dar. Ansätze, die potentiell zur Beschleunigung der Kollisionsdetektion beim Vertex-Tracing beitragen, sollen im Folgenden näher beschrieben werden:

1. In erster Linie ist auch beim Vertex-Tracing der Einsatz traditioneller Beschleunigungsverfahren notwendig, um einen signifikanten Speed-Up im Strahlen-Schnittpunkttest zu erhalten. Verfahren wie zum Beispiel hierarchische Datenstrukturen [MH99, HB00, RSH00] oder Strahlenklassifizierungen [Ama84] zählen ebenso dazu wie Optimierungen bezüglich der Systemarchitektur. Letzteres bezieht sich vor

allem auf die Wahrung von Cache-Kohärenzen oder auf die Ausnutzung von spezialisierten Prozessor-Einheiten (SIMD) [WBWS01]. Die Wahl der geeigneten Datenstruktur ist wie beim Standard-Ray-Tracing stark von der Anwendung abhängig. Der Einsatz des Vertex-Tracings in dynamischen Szenen verlangt entsprechend flexible Hierarchien, wie es beispielsweise OOB-Hierarchien[4] erfüllen. Der Einsatz von statischen Datenstrukturen wie *BSP-Trees* oder *kd-Trees* [MH99] ist dagegen eher für reine Walk-Throughs durch ebenso statische Szenen prädestiniert, kann aber mit einem effizienteren Kollisionstest aufwarten.

2. Ein anderer Ansatz zur Beschleunigung der Kollisionsdetektion, der sich jedoch stärker auf die Charakteristik des Vertex-Tracings bezieht, liegt in einer Vorsortierung der zu testenden Strahlen. Kollisionssysteme, die insbesondere ihre Effizienz aus der vorliegenden Strahlen- beziehungsweise Szenenkohärenz beziehen, benötigen zueinander ähnliche Strahlen in der Reihenfolge ihrer Abarbeitung. Das adaptive Sampling des Vertex-Tracings steht jedoch konträr zu dieser Bedingung. Hier werden aufgrund des schrittweisen Refinements (↗ Abschnitt 3.3) zum Teil stark inkohärente Strahlen generiert. Eine Vorsortierung der zu testenden Strahlen nach Zugehörigkeit zum jeweiligen VT-Objekt sowie ihrer Rekursionstiefe würde in diesem Fall eine bessere Strahlenkohärenz hervorrufen, so dass Kollisionssysteme davon profitieren können.

3. Die Methode Shaft-Culling stellt eine weitere Variante zur Beschleunigung des Strahlen-Schnittpunkttestes beim Vertex-Tracing dar. Wird das in Abschnitt 6.1.2 beschriebenen Shaft-Culling zur Detektion von Diskontinuitäten verwendet, kann es darüber hinaus auch für eine effizientere Kollisionsdetektion eingesetzt werden. Wurde einmal bestimmt, welche Objekte sich im Shaft der betrachteten Face befinden, muss lediglich ein Schnittpunkttest aller, von der Face ausgehenden Strahlen, mit diesen Objekten vorgenommen werden. Das heißt, auch Strahlen, die aus der Unterteilung einer Face resultieren, lassen sich relativ schnell testen. Dieser Ansatz ist vergleichbar mit der Vorgehensweise bei einer Strahlenklassifizierung. Potentiale solcher Klassifizierungen können insofern als Built-In des Vertex-Tracings angesehen werden.

6.2.3 Parallelisierung

Eine Optimierung des Laufzeitverhaltens beim parallelen Vertex-Tracing kann im Wesentlichen durch die Faktoren Load-Balancing und Kommunikation erzielt werden. Die Effizienz des Load-Balancing hängt wiederum signifikant von der Kommunikationsleistung des Systems ab. Nur bei kurzen Latenzzeiten und hohen Bandbreiten kann ein hoch dynamisches Load-Balancing wie das hier implementierte Queue Load-Balancing (↗ Abschnitt 4.3.3) einen guten Speed-Up des parallelen Systems erreichen. Potential

[4]Der Einsatz von *Object-Oriented* Bounding-Boxen (OOB) stellt einen klassischen Ansatz für eine Hierarchie bei dynamischen Kollisionsdetektionen dar.

zur Reduzieren des Kommunikationsaufwands beim Vertex-Tracings gibt es vor allem im Übertragungsvolumen der generierten Faces. Denkbar ist hier der Einsatz von Komprimierungsverfahren, die vor dem eigentlichen Datenversand ein entsprechendes Verpacken der zu übertragenden Informationen veranlassen. Um ein möglichst hohen Komprimierungsfaktor zu erreichen, sollte eine Sortierung des resultierenden Datenstroms erfolgen. Insbesondere Farbinformationen oder Textur-Information lassen sich im Verbund höher komprimieren.

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung und weiterführende Arbeiten

Das Ziel des in dieser Arbeit publizierten Verfahrens *Vertex-Tracing* bestand darin, spekulare Reflexionen beziehungsweise spekulare Refraktionen physikalisch korrekt in einer interaktiven Umgebung generieren zu können. Basierend auf der Tatsache, dass im Allgemeinen nur ein geringer Prozentsatz aller Szenenobjekte spekulares Verhalten aufweist, erfolgt eine Sonderbehandlung dieser Objekte, der sogenannten VT-Objekte. Die Abbildung der finalen Szene resultiert in einem hybriden Rendering, das sich einerseits aus OpenGL-Shading mittels Hardware und andererseits aus Software-Shading zusammensetzt.

Kern der Behandlung von VT-Objekten bildet ein adaptiv progressives Sampling der Radiance-Funktion, das sich auf einem Refinement im Objektraum stützt. Ausgangspunkt ist die Objektgeometrie aller VT-Objekte. Jeder ihrer Vertices dient als Initial-Sample-Position und damit als Startpunkt eines weiteren Ray-Tracings zur Bestimmung des endgültigen Sample-Wertes. Die Auswertung der ermittelten Samples erfolgt unter der Kombination verschiedener sogenannter Refinement-Kriterien. Sie bewirken ein effizientes Sampling der Radiance-Funktion in nur mit hoher Varianz versehenen Regionen. Die Einführung eines mehrstufigen Visibility-Tests erbrachte darüber hinaus eine weitere Reduktion der Sample-Anzahl. Er trägt dafür Sorge, dass nur in den tatsächlich sichtbaren Bereichen neue Samples generiert werden.

Die Rekonstruktion der Radiance-Funktion erfolgt durch bilinearer Interpolation mit Hilfe von OpenGL-Hardware, welches zugleich ein hybrides Rendering mit allen Nicht-VT-Objekten gestattet. Ferner eröffnete die Nutzung des Hardware-Renderings weitere Potentiale zur Beschleunigung des Vertex-Tracings. Gemeint ist damit die Einführung einer speziellen Textur-Behandlung innerhalb der ersten Rekursionsstufe des Vertex-Tracings. Die Verzögerung des Textur-Lookups bis in die Hardware verhindert vollständig ein Sampling von Texturen, das vor allem bei hochfrequenten Texturen zu erheblichem Mehraufwand führen würde. Die Darstellung von animierten Texturen kann auf diese Weise ebenfalls äußerst effizient durchgeführt werden. Im Falle einer unveränderlichen Kameraperspektive bedarf es bei animierten Texturen keinerlei Neuberechnung.

Im Hinblick auf die Beschleunigung des Vertex-Tracings wurden ferner die Aspekte einer verteilt, parallelen Berechnung untersucht beziehungsweise umgesetzt. Im Vordergrund stand dabei die Verteilung des Vertex-Tracings im Rechner-Cluster sowie eine parallele Abarbeitung auf Shared-Memory-Maschinen. Beide Ansätze zeigten, dass sich selbst mit dem adaptiv progressiven Charakter des Vertex-Tracings ein signifikanter Speed-Up erzielen lässt.

Trotz der Tatsache, dass sich das Verfahren Vertex-Tracing in Bezug auf sein Sampling dem Prinzip des Ray-Tracings bedient, stellt es jedoch mehr als ein weiteres Derivat eines Ray-Tracers dar. Das Vertex-Tracing schlägt vielmehr eine Brücke zwischen traditionellem Hardware-Rending und Standard-Ray-Tracing. Es schafft vor allem effiziente Erweiterung herkömmlicher Scan-Line-Verfahren im Hinblick auf globale Beleuchtungsphänomene. Ob Schatten, Reflexionen oder Refraktionen, alle Effekte lassen sich nun mit sub-linearer Zeitkomplexität bezüglich der Polygonanzahl hinzufügen und erfordern nicht jeweils einen weiteren Rendering-Pass der gesamten Szene, wie bei herkömmlichen Hardware-Rendering in der Regel der Fall. Zusammenfassend kann darüber hinaus auf folgende Vorteile des Vertex-Tracings verwiesen werden:

- Die Beschränkung auf ausgewählte Objekte der Szene ermöglicht eine äußerst effiziente und physikalisch korrekte Simulation spekularer Beleuchtungscharakteristiken.

- Das Vertex-Tracing stellt im weitesten Sinne eine Erweiterung von traditionellem Hardware-Rendering dar und kann somit problemlos in vorhandene Systeme dieser Art integriert werden.

- Das Vertex-Tracing skaliert gegenüber der Pixelanzahl der Bildebene sub-linear im Gegensatz zum Ray-Tracing, das sich in diesem Punkt linear verhält.

- Die Eigenschaft eines progressiven Vorgehens eröffnet die Möglichkeit einer Anwendung in hoch interaktiven Umgebungen wie beispielsweise in Virtual Reality der Fall.

- Das Vertex-Tracing verfolgt konsequent die Ausnutzung existierender Image-Kohärenz und benötigt demzufolge nur einen Bruchteil der Samples im Vergleich eines Standard-Ray-Tracers.

- Die enge Abhängigkeit des Samplings und des Refinements von der Objektgeometrie (Vertices, Edges, Faces) ermöglichen eine schnelle, qualitativ hochwertige Rekonstruktion der Radiance-Funktion. Die Detektion von Objektsilhouetten entfällt vollständig. Die Preview-Darstellung verschafft bereits eine exakte Repräsentation der Objektformen entsprechend der vorgegebenen Geometrie.

- Das Edge-basierte Refinement gewährleistet bereits ohne zusätzlich aufwendige Verfahren eine nahezu sichere Detektion von Diskontinuitäten und Nichtlinearitäten, da es implizit eine Zwangsverfeinerung adjazenter Faces unterstützt.

- Die Kollisionstests aller Primärstrahlen entfallen, da die Vertices der VT-Objekte und nicht die Pixel der Bildebene als Ausgangspunkt eines Basis-Ray-Tracings dienen.

- Der verzögerte Textur-Lookup für Texturen bis einschließlich zur ersten Rekursionsstufe des Ray-Tracings verhindert ein aufwendiges Sampling hochfrequenter Texturen und ermöglicht darüber hinaus die Anwendung von animierten Texturen ohne die Notwendigkeit einer Neuberechnung.

Neben den aufgeführten Vorteilen, sollen jedoch auch die Schwächen des Verfahrens an dieser Stelle zusammengefasst werden. Erstens, das Vertex-Tracing skaliert linear mit der Szenenkomplexität. Im Gegensatz zum traditionellen Ray-Tracing, dass sich sub-linear gegenüber der Szenenkomplexität verhält, bedarf es mindestens einem Rendering-Pass der darzustellenden Szene mittels Hardware-OpenGL. Solange dabei keine Occlusion-Culling-Mechanismen angewendet werden, dominiert der lineare Faktor des Hardware-Renderings die Zeitkomplexität des gesamten Vertex-Tracings. Zweitens, die Parallelisierung des Verfahrens gestaltet sich weitaus schwieriger als die eines Standard-Ray-Tracings. Aufgrund des adaptiven Charakters liegt es in der Natur des Verfahrens, stets eine sequentielle Abhängigkeit beizubehalten. Das heißt, eine parallele Berechnung ist immer mit einem weitaus höherem Kommunikationsbedarf verbunden, als beim Standard-Ray-Tracing der Fall.

Neben den inhärent, unausweichlichen Schwächen des Vertex-Tracings, gibt es jedoch eine Vielzahl an Möglichkeiten, die Effizienz des Verfahrens durch weiterführender Arbeiten zu steigern. Hierzu zählen in erster Linie Optimierungen bezüglich der Laufzeit, wie beispielsweise eine Beschleunigung des Samplings. Die Anwendung einer „State-Of-The-Art“-Kollisionserkennung, würde für das Vertex-Tracing enorme Laufzeitvorteile mit sich bringen, macht sie doch in etwa 95 % der Gesamtlaufzeit aus. Außer der Kollisionserkennung besitzt vor allem die Parallelisierung vielfältige Ressourcen zur Optimierung. Hier sind eine weitere Reduktion des Kommunikationsbedarfs sowie des erforderlichen Datenvolumens denkbar.

Potentiale zur Verbesserung des Verfahrens liegen ferner in der Rendering-Qualität. Hier gilt es insbesondere, den Fokus, auf eine noch sichere Detektion von Diskontinuitäten beziehungsweise Nichtlinearitäten zu lenken. Die Implementation von Shaft-Culling-Verfahren oder Intervall-Arithmetiken wäre hier durchaus viel versprechend. Eine ausführlichere Beschreibung möglicher Erweiterungen des Vertex-Tracings wurde bereits in Kapitel 6 erörtert.

7.2 Ausblick

Die Zukunft wird zeigen, wohin die Reise im Hinblick auf die Entwicklung von Rendering-Verfahren geht oder vielmehr, welche der bereits eingeschlagenen Richtungen sich als die pragmatischste erweist. Sicher ist nur, dass mit dem stetig wachsenden Bedarf an Szenenkomplexität und Realismus das traditionelle Scan-Line-Verfahren nicht die einzige

Alternative darstellen kann, stößt es doch durch seine stark redundante Charakteristik insbesondere bei hohen Szenenkomplexitäten an die Grenzen seines Basisalgorithmus. Hier können nur verschiedenste Optimierungsverfahren wie Dynamic Multi-Resolution, Occlusion-Culling oder hochgezüchtete, massiv parallele Graphikprozessoren helfen, den Anforderungen von zukünftigem Echtzeit-Rendering gerecht zu werden. Es liegt auf der Hand, dass in Zukunft das Thema „Visibility“ immer mehr an Relevanz erfährt. Auf längere Sicht werden sich die Verfahren durchsetzen, die diese Problematik am effizientesten beherrschen.

Ein solcher Kandidat ist sicherlich das Ray-Tracing in seiner ursprünglichen Form. Es verbindet eine effiziente Visibility-Betrachtung gleichsam mit der Möglichkeit globaler Beleuchtungsphänomene und einem charmanten Algorithmus. Dennoch, trotz der Existenz verschiedenster aktueller Real-Time Ray-Tracing-Systeme, kann von einer Salonfähigkeit des Algorithmus noch nicht unbedingt gesprochen werden. Vor allem im Hinblick auf Frame-Raten > 20 Hz, Stereoskopie, Auflösungen jenseits der SVGA oder hoch dynamischer Szenen verlangt das Ray-Tracing noch intensive Forschungsarbeit.

Eine Möglichkeit, die Lücke zwischen Ray-Tracing und Hardware-Rendering zu schließen, bietet zweifellos das im Rahmen dieser Arbeit entwickelte Verfahren Vertex-Tracing. Solange ein reinrassiges Ray-Tracing nicht hinreichend echtzeitfähig ist, wird man immer auf Hybridlösungen zurückgreifen müssen, um einerseits die nötige physikalische Korrektheit und andererseits die erforderliche Echtzeitfähigkeit zu wahren. Im Bezug auf das dabei stets auszuführende Spagat stellt das Vertex-Tracing mit seiner Einzigartigkeit im internationalen Vergleich eindeutig einen Beitrag auf diesem Forschungsgebiet dar.

Literaturverzeichnis

[AB91] E.H. Adelson and J.R. Bergen, *The Plenoptic Function and the Elements of Early Vision*, In Computation Models of Visual Processing, MIT Press, Cambridge (1991).

[AB99] Peter Apian-Bennewitz, *Studie zur Optimierung der Abdeckscheibe II*, Tech. report, DaimlerChrysler AG, November 1999.

[AC] John Amanatides and Kin Choi, *Ray Tracing Triangular Meshes*, Dept. of Computer Science York University North York, Ontario, http://citeseer.nj.nec.com/289766.html.

[AF84] John Amanatides and Alain Fournier, *Ray Casting Using Divide and Conquer in Screen Space*, International Conference on Engineering and Computer Graphics, August 1984, pages 290–296. f21g (1984).

[Ama84] John Amanatides, *Ray Tracing with Cones*, In Proceedings of SIGGRAPH'84 (1984).

[Amd67] G. M. Amdahl, *Validity of single-processor approach to achieving large-scale computing capability*, Proceedings of AFIPS Conference, Reston, page 483–485 (1967).

[AMS91] T. Akimoto, K. Mase, and Y. Suengaga, *Pixel-selected ray tracing*, IEEE Computer Graphics (1991), 11:14–22.

[AW87] J. Amanatides and A. Woo, *A Fast Voxel Traversal Algorithm for Ray Tracing*, Tech. report, Dept. of Computer Science, University of Toronto, 1987.

[Bad90] Didier Badouel, *An efficient ray polygon intersection*, In A. S. Glassner, editor, Graphics Gems. Academic Press Professional (1990).

[Bal99] Kavita Bala, *Radiance Interpolants for Interactive Scene Editing and Ray Tracing*, Phd thesis, Massachusetts Institute of Technology, 1999.

[Bei01] Daniel Beier, *Parallel progressives Vertex-Tracing mit dynamischem Load-Balancing*, Masters thesis, Technical Universität Ilmenau, 2001.

[BG89] Peter Burger and Duncan Gillies, *Interactive Computer Graphics*, Addison-Wesley, Wokingham, England, 1989.

[BN76] J. F. Blinn and M. E. Newell, *Texture and reflection in computer generated images*, Communications of the ACM 19, pages 542–546 (1976).

[BTB91] Christian Bouville, Pierre Tellier, and Kadi Bouatouch, *Low sampling densities using a psychovisual approach*, In Proceedings Eurographics '91, page 167-182, Amsterdam (1991).

[BW89] M. Born and E. Wolf, *Principles of Optics*, Pergamon, Oxford, 6th edition, 1989.

[Che95] Shenchang Eric Chen, *QuickTime VR – An image-based approach to virtual environment navigation*, In Computer Graphics Proceedings (SIGGRAPH '95), pages 29–38 (1995).

[CON99] Brian Cabral, Marc Olano, and Philip Nemec, *Reflection Space Image Based Rendering*, Siggraph 1999, Computer Graphics Proceedings (1999).

[Coo86] R. L. Cook, *Stochastic Sampling in Computer Graphics*, ACM Transactions on Graphics, vol. 5(1)), pages 51-72 (1986).

[CS93] J. L. D. Comba and J. Stolfi, *Affine arithmetic and its applications to computer graphics*, In Proceedings of SIBGRAPI '93, pages 9–18 (1993).

[CW93] Michael F. Cohen and John R. Wallace, *Radiosity and Realistic Image Synthesis*, Morgen Kaufmann Publisher, Inc., San Francisco, California, 1993.

[Die96] Paul Joseph Diefenbach, *Pipeline Rendering: Interaction And Realism Through Hardware-Based Multi-Pass Rendering*, Phd thesis, University of Pennsylvania, 1996.

[Dip85] M. A. Z. Dippé, *Antialiasing Through Stochastic Sampling*, Computer Graphics, vol. 19(3), pages 69-78 (1985).

[DS00] Sebastien Domine and John Spitzer, *Texture Shaders*, Tech. report, nVidia Corporation, http://developer.nvidia.com, 2000.

[ELPZ94] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi, *The farthest-point strategy for progressive image sampling*, In Proceddings, 12th International Conference on Pattern Recognition, Jerusalem (1994).

[Fel92] Wolf-Dietrich Fellner, *Computergrafik*, 2 ed., Wissenschaftsverlag Mannheim/Leipzig/Wien/Zürich, 1992.

[FvDFH90] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1990.

[GGSC96] Steven Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael Cohen, *The Lumigraph*, In Computer Graphics (SIGGRAPH '96 Proceedings), pages 43–54 (1996).

[GHC97] Steven J. Gortler, Li-Wei He, and Micheal F. Cohen, *Rendering layered depth images*, Tech. report, Microsoft Research, Redmond, WA, 1997.

[GKP96] A. Geist, J. A. Kohl, and P. M. Papdopoulos, *PVM and MPI: A Comparison of Features*, http://www.epm.ornl.gov/pvm/PVM (1996).

[Gla89] Andrew S. Glassner, *An Introduction to Ray Tracing*, Academic Press, 1989.

[Gla95] ———, *Principles of Digital Image Synthesis, Volume 1+2*, Morgan Kaufmann Publishers, Inc., San Francisco, 1995.

[Goe01] Steffen Goerzig, *CPPvm - C++ and PVM*, 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, Proceedings (2001).

[GP95] E. Gröller and W. Purgathofer, *Coherence in Computer Graphics*, Tech. report, Institute for Computer Graphics, Technical University Vienna, http://www.cg.tuwien.ac.at/research/TR/95/TR-186-2-95-04Abstract.html, 1995.

[Gre86] Ned Greene, *Environment mapping and other applications of world projections*, IEEE Computer Graphics and Applications, 6(11):21–29 (1986).

[Guo98] Baining Guo, *Progressive Radiance Evaluation Using Directional Coherence Maps*, Computer Graphics (SIGGRAPH 1998 Proceedings) (1998), 255–266.

[Han75] E. R. Hansen, *A generalized interval arithmetic*, In Interval Mathematics: Proceedings of the International Symposium, pages 7–18, Springer-Verlag (1975).

[Han86] Pat Hanrahan, *Using caching and breadth-first search to speed up ray tracing*, In Proceedings of Graphics Interface '86, pages 56–61 (1986).

[Hav00] Vlastimil Havran, *Heuristic Ray Shooting Algorithms*, PhD thesis, submitted to the Czech Technical University (2000).

[HB00] Vlastimil Havran and Jiri Bittner, *LCTS: Ray Shooting using Longest Common Traversal Sequences*, Proceedings of Eurographics (EG'00) (2000).

[Hec92] Paul S. Heckbert, *Discontinuity Meshing for Radiosity*, In Proceedings of the Third Eurographics Workshop on Rendering, pages 203–216 (1992).

[HG83] R. A. Hall and D. P. Greenberg, *A testbed for realistic image synthesis*, IEEE Computer Graphics and Application, 3(8) (1983).

[HH84] Paul S. Heckbert and Pat Hanrahan, *Beam tracing polygonal objects*, Computer Graphics (Proceedings SIGGRAPH'84), pages 119-127 (1984).

[HH97] Cameron Hughes and Tracey Hughes, *Object-Oriented Multithreading Using C++*, John Wiley and Sons, 1997.

[HK96] Ulrich Herzog and Rainer Klar, *Grundbegriffe der Leistungsbewertung*, In K. Waldschmidt, Seiten 41-62, Teubner Verlag, 1996.

[HKSS98] Wolfgang Heidrich, Jan Kautz, Philipp Slusallek, and Hans-Peter Seidel, *Canned Lightsources*, In Proceedings of the Eurographics Workshop on Rendering '98 (1998).

[HLCS99] Wolfgans Heidrich, Hendrik Lensch, Michael F. Cohen, and Hans-Peter Seidel, *Light field techniques for reflections and refractions*, In Eurographics Rendering Workshop 1999 (1999).

[HS98] Wolfgang Heidrich and H.-P. Seidel, *View-independent Environment Maps*, Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '98 (1998).

[Hub97] Walter Huber, *Paralleles Rechnen, Eine Einführung*, R. Oldenbourg Verlag München Wien, 1997.

[HW91] E. Haines and J. Wallace, *Shaft culling for efficient ray-traced radiositiy*, In Proceedings of 2nd Eurographics Workshop on Rendering (1991).

[JvW83] Frederik W. Jansen and Jarke J. van Wijk, *Fast Previewing Techniques in Raster Graphics*, Proceedings Eurographics (1983), 195–202.

[Kaj86] J. T. Kajiya, *The rendering equation*, In Computer Graphics (SIGGRAPH '86 Proceedings) page 143–150 (1986).

[KGGK94] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introdution to Parallel Computing*, Benjamin/Cummings, Redwood City CA, 1994.

[KK96] H.-J. Kim and C.-M. Kyung, *A new parallel ray-tracing system based on object decomposition*, The Visual Computer, 12(5):244–253 (1996).

[LH96] Marc Levoy and Pat Hanrahan, *Light Field Rendering*, In ACM Computer Graphics (Proceedings SIGGRAPH), pages 31–42 (1996).

[LKM01] E. Lindholm, M. Kligard, and H. Moreton, *A user-programmable vertex engine*, Proceedings of SIGGRAPH 2001, pp. 149–158 (2001).

[LR98] D. Lischinski and A. Rappoport, *Image-based rendering for non-diffuse synthetic scenes*, In Rendering Techniques '98, pages 301–314 (1998).

[LTG92] Dani Lischinski, Fillipo Tampieri, and Donald P. Greenberg, *Discontinuity meshing for accurate radiosity*, IEEE Computer Graphics and Applications, pages 25–39 (1992).

[MB95] L. McMillan and Gary Bishop, *Plenoptic Modeling: An Image-Based Rendering System*, In Computer Graphics (SIGGRAPH '95 Proceedings), pages 39–46. ACM SIGGRAPH (1995).

[MB99] Tom McReynolds and David Blythe, *Lighting and Shading Techniques for Interactive Applications*, SIGGRAPH'99 Course 12 (1999).

[MH99] Thomas Möller and Eric Haines, *Real-Time Rendering*, AK Peters Ltd, July 1999.

[MHE01] Marcelo Magallon, Matthias Hopf, and Thomas Ertl, *Parallel Volume Rendering Using PC Graphics Hardware*, In Proceedings of Pacific Graphics '01, Tokyo, Japan (2001).

[Mul93] S. Mullender, *Distributed Systems*, New York: Addison-Wesley, 1993.

[nC99] nVidia Corporation, *Perfect Reflections and Specular Lighting Effects with Cube Environment Mapping*, Tech. report, nVidia Corporation, http://developer.nvidia.com, 1999.

[NG95] Irena Notkin and Craig Gotsman, *Parallel Adaptive Ray-Tracing*, Proceedings of the Third International Conference in Central Europe on Computer Graphics and Visualisation 95, volume 1, pages 218– 226 (1995).

[NG97] ———, *Parallel Progressive Ray-Tracing*, Computer Graphics Forum, 16(1), pages 43–55 (1997).

[OR98] E. Ofek and A. Rappoport, *Interactive reflections on curved objects*, Proceedings of Siggraph 98. In Computer Graphics Proceedings, Annual Conference Series, 1998, ACM SIGGRAPH, pages 333-342 (1998).

[OW93] T. Ottmann and O. Widmayer, *Algorithmen und Datenstrukturen*, BI Wissenschaftsverlag, 1993.

[Pho75] Bui-Tuong Phong, *Illumination for Computer Generated Pictures*, Communications of the ACM, pages 311-317 (1975).

[PKGH97] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan, *Rendering complex scenes with memory-coherent ray tracing*, In SIGGRAPH 97 Conference Proceedings, Annual Conference Series, pages 101–108 (1997).

[PLS97] F. Pighin, Dani Lischinski, and David Salesin, *Progressive Previewing of Ray-Traced Images Using Image-Plane Discontinuity Meshing*, Rendering Techniques (1997), 115–126.

[PMSS99] S. Parker, W. Martin, P. Sloan, and P. Shirley, *Interactive Ray Tracing*, Interactive 3D (April 1999).

[PPL+99] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley, *Interactive ray tracing for volume visualization*, IEEE Transactions on Visualization and Computer Graphics, 5(3):238–250 (1999).

[PR] POV-Ray, *Persistence Of Vision Raytracer*, www.povray.org.

[PS89] James Painter and Kenneth Sloan, *Antialiased Ray Tracing by Adaptive Progressive Refinement*, Computer Graphics (SIGGRAPH '89 Proceedings) (1989), 281–288.

[PSL+98] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan, *Interactive Ray Tracing for Isosurface Rendering*, In Visualization '98, pages 233–238. ACM Press (1998).

[Rau93] Thomas Rauber, *Algorithmen in der Computergraphik*, B.G. Teubner Stuttgart, 1993.

[RB96] Gunther Raidl and Wilhelm Barth, *Fast Adaptive Previewing by Ray Tracing*, In Proceedings of 12th Spring Conference on Computer Graphics (1996).

[RGS97] A. Reisman, C. Gotsman, and A. Schuster, *Interactive-Rate Animation Generation by Parallel Progressive Ray-Tracing on Distributed-Memory Machines*, Proceedings of The Parallel Rendering Symposium, Phoenix (October 1997).

[RJ97] Erik Reinhard and Frederik W. Jansen, *Rendering Large Scenes Using Parallel Ray Tracing*, First Eurographics Workshop on Parallel Graphics and Visualisation, Bristol, Alpha Books, pages 67–80 (1997).

[Rot82] Scott D. Roth, *Ray casting for modeling solids*, Computer Graphics and Image Processing, page 109-144 (1982).

[RSH00] Erik Reinhard, Brian Smits, and Charles Hansen, *Dynamic Acceleration Structures for Interactive Ray Tracing*, In Proceedings Eurographics Workshop on Rendering (2000).

[Shi00] Peter Shirley, *Realistic Ray Tracing*, AK Peters, 2000.

[SKM98] László Szirmay-Kalos and Gábor Márton, *Worst-Case Versus Average Case Complexity of Ray-Shooting*, Computing, 61(2):103-131 (1998).

[SL91] Renben Shu and Alan Liu, *A fast ray casting algorithm using adaptive isotriangular subdivision*, Proceedings Visualisation '91, IEEE Computer Society, Los Alamitos (1991).

[Spi00a] John Spitzer, *Programmabel Texture Blending*, Tech. report, nVidia Corporation, http://developer.nvidia.com, 2000.

[Spi00b] _____, *Register Combiners*, Tech. report, nVidia Corporation, http://developer.nvidia.com, 2000.

[SSPS] Annette Scheel, Marc Stamminger, Jörg Pütz, and Hans-Peter Seidel, *Enhancements to Directional Coherence Maps*, url: citeseer.nj.nec.com/492134.html.

[TBD96] Seth Teller, Kavita Bala, and Julie Dorsey, *Conservative Radiance Interpolants for Ray Tracing*, Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering) (1996).

[UBSB01] Thomas Ullmann, Daniel Beier, Alexander Schmidt, and Beat Brüderlin, *Adaptive Progressive Vertex Tracing in Distributed Environments*, In Proceedings of Pacific Graphics '01, Tokyo, Japan (2001).

[USBB01] Thomas Ullmann, Alexander Schmidt, Daniel Beier, and Beat Brüderlin, *Adaptive Progressive Vertex Tracing for Interactive Reflections*, In Proceedings of EuroGraphics '01, Short Presentations, Manchester, UK (2001).

[vWvNJ91] Theo van Walsum, Peter R. van Nieuwenhuizen, and Frederik W. Jansen, *Refinement criteria for adaptive stochastic ray tracing of textures*, In Proceedings of Eurographics '91, page 155-166, Amsterdam (1991).

[Wal99] Mark Walmsley, *Multi-threaded Programming in C++*, Springer-Verlag UK, 1999.

[War94] Greg Ward, *The* RADIANCE *lighting simulation and rendering system*, In Proceedings of SIGGRAPH'94, pages 459–472 (1994).

[WBWS01] I. Wald, C. Benthin, M. Wagner, and P. Slusallek, *Interactive rendering with coherent ray tracing*, Proceedings of EuroGraphics 2001 (2001).

[WC93] Lance Williams and Shenchang Eric Chen, *View Interpolation for Image Synthesis*, In Computer Graphics, Annual Conference Series, pages 279–288 (1993).

[WFP+01] Michael Wand, Matthias Fischer, Ingmar Peter, Friedhelm Meyer auf der Heide, and Wolfgang Straßer, *The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes*, Computer Graphics (SIGGRAPH 01 Conference Proceedings), pages 361-370 (2001).

[WG84] H. Weghorst and G. Greenberg, *Improved Computational Methods for Ray Tracing*, ACM Transactions on Graphics (1984), 52–69.

[Whi80] Turner Whitted, *An Improved Shading Illumination Model for Shaded Display*, Communications of the ACM (1980), 23(6), S.343–349.

[WNDS99] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner, *OpenGL Programming Guide, Third Edition*, Addison Wesley, 1999.

[WW92] Alan Watt and Mark Watt, *Advanced Animation and Rendering Techniques, Theory and Practice*, Addison-Wesley Publishing Company, Inc., 1992.