Zusammenfassung Graphische Datenverarbeitung I

Contents

Computergrafik-Basics

Bildverarbeitung: Zur Bildverarbeitung gehören unter anderem Filter, Textursynthese, Dehazing, Image Descriptors and Retrieval und Objekterkennung

Modellierung: Unter dem Begriff Modellierung versteht man vor allem 3-dimensionale Modelle, welche durch z.B. Sketching erstellt werden können. Weitergehend gibt es Prozedurale Modelle, welche durch Anpassen von Parametern verändert, werden können und Punktwolken.

Rendering: Unter Rendering versteht man die Übertragung von geometrischen Primitiven in ein zweidimensionales rasterisiertes Bild. Auch Raytracing, Radiosity und globale Beleuchtung gehören zum Rendering. Beim Real-Time Rendering kommen Techniken wie Cel Shading, Dynamic Scenes, LOD, Subdiv und Bump Mapping zum Einsatz.

Simulation: Simulation physikbasierter Abläufe mittels Partikel oder Festkörper wie Stoff.

Visualisierung: Zum Beispiel Volume Rendering (dreidimensionale Daten werden in geometrischen Oberflächen visualisiert) und Scientific Visualization (visualisieren von wissenschaftlichen Daten, zB. Wettermodelle darstellen).

Hardware: moderne Hardware mit unified shader model

Ein- und Ausgabegeräte: Unter anderem VR und AR, haptisches Feedback und Tracker, wie Handschuhe oder Körpertracker aber auch Beamer und Wii-Remotes.

Benutzerschnittstellen: Multi-Touch Table

Digitalisierung: Übertragen von analogen Objekten in eine digitale Repräsentation.

Wahrnehmung: High-Dynamic Range

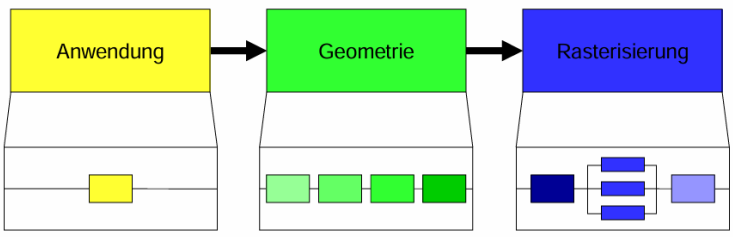
Pipeline

Erzeugung eines 2D Bildes für eine gegebene Szenenbeschreibung (Kamera, Modelle, Licht, Texturen, …)

Die Pipeline besteht aus 3 konzeptuellen Abschnitten, die jeweils aus mehreren funktionalen Abschnitten bestehen.

Die einzelnen Abschnitte der Pipeline können parallel ausgeführt werden. Dadurch lässt sich idealerweise die n-fache Geschwindigkeit beim Einsatz von n Abschnitten erreichen. In der Praxis ist die Geschwindigkeit der Pipeline durch den langsamsten Abschnitt beschränkt.

Die Grafikpipeline kann entweder als Fixed-Function Pipeline implementiert, sein oder als programmierbare Pipeline. Moderne Grafikkarten verwenden programmierbare Pipelines, da diese mehr Flexibilität ermöglichen.

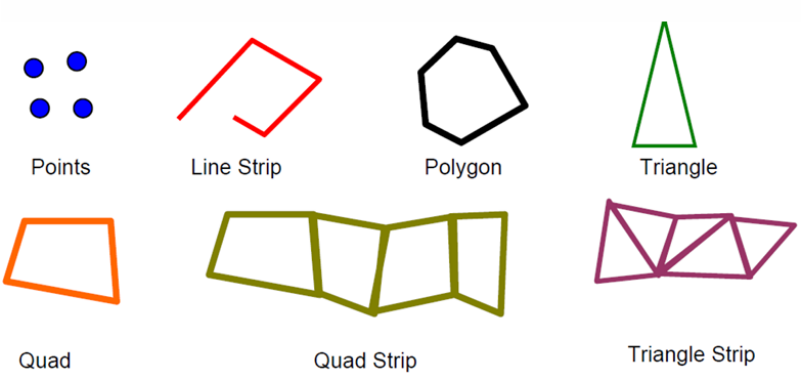


Anwendung

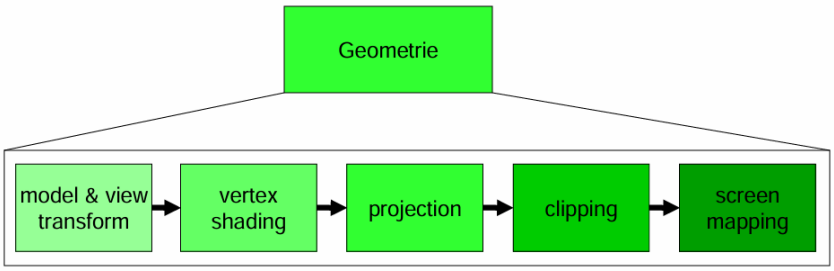
Die Anwendung wird auf der CPU ausgeführt und hat meistens wenig bis keine Parallelität. Die Performanz kann einfach optimiert werden, da die Anwendung komplett unter der Kontrolle des Anwendungsentwicklers ist. Die Anwendung übergibt die zu zeichnende Geometrie an den nächsten Abschnitt.

Geometrie

Graphische Primitiven in Open GL:



Der Geometrieschritt führt die meisten Operationen pro Polygon oder Vertex aus. Der Geometrieabschnitt ist in eine Reihe funktionaler Abschnitte unterteilt:



Model & View Transformation

Die Model-View Transformation dient dazu, Objekte aus ihrem lokalen Koordinatensystem in ein Weltkoordinatensystem und dann in das Kamerasicht-Koordinatensystem zu transformieren.

Die Model-Transformation beschreibt dabei die Transformation vom Model Space in den World Space und die View-Transformation beschreibt die Transformation vom World Space in den View Space.

Vertex Shading

Beim Vertex Shading werden die Beleuchtung und die Materialeigenschaften pro Vertex ausgewertet, um die Farbe des Vertex zu bestimmen. Eigenschaften wie Position, Normale, Farbe etc. können pro Vertex gespeichert werden. Diese Daten werden bei der Rasterisierung zur endgültigen Berechnung der Farbe eines Pixels verwendet. Die Berechnung findet meistens im Weltkoordinatensystem statt.

Projection

Bei der Projektion werden sichtbaren Volumen (View Frustum) in den „Einheitswürfel“ mit Extrempunkten (-1,-1,-1) und (1,1,1) transformiert. Dieser Schritt erfolgt, da dadurch alle Objekte und Szenen einheitlich dargestellt werden, auch kann man hier perspektivisch Projizieren.

Parallelprojektion: parallele Linien bleiben parallel, Kamera ist „im Unendlichen“. Kombination aus Translation und Skalierung.

Perspektivische Projektion: sichtbares Volumen bildet Pyramidenstumpf und wird ebenfalls auf den Einheitswürfel abgebildet.

Clipping

Beim Clipping werden Primitiven, die außerhalb des sichtbaren Volumens liegen verworfen. Primitive, die teilweise innerhalb des sichtbaren Volumens liegen, werden durch Primitive ersetzt, die ganz innerhalb des sichtbaren Volumens liegen.

Durch Projektion auf den Einheitswürfel erfolgt Clipping nur bei den sechs Ebenen, die diesen Würfel begrenzen.

Andere, benutzerdefinierte Clippingebenen sind möglich (genannt Sectioning).

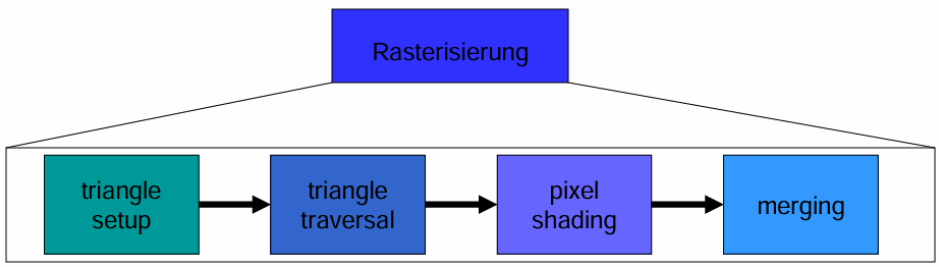
Screen Mapping

Hierbei werden die x- und y-Koordinaten der verbleibenden Primitiven auf Bildschirmkoordinaten abgebildet. Die z-Koordinate bleibt unverändert (also zwischen -1 und 1). Beim Screen Mapping wird zunächst eine Translation und dann eine Skalierung durchgeführt.

Rasterisierung

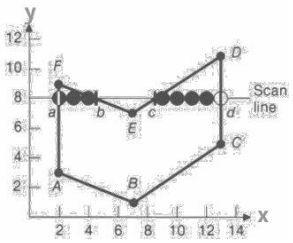
Der Rasterisierungsschritt bekommt als Input die verbleibenden Primitiven als 2D Vertices plus z-Koordinate, Shading Parameter etc.

Die Operationen werden pro Pixel bzw pro Fragment durchgeführt. Die Rasterisierung wird auch als Scan Conversion benannt. Die funktionalen Teilabschnitte dieses Schrittes lassen sich parallelisieren und sind die folgenden:



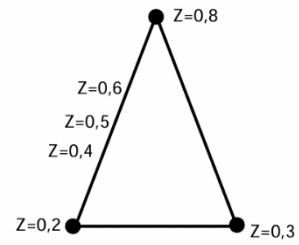
Triangle Setup

In diesem Schritt werden die Datenstrukturen zur Konvertierung der geometrischen Primitiven in Pixel vorberechnet. Es wird die Steigung (Slope) der Primitivkanten berechnet (für die Schnittberechnung). Dann wird die Schnittpunktberechnung zwischen Bildschirmzellen und der Primitivkanten ausgeführt. Durch die Sortierung der Schnittpunkte nach aufsteigenden x-Koordinaten erhalten wir so eine sortierte Zahlenliste zu jeder Bildschirmzeile des Primitivs. Diese Berechnung findet durch Fixed-Function Hardware statt.



Triangle Traversal

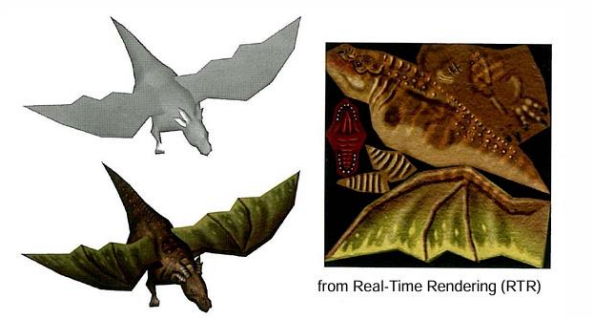
Beim Triangle Traversal (Scan Conversion) werden die Fragmente für alle Pixel, die ganz oder teilweise mit der gezeichneten Geometrie überlappen generiert. Es können mehrere Fragmente einer Pixelposition zugeordnet sein. Die Fragmente, welche durch Interpolation der Daten an den Vertices des Eingabedreiecks generiert werden, haben folgende Eigenschaften: Tiefe (z-Wert), Shading, Normale, Texturkoordinate etc. Auch dieser Schritt findet durch Fixed-Function Hardware statt.



Interpolation des z-Wertes eines Primitives

Pixel Shading

Beim Pixel Shading (Fragment Shading) wird das Shading berechnet. Das Ergebnis ist eine oder mehrere Farben pro Fragment, die an den nächsten Abschnitt der Pipeline weitergegeben werden. Dieser Schritt findet auf programmierbarer Hardware statt.



from Real-Time Rendering (RTR)

Merging

Das Ergebnis der ganzen Berechnung soll im **Color Buffer** gespeichert werden. Dabei wird ein Wert pro Pixel im Color Buffer gespeichert. Der Color Buffer ist ein 2D Array von RGB-Werten.

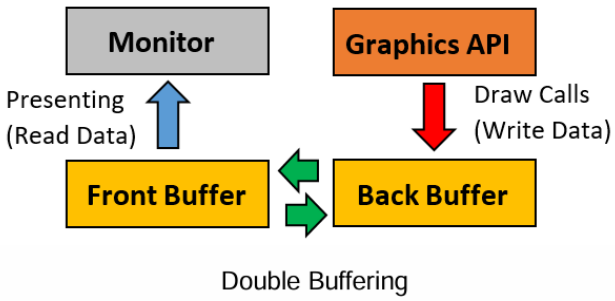
Beim Merging wird die Sichtbarkeit aller Fragmente berechnet. Der Tiefenwert jedes Pixels wird im **z-Buffer** (Depth Buffer) gespeichert. Der Tiefenwert wird mit dem aktuellen Tiefenwert im z-Buffer verglichen, bevor der Color Buffer und der z-Buffer geändert werden. Fragmente werden also verworfen, falls sie von anderen Fragmenten überdeckt werden.

Der z-Buffer Algorithmus berechnet die Sichtbarkeit mit Aufwand O(n) bei n gerenderten Primitiven. Die Reihenfolge der Abarbeitung der Primitive bzw. deren Fragmente ist irrelevant, außer es existieren teilweise transparente Primitiven bzw. Fragmente. Dann wird zusätzlich zu den RGB-Werten ein α-Kanal für die Transparenz benötigt. Der Farbwert eines Fragments wird mit dem vorhandenen Farbwert im Frame Buffer kombiniert (**Blending**). Nach dem Zeichnen aller undurchsichtigen Primitive müssen die restlichen von hinten nach vorne gezeichnet werden

Der **Stencil Buffer** erlaubt das Zeichnen von Schablonen (Stencil) in einem speziellen Buffer. Die Fragmente werden nur gezeichnet, wenn sie bestimmte Bedingungen im Stencil Buffer erfüllen.

Der **Accumulation Buffer** kombiniert mehrere gerenderte Bilder um z.B. Motion Blur, Depth of Field und Antialiasing zu ermöglichen. Diese Raster- bzw. Blend-Operation wird am Ende der Pipeline ausgeführt.

Der **Frame Buffer** speichert die Color und z-Buffer bzw. die Summe aller Buffer im System. Der Color Buffer wird auf dem Bildschirm angezeigt. Der Frame Buffer ist zweifach vorhanden (double buffering) um flimmern zu verhindern. Der Front Buffer wird momentan angezeigt und der Back Buffer wird zum Rendering verwendet. Die beiden Buffer werden ausgetauscht, nachdem das Rendering abgeschlossen ist.



Double Buffering

Bufferwechsel mit VSync off: Der Buffer wird gewechselt, sobald das Bild fertig gerendet ist. Dies kann zu **Tearing** fürhen. Bei Tearing besteht das auf dem Monitor sichtbare Bild aus mehreren Frames, wenn der Swap-Befehl während des Bildaufbaus kommt.

Bufferwechsel mit VSync on: Swap-Befehl zurückhalten, bis der Bildaufbau abgeschlossen ist (abhängig von der Monitor Refresh-Rate). Das sorgt für eine Verzögerung des Renderings, die GPU muss auf Swap warten, in extremen Fällen gehen dadurch mehrere Frames verloren.

Lösung: drei Buffer (tripple buffering) mit zwei Back Buffern. Der Front Buffer wird zum Bildaufbau verwendet und es wird abwechselnd in die beiden Back Buffer gerendert. Somit hat ein Buffer immer ein fertig gerendertes Bild, dieses wird mit dem Front Buffer geswappt.

Input

Interaktive Eingabegeräte 2D: Können entweder

- direkt (Touchscreen, Lichtgriffel) oder
- indirekt (Maus, Trackball (Optisch, Mechanisch), Grafiktablett) sein.

Interaktive Eingabegeräte 3D:

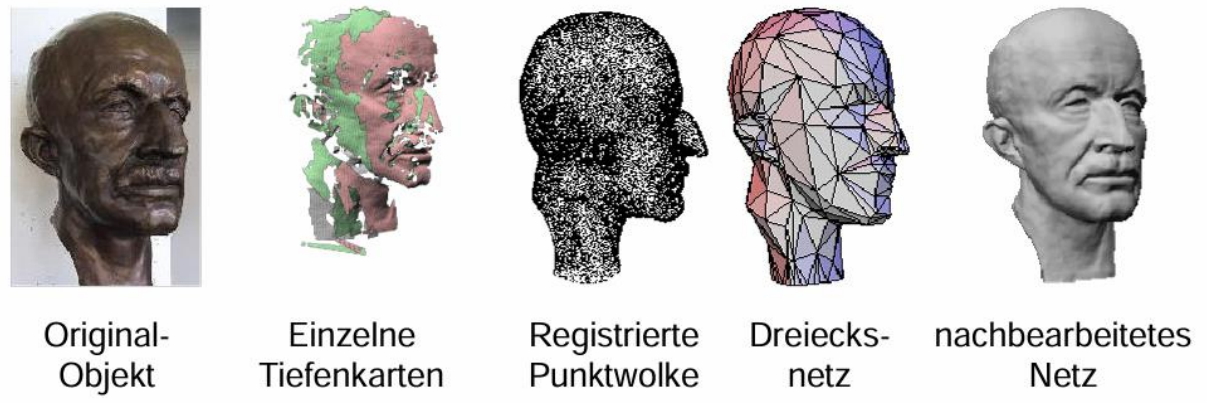
- Tracker mit 3 oder 6 Freiheitsgraden, absolute Position plus optional Richtung. Diese Tracker können mechanisch, optisch, magnetisch oder mit anderen Technologien funktionieren.
- Phantom (Force Feedback) mit 6 Freiheitsgraden über Gelenkwinkel, ein haptisches Feedback erlaubt virtuelles Berühren eines Objekts.
- Datenhandschuh: Bewegungsinformationen der Beugungs- und Abduktionssensoren werden erfasst und zur Schnittstelle des PCs geleitet, mechanisch oder optisch möglich.
- Motion Capture-Systeme erfassen die Position und Bewegung des Körpers. Können auch mechanisch oder optische Systeme sein. Bei optischen Systemen werden reflektierende Markierungen von speziellen IR-Hochgeschwindigkeitskameras erfasst. Die 3D-Positionen werden durch Triangulierung berechnet.

Akquisition

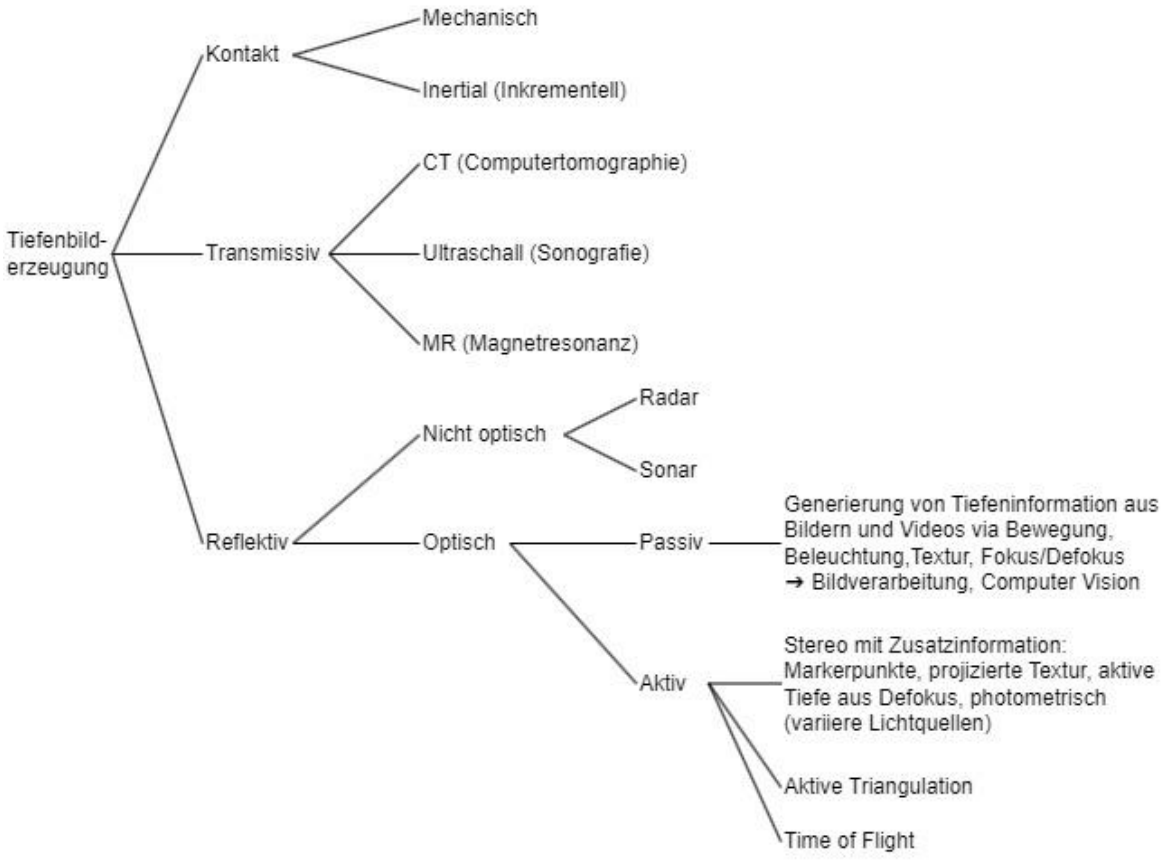
Erfassen von 3D-Modellen, Material, Texturen, Umgebung, Beleuchtung.

Abtastung realer Objekte

Aufnahme von mehreren Tiefenbildern (=diskrete Menge von Abtastpunkten/3D-Daten aus einer Sicht). Dann Registrierung, bei diesem Schritt werden die Tiefenbilder zu einer Punktwolke bzw. einem Dreiecksnetz überlappt und in ein gemeinsames Koordinatensystem transformiert. Danach folgen weitere Bearbeitungsschritte zum Erzeugen des endgültigen Modells.



Taxonomie



Optische Verfahren

Vorteile	**Nachteile**
Kontaktfrei	Abhängigkeit von optischen Eigenschaften des Objekts, Empfindlich gegenüber Transparent und Messfehler durch Spiegelungen
Sicher (Beschädigung des Objekts unwahrscheinlich)	Verdeckungsproblematik
Typischerweise kostengünstig und schnell	Textur

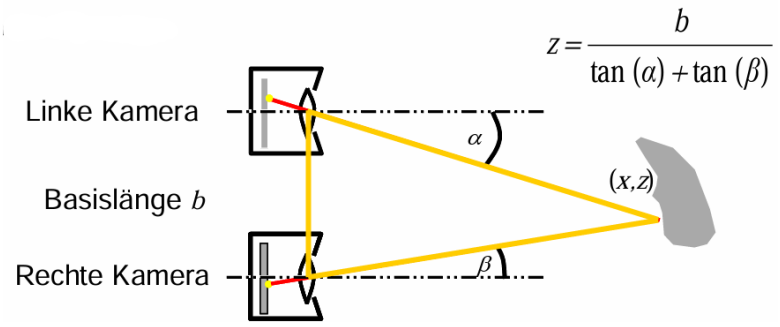
Aktive Optische Verfahren

Bei aktiven optischen Verfahren wird ein zusätzliches Signal in die Szene projiziert. Dieses zusätzliche Bauteil benötigt oftmals zusätzliche Kalibrierung aber erzielt oft eine hohe Präzision.

Vorteile	**Nachteile**
Erzeugen dichte Menge von Abtastpunkten (fast beliebige Feinheit des Abtastens)	Zusätzliche Lichtquelle (Projektor, Laser) in der Szene wird benötigt
Normalerweise robuster und genauer als passive Verfahren, insensitiver hinsichtlich Rauschens	Normalerweise teurer als passive, optische Methoden
Weniger rechenintensiv	

Optische Triangulation

Erzeugung von Tiefeninformationen aus Stereoinformationen (ähnlich dem menschlichen Sehen und kann Objekte, bis etwas zur Größe eines Menschen einscannen). Auch die optische Triangulation kann aktiv und passiv geschehen.



Bei passiven Systemen werden n≥2 Kameras verwendet. Die Herausforderung ist, die Korrespondenz in den Bildern zu finden.

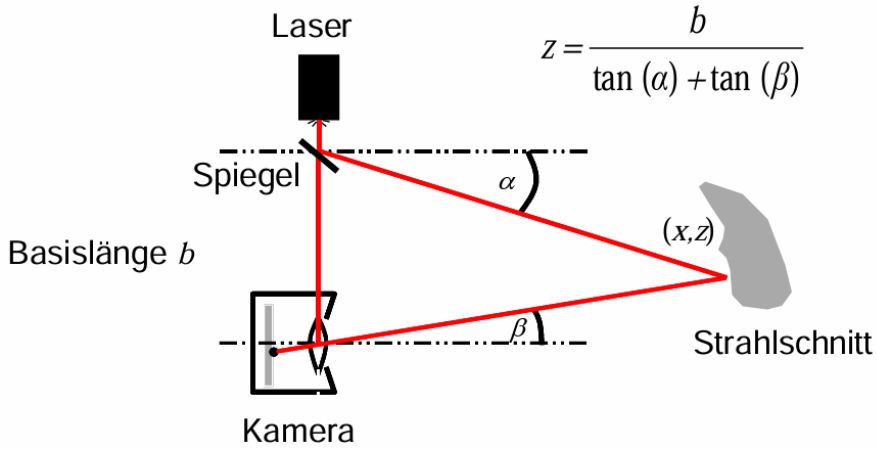
Epipolargeometrie: Zur Einschränkung des Suchraums beim Lösen des Korrespondenzproblems wird eine Kamerakalibrierung vorausgesetzt, dies wird mit Epipolargeometrie erreicht.

Wir haben einen Punkt X in der Szene, O_L beschreibt den Fokus der linken und O_R den Fokus der rechten Kamera. X_L und X_R sind die Projektionen von X auf die Bildebene der Kameras und e_L und e_R die Epipolarpunkte, also die Projektion der Fokusse auf die Bildebenen

Die Gerade $O_L X$ wird in der linken Kamera als Punkt X_L wahrgenommen, in der rechten Kamera als Gerade $e_R X_R$ (Epipolarlinie). Um die Kameras zu kalibrieren, wird zu einem Pixel in einem Bild die Epipolarlinie im anderen Bild berechnet und auf dieser eine Übereinstimmung gesucht.

Aktive optische Triangulation

Aktive Triangulation mit einem Laser und einer Kamera. Der Laser wird über die Szene bewegt. Der reflektierte Punkt kann eindeutig identifiziert und zur Triangulation verwendet werden. Zur verbesserten Abtastung wird häufig eine Laserlinie (aufgefächerter Laser) anstelle eines Laserstrahls verwendet.



Probleme optischer Triangulation:

- Ungünstige Materialeigenschaften des Objekts (dunkel, spiegelnd)
- Mehrfachreflexion des Objekts
- Ausgefranste Kanten am Rand des Objekts
- Texturrelief (erzeugt ungewollt real nicht vorhandene Geometrie)
- Verdeckung:
    - Kamera sieht den beleuchteten Punkt nicht (Kürzere Basislinie führt zu weniger Verdeckungen)
    - Punkt liegt im Schatten (Längere Basislinie führt zu erhöhter Präzision)

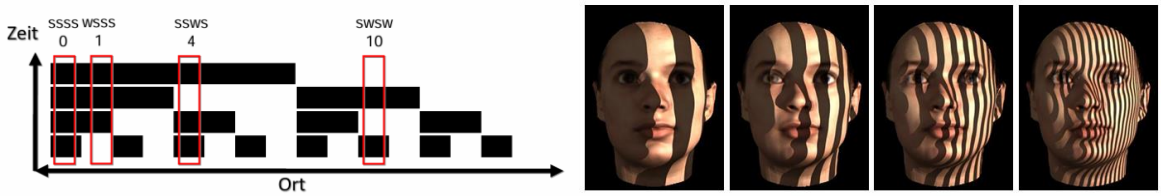
Streifenprojektionsverfahren

Beim Verwenden eines Streifenlichtprojektors (Lampe statt Laser) wird das reflektierte Streifenmuster analysiert.

Durch simultanes Erzeugen vieler Abtastpunkte kann die Aufnahmezeit reduziert werden.

Zunächst werden die Streifen in den Bildern der Kamera identifiziert, dann wird die Tiefe zurückgerechnet. Man nimmt hierfür an, dass die Projektor- und Kameraposition bekannt sind und die Oberfläche glatt ist (Streifen werden auf konsistente Kurven abgebildet).

Der Projektor beleuchtet das Objekt zeitlich sequenziell mit Mustern paralleler schwarz-weiß Streifen unterschiedlicher Breite. Die Kamera registriert das projizierte Streifenmuster unter bekanntem Blickwinkel. Für jedes Projektionsmuster wird ein Bild aufgenommen und für jeden Bildpunkt entsteht so eine zeitliche Folge unterschiedlicher Helligkeitswerte. Diese zeitliche Abfolge erlaubt das Berechnen der eindeutigen Nummer (Code) eines Streifens. Die Nummer dieses Streifens spezifiziert Lichtebene, Bildpunkt spezifiziert Lichtstrahl, Schnittpunkt ergibt Abtastpunkt.



Zur Verringerung/Erkennung von möglichen Ablesefehlern wechselt in jedem Punkt der Code nur einmal. Benachbarte Codes unterscheiden sich nur um ein Bit.

Das Streifenprojektionsverfahren lässt sich auch mit farblichen Streifen durchführen. Dabei werden die Farbstreifen zur Gewinnung der Tiefeninformationen verwendet. Die Abfolge der ausgesendeten Farbstreifen ist bekannt.

Besonderheiten:

- Genauigkeit abhängig vom Arbeitsbereich
    - typisch ~1000:1 – also: Genauigkeit etwa 1/1000 der gemessenen Distanz, z.B. 50 cm Arbeitsbereich, 0.5 mm Genauigkeit
- Funktioniert gut für kleine oder relativ nah abgetastete Objekte
- Schwierigkeiten für sehr große und weiter entfernte Objekte
    - Basislänge zu groß, Reflektion wird zu schwach
- Problem: mögliche Verdeckung des Kamerabildes oder der Lichtquelle
- Gradwanderung zwischen schlechter Abtastung für zu kleine Winkel und mehr Verdeckungsproblemen bei großen Winkeln
    - Praxis: sinnvoller Bereich für Triangulationswinkel: 15°-30°
- Komplexe Objekte erzeugen sehr große Mengen von Abtastpunkten; speicherintensiv
    - z.B. David, circa 5 Meter hoch, etwa 2 Milliarden Abtastpunkte, hunderte Scans
- Weiterverarbeitung erfordert Mesh-Dezimation, LOD (Level of detail techniques)

Time of Flight

Bei der Time of Flight Technik wird die Zeit Δt berechnet, die ein Lichtpuls (mit Lichtgeschwindigkeit c) von der Quelle zum Objekt und zurück benötigt. Daraus ergibt sich der Abstand r des Objekts, denn $r = \frac{1}{2} c\Delta t$. Dieses Verfahren wird bei diffus reflektierenden Objekten eingesetzt, z.B. bei Vermessung von Gebäuden, oder zur Abstandsbestimmung zwischen z.B. Flugzeug-Boden oder Mond-Erde.

Vorteile	Nachteile
Große bis sehr große Arbeitsvolumen	Mäßige Genauigkeit (~5 mm.)
Relativ kostengünstig	Schnelligkeit der Lichtausbreitung erfordert sehr genaue Zeitmessung im 30 Pico-Sekunden Bereich
Einfacher Aufbau	Unabhängig von der Objektgröße

Transformationen

Koordinatensysteme

n-dimensionales Koordinatensystem ist festgelegt durch einen ausgezeichneten Punkt w (Ursprung) und n linear unabhängige Vektoren $e_1, \dots, e_n$ (Basis). Der Ursprung und Vektoren sollten problemabhängig gewählt werden können.

In der Rendering-Pipeline werden mehrere Koordinatensysteme unterschieden:

- Weltkoordinaten (eye coordinates): beschreiben die gesamte Szene in 3D
- Objektkoordinaten (object coordinates): legen Lage von 3D-Objekten lokal fest
- Projektionskoordinaten (clip coordinates): erhält man nach Anwendung der Projektionstransformation (parallel oder perspektivisch)
- Normierte Koordinaten (normalized device coordinates)
- Bildschirmkoordinaten (window coordinates): stellen Szene in Fenster einer gewählten Größe und Position dar

Vektorräume

v heißt (reeller) Vektorraum, falls aus $u, v \in V$ und $a, b \in R$ stets $au + bv \in V$ folgt und $0 \in V$.

U heißt Unter-Vektorraum von V falls U eine Teilmenge von V mit Vektorraum-Struktur ist.

Affine Unterräume

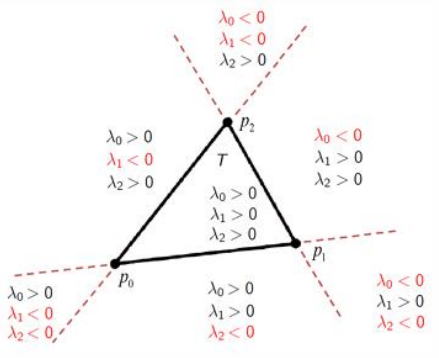
Affine Räume sind um einen festen Vektor w verschobene Unterräume U von V.

Definition: Eine Teilmenge A eines Vektorraums V heißt affiner Unterraum von V, falls $w \in V$ und ein Unterraum U von V existiert, sodass $A = w + U \coloneqq \{v \in V | es\ gibt\ ein\ u \in U\ mit\ v = w + u\}$. Affine Räume sind somit durch einen Ortsvektor w und eine Basis (Menge von linear unabhängigen Vektoren) von U festgelegt

Baryzentrische Koordinaten

Ist A ein affiner Raum des R^n festgelegt durch die Punkte $p_0, \dots, p_k$, so ist jeder Punkt p aus A darstellbar als $p = \sum_{i=0}^{k} \lambda_i p_i, \sum_{i=0}^{k} \lambda_i = 1$.

Die Koeffizienten λ_i heißen baryzentrische Koordinaten oder Schwerpunktskoordinaten von p hinsichtlich dem durch $p_0, \dots, p_k$, festgelegten k-Simplex im R^n.



Sind p_0, p_1 und p_2 drei Punkte in der Ebene $A \in R^2$, die ein Dreieck T bilden, so kann man jeden Punkt p aus R^2 schreiben als $p = \lambda_0 \cdot p_0 + \lambda_1 \cdot p_1 + \lambda_2 \cdot p_2$.

Baryzentrische Koordinaten sind dann lineare Polynome, die an genau einem Eckpunkt von T den Wert 1 haben und an den verbleibenden Eckpunkten 0 sind

Jeder Punkt in A lässt sich eindeutig durch baryzentrische Koordinaten beschreiben.

Baryzentrische Koordinaten besitzen das im Bild gezeigten Vorzeichenveralten.

Konvexe Hüllen

Konvexe Hülle von Punkten $p_0, \dots, p_m$ $co[p_0, \dots, p_k] = \{p = \sum_{i=0}^{k} \lambda_i p_i : \sum_{i=0}^{k} \lambda_i = 1 \ und \ \lambda_i \geq 0, i = 0, \dots, k\}$ besteht aus allen Konvexkombinationen (Affinkombinationen, bei denen alle baryzentrischen Koordinaten nicht negativ sind)**.**

Lineare Abbildungen

Eine Abbildung f ist linear, wenn sie folgende Eigenschaften für Vektoren x und y und Skalar k erfüllt

$$f(\vec{x}) + f(\vec{y}) = f(\vec{x} + \vec{y})$$

$$k \cdot f(\vec{x}) = f(k \cdot \vec{x})$$

Lineare Abbildungen lassen Ursprung invariant.

Skalierung: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} s_1 \cdot x \\ s_2 \cdot y \\ s_3 \cdot z \end{pmatrix}$

Rotation um Z: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Rotation um X: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Rotation um Y: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Scherung: $\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & s_2 & s_5 \\ s_1 & 1 & s_6 \\ s_3 & s_4 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Normalen können nicht mit der gleichen Transformationsmatrix multipliziert werden. Die Normalen-Eigenschaft (senkrecht auf der Oberfläche) wird gebrochen. Die Normale muss mit der transponierten inversen Matrix multipliziert werden: Sei $p' = M \cdot p$, dann gilt $n' = (M^{-1})^T \cdot n$

Affine Abbildungen

Eine Abbildung f ist affin, wenn sie in der Form $f(\vec{x}) = g(\vec{x}) + \vec{y}$ darstellbar ist und g eine lineare Abbildung ist.

Affine AbbildungenfürVektorenmit3 Elementenkönnennichtals3x3-Matrizen dargestellt werden.

$$\underbrace{\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}}_{f(\vec{x})} = \underbrace{\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{g(\vec{x})} + \underbrace{\begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}}_{\vec{y}}$$

Affine Abbildungen bilden Geraden auf Geraden ab, beschränkte Objekte bleiben beschränkt, Verhältnisse von Längen, Flächen, Volumen bleiben erhalten und parallele Objekte bleiben parallel.

Rotation, Skalierung und Scherung können durch lineare Abbildungen von R^3 nach R^3 (3 x 3 Matrix Vektor-Multiplikation) beschrieben werden.

Für Translationen (Verschiebungen) ist dies nicht der Fall, diese entsprechen “echten” affinen Abbildungen und benötige Vektor-Addition. Affine Abbildungen (und damit alle Transformation im 3D-Raum) können durch homogene (erweiterte) 4 x 4 - Matrizen beschrieben werden.

Homogene Koordinaten

Definiere Äquivalenzklasse:

$$\underbrace{\begin{pmatrix}x\\y\\z\end{pmatrix} \rightarrow \begin{bmatrix}x\\y\\z\\1\end{bmatrix} \cong \begin{bmatrix}sx\\sy\\sz\\s\end{bmatrix}}_{\substack{3D\ (inhomogene)\ Koordinate\\(runde\ Klammer)}} \quad \underbrace{\begin{bmatrix}x\\y\\z\\w\end{bmatrix} \cong \begin{bmatrix}x/w\\y/w\\z/w\\1\end{bmatrix} \rightarrow \begin{pmatrix}x/w\\y/w\\z/w\end{pmatrix}}_{\substack{4D\ homogene\ Koordinate\\(eckige\ Klammer)}}$$

Skalierungsfaktor s bzw. Gewicht w ungleich 0. Normalisierte homogene Koordinaten: s=w=1

Affine homogene Abbildung

Gründe für die Verwendung homogener 4x4-Matritzen in der GDV:

- Hintereinander Ausführung verschiedener Transformationen ->Lediglich Multiplikation von Matrizen gleicher Bauart
- Einheitliche Darstellung/Operationen -> einfache Implementierung, Hardwareumsetzung
- Homogene Koordinaten sind einfacher deutbar als ein erweitertes Rechenschema

Allgemeine Form von 3D-Transformationen:

$$\begin{bmatrix}x'\\y'\\z'\\w'\end{bmatrix} = \begin{bmatrix}a_{00} & a_{01} & a_{02} & t_x\\a_{10} & a_{11} & a_{12} & t_y\\a_{20} & a_{21} & a_{22} & t_z\\0 & 0 & 0 & a_{33}\end{bmatrix} \cdot \begin{bmatrix}x\\y\\z\\w\end{bmatrix}$$

linearer Anteil

affiner Anteil

inverse Streckungsfaktoren

Diese Definition (mit w‘=w=a33=1) ist äquivalent zur Definition der affinen Abbildung.

Translation als Matrix-Multiplikation in homogenen Koordinaten, der lineare Teil einer Translation ist die Identität.

Homogenisierter Vektor/Punkt: $(p) \rightarrow \begin{bmatrix}p\\1\end{bmatrix} \cong \begin{bmatrix}sp\\s\end{bmatrix}$

Punkte p im R^n werden durch die Ursprungsgerade im R^{n+1} repräsentiert, welche $\begin{bmatrix}p\\1\end{bmatrix}$ enthält.

Skalierung, Scherung und Rotation lassen den Ursprung invariant, d.h. sie besitzen keinen Translationsanteil und sind lineare Transformationen, in R^4 wird die 3x3-Matrix eingebettet. Nur die Skalierung hat hier einen Sonderfall, wenn die Skalierung in allen Koordinaten gleich ist, $s_1 = s_2 = s_3 = s$.

$$\begin{bmatrix}s & 0 & 0 & 0\\0 & s & 0 & 0\\0 & 0 & s & 0\\0 & 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & 0\\0 & 0 & 0 & 1/s\end{bmatrix}$$

Dreht man im Uhrzeigersinn um den Vektor r=(x,y,z)und den Winkel α, so gilt mit den Abkürzungen s=sin(α), c=cos(α)und t=1-cos(α)

$$R_{(x,y,z)} = \begin{bmatrix} t \cdot x^2 + c & t \cdot x \cdot y - s \cdot z & t \cdot x \cdot z + s \cdot y & 0 \\ t \cdot x \cdot y + s \cdot z & t \cdot y^2 + c & t \cdot y \cdot z - s \cdot x & 0 \\ t \cdot x \cdot z - s \cdot y & t \cdot y \cdot z + s \cdot x & t \cdot z^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Falls Rotationsachse nicht durch den Ursprung geht:

1. Verschiebe das Rotationszentrum in den Ursprung (Translation T)
2. Anschließende Rotation
3. Zurückverschieben in das Rotationszentrum (Translation T-1)

Kommutativität

Reihenfolge der Transformationen darf nicht vertauscht werden, da die Matrixmultiplikation nicht kommutativ ist.

Transformationen in OpenGL

OpenGL ist ein Zustandsautomat (state machine), befindet sich immer in einem aktuellen Zustand, welcher durch Zustandsvariablen (z.B. “Farbe”, ”Materialeigenschaften”,”Licht-quellen-Eigenschaften”,…) und Transformationen festgelegt ist, auch wenn gewisse Zustandsvariablen nicht explizit spezifiziert wurden.

Nicht jeder Funktionsaufruf enthält alle Parameter, sondern Werte werden so lange verwendet bis sich entsprechende Zustände ändern. Vorteil: kein permanentes Reorganisieren des Renderings.

OpenGL Programme bestehen aus 2 Teilen

- Aktueller Zustand: wie sollen Objekte gerendert werden?
- Festlegung der Objekte: was soll aktuell gerendert werden?

Jeder Vertex v einer Szene, bzw. jedes grafische Primitiv durchläuft alle dargestellten Transformationsstufen.

- Typisch: $v' = (VHPM)v$, wobei M,P,H,V = Matrix Modelview-Transformation, Projektionstransformation, Normalisierung und Viewport-Transformation
- Ungewohnt: Reihenfolge der Transformationsanwendung wird durch entsprechende Befehlsfolge in OpenGL umgekehrt (“von unten nach oben lesen”)

Transformationen T werden immer mit der gerade aktuell gültigen Transformation T_A akkumuliert. “Zurückgehen” in komplexen, hierarchischen Szenen wird durch Matrizen-Stack effizient realisiert.

OpenGL Matrix-Stack

Idee: speichere aktuelle Matrix vor Eintritt ins lokale System K und hole diese bei Verlassen von K wieder zurück. OpenGL-Befehle zur Realisierung des Matrizen-Stack

- glPushMatrix(): legt Kopie der aktuellen Matrix auf Stack („merke dir wo du bist“)
- glPopMatrix(): entfernt die oberste Matrix („gehe dahin zurück, wo du warst“)

Push und Pop bilden Klammern um die Transformationen in den untergeordneten (lokalen) Koordinatensystemen.

Space

Um Realtimerendering zu Beschleunigen von z.B. Culling verwendet man Hüllkörperhierarchien du Raum-Unterteilung, um festzustellen, welche Objekte nicht unnötig gerendert werden müssen.

Culling

Die Aufgabe des Cullings ist nicht sichtbare Teile der Szene abtrennen.

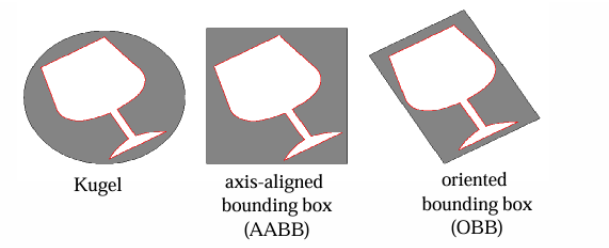
Dabei gibt es 4 Varianten:

- **view frustum:** Test des View Frustums (Sichtbarkeitsvolumen) gegen das jeweilige Bounding Volume. Wenn BV außerhalb des Frustums liegt, braucht der darunter liegende Teilbaum für das Rendering nicht weiter berücksichtigt werden.
- **Backface:** Zeige nur Vorderseiten
- **occlusion, portal:** Culling für Szenen mit gewissen räumlichen Eigenschaften. Wände verdecken ein Großteil der restlichen Objekte. Knoten im Szenengraph sind „Räume“ und „Portale“ zugeordnet. Portale sind Übergang zu anderen Räumen (Türen, Fenster)
- **detail:** Objekte unterhalb (oder nah bei) der Pixelauflösung übergehen

Hüllkörperhierarchie

Um jedes Objekt wird ein Hüllkörper (Bounding Volume) einfacher Geometrie berechnet, welcher es umschließt Test gegen BV ist oft effizienter als Test gegen alle graphischen Primitive der Objekte. Die Hüllkörperhierarchie (Bounding Volume Hierarchy, BVH) ist eine Verschachtelung mehrerer Hüllkörper.

Hüllkörper sollten einfach sein, Schnitt-Tests mit anderen Primitiven (Sichtvolumen, Sehstrahl) müssen sich sehr einfach berechnen lassen Hüllkörper sollten die Objekte genügend genau beschreiben. Standard-Hüllkörper:



Zusammengesetzte Hüllkörper (CSG= Constructive Solid Geometry) ergeben sich aus Standard Hüllkörpern durch Vereinigung, Schnitt und Differenzenbildung. Abwägen zwischen Passgenauigkeit und Geschwindigkeit/Aufwand der Berechnung.

BVH werden beim **View Frustum Culling** eingesetzt, man beginnt bei der Wurzel des Hierarchie-Graphens und testet jeden Knoten auf Schnitt mit der Blickpyramide. Falls der Schnitt nicht leer ist, traversiert man den Graphen rekursiv weiter mit den Kindern. Ansonsten stellt man die weitere Bearbeitung des Knoten ein. BVH erlaubt also effizientes Abschneiden der nicht sichtbaren Teilgraphen, aber BVH ist nicht effizient für Occlusion Culling.

BV können in **Weltkoordinaten** (Lokale Änderungen in der Szene erfordert globales Update) und in **Modellkoordinaten** (Erfordert explizite Rückrechnung in Weltkoordinaten) benötigt werden. Wichtig ist, dass die Hierarchie der Szene eine räumliche Aufteilung bzw. Organisation aufweist. Objekte, die nah beieinander sind, sollten auch strukturell benachbart sein. Die BVH können Bottom-up, Top-down oder per Insertion konstruiert werden.

Schnitt-Test für einfache Hüllkörper

Schnitt-Tests mit Strahl $r(t)$: $p = a + tb, t, c \in R, p, a, b, n \in R^3$

Für Ebene E: $n^T p + c = 0$ (n = Ebenennormale)

- $n^T(a + tb) + c = 0$

- $-n^T a - c = t(n^T b)$
- 3 Fälle: keine, genau eine, unendlich viele Lösung(en)

Für Kugel K: $(p - m)^2 = c^2$

- quadratische Gleichung: $(a + tb - m)^2 = c^2$
- 3 Fälle: keine, genau eine, genau zwei Lösung(en)

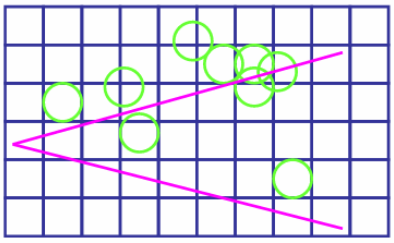
Raum-Unterteilung

Die Wahl der optimalen, räumlichen Datenstruktur ist **problembezogen**.

Für die Nachbarsuche eignet sich Octree, denn die Teilung im Objektmedian sorgt für Strukturierung nah beieinander liegender Objekte.

Für die Traversierung (view frustum culling) eignet sich kd-tree, denn große leere Gebiete sollten weit oben im Baum zu finden sein und Objekte selbst sollten „eng“ eingeschlossen werden.

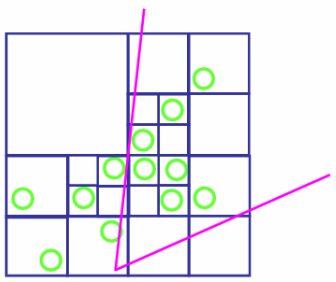
Gitter



Reguläre Gitter z.B. achsenparallel sind sehr einfach. Objekte sind im Allgemeinen in mehreren Zellen enthalten (Objektredundanz). Das reguläre Gitter kann sich der Geometrie nicht anpassen und ist sehr speicheraufwändig. Dafür aber effizient traversierbar, erlaubt einen schnellen Zugriff auf den Nachbaren und schnelle Updates.

Octree

Ein Octree ist eine hierarchische, objektabhängige Unterteilung des Raums mit achsenparallelen Boxen unterschiedlicher Größe.



Octree-Algorithmus:

- Schließe Szene durch Box minimaler Größe ein.
- Falls Stop-Kriterium erreicht: Binde Objekte an aktuelle Box, beende Rekursion.
- Sonst: unterteile aktuelle Box gleichmäßig am Objektmedian in acht achsenparallele Sub-Boxen und verfahre mit diesen genauso.
- Stop-Kriterium: maximaler Wert für Rekursionstiefe, Anzahl der Primitive liegt unter Threshold.

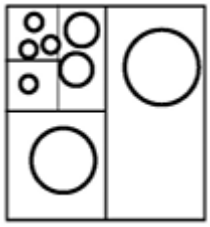
Aber:

- Problem mit Objekten, die Mittelpunkt einer Zelle enthalten (Grund: starre Unterteilung am Median der aktuellen Box)
- Suboptimal, falls Objekte sich sehr ungleichmäßig verteilen
- Entstehender Baum ist im Allgemeinen nicht balanciert (somit: keine logarithmische Tiefe)

Kd-Tree

Der kd-Tree ist eine flexiblere Variante des Octrees.

Kd-Tree-Algorithmus:



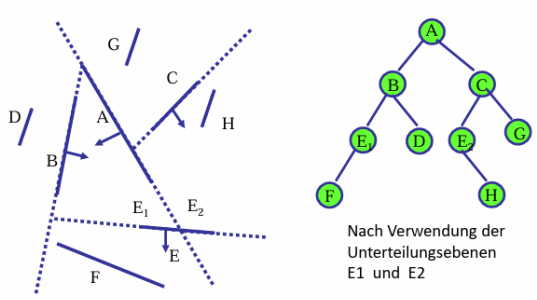
- Objekte werden nach den Koordinaten der längsten aktuellen Bounding Box B sortiert
- Suche B und wähle entsprechende Achse
- Eventuell zyklische Vertauschung der Achsen: x,y,z,x,y,z,…
- Rekursive Unterteilung der aktuellen Objektmenge in zwei disjunkte (etwa gleich große) Teilmengen durch angepasste achsen-parallele Schnittebene
- Abbruchkriterien: wie bei Octree

BSP-Tree

BSP-Bäume sind eine rekursive Unterteilung des Raums in zwei Halbräume durch Verwendung beliebiger Unterteilungsebenen. Die BSP-Trees verallgemeinern kd-trees (= axis-aligned BSP trees). In jedem Schritt wird der Raum in zwei Halbräume unterteilt und jeder Knoten entspricht einer Unterteilungsebene

BSP-Algorithmus:

- Ebenensegmente (mit Normalen) werden in alphabetischer Reihenfolge eingelesen
- Rekursives Einsortieren der verbleibenden Ebenensegmente in linken oder rechten Halbraum
- Splitte nicht entscheidbare Ebenensegmente



Probleme des BSP-Trees:

- Optimierung der Auswahl von Unterteilungs-Ebenen ist für allgemeine Szene komplex
- Selbst wenn Ebenen bekannt sind: welche Reihenfolge? Ebene Objekte (z.B. Dreiecks-Primitive) können als Splitting-Ebene verwendet, werden
- Regeln/Heuristiken: Balanciere Größe der entstehenden Zellen und Schnitte von Objekten durch Unterteilungsebenen möglichst meiden

Vorteil BSP-Tree: Test in welchem durch eine Ebene E erzeugten Halbraum ein Punkt p (z.B. eyepoint) liegt ist sehr einfach. Annahme E liegt in parameterfreier Form vor: E(p)=0.

Für den BSP gibt es **6** (feste) **Durchlaufstrategien**: LWR, RWL, WLR, WRL, LRW, RLW (R-Rechts, L-Links, W-Wurzel). BSP-Trees sind zur Traversierung der Geometrie aus einer festen Betrachter Sicht besonders geeignet, da sie unabhängig von der gewählten Blickrichtung verwendbar sind.

Hüllkörperhierarchien sind Objektzentrisch und räumlich redundant.

Raumunterteilung ist Raumzentrisch und objektredudant.

Methode	Nachteile	Vorteile
Hüllkörper-hierarchie	- Finden von optimalen BVs und BV-Hierarchien ist nicht trivial - BV-Test auf jeder Hierarchiestufe - Keine besonderen Strategien für Nachbarsuche und Traversierung	- Einfache Implementierung - Unabhängig von räumlicher Verteilung der Objekte - Schnelle Baumkonstruktion
Reguläres Gitter	- Speicherintensiv - Schlecht bei ungleichmäßig verteilten Objekten	- Sehr einfache Wartung (z.B. bei dynamischen Szenen)

			- Schnelle Nachbarsuche, einfache Traversierung
Octree	- Nachbarsuche nicht trivial (Baum Traversierung)	- Ungünstige Objekt Konstellationen möglich - Hoher Verzweigungsgrad	- Schnelle Baumkonstruktion - Schnelle Traversierung möglich
Kd-Tree BSP-Tree		- Balancieren der Bäume problematisch	- Im Schnitt schneller als Octree

Szenengraph

Der **Szenengraph** ist eine hierarchische Datenstruktur zur logischen und räumlichen Organisation von Szenen mit dem Ziel effizientes Rendering der (Teil-) Szenen zu ermöglichen. Der Szenengraph ist **gerichtet** und **azyklisch**. Der Wurzelknoten enthält die gesamte Szene (Universum) und die inneren Knoten beinhalten die logische Struktur und Kontrolle der Teilbäume. Die Blätter beinhalten die Geometrie /die Primitiven.

Eine **Szene** besteht aus den räumlichen Datenstrukturen und Animationen, Sichtbarkeit, Zuständen zum Rendern der Geometrie wie Materialeigenschaften, Transformationen, Texturen, Lichtquellen, Level of Detail, …

Zu den **Aufgaben** des Szenengraphs zur Performanz Steigerung gehören:

- Nur das aktuell Sichtbare soll dargestellt werden -> Culling Algorithmen
- LOD: sehr weit entfernte Objekte einfacher darstellen
- Ausrichtung von Billboards zum Augenpunkt hin
- State-Changes minimieren
- Redundanzen vermeiden

Sichten

Transformationssicht:

- Jeder Knoten hat eigene Transformation
- Fasse Teilbaum als Szene in lokalem Koordinatensystem auf
- Traversierung der Hierarchie akkumuliert die Transformationen der gesamten Welt und positioniert die „lokalen“ Koordinatensysteme an die richtige Stelle
- Transformationen werden gespeichert, damit sie nach Verlassen des Teilbaums restauriert werden können

Allgemeinere Betrachtung:

- Innere Knoten des Szenengraphen können für viel allgemeinere Zwecke (als lediglich Transformationen) designed sein

Allgemeine Konstruktion

Interne Knoten für generellere Zwecke (Beispiele):

- Gruppenknoten (Landschaft wird von Baum und Straße gebildet)
- Switch-Knoten (Bahnschranke offen/zu)
- Sequence-Knoten (Zuschalten von Ampelsignalen in Zyklen)
- Transformationsknoten
- LOD (level-of detail) (wähle Detailstufe eines Objekts in Abhängigkeit des Betrachter Standpunkts)

Aufteilung in Geometrie, Aktionen, Transformationen

OpenSG

OpenSG (Open SceneGraph, IGD) ist kein gerichteter, azyklischer Graph (DAG), sondern lediglich ein Baum. Der Szenengraph besteht lediglich aus logischen Knoten und Geometrie („backbones“, Zeiger – kein Knoten)

Warum kein DAG? Position/Transformation sollte eindeutig sein, auch bei Durchlauf „von unten nach oben“

Multi-Threading: Verlässliche Simultan-Manipulation des Szenengraphen, ermöglicht Datenzugriff von mehreren Clients/Threads, Verteilung der Last auf einem Prozessor

Gruppenknoten:

- Group: Basis aller Gruppen, traversiert alle Teilbäume
- Transform: beschreibt Transformation in lokale Koordinaten
- Switch: erlaubt flexible Auswahl der zu traversierenden Teilbäume
- Billboard: zeigt Bild eines Objekts um Geometrie vorzutäuschen
- DistanceLOD: wählt Auflösung eines Objekts relativ zur Betrachter Position
- Volume: Einbindung von Volume Rendering Optionen
- Lights

Billboards: Verwende Fototexturen (Bilder) um Geometrie vorzutäuschen

- Normale Billboards: ein Bild
- Blickwinkelabhängige Billboards: richten sich zum Betrachter hin aus
- Verwendung unter anderem bei entfernter komplexer Geometrie und Partikelsystemen

Volume Rendering Option: Darstellung und Weiterverarbeitung von Voxeldaten Z.B. medizintechnische Anwendungen.

Light: Lichtquellen besitzen zwei Bezugsknoten im Szenengraphen. Erstens, Teilbäume, die von der Lichtquelle beleuchtet werden (was wird beleuchtet), diese sind definiert durch den Knoten im Baum und alle Teilbäume werden (auf eventuell unterschiedliche Art) beleuchtet. Zweitens, Knoten der durch sein lokales Koordinatensystem Position und Orientierung beschreibt (wo ist das Licht).

DistanceLOD

DistanceLOD (Level of Detail) wählt einen darstellenden Teilbaum in Abhängigkeit zur aktuellen Entfernung des Betrachters.

- Rechnet aktuelle Entfernung aus
- Enthält eine sortierte Liste von Entfernungen
- Einen Wert mehr als vorhandene Teilbäume
- Liste definiert Intervalle für die Teilbäume

Für DistanceLOD gibt es 3 Standard-Ansätze: switch, fade und morph

Switch LOD: in Abhängigkeit der Distanz r des Betrachters wird genau ein Detailknoten zu 100% dargestellt, alle anderen zu 0%. Schlagartiger Wechsel der Detailstufen führt zu unerwünschten pop-up Effekten

Fade LOD: langsames Überblenden zwischen benachbarten Detailstufen, $fade_i$ legt Breite des Übergangsbereich fest, dadurch wird der Pop-up-Effekt eliminiert aber Überblenden erfordert, dass an diesen Stellen 2 LOD-Stufen gleichzeitig gerendert werden.

Morph LOD: verwende lokale Kriterien zur Mesh-Dezimierung, interpoliere umgekehrt lokale Surface-Details durch geeignete Anzahl linearer Übergänge (Morph-Operationen).

Projektionen

Projektiver Raum

Menge aller 1-dimensionalen Unterräume $P(V) = \{U: U = \{t\ v\}: v\ aus\ V\ \backslash\{0\}\}$ wird projektiver Raum eines (reellen) Vektorraums V genannt.

$V = R^n$: $P(R^n) = \{\{g: x = tv\}: v\ aus\ R^n\backslash\{0\}\}$ besteht aus der Menge aller Ursprungsgeraden.

Elemente $g = tv$ von $P(V)$ heißen "Punkte" des projektiven Raums $P(V)$.

Bezeichne mit $dim\ P(R^n) = dim\ R^{n-1} = n - 1$ die Dimension des Projektiven Raums.

Homogene Koordinaten

Ist v ein nicht-verschwindender Vektor aus R^4, so ist $g: x = tv$ eine Gerade im R^4 durch den Ursprung des R^4.

Erhalte Abbildung von $R^4\backslash\{0\}$ in den projektiven Raum $P(R^4)$, die jedem v aus $R^4\backslash\{0\}$ den projektiven Punkt $g = tv$ aus $P(R^4)$ zuordnet.

Zwei verschiedene, nicht-verschwindende Vektoren u und v bestimmen denselben projektiven Punkt, genau dann, wenn u und v linear abhängig sind, das heißt: $u = sv$ für ein s aus R

Ist $v = (x, y, z, w)^T$ wie oben, so bezeichnet $[x, y, z, w]^T = t\ (x, y, z, w)^T$ aus $P(R^4)$ die homogenen Koordinaten von v: $[u]^T = [v]^T$ genau dann, wenn $u = sv$ für ein s aus R

Einbettung

R^3 lässt sich in den Projektiven Raum $P(R^4)$ einbetten.

Die Abbildung von R^3 nach $P(R^4)$ festgelegt durch $(x, y, z)^T \rightarrow [x, y, z, 1]^T$ ordnet jedem Punkt im R^3 einen projektiven Punkt von $P(R^4)$ zu.

Bildpunkt von $(x, y, z)^T$ ist Ursprungsgerade im R^4 durch $(x, y, z, 1)^T$. Der Ursprung $(0,0,0)^T$ wird dabei auf den projektiven Punkt mit den homogenen Koordinaten $[0,0,0,1]^T$ abgebildet und bei dieser Abbildung treten projektive Punkte $[x, y, z, 0]^T$ nicht als Bilder auf (vierte homogene Koordinate $w = 0$)

Zusammenhang/Vorstellung: R^3 ist assoziiert mit Ebene $w = 1$ im R^4.

Zentralprojektion

Umgekehrt wird jedem projektiven Punkt $[x, y, z, w]^T$ aus $P(R^4)$, mit $w \neq 0$, ein Punkt im R^3 wie folgt zugeordnet: $[x, y, z, w]^T \rightarrow \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)^T$

Geometrische Interpretation: R^3 erhält man durch Zentralprojektion der Punkte im vierdimensionalen Raum mit $w \neq 0$ auf die Ebene $w = 1$, wobei der Ursprung des R^4 das Projektionszentrum ist.

Projektive Fernpunkte

Was passiert mit den projektiven Punkten der Form $[x, y, z, 0]^T$, also Ursprungsgeraden mit 0 in letzter Komponente des Richtungsvektors?

Antwort: diese werden als “unendlich ferne Punkte” im R^3 interpretiert

Intuitiv: $\left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)^T$ verhält sich in dieser Weise, falls w gegen 0 geht

Etwas präziser: $[x, y, z, 0]^T$ ist offenbar der Grenzwert von $\left[x, y, z, \frac{1}{s}\right]^T = [sx, sy, sz, 1]^T$ für $s \to \infty$

Die Punkte $(sx, sy, sz, 1)^T$ liegen auf einer Geraden $g(s)$ in der Ebene $w = 1$, welche durch den Punkt $(0,0,0,1)^T$ in Richtung $(x, y, z, 0)^T$ verläuft: $g(s) = (0,0,0,1)^T + s(x, y, z, 0)^T$

Daher ist $[x, y, z, 0]^T = \lim_{s \to \infty} g(s)$ ein unendlich ferner Punkt in der Ebene $w = 1$, bzw. im R^3

Bedeutung: beschreibt Punkte, die unendlich weit entfernt sind; diese fasst man als eine Richtung auf.

$H = \{[x, y, z, 0]^T\}$ heißt unendlich ferne Hyperebene in $P(R^4)$. $P(R^4)$ enthält affinen Raum R^3 und die “Richtungen” aus H. $P(R^4)$ hat keine Vektorraum-Struktur!

Addition und skalare Multiplikation lassen sich nicht sinnvoll auf projektive Punkte (= Ursprungsgeraden) übertragen.
Jedoch: projektive Punkte können im Projektiven Raum mittels projektiver Abbildungen verändert werden

Projektive Abbildung

Φ heißt projektive Abbildung von $P(R^4)$ nach $P(R^4)$, falls es eine lineare, injektive Abbildung Ψ von R^4 gibt, so dass $([v]^T) = [\Psi(v)]^T$ für alle $v \neq 0$

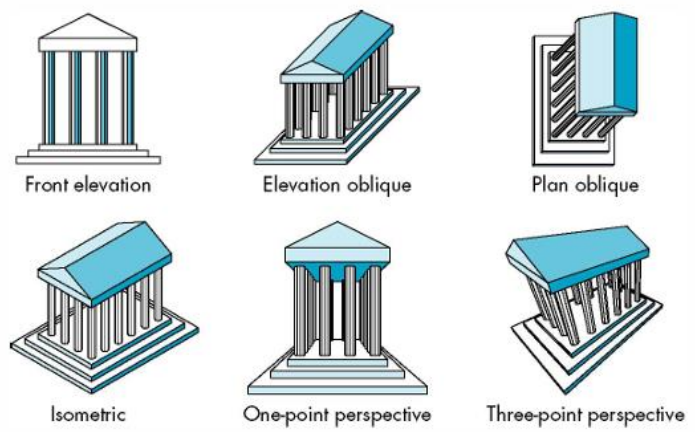
Projektive Abbildungen Φ von $P(R^4)$ in den $P(R^4)$ können durch homogene 4×4-Matrizen beschrieben werden.
Zwei injektive, lineare Abbildungen bestimmen dieselbe projektive Abbildung Φ genau dann, wenn deren zugehörigen 4x4-Matrizen A und A’ die Bedingung $A = \lambda \cdot A$’ für ein $\lambda \neq 0$ gilt (d.h. homogen).

Im Allgemeinen lassen projektive Abbildungen die unendlich ferne Hyperebene H nicht invariant!
Eine projektive Abbildung Φ ist genau dann affin, falls $\Phi(H) = H$ gilt.

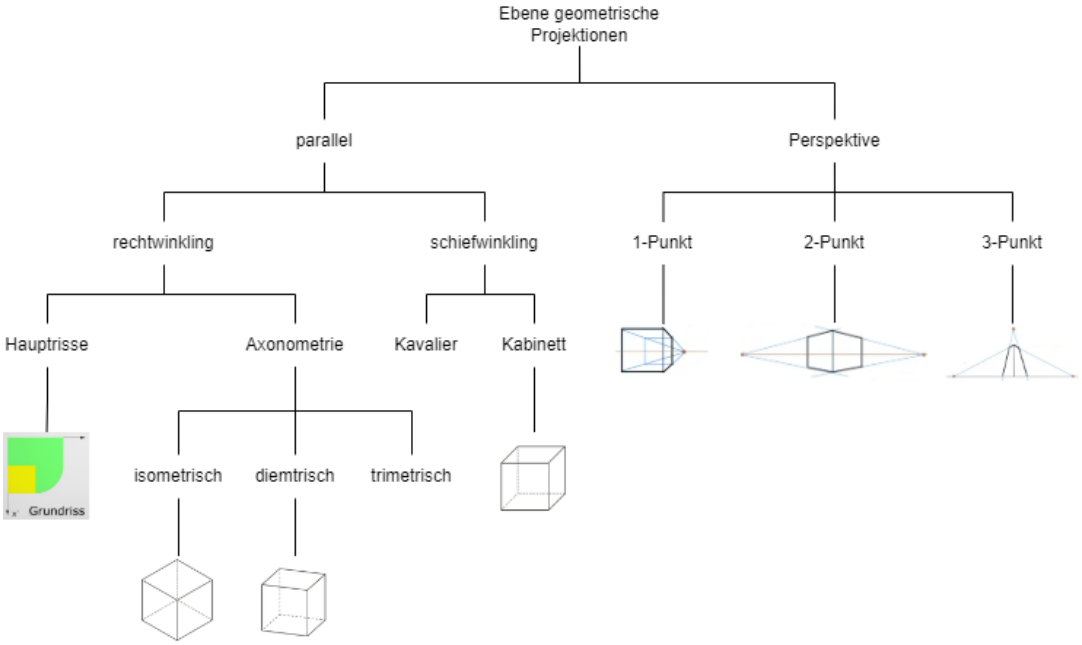
$$H := \{[x, y, z, w], w = 0\}$$

$$\begin{pmatrix} L & t^T \\ p & h \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} L(x, y, z)^T \\ p(x, y, z)^T \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} x' \\ y' \\ z' \\ 0 \end{pmatrix} \Leftrightarrow p = 0 \; A(H) = H \Leftrightarrow A = \begin{pmatrix} L & t^T \\ 0 & h \end{pmatrix}$$

Klassische Projektionen



Taxonomie



Eigenschaften

- Geraden werden auf Geraden abgebildet
- Schnitte von Geraden bleiben erhalten.
- Flächen werden auf Flächen abgebildet.
- Reihenfolge von Punkten auf projektiven Geraden bleiben erhalten.
- Winkel werden verändert.
- Parallelität geht verloren, Parallelen schneiden sich in Fluchtpunkten.
- Rechtecke werden auf Vierecke transformiert.
- Fernpunkte werden zu endlichen Fluchtpunkten.

Parallele Projektion

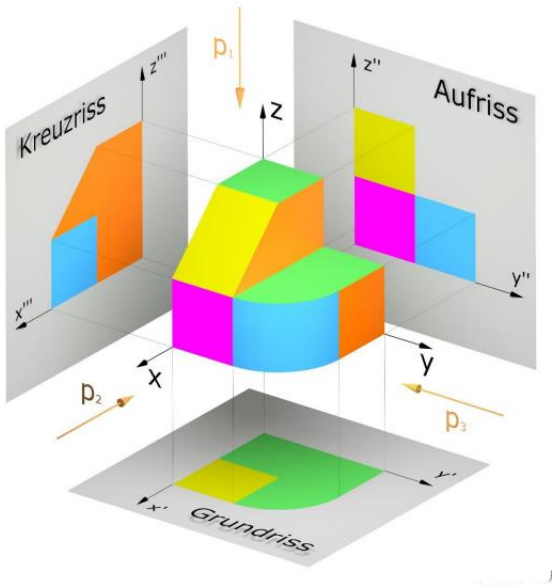
Wir nehmen an, dass wir entlang der negativen z-Achse blicken und ein rechtshändiges Koordinatensystem haben und parallel auf die Ebene $z = 0$ projizieren.

Projektion des sichtbaren Volumens (l, r, b, t, n, f) in den "Einheitswürfel" mit Extrempunkten $(-1, -1, -1)^T$ und $(1,1,1)^T$. Translation gefolgt von einer Skalierung.

$$P_0 = S \cdot T = \begin{pmatrix} \frac{2}{r-1} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -\frac{1+r}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{r-1} & 0 & 0 & -\frac{1+r}{2} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{2} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bei der parallelen Projektion wird zwischen rechtwinkliger und schiefwinkliger Projektion unterschieden. Bei der rechtwinkligen parallelen Projektion stimmt der Richtungsvektor der Projektionsstrahlen (Projektions-Richtung) mit dem Normalenvektor der Projektionsebene überein, bei der schiefwinkligen, parallelen Projektion ist dies nicht so.

Rechtwinklige, parallele Projektion



Unterscheidung in Hauptriss-Projektion und axonometrische Projektion Bei einer Hauptriss-Projektion ist die Projektionsebene orthogonal zu einer der Koordinatenachsen, bei der axonometrischen Projektion ist dies nicht so 3-Tafel-Projektion bildet.

Betrachte den Hauptriss mit Projektionsebene $z = z_0$

$$[x, y, z, w]^T \rightarrow [x, y, z_0 w, w]^T = \left[\frac{x}{w}, \frac{y}{w}, z_0, 1\right]^T$$

Dimensionsverlust/ Unmöglichkeit der Rücktransformation auf Originalpunkt zeigt sich durch verschwindende Determinante.
Oftmals ist $z_0 = 0$ oder z_0 in $[n, f]$: n, f = near (far) plane, wobei $0 > n > f$ (OpenGL: 0 < n < f, obgleich default-Kamera auch entlang der negativen Achse schaut; verwendet interne Negation).

Perspektivische Projektion

Perspektivische Projektionen werden durch projektive Abbildungen verwirklicht, affine Abbildungen reichen hierfür nicht mehr aus.
Vom Blickpunkt (Augpunkt, Beobachtungspunkt) werden weit entfernte Objekte kleiner dargestellt als nahe am Blickpunkt befindliche Objekte.
Konsequenzen: realistischer räumlicher Eindruck; daher keine Längeninvarianz.

Zunächst: perspektivische Projektion des Punkts p auf die Ebene $z = -d$. Bildpunkt q soll von der Form $q = (q_x, q_y, q_z)^T$ mit $q_z = -d$ sein. Der Strahlensatz ergibt:

$$\frac{q_x}{p_x} = -\frac{d}{p_z} \Rightarrow q_x = -d\frac{p_x}{p_z}, q_y = -d\frac{p_y}{p_z}$$

Somit ergibt sich die perspektivische Transformationsmatrix P_p:

$$P_p p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ -\frac{p_z}{d} \end{pmatrix} \Rightarrow q = \begin{pmatrix} -d \cdot \frac{p_x}{p_z} \\ -d \cdot \frac{p_y}{p_z} \\ -d \cdot \frac{p_z}{p_z} \\ 1 \end{pmatrix} = \begin{pmatrix} -d \cdot \frac{p_x}{p_z} \\ -d \cdot \frac{p_y}{p_z} \\ -d \\ 1 \end{pmatrix}$$

Eigenschaften:

- Alle Punkte auf der Geraden durch q und den Augenpunkt werden auf q abgebildet
- Für d gegen unendlich sind alle Punkte (im projektiven Sinne) unendlich fern
- Punkte der xy-Ebene $p_z = 0$ sind (für festes d) unendlich fern
- Punkte mit $p_z \leq 0$ bilden das “Sichtfeld”
- Punkte in der Projektionsebene $p_z = -d$ bleiben unverändert, diese heißen daher Fixpunkte
- Ebenen parallel zur Projektions-ebene, d.h. $p_z = -a$ bleiben solche Ebenen

Sichtbarkeitsbereich: Sichtbarkeitsbereich unter perspektivischer Projektion= Pyramidenstumpf bestehend aus 6 clipping planes: n, f, l, r, b,t = near, far, left, right, bottom und top clipping plane.

Vertigo-Effekt: "Vertigo-Effekt" aus gleichnamigem Film von Alfred Hitchcock. Kamerazoom während sich die Kamera von der Szene entfernt.

Perspektivische Projektion unter Einbeziehung der **near und far clipping planes**:

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1+\frac{f}{n} & f \\ 0 & 0 & -\frac{1}{n} & 0 \end{pmatrix}}_{R} \cdot \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x \\ y \\ \left(1+\frac{f}{n}\right) \cdot z + fw \\ -\frac{z}{n} \end{pmatrix} = \begin{pmatrix} -\frac{n}{z} \cdot x \\ -\frac{n}{z} \cdot y \\ -\frac{n}{z} fw - (f+n) \\ 1 \end{pmatrix}$$

Die z-Werte aus $[-f,-n]$ landen immer in $[-f,-n]$ allerdings nicht-linear. Tendenziell wird näher zur far clipping plane hin abgebildet.

Eigenschaften:

- Punkte auf der Ebene $z=-n$ bleiben unverändert (Fixpunkte)
- x,y-Koordinaten der far clipping plane werden um Faktor $\frac{n}{f}$ verkleinert-n-f Fluchtpunkt
- Fluchtpunkt ergibt sich aus der Projektion des unendlich weit entfernten Punktes in Blickrichtung, also aus der Projektion von $(0,0,-z,0)^T$
- $(0,0,-z,0)^T \rightarrow (0,0,-f-n,1)^T$

Nach perspektivischer Transformation (unter Einbeziehung der near und far clipping plane) ist noch eine Transformation auf das kanonische Sichtvolumen $[-1,1] \times [-1,1] \times [-1,1]$ notwendig. Transformation die dieses verwirklicht ist P_0 (siehe Parallele Projektion). Insgesamt ergibt sich:

$$P_0 R = \begin{pmatrix} \frac{2}{r-1} & 0 & 0 & \frac{r+1}{1-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{b-t} \\ 0 & 0 & \frac{2}{f-n} & \frac{f+n}{n-f} \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{n+f}{n} & f \\ 0 & 0 & -\frac{1}{n} & 0 \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-1} & 0 & \frac{r+1}{r-1} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

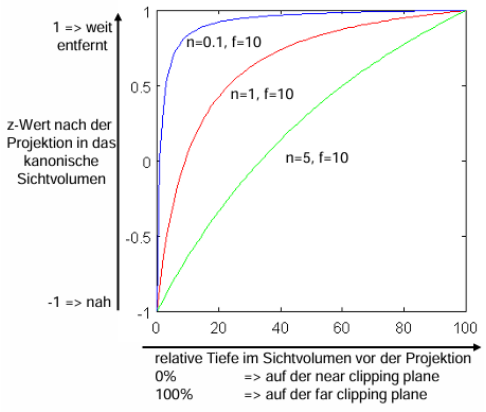
Tiefenwerte im kanonischen Sichtvolumen (KSV):

Tiefenwerte tendieren näher zur far clipping plane zu rutschen beim perspektivischen Transformieren.

Verhältnis von n zu f charakterisiert die Kurve (z.B.: n,f und 2n,2f erzeugen die gleiche Kurve)

Wird die near clipping plane nah zum Ursprung gesetzt, so werden entfernte Objekte stark gestaucht (Schlechte z-Buffer Auflösung in der Tiefe).

Dies kann zu Z-Buffer Artefakten führen bei dem mehrere Objekte überlappen und in einander angezeit werden.



Perspektivische Transformation

Standort des Betrachters liegt beliebig im Objektraum und Projektionsrichtung ist ebenfalls allgemein (OpenGL: gluLookAt()). Lege perspektivische Transformation durch Fixierung des Blickpunktes (Eye), des Blickbezugspunktes (V_{Ref}) und der Angabe einer Oben-Richtung (ViewUp) fest.

Vorgehensweise:

- Bilde rechtshändiges, orthogonales Koordinatensystem v'_x, v'_y und v'_z
- v'_z wird durch (normierte) Differenz von V_{Ref} und Eye definiert
- v'_x = (normiertes) Kreuzprodukt von ViewUp und v'_z
- ersetze ViewUp durch den Vektor v'_y = (normiertes) Kreuzprodukt von v'_z und v'_x
- Translation mit V_{Ref} in den Ursprung
- Rotiere auf das Weltkoordinatensystem: $v'_x \rightarrow x$; $v'_y \rightarrow y$; $v'_z \rightarrow z$;
- Wende auf diese Transformation P_0R an

Koordinaten des KSV werden noch homogenisiert, das heißt durch den inversen Streckungsfaktor w dividiert.

Viewport Transformation

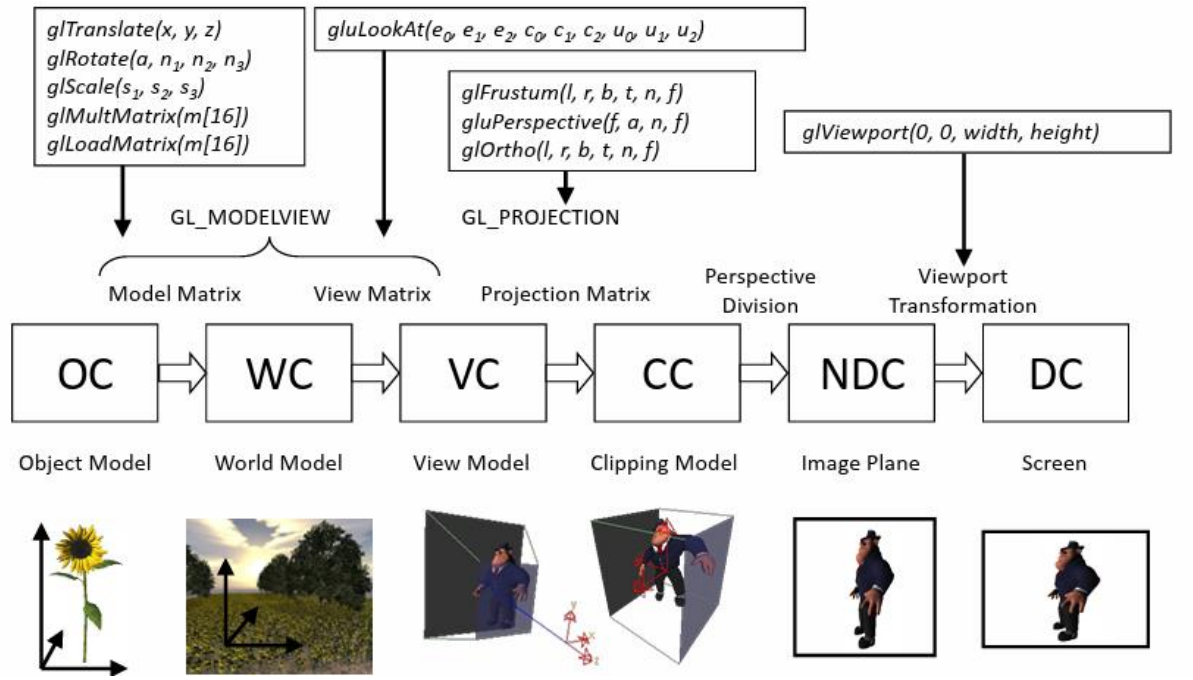
Bildschirm verwendet Pixelkoordinaten; Viewport (auch: Bildschirmausschnitt) ist relativ zu Window-Koordinaten (auch: Bildschirm-Koordinaten) festgelegt.

NDC müssen noch auf festgelegten Viewport transformiert werden.
Viewport-Transformation rechnet NDC in Window-Koordinaten um.

Ist $(x_{offset}|y_{offset})$ der linke, untere Eckpunkt des Viewports, so lautet die Umrechnungsformel: $x' = x_{offset} + \left(\frac{width}{2}\right) + \left(\frac{width}{2}\right) \cdot x$, y analog.

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{width}{2} & 0 & 0 & x_{offset} + \frac{width}{2} \\ 0 & \frac{height}{2} & 0 & y_{offset} + \frac{height}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Geometrieverarbeitung in OpenGL



Culling

Culling kann entweder von der Anwendung, der Geometrieverarbeitung oder der Rasterisierung durchgeführt werden.

Backface

Beim Backface-Culling werden die abgewandten Primitiven im kanonischen Sichtvolumen entfernt. Hierbei sind die Sehstrahlen parallel und die Betrachtungsrichtung definiert alle sichtbaren Primitiven.

Ein grafisches Primitiv G liegt hinsichtlich der Betrachtungsrichtung (auch Sehstrahl) r auf der Rückseite eines 3D-Objekts, falls die äußere Normale n von G vom Betrachter wegweist.

Rückseiten-Erkennung: Winkel zwischen r und n liegt in zwischen -90 und 90 Grad (Skalarprodukt $n \cdot r$ aus Sehstrahl und Normale ist positiv). Ist $n \cdot r = 0$, so degeneriert das Primitiv zu Liniensegment(en).

Besonders einfach wenn Betrachter entlang der negativen z-Achse $r = (0,0,-1)^T$ schaut, da dann die Rückseiten anhand einer negativen z-Komponente erkennbar sind ($n \cdot r = \left(n_x, n_y, n_z\right) \cdot (0,0,-1)^T = -nz$). **Normalen müssen konsistent orientiert sein!**

Front und Backface-Culling wird zum Beispiel für Comic Rendering verwendet, die Frontfaces werden gefüllt gezeichnet und die Backfaces als Wireframe.

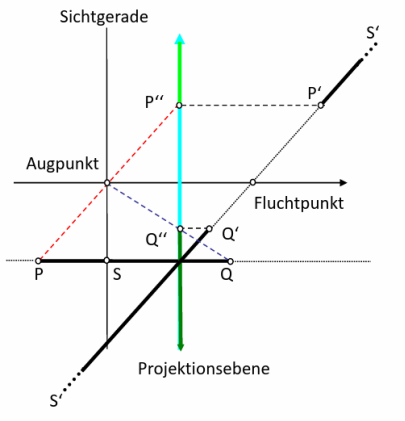
Clipping

Clipping beschreibt das Entfernen (von Teilen) von Primitiven, die außerhalb des darstellbaren/ sichtbaren Bereichs liegen.

Clipping wird nach Projektion in KSV angewandt (konsistent, da immer gegen Einheitswürfel geclippt wird) und wird somit vor Projektion auf 2D-Koordinaten angewandt. Clipping kann auch mit mehr als 6 Clipping Ebenen anwendungs-spezifisch verallgemeinert werden, dies kann Zeit sparen und Artefakte vermeiden.

Viewport-Transformation bildet KSV auf Bildschirmkoordinaten ab, oft ist der Bildschirm in mehrere Viewports aufgeteilt. Ohne Clipping kann es zu ungewünschten Überschneidungen als unerwünschte Artefakte kommen, wie die Wrap-around Problematik oder Primitive die im falschen Window sichtbar sind.

Wrap-Around Problematik:



Punkte in der Sichtebene bilden auf unendlich ferne Punkte ab (z.B. S). Bild von PQ kann dann nicht Apriori zugeordnet werden. P landet nach perspektivischer Transformation auf P‘ und nach Projektion auf P‘‘, analog Q auf Q‘ und dann auf Q‘‘ Die einfache Zuordnung PQ → P’’Q’’ stimmt hier nicht! Der unendlich ferne Punkt S’ hat kein eindeutiges Vorzeichen mehr. So könnten zwei weitere Segmente die richtige Lösung sein.

Problem tritt auf weil durch die Verwendung homogener Koordinaten (x,y,z,w)T die vierte Koordinate w kleiner als 0 sein kann Punkte auf der Sichtgeraden werden bei der perspektivischen Transformation auf unendlich ferne Punkte abgebildet (w=0).

Lösung w-Clip: Teile Liniensegmente durch Sichtebene in zwei Segmente auf, clippe zunächst an den Ebenen $w = \varepsilon$ und $w = -\varepsilon$ ▪ Beide Liniensegmente ($PQ(-\varepsilon)$ und $PQ(\varepsilon)$) werden zu Liniensegmenten mit Endpunkten, die nicht unendlich fern liegen und sind damit unterscheidbar.

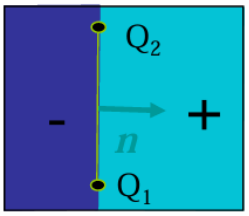
Clipping Grundidee

Standard/Realität: 3D-Clipping von grafischen Primitiven (Liniensegmente, Dreieck, Viereck, Polygon) am KSV.

Allgemeiner behandelbar: n-dimensionales Clipping von einfachen Objekten (Hyperebenen, n Simplexe) an (konvexen & konkaven) Körpern:

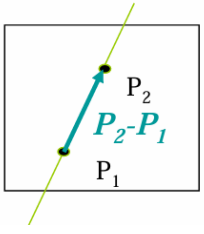
- Es gibt maximal einen sichtbaren Teil jedes Liniensegments
- Liniensegmente, deren Endpunkte beide oberhalb, unterhalb, rechts oder links liegen, sind völlig unsichtbar
- Liniensegmente deren Endpunkte beide innerhalb des Fensters liegen sind komplett sichtbar
- Verbleibende Liniensegmente erfordern die Berechnung von Schnittpunkten

2D-Clipping (Liang-Barsky-Algorithmus)



Stelle alle Fenster-Kanten als Geraden E in impliziter Form $E(P) = 0$ dar, wobei E wie folgt bestimmt wird:

$$Q_1 = (x_1, y_1), Q_2 = (x_2, y_2)$$
$$n = (\Delta y, -\Delta x) = \big(y_2 - y_1, -(x_2 - x_1)\big)$$
$$P = (x, y)$$
$$E(P) = n \cdot (P - Q_1) = n \cdot P - n \cdot Q_1$$



Konvention: Normale n zeigt ins Innere des Fensters, also $E(P) < 0$ genau dann, wenn P außerhalb des Fensters. Liniensegment durch zwei verschiedene Punkte P_1 und P_2 wird parametrisch dargestellt: $I(t) = P_1 + t \cdot (P_2 - P_1)$. Das Einsetzen von I in E ergibt den Schnitt.

Im Wesentlichen gibt es die folgenden drei Fälle:

1. P_1 und P_2 liegen außen: $E(P_1) < 0, E(P_2) < 0$
2. P_1 und P_2 liegen innen: $E(P_1) \geq 0, E(P_2) \geq 0$
3. P_1 und P_2 liegen auf verschiedenen Seiten: $E(P_1) \cdot E(P_2) < 0$

- Sonderfall $E(P_1) = 0$ oder $E(P_2) = 0$ wird normalerweise extra betrachtet.

Im Fall 3 muss der Schnittpunkt P berechnet werden. Einsetzen der parametrischen in die implizite Gleichung liefert:

$$
\begin{aligned}
& E(P_1 + t \cdot (P_2 - P_1) = 0 \\
\Leftrightarrow\ & \big(P_1 + t \cdot (P_2 - P_1)\big) \cdot n - Q_1 \cdot n = 0 \\
\Leftrightarrow\ & t = \frac{Q_1 \cdot n - P_1 \cdot n}{(P_2 - P_1) \cdot n} \\
\Rightarrow\ & P = P_1 + \frac{Q_1 \cdot n \dot{-} P_1 \cdot n}{(P_2 - P_1) \cdot n}(P_2 - P_1)
\end{aligned}
$$

Cohen-Sutherland-Algorithmus (CSA)

1001	1000	1010
0001	0000	0010
0101	0100	0110

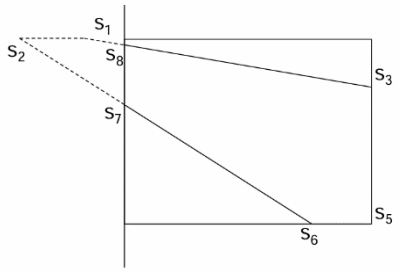
Idee: Endpunkte P eines Liniensegments erhalten einen 4-Bit Code (“Outcode”), abhängig von der Lage von P. Stellen des 4-Bit-Codes identifizieren die 4 benötigten Halbräume (oberer, unterer, rechter und linker) eindeutig. Punkte im Fenster werden mit 0000 codiert. Steht erstes Bit für oberen Halbraum, so bedeutet 1xxx ”im oberen Halbraum”. Im Beispiel identifizieren die Stellen der Codes der Reihe nach die Halbräume Oben|Unten|Rechts|Links.

Liniensegment liegt völlig innerhalb des Fensters, wenn der Code für beide Endpunkte 0000 ist (logisches ODER ist 0000) oder völlig außerhalb des Fensters, wenn der Code sich an einer Stelle nicht unterscheidet (logisches UND ist ungleich 0000. Verbleibender Fall: das logische ODER der Codes ist ungleich 0000 und das logische UND ist gleich 0000.

Schnittpunkte werden nach Liang & Barsky berechnet. Der Schnittpunkte zerlegen das Liniensegment in neue (kürzere) Liniensegmente, dabei werden Schnittpunkte dupliziert und die Duplikate liegen auf “verschiedenen” Seiten der Schnittgeraden und erhalten daher verschiedene Outcodes. Jedes so entstehende (kürzere) Liniensegment wird potenziell analog behandelt.

Anzahl der Stellen des Codes entspricht der Anzahl der Halbräume und ist unabhängig von der (Raum)Dimension.

Sutherland-Hodgman-Algorithmus (SHA)

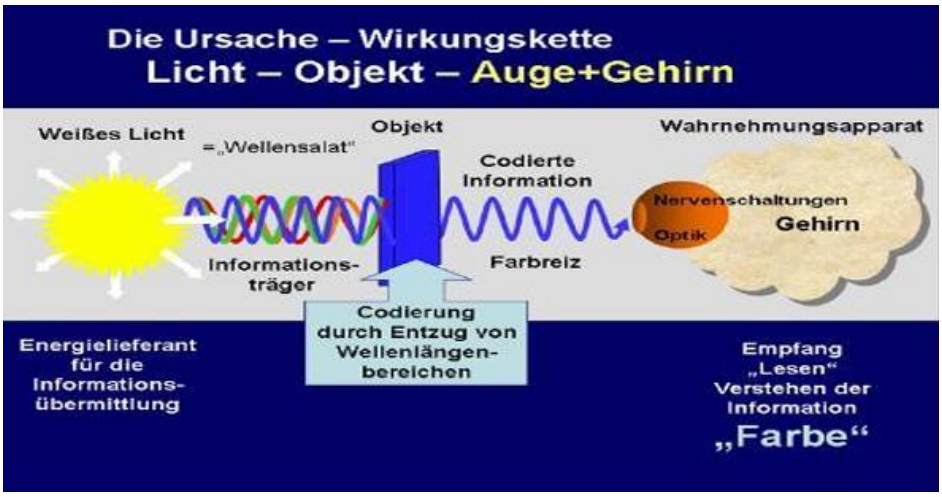


Clipping eines geschlossenen Polygons muss ein (eventuell neues) geschlossenes Polygon liefern. Hier ist die einfache Variante des CSA nicht anwendbar. Das Hauptproblem ist, sind Polygone in den Ecken des Fensters und “große” Polygone. Das ,,geclippte'' Polygon besteht aus Teilen des Fensters. Eine einfache Lösung ist der Sutherland-Hodgman-Algorithmus, bei dem das gesamte Polygon sukzessive an den Fenstergrenzen geclippt wird. Im Bild rechts als Beispiel nur die linke Seite des Fensters.

Licht

Beim Rendern von 3D-Modellen hat man im Wesentlichen zwei Komponenten, die Geometrie und die Erscheinung (also die Art der Interaktion mit dem Licht). Häufig ist das Ziel, der Photorealismus, man versucht also die Realität zu simulieren aber muss vereinfachen, um effizientere Algorithmen zu verwenden.

Licht-Wirkungskette:



Visuelle Phänomene:

- Licht wird von Lichtquelle ausgestrahlt
- Licht interagiert mit der Szene, Absorption, Streuung
- Licht wird von einem Sensor absorbiert.

Simulation der Beleuchtung:

Realistische, 3-dimensional erscheinende Darstellungen in der GDV benötigen geeignete Beleuchtungsmodelle, diese ergeben sich aus Farbmodell, Modelle für Lichtausbreitung, Modelle für Lichtreflexion (Interaktion mit Objekten) und Eigenschaften der Lichtquellen.

Man unterscheidet zwischen lokalen und globalen Beleuchtungsmodellen. Lokale Beleuchtungsmodelle haben keine Wechselwirkung durch Mehrfachreflexion, daher nur direktes Licht, keine Verdeckung (Schatten), keine Spiegelung, nur einfache Lichteffekte aber schnell zu berechnen.

Licht ist ein eine sich sehr schnell ausbreitende Strahlung. Licht ist eine wellenförmig ausbreitende elektromagnetische Schwingung.

Monochromatisches (einfarbiges) Licht ist festgelegt durch eine Frequenz f bzw. eine Wellenlänge λ, wobei $f \cdot \lambda = c$.

Spektrum des sichtbaren Lichts

Das sichtbare Licht ist nur ein kleiner Ausschnitt des elektromagnetischen Spektrums (380nm – 720nm Wellenlänge). Farbe ist ein Sinneseindruck.

Der Mensch ist ein Trichromat (hat drei Arten von Zapfen). Die Zapfen reagieren auf unterschiedliche Wellenlängen verschieden stark. Weißes Licht ist nicht monochromatisch, es wird durch die Mischung erzeugt. Zwei Farbreize sind gleich, falls die Erregungszustände der Farbzapfen

gleich sind. Gleiche Farbreize können durch Licht mit unterschiedlichen Wellenlängenanteilen erzeugt werden.

Tristimulus-System: Additive Farbmischung

Das erste Graßmannsche Gesetz besagt, dass je vier Farben stets in einer eindeutigen linearen Beziehung zueinanderstehen und zur Beschreibung einer beliebigen Farbe werden drei voneinander unabhängige Primärfarben benötigt.

Farben werden wie Elemente eines 3-diemensionalen Vektorraumes (Farbraum) behandelt.

- „Vektoren“ = Farben heißen Farbvalenzen
- „Richtung“ eines Vektors heißt Farbart
- „Länge“ eines Vektors heißt Farbwert: bestimmt Leuchtdichte, d.h. Helligkeit der Farbe
- 3 „linear unabhängige“ Vektoren heißen Primärvalenzen
- Linear unabhängig: jede der 3 Primärvalenzen ist nicht durch Mischung der beiden anderen Primärvalenzen darstellbar

Farbmodell

Sei F eine beliebige Farbvalenz und R,G.B Primärvalenzen, dann existieren r,g,b (reelle Zahlen), so dass $F = rR + gG + bB$.

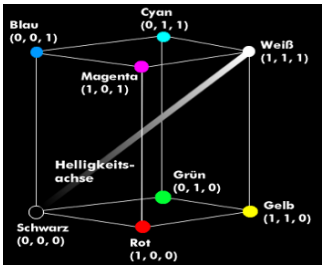
Durch große Anzahl von Mischexperimenten wurde Vektorraum-Eigenschaften des Farbraums bestätigt:

$$F_i = r_i R + g_i G + b_i B$$
$$\alpha F_1 + \beta F_2 = (\alpha r_1 + \beta r_2)R + (\alpha g_1 + \beta g_2)G + (\alpha b_1 + \beta b_2)B$$

Farbvalenzen erlauben Rechnen wie mit Vektoren. Insbesondere: Umrechnung der Darstellung bezüglich verschiedener Primärvalenztripel (Basiswechsel) möglich.

Das RGB-Modell ist das Standardmodell additiver Farbmischung und der Farbdarstellung am Display

Einheitswürfel-Modell:



- Ecken sind gesättigte Primärfarben Rot, Grün & Blau und gesättigte Sekundärfarben Cyan, Magenta & Gelb
- Summenfarbe: Weiß (1,1,1)
- Hintergrundfarbe: Schwarz (0,0,0)
- Achromatische Farben (Gleiche Anteile von R, G und B) liegen auf der Hauptdiagonalen

Farbe ergibt sich durch Spezifizierung der Anteile von R, G und B, welche zu Schwarz hinzuaddiert werden.

8-Bit-Darstellung, also mit 256 Sättigungswerten für jede der Primärfarben ergibt etwa 16.7 Millionen unterschiedliche Farben.

Lichtausbreitung

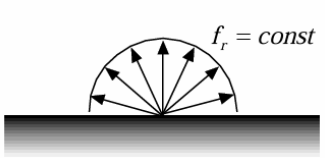
Physikalische Betrachtungsmöglichkeiten:

- Quantenoptik (atomare Betrachtung)
- Wellenoptik (mikroskopische Betrachtung)
- Geometrische Optik (makroskopische Betrachtung)

In der GDV wird Licht als Lichtstrahl betrachtet (geometrische Optik). So wird das Licht vereinfacht. Man kann dem Licht entlang der Strahlen folgen. Man kann Wellen-Phänomene auf mikroskopischer Ebene ignorieren. Die Intensität des Lichtes nimmt nicht mit der Entfernung ab. Ignorieren der mikroskopischen Wechselwirkung zwischen Licht und Materie. Anstatt dem kontinuierlichen Spektrum werden nur n (typischerweise n=3) diskrete Wellenlängen als monochromatisches Licht verwendet, die Lichtquellen senden lediglich Anteile von R, G, B.

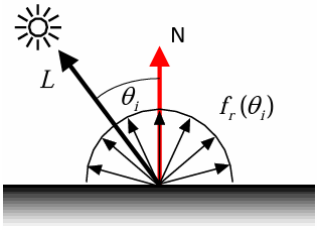
Reflexionsmodell

Ambiente Reflexion



Ambiente Beleuchtung ist unabhängig von Einfallswinkel und Lichtposition, ein Minimum an Licht scheint immer vorhanden, das Licht wirkt flach.

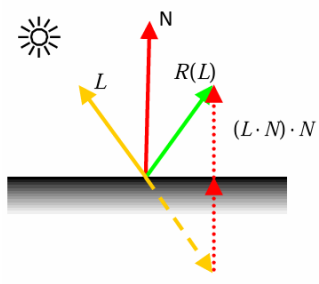
Ideal diffuse Reflexion



Hierfür verwendet man das Lambert'sche Beleuchtungsmodell. Die Reflektion ist abhängig vom Winkel, in welchem das Licht auf die Oberfläche trifft, aber unabhängig von der Richtung zum Betrachter. Flächen deren Normale zur Lichtquelle zeigen erscheinen heller.

$f_r(\theta_i) = \cos(\theta_i) \cdot E = (L \cdot N) \cdot E$, L, N müssen normiert sein.

Ideal spiegelnde Reflexion



Reflexion ist spiegelnd, das heißt der reflektierende Strahl geht nur in Richtung $R(L)$.

$$R(L) = -L + 2 \cdot (L \cdot N) \cdot N$$

Reflexionsgesetz:

- Lichtstrahl, Flächennormale und reflektierter Strahl liegen in einer gemeinsamen Ebene
- Winkel zwischen L und N stimmt mit Winkel zwischen N und R(L) überein

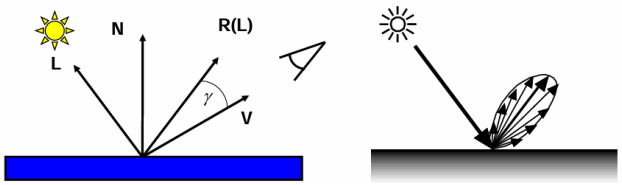
Spekulare Reflexion

Die spekulare Reflektion ist glänzend, aber nicht spiegelnd. Typischerweise reflektierend in Vorzugsrichtung r. Der Winkel zwischen idealer Reflexionsrichtung und Betrachter Position ist wichtig. Beim der Phong Reflexion wird mit dem Term $\cos^s()$ modelliert, wobei s den Shininess-Faktor angibt. Ist s groß, so ist die Fläche fast ideal spiegelnd und die Glanzlcihter (specular highlights) sind punktförmig. Ist s klein, erscheint die Reflexion matt und die Glanzlichter sind großflächig.

Spiegelnde Reflexion (Phong)

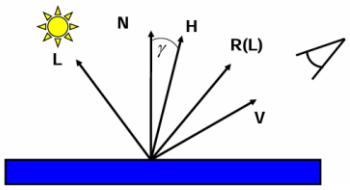
Skalarprodukt zwischen reflektierter Lichtrichtung und Richtung zum Betrachter hoch Exponent (k_e).

$$f_r(\gamma) = \cos^{k_e}(\gamma) \cdot E$$
$$f_r(\gamma) = (R(L) \cdot V)^{k_e} \cdot E$$



Beim Blinn-Phong Reflexionsmodell verwendet man den Halbierungsvektor H zwischen der einfallenden Lichtrichtung und der Blickrichtung, so reduziert man den Berechnungsaufwand.

$$f_r(\gamma) = (H \cdot N)^{k_e} \cdot E, H = \frac{L+V}{\|L+V\|}$$



Kombination von Beleuchtungsmodellen

Addition der Anteile aus ambienter, diffuser und spiegelnder Reflexion.

$$k_a E + k_d E(L \cdot N) + k_s E(H \cdot N)^{k_e} = I$$

- E: einfallende Intensität
- I: reflektierte Intensität
- k_a, k_d, k_s: Tupel für jeweilige Anteile in R, G, B
- k_e: Spiegelungsexponent
- L: Richtung Lichtquelle
- N: Oberflächennormale
- H: Halfway-Vektor

BRDF bis 12D-Modell

Bidirectional Reflectance Distribution Function (BRDF, 4D) beschreibt das Verhältnis von reflektierter Radiance (I) zu einfallender Radiance (E) in Abhängigkeit von Einfallsrichtung und Betrachtungsrichtung. Es gilt für homogenes Material, bei dem die Reflexionseigenschaften unabhängig von der Position auf der Oberfläche sind. Licht verlässt die Oberfläche am gleichen Punkt, an dem es eingetroffen ist, ohne Änderung der Wellenlänge oder Zeitabhängigkeiten wie Phosphoreszenz.

$$f_r(\overrightarrow{\omega_i}, \overrightarrow{\omega_o}) = \frac{dI(\overrightarrow{\omega_o})}{dE(\overrightarrow{\omega_i})}$$

Eine **isotrope BRDF (3D)** ist invariant gegenüber einer Rotation um die Normale der Oberfläche. Dies bedeutet, dass das Material isotrop ist, also in alle Richtungen gleich reflektiert.

$$f_r(\theta_i, \theta_o, \varphi_o)$$

Eine **räumlich variierende BRDF (6D)** berücksichtigt, dass die Reflexionseigenschaften eines Materials über die Oberfläche variieren können. Dies ermöglicht eine detailliertere und realistischere Darstellung von Materialien mit unterschiedlichen Reflexionseigenschaften an verschiedenen Punkten.

$$f_r(\overrightarrow{x_i}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o})$$

Bidirectional Scattering Surface Reflectance Distribution Function (BSSRDF, 8D) berücksichtigt nicht nur Reflexion, sondern auch das Eindringen von Licht in eine Oberfläche und das anschließende Verlassen an einem anderen Punkt (Subsurface Scattering). Dies ist besonders wichtig für die Darstellung von Materialien wie Haut, Marmor oder Milch, bei denen Licht unter der Oberfläche gestreut wird.

$$f_r(\overrightarrow{x_i}, \overrightarrow{\omega_i}, \overrightarrow{x_o}, \overrightarrow{\omega_o})$$

Eine **Scattering Function (9D)** erweitert die Betrachtung um die Abhängigkeit von der Wellenlänge des Lichts und berücksichtigt Phänomene wie Fluoreszenz, bei denen die Wellenlänge des Lichts nach der Interaktion mit dem Material ändern kann.

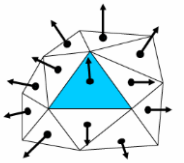
$$f_r(\overrightarrow{x_i}, \overrightarrow{\omega_i}, \lambda_i, \overrightarrow{x_o}, \overrightarrow{\omega_o})$$

Das **allgemeine Reflexionsmodell (12D)** fügt eine zeitliche Dimension hinzu, um Phänomene wie Phosphoreszenz zu berücksichtigen, bei denen die Lichtreflexion zeitabhängig ist.

$$f_r(\overrightarrow{x_i}, \overrightarrow{\omega_i}, t_i, \lambda_i, \overrightarrow{x_o}, \overrightarrow{\omega_o}, t_o, \lambda_o)$$

Flat Shading

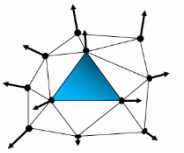
Beim Flat Shading wird die Reflexion pro Dreieck berechnet mit Hilfe dessen Normale. Dies erzeugt einen Farbwert je Dreieck.



$$I_\Delta = f(\overrightarrow{\omega_o}, \overrightarrow{n_\Delta}, \overrightarrow{\omega_i})E$$

Gouraud Shading

Beim Gouraud Shading wird die Reflexion pro Eckpunkt berechnet mit Hilfe dessen Normale. Dies erzeugt ein Farbwert je Eckpunkt, welcher linear über das Dreieck mittels baryzentrischer Koordinaten interpoliert wird.



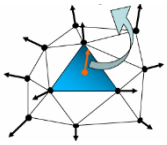
$$I_v = f(\overrightarrow{\omega_o}, \overrightarrow{n_v}, \overrightarrow{\omega_i})E$$
$$I_p = \lambda_1 I_{v1} + \lambda_2 I_{v2} + \lambda_3 I_{v3}$$

Probleme von Gouraud Shading:

- Highlights (spiegelnde Reflexion) variieren stark in dynamischen Szenen, je nachdem wie die Farbwertberechnung in den Eckpunkten ausfällt
- Glanzpunkt mitten auf Eckpunkt: starkes Highlight.
- Glanzpunkt in der Mitte eines Dreiecks: Eckpunkte haben womöglich keine Glanzpunktausprägung. Das Highlight verschwindet.
- Mach'sche Streifen, bedingt durch laterale Inhibition (Hemmung). Rezeptoren hemmen benachbarte Rezeptoren
    - Effekt: Kontrastverstärkung, unstetige Übergänge fallen auf, gilt auch für Sprünge in der ersten Ableitung!

Phong Shading

Beim Phong Shading wird die Reflexion pro Fragment mit interpolierter Normale berechnet. Die Eckpunkte werden über das Dreieck interpoliert.



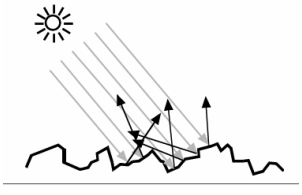
$$\overrightarrow{n_p} = \frac{\lambda_1 \overrightarrow{n_a} + \lambda_2 \overrightarrow{n_b} + \lambda_3 \overrightarrow{n_c}}{\|\lambda_1 \overrightarrow{n_a} + \lambda_2 \overrightarrow{n_b} + \lambda_3 \overrightarrow{n_c}\|}$$
$$I_p = f(\overrightarrow{\omega_o}, \overrightarrow{n_p}, \overrightarrow{\omega_i})E$$

Beleuchtungsmodelle

Bei Beleuchtung mit mehreren Lichtquellen addiert man die einzelnen Lichtquellen und beschränkt sie auf ein gültiges Intervall.

Cook-Torrance Modell

Oberfläche wird als Menge kleiner Facetten (Spiegel) interpretiert mit spiegelnder Reflexion an dieser Oberfläche. Mit jeweils einem perfekten Spiegel. Die Eigenschaften sind ableitbar von der physikalischen Theorie der Reflexion. Die Verteilung der Normalen bestimmt die Breite des Glanzlichts. Modellierung der Rauheit durch Beckmannsche Wahrscheinlichkeitsdichtefunktion (Microfacettenverteilung).



Globale Beleuchtungsmodelle

Raytracing, Radiosity, Photon Mapping, Path Tracing und Precomputed Radiance Transfer und Image Based Rendering, werden z.T. später noch behandelt.

Bestimmung von BRDF-Parametern

Die BRDF-Parameter werden typischerweise mit einem Gerät namens Gonioreflektometer bestimmt. Dieses Gerät ermöglicht es, die Richtungsabhängigkeit der Reflexion eines Materials zu messen, indem es systematisch den Winkel zwischen der einfallenden Lichtquelle und dem Sensor variiert, während es die reflektierte Intensität aufzeichnet. Das Material, dessen Reflexionseigenschaften gemessen werden, wird in die Messanordnung eingebracht, und das Gonioreflektometer misst, wie das Material Licht unter verschiedenen Einfall- und Beobachtungswinkeln reflektiert.

Raytracing

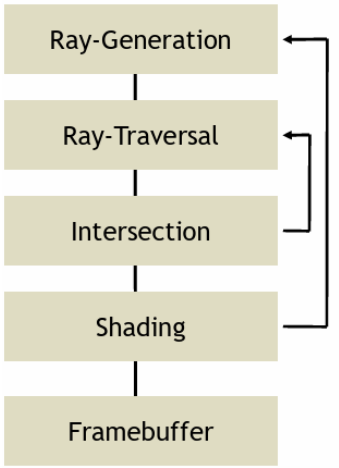
1. Erzeugung von Primärstrahlen: Strahlen werden von der Kameraposition aus in die Szene geschossen, mindestens ein Strahl pro Pixel
2. Ray Tracing: Schnitt der Strahl mit der Geometrie der Szene, in der Regel Verwendung von Beschleunigungsstrukturen
3. Shading: Auswertung der Farbe am ersten Auftreffpunkt des Primärstrahls in der Szene, benötigt Informationen über die Reflektionseigenschaften und die einfallende Beleuchtung

Ein Strahl ist festgelegt durch einen Startpunkt und eine Richtung (und evtl. maximale Länge). Der Strahl ist nur eine Richtung, der Richtungsvektor ist oft normalisiert.

$$\vec{r}(t) = \vec{o} + t \cdot \vec{d}, t > 0, \left\|\vec{d}\right\| = 1$$

Schnitt mit Primitiven (Kugel, Ebene, Dreieck) durch Einsetzen der Strahlgleichung in die entsprechende Beschreibung des Primitivs, aber numerische Probleme beim exakten Schnitt (z.B. mit Objektoberfläche).

Raytracing-Pipeline



1. **Ray-Generation:** In dieser Phase werden Strahlen von der Kameraposition aus in die Szene geschossen. Für jeden Pixel auf dem Bildschirm wird mindestens ein Primärstrahl generiert, der durch die Szene verfolgt wird, um zu bestimmen, welche Objekte der Strahl schneidet.
2. **Ray-Traversal:** Nachdem die Strahlen generiert wurden, müssen sie durch die Szene traversiert werden, um Interaktionen mit Objekten zu identifizieren. Dieser Prozess wird oft mit Beschleunigungsstrukturen wie BVH (Bounding Volume Hierarchies), kd-Bäumen oder Gittern optimiert, um die Effizienz der Schnittpunktberechnungen zu erhöhen.
3. **Intersection:** In dieser Phase wird der Schnittpunkt des Strahls mit den Objekten in der Szene berechnet. Es wird bestimmt, an welcher Stelle der Strahl ein Objekt trifft, und diese Informationen werden für die spätere Schattierungsberechnung gespeichert.
4. **Shading:** Nachdem die Schnittpunkte bestimmt wurden, wird die Farbe und Helligkeit der Schnittpunkte aufgrund der Materialien der Objekte, der Lichtquellen in der Szene und der Schattierungsalgorithmen wie Phong, Gouraud oder physikalisch basierten Rendering-Techniken berechnet.

5. **Framebuffer:** Schließlich werden die berechneten Farben und Helligkeitswerte der Schnittpunkte im Framebuffer gespeichert, der das finale Bild repräsentiert. Dieses Bild wird dann auf dem Bildschirm angezeigt.

Raytracing-Varianten

Nur Primärstrahlen: Nur Verdeckungsberechnung, keine Schatten, Reflexion oder Beleuchtung

$$I_p = I_a k_a$$

Primärstrahlen mit einem Schattenstrahl: Verdeckungsberechnung, Harte Schatten, keine Reflexion und nur direkte Beleuchtung.

$$I_p = I_a k_a + \sum_{i=1}^{n} S_{i,p} I_i (k_d f_d + k_s f_s)$$
$$f_d = L_i \cdot N_p$$
$$f_s = (H_i \cdot N_p)^{k_s}$$

Recursive Ray Tracing: Von der Kamera ausgesandte Primärstrahlen erzeugen an den Schnittpunkten mit Geometrie rekursiv Sekundärstrahlen. Berechnung von Sichtbarkeit (Schlagschatten), Spiegelungen, Lichtbrechung, …

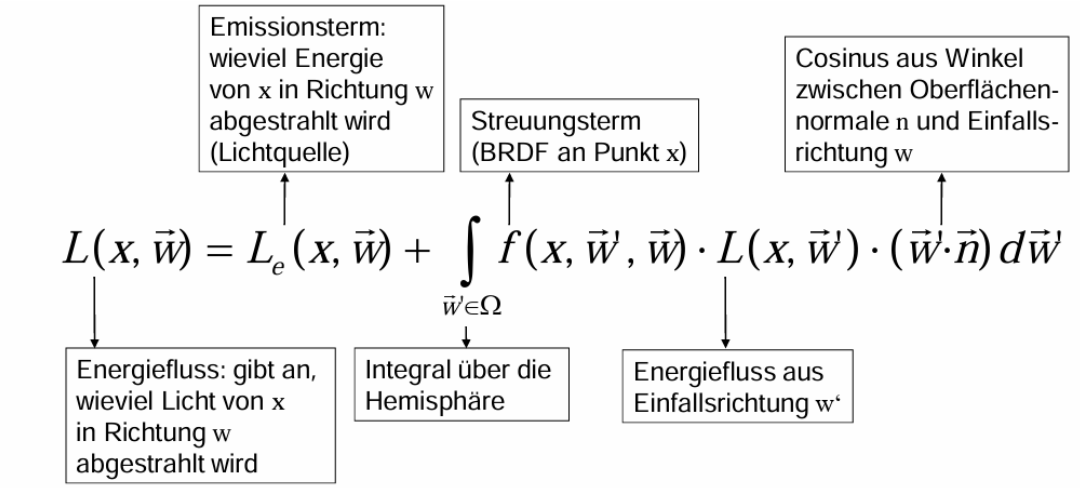
$$I_p = I_a k_a + \sum_{i=1}^{n} S_{i,p} I_i (k_d f_d + k_s f_s) + k_r I_r + k_t I_t$$

Distributed Ray Tracing: Mehrere Sekundärstrahlen mit anschließender Mittelung der Farbwerte. Stochastische Modellierung verschiedenster Effekte. Vermeidung von Rauschen durch Verwendung sehr vieler Strahlen. Weiche Schatten, Spiegelnde und glänzende Reflexionen mit BRDF. Direkte Beleuchtung und über spiegelnde und glänzende Reflexion.

Path Tracing: Verdeckungsberechnung mit weichen Schatten, vollständiger Reflexion durch BRDF, vollständiger globaler Beleuchtung und inklusive Kaustiken. Basiert auf der Rendergleichung.

Rendergleichung

Mathematische Basis für globale Beleuchtungsmethoden



Raytracing Aufwand

Ray Tracing ermöglicht eine realitätsnahe Simulation der Beleuchtung durch ein globales Beleuchtungsmodell. Es kann riesige Modelle handhaben, mit logarithmischer Skalierung bezüglich

der Szenengröße, und ist hochgradig skalierbar, mit linearer Skalierung hinsichtlich der Anzahl der Prozessoren. Ray Tracing unterstützt die Verwendung realistischer Materialien und ermöglicht Effekte wie Motion Blur.

Der **naive Ansatz des Ray Tracing** hat eine lineare Komplexität. Für ein Bild mit 1024x768 Pixeln und 6320 Dreiecken in der Szene ergeben sich beispielsweise 4.970.250.240 Tests auf Strahl-Dreiecksschnitt. Unstrukturierte Daten erfordern mindestens linearen Aufwand, da jedes Primitiv potenziell das erste sein könnte, das von einem Strahl geschnitten wird, und somit alle Primitive einzeln getestet werden müssen. Die **Komplexität** lässt sich durch räumliche Sortierung reduzieren, was den Einsatz effizienter Suchstrategien mit $O(\log n)$ Komplexität ermöglicht, allerdings mit einem Trade-off zwischen dem Sortieraufwand und dem Aufwand zur Laufzeit, besonders in dynamischen Szenen.

Beschleunigungsmöglichkeiten

- Weniger Schnittberechnungen: Der Einsatz räumlicher Datenstrukturen wie Bounding Volume Hierarchien (BVH), (reguläre) Gitter, hierarchische Gitter, Octrees, Binary Space Partitions (BSP) und kd-Bäume kann die Anzahl der notwendigen Schnittberechnungen signifikant reduzieren.
- Tracing von zusammenhängenden Strahlbündeln: Diese Methode nutzt die Kohärenz zwischen benachbarten Strahlen aus und kann Techniken wie Cone Tracing oder Beam Tracing umfassen.

Simulation des Lichttransports entweder von der Kamera aus oder von den Lichtquellen aus.

Vorwärts-Lichttransport (Photon Tracing / Photon Mapping): Photonen werden von der Lichtquelle aus verschossen, Reflexion, Refraktion, Absorption an Oberflächen und Photonen, die die Kamera treffen, werden aufgezeichnet. -> Problem: die meisten ausgesendeten Photonen treffen nicht die Kamera.

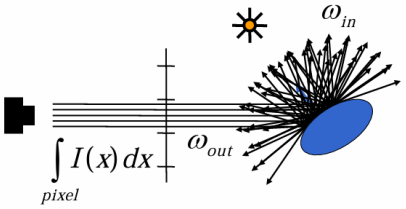
Rückwärts-Lichttransport (Ray Tracing / Path Tracing): Startet an der Kamera, suche von Pfaden, auf denen das Licht die Kamera erreichen kann.

Photon Mapping

Physikalische korrekte Simulation der Beleuchtung, insbesondere korrekte Kaustiken. Die Photonen werden in einer Datenstruktur gespeichert (Photon Map). Die Szene wird dann anschließend mit Rückwärts-Lichttransport gerendert. Die Photon Map wird dann zur Farb- und Helligkeitswertbestimmung verwendet.

Anti-Aliasing

Anti-Aliasing ist eine Technik, die in der Computergrafik verwendet wird, um das Problem des "Aliasing" zu mindern. Aliasing tritt auf, wenn hohe Frequenzen in einem Bild (z.B. scharfe Kanten) durch die begrenzte Auflösung eines Rasters (Pixelgitter) nicht korrekt wiedergegeben werden können und unerwünschte Effekte wie Treppenbildung oder Moiré-Muster erzeugen. Approximation von Gloss und Translucency durch perfekte Spiegel führt zu einer Art Aliasing, da die Reflexion und Refraktion nicht ausreichend abgetastet werden.



Vollständiges Anti-Aliasing erfordert die Auswertung sehr vieler, geschachtelter Integrale und ist daher sehr aufwendig.

$\int_{\Omega} I(\omega_{in}) f_r(\omega_{in}, \omega_{out}) d\omega_{in}$

Approximation der Integrale

Eine Funktion $f(x)$ ist nicht analytisch bekannt, aber auswertbar, das Integral kann folgendermaßen approximiert werden:

$$F \approx \sum_{i=0}^{n-1} f(i \cdot \Delta x)\Delta x \text{ mit } \Delta x = \frac{W}{n} \text{ gilt } F \approx \frac{W}{n}\sum_{i=0}^{n-1} f(i \cdot \Delta x)$$

Wobei $f(x)$ an n Stellen mit Abstand Δx ausgewertet worden ist (Auswertung mit Quadraturformel).

Monte-Carlo Integration: Alternativ kann die Auswertung von $f(x)$ an zufällig ausgewählten anstatt regelmäßig verteilten Stellen erfolgen. Die Position x sei eine Zufallsvariable X, die entsprechend der Wahrscheinlichkeitsverteilung $p(x)$ verteilt ist, d.h. $p(x)$ ist immer positiv und das Integral von $p(x)$ über den Definitionsbereich ist 1. Das Integral lässt sich damit für eine Menge von Sample-Positionen X_i mittels Monte Carlo Integration berechnen:

$$F \approx \frac{1}{n}\sum_{i=0}^{n-1} \frac{f(X_i)}{p(X_i)}$$

Das Ziel ist eine hohe Genauigkeit mit wenigen Samples, weitere Varianten sind Importance Sampling und stratified Sampling.

Textur

Reale Oberflächen haben ein großes Spektrum geometrischer Formen, Materialien, (Fein-) Strukturen, diese können regelmäßig und unregelmäßig sein. Eine exakte Nachbildung der geometrischen Detailform ist oftmals zu aufwendig, daher werden Oberflächen mit Texturen überzogen.

Texturen erlauben natürliche Gestaltung von komplexen 3D-Szenen ohne die Geometrie zu ändern. Stattdessen werden die Werte einer Textur den graphischen Primitiven der 3D-Objekte zugeordnet und fast alle Parameter können durch eine Textur modelliert werden. Die Texturen werden dabei wie eine Art Gummi-Haut über das Objekt gezogen.

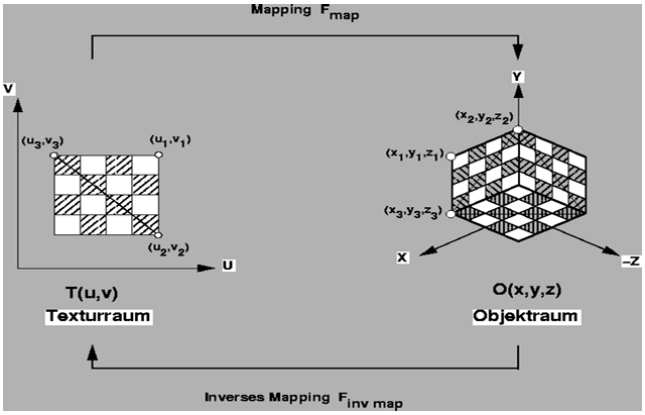
Texture Mapping

Es gibt zwei Grundsätze beim Texture Mapping:

- Non-Parametric Texture Mapping: Größe und Orientierung der Texturen bleiben konstant
- Parametric Texture Mapping: Größe und Orientierung der Textur hängen von der Objektgeometrie ab.

2D-Texturen sind Funktionen, die Punkte der (u, v)-Textur-Ebene auf (r, g, b)-Farben (Texturwerte) abbilden: $(r, g, b) = C_{tex}(u, v)$.

Normalerweise ist C_{tex} nur an diskreten Stellen $C[i, j]$ bekannt (Bildmatrix) mit einem normierten Textur-Koordinatensystem u, v aus $[0,1]$. Manchmal nur mit Grauwerten oder zusätzlichem Alpha-Kanal.



Die Textur-Abbildung beschreibt, wie eine 2D-Textur auf eine vorgegebene Oberfläche aufgebracht wird. Texturierung erfordert

Lösung des „inversen Mapping“. Jedem Flächenpunkt P mit Koordinaten (x, y, z) müssen (u, v)-Textur-Koordinaten zugeordnet werden, dies geschieht mit einer zu spezifizierenden Funktion $F_{inv\ map}$: $(u, v) = F_{inv\ map}(x, y, z)$.

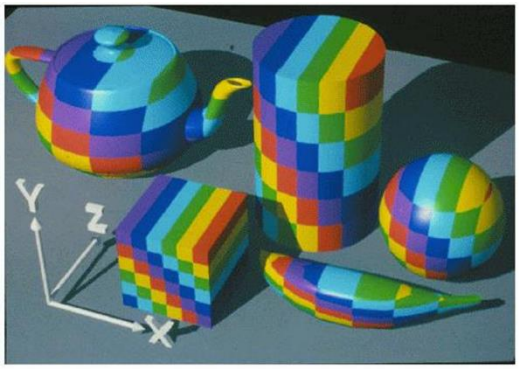
Textur-Mapping mit einer 2D-Textur (mathematisch): $(r, g, b) = C_{tex}(F_{inv\ map}(x, y, z))$.

In der Praxis: Jedem Flächenpunkt P eines 3D-Objekts mit Koordinaten müssen durch $F_{inv\ map}$ geeignete Textur Koordinaten zugeordnet werden. Polygonale Modelle erfordern (in einfachem Kontext) lediglich die Zuordnung von geeigneten Textur Koordinaten für deren Eckpunkte. Beim Pixel Shading wird die Textur-Koordinaten u, v durch lineare Interpolation zwischen den Eckpunkten berechnet.

Arten von Textur-Mapping

Manuelle Zuweisung der Textur-Koordinaten ist sehr aufwändig. Das Zweischrittverfahren von Bier und Sloan soll dies vereinfachen. Ein komplexes Objekt wird mit einer einfachen parametrisierbaren Fläche umhüllt. 2D-Textur wird auf diese umhüllende Fläche abgebildet. Anschließend wird von dort aus auf das Objekt projiziert.

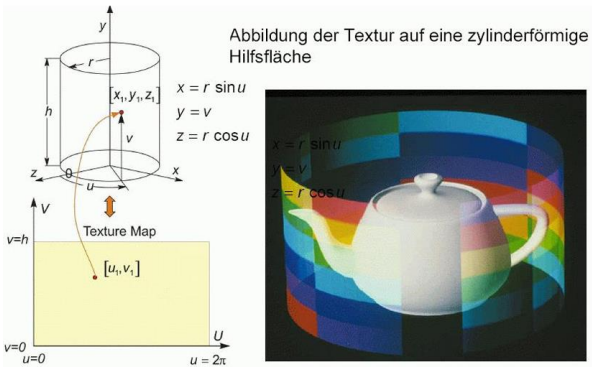
Box-Mapping



- Umhüllende Fläche: Quader (z.B. achsenparallele Bounding Box)
- Parametrisierung: längste Achse u, zweitlängste Achse v
- Projektion: entlang drittlängster Achse

Zylinder Mapping

- Umhüllende Fläche: Zylinder
- Parametrisierung: Höhe und Rotationswinkel
- Projektion: senkrecht zur Zylinderachse



Kugel-Mapping

- Umhüllende Fläche: Kugel
- Parametrisierung: Kugelkoordinaten.
- Projektion: Zentralprojektion.

Berechnung:

- Invers: $(u, v) = F_{inv\ map}(x, y, z)$
- Ursprung in Kugelmitte: $v = \sin^{-1}\left(\frac{z}{\sqrt{x^2+y^2+z^2}}\right), v \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$

- Falls $x \geq 0 \Leftrightarrow u \in [0,\pi]: u = \cos^{-1}\left(\frac{z}{\sqrt{x^2+y^2}}\right), u \in [0,\pi]$
- Sonst $x < 0 \Leftrightarrow u \in [\pi, 2\pi]: u = 2\pi - \cos^{-1}\left(\frac{z}{\sqrt{x^2+z^2}}\right), u \in [\pi, 2\pi]$
- Alternativ: $u = \arctan 2(x,z), u \in [-\pi,\pi]$

3D-Texturen

3D-Texturen werden auch als Festkörpertexturen bezeichnet. Sie werden analog behandelt und verwendet: 3D-Texturen sind Funktionen, die Punkte eines (u,v,w)-Textur-Raumes auf (r,g,b)-Farben abbilden: $(r,g,b) = C_{tex}(u,v,w)$

Inverses Mapping ordnet (x,y,z)-Flächenpunkten P (u,v,w)-Textur-Koordinaten zu. Man schnitz so zusagen den Körper aus dem (u,v,w)-Texturkörper heraus.

Rekonstruktion aus diskreten Texturen

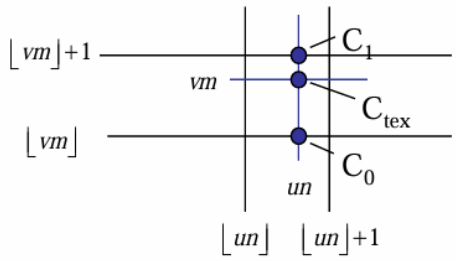
Die (R,G,B)-Farbwerte von C_{tex} sind normalerweise nur an einer Menge von diskreten Stellen (Bildmatrix) bekannt. $\{C[i,j]|0 \leq i < n, 0 \leq j < m\}$

Ist (u,v) aus $[0,1] \times [0,1]$ beliebig, so muss der Wert $C_{tex}(u,v)$ aus diesen diskreten Informationen sinnvoll rekonstruiert werden, also möglichst schnell und genau. Dies führt zu einem klassischen Approximationsproblem: modelliere bivariate Funktion aufgrund vorgegebener (strukturierter) Daten.

Alternativ kann man prozedurale Texturen verwenden, bei denen man eine Funktion $C_{tex}(u,v)$ verwendet, um spezifische Texturen zu erzeugen. Diese sind in jeder Auflösung darstellbar aber nur schwierig oder gar nicht durch eine geeignete Funktion beschreibbar.

Die Rekonstruktion aus diskreten Texturen beschäftigt sich mit der Herausforderung, kontinuierliche Farbwerte aus einer diskreten Menge von Texturdaten zu generieren. Hierbei werden zwei Hauptmethoden verwendet:

1. **Stückweise konstante Rekonstruktion (nächster Nachbar)**: Bei dieser Methode wird jedem Pixel der Farbwert des nächstgelegenen Texels (Textur-Element) zugewiesen. Dies ist eine sehr einfache Herangehensweise, kann jedoch zu deutlich sichtbaren Pixelgrenzen führen, besonders wenn die Textur vergrößert dargestellt wird. $C_{tex}(u,v) = C[\lfloor un \rfloor, \lfloor vm \rfloor]$
2. **Bilineare Interpolation:** Diese Methode bietet eine weichere und natürlichere Rekonstruktion. Hierbei wird der Farbwert eines Pixels durch eine gewichtete Kombination der Farbwerte der vier nächstgelegenen Texel berechnet. Die Gewichtung hängt von der relativen Position des Pixels zu diesen Texeln ab.

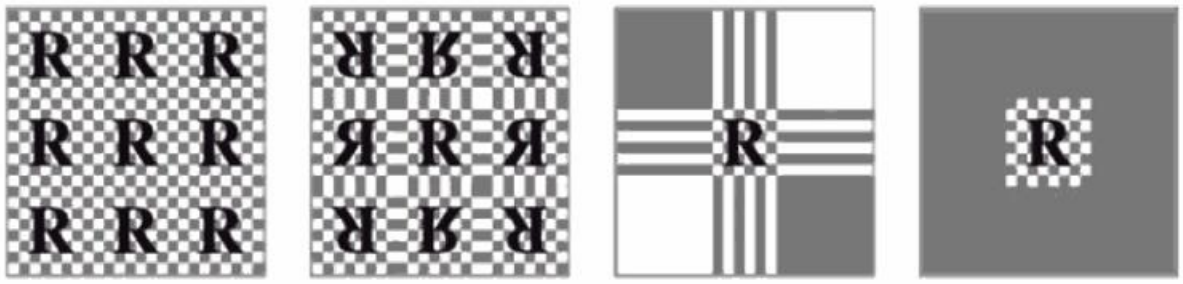


$$\hat{u} = un - \lfloor un \rfloor$$
$$\hat{v} = vm - \lfloor vm \rfloor$$
$$C_0(u,v) = \hat{u} \cdot C[\lfloor un \rfloor + 1, \lfloor vm \rfloor] + (1-\hat{u}) \cdot C[\lfloor un \rfloor, \lfloor vm \rfloor]$$
$$C_1(u,v) = \hat{u} \cdot C[\lfloor un \rfloor + 1, \lfloor vm \rfloor + 1] + (1-\hat{u}) \cdot C[\lfloor un \rfloor, \lfloor vm \rfloor + 1]$$
$$C_{tex}(u,v) = \hat{v} \cdot C_1(u,v) + (1-\hat{v}) \cdot C_o(u,v)$$

Texturwiederholung

Umgang mit dem Fall, dass eine Textur außerhalb ihres Definitionsintervall ausgewertet wird.

- Repeat: Textur wird gekachelt
- Mirror: jeweils gespiegelte Kachelung
- Clamp: der Texturparameter wird auf das gültige Intervall abgeschnitten
- Border: außerhalb gilt ein fester Farbwert



Manche Texturen lassen sich nicht Kacheln, ohne dass es sofort auffällt. Dafür kann man zufällige Variation (z.B. Rauschen) einbringen oder alternativ Textursynthese verwenden.

Bei der Textursynthese wird eine große Textur anhand eines kleinen Ausschnitts generiert. Zufallsaspekte zur Reduktion von klaren Mustern und Farbhistogramme, um Ähnlichkeit zum Ausgangsmuster zu verbessern.

Probleme diskreter Texturen

Große Texturen mit hoher Auflösung haben einen hohen Speicherbedarf. Beim Vergrößern von Texturen können Artefakte auftreten und die Fortsetzung ist häufig problematisch. Häufig erkennt man, dass es sich um 2D-Bilder in einer 3D-Szene handelt.

Perspektive

Abstände im 2D-Bildraum korrespondieren nicht mit den Abständen im 3D-Raum (vor der perspektivischen Projektion), so können ungewünschte Verzerrungen der Textur entstehen, falls lediglich eine einfache lineare Interpolation die Textur-Koordinaten zwischen den Eckpunkten zur Bestimmung der Texturwerte der Pixel verwendet wird.

Abtastfehler

Textur Mapping auf beliebige Flächen resultiert in sichtbaren Verschmierungs- und Aliasing Effekten (Abtastfehler). Die Ursache hierfür ist, dass die Größe von Texel und Pixel quasi nie übereinstimmen. Bei der Vergrößerung gibt es zu viele Pixel für einen Texel und bei der Verkleinerung entspricht ein Pixel mehreren Texel.

Zur Darstellung von Textur reicht es nicht aus, für einen Bildschirmpixel (x', y') die Texturkoordinaten (u, v) zu berechnen.

Filterung

Oversampling: Texel werden auf mehrere Pixel verschmiert, Bildschirmbild wird unschärfer, kaum vermeidbar, denn Texturauflösung ist fest

Undersampling: Mehrere Texel fallen auf einen Pixel, tritt häufig in 3D-Grafik auf (perspektivische Projektion), Artefakte oftmals vermeidbar durch Verwendung von geeigneten Filterstrategien.

Footprint

Der Footprint ist die Projektion eines Pixels auf die Texturebene und ist näherungsweise ein Parallelogramm aus den Vektoren $r_1 = \left(\frac{\partial u}{\partial x}, \frac{\partial v}{\partial x}\right)^t, r_2 = \left(\frac{\partial u}{\partial y}, \frac{\partial v}{\partial y}\right)^t$.

Mip-Mapping

Sukzessive Mittelung der Texel ergibt verschiedene Level (= gröbere Darstellungen der Textur)

Mip-Map $C_{mip}[i,j]$ speichert eine quadratische Textur $C[i,j]$ der Größe $n \times n$, wobei $n = 2k$ in fortlaufend halbierten Auflösungsstufen, dadurch entstehen k+1 Auflösungsstufen, benötigt etwa 1/3 mehr Speicheraufwand als die Original-Textur.

MipMap-Level d=0 enthält die vorgegebene Textur $C^0_{mip}[i,j] = C[i,j], 0 \leq i,j < 2^k$.

MipMap-Level d entsteht durch Filterung von je vier (verschiedenen) Werten des MipMap-Level d-1 $C^d_{mip}[i,j] = \frac{1}{4}\left(C^{d-1}_{mip}[2i,2j] + C^{d-1}_{mip}[2i+1,2j] + C^{d-1}_{mip}[2i,2j+1] + C^{d-1}_{mip}[2i+1,2j+1]\right), 1 \leq d < k-1, 0 \leq i,j < 2^{k-d}$.

Auf Level d der Texturhierarchie werden 2^{2d} Texel der Originaltextur als ein einziges (gemitteltes) Texel dargestellt.

Bestimmung des MipMap-Levels durch berechnen der Texturkoordinaten (u,v) des Pixelmittelpunkts sowie Ableitung nach den Bildschirmkoordinaten. Die Projektionen des quadratischen Pixels auf die Texturebene hat die Kantenlängen $a = \|r_1\|\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2}, b = \|r_2\|\sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}$, wenn d nicht ganzzahlig, kann zwischen zwei Stufen interpoliert werden.

Anisotropie

Die Richtungsabhängigkeit der Projektion (Anisotropie) des Footprints kann durch das Mip-Mapping nicht direkt berücksichtigt werden. Footprints werden durch ein Quadrat approximiert. Manchmal unterscheiden sich Footprint und Texelraster stark (z.B. Betrachtung eines Polygons aus flachem Blickwinkel). Footprints können zu gross sein und zu “Verschmierungen” führen. Ein Ausweg: ist Footprint-Assembly.

Footprint-Assembly (FPA)

Näherung des Footprints durch einen quadratischen (Mip-Map) oder rechteckigen (SAT) Texturausschnitt ist ungenau, besonders wenn die Achsen voneinander stark abweichen. Beim Footprint-Assembly werden mehrere Texturausschnitte gemittelt, um das Parallelogramm besser zu approximieren.

Anzahl der Texturausschnitte ist eine Zweierpotenz: $N = 2^m$

Mit Mip-Map-Level $d = \log_2(\min(r_1, r_2))$, Anzahl Texturausschnitte $N = \frac{\max(r_1,r_2)}{\min(r_1,r_2)}$ (gerundet auf nächste Zweierpotenz). Schrittrichtung Δr ist der vom Betrag größere Vektor aus r_1 und r_2, skaliert mit $\frac{1}{N}$. Auswertung an den Punkten $p_n = p + \frac{n}{2} \cdot \Delta r, n \in \{\pm 1, \pm 3, \ldots, \pm(N-1)\}$

Summed Area Tables

Eine alternative Möglichkeit der Berücksichtigung der Richtungsabhängigkeit der Projektion des Footprints. An Stelle von Quadraten werden hierbei (u,v)-achsenparallele Rechtecke zur Approximation der Footprints verwendet.

Idee: berechne arithmetisches Mittel aller Texturwerte in einem (u,v) -achsenparallelen Rechtecks mit Eckpunkten (i0,j0) und (i1,j1) als

$$C_{avg}(i_0, j_0, i_1, j_1) = \frac{\sum_{i=i_0}^{i_1} \sum_{j=j_0}^{j_1} C[i,j]}{(i_1 - i_0 + 1)(j_1 - j_0 + 1)}$$

Aufwand ist proportional zur Größe des Rechtecks: $O((i_1 - i_0) \cdot (j_1 - j_0))$. Um den Aufwand zu reduzieren bestimmt man vorab Summed Area Table mit Einträgen

$$C_{sat}(i_s, j_s) = \sum_{i=0}^{i_s} \sum_{j=0}^{j_s} C[i,j]$$

Wobei $i_s < m$ und $j_s < n$ und m, n fix sind. Dann wird eine konstante Anzahl von Operationen zur Bestimmung der obigen arithmetischen Mittel C_{avg} benötigt.

Tunneltest

Der Tunneltest ist eine Qualitätsprüfung für Filteralgorithmen. Der Betrachter blick in einen unendlich langen Zylinder, verschiedene Mip-Map Stufen werden farbig gekennzeichnet, damit der Übergang klar erkennbar wird. Die Filteralgorithmen verursachen einen verwaschenen Übergang durch trilineare Interpolation und Footprint-Approximation. Im Ideal hat man runde Kreise mit sanften Übergängen.

Perspektivische Korrektur

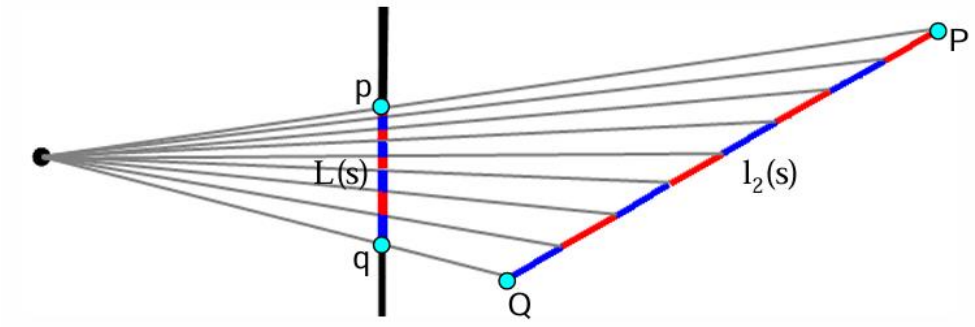
Die Textur muss perspektivisch korrekt dargestellt werden, aber Rasterisierung erst nach der perspektivischen Transformation. Einfache (affine) Interpolation der Texturwerte aus den Texturwerten an den Eckpunkten führt zu ungewünschten Verzerrungen.

Die perspektivische Projektion der Gerade $l_2(s) = P + s(Q - P)$ entspricht nicht der Gerade $l_1(t) = p + t(q - p)$ mit s und t in [0,1].

Projektion L des Liniensegments l_2 auf $z = 1$mit $P = (x_1, y_1, z_1)$ und $Q = (x_2, y_2, z_2)$

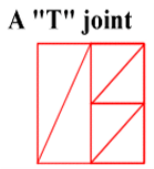
$$L(s) = \frac{P + s(Q - P)}{z_1 + s(z_2 - z_1)} = \frac{1}{z_1 + s(z_2 - z_1)} \begin{pmatrix} x_1 + s(x_2 - x_1) \\ y_1 + s(y_2 - y_1) \\ z_1 + s(z_2 - z_1) \end{pmatrix}$$

$$= \left(\frac{x_1 + s(x_2 - x_1)}{z_1 + s(z_2 - z_1)}, \frac{y_1 + s(y_2 - y_1)}{z_1 + s(z_2 - z_1)}, 1 \right)^T$$



Da Rasterisierung in Bildschirmkoordinaten stattfindet, muss eine Abbildung zwischen s und t gefunden werden. Alle Punkte von $L(s)$ projizieren auf $l_1(t)$.

$$s = \frac{tz_1}{z_2 + t(z_1 - z_2)}$$

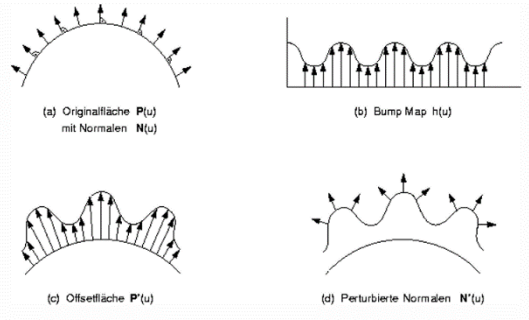


Die Integration der Texturierung in der Rasterisierung zur perspektivisch korrekten Darstellung ist aufwendig. Eigentlich müssten Helligkeit und Farbwerte auch perspektivisch korrigiert werden, aber es geht normalerweise auch so, und ist schneller. Fehler leicht erkennbar bei T-Knoten.

Weitere Mappings

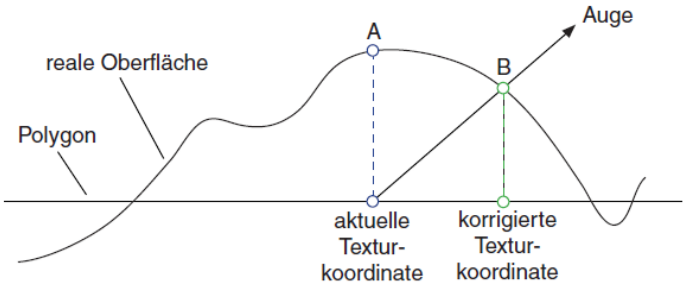
Bump Mapping

Simulation von Oberflächenunebenheiten, geht nur, wenn das Beleuchtungsmodell pro Pixel ausgewertet wird. Die Bump Map muss extra gespeichert werden oder die Textur braucht perturbierte (Perturbation = Störung (positiv)) Normalen. Die Vereinigung von 3D-Objekt und der Bump-Map ergibt ein detailliertes Modell. Perturbierte Normalen beeinflussen die Beleuchtung, aber nicht die Geometrie.



Parallax Mapping

Texturkoordinaten werden auf Per-Pixel-Basis modifiziert. Berechnung aus einem Höhenfeld (als Textur) und Kameraposition. Geometrisch flach wie Bump Mapping, aber wirkt plastischer. Schneller zu berechnen als Displacement Mapping.



Displacement Mapping

Displacement Mapping verändert die Geometrie des Objektes.

View-dependent Displacement Mapping: Berechnet die Abstände nicht entlang der Oberflächennormale, sondern entlang der Blickrichtung. Für mehrere Blickrichtungen V, ausgedrückt in sphärischen Koordinaten (θ,ϕ), wird ein Punkt P auf der Referenzoberfläche mit den Texturkoordinaten (u,v) auf den Punkt P' projiziert. Punkt P' besitzt dabei Texturkoordinaten (u',v'). Krümmung c einer Oberfläche kann ebenfalls mit einbezogen werden. Der Abstand d ist somit abhängig von θ, ϕ, u, v und c (5D-Textur).

5D-Texturen sind sehr speicherintensive und Grafikhardware ist nur auf 2D-Texturen spezialisiert. Um dieses Problem zu lösen, werden die 5D-Texturen in zwei niedrig-dimensionale Texturen aufgeteilt.

Environment Mapping

Berechnung von Spiegelungen einer Scene mit Textur-Hardware. Ist ein Objekt verglichen mit dem Abstand zu umgebenden Objekten klein, so hängt die einfallende Beleuchtungsstärke nur von der Richtung, nicht von der Position eines Punktes auf dem Objekt ab. Daher kann die einfallende Beleuchtung für ein Objekt vorberechnet und in einer 2D-Textur, der Environment Map, gespeichert werden.

Erstellen von Environment Map: Das reflektierende Objekt wird von einer virtuellen Fläche umhüllt (analog Two-Part Mapping). Berechne Zentralprojektion der Scene auf diese Fläche und speichere diese als diskrete Textur.

Berechnung der Farbwerte: Berechne die ideale Reflektionsrichtung (abhängig von der Normalen der Oberfläche und der Kameraposition). Berechne Schnitt der Reflektionsrichtung mit der umhüllenden Fläche und entnehme dort den Farbwert.

Vorteile:

- Schnell und einfach zu berechnen
- Liefert gute Ergebnisse, wenn die Textur z.B. den Himmel oder einen weit entfernten Horizont repräsentiert

Nachteile:

- Die Reflektionsberechnung ist nur dann korrekt, wenn sich der Objektpunkt P im "Weltmittelpunkt" (Zwischenflächen-Mittelpunkt) W befindet. Je größer der Abstand zwischen P und W, desto stärker die Verzerrungen
- Ist die Environment Map schlecht parametrisiert, können erhebliche Aliasing-Probleme auftreten
- Es wird keine Verdeckungsrechnung durchgeführt
- Szenenobjekte können sich nicht gegenseitig widerspiegeln
- Vorsicht vor Artefakten an den Kanten und "Nahtstellen" des Projektionskörpers und durch Interpolation

Radiosity

Verfahren zur Berechnung der Verteilung von Wärme- oder Lichtstrahlung innerhalb einer Szene mittels physikalischen Modells (Optik, Wärmelehre). Beruht auf Energieerhaltungssatz: Strahlung, die auf Fläche fällt und nicht absorbiert wird, wird von ihr zurückgeworfen. Dafür wird angenommen, dass alle Oberflächen ideal diffuse Reflektoren sind und die Quellen ideal diffuse Strahler sind. Das bedeutet eine gleichmäßige Reflexion der Strahlung in alle Richtungen über die Hemisphäre, dabei ist die Strahlenverteilung unabhängig von der Position des Betrachters.

Die Radiometrie beschreibt den Lichttransport in einem vereinfachten physikalischen Modell, unabhängig von Zeit, Polarisation und Wellenlänge.

Größe	Einheit	Beschreibung	Fotometrische Entsprechung
Strahlungsenergie	J	Die Energie einer Anzahl von Photonen	Lichtmenge
Strahlungsfluss	W	Strahlungsenergie pro Zeit	Lichtstrom
Strahlstärke (Radiosity)	$\frac{W}{sr}$	Strahlungsfluss pro Raumwinkel	Lichtstärke / Lichtintensität
Bestrahlungsstärke	$\frac{W}{m^2}$	Strahlungsfluss pro Empfängerfläche	Beleuchtungsstärke

Raumwinkel

Erweitern des Winkels in die dritte Dimension, Fläche der Projektion eines Objekts auf die Einheitskugel, an sich dimensionslos aber Einheit Steradiant, diese entspricht der Bogenlänge in 2D.

$$\omega = \frac{A}{r^2}, [\omega] = \frac{m^2}{m^2} = sr$$

Der differentielle Raumwinkel ist die Projektion auf die Tangentialebene, Approximation: $\Delta\omega = \frac{\Delta A \cos\theta}{r^2}$

Rendering Equation mit diffuser Reflexion:

$$B(x) = B_e(x) + \rho(x) \int_S B(x') \cdot \frac{\cos\theta \cos\theta'}{\pi r^2} V(x, x') dA'$$

Finite-Element Methode

Radiosity-Gleichungssystem:

$$B_i = B_{e,i} + \rho_i \sum_{j=1}^{n} F_{ij} B_j \, , mit \; F_{ij} = \frac{1}{A_i} \int_{x \in S_i} \int_{x' \in S_j} \frac{\cos\theta \cos\theta'}{\pi r^2} V(x, x') dx' dx$$

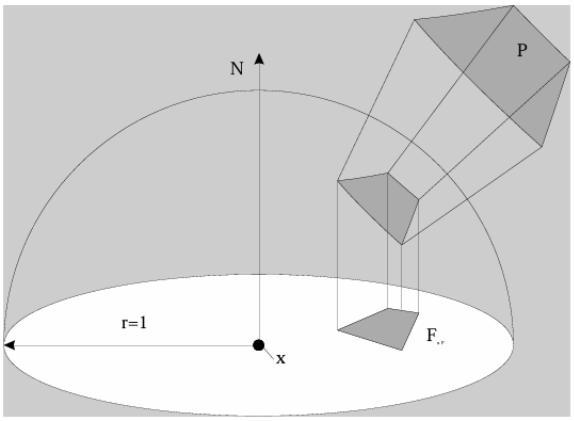
Rein geometrisch misst man den eintreffenden Lichtanteil. Der differenzielle Formfaktor lautet:

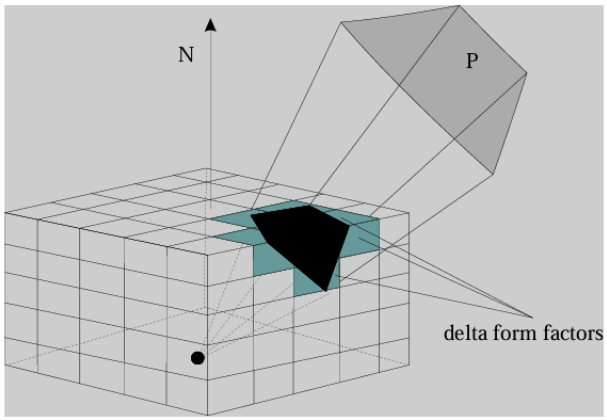
$$F_{dAdA'} = \frac{\cos\theta \cos\theta'}{\pi r^2} dA'$$

Durch die Finite-Element Methode kann man unbekannte Formfaktoren berechnen. Die FE-Methode ist Reziprok ($A_i F_{ij} = A_j F_{ji}$) und Additiv ($F_{i(j \cup k)} = F_{ij} + F_{ik}$). Beim Berechnen des Formfaktors ist die analytische Lösung meistens unbekannt.

Hemicube

Der Anteil der bedeckten Grundfläche entspricht dem Formfaktor (projizierter Raumwinkel). Ersetzen der Hemisphäre durch einen Hemicube mit Delta-Formfaktoren und Itembuffer, so erhält man die Formfaktoren zur gesamten Szene aber mit Aliasing Effekten.





Monte Carlo Integration

Die Monte Carlo Integration verwendet die Unterteilung in differentielle Subpatches (Point-to-Disc Formfaktor) $F_{dA,A'} = \frac{A'}{\pi r^2 + A'}$.

Somit ergibt sich der folgende Formfaktor, wenn die Sichtbarkeit mittels Raytracer berechnet wird.

$$F_{ij} = \frac{A_j}{n} \sum_{k=1}^{n} \frac{\cos\theta_i^k \cos\theta_j^k}{\pi r^2 + \frac{A_j}{n}} V(x_i, x_j)$$

Radiosity Matrix

$B_i = B_{e,i} + \rho_i \sum_{j=1}^{n} F_{ij} B_j$ für jeden Patch berechnen, man erhält n Gleichungen mit n Unbekannten $MB = B_e$. Voraussetzung für die Konvergenz divergenter Lösungsmethoden: $\sum_{j=1, j \neq i}^{n} |M\_ij| \leq$

$|M_{ii}|, \forall i \ (F_{ii} = 0)$. Summe der Formfaktoren einer Reihe (Diagonalelement ausgenommen) ist kleiner als 1 (Energieerhaltung).

Gauss-Seidel Iteration

Entspricht dem „Einsammeln“ von Energie. Pro Iteration eine oder mehrere Reflektionen, Quadratische Anzahl an Formfaktoren.

```
/* initialize starting guess */
for (each i)
   B_i = B_ei
while(not converged) {
   /* loop over all patches */
   for (each i)
      B_i = B_ei + ρ_i Σ_{j=1,j≠i}^{n} F_ij B_j
   /* display current solution */
   render(B);
}
```

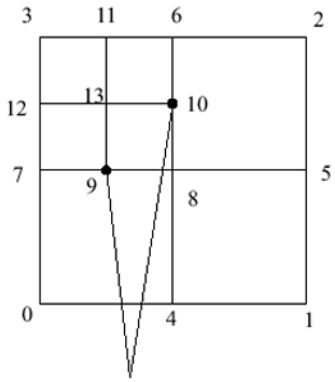
Rekonstruktion

Radiosity liegt für jedes Flächenelement vor mit einem konstanten Wert über die gesamte Fläche. Bilineare Interpolation hat zwar Artefakte aber ist Hardwareunterstützt. T-Vertices werden aufgebrochen.

Radiosity Texture

Verhindern von T-vertices aber mit Tradoff zwischen Textur und Geometrie.

Radiosity Mesh



T-vertices

Radiosity Texture

3	11	6	a	2
12	13	10	b	c
7	9	8	d	5
e	f	g	h	i
0	j	4	k	1

Durch Buchstaben gekennzeichnete Texel werden durch Filtern erzeugt.

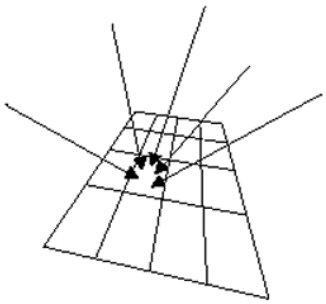
In der Berechnung werden nur Durchschnittswerte der Textur verwendet und anstelle der Radiosity wird Irradiance ausgegeben.

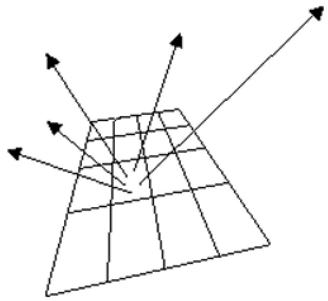
$$B_i(x) = B_{ei} + \rho_i(x)E_i \Rightarrow E_i = \frac{B_i - B_{ei}}{\rho_i}$$

Die Multiplikation mit ρ erfolgt in der Grafikhardware und liefert die Radiosity.

Southwell Iteration

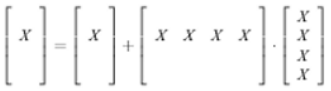
Ein weiteres Verfahren, um die Formfaktor-Kosten zu reduzieren, das Prinzip liegt hierbei aber dabei, die Energie zu verteilen.

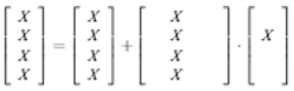




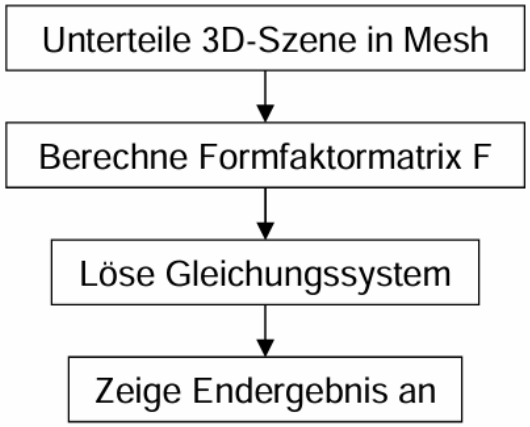
Gathering (Gauss-Seidel): aktualisiert einen Eintrag

Shooting (Southwell): aktualisiert gesamten Lösungsvektor

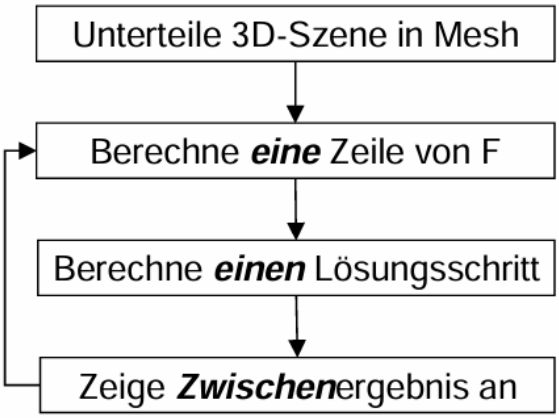




Progressive Refinement



Full Matrix Radiosity



Progressive Refinement
(wählt immer den hellsten Patch)

Ambient

Term: Flächen ohne Energie bleiben anfangs Dunkle, Abschätzung der zukünftigen Energie.

Durchschnittlich unverteilte Energie: $U = \frac{\sum_{i=1}^{N} \Delta B_i A_i}{\sum_{i=1}^{N} A_i}$

Durchschnittliche Reflektivität: $\rho_{ave} = \frac{\sum_{i=1}^{N} \mathrm{A}_i \rho_i}{\sum_{i=1}^{N} A_i}$

Die ambiente Korrektur ist die Abschätzung aller zukünftigen Reflektionen.

Abnahme der durchschnittlichen Reflektivität: $R = 1 + \rho_{ave} + \rho_{ave}^2 + \rho_{ave}^3 + \cdots = \frac{1}{1-\rho_{ave}}$

Korrigierter Radiositywert (nur für die Anzeige): $B_i^{disp} = B_i + \rho_i RU$

Unterteilung

Bisher gab es eine reguläre Unterteilung, diese muss feingenug sein, um Schatten herauszubilden, unnötige Unterteilung erhöht Rechen bzw. Speicheraufwand. Empfänger sollte feiner unterteilt sein als der Sender.

Hierarchische Unterteilung: Hierfür wird eine 2-stufige Hierarchie angewandt. Der Emitter/Reflektoren ist grob unterteilt (wenige Iterationen, da Shooting) und der Empfänger verwendet eine höhere Auflösung.

Adaptive Unterteilung: A priori Unterteilung kennt Radiosity-Funktion nicht, stattdessen eine adaptive Unterteilung wie z.B. Quadtree wählen. Unterteilung basiert auf Radiosity-Gradienten.

Hierarchical Radiosity

Kombination aus Hierarchie und adaptiver Unterteilung, verwenden des optimalen Levels für jede Interaktion.

Zu beginn hat man keine Unterteilung und baut eine adaptive Unterteilung in einer Quadtree Struktur aufgrund eines Orakels auf. Erzeugung von Links über die Energie zwischen den dadurch verbundenen Hierarchiestufen.

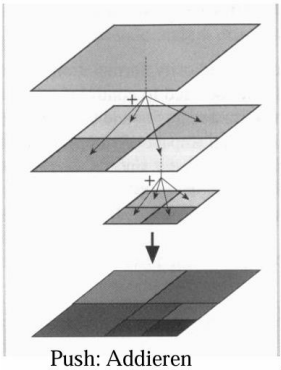
Algorithmus: Für jedes Paar von Eingabepolygonen aufgerufen, link() Funktion berechnet den Formfaktor.

```
refine(p, q)
{
   if(oracle(p,q) == OK or
           (area(p) < A_ε and area(q) < A_ε)) {
      link(p,q)
   } else {
      if(subdiv(p,q) == p) {
         // p was subdivided
         for all children c of p
            refine(c,q)
      } else {
         // q was subdivided
         for all children c of q
            refine(p,c)
      }
}
```

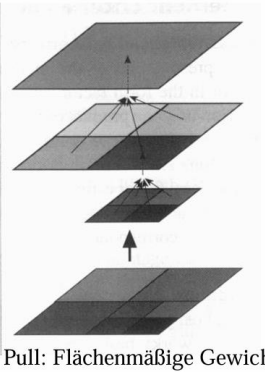
Anschließend einsammeln der Energie über alle links und anteiliges Verteilen innerhalb der Hierarchie.

```
/* initial linking */
for (each polygon p)
   for (each polygon q)
      if(p != q)
         refine(p,q);

/* solving */
while (not converged) {
   for (each input polygon p) gather(p);
   for (each input polygon p) pushpull(p, 0);

}

gather(p)
{
   p.B_g = 0    /* initialize gathered radiosity */
   for (all incoming links l) {
      p.B_g = p.B_g + l.formfactor * l.q.B_s * ρ_p
   }
   for (each child c of p)
      gather(c);
}
```

```
pushpull(p, B_down)
{
   if(p has children) {
      B_up = 0
      for (each child c of p)
         B_up = B_up + A_c/A_p * pushpull(c, p.B_g+B_down)
   } else {
      /* p is leave node */
      B_up = p.E + p.B_g + B_down
   }
   p.B_s = B_up
   return B_up
}
```



Push: Addieren



Pull: Flächenmäßige Gewichtung

Orakel

Das Orakel bestimmt die Qualität der Lösung und ist Laufzeitkritisch und kann direkt, als Fehlermaß verwendet werden, indem der Nutzer einen Schwellwert zur Steuerung der Unterteilung angibt.

BF-Refinement

BF ist die Multiplikation der Radiosity mit dem Formfaktor, um eine gleiche Energiemenge pro Link zu erhalten. Wird auch als adaptives Link-Refinement bezeichnet.

```
tmp = A_ε
A_ε = ∞
/* initial linking */
for (each polygon p)
   for (each polygon q)
      if(p != q)
         refine(p,q);
A_ε = tmp

done = FALSE;
while(done == FALSE) {
   /* solving */
   while (not converged) {
      for (each input polygon p) gather(p);
      for (each input polygon p) pushpull(p, 0);
   }
   done = TRUE;
   /* refinement */
   for all links l
      if(refineLink(l) == FALSE)
         done = FALSE;
}
```

Laufzeit

Links zwischen höheren Ebenen entsprechen Blöcken in der FF-Matrix. Anzahl der Links ist linear in der Anzahl der Blattknoten mit einer quadratischen Startphase erhalten wir $O(k^2 + n)$

Rasterisierung

Ein Pixel ist 1.0 hoch und 1.0 breit

DirectX 9 und davor:

- Pixelzentrum von Pixel (i,j) bei (i.0,j.0)
- 10 Pixel (Pixel 0 bis 9) decken das Intervall [-0.5,9.5) ab
- Umwandlungen zwischen Fließkommazahl c und Ganzzahl d:
    - d = round(c)
    - c = d

OpenGL, DirectX 10 und Nachfolger:

- Pixelzentrum von Pixel (i,j) bei (i.5,j.5)
- 10 Pixel (Pixel 0 bis 9) decken das Intervall [0.0,10.0) ab
- Umwandlungen zwischen Fließkommazahl c und Ganzzahl d:
    - d = floor(c)
    - c = d + 0.5
- OpenGL: kartesisches Koordinatensystem (Ursprung links unten)
- DirectX: Ursprung links oben (Windows), nicht konsistent eingehalten

Die Rasterisierung führt ihre Operationen pro Pixel (Picture Element) aus. Die Inputinformationen sind Transformierte und projizierte Eckpunkte in Bildschirmkoordinaten, Tiefenwerte der Eckpunkte und assoziiert Beleuchtung und Normale. Die Aufgabe der Rasterisierung ist die Berechnung der Farbwerte des Color Buffers.

Triangle Setup

Siehe Triangle Setup

Triangle Traversal

Siehe Triangle Traversal

Annahmen für die folgenden Algorithmen:

- Anfangspunkt (x_a, y_a) und Endpunkt (x_e, y_e) liegen auf dem Raster, sonst werden sie auf das Raster abgeschnitten
- $x_e - x_a > 0$, sonst Rollen der Punkte tauschen
- Steigung m liegt in $[-1,1]$, sonst x mit y tauschen.

Differential Digital Analyzer (DDA)

Das Ziel ist ein Pixelmuster, welches ein vorgegebenes Liniensegment annähert.

Berechnen von:

- $m = \frac{y_e - y_a}{x_e - x_a}$
- $b = y_a - mx_a$
- $y = mx + b$ für $x = x_a, \dots, x_e$

Zeichnen der Pixel an der Stelle $(x, round(y))$

Der **Aufwand** dieses Algorithmus sind Addition, Multiplikation (je Fließkomma) und Rundung je Pixel. Der Algorithmus ist also relativ ineffizient.

Der inkrementelle DDA verwendet keine Fließpunkt-Multiplikation und ist daher schneller:

- Übergang von $x_a + i$ zu $x_a + i + 1$ ergibt sich durch die Addition von m: $m(x_a + i + 1) = m(x_a + i) + m$

Inkrementell beginne mit $y = y_a$

- Zeichne Pixel an der Stelle $(x, round(y))$
- Berechne für nächstes x neues $y = y + m$

Aber auch dieser Algorithmus hat noch Probleme mit Fließkommaaddition und Rundungen und Fehlerakkumulation durch ungenaues m.

Bresenham Algorithmus

Der Bresenham Algorithmus ist ein klassischer Algorithmus mit ganzzahliger Arithmetik. Benötigt lediglich Integer Addition und binäre Sifts.

Idee:

- Nachfolger von Pixel (x, y) kann nur Pixel $(x + 1, y)$ und $(x + 1, y + 1)$ sein
- Wähle Nachfolge-Pixel so aus, dass dieses dem (kontinuierlichen) Liniensegment am nächsten liegt

Mittelpunkt-BA

Mittelpunkt $M = (x + 1, y + 0.5)$ der beiden Kandidaten $(x + 1, y)$ und $(x + 1, y + 1)$ für ein Nachfolge-Pixel von (x, y). Einfache algorithmische Vorschrift: ist $y + 0.5$ größer als $mx + b$ so wähle $(x + 1, y)$, sonst $(x + 1, y + 1)$.

Es wird die implizite Form $F(x, y) = dy\ x - dx\ y + dx\ b$ verwendet

- $m = \frac{(y_e - y_a)}{(x_e - x_a)} = \frac{dy}{dx}$ und $b = ya - m\ xa$
- Punkt auf Linie: $F(x, y) = 0\ y = mx + b$
- Punkt unter Linie: $F(x, y) > 0\ y < mx + b$
- Punkt über Linie: $F(x, y)\ mx + b$
- Ganzzahlige Entscheidungsvariable E
  - $E(M) = 2F(x + 1, y + 0.5) = dy(2x + 2) - dx(2y + 1) + 2b\ dx$
  - wird durch ganzzahlige Inkremente $2dy$ und $2dy - 2dx$ aktualisiert

Algorithmische Vorschrift:

- $E(M) < 0$
  - $y + 0.5$ ist größer als $m(x + 1) + b$
  - Wähle $(x + 1, y)$ als nächsten Pixel
  - Neuer Mittelpunkt ist $M_{neu} = (x + 2, y + 0.5)$
  - Neue Entscheidungsvariable ergibt sich als
  - $E(M_{neu}) = 2F(x + 2, y + 0.5) = dy(2x + 4) - dx(2y + 1) + 2b\ dx = E(M) + 2dy$
  - Inkrement: $2dy$
- sonst
  - Wähle $(x + 1, y + 1)$
  - Neuer Mittelpunkt ist $M_{neu} = (x + 2, y + 1.5)$
  - Neue Entscheidungsvariable ergibt sich als
  - $E(M_{neu}) = 2F(x + 2, y + 1.5) = dy(2x + 4) - dx(2y + 3) + 2b\ dx = E(M) + 2dy - 2dx$
  - Inkrement: $2(dy - dx)$

Rasterisierung von Polygonen

Algorithmus zur Rasterisierung auf die Kanten des Polygons anwenden, Mittelpunkt-Ba, Scanline oder Konvexe Polygone durch Schnitt von Halbräumen (Innen und Außen) beschreiben.

Aliasing

Aliasing ist ein Abtastartefakt und entsteht durch Nichtbeachtung des Abtasttheorems:

> Damit ein Ursprungssignal korrekt wiederhergestellt werden kann, dürfen im abzutastenden Signal nur Frequenzanteile vorkommen, die kleiner als die Nyquist-Frequenz (halbe Abtastfrequenz) sind

In der GDV entsprechen harte Objektkanten stellenweise unendlicher Ortsfrequenz. In der GDV hervorgerufen durch begrenzt aufgelöstes Abtastgitter (Pixel).

Antialiasing

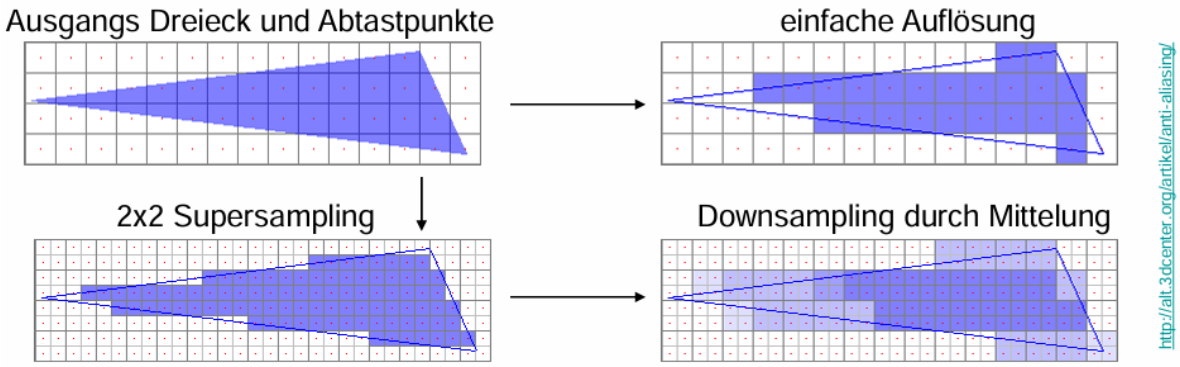
Antialiasing ist eine Variante des Bresenham-Algorithmus:

- Setze für jedes Pixel die Helligkeit der beiden potenziellen Nachfolge-Pixel
- Helligkeit ist von der Entfernung des Pixelmittelpunktes zur Geraden abhängig
- Nimmt linear mit steigender Entfernung zu M ab.
- Gesamthelligkeit ist konstant

Supersampling Antialiasing

Full-scene antialiasing (FSAA) / supersampling antialiasing (SSAA)

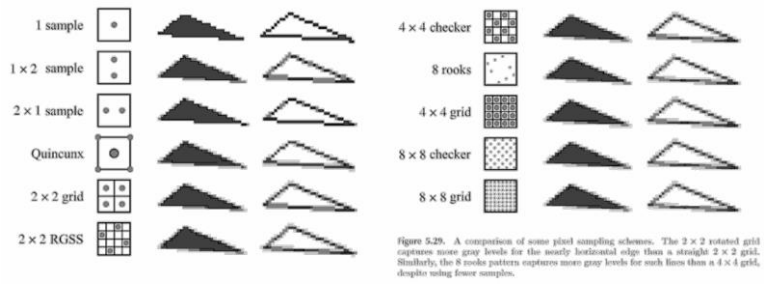
- Mehrere Abtastpunkte zur Farbwertbestimmung auswerten und mitteln
- Sehr einfach umzusetzen, aber kostspielig
- Jedes Subpixel unterläuft das komplette Shading!
- Hohe Qualität: Jedes Subpixel wertet auch Texturinformationen aus



Multisample Antialiasing

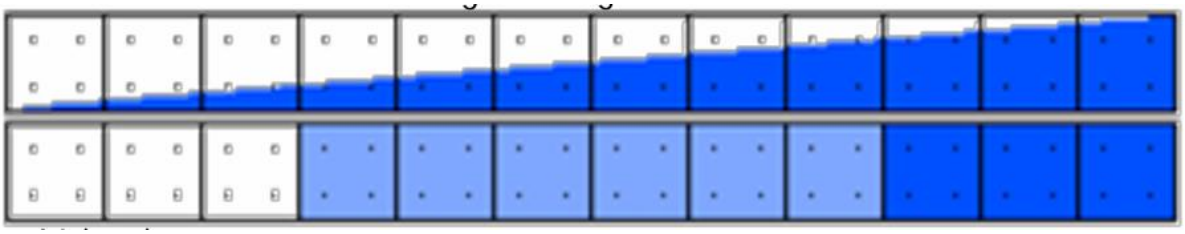
Bei MSAA werden keine Subpixel erzeugt, die das Shading durchlaufen, pro Pixel werden mehrere Samples ausgewertet. Dadurch können alle Samples parallel erzeugt werden und manche Vorrechnungen können für alle Samples übernommen werden. Außerdem benötigt man nur einen Texturzugriff für alle Samples des Pixels.

Wenn alle Samples in einem Polygon liegen ist MSAA identisch zu keinem Antialiasing, MSAA glättet nur Kanten anteilig zu den Samples innerhalb und außerhalb des Polygons.



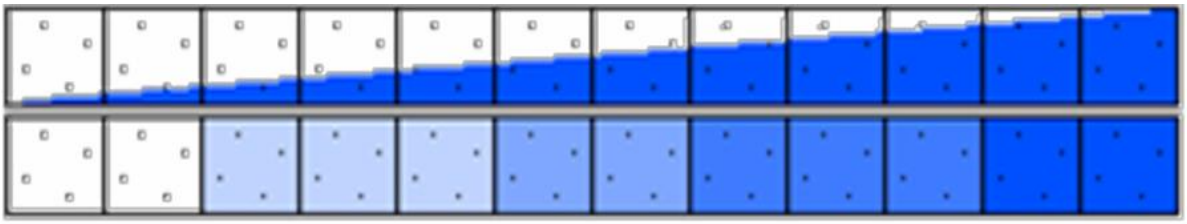
Ordered Grid (2x2)

- Löst nahezu horizontale und vertikale Übergänge schlecht auf
- Gerade diese sind für das menschliche Auge auffällig



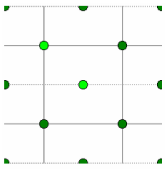
Rotated Grid (2x2)

- Mehr Abstufungen bei nahezu horizontalen oder vertikalen Übergängen



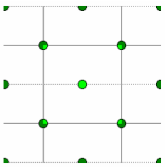
Mehrfachverwendung von Samples

2 x Multisampling



- Pro Pixel ein Sample in der Mitte und ein Sample in der linken oberen Ecke
- Drei Abstufungen: Pixel zu 0%,50%,100% im Polygon
- Farbmittelung pro Pixel direkt und unabhängig von anderen Pixeln möglich

Quincunx (NVidia)

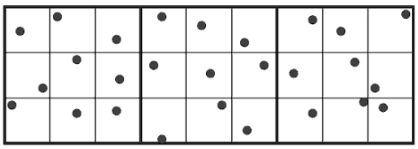


- Nutzt Eck-Samples benachbarter Pixel bei der Mittelung (Eck-Samples gehen zu 25% ein, Mittelpunkt zu 100%)
- Mehr Abstufungen mit gleicher Anzahl Samples
- Zwei Durchläufe
    - 1) Farbwert je Sample (incl. Texturwert des Pixels)
    - 2) Farbmittelung
- Fügt Unschärfe hinzu (Texturinformationen werden ebenfalls gemittelt)

Jitter Pattern

Regelmäßig verteilte Samples werden immer von Aliasing betroffen sein, daher kann man zufallsverteilte Samples verwenden, dies ersetzt periodisch wiederholende Alias-Effekte mit Rauschen, was sich positiv auf die menschliche Wahrnehmung auswirkt.

Jittering beschreibt die Unterteilung des Pixels in gleich große Bereiche in Anzahl der Samples und positioniert ein Sample zufällig je Bereich.



Pixel Shading

Siehe Pixel Shading

Distance Falloff: Entfernte Pixel werden relativ dunkler dargestellt und wirken so natürlicher

Bloom: Simulation eines Sensorartefakts, da Linsen nicht perfekt fokussieren können, beeinflussen sehr helle Lichtquellen die Nachbarregionen des Sensors.

Lens Flare: Sichtbare Reflexion und Streuung des Lichtes im Linsensystem.

Merging

Siehe Merging

Sichtbarkeit: Mehrere 3D-Objekte erhalten durch Projektion gleiche 2D-Koordinaten und verdecken sich so gegenseitig, der am Auge nächsten liegende Punkt ist sichtbar. In einer Szene mit n Polygonen können n^2 sichtbare Fragmente entstehen.

Painters Algorithmus

Die Idee beim Painters Algorithmus ist, dass Polygone so dargestellt werden wie durch einen Maler also Back-To-Front. Hierfür müssen die Polygone erst nach ihrem Abstand sortiert werden und dann Back-To-Front gezeichnet werden.

Das Sortieren ist aufwändig $O(n \cdot \log n)$ und jede Änderung erfordert neues Sortieren und gegenseitiges Durchdringen wird nicht korrekt dargestellt. Unterteilen dieser Primitiven erhöht die Komplexität auf $O(n^2)$. Die Binary Space Partition (BSP) erlaubt effizienten back-to-front-order Durchlauf, unabhängig vom Augpunkt.

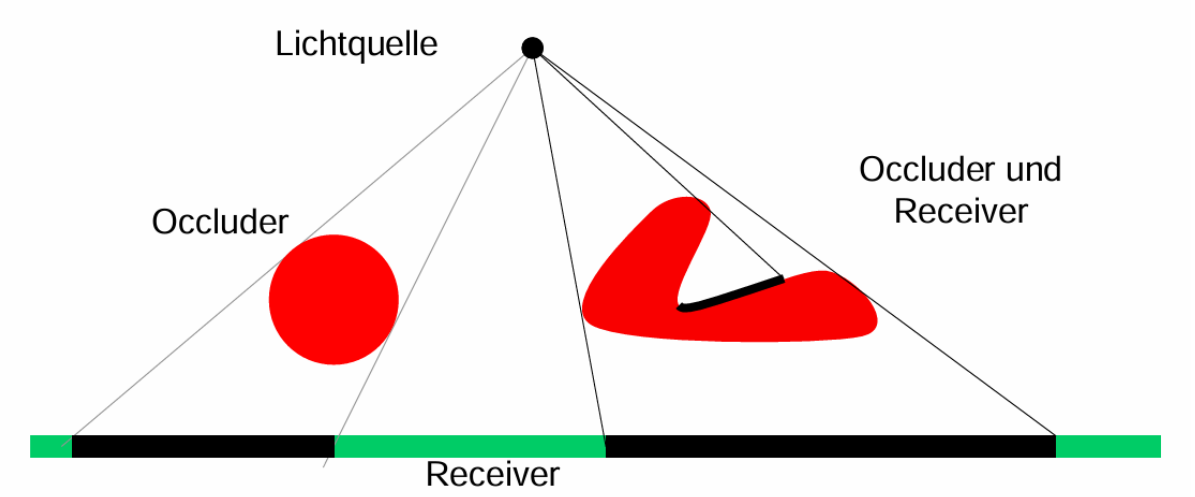
z-Buffer

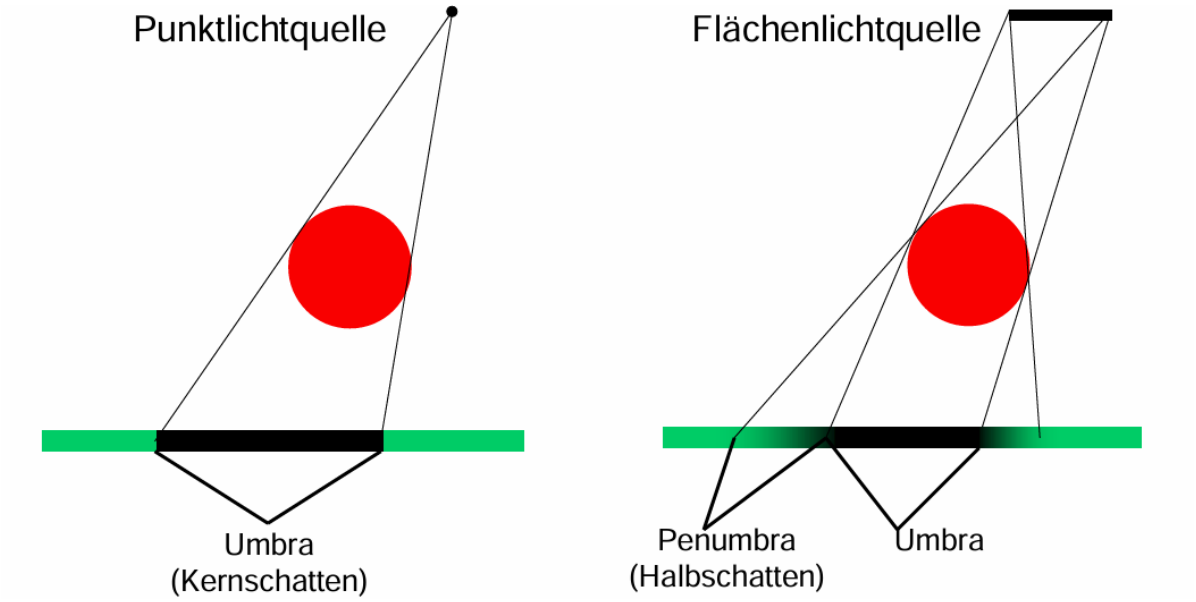
Beim z-Buffer wird das Sichtbarkeitsproblem auf Pixelebene gelöst. Der Algorithmus von Catmull und Strasser speichert für jedes Pixel einen Tiefenwert (z-Wert), dann wird für jedes dem aktuellen Objekt entsprechende Pixel überprüft, ob es näher am Augpunkt liegt und die Daten entsprechend aktualisiert.

Dieser Algorithmus löst das Sichtbarkeitsproblem mit Komplexität $O(n)$ aber benötigt mehr Speicher.

Schatten

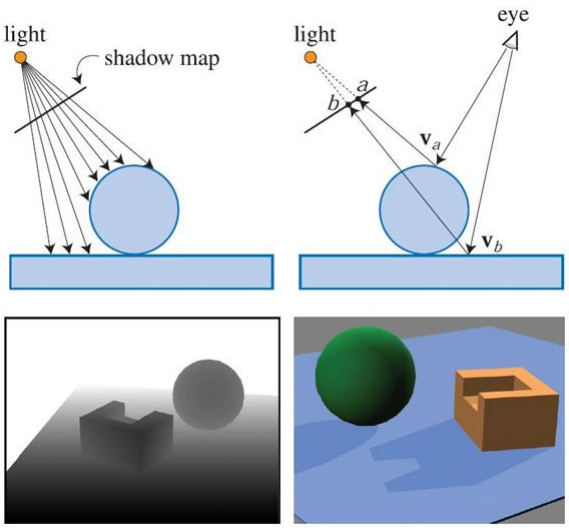
Schatten bieten Hinweise zu gegenseitiger Beziehung zwischen Objekten (Orientierung und Position). Oft werden weiche Schatten bevorzugt, da sie realistischer sind. Harte Schatten können als geometrisches Detail missinterpretiert werden aber lieber harte oder ungenaue Schatten als keine Schatten.





Shadow Maps

Shadow Maps sind ein Echtzeitverfahren und verwendet die Standard-Funktionen der Rasterisierung, dabei wird die Szene aus der Sicht der Lichtquelle gerendert und nur die Tiefenwerte in Schwarz-Weiß gespeichert.



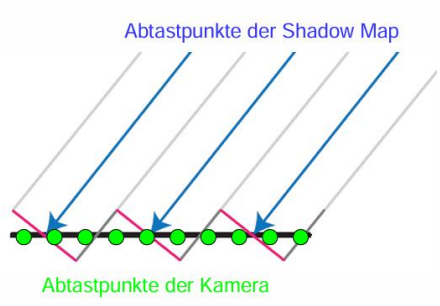
Bei der Rasterisierung der Primitive aus der Sicht der Kamera wird dann zu jedem Fragment die Distanz zur Lichtquelle berechnet und welcher Wert der Shadow Map zum aktuellen Fragment gehört. Dann wird die berechnete Distanz mit dem Tiefenwert der Shadow Map verglichen. Bei Abweichung größer als Epsilon befindet sich das Fragment im Schatten.

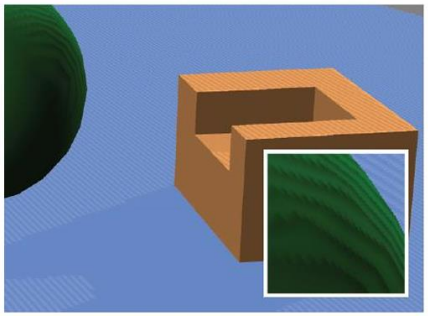
Vorteile: Shadow Maps können bei der Texturierung mit speziellem Texture Mapping angewendet werden und der Berechnungsaufwand ist linear zur Anzahl der Primitive.

Nachteile: Die Qualität des Schattens ist abhängig von der Auflösung des z-Buffers (Anzahl Pixel und Präzision), dies kann zu Self-Shadow Aliasing führen (Tiefenwert eines Abtastpunktes ist Repräsentant für eine ganze Region). Auch Treppeneffekte können entstehen.

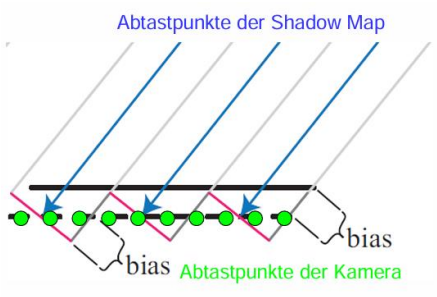
Self-Shadow Aliasing

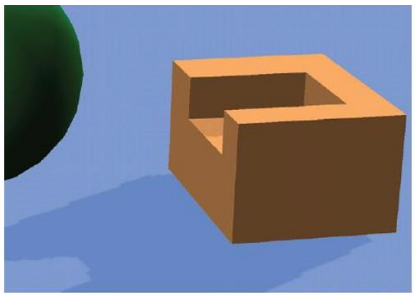
Da die Tiefenwerte nur diskret vorliegen und die Abtastpunkte von der Kamera und der Lichtquelle selten übereinstimmen entstehen beim Vergleich fälschlicher Wiese zu großen Differenzen.





Eine mögliche Lösung ist, ein Bias in der Shadow Map abzuziehen, aber kann andere Artefakte hervorrufen, z.B. Schwebeeffekt.

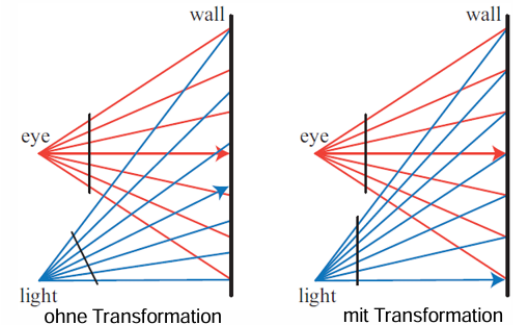


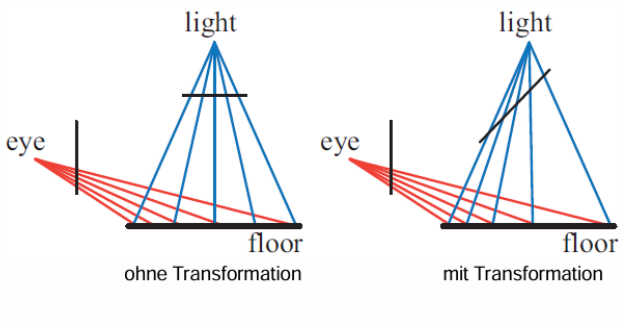


Treppeneffekt

Der Treppeneffekt tritt auf, wenn ein Texel der Shadow Map eine große Anzahl Pixel überdeckt, durch z.B. zu geringer Shadow-Map-Auflösung, Perspektivischen Aliasing oder Projektives Aliasing.

Eine mögliche Lösung ist, die Shadow-Map-Auflösung zu erhöhen oder das View Frustum der Lichtquelle auf sichtbaren Szenenausschnitt anzupassen. Idealerweise überdeckt ein Texel der Shadow Map einen Pixel in der Szene. View Matrix und Projection Matrix der Lichtquelle können angepasst werden, dies reduziert perspektivisches Aliasing.



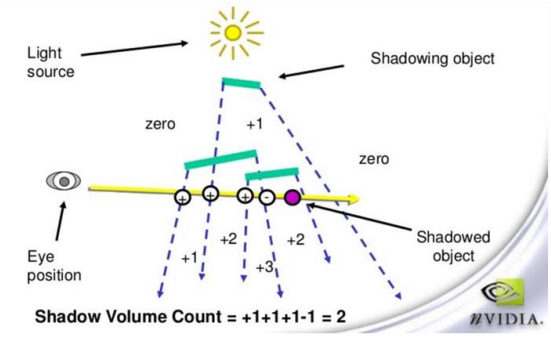


Shadow Volumes

Shadow Volumes sind ein alternatives Echtzeitverfahren, bei dem ein Volumen von Schatten für jedes Dreieck vom Licht aus erzeugt wird. Jedes Dreieck erzeugt drei Quads (Normalen N zeigen nach außen).

Man verfolgt den Strahl vom Augpunkt durch einen Pixel in die Szene bis zu einem Schnittpunkt, mit den Normalen der Quads kann man zwischen dem Betrachter zu- und abgewandten Quads entscheiden. Für jeden Strahl: starte mit einem Zähler = 0:

- Erhöhe den Zähler, falls der Strahl ein zugewandtes Quad passiert
- Reduzier den Zähler, falls der Strahl ein abgewandtes Quad passiert
- Ist der Zähler größer 0, dann befindet sich der Punkt im Schatten



Shadow Maps mit Stencil Buffer

- Stencil Buffer auf 0 setzen
- **Erster Durchgang:** Szene mit ambienter Beleuchtung zeichnen (Color Buffer und z-Buffer werden normal beschrieben)
- z-Buffer und Color Buffer beschreiben ausschalten, z-Buffer Tests aber weiterhin durchführen
- **Zweiter Durchgang:** zugewandte Quads zeichnen, dabei die Einträge für jedes gerasterte Pixel im Stencil Buffer um eins erhöhen (z-Buffer Test durchführen, damit nur „sichtbare" Quads den Zähler beeinflussen)
- **Dritter Durchgang:** abgewandte Quads zeichnen, dabei die Einträge für jedes gerasterte Pixel im Stencil Buffer um eins verringern
- Nun enthält der Stencil Buffer den Zähler für die Sichtstrahlen durch die Shadow Volumes
- **Vierter Durchgang:** Szene mit diffuser and spekularer Beleuchtung zeichnen für Pixel mit 0-Einträgen im Stencil Buffer (Bereiche im Licht)

Probleme von Shadow Maps

- Zähler muss angepasst werden, falls die Kamera in einem Shadow Volume liegt
- Falls ein Quad die near plane des view frustum schneidet, wird es abgeschnitten (clipping in der Geometrieverarbeitung), was den Zähler fälscht
    - Ausweg: capping (an near plane)
    - Anstatt ein Teilprimitiv (hier außerhalb liegenden Teils eines Quads) wegzuwerfen, wird es parallel zur clipping plane gelegt

Bisherige Zählweise nennt man **z-pass** Methode: „Sichtbare" Quads (bestehen z Buffer-Test) verändern den Zähler.

Z-Fail

z-fail: Verändern des Zählers nur für Pixel, welche den z-Buffer-Test nicht bestehen, „Verdeckte" Quads und reduzieren des Zählers für zugewandte Quads, erhöhen des Zählers für abgewandte Quads, aber Capping notwendig!

Vorteile: Alle Shadow Volumes vor einer Oberfläche beeinflussen nicht den Zähler (Shadow Volumes, die die Kamera umgeben), dadurch werden beim z-fail Verfahren die Probleme im Zusammenhang mit der Kameraposition innerhalb eines Shadow Volumes behoben.

Nachteile: Inverses z-pass-Problem, beim z-pass können Shadow Volumes die near plane durchdringen und den Zähler verfälschen, beim z-fail besteht das Problem bei der far plane. Die Shadow Volumes müssen geschlossen sein, analog zu capping an der near plane. Heißt Depth Clamping an der far plane, bei der z-fail Methode müssen Schatten werfende Primitive (Occluder) ebenfalls im Stencil Buffer gerastert werden, damit der Zähler korrekt ist.

Trick mit dem Unendlichen: Alternativ zum Depth Clamping kann die far plane in das Unendliche gesetzt werden. Quads der Shadow Volumes müssen ebenfalls in das Unendliche extrudiert werden (mit homogener Koordinate w = 0). Die Projektion ins kanonische Sichtvolumen ist dann beschrieben durch:

$$P_0 R_\infty = \begin{pmatrix} \frac{2n}{r-1} & 0 & -\frac{r+1}{r-1} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & 1 & -2n \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Dadurch wird die Präzision kaum beeinflusst, es ist also kein Depth Clamping mehr notwendig, denn die in das Unendliche extrudierten Quads transformieren korrekt. Z-pass ist schneller und sollte verwendet werden, wenn die Kamera nicht in einem Shadow Volume liegt, aber Z-fail ist robuster

Merging Volumes: Falls zwei der Lichtquelle zugewandten Dreiecke eine Kante teilen, werden dort ein zu- und ein abgewandtes Quad erzeugt, welche sich gegenseitig aufheben

Silhouettenkanten: Aus Sicht der Lichtquelle heben sich alle inneren Kanten gegenseitig auf, nur die Kanten der Silhouette tragen zum Shadow Volume bei. Diese zu finden reduziert viele nutzlose Quads der Shadow Volumes und hebt die Performanz drastisch an.

Shadow Volumes Vor- und Nachteile

Vorteile:

- Können auf jeder GPU mit Stencil Buffer berechnet werden
- Sind nicht Bildbasiert (keine zusätzlichen Texturierungsberechnungen)
- Umgehen das Abtastproblem der Shadow Maps ->Scharfe Schatten überall ohne Treppeneffekte
- Können erweitert werden, um weiche Schatten zu berechnen

Grenzen:

- Transluzente Occluder sind nicht möglich
- Primitive mit teils transparenten Texturen werfen keine korrekten Schatten
- Rasterisierung wird zum Bottleneck
    - Quads der Shadow Volumes belegen viele Pixel und liegen vielfach übereinander
    - Berechnungsaufwand variiert stark und ist schwer vorherzusagen