Rendering VO SS 2006 (186.101)

Version: 1.3
Zusammenfassung: Martin Tintel (mtintel)

Allgemeines zur VO

- Unterlagen: Folien werden unter http://www.cg.tuwien.ac.at/courses/Rendering/VO.html ins Netz gestellt.
- Bei Interesse gibt es auch auf der Webseite weiterführende Bücher.
- Die Prüfung findet mündlich statt (siehe Webseite).
- Buchtipp von mir: „Radiosity & Global Illumination“ von Francois X. Sillion und Claude Puech (das es bei Ebay oft um paar Euro gibt (zahlte um die 3 Euro dafür))

Renderingverfahren im Überblick (von mir zusammen gefasst)

Eine kurze Zusammenfassung von mir aus einem **Maya Buch** da ich denke dass das auch sehr interessant ist, kurz erläutert um was es bei welchem Verfahren geht, und welche Vor und Nachteile sie haben!

Raycasting ist einfacher als Raytracing, da er nur Lichtreflexionen berücksichtigt und die Modelle schattiert. Das Problem aber ist bei Raycasting, dass es weder Lichtbrechungen, noch physikalisch richtige Spiegelungen berechnen kann! Raycasting kennt nur Shadow Maps, die lediglich eine homogen Schatten und Ausmaß festlegen können!

An sich ist es in den meisten Fällen kein Problem, außer man verwendet lichtdurchlässige Flächen („Transparenz“) dann muss man zu **Raytrace** greifen, da nur er es kann (im Gegensatz zu Raycasting).

Bei **Global Illumination** (GI) versteht man die Methode der indirekten Beleuchtung. Dabei wird Licht in Photonenenergie und Anzahl der Photonen angegeben. In der Natur hat Licht eine bestimmte Energie und wird von jedem Objekt in der Umwelt reflektiert. Dabei wird je nach Art des Materials ein Quantum des Lichtenergie absorbiert. Das Licht oder die Photonen prallen so oft von den Oberflächen der Körper ab, bis sie komplett absorbiert sind. Die Berechnung dauert sehr lange, führt aber zu fotorealistischen Effekten.

Caustics können oft im Zusammenhang mit GI gemacht werden, und erzeugen Glanzlichter auf Objekten, da sie dort gebrochen werden und somit reflektiert werden (z.b. Glas Wasser am Tisch oder ein mit Wasser gefülltes Swimming Pool im Sommer)

Final Gather: Ist eine Methode des GI und kann z.b. im Renderer Mental Ray hinzugeschaltet werden. Es erzeugt sehr diffuse Szenensituationen, wo geringe oder nur langsame Wechsel der indirekten Beleuchtung entstehen, also typische Außenszenen. Das Verfahren ist auch extrem rechenintensiv!

Bei vielen 3D Programmen (wie z.B. bei Maya) bzw Renderer kann man auch oft Raycasting bzw. Raytracing gleichzeitig machen, indem man sagt, dass ein bestimmtes Objekt bzw. Material mit Raycasting oder Raytracing berechnet werden soll, ob ein spezielles Licht einen Depth Map Shadow oder einem Raycast Shadow erzeugen soll.

Kurze Kunstgeschichte Einführung

- Antike: Realistische Bilder und handwerklich hoch entwickelt.
- Mittelalter: ganz bewusst Abstraktion auf hohem Niveau und wollten nicht realistisch (war aber bei einigen vorhanden) malen da alles für sie eine Bedeutung hatte.
- Renaissance: „Wiedergeburt“, Antikekunst neu entdeckt (Verzerrung, Schattenwurf,…)
- 19. Jahrhundert: „Kunst“ ist Ausdruck und nicht realistische Abbildung (da dann eigentlich das selbe wie Fotografie).
- 1848: schon schwarz- weiß Fotos
- 1910: schon Farbfotos

Computergrafik und „Realitätstreue“

- Photoshop: Bilder sind vorher schon realistisch und werden nur manipuliert.
- 3D Programme: „glaubwürdig“, muss NICHT korrekt sein!
- 3D Spiele: virtuelle Welt, Frame Rat wichtiger als Realismus (der einzelnen Frames) (Anmerkung: VOs von Michael Wimmer zu dem Thema!!!)
- 3D Renderer: Genau wissen wie es dann aussehen wird z.b. CAD Datei und dann rendern um zu wissen wie es dann in der Realität 100%ig aussehen würde (z.b. in einem speziellen Raum mit speziellen Sonnenlicht, ein bestimmtes Leder in einem neuen Auto aussehen wird, Eigenschaften haben wird und rüber kommen wird (verbindliche Simulation da man Zusicherung haben will das es dann so 100%ig stimmen wird)).

Oft soll ein Rendering realistisch (physikalisch korrekt) aussehen aber der Schatten z.b. soll anders aussehen=> muss vom Renderer unterstützt werden (z.b. bei Pixar war das Problem als viel „physikalisch“ korrektes eingebaut wurde, dass viele sich beschwerten das man dann nicht mehr Schatten und ähnliches machen kann wie man es will sondern es eben nur „physikalisch korrekt“ geht…)

Kajia- mathematische„Beschreibung“ der Rendering Grundüberlegung

Kann man nicht „ausrechnen“ in normalen Sinn sondern ist eben mathematische Beschreibung damit man das Problem ein Mal unmissverständlich aufgeschrieben hat.
Beschreibt die Lichtausbreitung in einer Szene. Wenn man mit dem Augen in eine Szene schaut auf einen bestimmten Punkt, wie hell ist dann dieser Punkt?
g wenn z.b. ein Punkt nicht sichtbar ist da er verdeckt ist=> gleich 0 da man ihn ja nicht sehen kann.
rho: Oberfläche
Eigentlich eh alles logisch da es genau so ist wie in der Realität nur schaut die Gleichung an sich „kompliziert“ aus! In der normalen Alltags Sprache ausgedrückt wäre es total einfach zu verstehen, aber da der PC das ja nicht versteht, muss man es so machen und vor allem da man dann die mathematische Maschinerie verwende kann um es sich ausrechnen zu können.

Renderingverfahren im Überblick

- Scanline: keine Spiegelungen, kein richtiges Licht
- Raytracing: Oberflächen nicht richtig, Rekursion nur für perfekte Spiegel (harte schatten, perfekte Spiegelung)
- Radiosity: alles Geometrien, aber Polygone, Integrale nur für Diffuse Oberflächen und auch bei Oberfläche an sich. Aber total konfus und daher oft „viele Klötze“ im Raum

- Global Illumination: sehr schön da die Kanten und Ecken schön sind, nur geht das ohne aufwändigen Gewichtungen (unter anderem) nicht!
- Photon Tracing: Integral teilweiße da es vom Algorithmus abhängt. Ist so wie z.b. GI, nur umgekehrt. . In der Realität kaum einsetzbar.
- Path Tracing: Extrem aufwändig da man nicht weiß wie man am besten das allgemeiner machen kann. KEINE Fakelights => toll heutzutage da man sich nicht um Algorithmus kümmern muss sondern einfach die Szene hinstellt und dann los rendern lässt.
- Light Tracing ist so wie Path Tracing, nur das man von der Lichtquelle aus geht. In der Realität kaum einsetzbar.
- Shooting Algorithmen an sich nicht sinnvoll, sonder nur in Zusammenhang mit anderen Algorithmen wo man z.b vom Auge aus den Strahl schickt=>
- Shooting Type Random (Walk): Schießen Strahlen von der Lichtquelle aus.

Problem der Darstellung (Tone Mapping)

Problem das viele physikalische Bilder nicht angezeigt werden können z.b. wenn Licht hell ist (Sonnenaufgang/Sonnenuntergang) und da man die total helle Sonne nicht anzeigen kann sondern nur „dunkler“, muss man den Rest weniger heller machen, damit es dann verhältnismäßig hell aussieht! Bei Grafikkarten gibt es z.b. HDR oder bei den Monitoren gibt es auch schon welche, die es anzeigen können, aber sehr teuer!

Farbkonstanz

Wenn eine Szene beleuchtet wird und ein Mensch es sieht, ist er auch in der „Beleuchtung“ drinnen oder ist er außerhalb und bekommt somit die Farbe anders mit?
Time Depending Tone Mapping: wenn das licht eingeschaltet wird, und man geblendet wird, dann sieht man alles viel heller und es dauert bis das Auge sich daran gewöhnt!

Global Illumination (GI)/ Raytracing/ Raycasting/

- Inversion: „Abkürzungen“ in der Formel damit es übersichtlicher ist und man einfacher mathematische Verfahren anwenden kann. T ist aber nicht ein einfacher Buchstaben und hat eine spezielle Eigenschaft=> macht Probleme und ist nicht brauchbar. Ist einziges Verfahren was nicht abhängt z.b. auch anwendbar wenn Oberflächen Energie abgeben/ produzieren=> ist aber eben nicht effizient
- Expansion: Ursprüngliche Gleichung => man kann immer wieder einsetzen=> wenn man es n mal macht, kommt eine Neumann Serie heraus=> es gibt einen Limes und es konvergiert.

Rekursive Substitution macht genau das was das Auge macht (bis der Strahl über so viele absorbierende Oberflächen geschickt wurde, damit das Ergebnis nicht mehr relevant ist)

Raytracing: Photorealistischer Algorithmus
Raycasting: Strahl wird mit etwas aus der Szene geschnitten

Gathering Expansion Steps

Man integriert über die gesamte Halbkugel numerisch, indem man das Integral (numerisch und hochintegral, da man nicht nur über Hemisphäre sondern über alles was noch kommt, also den gesamten Pfad, integriert) über ein Monte Carlo Integral macht. Monte Carlo Integral hat aber einen „schlechten Namen“.

Numerisches integrieren, gewisse Menge an Samples nehmen und eine gewichtet Summe machen (man nähert sich über „Kasterln“ an und zählt sie zusammen).

Problem ist das es exponential ansteigt, und man deswegen den Integrant nicht regelmäßig abtasten will, da ansonsten der Aufwand um das vielfache ansteigt. Wenn man z.b. einen Lichtstrahl hat der auf eine Fläche stößt, und dann die Halbkugel macht, und man dann z.b. die Halbkugel in 10° Schritten macht, und der Lichtstrahl tritt aus, muss man es auf der Fläche wo dann wieder der Lichtstrahl auftritt, es ebenfalls machen.

Man nimmt also zufällig verteilte Schätzwerte. Die Frage ist, wieviele Samples man nehmen muss, damit man eine bestimmte Genauigkeitsschranke unterschreitet.

Numerisches Integrieren geht nur bei niedriger (????) deterministisch, und bei höheren muss man Monte Carlo Integration machen.

Monte Carlo (MC) Integral

Monte Carlo heißt immer, dass man Zufallszahlen zum integrieren nimmt (sehr vereinfacht ausgedrückt). Die Frage ist welche Zufallszahlen man verwenden soll, die man auch auf den ersten Blick nicht so einfach sehen kann! Genauer gesagt lautet die Frage: welche Punkte soll man nehmen.

Man hat eine Funktion die man integrieren will und es gibt 2 Strategien:

- Man verteilt die Punkte so, sodass die Samplepunkte so aussehen wie der Integrant. Wo er groß ist schießt man viele, und wo er gering ist schießt man wenige.
- Nicht regelmäßige, aber möglichst Gleichverteilte Punkte zu verwenden.

Beide Strategien haben ihre Vor und Nachteile und somit auch ihre Einsatzgebiete.
Verwendet auch keine richtigen Zufallszahlen (außer am Anfang beim Testen des Renderer)

Diskrepanz kann man auch ausrechnen (?????)

Zufallszahlen

Zufallszahlen werden mit verschiedenen Algorithmen und mathematischen Methoden erzeugt. Echte Zufallszahlen (auch die physikalisch erzeugt werden) haben keinen Mindest/ Maximal Distanz! Man will aber oft, dass sie nicht beisammen liegen, also einen bestimmten Abstand haben, aber trotzdem „Zufallszahlen“ sind=> sind also keine „richtigen“ Zufallszahlen mehr.

- Monte Carlo verwendet man auch keine richtigen Zufallszahlen (außer am Anfang beim Testen des Renderer)
- QMC (Quasi Monte Carlo): Kein Dropout Zufallsgenerator wie random (???)
- Hammersley Sequenz: wenn man weiß wieviel Punkte man haben will
- TMS Netze: Ist gut für 2D Punktemengen

Wenn man eine Punktemenge in einem Quadrat machen will, braucht man 2 Generatoren: einen für x und einen für y. Man verwendet Primzahlen, hat aber das Problem das man bestimmte nehmen muss und sich die „merken“ muss (lebt davon das man schön angeordnete Pools hat die einander nicht in die Quere kommen)!

Rand()… ist nicht Threadsave! Wenn man also 2 Threads hat die gleichzeitig laufen und

Zufallszahlen haben will, und auf einen Zufallsgenerator zugreift (der ja wissen muss was schon für Zahlen „gewürfelt“ wurden) dann machen die beiden sich Probleme, und können auch statt üblich, Zahlen zwischen 0 und 1, plötzlich auch andere Zahlenwerte haben und somit schaut dann alles total anders und falsch aus.

Raycasting

Raycasting ähnlich wie OpenGl, und hat den Vorteil das Strahlenbasierte Algorithmen besser sind als räumlich basierte. Daher kann man z.b. eine Szene mit tausenden Sonnenblumen schnell berechnen, und das auch noch in Echtzeit.

Raytracing

Raytracer kommen langsam auch bei Echtzeitberechnungen vor.
Ist ein Hybridalgorithmus: wenn man auf einen perfekten Spiegel mit dem Lichtstrahl kommt, macht man weiter und wenn nicht, bricht man ab! => Problem das in der Welt es perfekte Spiegel nicht gibt und deswegen sehr unrealistisch sind, genauso wie bei Lack, Papier,… die ebenfalls nicht perfekt spiegelnd sind!

Funktioniert so, dass man einen Punkt berechnen will, und genau dort einen Strahl durch die Szene schickt, und schaut was getroffen wird=> Hitliste. Dann nimmt man alle Lichtwellen her, und schaut welche Auswirkungen sie haben (bei vielen Lichtern ist das sehr aufwändig, und deswegen schaut man oft, welche Lichter überhaupt relevant sind, wie stark sie sind, was dominiert, um Rechenzeit zu sparen). Dann geht der Strahl weiter und von dort geht es rekursiv weiter („wo trifft dann der Strahl auf?“). Ein sehr mächtiger Algorithmus, da wenn man den Algorithmus „richtig“ nimmt, das Ergebnis sehr echt aussieht.

Kann aber nicht alles, was wichtig ist in der Szene her nehmen! Erkennt man daran das es oft so aussieht, wie wenn man alles perfekt hätte mit einem kleinen Lack darüber. Oder das Oberflächen (z.b. ein Haus im Wasser) perfekt gespiegelt wird und nicht „verzerrt“.

Distributed Raytracing

Oder auch Distribution Raytracing genannt: Man schießt einen Strahl in die Szene, und bei allem was getroffen wir, gibt es eine hemisphärische Abtastung die rekursiv ist=> Problem ist das die meisten Strahlen nichts bringen, da eigentlich nur die erste direkte Eben relevant ist, und die der zweiten, aber danach ist es kaum mehr relevant. Ist sehr aufwendig und auch heute kaum machbar, obwohl es in den 80iger Jahren entwickelt wurde! Der Algorithmus an sich ist sehr leicht und nahe liegend, aber kaum ausrechenbar wegen den langen Rechenzeiten.

Path Tracing

Wird nicht für Production Works verwendet, ist aber trotzdem von Bedeutung, da es der effizienteste Algorithmus ist, der auch „richtig rechnet.

Man geht vom Auge aus, und verfolgt den Strahl bis er verloren geht. Wichtig ist, dass der Pfad nicht geteilt wird! Wenn man also in die Kugel die transparent ist einen Strahl schickt, wird „gewürfelt“, ob er den Hauptstrahl verfolgt oder andere!

Path Trace steckt gleich viel „Energie“ in die erste und 2te Ebene.

- Möglichkeit 1: Man schießt viele Strahlen hinein, und sehr schnell, da der Algorithmus nur rekursiv geht und nicht mit vielen „If Schleifen“ voll gestopft ist und daher recht schnell sein kann. Problem wenn nur eine Lichtquelle da ist, die auch noch sehr klein ist. Umso kleiner die Lichtquelle ist, umso schlechter die Qualität, da die Wahrscheinlichkeit sinkt, auf Lichtquelle (auf Oberflächen) zu treffen
- Möglichkeit 2: Auf jedem Auftrittspunkt schießt man zusätzlich ein Lichtquellenstrahl („Fühler“) drauf, hat aber das Probleme das man Energie „doppelt“ zählen kann=> man muss Rekursion weglassen, oder Lichtfühler, aber das geht nicht=> „Multiple Imporance Sampling“.

Path Tracer sind nicht schwer zu Programmieren, und die Gewichtung der Samples ist sehr wichtig, da man einen flüssigen Übergang braucht.

Bi- Directional Path Tracing

Die meisten Pfade bringen es nicht, aber im statistischen Mittel sehr wohl. Das Problem liegt darin, dass man den Algorithmus schwer nachbauen kann, da die Infos nicht öffentlich sind und man kaum Infos über den Aufbau und Gewichtung finden kann! Es steht eigentlich nirgendwo drinnen, wie es genau geht sodass man es dann wirklich nachbauen kann.

Metropolis Light Transport

Monte Carlo und Metropolis Light Transport wurden eigentlich in den 40igern in den USA entwickelt für Atombombenprogramm. Problem ist das es ursprünglich eine Ad- Hoc Lösung war.

Metropolis Light Transport: Da die meisten Connections nichts bringen, oft nicht mal eine einzige und es viel wichtiger ist einen Pfad zu finden, wo viel Licht ist, sollte man eher das „finden“.

Pfad wird mutiert/ geschüttelt. Metropolis „schlecht“ da es zum Starten sehr viele Samples nehmen muss, wobei er dann die meisten nie brauchen wird! Bringt aber beinahe rauschfreies Bild zustande, im Gegensatz zum bidirektionalen Tracer und kann somit gut mit allem umgehen! Ist aber schwer zu implementieren und für die meisten Szenen unnötig, da es dort wie ein bidirektionaler ist, aber um einiges langsamer.

Radiosity

Großes Lineares Gleichungssystem wird aufgestellt und gelöst.

Vektor der Radiosity wird mit B abgekürzt, was an sich keinen Sinn hat, aber beim ersten Paper über das Thema so gemacht wurde, und dann immer weiter beibehalten wurde.

Radiosity war das erste Verfahren, dass ziemlich einfach und zuverlässig war=> deswegen auch sehr beliebt bei Architekten, teilweiße bis heute!

Radiosity beschäftigt sich schon seit langem mit Beleuchtung- Repräsentation. Man will es aber nicht in viele Kästchen auflösen, sondern lieber in bessere, da es viel Rechenzeit spart. Eine der Möglichkeiten sind Basisfunktionen, sodass man versucht für jede Basisfunktion es zu lösen. Nur kann man dadurch auch Probleme bekommen, da die Berechnung und das Darstellen unterschiedlich ist, und da man beides zusammenführen und interpolieren muss,

was bei komplizierten Funktionen und vor allem Komplexen Meshes sehr Problematisch sein kann.

Das Mesh für Radiosity

Meshes sollte man mit Gefühl machen und wenn es hochaufgelöst ist, man sehr dünne und lange Übergänge vermeiden soll, und besser grobe und dafür gut abgestufte Übergänge machen sollte.

Cornell Box

Die Box hat den Vorteil, dass man es in der Realität leicht bauen kann, und mit einer geeichten Kamera aufnehmen kann und schauen kann, wie sehr das Rendering der Realität ähnelt. Normalerweise ist es eine Box, in der eine Lichtquelle ist und meistens auch Körper (Kugel oder Rechtecke) drinnen sind.

Form Faktor

Form Factor Fij: Kann man beibehalten, auch wenn man das Licht ab und anschaltet bzw. Patches als Lichtquellen definiert. Man darf aber keine Objekte bewegen, da sich dadurch es ändert.

Am Form Factor ist es sehr interessant, wenn es um die Projektion auf eine andere Fläche geht! Die Projektion eines Patches (i) auf j ist eigentlich eine Projektion auf eine Einheits-Halbkugel. Es geht nicht nur darum, wieviele Strahlen durchkommen, sondern auch welche Winkel sie haben und welchen Formfaktor.

Monte Carlo Form Factor: ist ineffizienter als Raytracing, aber weniger Fehleranfällig auf perverse Sonderfälle, also es funktioniert immer, auch wenn es sehr ineffizient ist.

In der Praxis ist am wichtigsten das **Hemicube Verfahren:** man will einen Punkt wissen, baut darum herum einen Cube auf, und nimmt 5 Kameras her, die in alle Richtungen schauen und misst wieviel % von dem Patch über einem beansprucht wird (???)

Hemicube Aliasing: wenn es (Hemicube Aliasing) ein zu geringe Auflösung hat, auch wenn die Punkte an sich hoch genug aufgelöst sind, schaut es nicht gut aus. Wird aber „gerne“ in Echtzeitrendering verwendet.

Jakobiiteration

Alle Bi (???) will man haben und alle E sind das Licht. Die Patches holen Licht und das Licht prallt dann auch auf Flächen die schon Licht abbekommen haben.

Hierarchische Radiosity

Wenn man die Anzahl der Patches erhöht, erhöht man die Rechenzeit quadratisch. Deswegen unterteilt man bei der Schattenkante eine sehr hohe Auflösung, aber ansonsten nicht, und es kommt drauf an was man genau „lösen“ will um die feine Auflösung einzuschalten oder nicht! Man baut auf jedem Patch eine Hierarchie auf, die man nicht immer auf ein Mal braucht, und schaltet dann zwischen den „Modis“ hin und her, was man halt braucht.

Push und Pull: man wirft die ??????

Global Lines

Man schießt Strahlen durch die Szene und man rechnet sich den Lichttransport aus sprich alle Interaktionen des Strahles. Der Vorteil ist, was auch teuer von der Rechenleistung ist, welche Schnittpunkte es gibt. Bei Animationen nimmt man die Objekte her und bewegt das Objekt durch die Szene. Bei jedem Frame nimmt man eine Instanz des Objektes her. Man zählt mit in welchem Frame wie der Strahl ist und kann somit die Interaktionskette auflösen. Der Vorteil ist, dass wenn sich wenig verändert, man quasi nur ein Mal Raycasten muss, und dann nur die einzelnen, bewegten Objekte noch berechnen muss! Ist also ziemlich effizient. Trotzdem nicht weit verbreitet, da es kein Amerikaner entwickelt hat und somit in den USA nicht „bekannt" ist. Weiters gibt es ein leichtes Rauschen. Vor allem wurde es erst Ende der 90er Entwickelt. Da es aber eine Große Ersparnis bei Animationen hat, könnte es vielleicht wieder ein Mal aus der Schublade raus kommen für Animationsrendering.

Photon Tracing und Photon Maps

Photonen werden in die Szene geschickt, und auf einer Photon Map aufgezeichnet.

Photon Maps: Extrem effizient für Caustics, hat aber eine sehr große Datenstruktur! Bietet aber die Möglichkeit, die Photonen zu trennen in normale Photonen, Caustics Photonen und Schatten Photonen! Photon Maps sind auf dem Blatt „besser" (als andere) aber ist auch kein Allheilmittel und hat seine Probleme. z.b. muss man sehr viel Vorberechnen, braucht sehr viel Speicher, und man muss ebenfalls sehr viele Photonen „schießen".

Caustics Photonen: Photonen die durch das Glas gehen, die Brechung berechnen, den veränderten Schatten durch die Brechung,… Bei jedem Photon wird auch berechnet, aus welcher Richtung es kommt!

Global Photon Map: Alle Photonen die nicht einen Schatten werfen oder Caustics erzeugen werden verwendet, um zu steuern ?????

Shadow Photon Map: beim ersten Hit wird es als Licht gekennzeichnet, und danach dann immer als „Schatten".

Light Maps: Man hat Zähler für die „Patches", ähnlich wie GI, nur ist der Unterschied, dass mit den aufgezeichneten Stats nicht gerechnet wird und das es direkt auf den gekrümmten Oberflächen aufgebracht ist, und nicht wie bei GI designiert. Sind also Abziehbilder für das Licht. Kann auch Caustics ausrechnen. Die Auflösung der Light Maps ist aber unten (Schatten eines Diamanten z.b.) um einiges höher, und die Auflösung der Wand unnötiger weiße auch recht hoch! Es wird die Helligkeit berechnet, indem man schaut, wieviele Photonen auf eine Oberfläche auftreffen und dann muss man sie noch in Relation zu den anderen Elementen stellen, da die Wahrscheinlichkeit, eine kleine Fläche zu treffen, um einiges niedriger ist als eine große Fläche zu treffen.

Das Problem ist, dass wenn man Photonen sucht (bei Caustics ????), man nur die Photonen dann ausschließen kann, die nicht den selben Vektor haben, oder auf einem anderen Objekt sind, aber wenn man eine Wand hat die im 9 Grad Winkel auf einem Boden steht, und man such am Boden nahe der Wand Photonen, kann man die auf der Wand ausschließen, aber nicht die die hinter der Wand am Boden sind, obwohl man die eh nicht sehen kann! Bei einer normalen Oberfläche ist es aber sehr gut und recht schnell und funktioniert einwandfrei!

Photonen haben den Vorteil, dass man volumetrisches wie Rauch, Ausbreitung von Licht in rauchen nicht nur aufzeichnen, sondern wiedergeben!

Unbekannt ist, wie genau die Aufsammlung der Informationen erfolgt.

Problemlösung für das Photon Problem wäre es, den Strahl aufzuzeichnen und im Dualen Space zu lösen.

Steinersche Römerfläche: schön zum Rendern (wegen Caustics) und für ??? (Beruf)

Photon Tracing ist ein gutes Verfahren, egal ob man Light Maps oder Photon Maps (~1998, 1999) verwendet.

Bei einem Baum ist das Problem, dass wenn man Light Maps verwendet, man viel zu viele Flächen hat und dass das kaum möglich ist (Path Tracing ist auch nicht gut in dem Fall).

Das Problem beim Rendern ist, dass wenn man zum Beispiel eine Landschaft, Wald, Haus,… rendern will, die Wahrscheinlichkeit sehr gering ist, dass Licht auf das „Objekt“ fällt. Man könnte also nur das Licht „manipulieren“, sodass es auf das Objekt auftrifft, was wiederum das Problem ergibt, dass Monte Carlo Renderer das nicht gut vertragen.

Orientation Lightmaps (OL)

OL Prinzip: nicht nur Vereinfachung, sondern eigentlich total falsch. Da man es aber mit anderen Verfahren mischen kann, trotzdem sehr gut. Man kann zb. einen Christbaum in einer Wohnung rein stellen, kann dann dort sehr gut alles mit Photon Tracing machen, und somit alles „zum laufen bringen“. Man kann also recht einfach ein Objekt mit der Datenstruktur umhüllen, und beeinträchtigt somit nicht die restliche Szene.

Hat den Vorteil, dass es kaum Speicher brauch, man die Datenstruktur um mehrere Objekte hüllen kann.

Problem: das Licht wird verwischt und ist falsch, aber es funktioniert und ist erstaunlicherweise nicht sooooo falsch und daher recht gut.

Bei Animationen ist der Mensch viel anfälliger auf Artefakte.

Bei OL kann man auch Hierarchien machen und schauen, welche Zweige wieviele Photonen abbekommen. Man geht dann immer den Weg weiter rauf (den Pfad entlang) bis man auf genug Photonen trifft.

Man erkennt es aber daran, dass „gemittelt“ wird und somit helle Körper oft unten wo sie dunkel sein sollten, zu hell sind bzw. recht dunkle Gegenstände, wo aber Licht vom Boden auf die Unterseite des Objekts abprallt, nicht heller macht wie er es sollte.

Renderman

Hauptziel ist nicht das realistische rendern, sondern das es „echt“ aber vor allem wie ein 3D Comic aussieht. Könnte auch realistisch aussehen, aber das ist nicht das Ziel.

Software Shaders haben dort ihren Ursprung.

Renderman kann entweder der Szenenbeschreibungsstandard sein oder ein Renderer. Der Standard ist kostenlos, somit man rein theoretisch einen Renderer programmieren kann. Da aber fast 1 Jahre lang das niemand geschafft hat, hat Lary Carpment (???) eine Diplomarbeit darüber geschrieben, wurde bei Pixar aufgenommen, ist später dort ausgestiegen, gründet eine eigenen Firma und wurde von Pixar verklagt.

Pixar hat ein Patent auf Zufallszahlen in Computergrafik (Programmen)=> kann rein theoretisch einem klagen. Deswegen recht problematisch.

Renderman ist aus den 80igern, sehr verbreitet und vor allem sehr interessant, da man kaum was ändern musste, damit es in der „Jetztzeit" gut verwendet werden kann.

Das Grundlegende Verfahren von Renderman hat sich aber seit vielen Jahren nicht verändert.

Ursprungsüberlegung war: man will einen Kinofilm rendern, und wie kann man das am besten machen:

Komplexe Szenen: alle Geometrien sollten möglich sein zu rendern

Shaders wurden von Anfang an als notwendig erachtet, damit man von Anfang an die Szenenkomplexität erreichen konnte.

Raytracing kam nicht in Frage, da wenn man einen 2 Stunden Film in einem Jahr rendern will, man pro Frame nur 3 Minuten pro Frame hat=> kam also nicht in Frage sondern man musst ein besseres Verfahren finden.

Wichtig war aber, dass man Filterung und Anti Aliasing gut macht, da es nichst bringt, alles schnell berechnet zu haben und an sich gut aussieht, aber dann viele Kanten und Kasterln hat!

REYES Algorithmus: Man teilt die Objekte auf in Primitive, welche wiederum in Micropolygone (Alle Geometrien werden in Micropolygone zerteilt) zerteilt werden.

Vorteil: man kann es parallelisieren und somit recht schnell auf vielen Rechnern berechnen lassen bzw. es sequenziell abarbeite („Batch rendern"), sodass man auch sehr komplexe Szenen auf Rechnern mit wenig RAM rendern lassen kann.

Was Renderman auch nicht macht ist Clipping, da Polygone immer kleiner sind als ein Pixel (Polygone sind maximal ein halbes Pixel groß) oder sie eh schon weggefallen sind.

Man hat in Renderman 2 Arten von Texturen definiert (intern), damit es gut geht. CAT und RAT:

- CAT: ????
- RAT: ????

Um einen „Renderman" zu machen, braucht man nur 4 „Sachen" implementieren.

- ??? (siehe Folien „Geometric Primitives Routines")
- 2) Muss Micropolygone machen
- Muss splitten können
- ??? (siehe Folien „Geometric Primitives Routines")

RiPixelSamples(): umso mehr Samples, umso weniger Rauschen
RiPixelVariance(): ???

REYES Advantages: Kann mit Arbitrary (deutsch????) Nummern von Primitiven umgehen. ?????.
REYES Disadvantages: ?????

Vorteile:
- Kommt mit beliebigen Anzahl an Geometrien zurande.
- Ist „3D Zeichenprogramm“
- Hat sehr gute Subdivision

Nachteile:
- keine GI

Man sollte Komplexität in die Shader stecken, und nicht in die Geometrie.

Alle Aufrufe in Renderman beginnen mit RI (bei der Programmierung)

Echtzeit Renderer

Echtzeit Renderer waren lange Zeit unmöglich, aber heute schon teilweiße recht gut machbar.

Shader

Es gibt verschiedene Arten von Shadings Ein Oberflächen Shader bekommt andere „Inputs“ als ein Displacemet Shader und gibt auch anderes zurück

- Light Source Shader: wieviel Licht („Intensität“) und welche Farbe kommt auf einen Punkt.Man muss allen Objekten den Light Source Shader zuweisen, da es auch auf allen Objekten wirkt!!!
- Shadow Maps: weitgehen Automatisiert Reflections: man macht einen Würfel, rendert alle 6 Seiten und mapt dann das Ergebnis auf den Würfel

Spectral Rendering

Die Frage ist, wenn man einen Rendering Algorithmus anwendet, was man dann eigentlich für ein „Ergebnis“ ausrechnet… z.b. könnte die „Antwort“ RGB lauten.

Da das Licht sich aber aus einer Mischung von Frequenzen zusammensetzt, und das eigentlich nichts mit RGB zu tun hat ist das Problematisch!

Licht ist auch eine Welle, und diese Eigenschaft macht in der Regel nicht viel aus, da man kaum Effekte sehen kann, die nur auf die Wellenform zurück zu führen sind, kann man auch so simulieren lassen (z.b. das schimmern einer CD) und spielt somit keine Rolle in der Computergrafik. Eigentlich nicht mehr mit der „abgeschwemmten“ Rechnung dürfen Leute wie z.B. die Sendemasten, Handynetze,… arbeiten, da Mikrowellen noch mehr wellenförmig sind und somit der Fehler um einiges größer ist.

Colour Space based Renderer vs. Spectral Renderer

Es gibt Colour Space based Renderer und Spectral Renderer (wie Maxwell), wobei es keinen einzigen kommerziellen Spectral Renderer gibt (Maxwell wurde auch noch nicht fertig released (Anmerkung von mir: gibt es jetzt schon zu kaufen als 1.0!)).

Wenn man im RGB Raum rechnet, und nur eine Lichtquelle hat, kann man auch so extrem gute Ergebnisse erzielen, sodass Spectral Renderer sich eigentlich nicht auszahlen. Und mit mehreren Lichtquellen erzielt man auch sehr gute Ergebnisse, wenn man die Durchführung sehr gut macht!

Spectral Rendering: ???
Component Rendering: ???
RGB (Tristimulus) Rendering: ???

Das Problem ist, welche Stützstellen und wieviel man nimmt, da man oft Spektren wie bei Neonröhren, wo man sehr extreme Peaks (???) hat und somit kann man keinen guten und intelligenten Renderer machen.

Spectral hat den Vorteil, dass es genau ist und man genau vorhersagen kann, wie etwas unter einen bestimmten Lichtquelle aussehen wird. Hat aber den Nachteil, dass es in der Regel ein wenig länger braucht zu berechnen, aber z.b. wenn man es in Kombination mit GI macht, ist es kaum langsamer da es um die 3- 5% länger dauert… Weiters kann man recht schwer z.b. ein „ein wenig grüneres“ Bild erzeugen. Aber wenn man eine Datenbank hat mit genau den Lichtquellen(Messungen) die man haben will, interessiert das einem kaum.

RGB ist allgemein Bekannt und daher auch gut. Man kann nicht vorhersagen, wie es dann tatsächlich anschauen wird. In den meisten Fällen ist es ausreichend zu wissen, wie die Szene unter einen weißen Licht ausschaut, da man kaum mit färbigen Lichtquellen arbeitet und wenn ja, man nicht unbedingt es genau wissen muss, sondern nur recht grob.

Bei Innenarchitekten ist das Problem, dass man dem Kunden sehr genau im Vorhinein sagen/ zeigen will, wie es dann wirklich aussehen wird! Dadurch muss man Spectral Renderer verwenden, wenn man Bunte Lichtquellen hat, starke Peaks,….

RGB kann man verbessern, indem man schaut wann und wo der Fehler passiert, und man den Leuten sagt, was man da genau macht, sodass man von Haus aus Probleme vermeiden kann.

Rein theoretisch ist jede Lichtabstrahlung von der Wand/ Oberfläche die sich dann wieder mit einem anderen Licht(Farbe) vermischt ein Fehler, aber da hauptsächlich die eigenen Lichtstrahlen von der Quelle aus zählen, ist der Fehler kaum wahrnehmbar.

Punkt 2 (Perform Tristimulus RGB Rendering) und 3 (Apply white balance transform….)
??????????

Wenn man in einer Szene ist, dann stellt sich das Auge ein und würde die Farben nicht so wahrnehmen wie sie eigentlich sind, sondern anders (z.b. Glühbirnen schauen wenn man in der Nacht durch die Straße geht, und man sie in einem Haus leuchten sieht, „Geld/ orange“ aus, obwohl sie eigentlich weiß sind).Deswegen muss man es nachher alles umrechnen, sodass das Bild nicht aussieht, wie wenn man es „von außen“ sieht, sondern aus/in der Szene=> Weißpunktabgleich=> von Kries Transformation (for Chromatic Adaptation)

Sharp RGB vs. sRGB:

sRGB: Es gibt keinen vernünftigen Grund, einen Renderer für sRGB zu schreiben, außer das man schon in dem Farbraum ist
Sharp RGB umspannt einen viel größeren Raum und ist deswegen um einiges besser als sRGB

Picture Perfect: ????

CIE XYZ Farbraum: es kann nur positive Farbwerte geben.
Metamerism: Metamere Farben

Absorption: Je dicker Glas wird, umso intensiver nimmt das Licht die Farbe an des Glases. Im Farbraum funktioniert es trotzdem recht gut und von daher sieht man keine großen „Fehler".

Bei Brechungen wird der „undurchsichtige" (oder durchsichtige????) Bereich strichliert gezeichnet.

Sellmeiers Formel: ?????

Polarisierendes Licht

auch ohne schaut es glaubwürdig aus, aber mit schaut es „perfekter" aus. Bei Kristallen zb. schaut es auch ohne Polarisation glaubwürdig aus, auch wenn einiges an Farbe (z.b. bei Fassetten) nicht stimmt.

Beim Fotografieren mit einem Polarisationsfilter verschwinden die weißen „Lichtpunkte" auf Pflanzen und ähnliches.

Polarisation ist wichtig wenn man z.b. Objekte mit starkem Licht als Input versehen will oder wenn man Himmelszenen macht (wenn der Himmel sehr blau ist), da der Himmel unterschiedlich stark polarisiert ist.

Bei der Polarisation muss man nicht nur die Lichtstärke, sondern auch die xy Koordinatenausbreitung beachten.

Licht ist nichts anderes als Wellen die eigentlich nichts miteinander zu tun haben.

Wenn man mit Polarisation rechnet, ist nicht nur die Abschwächung wichtig, sondern auch die Phasenverzögerung (!!!)

Es gibt 2 verschiedene Daten- Typen: Intensität und Abschwächung

Strokes Vectors: wird mit 4 Vektoren beschrieben, wobei die 3 hinteren Werte angeben wie es nachschwingt und der erste Vektor die Lichtstärke/ Intensität angibt.

Fluoressens und optische Aufheller

Fluoressens ist überall anzutreffen und ist für Spektakuläre Effekte/ Oberfläche/ Farben zuständig…

Wenn man eine Oberfläche anleuchtet, und es gelb reflektiert, kann nicht mehr gelbes Licht raus kommen, als auf die Oberfläche „rein“ kommt.. deswegen könnte man andere Lichtwellen her nehmen, die „gelb“ umformen und erhält somit stärkeres Gelb als man es eigentlich bekommen kann.

Ein optischer Aufheller in Papier, weißer Kleidung,… ist nichts anderes als Fluoressens damit es schöner aussieht.

Oberflächen Absorption kann man normalerweise so ausrechnen, dass man Input minus Output rechnet und somit weiß, wieviel reflektiert wird und wieviel verloren geht… aber bei Fluoressens geht das nicht

Combined Filter: alle Kästchen über M sind leer und braucht man nicht.

Streuung: 2 Arten der Lichtstreuung:

- Streuung an Molekülen
- Streuung an Partikeln

Die Streuung von Licht an Molekülen ist der Grund dafür, dass der Himmel blau ist, und das absolut reines Wasser trotzdem blau ist

Interferenz: Wellenlichtabhängigre Effekt: das Problem ist, dass die Gleitkommaarithmetik nicht genau genug ist, um auf Nanometer Genauigkeit arbeiten zu können, und somit niemals genug signifikante Stellen hat, um es genau zu rechnen.

Aber die Interferenz Effekte die normalerweise für uns interessant sind z.b. Schmetterlingsflügel,,… kann man trotzdem machen indem man die Wellen „dehnt“ sodass man sie dann rechnen kann

Makroskopische Effekte gehen von der Geometrie her nicht, da es eben zu ungenau ist.

Pearlmutfarben: ändern Farben wenn man den Blickwinkel auf die Farben verändert. Z.b. ein Auto kann rot sein und dann an einigen Stellen deswegen blau ausschauen wenn man darum herum geht.

Reflection Models

Einerseits hat man die Geometrie, anderseits die Reflexionen.
BRDF: das selbe wie BDTF, nur das man dort Strahlen hat die ???? (Folien)
BSDF: BRDF+ BDTF
BTF: so wie BRDF, nur das man die Texturen erfasst „Image based Rendering“ wobei man wissen will wie eine Oberfläche unter verschiedene Lichtern ausschaut und z.b. Stoffbezüge in einem Auto aussehen wird.

Bidirectional Reflectance Distribution Function (BRDF)

Man kann den Strahl nach BRDF machen oder Lichtquelle sampeln. Das Problem ist, dass man nicht von Haus aussagen kann was „wichtig“ ist. Ist ein Prinzipielles Problem, dass es nicht automatisch entscheiden kann, sondern „abwägen“ muss und das alles komplexer und

somit auch Leistungsaufwändiger sind.

BRDF: umso weiter man von der Lichtquelle weg ist, umso verrauschter ist es. Bei Lightsource ist es umgekehrt, dass es umso verrauschter ist, umso näher man an der Lichtquelle ist! Man könnte es folglich mitteln, nur das Problem ist, dass es dann nicht die „besten" Punkte nimmt, sondern den „Durchschnitt" und es somit nur noch ärger macht!

Man braucht also eine besser Möglichkeit es zu „mixen" Die Idee ist es, zu schauen wie wahrscheinlich es ist, dass dieser Spezieller Schätzwert zu Stande kommt und nimmt dann den, der Wahrscheinlicher ist. Gewichtung mit WA und WB, wobei zusammen es 1 ergibt=> man weiß also was wichtiger/ wahrscheinlicher ist!

Es soll also am Schluss ein Faktor raus kommen, damit dass richtige raus gecancelt wird.

Man nimmt die Summer der Wahrscheinlichkeiten beider, dann die „einfache Wahrscheinlichkeit" (??????) und dividiert.

Unterhalten sich teilweiße sehr unterschiedlich, abhängig von der Distanz („Größenordnung")… z.b. aus weiter Entfernung kann es nach einer sehr „einfachen" Reflexion anschauen, während sie aus der Nähe ganz anders ist.

Wie kann man BRDFs machen: 2 Ansätze
1) Man misst eine Oberfläche ab
2) Man baut sich ein Analytisches Modell das sich so Verhält wie ein reales.

Ist für beides wichtig:
- Messwerte + Formel sind notwendig und sollten auch nur vernünftig viel Speicherplatz brauchen.
- Sollte auch schnelles Monte Carlo Sample Methoden berücksichtigen.
- Sollte auch Messergebnisse der Oberfläche gut einfangen.
- Intuitive Benutzung durch Benutzer

Da man die ganze Hemisphäre abtasten muss, fallen große Datenmengen an, und auch das messen dauert sehr lange. Man muss auch im Stande sein eine Verteilung zu würfeln!

Analytische BRDFs sind eigentlich recht gut.
Es gibt Physikalisch basierte Modelle, die sehr brauchbar sind

Empirisches Model: zu sampeln ist es leicht, und Verallgemeinerungen sind auch leicht zu machen.

„Wie schaut die BRDF einer diffusen Halbkugel aus?" (Frage kommt gerne zur Prüfung): man sieht es nicht an der Halbkugel sondern daran, dass die Oberfläche schrumpft.

Oberflächen Modelle

- Lambert: perfekt Diffuses Modell
- Oren- Nayar- Model: ein diffuses Modell auf Mirko Facetten Ebene. Man kann einen Rauheitsfaktor angeben, damit es nicht „perfekt" ist
- Phong: neigt dazu metallisch auszuschauen, abhängig vom Highlight

Beim Mond sieht man nicht dass es eine Kugel ist, da es eine starke retroreflektierende „Schicht“ hat auf der Oberfläche.

Bei ansteigenden Winkel wird bei den Modellen der ????? kleiner.

Je flacher man auf ein Objekt schaut/ sieht, umso spiegelnder ist es.

Das Problem ist das die analytischen Modelle es nicht mehr spiegelt (obwohl er es sollte, wenn man es flacher sieht), sondern weniger, was aber unser Auge nicht stört (aber theoretisch falsch ist)!

Highlight ist wichtig damit man auch „sieht“ welche Oberflächen Art es ist… Glas, Plastik, Metal,…. Deswegen schaut Phong auch meistens „Plastik mäßig“ aus (Ein Highlight das weiß ist schaut nach Plastik aus).

Torrance- Sparrow Model

physikalisch plausibles BRDF Model das mit statistischen Methoden arbeitet. Ist Mikro Facetten Model + Rauheitsfaktor + Fresnel Term für Reflexion
Da es viele sehr kleine Oberflächen gibt, und der Lichteinfallswinkel geringer wird, also flacher, so wird das Modell dunkler als bei perfekter Oberfläche, da statistisch gesehen kleine Oberflächen höher liegen können, und dadurch „dunkel“ sind.

Unterschied zum Oren- Nayar- Model: Oren- Nayar- Model geht davon aus, dass einzelne Facetten diffus sind und beim Torrance- Sparrow Model ist es glatt und besitzt perfekte Reflektoren („Fresnel“).

Torrance- Sparrow Model schaut bei der „Analyse“ (BRDF) aus wie eine Mischung aus Phong + Lambert, und nimmt auch mit der Flachheit des Lichteinfalls ab (???), aber bekommt dann einen Höcker wie er es sollte.

Bei rauer Oberfläche schaut es diffus am Anfang aus, aber dann hat es einen Höcker unterhalb der Spiegelrichtung, was auch sehr real ist

Wenn man das anwendet, kann man sehr gute Oberflächen z.b. für raues Metal machen.

Torrance- Sparrow Model schaut gut aus, aber bei kleinen Objekten kaum besser als andere Modelle.

Torrance- Sparrow Model ist physikalisch korrekt, liefert gute Ergebnisse, ist aber nicht wirklich echtzeittauglich, schwer auszuwerten, und hängt von Materialkonstanzen ab.

WARD BRDF Model: basiert auf echten Messungen. Ist mathematisch ein wenig aufwendiger. Gibt eine anisotrope Version .z.b. Aluminium (da dort ganz kleine Rillen in Längsrichtung sind) oder Holz (da das Highlight dann dort länglich ist)

BRDF hat das Problem, dass bestimmte Oberflächen nicht charakterisiert werden z.b. Phosphorisierende, Fluoreszierende,…. Oberflächen

Modeling Natural Phenomena

Bei Außenszenen gibt es oft Probleme z.b. bei Krieg der Sterne die Außenszene mit den Druiden und der grünen Wiese.

Sehr problematisch sind Außenaufnahmen deswegen, weil es z.b. viele Häuser gibt, die Sonne weit weg ist, und wenn man Photonen von der Sonne aus schickt, die Photonen nicht auf der Erde auftreffen. Man könnte selektiv nur auf die Häuser die Photonen schießen, aber wenn z.b. wo ein großes spiegelndes Haus weit weg ist, dass Caustics wirft, dann würde das weg fallen.
Weiters ist es so dass wenn man realistische Bilder haben will, die Verfahren die in der Echtzeitgrafik gut funktionieren, weg fallen.

Sie sind auch sehr komplex, und das Terrain muss per Hand modelliert werden und wenn man es automatisch machen will, weil es auf „Regeln" beruht, kann es auch zu Problemen kommen.

Implicit Surface Modeling

Implicit Surface Modeling (in Bryce Metaballs und in Cinema 4D Metaball Object)
Eignet sich sehr gut um Naturphänomene zu modellieren, Schnecken oder Gummibärchen.

- Vorteil dass es wenige Polygone hat,
- Nachteil dass es nicht mit Scanline geht sondern nur zb. mit Raytracern oder man es extrahieren muss (damit es z.b. renderbare Polygone sind).

Weiters ist es schwer genau damit zu modellieren, da die Parameter für Distanz und ähnliches recht schwer abzuschätzen und einzustellen sind, dass es die Ergebnisse erzielt die man haben will.

Bäume kann man auch gut machen oder sehr komplexe Meeresschnecken (mit einem prozedural geschriebenem Programm bei dem man über Parameter angibt wie die Schnecke aussehen soll)

Terrain Generation

Eine Landschaft sollte nicht nur aus Bergen und Tälern bestehen, sondern auch durch „Fehler" die durch Regen, Erosion, Gesteinszerfall, … entstehen

(Praktikumsidee des Vortragenden: Terrain Generation Programm schreiben)

Himmel

Der Himmel (Skydom Luminance) ist auch ein großes Problem, da der „Hausverstand" sagt das man einfach eine Blaue Kugel in der die Szene macht, aber der Himmel ist nicht gleichmäßig blau. Deswegen sollte man mit einem HDR Panorama arbeiten, dass „out of the Box" realistisch von Farbe und Himmel her ist, aber den Nachteil hat dass man darauf fixiert ist und keine Parameter ändern kann z.b. die Uhrzeit, Ort, Jahreszeit, Wolken, Licht, ….
Weiters ist eine Animation über längeren Zeitraum auch nicht möglich, da man nur ein Bild hat und man dann von jedem Frame eine geeignete HDR Map haben muss damit es wirklich 100% realistisch ist bzw. gut aussieht.

Die Alternative ist eine Funktion, bei der man alles einstellen kann (wie z.b. bei Cinema 4D wo man Wolken, Jahreszeit, Sterne,… einstellen kann) was aber normalerweise immer den Nachteil hat, dass Wolken nicht bei den Parametern vorkommen und man deswegen die

anders machen muss (z.b. Himmel und dann separat Wolken erstellen).

Wolken

Echte Wolken sind unten flach.

Man könnte Wolken Pathtracen aber viel zu langsam. In Echtzeitgrafik werden oft Vorberechnete „Boxes“ verwendet wo die Wolken drinnen sind.

Atmosphärische Effekte

Alle analytischen Modelle können „Diesigkeit“ (wie weit kann man schauen, wie „dreckig“ ist die Atmsphäre,…) nicht wirklich erstellen und berechnen oder Nebel

CIE Model gibt nur Helligkeit an und man färbt es dann (meistens blau) ein aber trotzdem schaut es dann nicht wirklich gut aus.

Atmosphärische „Effekte“ werden meisten nicht berechnet da es sehr kompliziert ist, meistens „zu schwach“ ist wenn man es hervorheben will, und es meistens gefakt wird da es auch noch „billiger“ ist.

Der Nachthimmel ist niemals komplett schwarz sondern hat immer ein sehr leichtes Licht!

Wolken werden meistens „schnell und nicht echt“ gemacht, da die Wolken nicht nur „random“ sind sondern sie durch bestimmte Prozesse entstehen und die schwer erfassbar sind oder sehr rechenintensiv sind. Vor allem bei Animationen und Zeitraffer fallen solche „Fehler“ dann auf.

Wir sind Experten im einschätzen von Menschen und schauen vor allem auf die Gesichtszüge und deswegen fallen so gut wie alle 3D Menschen leicht auf.

Wasser

Wasser ist problematisch da es nie 100% flach ist. Die Spiegelung ist recht „harmlos“ und die Farbe ist auch nicht sooo schwierig, sondern eben die Form und die Wellen sind problematisch, da die von einer Reihe von Faktoren abhängen, und das händische modellieren zu vergessen ist und man folglich auf prozedurale Programme zurückgreifen muss

Die Formeln von Jerlov für Wasser und deren Farbe ist sehr gut, aber wenn man Objekte im Wasser hat z.b. Hai, Wal,… dann sieht man die nicht!

Bump Mapping

Bump Mapping (man verbiegt die Oberflächenormalen z.b. bei einer Orange) wobei dadurch leicht „Unebenheiten“ entstehen und ist auch in der Regel sehr gut, außer man verwendet einen Raytracer. Man könnte damit z.b. eine Wasseroberfläche damit machen, indem man eine gerade Oberfläche macht, und über Bump Mapping dann die „Höhen und Tiefen“ rein macht. Aber da Pathtracen nicht geht, da wenn man z.b. die Normalen verändert, man „unterhalb“ ist und der Tracer glaubt dass er dann von unten kommt, obwohl er von oben kommt.

Einen Ozean mit Bump Mapping geht deswegen auch so nicht zu machen, sondern man muss es aus Geometrie machen, da sonst Raytracer nicht geht. Geometrie aber bis zu Horizont zu machen ist auch Problematisch, da dass SEHR viel Geometrie ist und das gerendert bzw. in der Grafikkarte gehalten werden muss.

Explicit Wavetrain Simulation

Frequency Based Approaches: Wellen haben auch ein Frequenz Spektrum und wenn man das kennt, kann man dadurch und durch Furie Transformation dann Meshes machen die dieselben Wellenkonturen haben. Man kann somit eine Kachel machen und dann z.b. die immer wieder wiederholen. Das Problem ist aber dass das Schiff im Wasser dann trotzdem unrealistisch ist da es nicht mit dem Schiff zusammenarbeitet also sich nicht mit dem Wasser schneidet, „ein und untertaucht", keinen „Schweif" hinter sich her zieht, einen „Wasser Schaum" um sich hat (bei Titanic wurde das hin gepinselt bzw. ist das Meer ganz ruhig damit man es nicht sieht)..

Penrose Muster: siehe Link.

Pflanzen

Blätter sind das wichtigste an Pflanzen und sehr schwer gut zu machen, da Licht durch Blätter durchscheint, aber an einigen Stellen mehr als an anderen Stellen und das Lichtverhalten im Blatt auch sehr komplex ist. Das liegt auch daran dass die Struktur von Blättern sehr komplex und vielschichtig ist.

Oliver Deussen hat Xfrog entwickelt und das Programm ist sehr gut, auch wenn es manchmal „zu perfekt" aussieht.

L System

L System werden gerne z.b. für Bäume, Pflanzen,… verwendet da es auf „mathematische" Regeln aufbaut.

Man startet mit einem Startsymbol und hat die Regel, dass man das Startsymbol immer in 2 Symbole umwandelt, wobei man durch Zufall bei der nächsten Iteration aus einem Strich 2 macht und eines eine Abzweigung bekommt=> man kann es also sehr gut „wachsen" lassen wie eine Pflanze, die sehr gut, „einfach" und vor allem sehr gut aussieht.
Ein anderer Vorteil ist, dass Sedlinski (??????) Tetraeder

Rendering Systems

Radiance: Toolkit das sehr wichtig ist. Ist „sehr alt" (Entwicklung begann 1988 und 1994 war es fertig, wobei es danach kaum weiterentwickelt wurde vom Kernalgorithmus her.). Ist Open Source aber eben nicht am aktuellen Stand. Ist eine Sammlung an Command Line Programmen und ein Stochastischer Ray- Tracer. Kann keine Caustics rechnen! Geht sehr intelligent vor, da er Gradient, räumliche Begebenheiten, …. beachtet und Bildergebnis ist sehr gut. Man kann auch anzeigen lassen, wo wieviel Lux sind und dadurch kann man sagen wie hell es wo sein würde wenn die Sonne so und so stark ist und das Licht auch so und so stark eingestellt wäre. Caustics ist in Architekturszenen unwichtig, wo hingegeben Diffuses ehr wichtig ist, und aus Optimierungsgründen wurde Caustics deswegen auch bewusst ausgeschlossen. Ein anderes Problem ist dass es ein Farbwertraum- Renderer ist.

Da das Programm von nur einem Menschen geschrieben wurde, sehr optimiert ist,… ist es kaum veränderbar. Das Parallelisieren ist auch sehr schwer, da man zwischendurch beim

rendern ein Ergebnis sehen kann, dass aber kaum mit Threads parallelisierbar ist.

Maya, 3DS Max, Renderman… sind nicht dazu gemacht bzw taugen nichts für eine Realistische Vorhersage, sondern sind meistens Hybriden und für Kreatives Arbeiten gedacht und man muss von der Hand aus viel machen (im Gegensatz zu Radiance) da man es mit der Größtmöglichen Freiheit verwenden soll/ will, was aber bei Radiance nicht wichtig ist, da man es wirklich realistisch und echtheitsgetreu will!

Man kann auch bei den 3D Programmen mithilfe von Plugins den Renderer austauschen.

Brazil ist eines der ersten Stochastischer Renderer gewesen.

Maxwell: erster Kommerzieller Spektral- Renderer.

Mental Ray: Raytracer das schnell ist und mit sehr vielen Polygonen (nur Dreiecke) zurechtkommt, und man es „vorkaut" da man die Kern- Engine hochoptimiert machen kann und deswegen vorgekaut werden muss, aber schnell ist. Photon Maps wurden integriert, wird aber kaum mehr weiterentwickelt da der Mitarbeiter der das Programm geschrieben hat rausgeworfen wurde und jetzt Uni Prof ist.

POVRay: schon sehr alt, aber trotzdem sehr fähiges Tool wenn man sich damit auskennt.

ART: Advanced Rendering Toolkit der TU Wien (CG Institut)

RenderPark: Gegenpart zu ART, dass auch sehr viele Rendering Algorithmen hatte. Man konnte damit auch sehr leicht mit verschiedenen Renderer rendern und dann die Ergebnisse vergleichen lassen.

PBRT: Subsurface Scuttering kann er rechnen (also wie Licht sich unter einen Oberfläche ausbreitet)

Was kann welcher Renderer?

Das Problem ist bei vielen Renderer das man weiß dass sie angeblich das und das können, dies aber oft nicht können oder nur sehr eingeschränkt. Viele Renderer verwenden auch z.b. „Quasi Monte Carlo" weil Stochastisches Rendering von Pixar patentiert ist und auch viele Firmen Klagen ersparen wollen und deswegen nicht sagen was eigentlich wirklich in ihrem Renderer drinnen steckt.

Lochkamera

Raytracing Bilder schauen oft unrealistisch perfekt aus. Es arbeitet wie eine Lochkamera, nur da wenn das Loch zu klein ist, kein Licht reinkommt, und man es größer macht, es unscharf ist, braucht man eine Blende. Dadurch dass das Loch aber „groß" ist, bildet man nicht durch ein Loch ab sondern durch ein großes und dadurch hat man ein „anderes Ergebnis". Durch Linse kann man es scharf bekommen und auch genug Licht bekommen, hat aber den Nachteil dass es nur für eine bestimmte Tiefe geht. Man kann also nicht Objekte aus unterschiedlichen Distanzen Fokussieren.

Relativ vs. Absolut

Pixel in einem Rasterbild können entweder absolute Helligkeitswerte sein („nach oben offen") oder relativ im Bezug auf ein Ausgabesystem (z.b. bei RGB gibt es XY Werte und wenn etwas 50% rot ist, dann ist es in der Mitte der XY Werte).

Photoshop und GIMP sind sehr gute Programme um relative Infos zu be- und verarbeiten, aber für absolute Werte gibt es nur PlugIns, die aber nicht so gut sind!

Diverses

Rendering Equation: aus der Sicht des Auges. Was kommt aus der Szene ins Auge. Man schießt Licht in die Szene und schaut was wo hängen bleibt.

Potential Equation: Rechnet umgekehrt. Rechnet Beleuchtungszustand in der Szene aus. Wo also in der Szene Licht ist zu einem bestimmten Zeitpunkt. Daten werden gespeichert und müssten dann noch verarbeitet werden.

Photontracing probiert Potentialgleichung auszurechnen und nicht Rendering Equation.

Radiosity: Überlegung: speichern der ganzen Lichtinformationen.

L = Le + TL …. Wenn man es mehrfach ausführt wird es nicht größer, sondern höchstens kleiner. Liegt daran das man einen Bounds (???) macht auf die Oberfläche, und da nie mehr Licht entstehen kann, kann es auch nicht kleiner werden.

Open GL, Scanline: ignoriert Kopplung
Raytracing: Folgt bestimmten leichten Kopplungen
Global Illumination: macht die Kopplung richtig

(Area) Light Source: Wahrscheinlichkeit das man Licht trifft ist umso schwächer, kleiner, weiter entfernter und flacher die Lichtquelle ist.

Reflectivity: wenn eine Ebene 0.1 hat wird der Strahl nur weiter geleitet, wenn die Zufallszahl mehr/ weniger (???????) als 0,1 hat

Russisches Roulett: Bilder sind „schircher“, obwohl es keinen fehler „macht“.
Expansion: kann man super parallelisieren da es genau teilbar ist!

Iteration: Mit Beleuchtungszustand muss man rechnen können (zb. Bei Radiosity).

Random Walk Algorithmen: ?????
Heckbert’s Taxonomy: Beschreibt welche Pfade es gibt.

T- Vertices: Immer achten das die Meshes wohlgeformt sind, und keine Löcher hat.

Ein leuchtender Patch (Bilder bei „Discomesh“) der Schatten erzeugt und von oben gesehen wird, schaut man wie die Ecken des oberen Objektes zu denen des unteren Objekts stehen, und schaut wo die unten auf dem Boden aufkommen und schaut, wo die endgültigen Kanten des Schattens sind!

OpenEXR: HDR in Hardware!

Rechtliches

Bitte beachtet dass diese Mitschrift für mich (Martin Tintel/ mtintel) gedacht ist und somit weder vollständig noch "richtig" sein muss. Es ist dafür gedacht dass wenn man ein Mal krank ist, und wissen will was durchgenommen wurde, man leicht nachsehen kann oder vor

der Prüfung auch noch andere Mitschriften hat wo man nachschauen kann, falls man sich unsicher ist oder es nicht ganz mitgeschrieben hat bzw. damit ich was habe nachdem ich lernen kann.

Wenn es Fehler oder sonst irgendwelche Probleme, Vorschläge,... gibt dann einfach posten oder mir eine Mail schicken!

Versionshistory:

1.0: Alles ist fertig (alle VOs abgehalten) und Zusammenfassung geschrieben.
1.1: Rechtschreibfehler ausgebessert, VO Termine raus genommen (Seitenanzahl reduziert) und Layout leicht verändert.
1.2: Rechtschreibfehler ausgebessert, Inhaltliches ausgebessert, begonnen Layout weiter zu ändern.
1.3: geplant: weitere Rechtschreibfehler auszubessern und Layout komplett zu ändern bzw Themenreihenfolge (so wie in einem Buch, wo man in jedem Kapitel alles geordnet hat und nicht wild über alle Kapitel verteilt hat) zu machen.
1.4: geplant: Restlich Rechtschreibfehler und inhaltliche Fehler ausbessern. PDF Version erstellen