Computergrafik II, Prof. Dr. Gitta Domik, Zusammenfassung von Florian Schoppmann

Zusammenfassung Computergrafik II

1. Modellierung

Rendering: Objektbeschreibung → Abbildungsparameter und Beleuchtung → Culling → Hidden Surface Removal, Rastern, Schattieren → Bildschirm

Gegeben eine Folge von Punkten: Man spricht bei einer Kurve von Approximation: Punkte werden nur annähert, aber nicht berührt (Idee: Punkte sind potentiell Fehler behaftete Messwerte) ↔ Interpolation Kurve verläuft durch alle Punkte

Repräsentation: Kriterien für Repräsentation beliebiger Kurven/Flächen: Stabilität, Glattheit, Einfachheit (zum Berechnen), Interpolation notwendig/reicht Approximation, Ableitungen benötigt?

explizit: $z = f(x, y) \leftrightarrow$ implizit: $g(x, y) = 0$

Normalenvektor: $n = \frac{\partial p(u,v)}{\partial u} \times \frac{\partial p(u,v)}{\partial v}$ (wenn u, v, z ein Rechtssystem bilden)

„Algorithmus“ von De Castlejau:
Eingabe: $n+1$ Stützpunkte $p_0, \ldots, p_n$, $t \in [0, 1]$
Ausgabe: Punkt auf Bézier-Kurve

1: Setze $p_i^{(0)} := p_i$ für alle $i \in \{0, \ldots, n\}$
2: **for** k := 1, . . . , n **do**
3: Setze $p_i^{(k)} := (1-t)p_i^{(k-1)} + tp_{i+1}^{(k-1)}$ für alle $i \in \{0, \ldots, n-k\}$
4: **end for**
5: Gebe $p_0^{(n)}$ aus

Verallgemeinerung durch Bernstein-Polynome: Eine Bézier-Kurve durch $n+1$ Punkte $p_1, \ldots, p_{n+1}$ ist gegeben durch:

$$p(t) = \sum_{k=0}^{n} p_k B_k^n(t),$$

wobei B_k^n dasjenige Bernstein-Polynom ist, das wie folgt berechnet wird:

$$B_k^n(t) = \binom{n}{k}(1-t)^{n-k}t^k$$

Beweis per Induktion über die Anzahl der Stützpunkte n: Sei $(p_i)_{i\in\mathbb{N}}$ Punktfolge. Induktionsbehauptung:
De Castlejau$((p_0, \ldots, p_n), t) = \sum_{k=0}^{n} p_k B_k^n(t)$

IA: $n = 1$: $p_0^{(1)} = (1-t)p_0 + tp_1 = p_0 B_0^1 + p_1 B_1^1$
IS: $n \to n+1$

$$\begin{aligned}
&p_0^{(n+1)} = (1-t)p_0^{(n)} + tp_1^{(n)} \\
=&(t-1)\sum_{k=0}^{n} p_k B_k^n(t) + t\sum_{k=0}^{n} p_{k+1} B_k^n(t) \\
=&\sum_{k=0}^{n} p_k \binom{n}{k}(1-t)^{n+1-k}t^k \\
+&\sum_{k=1}^{n+1} p_k \binom{n}{k-1}(1-t)^{n-(k-1)}t^k \\
=&\sum_{k=0}^{n+1} p_k \underbrace{\binom{n+1}{k}(1-t)^{n+1-k}t^k}_{=B_k^{n+1}(t)}
\end{aligned}$$

Problem bei Bernstein-Polynomen $B_k^n(t)$: Trägermenge ist $[0, 1]$. Abhilfe z. B. segmentieren.

Alternative: Splines als stückweise definierte Polynome.

Bézier	*B-Splines*
globaler Einfluss auf Kurve	lokaler Einfluss
Innerhalb konvexer Hülle	
stetig	
Approximation und Interpolation	nur Approximation

Cox-deBoor „Algorithmus“ zur Approximation von $n+1$ Punkten $p_1, \ldots, p_n$, wobei Spline-Ordnung m (Grad $m-1$) und Knoten $t_0, \ldots, t_{n+m}$ zwischen den Polynomen vorgegeben: Berechne für $k = 0, \ldots, n$:

$$\begin{aligned}
N_{k,m}(t) =& \frac{t - t_k}{t_{k+m-1} - t_k} N_{k,m-1}(t) \\
&+ \frac{t_{k+m} - t}{t_{k+m} - t_{k+1}} N_{k+1,m-1}(t)
\end{aligned}$$

Dann:

$$Q(t) = \sum_{k=0}^{n} p_k N_{k,m}(t)$$

NURBS (Non-uniform rational B-Splines): Gebrochen ratoinale Basisfunktionen anstatt von Polynomen

B-Spline-Flächen: Fläche gegeben durch Punkte $p_{0,0}, \ldots, p_{n,n}$. Segmente $Q_{s,t}$, $3 \le s, t \le n$.

$$\begin{aligned}
Q_{m,n}(u, v) = \sum_{i=0}^{3}\sum_{j=0}^{3} \big(&p_{m-3+i,n-3+j} \\
&\cdot B_{m-3+i}(u) \cdot B_{n-3+i}(v)\big)
\end{aligned}$$

2. Beleuchtung

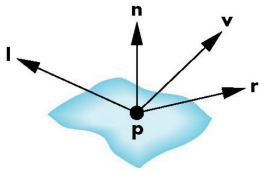
Reflexionen nicht mit Pipeline-Modell (jedes Polygon wird unabhängig berechnet), globale Effekte z. B. Schatten, Mehrfach-Reflexionen, Transparenz

Modellierung einer Lichtquelle, Auswirkung auf einen Punkt (x, y, z): $I_L = (I, I_x, I_y, I_z, x, y, z, \theta, \lambda, \dots)$, wobei I Energiestärke, (I_x, I_y, I_z) Ort der Lichtquelle, (θ, λ) Richtung, etc.

Empirisches Modell: Messen der Energie für „alle Situationen“

„Standard“ Lichtquellen: Punktlicht (gleichmäßige Abstrahlung in alle Richtungen), Gerichtetes Licht (von einer weit entern liegenden Punktlichtquelle, bspw. Sonne), Spotlicht (strahlt Licht innerhalb eines begrenzten Winkels ab), Flächenlichtquellen

Phong: $I = k_d I_d \langle l, n\rangle + k_s I_s \langle r, v\rangle^\alpha + k_a I_a$ (Diffuse, Specular, Ambient)



Problem bei Phong: Specular-Term muss bei jedem Knoten neu berechnet werden, r zu berechnen ist aufwendig. Deshalb Ersetzung von $\langle r, v\rangle$ durch $\langle n, h\rangle$, wobei $h = \frac{l+v}{|l+v|}$ der sogenannte, von Blinn 1977 vorgeschlagene, Halfway Vector ist. Fehler wird durch neuen Exponenten α' wettgemacht.

Schatten-Realisierung für Punkt-Lichtquelle durch Projektion der Polygone mit Lichtquelle als Projektionszentrum

Bidirectional Reflection Distribution Functions: Materialeigenschaft, die den Anteil des reflektierenden Lichts für alle Kombinationen von l, und v definiert.

Cook-Torrance

3. Animation, Morphing, Texture-Mapping

Animation

Rigid Body Animation Bewegung starrer Körper

Articulated Structure Animation hierarchische Modellierung

Behavioral Animation auch gemeinsames Agieren vieler Einzelteile, z. B. Fischschwarm

Particle Animation Individuelle Animieren, bspw. Luftpartikel im Sturm

Procedural Animation Math. Modell, das Bewegung kontrolliert

Facial Animation

Soft Body Animation

Morphing: In k Schritten den Grauwert der Position im Urbild auf Grauwert der Position im Endbild umrechnen

Texture Mapping: Motivation mit Beispiel: Darstellung von Orange mit Polygonen zu aufwändig

Texture Mapping Bilder füllen Polygone

Environmental (Reflection) Mapping
: Objektumfeld als Texture Map nutzen (Multipass-Rendering):
Erst Szene ohne Spiegel berechnen, wobei Kamera im Zentrum des Spiegels. Dann Bild aus dem ersten Durchgang als Textur verwenden.

Bump Mapping Veränderung von Normalvektoren simulieren

Ausführung des Mapping am Ende Rendering Pipeline (→ weniger Polygone betroffen)

Koordinatensysteme:

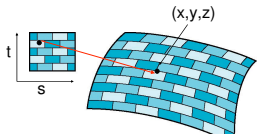
Parametrische Koordinaten Zur Modellierung von gekrümmten Oberflächen

Texturkoordinaten

Weltkoordinaten

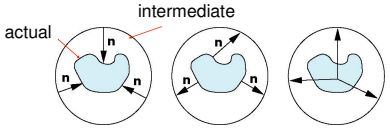
Bildkoordinaten Koordinaten des Ergebnisbildes

Backward Mapping: Zuordnungen $s = s(x, y, z)$, $t = t(x, y, z)$ gesucht.



Two-party mapping mit Zwischenoberfläche: Bspw. Cylindrical Mapping, Spherical Mapping, Box Mapping

Alternativen beim Second Mapping (von Zwischenoberfläche auf echte Oberfläche): Normale von Zwischenoberfläche auf echte, Normale von echter Oberfläche auf Zwischenoberfläche, Vektor vom Zentrum der Zwischenoberfläche aus



Vermeidung von Aliasing beim Backward Mapping durch Durchschnittsbildung von Pixeln in Textur

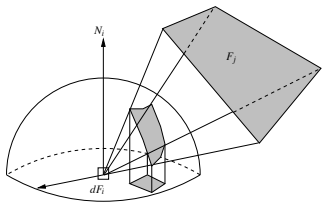
4. Radiosity

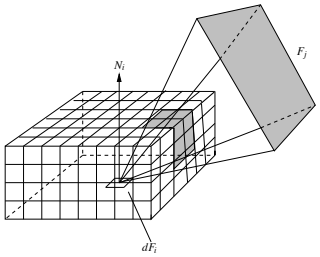
Idee: Zerteilung von Objekten in Oberflächenelemente („Patches“), jeder Patch hat homogene Eigenschaften in Bez. auf Emission, Reflexion, ..., Berechnung der Interaktion der Energie

„Radiosity mal Fläche eines Patches = emittierte Energie + reflektierte Energie“. Radiosity ist die Energie, die von einem Patch ausgeht, beschrieben pro Einheitsfläche pro Zeiteinheit.

Formfaktor F_{ij} beschreibt Geometrie zwischen Patch j und i, wird nur einmal berechnet und drückt den Energieaustausch der beiden Patches A_i und A_j aus – abhängig von Orientierung und Distanz. Es gilt:
$F_{ij} = \frac{\text{Abstrahlende Energie von } A_i\text{, die direkt auf } A_j \text{ landet}}{\text{Abstrahlende Energie von } A_i \text{ in alle Richtungen}}$
(„alle“: nur hemisphärischer Bereich)

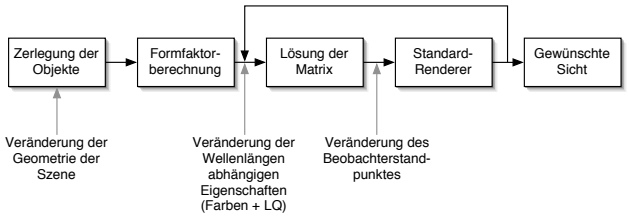
Für eine endliche Fläche ist der Formfaktor gleich dem Verhältnis der Basisfläche der Halbkugel zur Orthogonalprojektion der auf die Halbkugel projizierten Fläche.





In der Praxis Hemicube-Methode. Bei Doppelprojektion wird Tiefe verglichen und das entferntere eliminiert. Vorteil: Für jeden Pixel des Hemicubes kann Formfaktor ΔF berechnet werden. Alle ΔF können dann in einer Tabelle zwischengespeichert werden.

Zusammenfassung:



Progressive Refinement Algorithmus

1: **repeat**
2: **for** each patch i, starting with biggest emission **do**
3: position hemicube on patch i
4: calculate formfactors F_{ij} {nur erste Iteration}
5: **for** each patch $j \neq i$ **do**
6: $\Delta R := \rho_j \cdot \Delta B_i \cdot F_{ij} \cdot A_i/A_j$ {Strahlung von Fläche i}
7: $\Delta B_j := \Delta B_j + \Delta R$ {Differenz erhören}
8: $B_j := B_j + \Delta R$ {Strahlung erhöhen}
9: **end for**
10: $\Delta B_i := 0$ {Überschuss ist verteilt}
11: **end for**
12: **until** convergence

5. Raytracing

Erweitertes Raytracing: Es wird zuerst ein Strahl verfolgt, der sich dann im Schnittpunkt mit einem Objekt in weitere Strahlen auflöst:

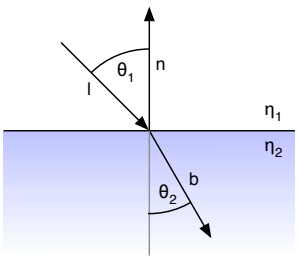
Schattenstrahl Gibt es ein Hindernis/Objekt auf dem Strahl zwischen Schnittpunkt und Lichtquelle?

Lichtbeugungsstrahl Falls das Objekt transparent ist

Reflexionsstrahl Bei Reflektor in Reflexionsrichtung

Lichtbrechung: Nach Gesetz von Snellius gilt:

$$\frac{\sin\theta_2}{\sin\theta_1} = \frac{\eta_1}{\eta_2}$$



Die Vektoren in der Grafik seien normiert. Dann folgt für den Ausfallsvektor:

$$\begin{aligned} b &= -n\cos\theta_2 + \frac{l + n\cos\theta_1}{\sin\theta_1}\sin\theta_2 \\ &= \frac{\eta_1}{\eta_2} l + (\cos\theta_2 - \frac{\eta_1}{\eta_2}\cos\theta_1) n \end{aligned}$$

Raytracing ist empirisch, physikalisch falsch. Z. B. lokal ambientes Licht, direkte Beleuchtung, global aber nur einzelne Strahlen, die zurückverfolgt werden

Lichtspeicher nach Haines und Greenberg: Alle Polygone werden auf den Lichtspeicher projiziert, Projektionszentrum ist Lichtquelle. Lichtspeicher enthält Tupel (*Objekt-ID*, *Polygon-ID*, *Entfernung*), sortiert nach Entfernung. Bei Schattenberechnung wird in der Menge der geschnittenen Objekte nach undurchsichtigen Polygonen gesucht.

Problem Aliasing auch beim Raytracing aufgrund unendlich dünner Strahlen. Abhilfe durch Erhöhung der Anzahl der Strahlen, z. B. Supersampling, dem Aussenden mehrerer Strahlen pro Pixel.

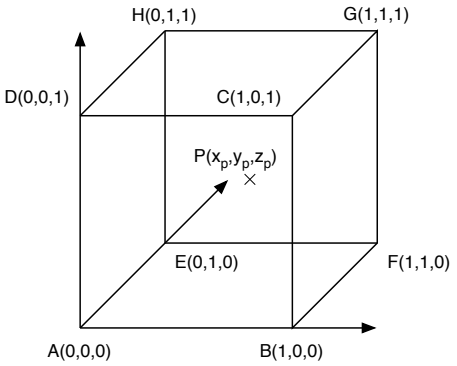
Disitributed Raytracing: Supersampling mit zufälliger Ablenkung einzelner Strahlen.

Adaptive Tiefenkontrolle beim rekursiven Raytracing.

Umschreibende Hüllen für einfacheren Ausschluss: Kostenfunktion $T = bB + iI$, wobei b Anzahl der Schnittpunkttests für Bounding Volume, B Kosten dafür, i Anzahl der Tests für Objekt, I Kosten dafür.

	Raytracing	*Radiosity*
gut	spiegelnder Reflexion, Lichtbeugung, Schatten	diffuser Reflexion
schlecht	diffuser Reflexion	spiegelnder Reflexion, Lichtbeugung

6. Volumen-Rendering



Trilineare Interpolation: Voxelwerte $V_A, \ldots, V_H$ gegeben:

$$\begin{aligned} V_P =& (1-x_p)(1-y_p)(1-z_p)V_A \\ &+x_p(1-y_p)(1-z_p)V_B \\ &+x_p(1-y_p)z_pV_C \\ &+(1-x_p)(1-y_p)z_pV_D \\ &+(1-x_p)y_p(1-z_p)V_E \\ &+x_py_p(1-z_p)V_F \\ &+x_py_pz_pV_G \\ &+(1-x_p)x_pz_pV_H \end{aligned}$$

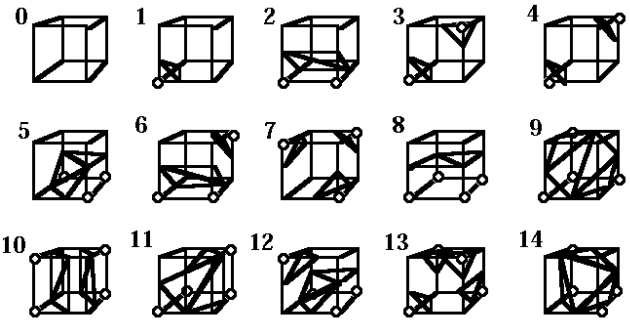
Anwendung: Suche Daten mit bestimmten Voxelwerten, z. B. Knochen: $180 \leq$ Wert ≤ 220

Ray Casting: Klassifizieren der Voxel (Farbwert C, Opacity α) feststellen, Transformieren der Volumendaten, Ray Casting: Für jeden von einem Strahl durchlaufenen Voxel: $c_{\text{out}} = c_{\text{in}}(1-\alpha) + C\alpha$. Bei Farbe: $\alpha = (\alpha_r, \alpha_g, \alpha_b)$

Volumen Rendering: Backward mapping, Forward Mapping

Oberflächen-Gewinnung durch Gradient

Marching Cubes Algorithmus: Zeichne innerhalb jedes Voxelwürfels die in der Code-Tabelle angegebenen Isoflächensegmente



Es kann bei isolierter Betrachtung von Voxelwürfeln Mehrdeutigkeiten geben, in diesem Falle müssen die Nachbarwürfel betrachtet werden.