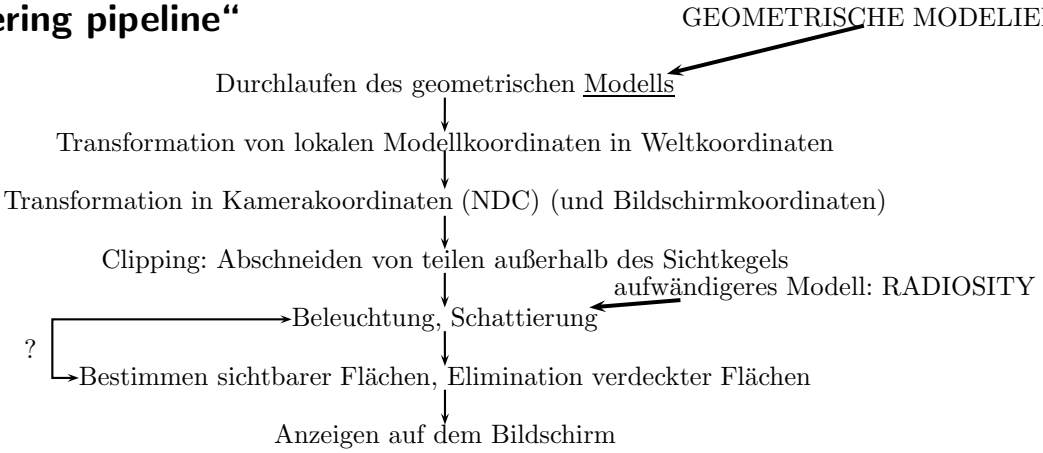
1 Geometrische Modellierung

1.1 „Rendering pipeline“



nur eine typische Reihenfolge!

Alle diese Operationen werden heute von Hardwareausgeführt

Alternative für Beleuchtung, Schattierung und Bestimmung sichtbarer Flächen: Raytracing (Strahlverfolgung)

1.2 Geometrische Modellierung

1.2.1 Wie kann man geometrische Formen darstellen?

Möglichkeit 1) *geometrische Grundformen* (Kugel, Zylinder, Polygone) werden zusammengesetzt

Grundmodell: triangulierte Flächen.

Mit triangulierten Flächen lässt sich im Prinzip alles genügend fein approximieren

(flache Polygone mit mehr als 3 Ecken sind in der Regel auch zugelassen)

Möglichkeit 2) *Implizite Flächen* Fläche ist durch eine Gleichung gegeben

$$(x^2 + y^2)^2 + 3z^3xy - 28xz^2 = 0$$

Möglichkeit 3) parametrische Fläche
z. B. Kugel:

$$\begin{aligned} x &= \cos\varphi \cos\vartheta \\ y &= \sin\varphi \cos\vartheta \\ z &= \sin\vartheta \end{aligned}$$

$$0 \le \varphi \le 2\pi \qquad -\pi \le \vartheta \le \pi$$

Operation	**Dreicksgitter**	**implizite Fläche**	**parametrische Fläche**
Erzeugen eines Punktes auf der Fläche	sehr leicht	schwierig	sehr leicht
Durchlaufen aller Punkte der Fläche	sehr leicht	sehr schwierig	leicht
Schnitt der Fläche mit einem Sichtstrahl (z. B. beim Raytracing)	durchschnittlich[1] (Modell durchlaufen)	leicht (Lösen eines Gleichungssystems in einer Variablen)	sehr schwierig (Gleichungssystem mit 3 Variablen)
geometrische Manipulationen	sehr leicht	schwierig	durchschnittlich
Speicherverbrauch	ist abhängig von der Genauigkeit	kompakt	kompakt

Umwandlung von impliziten oder paramtrischen Flächen in ein Dreicksgitter: „Gittererzeugung“ („Meshing“)

1.2.2 Darstellung polyedrischer Flächen bzw. Polygongittern

1. **primitiv:** Liste von Dreiecken, Vierecken, etc.

$$D_1 : (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)$$
$$D_2 : (x_4, y_4, z_4), (x_5, y_5, z_5), (x_6, y_6, z_6)$$
$$\vdots$$

   Problem: in einem typischen Dreiecksgitter kommen die gleichen Punkte mehrfach vor. Es ist Verschwendung, die gleichen Daten mehrfach zu speichern bzw. zu übertragen.

2. Trennung der Koordinaten von der kombinatorischen Struktur:

$$P_1 : (x_1, y_1, z_1) \qquad D_1 : 1, 2, 4$$
$$P_2 : (x_2, y_2, z_2) \qquad D_2 : 2, 3, 4$$
$$\vdots \qquad\qquad \vdots$$

   Flächen sind typischerweise *orientiert*: Dreieck P_1, P_2, P_3 und Dreieck P_1, P_3, P_2 sind Vorder- und Rückseite des gleichen Dreiecks im Raum, un sie können unterschiedliche Materialeigenschaften haben.

   Um noch mehr zu sparen, werden Dreiecke zu alternierenden Dreiecksstreifen bzw. Dreiecksfächern zusammengefasst. (und Vierecke zu Viereckssstreifen)

   Für n Dreicke benötigt man nur $n + 2$ Punkte. Für isolierte Dreicke werden $3n$ Punkte benötigt.

Datenstruktur zum Speichern der kombinatorischen Struktur einer polyedrischen Fläche

- Doubly-connected edge list (*DECL*, doppelt-verkettete Kantenliste)
- Quad-edge data structure
- „winged-edge“ data structure

Jede Kante wird durch zwei *Halbkanten* repräsentiert, eine in jede Richtung

Jede Halbkante hat 3 Zeiger:

- $l(e), r(e), ...$ linke und rechte Nachbarkante mit demselben Startknoten

[1] weder leicht noch schwer

- $u(e)$... Umkehrkante

e	$r(e)$	$l(e)$	$u(e)$
1	2	3	4
2	$\emptyset$	1	5
3	1	$\emptyset$	6
4	7	8	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$

- Jede Kante liegt an bis zu 23 Flächen an, eine links und eine rechts.
- Durchlaufen aller Kanten, die von einem Knoten ausgehen
- Wir benötigen eine erste Kante e_1

$$\begin{aligned} e_2 &= r(e_1) \\ e_3 &= r(e_2) \\ &\vdots \end{aligned}$$

bis $e_i = e_1$ oder $e_i = \emptyset$

Durchlaufen aller Kanten einer Fläche, die rechts von e_1 liegt.

$$\begin{aligned} e_2 &= l(u(e_1)) \\ e_3 &= l(u(e_2)) \\ &\vdots \end{aligned}$$

bis $e_i = e_1$

$$\begin{aligned} l(r(e)) &= e && \text{falls } r(e) \neq 0 \\ r(l(e)) &= e && \text{falls } l(e) \neq 0 \\ u(u(e)) &= e \end{aligned}$$