LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

Computergrafik I

Sichtbarkeitsbestimmung

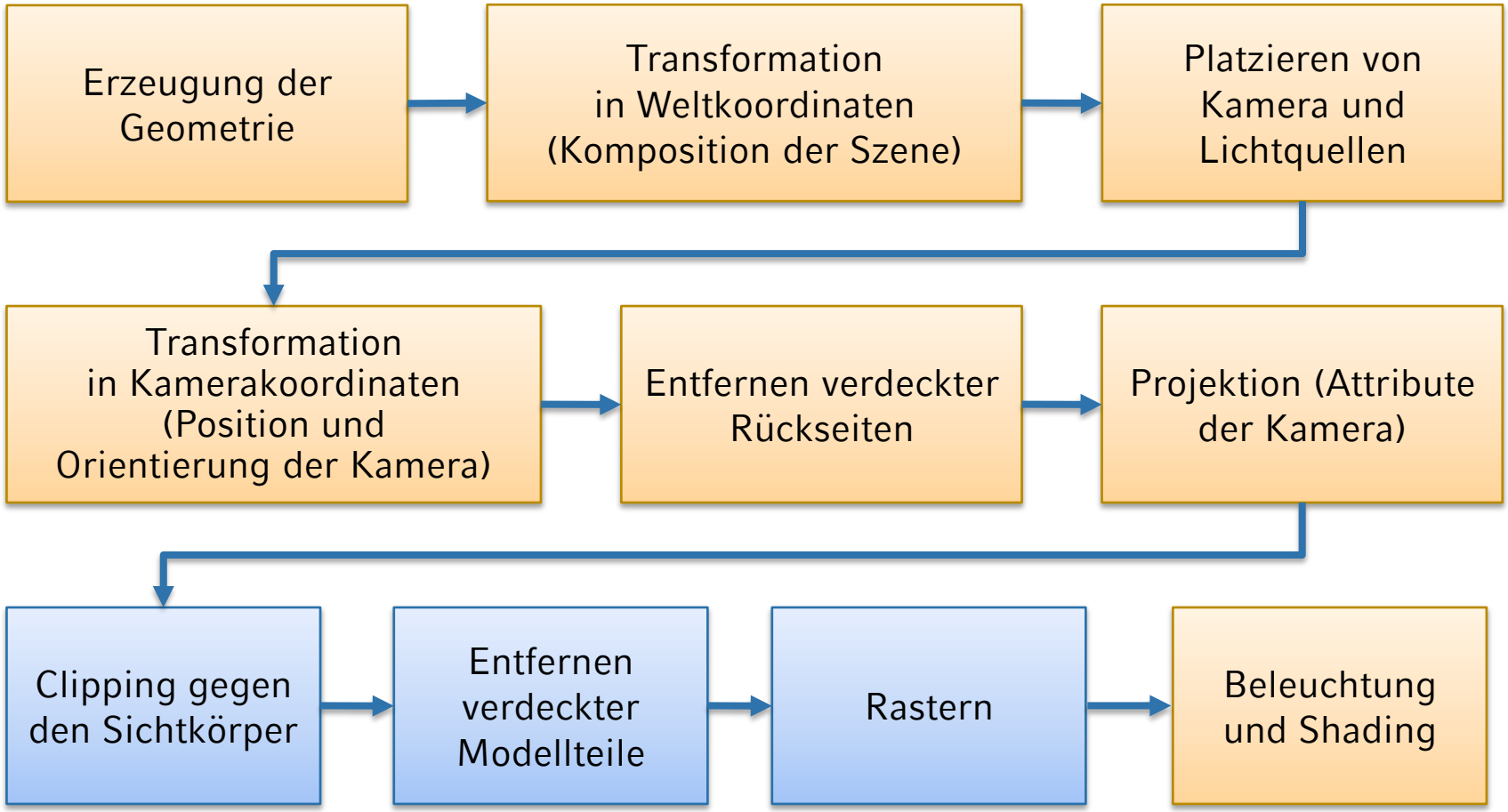
Letztes Mal

- Shading
- Lokales Beleuchtungsmodell
- Globale Beleuchtungsmodelle

Fahrplan für heute

- Nachsatz: Prozedurale Modellierung
- Animationstechniken
- Sichtbarkeitsbestimmung
- Rastern (und Shading)

Rendering-Pipeline



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

von 3D nach 2D

Prozedurale Modellierung

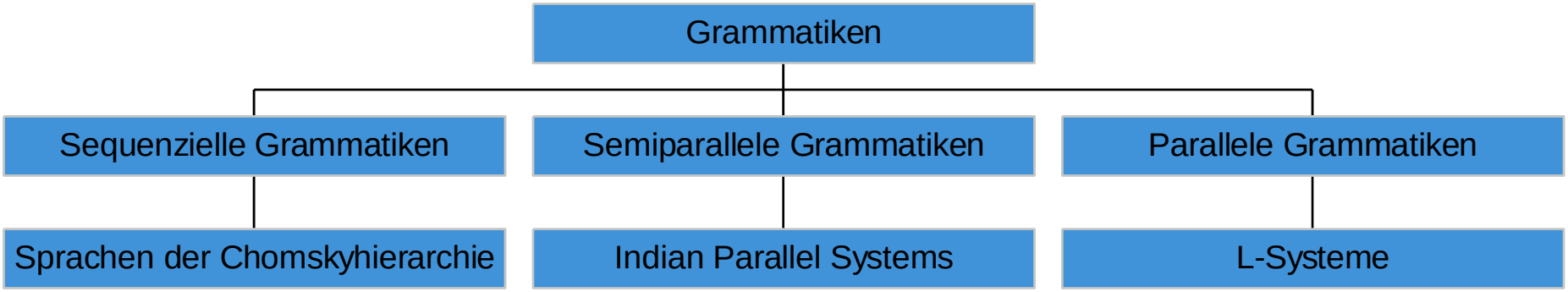
L-Systeme

- Entwickelt von Aristid Lindenmayer
- Ende der 60iger Jahre
- Entwickelt als Automat und später als Teil der Formalen Sprachen definiert
- Mathematisches Modell zur Beschreibung von biologischen Prozessen
  - Zelltrennung
  - Pflanzenwachstum

L-Systeme

- Bereich der Theoretische Informatik
- Formale Sprachen
- Parallele Ersetzungsgrammatik

L-Systeme


Grammatiken
Sequenzielle Grammatiken
Semiparallele Grammatiken
Parallele Grammatiken
Sprachen der Chomskyhierarchie
Indian Parallel Systems
L-Systeme

L-Systeme

- Mittels eines L-Systems wird ein Wort generiert
- Wortgenerierung erfolgt über Regeln
- Beispiel
  - Startwort: F
  - Regel: F → FF
  - Wortgenerierung
    - 1. Ableitung: F
    - 2. Ableitung: FF
    - 3. Ableitung: FFFF
    - 4. Ableitung: FFFFFFFF
    - 5. Ableitung: FFFFFFFFFFFFFFFF

L-Systeme

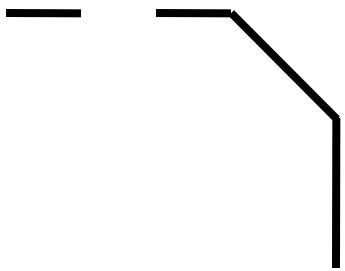
- Bestimmte Zeichen können als Kommandos interpretiert werden
- Beispiele
  - F0
    - Gehe eine bestimmte Wegstrecke und hinterlasse eine Linie (im 2D-Raum)
  - M0
    - Gehe eine bestimmte Wegstrecke und hinterlasse keine Linie (im 2D-Raum)
  - L(x)
    - Änder die Länge der Wegstrecke
  - LMul(x)
    - Multipliziere die aktuelle Länge der Wegstrecke mit x
  - RU(x)
    - Rotiere um x-Grad auf der y-Achse

L-SYSTEME

- Was ist das graphische Ergebnis aus dem folgenden Wort?
- `L(10) F0 RU(45) F0 RU(45) LMul(0.5) F0 M0 F0 ?`

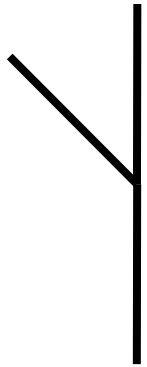
L-Systeme

- `L(10) F0 RU(45) F0 RU(45) LMul(0.5) F0 M0 F0`



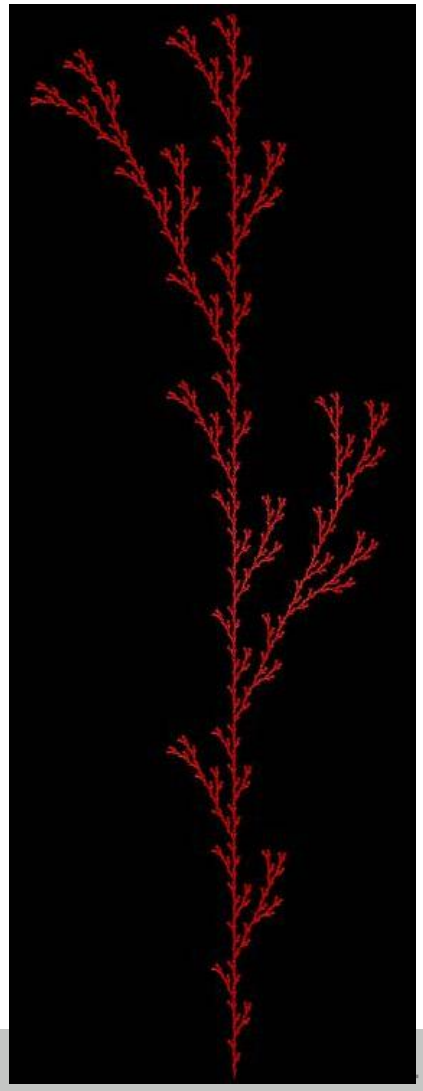
L-Systeme

- Realisierung von Verzweigungen mittels []
- Beispiel: F [RU(45) F] F



L-Systeme

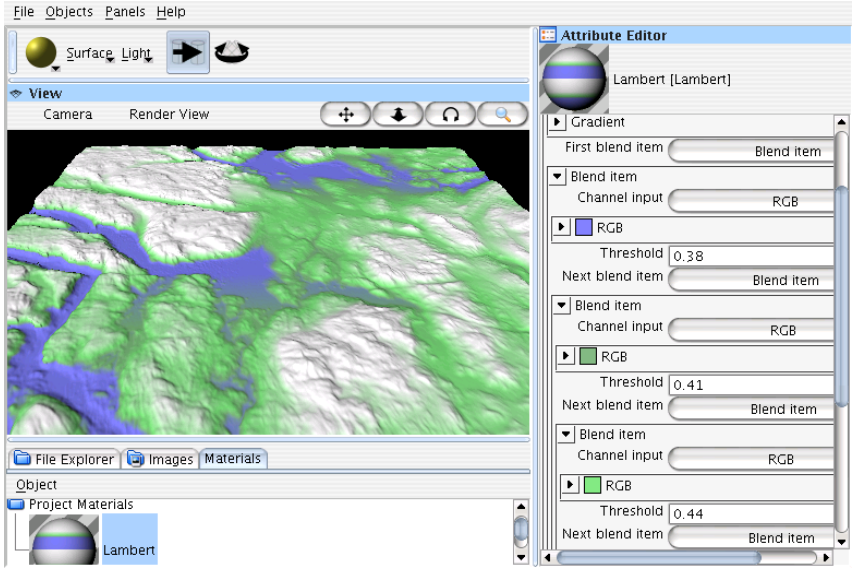
- Ableitungsschritte = 5
- Basisstärke = 60
- Startaxiom = F
- Regel: F → F[RU(25.7)F]F[RU(-25.7)F]F



L-Systeme

Die Plattform GroIMP

- Freie Entwicklungsplattform (Open Source) für 3D Modellierung



Wie kann GroIMP eingesetzt werden?

- Büsche, Bäume, Blumen, …, jegliche Art von Pflanzenstrukturen
- Steine, Felsen und Gebirge
- Verschiedene Arten von Landschaften
  - Mondlandschaft, Vulkanlandschaft,etc.

Warum prozedurale Modellierung?

- Biologische Korrektheit
  - Integration des Öko-Systems
  - Wettkampf um Ressourcen
  - Realistischere Modelle
- Genaues Modellieren
  - Genaue Dimensionierung von 3D-Teilen
  - Die 3D-Primitiven sind alle miteinander verbunden
  - Keine fehlenden 3D-Teile

Beispiele

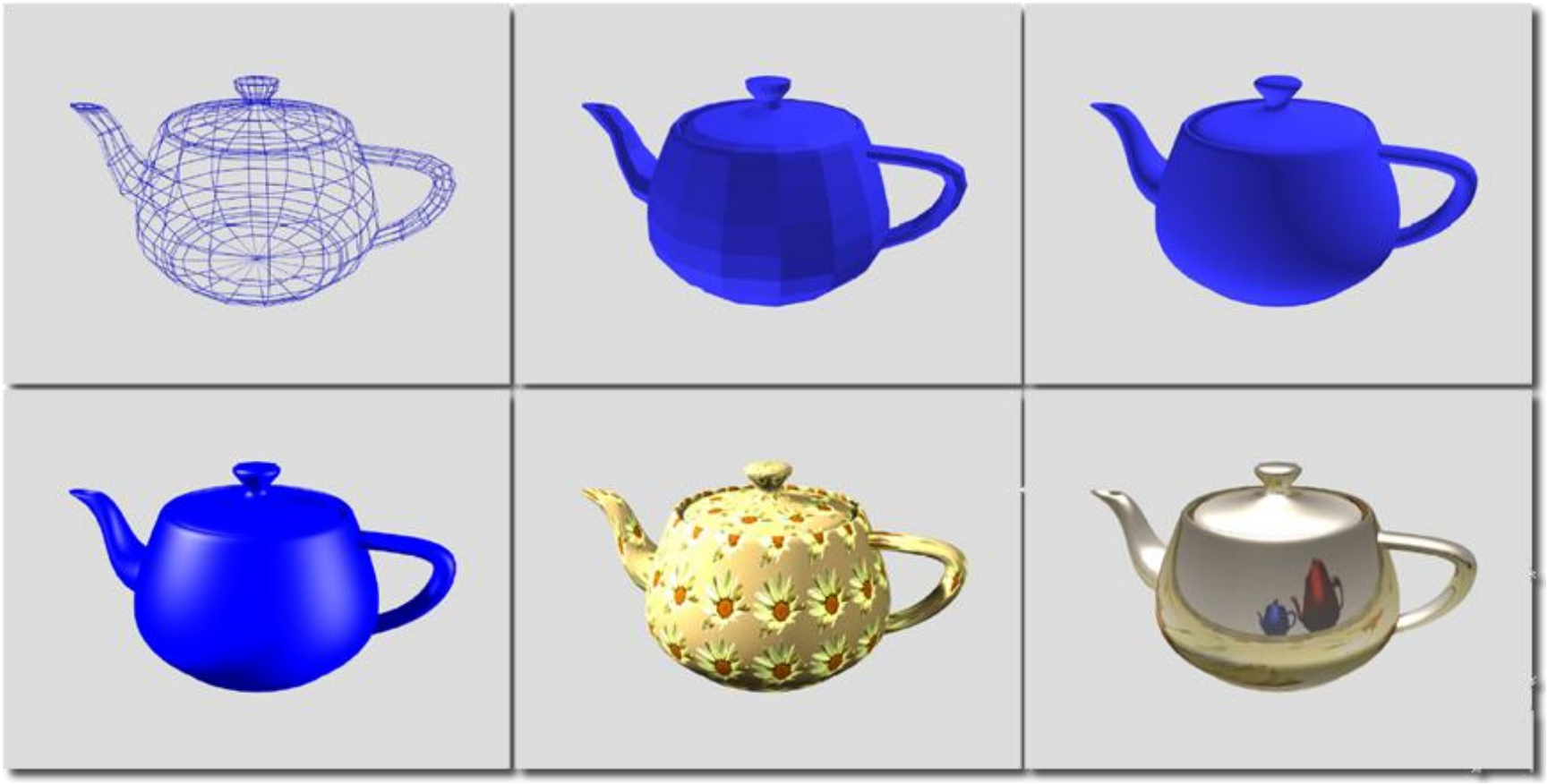
Oliver Deußen et al.,
Universität Magdeburg,
1997

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

von 3D nach 2D

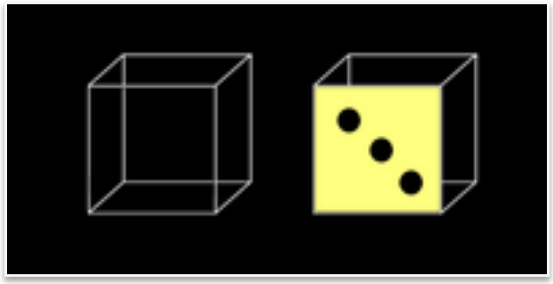
Materialien

Weitere Techniken



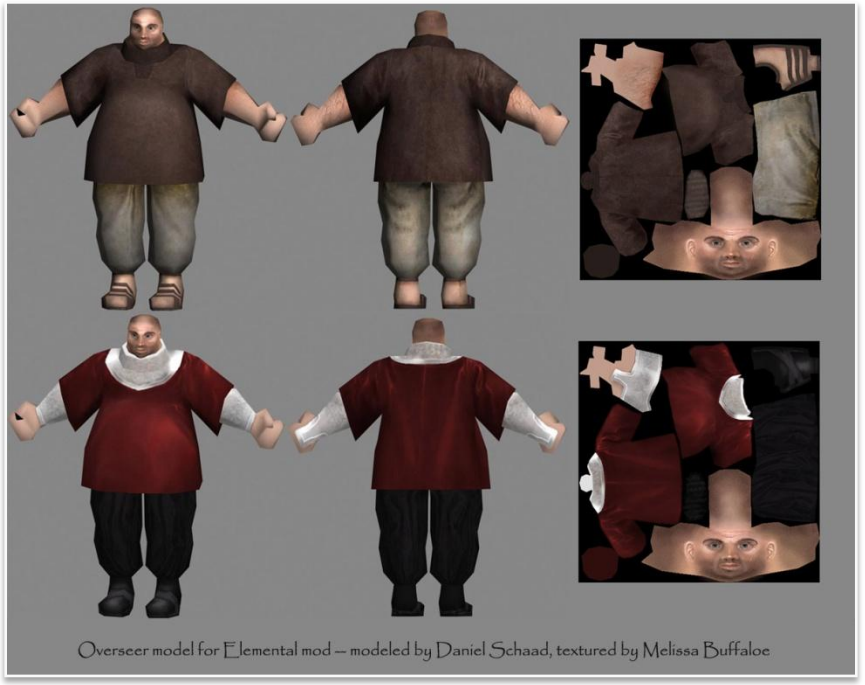
Mapping

- Auftragen einer 2D Map auf 3D Polygon(e) (Drahtgitter)
- Map/Textur: Bild/Muster oder Oberflächeneigenschaften
- erhöht Realismus ohne Anzahl der Polygone (Rechenaufwand) zu erhöhen
- Jeder 3D Vertex(x,y,z) bekommt Texel zugeordnet
- Texel:
  - u-,v-Koordinaten
  - Position auf der Textur (0,0 links unten, 1,1 rechts oben)
- Interpolation von Eckpunkt zu Eckpunkt eines Polygons



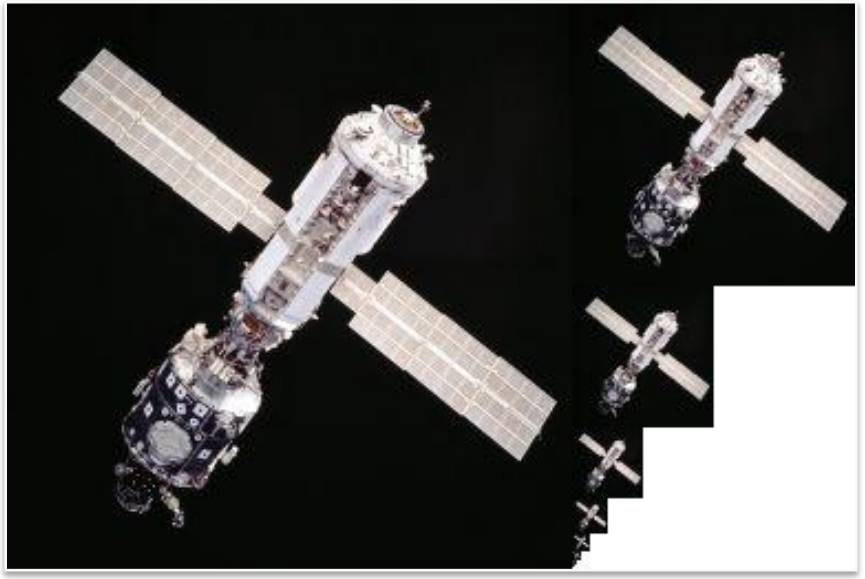
Texture Mapping (Diffuse Color Map)

- Textur: Bild oder Muster
- Problem: Ansicht auf weit entferntes oder nahe Objekte
  - Pixel können wesentlich größer/kleiner als Texel sein
  - Mittelwertberechnung meist zu aufwendig
  - Lösung: Mip Mapping



MIP-Mapping

- Ursprüngliche Textur wird abgespeichert und verkleinert
  - Verhältnis 4:1 (halbe Höhe, halbe Breite)
  - mit entstandenen Texturen rekursiv fortgeführt bis Auflösung 1:1
  - häufig von Grafikhardware beim Texturladen angewandt
- Textur mit passender Größe wird verwendet
- 1/3 höherer Speicherbedarf



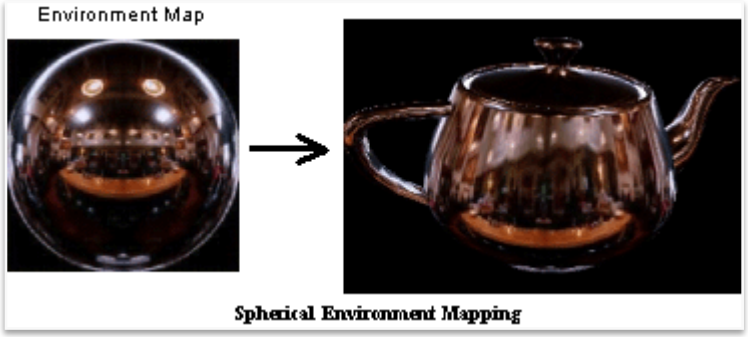
Environment Mapping

Spiegeleffekte der Umgebung simuliert, ohne eine exakte Lichtverteilungsberechnung

durchzuführen

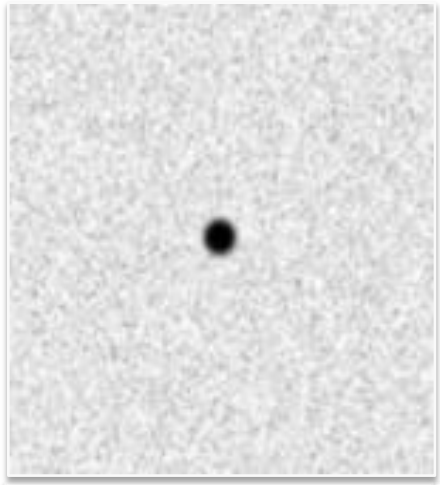
Cubic EM: 6 Maps (vorn, hinten, oben,...)

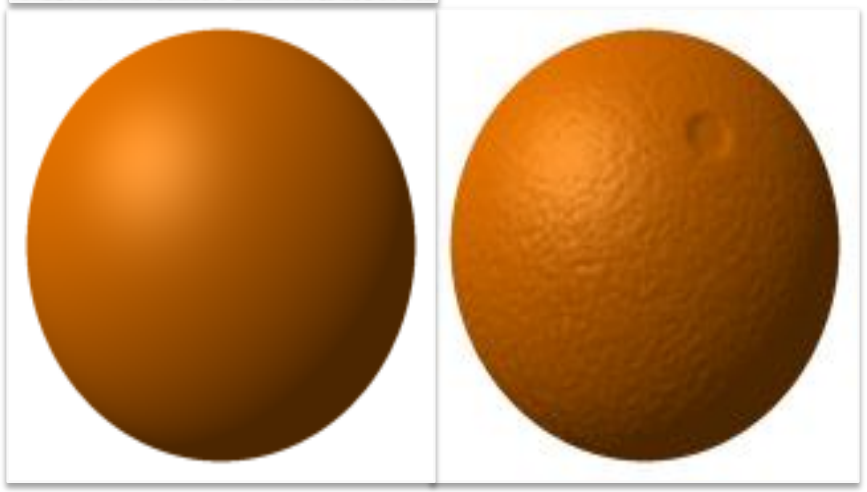
Spherical EM: eine Map für alle Seiten



Bump Mapping

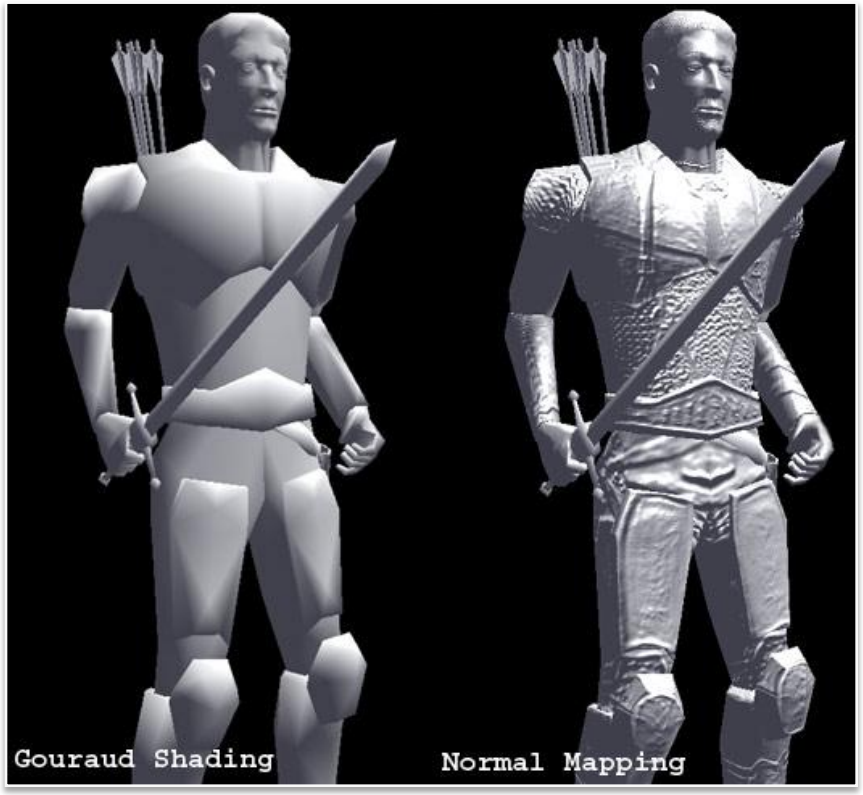
- Höhenunterschiede simuliert, die in der Geometrie des Modells gar nicht vorhanden sind
- Bump-Map:
  - besteht aus Grauwerten
  - Schwarz (Farbwert 0) die „tiefste“ Stelle
  - Weiß (Farbwert: 255) die „höchste“ Stelle





Normal Mapping

- Oberflächenunebenheiten simuliert, die in der Geometrie des Modells gar nicht vorhanden sind
- Normal-Map:
  - RGB-Werte als XYZ Werte interpretiert
  - normalisierter Vektor ist Normale für Helligkeitsberechnung
  - von Hi-Res-Modell errechnet und für Low-Res-Modell verwendet

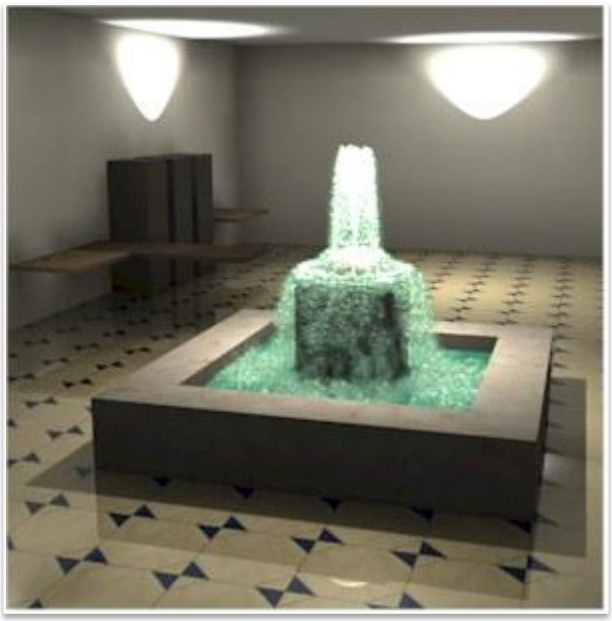


Displacement Mapping

- Höhenwerte aus Bump-Map werden zum erzeugen neuer und verschieben vorhandener Vertices benutzt
- Vorteil: Geometrie wird verändert
- Nachteil: deutlich mehr Rechenaufwand

Partikel-Systeme

- einfache Weise eine große Anzahl von Objekten zu animieren
  - Effekte (Feuer, Wolken, Rauch, Wasser)
  - Bewegegung vieler Objekte (Schwärme, Asteroidenfelder)



Zusammenfassung

- Rendering besteht aus Viewing und Shading
- Wichtige Bestandteile:
  - Transformationen bzw. Kameramodell
  - Entfernen verdeckter Modellteile
  - Beleuchtungsmodell
  - Shading (Anwenden des Beleuchtungsmodells)
- Unterschiedliche Verfahren in jedem Schritt möglich, daher große Breite von Effekten
- Effiziente Algorithmen und Datenstrukturen (effizienter Zugriff) für die jeweiligen Schritte nötig.

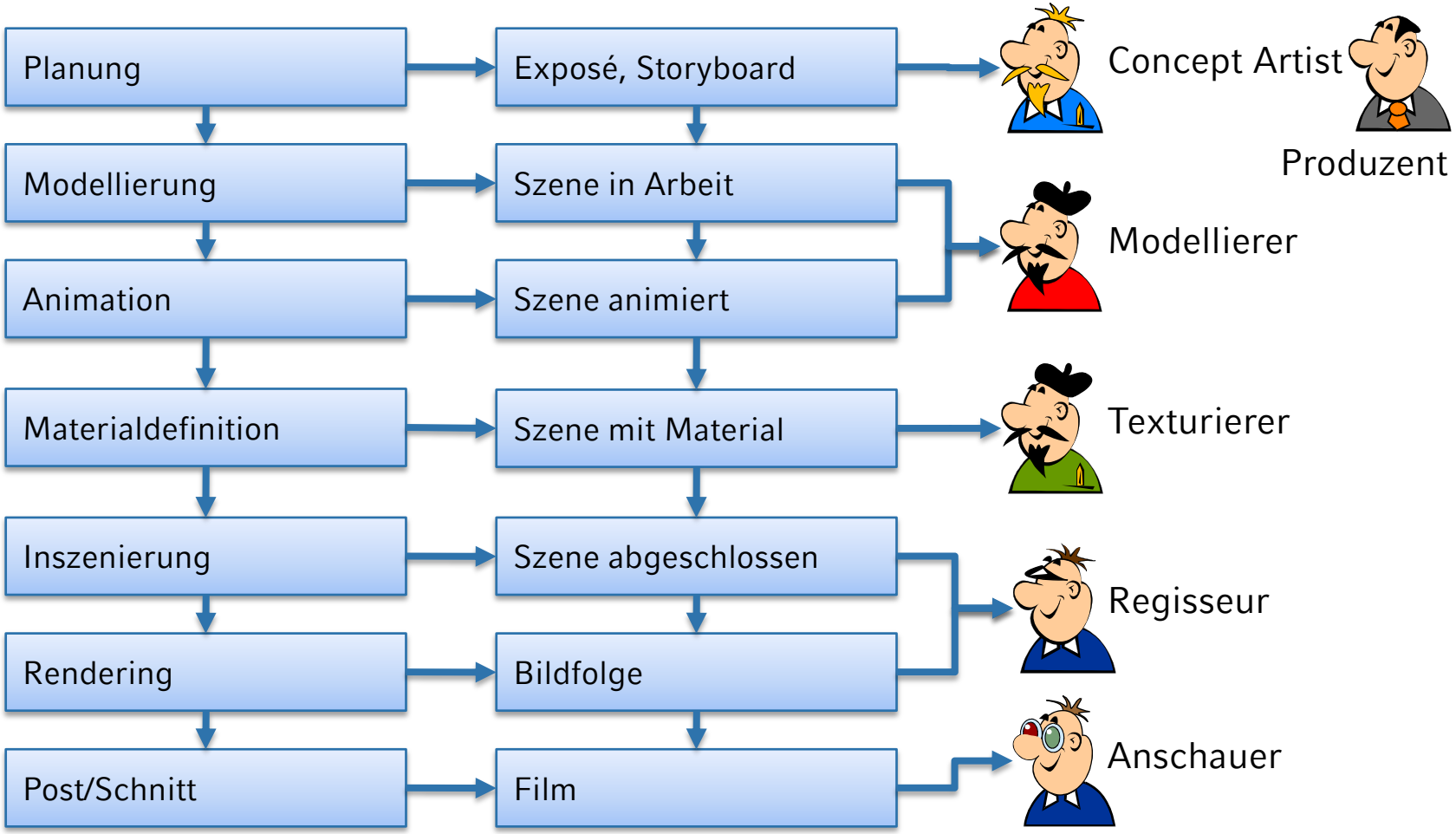
Zusammenfassung: Gestalterische Relevanz

- keine unendlich scharfen Kanten
- diffuse Reflexionen
- Schattenwurf
- Varianz
  - in Gestalt und
  - Oberfläche
- Ambient Occlusion (Dirt Maps)

Von der Vision zum Bild

Bildproduktions-Prozess

Bildproduktionsprozess



Vor der Animation: Erstellung der Geometrien

- Konstruktion
  - Konstruieren mit 3D-Primitiven
  - Konstruktion mit Freiformflächen
  - Extrusions- und Rotationskörper
  - Zusammensetzen aus 2D-Formen
  - zusammengesetzte 3D-Körper
- Modellieren auf polygonaler Ebene

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

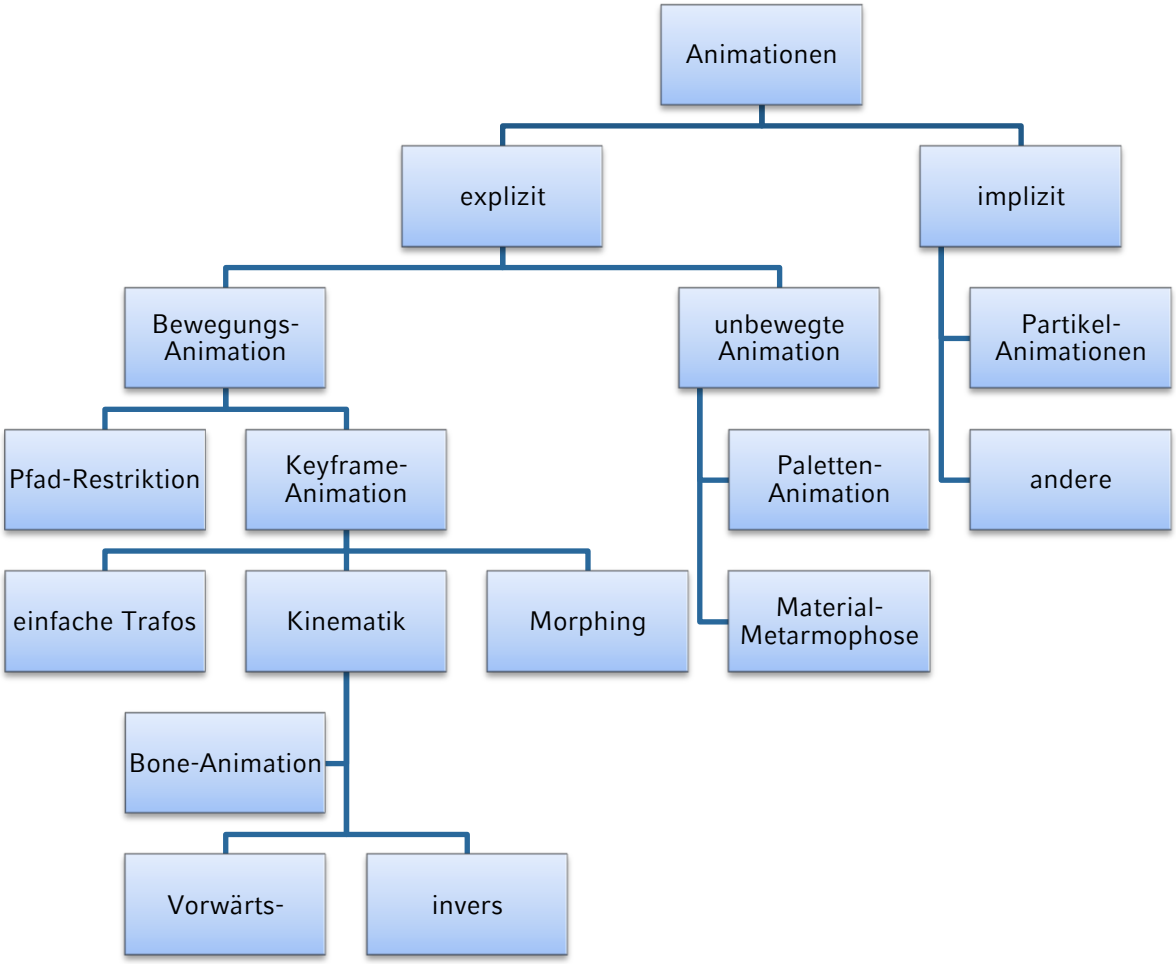
„Belebtes“ 3D

Animations-techniken

Animationen – Systematik

- Unterscheidung nach den Dimensionen:
  - 2D-Animation (Objekte und Kulisse)
  - 2½-Animationen (Eindruck von Räumlichkeit dank mehrerer Ebenen für u. a. Parallaxe-Scrolling)
  - 3D-Animationen

Animationstechniken



von 3D nach 2D

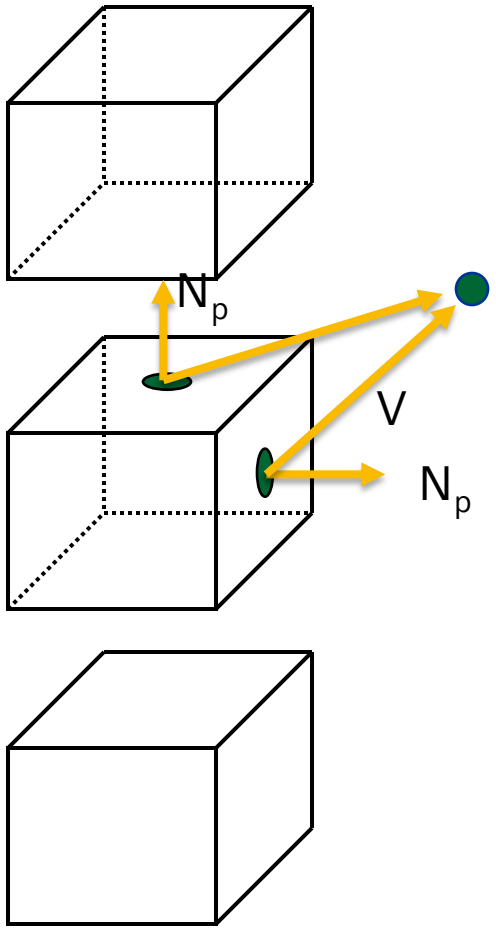
Sichtbarkeits-Bestimmung

Sichtbarkeitsbestimmung: Entfernen verdeckter Rückseiten

- Polygone entfernen, die vom Betrachter wegzeigen (Rückseiten), da sie mit Sicherheit nicht sichtbar sind (engl.: *back face culling*)
- Generelles CG-Prinzip: Versuche mit einem schnellen (einfachen) Test eine aufwändigere Berechnung in vielen Fällen einzusparen.

Sichtbarkeitsbestimmung: Entfernen verdeckter Rückseiten

- Einfacher Test auf Sichtbarkeit eines Polygons mit Hilfe von:
  - Polygonnormale N_p (nach außen zeigend)
  - Vektor v(iew) vom Polygon zum Betrachter-Standpunkt
- Ist der Winkel > 90°, dann zeigt die Normale (und damit die Außenseite des Polygons) vom Betrachter weg und das Polygon ist nicht sichtbar.

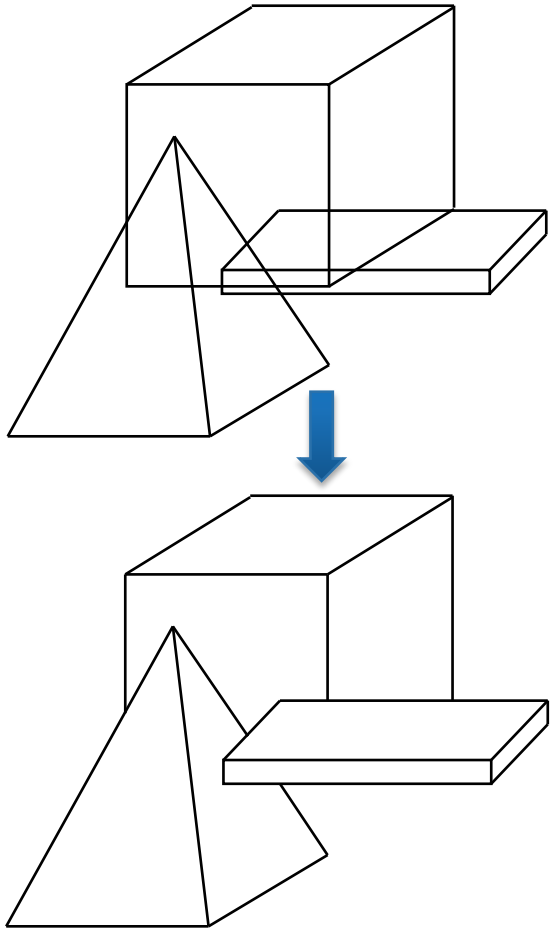


Sichtbarkeitsbestimmung: Entfernen verdeckter Rückseiten

- Backface Culling (Eliminieren von Polygonen mit einer vom Betrachter abgewandten Normale)
- Erster Schritt beim Entfernen verdeckter Polygone
- Vorgehen:
  - Bilde Skalarprodukt aus $N_p \cdot V$
  - Polygon ist prinzipiell sichtbar, wenn $N_p \cdot V > 0$ (und damit cos zwischen beiden Vektoren < 90°. Vorzeichentest ist schneller als Winkelberechnung)
  - wichtig: Vektoren vorher normalisieren
- Einfacher Test, weil Betrachterstandpunkt im Ursprung des Kamerakoordinatensystems
- Back face culling entfernt ~ 50% aller Polygone.

Sichtbarkeitsbestimmung: Entfernen verdeckter Rückseiten

- Bestimmen der Teile des Modells, die nicht von anderen Teilen des Modells verdeckt werden (oder der Teile, die verdeckt werden)
- Hidden Line bzw. Hidden Surface Removal



Sichtbarkeitsbestimmung: Entfernen verdeckter Kanten/Flächen

- Viele Verfahren zur Lösung der Aufgabe:
  - Unterschiedliche Herangehensweise
    - mit Objektgenauigkeit (d. h. analytische Verfahren in 3D)
    - mit Pixelgenauigkeit (d. h. Verfahren in „2D“)
  - Unterschiedliche Komplexität
  - Unterschiedliche Algorithmenklassen
  - Keine detaillierte Behandlung hier, nur depth-sort und z-Buffer-Algorithmus (später)

Sichtbarkeitsbestimmung: Entfernen verdeckter Kanten/Flächen

- Aus dem Kamara-Koordinatensystem werden die Koordinaten in der (2D-)Bildebene berechnet
- Projektionstransformation
- Änderung zu den bisher behandelten Transformationen: *z*-Werte bleiben erhalten, so dass jeder Objektpunkt einen Tiefenwert besitzt.
- Tiefenwerte werden für das Entfernen verdeckter Teile des Modelles beim *z*-Buffer-Algorithmus benötigt.

Sichtbarkeitsbestimmung: Entfernen verdeckter Kanten/Flächen

- Mögliche Konstellationen

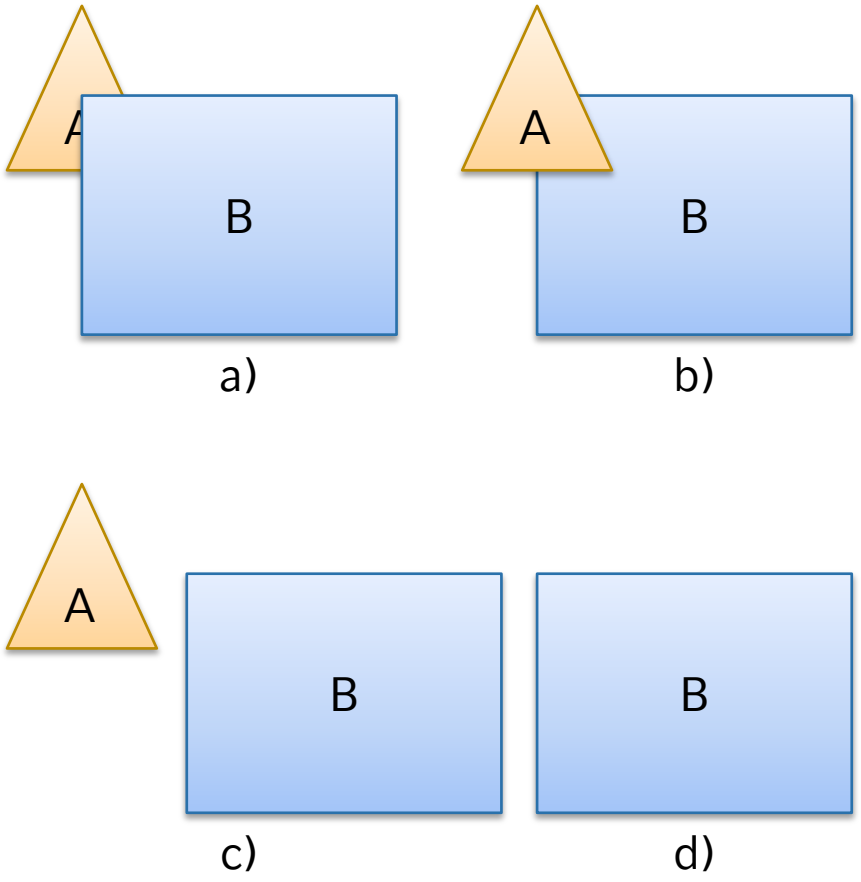
a) B vor A und Projektion B und A überlappt (B verdeckt A teilweise)

b) A vor B und Projektion A und B überlappt (A verdeckt B teilweise)

c) Projektion von A und B überlappt nicht (beide komplett sichtbar)

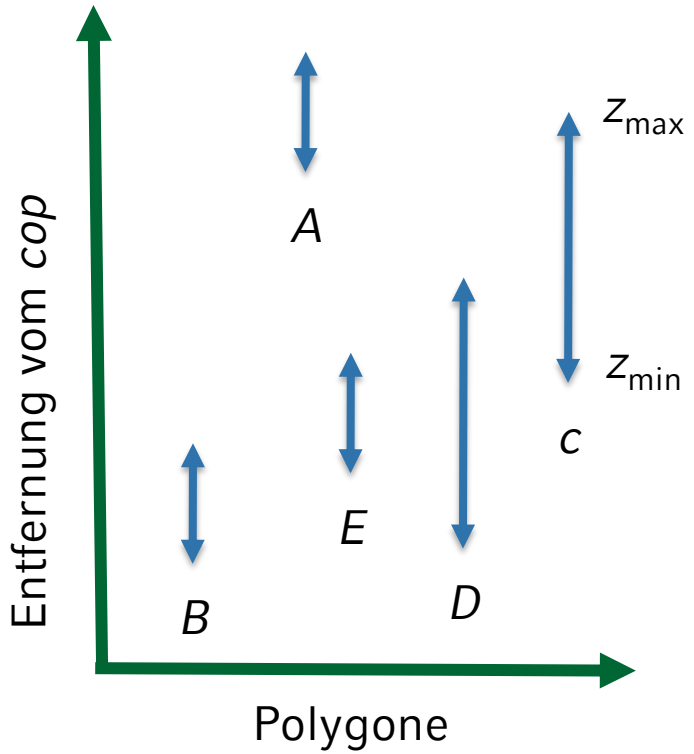
d) Ein Polygon vor dem anderen und Projektion des vorderen umschließt Projektion des hinteren (hinteres Polygon komplett verdeckt)

e) Projektion des vorderen Polygones komplett in der Projektion des hinteren enthalten (vorderes Polygon wird „über“ das hintere gezeichnet.)



Sichtbarkeitsbestimmung: Entfernen verdeckter Kanten/Flächen

- Depth-Sort (Objektbasierter Algorithmus)
- Sortieren der Polygone nach ihrer Entfernung zum *cop*
- Zeichnen von hinten nach vorn
- **Problem**: Bei Polygonen, bei denen sich die Abstände zum *cop* und die Projektion überlappen, ist die Zeichenreihenfolge unklar.



Sichtbarkeitsbestimmung: Entfernen verdeckter Kanten/Flächen

- **Lösung**:
  - Teile solche Polygone solange bis Polygonteile entstehen, die sich eindeutig sortieren lassen.
  - Dreiecke: Schwerpunkt bilden; 3 kleinere Dreiecke konstruieren

Sichtbarkeitsbestimmung: Z-Buffer-Algorithmus

- Effizienter Algorithmus zum Erkennen der sichtbaren Teile einer Szene, der in Bildschirmkoordinaten arbeitet (besser: im Bildraum)
- Ist in Hardware implementiert (NVidia, ATI, etc.).
- Voraussetzung: Alle Polygone liegen transformiert vor, wobei die Bildschirmkoordinaten eine transformierte z-Koordinate besitzen, die die Tiefe angibt.
- Technische Voraussetzung: zwei „Bitmaps“
  - Framebuffer enthält das eigentliche Bild
  - Z-Buffer enthält Tiefenwerte

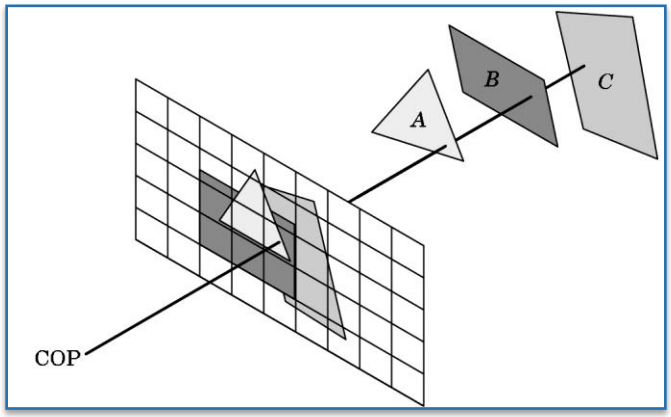
Sichtbarkeitsbestimmung: Z-Buffer-Algorithmus

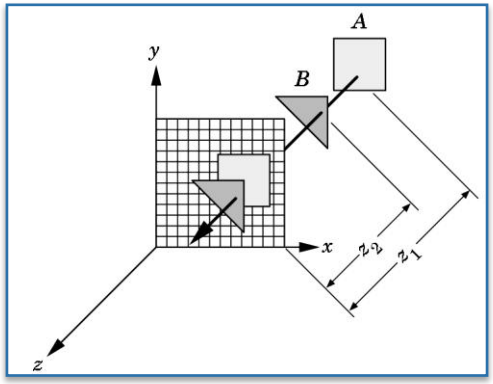
1. Fülle den Framebuffer mit der Hintergrundfarbe
2. Fülle den z-Buffer mit der maximal möglichen Tiefe
3. Zeichne alle Polygone nacheinander:
   - Bestimme für jeden Punkt des Polygons die Position im Bild, die Farbe und den Tiefenwert
   - Vergleiche den Tiefenwert an der Position mit dem, der dort im z-Buffer gespeichert ist
   - Wenn der Tiefenwert des Polygonpunktes kleiner ist (weiter vorne), dann zeichne die Farbe in den Framebuffer und setze den Tiefenwert im z-Buffer auf den neuen Tiefenwert, sonst wird nichts verändert

Sichtbarkeitsbestimmung: Z-Buffer-Algorithmus

- Was heißt „jeder Punkt des Polygons“?
  - Polygon wird abgetastet;
  - Berechnung wird für diskrete Punkte des Polygons durchgeführt.
  - Abtastrate hängt von der Größe des projizierten Polygones ab.

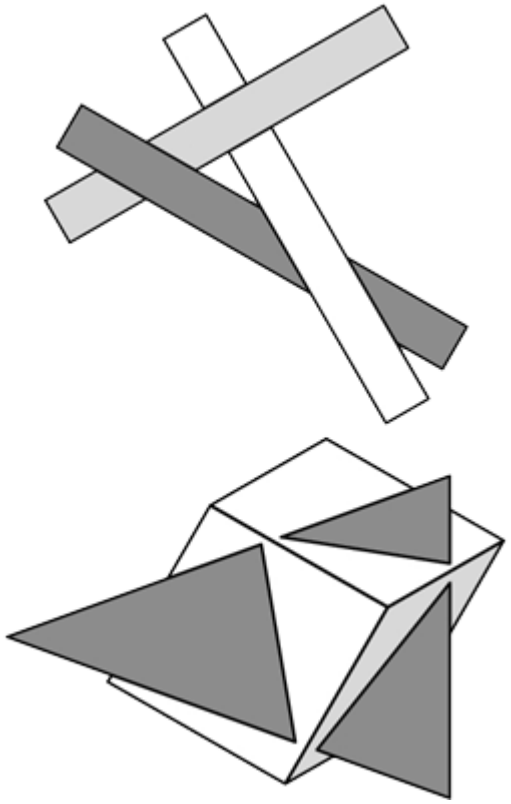
Sichtbarkeitsbestimmung: Entfernen verdeckter Kanten/Flächen





Sichtbarkeitsbestimmung: Entfernen verdeckter Kanten/Flächen

- Probleme bei der Sichtbarkeitsbestimmung:
  - Zyklisch überlappende Polygone
  - Ein Polygon „zersticht“ ein anderes (engl. *piercing*).
- Lösung: Teilen in solche Polygone, die sich eindeutig in Tiefenrichtung sortieren lassen (schwierig)



Zusammenfassung

- Entfernen von Rückseiten der Polygone (Normale und Vektor zur Kamera betrachten)
- Entfernen verdeckter Flächen(teile) durch bildbasierte oder objektbasierte Algorithmen
- Beispiele:
  - Objektbasiert: Sortierung der Polygone nach Abstand zur Kamera (Sortierreihenfolge nicht immer eindeutig)
  - Bildbasiert: z-Buffer

Weiter: Rasterung und Shading

- Nach der Projektion sind die Eckpunkte der Polygone in Bildschirmkoordinaten gegeben.
- Noch zu tun:
  - Umsetzung dieser Koordinaten auf Pixelpositionen
  - Bestimmen der entsprechenden Farbe des Pixels
  - „Zeichnen“ der Objekte = Rasterung
- Farbe des Pixels ist abhängig von:
  - Einfallendem Licht in der 3D-Szene
  - Material und Oberflächeneigenschaft der Objekte
  - Betrachterstandpunkt
- Bestimmen der Farbe an einem Punkt im 3D-Modell über Beleuchtungsmodelle, Umsetzen auf Pixelfarbe: Shading

von 3D nach 2D

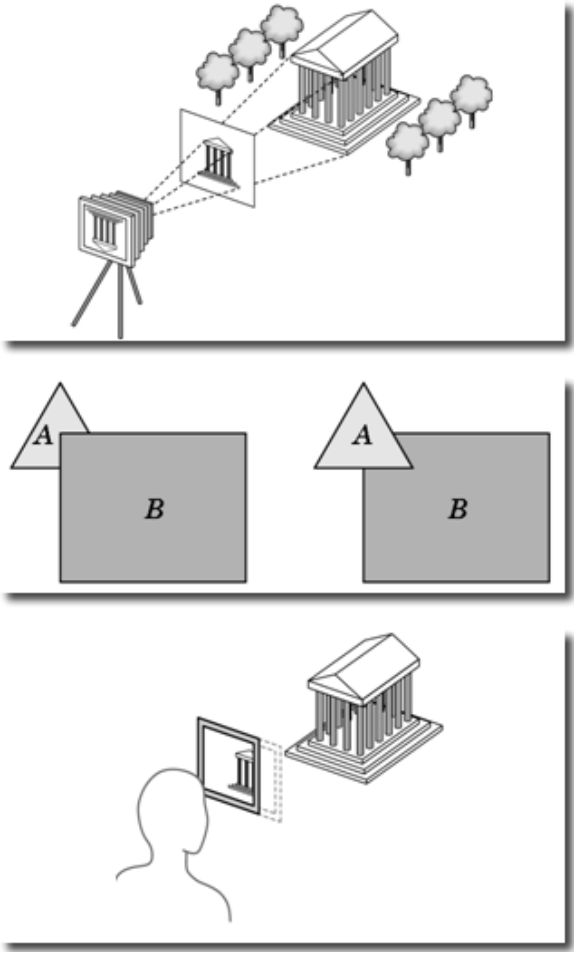
Clipping

Einleitung

- Motivation und Anwendung:
  - Clippen: Einleitung „Abschneiden“ einer geometrischen Beschreibung an einem Clipkörper.
- Anwendungen:
  - Fenstersysteme: Abschneiden der dargestellten Geometrie an den Fenstergrenzen (Bestandteil des Graphiksystems von Fenstersystemen)
  - Abschneiden von 3D-Geometrien am Sichtkörper (Pyramidenstumpf bzw. Quader)
  - Abschneiden eines verdeckten Polygons an einem davor liegenden Polygon (HSR)
  - Exploration von 3D-Daten mit Clipebenen

Einleitung: Anwendungen für Clipping

- *Oben:* Clippen am Sichtkörper
- *Mitte:* Clippen bei der Bestimmung verdeckter Kanten
- *Unten:* Clippen am Viewport



Einleitung

- Was heißt Clippen?

1. Entscheiden, welche Graphikprimitive komplett gezeichnet werden
2. Entscheiden, welche Teile von Graphikprimitiven gezeichnet werden

- → Schnittpunktberechnungen zwischen Graphikprimitiven und Clipgeometrie
- → Geschlossene Polygone bzw. Polyeder sind nach dem Abschneiden nicht mehr geschlossen. Berechnung von verbindenden Kanten/Flächen
- Beim Rendering muss das Clippen für jedes Polygon in 3d und für die sichtbaren Polygone nach Projektion in 2d durchgeführt werden. → Zahl der Schnittpunktberechnungen minimieren

Clippen von Liniensegmenten an Rechtecken

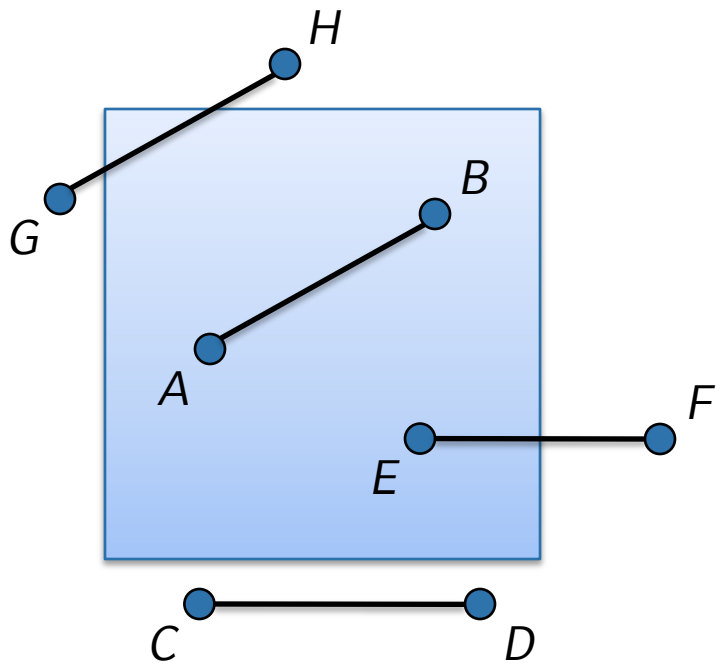
1. Punkte p_1 und p_2 innerhalb (*AB*)
   Linie komplett innerhalb (*accept*)

2. Punkte p_1 und p_2 außerhalb und jenseits der gleichen Kante (*CD*)
   Linie komplett außerhalb (*reject*)
   Bei 1. und 2. nutzt man Konvexität der Linie und des Clipkörpers

3. Punkte p_1 und p_2 außerhalb aber in Bezug auf verschiedene Kanten
   → testen, ob es zwei Schnittpunkte gibt (*GH*)

4. Punkte p_1 innerhalb p_2 außerhalb (*EF*) Berechne Schnittpunkt zwischen p_1, p_2 und dem Rechteck

5. Punkte p_1 außerhalb und p_2 innerhalb Vertauschen von p_1 und p_2 und dann vorgehen wie Fall 4



Clippen von Liniensegmenten an Rechtecken

- Test, ob Punkt p im achsenparallelen Rechteck (x_{min}, x_{max}, y_{min}, y_{max}):

```
If (p(x)>=xmin && p(x)<=xmax && p(y)>=ymin &&
p(y)<=ymax)

inside = true;
```

- Schnittpunktberechnungen:
  1. Identifizieren der geschnittenen Rechteckkante Tests separat durchführen, ggf. kommen zwei Kanten in Frage (siehe EF)
  2. Schnittpunktberechnung mit dieser (diesen) Kante(n)

Clippen von Liniensegmenten an Rechtecken

- Cohen-Sutherland-Algorithmus: Bestimme für jeden Eckpunkt (x, y) ein Bitfeld (outcode), das angibt, ob Eckpunkt jenseits des entsprechenden Wertes ist.
- 1. Bit: $y > y_{max}$,
- 2. Bit: $y < y_{min}$,
- 3. Bit: $x > x_{max}$,
- 4. Bit: $x < x_{min}$

1001	1000	1010
0001	0000	0010
0101	0100	0110

Clippen von Liniensegmenten an Rechtecken

- Cohen-Sutherland-Algorithmus:
- outcode (p1) und outcode (p2) seien die Bitfelder der Eckpunkte:

```
if (outcode (p1) == 0000 && outcode (p2) == 0000)
   accept; //p1 und p2 innerhalb
if (outcode (p1) & outcode (p2) != 0000) // bitweises „und"
   reject;
if (outcode (p1) | outcode (p2) != 0000) // bitweises oder
   calculate_intersection (p1, p2, x_min, x_max, y_min,
   y_max);
```

- Schnittpunktberechnung hängt von der Linienrepräsentation ab. Parametrische Darstellung günstig, weil keine Sonderfälle (z. B. vertikale Linien) auftreten.

Clippen von Liniensegmenten an Rechtecken

Parametrische Liniendarstellung:

$$p(t) = (1 - t)\, p_1 + t\, p_2$$

Für $t \geq 0$ und $t \leq 1 \rightarrow$ Linie von p_1 zu p_2.

Wenn es einen Schnittpunkt *schn* zwischen (p_1, p_2) und einer Rechteckkante (r_1, r_2) gibt, dann gibt es Parameter s und t, mit s in (0, 1) und t in (0, 1) so dass

$$schn = (1 - s)\, r_1 + s\, r_2 = p(t).$$

Lösung: Gleichungen für x und y-Komponente der Punkte aufstellen und gleichsetzen (2 Gleichungen; 2 Unbekannte)

Clippen von Liniensegmenten an Rechtecken

- **Beispiel:** Schnittpunkt-Berechnung der Linie (x_1, y_1) (x_2, y_2) mit der unteren Rechteck-Kante (x_{min}, y_{min}) (x_{max}, y_{min})
- Schnittpunkt-Berechnung an horizontalen und vertikalen Kanten vereinfacht sich

$$\text{I} \quad (1-t)x_1 + tx_2 = (1-s)x_{min} + sx_{max}$$

$$\text{II} \quad (1-t)y_1 + ty_2 = (1-s)y_{min} + sy_{max}$$

$$(1-t)y_1 + ty_2 = y_{min}$$

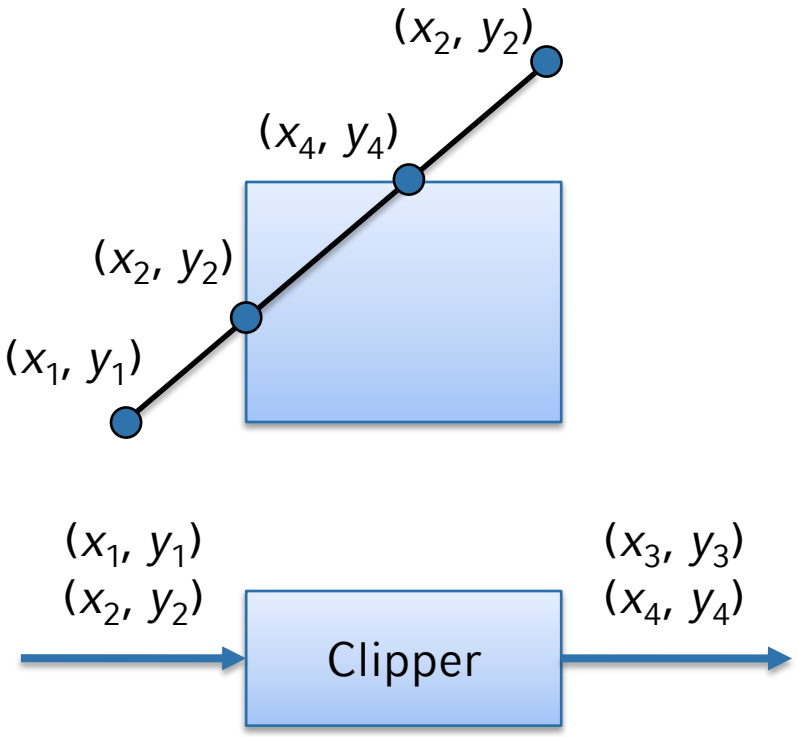
$$t - y_1 + ty_2 = y_{min} - 1$$

$$t = \frac{y_{min} - y_1}{y_2 - y_1}$$

Clippen von Liniensegmenten an Rechtecken

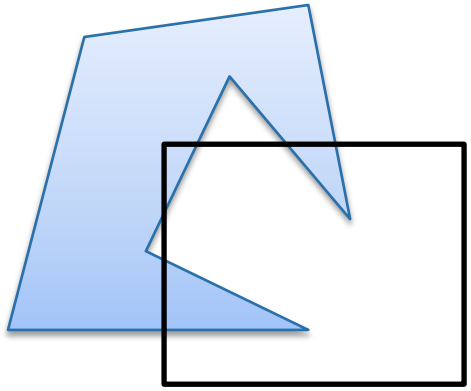
- Clipprozedur für eine Linie (x_1, y_1) (x_2, y_2) an einem Rechteck erzeugt neue Linie (x_3, y_3) (x_4, y_4), wobei
  - (x_3, y_3) und (x_1, y_1) sowie
  - (x_2, y_2) und (x_4, y_4)

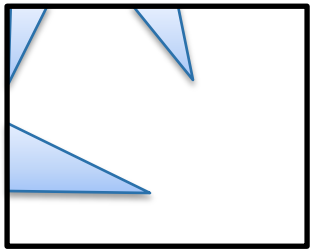
  gleich sein können



Clippen von Polygonen an Rechtecken

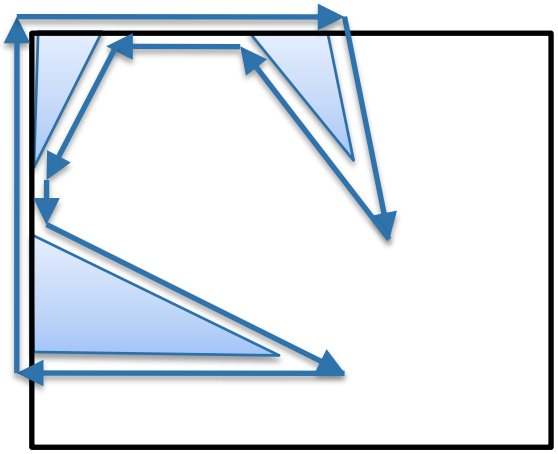
- Grundsätzlich kann das Clippen von Polygonen (p_1, p_2, …, p_n) an Rechtecken durch sukzessives Clippen der Linien (p_i, p_{i+1}) durchgeführt werden.
- **Problem:** Konkave Polygone → Durch Clippen kann ein konkaves Polygon in mehrere Polygone zerfallen.
- **Lösungsmöglichkeiten:**
  - Unterteilung des Polygons in konvexe Polygone (z. B. Triangulierung) oder
  - Einführen von zusätzlichen Kanten



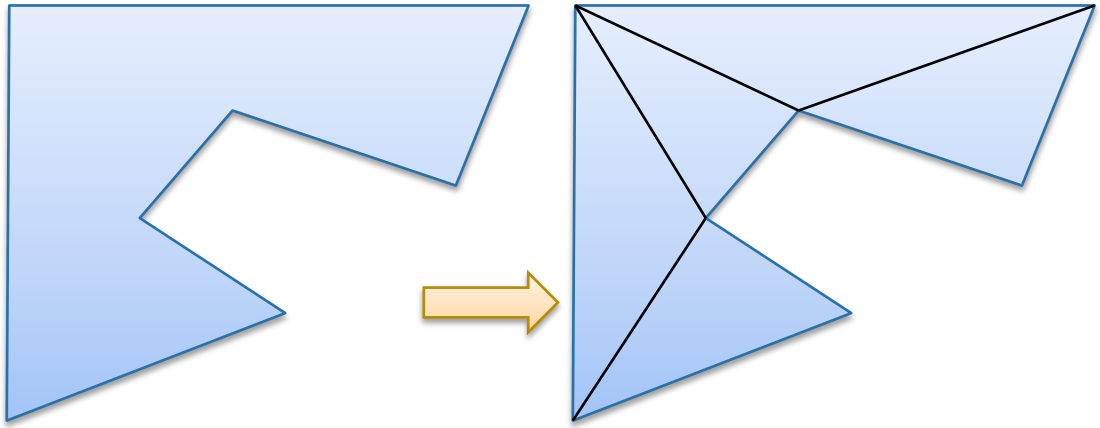


Clippen von Liniensegmenten an Rechtecken

Clippen von konkaven Polygonen



Einfügen von zusätzlichen Kanten, um geschlossenes Polygon zu repräsentieren



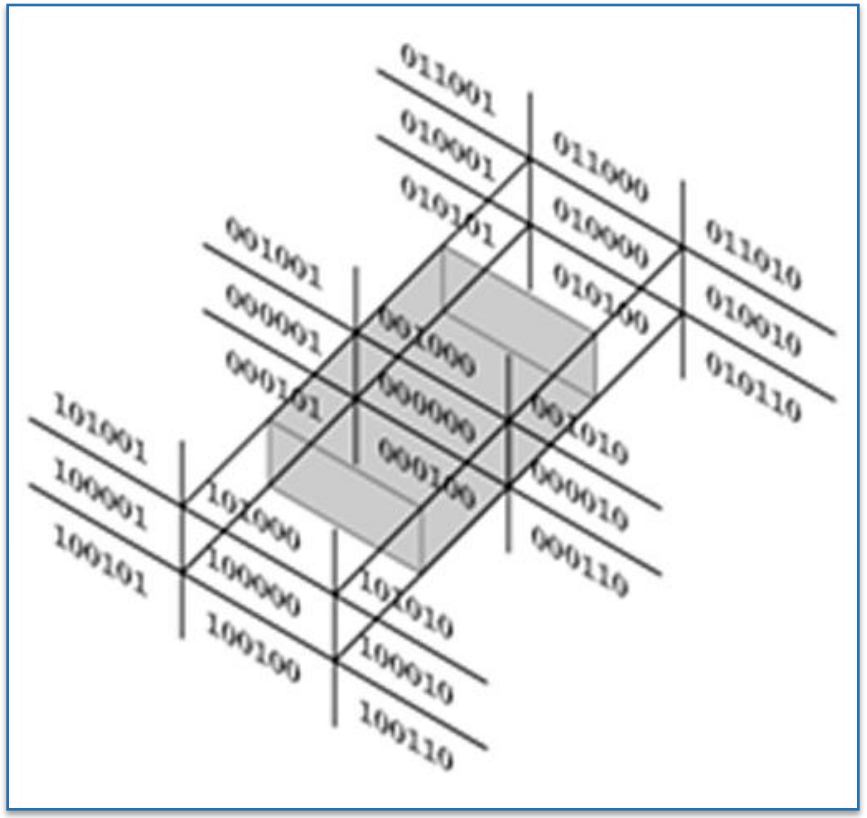
Triangulierung eines konkaven Polygons. Zusammenhang zwischen Dreiecken und Polygon muss repräsentiert werden

Clippen in 3D

- Wesentlicher Unterschied zwischen 2D und 3D:
- 2D: Schnittpunktberechnungen zwischen zwei Linien (Clipgeometrie durch Begrenzungslinien charakterisiert)
- 3D: Schnittpunkte liegen i. a. nicht auf den Kanten der Clipgeometrie. Schnittpunktberechnungen zwischen Linien und Flächen.

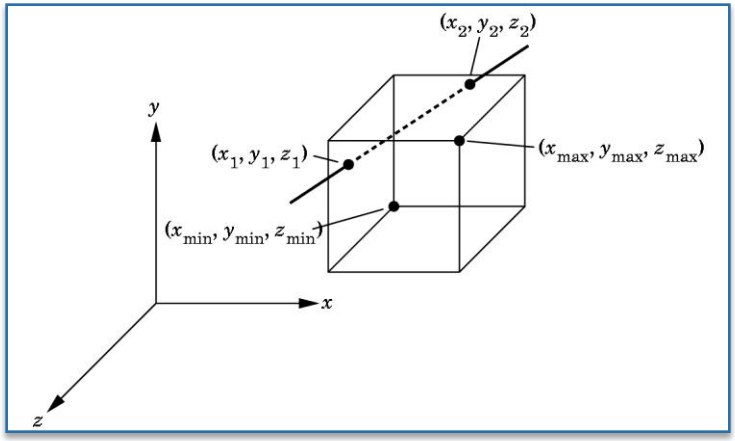
Clippen in 3D

- Erweiterung des Cohen-Sutherland-Algorithmus durch Ergänzen der Bitfelder pro Eckpunkt (zwei zusätzliche Bits repräsentieren Vergleich mit z_{min} und z_{max})



Clippen in 3D

- Für Schnittpunkt-Berechnung in parametrischer Form wird eine 3. Gleichung ergänzt, die sich auf die *z*-Koordinaten bezieht.



Clippen in 3D

- Clippen von Linien gegen achsenparallele Quader (Sichtbereich bei der Parallelprojektion). Vereinfachung wie im 2D-Fall.
- Clippen von Linien gegen Pyramidenstumpf (perspekt. Projektion): Scherung des Sichtkörpers → achsenparalleler Quader entsteht.
- Bei einer Pipeline-Architektur wird sukzessive gegen die sechs Begrenzungsflächen eines Quaders getestet.

Effizientes Clippen komplexer geometrischer Objekte

- Schnittpunktberechnungen sind sehr aufwändig für
  - Große Polygone
  - Parametrische Kurven
  - Text
- Intern wird Text häufig durch parametrische Kurven repräsentiert (Skalierbarkeit) und diese werden in Polygone umgewandelt.
- Schnittpunktberechnungen sind sehr effizient für einfache Formen (Kugeln, Quader).
- Effiziente Verfahren nutzen einfache (konvexe) Formen als Hüllkörper und testen, ob Hüllkörper geclippt werden muss.
- Nur wenn Hüllkörper teilweise innerhalb und teilweise außerhalb der Clipgeometrie ist, werden aufwändigere Tests durchgeführt (Raytracing).

Quellen

- Foley, van Dam, Feiner, Hughes. *Computer Graphics, Principles and Practice*. Zweite Auflage, Addison Wesley. ISBN 0-201-84840-6.
- Bernhard Preim. *Computergraphik 1*. Universität Magdeburg, Vorlesungsskript, Juli 2005.