Seminar Computergraphik I
Spezielle Probleme und Verfahren der Sichtbarkeitsbestimmung

Matthias Köppe

20. Januar 1997

1 Grundlagen

Gegeben sei ein geometrisches Objekt im $\mathbb{R}^3$ und eine Projektion $\psi : \mathbb{R}^3 \to \mathbb{R}^2$, die die Punkte des Objektes in einen 2dimensionalen Bildraum abbildet. Ziel der Sichtbarkeitsbestimmung ist es, die Teile des Objektes zu ermitteln, die bei ψ nicht von anderen Teilen „verdeckt“ werden.

Der Anschaulichkeit halber beschränken wir uns bei den folgenden Betrachtungen auf *ebene geometrische Projektionen*, welchen das folgende Konzept zugrunde liegt:

In den $\mathbb{R}^3$ ist eine *Projektionsebene* eingebettet. Durch jeden Objektpunkt P verläuft eine Gerade, der *Projektor* von P, welche sich mit der Projektionsebene in einem Punkt, dem *Bildpunkt* P', schneidet. Unter der *Projektorstrecke* eines Punktes P wollen wir die Strecke PP' verstehen.

Die Klasse der ebenen geometrischen Projektionen umfaßt die *Parallelprojektionen*, in denen alle Projektoren zueinander parallel sind, und die *perspektivischen Projektionen*, in denen die Projektoren sich in einem gemeinsamen Punkt, dem *Augpunkt*, schneiden.

Die eingeführten Begriffe gestatten nun eine anschauliche Definition der Sichtbarkeit eines Punktes:

Definition Ein Punkt P heißt *sichtbar* (bezüglich der Projektion ψ), wenn die Projektorstrecke von P von keinem anderen Objekt geschnitten wird.

Es werden die folgenden Zielstellungen der Sichtbarkeitsbestimmung unterschieden:

Visible-line determination Ermittlung der sichtbaren Teile der Kanten

Visible-surface determination Ermittlung der sichtbaren Teile der Flächen

Die dazu verwendeten Verfahren lassen sich in zwei Klassen einteilen:

Objektraum-Algorithmen Analytische Verfahren zur Bestimmung der sichtbaren Teile in der Genauigkeit der Objektdefinition

Bildraum-Algorithmen Verfahren zur Bestimmung der Sichtbarkeit für ein Pixelraster

2 Übersicht über verfügbare Algorithmen

2.1 Objektraum-Algorithmen

Visible-line-Reporting mit Brute Force In der Vorlesung wurde ein Verfahren vorgestellt, das jede Kante mit der Projektion jedes Polygons schneidet und die unsichtbaren Abschnitte in einem AVL-Baum verwaltet. Dieses Verfahren wurde durch Techniken wie Divide-and-Conquer und Einteilung des Objektraums mittels eines Zellrasters in der Laufzeit verbessert.

Visible-line-Reporting nach Appel Hier wird ein alternatives Grundverfahren vorgestellt, das Kantenkohärenz und Back-face-Culling ausnutzt. Es kann mit den gleichen Techniken in der Laufzeit verbessert werden.

2.2 Bildraum-Algorithmen

Scan-line-Verfahren Hier wird ein verallgemeinertes Scan-line-Verfahren zur Rasterkonvertierung von Polygonen vorgestellt, das Kantenkohärenz ausnutzt.

z-Buffer-Verfahren In der Vorlesung wurde die Verwendung eines z-Buffers zur Sichtbarkeitsbestimmung in Szenen aus allgemeinen Oberflächen vorgestellt. Das Verfahren benötigt sehr viel Speicher, läßt sich aber leicht in Hardware implementieren.

Scan-line-z-Buffer-Verfahren Hier wird ein Verfahren vorgestellt, das die Vorteile eines z-Buffers mit dem eines Scan-line-Algorithmus verbindet und für extrem hohe Auflösungen geeignet ist.

2.3 Hybrid-Algorithmen: Listenprioritätsverfahren

Die Listenprioritätsverfahren stehen als Hybride zwischen den Objektraum- und Bildraumverfahren. Es wird analytisch eine zulässige Darstellungsreihenfolge für die Polygone ermittelt. Bei der eigentlichen Darstellung wird dann die Fähigkeit der Ausgabegeräte genutzt, Pixel überschreiben zu können.

Tiefensortierung Dieses Verfahren arbeitet in drei Schritten:

1. Sortiere die Polygone nach der Entfernung vom Augpunkt
2. Behebe Probleme (etwa bei zyklischen Überlappungen oder einander durchdringenden Polygonen) durch Zerteilung der Polygone
3. Stelle die Polygone von hinten beginnend dar.

Der Schritt 2 erfordert recht aufwendige Tests und etwas Computational Geometry. Das Verfahren wird hier nicht weiter erörtert.

Painter's Algorithm Dieses Verfahren ist eine Vereinfachung des Algorithmus mit Tiefensortierung für Probleme, in denen explizite Prioritäten vorgegeben sind („$2\frac{1}{2}D$-Probleme"), etwa für Fenstersysteme oder VLSI-Layout.

Verwendung von Binary Space-Partitioning Trees Dies ist ein sehr elegantes Verfahren, das sich besonders gut eignet für Anwendungen, in denen die Szene konstant bleibt, aber der Augpunkt sich häufig ändert. Es erfordert eine aufwendige Vorverarbeitung zum Aufbau der Datenstruktur. Für jede Wahl des Augpunkts läßt sich dann effizient eine zulässige Darstellungsreihenfolge ermitteln.

Algorithm Appel;
begin
  for jedes Objekt O **do**
  begin
    Sei v eine beliebige Ecke von O;
    Berechne die Unsichtbarkeit $\delta(v)$ von v;
    $U := \emptyset$; $V := \{v\}$; $W := \emptyset$;
    while $V \neq \emptyset$ **do**
    begin
      Sei $v \in V$;
      for alle Nachbarn w von v mit $vw \notin U$ **do**
      begin
        Folge der Kante vw und ermittle $\delta(w)$;
        $U := U \cup \{vw\}$;
        if $w \notin W$ **then** $V := V \cup \{w\}$
      end;
      $V := V \setminus \{v\}$; $W := W \cup \{v\}$
    end
  end
end.

Abbildung 1: Das Verfahren von Appel

3 Das Verfahren von Appel et al. (1967)

Das Verfahren von Appel ist ein Visible-Line-Algorithmus für Szenen aus (durchdringungsfreien) ebenen Polygonen. Um die Beschreibung zu vereinfachen, wollen wir davon ausgehen, daß jedes Polygon eine Seitenfläche eines Polyeders ist. Zudem möge kein Clipping der Szene stattgefunden haben, so daß alle Seitenflächen der Polyeder in der Szene enthalten sind.

Unter diesen Voraussetzungen können wir die Eigenschaft verwenden, daß die Rückseiten von Polyedern auf die Sichtbarkeit von Punkten keinen Einfluß haben.

Definition Ein Polygon P heißt *back-facing*, falls nach außen orientierte Flächennormalen n_P des Polygons vom Betrachter wegzeigen. Anderenfalls heißt das Polygon *front-facing*.

Ist also eine Szene mit Polyedern gegeben, in der alle Flächennormalen nach außen orientiert sind, so kann man alle Polygone mit vom Betrachter wegzeigenden Flächennormalen weglassen. Diese allgemeine Technik wird als *Back-Face Culling* bezeichnet.

Das Verfahren von Appel schreibt nun jedem Punkt v der Szene seine *quantitative Unsichtbarkeit* $\delta(v)$ zu. Sie ist definiert als die Anzahl der front-facing Polygone, die den Punkt verdecken.

Die quantitative Unsichtbarkeit wird zunächst für eine Ecke mit einem Brute-Force-Verfahren bestimmt und auf den Kanten verfolgt. Dabei wird ausgenutzt, daß eine Veränderung nur an *Umrißkanten* stattfindet, d. h. an Kanten, an denen front-facing und back-facing Polygone zusammenstoßen.

In der Abbildung 2 ist die Kante AB keine Umrißkante; an ihr verändert sich die quantitative Unsichtbarkeit nicht.

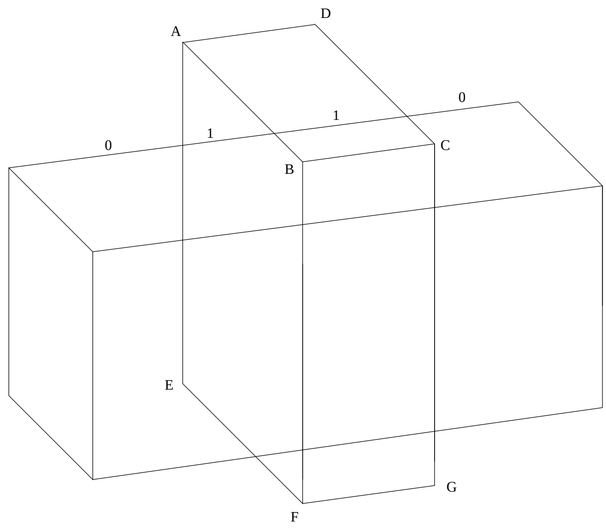

Abbildung 2: Quantitative Unsichtbarkeit von Segmenten

$$\begin{tikzcd}
\mathbb{P}^3 \arrow[r, "T_1"] & \mathbb{P}^3 \arrow[r, "P"] & \mathbb{P}^3 \arrow[r, "T_2"] & \mathbb{P}^3 \arrow[d, "\varphi^{-1}"] \\
\mathbb{R}^3 \arrow[u, "\varphi"] \arrow[rrr] \arrow[rrrd, "\psi"] & & & \mathbb{R}^3 \arrow[d, "\pi"] \\
 & & & \mathbb{R}^2
\end{tikzcd}$$

Abbildung 3: Projektion als Vorverarbeitungsschritt

4 Projektion als Vorverarbeitungsschritt

Angenommen, es ist eine perspektivische Projektion vorgegeben. Um zu bestimmen, ob ein Objektpunkt einen anderen verdeckt, müssen Algorithmen wie das Verfahren von Appel die Entfernungen der Punkte vom Augpunkt miteinander vergleichen. Diese Vergleichsoperation ist recht aufwendig; sie erfordert zwei Divisionen.

Die Laufzeiteigenschaften lassen sich verbessern, indem als Vorverarbeitungsschritt alle Punkte einer perspektivischen Transformation derart unterworfen werden, daß die kanonische Projektion π der Transformierten,

$$(x, y, z) \overset{\pi}{\longmapsto} (x, y),$$

gleich der perspektivischen Projektion ψ der Objektpunkte ist. Dadurch muß die aufwendige perspektivische Transformation nur einmal je Punkt, nicht je Vergleich ausgeführt werden. Sichtbarkeitstests lassen sich in transformierten Koordinaten mit einfachen Vergleichen realisieren.

In dem zugehörigen Diagramm in Abbildung 3 ist $\mathbb{P}^3$ der Raum der Äquivalenzklassen des $\mathbb{R}^4_* := \{(x, y, z, w) \in \mathbb{R}^4 \mid w \neq 0\}$ modulo Homogenität, die vermöge der Bijektion φ,

$$(x, y, z) \overset{\varphi}{\longmapsto} (x, y, z, 1) \cdot \mathbb{R}^*,$$

mit den Punkten des $\mathbb{R}^3$ identifiziert sind.

Die Transformation ist ein projektiv-lineare Abbildung (sie hat eine Matrix-Repräsentation in homogenen Koordinaten) und läßt sich aus geometrischen Transformationen T_1, T_2 und der perspektivischen Transformation P eines kanonischen Objektraums (siehe [1]),

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{\min}} & \frac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{pmatrix},$$

zusammensetzen.

5 Scanline-Verfahren

In der Vorlesung wurde ein Scanline-Verfahren zur Scan-Conversion gefüllter Polygone vorgestellt, welches Kantenkohärenz ausnutzt. Unter Verwendung ähnlicher Datenstrukturen können wir ein Verfahren angeben, welches das Visible-Surface-Problem für Szenen aus durchdringungsfreien Polygonen effizient löst.

Das Verfahren legt zunächst eine Kantentabelle (Edge Table, ET) an, die zu jeder Scanzeile die dort beginnenden Kanten auflistet. Horizontale Kanten müssen nicht gesondert betrachtet werden. Die Kanten werden in Bildraum-Koordinaten beschrieben.

Die Scanzeilen werden nun in ihrer natürlichen Ordnung behandelt. Dabei wird eine Tabelle aktiver Kanten (Active-Edge Table, AET) verwaltet. Wir betrachten nun die für jede Scanzeile ausgeführten Operationen.

Die in der aktuellen Scanzeile beginnenden Kanten aus der Kantentabelle werden hinzugefügt und die bereits abgelaufenen Kanten entfernt. Nach diesen Schritten wird sichergestellt, daß AET sortiert ist. Gewöhnlich wird AET mit einer doppelt verketteten Liste implementiert. Da angenommen werden kann, daß sich beim Übergang von einer Scanzeile zur nächsten nur wenig an der x-Ordnung ändert, ist Insertion Sort eine gute Implementation des Sortierens. Für sehr große Szenen könnte sich die Implementation von AET und der Einträge von ET als binomische Heaps lohnen: Der Sortierschritt entfällt, und die $\cup$-Operation ist effizient möglich.

Es wird nun eine Tabelle aktiver Polygone (Active-Polygon Table, APT) angelegt und nach den z-Werten an den aktuellen Stellen sortiert gehalten. Die AET wird von links nach rechts durchgegangen und die zugehörigen Polygone betrachtet:

- War das Polygon aktiv, so wird es hinter der aktuellen Kante inaktiv. War das Polygon unter den aktiven Polygonen am dichtesten am Betrachter, so muß die aktuelle Spanne gefüllt werden.

- War das Polygon inaktiv, so wird es hinter der aktuellen Kante aktiv. Ist das Polygon dichter am Betrachter als alle bisher aktiven Polygone, so muß die aktuelle Spanne gefüllt werden. Die lineare Ordnung in APT wird nun an den z-Werten an der aktuellen Stelle gemessen.

Die Eigenschaft eines Polygons, in APT enthalten zu sein, sollte in einem Flag des Polygons gespeichert werden.

Da wir von durchdringungsfreien Polygonen ausgehen, muß die lineare Ordnung nur dann neu definiert werden, wenn ein Polygon hinzugefügt wird. Selbst dann ist keine Neusortierung der Tabelle, sondern nur geeignete Einfügung des neuen Polygons erforderlich.

Nachdem alle Kanten aus AET behandelt wurden, ermittelt das Verfahren mit inkrementellen Berechnungen die x-Stellen auf den Kanten in AET für die nächste Scanzeile.

```
Algorithm Scanline;
begin
  Bucket-Sortiere die Kanten nach y_min in ET;
  AET := ∅;
  for alle Scan-Zeilen y do
  begin
    AET := AET ∪ ET[y];
    Lösche alle Kanten in AET mit y = y_max;
    Sortiere AET nach x;
    APT := ∅;
    Sei E eine Kante mit minimalem x in AET;
    while E existiert do
    begin
      Sei P das Polygon zu E;
      Sei x die Schnittstelle von E mit Scanzeile y;
      if P ∈ APT
      then begin
        if P ist erstes Polygon in APT
        then fülle die aktuelle Spanne in der Farbe von P;
        APT := APT \ {P};
      end
      else begin
        Sei Q das erste Polygon in APT;
        Sei die lineare Ordnung in APT bestimmt durch z an (x,y);
        APT := APT ∪ {P};
        if P ist jetzt das erste Polygon in APT
        then fülle die aktuelle Spanne in der Farbe von Q;
      end;
      Sei E die nächste Kante in AET
    end
  end
end.
```

Abbildung 4: Scanline-Algorithmus zur Lösung des Visible-Surface-Problems

```
Algorithm Scan-Line z-Buffer;
begin
  for alle Scan-Zeilen y do
  begin
    frameBuffer[x] := background   ∀x;
    zBuffer[x] := ∞   ∀x;
    for alle Polygone P, die die Scan-Zeile y schneiden do
    begin
      Sei z die Tiefe für P an (x, y);
      if z < zBuffer[x]
      then begin
        zBuffer[x] := z;
        Sei c die Farbe für Pixel (x, y) in P;
        frameBuffer[x] := c;
      end
    end
  end
end.
```

Abbildung 5: Prototyp eines Scan-line-z-Buffer-Algorithmus

6 Scan-line-z-Buffer-Verfahren

Ein z-Buffer-Verfahren führt zu einem vorgegebenen Pixelraster nicht nur einen Frame-Buffer (für Farbwerte), sondern auch einen z-Buffer (für Tiefenwerte). Objekte werden in einer beliebigen Reihenfolge scan-konvertiert, ein Pixel aber nur dann in den Frame-Buffer geschrieben, wenn die zugehörige Tiefe geringer ist als der bisherige z-Buffer-Eintrag.

Ein z-Buffer für komplette Szene benötigt einen Speicherplatz, der in der Auflösung quadratisch ist. Falls in speziellen Anwendungen oder bei extrem hohen Auflösungen nicht genügend Speicherplatz zur Verfügung steht, empfiehlt sich die Verwendung eines einzeiligen z-Buffers in einem Scan-line-z-Buffer-Verfahren. Ein Prototyp ist in Abbildung 5 angegeben.

Offensichtlich läßt sich dieser Algorithmus mit Datenstrukturen aus dem vorgestellten Scan-line-Verfahren effizient implementieren. Eine Tabelle aktiver Kanten und ein Zähler für die aktiven Kanten eines jeden Polygons ermöglichen es etwa, die in einer Scan-Zeile höchstens sichtbaren Polygone zu ermitteln.

Für die Tiefenberechnung (z-Interpolation) verwendet man inkrementelle Verfahren.

7 Aliasing

Bei den vorgestellten Objektraum-Verfahren können bei sorgloser Implementation Aliasing-Effekte auftreten.

Betrachte etwa eine Kante, in der zwei Polygone zusammenstoßen: Im angegebenen Scanline-Algorithmus entscheidet die z-Koordinate, welches der beiden Polygone vorn liegt, und zwar für die gesamte Strecke, auf der die Polygone aktiv sind. Da die z-Koordinaten bei exakter Rechnung gleich sind, können numerische Probleme den Vergleich ein zufälliges Ergebnis liefern lassen. Dadurch könnte eine ganze Spanne in einer falschen Farbe gefüllt werden.

Das Problem läßt sich etwa durch geeignete Wahl einer ε-Schranke für z-Differenzen beseitigen, unterhalb derer nicht die z-Differenz betrachtet wird, sondern die

Differenz der Gradienten $\frac{\partial z}{\partial x}$, in einem gewissen Sinn also z-Differenzen für innerhalb beider Polygone gelegene Punkte.

Auch bei z-Buffer-Verfahren können an gemeinsamen Kanten Aliasing-Effekte auftreten. Es ist aber dem z-Buffer inhärent, daß die Effekte auf einzelne Pixel begrenzt sind.

Literatur

[1] Foley, James D., Andries van Dam, Steven K. Feiner und John F. Hughes: *Computer Graphics: Principles and Practice. Second Edition in C.* The Systems Programming Series. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1996.

[2] Schmitt, A., O. Deussen und M. Kreeb: *Einführung in graphisch-geometrische Algorithmen.* B. G. Teubner, 1996.