Proseminar schnelle 3D–Grafik

Vortrag von Christian Schmidbauer

Linien– und Polygon–Clipping

1 Linien–Clipping

1.1 Grundlagen

Zuerst wollen wir einige Grundlagen zum Linien–Clipping betrachten. Erstens will ich expli– zit festhalten, wann ein einzelner Punkt (x,y) im Sichtfenster liegt. Dies trifft zu, wenn alle der folgenden Ungleichungen erfüllt werden:

$$x_{min} \leqslant x \leqslant x_{max} \quad \text{sowie} \quad y_{min} \leqslant y \leqslant y_{max} \quad .$$

Dabei bezeichnen (x_{min},y_{min}) und (x_{max},y_{max}) die Koordinaten der linken unteren sowie der rechten oberen Ecke des Sichtfensters, gegen das ein Clipping durchgeführt wird. Die x–Achse zählt dabei (wie gewohnt) von links nach rechts positiv und die y–Achse von unten nach oben positiv.

Liegen beide Endpunkte einer darzustellenden Linie im Sichtfenster, so kann man sie »trivial akzeptieren« und damit zeichnen (Dieser Test wird von allen Algorithmen durchgeführt, bevor sie eine Linie weiter bearbeiten, ich werde ihn aber bei der Beschreibung der einzelnen Algorithmen nicht mehr erwähnen). Liegt ein Endpunkt innerhalb und einer außerhalb des Sichtfensters, so muß der Schnittpunkt mit der von der Linie geschnittenen Begrenzungskante des Sichtfensters gefunden werden. Im dritten und letzten Fall, wenn beide Endpunkte der zu clippenden Linie außerhalb des Sichtfensters liegen, kann es keinen oder aber zwei Schnitt– punkte mit den Sichtfensterkanten geben, die bestimmt werden müssen.

Die »Brute–Force« Methode, um diese Schnittpunkte zu erhalten, besteht darin, die Gerade, welche unsere darzustellende Linie enthält, mit jeder der Geraden zu schneiden, die eine der Begrenzungskanten des Sichtfensters enthalten. Von den erhaltenen Schnittpunkten sind nur diejenigen relevant, welche sowohl in der zu clippenden Linie als auch in der entsprechenden Sichtfensterkante enthalten sind. Man kann dazu die Geradengleichungen beider Geraden in der Achsenabschnittsform aufstellen und eine gemeinsame Lösung für den Schnittpunkt bestimmen (man erhält also ein lineares Gleichungssystem mit zwei Gleichungen und zwei Unbekannten, das zu lösen ist). Bei dieser Art der Berechnung stößt man bei den vertikalen Begrenzungslinien des Sichtfensters allerdings auf Schwierigkeiten (unendliche Steigungen). Abhilfe schafft hier die Verwendung einer parametrisierten Beschreibung der Linien, sowohl der zu zeichnenden als auch der Sichtfensterkanten. Es bietet sich hier eine Formulierung der Liniengleichungen durch die vektorielle Punkt–Richtungs–Form an. In beiden Fällen ist aber vorab zu prüfen, ob die zu schneidenden Linien parallel sind und es deshalb keinen Schnittpunkt geben kann. Diese Art der Berechnung ist jedoch sehr aufwendig, und daher ineffizient. Es werden nun drei Möglichkeiten vorgestellt, wie man einige der Schnittpunktberechnungen beziehungsweise Vergleiche einsparen kann.

1.2 Entscheidungsfindung durch Zoneneinteilung

1.2.1 Der Cohen–Sutherland Algorithmus

Beim Cohen–Sutherland–Algorithmus werden zuerst Tests durchgeführt, ob eine Weiterbehandlung der gegebenen Linien sinnvoll ist. Die Linie kann für die obere Kante des Sichtfensters »trivial verworfen« werden, wenn beide y–Koordinaten der Endpunkte der zu zeichnenden Linie größer als die y–Koordinate der genannten Sichtfensterlinie sind. Für die untere Begrenzung müssen beide y–Koordinaten der Linie entsprechend kleiner sein als die y–Sichtfensterkoordinate. Für die vertikalen Kanten des Sichtfensters ergeben sich unter Verwendung der x–Koordinaten entsprechende Bedingungen. Sehr leicht lassen sich diese Entscheidungen treffen, wenn man für die beiden Endpunkte der zu zeichnenden Linie Outcodes mit vier Bit berechnet. Diese ergeben sich nach folgenden Bedingungen:

Erstes Bit gesetzt, wenn:	$y > y_{max}$
Zweites Bit gesetzt, wenn:	$y < y_{min}$
Drittes Bit gesetzt, wenn:	$x > x_{max}$
Viertes Bit gesetzt, wenn:	$x < x_{min}$

Für jede Gerade, in der eine der Sichtfensterkanten enthalten ist und die von der zu zeichnenden Linie geschnitten wird, wird also ein Bit gesetzt. Der oberen Kante des Sichtfensters entspricht somit das erste Bit, der unteren das zweite Bit, der rechten das dritte und der linken das vierte Bit des Outcodes.

Hiermit können dann zu zeichnende Linien sehr einfach »trivial verworfen« werden, denn es ist nur eine logische UND–Verknüpfung zwischen den Outcodes der Endpunkte zu berechnen, um dies entscheiden zu können. Ist das Ergebnis dieser boolschen Operation verschieden von Null, kann die Linie »trivial verworfen« werden. Ist dies nicht der Fall, so wird nach folgender Vorschrift verfahren:

- Wähle einen äußeren Endpunkt.
- Suche das erste Bit (von links) von dessen Outcode, das gesetzt ist, und unterteile die zu zeichnende Linie am Schnittpunkt mit der Geraden, welche die diesem Bit entsprechende Sichtfensterkante enthält.
- Ersetze den äußeren Endpunkt durch den eben berechneten Schnittpunkt, berechne den Outcode für diesen Schnittpunkt und behandle die erhaltene Linie wie eine zu zeichnende Linie.

Dabei kann es wie bei der Brute−Force−Berechnung vorkommen, daß in Zwischenschritten Schnittpunkte berechnet werden, die nicht auf einer Kante des Sichtfensters liegen. Da dies zu unnötigen Berechnungen führt, ist der Cohen−Sutherland−Algorithmus nicht der effizienteste unter den bekannten Algorithmen. Wegen des »trivialen Verwerfens« und des Einsparens von bis zu zwei Schnittpunktberechnungen bei günstigem Verlauf der zu zeichnenden Linie ist dieser Algorithmus aber effizienter als die Brute−Force−Berechnung.

1.2.2 Der Nicholl−Lee−Nicholl Algorithmus

Beim Nicholl−Lee−Nicholl Algorithmus wird ebenfalls wieder eine Zoneneinteilung der Zeichenebene vorgenommen. Jedoch ist hier das Vorgehen wesentlich komplexer als beim Cohen−Sutherland Algorithmus. Es werden grundsätzlich drei nichttriviale Grundfälle unterschieden:

1. Der betrachtete Anfangspunkt der zu zeichnenden Linie liegt ober− oder unterhalb des Sichtfensters.
2. Der betrachtete Anfangspunkt der zu zeichnenden Linie liegt links oder rechts vom Sichtfenster.
3. Der betrachtete Anfangspunkt der zu zeichnenden Linie liegt diagonal zwischen diesen Fällen.

Innerhalb dieser Grundfälle kann man Symmetriebetrachtungen verwenden, um jeweils nur einen Fall zu implementieren. Für jeden dieser Grundfälle wird jetzt die Zeichenebene durch die Verbindungsgeraden vom Anfangspunkt der Linie zu den Eckpunkten des Sichtfensters und durch die Begrenzungslinien des Sichtfensters in Zonen eingeteilt, für die explizit

angegeben werden kann, welche Sichtfensterkanten geschnitten werden, falls der Endpunkt der Linie in dieser Zone liegt. Diese Schnittpunkte können dann sehr effizient berechnet werden, da hierfür zahlreiche Zwischenergebnisse aus der Zonenzuordnung weiter verwendet werden können.
Um zu bestimmen, in welcher Zone der Endpunkt unserer zu zeichnenden Linie liegt, ist also eine Funktion nötig, die angibt, ob der Endpunkt auf einer bestimmten Seite einer Geraden liegt. Diese erhält man zum Beispiel, indem man das Vorzeichen des Skalarprodukts aus dem Vektor von einem beliebigen Punkt auf der zu vergleichenden Geraden zu dem Endpunkt und dem Normalenvektor der zu vergleichenden Geraden betrachtet.
Da beim Nicholl–Lee–Nicholl–Algorithmus nur diejenigen Schnittpunkte berechnet werden, die auch gebraucht werden, und dafür auch Zwischenergebnisse aus der Zonenzuordnung des Endpunktes wiederverwendet werden können, ist dieser Algorithmus der effizienteste unter den vorgestellten Algorithmen

1.3 Parametrische Linien–Clipping Algorithmen

1.3.1 Der Cyrus–Beck–Algorithmus

Unter Verwendung mathematischer Grundlagen aus der Vektorrechnung und einer parametrisierten Darstellung der zu zeichnenden Linie erhält man zur Berechnung des Parameters t_i eines Schnittpunktes der zu zeichnenden Linie und einer beliebigen Sichtfensterkante i untenstehende Formel, indem man ansetzt, daß das Skalarprodukt aus dem Normalenvektor der Sichtfensterkante und dem Verbindungsvektor von einem beliebigen Punkt E_i auf der Sichtfensterbegrenzung zu unserem gesuchten Schnittpunkt gleich 0 ist.
Als Formel geschrieben: $\vec{N}_i * (\vec{P}_0 + t_i \cdot \vec{D} - \vec{P}_{E_i}) = 0$.

Man erhält sodann durch Umformung: $t_i = \dfrac{\vec{N}_i * [\vec{P}_0 - \vec{P}_{E_i}]}{-\vec{N}_i * \vec{D}}$,

wobei $\vec{N}_i$ den nach außen gerichteten Normalenvektor zur Kante i und $\vec{P}_{E_i}$ den Orts–vektor eines Punktes (eine günstige Wahl sind hier natürlich die bekannten Eckpunkte des Sichtfensters) auf der Kante i bezeichnet. $\vec{D} = \vec{P}_1 - \vec{P}_0$ ist der Differenzvektor zwischen den beiden Endpunkten der zu zeichnenden Linie. Damit der Nenner nicht Null wird, müssen folgende Bedingungen geprüft werden:

$\vec{N}_i = 0$	Dies kann nur aus Versehen vorkommen, da $\vec{N}_i$ von uns für jede Kante frei (aber sinnvoll) selbst gewählt werden kann.
$\vec{D} = 0$	Dies würde bedeuten, daß beide Endpunkte der zu zeichnenden Linie identisch sind. Sollte als Eingabe für den Algorithmus ausgeschlossen sein.
$\vec{N}_i * \vec{D} = 0$	Die zu zeichnende Linie ist parallel zur gewählten Kante. Es kann keinen Schnittpunkt geben, man kann zur nächsten Kante oder Linie wechseln.

Aus den so berechneten Schnittpunktparametern für jede Kante des Sichtfensters werden nun alle aussortiert, die einen Wert außerhalb des Intervalls [0 , 1] haben, denn diese Schnittpunkte können nicht innerhalb der zu zeichnenden Linie sein. Für die verbliebenen Schnittpunkte wird nun bestimmt, ob sie aus Sicht eines Endpunktes die betrachtete Sichtfensterkante »potentiell eintretend« (falls $\vec{N}_i * \vec{D} < 0$) oder »potentiell verlassend« (für den Fall $\vec{N}_i * \vec{D} > 0$) schneiden.

Diese Information ist aber schon vorhanden, nämlich im Vorzeichen des Nenners in der Be–rechnung von t_i . Mit dieser Kategorisierung der Schnittpunktparameter können wir nun das Parameterpaar (oder den Parameter, falls ein Endpunkt im Inneren des Sichtfensters liegt) auswählen, mit welchem die sichtbare Linie gezeichnet werden kann. Konkret sind dies:

- Der »potentiell eintretende« Parameter mit dem größten Wert für t_i und
- Der »potentiell verlassende« Parameter mit dem kleinsten Wert für t_i .

Dieser Algorithmus unterscheidet sich nur wenig von der »Brute–Force«–Berechnung und ist daher kaum effizienter als diese. Nur durch die Verwendung von Zwischenergebnissen können hier Pluspunkte gesammelt werden. Jedoch ist bei diesem Algorithmus die Form des Sichtfensters beliebig.

1.3.2 Der Liang–Barsky Algorithmus

Der Liang–Barsky Algorithmus ist nur eine Abwandlung des Cyrus–Beck Algorithmus für rechteckige und aufrechte Sichtfenster. Es werden dabei in der Berechnung des Schnittpunkt–parameters t_i die mathematischen Konsequenzen dieser speziellen Sichtfenstergeometrie berücksichtigt und unnötige Berechnungen vermieden.

Es sind dann nur noch folgende Berechnungen durchzuführen:

Kante	Normalenvektor	$\vec{P}_{E_i}$	$\vec{P}_0 - \vec{P}_{E_i}$	$t = \frac{\vec{N}_i * [\vec{P}_0 - \vec{P}_{E_i}]}{-\vec{N}_i * \vec{D}}$
links	(–1,0)	(x_{min},y)	$(x_0-x_{min},\ y_0-y)$	$\frac{-(x_0 - x_{min})}{x_1 - x_0}$
rechts	(1,0)	(x_{max},y)	$(x_0-x_{max},\ y_0-y)$	$\frac{x_0 - x_{max}}{-(x_1 - x_0)}$
unten	(0,–1)	(x,y_{min})	$(x_0-x,\ y_0-y_{min})$	$\frac{-(y_0 - y_{min})}{y_1 - y_0}$
oben	(0,1)	(x,y_{max})	$(x_0-x,\ y_0-y_{max})$	$\frac{y_0 - y_{max}}{-(y_1 - y_0)}$

Durch die Einschränkungen, die sich aus der Wahl eines senkrechten, rechteckigen Sichtfensters ergeben, erhält man eine bessere Effizienz im Vergleich zum Cyrus–Beck–Algorithmus.

2 Polygon–Clipping

2.1 Der Sutherland–Hodgman Algorithmus

Dieser Algorithmus kann verwendet werden, um beliebige, auch konkave, Polygone gegen beliebige konvexe Sichtfenster zu clippen. Er arbeitet wie folgt:

- Clippe alle Kanten des zu clippenden Polygons gegen die erste Kante des Sichtfensters.
- Dann clippe das Ergebnis gegen die nächste Kante des Sichtfensters.
- Wiederhole dies bis zur letzten Kante des Sichtfensters. Das Resultat ist das geclippte Polygon.

Dabei liegt das zu clippende Polygon als Liste von Ecken vor. Wenn Ecke s als »bereits in einer Iteration bearbeitet« betrachtet wird, so treten beim Clippen der darauf folgenden Polygonkante s nach p gegen eine Sichtfensterkante vier zu unterscheidende Fälle auf:

1. Liegen sowohl s als auch p innerhalb der Sichtfensterkante, so wird p der Liste der Ecken des Ausgabepolygons hinzugefügt.
2. Liegt s innerhalb und p außerhalb der Sichtfensterkante, so wird der Schnittpunkt der Kante s nach p mit der Sichtfensterkante als Ecke des Ausgabepolygons hinzugefügt.
3. Liegen beide außerhalb, so wird dem Ausgabepolygon keine Ecke hinzugefügt.
4. Liegt s außerhalb und p innerhalb, so wird zuerst der Schnittpunkt der Kante s nach p mit der Sichtfensterkante hinzugefügt und anschließend auch p zur Eckenliste des Ausgabepolygons hinzugefügt.

2.2 Der Liang–Barsky Algorithmus

Auch bei diesem Algorithmus gibt es eine Zonen–Aufteilung ähnlich der beim Cohen–Sutherland–Algorithmus. Jedoch wird hier nicht nach drei Klassen von Zonen (unten/oben, links/rechts, diagonal) unterschieden, sondern nur nach diagonal / nicht diagonal.
Der Algorithmus kann beliebige Polygone, auch konkave, gegen rechteckige, aufrechte Sicht–fenster clippen. Dazu geht er wie folgt vor:

- Clippe die zu bearbeitende Polygonkante gegen das Sichtfenster.
- Füge eventuelle Schnittpunkte mit dem Sichtfenster der Eckenliste des Ausgabepolygons hinzu. Beachte dabei die Reihenfolge vom Ausgangspunkt der Linie zum Endpunkt (Man sieht, parametrisiert zu arbeiten ist hier klar von Vorteil).
- Ist der Endpunkt der bearbeiteten Kante in einer Eckzone, so füge die zu dieser Eckzone des Sichtfensters gehörende Ecke zu der Eckenliste des Ausgabepolygons hinzu.
- Bearbeite die nächste Kante.
- Entferne in einem Nachbearbeitungsschritt eventuell vorhandene überschüssige Ecken und Kanten.

Es ist offensichtlich, daß für den ersten Schritt ein parameterisiert arbeitender Linien–Clip–ping Algorithmus die Probleme bei vertikalen Kanten (unendliche Steigung) löst.

2.3 Der Weiler Algorithmus

Der Weiler Algorithmus bestimmt für eine (theoretisch) beliebige Anzahl von Eingabepo– lygonen eine Datenstruktur, die aus folgenden Elementen besteht:

- Einem Satz von Polygonen, also einer Anzahl von Eckenlisten.
- Zu jedem dieser Polygone gibt es Informationen, zu welchem/n der Eingabepolygone dieses Ausgabepolygon gehört hat.
- Und zuletzt eine Baumstruktur, die wiedergibt, welche Polygone mit welchen koexistieren, und welche von anderen eingeschlossen werden.

Dazu sind mehrere komplexe Bearbeitungsschritte erforderlich.
Zuerst werden die Schnittpunkte der Kanten aller Eingabepolygone mit Kanten anderer Ein– gabepolygone den Eckenlisten beider betroffener Polygone hinzugefügt. Sodann werden alle Kanten der Eingabepolygone verdoppelt. Die Kanten tragen fortan Markierungen, wobei bei der eben erwähnten Verdopplung eine Kante eine »innere« Markierung erhält, die das jewei– lige Eingabepolygon eindeutig identifiziert. Die zweite Kante erhält bei der Verdopplung eine »äußere« Markierung. Aus diesen Informationen, Kanten und Markierungen, werden nun so– genannte »Konturen« generiert. Dabei handelt es sich um eine Menge disjunkter Polygone, welche die bei der Kantenverdopplung verteilten Markierungen in einer Art und Weise geerbt haben, daß nun jedes Konturpolygon die Markierungen aller Eingabepolygone trägt, in denen es enthalten war. Aus den Konturpolygonen läßt sich nun eine Baumstruktur bilden, welche die Information, welches Eingabepolygon neben welchem anderen koexistiert und welche Polygone in einem anderen enthalten sind, leicht verarbeitbar zugänglich macht.
Aus der mit dem Weiler–Algorithmus erhaltenen Datenstruktur lassen sich besonders leicht boolsche Verknüpfungen zwischen Polygonen bestimmen.

3 Literatur

Alle hier besprochenen Algorithmen findet man, zusammen mit weiterführenden Themen zur Grafikprogrammierung, in folgendem Werk:

- Foley, J.D.; van Dam, A.; Feiner, St.K.; Hughes, J.F.: Computer graphics – principles and practice. 2. Aufl. Reading, Masss.: Addison–Weseley 1990

Zum Verständnis der mathematischen Grundlagen:

- Meyberg, K.; Vachenauer, P.: Höhere Mathematik 1, Differential– und Integralrechnung, Vektor– und Matrizenrechnung. 4. Aufl. Berlin: Springer 1997