Proseminar Computergraphik: Rastergrafik-Algorithmen

Autor: Christian Vonsien

Technische Universität Dresden
Fakultät Informatik
Studiengang Informatik (Diplom)

0. Inhalt

0 Inhalt 1

1 Einleitung 2

1.1 Was ist eine Rastergrafik?

2 Rasterung von Linien und Kurven 3

2.1 Rasterung von Linien

2.2 Rasterung von Kreisen

2.3 Antialiasing

2.3.1 Methode von Gupta und Sproull

2.3.2 Methode von Wu

3 Füllen beschränkter Flächen 10

3.1 Scangeraden-Methode

3.2 Saatfüllen

4 Clipping von Linien und Polygonen 13

4.1 Clipping von Linien an rechteckigen Clip-Bereichen

4.2 Clipping von Polygonen an rechteckigen Clip-Bereichen

5 Quellen 16

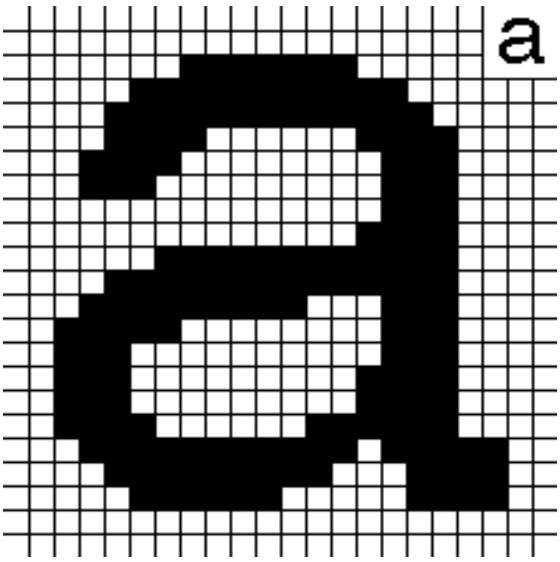
1 Einleitung

Die folgende Ausarbeitung behandelt Rastergrafik-Algorithmen, insbesondere die Rasterung von Linien und Kurven, sowie das Füllen und Clippen von Polygonen. Dazu wird zuerst die Rastergrafik an sich definiert und dann auf die einzelnen Algorithmen eingegangen.

1.1 Was ist eine Rastergrafik?

Eine Rastergrafik wird definiert als eine Matrix aus gleichgroßen, gleichmäßig angeordneten Pixeln. Die Eigenschaften einer Rastergrafik werden zum einen durch die Zahl der Pixel, zum anderen durch Höhe und Breite angegeben. Dadurch kann auch die Größe eines einzelnen Pixels bestimmt werden.
Ein Pixel besitzt immer einen Farbwert und meistens einen Alphawert, der die Intensität der Farbe bestimmt. Die Anzahl der Farbwerte wird durch die Anzahl der Bitebenen und der Anzahl der Kanäle pro Pixel bestimmt. Normalerweise hat ein Pixel 8 Bit Farbtiefe. Heutzutage hat zum Beispiel True Color 3 Kanäle bei RGB-Farbgebung, das heißt, es kann bis zu 24 Bit, bzw. etwa 16,78 Millionen Farben darstellen. Meist wird True Color auch mit 32 Bit angegeben, dabei sind die letzten 8 Bit aber für die Alphawerte zuständig.

Beispiel einer Rastergrafik:



2 Rasterung von Linien und Kurven

Um ein Bild auf den Bildschirm zu bekommen muss dieses zuerst in eine Rastergrafik umgewandelt werden, da der Bildschirm nur einzelnen Pixeln Farb- und Alphawerte zuweisen kann. Dabei tritt das Problem auf, dass Grafiken normalerweise keine festen Pixel oder Farbwerte haben, bzw. keine festen x- und y-Werte an denen sich die Farbwerte der Grafik befinden. Dazu wurden verschiedene Algorithmen entwickelt, um zum einen die äußeren Linien einer Grafik zu rastern, das heißt in eine Rastergrafik umzuwandeln, und zum anderen die Bereiche zwischen den gerasterten Linien zu füllen. Daher wird zuerst auf die Rasterung von Linien und Kurven eingegangen und dann auf das Füllen dieser gerasterten Linien.

2.1 Rasterung von Linien

Eine naive Herangehensweise an die Rasterung von Linien ist die y-Werte zu jedem x-Wert der Geradengleichung der Linie zu berechnen und dann am gerundeten Wert den Pixel einzufärben. Dies hat natürlich große Nachteile für eine Berechnung am Computer, da Multiplikations- und Fließkommaoperationen ausgeführt werden müssen. Diese sind sehr langsam an einem Computer gegenüber zum Beispiel der Addition.
Daher ersetzen wir die Multiplikation auch durch die Addition, da die y-Werte der Bildschirmkoordinaten sowieso nur ganzzahlig sein können. Wir berechnen die Steigung also nur einmal und addieren sie in jedem Schritt auf den vorherigen Wert auf. Das Runden bleibt leider weiter bestehen.
Um auch das Runden zu umgehen, wird eine Kontrollvariable eingeführt. Im weiteren wird diese als d_i bezeichnet.
Als Ansatz für den Algorithmus wird zuerst einmal festgelegt, dass nur Steigungen zwischen 0 und 1 betrachtet werden. Alle anderen Steigungen können dann durch Symmetrie herausgefunden werden. Dazu am Ende der Herleitung noch eine Veranschaulichung.
Diese Vereinfachung ermöglicht es, dass beim Zeichnen der Linie immer nur 2 weitere Pixel als nächstes einzufärbendes Pixel in Frage kommen. Das Pixel rechts neben und das Pixel rechts oberhalb des zuletzt eingefärbten Pixels. Um nun eine Aussage darüber zu treffen, wie weit die gerasterte Linie von der eigentlichen abweicht, wird der Punkt genau zwischen den beiden nächstmöglichen Pixeln betrachtet (weiterhin als Mittelpunkt bezeichnet). Befindet sich dieser Punkt unterhalb der Originallinie wird der Punkt rechts oberhalb genommen und umgekehrt.
Die ganze Überlegung muss natürlich in eine Formel übertragen werden, damit es auch berechnet werden kann.
Zur Herleitung der Formel, sowie der Kontrollvariable, die kontrolliert welcher Punkt als nächstes eingefärbt werden muss in Abhängigkeit der Abweichung von der Originallinie, wird die Geradengleichung in folgender Form betrachtet:

$$F(x,y) = ax+by+c = \Delta y\, x - \Delta x\, y + c$$

Für Punkte, die auf der Originallinie liegen, gilt: $F(x,y)=0$

Der Mittelpunkt hat die Koordinaten $x+1,\ y+1/2$. Dies wird jetzt in $F(x,y)$ eingesetzt.

$$F(x+1,y+1/2)=\Delta y(x+1)-\Delta x(y+1/2)+c=d_i$$

Dies liefert den Fehler, der mit jedem Schritt gemacht wird. Das heißt die Kontrollvariable ist damit gefunden und durch eine Formel definiert. Die Entscheidung des nächsten Schrittes fußt auf das Vorzeichen dieser Variable. Ist $d_i<0$ wird der Punkt rechts neben dem letzten genommen und umgekehrt.
Nun ist diese Formel immer noch zu kompliziert und zeit fressend um in jedem Schritt mit berechnet werden zu können. Daher wird d_i inkrementell aus dem nächsten und übernächsten Schritt berechnet. Die Herleitung der zugehörigen Formeln ist wie folgt.
Geht der nächste Schritt nach rechts (als ΔO bezeichnet), das heißt wird als nächstes das rechte Pixel eingefärbt, so berechnet sich der Fehler nach:

$$\Delta O=d_{i+1}-d_i=F(x+2,y+1/2)-d_i=\Delta y$$

Geht der nächste Schritt nach rechts oben (als ΔNO bezeichnet), das heißt wird als nächstes das rechte, obere Pixel eingefärbt, so berechnet sich der Fehler nach:

$$\Delta NO=d_{i+1}-d_i=F(x+2,y+3/2)-d_i=\Delta y-\Delta x$$

Jetzt kann der Fehler in jedem Schritt je nach Wahl des nächsten Pixel berechnet werden ohne Multiplikation oder Fließkommaoperationen zu verwenden. Es fehlt nur noch die Initialisierung des Fehlers. Dazu wird der Fehler am Anfangspunkt der Rasterung betrachtet:

$$d_0=F(x_0+1,y_0+1/2)=\Delta y\,x_0-\Delta x\,y_0+c+\Delta y-\Delta x/2$$

Dabei ist $\Delta y\,x_0-\Delta x\,y_0+c=F(x_0,y_0)=0$. Das heißt der Fehler und d_i werden wie folgt initialisiert:

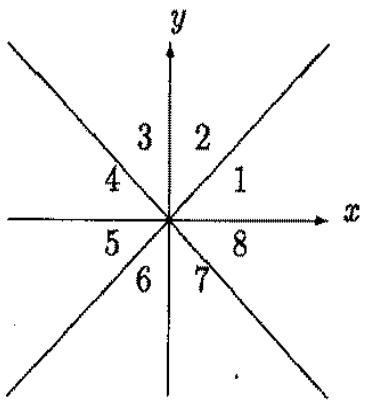
$$\begin{aligned}
&d_i=2\Delta y-\Delta x\\
&\Delta O=2\Delta y\\
&\Delta NO=2(\Delta y-\Delta x)
\end{aligned}$$

Dies ist eine Herleitung des sogenannten Bresenham-Algorithmus.
Daraus ergibt sich dann folgender Ablauf im Algorithmus:

$$\begin{aligned}
&d_i=2\Delta y-\Delta x\\
&\Delta O=2\Delta y\\
&\Delta NO=2(\Delta y-\Delta x)\\
&y=y_a\\
&Pixel(x_a,y_a)\,einfärben
\end{aligned}$$

$$
\begin{aligned}
&FOR(x = x+1, x \leq x_e)\\
&\quad IF\ (d_i \leq 0)\\
&\qquad d_i = d_i + \Delta O\\
&\quad ELSE\\
&\qquad d_i = d_i + \Delta NO\\
&\qquad y = y+1\\
&\quad Pixel\ (x, y)\ einfärben
\end{aligned}
$$

Die Initialisierung findet wie oben hergeleitet statt. Das erste Pixel wird eingefärbt und ab da wird die Kontrollvariable in jedem Schritt um die zugehörigen Fehlerwerte verändert. Je nachdem welcher Schritt dann ausgeführt wird, wird y um eins erhöht oder nicht. Das Pixel wird am Ende der Berechnung von x und y eingefärbt. Die Schleife läuft durch, bis das letzte Pixel erreicht ist.
Der Algorithmus gilt bisher nur für Steigungen zwischen 0 und 1. Um auch alle anderen Steigungen zeichnen zu können, nutzt man die Symmetrie aus.



Oktand	treibende Achse	andere Achse
1	x	inkrementiert
2	y	inkrementiert
3	y	dekrementiert
4	x	dekrementiert
5	x	inkrementiert
6	y	inkrementiert
7	y	dekrementiert
8	x	dekrementiert

Das heißt, man ändert im Algorithmus die treibende Achse. Diese wird im vorher beschriebenen Schleifenkopf mit jedem Schritt inkrementiert. Wenn man aber zum Beispiel eine Linie im dritten Oktanden berechnen möchte, so muss statt x, die y-Achse inkrementiert werden. In diesem Fall würde x dann auch dekrementiert werden, wenn die Kontrollvariable das so vorschreibt.
Der Bresenham-Algorithmus ist heutzutage nicht mehr softwareseitig implementiert, sondern wird direkt in die Hardware geschrieben.

2.2 Rasterung von Kreisen

Die Rasterung von Kurven wird hier durch die Rasterung von Kreisen repräsentiert. Das heißt, es wird ein Algorithmus zur Rasterung von Kreisen entwickelt, welcher eine Vereinfachung zur Zeichnung von Kurven ist.
Der folgende Algorithmus wurde wieder von Bresenham entwickelt, jedoch geht der Ansatz nicht auf ihn selbst zurück, sondern auf „Die Methode von Horn“.
Auch hier betrachtet man zuerst den ersten Oktanden und nutzt dann die Symmetrie aus. Man fängt beim Punkt $(r,0)$ an und geht dann entlang der Kreislinie bis hoch zu einem Winkel von 45°. Hierbei ist die treibende Achse die y-Achse, während x ab und zu dekrementiert wird. Die durch den Algorithmus berechneten x- und y-Werte werden auf den Kreis Mittelpunkt aufaddiert und dann gezeichnet.
Um einen geeigneten Algorithmus zu finden, muss man die umständlichen Berechnungen der Kreisgleichung umgehen. Dazu wird die Kreisgleichung betrachtet:

$$0 = x^2 + y^2 - r^2$$

Daraus kann man die Änderung in x- und y-Richtung in jedem Schritt berechnen:

$$x^2 = (x_{vorher} - 1)^2 = x_{vorher}^2 - 2\,x_{vorher} + 1$$
$$y^2 = (y_{vorher} + 1)^2 = y_{vorher}^2 + 2\,y_{vorher} + 1$$

Daraus ergibt sich für die Änderung in jede Richtung:

$$\Delta y = 2\mathrm{y} + 1$$
$$\Delta x = 2\mathrm{x} - 1$$

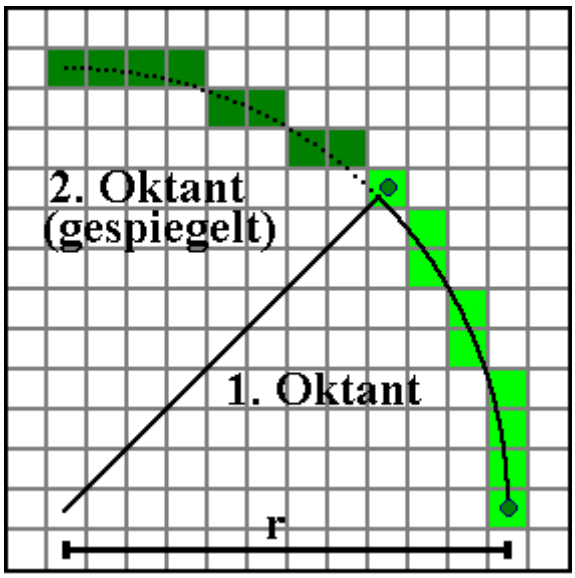
Die Änderung in x-Richtung wird nicht weiter berechnet werden, sondern fließt in das Fehlerglied ein. Auch r^2 muss nicht berechnet werden, wie man am Ende der Herleitung sehen wird.
Die Initialisierung der Kontrollvariable d_i muss nun noch hergeleitet werden. Die Frage dazu ist, ab welcher Abweichung von der Originalkurve soll x dekrementiert werden? Zum ersten Mal soll dies geschehen, wenn $x = r - 1/2$ gilt. Dann ist die Originalkurve um einen halben Pixel von der bisher gezeichneten entfernt. Aus dieser Überlegung heraus kann man folgendes herleiten:

$$y = \sqrt{r^2 - x^2} \quad \text{und} \quad x = r - 1/2 \;\Rightarrow\; y = \sqrt{r - 1/4}$$

Bis zu diesem Punkt hat noch kein x-Schritt stattgefunden, das heißt, dass sich die Kontrollvariable nur aus allen bisher berechneten y-Werten zusammensetzt, was laut Herleitung gleichzusetzen ist mit y^2. Da 1/4 vernachlässigbar ist, da nur im Bereich der ganzen Zahlen gerechnet wird, kann d_i mit $d_i = r$ initialisiert werden.
Aus diesen Überlegungen lässt sich der Bresenham-Algorithmus zur Rasterung von Kreisen aufstellen:

$$
\begin{aligned}
&x = r\\
&y = 0\\
&d_i = r\\
&Pixel(x_{mittel} + x, y_{mittel} + y)\ einfärben\\
&WHILE(y < x)\\
&\qquad \Delta y = 2y + 1\\
&\qquad y = y + 1\\
&\qquad d_i = d_i - - \Delta y\\
&\qquad IF\ d_i < 0\\
&\qquad\quad \Delta x = 1 - 2x\\
&\qquad x = x - 1\\
&\qquad d_i = d_i - \Delta x\\
&\quad Pixel(x_{mittel} + x, y_{mittel} + y)\ einfärben\\
&\quad Pixel(x_{mittel} + y, y_{mittel} + x)\ einfärben
\end{aligned}
$$

Die Symmetrie geht dann in die Zeilen ein, die die Pixel färben, je nachdem in welchem Oktanden man sich befindet, werden die berechneten x- und y-Werte auf den Mittelpunkt aufaddiert oder abgezogen.



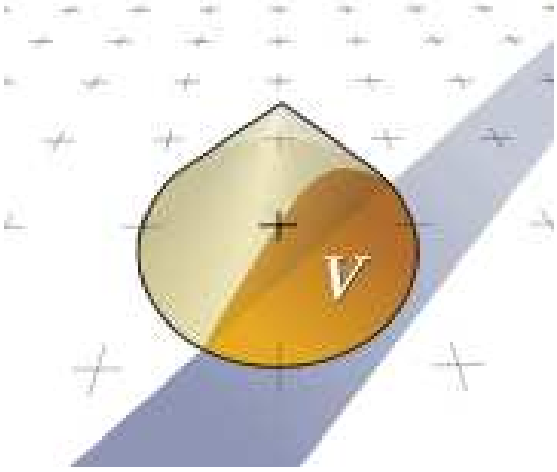
Im Algorithmus oben werden zum Beispiel die Pixel für den ersten und zweiten Oktanden gleichzeitig berechnet, indem sie über Kreuz auf den Mittelpunkt addiert werden. Wenn alle Oktanden in das Pixeleinfärben aufgenommen werden, so kann ein ganzer Kreis mit der Berechnung von nur einem Achtel-Kreis gezeichnet werden.

2.3 Antialiasing

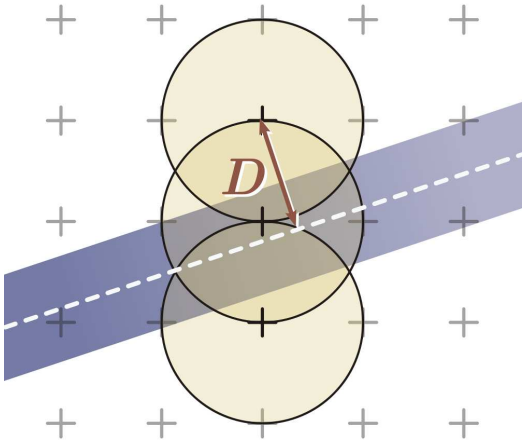
Durch den Bresenham-Algorithmus gezeichnete Linien haben immer nur ein Pixel Breite, das heißt, dass sie zum einen vom Weiten sehr dünn wirken und zum Anderen vom Nahen ein sehr treppenartiges Aussehen haben.
Um dieses Problem zu umgehen wurde das sogenannte Antialiasing entwickelt, welches um die Linien hellere Pixel legt, die die Linie zweidimensional machen und besser aussehen lassen.
Dazu werden 2 verschiedene Algorithmen betrachtet, die sich mit genau diesem Problem auseinander setzen.

2.3.1 Methode von Gupta und Sproull

Diese Methode basiert auf dem Bresenham-Algorithmus, das heißt die Rasterung der Linien wird nach dem Bresenham-Algorithmus berechnet. Dabei wird aber noch zusätzlich ein D für jedes angrenzende Pixel berechnet. Um die angrenzenden Pixel heraus zu finden, werden über jedes Pixel Kegel mit einem Radius von einem Pixel (Glättungskern) gelegt. Wenn der Kegel ein Pixel der gerasterten Linie schneidet, zählt das dazugehörige Pixel als angrenzend. Bei einer Linienstärke von einem Pixel (wie beim Bresenham-Algorithmus üblich) sind dies also höchstens drei Pixel innerhalb der Linie und sechs Pixel an den Enden.
Die Farbwerte der einzelnen angrenzende Pixel werden nun über D bestimmt. D ist der Abstand des Pixels von der zu rasternden Linie. Damit D nicht zu oft berechnet werden muss, werden nur 24 mögliche Distanzen aus einer Tabelle betrachtet. Die Intensität des Farbwertes ist proportional zu Distanz D bzw. proportional zum Volumen V des Kegels. Das heißt, je weiter das Pixel von der eigentlichen Linie entfernt ist, desto heller wird es.
Die Helligkeit der Pixel an den Linienenden hängt von der Steigung der Linie ab. Auch hier werden die Helligkeitswerte für einige Steigungen vorher berechnet und in einer Tabelle abgelegt.
Ein Problem, dass sich mit diesem Algorithmus ergibt, ist das seilartige Aussehen der Linien, da der Kegel als Glättungskern nicht geeignet ist.



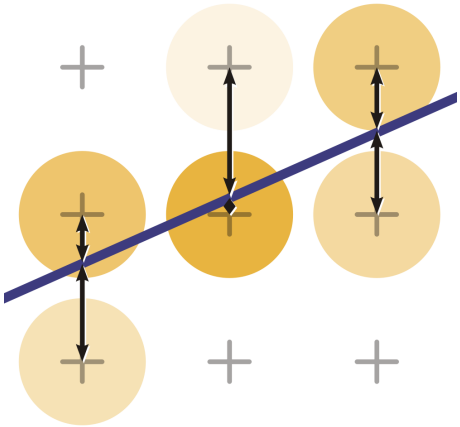
Hier sieht man einmal den Glättungskern bzw. Kegel auf einem Pixel mit dem Radius vom einem Pixel.



Hier sind die angrenzenden Pixel des Pixels der Linie zu sehen.

2.3.1 Methode von Wu

Diese Methode geht nicht von einem Glättungskern aus, sondern basiert auf einem Fehlermaß. In der Grundform ist dieser Algorithmus nur auf ideale, unendlich dünne Linien anwendbar.
Im Bresenham-Algorithmus wird versucht die Abweichung von der Originallinie durch die Kontrollvariable zu minimieren. Hier nimmt jedes Pixel ober- und unterhalb der Linie abhängig vom Abstand von der Originallinie eine Farbintensität an. Der Fehler lässt sich theoretisch ganz beseitigen, wenn jedes Pixel beliebig viele Intensitäten annehmen kann. Wu gab dazu einen besonders schnellen Algorithmus an, der nur eine Kontrollvariable benötigt und gleichzeitig die Position der Pixel ober- und unterhalb der Linie und deren Intensität bestimmt.



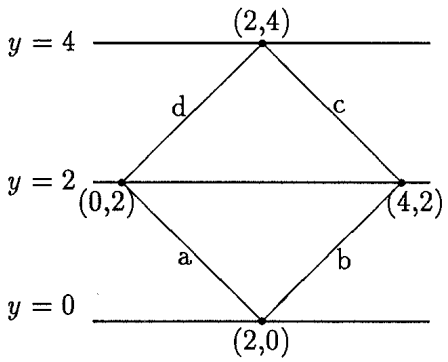
Hier ist ein Beispiel der Arbeitsweise des Wu-Algorithmus zu sehen. Je nach Abstand von der Originallinie haben die Pixel unterschiedliche Intensitäten.

3 Füllen beschränkter Flächen

Durch die oben genannten Algorithmen können Linien und Kurven erzeugt werden. Diese dienen aber oft nur zur Begrenzung von Flächen, die ausgefüllt werden müssen. Das heißt, es muss auch Algorithmen geben, die die Flächen zwischen diesen Linien und Kurven ausfüllen. Dazu werden zwei verschiedene Methoden vorgestellt.

3.1 Scangeraden-Methode

Die Voraussetzung für die Scangeraden-Methode ist die geometrische Beschreibung des zu füllenden Polygons. Ein Polygon ist eine Ebene oder ein Körper, der durch die Begrenzung durch Linien und Kurven entstanden ist.
Bei dieser Methode wird eine Scangerade von unten nach oben parallel zur x-Achse über das Polygon geschoben. Für jede Lage der Gerade werden die Schnittpunkte mit dem Polynom berechnet. Für konvexe Polygone gibt es so zwei oder gar keinen Schnittpunkt, für konkave eine gerade Anzahl an Schnittpunkten. Diese werden nach aufsteigendem x geordnet. Weiterhin wird jedem Schnittpunkt eine Parität zugeordnet. Diese beginnt bei Null und wird mit jedem Schnittpunkt um eins erhöht.
Nach dem Einlesen aller Schnittpunkte werden alle Pixel zwischen einem Paar von Schnittpunkten eingefärbt. Außerdem wird jedes Pixel gesetzt, falls dessen Parität ungerade ist.
Es gibt Sonderfälle, die bei der Scangeraden-Methode beachtet werden müssen. Wenn ein Schnittpunkt eine Fließkommazahl als x-Wert hat, so wird abgerundet, wenn man innerhalb des Polygons ist und aufgerundet, wenn man außerhalb ist.
Ist der x-Wert eine ganz Zahl, so wird der linke Endpunkt des zugehörigen Paares als innerhalb betrachtet, der rechte Endpunkt als außerhalb.
Ein Eckpunkt, mit einem y_{max} wird nicht zur Parität hinzu gezählt.
Horizontale Kanten werden auch nicht bei der Berechnung der Parität hinzu gezählt und fallen bei weiteren Berechnung raus.

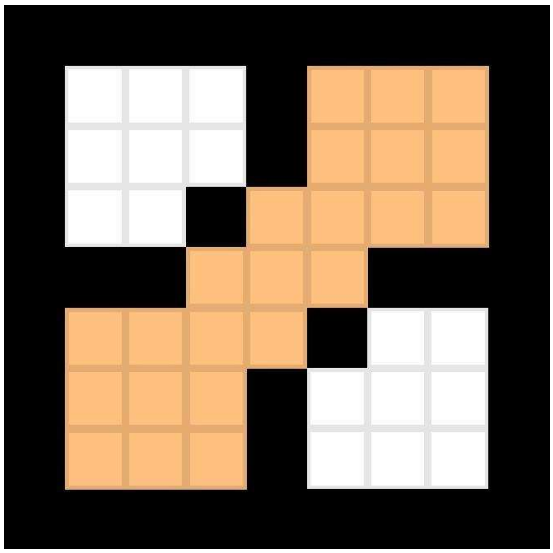


Ein Beispiel zur y_{max} Regel. Für $y = 0$ werden die Kanten a und b, sowie für $y = 2$ die Kanten c und d geschnitten, während für $y = 4$ keine Kante geschnitten wird.
Eine genauere Beschreibung des Einfärbe-Algorithmus wird jetzt noch einmal gegeben.

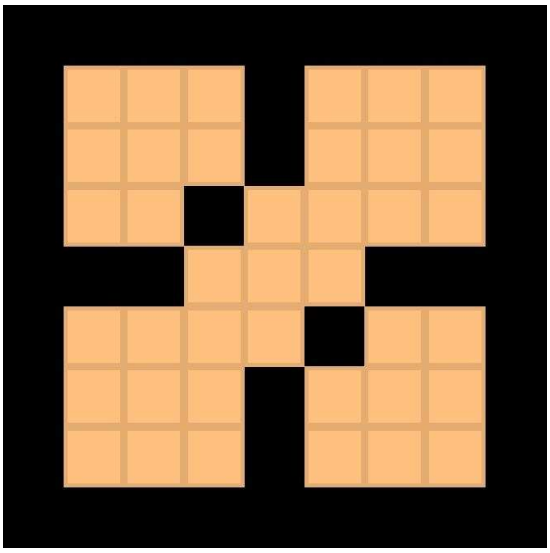
Eine äußere Schleife läuft über die Polygonkanten p. Eine innere Schleife berechnet alle Schnittpunkte mit den Scangeraden zwischen $y_{min}(p)$ und $y_{max}(p)$. Diese werden sortiert gespeichert. Nach dem Durchlauf werden die gespeicherten Schnittpunkte durchlaufen und die entsprechenden Pixel zwischen den Schnittpunkten eingefärbt.

3.2 Saatfüllen

Im Gegensatz zur Scangeraden-Methode wird hier keine geometrische Beschreibung benötigt. Diese Methode wird vorrangig benutzt um ein Polygon, dass von anderen Polygonen begrenzt wird, zu füllen. Das heißt es wird eher in einer interaktiven Grafik benutzt. Es wird aber eine Grenzfarbe benötigt. Diese Farbe gibt an, an welcher Stelle der Algorithmus stoppen soll.
Die Grundidee des Algorithmus ist zuerst ein Pixel innerhalb des zu füllenden Bereiches zu setzen und dann dessen Nachbarpixel auch einzufärben, bis alle Pixel eingefärbt sind. Nachbarpixel können zu einen nur vertikale und horizontale Pixel sein, oder auch zusätzlich diagonale.
Wird nur die Variante der horizontalen und vertikalen Pixel gewählt, so können damit auch Linien, die durch den Bresenham-Algorithmus entstanden sind gefüllt werden. Der Nachteil ist aber, dass bei „engen“ Polygonen nicht weiter gesprungen wird. Das heißt, dass die Fläche innerhalb des Polygons nicht komplett ausgefüllt wird. Abhilfe schafft dabei das Setzen einer zusätzlichen Saat, von welcher aus der Algorithmus fortgesetzt wird.



Hier ein Beispiel zu den Füllarten. In diesem Bild werden nur die vertikalen und horizontalen Nachbarpixel aufgefüllt.



Hier werden alle Nachbarpixel aufgefüllt. Dadurch werden auch die beiden Bereiche links oben und rechts unten ausgefüllt.

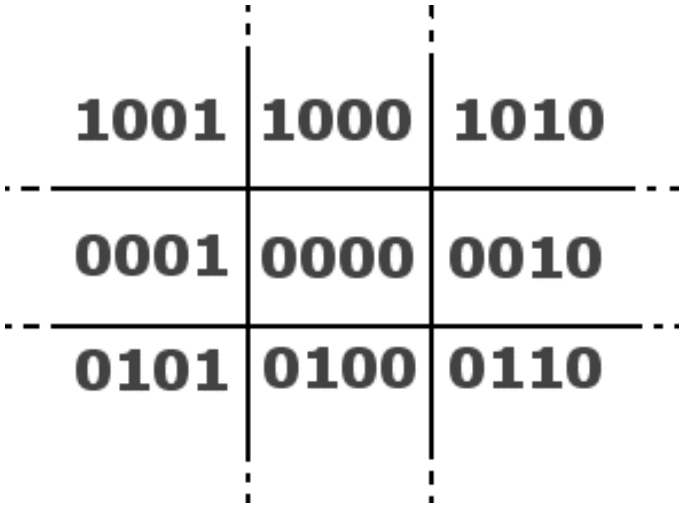
Man sieht außerdem gut an beiden Bildern, dass Linien, die durch den Bresenham-Algorithmus gezeichnet wurden, durchlässig für die zweite Art des Algorithmus sind.

4 Clipping von Linien und Polygonen

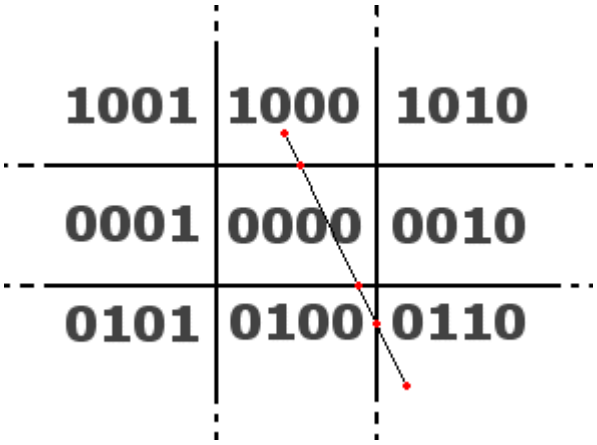
Auf dem Bildschirm sind oft mehrere Objekte dargestellt. Und oft sind diese auch übereinander bzw. hintereinander. Damit nun nicht alles mit einander vermischt wird, müssen die Objekte im Hintergrund „zurecht geschnitten“ werden, sodass nur das eigentlich Sichtbare wirklich sichtbar ist. Diese Technik wird Clipping genannt.
Sie wird aufgrund des hohen Bedarfs sehr oft angewandt, meist um Geraden an Polygonen zu schneiden. Es gibt 2 Varianten des Clippings. Zum einen das Clipping während der Darstellung oder das analytische Clipping.
Das Clipping während der Darstellung betrachtet jedes Pixel einzeln und überprüft für jedes einzelne Pixel, ob es noch angezeigt werden darf oder nicht. Das ist natürlich sehr aufwändig und deshalb kaum genutzt.
Das analytische Clipping berechnet zuerst alle Schnittpunkte und clippt dann. Dazu wird zuerst das Clipping von Linien an rechteckigen Clip-Bereichen betrachtet.

4.1 Clipping von Linien an rechteckigen Clip-Bereichen

Wie vorher schon beschrieben dürfen auf dem Bildschirm bestimmte Bereiche nicht angezeigt werden. Hier geht es jetzt speziell um Linien die innerhalb eines rechteckigen Bereiches liegen, wie zum Beispiel der Bildschirm oder auch nur ein Bildschirmbereich.
Eine naive Herangehensweise wäre folgende: Man berechnet von jeder Linie die Schnittpunkte mit dem Clip-Bereich und zeigt den Teil der Linie an, der innerhalb des Clip-Bereiches liegt.
Das Problem dabei ist, dass eine Linie auch komplett außerhalb oder komplett innerhalb des Clip-Bereiches liegt. Dann werden unnötige Schnittpunktberechnungen durchgeführt. Also wird ein Algorithmus entwickelt, der eine Aussage über die Lage der Linien zum Clip-Bereich machen kann.
Dazu werden 9 Bereiche in Bezug auf den Clip-Bereich angelegt. Jeweils vier Bereiche diagonal und vertikal bzw. horizontal zum Clip-Bereich. Diesen werden dann vierstellige Bitkodes zugeordnet. Die erste Stelle bedeutet links, die zweite rechts, die dritte unterhalb und die vierte oberhalb des Clip-Bereiches:



Die Bitkodes von benachbarten Bereichen unterscheiden sich also nur um eine Stelle.
Der Algorithmus ordnet nun jedem Endpunkt der zu clippenden Linien einen Bereich zu. Dazu werden die x- und y-Werte der Endpunkte betrachtet. Ist zum Beispiel der x-Wert des Endpunktes größer als der maximal x-Wert des Clip-Bereiches, wird für diesen Punkt das Flag für den rechten Bereich gesetzt. Ist dann noch der y-Wert kleiner als der minimale y-Wert des Clip-Bereiches, so wird auch noch das Flag für den unteren Bereich gesetzt. Dieser Punkt hätte also das Flag 0110.
Nachdem jeder Punkt ein Flag besitzt, können logische Operationen auf jeweils 2 zu einer Linie gehörigen Endpunkten ausgeführt werden:
Ist die bitweise Oder-Verknüpfung gleich 0000, so muss nicht geclippt werden, denn die Linie liegt komplett innerhalb des Fensters.
Ist die bitweise Und-Verknüpfung ungleich 0000, so geht die Linie am Fenster vorbei.
Tritt keiner dieser Fälle ein, so wird überprüft, welche der Flags des Endpunktes gesetzt sind. Aus diesen Flags werden dann die Schnittpunkte mit den Geraden des Clip-Bereiches berechnet.
Die daraus resultierenden Koordinaten sind die Koordinaten des Schnittpunktes der Linie mit dem Clip-Bereich und damit auch die neuen Endpunkte der geclippten Linie.
Dieser Algorithmus wurde von Dan Cohen und Ivan Sutherland entwickelt und trägt daher auch deren Namen: Algorithmus von Cohen-Sutherland.

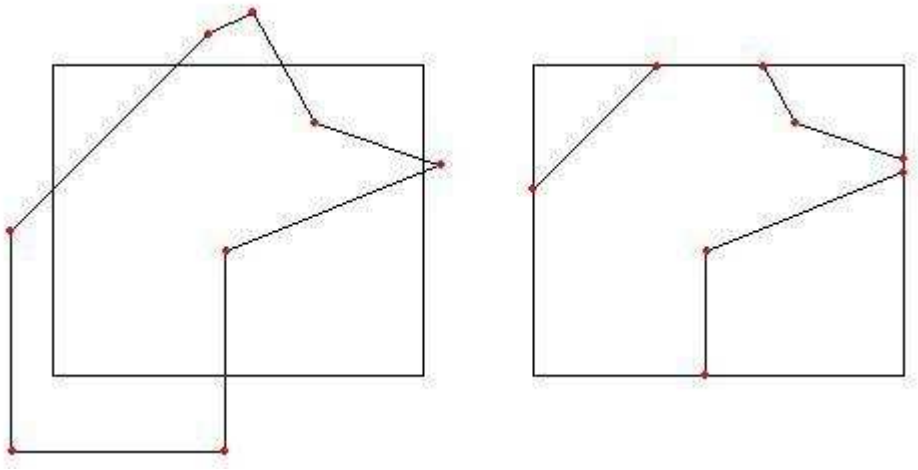


Hier ein Beispiel. Der Endpunkt rechts unten ist wie vorher schon beschrieben rechts unterhalb des Clip-Bereiches und die dazugehörige Linie geht durch den Clip-Bereich hindurch. In diesem Fall wird zuerst der Schnittpunkt mit der verlängerten rechten Außenkante des Clip-Bereichs und dann der Schnittpunkt mit der unteren Kante des Clip-Bereiches berechnet. Daraus entsteht dann die untere Begrenzung der Linie.

4.2 Clipping von Polygonen an rechteckigen Clip-Bereichen

Nun existieren nicht nur Linien auf dem Bildschirm. Meistens sind es doch Polygone, die angezeigt werden sollen, die jedoch auch für einen bestimmten Bereich zu groß sein können. Deshalb muss es auch eine Möglichkeit geben diese zu clippen und damit auf den Bereich anzupassen.
Dazu wurde ein Algorithmus von Ivan Cohen und G.W. Hodgman entwickelt. In der Grundversion wird das Problem wieder auf ein einfacheres Problem zurückgeführt: Linien an rechteckigen Clip-Bereichen schneiden. Dazu wird jede Kante des Clip-Bereiches zu einer Gerade erweitert. An dieser werden dann die Polygonkanten geschnitten. Die Menge der neuen Schnittpunkte beschreibt dann das neue Polygon.
Die verbesserte Version bezieht sich dann nur noch auf die Ecken des Polygons, kann aber nur Polygone an einem konvexen Clip-Bereich clippen. Da hier aber nur rechteckige Clip-Bereiche betrachtet werden, kann auch dieser Version benutzt werden.
Wie schon angedeutet benötigt die erweiterte Version alle Eckpunkte des zu clippenden Polygons. Man läuft nun alle Eckpunkte v_i einmal komplett ab, bis man wieder am Ausgangspunkt angelangt ist und behandelt diese nach folgenden Regeln:

- v_i und v_{i+1} liegen im Fenster $\Rightarrow$ v_{i+1} wird übernommen
- v_i innerhalb, v_{i+1} außerhalb $\Rightarrow$ Schnittpunkt mit der Fensterkante wird übernommen
- v_i außerhalb, v_{i+1} innerhalb $\Rightarrow$ Schnittpunkt mit der Fensterkante wird übernommen
- v_i außerhalb, v_{i+1} außerhalb $\Rightarrow$ entweder kein neuer Punkt wird übernommen oder beide Schnittpunkte mit den Clip-Bereichskanten werden übernommen, wenn die Gerade von v_i nach v_{i+1} durch den Clip-Bereich geht



Hier noch ein Beispiel zu diesem Algorithmus. Das linke Bild zeigt das Polygon mit seinen Eckpunkten vor dem Clippen. Das rechte Bild zeigt das geclippte Polygon mit seinen neuen Eckpunkten.

5 Quellen

[1] Piesch, Dieter: Skript Rastergrafik, Uni Regensburg, 2002
[2] Rauber. Thomas: „Algorithmen in der Computergraphik“, 1993
[3] Fellner, Dieter W.: „Computergrafik“, 1992
[4] Vandevenne, Lode: „Computer Graphics Tutorial“, 2004
[5] Wikipedia-User: Phrood, Peter Frankfurt und André Karwath