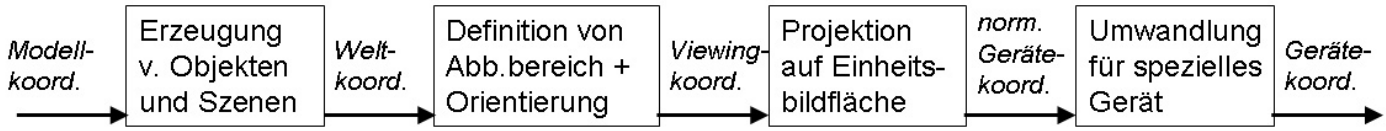
2D-Viewing

2D Viewing-Pipeline

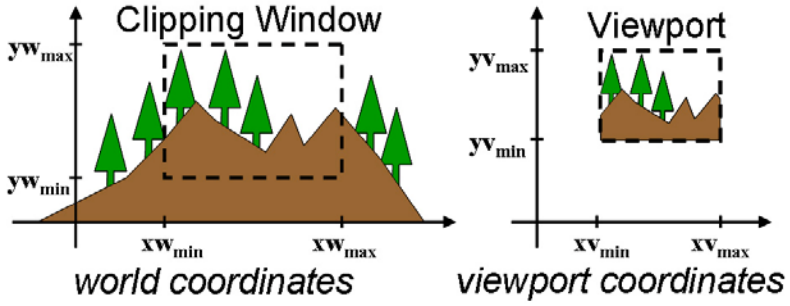
Unter Viewing-Pipeline versteht man die Folge von Umwandlungen, die geometrische Daten durchlaufen um schließlich als Bilddaten auf einem Gerät dargestellt zu werden. Die 2D-Viewing-Pipeline beschreibt diesen Vorgang für 2D-Daten:



Die Koordinaten, in denen einzelne Objekte konstruiert werden, nennt man ***Modellkoordinaten***,
aus diesen Objekten werden Szenen zusammengestellt, diese befinden sich in ***Weltkoordinaten***,
diese nennt man nach der Transformation in das Kamerakoordinatensystem die ***Viewingkoordinaten***,
die Abbildung eines Fensters der Szene erfolgt in geräteunabhängiger Weise in ***normalisierte Koordinaten***,
und schließlich werden diese normalisierten Werte in gerätespezifische ***Gerätekoordinaten*** abgebildet.

Window-Viewport Transformation

Eine Window-Viewport-Transformation beschreibt den Übergang eines (rechteckigen) Fensters in einem Koordinatensystem in ein anderes (rechteckiges) Fenster in einem anderen Koordinatensystem. Dabei wird definiert welcher Ausschnitt transformiert wird, wo das Ergebnisfenster liegt, sowie wie das Fenster verschoben, skaliert oder verdreht wird.



Die folgende Ableitung zeigt, wie einfach diese Transformation im Allgemeinen (d.h. ohne Rotation) ist:

Die Abbildung ist linear in x und y, und $(xw_{min}/yw_{min}) \rightarrow (xv_{min}/yv_{min})$, $(xw_{max}/yw_{max}) \rightarrow (xv_{max}/yv_{max})$.
Für einen beliebigen Punkt (xw/yw) der nach (xv,yv) transformiert wird, gilt:

$$xw = xw_{min} + \lambda(xw_{max}-xw_{min}) \quad \text{where } 0<\lambda<1 \quad \Rightarrow \quad xv = xv_{min} + \lambda(xv_{max}-xv_{min})$$

Wenn man λ aus der 1. Gleichung berechnet und in die 2. Gleichung einsetzt, erhält man:

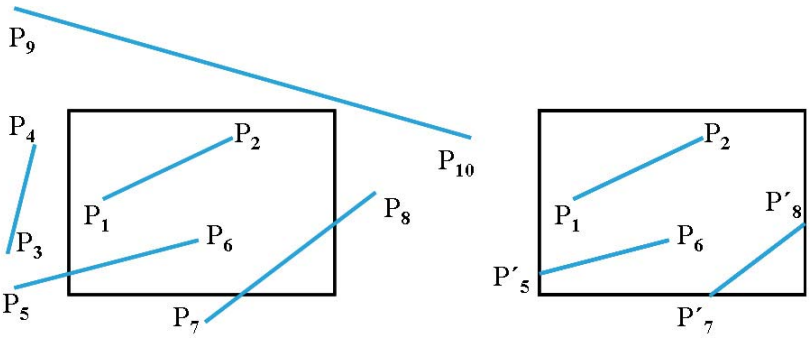
$$xv = xv_{min} + (xv_{max} - xv_{min})(xw - xw_{min})/(xw_{max} - xw_{min}) = xw(xv_{max} - xv_{min})/(xw_{max} - xw_{min}) + t_x$$

wobei t_x eine für alle Punkte gleiche Konstante ist, ebenso der Faktor $s_x = (xv_{max} - xv_{min})/(xw_{max} - xw_{min})$.
Verfährt man mit y analog, erhält man die einfache lineare Transformation: $\mathbf{xv = s_x xw + t_x}$, $\mathbf{yv = s_y yw + t_y}$,

Im Kapitel Transformationen wird beschrieben, wie man solche Transformationen noch einfacher anschreiben kann.

Linien-Clipping

Clipping heißt das Abschneiden von Bildteilen, die außerhalb des Darstellungs-fensters liegen (siehe auch Abb. oben). Je früher man die Clippingoperation in der Viewing-Pipeline durchführt, desto mehr unnötige nachfolgende Umformung von



ohnehin nicht sichtbaren Teilen erspart man sich:

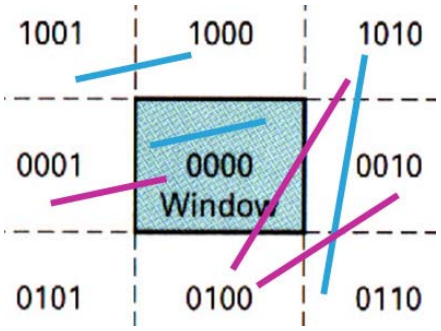
- **in Weltkoordinaten**, also analytische Berechnung zum frühest möglichen Zeitpunkt,
- **bei der Rasterkonversion**, also innerhalb d. Algorithmus, der ein Graphikprimitiv in Punkte umformt,
- **pixelweise**, also nach allen Berechnungen erst unmittelbar vor der Zeichnung.

Clipping ist eine sehr häufige Operation, daher muss sie einfach und schnell sein.

Clippen von Linien: Cohen-Sutherland-Verfahren

Algorithman zum Clippen von Linien nützen i.A. die Tatsache aus, dass jede Linie in einem rechteckigen Fenster höchstens einen sichtbaren Teil besitzt. Weiters gilt es Grundprinzipien der Effizienz auszunutzen, etwa häufige einfache Fälle früh zu eliminieren und unnötige teure Operationen (Schnittpunkt-Berechnungen) zu vermeiden. Einfachstes Linienclipping könnte etwa so aussehen:

```
for endpoints (x0,y0), (xend,yend)
intersect parametric representation
     x = x0 + u*(xend - x0)
     y = y0 + u*(yend - y0)
with window borders:
     intersection ⇔ 0 < u < 1
```



Der Cohen-Sutherland-Algorithmus klassifiziert zuerst die Endpunkte einer Linie hinsichtlich ihrer Lage zum Clippingfenster: oben, unten, links, rechts, und codiert diese Information in 4 Bit. Nun kann man schnell überprüfen:

1. OR der beiden Codes = 0000 ⇒ Linie ganz sichtbar
2. AND der beiden Codes ≠ 0000 ⇒ Linie ganz unsichtbar
3. andernfalls mit einer relevanten Fensterkante schneiden, und den weggeschnittenen Punkt durch den Schnittpunkt ersetzen. GOTO 1.

Schnittpunktberechnungen:

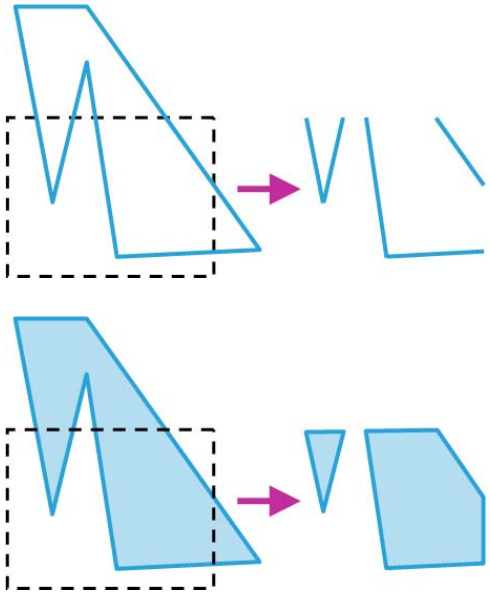
mit vertikalen Fensterkanten: $y = y_0 + m(xw_{min} - x_0)$, $y = y_0 + m(xw_{max} - x_0)$

mit horizontalen Fensterkanten: $x = x_0 + (yw_{min} - y_0)/m$, $x = x_0 + (yw_{max} - y_0)/m$

Punkte genau auf den Fensterkanten müssen natürlich als innerhalb gelten, dann kann es zu höchstens 4 Schleifendurchläufen kommen. Wie man auch sieht, werden nur dann Schnittpunktberechnungen durchgeführt, wenn es wirklich notwendig ist.

Für das Clippen von Kreisen gibt es ähnliche Verfahren. Man muss jedoch berücksichtigen, dass Kreise beim Clippen in mehrere Teile zerfallen können.

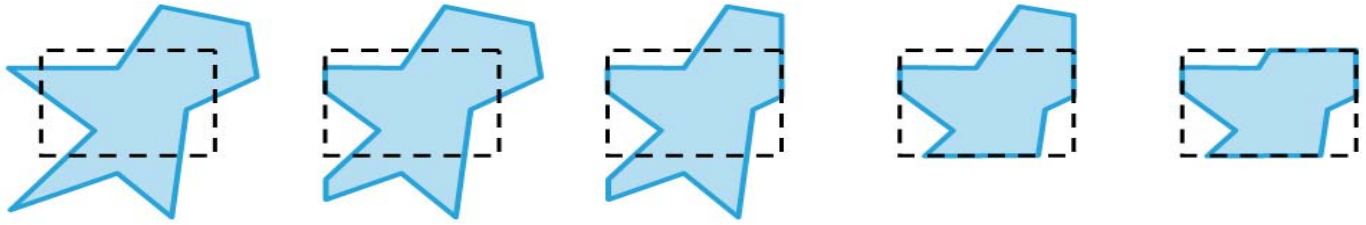
Polygon-Clipping



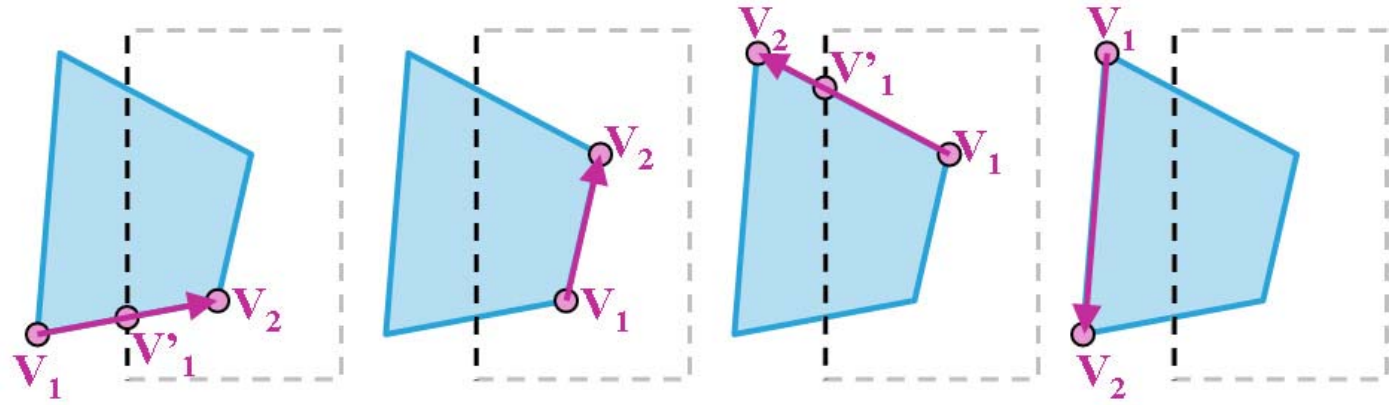
Das Clippen von Polygonen hat zu berücksichtigen, dass nach dem Clippen als Resultat wieder *ein* Polygon erzeugt wird, auch wenn durch den Clipping-Vorgang mehrere Teile entstehen. Die obere Abbildung zeigt ein Polygon, dass mit einem Linien-Clipping-Algorithmus geclippt wurde. Es ist nicht mehr erkennbar, was innen und was außen ist. Das untere Bild zeigt das Ergebnis eines korrekten Polygon-Clipping-Verfahrens. Das Polygon zerfällt in mehrere Teile, die alle korrekt gefüllt werden können.

Clippen von Polygonen: Sutherland-Hodgman-Verfahren

Die Grundidee kommt von der Erkenntnis, dass beim Clipen an nur einer Kante keine gröberen Komplikationen entstehen. Daher wird das Polygon nacheinander an allen 4 Fensterkanten geclippt, und das Ergebnis jeweils als Input für die nächste Kante verwendet:

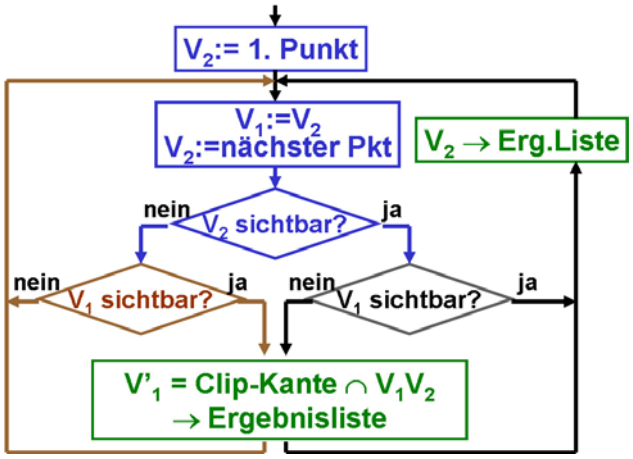


Es gibt 4 verschiedene Fälle, wie sich eine Kante (V_1, V_2) bezüglich einer Fensterkante verhalten kann. Bei der sequenziellen Abarbeitung werden dabei folgende Resultate erzeugt:



1. out→in (output V'_1, V_2) 2. in→in (output V_2) 3. in→out (output V'_1) 4. out→out (kein output)

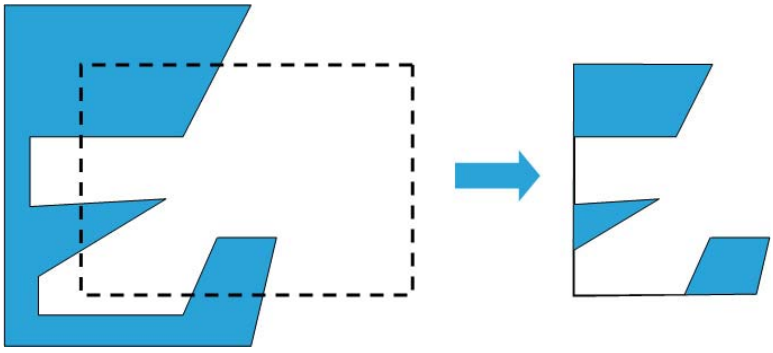
Der Algorithmus ***für eine Kante*** läuft dann so ab:



Die Eckpunkte des Polygons werden sequenziell bearbeitet. Für jede Polygonkante wird festgestellt, welchem der 4 Fälle sie zuzurechnen ist und der entsprechende Eintrag in die Ergebnisliste erzeugt. Die Ergebnisliste enthält nach dieser Bearbeitung die Eckpunkte des an dieser Kante geclippten Polygons, ist also wieder ein gültiges Polygon und dient als Input für die Clippingoperation ander nächsten Fensterkante.

Diese drei Zwischenresultate kann man vermeiden, indem man die Prozedur für die 4 Fenstergrenzen *rekursiv* aufruft, und so jeden Ergebnispunkt sofort wieder als nächsten Eingabepunkt der nächsten Clippingoperation verwertet. Alternativ kann natürlich auch eine Pipeline durch die 4 Operationen geschleust werden, sodass am Ende nur ein Polygon entsteht – das an dem Fenster korrekt geclippte Polygon.

Wenn ein Polygon beim Clippen in mehrere Teile zerfällt, dann erzeugt dieses Verfahren Verbindungskanten entlang des Clippingfensters. Eine nachträgliche Kontrolle und eventuelle Nachbearbeitung ist in solchen Fällen notwendig.



Text-Clipping

Das Clippen von Text erscheint auf den ersten Blick trivial, es muss allerdings eine kleine Feinheit beachtet werden. Je nach Erzeugungsweise der Buchstaben kann es passieren, dass nur Texte angezeigt werden, die komplett lesbar sind (wo also alle Buchstaben komplett im Fenster liegen), dass Text nur buchstabenweise geclippt wird (also alle Buchstaben ganz verschwinden, die nicht ganz im Fenster sind), oder dass Text korrekt abgeschnitten wird (also auch halbe Buchstaben erzeugt werden).