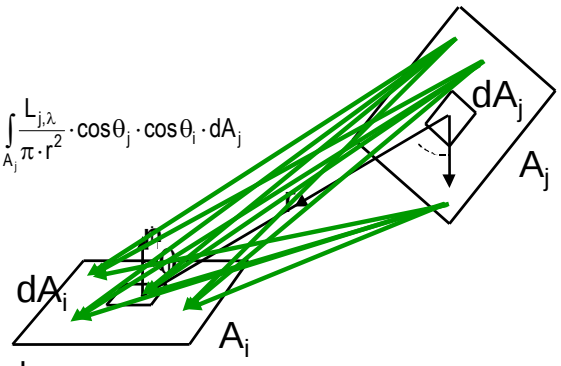
3D-Rendering

Privatdozent (PD) Dr.-Ing. habil. K.-H. Franke

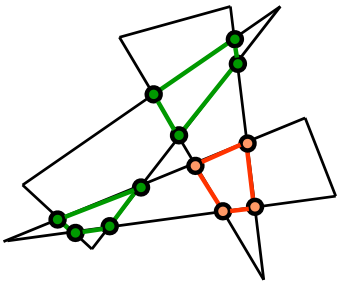
Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Prakt. Inf. und Medieninf.

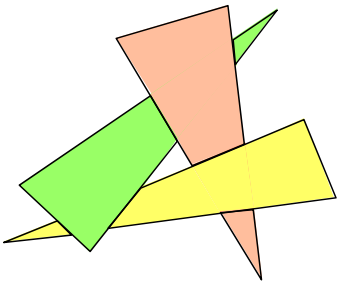
Fachgebiet Graphische Datenverarbeitung

$$L_{i,\lambda} = \beta_i(\lambda) \cdot \int_{A_j} \frac{L_{j,\lambda}}{\pi \cdot r^2} \cdot \cos\theta_j \cdot \cos\theta_i \cdot dA_j$$



ZBS, Werner-von-Siemens-Str. 10, Sekretariat, Monika Stübchen,
Tel.: (03677) 6897 680
oder Sekretariat **FG GDV, Frau Franziska Katzki, Zusebau**, Zi. 2039,
Tel.: (03677) 69 4119





Tel.: (03677) 6897-681
(0172) 36 27 701
Fax: (03677) 6897-682
email: karl-heinz.franke@tu-ilmenau.de
Internet: http://www.zbs-ilmenau.de
http://kb-bmts.rz.tu-ilmenau.de/Franke

Ilmenau, 11.12.2011

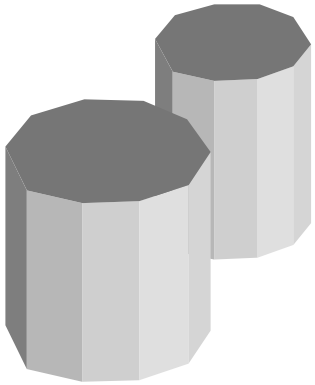
ISSN 1432-3346

3D-Rendering (Inhalt)

- 3D-Rendering (Übersicht)
- Lokale 3D-Schattierungsverfahren
  - Painters-Algorithmus
  - Z-Puffer-Verfahren
  - Transparenz
  - Kontinuierliche Schattierung von Polygonen (Gouraud-Shading)
  - Phong-Shading
  - 3D-Grafikhardware (Übersicht)
- Globale 3D-Schattierungsverfahren
  - Ray Tracing (Strahlenverfolgung)
    - Grundprinzip
    - Rekursives Ray Tracing, Monte Carlo Ray Tracing
    - Lichtschatten, Transparenz und Brechung
    - Implementierung des Ray-Tracing-Verfahrens
  - Radiosity
    - Grundprinzip des Radiosity-Verfahrens
    - Progressive Refinement, Adaptives Radiosity
    - Eigenschaften des Radiosity-Verfahrens, Erweiterungen

ISSN 1432-3346

Direkte 3D-Schattierungsverfahren (List-Priority Algorithms / Painters)



Flachschattierte Polygone, gezeichnet mit dem Painters-Algorithmus: Zuerst werden die entfernten Objekte geschrieben, dann die Nahen im Overwrite-Modus. Verdeckungsprobleme lösen sich dadurch quasi automatisch, → aber

- List-Priority Algorithmen sind hybride Algorithmen, die *Objektgenauigkeit (Tiefenstaffelung der Objekte) und Bildgenauigkeit (Zeichnen der Pixel)* kombinieren →
- *Aus dieser Kombination resultieren die Probleme beim Painters-Algorithmus:* Widerspruch zwischen *Dreidimensionalität der Polygone* einerseits und *dem Ordnungsprinzip für Objekte nach den Koordinaten einzelner Punkte* (Schwerpunkte) andererseits.

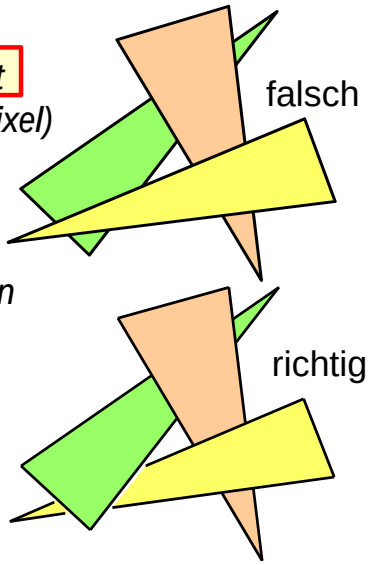


Abb. rechts: Beispiel für sich zyklisch überlappende Polygone, die bei Anwendung des Painters falsch dargestellt werden (es gibt nur eine Reihenfolge der Schwerpunkte).

Direkte Schattierungsverfahren (Depth-Sort Algorithm /Newell and Sancha/)

Lösung des Problems der sich überlappenden Polygone:

1. Sortiere alle Polygone, beginnend mit der kleinsten z-Koordinate (entferntestes Polygon)
2. Löse alle Mehrdeutigkeiten auf, die sich aus überlappenden Polygonen ergeben können (wie?)
3. Zeichne die aufgelösten Polygone in der Reihenfolge der z-Werte (vom entferntesten zum nächsten Polygon)

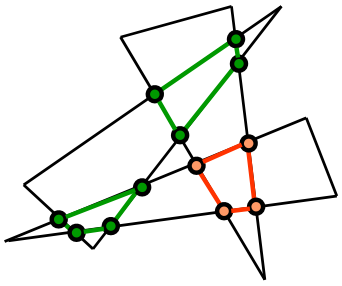
Der Schritt 2 beginnt mit einfachen Tests um zu entscheiden, ob *Schnitte zur Auflösung der Mehrdeutigkeiten* erfolgen müssen (P sei das entfernteste Polygon, Q das nächst nähere in der Reihenfolge der Liste):

2.1 gibt es x-Überlappung von P und Q *in der Projektionsebene* (nein → Testende / ja → weiter)
2.2 gibt es y-Überlappung von P und Q *in der Projektionsebene* (nein → Testende / ja → weiter)
2.3 liegt P vollständig hinter der (gesamten) *3D-Ebene von Q* (ja → Testende / nein → weiter)
2.4 liegt Q vollständig vor der *3D-Ebene von P* (ja → Testende / nein → weiter)
2.5 gibt es *Überlappungen* der Polygone P und Q *in der Projektionsebene* (nein → zeichne P, weiter mit den verbliebenen Polygonen der Liste, ja → Zerlegung der Polygone)

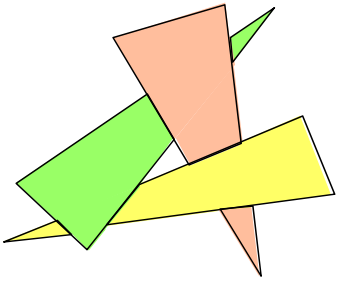
Direkte Schattierungsverfahren (Depth-Sort-Algorithm)

Zerlegung sich überlappender Polygone:

- Unterteilung in sich *nicht überlappende* und *vollständig überdeckende* Teilpolygone
- Ist in der Projektionsebene ohne Probleme möglich durch gegenseitigen Schnitt aller Polygone.



- Die sichtbaren Teilpolygone können nun ausgegeben werden:
  - Zeichnen der nicht überlappenden Teilpolygone
  - Von den sich vollständig überlappenden Teilpolygonen wird nur das vordere gezeichnet.

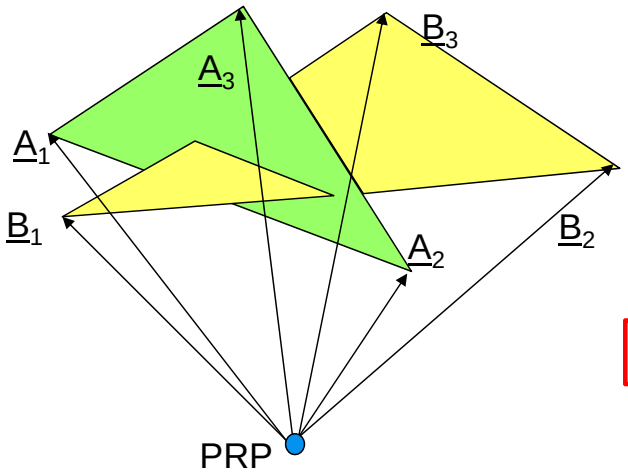


- Befinden sich nicht nur P und Q, sondern mehrere Polygone in der Szene, so müssen in der weiteren Abfolge des Depth-Sort-Algorithmus P und Q auch zu den anderen Polygonen auf Überlappungen geprüft werden, die im Widerspruch zur Tiefenstaffelung der Polygonschwerpunkte stehen.

Direkte Schattierungsverfahren (Depth-Sort Algorithm)

Achtung:

- Verfahrensweise schlägt fehl bei *sich durchdringenden Polygonen.*
- In diesem Fall müssen *neue Schnittkanten in 3D (nicht in der Projektionsebene)* berechnet und dann entsprechend aufgeteilt werden.



$$\frac{(\underline{A}_2-\underline{A}_1)\times(\underline{A}_3-\underline{A}_1)}{|(\underline{A}_2-\underline{A}_1)\times(\underline{A}_3-\underline{A}_1)|}=\underline{n}_{g(reen)}$$

$$\underline{n}_g^T\cdot\underline{P}-\underline{n}_g^T\cdot\underline{A}_1=\underline{n}_g^T\cdot\underline{P}-a_g=0$$

$$\frac{(\underline{B}_2-\underline{B}_1)\times(\underline{B}_3-\underline{B}_1)}{|(\underline{B}_2-\underline{B}_1)\times(\underline{B}_3-\underline{B}_1)|}=\underline{n}_{y(ellow)}$$

$$\underline{n}_y^T\cdot\underline{P}-\underline{n}_y^T\cdot\underline{B}_1=\underline{n}_y^T\cdot\underline{P}-a_y=0$$

$$\underline{r}=(\underline{n}_g\times\underline{n}_y),\qquad \underline{n}_g^T\cdot\underline{P}_0=a_g,\qquad \underline{n}_y^T\cdot\underline{P}_0=a_y$$

Zwei Gleichungen für die drei Koordinaten X_0, Y_0, Z_0 → eine frei wählbar (z.B. Z_0=0)

Schnittgerade → $\underline{P}=\underline{r}\cdot t+\underline{P}_0$

Schnitt mit Polygonkanten → Schnittkante

Direkte Schattierungsverfahren (Depth-Sort Algorithm)

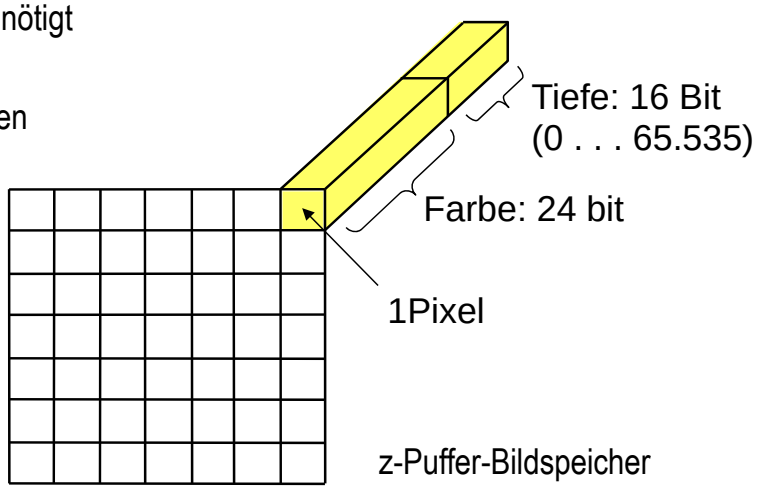
Wann wendet man den Painter´s Algorithmus / Depth-Sort Algorithmus an?

- Einfache Szenen, kleine Objekte, die sich in den z-Werten hinreichend unterscheiden.
- Dort, wo keine 3D-Unterstützung angeboten wird (begrenzter Speicher, keine Z-Buffer Unterstützung).
- *Für das Schattieren von semitransparenten Flächen !!*

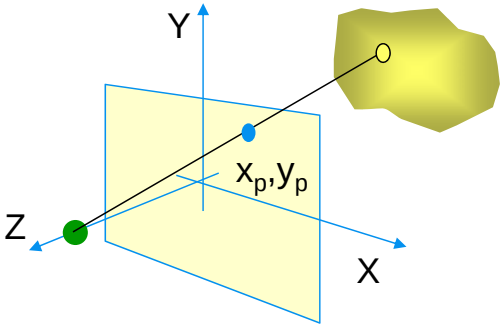
Direkte Schattierungsverfahren

z-Puffer-Verfahren (z-Buffer)

- Einer der einfachsten „visible surface“-Algorithmen (CUTMULL 1974)
- Probleme des Painters-Algorithmus werden überwunden durch *zusätzliche Berechnung des z-Wertes für jeden Punkt* jedes Polygons und Speicherung des zur Projektionsebene nächstliegenden Farb- und Z-Wertes.
- Dazu ist ein zusätzlicher z-Speicher benötigt (z-Buffer).
- Es sind weder Vorsortieren von Objekten noch Objektvergleiche erforderlich.



Direkte Schattierungsverfahren (Z-Puffer)



Initialisierung: Für alle Pixel

- Setze Farbe auf Hintergrundfarbe (z. B. Weiß)
- Setze alle Z-Werte auf -∞ (minimaler ganzzahliger Wert)
- Setze Z_{max} auf Wert der kameraseitigen Clipping-Plane

Für alle Polygone (nach der Ansichten-Transformation, d.h. im 3D-Kamerakoordinatensystem)

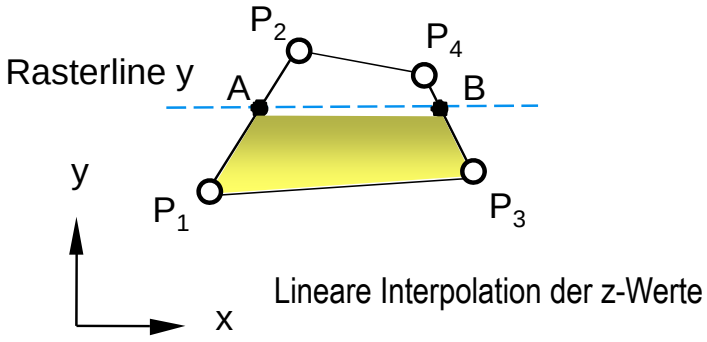
- Rasterumwandlung in der Projektionsebene *(x_p / y_p Koordinaten)* durch modifizierten 2D-Polygonfüllalgorithmus. Modifiziert heißt: *zusätzliche Berechnung des z-Wertes für jedes Pixel*
- Anwendung einer Write_Pixel_ZB-Prozedur. *Wenn der z-Wert* des aktuellen Pixels im abzuarbeitenden Polygon *größer* als der bereits abgespeicherte z-Wert an dieser Position ist, wird *mit (x_p, y_p → Farbe, z_p) überschrieben*.
- *Die näher an der Kamera liegen Pixel überschreiben also die ferneren*.

Direkte Schattierungsverfahren (Z-Puffer)

Interpolation der z-Werte:

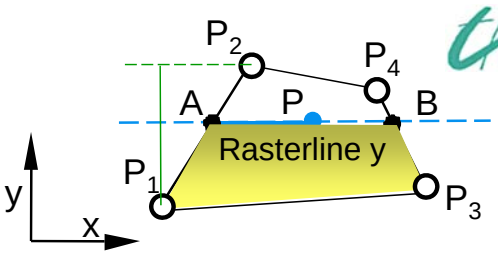
- Die Tiefenwerte sind auch nach der Ansichten-Transformation (View-Transformation) nur für die Eckpunkte (hier P_1 bis P_4) gegeben.
- *Zunächst* erfolgt die *lineare Interpolation der z-Werte entlang der Polynomkanten* P_i–P_j ($i \neq j$) für die y-Position der gerade aktuellen Scanline (z.B. Punkte A und B).
- Danach wird mit dem *Füllen der Bildzeile* (z.B. durch einen konventionellen Polyfill-Algorithmus) *die Interpolation der z-Werte entsprechend der x-Position in der Scanline* (Bildzeile) fortgesetzt (pixelgenaues Befüllen des z-Buffers).

Immer Ganzzahlarithmetik!
(Midpoint- oder Bresenham-Algorithmus,
der vom Zeichnen digitaler Geraden bekannt ist.)



Lineare Interpolation der z-Werte

Direkte Schattierungsverfahren (Z-Puffer)



Berechnung der z-Werte eines Pixels:

a)
$$z_A = z_1 + \frac{z_2 - z_1}{y_2 - y_1} \cdot (y_A - y_1)$$
$$z_B = z_3 + \frac{z_4 - z_3}{y_4 - y_3} \cdot (y_A - y_3)$$

b)
$$z_p = z_A + \frac{z_B - z_A}{x_B - x_A} \cdot (x_p - x_A)$$

- Eine Koordinate reicht zur Interpolation von Z_A und Z_B (Strahlensatz). Beim füllen in x-Richtung ist y sinnvoll (charakterisiert die Scanline).
- Der Pixel-z-Wert z_p wird völlig äquivalent ermittelt, allerdings die Interpolationskoordinate jetzt x (y = const für die Rasterlinie)
- Die Werte z_A, z_B, x_A, x_B, in z_p werden gleichzeitig mit den y_A-Werten (Schnitte) von einer Rasterlinie zur nächsten inkrementiert, wenn dies nach Brasanham erforderlich wird.

- Die Brüche bleiben in allen Ausdrücken rational !! (Bresenham)
- Die Ausdrücke für die z-Werte haben identische Form wie die der I-Werte beim Polygonfüllalgorithmus

Immer Ganzzahlarithmetik! (Midpoint- oder Bresenham-Algorithmus)

Man beachte: die Farbe des Pixels (aufwendige Schattierungsrechnung) muss nicht berechnet werden, *wenn klar ist, dass das Pixel verdeckt ist*. → Aus Effizienzgründen ist ein *grobes Vorsortieren* und Beginn mit den vorderen Polygonen Rechenzeit sparend !

Berechnung der Werte für den Z-Buffer (Bresenham)

- Z-Buffer i.a. für ganzzahlige z-Werte → Berechnung mit Rundung → Bresenham
- Ohne Einschränkung der Allgemeinheit darf $\Delta z/\Delta x < 1$ angenommen werden

$x := x_A, \quad y := y_A, \quad z := z_A$	$x := x_A, \quad y := y_A, \quad z := z_A$
$\Delta x = x_B - x_A, \quad \Delta z = z_B - z_A$	$\Delta x = x_B - x_A, \quad \Delta z = z_B - z_A$
Schreibe $z(x, y)$	$Schreibe \;\; z(x, y)$
$dz := \frac{\Delta z}{\Delta x}$ dz=2/9=0,22	$2 \cdot dz \cdot \Delta x := 2 \cdot \Delta z$ 2·dz·**Δx**=4
while $x < x_B$	$while \; x < x_B$
$x := x + 1$	$x := x + 1$
if $dz < 0.5$	$if \;\; 2 \cdot dz \cdot \Delta x < \Delta x$ 2·dz·**Δx**<9
then $dz := dz + \frac{\Delta z}{\Delta x}$	$then \; 2 \cdot dz \cdot \Delta x := 2 \cdot dz \cdot \Delta x + 2 \cdot \Delta z$
else $dz := dz - 1 + \frac{\Delta z}{\Delta x}$, $z := z + 1$	$else \; 2 \cdot dz \cdot \Delta x := 2 \cdot dz \cdot \Delta x - 2 \cdot \Delta x + 2 \cdot \Delta z \;\; , \;\; z := z + 1$
Schreibe $z(x, y)$	$Schreibe \; z(x, y)$
end	end

Berechnung der Werte für den Z-Buffer (Bresenham)

- Z-Buffer i.a. für ganzzahlige z-Werte → Berechnung mit Rundung → Bresenham
- Ohne Einschränkung der Allgemeinheit → $\Delta z/\Delta x < 1$ darf angenommen werden

$x := x_A, \quad y := y_A, \quad z := z_A$

$\Delta x = x_B - x_A, \quad \Delta z = z_B - z_A$

Schreibe $z(x, y)$

$dz := \frac{\Delta z}{\Delta x}$

while $x < x_B$

$x := x + 1$

if $dz < 0.5$

then $dz := dz + \frac{\Delta z}{\Delta x}$

else $dz := dz - 1 + \frac{\Delta z}{\Delta x}$, $z := z + 1$

Schreibe $z(x,y)$

end

$x := x_A, \quad y := y_A, \quad z := z_A$

$\Delta x = x_B - x_A, \quad \Delta z = z_B - z_A$

Schreibe $z(x, y)$

$d^* := 2 \cdot \Delta z$ $d^*=4$

while $x < x_B$

$x := x + 1$

if $d^* < \Delta x$ $d^*<9$

then $d^* := d^* + 2 \cdot \Delta z$

else $d^* := d^* - 2 \cdot \Delta x + 2 \cdot \Delta z$, $z := z + 1$

Schreibe $z(x,y)$

end

ISSN 1432-3346

Berechnung der Werte für den Z-Buffer (Bresenham)

- Z-Buffer i.a. für ganzzahlige z-Werte → Berechnung mit Rundung → Bresenham
- Ohne Einschränkung der Allgemeinheit → $\Delta z/\Delta x < 1$ darf angenommen werden

$x := x_A, \quad y := y_A, \quad z := z_A$

$\Delta x = x_B - x_A, \quad \Delta z = z_B - z_A$

Schreibe $z(x, y)$

$dz := \frac{\Delta z}{\Delta x}$

while $x < x_B$

$x := x + 1$

if $dz < 0.5$

then $dz := dz + \frac{\Delta z}{\Delta x}$

else $dz := dz - 1 + \frac{\Delta z}{\Delta x}$, $z := z + 1$

Schreibe $z(x,y)$

end

$x := x_A, \quad y := y_A, \quad z := z_A$

$\Delta x = x_B - x_A, \quad \Delta z = z_B - z_A$

Schreibe $z(x, y)$

$d := 2 \cdot \Delta z - \Delta x$ $d= -5$

while $x < x_B$

$x := x + 1$

if $d < 0$

then $d := d + 2 \cdot \Delta z$

else $d := d - 2 \cdot \Delta x + 2 \cdot \Delta z$, $z := z + 1$

Schreibe $z(x,y)$

end

Multiplikation aller dz-(Un)Gleichungen mit $2 \cdot \Delta x$ und Offset von $-\Delta x$ bei der Initialisierung von d zur Vereinfachung des Tests (if-Zeile)

ISSN 1432-3346

Direkte Schattierungsverfahren (Double Buffer)

Double Buffering

- Ziel: Illusion der Bewegung → erfordert schnelle, ungestörte Bildfolgen →
- Bedingung: Betrachter sieht immer nur komplette Bilder, niemals das Bild während der Entstehung (Bildaufbau)

Verwendung von zwei Bildspeichern (Double-Buffer-Prinzip):

- Aktueller auf dem Bildschirm angezeigter Bildspeicher enthält die vollständige Abbildung der zuvor schattierten Szene.
- Im zweiten Puffer wird das neue Bild berechnet.
- Nach Fertigberechnung des neuen Bildes wird der Bildschirm auf den zweiten Puffer umgeschaltet.
- Der erste dient nun zum Aufbau des nächsten Bildes.

Direkte Schattierungsverfahren (Transparenz)

Transparenz beim z-Puffer-Verfahren

C_f Farbe des Polygons im Vordergrund
α Opazität der Vordergrundfarbe
C_b Hintergrundfarbe (die im Bildspeicher für das entsprechende Pixel zuletzt eingetragene Farbe)

Die resultierende Farbe C ergibt sich zu:

$$C = \alpha \cdot C_f + (1 - \alpha) \cdot C_b$$

z-Wert des Vordergrund-Polygons (transparentes Polygon)

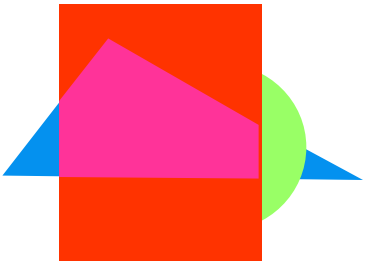
Problem:

Schiebt sich nach Behandlung des semitransparenten Objektes ein weiteres Polygon zwischen das ursprüngliche Hintergrundpolygon und das transparente, kann die Farbe nicht mehr korrekt bestimmt werden.

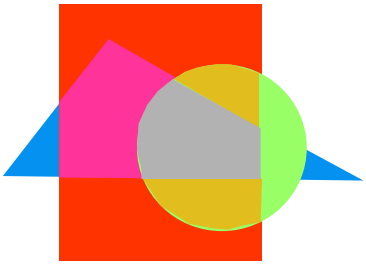
- Der Grundalgorithmus des z-Buffering ignoriert Objektepixel, die weiter entfernt sind als das semitransparente.
- Selbst bei differenziertem Vorgehen auf der Basis eines α - Speichers steht im Bildspeicher die Mischfarbe (ist nicht die Farbe des semitransparenten Objektes)

Direkte Schattierungsverfahren (Transparenz)

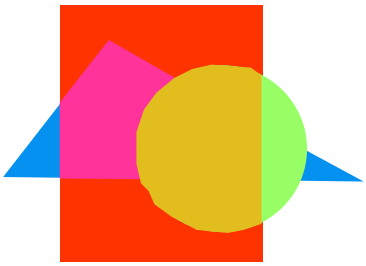
Opake grüne Scheibe schiebt sich zwischen halbtransparentes rotes Rechteck und opakes blaues Dreieck.



Reine z-Pufferung (fernere Objekte werden nicht berücksichtigt)



Berücksichtigung des α-Wertes, jedoch falscher Vordergrund



korrekte Lösung

Lösung:

- Darstellung *aller opaken Objekte (α=1) nach dem z-Pufferverfahren*.
- *Sortieren aller semitransparenten Polygone* nach der Tiefe und zeichnen nach dem *Painters-Algorithmus unter Berücksichtigung des z-Puffers*.
- *Restfehler:* Sich zyklisch überlappende oder sich durchdringende semitransparete Flächen → exakte Behandlung durch die vorn beschriebenen Maßnahmen.