

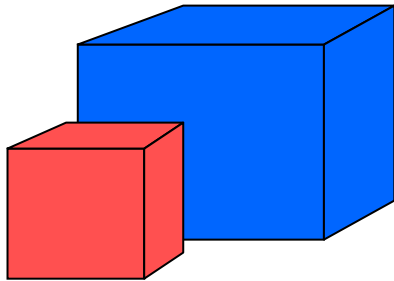
Computergrafik 2008

Oliver Vornberger

Kapitel 17:
Culling

Definition

to cull something = etwas loswerden



Objektraum:

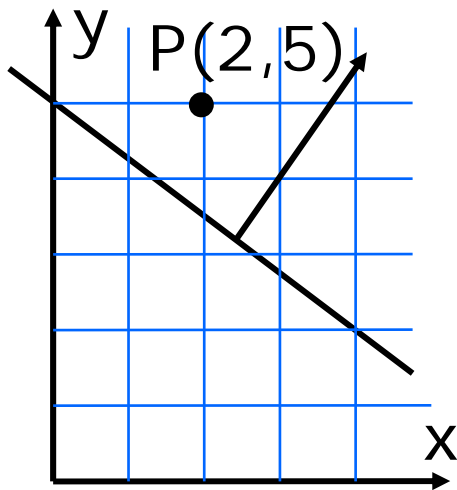
- back face culling
- Vergleich von Flächen

Bildraum:

- hidden surface removal
- Vergleich von Pixeln

Geradengleichung

- $Q(0, \infty)$



$$y = -\frac{3}{4}x + 5$$

$$\frac{3}{4}x + y - 5 = 0$$

$$3x + 4y - 20 = 0$$

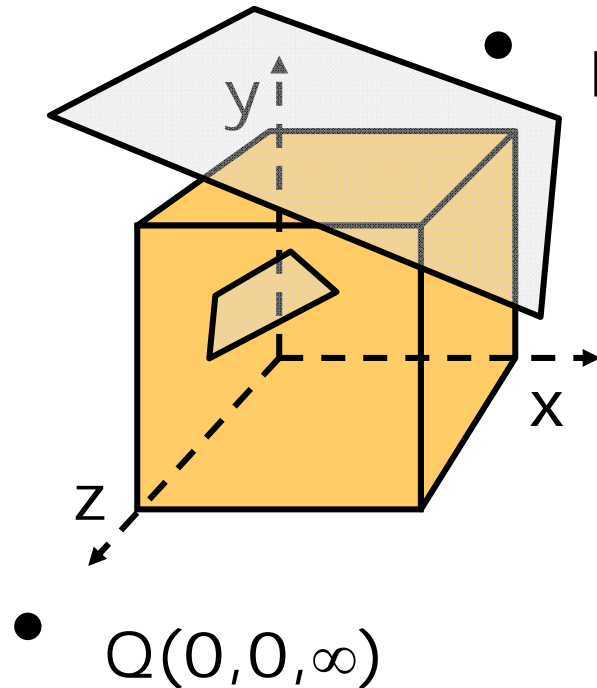
$$Ax + By + C = 0$$

$$\vec{p} = \begin{pmatrix} 2 \\ 5 \\ 1 \end{pmatrix} \quad \vec{n} = \begin{pmatrix} 3 \\ 4 \\ 0 \end{pmatrix}$$

sichtbar von P, falls $\vec{p} \cdot \vec{n} + C \geq 0$

sichtbar von Q, falls $B \geq 0$

Ebenengleichung



$P(x, y, z)$

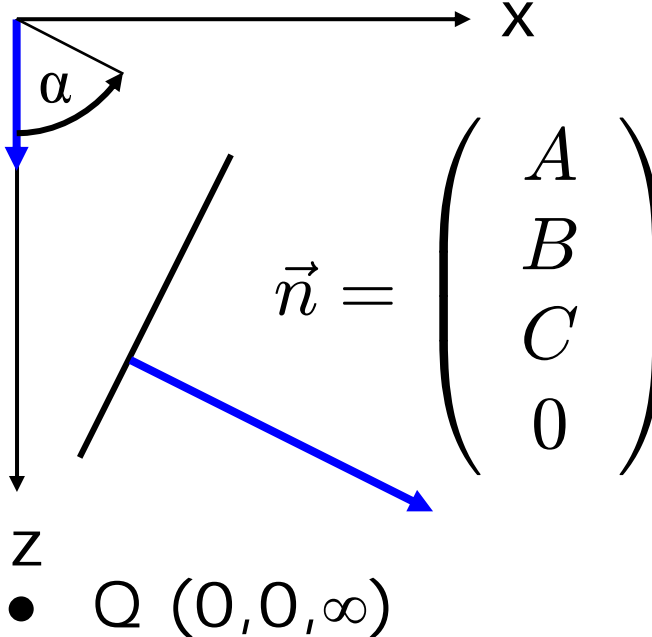
$$Ax + By + Cz + D = 0$$

$$\vec{p} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \vec{n} = \begin{pmatrix} A \\ B \\ C \\ 0 \end{pmatrix}$$

sichtbar von P, falls $\vec{p} \cdot \vec{n} + D \geq 0$

sichtbar von Q, falls $C \geq 0$

Alternative über Winkel

$$\vec{p} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$


$$\vec{n} = \begin{pmatrix} A \\ B \\ C \\ 0 \end{pmatrix}$$

$$\cos(\alpha) = \frac{\vec{p} \cdot \vec{n}}{|\vec{p}| \cdot |\vec{n}|}$$

• Q (0,0, ∞)

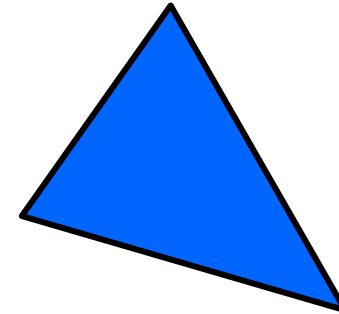
Fläche sichtbar von Q, falls Winkel $\alpha \leq 90^\circ$

Winkel $\alpha \leq 90^\circ$, falls $\vec{p} \cdot \vec{n} \geq 0$

Back Face Culling

für jede Polygonfläche:

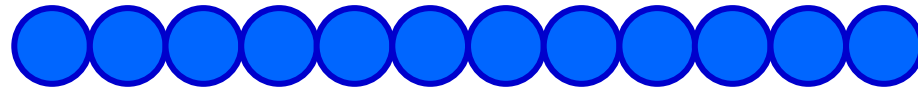
- berechne z-Komponente der Flächennormale im NPC



$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \cdot$$

- falls Ergebnis $< 0 \Rightarrow$ Face nicht sichtbar
- \Rightarrow vergiss es !

Hidden Surface Removal

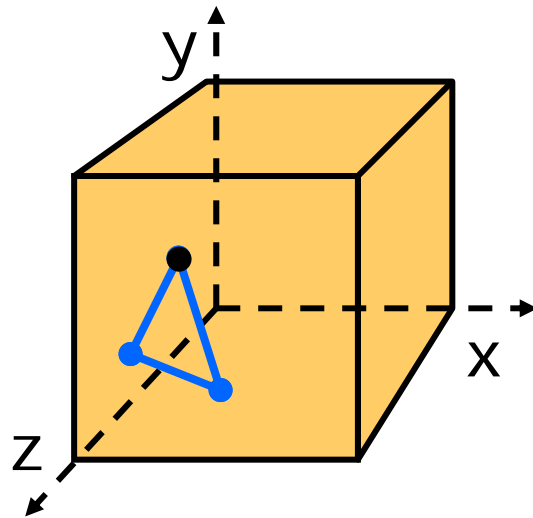


Berechne sichtbare Pixel einer Rasterzeile

- z-Buffer
- Painter's Algorithm
- Span-Buffer Algorithm
- Binary Space Partition Tree

z-Buffer

Ergebnis von Device Mapping: $P(x,y,z)$



große z-Werte vorne
(näher am Betrachter)

```
double[][] tiefe;    // z-Buffer  
Color[][] bild;     // frame buffer
```


z-Buffer-Algorithmus

```
initialisiere bild[][] mit Hintergrundfarbe
initialisiere tiefe[][] mit 0.0

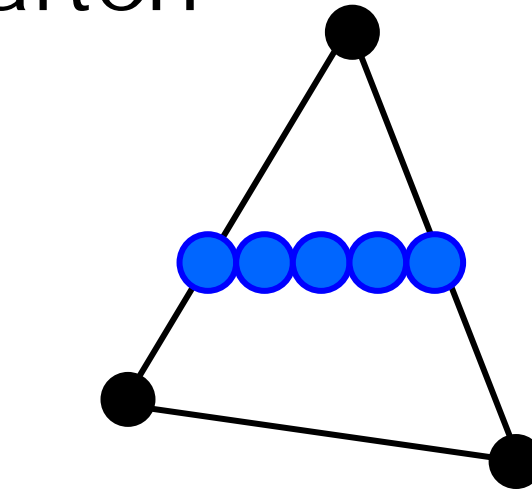
für jede Fläche F tue {
    für jedes Pixel (x,y) auf F tue {
        berechne Farbe c und Tiefe z
        if (z > tiefe[x][y]) {
            tiefe[x][y] = z;
            bild [x][y] = c;
        }
    }
}
```

Nachbarschaften

$$Ax + By + Cz + D = 0$$

$$z = -\frac{Ax + By + D}{C}$$

$$z_i = -\frac{Ax_i + By_i + D}{C}$$



$$= x_i + 1$$

$$z_{i+1} = -\frac{Ax_{i+1} + By_i + D}{C}$$

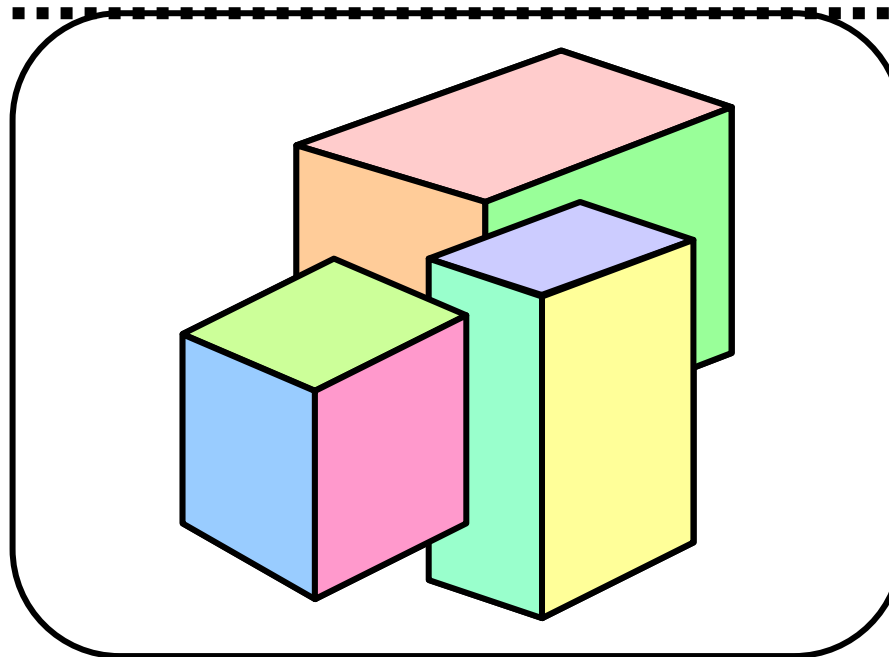
$$= z_i - \frac{A}{C}$$

selbe Zeile, nächster x-Wert

Speicherbedarf

- z-Werte nahe Backplane dicht beieinander
- Auflösung für Tiefe: 32 Bit Double
- Auflösung für Farbe: 24 Bit Integer
- Auflösung für Transparenz: 8 Bit Integer
- pro Pixel \Rightarrow 8 Byte
- bei 1024×768 Pixeln \Rightarrow 6 MB

Economy-Version



Nutze den Z-Buffer jeweils für eine Zeile

Analyse z-Buffer

pro Pixel (x,y):

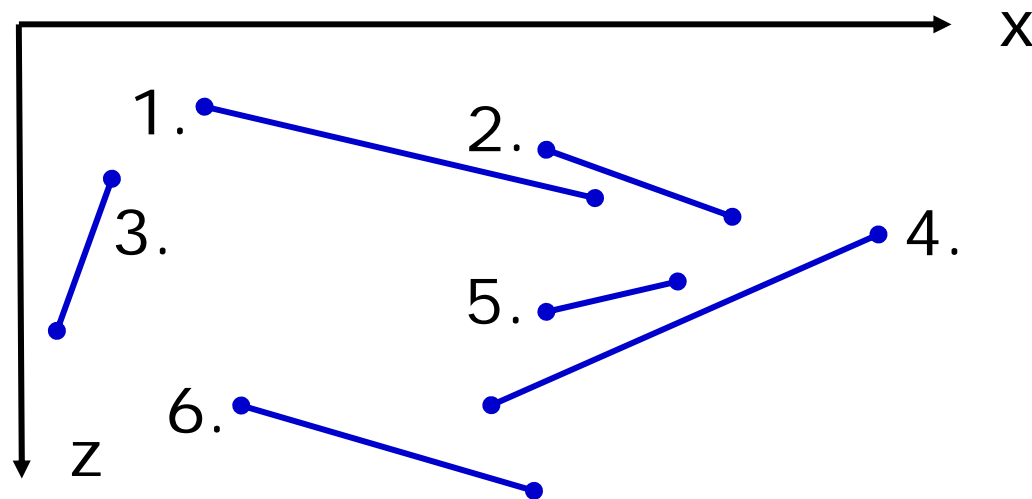
- z-Wert berechnen + testen,
ob z-Wert größer als tiefe[x][y]
- Pixel wird ggf. später übermalt

Wunsch:

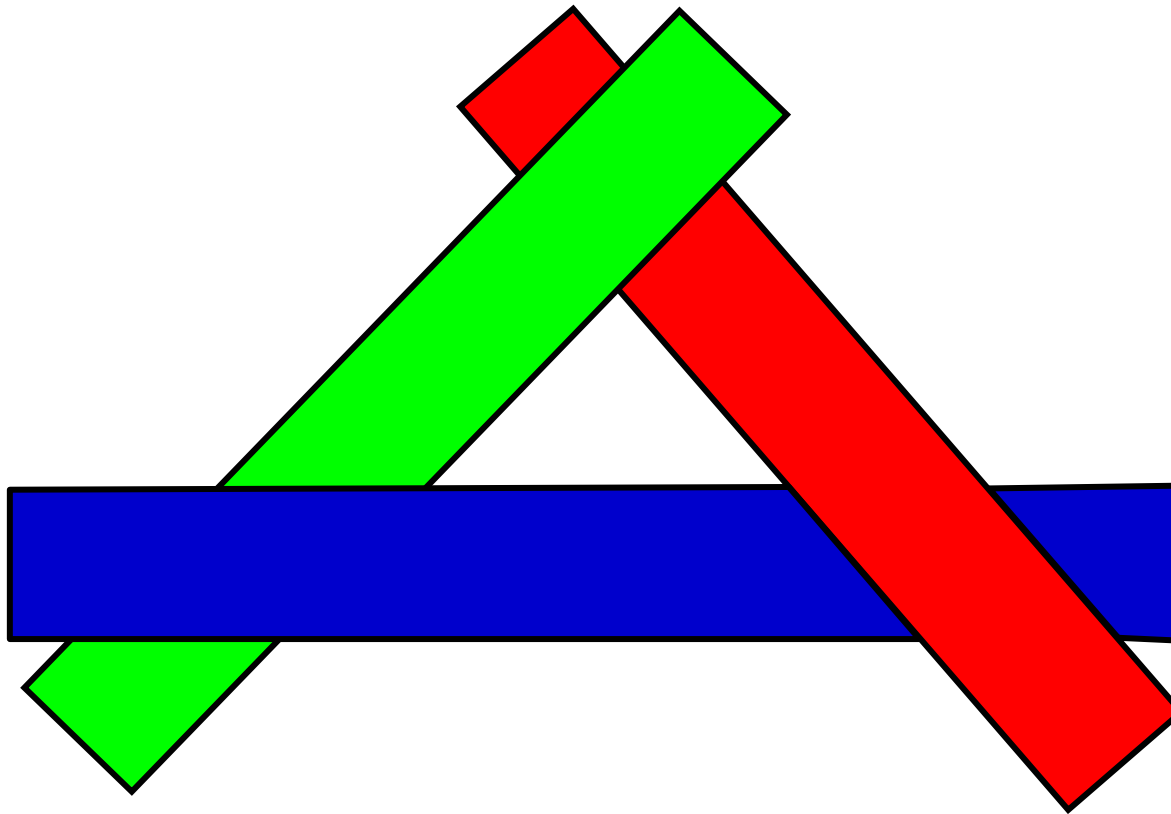
- Tiefentest vermeiden
- doppeltes Rendern vermeiden

Painter's Algorithm

- ordne alle Polygone nach kleinstem z-Wert
- Polygone mit überlappender z-Ausdehnung ggf. umordnen
- Ausgabe von hinten nach vorne

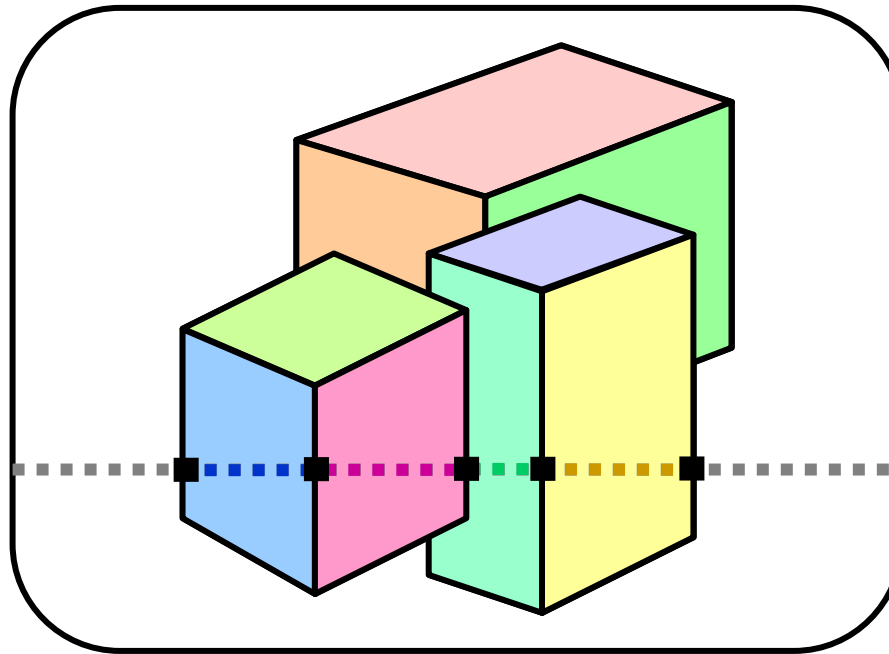


Problem beim Painter

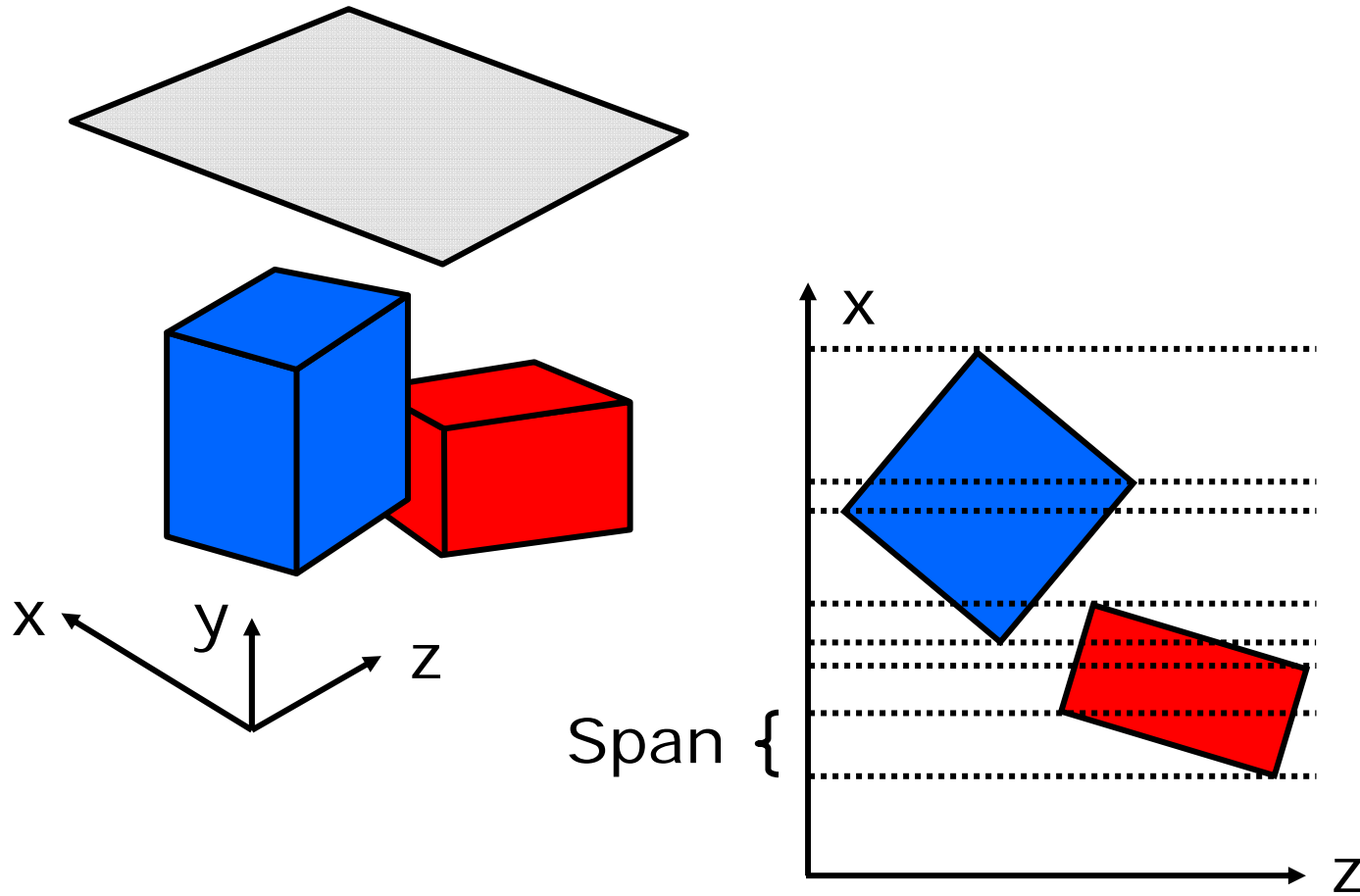


Span-Buffer-Algorithmus

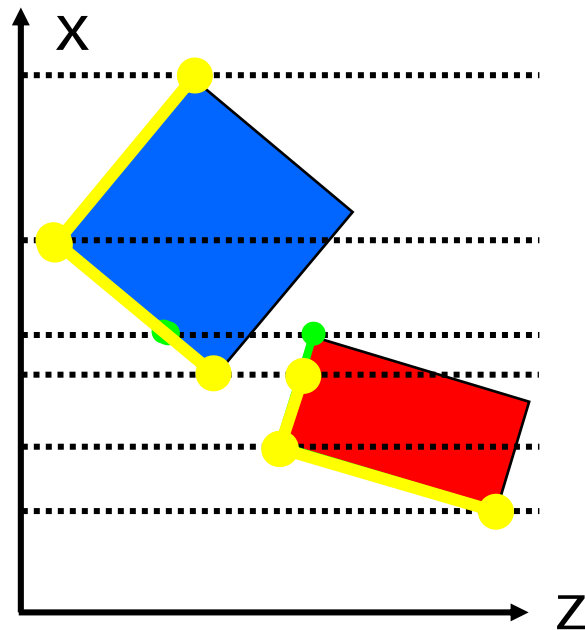
- Scanline durchläuft Bild
- Scanline zerfällt in Abschnitte = Spans
- pro Span ist genau ein Polygon zuständig



Scanline



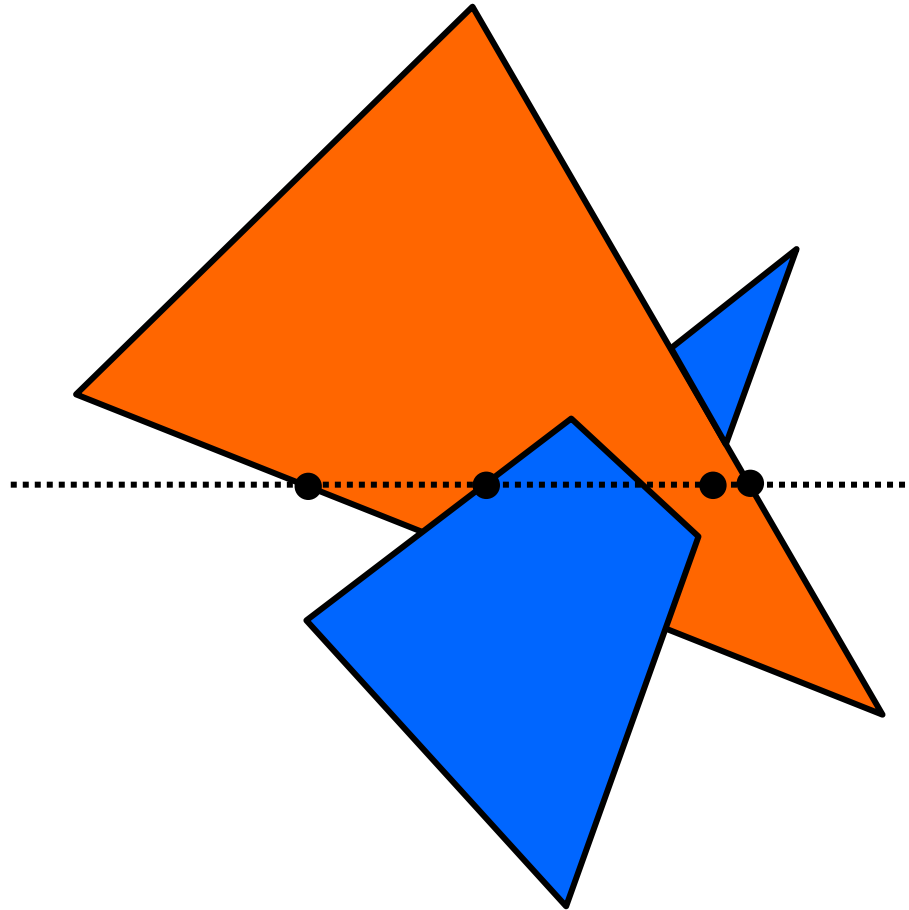
Spans



nur Vorderflächen
nach x sortieren
ggf. zerschneiden
vordersten finden
Spans vereinigen

- + Rendern eines Spans ohne Test auf Tiefe
- + Rendern eines Pixels ohne Überschreiben
- hoher Aufwand für Ermittlung der Spans

Probleme mit Spans



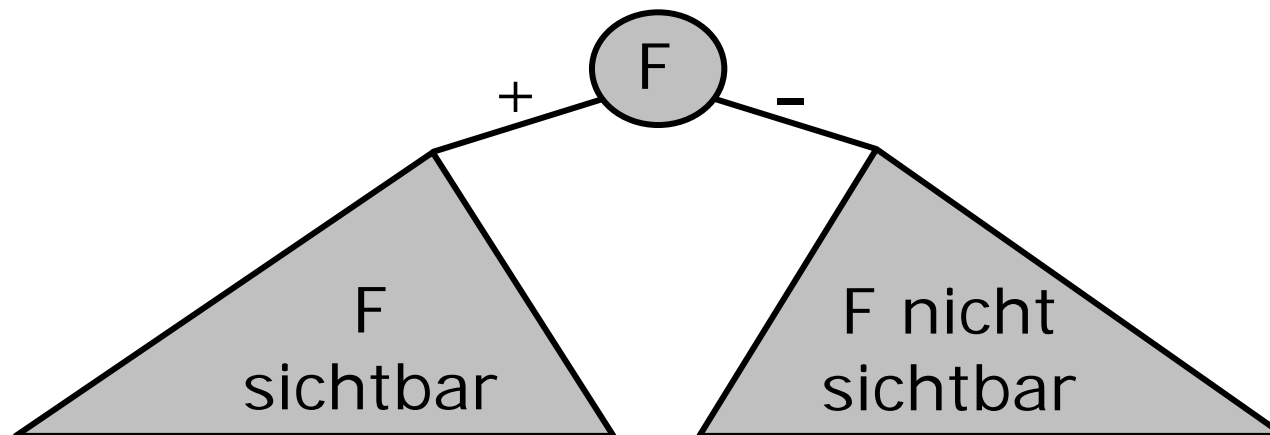
gegenseitige
Durchdringung
wird nicht
erkannt !

Binary Space Partition

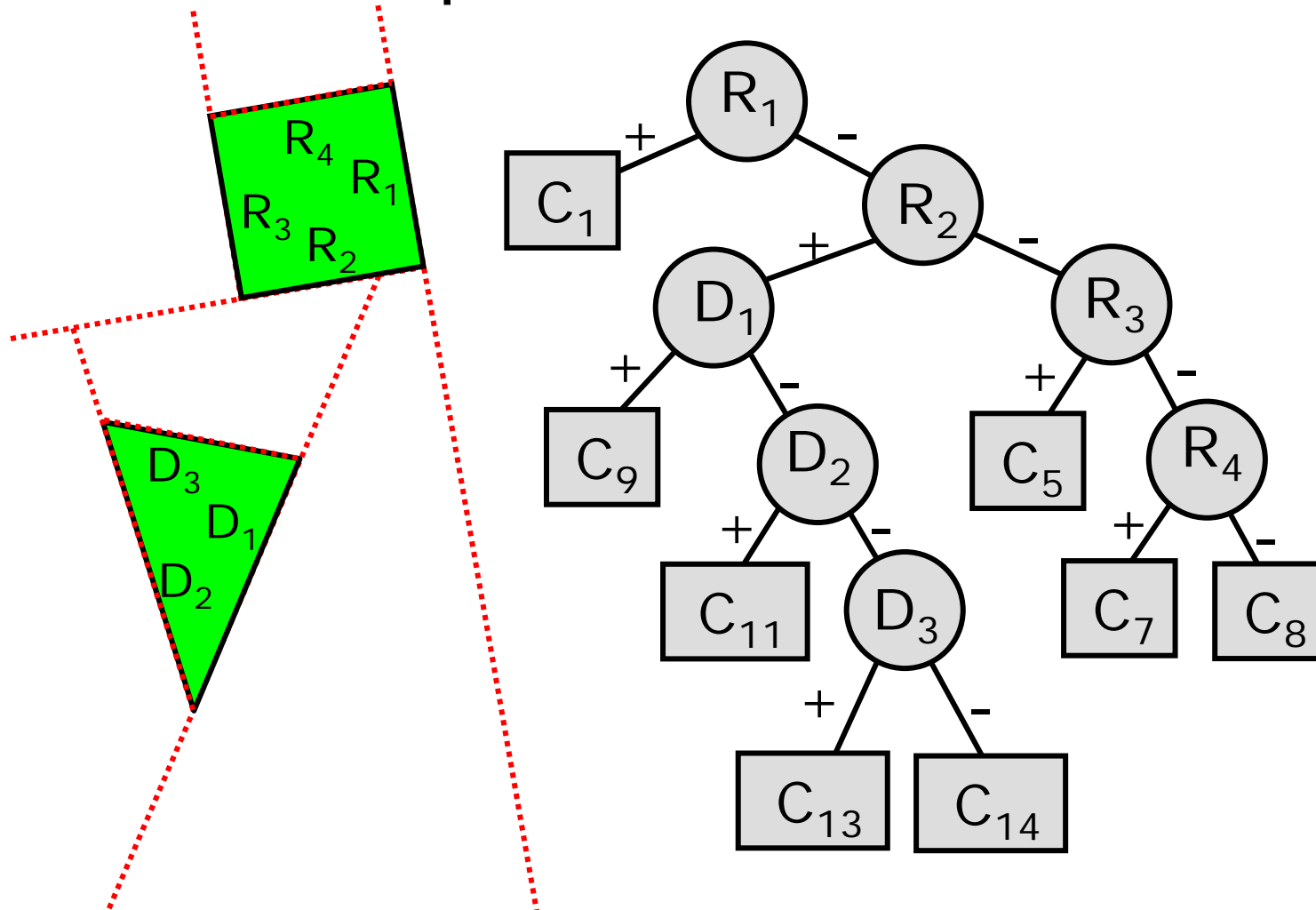
- Analyse der räumlichen Beziehungen
- hoher Aufwand für Vorbereitung
- unabhängig vom Betrachterstandpunkt
- nutzbar für beliebige Augenpunkte
- geeignet bei Kamerafahrt

BSP-Tree

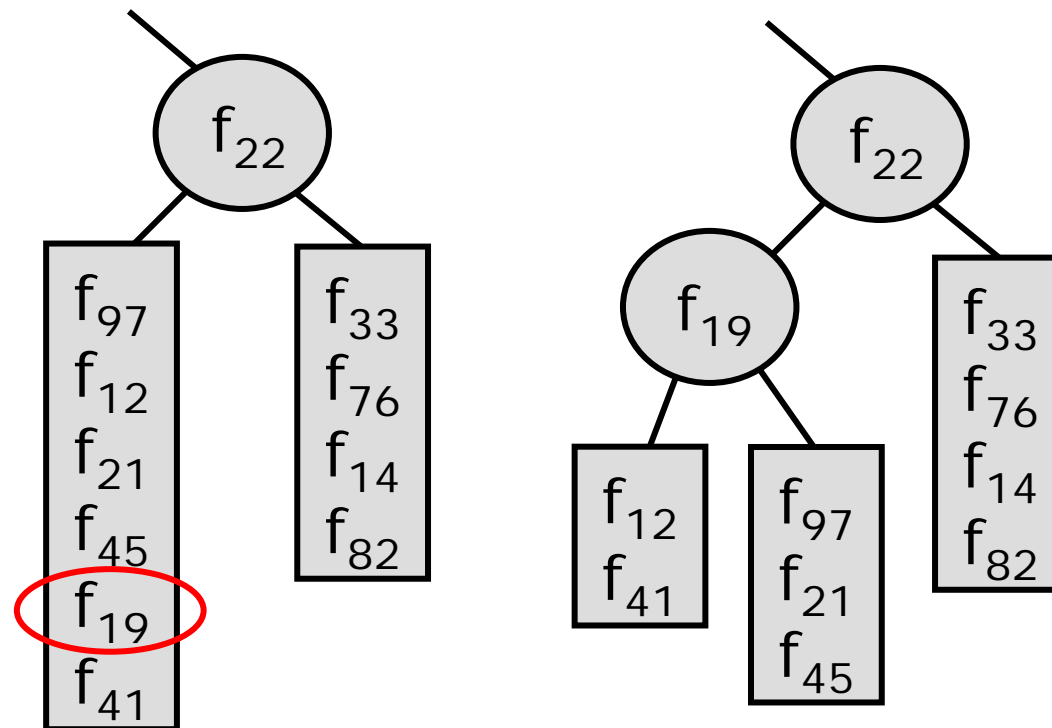
- jeder innere Knoten repräsentiert eine Polygonfläche F , welche die Szene aufteilt in "vorderen" Teil (von dort ist F sichtbar) und "hinteren" Teil (von dort ist F nicht sichtbar)
- jedes Blatt repräsentiert einen Teilraum



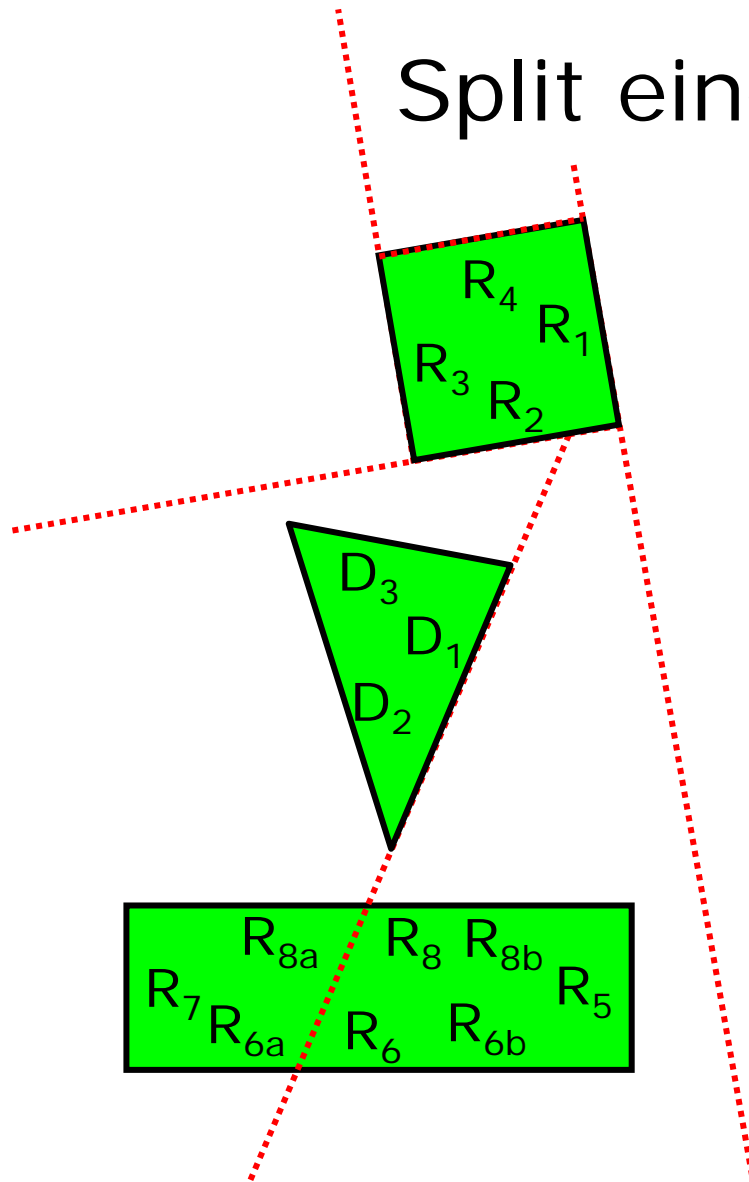
Beispiel für BSP-Tree



Split in BSP-Tree



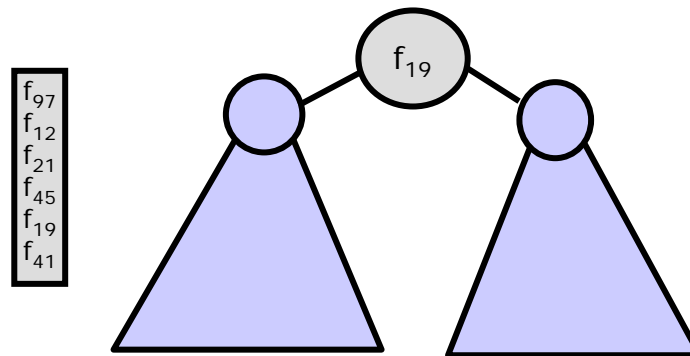
Split eines Polygons



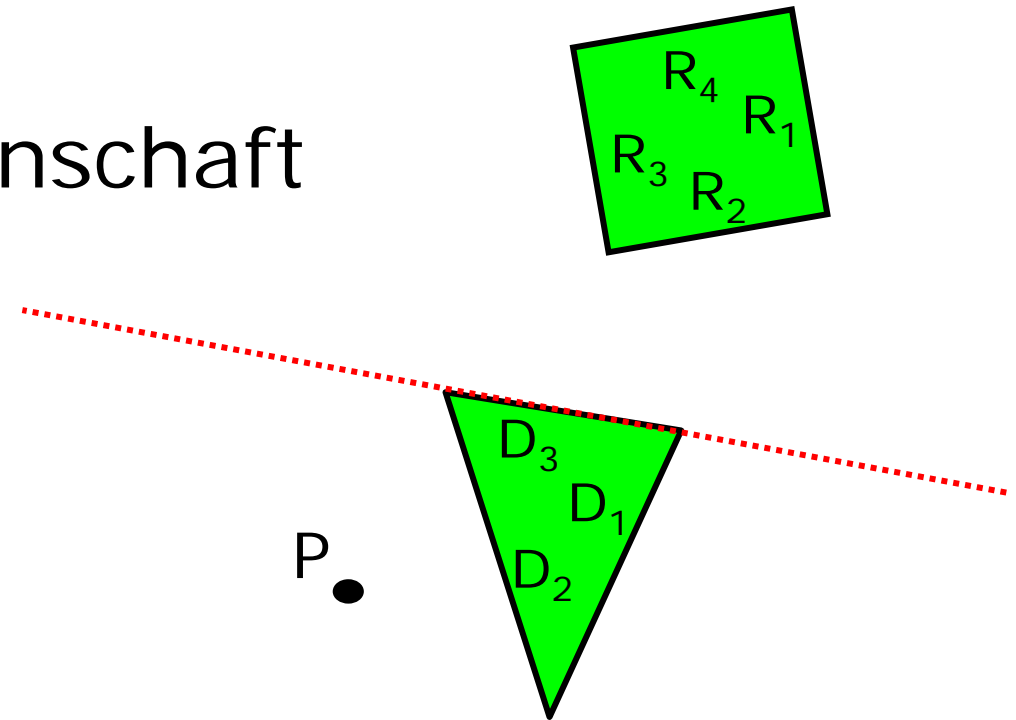
geht Trennebene
durch Polygon,
so wird es in zwei
Polygone zerlegt

BSP-Tree erzeugen

```
bspTree makeTree(PolygonList L){  
    wähle Polygon root aus L;  
    bilde PolygonList front;  
    bilde PolygonList back;  
    bspTree f = makeTree(front);  
    bspTree b = makeTree(back);  
    return new bspTree(f,root,b);  
}
```



BSP-Tree-Eigenschaft



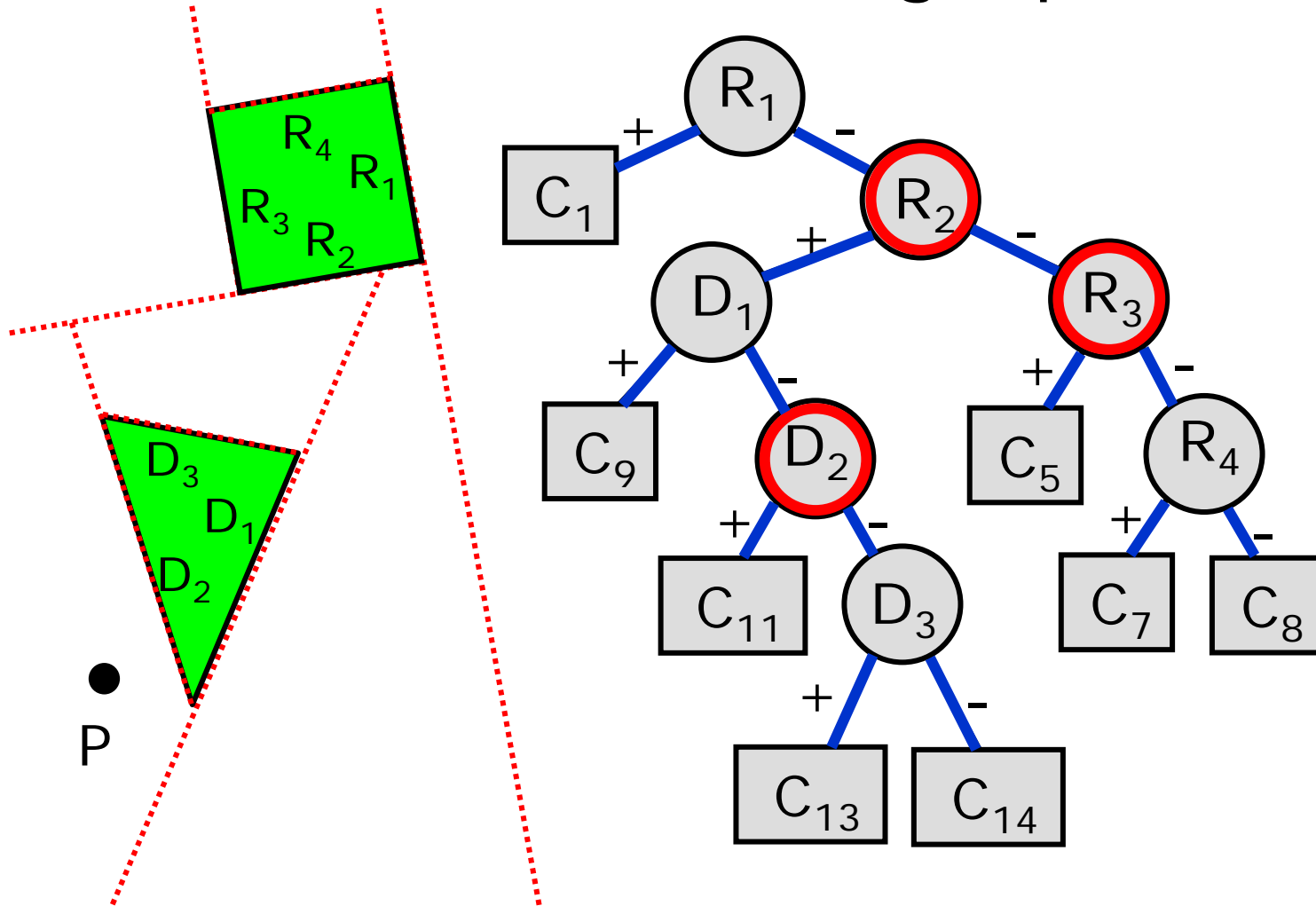
Flächen, die auf derselben Seite liegen wie der Augenpunkt,

- können Flächen auf der anderen Seite verdecken
- können nicht verdeckt werden von Flächen auf der anderen Seite

BSP-Tree-Traversierung

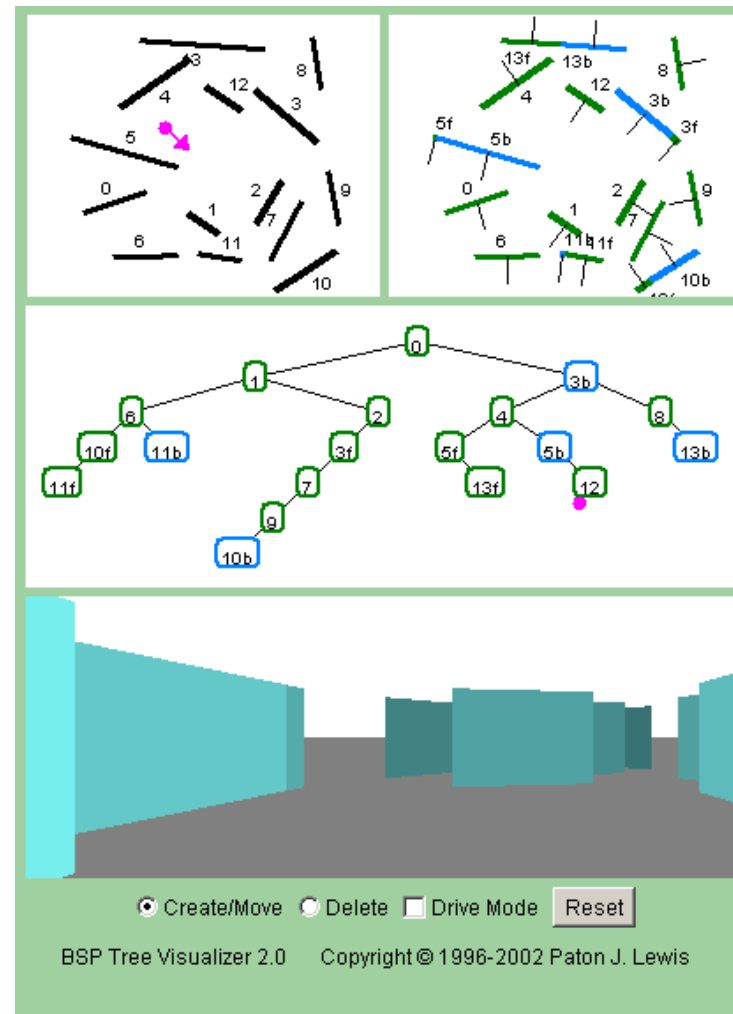
```
void bspOrder(bspTree b, Point P){
    if (!b.empty()) {
        if (P liegt vor b.root()) {
            bspOrder(b.back(),P);
            display(b.root());
            bspOrder(b.front(),P);
        } else {
            bspOrder(b.front(),P);
            // Rückseite unterdrückt
            bspOrder(b.back(),P);
        }
    }
}
```

Sichtbarkeit vom Augenpunkt



Applet zum BSP-Tree

von
Paton J. Lewis,
Symbolcraft



~cg/2008/skript/Applets/bsp-tree/bsp-tree.html