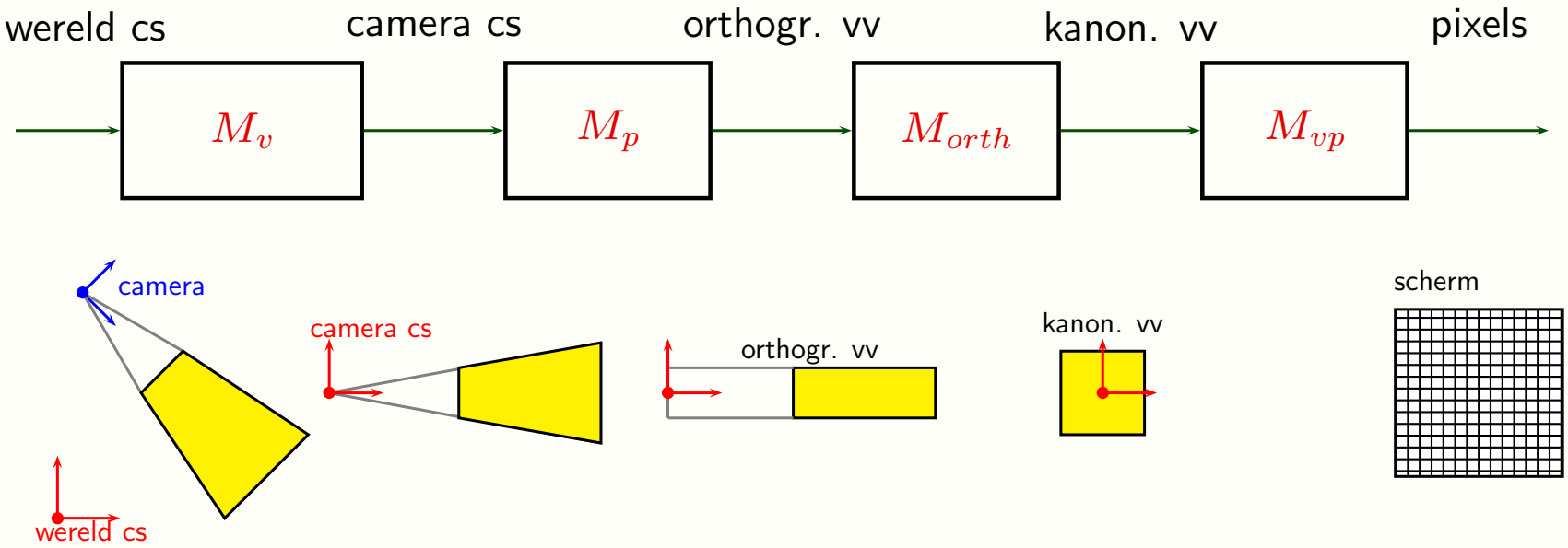
Viewing, BSP trees en Bezier krommen

Doel: creëren van 2D beelden van 3D scenes

- Grafische pijplijn
- **Orthografische** projectie
- **Perspectief** projectie
- BSP-trees voor hidden surface elimination
- Bézier krommen

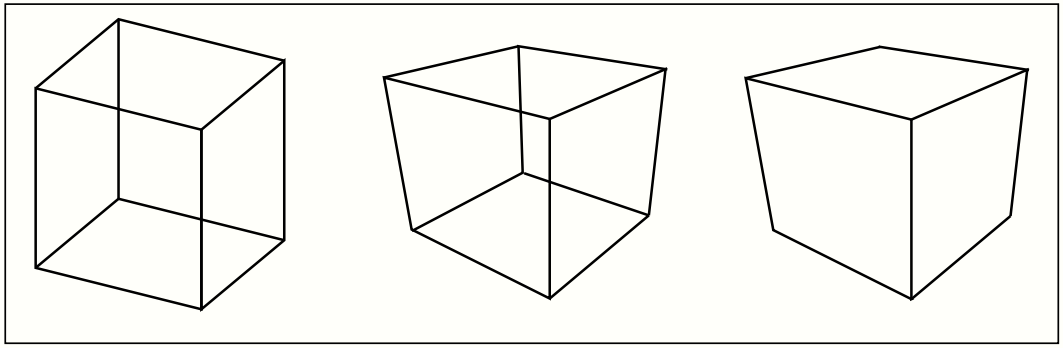
Grafische pijplijn



Projecties

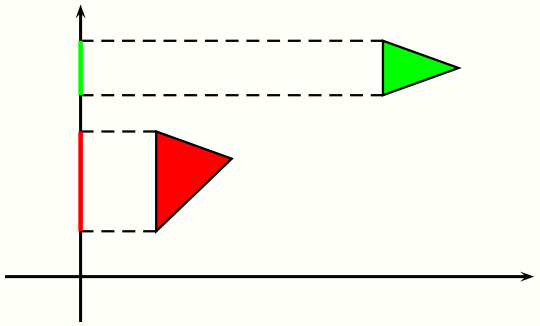
Methoden voor genereren van **projecties** van lijnsegmenten

- `orthografische` projectie (links)
- `perspectief` projectie (midden en rechts)



Parallelle projectie

Transformeert 3D punt naar 2D langs projectie richting op beeldvlak

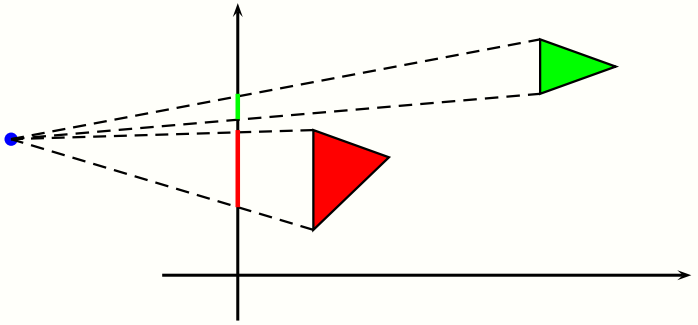


beeldvlak $\perp$ projectie richting:
orthografische projectie

- parallelle lijnen blijven parallel
- grootte en vorm van objecten blijven behouden

Perspectief projectie

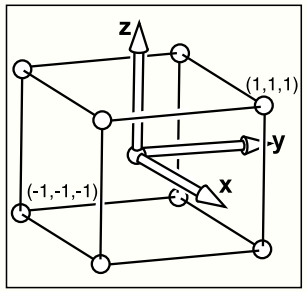
Transformeert 3D punt naar 2D langs lijn door viewpunt tot snijpunt met beeldvlak

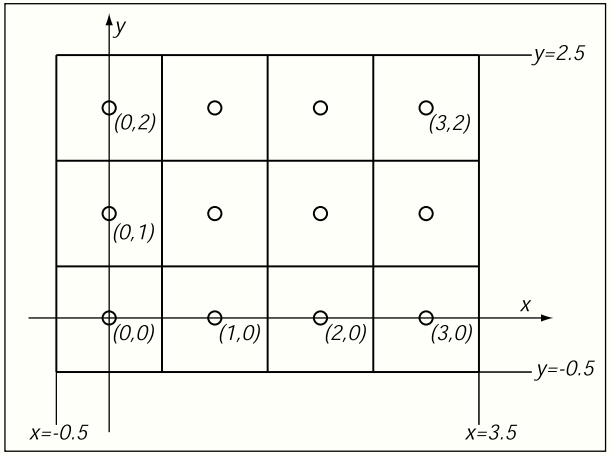


beeldvlak $\perp$ z-as:
perspectief projectie

- objecten verder van viewpunt vandaan worden kleiner

M_{vp}: kanonieke view volume $\rightarrow$ scherm

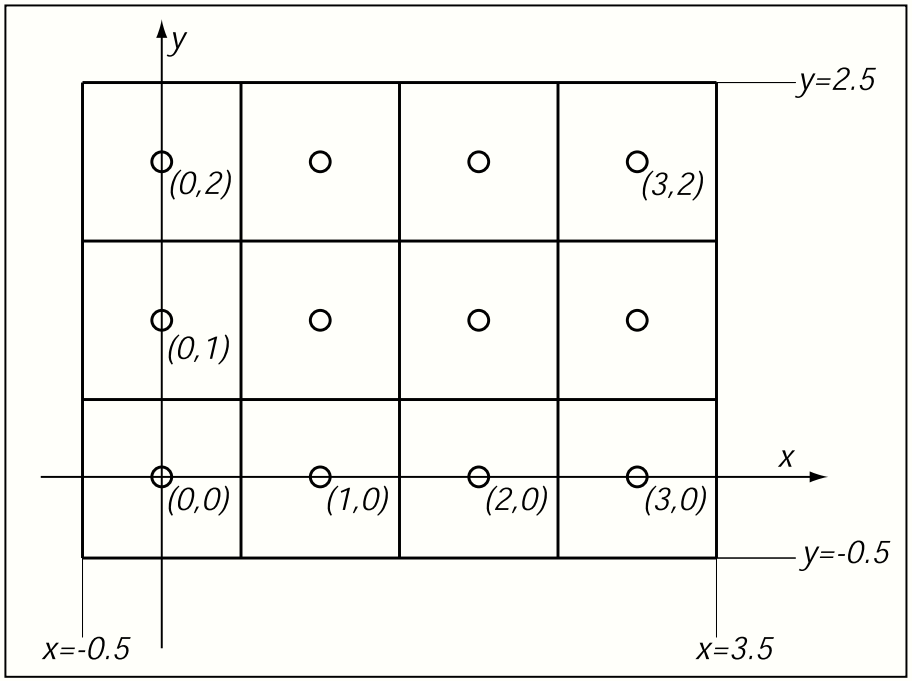




Zoek afbeelding **view volume** naar `scherm` langs z-as

Punten in kanonieke view volume worden orthografisch geprojecteerd op $n_x \times n_y$ beeld

Schermcoördinaten



Window van $[-0.5, n_x - 0.5]$ tot $[-0.5, n_y - 0.5]$
$(n_x = 4, n_y = 3)$

Transformatie van viewcoördinaten naar schermcoördinaten

rechthoek $(a, b)(A, B) \rightarrow (c, d)(C, D)$

`transformatie` $=$
transleer (c, d) schaal $(\frac{C-c}{A-a}, \frac{D-d}{B-b})$ transleer $(-a, -b)$

rechthoek $(-1, -1)(1, 1) \rightarrow (-0.5, -0.5)(n_x - 0.5, n_y - 0.5)$

`transformatie` $=$
transleer $(-0.5, -0.5)$ schaal $(\frac{n_x}{2}, \frac{n_y}{2})$ transleer $(1, 1)$

Transformatie van view volume naar scherm

`transformatie` $=$
transleer $(-0.5, -0.5)$ schaal $(\frac{n_x}{2}, \frac{n_y}{2})$ transleer $(1, 1)$

$$\begin{pmatrix} 1 & 0 & -0.5 \\ 0 & 1 & -0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{n_x}{2} & 0 & 0 \\ 0 & \frac{n_y}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} =$$

$$\begin{pmatrix} \frac{n_x}{2} & 0 & -0.5 \\ 0 & \frac{n_y}{2} & -0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

Transformatie M_{vp}

$$\begin{pmatrix} x_{pixel} \\ y_{pixel} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{n_x}{2} & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_y-1}{2} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{canonical} \\ y_{canonical} \\ 1 \end{pmatrix}$$

Orthografische projectie heeft z-waarden "weggegooid"

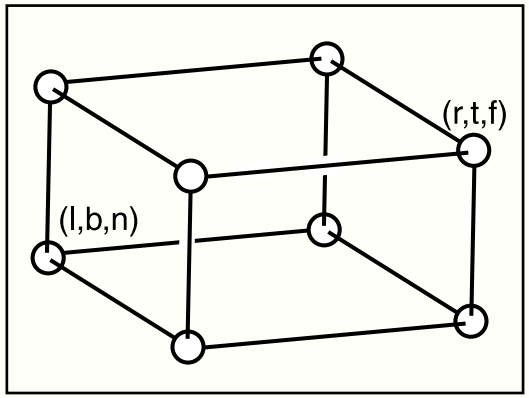
Vaak z-waarden gebruikt voor **ordenen** van oppervlakken

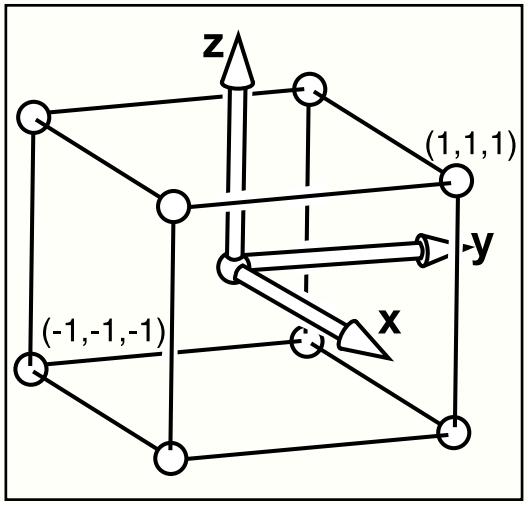
$$\begin{pmatrix} x_{pixel} \\ y_{pixel} \\ z_{canonical} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{canonical} \\ y_{canonical} \\ z_{canonical} \\ 1 \end{pmatrix}$$

M_{orth}: orthografische vv $\rightarrow$ kanonieke vv

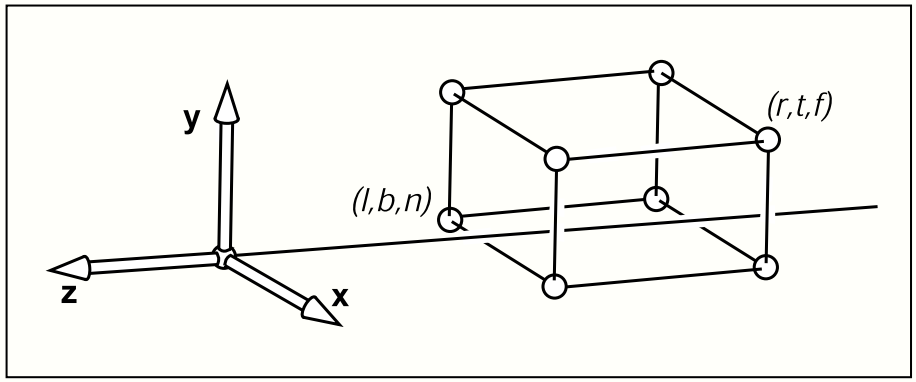
Zoek **afbeelding** van orthografische view volume naar `kanonieke view volume`

box $[\texttt{l, r}] \times [\texttt{b, t}] \times [\texttt{n, f}]$





Orthografische view volume



Viewer kijkt in richting van **negatieve z-as**, dus $n > f$

`orthografische view volume` $\rightarrow$ **standaard view volume** is 3D versie van windowtransformatie: $\mathbf{S_c T_O}$

M_{orth}: orthografische vv $\rightarrow$ kanonieke vv

$$x = l \rightarrow x = -1 \qquad x = r \rightarrow x = 1$$
$$y = b \rightarrow y = -1 \qquad y = t \rightarrow y = 1$$
$$z = n \rightarrow z = -1 \qquad z = f \rightarrow z = 1$$

schaal **transleer naar oorsprong**

$$\begin{pmatrix} x_{can} \\ y_{can} \\ z_{can} \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$M_O = M_{vp} M_{orth}$: orthografische vv $\rightarrow$ scherm

3D lijnsegmenten projecteren op scherm, **z**-waarden?

$$M_O = \begin{pmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{n-f} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{n+f}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} x_{pixel} \\ y_{pixel} \\ z_{canonical} \\ 1 \end{pmatrix} = M_O \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

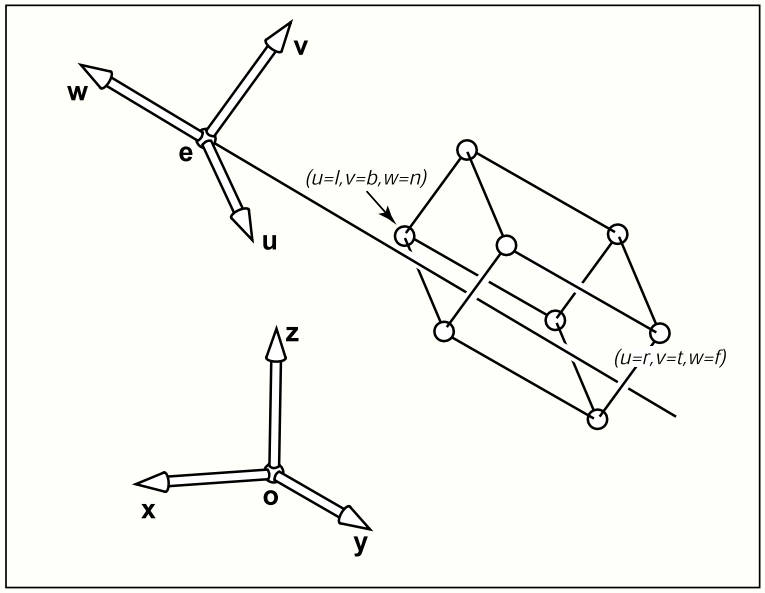
M_O: orthografische vv $\rightarrow$ scherm

$$\begin{pmatrix} x_{pixel} \\ y_{pixel} \\ z_{canonical} \\ 1 \end{pmatrix} = M_O \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$z_{canonical}$ in [-1,1] (gebruikt in **z-buffer algoritme**)

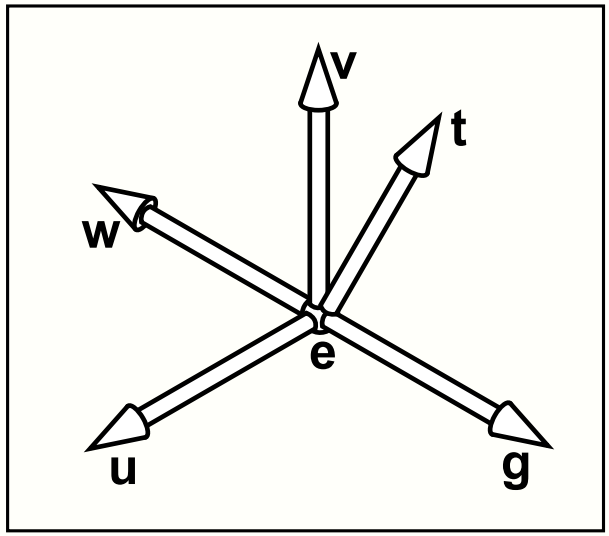
```
bereken MO
for elk lijnsegment (a, b) doe
    p = MO a
    q = MO b
    draw(p, q)
```

M_v: wereldcoördinaten naar cameracoördinaten



- wereldcoördinaten in standaardbasis x, y en z
- cameracoördinaten met camerapositie e en basis u, v en w

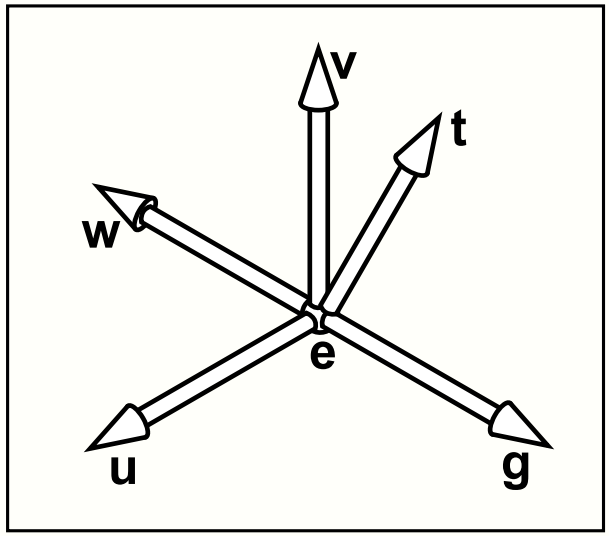
Orthonormale basis van cameracoördinaten



Gebruiker specificeert:
oogpositie $\mathbf{e}$
`kijkrichting` $\mathbf{g}$
`view-up` $\mathbf{t}$

Construeer **orthonormale basis** $\mathbf{uvw}$ met $\mathbf{w}$ tegengesteld aan kijkrichting en $\mathbf{v}$ in vlak van $\mathbf{g}$ en $\mathbf{t}$

Coördinatensysteem met oorsprong $\mathbf{e}$ en basis $\mathbf{uvw}$



- $\mathbf{w} = -\frac{\mathbf{g}}{||\mathbf{g}||}$
- $\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{||\mathbf{t} \times \mathbf{w}||}$
- $\mathbf{v} = \mathbf{w} \times \mathbf{u}$

Orthonormale basis uvw

1. Alignen van uvw naar xyz

$$\mathbf{u} \xrightarrow{R} x$$

$$\mathbf{v} \xrightarrow{R} y$$

$$\mathbf{w} \xrightarrow{R} z$$

2. Veranderen coördinatensysteem

$$\begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{pmatrix}_{x,y,z} \xrightarrow{R} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix}_{\mathbf{u},\mathbf{v},\mathbf{w}}$$

$\mathbf{R} = \mathbf{R_{uvw}}$

$$\mathbf{R_{uvw}} = \begin{pmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{pmatrix}$$

$\mathbf{R_{uvw}}$ voert punt in `wereldcoördinaten` over naar punt in `cameracoördinaten` op **offset** na: namelijk de positie van camera

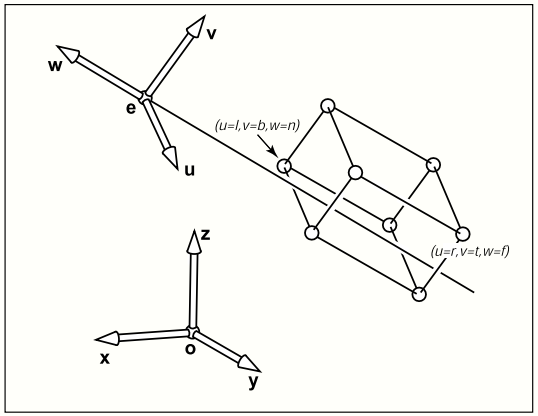
Transformatie:

1. translatie van $\mathbf{e}$ naar oorsprong

2. $\mathbf{uvw}$ roteren naar $\mathbf{xyz}$

Wereldcoördinaten $\rightarrow$ cameracoördinaten

(View volume t.o.v. $\mathbf{uvw}$)



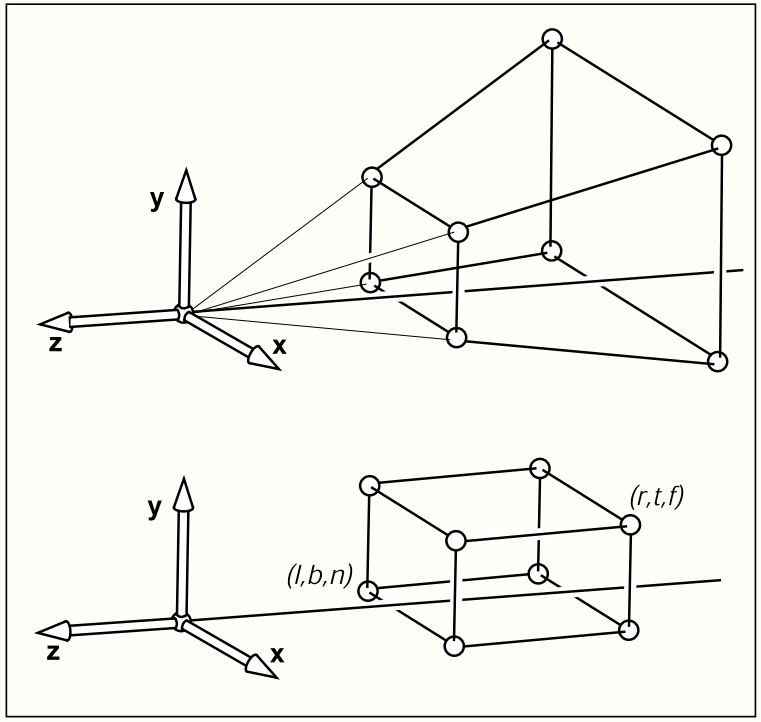
$$\mathbf{M_v} = \begin{pmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Wereld $\xrightarrow{orthogr.projectie}$ scherm

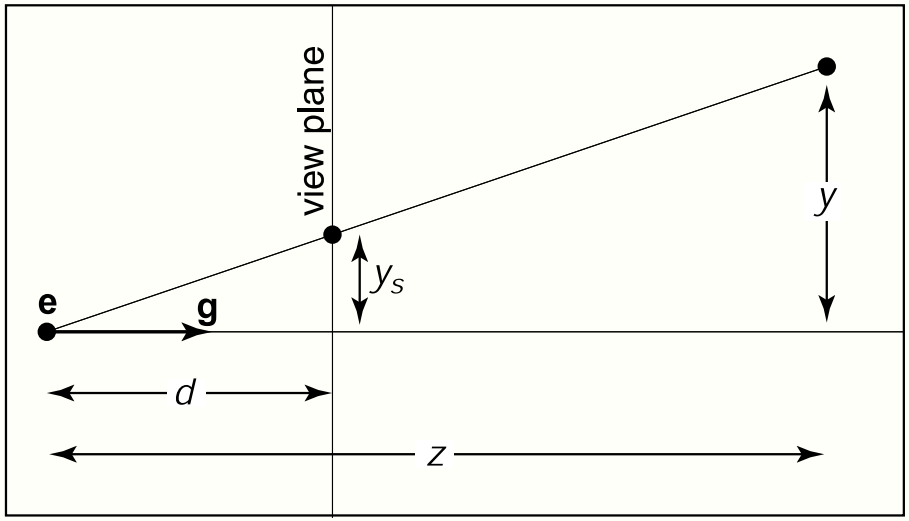
```
bereken MO
bereken Mv
M = MO Mv
for elk lijnsegment (a, b) doe
    p = M a
    q = M b
    draw(p, q)
```

$$\begin{pmatrix} x_{pixel} \\ y_{pixel} \\ z_{canonical} \\ 1 \end{pmatrix} = M_{vp} M_{orth} M_v \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

M_p: transformeren van view frustrum



Perspectief projectie

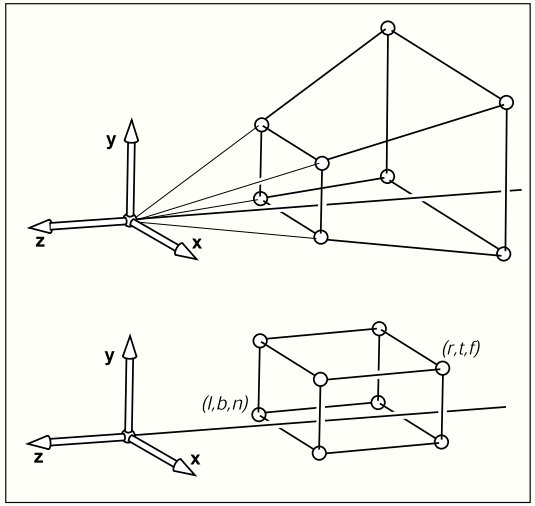


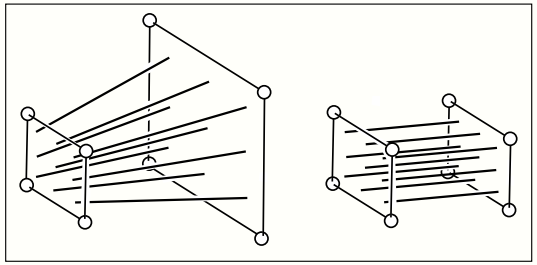
positie oog op $\mathbf{e}$
`kijkrichting` $\mathbf{g}$

`viewplane` op afstand **d**

$$y_s = \frac{d}{z} y$$

Perspectief transformatie





We hebben matrix nodig die het volgende doet:

1. $x_s = \frac{d}{z}x$ en $y_s = \frac{d}{z}y$
2. z-waarde blijft hetzelfde voor alle punten in near en far vlak

Perspectief transformatie $\mathbf{M_p}$

$$\mathbf{M_p} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Opmerkingen:

- We kijken in negatieve z-richting
- n en f geven voor- en achtervlak aan van view frustrum
- n dient als projectievlak

Perspectief transformatie $\mathbf{M_p}$

$$\begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ z(n+f)-fn \\ z \end{pmatrix} \xrightarrow{homogeen} \begin{pmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n+f-\frac{fn}{z} \\ 1 \end{pmatrix}$$

`z = n` $\xrightarrow{\mathbf{M_p}}$?

`z = f` $\xrightarrow{\mathbf{M_p}}$?

Perspectief transformatie $\mathbf{M_p}$

$$\mathbf{M_p}(\texttt{z = n}) = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ (n+f)n - fn \\ n \end{pmatrix} = \begin{pmatrix} x \\ y \\ n \\ 1 \end{pmatrix}$$

$$\mathbf{M_p}(\texttt{z = f}) = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ f \\ 1 \end{pmatrix} = \begin{pmatrix} nx \\ ny \\ (n+f)f - fn \\ f \end{pmatrix} = \begin{pmatrix} \frac{nx}{f} \\ \frac{ny}{f} \\ f \\ 1 \end{pmatrix}$$

x en y in achtervlak geschaald met $\frac{n}{f}$

Homogene perspectief transformatie $\mathbf{M_p}$

$$\mathbf{M_p} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

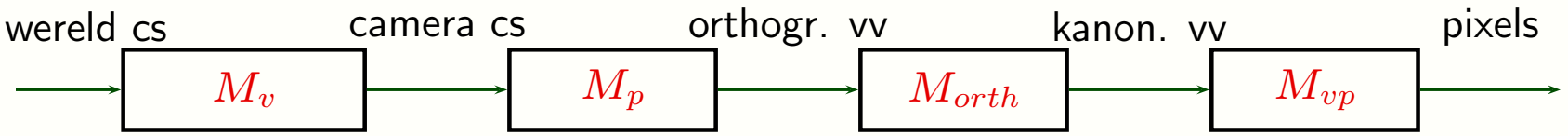
Homogene perspectief transformatie $\mathbf{M_p}$ zet **perspectief view volume** om in **orthografisch view volume**

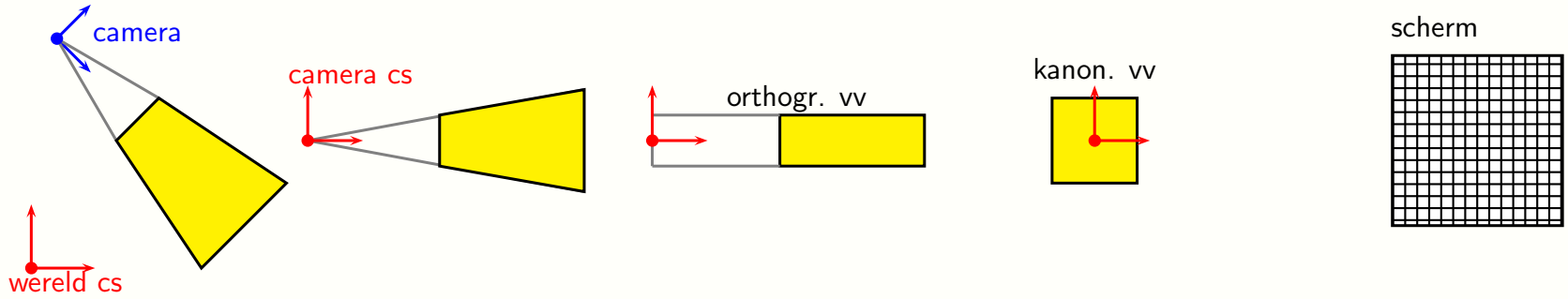
Wereldcoördinaten $\xrightarrow{M}$ schermcoördinaten

```
bereken MO
bereken Mv
bereken Mp
M = MO Mp Mv
for elk lijnsegment (a, b) doe
    p = M a
    q = M b
    deel homogene coordinaten van p en q door w
    draw(p, q)
```

wereld $\xrightarrow{\mathbf{M_v}}$ camera $\xrightarrow{\mathbf{M_p}}$ orthogr.view v $\xrightarrow{\mathbf{M_O}}$ scherm

Grafische pijplijn



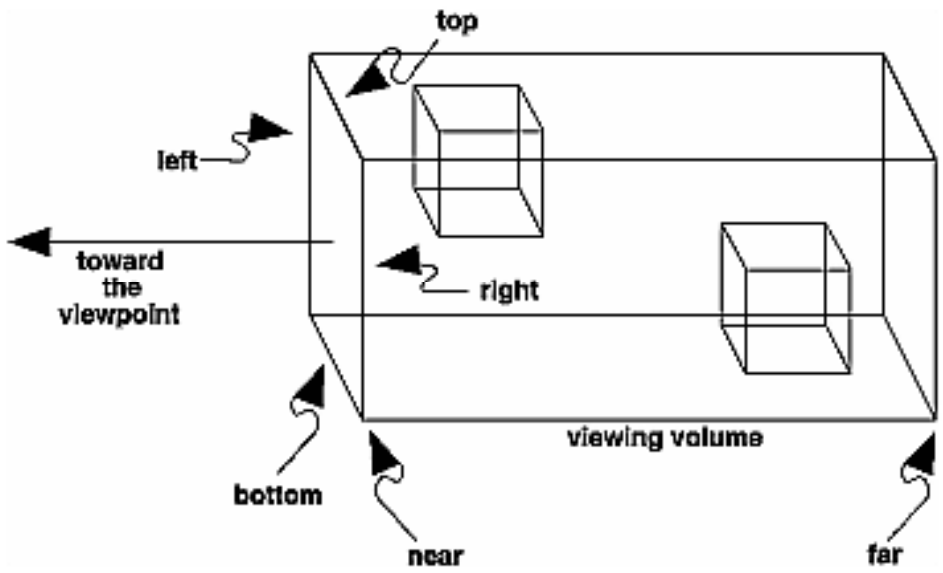


Andere aspecten van de grafische pijplijn

We kunnen nu punten en lijnen tekenen

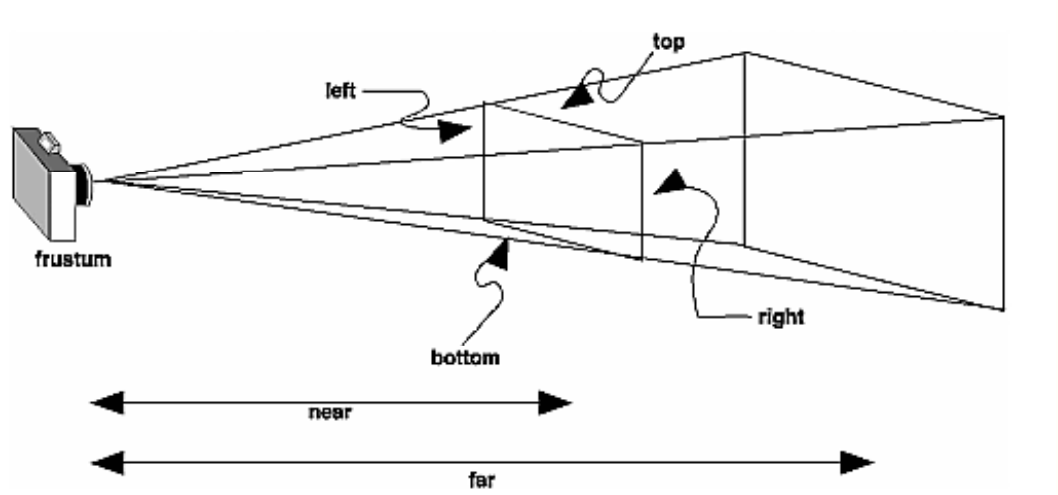
- Driehoeken die gedeeltelijk buiten view frustrum liggen moeten geclipped worden
- Overblijvende driehoeken moeten geprojecteerd worden als ze front facing zijn
- Geprojecteerde driehoeken moeten ingekleurd en/of textured worden

Orthografische projectie in OpenGL ($\mathrm{M}_{\mathrm{orth}}$)




```
 glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho(left, right, bottom, top, near, far);
```

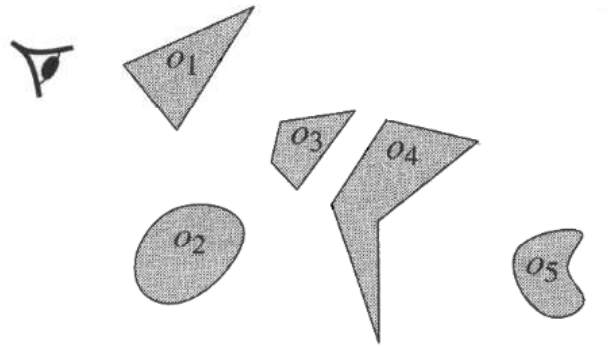
Perspectiefprojectie in OpenGL ($\mathbf{M_{orth}M_p}$)




```
 glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(left, right, bottom, top, near, far); of
gluPerspective(fovy, aspect, near, far);
```

Visbility problem

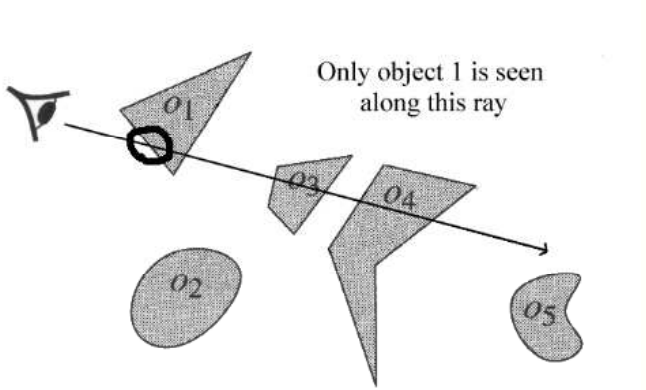
- we hebben verzameling objecten
- we hebben "oog" in de ruimte en kijken in bepaalde richting



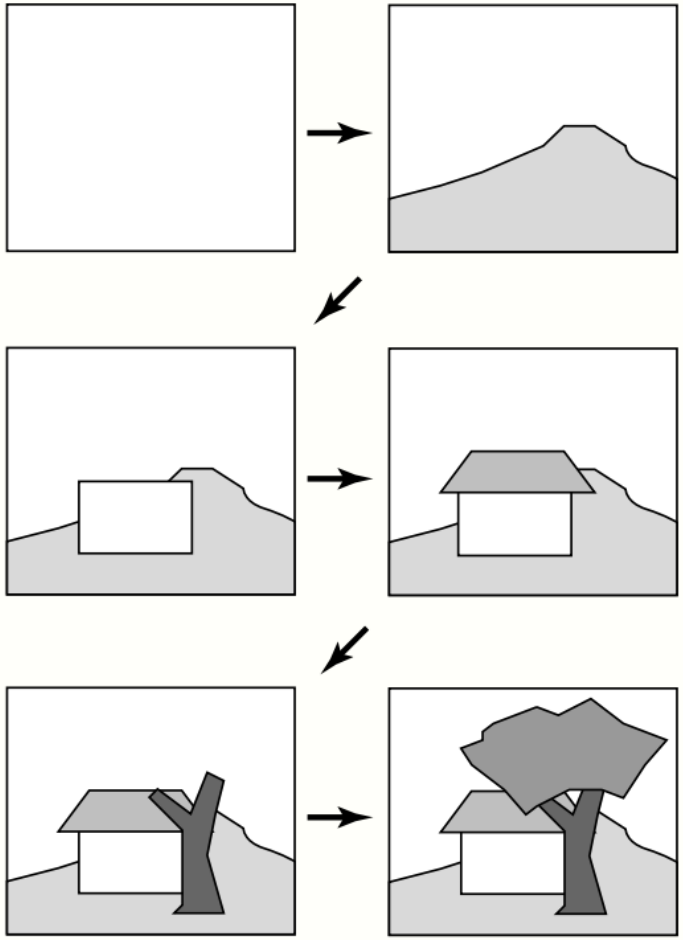
Tekenen van zichtbare objecten

- we willen beeld genereren dat oog zou zien, gegeven de objecten in de ruimte

- hoe tekenen we het correcte object op iedere pixel?



Painter's algoritme: van back-to-front



BSP tree algoritme

BSP tree algoritme is voorbeeld van painter's algoritme met polygonen die andere polygoonvlakken niet **kruisen**

Als polygonen wel snijden, doe **preprocessing** stap

1. Constructie BSP boom

2. Gebruik BSP boom bij tekenen

Een geconstrueerde BSP boom geeft ons correcte volgorde van polygonen voor elk viewpunt

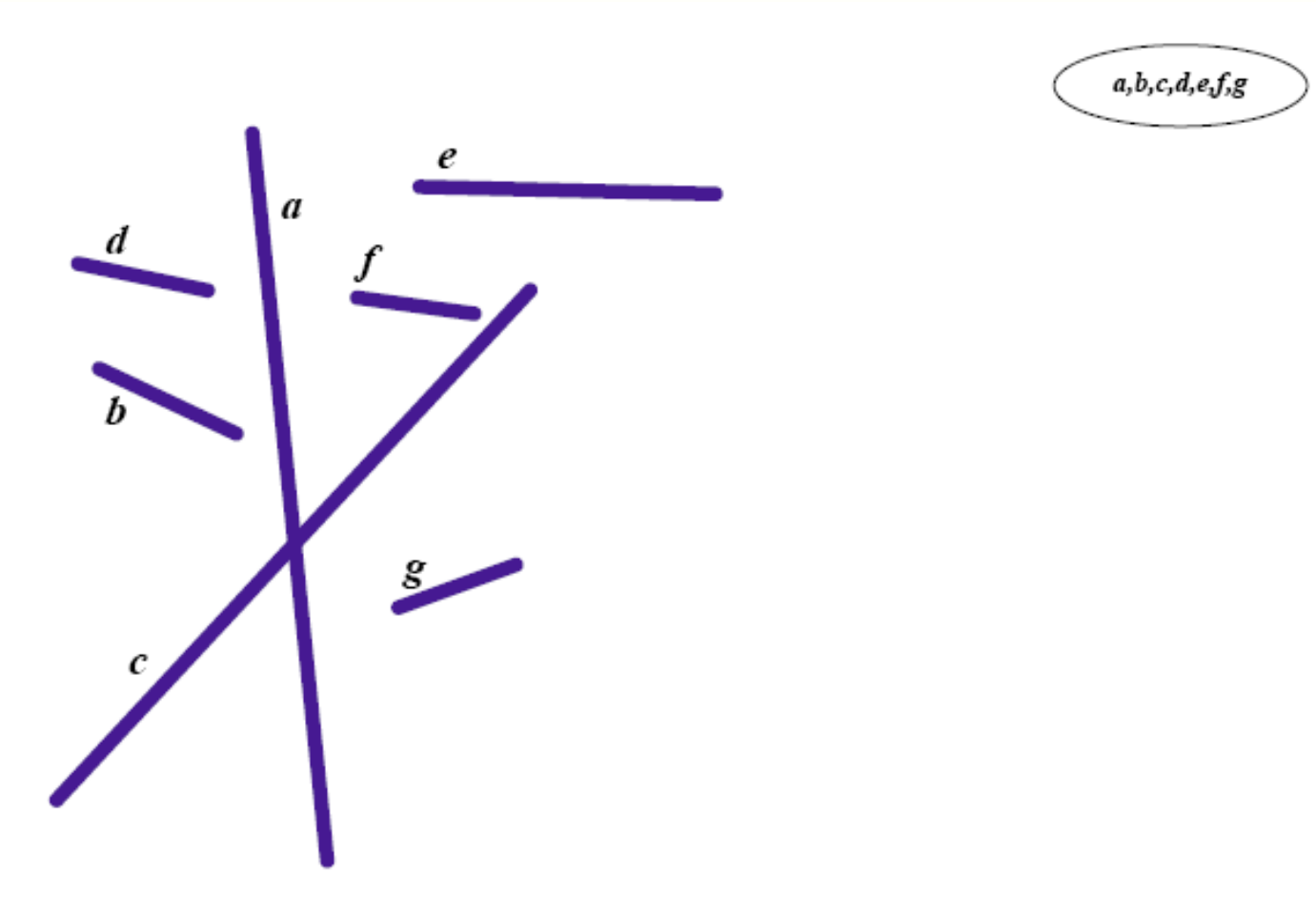
Bouwen van goede BSP boom

- Onhandige keuze van n polygonen is $\mathcal{O}(n^3)$ vanwege splitsen
- Er zijn algoritmen die opdeling vinden van orde $\mathcal{O}(n^2)$
  - Bijvoorbeeld, probeer alle overgebleven polygonen en kies die met minste splitsingen
  - Minder splitsingen $\rightarrow$ grotere polygonen $\rightarrow$ efficienter
- Ook gebalanseerde boom

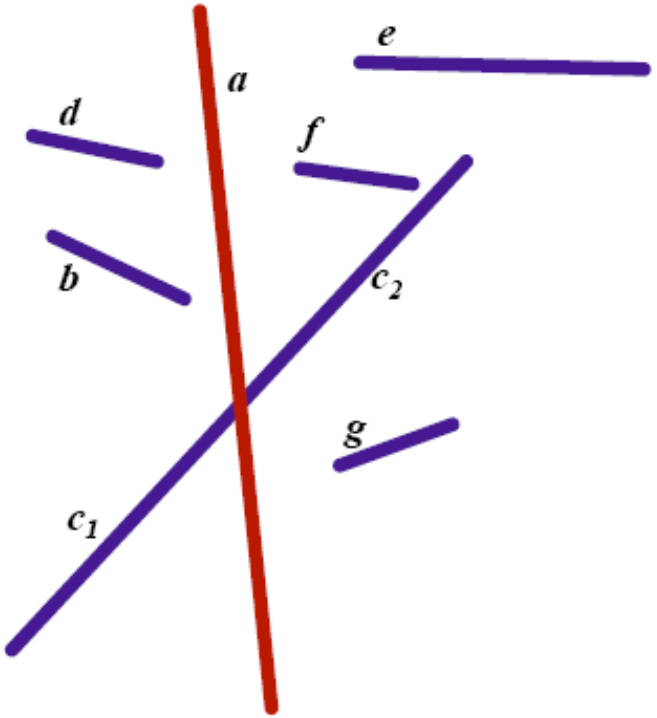
Painter's algoritme met BSP bomen

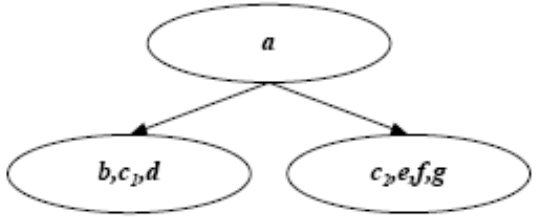
- Bouw de boom
  - dan moeten polygonen gesplitst worden
  - langzaam, maar eenmalig
- Correcte traversal tekent polygonen in back-to-front of front-to-back volgorde voor elk gezichtspunt
  - Volgorde is afhankelijk van view
  - Van te voren boom berekenen
  - Tekenen polygonen bij traversal
- Niet voor veranderende scenes

Constructie BSP tree (1)

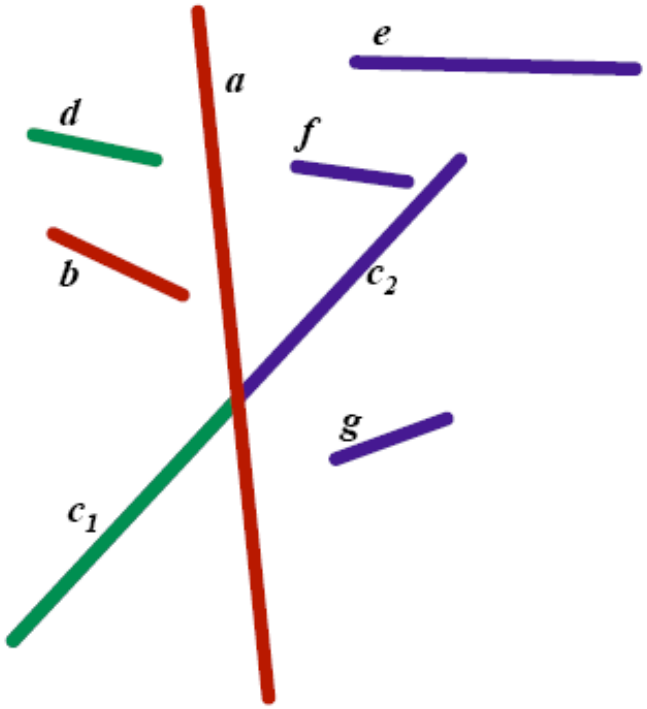


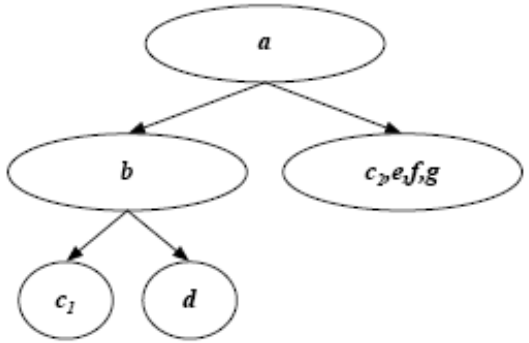
Constructie BSP tree (2)



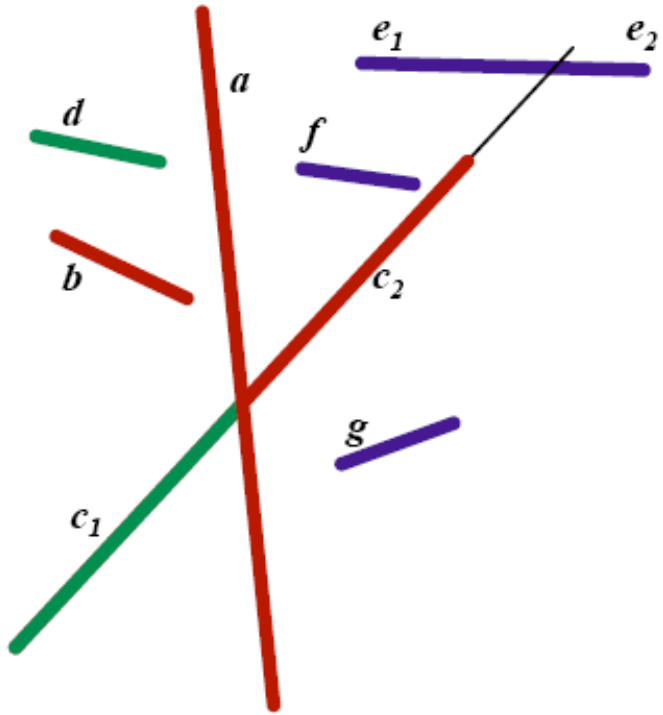


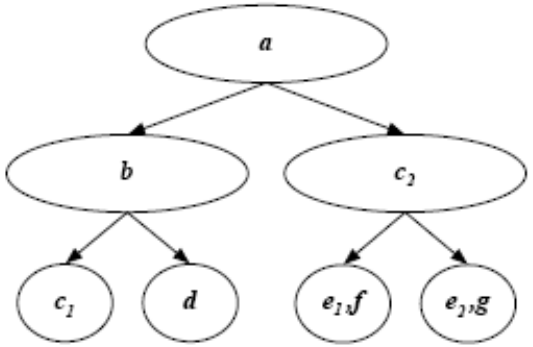
Constructie BSP tree (3)



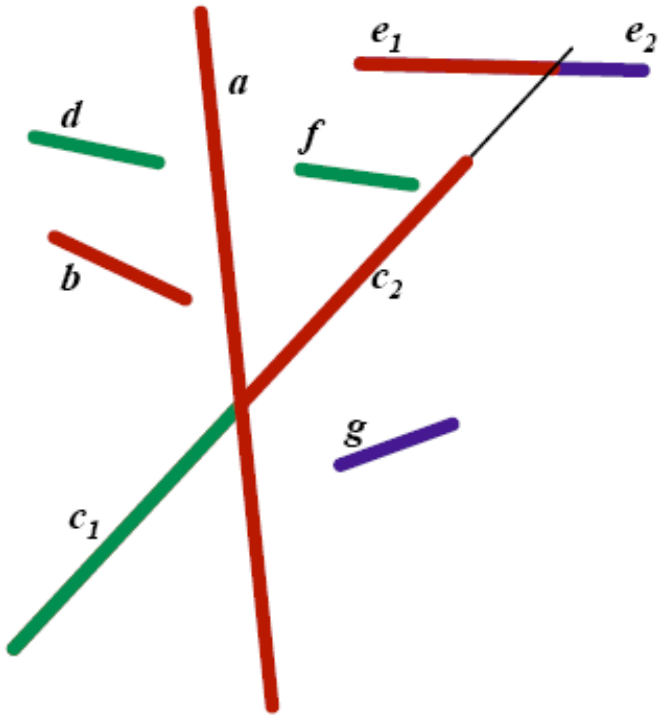


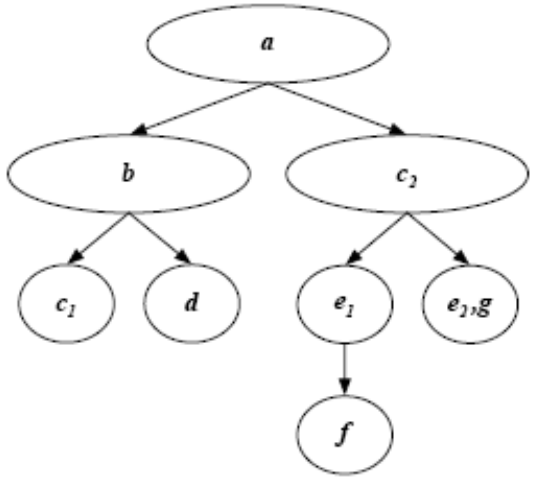
Constructie BSP tree (4)



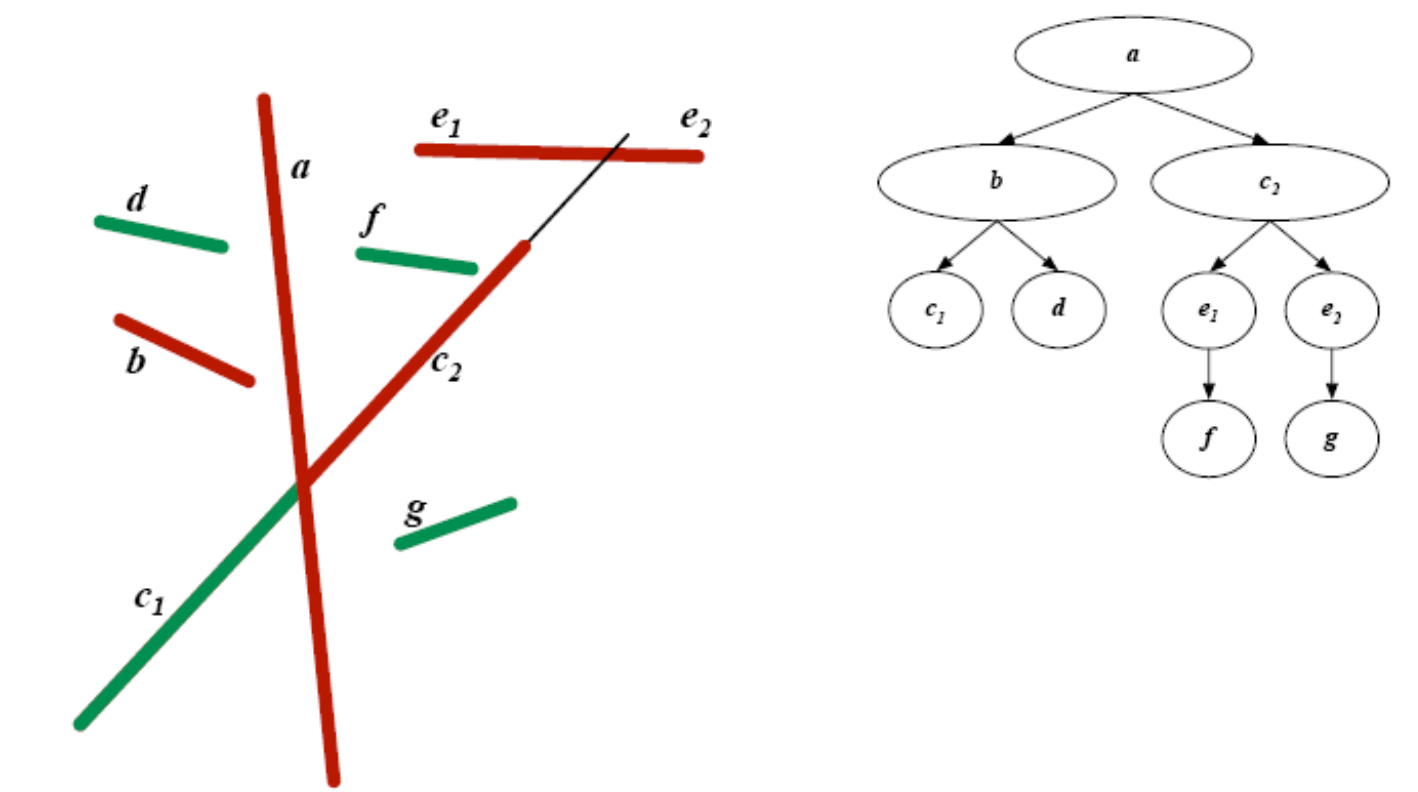


Constructie BSP tree (5)

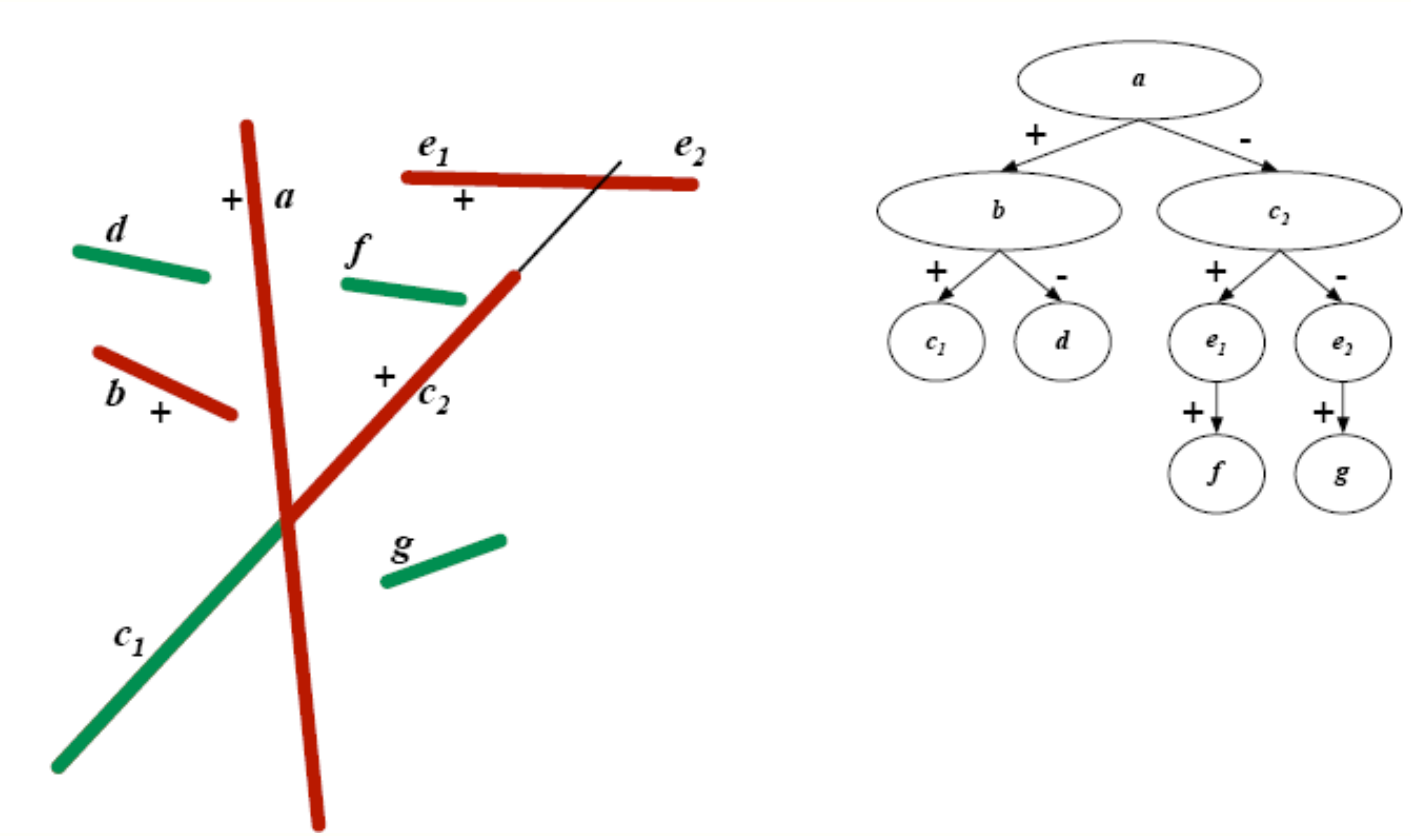




Constructie BSP tree (6)



Constructie BSP tree (7)



Visibility traversal

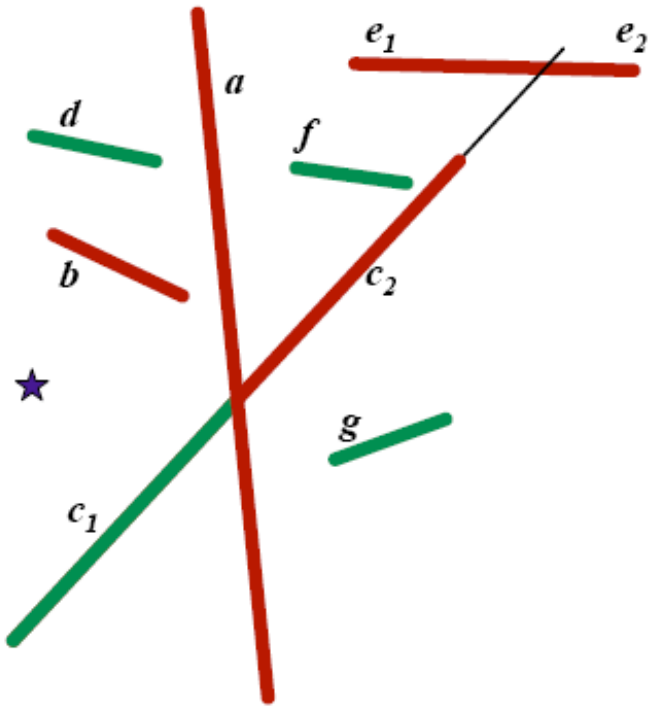
Variatie op **inorder** traversal:

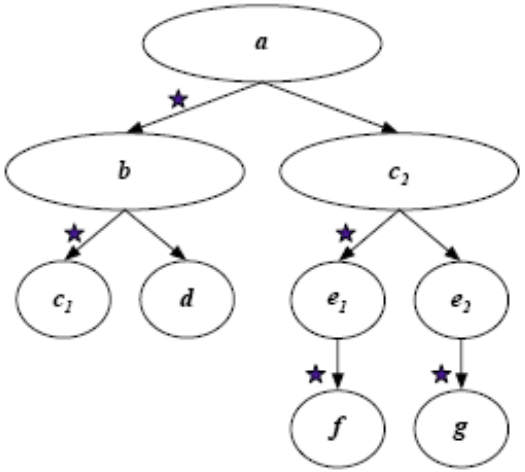
- kind aan zelfde kant als viewpunt
- doe root
- kind aan andere kant

Kies eerste kind gebaseerd op locatie van viewpunt

Eerste kind moet aan zelfde kant van sub-tree wortel liggen als viewpunt

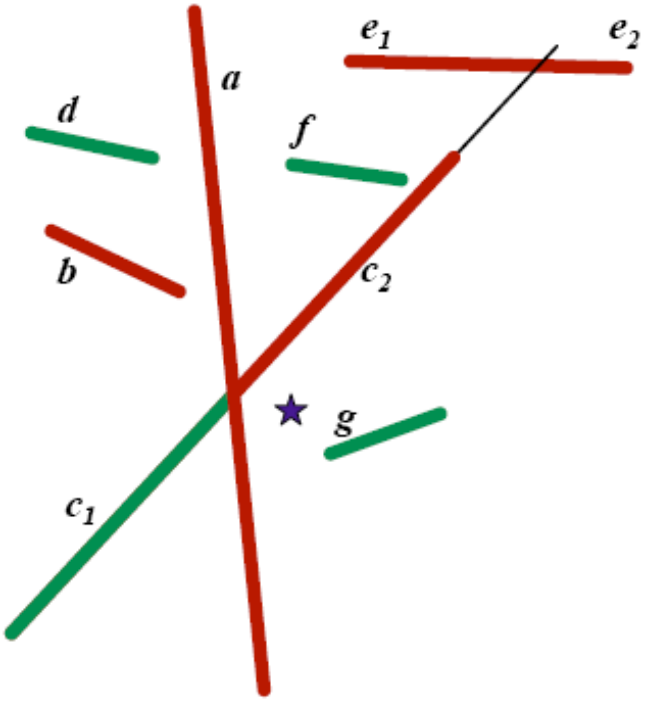
Visbility traversal BSP tree (1)

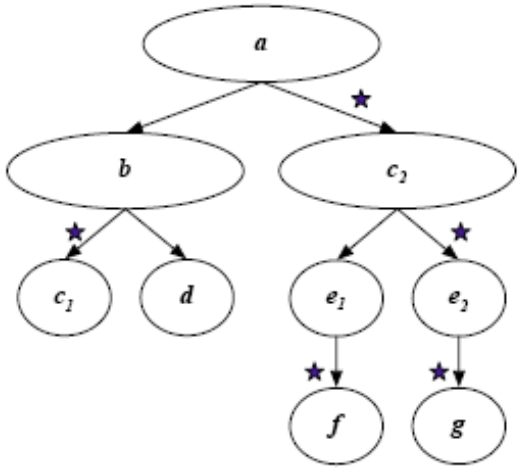




$c_1{:}b{:}d{:}a{:}f{:}e_1{:}c_2{:}g{:}e_2$

Visbility traversal BSP tree (2)





$g{:}e_2{:}c_2{:}f{:}e_1{:}a{:}c_1{:}b{:}d$

Bézier krommen

Bézier kromme wordt bepaald door

- orde
- aantal controlpunten

Een kromme van graad n heeft $n+1$ controlpunten P_0, P_1, ..., P_n

- Lineaire kromme (graad $= 1$) heeft 2 controlpunten
- Kwadratische kromme (graad $= 2$) heeft 3 controlpunten
- Kubische kromme (graad $= 3$) heeft 4 controlpunten

Lineaire Bézier kromme

Gegeven punten P_0 en P_1

Lineaire Bézier kromme is rechte lijn tussen P_0 en P_1

$$B(t) = P_0 + t(P_1 - P_0) = (1-t)P_0 + tP_1 \ , \qquad t \in [0,1]$$

Opmerking : som van gewichten is 1

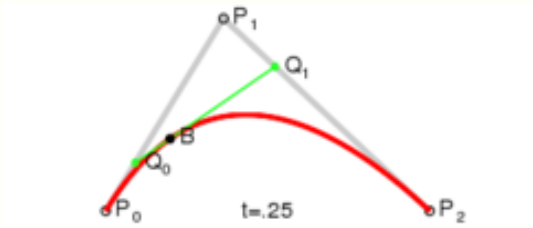
Kwadratische Bézier kromme

Gegeven punten P_0, P_1 en P_2

Kwadratische Bézier kromme is parabool door P_0 en P_2 (begin- en eindpunt)

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2 \ , \qquad t \in [0,1]$$

Opmerking : som van gewichten is 1



Kwadratische Bézier kromme

Wat doet P_1?

P_1 bepaalt de raaklijn in begin- en eindpunt

$$B(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2 \ , \qquad t \in [0,1]$$

$$B'(t) = -2(1-t)P_0 + 2(1-2t)P_1 + 2tP_2$$

$B'(0) = -2P_0 + 2P_1 = 2(P_1 - P_0)$ raaklijn in P_0

$B'(1) = -2P_1 + 2P_2 = 2(P_2 - P_1)$ raaklijn in P_2

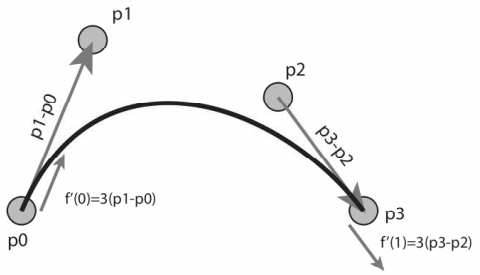
Derdegraads Bézier kromme

Gegeven punten P_0, P_1, P_2 en P_3

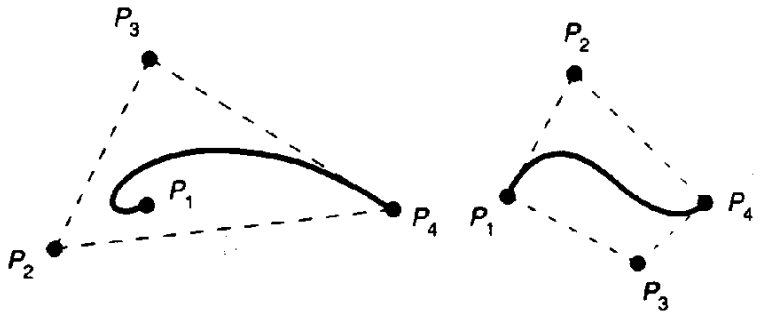
Derdegraads Bézier kromme is 3degraads kromme door P_0 en P_3

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \ , \qquad t \in [0,1]$$

Opmerking : som van gewichten is 1



Convexe omhulsel eigenschap



Kromme binnen `convexe omhulsel` (stippellijnen)

Bernstein basis polynomen

Vier weegfuncties (blending functions) van derdegraads Bézier kromme heten **Bernstein basis polynomen**

$$\begin{aligned} b_{0,3} = (1-t)^3 &= 1 - 3t + 3t^2 - t^3 \\ b_{1,3} = 3t(1-t)^2 &= 3t - 6t^2 + 3t^3 \\ b_{2,3} = 3t^2(1-t) &= 3t^2 - 3t^3 \\ b_{3,3} = t^3 &= t^3 \end{aligned}$$

Som weegfuncties 1 en niet-negatief $\Rightarrow$ **convex-hull** eigenschap (denk aan barycentrische coördinaten: elk punt op kromme ligt binnen vierhoek)

Bézier kromme door meerdere punten?

Driehoek van Pascal: binomiaalcoëfficiënten $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

$$
\begin{array}{ccccccccccc}
 & & & & 1 & & & & & & (n=0) \\
 & & & 1 & & 1 & & & & & (n=1) \\
 & & 1 & & 2 & & 1 & & & & (n=2) \\
 & 1 & & 3 & & 3 & & 1 & & & (n=3) \\
1 & & 4 & & 6 & & 4 & & 1 & & (n=4) \\
\binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4} & &
\end{array}
$$

$$\binom{n}{0} \quad \binom{n}{1} \quad \binom{n}{2} \quad \binom{n}{3} \quad \cdots \quad \binom{n}{n} \text{ rij } n$$

Binomiaalformule: $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$

$$1 = 1^n = ((1-t)+t)^n = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k$$

Voorbeeld $n = 3$

$(x+y)^3 = \binom{3}{0}x^3 + \binom{3}{1}x^2 y + \binom{3}{2}xy^2 + \binom{3}{3}y^3 = x^3 + 3x^2 y + 3xy^2 + y^3$

$1 = (1-t)^3 + 3t(1-t)^2 + 3t^2(1-t) + t^3$ (som weegfuncties 1)

$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3$

Dus **coëfficiënten Bézier kromme:** $\binom{n}{k}(1-t)^{n-k}t^k$

n^egraads Bézier kromme

n^egraads Bézier kromme door $n+1$ punten P_0 t/m P_n:

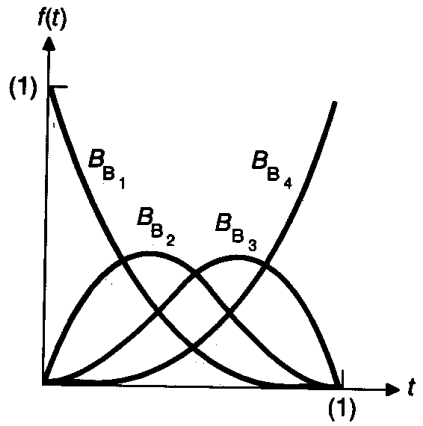
$$B(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k P_k$$

$n = 3$ en Bernstein basis polynomen

$$B(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k P_k$$

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$

Coëfficiënten: $\binom{3}{0}$ $\binom{3}{1}$ $\binom{3}{2}$ $\binom{3}{3}$



$n = 4$

$$B(t) = \sum_{k=0}^{n} \binom{n}{k} (1-t)^{n-k} t^k P_k$$

4^egraads Bézier kromme door 5 punten P_1 t/m P_5:

$$B(t) = (1-t)^4 P_0 + 4t(1-t)^3 P_1 + 6t^2(1-t)^2 P_2 + 4t^3(1-t) P_3 + t^4 P_4$$

$B(0) = P_0$, dus gaat door P_0
$B(1) = P_4$, dus gaat door P_4