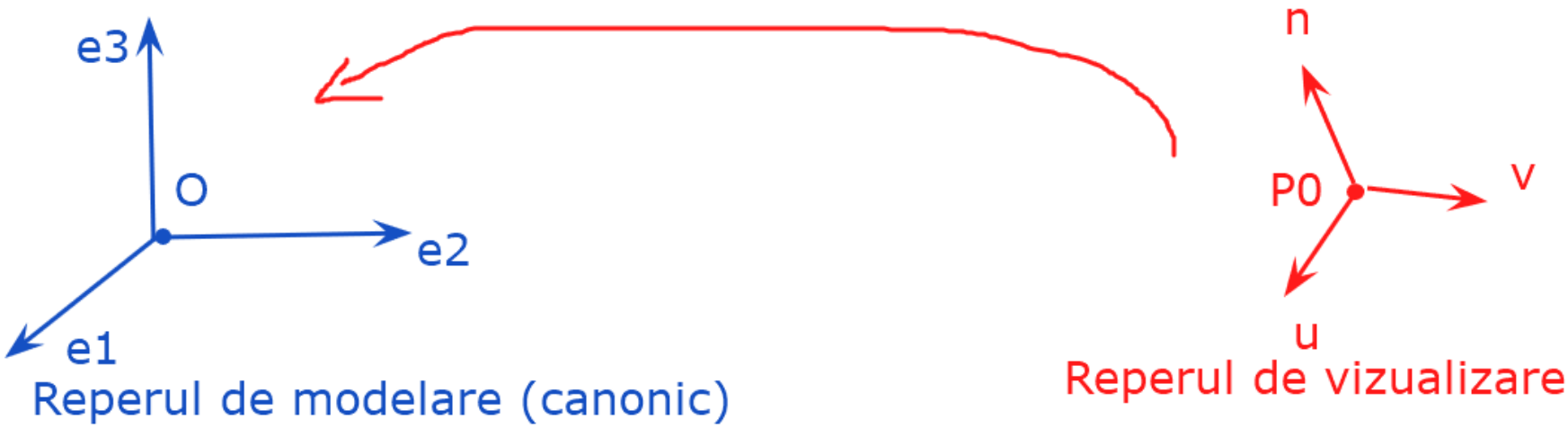
Transformări (IV). Proiecții

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Schimbarea reperului ca transformare

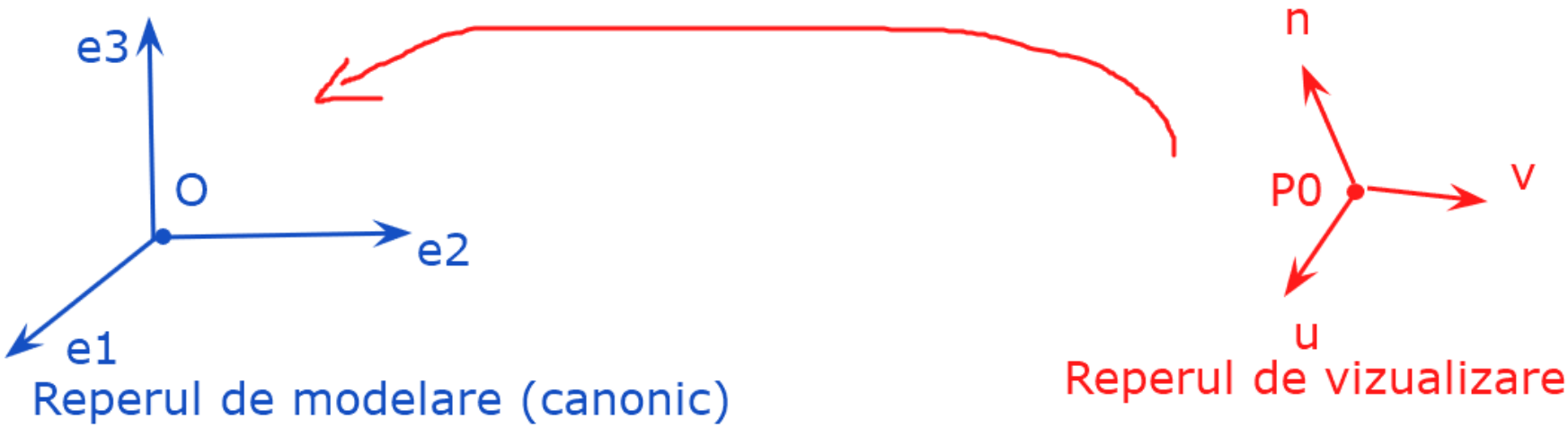
Schimbarea de reper $\leftrightarrow$ Efectuarea unei transformări



"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Schimbarea reperului ca transformare

Schimbarea de reper $\leftrightarrow$ Efectuarea unei transformări

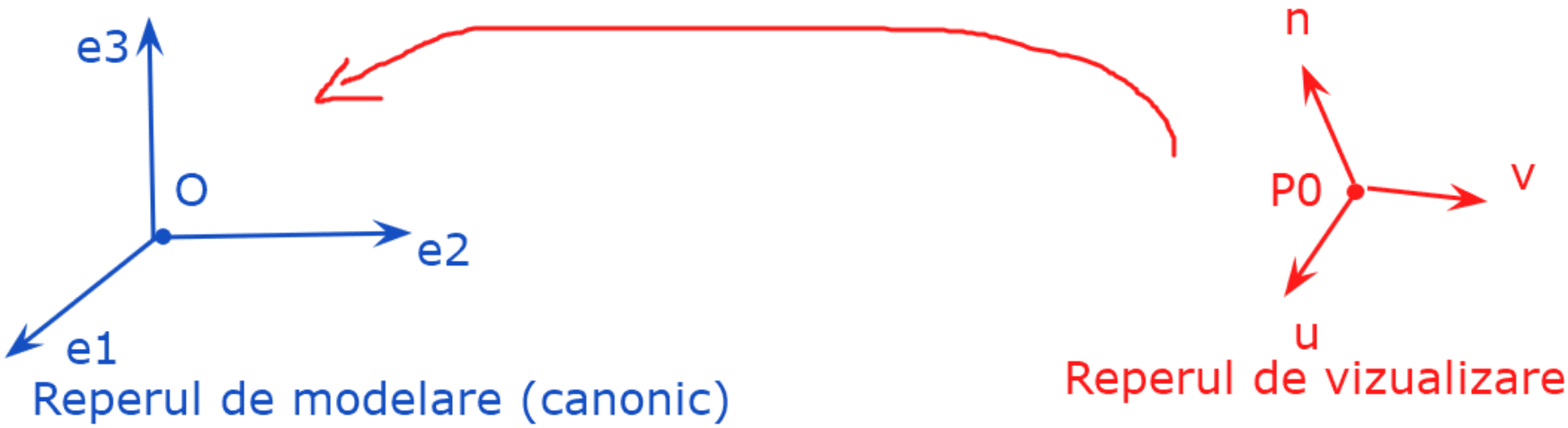


"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

Schimbarea reperului ca transformare

Schimbarea de reper $\leftrightarrow$ Efectuarea unei transformări



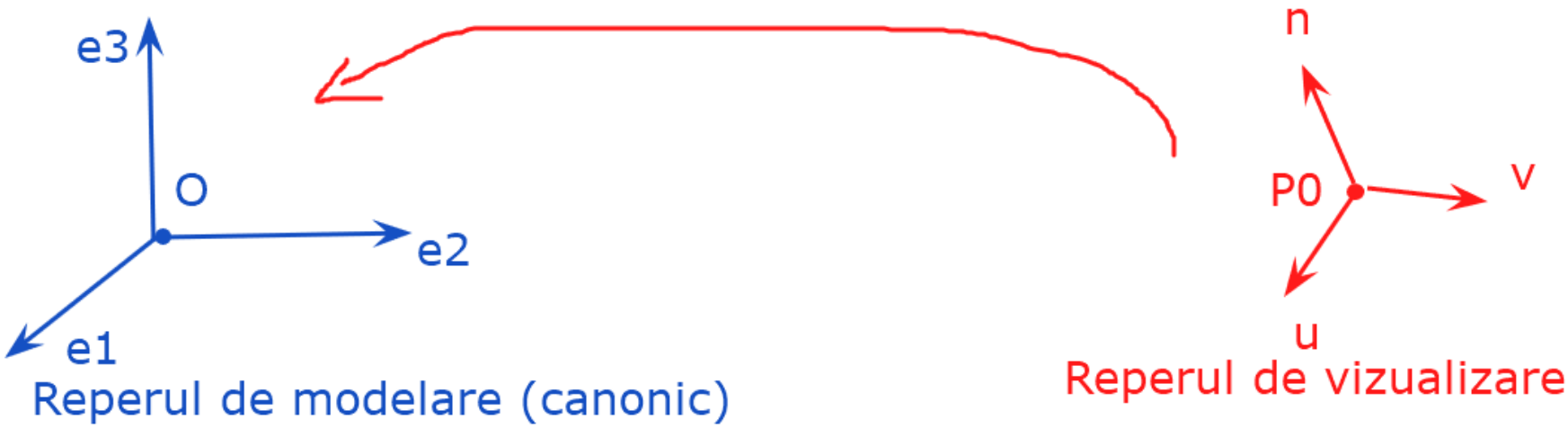
"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

- translatăm astfel încât P_0 să devină originea, adică aplicăm $\mathbf{T}_{(-x_0,-y_0,-z_0)}$

Schimbarea reperului ca transformare

Schimbarea de reper $\leftrightarrow$ Efectuarea unei transformări



"Aducem reperul de vizualizare ca să se suprapună peste reperul de modelare".

Descrierea transformărilor:

- translatăm astfel încât P_0 să devină originea, adică aplicăm $\mathbf{T}_{(-x_0,-y_0,-z_0)}$
- aplicăm o rotație 3D $\mathbf{R}$ astfel încât reperul ortonormat $(\mathbf{u}, \mathbf{v}, \mathbf{n})$ să se suprapună cu reperul ortonormat (e_1, e_2, e_3) (reperul canonic) - un reper este ortonormat dacă vectorii sunt perpendiculari 2 câte 2 și de normă 1.

- Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.

- Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.
- Despre aplicarea proiecțiilor:

- Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.
- Despre aplicarea proiecțiilor:
  - dacă nu a fost efectuată nicio transformare de vizualizare, proiecția este aplicată în raport cu reperul de modelare, fiind decupat pătratul "standard" $[-1, 1] \times [-1, 1]$,

- Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.
- Despre aplicarea proiecțiilor:
  - dacă nu a fost efectuată nicio transformare de vizualizare, proiecția este aplicată în raport cu reperul de modelare, fiind decupat pătratul "standard" $[-1, 1] \times [-1, 1]$,
  - dacă a fost efectuată o transformare de vizualizare (observatorul a fost "adus" în origine, axele au fost "aliniate", etc.): din punctul de vedere al logicii imaginii, decuparea / proiecția sunt realizate în raport cu observatorul și reperul de vizualizare. Altfel spus, proiecția este aplicată după transformarea de vizualizare.

- Rolul transformărilor de proiecție: de a permite reprezentarea unei lumi 3D, teoretic nemărginite, pe un monitor 2D mărginit.
- Despre aplicarea proiecțiilor:
  - dacă nu a fost efectuată nicio transformare de vizualizare, proiecția este aplicată în raport cu reperul de modelare, fiind decupat pătratul "standard" $[-1, 1] \times [-1, 1]$,
  - dacă a fost efectuată o transformare de vizualizare (observatorul a fost "adus" în origine, axele au fost "aliniate", etc.): din punctul de vedere al logicii imaginii, decuparea / proiecția sunt realizate în raport cu observatorul și reperul de vizualizare. Altfel spus, proiecția este aplicată după transformarea de vizualizare.
- O proiecție este o transformare care implică (i) decuparea, (ii) proiecția propriu-zisă, fiind necesară o matrice 4×4 adecvată. Din punct de vedere al implementării: dacă `matrVizualizare` este matricea de vizualizare (dată de `glm::lookAt()`) și `matrProiectie` este matricea de proiecție, atunci în codul sursă trebuie să apară `matrProiectie * matrVizualizare`.

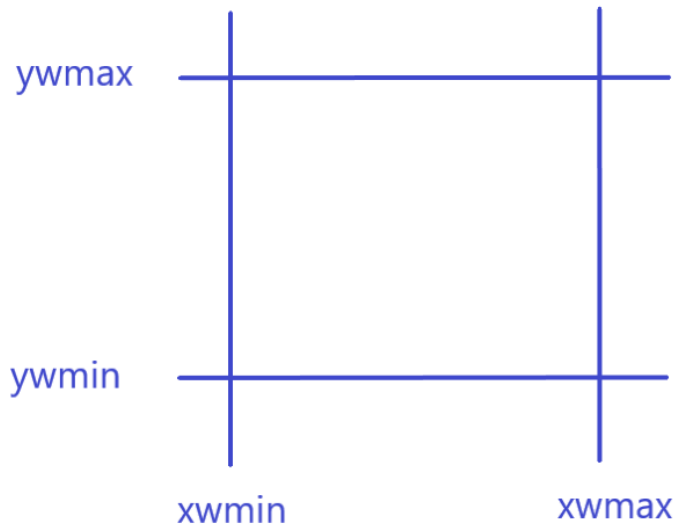
Cazul 2D

- `glm::ortho (xwmin, xwmax, ywmin, ywmax);`

Cazul 2D

- `glm::ortho (xwmin, xwmax, ywmin, ywmax);`
- Efectul: este decupat un dreptunghi $\mathcal{D}$ din planul orizontal Oxy (se presupune că nu au fost aplicate alte transformări). Dreptunghiul $\mathcal{D}$ are laturile paralele cu axele de coordonate, fiind delimitat de dreptele

$$x = xwmin,\ x = xwmax,\ y = ywmin,\ y = wymax.$$



Apoi este realizată o transformare a dreptunghiului $\mathcal{D}$ în pătratul "standard" $[-1,1] \times [-1,1]$. Matricea 4×4 asociată transformării poate fi determinată explicit.

Proiecții ortogonale 2D - exemple

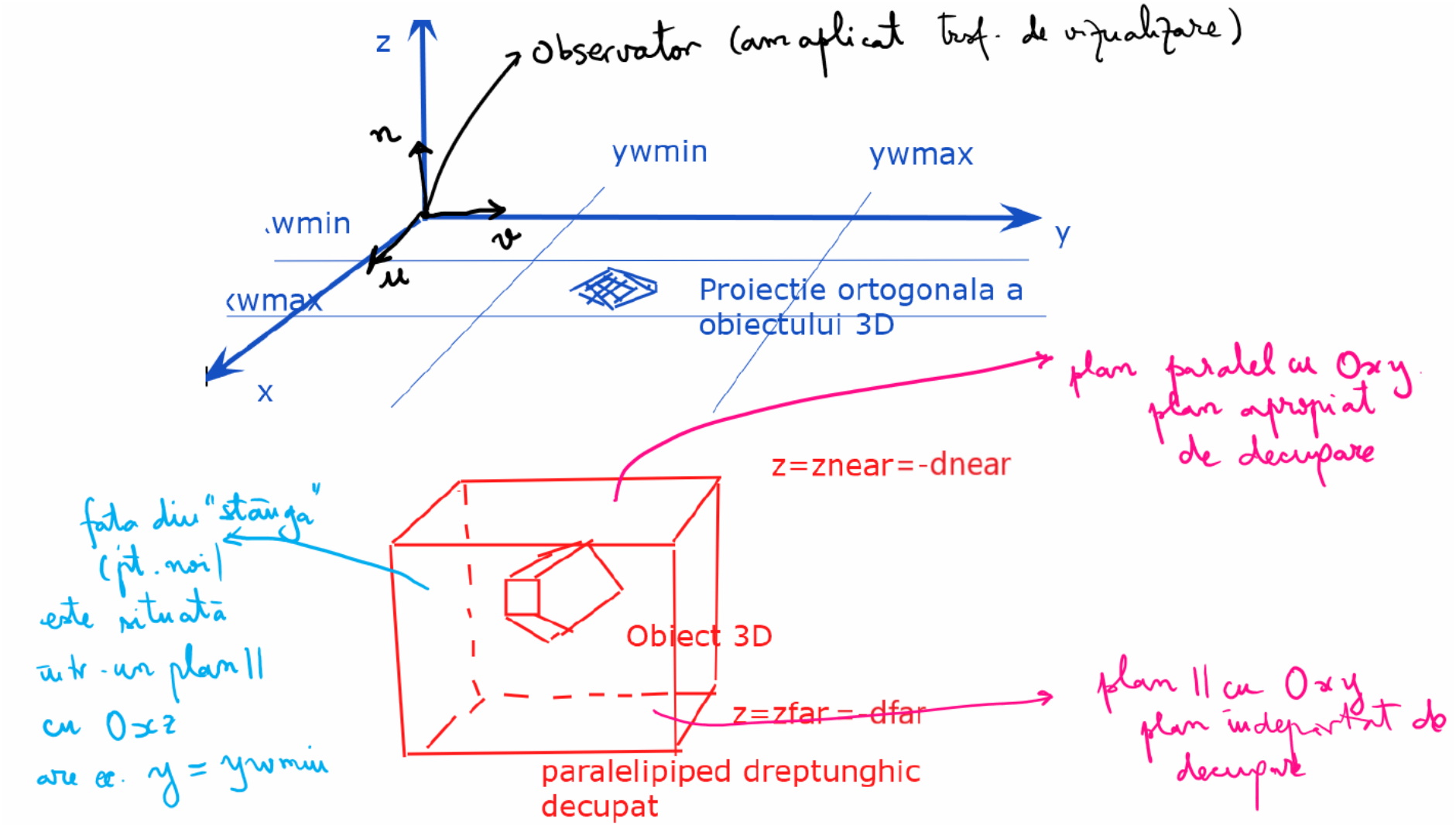
- Care este aria dreptunghiului decupat dacă se apelează funcția `glm::ortho (a, b, c, d)`? ($a < b, c < d$).

Proiecții ortogonale 2D - exemple

- Care este aria dreptunghiului decupat dacă se apelează funcția `glm::ortho (a, b, c, d)`? ($a < b, c < d$).
- Ce diferențe sunt (din punctul de vedere al (i) dimensiunii scenei decupate, (ii) obiectelor - dimensiune, etc.) între apelarea funcției `glm::ortho (a, b, c, d)` și apelarea funcției `glm::ortho (2a, 2b, 2c, 2d)`? ($a < b, c < d$)

Cazul 3D - proiecții ortogonale

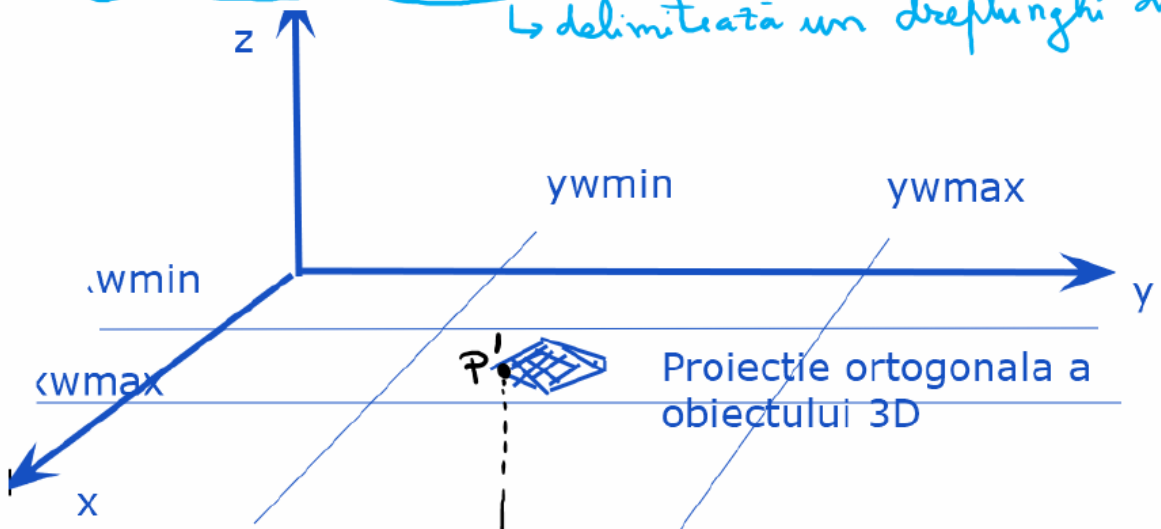
```
glm::ortho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```



Ce este o proiecție ortogonală?

```
glm::ortho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

↳ delimitează un dreptunghi din planul de vizualizare



P' = proiecția ortogonală a lui P (se duce o dreaptă care trece prin P și e ⊥ Oxy ...)

P' este și pr. ortogonală a lui P_1

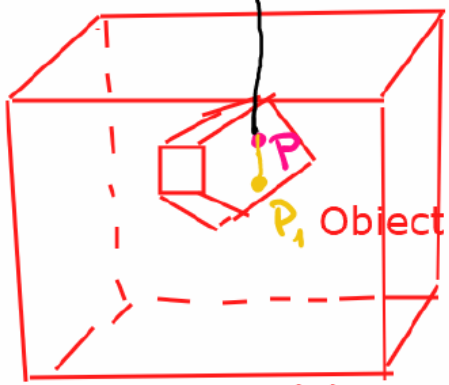
formula:

$(x_P, y_P, z_P) \mapsto$

$\mapsto (x_P, y_P, 0)$

$\equiv$

(x_P, y_P).



paralelipiped dreptunghic decupat

Cazul 3D - proiecții ortogonale

```
glm::ortho (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

Este decupat un **paralelipiped dreptunghic** delimitat de planele $x = xw_{\min}, x = xw_{\max}$; $y = yw_{\min}, y = yw_{\max}$, respectiv $z = z_{\text{near}}$, unde $z_{\text{near}} = -d_{\text{near}}$, $z = z_{\text{far}}$, unde $z_{\text{far}} = -d_{\text{far}}$. Valorile implicite sunt $-1.0, 1.0, -1.0, 1.0, -1.0, 1.0$ (valori normalizate).

Matricea 4×4 asociată este

$$\mathcal{M}_{\text{ortho}} = \begin{pmatrix} \frac{2}{xw_{\max} - xw_{\min}} & 0 & 0 & -\frac{xw_{\max} + xw_{\min}}{xw_{\max} - xw_{\min}} \\ 0 & \frac{2}{yw_{\max} - yw_{\min}} & 0 & -\frac{yw_{\max} + yw_{\min}}{yw_{\max} - yw_{\min}} \\ 0 & 0 & -\frac{2}{d_{\text{far}} - d_{\text{near}}} & -\frac{d_{\text{far}} + d_{\text{near}}}{d_{\text{far}} - d_{\text{near}}} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Matricea $\mathcal{M}_{\text{ortho}}$ are rolul de a transforma paralelipipedul dreptunghic decupat în paralelipipedul "standard" $[-1, 1] \times [-1, 1] \times [-1, 1]$, apoi, în mod implicit, sunt reținute primele două coordonate.

Proiecții ortogonale 3D - comentarii

- De discutat rolul elementelor care definesc paralelipipedul dreptunghic decupat.

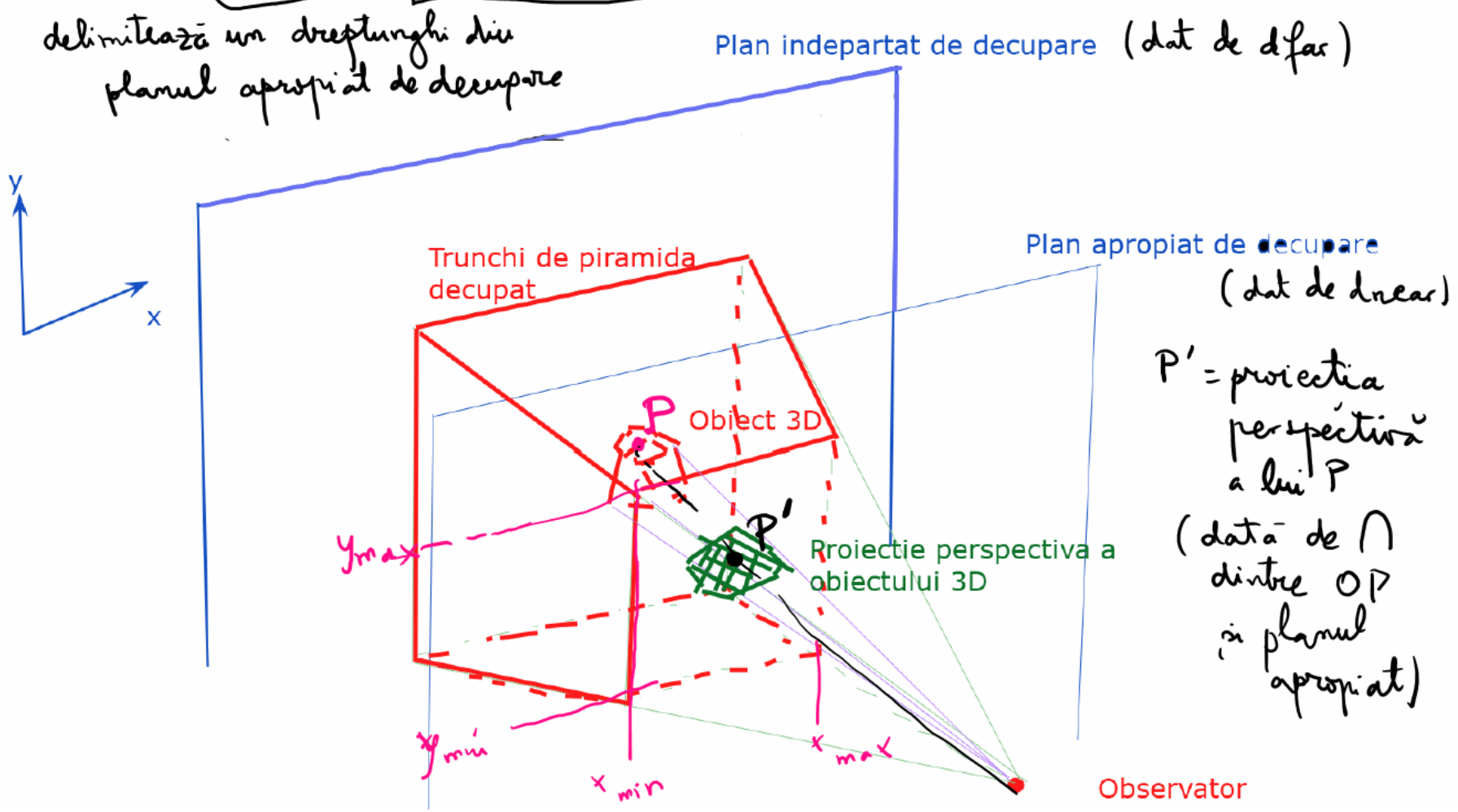
Proiecții ortogonale 3D - comentarii

- De discutat rolul elementelor care definesc paralelipipedul dreptunghic decupat.
- De testat pe codul sursă `07_01_desenare_cub.cpp` cum este realizată proiecția dacă modificăm diverși parametri ai funcției `glm::ortho();`. De urmărit: (i) cum este realizată decuparea; (ii) cum arată obiectul randat. De exemplu: ce se întâmplă dacă modificăm `dnear`?

Cazul 3D - proiecții perspective

```
glm::frustum(xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

delimitează un dreptunghi din planul apropiat de decupare



P' = proiectia perspectivă a lui P (dată de ∩ dintre OP și planul apropiat)

Cazul 3D - proiecții perspective

```
glm::frustum (xwmin, xwmax, ywmin, ywmax, dnear, dfar);
```

Este decupat un **trunchi de piramidă** având vârful în origine (poziția observatorului). Bazele sunt date de planele $z = z_{\text{near}}$, unde $z_{\text{near}} = -d_{\text{near}}$, $z = z_{\text{far}}$, unde $z_{\text{far}} = -d_{\text{far}}$

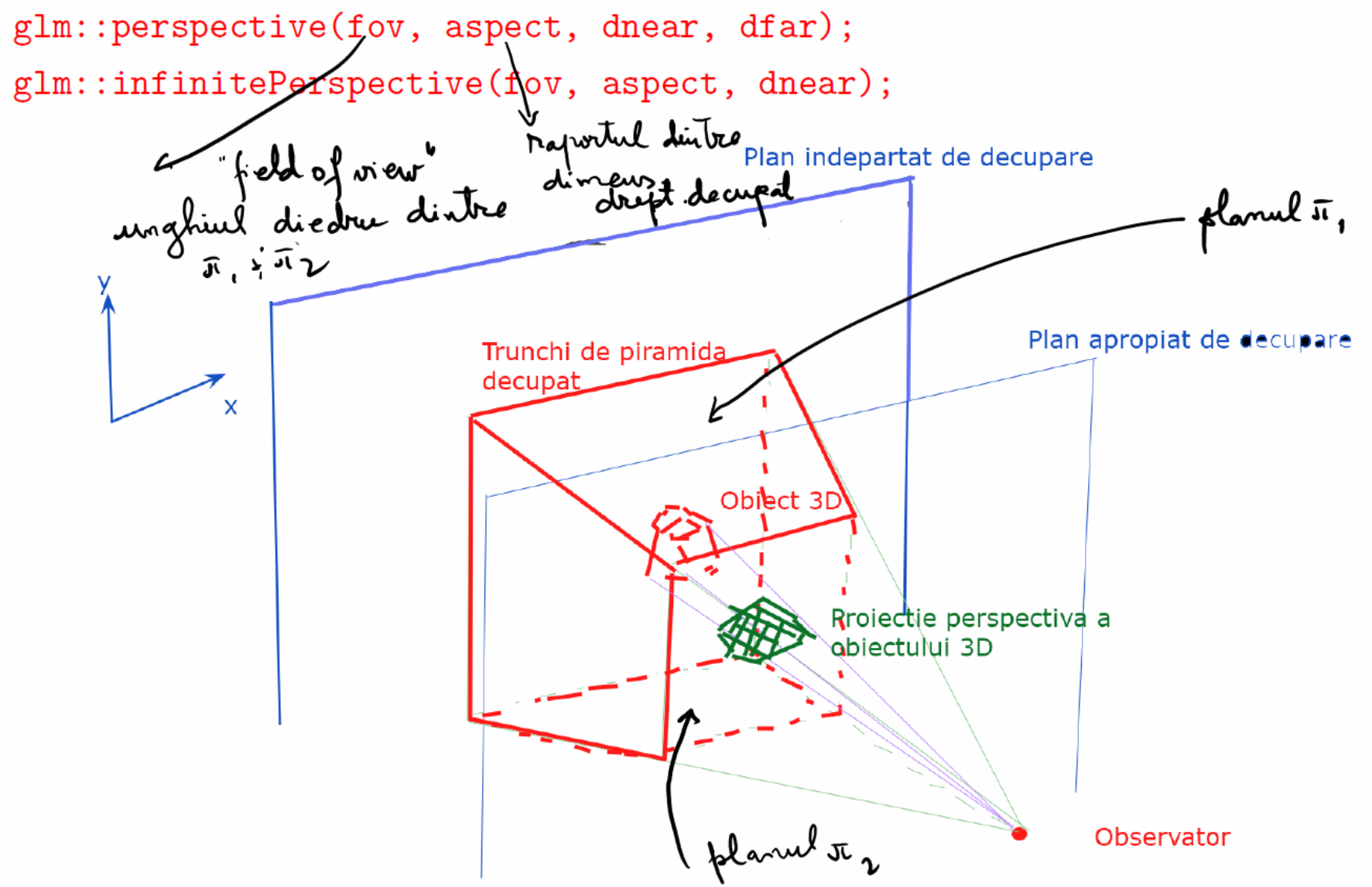
Important: se delimitează dreptunghiul din planul apropiat, având ecuațiile dreptelor de forma $z = z_{\text{near}}, x = xw_{\text{min}}$, etc., **apoi** se determină dreptunghiul din planul îndepărtat și trunchiul de piramidă.
Matricea 4×4 asociată este

$$\mathcal{M}_{\text{frustum}} = \begin{pmatrix} \frac{2d_{\text{near}}}{xw_{\text{max}} - xw_{\text{min}}} & 0 & \frac{xw_{\text{max}} + xw_{\text{min}}}{xw_{\text{max}} - xw_{\text{min}}} & 0 \\ 0 & \frac{2d_{\text{near}}}{yw_{\text{max}} - yw_{\text{min}}} & \frac{yw_{\text{max}} + yw_{\text{min}}}{yw_{\text{max}} - yw_{\text{min}}} & 0 \\ 0 & 0 & -\frac{d_{\text{far}} + d_{\text{near}}}{d_{\text{far}} - d_{\text{near}}} & -\frac{2d_{\text{near}}\, d_{\text{far}}}{d_{\text{far}} - d_{\text{near}}} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Matricea $\mathcal{M}_{\text{frustum}}$ are rolul de a transforma trunchiul de piramidă decupat în paralelipipedul "standard" $[-1, 1] \times [-1, 1] \times [-1, 1]$, apoi, în mod implicit, sunt reținute primele două coordonate.

Cazul 3D - proiecții perspective

```
glm::perspective(fov, aspect, dnear, dfar);
glm::infinitePerspective(fov, aspect, dnear);
```



Cazul 3D - proiecții perspective

```
glm::perspective(fov, aspect, dnear, dfar);
```

`glm::perspective()` - este decupat un **trunchi de piramidă** decupat dintr-o **piramidă cu baza dreptunghi în care înălțimea dusă din vârful piramidei inițiale (observator) cade în centrul dreptunghiului.** "Deschiderea" piramidei este dată de `fov`, iar raportul lungimilor laturilor este dat de `aspect`.

Important: se delimitează dreptunghiul din planul apropiat, **apoi** se determină dreptunghiul din planul îndepărtat și trunchiul de piramidă.

Matricea 4×4 asociată este

$$\mathcal{M}_{\text{perspective}} = \begin{pmatrix} \frac{\text{ctg}(\frac{\text{fov}}{2})}{\text{aspect}} & 0 & 0 & 0 \\ 0 & \text{ctg}(\frac{\text{fov}}{2}) & 0 & 0 \\ 0 & 0 & -\frac{d_{\text{far}} + d_{\text{near}}}{d_{\text{far}} - d_{\text{near}}} & -\frac{2 d_{\text{near}} d_{\text{far}}}{d_{\text{far}} - d_{\text{near}}} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Matricea $\mathcal{M}_{\text{perspective}}$ are rolul de a transforma trunchiul de piramidă decupat în paralelipipedul "standard" $[-1, 1] \times [-1, 1] \times [-1, 1]$, apoi, în mod implicit, sunt reținute primele două coordonate.

Proiecții perspective - comentarii

- De discutat rolul elementelor care definesc trunchiul de piramidă decupat. De comentat diferența dintre `glm::frustum( );` - se poate decupa un trunchi de piramidă arbitrar și `glm::perspective( );` - se poate decupa un trunchi de piramidă care provine dintr-o piramidă având baza un dreptunghi, iar piciorul înălțimii piramidei (dusă din vârf) coincide cu centrul bazei. În cazul funcției `glm::perspective();` sunt considerați ca parametri `fov`=*field of view*, un unghi între plane și `aspect`=raportul dintre lungimile laturilor dreptunghiului decupat.

Proiecții perspective - comentarii

- De discutat rolul elementelor care definesc trunchiul de piramidă decupat. De comentat diferența dintre `glm::frustum( );` - se poate decupa un trunchi de piramidă arbitrar și `glm::perspective( );` - se poate decupa un trunchi de piramidă care provine dintr-o piramidă având baza un dreptunghi, iar piciorul înălțimii piramidei (dusă din vârf) coincide cu centrul bazei. În cazul funcției `glm::perspective();` sunt considerați ca parametri `fov`=*field of view*, un unghi între plane și `aspect`=raportul dintre lungimile laturilor dreptunghiului decupat.
- De testat pe codul sursă `07_01_desenare_cub.cpp` cum este realizată proiecția dacă modificăm diverși parametri ai funcției `glm::frustum( )`. De urmărit: (i) cum este realizată decuparea; (ii) cum arată obiectele randate. De exemplu: ce se întâmplă dacă modificăm `dnear`?
  Aceleași întrebări pentru `glm::perspective ( )`.

Cazul 3D - proiecții perspective. Valoarea *dnear* și efectul asupra desenului în cazul `glFrustum`

Presupunem că nu modificăm valoarile *xwmin*, *xwmax*, *ywmin*, *ywmax*. Dacă *dnear* este mic, atunci trunchiul de piramidă decupat are "deschidere mai mare" și obiectele vor părea mai mici. Dacă *dnear* este mare, atunci trunchiul de piramidă decupat este "mai îngust" și obiectele vor părea mai mari.

Concluzie - vizualizare și proiecție

- Este necesară trecerea de la scena 3D (obiectele care se doresc a fi reprezentate), teoretic infinită, la imaginea 2D (primitivele care sunt randate), inclusă într-un dreptunghi cu dimensiuni date. Pentru a realiza acest lucru, sunt urmați doi pași. Acești pași corespund unor transformări specifice:

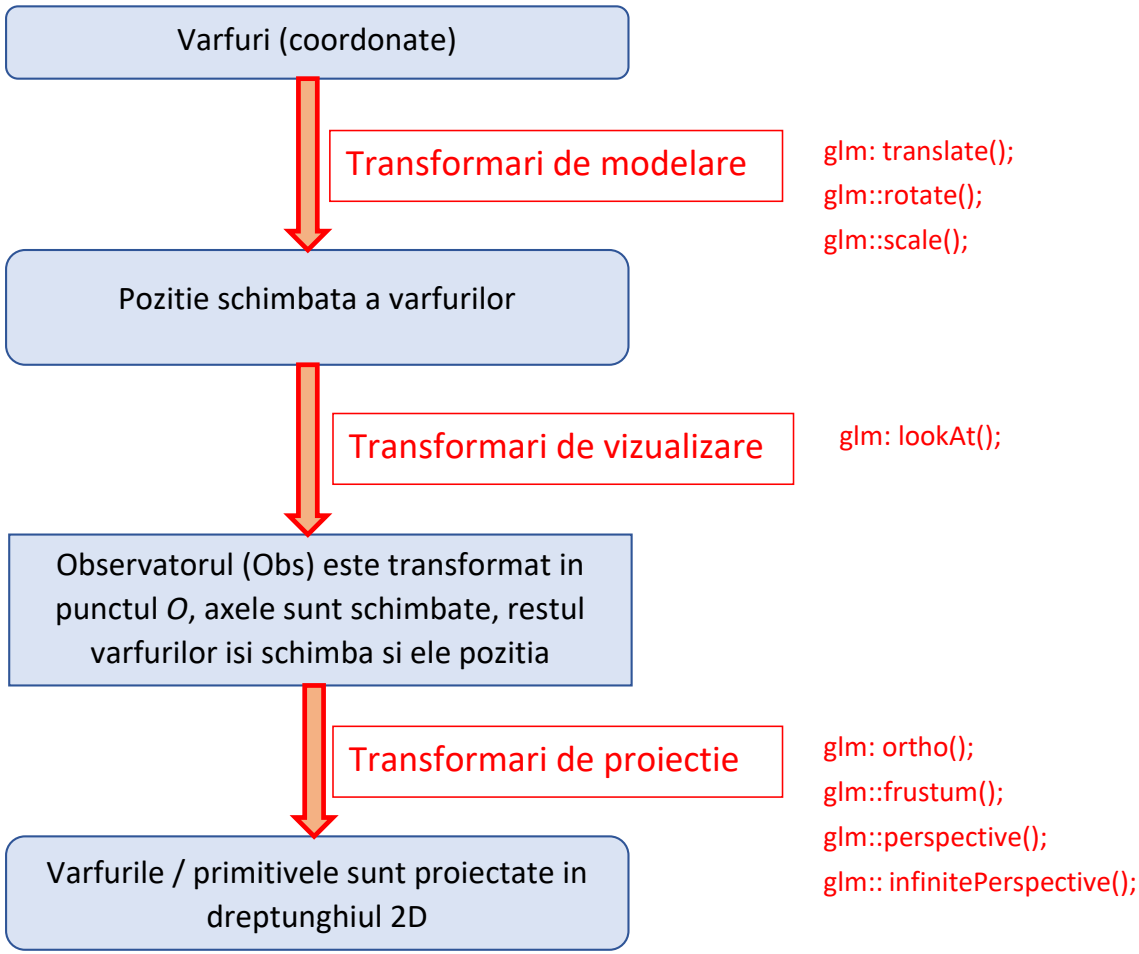
Concluzie - vizualizare și proiecție

- Este necesară trecerea de la scena 3D (obiectele care se doresc a fi reprezentate), teoretic infinită, la imaginea 2D (primitivele care sunt randate), inclusă într-un dreptunghi cu dimensiuni date. Pentru a realiza acest lucru, sunt urmați doi pași. Acești pași corespund unor transformări specifice:
  - Este stabilit modul în care este vizualizată scena 3D (poziția observatorului), prin introducerea coordonatelor de vizualizare și schimbarea reperului. Funcția specifică din GLM este `glm::lookAt();`.

Concluzie - vizualizare și proiecție

- Este necesară trecerea de la scena 3D (obiectele care se doresc a fi reprezentate), teoretic infinită, la imaginea 2D (primitivele care sunt randate), inclusă într-un dreptunghi cu dimensiuni date. Pentru a realiza acest lucru, sunt urmați doi pași. Acești pași corespund unor transformări specifice:
  - Este stabilit modul în care este vizualizată scena 3D (poziția observatorului), prin introducerea coordonatelor de vizualizare și schimbarea reperului. Funcția specifică din GLM este `glm::lookAt();`.
  - Este stabilit modul în care se realizează (i) decuparea (infinit $\longrightarrow$ finit); (ii) proiecția (3D $\longrightarrow$ 2D). În GLM sunt mai multe funcții specifice, depinzând de obiectivul urmărit: `glm::ortho; glm::frustum(); glm::perspective(), glm::infinitePerspective()`.

Concluzie - fluxul transformărilor

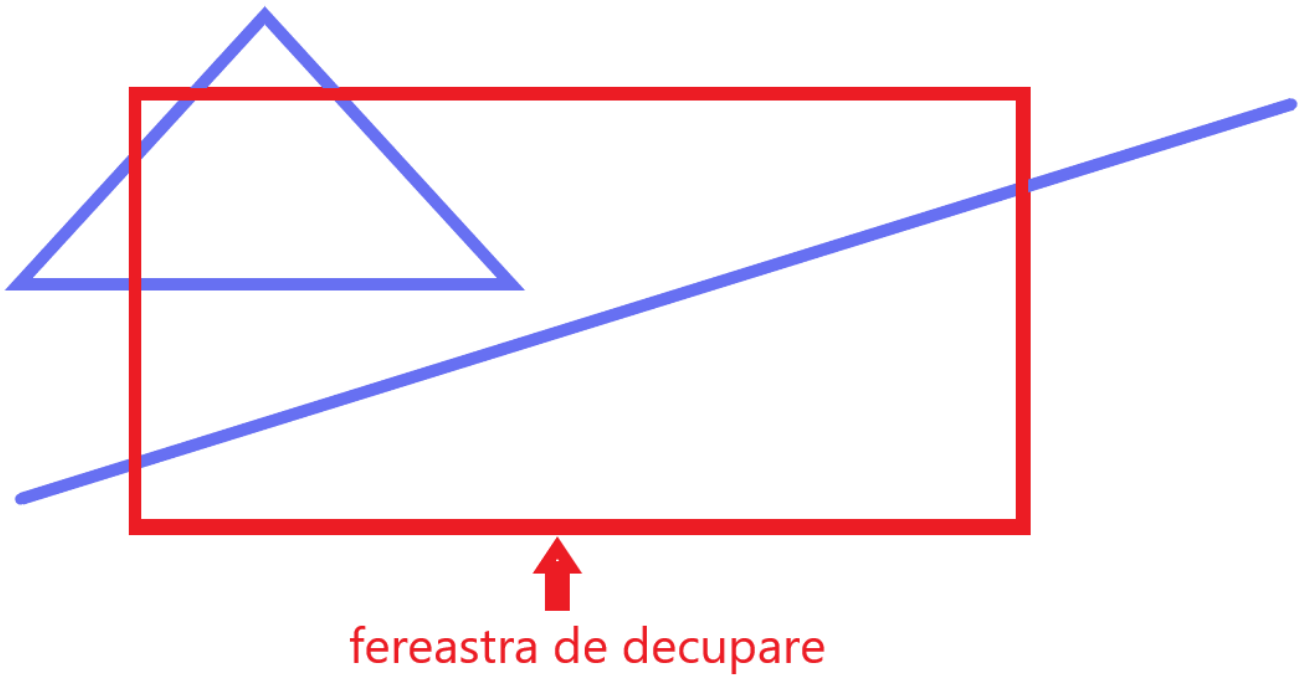


Atenție la ordinea înmulțirii matricelor corespunzătoare din shader:

`matrProiectie * matrVizualizare * matrModelare.`

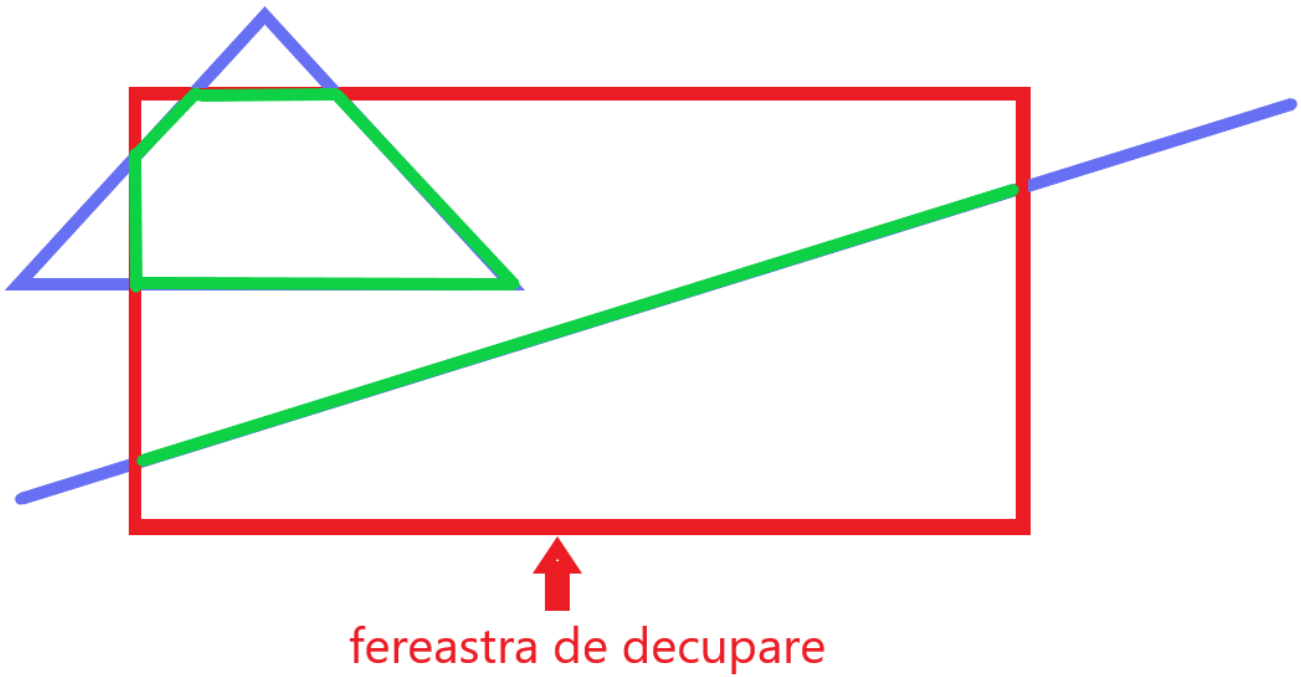
Algoritmi de decupare - motivație

În cazul primitivelor din desen, doar o parte a acestora intersectează fereastra de decupare și urmează să fie randate.



Algoritmi de decupare - motivație

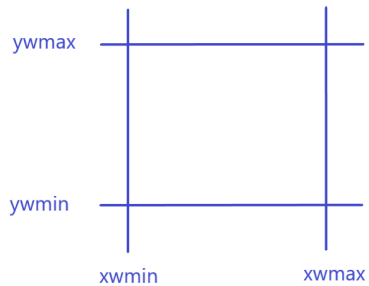
Pentru segment va fi randată doar o porțiune. În cazul triunghiului, va fi decupată o porțiune a sa, fiind randat un pentagon.



Algoritmi de decupare - observații generale

- Trebuie stabilit dacă o primitivă geometrică intersectează sau nu fereastra de decupare. În continuare presupunem că suntem în cazul 2D, fereastra de decupare fiind delimitată de dreptele

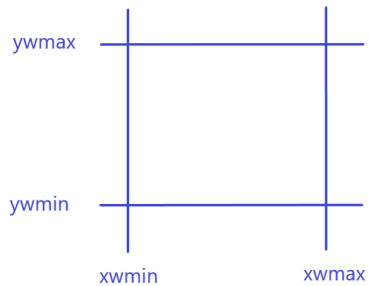
$$x = xwmin,\ x = xwmax,\ y = ywmin,\ y = wymax.$$



Algoritmi de decupare - observații generale

- Trebuie stabilit dacă o primitivă geometrică intersectează sau nu fereastra de decupare. În continuare presupunem că suntem în cazul 2D, fereastra de decupare fiind delimitată de dreptele

$$x = xwmin,\ x = xwmax,\ y = ywmin,\ y = wymax.$$

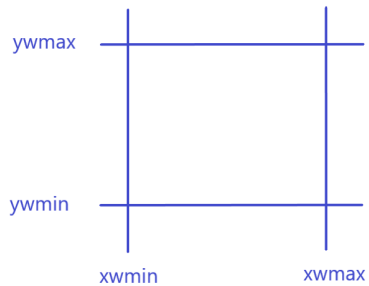


- Algoritmii de decupare depind de tipul primitivei. În cazul punctelor: se bazează pe inegalități.

Algoritmi de decupare - observații generale

- Trebuie stabilit dacă o primitivă geometrică intersectează sau nu fereastra de decupare. În continuare presupunem că suntem în cazul 2D, fereastra de decupare fiind delimitată de dreptele

$$x = xwmin,\ x = xwmax,\ y = ywmin,\ y = wymax.$$

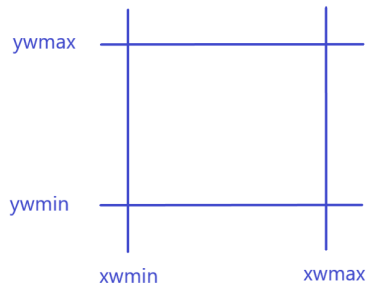


- Algoritmii de decupare depind de tipul primitivei. În cazul punctelor: se bazează pe inegalități.
- În continuare: algoritmi pentru segmente de dreaptă (pornind de la aceștia se poate ajunge la algoritmi pentru poligoane). Sunt descriși doi algoritmi: (i) Cohen-Sutherland (calitativ), (ii) Liang-Barski (cantitativ).

Algoritmi de decupare - observații generale

- Trebuie stabilit dacă o primitivă geometrică intersectează sau nu fereastra de decupare. În continuare presupunem că suntem în cazul 2D, fereastra de decupare fiind delimitată de dreptele

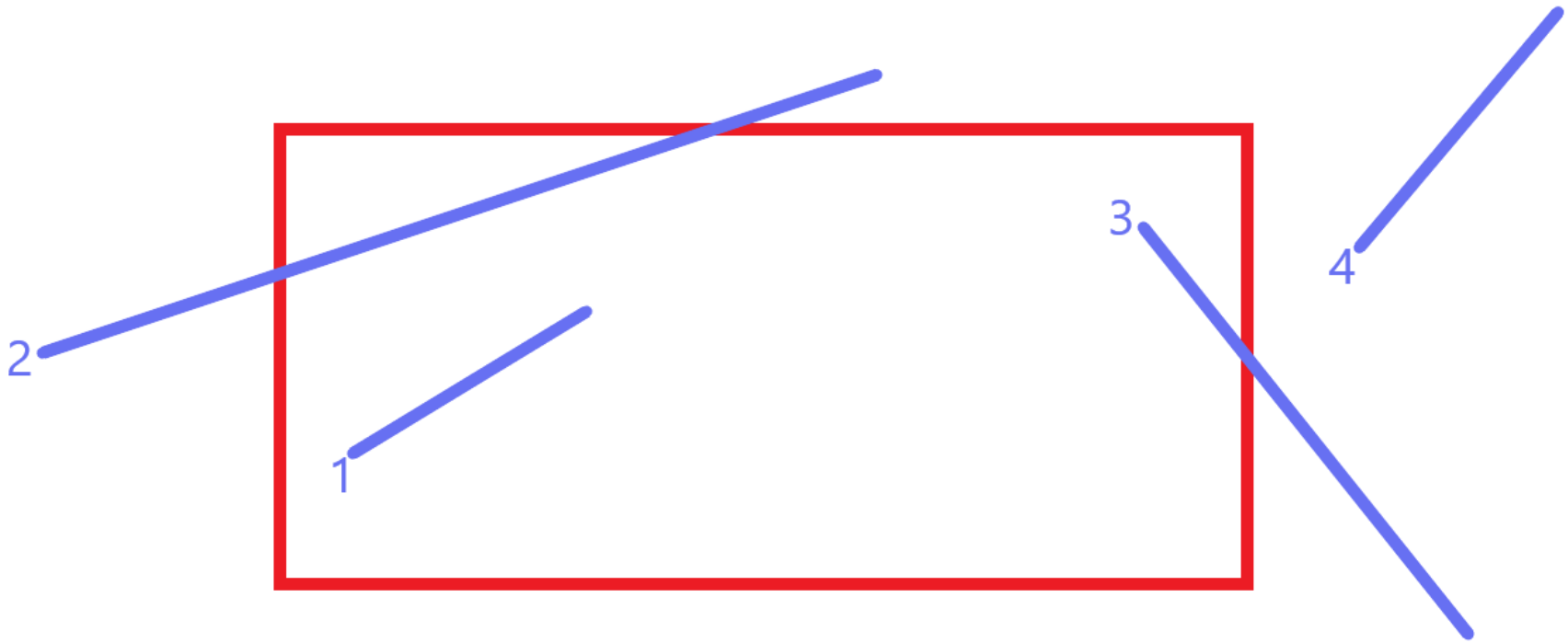
$$x = xwmin,\ x = xwmax,\ y = ywmin,\ y = wymax.$$



- Algoritmii de decupare depind de tipul primitivei. În cazul punctelor: se bazează pe inegalități.
- În continuare: algoritmi pentru segmente de dreaptă (pornind de la aceștia se poate ajunge la algoritmi pentru poligoane). Sunt descriși doi algoritmi: (i) Cohen-Sutherland (calitativ), (ii) Liang-Barski (cantitativ).
- Alte metode / referințe se găsesc în *Line Clipping in 2D: Overview, Techniques and Algorithms*, 2022

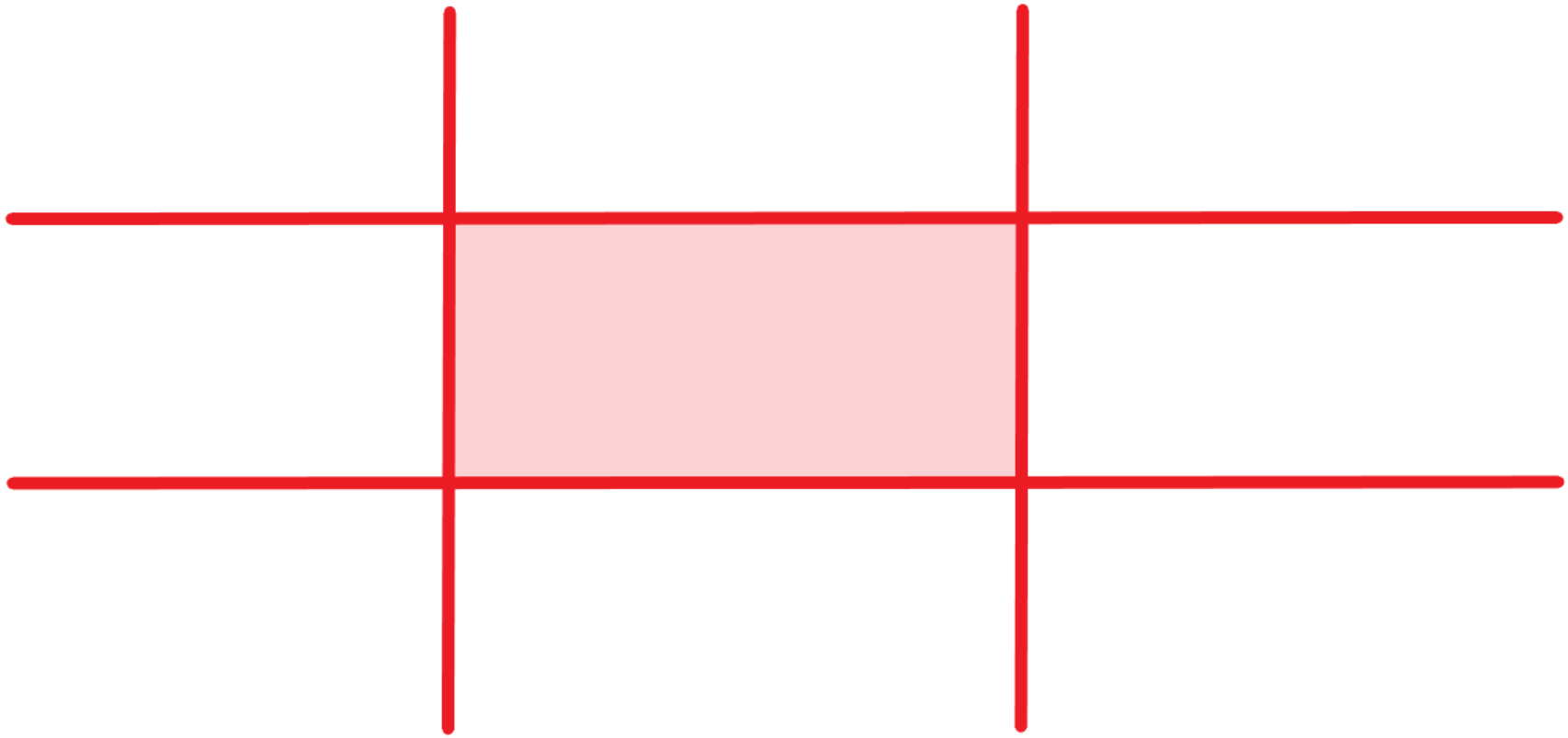
Abordare calitativă. Algoritmul Cohen-Sutherland

Diverse configurații ale segmentelor față de fereastra de decupare.



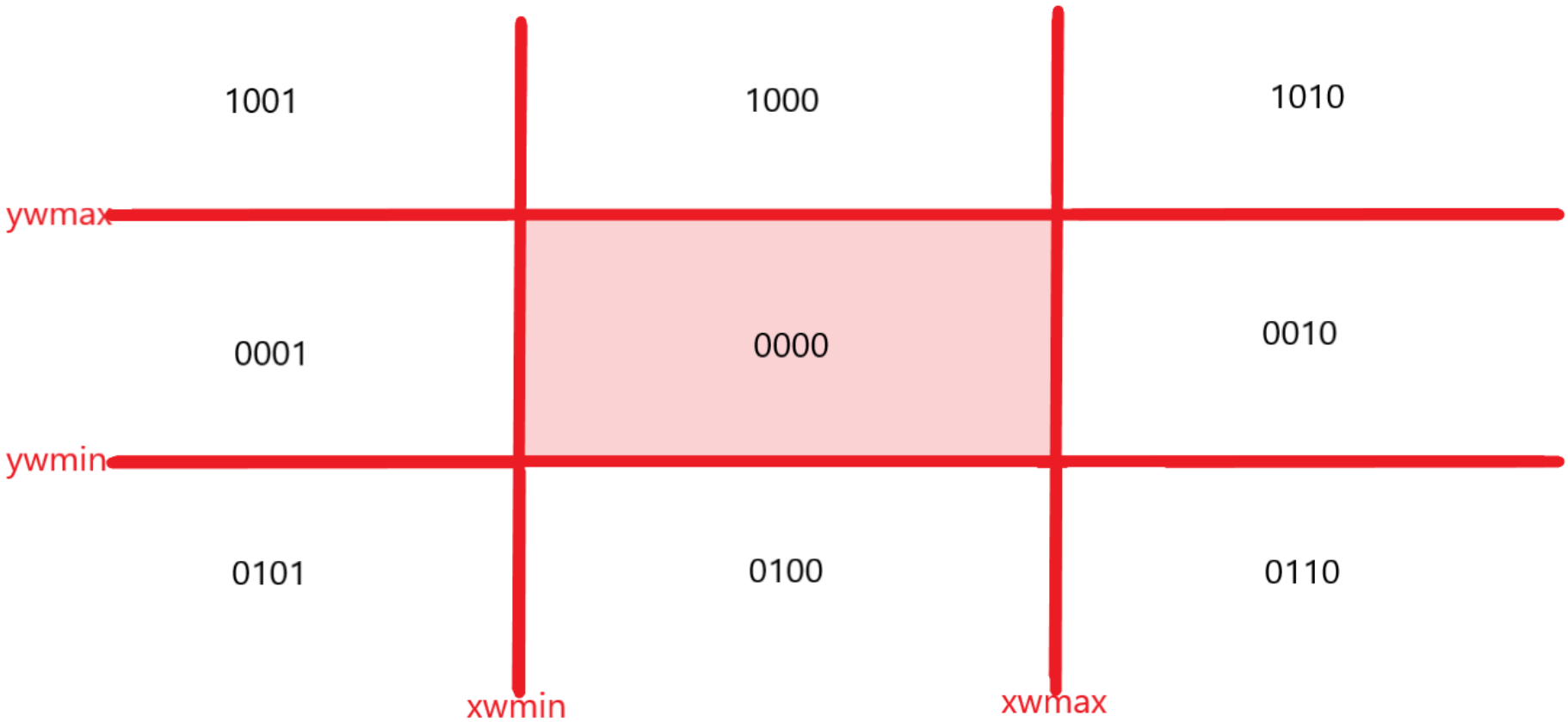
Algoritmul Cohen-Sutherland - împărțirea planului

Planul este împărțit în 9 regiuni, fiecareia dintre ele îi este asociat un cod pe 4 biți (TBRL – Top Bottom Right Left).



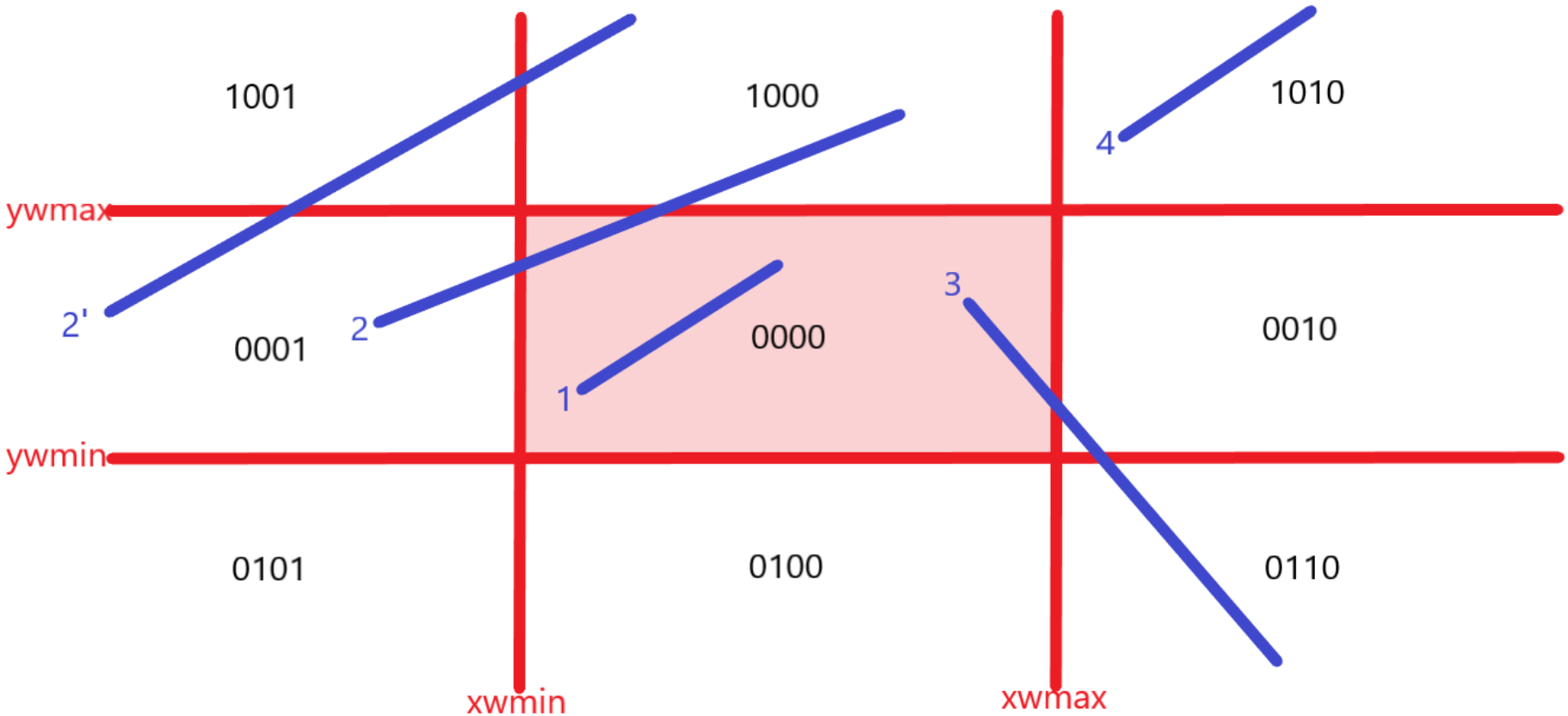
Algoritmul Cohen-Sutherland - codurile regiunilor

Planul este împărțit în 9 regiuni, fiecareia dintre ele îi este asociat un cod pe 4 biți (TBRL – Top Bottom Right Left).



Algoritmul Cohen-Sutherland - coduri și segmente

Dat un segment *s*, codurile asociate extremităților sale dau o primă informație despre poziția segmentului *s* față de fereastra de decupare.



Algoritmul Cohen-Sutherland - sinteză

- Pentru un punct $M = (x_M, y_M)$ se stabilește codul regiunii:
  $x < xwmin \Rightarrow L = 1, R = 0$, deci $**01$
  $x > xwmax \Rightarrow L = 0, R = 1$, deci $**10$, etc

Algoritmul Cohen-Sutherland - sinteză

- Pentru un punct $M = (x_M, y_M)$ se stabilește codul regiunii:
  $x < xwmin \Rightarrow L = 1, R = 0$, deci $**01$
  $x > xwmax \Rightarrow L = 0, R = 1$, deci $**10$, etc
- Date două puncte, P și Q, cele două coduri asociate dau o primă informație referitoare la poziția segmentului $[PQ]$ față de fereastra de decupare.

Algoritmul Cohen-Sutherland - sinteză

- Pentru un punct $M = (x_M, y_M)$ se stabilește codul regiunii:
  $x < xwmin \Rightarrow L = 1, R = 0$, deci $**01$
  $x > xwmax \Rightarrow L = 0, R = 1$, deci $**10$, etc
- Date două puncte, P și Q, cele două coduri asociate dau o primă informație referitoare la poziția segmentului $[PQ]$ față de fereastra de decupare.
- Există cazuri în care folosind codurile se poate lua o decizie referitoare la poziția relativă a segmentului față de fereastră (de exemplu două coduri de tipul $**10$, ambele coduri 0000, etc.).
- Există cazuri în care folosind codurile nu se poate lua o decizie (de exemplu 0001 și 1000), fiind necesari alți algoritmi, cantitativi (se determină explicit coordonatele intersecțiilor dintre segment și dreptele suport ale ferestrei de decupare, se stabilește dacă aceste puncte sunt pe laturile dreptunghiului, etc.).

Abordare cantitativă. Algoritmul Liang-Barsky

- **Principiu:** Utilizează **reprezentarea parametrică** a unei drepte. Ideea centrală: fiecărui punct de pe dreaptă îi corespunde exact un număr real (parametru) și reciproc.

Abordare cantitativă. Algoritmul Liang-Barsky

- **Principiu:** Utilizează **reprezentarea parametrică** a unei drepte. Ideea centrală: fiecărui punct de pe dreaptă îi corespunde exact un număr real (parametru) și reciproc.
- **Explicit:** Fie $P_0 = (x_0, y_0), P_1 = (x_1, y_1) \in \mathbb{R}^2$, presupunem (fără a restrânge generalitatea) că $x_0 < x_1, y_0 < y_1$.
  Notăm $\Delta x = x_1 - x_0, \Delta y = y_1 - y_0$.
  Dreapta P_0P_1 are reprezentarea parametrică

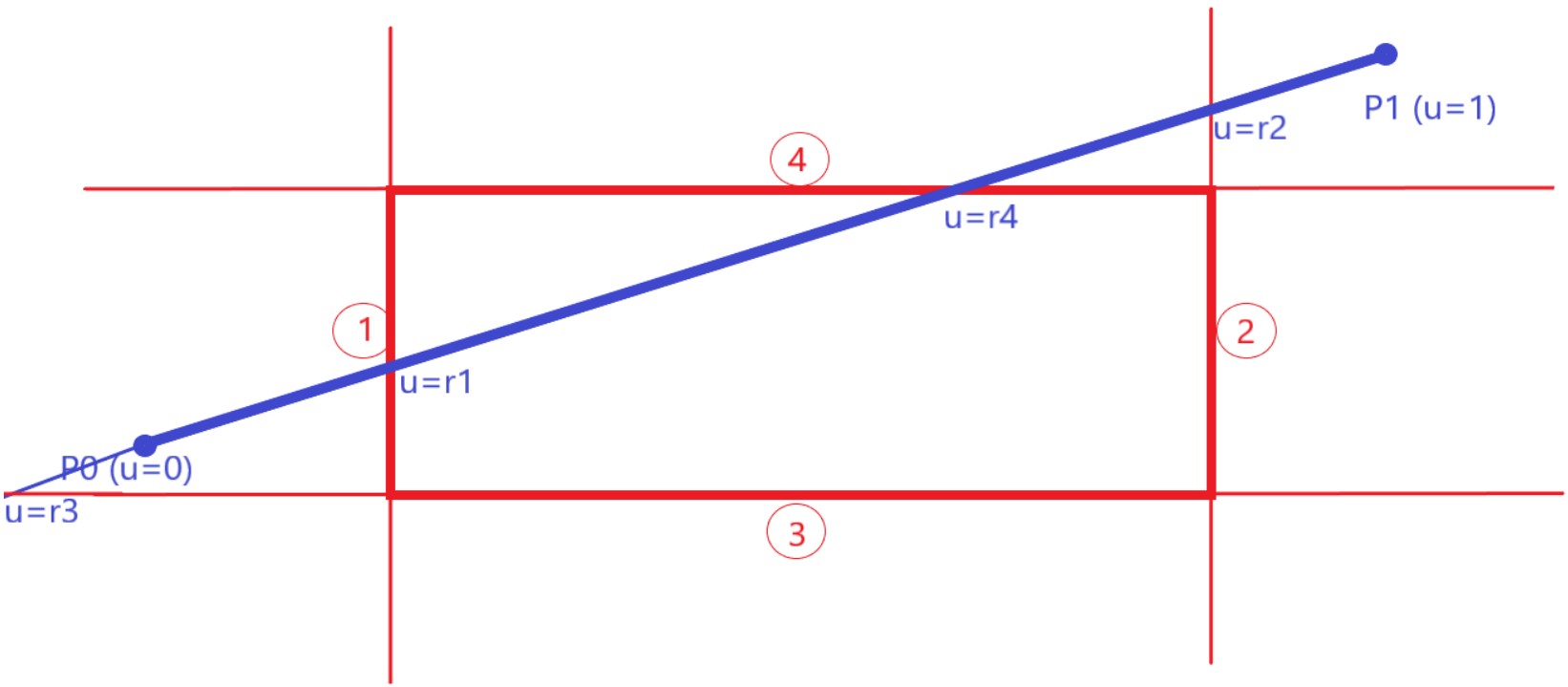
$$\begin{cases} x = x_0 + u \cdot \Delta x \\ y = y_0 + u \cdot \Delta y \end{cases}, u \in \mathbb{R}.$$

  A da un punct de pe dreapta P_0P_1 revine la a indica o valoare a lui u și reciproc.

Algoritmul Liang-Barsky - intersecții cu fereastra

Intersecții cu fereastra de decupare: Dreapta P_0P_1 intersectează dreptele suport ale laturilor dreptunghiului (notate $1, 2, 3, 4$) în puncte care corespund unor valori r_1, r_2, r_3, r_4 ale parametrului (calculabile explicit!). De exemplu, r_1 se determină din condiția

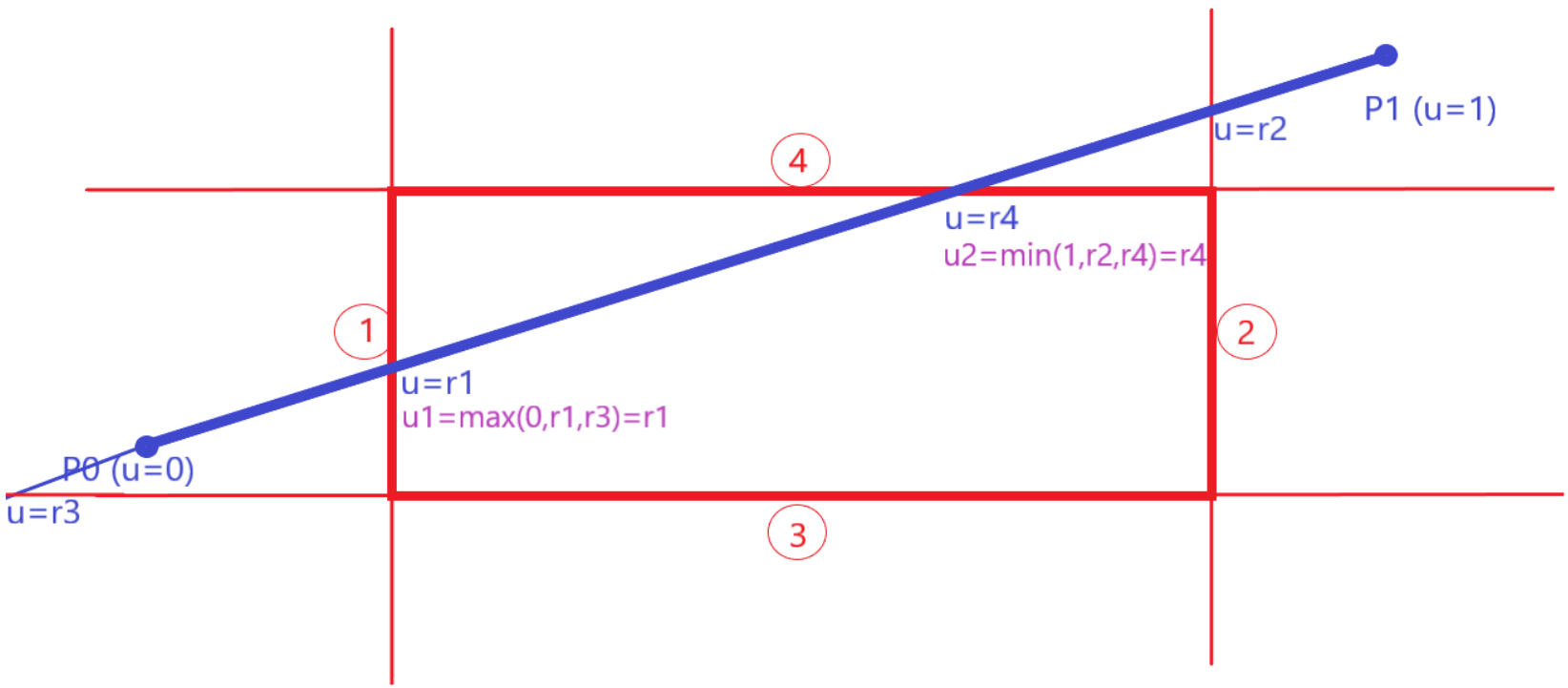
$$xwmin = x_0 + r_1 \cdot \Delta x, \text{ deci } r_1 = \frac{xwmin - x_0}{\Delta x}, \text{ etc.}$$



Algoritmul Liang-Barsky - valori importante ale parametrului

"Intrare" în dreptunghi: $u_1 = \max(0, r_1, r_3)$
(start segment, fâșia verticală, fâșia orizontală)

"Ieșire" din dreptunghi: $u_2 = \min(1, r_2, r_4)$
(stop segment, fâșia verticală, fâșia orizontală)



Algoritmul Liang-Barsky - condiția de intersecție

Condiția ca **segmentul** $[P_0P_1]$ să intersecteze dreptunghiul de decupare:

$$u_1 < u_2$$

(intuitiv: "intrăm în dreptunghiul de decupare înainte de a ieși").
Mai mult, segmentul care urmează să fie decupat are extremitățile corespunzătoare parametrilor u_1 și u_2.