

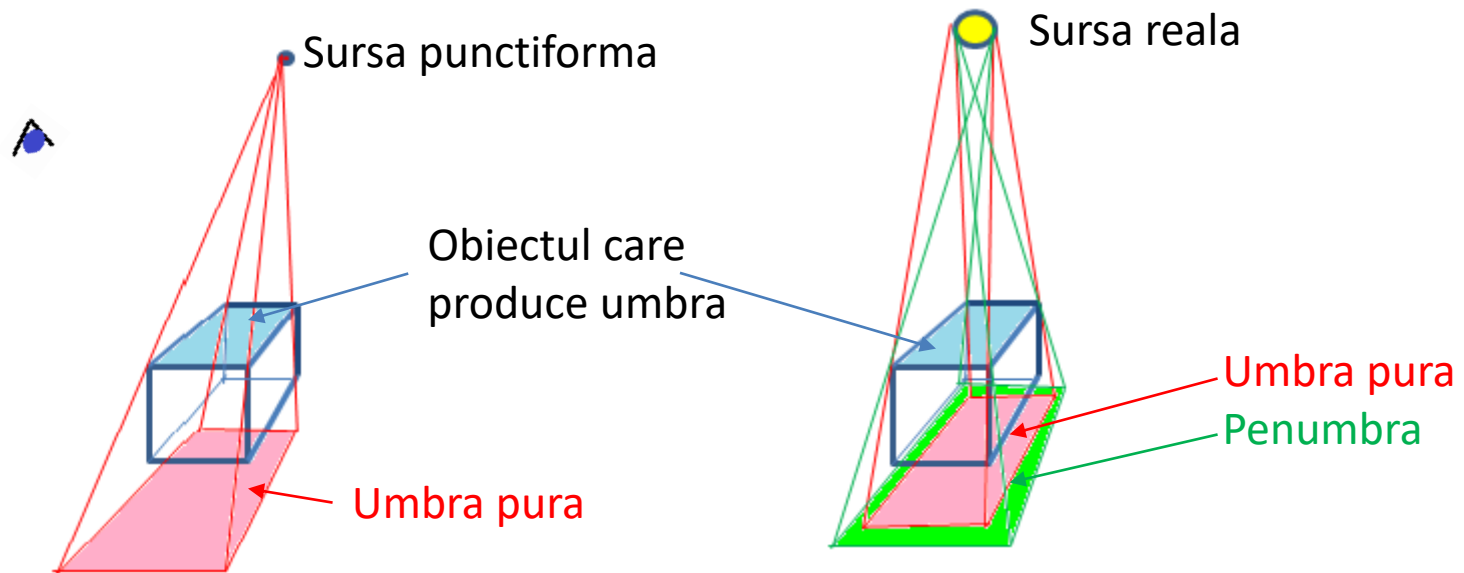
# *Redarea umbrelor în imagini*

*Prof. univ. dr. ing. Florica Moldoveanu*

*Curs Elemente de Grafică pe Calculator – UPB, Automatică și Calculatoare*  
2020-2021

# INTRODUCEREA UMBRELOR IN IMAGINI(1)

- Atunci când un observator privește o scenă 3D luminată de o sursă de lumină dintr-o poziție diferită de aceea a sursei de lumină, va vedea umbre produse de obiectele scenei.
- Umbrele au o contribuție însemnată la realismul imaginii, îmbunătățind percepția profunzimii.



In general, se considera numai umbra pura: calcule mai simple.

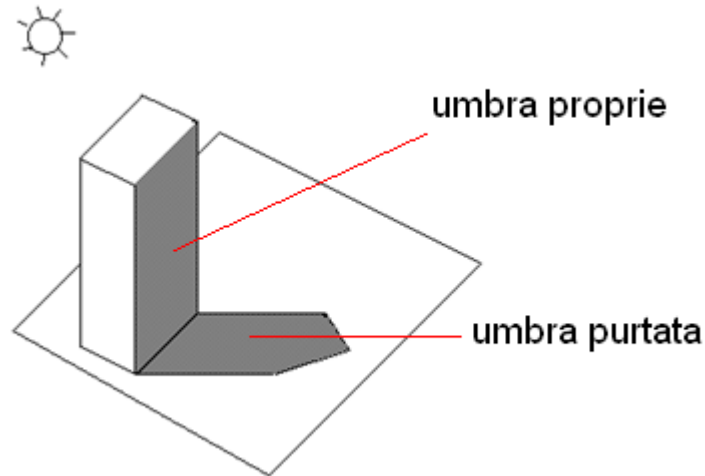
# INTRODUCEREA UMBRELOR IN IMAGINI(2)

- ❑ **Volumul de calcule necesar introducerii umbrelor depinde de poziția sursei de lumină:**
  - Dacă sursa este la infinit calculele sunt mai simple: se efectuează o proiecție paralelă din poziția sursei
  - În cazul unei surse situate la distanță finită dar în afara scenei, este necesară o proiecție perspectivă din poziția sursei.
  - Cazul cel mai dificil este acela în care sursa este situată în scenă.
- ❑ Problema determinării umbrelor este similară celeia de determinare a părților nevizibile ale unei scene 3D: **partile scenei 3D care nu sunt vizibile din poziția sursei de lumină sunt în umbră.**
- Crearea unei imagini cu umbre presupune rezolvarea de două ori a problemei suprafețelor nevizibile: o dată privind scena din poziția sursei de lumină, a doua oară privind-o din poziția observatorului.

# UMBRE PLANARE (1)

Umbrele produse de obiectele scenei 3D sunt de două tipuri:

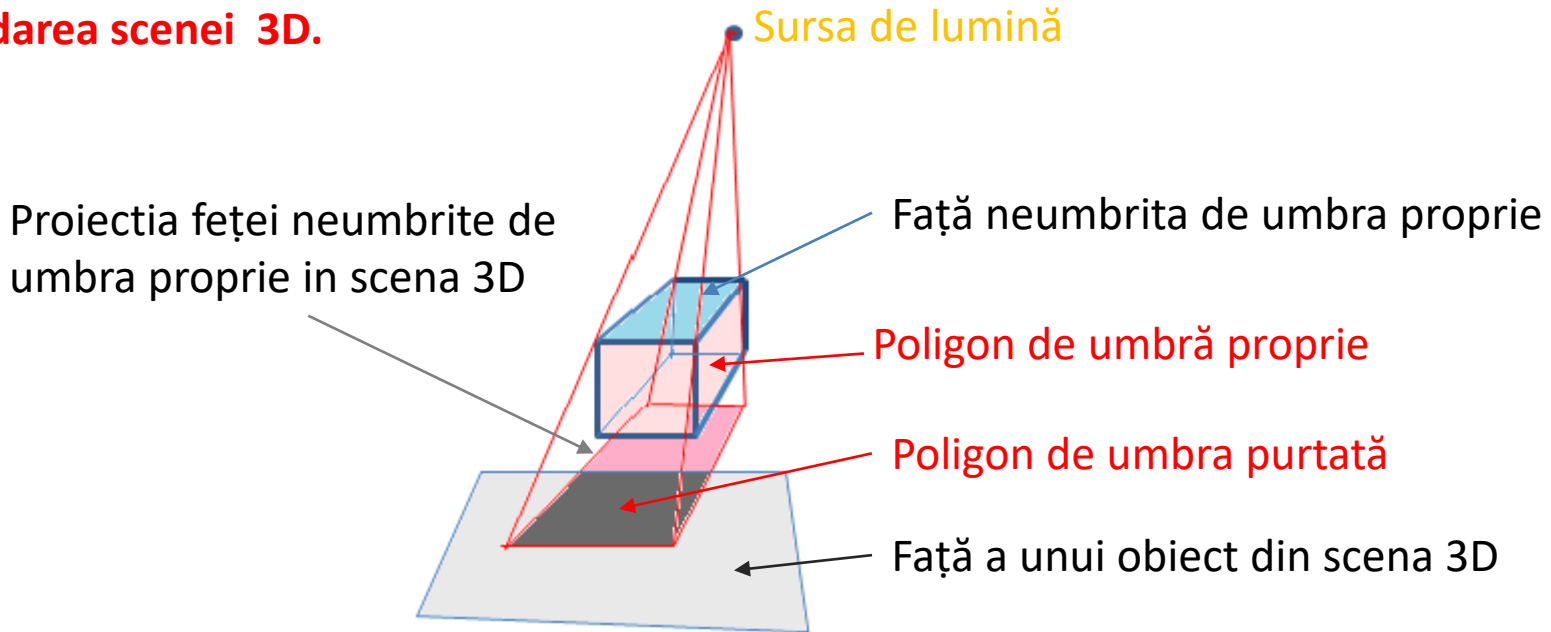
**umbre proprii și umbre purtate.**



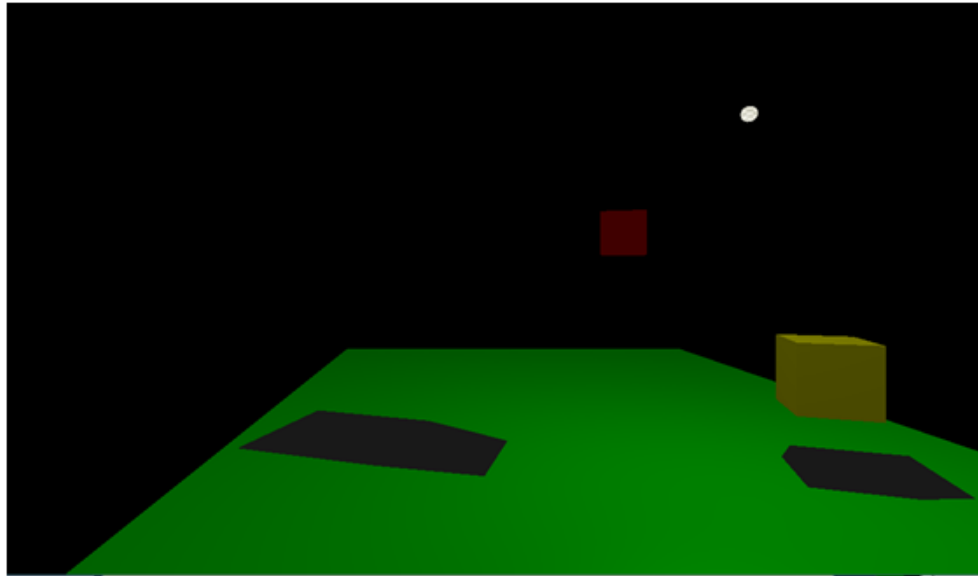
- **Umbrele proprii** sunt generate de obiectul însuși, care împiedică lumina să ajungă la unele dintre fețele sale: **fețele umbrite de umbra proprie sunt fețele auto-obturate** atunci când scena este văzută din poziția sursei de lumină.
- **Umbrele purtate** sunt umbrele pe care un obiect le produce pe alte obiecte ale scenei, la care lumina nu ajunge din cauza obiectului.

## UMBRE PLANARE (2)

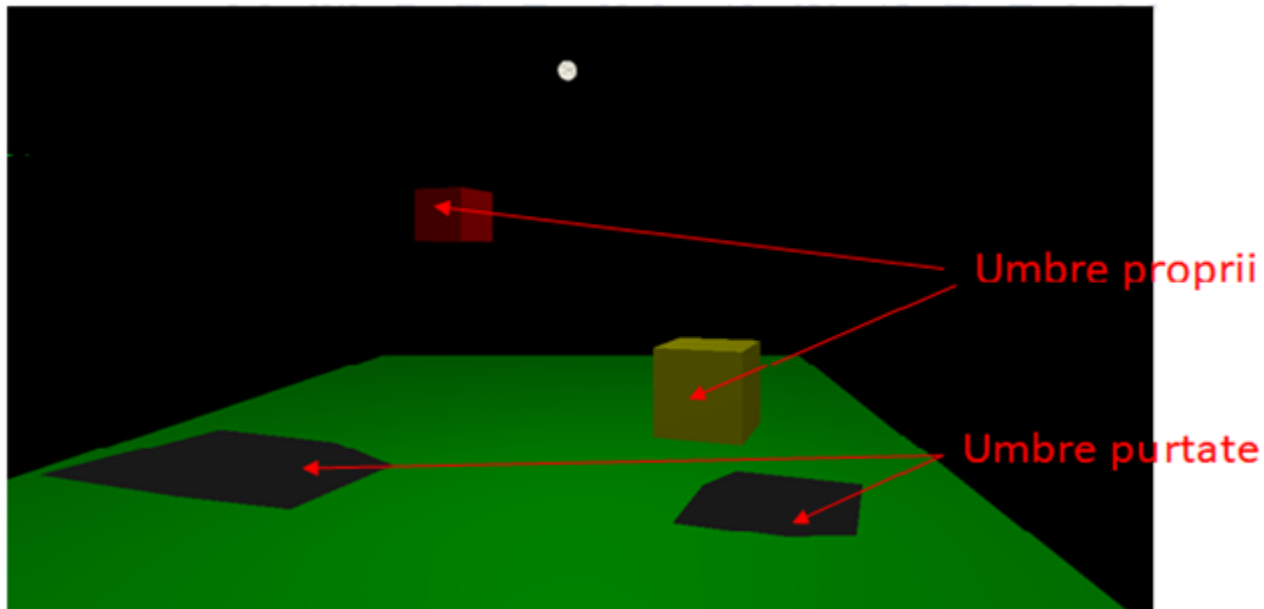
- ❑ Un poligon de umbră purtată este coplanar cu fața obiectului care conține umbra purtată.
- ❑ Un poligon de umbră proprie este un poligon identic cu fața umbrită de umbra proprie.
- ❑ Poligoanele de umbră proprie și cele de umbra purtată se folosesc ca poligoane detaliu la redarea scenei 3D.



# UMBRE PLANARE (3)



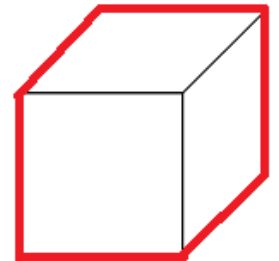
Scena cu umbre  
vazuta dintr-o pozitie  
a observatorului  
diferita de cea a  
sursei.



# UMBRE PLANARE (4)

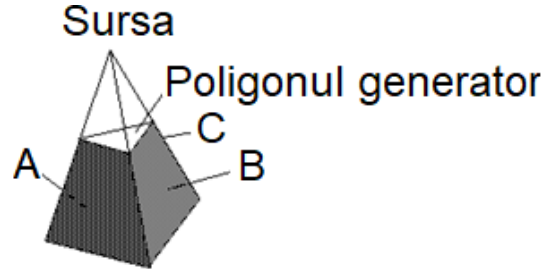
## Redarea scenei cu umbre (functia Update)

1. Se cere rasterizarea poligoanelor scenei.
  2. Se dezactivează testul de adâncime (astfel incat fragmentele poligoanelor de umbra să suprascrie unele dintre fragmentele poligoanelor scenei): `glDisable(GL_DEPTH_TEST)`.
  3. Se cere rasterizarea poligoanelor de umbră (acestea au o culoare de umbra)
- Pot fi generate mai multe vederi ale scenei fără a recalcula umbrele, deoarece umbrele depind numai de poziția sursei (surselor) de lumină.
  - Numărul de poligoane de umbră este mai mic, dacă în loc să se proiecteze fiecare față luminată de sursă, se proiectează silueta fiecărui obiect văzută din poziția sursei de lumină.
  - Calculul poligoanelor de umbra este destul de complicat, de aceea în prezent se folosesc alte metode pentru introducerea umbrelor în imagini.



# VOLUME DE UMBRA (1)

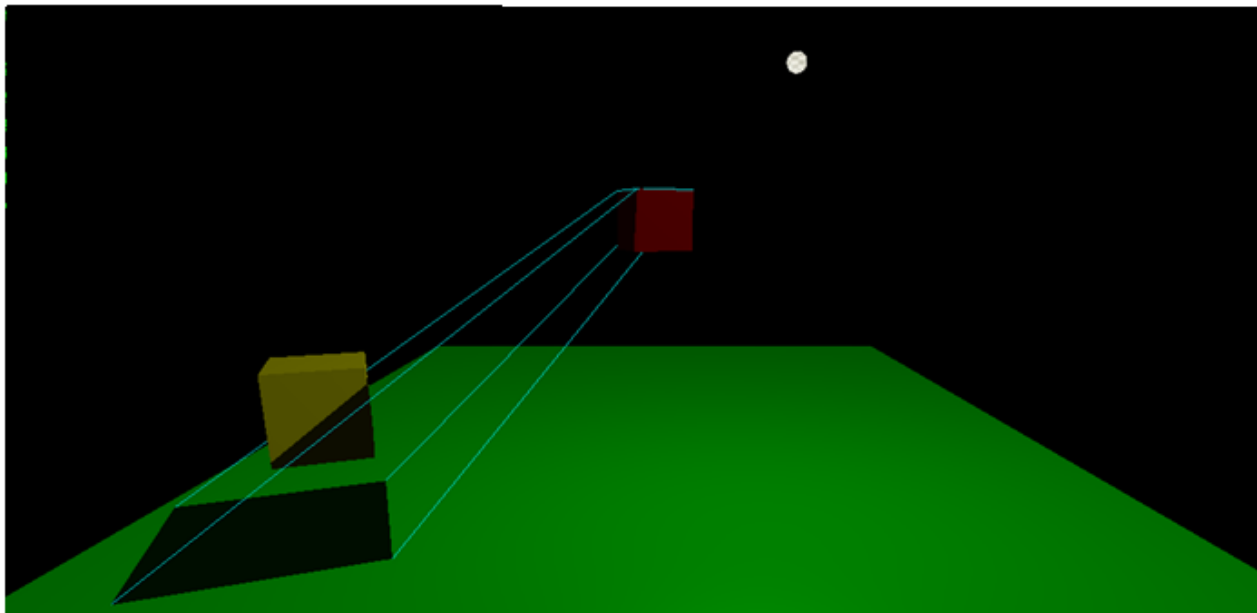
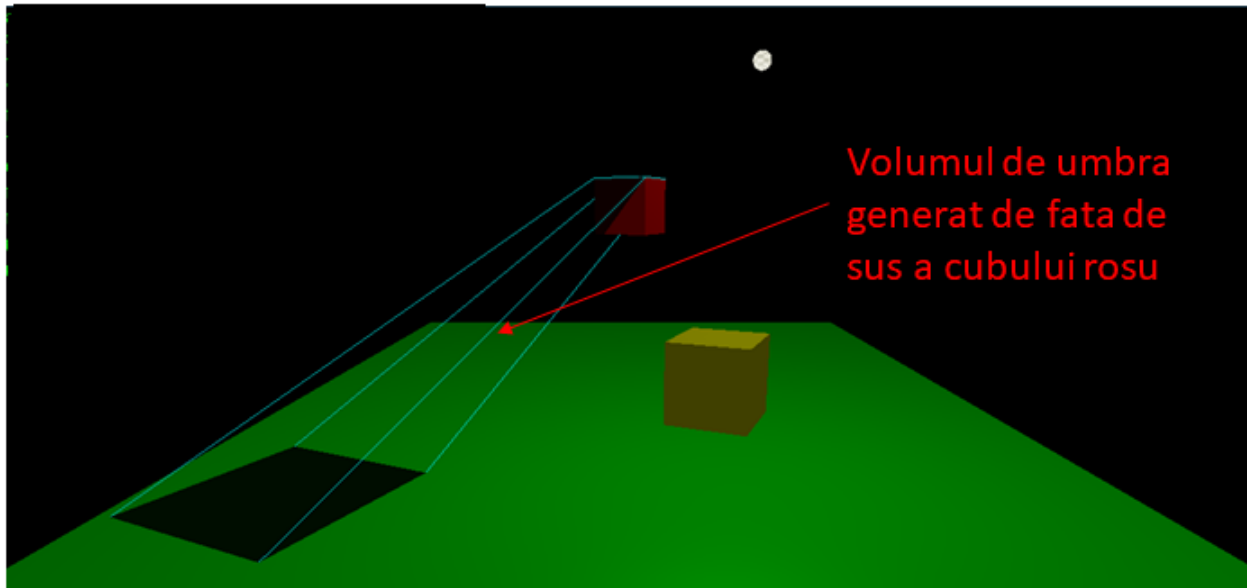
- ❑ Sursa de lumină este considerată punctiformă iar scena 3D alcatuita din poligoane.
- ❑ Un **volum de umbră** este definit de o sursă de lumină și un poligon iluminat (vizibil din poziția sursei de lumină), pe care-l vom numi **poligonul generator**.



- ❑ Volumul de umbra determinat de o sursă și un poligon generator este delimitat de o față care reprezintă poligonul generator scalat. Această față este situată la o distanță față de sursă dincolo de care intensitatea luminii sursei este neglijabilă, deci orice punct aflat dincolo de această limită este umbrit.
- ❑ Volumul de umbră poate fi decupat la marginile volumului vizual.



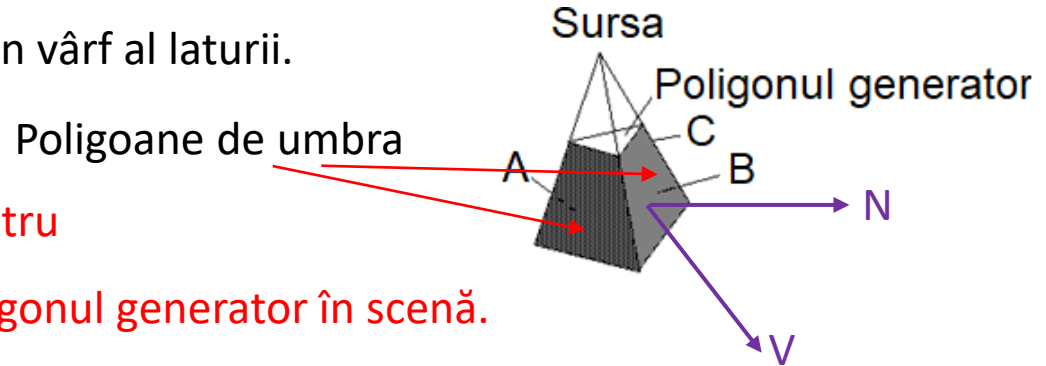
# VOLUME DE UMBRA (2)



# VOLUME DE UMBRA (3)

## Determinarea umbririi produse in scena 3D de un volum de umbra

❑ Fețele laterale ale volumului sunt numite **poligoane de umbră**. Un poligon de umbră este determinat de o latură a poligonului generator și cele două drepte care pleacă din sursa de lumină, fiecare trecând printr-un vârf al laturii.



❑ Poligoanele de umbră se folosesc pentru determinarea umbririi produse de poligonul generator în scenă.

❑ Normalele la poligoanele de umbra trebuie sa fie orientate spre exteriorul volumului.

Notăm cu :

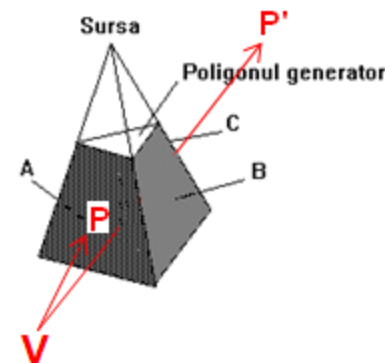
- PUV poligoanele de umbră care sunt vizibile din poziția observatorului (A și B în figura) și cu
- PUN poligoanele de umbră care nu sunt vizibile din poziția observatorului (poligonul C).
- Clasificarea poligoanelor de umbra in PUV si PUN se poate face pe baza normalelor la poligoane si a pozitiei observatorului (back face culling).

# VOLUME DE UMBRA (4)

Fie

- un punct  $P$  al unei suprafețe din scena 3D și
- VP vectorul din poziția observatorului ( $V$ ) în punctul  $P$ .

**Punctul  $P$  este umbrit dacă numărul de poligoane de tip PUV intersectate de vectorul VP este mai mare decât numărul de poligoane de tip PUN intersectate de vector.**



Acesta este singurul caz, atunci când punctul  $V$  nu este în volumul de umbră.

□ In general, pentru a determina dacă un punct  $P$  este în umbră, se poate folosi un contor în care inițial se memorează numărul de volume de umbră care conțin poziția observatorului.

- Se asociază poligoanelor de tip PUV valoarea  $+1$  iar celor de tip PUN valoarea  $-1$ .
- Atunci când vectorul VP traversează un poligon de umbră, se adună la contor valoarea asociată poligonului.
- Punctul  $P$  este umbrit dacă valoarea contorului este pozitivă în  $P$ .

# VOLUME DE UMBRA (5)

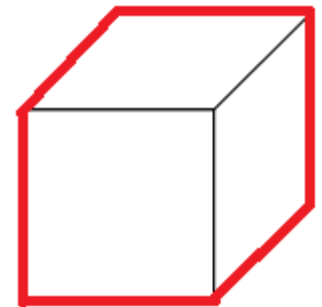
➤ Volumul de calcule presupus de acest algoritm poate fi redus dacă în loc să se calculeze volumul de umbră pentru fiecare poligon vizibil din poziția sursei, se calculează un singur volum de umbră pentru o suprafață poliedrală.

În acest scop, **se determină poligoanele de umbră numai pentru laturile care fac parte din silueta suprafeței, văzută din poziția sursei de lumina.**

❑ Silueta unei suprafețe, corespunzătoare unui punct de observare, este un set conectat de laturi care aparțin poligoanelor vizibile din punctul de observare.

❑ O latură de siluetă este fie o latură de margine a unei suprafețe deschise, fie o latură care separă un poligon vizibil de unul nevizibil.

➤ Pentru determinarea laturilor de siluetă, este necesar să se folosească o structură de date care reflectă adiacența poligoanelor. Cunoșcându-se poligonul adiacent pe fiecare latură a fiecărui poligon vizibil din poziția sursei, se pot determina rapid laturile de siluetă.



# VOLUME DE UMBRA (6)

## Implementarea metodei folosind stencil-buffer(1)

- ❑ **Buffer-ul de marcaje (stencil-buffer)** este prezent pe placile grafice actuale (in memoria GPU), alaturi de buffer-ul imagine (frame-buffer) si buffer-ul de adancime (z-buffer).
  - este o memorie de numere intregi asociata pixelilor imaginii, de regula un byte/pixel;
  - poate fi utilizat pentru a implementa diferite operatii grafice, de regula in combinatie cu z-buffer, cum este si cazul implementarii volumelor de umbra:
    - **valorile din stencil-buffer pot fi incrementate/ decrementate automat pentru fiecare fragment care trece/ nu trece testul de adancime.**
- ❑ Exista mai multe metode de a utiliza stencil-buffer pentru implementarea metodei volumelor de umbre.

# VOLUME DE UMBRA (7)

## Implementarea metodei folosind stencil-buffer(2)

❑ In metoda prezentata in continuare, se creaza in stencil-buffer o masca a umbrelor din imagine: orice pixel de umbra, (x,y), va fi reprezentat in stencil-buffer printr-o valoare  $\neq 0$ .

❑ Pixelii de umbra sunt afisati in culorile fragmentelor vizibile, calculate cu formula.

$$C_{\text{fragment}} = C_{\text{emisiva}} + C_{\text{cambianta}} = k_e + k_a * \text{AmbientLight}$$

$k_e, k_a$  – proprietati de material

Cemisiva: o culoare de umbra

- Se presupune ca s-au calculat poligoanele de umbra.
- Conturul fiecarui poligon de umbra trebuie sa fie orientat in sens trigonometric atunci cand este privit din exteriorul volumului de umbra. In acest fel, poligoanele PUV sunt cele “din fața” observatorului, iar poligoanele PUN sunt cele “din spate”, atunci cand observatorul nu este inclus in nici un volum de umbra.

# VOLUME DE UMBRA (8)

## Implementarea metodei folosind stencil-buffer(3)

Consideram cazul in care observatorul nu este in umbra (cazul uzual).

### Pasul 1

Initializeaza z-buffer si stencil-buffer; // nu trebuie programate, acestea sunt

// initializate in mod automat pentru fiecare cadru imagine

Activeaza scrierea in z-buffer si dezactiveaza scrierea in stencil-buffer ; //implicite

**Rasterizeaza scena ( folosindu-se algoritmul z-buffer) fara lumini**

### Efectul:

— in frame-buffer se obtine imaginea scenei in umbra (fara lumini). Culoarea fiecarui pixel este:

$C_{\text{pixel}} = \text{culoarea emisiva} + \text{culoarea ambienta};$

**La sfarsitul acestui pas, z-buffer contine coordonatele z ale fragmentelor vizibile in imagine.**

# VOLUME DE UMBRA (9)

## Implementarea metodei folosind stencil-buffer(4)

### Pasul 2

```
glDepthMask(GL_FALSE); // Dezactiveaza scrierea in z-buffer  
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE); // Dezactiveaza scrierea in frame-buffer  
glFrontFace(GL_CCW); // fețele “din față” sunt orientate trigonometric  
glCullFace(GL_BACK); // se vor elimina fețele din spate (nu sunt rasterizate poligoanele PUN)  
glStencilFunc(GL_ALWAYS, 0x0, 0xff); // se activeaza scrierea in stencil buffer  
glStencilOp(GL_KEEP, GL_KEEP, GL_INCR); // se incrementeaza stencil buffer pt toate  
// fragmentele care trec testul de adancime  
(https://learnopengl.com/Advanced-OpenGL/Stencil-testing)
```

### Rasterizeaza poligoanele de umbra

### Efectul:

**Sunt rasterizate poligoanele de umbra de tip PUV.**

Pentru fiecare fragment rezultat, f, se face testul:

```
daca f.z < z-buffer[f.y][f.x] // fragmentul f in fața fragmentului vizibil in pixelul (x,y)  
    atunci stencil-buffer[f.y][f.x]++; // se numara poligoanele PUV aflate intre  
    // observator si obiectele scenei
```



# VOLUME DE UMBRA (10)

## Implementarea metodei folosind stencil-buffer(5)

### Pasul 3

```
glCullFace(GL_FRONT); // se vor elimina fețele din față (nu sunt rasterizate poligoanele PUV)  
glStencilOp(GL_KEEP, GL_KEEP, GL_DECR); // se decrementeaza stencil buffer pt toate  
// fragmentele care trec testul de adancime
```

### Rasterizeaza poligoanele de umbra

### Efectul:

**Sunt rasterizate poligoanele de umbra de tip PUN.**

Pentru fiecare fragment rezultat,  $f$ , se face testul:

```
daca  $f.z < z\text{-buffer}[f.y, f.x]$  //  $f$  este in fața fragmentului vizibil in pixelul (x,y)  
    atunci stencil-buffer[f.y, f.x]--; // se numara poligoanele PUN aflate intre observator  
    // si obiectele scenei
```

In stencil-buffer s-a obtinut o masca cu “gauri” in punctele care primesc lumina:

valorile din stencil-buffer sunt diferite de zero pentru pixelii care trebuie afisati in umbra.

# VOLUME DE UMBRA (11)

## Implementarea metodei folosind stencil-buffer(6)

### Pasul 4

```
glEnable(GL_STENCIL_TEST); // Activeaza testul stencil
```

```
glStencilOp(GL_KEEP, GL_KEEP, KEEP); // Dezactiveaza scrierea in stencil-buffer
```

```
glStencilFunc(GL_EQUAL, 0x0, 0xff); // testul stencil: pixelul(x,y) se modifica daca  
// stencil-buffer [y][x]=0
```

```
glDepthMask(GL_TRUE); // Activeaza scrierea in z-buffer
```

```
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE); // Activeaza scrierea in frame-buffer
```

**Rasterizeaza scena (folosindu-se algoritmul zbuffer) cu lumini**

### Efectul:

**Se rasterizeaza scena modificandu-se in frame-buffer numai culorile pixelilor (x,y) pentru care stencil-buffer [y][x] =0 (fragmentul nu este in umbra)**

# VOLUME DE UMBRA (12)

## Implementarea metodei folosind stencil-buffer(7)

### Cazul general:

Se initializeaza stencil-buffer cu o valoare reprezentand numarul de volume de umbra care contin pozitia observatorului.

### Aprecieri asupra metodei volumelor de umbra

#### Pozitive:

- Este generală și poate fi aplicată pentru orice scena, pentru oricate surse de lumina.
- Produce forme de umbre exacte.

#### Negative:

- Complexa din punct de vedere al timpului de calcul:  
se rasterizeaza scena de 2 ori si poligoanele de umbra o singura data dar in 2 pasi!
- Sursele de lumina sunt considerate punctiforme, de aceea nu pot fi obtinute penumbre.

# SHADOW MAPPING (1)

Williams [WILL78] a propus o metodă de generare a umbrelor bazată pe rasterizarea de 2 ori a scenei, cu execuția algoritmului Z-Buffer:

- În prima etapă se construiește buffer-ul Z al scenei văzută din poziția sursei de lumina (în transformarea de vizualizare a scenei – funcția lookAt – poziția observatorului este poziția sursei).

Vom nota acest buffer cu ZS. El conține coordonatele z ale fragmentelor care primesc lumina de la sursa. Mai este numit și « **shadow map** », de unde și denumirea sub care este cunoscută metoda în prezent.

- În etapa a doua se construiește imaginea văzută din poziția observatorului, utilizând buffer-ul ZS pentru determinarea punctelor aflate în umbra.

**În cazul mai multor surse de lumină, se utilizează câte un buffer ZS pentru fiecare sursă.**

Implementarea modernă a metodei este numită “shadow mapping”.

# SHADOW MAPPING (2)

## Etapa I.

Se calculeaza matricea de transformare a varfurilor, MVPS, cu observatorul in pozitia sursei de lumina

Se transmite MVPS la vertex shader (ca variabila uniforma).

Se cere folosirea unui buffer ZS ca buffer de adancime, in locul buffer-ului Z implicit  
(In OpenGL, ZS este implementat ca un buffer textura)

Se cere rasterizarea scenei fara scriere in frame-buffer:

1. In vertex shader se transforma varfurile primitivelor folosind MVPS
2. Se rasterizeaza fiecare primitiva a scenei, cu testul de adancime:

pentru fiecare fragment  $f(x,y,z)$  rezultat din rasterizarea unei primitive

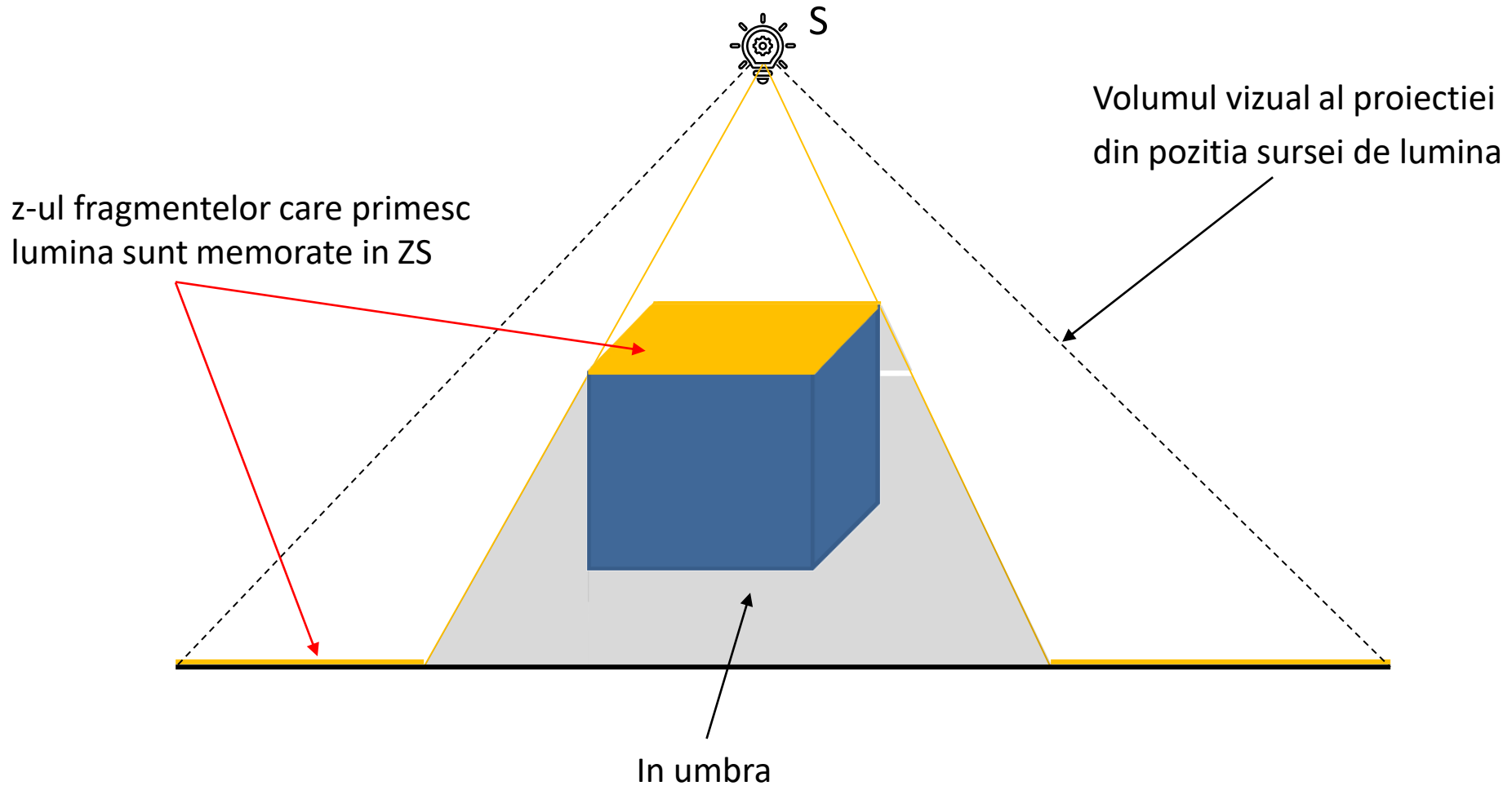
dacă  $z < ZS[y][x]$  atunci

$ZS[y][x] = z$  //fragmentul este vizibil în pixelul  $(x,y)$  din poziția sursei

La sfarsitul acestei etape, in ZS sunt memorate coordonatele z ale fragmentelor care primesc lumina de la sursa.

# SHADOW MAPPING (3)

## Etapa I



# SHADOW MAPPING (4)

## Etapa II.

Se calculeaza matricea de transformare a varfurilor MVPO, cu observatorul in pozitia sa.

Se transmite la vertex shader (ca variabile uniforme) MVPO si MVPS.

Se cere rasterizarea scenei:

1. In vertex shader se calculeaza varful transformat cu MVPO (gl\_Position) si MVPS (**poz\_sursa**)

**poz\_sursa** - pozitia varfului transformata cu MVPS este variabila de iesire din vertex shader

2. Se rasterizeaza fiecare primitiva a scenei, cu testul de adancime:

pentru fiecare fragment  $f(x,y,z)$  rezultat din rasterizarea unei primitive

dacă  $z < Z\text{-Buffer}[y][x]$  atunci //fragment vizibil în pixelul  $(x,y)$ , din poziția observatorului

$\{Z\text{-Buffer}[y][x] = z; \text{frame-Buffer}[y][x] = \text{Culoare fragment (întoarsa de fragment shader)};\}$

In fragment shader

**poz\_sursa** - variabilă de intrare pt fragm shader (poziție fragment in vederea din pozitia sursei)

Culoare\_fragment = culoare calculata folosind modelul de iluminare locală;

dacă  $\text{poz\_sursa}.z > ZS[\text{poz\_sursa}.y][\text{poz\_sursa}.x]$  atunci //fragmentul  $f(x,y,z)$  nu primește lumina

$\text{Culoare\_fragment} = \text{factor\_umbra} * \text{Culoare\_fragment}; (0 < \text{factor\_umbra} < 1)$

# SHADOW MAPPING (5)

Etapa II.

