

# Culori, vârfuri, primitive grafice. Program principal și *shader-e*

Mihai-Sorin Stupariu

Sem. I, 2024 - 2025

Spațiul culorilor

Programul principal: vârfuri

Programul principal: primitive grafice - generalități

Shader-e

Comunicarea dintre programul principal și shader-e

# Spațiul culorilor

- Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) — **Cubul RGB**.

# Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) — **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:

# Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) — **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
  - în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

— “deprecated”

Sufixul \* poate indica:

# Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
  - în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

– “deprecated”

Sufixul \* poate indica:

- dimensiunea  $n$  a spațiului de culori în care lucrăm,  $n = 3$  (RGB) sau  $n = 4$  (RGBA); A=factorul “alpha”, legat de opacitate/transparență.

# Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
  - în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

– “deprecated”

Sufixul \* poate indica:

- dimensiunea  $n$  a spațiului de culori în care lucrăm,  $n = 3$  (RGB) sau  $n = 4$  (RGBA); A=factorul “alpha”, legat de opacitate/transparență.
- tipul de date utilizat, care poate fi:
  - i (integer) ( $i \in \{0, 1, \dots, 255\}$ )
  - f (float)
  - d (double) ( $f, d \in [0.0, 1.0]$ )

# Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:
  - în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

– “deprecated”

Sufixul \* poate indica:

- dimensiunea  $n$  a spațiului de culori în care lucrăm,  $n = 3$  (RGB) sau  $n = 4$  (RGBA); A=factorul “alpha”, legat de opacitate/transparență.
- tipul de date utilizat, care poate fi:
  - i (integer) ( $i \in \{0, 1, \dots, 255\}$ )
  - f (float)
  - d (double) ( $f, d \in [0.0, 1.0]$ )
- (opțional) posibila formă vectorială, indicată prin sufixul v.



# Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.
- ▶ În OpenGL o culoare este indicată prin:

- în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

– “deprecated”

Sufixul \* poate indica:

- dimensiunea  $n$  a spațiului de culori în care lucrăm,  $n = 3$  (RGB) sau  $n = 4$  (RGBA); A=factorul “alpha”, legat de opacitate/transparență.
- tipul de date utilizat, care poate fi:
  - i (integer) ( $i \in \{0, 1, \dots, 255\}$ )
  - f (float)
  - d (double) ( $f, d \in [0.0, 1.0]$ )
- (opțional) posibila formă vectorială, indicată prin sufixul v.
- în OpenGL “nou”: culorile sunt manevrate într-un mod asemănător vârurilor (folosind VBO), odată cu acestea (attribute ale vârurilor).

# Spațiul culorilor

- ▶ Culorile sunt obținute combinând intensitățile de pe trei canale: R (red); G (green); B (blue) – **Cubul RGB**.

- ▶ În OpenGL o culoare este indicată prin:

- în OpenGL “vechi” folosind funcția

```
glColor* ( );
```

– “deprecated”

Sufixul \* poate indica:

- dimensiunea  $n$  a spațiului de culori în care lucrăm,  $n = 3$  (RGB) sau  $n = 4$  (RGBA); A=factorul “alpha”, legat de opacitate/transparență.
- tipul de date utilizat, care poate fi:
  - i (integer) ( $i \in \{0, 1, \dots, 255\}$ )
  - f (float)
  - d (double) ( $f, d \in [0.0, 1.0]$ )
- (opțional) posibila formă vectorială, indicată prin sufixul v.
- în OpenGL “nou”: culorile sunt manevrate într-un mod asemănător vârfurilor (folosind VBO), odată cu acestea (atribute ale vârfurilor).
- elementul comun este faptul că, în ambele cazuri, culorile conțin informații legate de canalele R, G, B, precum și de canalul A (factorul  $\alpha$ , legat de opacitate).

## Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

## Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

- în OpenGL “vechi” cu ajutorul funcției

```
glVertex* ( );
```

— “deprecated”

# Vârfuri - funcții pentru indicare

Primitivele grafice sunt trasate cu ajutorul **vârfurilor** (*entități abstracte, a nu fi confundate cu punctele!*). În OpenGL un vârf este definit:

- în OpenGL “vechi” cu ajutorul funcției

```
glVertex* ( );
```

— “deprecated”

- în OpenGL “nou”:
  - vârfurile sunt stocate în matrice;
  - împachetate în VAO (*Vertex Array Objects*);
  - trimise plăcii grafice sub formă de VBO (*Vertex Buffer Objects*).

O serie întreagă de funcții asociate (exemple mai jos, v. și slide-uri):

```
// Se creeaza un buffer pentru VÂRFURI;
glGenBuffers(1, &VboId);
glBindBuffer(GL_ARRAY_BUFFER, VboId);
glBufferData(GL_ARRAY_BUFFER, sizeof(Vertices), Vertices, GL_STATIC_DRAW);
// Se asociaza atributul (0 = coordonate) pentru shader;
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, 0);
```

# Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:

# Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
  - ▶ coordonate (fac parte din definiție),

# Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
  - ▶ coordonate (fac parte din definiție),
  - ▶ o culoare (v. secțiunea Culori...),



# Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
  - ▶ coordonate (fac parte din definiție),
  - ▶ o culoare (v. secțiunea Culori...),
  - ▶ o normală (legată de funcții de iluminare),

# Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
  - ▶ coordonate (fac parte din definiție),
  - ▶ o culoare (v. secțiunea Culori...),
  - ▶ o normală (legată de funcții de iluminare),
  - ▶ coordonate de texturare

# Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
  - ▶ coordonate (fac parte din definiție),
  - ▶ o culoare (v. secțiunea Culori...),
  - ▶ o normală (legată de funcții de iluminare),
  - ▶ coordonate de texturare
- ▶ În OpenGL “vechi”: pentru o anumită caracteristică, este considerată valoarea *curentă* a respectivei caracteristici. Altfel spus, ea trebuie indicată în codul sursă înaintea vârfului.

# Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
  - ▶ coordonate (fac parte din definiție),
  - ▶ o culoare (v. secțiunea Culori...),
  - ▶ o normală (legată de funcții de iluminare),
  - ▶ coordonate de texturare
- ▶ În OpenGL “vechi”: pentru o anumită caracteristică, este considerată valoarea *curentă* a respectivei caracteristici. Altfel spus, ea trebuie indicată în codul sursă înaintea vârfului.
- ▶ În OpenGL “modern”: pentru o anumită caracteristică, este generat un *buffer object*, la fel ca pentru coordonatele vârfurilor. Alternativ, caracteristica respectivă poate fi indicată în aceeași matrice cu coordonatele vârfurilor.

# Caracteristici ale vârfurilor

- ▶ Unui vârf îi sunt asociate:
  - ▶ coordonate (fac parte din definiție),
  - ▶ o culoare (v. secțiunea Culori...),
  - ▶ o normală (legată de funcții de iluminare),
  - ▶ coordonate de texturare
- ▶ În OpenGL “vechi”: pentru o anumită caracteristică, este considerată valoarea *curentă* a respectivei caracteristici. Altfel spus, ea trebuie indicată în codul sursă înaintea vârfului.
- ▶ În OpenGL “modern”: pentru o anumită caracteristică, este generat un *buffer object*, la fel ca pentru coordonatele vârfurilor. Alternativ, caracteristica respectivă poate fi indicată în aceeași matrice cu coordonatele vârfurilor.
- ▶ Caracteristicile vârfurilor, indicate în programul principal, se regăsesc (sub un format adecvat) și în *shader*-ul de vârfuri.

## Funcții pentru primitive

Vârfurile sunt utilizate pentru trasarea primitivelor grafice. Funcțiile folosite sunt diferite pentru cele două moduri de randare:

- în OpenGL “vechi”, o funcție de tipul `glVertex ( )` poate fi apelată într-un cadru de tip

```
glBegin (*);
```

```
glEnd;
```

— “deprecated”

(unde \* reprezintă tipul de primitivă generat);

## Funcții pentru primitive

Vârfurile sunt utilizate pentru trasarea primitivelor grafice. Funcțiile folosite sunt diferite pentru cele două moduri de randare:

- în OpenGL “vechi”, o funcție de tipul `glVertex ( )` poate fi apelată într-un cadru de tip

```
glBegin (*);
```

```
glEnd;
```

— “deprecated”

(unde \* reprezintă tipul de primitivă generat);

- în OpenGL “nou” este utilizată o funcție de tipul

```
glDrawArrays (GLenum mode, GLint first, GLint count);
```

unde

mode: tipul primitivei;

first: primul vârf;

count: câte vârfuri se iau în considerare.

## Tipuri de primitive (corespund lui GLenum mode)

- ▶ Puncte: GL\_POINTS



## Tipuri de primitive (corespund lui GLenum mode)

- ▶ Puncte: GL\_POINTS
- ▶ Segmente de dreaptă: GL\_LINES, GL\_LINE\_STRIP, GL\_LINE\_LOOP

## Tipuri de primitive (corespund lui GLenum mode)

- ▶ Puncte: GL\_POINTS
- ▶ Segmente de dreaptă: GL\_LINES, GL\_LINE\_STRIP, GL\_LINE\_LOOP
- ▶ Triunghiuri: GL\_TRIANGLES, GL\_TRIANGLE\_STRIP, GL\_TRIANGLE\_FAN

## Tipuri de primitive (corespund lui GLenum mode)

- ▶ Puncte: GL\_POINTS
- ▶ Segmente de dreaptă: GL\_LINES, GL\_LINE\_STRIP, GL\_LINE\_LOOP
- ▶ Triunghiuri: GL\_TRIANGLES, GL\_TRIANGLE\_STRIP, GL\_TRIANGLE\_FAN
- ▶ Dreptunghiuri: GL\_QUADS, GL\_QUAD\_STRIP

## Tipuri de primitive (corespund lui GLenum mode)

- ▶ Puncte: `GL_POINTS`
- ▶ Segmente de dreaptă: `GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`
- ▶ Triunghiuri: `GL_TRIANGLES`, `GL_TRIANGLE_STRIP`,  
`GL_TRIANGLE_FAN`
- ▶ Dreptunghiuri: `GL_QUADS`, `GL_QUAD_STRIP`
- ▶ Poligoane (convexe!): `GL_POLYGON`

## Despre GLSL și obiecte-program

- ▶ **GLSL** este un limbaj de nivel înalt, cu sintaxă asemănătoare limbajului C: detalii în [specificațiile GLSL](#) - sunt prezentate funcțiile standard utilizate.

# Despre GLSL și obiecte-program

- ▶ **GLSL** este un limbaj de nivel înalt, cu sintaxă asemănătoare limbajului C: detalii în **specificațiile** GLSL - sunt prezentate funcțiile standard utilizate.
- ▶ Avantaje: utilizare pe platforme multiple, compatibilitate cu diverse tipuri de plăci grafice.
- ▶ Motivație: de a controla banda grafică (*graphics pipeline*). Finalitate: obținerea unui obiect-program (*Program Object*) care să fie rulat pe placa grafică.

## Despre GLSL și obiecte-program

- ▶ **GLSL** este un limbaj de nivel înalt, cu sintaxă asemănătoare limbajului C: detalii în **specificațiile** GLSL - sunt prezentate funcțiile standard utilizate.
- ▶ Avantaje: utilizare pe platforme multiple, compatibilitate cu diverse tipuri de plăci grafice.
- ▶ Motivație: de a controla banda grafică (*graphics pipeline*). Finalitate: obținerea unui obiect-program (*Program Object*) care să fie rulat pe placa grafică.
- ▶ Pentru a genera obiectul-program, trebuie parcurse o serie de etape prin intermediul programului principal - aplicația instalează codul GLSL pe placa grafică. Practic: v. *slide*-ul următor.

# Despre GLSL și obiecte-program

- ▶ **GLSL** este un limbaj de nivel înalt, cu sintaxă asemănătoare limbajului C: detalii în **specificațiile** GLSL - sunt prezentate funcțiile standard utilizate.
- ▶ Avantaje: utilizare pe platforme multiple, compatibilitate cu diverse tipuri de plăci grafice.
- ▶ Motivație: de a controla banda grafică (*graphics pipeline*). Finalitate: obținerea unui obiect-program (*Program Object*) care să fie rulat pe placa grafică.
- ▶ Pentru a genera obiectul-program, trebuie parcurse o serie de etape prin intermediul programului principal - aplicația instalează codul GLSL pe placa grafică. Practic: v. *slide*-ul următor.
- ▶ Codul este efectiv rulat la apelarea `glDrawArrays()` sau a unei funcții de desenare similare.



## Despre *shader*-e

- ▶ Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)
```

din fișierul `LoadShaders.cpp`

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

## Despre *shader*-e

- ▶ Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)
```

din fișierul `LoadShaders.cpp`

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

- ▶ Fluxul:

## Despre *shader*-e

- ▶ Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)
```

din fișierul `LoadShaders.cpp`

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

- ▶ Fluxul:
  - (dacă este cazul) obținerea codului din fișier și transformarea în *string*

## Despre *shader*-e

- ▶ Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)
```

din fișierul `LoadShaders.cpp`

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificator al obiectului-program generat.

- ▶ Fluxul:
  - (dacă este cazul) obținerea codului din fișier și transformarea în *string*
  - crearea unui obiect *shader* `glCreateShader()`, încărcarea *string*-ului în obiectul *shader* `glShaderSource()` și compilarea `glCompileShader()` - pas realizat separat pentru fiecare tip de *shader*

## Despre *shader*-e

- ▶ Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)
```

din fișierul `LoadShaders.cpp`

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificador al obiectului-program generat.

- ▶ Fluxul:
  - (dacă este cazul) obținerea codului din fișier și transformarea în *string*
  - crearea unui obiect *shader* `glCreateShader()`, încărcarea *string*-ului în obiectul *shader* `glShaderSource()` și compilarea `glCompileShader()` - pas realizat separat pentru fiecare tip de *shader*
  - crearea unui obiect-program `glCreateProgram()`, atașarea obiectelor *shader* `compile` `glAttachShader()` și legarea programului `glLinkProgram()`

## Despre *shader*-e

- ▶ Manevrare: în *template*-ul pus la dispoziție, etapele specifice sunt incluse în funcția

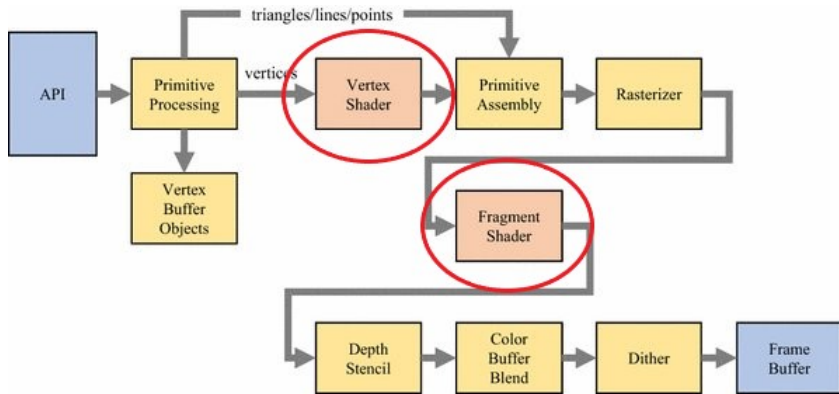
```
GLuint LoadShaders(const char *vertex_file_path,const char *fragment_file_path)
```

din fișierul `LoadShaders.cpp`

Ca date de intrare: calea către *shader*-ele de vârfuri, respectiv fragment. Funcția returnează un identificador al obiectului-program generat.

- ▶ Fluxul:
  - (dacă este cazul) obținerea codului din fișier și transformarea în *string*
  - crearea unui obiect *shader* `glCreateShader()`, încărcarea *string*-ului în obiectul *shader* `glShaderSource()` și compilarea `glCompileShader()` - pas realizat separat pentru fiecare tip de *shader*
  - crearea unui obiect-program `glCreateProgram()`, atașarea obiectelor *shader* `compile` `glAttachShader()` și legarea programului `glLinkProgram()`
  - utilizarea obiectului-program `glUseProgram()` - poate fi apelată inclusiv în funcția de desenare.

# “Programmable pipeline”



Sursa: Baek and Kim, 2019

## Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimite mai departe în banda grafică.



## Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimite mai departe în banda grafică.
- ▶ La apelarea unei funcții de desenare (de ex. `glDrawArrays()`) *shader*-ul de vârfuri procesează, unul câte unul (eventual în paralel) vârfurile din *VBO*.

## Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimite mai departe în banda grafică.
- ▶ La apelarea unei funcții de desenare (de ex. `glDrawArrays()`) *shader*-ul de vârfuri procesează, unul câte unul (eventual în paralel) vârfurile din *VBO*.
- ▶ Date de intrare / ieșire: informații despre vârfuri (poziție, culoare, etc.) și o serie de alte date (de ex. matrice ale transformărilor). În particular, în *shader*-ul de vârfuri pot fi aplicate diverse transformări (inclusiv pentru mișcare) și poate fi controlată geometria scenei. De asemenea, mai există variabilele uniforme, preluate din programul principal.

## Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimite mai departe în banda grafică.
- ▶ La apelarea unei funcții de desenare (de ex. `glDrawArrays()`) *shader*-ul de vârfuri procesează, unul câte unul (eventual în paralel) vârfurile din *VBO*.
- ▶ Date de intrare / ieșire: informații despre vârfuri (poziție, culoare, etc.) și o serie de alte date (de ex. matrice ale transformărilor). În particular, în *shader*-ul de vârfuri pot fi aplicate diverse transformări (inclusiv pentru mișcare) și poate fi controlată geometria scenei. De asemenea, mai există variabilele uniforme, preluate din programul principal.
- ▶ Datele furnizate de *shader*-ul de vârfuri trec printr-un proces de asamblare a primitivelor și de rasterizare, fiind transformate în fragmente (locații ale pixelilor) - proces realizat de API.

## Shader-ul de vârfuri (*vertex shader*)

- ▶ Rol: procesează vârfurile (și informațiile despre acestea) și le trimite mai departe în banda grafică.
- ▶ La apelarea unei funcții de desenare (de ex. `glDrawArrays()`) *shader*-ul de vârfuri procesează, unul câte unul (eventual în paralel) vârfurile din *VBO*.
- ▶ Date de intrare / ieșire: informații despre vârfuri (poziție, culoare, etc.) și o serie de alte date (de ex. matrice ale transformărilor). În particular, în *shader*-ul de vârfuri pot fi aplicate diverse transformări (inclusiv pentru mișcare) și poate fi controlată geometria scenei. De asemenea, mai există variabilele uniforme, preluate din programul principal.
- ▶ Datele furnizate de *shader*-ul de vârfuri trec printr-un proces de asamblare a primitivelor și de rasterizare, fiind transformate în fragmente (locații ale pixelilor) - proces realizat de API.
- ▶ Mai multe detalii: ulterior (de ex. texturare, modele de iluminare, etc.)

## Shader-ul de fragment (*fragment shader*)

- ▶ Rol: procesarea fragmentelor și stabilirea culorii pentru fiecare pixel din buffer-ul de cadru. Sunt realizate operații la nivel de fragment (de ex. amestecarea culorilor, combinarea cu texturile, etc.)

## Shader-ul de fragment (*fragment shader*)

- ▶ Rol: procesarea fragmentelor și stabilirea culorii pentru fiecare pixel din buffer-ul de cadru. Sunt realizate operații la nivel de fragment (de ex. amestecarea culorilor, combinarea cu texturile, etc.)
- ▶ Date de intrare: provin din *shader*-ul de vârfuri. Date de ieșire: legate de culoarea fragmentelor. De asemenea, mai există variabilele uniforme, preluate din programul principal.

## Shader-ul de fragment (*fragment shader*)

- ▶ Rol: procesarea fragmentelor și stabilirea culorii pentru fiecare pixel din buffer-ul de cadru. Sunt realizate operații la nivel de fragment (de ex. amestecarea culorilor, combinarea cu texturile, etc.)
- ▶ Date de intrare: provin din *shader*-ul de vârfuri. Date de ieșire: legate de culoarea fragmentelor. De asemenea, mai există variabilele uniforme, preluate din programul principal.
- ▶ Mai multe detalii: ulterior (de ex. texturare, modele de iluminare, etc.)

# Elemente de principiu - comunicarea dintre progr. princ. și *shader-e*

Două moduri de comunicare

- ▶ Prin intermediul *buffer*-elor (vârfuri și caracteristici asociate) - C++ și al *vertex attributes* - *shader*.
- ▶ Folosind variabile uniforme.



Pentru vârfuri și caracteristici ale acestora - prog. princ.

► Pași realizați:

## Pentru vârfuri și caracteristici ale acestora - prog. princ.

### ► Pași realizați:

- creare vectori cu vârfuri, caracteristici (culoare, etc.) - pot fi indicate separat sau în aceeași matrice, indici;
- generare nume ptr. buffer-objects: `glGenBuffers( )`;
- activare/"legare" buffer `glBindBuffer( )` și transfer de date în buffer: `glBufferData( )`.

## Pentru vârfuri și caracteristici ale acestora - prog. princ.

### ► Pași realizați:

- creare vectori cu vârfuri, caracteristici (culoare, etc.) - pot fi indicate separat sau în aceeași matrice, indici;
- generare nume ptr. buffer-objects: `glGenBuffers( )`;
- activare/"legare" buffer `glBindBuffer( )` și transfer de date în buffer: `glBufferData( )`.
- asociere cu un *vertex attribute* și indicarea locațiilor (ptr. *shader!*): `glVertexAttribPointer( )` - parametrul `GLuint index` trebuie să se regăsească, cu aceeași valoare, și în *shader*,
- activare a *vertex attribute*: `glEnableVertexAttribArray( )` - parametrii `GLuint index` și `GLint size` trebuie să se regăsească, cu aceeași valoare, și în *shader*,

- Structura specifică pentru *buffer*: *VBO (Vertex Buffer Object)*. Pot fi utilizate mai multe *VBO* (inclusiv pentru vârfuri).

## Pentru vârfuri și caracteristici ale acestora - prog. princ.

### ► Pași realizați:

- creare vectori cu vârfuri, caracteristici (culoare, etc.) - pot fi indicate separat sau în aceeași matrice, indici;
- generare nume ptr. buffer-objects: `glGenBuffers( )`;
- activare/"legare" buffer `glBindBuffer( )` și transfer de date în buffer: `glBufferData( )`.
- asociere cu un *vertex attribute* și indicarea locațiilor (ptr. *shader*!): `glVertexAttribPointer( )` - parametrul `GLuint index` trebuie să se regăsească, cu aceeași valoare, și în *shader*,
- activare a *vertex attribute*: `glEnableVertexAttribArray( )` - parametrii `GLuint index` și `GLint size` trebuie să se regăsească, cu aceeași valoare, și în *shader*,
- apelare funcție de desenare.

► Structura specifică pentru *buffer*: *VBO (Vertex Buffer Object)*. Pot fi utilizate mai multe *VBO* (inclusiv pentru vârfuri).

► O structură asociată: *VAO (Vertex Array Object)* - are rolul de a permite gestionarea eficientă și organizarea *VBO* - necesar cel puțin un *VAO*. Funcții: `glGenVertexArrays`, `glBindVertexArray( )`.

## Pentru vârfuri și caracteristici ale acestora - *shader*

- ▶ În *shader*-ul de vârfuri:

## Pentru vârfuri și caracteristici ale acestora - *shader*

- ▶ În *shader*-ul de vârfuri:
  - Informațiile despre vârfuri sunt transferate în *shader* ca *vertex attributes*, aici devenind valori de intrare (*input*). Parametrii *index* și *size* din funcția `glEnableVertexAttribArray( )` se regăsesc (cu aceleași valori) apelați în `location = index`, respectiv `vec*`, unde `*=size`. Exemplu:  

```
layout (location = 0) in vec4 in_Position;  
layout (location = 1) in vec3 in_Color;
```

Indexul 0 corespunde unui vector cu 4 componente (coordonate).  
Indexul 1 corespunde unui vector cu 3 componente (culori).

## Pentru vârfuri și caracteristici ale acestora - *shader*

### ► În *shader*-ul de vârfuri:

- Informațiile despre vârfuri sunt transferate în *shader* ca *vertex attributes*, aici devenind valori de intrare (*input*). Parametrii *index* și *size* din funcția `glEnableVertexAttribArray( )` se regăsesc (cu aceleași valori) apelați în `location = index`, respectiv `vec*`, unde `*=size`. Exemplu:

```
layout (location = 0) in vec4 in_Position;
```

```
layout (location = 1) in vec3 in_Color;
```

Indexul 0 corespunde unui vector cu 4 componente (coordonate).

Indexul 1 corespunde unui vector cu 3 componente (culori).

- Trebuie să existe o corespondență perfectă între valorile din programul principal și cele din *shader*, altminteri efectul nu va fi cel dorit.

## Variabile uniforme

- ▶ În programul principal:



## Variabile uniforme

► În programul principal:

- obținerea unei referințe către variabila uniformă

`glGetUniformLocation(GLuint program, const GLchar *name)`

Variabila `program` și string-ul cu numele variabilei trebuie modificate.

Exemplu:

```
codColLocation = glGetUniformLocation(ProgramId,  
"codColShader");
```

Această funcție poate fi apelată și în `init`

## Variabile uniforme

► În programul principal:

- obținerea unei referințe către variabila uniformă

`glGetUniformLocation(GLuint program, const GLchar *name)`

Variabila `program` și string-ul cu numele variabilei trebuie modificate.

Exemplu:

```
codColLocation = glGetUniformLocation(ProgramId,
"codColShader");
```

Această funcție poate fi apelată și în `init`

- specificarea valorii variabilei `glUniform*()` Sufixul `*` indică informații despre variabilă. De asemenea, trebuie indicată locația utilizată.

**Această funcție trebuie apelată la fiecare modificare a valorii variabilei.**

Exemplu:

```
int codCol = 0;
glUniform1i(codColLocation, codCol);
```

## Variabile uniforme

► În programul principal:

- obținerea unei referințe către variabila uniformă

`glGetUniformLocation(GLuint program, const GLchar *name)`

Variabila `program` și string-ul cu numele variabilei trebuie modificate.

Exemplu:

```
codColLocation = glGetUniformLocation(ProgramId,
"codColShader");
```

Această funcție poate fi apelată și în `init`

- specificarea valorii variabilei `glUniform*()` Sufixul `*` indică informații despre variabilă. De asemenea, trebuie indicată locația utilizată.

**Această funcție trebuie apelată la fiecare modificare a valorii variabilei.**

Exemplu:

```
int codCol = 0;
glUniform1i(codColLocation, codCol);
```

► În *shader*:

## Variabile uniforme

### ► În programul principal:

- obținerea unei referințe către variabila uniformă

`glGetUniformLocation(GLuint program, const GLchar *name)`

Variabila `program` și string-ul cu numele variabilei trebuie modificate.

Exemplu:

```
codColLocation = glGetUniformLocation(ProgramId,
"codColShader");
```

Această funcție poate fi apelată și în `init`

- specificarea valorii variabilei `glUniform*()` Sufixul `*` indică informații despre variabilă. De asemenea, trebuie indicată locația utilizată.

**Această funcție trebuie apelată la fiecare modificare a valorii variabilei.**

Exemplu:

```
int codCol = 0;
glUniform1i(codColLocation, codCol);
```

### ► În *shader*:

- declararea variabilei uniforme și a tipului acesteia.

Exemplu:

```
uniform int codColShader
```