



Računarska grafika

Generacija 2023/2024

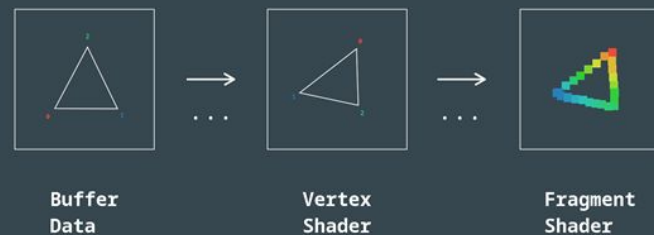
2. Sedmica

Još jednom: *OpenGL je automat stanja sa klijent-server arhitekturom.*

Četiri šejdera su programabilna u rendering pajplajnu OpenGL-a:

- Vertex
 - (Većinom) za obradu prostornih podataka
- ~~Geometry~~
- ~~Tessellation~~
- Fragment
 - (Većinom) za “farbanje piksela”

Pajplajn





Specifikacija tjemena (Vertex specification)

- VertexBufferObject - bafer tjemena koji se čuva na GPU
- VertexArrayObject - objekat u kom se čuva jedan ili više VBO - a

Iscrtavanje putem VAO i VBO takođe nudi značajno bolje performanse u odnosu na immediate - mode rendering(iscrtavanje).

Unutar VBO možemo čuvati podatke o poziciji, boji, normalni, itd...

VAO - primer 0





```
//Primjer VAO
float Vertices[] = {
    //      position      |      colour
    // X   Y       Z       R       G       B
    -0.5f, -0.5f, 0.0f, 0.905f, 0.298f, 0.235f, // v0
    0.5f, -0.5f, 0.0f, 0.180f, 0.8f,   0.443f, // v1
    -0.5f,  0.5f, 0.0f, 0.160f, 0.501f, 0.725f, // v2

    -0.5f,  0.5f, 0.0f, 0.160f, 0.501f, 0.725f, // v2
    0.5f, -0.5f, 0.0f, 0.180f, 0.8f,   0.443f, // v1
    0.5f,  0.5f, 0.0f, 0.945f, 0.768f, 0.058f, // v3
};

// Generate VBOs
unsigned VBOs[2];
int Stride = 6 * sizeof(float);
glGenBuffers(2, VBOs);
// Bind first VBO to GL_ARRAY_BUFFER and load data into it
glBindBuffer(GL_ARRAY_BUFFER, VBOs[0]);
glBufferData(GL_ARRAY_BUFFER, Vertices, GL_STATIC_DRAW);
// Set pointers to buffer segments and enable them
glVertexAttribPointer(0, 3, GL_FLOAT, false, Stride, (void*)(0));
glVertexAttribPointer(1, 3, GL_FLOAT, false, Stride, (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(0); glEnableVertexAttribArray(1);
...
// Drawing
glEnableClientState(GL_VERTEX_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, m_modelsVBO[0]);
glDrawArrays(GL_TRIANGLES, 0, ArrayCount(Vertices) / 6);
```

VAO



VBO 0

`pos[0], col[0], pos[1], col[1], ...`


VBO 1

`nrm[0], nrm[1], nrm[2], nrm[3], ...`

VAO - primer 1

```
uint Indices = {  
    0, 1, 2,  
    2, 1, 3  
};
```





```
// Crtanje tjemena preko indeksa (EBO)
// . . . VBO generation, buffering, vertex attrib. pointers...
// Bind second VBO(EBO) to GL_ELEMENT_ARRAY_BUFFER and load indices
into it
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, VBOs[1]);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, Indices, GL_STATIC_DRAW);
. . .
// Drawing
glEnableClientState(GL_VERTEX_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER, VBOs[0]);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, VBOs[1]);
glDrawElements(GL_TRIANGLES, ArrayCount(Vertices) / 6, GL_UNSIGNED_INT,
(void*)0);
```


VAO



VB0 0

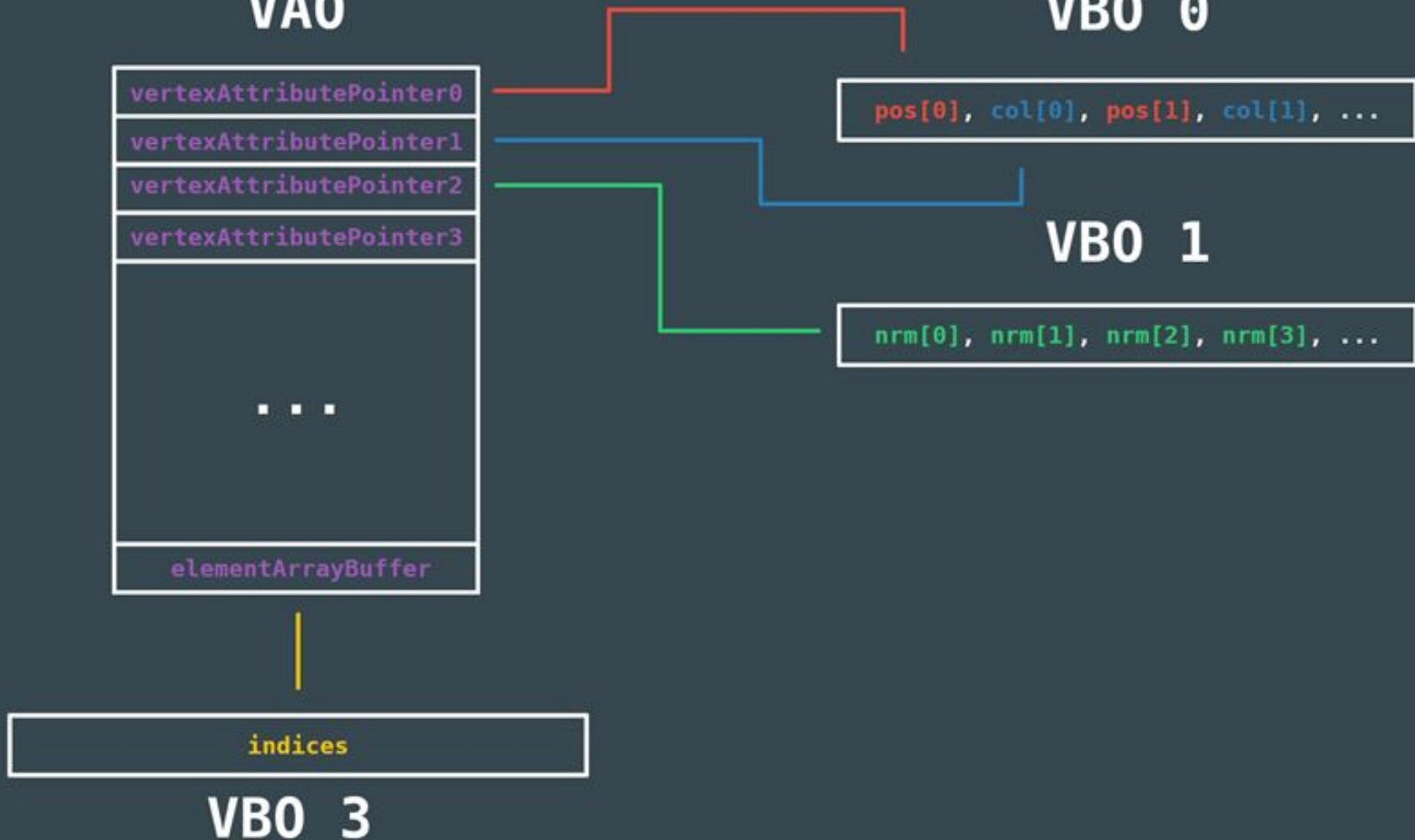
`pos[0], col[0], pos[1], col[1], ...`

VB0 1

`nrm[0], nrm[1], nrm[2], nrm[3], ...`

`indices`

VB0 3

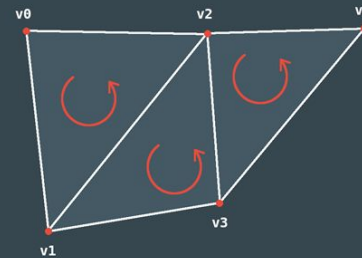
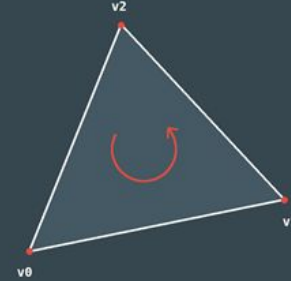


Poligoni i primitive

Osnovni poligon u računarskoj grafici jeste trougao (**GL_TRIANGLES**). Lice poligona određeno je normalom čiji pravac zavisi od redosleda definisanja temena i određuje se pravilom desne ruke (moguće menjati).

Nadovezani trouglovi (**GL_TRIANGLE_STRIP**) na slici desno je iscrtan je u redosledu: v0, v1, v2, pa zatim v2, v1, v3, pa v2, v3, v4 ...

Postoje i **GL_POINTS**, **GL_LINES**, **GL_LINE_STRIP**, **GL_LINE_LOOP**, **GL_TRIANGLES_FAN** i dr.





Unos sa tastature

Preko callback funkcije:

```
static void  
KeyCallback(GLFWwindow* window, int key, int scancode, int action, int mode) {  
    bool IsDown = action == GLFW_PRESS || action == GLFW_REPEAT;  
    switch (key) {  
        case GLFW_KEY_ESCAPE: glfwSetWindowShouldClose(window, GLFW_TRUE); break;  
    }  
}
```

Bez callback funkcije:

```
if (glfwGetKey(Window, GLFW_KEY_ESCAPE) == GLFW_PRESS) glfwSetWindowShouldClose(Window, GLFW_TRUE);
```



Ograničavanje brzine crtanja

Bez ikakvih granica, OpenGL će se izvršavati najbrže što mu hardver mašine dozvoljava.

Razmotriti zašto je to loše i osmisliti način da se OpenGL program izvršava sa podesivim brojem renderovanih slika u sekundi (Frame Per Second) koristeći `glfwGetTime` funkciju.