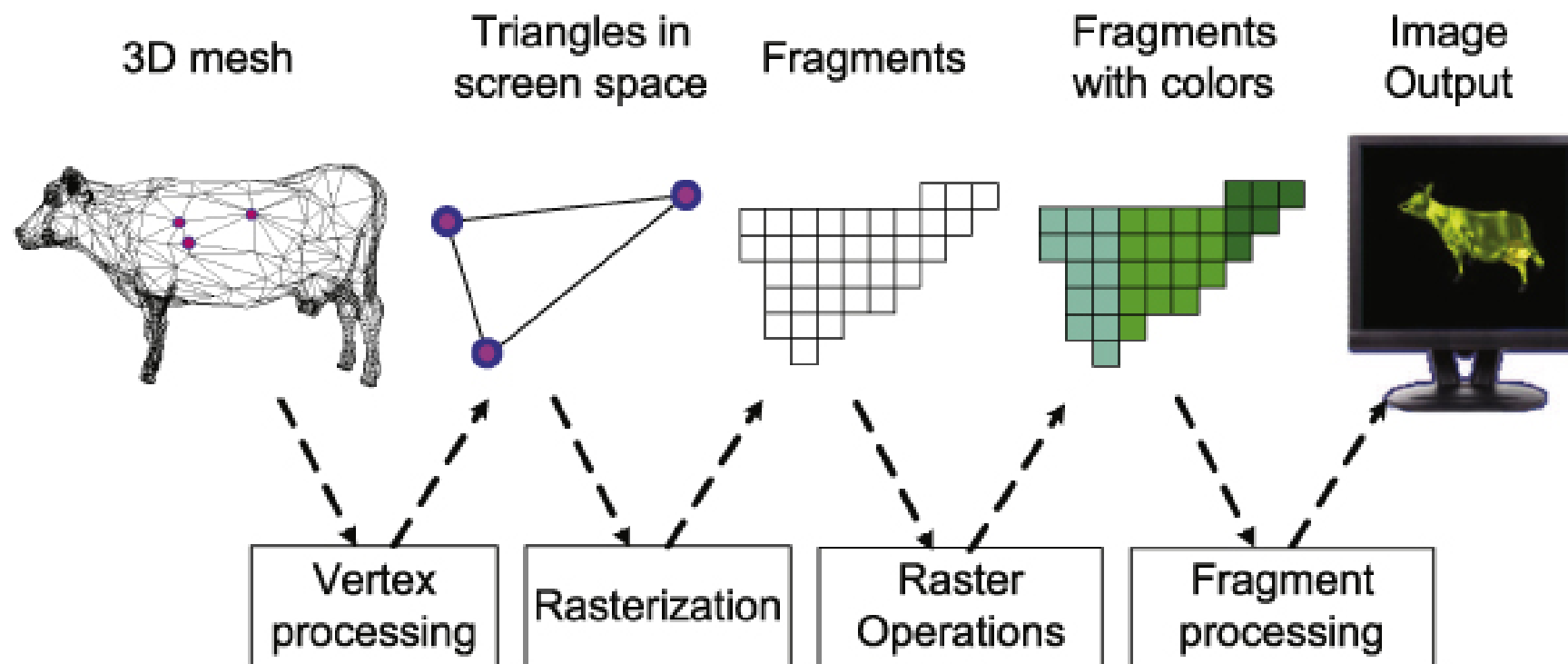


Koraki grafičnega cevovoda




<http://iopscience.iop.org/1742-5468/2009/06/P06016/fulltext/>

- Glavna naloga: Tvorba 2D slike iz 3D geometrijskih podatkov
- Fragment \leftarrow piksel
 - En fragment lahko prekrije drugi fragment (depth testing, več fragmentov tvori en piksel (antialiasing)).

OpenGL

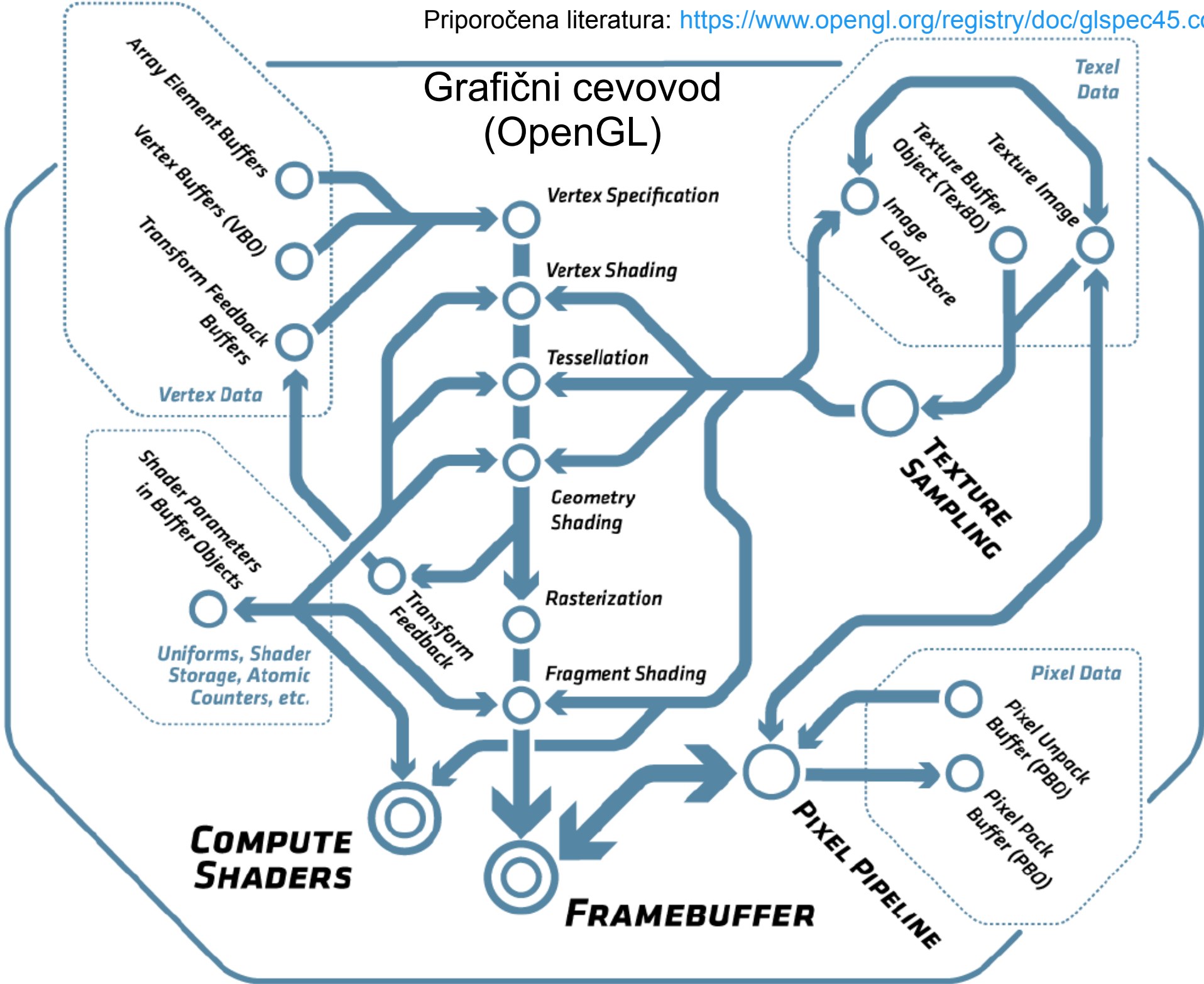
- Najbolj razširjena tehnologija - API za dostop do grafičnega cevovoda (GPU)
 - Za programski jezik C → dostopen v vseh ostalih programskih jezikih
 - Enostaven za uporabo (npr. proti Vulkan/Direct3D12)
- Neodvisen od platforme (Windows/Linux/macOS, Android – OpenGL ES,....)
- Ponavadi priložen zraven gonilnikov grafične kartice
 - <http://www.mesa3d.org/> (odprto-kodni gonilniki in tudi emulator za OpenGL na CPU)
- Možna tudi uporaba na platformah, ki nimajo podpore za OpenGL (npr. Xbox)
<https://docs.mesa3d.org/drivers/zink.html>,
<https://github.com/google/angle>
- <https://www.opengl.org/wiki/>
- <https://www.khronos.org/opengl/>

Zgodovinski pregled

- https://www.opengl.org/wiki/History_of_OpenGL
- 1992, 1.0 → evolucija, dodajanje novih funkcionalnosti
 - Risanje (glBegin, glEnd)
 - Fiksen cevovod 
• Nastavljanje izrisa (premik točk, osvetlitveni model, senčenje, luči, megla,...) klici funkcij
 - 2003, OpenGL 2.0
 - Senčilniki: programirljivost delov grafičnega cevovoda (senčilnik oglišč in fragmentov) - GLSL
- 2008, OpenGL 3.0
 - Problem z OpenGL 1.0 → 2.x: veliko /funkcij za izris (vklop/izklop luči, ...) → počasnost, API overhead
 - Fiksen cevovod odsvetovan (deprecated) kot tudi neučinkovit način izrisa (glBegin, glEnd)
- 2009, OpenGL 3.1
 - Fiksen cevovod odstranjen in premaknjen v **opcijsko** razširitev: GL_ARB_compatibility (ni na voljo na Mac OSX)
- 2009, OpenGL 3.2
 - **Core** profile
 - Compatibility profile: optional
- 2014, OpenGL 4.5
- 2015/2016, Vulkan
 - Nov vmesnik, učinkovitejši vendar **težji** za uporabo
- 2017, OpenGL 4.6

```
glBegin(GL_TRIANGLES);  
glColor3f(r1, g1, b1);  
glVertex3f(v1.x, v1.y, v1.z);  
glColor3f(r2, g2, b2);  
glVertex3f(v2.x, v2.y, v2.z);  
glColor3f(r3, g3, b3);  
glVertex3f(v3.x, v3.y, v3.z);  
glEnd();
```

Grafični cevovod (OpenGL)



OpenGL osnovni koncept

- Opis stanja grafičnega cevovoda s parametri
- Nadziranje grafičnega cevovoda s funkcijami (nastavljanje parametrov)
 - `glEnable(GL_DEPTH_TEST);`
 - Bližnji objekti prekrijejo bolj oddaljene
 - `glDisable(GL_DEPTH_TEST);`
 - Nazadnje narisani objekti prekrijejo obstoječe objekte
 - `glEnable(GL_CULL_FACE);`
 - Vidna naj bo samo ena stran trikotnika
 - ...
- Uporaba objektov (kompleksnejših podatkovnih struktur)
 - **Senčilni programi** (koda, ki se izvede na GPU), **buffer** (npr. medpomnilnik geometrijskih podatkov), **teksture** (slike, ki jih lepimo na objekte),
 - Vsebujejo stanje/parametre in podatke
 - Dostopni preko številke (ID)

Delo z objekti (npr. medpomnilnik)

- https://www.opengl.org/wiki/OpenGL_Object
- Kreiranje
 - void **glGenBuffers**(GLsizei n, GLuint * buffers); // *n* objektov, rezultat: ID-ji objektov v buffers
- Uporaba
 - void **glBindBuffer**(GLenum target, GLuint buffer); // priklopimo za uporabo
 - ...
 - void **glBufferData**(GLenum target, GLsizeiptr size, const GLvoid * data, GLenum usage); // primer: nalaganje podatkov v buffer na GPU
 - void **glDrawArrays**(GLenum mode, GLint first, GLsizei count);
- Brisanje - sprostitvev pomnilnika
 - void **glDeleteBuffers**(GLsizei n, const GLuint * buffers);

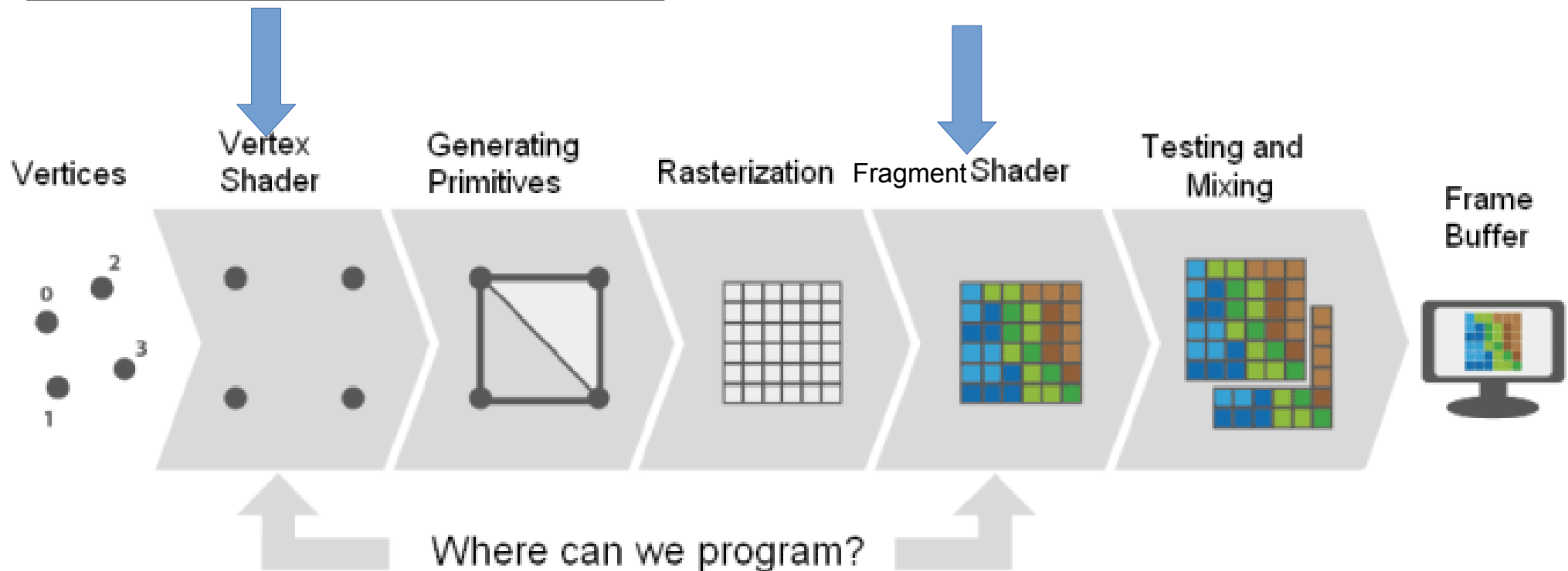
Programski jezik GLSL

- Preprost programski jezik za programiranje posameznih stopenj grafičnega cevovoda - senčilnikov: kaj se zgodi z oglišči, fragmenti, ...
- Podoben C-ju, razen:
 - Vgrajeni tipi: `vec3`, `mat4`
 - Ni kazalcev, `#include`, nizov, ...
- Vsak senčilni program ima vstopno točko: `void main(){...}`
- Podpora za delo z matrikami in vektorji
- Ponavadi se izvaja na GPU (grafična procesna enota)!

GLSL – umestitev znotraj grafičnega cevovoda

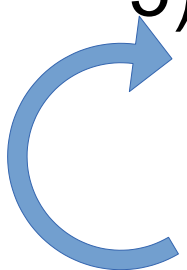
```
#version 330
layout(location=0) in vec3 in_Pos;
uniform mat4 PVM;
void main(){
    gl_Position = PVM * vec4(in_Pos, 1);
}
```

```
#version 330
out vec4 out_Color;
void main(){
    out_Color = vec4(0, 1, 0, 1);
}
```



Pregled implementacije grafične aplikacije

- 1) Inicializacija površine za izris v grafični aplikaciji in konteksta od OpenGL
 - Definiramo katero različico OpenGL uporabljamo, double buffering, multisampling, ...
- 2) Dinamično nalaganje funkcij, saj vse funkcije niso dostopne na vseh platformah oz. različicah OpenGL
 - Odvisno tudi od podprtih razširitev
- 3) Prevajanje senčilnikov
- 4) Nalaganje geometrijskih podatkov in tekstur v pomnilnik
- 5) Izris
 - 1) Priklop senčilnikov (bind)
 - 2) Priklop dela pomnilnika (vertex array)
 - 3) Izris (npr. `glDrawArrays`)



Implementacija – osnovno okno knjižnice

- ~~Uporaba sistemskih klicev operacijskega sistema:~~
 - [https://www.khronos.org/opengl/wiki/Creating_an_OpenGL_Context_\(WGL\)](https://www.khronos.org/opengl/wiki/Creating_an_OpenGL_Context_(WGL))
 - <https://en.wikipedia.org/wiki/GLX>
- Osnovno okno (tudi vhod):
 - <http://www.glfw.org/>
 - <http://freeglut.sourceforge.net/>
- Osnovno okno + dodatki (zvok,...):
 - <http://www.sfml-dev.org/>
 - <http://www.libsdl.org/index.php>
 - <http://liballeg.org/>
- Splošna gradnja grafičnih vmesnikov
 - <http://www.qt.io/developers/>
 - <http://www.wxwidgets.org/>
 - <http://www.fltk.org/index.php>

Java in C#?

- Potrebna knjižnica, ki omogoča dostop do OpenGL (wrapper library)
 - Java → C
- https://www.opengl.org/wiki/Language_bindings
- <http://jogamp.org/>
- <http://jogamp.org/jogl/www/>
- <http://www.opentk.com/>
- Tudi za Python, Lisp...

Implementacija - nalaganje funkcij

- Različne platforme (GPU + OS) imajo podporo za različne funkcionalnosti OpenGL (3.3, 4.5, ...)
 - Nekatere funkcije OpenGL niso dostopne, zato se naložijo ob zagonu aplikacije glede na podprtost
- Knjižnice
 - <http://glew.sourceforge.net/>
 - <https://github.com/skaslev/gl3w>
 - Qt
 - Že vsebuje nalagalnik funkcij (npr. `QOpenGLFunctions_3_3_Core`)

Implementacija: prevajanje senčilnega programa

- `p=glCreateProgram()`

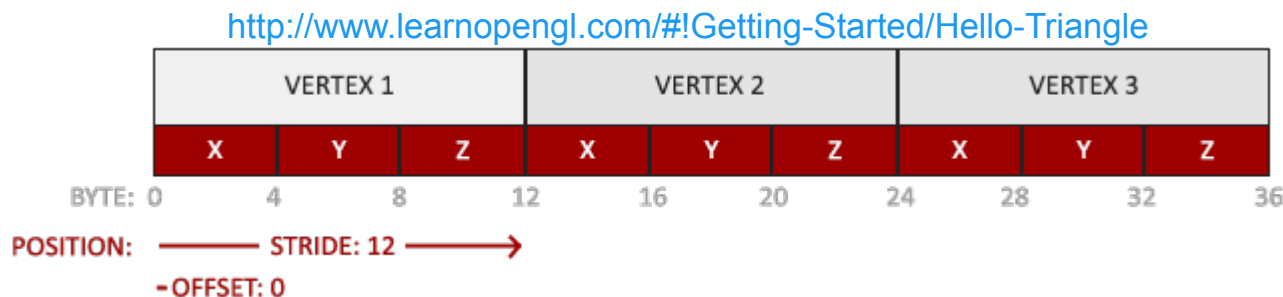
- `uint s=glCreateShader(GL_VERTEX_SHADER ali GL_FRAGMENT_SHADER)`
- `glShaderSource(s, 1, &izvorna_koda, nullptr)`
- `glCompileShader(s)`
- `glAttachShader(p, s)`

- `glLinkProgram(p)`

shader program = vertex shader + fragment shader + ostali senčilniki

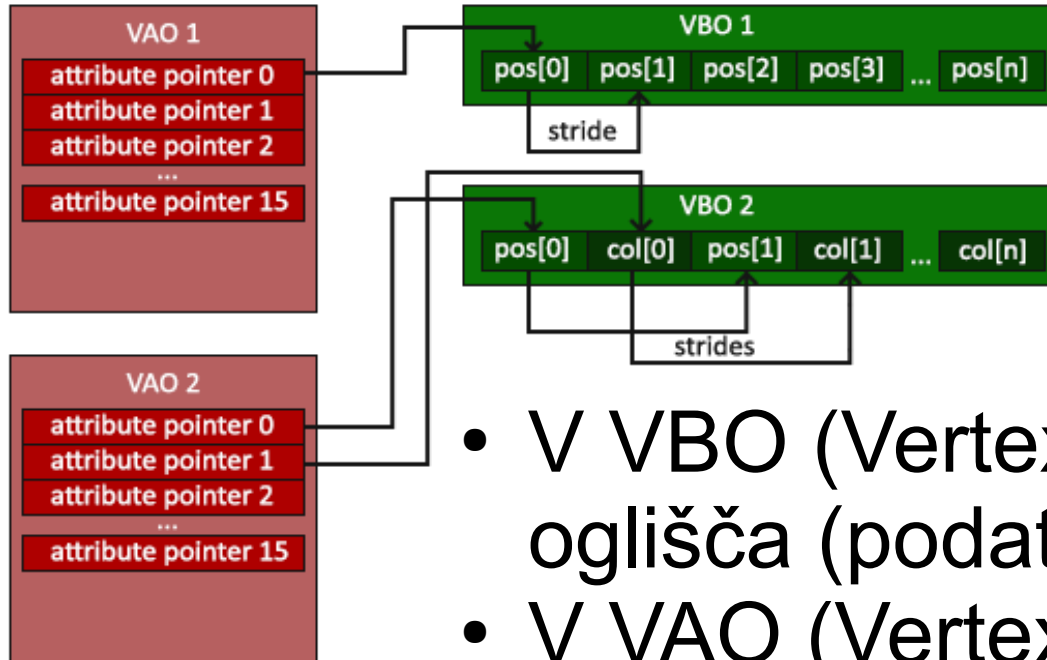
Nalaganje podatkov (trikotniška mreža)

- Prenos oglišč med GPU in CPU je počasen, zato je geometrijske podatke potrebno naložiti v pomnilnik na grafični kartici (za večkratno uporabo)
- To izvedemo v času inicializacije in ne tekom izrisa!
- Potrebujemo smiselno predstavitev geometrijskih podatkov/oglišč



Nalaganje podatkov (trikotniška mreža)

<http://www.learnopengl.com/#!Getting-Started/Hello-Triangle>



- V VBO (Vertex buffer object) hranimo oglišča (podatke)
- V VAO (Vertex Array Object) hranimo opis organizacije podatkov (podatki o podatkih):
 - Razmik med posameznimi oglišči
 - Uporabljene lastnosti oglišč (lokacija, barva, ...)

Implementacija: nalaganje trikotnika na GPU

<http://www.learnopengl.com/#!/Getting-Started/Hello-Triangle>



- `vec3 trikotnik[]={vec3(0,0,0), vec3(0,0.5,0), vec3(0.5,0.5,0)};`
- `glGenVertexArrays(1,&vertex_array); // VAO – opis organizacije podatkov`
- `glBindVertexArray(vertex_array);`
- `glGenBuffers(1,&id_buffer_trikotniki); // VBO - podatki`
- `glBindBuffer(GL_ARRAY_BUFFER, id_buffer_trikotniki);`
- `glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3)*3, trikotnik, GL_STATIC_DRAW);`
`glEnableVertexAttribArray(0); // položaj so prvi prvi atribut oglišč v GLSL`
`// uporabljamo: layout(location=0) in vec3 in_Pos; v GLSL`
- `glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(glm::vec3), 0); // opis podatkov`
`// to se samodejno zapiše v priključen VAO vertex_array`

Implementacija: izris

- `glClearColor(0, 0, 0.5, 1);`
- `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
- `glUseProgram(p);`
- `glUniformMatrix4fv(glGetUniformLocation(p, "PVM"),
1, GL_FALSE, glm::value_ptr(PVM)) // naloži matriko »v senčilnike«`
- `glUniform4f(glGetUniformLocation(p, "BarvaRGBA"), 1,0,0,0);`
- `// https://registry.khronos.org/OpenGL-Refpages/gl4/html/glUniform.xhtml`
- `glBindVertexArray(vertex_array); // priklapi VAO: objekt za izris`
- `glDrawArrays(GL_TRIANGLES, 0, 3); // od prvega oglišča (0) nariši 3 oglišča`
- `// https://registry.khronos.org/OpenGL-Refpages/gl4/html/glDrawArrays.xhtml`

Koristne knjižnice

- <http://glm.g-truc.net/>
 - delo z matrikami in vektorji
- <http://assimp.sourceforge.net/>
 - nalaganje 3D modelov iz različnih animacijskih paketov (Blender, LightWave, 3ds Max, ...)
- <http://freeimage.sourceforge.net/>
 - nalaganje slikovnih datotek za lepljenje na ploskve 3D objektov
- <http://www.qt.io/download-open-source/>
 - Vključuje tudi nalaganje slikovnih datotek

Dodatna literatura

- <https://open.gl/>
- <http://www.learnopengl.com/>
- <http://docs.gl/>
- <http://www.swiftless.com/opengl4tuts.html>
- <http://www.opengl.org/wiki/Tutorials>
- <https://www.shadertoy.com/>
- Koristna orodja
 - <https://renderdoc.org/> - razhroščevalnik (vpogled v izvedene klice na GPU)
 - <http://www.mesa3d.org/> (odprto-kodni gonilniki in tudi programski emulator za OpenGL)