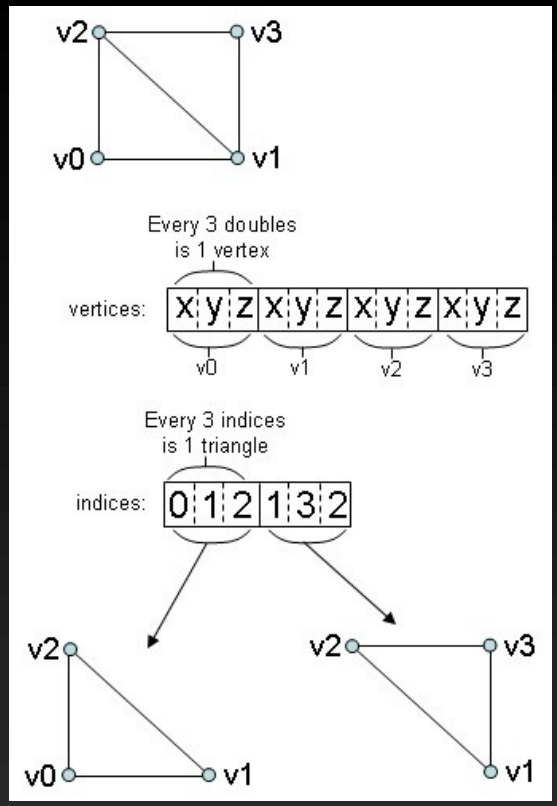
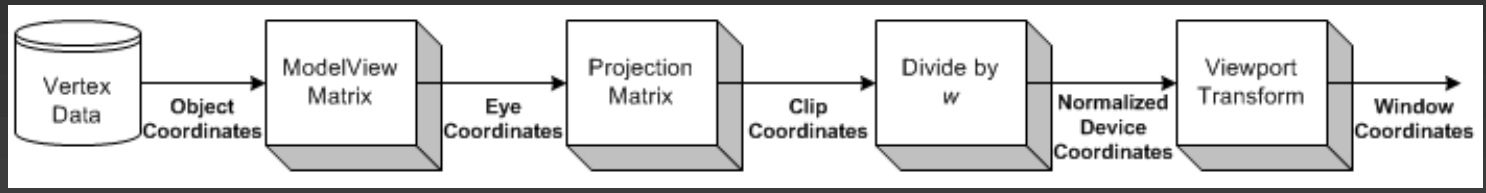
Opakování

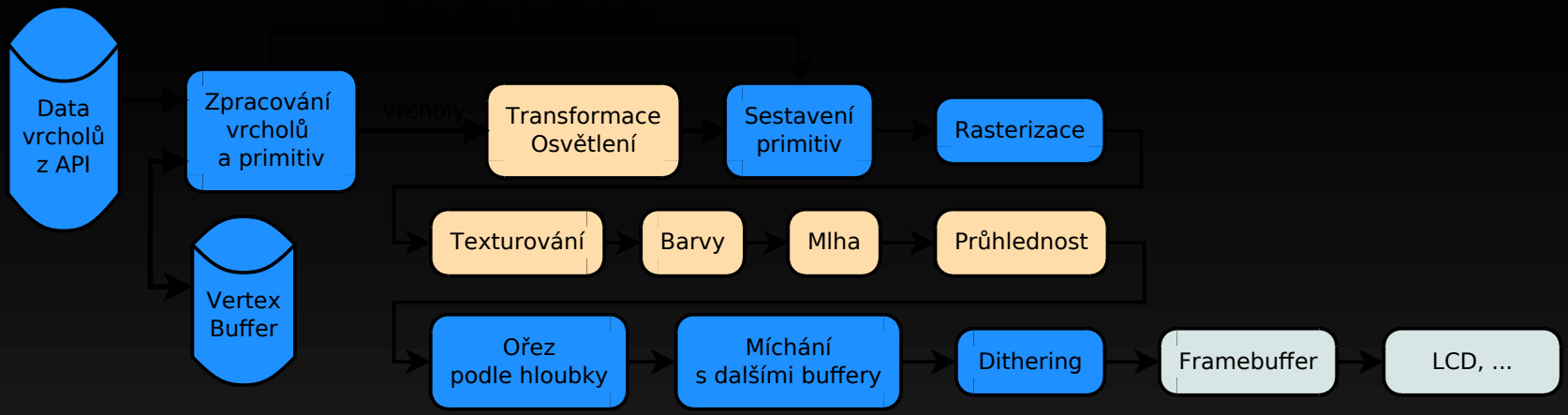


- Reprezentace
  - 3D rastr
  - obálka
    - vrchol (vertex)
    - hrana (edge)
    - ploška (face)
- 3D zobrazování
  - načtení a transformace souřadnic zadaných vertexů



  - rasterizace
  - výpočet barvy fragmentu
  - průhlednost a zakrývání podle Z

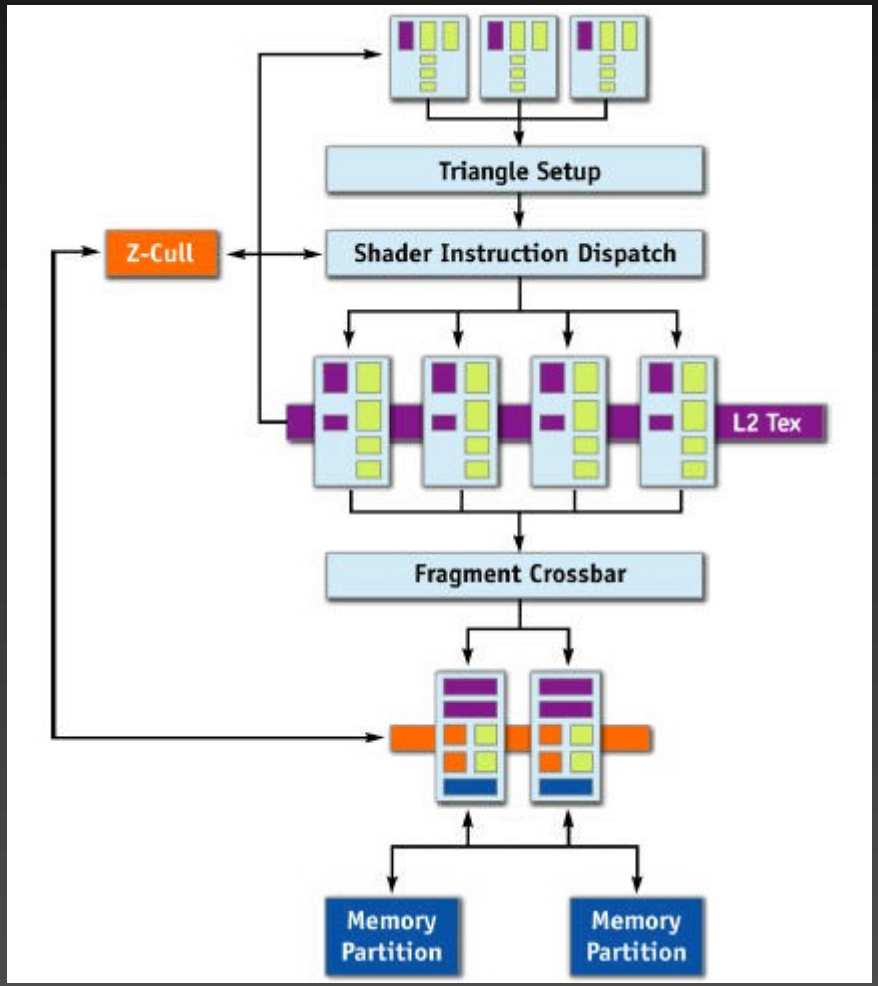
3D graphic pipeline
OpenGL pipeline



Pipeline v hardware

- pevná

- programovatelná
(specializované shader jednotky)



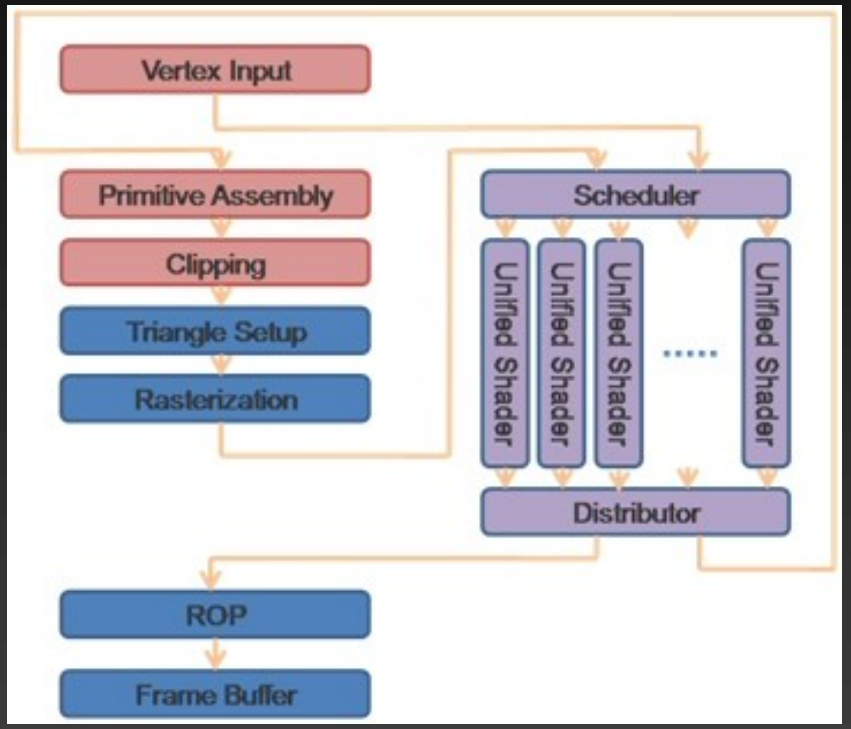
Why unify?

Unified Shader

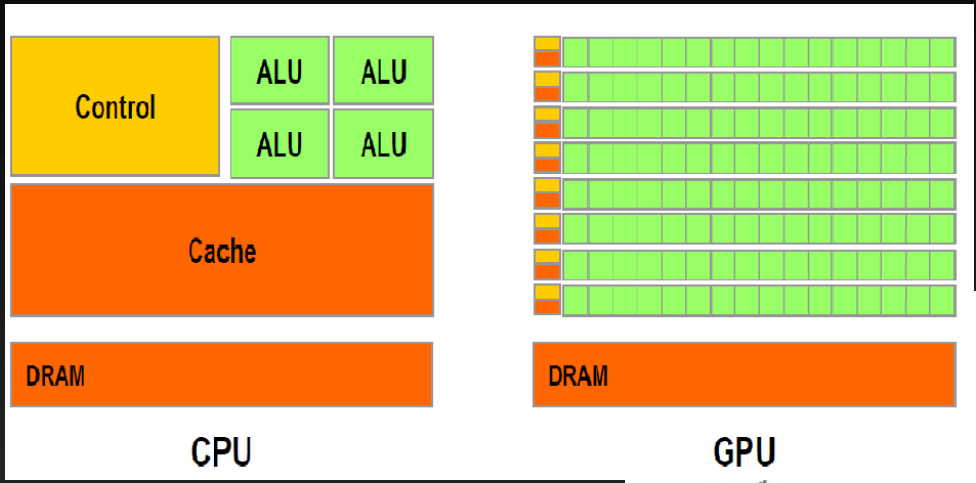
Heavy Geometry
Workload Perf =12

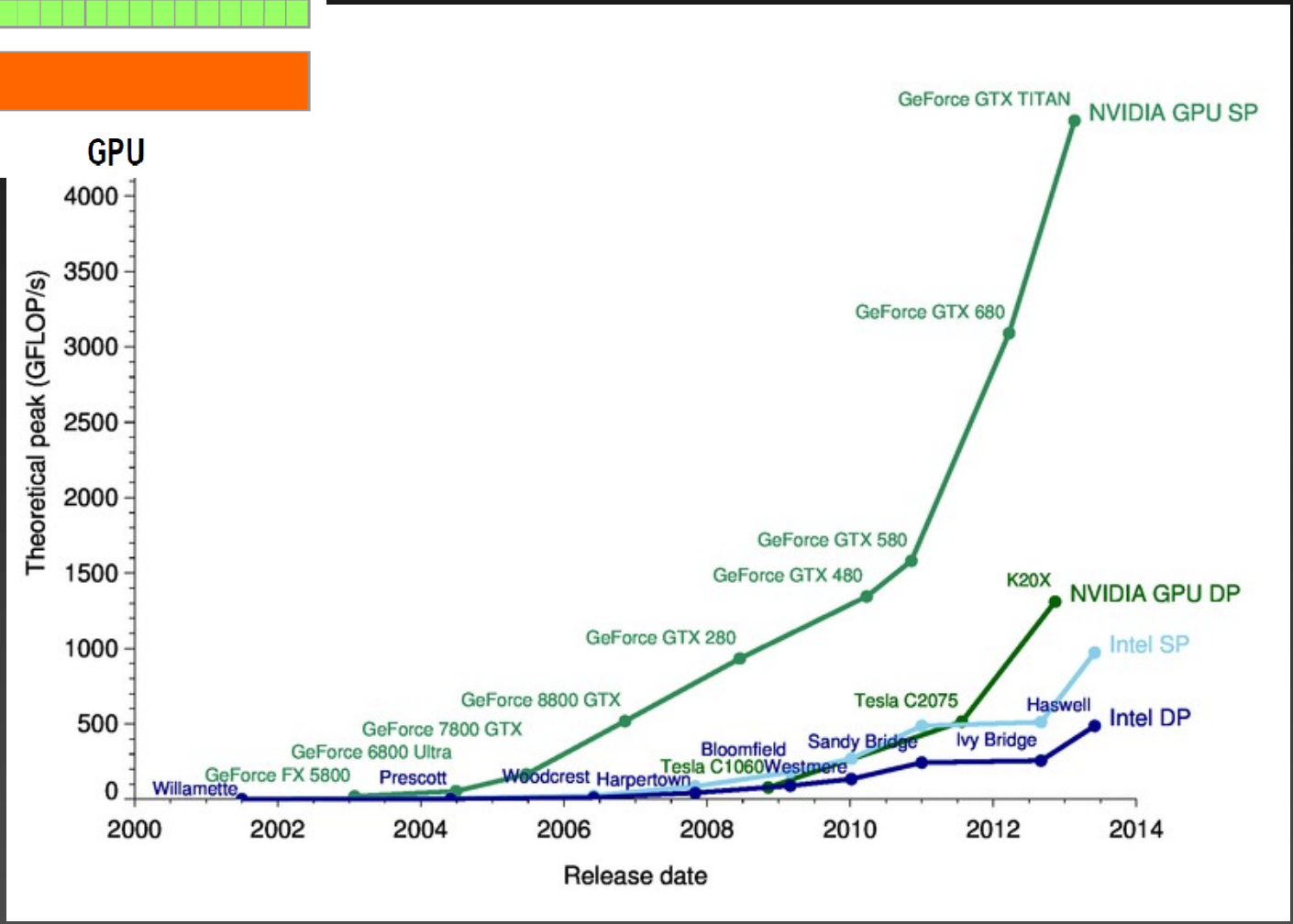
Unified Shader

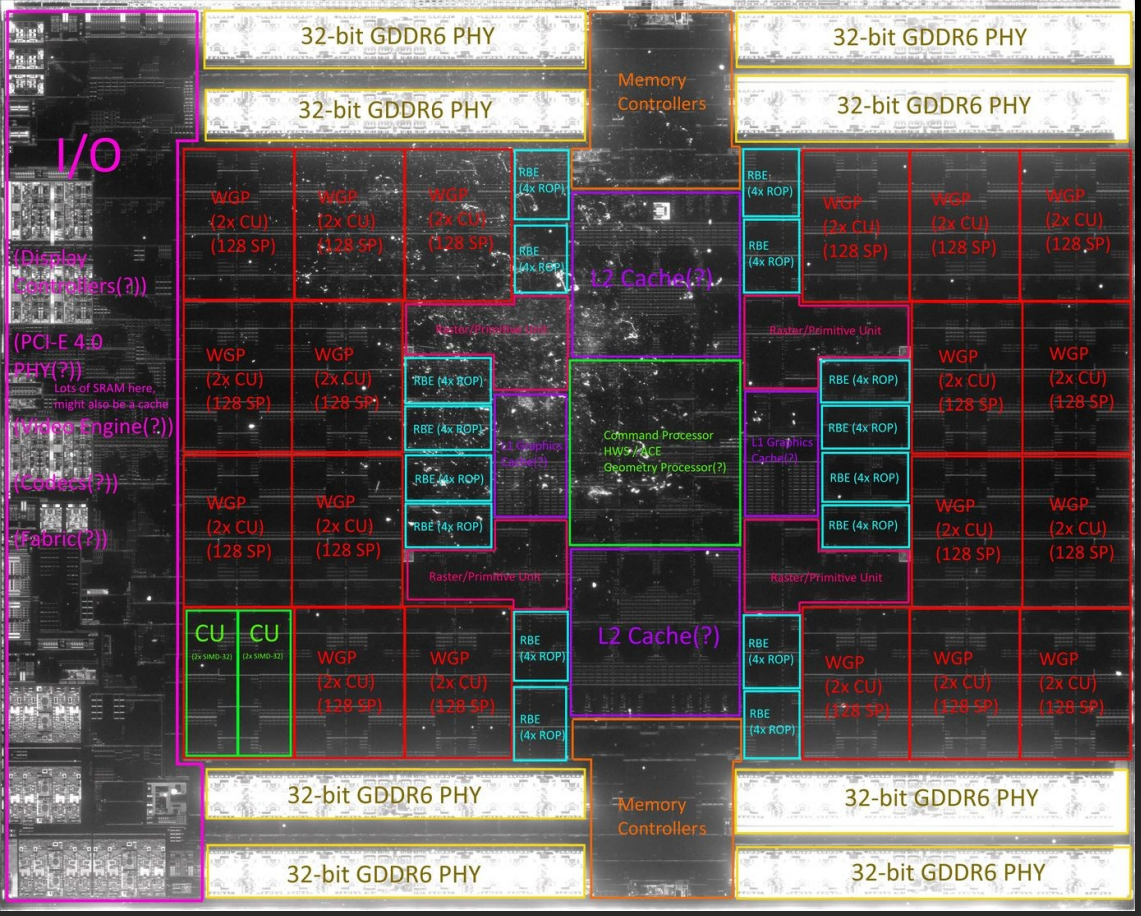
Heavy Pixel
Workload Perf = 12

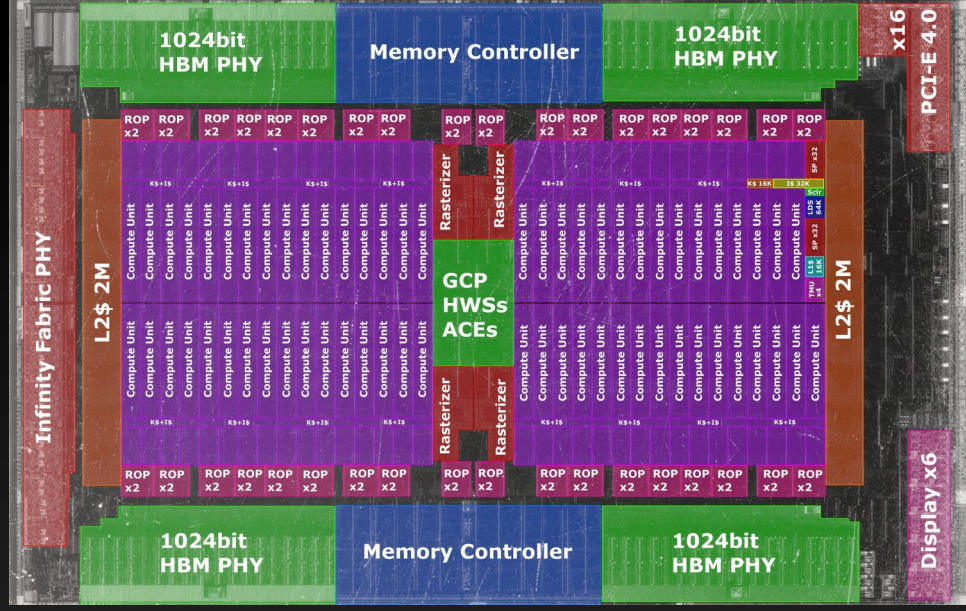


CPU vs. GPU


Control
ALU
ALU
ALU
ALU
Cache
DRAM
DRAM
CPU
GPU

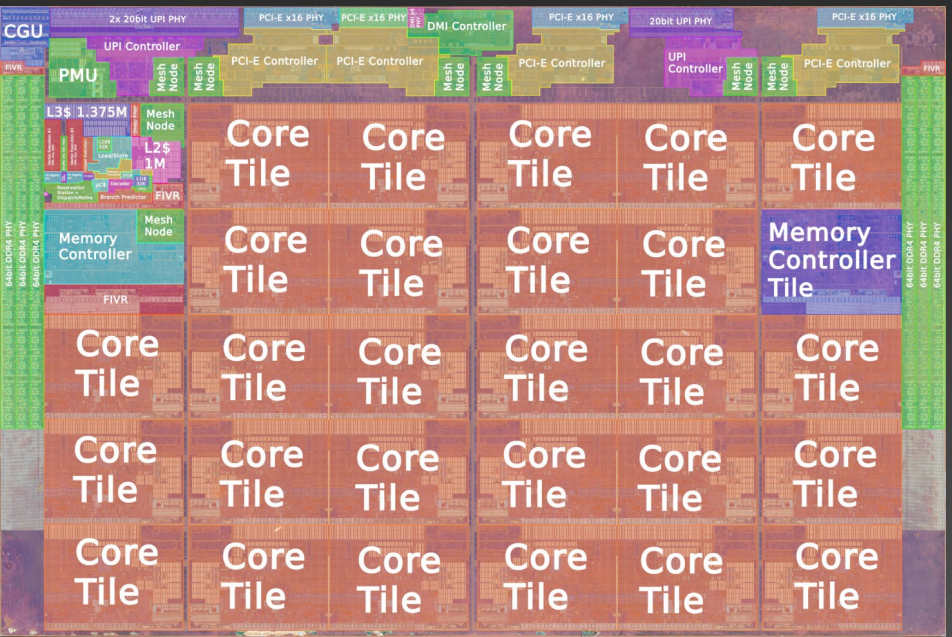

Theoretical peak (GFLOP/s)
Release date
0
500
1000
1500
2000
2500
3000
3500
4000
2000
2002
2004
2006
2008
2010
2012
2014
GeForce GTX TITAN
NVIDIA GPU SP
GeForce GTX 680
GeForce GTX 580
GeForce GTX 480
GeForce GTX 280
GeForce 8800 GTX
GeForce 7800 GTX
GeForce 6800 Ultra
GeForce FX 5800
Willamette
Prescott
Woodcrest
Harpertown
Bloomfield
Tesla C1060
Westmere
Sandy Bridge
Ivy Bridge
Haswell
Tesla C2075
K20X
NVIDIA GPU DP
Intel SP
Intel DP





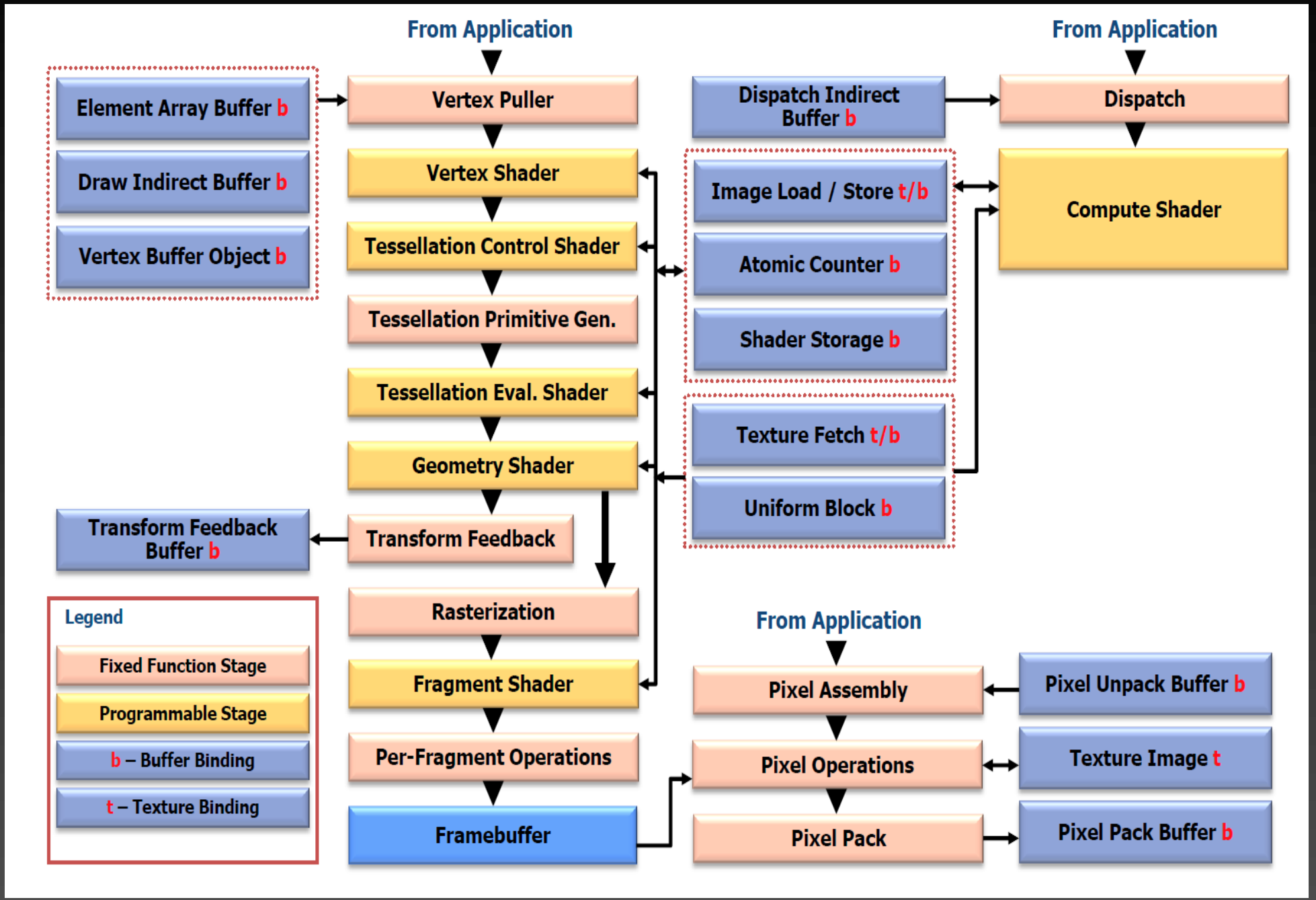
AMD VegaII (Radeon 7)
7nm, 331 mm², 3480 core

AMD Navi10 (Radeon 5700XT)
7nm, 251 mm², 2560 core

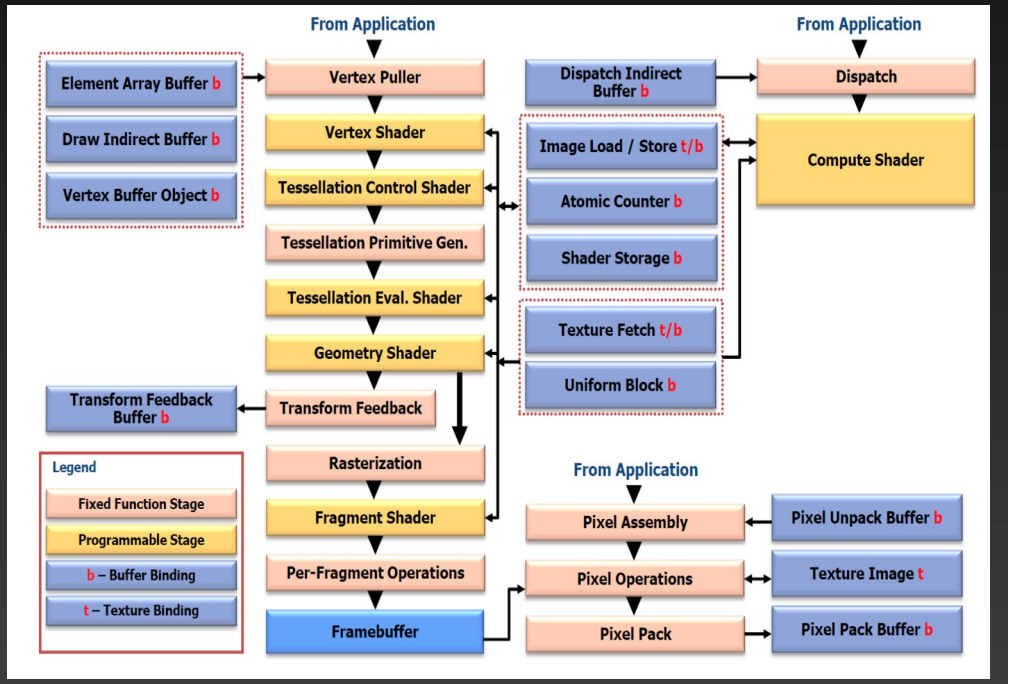


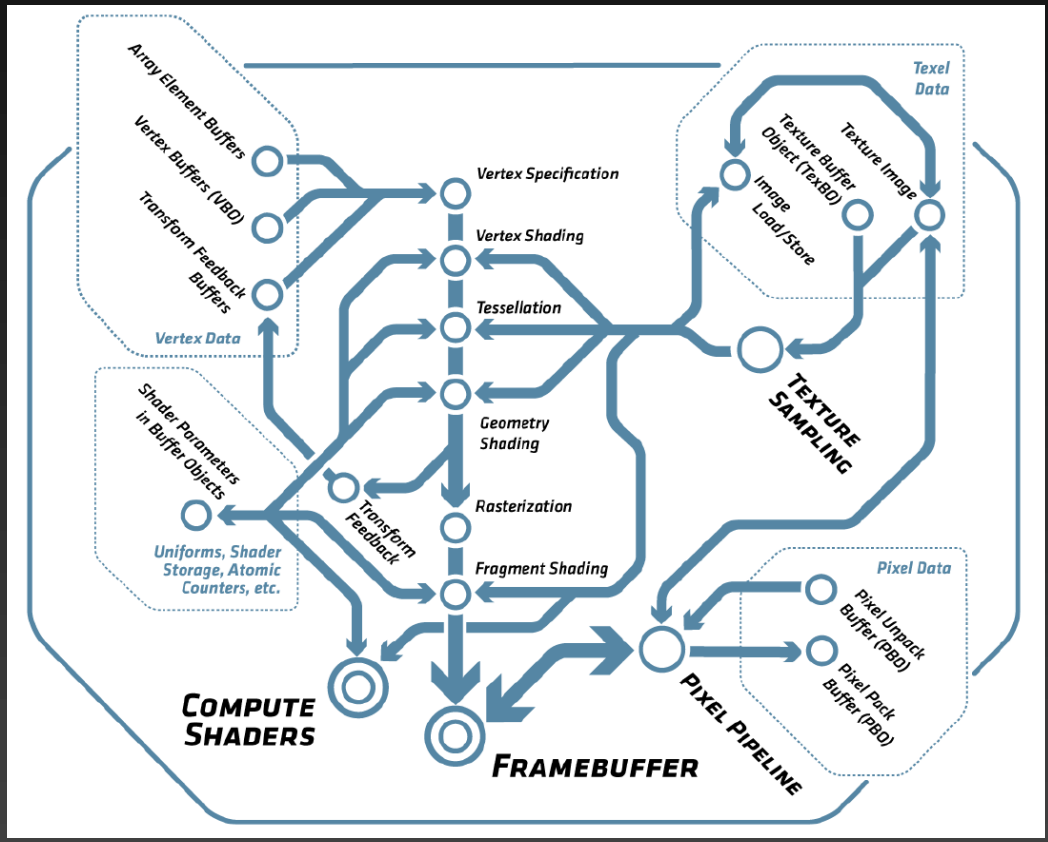
Intel Skylake (Xeon Platinum)
14nm, 694 mm², 28 core + SMT2

OpenGL 4.6 (Core Profile) - May 5, 2022



OpenGL 4.6 (Core Profile) - May 5, 2022





OpenGL pipeline functions

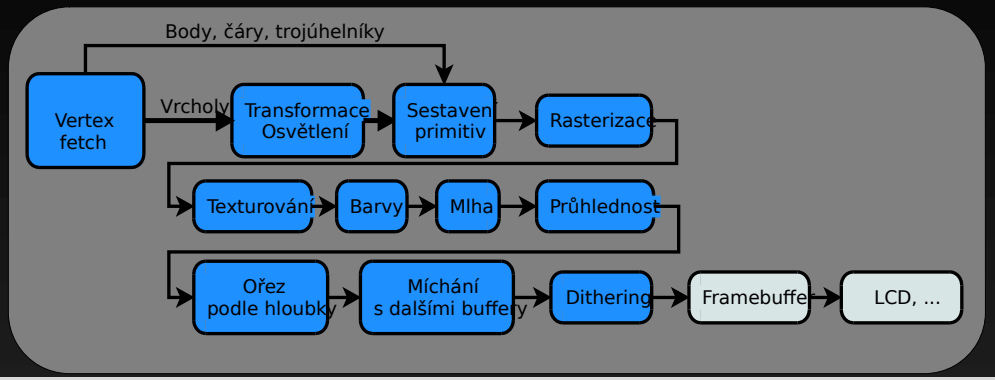
- Vertex processing
  - coordinates transformation, calculation of normals, colors, UV textures, …
    - directly or from lighting model
- Geometry processing
  - clipping, culling
  - perspective projection (larger distance → smaller object)
  - enabling of vertex/edges/faces rasterization
- Rasterisation = conversion to fragments
  - first: cull back side of polygons, clip by 6 planes, w division (perspective)
  - viewport, antialiasing
  - fragment = complex entity, set of information
    - similar to pixel, but not stored yet
    - each fragment has [x,y,z] coordinates and color
  - all information taken into account
    - line width, point size, lights, materials, antialiasing, …
  - drawing of edges, filling polygons
- Pixel and textures operations
  - decompression, format conversion, filtration
  - math operations (+,*, saturation, …)

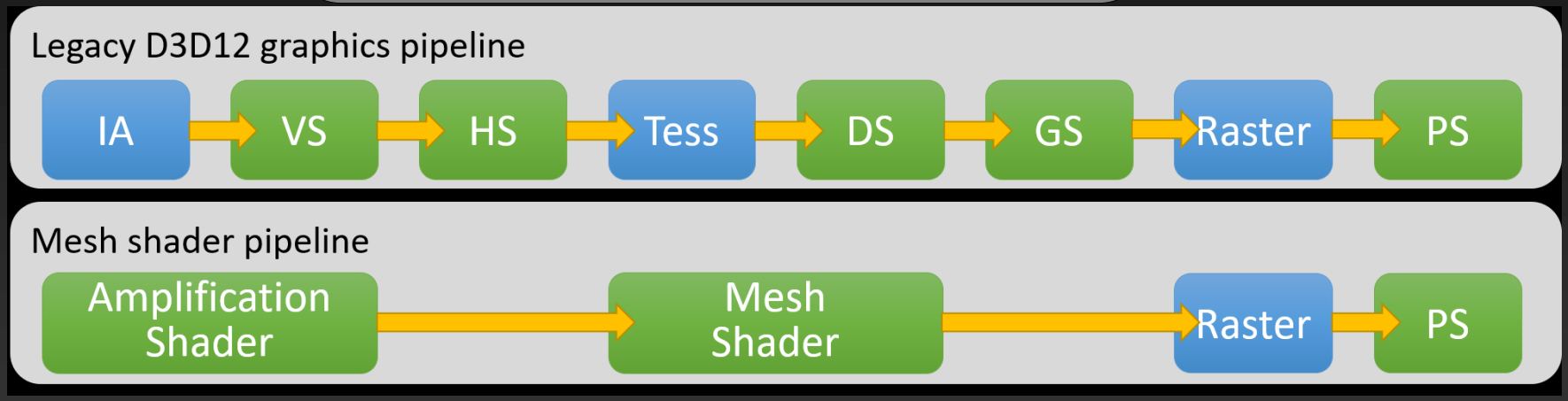
OpenGL pipeline functions (cont.)

- Fragment operations
  - texturing, fog
  - clipping by stencil, depth
  - blending with already existing fragment (alfa blending)
  - dithering
  - math ops

2020+ vývoj k plné programovatelnosti

Pevná pipeline (legacy)

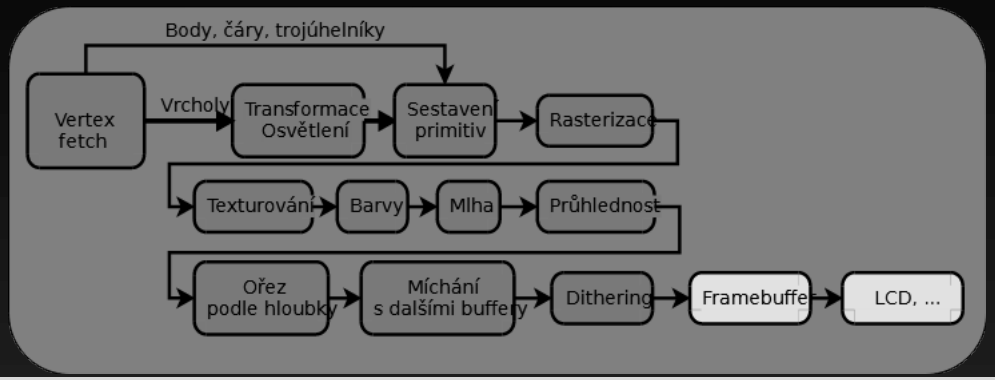


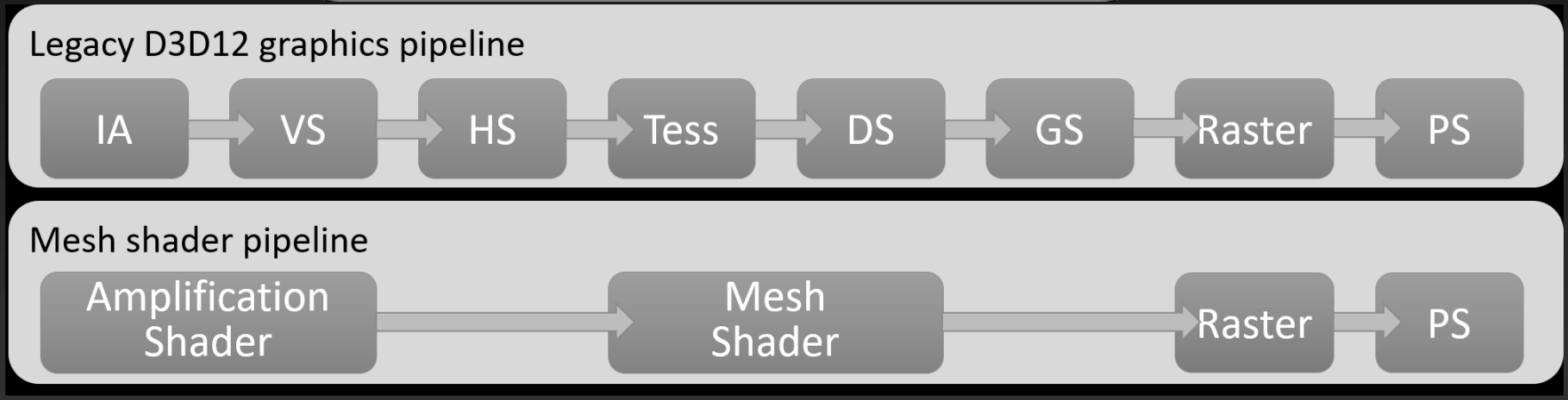


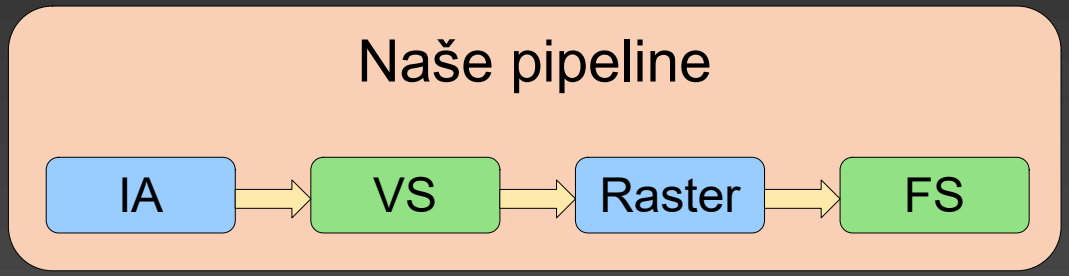
- Rozhraní
  - OpenGL, DX12 Ultimate, Vulkan
- HW
  - PS5, Xbox series X
  - NVidia RTX 3000, AMD Radeon 6000
- Předchůdce
  - AMD Next-Generation Geometry (NGG), cca 2017

Pipeline pro nás

Pevná pipeline (legacy)







Modern OpenGL

- relativelly new
  - check your graphics card for support
- create OpenGL context with latest version
  - no version specification → select latest
  - latest version: 4.6
- create OpenGL context with specific version
  - e.g. OpenGL Core version **4.5+**

```
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 5);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
```

- version specification in shaders

```
#version 450 core
```

One Triangle: Compatibility vs. Core profile

```
glBegin(GL_TRIANGLES);
  glTexCoord2f(0.0f, 0.0f);
  glVertex2i(200, 50);

  glTexCoord2f(0.0f, 1.0f);
  glVertex2i(50, 250);

  glTexCoord2f(1.0f, 1.0f);
  glVertex2i(350, 250);
glEnd();
```

```
#version 330 core
layout (location = 0) in vec3 aPos; // Positions/Coordinates
layout (location = 1) in vec2 aTex; // Texture Coordinates

uniform mat4 uProj_m,uV_m,uM_m;

out VS_OUT {
	vec3 color; // Outputs color for FS
	vec2 texCoord; // Outputs texture coordinates for FS
} vs_out;

void main() {
	// Outputs coordinates of all vertices
	gl_Position = uProj_m * uV_m * uM_m * vec4(aPos,1.0f);

	// Assigns the colors somehow
	vs_out.color = vec3(1.0); //white

	// Pass the texture coordinates to "texCoord" for FS
	vs_out.texCoord = aTex;
}
```

```
#version 330 core
in VS_OUT {
	vec3 color; // color for FS
	vec2 texCoord; // texture coordinates for FS
} fs_in;

uniform sampler2D tex0; // texture unit from C++

out vec4  FragColor; // Final output

void main() {

	FragColor = fs_in.color * texture(tex0, fs_in.texcoord);
}
```

```
	//existing data
	struct my_vertex {
		glm::vec3 position;  // Vertex
		glm::vec2 texcoord;  // Texcoord0
	};
	std::vector<my_vertex> vertices = {
			{ {200,50,0}, {0,0} },
			{ {50,250,0}, {0,1} },
			{ {350,250,0}, {1,1} } };
	std::vector<GLuint> indices = {0,1,2};

	//GL names for Array and Buffers Objects
	GLuint VAO, VBO, EBO;
//--------------------------------------------------------------------------------
	// Generate the VAO and VBO
	glGenVertexArrays(1, &VAO);
	glGenBuffers(1, &VBO);
	glGenBuffers(1, &EBO);
	// Bind VAO (set as the current)
	glBindVertexArray(VAO);
	// Bind the VBO, set type as GL_ARRAY_BUFFER
	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	// Fill-in data into the VBO
	glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(vertex),
		vertices.data(), GL_STATIC_DRAW);
	// Bind EBO, set type GL_ELEMENT_ARRAY_BUFFER
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
	// Fill-in data into the EBO
	glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(GLuint),
		indices.data(), GL_STATIC_DRAW);
	// Set Vertex Attribute to explain OpenGL how to interpret the VBO
	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(vertex),
		(void*)(0 + offsetof(vertex, position)));
	// Enable the Vertex Attribute 0 = position
	glEnableVertexAttribArray(0);

	// Set end enable Vertex Attribute 1 = Texture Coordinates
	glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(my_vertex),
	(void*)(0 + offsetof(my_vertex, normal)));
	glEnableVertexAttribArray(1);

	// Bind VBO and VAO to 0 to prevent unintended modification
	glBindBuffer(GL_ARRAY_BUFFER, 0);
	glBindVertexArray(0);
	glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
//--------------------------------------------------------------------------------
    // create and use shaders
    GLuint VS_h, FS_h, prog_h;
    VS_h = glCreateShader(GL_VERTEX_SHADER);
    FS_h = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(VS_h, 1, &VS_string, NULL);
    glShaderSource(FS_h, 1, &FS_string, NULL);
    glCompileShader(VS_h);
    glCompileShader(FS_h);
    prog_h = glCreateProgram();
    glAttachShader(prog_h, VS_h);
    glAttachShader(prog_h, FS_h);
    glLinkProgram(prog_h);
    glUseProgram(prog_h);
//--------------------------------------------------------------------------------
	// USE buffers
	glBindVertexArray(VAO);

   glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0)
```