Language for Prototyping of Visualizations

Author: Bc. Daniel Kříž

Supervisor: Ing. Tomáš Milet, Ph.D.

1. What is VPSL?

VPSL is a language based on high-level directives that is used together with some shading language targeting GPUs. It aims to be minimal and easy to grasp, making it possible to focus on the GPU program development rather than on the CPU setup.

2. Why is it useful?

- Based on well-known concepts.
- Less work on the CPU setup and more focus on the GPU.
- No need to learn a whole new language.
- Extensible to other shading languages than just GLSL.

3. The Basic Idea

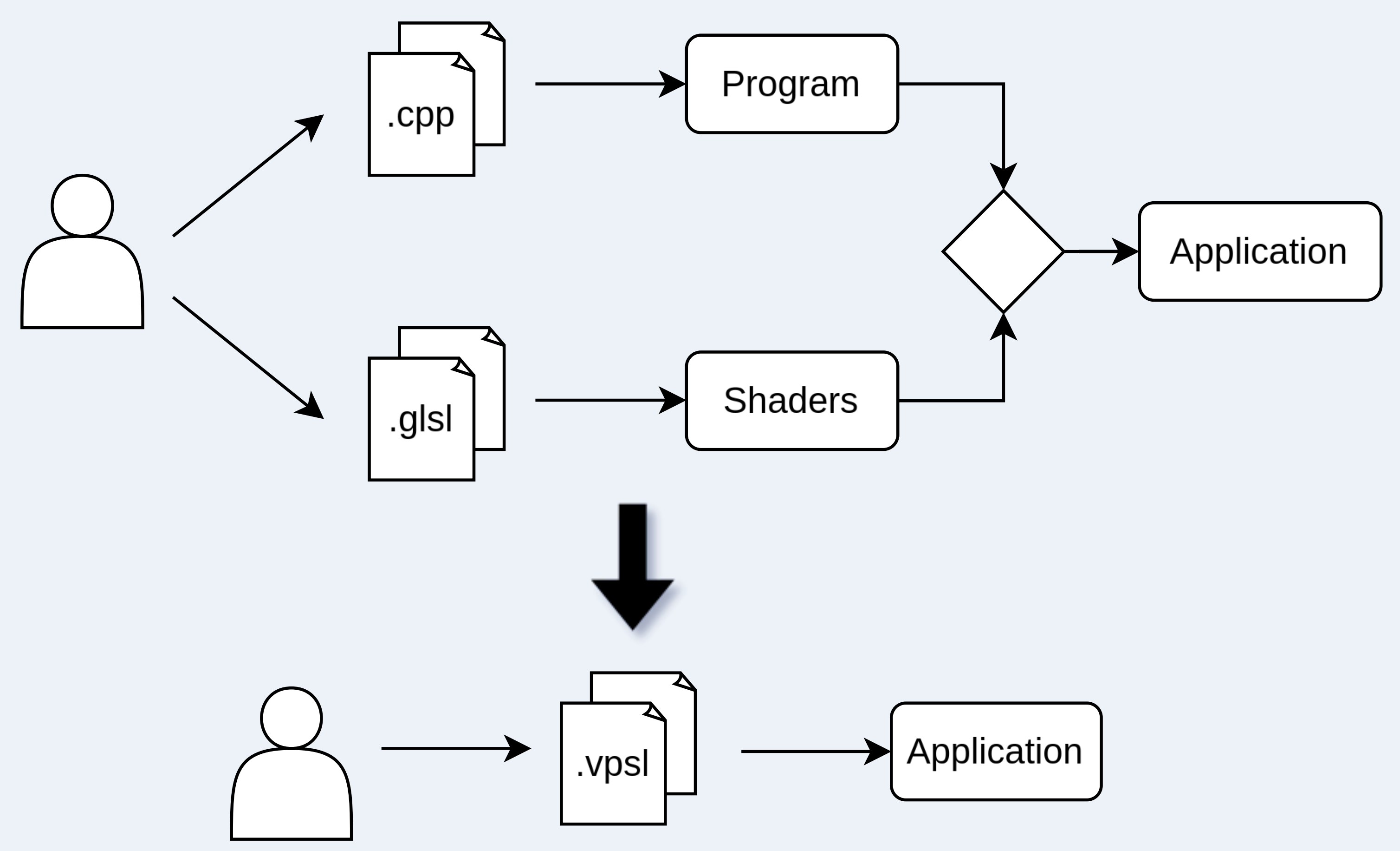

Figure 1: The difference between traditional and new approach.

4. Usage

```
#pragma vp load texture name(triangle_texture) path(...)

#pragma vp program
#pragma vp begin
#pragma vp shader type(vertex) name(main_vs)
#pragma vp begin
#version 450 core
layout (location = 0) out VS_OUT { vec2 tex_coords; } vs_out;

void main() {
    const vec2 tex_coords[3] = vec2[3](
        vec2(0.0f, 0.0f),
        vec2(1.0f, 0.0f),
        vec2(0.5f, 1.0f)
    );
    const vec4 vertices[3] = vec4[3](
        vec4(-0.5f, -0.5f, 0.0f, 1.0f),
        vec4( 0.5f, -0.5f, 0.0f, 1.0f),
        vec4( 0.0f,  0.5f, 0.0f, 1.0f)
    );
    gl_Position = vertices[gl_VertexID];
    vs_out.tex_coords = vec2(
        tex_coords[gl_VertexID].x,
        tex_coords[gl_VertexID].y
    );
}

#pragma vp end // main_vs

#pragma vp shader type(fragment) name(main_fs)
#pragma vp begin
#version 450 core

#pragma vp texture name(triangle_texture)
uniform sampler2D u_texture;

layout (location = 0) out vec4 color;

layout (location = 1) in VS_OUT {
    vec2 tex_coords;
} fs_in;

void main() {
    color = texture(u_texture, fs_in.tex_coords);
}

#pragma vp end // main_fs
#pragma vp end // triangle program
```

Listing 1: An example of simple shader program in VPSL.

5. Directives

- **Begin-end** - encloses directive's scope
- **Shader** - declares a new shader with a type, can append or prepend other shaders
- **Program** - declares a new program, can be created from other shaders. It is also possible to define their order of execution
- **Load** - loads resource (material, texture, mesh) from a file
- **Texture** - declares a texture in shader code
- **Buffer** - defines `std430` buffer with a size
- **Resource store** - appends some location to the search path
- **Include** - appends other `.vpsl` file
- **CopyIn** - defines shader attributes
- **Option** - enables/disables some settings (e.g., face culling)

6. How it works?

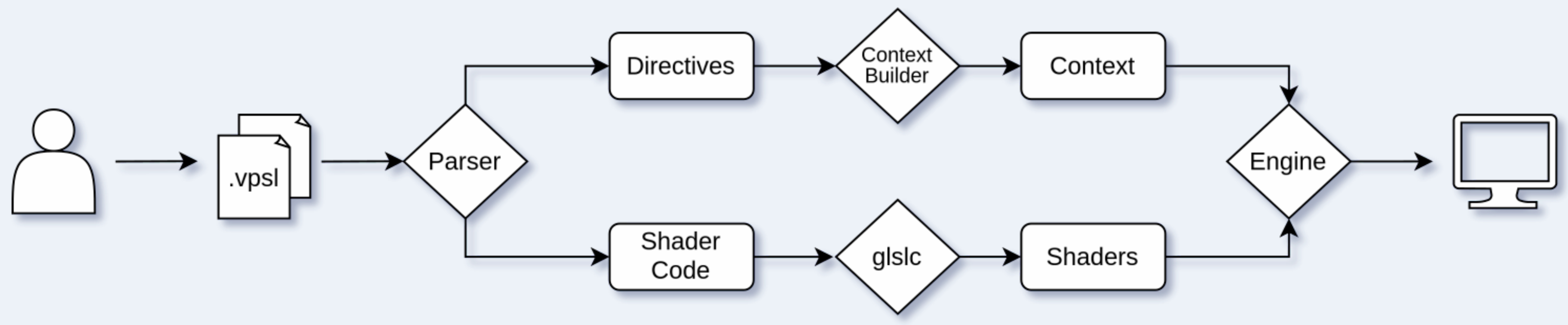

Figure 2: A diagram showing a high-level architecture of the VPSL.

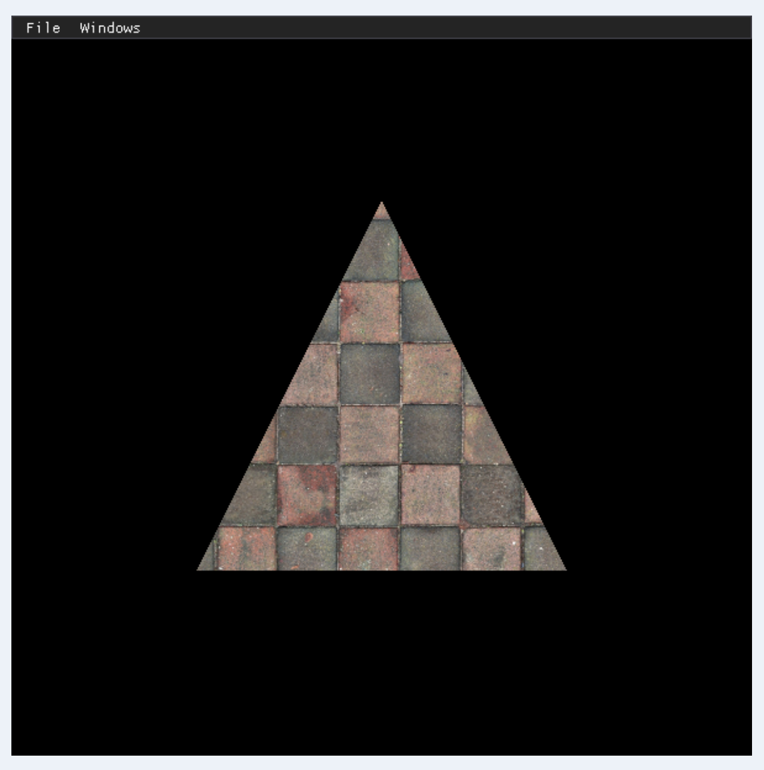

Figure 3: A screenshot of output.