LMS Moodle VŠB-TUO: 460-2021/03 Základy počítačové grafiky (2024/2025 ZS): Sekce: Cvičení 2

Toto cvičení budeme věnovat doplnění a úpravě zdrojového kódu o načtení modelu, shaderů, inicializace OpenGL 3.3+ atd. (přechod na moderní OpenGL).

Další modely budeme používat později, jsou složené z pozice a normály.

Nejdříve budeme potřebovat inicializovat knihovnu GLEW (můžete využít ukázku z minulého cvičení), include knihovny GLEW použijte před knihovnou GLFW.

```
//Include GLEW
#include <GL/glew.h>
//Include GLFW
#include <GLFW/glfw3.h>
```

Dále budeme potřebovat pole vrcholů, které bude definovat náš model (trojúhelník) a shadery (vertex a fragment).

```
 float points[] = {
   0.0f, 0.5f, 0.0f,
   0.5f, -0.5f, 0.0f,
  -0.5f, -0.5f, 0.0f
  };

 const char* vertex_shader =
"#version 330\n"
"layout(location=0) in vec3 vp;"
"void main () {"
"    gl_Position = vec4 (vp, 1.0);"
"}";



const char* fragment_shader =
"#version 330\n"
"out vec4 frag_colour;"
"void main () {"
"    frag_colour = vec4 (0.5, 0.0, 0.5, 1.0);"
"}";
```

Upravte main tak, aby jste inicializovali moderní OpenGL (3.3+).

```
GLFWwindow* window;
glfwSetErrorCallback(error_callback);
if (!glfwInit()) {
  fprintf(stderr, "ERROR: could not start GLFW3\n");
  exit(EXIT_FAILURE);
}
```

```
/* //inicializace konkretni verze
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
glfwWindowHint(GLFW_OPENGL_PROFILE,
GLFW_OPENGL_CORE_PROFILE);  //*/

window = glfwCreateWindow(800, 600, "ZPG", NULL, NULL);
if (!window){
  glfwTerminate();
  exit(EXIT_FAILURE);
}

glfwMakeContextCurrent(window);
glfwSwapInterval(1);

// start GLEW extension handler
glewExperimental = GL_TRUE;
glewInit();


// get version info
printf("OpenGL Version: %s\n",glGetString(GL_VERSION));
printf("Using GLEW %s\n", glewGetString(GLEW_VERSION));
printf("Vendor %s\n", glGetString(GL_VENDOR));
printf("Renderer %s\n", glGetString(GL_RENDERER));
printf("GLSL %s\n", glGetString(GL_SHADING_LANGUAGE_VERSION));
int major, minor, revision;
glfwGetVersion(&major, &minor, &revision);
printf("Using GLFW %i.%i.%i\n", major, minor, revision);

int width, height;
glfwGetFramebufferSize(window, &width, &height);
float ratio = width / (float)height;
glViewport(0, 0, width, height);
```

Vytvořte VBO (buffer objekt), což je blok paměti s daty a VAO (vertex array) pro následné vykreslování modelu.

```
//vertex buffer object (VBO)
GLuint VBO = 0;
glGenBuffers(1, &VBO); // generate the VBO
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof (points), points, GL_STATIC_DRAW);

//Vertex Array Object (VAO)
GLuint VAO = 0;
glGenVertexArrays(1, &VAO); //generate the VAO
glBindVertexArray(VAO); //bind the VAO
glEnableVertexAttribArray(0); //enable vertex attributes
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, NULL);
```

Vytvořte vertex a fragment shader a následně shader program

```
 //create and compile shaders
GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertex_shader, NULL);
glCompileShader(vertexShader);
GLuint fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragmentShader, 1, &fragment_shader, NULL);
glCompileShader(fragmentShader);
GLuint shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, fragmentShader);
glAttachShader(shaderProgram, vertexShader);
glLinkProgram(shaderProgram);
```

Upravte vykreslovací smyčku tak aby jste trojúhelník vykreslili

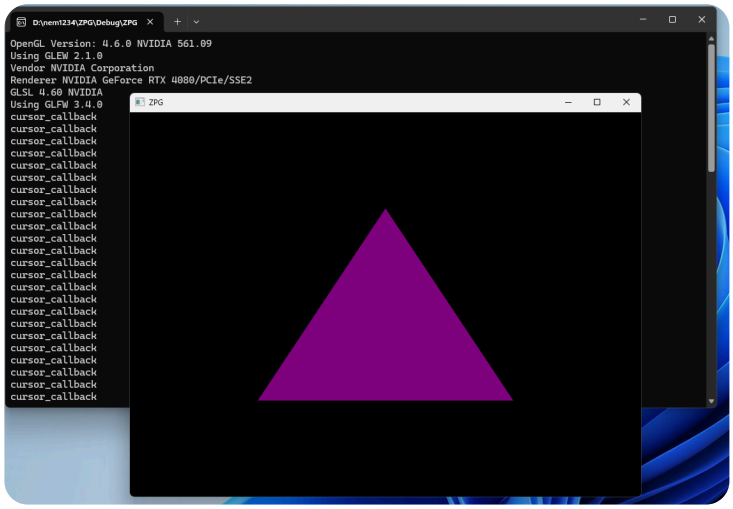
```
 while (!glfwWindowShouldClose(window)){
    // clear color and depth buffer
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glUseProgram(shaderProgram);
    glBindVertexArray(VAO);
    // draw triangles
    glDrawArrays(GL_TRIANGLES, 0, 3); //mode,first,count
    // update other events like input handling
    glfwPollEvents();
    // put the stuff we've been drawing onto the display
    glfwSwapBuffers(window);
 }

glfwDestroyWindow(window);

glfwTerminate();
exit(EXIT_SUCCESS);
}
```

Kontrola kompilace a linkování program shaderu

```
GLint status;
glGetProgramiv(shaderProgram, GL_LINK_STATUS, &status);
if (status == GL_FALSE)
{
  GLint infoLogLength;
  glGetProgramiv(shaderProgram, GL_INFO_LOG_LENGTH, &infoLogLength);
  GLchar *strInfoLog = new GLchar[infoLogLength + 1];
  glGetProgramInfoLog(shaderProgram, infoLogLength, NULL, strInfoLog);
  fprintf(stderr, "Linker failure: %s\n", strInfoLog);
  delete[] strInfoLog;
}
```



Ostraňte deprecated příkazy, které se používaly ve spojení s fixním vykreslovacím řetězcem v prvním cvičení.

glBegin(...)/glEnd(), glColor*, glVertex*, glTexCoord*, glRotate*, glTranslate*, glScale*, glMatrixMode(), glLoadIdentity(), glPushMatrix(), glPopMatrix(), glFrustum(), gluPerspective(...), gluLookAt(..) atd.

Pokud zjistíte, že chybí glew32.dll, můžete to vyřešit tak, že tento soubor z knihovny nakopírujete do adresáře s výsledným EXE souborem.

Možnost převodu callback funkcí z knihovny GLFW (C) na vlastní metody (C++).

```
void Application::cursor_pos_callback(GLFWwindow* window,double mouseX,double mouseY){
    printf("cursor_pos_callback %d, %d; %d, %d\n", (int)mouseX, (int)mouseY, (int)clickX, (int)clickY);
}
glfwSetCursorPosCallback(window, [](GLFWwindow* window,double mouseXPos,double mouseYPos)-> void {Application::getInstance()->cursor_pos_callback(window,mouseXPos,mouseYPos);});
```

Pokračujte v rozšiřování zdrojového kódu a úpravy na objektový kód.

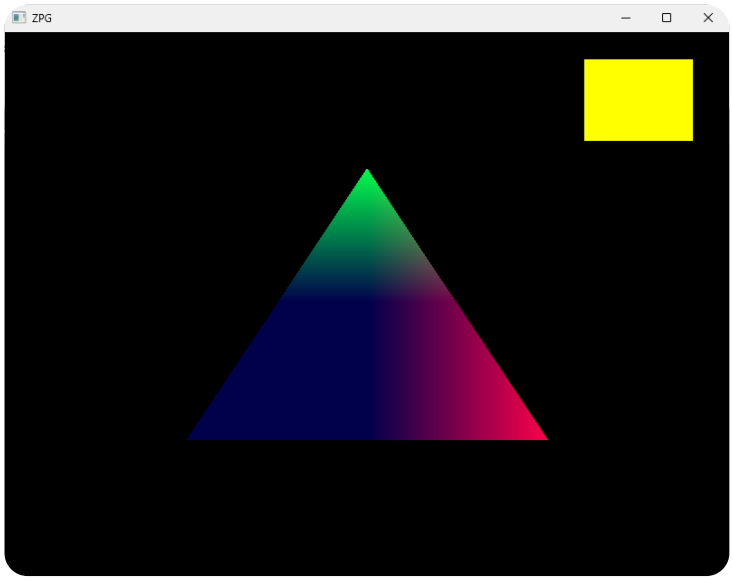
Vytvořte složitější model (čtverec, krychli), upravte barvu, projděte si jednotlivé příkazy, udělejte chybu v shaderu atd.
VS má možnost přeformátování zdrojového kódu (Ctrl+K, Ctrl+D).

//pokud budete mít další objekt (čtverec) můžete si zkusit přidal model s pozici a barvou, v tomto případě musíte upravit jak VBO, tak VAO.

```
const a b[] ={
  { { -.5f, -.5f, .5f, 1 }, { 1, 1, 0, 1 } },
  { { -.5f, .5f, .5f, 1 }, { 1, 0, 0, 1 } },
  { { .5f, .5f, .5f, 1 }, { 0, 0, 0, 1 } },
  { { .5f, -.5f, .5f, 1 }, { 0, 1, 0, 1 } },
};

glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, sizeof(b[0]), (GLvoid*)0);
```

Vyzkoušejte předat pozici z vertex shaderu jako proměnnou do fragment shaderu a tam ji použijte jako barvu pro aktuální fragment.



Zdrojové kódy si vhodně upravte na objektový kód (třídy, metody). Při návrhu OOP se řiďte věcmi, které jsme si řekli na přednášce a cvičeních.

Výsledné zdrojové kódy (CPP a H) odešlete podle termínů na https://kelvin.cs.vsb.cz/.

Příklad souboru main.cpp

```
/**
 * @file main.cpp
 *
 * @brief Main function
 *
 * @author ...
  **/
#include "Application.h"
int main(void)
{
    Application* app = new Application();
    app->initialization(); //OpenGL inicialization

    //Loading scene
    app->createShaders();
    app->createModels();
    app->run(); //Rendering
}
```

Úkoly k procvičování

1. Přidejte si do projektu knihovnu GLEW, abychom mohli používat programovatelný vykreslovací řetězec.
2. Upravte aplikaci tak, aby se vykresloval trojúhelník pomocí shaderu (kompletní návod máte na LMS).
3. Udělejte v shaderu schválně chybu a ověřte, že se aplikace nepustí a že vypíše chybu.
4. Můžete se pokusit změnit tvar, barvu atd.
5. Zaměřte se na svůj objektový kód podle domluvy na cvičení a upravte jej tak, aby jste si zpřehlednili funkčnost a každá třída podle svého názvu odpovídala za přidělenou část (aplikce, shadery, modely atd.).
6. Zkuste si vytvořit více modelů a vykreslovat je, vytvořte si jednoduchá primitiva, jako trojúhelník, čtverec, krychle atd. (příště již přidáme složitější modely).
7. Vyzkoušejte si vykreslete více modelů pomocí více shader programu (shader programy se můžou lišit třeba výslednou barvou ve fragment shaderu).