ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| к.т.н |  |  |  | С.В. Щекин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8 |
| Освоение работы с программируемыми шейдерами |
| по дисциплине: КОМПЬЮТЕРНАЯ ГРАФИКА |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

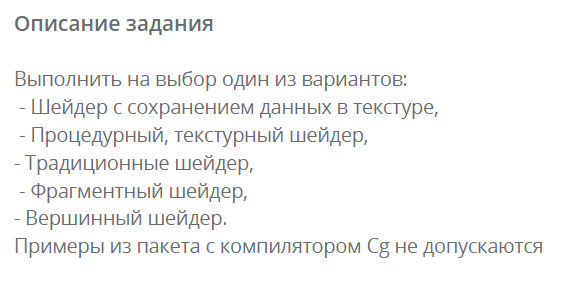
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4131 |  |  |  | Д.А. Кузнецов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**1. Цель работы:**

Реализовать один из предложенных шейдеров на практике

**2. Задание на лабораторную работу:**



**3. Листинг:**

**Main.cpp**

1. #include <iostream>

2.

3. #include <GL/glew.h>

4. #include <GLFW/glfw3.h>

5.

6. #include "Scene.h"

7. #include "Shader.h"

8. #include "Window.h"

9.

10. int main() {

11.

12. GLFWwindow\* window = getWindow();

13.

14. renderScene(window);

15.

16. glfwTerminate();

17.

18. return 0;

}

**Shader.frag**

1. #version 330 core

2.

3. in vec3 ourColor;

4. in vec2 TexCoord;

5.

6. out vec4 color;

7.

8. uniform sampler2D ourTexture;

9.

10. void main() {

11. color = vec4(ourColor, 1.0f);

12. };

**Shader.vert**

1. #version 330 core

2.

3. layout (location = 0) in vec3 position;

4. layout (location = 1) in vec3 color;

5. layout (location = 2) in vec2 texCoord;

6.

7. out vec3 ourColor;

8. out vec2 TexCoord;

9.

10. uniform mat4 model;

11. uniform mat4 view;

12. uniform mat4 projection;

13. uniform mat4 transform;

14.

15. void main() {

16. gl\_Position = projection \* view \* model \* transform \* vec4(position, 1.0f);

17. ourColor = color;

18. TexCoord = texCoord;

19. };

**ShaderRealization.cpp**

1. #include <string>

2. #include <fstream>

3. #include <sstream>

4. #include <iostream>

5.

6. #include <GL/glew.h>;

7. #include "Shader.h";

8.

9. Shader::Shader(const GLchar\* vertexPath, const GLchar\* fragmentPath) {

10.

11. // 1. Получаем исходный код шейдера из filePath

12. std::string vertexCode;

13. std::string fragmentCode;

14. std::ifstream vShaderFile;

15. std::ifstream fShaderFile;

16.

17. // Удостоверимся, что ifstream объекты могут выкидывать исключения

18. vShaderFile.exceptions(std::ifstream::badbit);

19. fShaderFile.exceptions(std::ifstream::badbit);

20. try

21. {

22. // Открываем файлы

23. vShaderFile.open(vertexPath);

24. fShaderFile.open(fragmentPath);

25. std::stringstream vShaderStream, fShaderStream;

26.

27. // Считываем данные в потоки

28. vShaderStream << vShaderFile.rdbuf();

29. fShaderStream << fShaderFile.rdbuf();

30.

31. // Закрываем файлы

32. vShaderFile.close();

33. fShaderFile.close();

34.

35. // Преобразовываем потоки в массив GLchar

36. vertexCode = vShaderStream.str();

37. fragmentCode = fShaderStream.str();

38. }

39. catch (std::ifstream::failure e)

40. {

41. std::cout << "ERROR::SHADER::FILE\_NOT\_SUCCESFULLY\_READ" << std::endl;

42. }

43.

44. const GLchar\* vShaderCode = vertexCode.c\_str();

45. const GLchar\* fShaderCode = fragmentCode.c\_str();

46.

47. // 2. Сборка шейдеров

48. GLuint vertex, fragment;

49. GLint success;

50. GLchar infoLog[512];

51.

52. // Вершинный шейдер

53. vertex = glCreateShader(GL\_VERTEX\_SHADER);

54. glShaderSource(vertex, 1, &vShaderCode, NULL);

55. glCompileShader(vertex);

56.

57. // Если есть ошибки - вывести их

58. glGetShaderiv(vertex, GL\_COMPILE\_STATUS, &success);

59. if (!success)

60. {

61. glGetShaderInfoLog(vertex, 512, NULL, infoLog);

62. std::cout << "ERROR::SHADER::VERTEX::COMPILATION\_FAILED\n" << infoLog << std::endl;

63. };

64.

65. // Фрагментный шейдер

66. fragment = glCreateShader(GL\_FRAGMENT\_SHADER);

67. glShaderSource(fragment, 1, &fShaderCode, NULL);

68. glCompileShader(fragment);

69.

70. // Если есть ошибки - вывести их

71. glGetShaderiv(fragment, GL\_COMPILE\_STATUS, &success);

72. if (!success)

73. {

74. glGetShaderInfoLog(fragment, 512, NULL, infoLog);

75. std::cout << "ERROR::SHADER::FRAGMENT::COMPILATION\_FAILED\n" << infoLog << std::endl;

76. };

77.

78. // Шейдерная программа

79. this->Program = glCreateProgram();

80. glAttachShader(this->Program, vertex);

81. glAttachShader(this->Program, fragment);

82. glLinkProgram(this->Program);

83.

84. //Если есть ошибки - вывести их

85. glGetProgramiv(this->Program, GL\_LINK\_STATUS, &success);

86. if (!success)

87. {

88. glGetProgramInfoLog(this->Program, 512, NULL, infoLog);

89. std::cout << "ERROR::SHADER::PROGRAM::LINKING\_FAILED\n" << infoLog << std::endl;

90. }

91.

92. // Удаляем шейдеры, поскольку они уже в программу и нам больше не нужны.

93. glDeleteShader(vertex);

94. glDeleteShader(fragment);

95.

96. }

97.

98. void Shader::Use() {

99. glUseProgram(this->Program);

100. }

**Window.cpp**

1. #include <iostream>

2.

3. #include <GL/glew.h>

4. #include <GLFW/glfw3.h>

5.

6. #include "Shader.h"

7.

8. void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode);

9.

10. GLFWwindow\* getWindow() {

11.

12. /\* Window \*/

13.

14. // GLFW initialization

15. glfwInit();

16.

17. // Min major and minor openGL version

18. glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

19. glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

20.

21. // Creating profile for context

22. glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

23.

24. // Blocking window resizing

25. glfwWindowHint(GLFW\_RESIZABLE, GL\_FALSE);

26.

27. GLFWwindow\* window = glfwCreateWindow(1920, 1080, "Scene", nullptr, nullptr);

28.

29. if (window == nullptr) {

30. std::cout << "Failed to create GLFW window" << std::endl;

31.

32. glfwTerminate();

33. return nullptr;

34. }

35.

36. glfwMakeContextCurrent(window);

37.

38. glewExperimental = GL\_TRUE;

39.

40. if (glewInit() != GLEW\_OK)

41. {

42. std::cout << "Failed to initialize GLEW" << std::endl;

43. return nullptr;

44. }

45.

46. int width, height;

47. glfwGetFramebufferSize(window, &width, &height);

48.

49. glViewport(0, 0, width, height);

50.

51. glfwSetInputMode(window, GLFW\_STICKY\_KEYS, GLFW\_TRUE);

52. glfwSetKeyCallback(window, key\_callback);

53.

54. return window;

55. }

56.

57. void key\_callback(GLFWwindow\* window, int key, int scancode, int action, int mode) {

58.

59. if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS) {

60. glfwSetWindowShouldClose(window, GL\_TRUE);

61. }

62. }

**Scene.cpp**

1. #include <iostream>

2.

3. #include <GL/glew.h>

4. #include <GLFW/glfw3.h>

5. #include "Shader.h"

6.

7. #include <glm/glm.hpp>

8. #include <glm/gtc/matrix\_transform.hpp>

9. #include <glm/gtc/type\_ptr.hpp>

10. #include <glm/gtx/string\_cast.hpp>

11.

12. #include <SOIL/SOIL.h>;

13.

14. void renderScene(GLFWwindow\* window) {

15.

16. // Window dimensions

17. const GLuint WIDTH = 1920, HEIGHT = 1080;

18.

19. /\* Pyramid \*/

20.

21. Shader my\_shader("D:\\Other\\C++ Scripts\\ComputerGraphicsLabs\\ComputerGraphicsLabs\\Shader.vert",

22. "D:\\Other\\C++ Scripts\\ComputerGraphicsLabs\\ComputerGraphicsLabs\\Shader.frag");

23.

24. // Pyramid

25.

26. GLfloat vertices[] = {

27.

28. //front triangle //Texture chords // Colors

29. 0.0f, 0.7f, 0.0f, 0.0f, 0.5f, 0.0f, 1.0f, 1.0f, // 0

30. -1.0f, -1.0f, 1.0f, 1.0f, 1.0, 1.00, 1.00, 0.00, // 1

31. 1.0f, -1.0f, 1.0f, 1.0f, 0.0f, 0.94, 0.00, 1.00, // 2

32.

33. //right triangle

34. 0.0f, 0.7f, 0.0f, 0.0f, 0.5f, 0.0f, 1.0f, 1.0f, // 3

35. 1.0f, -1.0f, 1.0f, 1.0f, 1.0, 1.00, 1.00, 0.00, // 4

36. 1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 0.94, 0.00, 1.00, // 5

37.

38. //back triangle

39. 0.0f, 0.7f, 0.0f, 0.0f, 0.5f, 0.0f, 1.0f, 1.0f, // 6

40. 1.0f, -1.0f, -1.0f, 1.0f, 1.0, 0.94, 0.00, 1.00, // 7

41. -1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 1.00, 1.00, 0.00, // 8

42.

43. //left triangle

44. 0.0f, 0.7f, 0.0f, 0.0f, 0.5f, 0.0f, 1.0f, 1.0f, // 9

45. -1.0f, -1.0f, -1.0f, 1.0f, 1.0, 1.00, 1.00, 0.00, // 10

46. -1.0f, -1.0f, 1.0f, 1.0f, 0.0f, 0.94, 0.00, 1.00, // 11

47.

48. //bottom triangle

49. -1.0f, -1.0f, 1.0f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, // 12

50. 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0.63f, // 13

51. -1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.59f, 0.0f, 1.0f, // 14

52.

53. //bottom triangle right

54. 0.0f, 0.0f, -1.0f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, // 15

55. 0.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 1.0f, 0.63f, // 16

56. 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.59f, 0.0f, 1.0f, // 17

57.

58. };

59.

60. GLuint indices[] = {

61. 0, 1, 2, // Triangle 1

62. 3, 4, 5, // Triangle 2

63. 6, 7, 8, // Triangle 3

64. 9, 10, 11, // Triangle 4

65. 12, 13, 14, // Triangle 5

66. 15, 16, 17, // Triangle 6

67. };

68.

69. // Texture

70.

71. // Texture atribute

72.

73. GLuint texture;

74. glGenTextures(1, &texture);

75. glBindTexture(GL\_TEXTURE\_2D, texture);

76. // Set the texture wrapping parameters

77. float borderColor[] = {1.0f, 0.0f, 0.0f, 1.0f};

78. glTexParameterfv(GL\_TEXTURE\_2D, GL\_TEXTURE\_BORDER\_COLOR, borderColor);

79. // Set texture filtering parameters

80. glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

81. glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

82. int width, height;

83. unsigned char\* image = SOIL\_load\_image("D:/Other/C++ Scripts/ComputerGraphicsLabs/Obamium/Front.png", &width, &height, 0, SOIL\_LOAD\_RGB);

84. if (image == nullptr) {

85. std::cout << "Texture didnt load";

86. }

87. glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, width, height, 0, GL\_RGB, GL\_UNSIGNED\_BYTE, image);

88. glGenerateMipmap(GL\_TEXTURE\_2D);

89.

90. glBindTexture(GL\_TEXTURE\_2D, 0);

91. SOIL\_free\_image\_data(image);

92.

93. // Generating buffer objects

94. GLuint VBO, VAO, EBO;

95. glGenVertexArrays(1, &VAO);

96. glGenBuffers(1, &VBO);

97. glGenBuffers(1, &EBO);

98.

99. glBindVertexArray(VAO);

100.

101. // Copy data from vertices to buffer

102.

103. glBindBuffer(GL\_ARRAY\_BUFFER, VBO);

104. glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices, GL\_STATIC\_DRAW);

105.

106. glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, EBO);

107. glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(indices), indices, GL\_STATIC\_DRAW);

108.

109. // Position attribute

110. glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 8 \* sizeof(GLfloat), (GLvoid\*)0);

111. glEnableVertexAttribArray(0);

112.

113. // Color attribute

114. glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 8 \* sizeof(GLfloat), (GLvoid\*)(5 \* sizeof(GLfloat)));

115. glEnableVertexAttribArray(1);

116.

117. glVertexAttribPointer(2, 2, GL\_FLOAT, GL\_FALSE, 8 \* sizeof(GLfloat), (GLvoid\*)(3 \* sizeof(GLfloat)));

118. glEnableVertexAttribArray(2);

119.

120.

121. glBindVertexArray(0);

122.

123. // Depth buffer

124. glEnable(GL\_DEPTH\_TEST);

125.

126. // Gaming loop

127. while (!glfwWindowShouldClose(window))

128. {

129. glfwPollEvents();

130.

131. glClearColor(1, 1, 1, 1);

132. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

133.

134. glActiveTexture(GL\_TEXTURE0);

135. glBindTexture(GL\_TEXTURE\_2D, texture);

136.

137. my\_shader.Use();

138.

139. // Create transformations

140. glm::mat4 model = glm::mat4(1);

141. glm::mat4 view = glm::mat4(1);

142. glm::mat4 projection = glm::ortho(0.0f, 100.0f, 0.0f, 600.0f, 0.1f, 100.0f);

143. glm::mat4 transform = glm::mat4(1);

144.

145. model = glm::rotate(model, 1.0f, glm::vec3(0.0f, 1.0f, 0.0f));

146. view = glm::translate(view, glm::vec3(0.0f, 0.0f, -4.0f));

147. projection = glm::perspective(45.0f, (GLfloat)WIDTH / (GLfloat)HEIGHT, 0.1f, 100.0f);

148. transform = glm::rotate(transform, (GLfloat)glfwGetTime() \* 1.6f, glm::vec3(0.0f, 1.0f, 0.0f));

149.

150. // Get their uniform location

151. GLint modelLoc = glGetUniformLocation(my\_shader.Program, "model");

152. GLint viewLoc = glGetUniformLocation(my\_shader.Program, "view");

153. GLint projLoc = glGetUniformLocation(my\_shader.Program, "projection");

154. GLint transformLoc = glGetUniformLocation(my\_shader.Program, "transform");

155.

156. // Pass them to the shaders

157. glUniformMatrix4fv(modelLoc, 1, GL\_FALSE, glm::value\_ptr(model));

158. glUniformMatrix4fv(viewLoc, 1, GL\_FALSE, glm::value\_ptr(view));

159. glUniformMatrix4fv(projLoc, 1, GL\_FALSE, glm::value\_ptr(projection));

160. glUniformMatrix4fv(transformLoc, 1, GL\_FALSE, glm::value\_ptr(transform));

161.

162. // Draw container

163. glBindVertexArray(VAO);

164. glDrawElements(GL\_TRIANGLES, 18, GL\_UNSIGNED\_INT, 0);

165. glBindVertexArray(0);

166.

167. // Swap the screen buffers

168. glfwSwapBuffers(window);

169. }

170.

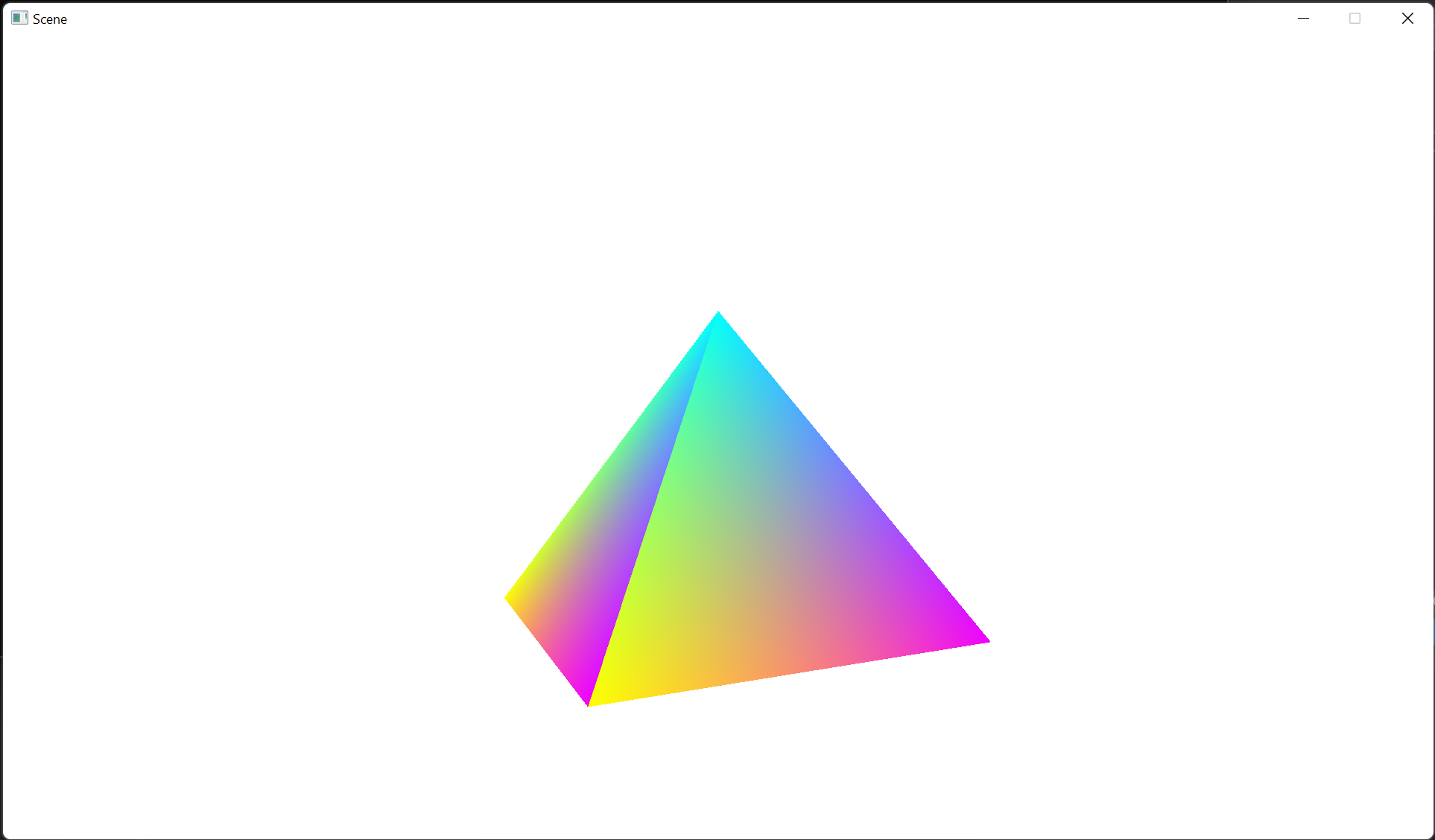
171. glDeleteVertexArrays(1, &VAO);

172. glDeleteBuffers(1, &VBO);

173. glDeleteBuffers(1, &EBO);

174. }

**4. Примеры работы программы**

****

**5. Вывод**

Были приобретены навыки написания собственных шейдеров с использованием OpenGL.