ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

Доцент, канд. техн. наук				В. А. Кузнецов
должность, уч. степень, звание		подпись, дата		инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №3

КОНВЕРТАЦИЯ КАРТЫ ГЛУБИНЫ

Вариант 5

по курсу: Моделирование трехмерных сцен и виртуальная реальность

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	4128				Воробьев В. А.
			подпись, дата		инициалы, фамилия

Санкт-Петербург 2024

СОДЕРЖАНИЕ

1 Введение **3**
1.1 Задание 3
1.2 Решение о смене языка 3

2 Выполнение работы **4**
2.1 Тестирование работы 8

ВЫВОД **12**

ПРИЛОЖЕНИЕ **13**

1 Введение

1.1 Задание

Разработать программу, используя язык программирования согласно варианту, позволяющую:

1. Считать из исходного файла карту глубины заданной размерности. Использовать исходный файл или поток данных стереокамеры. Использование сторонних файлов bmp не допускается!
2. Визуализировать трехмерную оболочку, относящуюся к рассматриваемому объекту (формат карты глубины представлен в разделе 1) с использованием библиотеки OpenGL. Критерием визуализации является возможность проверки правильности считанной карты глубины и возможность сравнения с результатом в выходном файле.
3. Экспортировать оболочку объекта в файл в соответствии с вариантом задания.

Вариант 5: `.obj`, `C#`.

1.2 Решение о смене языка

По причине того, что:

1) Разработка велась на `MacOS`;
2) Недавнему отказу поддержки `Visual Studio` на `MacOS` (Источник (URI - `https://devblogs.microsoft.com/visualstudio/visual-studio-for-mac-retirement-announcement/`));
3) Закончившийся в этом месяце лицензии `JetBrains Rider`;
4) Сырого, неудобного тулинга `C# + NuGet` в `VS Code`.

Было решено выбрать ЯП `Python`.

2 Выполнение работы

В ходе работы был реализован скрипт на Python с использованием библиотек `NumPy` и `PyOpenGl`. Программа была разделена на функции для удобства поддержки и расширения. Исходный код доступен на GitHub (URI - `https://github.com/vladcto/suai-labs/tree/main/7_semester/3D/3`) или в Приложении.

Для начала было составлена схема для `JSON`.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Depth Map Configuration",
  "type": "object",
  "properties": {
    "name": { "type": "string", "description": "Имя файла с картой глубины" }
  },
  "required": ["name"],
  "additionalProperties": false
}
```

Затем были реализованы функции для чтения `JSON` файла и его преобразовании в 2D массив карты глубины.

```
def read_json_file(json_filename):
    with open(json_filename, 'r') as file:
        data = json.load(file)
    return data['name']


def read_depth_map(depth_map_filename):
    with open(depth_map_filename, 'rb') as file:
        height = int(np.fromfile(file, dtype=np.float64, count=1)[0])
        width = int(np.fromfile(file, dtype=np.float64, count=1)[0])
        depth_map_array = np.fromfile(
            file, dtype=np.float64).reshape((height, width))
    return depth_map_array
```

После была написана функция для конвертации 2D массива в файл формата `.obj`. Для этого мы сначала составляли список вершин, а затем соединяли их в грани, по факту их смежности в массиве карты глубины.

```
def export_to_obj(filename, depth_map_array):
    height, width = depth_map_array.shape
    vertices = []
    indices = {}

    with open(filename, 'w') as file:
        vertex_id = 1
        for i in range(height):
            for j in range(width):
                depth = depth_map_array[i, j]
                if depth != 0:
                    vertex = (j - width / 2, height / 2 - i,
                      -depth)
                    indices[(i, j)] = vertex_id
                    vertices.append(vertex)
                    file.write(f'v {vertex[0]} {vertex[1]} {
                      vertex[2]}\n')
                    vertex_id += 1

        for i in range(height - 1):
            for j in range(width - 1):
                if (depth_map_array[i, j] != 0 and
                    depth_map_array[i, j + 1] != 0 and
                    depth_map_array[i + 1, j] != 0 and
                        depth_map_array[i + 1, j + 1] != 0):

                    v1 = indices[(i, j)]
                    v2 = indices[(i, j + 1)]
                    v3 = indices[(i + 1, j + 1)]
                    v4 = indices[(i + 1, j)]

                    file.write(f"l {v1} {v2}\n")
                    file.write(f"l {v2} {v3}\n")
                    file.write(f"l {v3} {v4}\n")
                    file.write(f"l {v4} {v1}\n")

                    file.write(f"f {v1} {v2} {v3} {v4}\n")
```

После был реализован код для инициализации холста `OpenGl`. Большая часть кода избыточна, по причине того, что она используется в лабораторной работе №4. Здесь важнее всего, что мы устанавливаем перспективную про-

екцию, устанавливаем цветовое буферизование с использованием RGBA, а также задаем функцию для рендеринга `display_func`.

```
def init_glut(display_func):
    glutInit()
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
    glutInitWindowSize(window_width, window_height)
    glutInitWindowPosition(100, 100)
    glutCreateWindow(b"3D Depth Map Visualization")
    glutDisplayFunc(display_func)
    glEnable(GL_DEPTH_TEST)
    glClearColor(0.0, 0.0, 0.0, 0.0)

    glMatrixMode(GL_PROJECTION)
    gluPerspective(45, (window_width / window_height), 0.1,
        1000.0)
    glMatrixMode(GL_MODELVIEW)
```

Затем реализовали последнюю функцию `display_func`, в которой мы очищаем изначальный кадр, устанавливаем позицию и направление камеры, настраиваем буфер, для отрисовки граней, и наконец используя такой же способ, как и в экспорте `.obj`, рисуем грани нашей оболочки.

```
def display(depth_map_array):
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glLoadIdentity()

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    # -500 as a crutch so not to deal with positioning
    gluLookAt(
        0, 0, -500,
        0, 0, 0,
        0, 1, 0,
    )

    height, width = depth_map_array.shape
    max_depth = np.max(depth_map_array) if np.max(
        depth_map_array) != 0 else 1
    scale_factor = 200

    glBegin(GL_QUADS)
```

```
20      for i in range(height - 1):
21          for j in range(width - 1):
22              if (depth_map_array[i, j] != 0 and
23                  depth_map_array[i, j + 1] != 0 and
24                  depth_map_array[i + 1, j] != 0 and
25                      depth_map_array[i + 1, j + 1] != 0):
26
27                  vertices = [
28                      ((j - width / 2) / width, (height / 2 - i
                            ) /
29                       height, -depth_map_array[i, j] /
                            max_depth),
30                      (((j + 1) - width / 2) / width, (height /
                            2 - i) /
31                       height, -depth_map_array[i, j + 1] /
                            max_depth),
32                      (((j + 1) - width / 2) / width, (height /
                            2 - (i + 1)) /
33                       height, -depth_map_array[i + 1, j + 1] /
                             max_depth),
34                      ((j - width / 2) / width, (height / 2 - (
                            i + 1)) /
35                       height, -depth_map_array[i + 1, j] /
                            max_depth)
36                  ]
37
38                  for vertex in vertices:
39                      glVertex3f(vertex[0] * scale_factor,
                            vertex[1]
40                                 * scale_factor, vertex[2] *
                                      scale_factor)
41      glEnd()
42
43      glutSwapBuffers()
```

И в заключение, собираем все наши функции для выполнения программы:

```
1  json_filename = 'depth_map_info.json'
2  depth_map_filename = read_json_file(json_filename)
3  depth_map_array = read_depth_map(depth_map_filename)
4
5  export_to_obj('output.obj', depth_map_array)
```

```
6
7 init_glut(lambda: display(depth_map_array))
8 glutMainLoop()
```

2.1 Тестирование работы

Для тестирования мы провели следующее:

1) Составили наш тестовый `JSON` файл.
2) Проверили отрисовку нашей оболочки, используя `OpenGl`.
3) Удостоверились в верности п.2, открыв наш `.obj` файл в `Blender`.

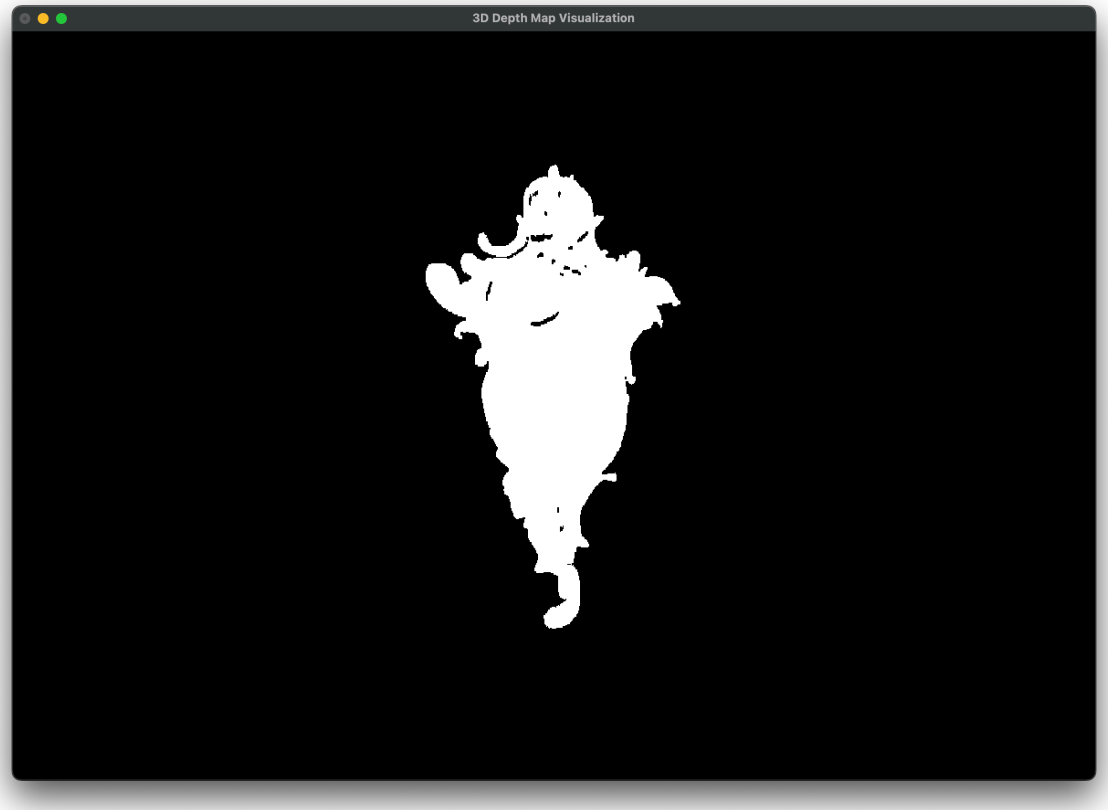

Рисунок 2.1 - Визуализация оболочки в нашей программе

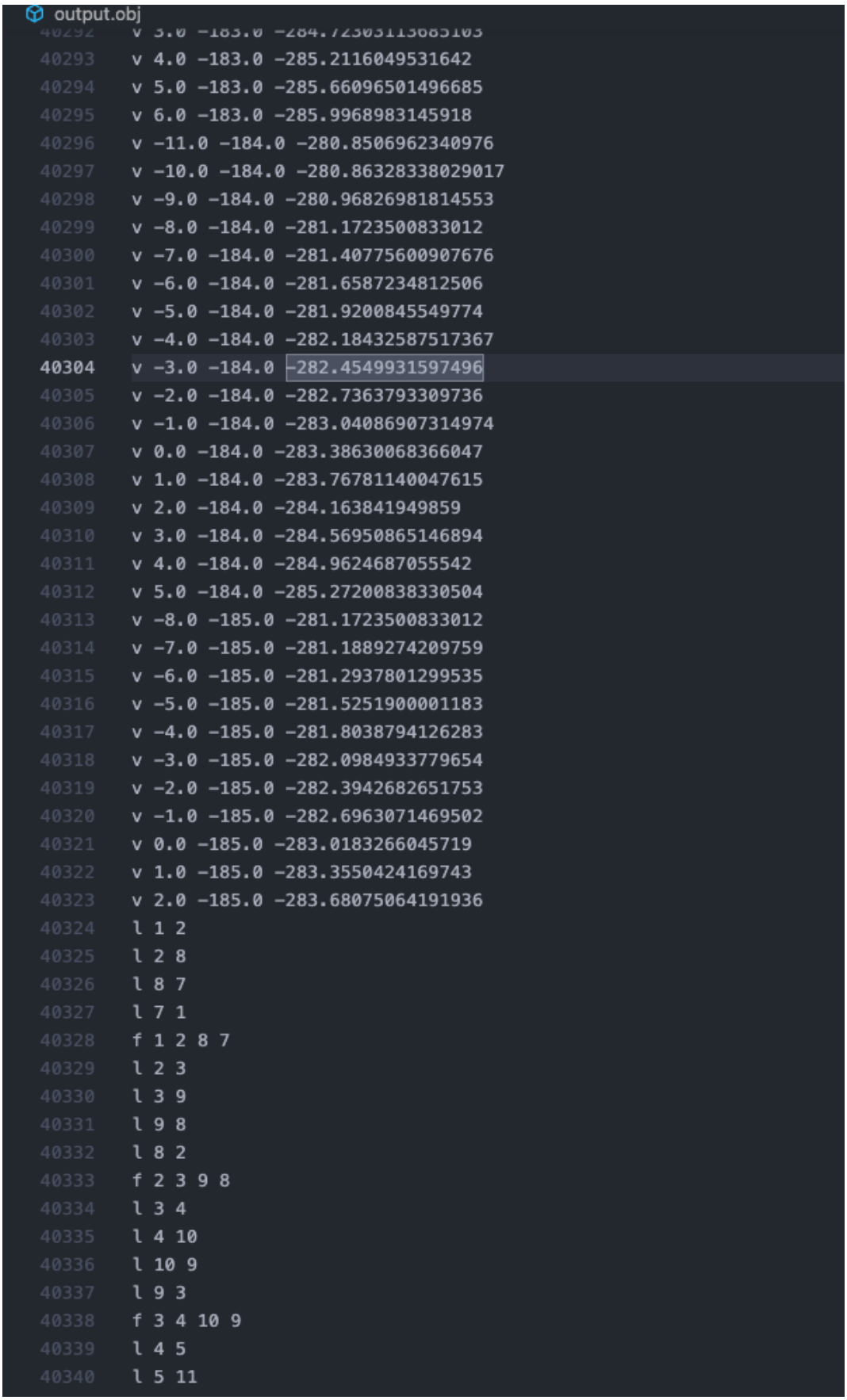

Рисунок 2.2 - Фрагмент .obj файла

Рисунок 2.3 - Модель в Blender, полученная из .obj

Как мы видим - модель совпадает, а значит программа верна. Сплошная

заливка белым цветом нашей модели обусловлена тем, что в этой лабораторной работе еще не настроено освещение.

ВЫВОД

В результате выполнения лабораторной работы была создана программа на языке `Python`, считывающая и визуализирующая карту глубины, а также преобразующая его в формат `.obj`. Также были освоены базовые навыки работы с картами глубины и библиотекой `OpenGL`.

Получившийся исходных код было выложен на GitHub (URI - `https://github.com/vladcto/suai-labs/tree/main/7_semester/3D/3`), а также представлен в Приложении.

ПРИЛОЖЕНИЕ

Листинг solve.py:

```
1  from OpenGL.GL import *
2  from OpenGL.GLUT import *
3  from OpenGL.GLU import *
4  import numpy as np
5  import json
6
7  window_width = 1020
8  window_height = 820
9
10
11 def read_json_file(json_filename):
12     with open(json_filename, 'r') as file:
13         data = json.load(file)
14     return data['name']
15
16
17 def read_depth_map(depth_map_filename):
18     with open(depth_map_filename, 'rb') as file:
19         height = int(np.fromfile(file, dtype=np.float64,
              count=1)[0])
20         width = int(np.fromfile(file, dtype=np.float64, count
              =1)[0])
21         depth_map_array = np.fromfile(
22             file, dtype=np.float64).reshape((height, width))
23     return depth_map_array
24
25
26 def export_to_obj(filename, depth_map_array):
27     height, width = depth_map_array.shape
28     vertices = []
29     indices = {}
30
31     with open(filename, 'w') as file:
32         vertex_id = 1
33         for i in range(height):
34             for j in range(width):
35                 depth = depth_map_array[i, j]
36                 if depth != 0:
```

```
37                      vertex = (j - width / 2, height / 2 - i,
                            -depth)
38                      indices[(i, j)] = vertex_id
39                      vertices.append(vertex)
40                      file.write(f'v {vertex[0]} {vertex[1]} {
                            vertex[2]}\n')
41                      vertex_id += 1
42
43          for i in range(height - 1):
44              for j in range(width - 1):
45                  if (depth_map_array[i, j] != 0 and
46                      depth_map_array[i, j + 1] != 0 and
47                      depth_map_array[i + 1, j] != 0 and
48                          depth_map_array[i + 1, j + 1] != 0):
49
50                      v1 = indices[(i, j)]
51                      v2 = indices[(i, j + 1)]
52                      v3 = indices[(i + 1, j + 1)]
53                      v4 = indices[(i + 1, j)]
54
55                      file.write(f"l {v1} {v2}\n")
56                      file.write(f"l {v2} {v3}\n")
57                      file.write(f"l {v3} {v4}\n")
58                      file.write(f"l {v4} {v1}\n")
59
60                      file.write(f"f {v1} {v2} {v3} {v4}\n")
61
62
63  def display(depth_map_array):
64      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
65      glLoadIdentity()
66
67      glMatrixMode(GL_MODELVIEW)
68      glLoadIdentity()
69
70      # -500 as a crutch so not to deal with positioning
71      gluLookAt(
72          0, 0, -500,
73          0, 0, 0,
74          0, 1, 0,
75      )
```

```
76
77      height, width = depth_map_array.shape
78      max_depth = np.max(depth_map_array) if np.max(
            depth_map_array) != 0 else 1
79      scale_factor = 200
80
81      glBegin(GL_QUADS)
82      for i in range(height - 1):
83          for j in range(width - 1):
84              if (depth_map_array[i, j] != 0 and
85                  depth_map_array[i, j + 1] != 0 and
86                  depth_map_array[i + 1, j] != 0 and
87                      depth_map_array[i + 1, j + 1] != 0):
88
89                  vertices = [
90                      ((j - width / 2) / width, (height / 2 - i
                            ) /
91                       height, -depth_map_array[i, j] /
                            max_depth),
92                      (((j + 1) - width / 2) / width, (height /
                            2 - i) /
93                       height, -depth_map_array[i, j + 1] /
                            max_depth),
94                      (((j + 1) - width / 2) / width, (height /
                            2 - (i + 1)) /
95                       height, -depth_map_array[i + 1, j + 1] /
                             max_depth),
96                      ((j - width / 2) / width, (height / 2 - (
                            i + 1)) /
97                       height, -depth_map_array[i + 1, j] /
                            max_depth)
98                  ]
99
100                 for vertex in vertices:
101                     glVertex3f(vertex[0] * scale_factor,
                            vertex[1]
102                                * scale_factor, vertex[2] *
                                    scale_factor)
103     glEnd()
104
105     glutSwapBuffers()
```

```
106
107
108 def init_glut(display_func):
109     glutInit()
110     glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
111     glutInitWindowSize(window_width, window_height)
112     glutInitWindowPosition(100, 100)
113     glutCreateWindow(b"3D Depth Map Visualization")
114     glutDisplayFunc(display_func)
115     glEnable(GL_DEPTH_TEST)
116     glClearColor(0.0, 0.0, 0.0, 0.0)
117
118     glMatrixMode(GL_PROJECTION)
119     gluPerspective(45, (window_width / window_height), 0.1,
          1000.0)
120     glMatrixMode(GL_MODELVIEW)
121
122     glPointSize(2.0)
123
124
125 json_filename = 'depth_map_info.json'
126 depth_map_filename = read_json_file(json_filename)
127 depth_map_array = read_depth_map(depth_map_filename)
128
129 export_to_obj('output.obj', depth_map_array)
130
131 init_glut(lambda: display(depth_map_array))
132 glutMainLoop()
```

Листинг depth_map_info.json:

```
1 {
2   "name": "test.dat"
3 }
```