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 №4

ФОРМИРОВАНИЕ ИЗОБРАЖЕНИЯ ТРЕХМЕРНОЙ ПОВЕРХНОСТИ

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

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, см. раздел 1. Использование сторонних файлов bmp не допускается! Имя файла карты глубины задается в файле JSON.
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. Сформировать согласно заданным в файле JSON положению источника света и наблюдателя изображение объекта в формате .bmp или попиксельно отобразить в Canvas в соответствии с вариантом. Возможная модель отражения поверхности выбираются в соответствии с вариантом и конфигурируется в файле JSON, обработка файла конфигурации JSON позволяет осуществлять выбор одной из трех моделей отражения.
3. Экспортировать оболочку объекта в файл в соответствии с вариантом задания. Имя и формат выходного файла задается в файле JSON, необходимо предоставить выбор как минимум из трех форматов выходного файла, для хранения трехмерной оболочки.

Вариант 5: `.obj`, `C#`, модели отражения: Ламберта; Фонга; Торенса-Сперроу, Canvas.

Также как два дополнительных формата вывода был выбран `.stl` и `.ply`.

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. Также для соблюдения инкапсуляции было решено разделить программу на несколько библиотек. Исходный код доступен на GitHub (URI - `https://github.com/vladcto/suai-labs/tree/main/7_semester/3D/4`) или в Приложении.

Для начала была 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" },
      "light_source":
        {
          "type": "object",
          "properties":
            {
              "position":
                {
                  "type": "array",
                  "description": "Трехмерная позиция
                     источника света",
                  "items": { "type": "number" },
                  "minItems": 3,
                  "maxItems": 3
                }
            },
          "required": ["position"],
          "additionalProperties": false
        },
      "viewer":
        {
          "type": "object",
          "properties":
            {
```

```
30              "position":
31                {
32                  "type": "array",
33                  "description": "Трехмерная позиция
                      наблюдателя",
34                  "items": { "type": "number" },
35                  "minItems": 3,
36                  "maxItems": 3
37                }
38            },
39          "required": ["position"],
40          "additionalProperties": false
41        },
42      "reflection_model":
43        {
44          "type": "string",
45          "description": "Модель отражения для использования
                в рендеринге",
46          "enum": ["lambert", "phong", "torrance-sparrow"]
47        },
48      "export_format":
49        {
50          "type": "string",
51          "description": "Формат для экспорта файлов",
52          "enum": ["obj", "stl", "ply"]
53        },
54      "export_name":
55        {
56          "type": "string",
57          "description": "Имя файла для экспорта без
                расширения"
58        }
59    },
60  "required":
61    [
62      "name",
63      "light_source",
64      "viewer",
65      "reflection_model",
66      "export_format",
67      "export_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'], data


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
```

Текущий код поддерживает настройку из JSON и немного забегая вперед покажем как это выглядит.

```
# Read map
json_filename = 'depth_map_info.json'
depth_map_filename, config = read_json_file(json_filename)
depth_map_array = read_depth_map(depth_map_filename)

# Export map
export_format = config.get('export_format', 'obj')
output_filename = config.get('export_name', 'output')
export_depth_map_to_file(depth_map_array, export_format,
    output_filename)

# Show object
light_position = config['light_source']['position']
viewer_position = config['viewer']['position']
reflection_model = config.get('reflection_model', 'phong')
```

Затем была реализована библиотека для выбора формата экспорта карты глубины. Всего было реализовано 4 функции, 1 функция-маппер и 3 функ-

ции экспорта, составленные на основе официальной спецификации. Высокоуровневое описание работы алгоритма было представлено в лабораторной работе №3. Выбор формата и имени определяется на основе входного JSON файла.

```
import numpy as np


def _calculate_normal(p1, p2, p3):
    u = np.array(p2) - np.array(p1)
    v = np.array(p3) - np.array(p1)
    return np.cross(u, v)

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)]
```

```
36                      v4 = indices[(i + 1, j)]
37
38                      file.write(f"l {v1} {v2}\n")
39                      file.write(f"l {v2} {v3}\n")
40                      file.write(f"l {v3} {v4}\n")
41                      file.write(f"l {v4} {v1}\n")
42
43                      file.write(f"f {v1} {v2} {v3} {v4}\n")
44
45
46  def _export_to_stl(filename, depth_map_array):
47      # TODO: extract open file in new method
48      height, width = depth_map_array.shape
49      with open(filename, 'w') as file:
50          file.write('solid depth_map\n')
51          for i in range(height - 1):
52              for j in range(width - 1):
53                  if (depth_map_array[i, j] != 0 and
54                      depth_map_array[i, j + 1] != 0 and
55                      depth_map_array[i + 1, j] != 0 and
56                          depth_map_array[i + 1, j + 1] != 0):
57
58                      vertices = [
59                          [j - width / 2, height / 2 - i, -
                                depth_map_array[i, j]],
60                          [j + 1 - width / 2, height / 2 -
61                              i, -depth_map_array[i, j + 1]],
62                          [j + 1 - width / 2, height / 2 -
63                              (i + 1), -depth_map_array[i + 1,
                                    j + 1]],
64                          [j - width / 2, height / 2 -
65                              (i + 1), -depth_map_array[i + 1,
                                    j]]
66                      ]
67
68                      for k in range(2):
69                          triangle = (
70                              vertices[0], vertices[k + 1],
                                    vertices[k + 2])
71                          normal = _calculate_normal(*triangle)
72                          normal_string = ' '.join(map(str,
```

```
                                normal))
73                              file.write(f'facet normal {
                                    normal_string}\n')
74                              file.write('  outer loop\n')
75                              for vertex in triangle:
76                                  vertex_string = ' '.join(map(str,
                                        vertex))
77                                  file.write(f'    vertex {
                                        vertex_string}\n')
78                              file.write('  endloop\n')
79                              file.write('endfacet\n')
80          file.write('endsolid depth_map\n')
81
82
83  def _export_to_ply(filename, depth_map_array):
84      height, width = depth_map_array.shape
85      vertices = []
86      indices = {}
87      faces = []
88
89      for i in range(height):
90          for j in range(width):
91              depth = depth_map_array[i, j]
92              if depth != 0:
93                  vertex = (j - width / 2, height / 2 - i, -
                        depth)
94                  indices[(i, j)] = len(vertices)
95                  vertices.append(vertex)
96
97      for i in range(height - 1):
98          for j in range(width - 1):
99              if (depth_map_array[i, j] != 0 and
100                 depth_map_array[i, j + 1] != 0 and
101                 depth_map_array[i + 1, j] != 0 and
102                     depth_map_array[i + 1, j + 1] != 0):
103
104                 v1 = indices[(i, j)]
105                 v2 = indices[(i, j + 1)]
106                 v3 = indices[(i + 1, j + 1)]
107                 v4 = indices[(i + 1, j)]
108
```

```
109                 faces.append((v1, v2, v3, v4))
110
111     with open(filename, 'w') as file:
112         file.write("ply\n")
113         file.write("format ascii 1.0\n")
114         file.write(f"element vertex {len(vertices)}\n")
115         file.write("property float x\n")
116         file.write("property float y\n")
117         file.write("property float z\n")
118         file.write(f"element face {len(faces)}\n")
119         file.write("property list uchar int vertex_index\n")
120         file.write("end_header\n")
121
122         for vertex in vertices:
123             file.write(f"{vertex[0]} {vertex[1]} {vertex[2]}\
                n")
124
125         for face in faces:
126             file.write(f"4 {face[0]} {face[1]} {face[2]} {
                face[3]}\n")
127
128
129 def export_depth_map_to_file(depth_map_array, export_format,
    output_filename):
130     output_path = f"{output_filename}.{export_format}"
131
132     if export_format == 'obj':
133         _export_to_obj(output_path, depth_map_array)
134     elif export_format == 'stl':
135         _export_to_stl(output_path, depth_map_array)
136     elif export_format == 'ply':
137         _export_to_ply(output_path, depth_map_array)
```

Затем по подобию с функциями для экспорта, были составлены функции для моделей отражения. Кастомизация параметров задается напрямую в исходном коде, за исключением направления источника освещения - оно задается из `JSON`. Текущая настройка позволяет создать максимально схожую модель освещения между тремя моделями.

```
1 import numpy as np
2
3
```

```
4  def _lambert_reflection(normal, light_source):
5      light_vector = np.array(light_source) / np.linalg.norm(
           light_source)
6      normal_vector = np.array(normal) / np.linalg.norm(normal)
7      dot = np.dot(normal_vector, light_vector)
8      intensity = max(dot, 0)
9      return (intensity, intensity, intensity)
10
11
12 def _torrance_sparrow_reflection(normal, light_source, viewer
       ):
13     light_vector = np.array(light_source) / np.linalg.norm(
           light_source)
14     normal_vector = np.array(normal) / np.linalg.norm(normal)
15     viewer_vector = np.array(viewer) / np.linalg.norm(viewer)
16
17     roughness = 0.5
18     f0 = 0.04
19
20     half_vector = (light_vector + viewer_vector) / \
21         np.linalg.norm(light_vector + viewer_vector)
22     dot_l_n = max(np.dot(light_vector, normal_vector), 0)
23     dot_v_n = max(np.dot(viewer_vector, normal_vector), 0)
24     dot_h_n = max(np.dot(half_vector, normal_vector), 0)
25     dot_h_v = max(np.dot(half_vector, viewer_vector), 0)
26
27     F = f0 + (1 - f0) * (1 - dot_h_v) ** 5
28
29     G = min(1, min((2 * dot_h_n * dot_v_n) / dot_h_v,
30             (2 * dot_h_n * dot_l_n) / dot_h_v))
31
32     alpha = roughness ** 2
33     denom = (dot_h_n ** 2) * (alpha ** 2) + (1 - (dot_h_n) **
           2)
34     D = alpha ** 2 / (np.pi * (denom ** 2))
35
36     specular_intensity = (F * D * G) / (4 * dot_l_n * dot_v_n
           + 1e-7)
37     diffuse_intensity = dot_l_n
38
39     intensity = 0.1 + 0.9 * (diffuse_intensity +
```

```
        specular_intensity)
40      intensity = min(intensity, 1)
41
42      return (intensity, intensity, intensity)
43
44
45  def _phong_reflection(normal, light_source, viewer):
46      light_vector = np.array(light_source) / np.linalg.norm(
            light_source)
47      normal_vector = np.array(normal) / np.linalg.norm(normal)
48      viewer_vector = np.array(viewer) / np.linalg.norm(viewer)
49
50      ambient = 0.1
51      diffuse_coefficient = 0.7
52      specular_coefficient = 0.2
53      shininess = 32
54
55      ambient_color = ambient
56
57      dot_l_n = max(np.dot(light_vector, normal_vector), 0)
58      diffuse_color = diffuse_coefficient * dot_l_n
59
60      reflection_vector = 2 * normal_vector * dot_l_n -
            light_vector
61      dot_r_v = max(np.dot(reflection_vector, viewer_vector),
            0)
62      specular_color = specular_coefficient * (dot_r_v **
            shininess)
63
64      intensity = ambient_color + diffuse_color +
            specular_color
65      intensity = min(intensity, 1)
66
67      return (intensity, intensity, intensity)
68
69  def get_color(normal, light_source, viewer, model):
70      if model == 'lambert':
71          return _lambert_reflection(normal, light_source)
72      elif model == 'phong':
73          return _phong_reflection(normal, light_source, viewer
                )
```

```
74      elif model == 'torrance-sparrow':
75          return _torrance_sparrow_reflection(normal,
              light_source, viewer)
```

Затем на основе библиотеки выше, мы смогли обновить код функции рендеринга для поддержки освещенности грани `get_color`. Также было добавлено управление камерой на основе входных данных. В остальном логика функции не изменилась от 3 лабораторной работы.

```
1   def calculate_normal(p1, p2, p3):
2       u = np.array(p2) - np.array(p1)
3       v = np.array(p3) - np.array(p1)
4       return np.cross(u, v)
5
6   def display(depth_map_array, light_position, viewer_position,
        reflection_model):
7       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
8       glLoadIdentity()
9
10      glMatrixMode(GL_MODELVIEW)
11      glLoadIdentity()
12
13      gluLookAt(viewer_position[0], viewer_position[1],
            viewer_position[2],
14                0, 0, 0,
15                0, 1, 0)
16
17      height, width = depth_map_array.shape
18      max_depth = np.max(depth_map_array) if np.max(
            depth_map_array) != 0 else 1
19      scale_factor = 200
20
21      glBegin(GL_QUADS)
22      for i in range(height - 1):
23          for j in range(width - 1):
24              if (depth_map_array[i, j] != 0 and
25                  depth_map_array[i, j + 1] != 0 and
26                  depth_map_array[i + 1, j] != 0 and
27                      depth_map_array[i + 1, j + 1] != 0):
28
29                  vertices = [
30                      ((j - width / 2) / width, (height / 2 - i
```

```
                        ) /
31                       height, -depth_map_array[i, j] /
                            max_depth),
32                      (((j + 1) - width / 2) / width, (height /
                            2 - i) /
33                       height, -depth_map_array[i, j + 1] /
                            max_depth),
34                      (((j + 1) - width / 2) / width, (height /
                            2 - (i + 1)) /
35                       height, -depth_map_array[i + 1, j + 1] /
                             max_depth),
36                      ((j - width / 2) / width, (height / 2 - (
                            i + 1)) /
37                       height, -depth_map_array[i + 1, j] /
                            max_depth)
38                  ]
39
40                  normal = calculate_normal(
41                      vertices[0], vertices[1], vertices[2])
42                  color = get_color(normal, light_position,
43                                    viewer_position,
                                        reflection_model)
44
45                  glColor3f(*color)
46                  for vertex in vertices:
47                      glVertex3f(vertex[0] * scale_factor,
                            vertex[1]
48                                 * scale_factor, vertex[2] *
                                    scale_factor)
49      glEnd()
50
51      glutSwapBuffers()
```

Затем мы подключили написанные нами библиотеки и составили программу из функций. На вход она принимает JSON конфигурацию, а затем на основе карты глубины составляет визуализацию модели с освещением, а также требуемый файл нужного формата.

```
1  # Read map
2  json_filename = 'depth_map_info.json'
3  depth_map_filename, config = read_json_file(json_filename)
4  depth_map_array = read_depth_map(depth_map_filename)
```

```
5
6  # Export map
7  export_format = config.get('export_format', 'obj')
8  output_filename = config.get('export_name', 'output')
9  export_depth_map_to_file(depth_map_array, export_format,
       output_filename)
10
11 # Show object
12 light_position = config['light_source']['position']
13 viewer_position = config['viewer']['position']
14 reflection_model = config.get('reflection_model', 'phong')
15
16 init_glut(
17     lambda: display(
18         depth_map_array,
19         light_position,
20         viewer_position,
21         reflection_model
22     )
23 )
24 glutMainLoop()
```

2.1 Тестирование работы

Для тестирования мы провели следующее:

1) Составили наш тестовый `JSON` файл. Сделали 3 дубликата, с разными типами форматов и моделей отражения.
2) Проверили отрисовку нашей оболочки используя `OpenGl`. Повторили для каждой модели отражения.
3) Проверили правильность экспорта карты глубины в 3 формата в `Blender`.

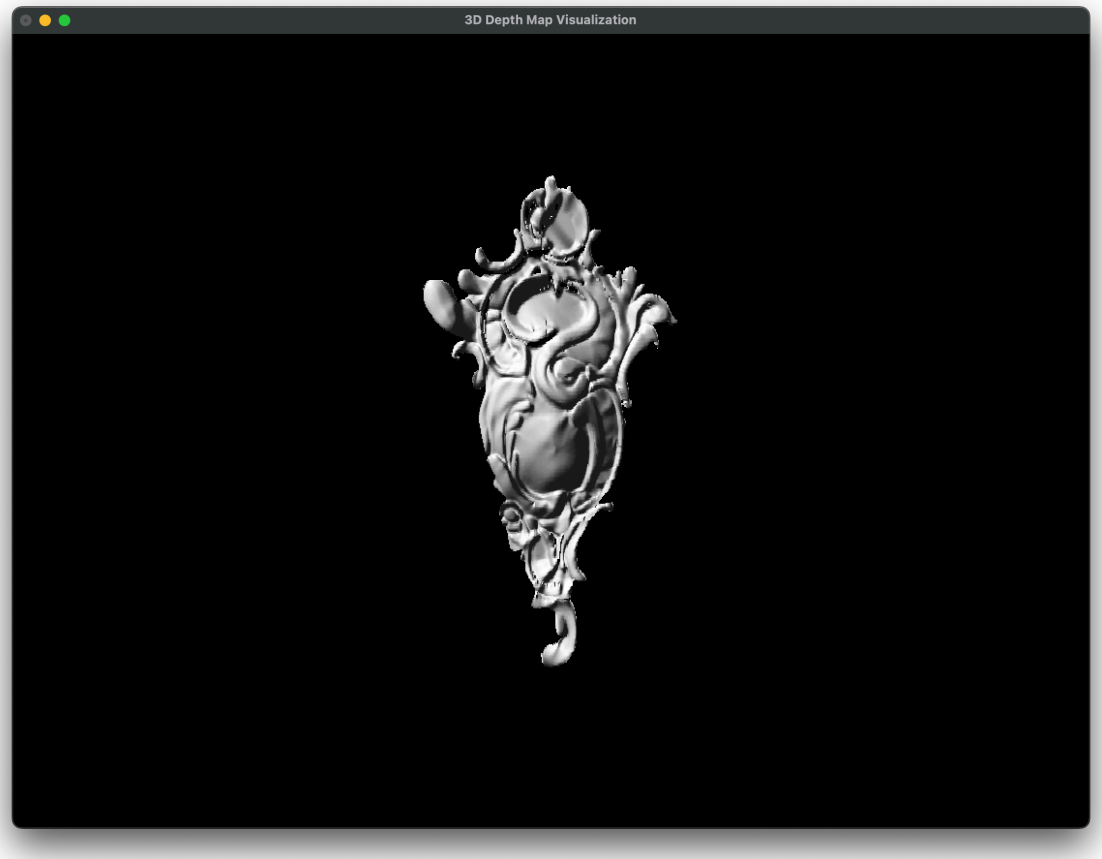

Рисунок 2.1 - торенс

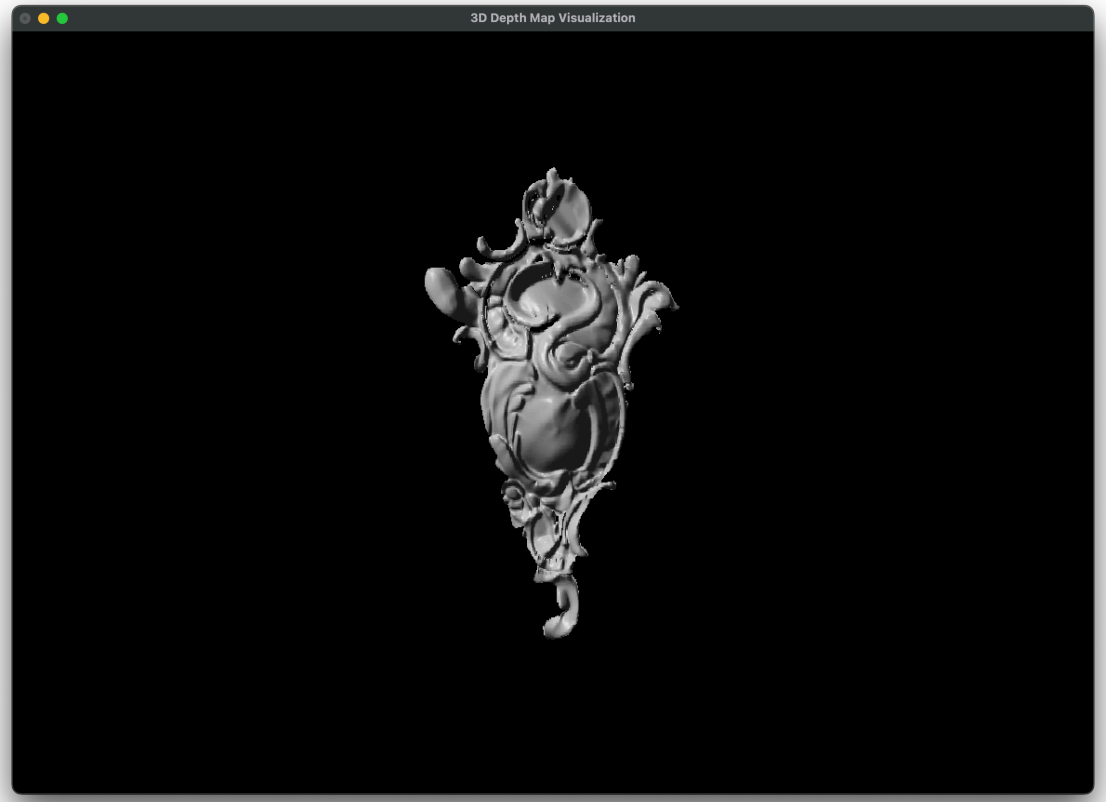

Рисунок 2.2 - фонг

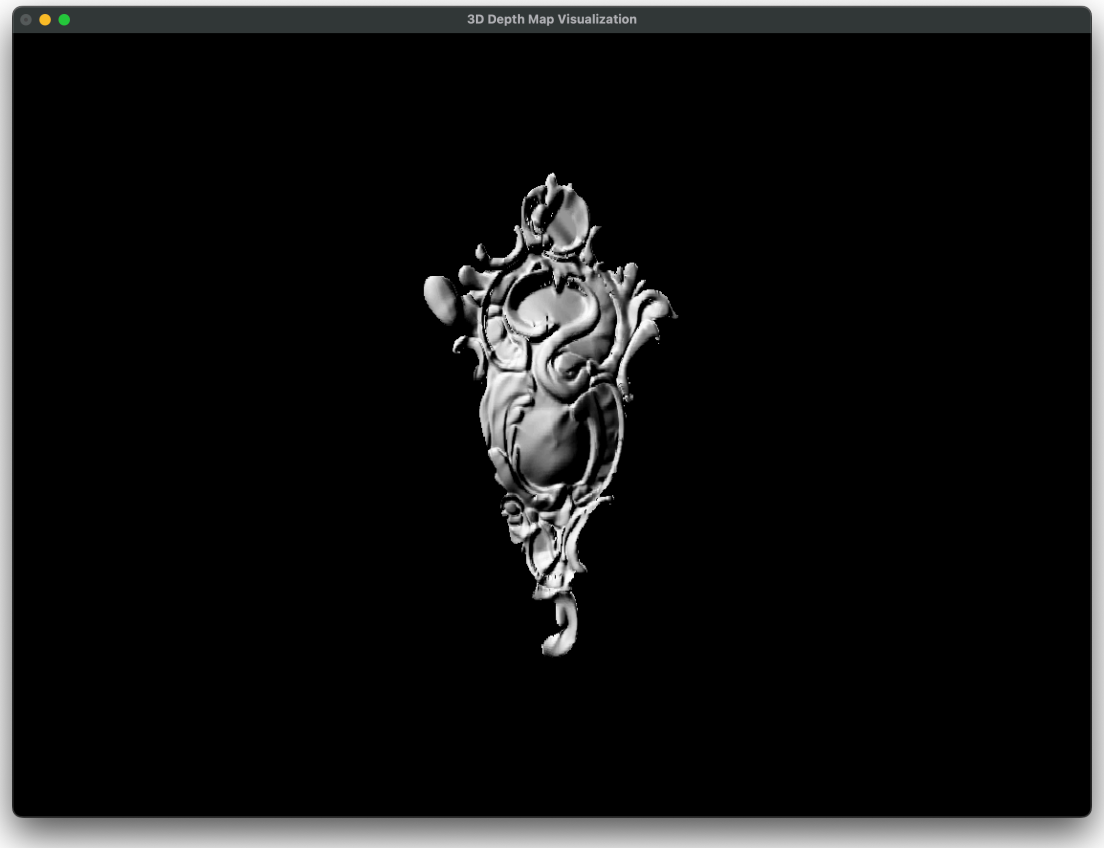

Рисунок 2.3 - Ламберт

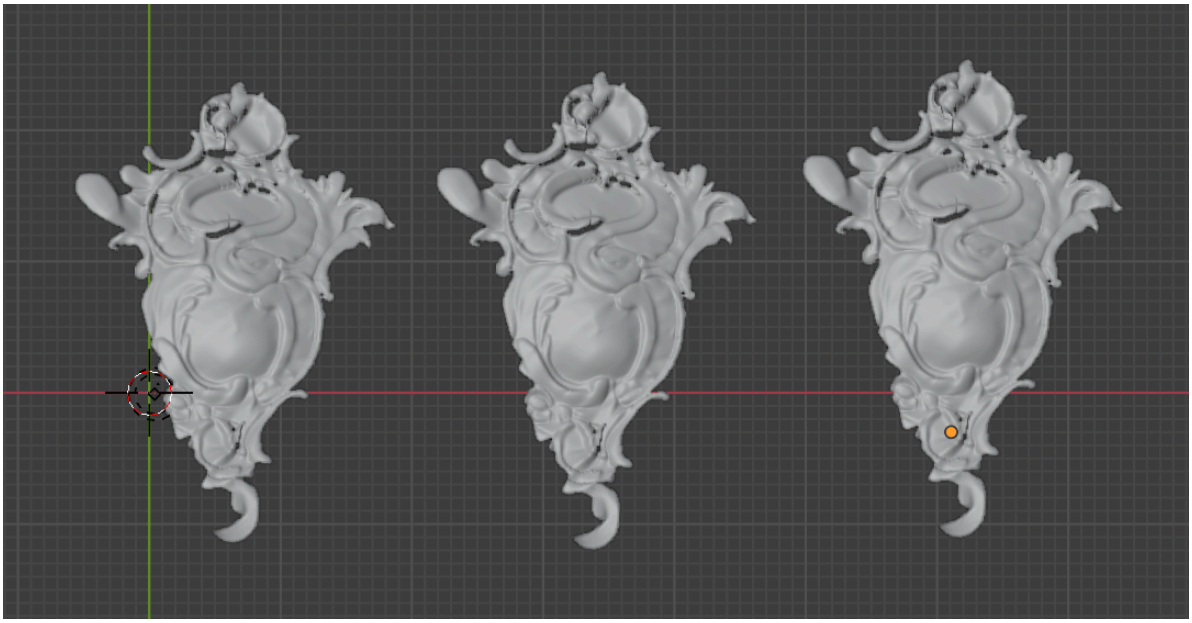

Рисунок 2.4 - Полученные модели в Blender

output.obj

```
40291    v 2.0 -183.0 -284.2350618210252
40292    v 3.0 -183.0 -284.72303113685103
40293    v 4.0 -183.0 -285.2116049531642
40294    v 5.0 -183.0 -285.66096501496685
40295    v 6.0 -183.0 -285.9968983145918
40296    v -11.0 -184.0 -280.8506962340976
40297    v -10.0 -184.0 -280.86328338029017
40298    v -9.0 -184.0 -280.96826981814553
40299    v -8.0 -184.0 -281.1723500833012
40300    v -7.0 -184.0 -281.40775600907676
40301    v -6.0 -184.0 -281.6587234812506
40302    v -5.0 -184.0 -281.9200845549774
40303    v -4.0 -184.0 -282.18432587517367
40304    v -3.0 -184.0 -282.4549931597496
40305    v -2.0 -184.0 -282.7363793309736
40306    v -1.0 -184.0 -283.04086907314974
40307    v 0.0 -184.0 -283.38630068366047
40308    v 1.0 -184.0 -283.76781140047615
40309    v 2.0 -184.0 -284.163841949859
40310    v 3.0 -184.0 -284.56950865146894
40311    v 4.0 -184.0 -284.9624687055542
40312    v 5.0 -184.0 -285.27200838330504
40313    v -8.0 -185.0 -281.1723500833012
40314    v -7.0 -185.0 -281.1889274209759
40315    v -6.0 -185.0 -281.2937801299535
40316    v -5.0 -185.0 -281.5251900001183
40317    v -4.0 -185.0 -281.8038794126283
40318    v -3.0 -185.0 -282.09849337796 54
40319    v -2.0 -185.0 -282.3942682651753
40320    v -1.0 -185.0 -282.6963071469502
40321    v 0.0 -185.0 -283.0183266045719
40322    v 1.0 -185.0 -283.3550424169743
40323    v 2.0 -185.0 -283.68075064191936
40324    l 1 2
40325    l 2 8
40326    l 8 7
40327    l 7 1
40328    f 1 2 8 7
```

output.ply

```
ply
format ascii 1.0
element vertex 40323
property float x
property float y
property float z
element face 39206
property list uchar int vertex_index    "uchar": Unknown wor
end_header
-2.0 203.0 -276.6078247043727
-1.0 203.0 -276.2445829564143
0.0 203.0 -275.4955165420197
1.0 203.0 -274.9213106131411
2.0 203.0 -274.39576528528085
-3.0 202.0 -276.64601231183707
-2.0 202.0 -276.6078247043727
-1.0 202.0 -276.1604437035929
0.0 202.0 -275.82319687955305
1.0 202.0 -275.5623265548546
2.0 202.0 -275.27477400639503
3.0 202.0 -274.8975849812199
4.0 202.0 -274.401972114168
-4.0 201.0 -276.60908531556265
-3.0 201.0 -276.64601231183707
-2.0 201.0 -276.6030378437452
-1.0 201.0 -276.52130667716017
0.0 201.0 -276.4393176025759
1.0 201.0 -276.3683304471415
2.0 201.0 -276.2358307350997
3.0 201.0 -275.9807160488073
4.0 201.0 -275.58005883847414
-4.0 200.0 -276.60908531556265
-3.0 200.0 -276.75175585951865
-2.0 200.0 -276.9700280967541
-1.0 200.0 -277.1010039073679
0.0 200.0 -277.1744669317922
1.0 200.0 -277.21611792933936
2.0 200.0 -277.18892028099685
3.0 200.0 -277.02072402265657
4.0 200.0 -276.671711330382
5.0 200.0 -276.06078111457515
-4.0 199.0 -276.60908531556265
-3.0 199.0 -276.92083911337795
-2.0 199.0 -277.4437365255043
-1.0 199.0 -277.74271715536094
0.0 199.0 -277.9146561816297
1.0 199.0 -278.0255426138645
2.0 199.0 -278.06798908947485
```

```
    outer loop
      vertex -5.0 194.0 -278.70811956875576
      vertex -4.0 193.0 -279.1084522370176
      vertex -5.0 193.0 -278.7369355312753
    endloop
  endfacet
  facet normal -0.3708756190928284 0.16646165003760416 -1.0
    outer loop
      vertex -4.0 194.0 -279.01934657586975
      vertex -3.0 194.0 -279.3902221949626
      vertex -3.0 193.0 -279.5566838450002
    endloop
  endfacet
  facet normal -0.4482316079825637 0.08910566114786889 -1.0
    outer loop
      vertex -4.0 194.0 -279.01934657586975
      vertex -3.0 193.0 -279.5566838450002
      vertex -4.0 193.0 -279.1084522370176
    endloop
  endfacet
  facet normal -0.3436946483084853 0.2319021140934865 -1.0
    outer loop
      vertex -3.0 194.0 -279.3902221949626
      vertex -2.0 194.0 -279.73391684327106
      vertex -2.0 193.0 -279.96581895736455
    endloop
  endfacet
  facet normal -0.4091351123643676 0.16646165003760416 -1.0
    outer loop
      vertex -3.0 194.0 -279.3902221949626
      vertex -2.0 193.0 -279.96581895736455
      vertex -3.0 193.0 -279.5566838450002
    endloop
  endfacet
  facet normal -0.29666977082530366 0.28958596041326246 -1.0
    outer loop
      vertex -2.0 194.0 -279.73391684327106
      vertex -1.0 194.0 -280.03058661409636
      vertex -1.0 193.0 -280.3201725745096
    endloop
  endfacet
  facet normal -0.3543536171450796 0.2319021140934865 -1.0
    outer loop
      vertex -2.0 194.0 -279.73391684327106
      vertex -1.0 193.0 -280.3201725745096
      vertex -2.0 193.0 -279.96581895736455
    endloop
  endfacet
  facet normal -0.26621409322996215 0.3093498147278524 -1.0
```

Как мы видим - модели совпадают, а освещение работает корректно => работа выполнена верно.

ВЫВОД

В результате выполнения лабораторной работы была создана программа на языке Python, считывающая и визуализирующая карту глубины, преобразующая карту глубины в формат .obj, .ply, .stl, а также применяющая 3 разные модели отражения. Вся конфигурация задается из JSON файла.

Получившийся исходный код было выложен на GitHub (URI - https://github.com/vladcto/suai-labs/tree/main/7_semester/3D/4), 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  from model_export import *
5  from reflection_models import *
6  import numpy as np
7  import json
8
9  window_width = 1020
10 window_height = 820
11
12
13 def read_json_file(json_filename):
14     with open(json_filename, 'r') as file:
15         data = json.load(file)
16     return data['name'], data
17
18
19 def read_depth_map(depth_map_filename):
20     with open(depth_map_filename, 'rb') as file:
21         height = int(np.fromfile(file, dtype=np.float64,
              count=1)[0])
22         width = int(np.fromfile(file, dtype=np.float64, count
              =1)[0])
23         depth_map_array = np.fromfile(
24             file, dtype=np.float64).reshape((height, width))
25     return depth_map_array
26
27
28 def calculate_normal(p1, p2, p3):
29     u = np.array(p2) - np.array(p1)
30     v = np.array(p3) - np.array(p1)
31     return np.cross(u, v)
32
33 def display(depth_map_array, light_position, viewer_position,
      reflection_model):
34     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
35     glLoadIdentity()
36
```

```
37      glMatrixMode(GL_MODELVIEW)
38      glLoadIdentity()
39
40      gluLookAt(viewer_position[0], viewer_position[1],
            viewer_position[2],
41                0, 0, 0,
42                0, 1, 0)
43
44      height, width = depth_map_array.shape
45      max_depth = np.max(depth_map_array) if np.max(
            depth_map_array) != 0 else 1
46      scale_factor = 200
47
48      glBegin(GL_QUADS)
49      for i in range(height - 1):
50          for j in range(width - 1):
51              if (depth_map_array[i, j] != 0 and
52                  depth_map_array[i, j + 1] != 0 and
53                  depth_map_array[i + 1, j] != 0 and
54                      depth_map_array[i + 1, j + 1] != 0):
55
56                  vertices = [
57                      ((j - width / 2) / width, (height / 2 - i
                            ) /
58                       height, -depth_map_array[i, j] /
                            max_depth),
59                      (((j + 1) - width / 2) / width, (height /
                            2 - i) /
60                       height, -depth_map_array[i, j + 1] /
                            max_depth),
61                      (((j + 1) - width / 2) / width, (height /
                            2 - (i + 1)) /
62                       height, -depth_map_array[i + 1, j + 1] /
                             max_depth),
63                      ((j - width / 2) / width, (height / 2 - (
                            i + 1)) /
64                       height, -depth_map_array[i + 1, j] /
                            max_depth)
65                  ]
66
67                  normal = calculate_normal(
```

```
68                      vertices[0], vertices[1], vertices[2])
69                  color = get_color(normal, light_position,
70                                    viewer_position,
                                      reflection_model)
71
72                  glColor3f(*color)
73                  for vertex in vertices:
74                      glVertex3f(vertex[0] * scale_factor,
                            vertex[1]
75                                 * scale_factor, vertex[2] *
                                      scale_factor)
76      glEnd()
77
78      glutSwapBuffers()
79
80
81  def init_glut(display_func):
82      glutInit()
83      glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
84      glutInitWindowSize(window_width, window_height)
85      glutInitWindowPosition(100, 100)
86      glutCreateWindow(b"3D Depth Map Visualization")
87      glutDisplayFunc(display_func)
88      glEnable(GL_DEPTH_TEST)
89      glClearColor(0.0, 0.0, 0.0, 0.0)
90
91      glMatrixMode(GL_PROJECTION)
92      gluPerspective(45, (window_width / window_height), 0.1,
            1000.0)
93      glMatrixMode(GL_MODELVIEW)
94
95      glPointSize(2.0)
96
97
98  # Read map
99  json_filename = 'depth_map_info.json'
100 depth_map_filename, config = read_json_file(json_filename)
101 depth_map_array = read_depth_map(depth_map_filename)
102
103 # Export map
104 export_format = config.get('export_format', 'obj')
```

```
105 output_filename = config.get('export_name', 'output')
106 export_depth_map_to_file(depth_map_array, export_format,
        output_filename)
107
108 # Show object
109 light_position = config['light_source']['position']
110 viewer_position = config['viewer']['position']
111 reflection_model = config.get('reflection_model', 'phong')
112
113 init_glut(
114     lambda: display(
115         depth_map_array,
116         light_position,
117         viewer_position,
118         reflection_model
119     )
120 )
121 glutMainLoop()
```

Листинг model_export.py:

```
1  import numpy as np
2
3
4  def _calculate_normal(p1, p2, p3):
5      u = np.array(p2) - np.array(p1)
6      v = np.array(p3) - np.array(p1)
7      return np.cross(u, v)
8
9  def _export_to_obj(filename, depth_map_array):
10     height, width = depth_map_array.shape
11     vertices = []
12     indices = {}
13
14     with open(filename, 'w') as file:
15         vertex_id = 1
16         for i in range(height):
17             for j in range(width):
18                 depth = depth_map_array[i, j]
19                 if depth != 0:
20                     vertex = (j - width / 2, height / 2 - i,
                           -depth)
21                     indices[(i, j)] = vertex_id
```

```
22                      vertices.append(vertex)
23                      file.write(f'v {vertex[0]} {vertex[1]} {
                            vertex[2]}\n')
24                      vertex_id += 1
25
26          for i in range(height - 1):
27              for j in range(width - 1):
28                  if (depth_map_array[i, j] != 0 and
29                      depth_map_array[i, j + 1] != 0 and
30                      depth_map_array[i + 1, j] != 0 and
31                          depth_map_array[i + 1, j + 1] != 0):
32
33                      v1 = indices[(i, j)]
34                      v2 = indices[(i, j + 1)]
35                      v3 = indices[(i + 1, j + 1)]
36                      v4 = indices[(i + 1, j)]
37
38                      file.write(f"l {v1} {v2}\n")
39                      file.write(f"l {v2} {v3}\n")
40                      file.write(f"l {v3} {v4}\n")
41                      file.write(f"l {v4} {v1}\n")
42
43                      file.write(f"f {v1} {v2} {v3} {v4}\n")
44
45
46  def _export_to_stl(filename, depth_map_array):
47      # TODO: extract open file in new method
48      height, width = depth_map_array.shape
49      with open(filename, 'w') as file:
50          file.write('solid depth_map\n')
51          for i in range(height - 1):
52              for j in range(width - 1):
53                  if (depth_map_array[i, j] != 0 and
54                      depth_map_array[i, j + 1] != 0 and
55                      depth_map_array[i + 1, j] != 0 and
56                          depth_map_array[i + 1, j + 1] != 0):
57
58                      vertices = [
59                          [j - width / 2, height / 2 - i, -
                                depth_map_array[i, j]],
60                          [j + 1 - width / 2, height / 2 -
```

```
                                i, -depth_map_array[i, j + 1]],
                            [j + 1 - width / 2, height / 2 -
                                (i + 1), -depth_map_array[i + 1,
                                    j + 1]],
                            [j - width / 2, height / 2 -
                                (i + 1), -depth_map_array[i + 1,
                                    j]]
                        ]

                        for k in range(2):
                            triangle = (
                                vertices[0], vertices[k + 1],
                                    vertices[k + 2])
                            normal = _calculate_normal(*triangle)
                            normal_string = ' '.join(map(str,
                                normal))
                            file.write(f'facet normal {
                                normal_string}\n')
                            file.write('  outer loop\n')
                            for vertex in triangle:
                                vertex_string = ' '.join(map(str,
                                    vertex))
                                file.write(f'    vertex {
                                    vertex_string}\n')
                            file.write('  endloop\n')
                            file.write('endfacet\n')
        file.write('endsolid depth_map\n')


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

    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)
```

```
94                  indices[(i, j)] = len(vertices)
95                  vertices.append(vertex)
96
97      for i in range(height - 1):
98          for j in range(width - 1):
99              if (depth_map_array[i, j] != 0 and
100                 depth_map_array[i, j + 1] != 0 and
101                 depth_map_array[i + 1, j] != 0 and
102                     depth_map_array[i + 1, j + 1] != 0):
103
104                 v1 = indices[(i, j)]
105                 v2 = indices[(i, j + 1)]
106                 v3 = indices[(i + 1, j + 1)]
107                 v4 = indices[(i + 1, j)]
108
109                 faces.append((v1, v2, v3, v4))
110
111     with open(filename, 'w') as file:
112         file.write("ply\n")
113         file.write("format ascii 1.0\n")
114         file.write(f"element vertex {len(vertices)}\n")
115         file.write("property float x\n")
116         file.write("property float y\n")
117         file.write("property float z\n")
118         file.write(f"element face {len(faces)}\n")
119         file.write("property list uchar int vertex_index\n")
120         file.write("end_header\n")
121
122         for vertex in vertices:
123             file.write(f"{vertex[0]} {vertex[1]} {vertex[2]}\
                n")
124
125         for face in faces:
126             file.write(f"4 {face[0]} {face[1]} {face[2]} {
                face[3]}\n")
127
128
129 def export_depth_map_to_file(depth_map_array, export_format,
    output_filename):
130     output_path = f"{output_filename}.{export_format}"
131
```

```
132     if export_format == 'obj':
133         _export_to_obj(output_path, depth_map_array)
134     elif export_format == 'stl':
135         _export_to_stl(output_path, depth_map_array)
136     elif export_format == 'ply':
137         _export_to_ply(output_path, depth_map_array)
```

Листинг reflection_models.py:

```
1  import numpy as np
2
3
4  def _lambert_reflection(normal, light_source):
5      light_vector = np.array(light_source) / np.linalg.norm(
           light_source)
6      normal_vector = np.array(normal) / np.linalg.norm(normal)
7      dot = np.dot(normal_vector, light_vector)
8      intensity = max(dot, 0)
9      return (intensity, intensity, intensity)
10
11
12 def _torrance_sparrow_reflection(normal, light_source, viewer
       ):
13     light_vector = np.array(light_source) / np.linalg.norm(
           light_source)
14     normal_vector = np.array(normal) / np.linalg.norm(normal)
15     viewer_vector = np.array(viewer) / np.linalg.norm(viewer)
16
17     roughness = 0.5
18     f0 = 0.04
19
20     half_vector = (light_vector + viewer_vector) / \
21         np.linalg.norm(light_vector + viewer_vector)
22     dot_l_n = max(np.dot(light_vector, normal_vector), 0)
23     dot_v_n = max(np.dot(viewer_vector, normal_vector), 0)
24     dot_h_n = max(np.dot(half_vector, normal_vector), 0)
25     dot_h_v = max(np.dot(half_vector, viewer_vector), 0)
26
27     F = f0 + (1 - f0) * (1 - dot_h_v) ** 5
28
29     G = min(1, min((2 * dot_h_n * dot_v_n) / dot_h_v,
30             (2 * dot_h_n * dot_l_n) / dot_h_v))
31
```

```
32      alpha = roughness ** 2
33      denom = (dot_h_n ** 2) * (alpha ** 2) + (1 - (dot_h_n) **
            2)
34      D = alpha ** 2 / (np.pi * (denom ** 2))
35
36      specular_intensity = (F * D * G) / (4 * dot_l_n * dot_v_n
            + 1e-7)
37      diffuse_intensity = dot_l_n
38
39      intensity = 0.1 + 0.9 * (diffuse_intensity +
            specular_intensity)
40      intensity = min(intensity, 1)
41
42      return (intensity, intensity, intensity)
43
44
45  def _phong_reflection(normal, light_source, viewer):
46      light_vector = np.array(light_source) / np.linalg.norm(
            light_source)
47      normal_vector = np.array(normal) / np.linalg.norm(normal)
48      viewer_vector = np.array(viewer) / np.linalg.norm(viewer)
49
50      ambient = 0.1
51      diffuse_coefficient = 0.7
52      specular_coefficient = 0.2
53      shininess = 32
54
55      ambient_color = ambient
56
57      dot_l_n = max(np.dot(light_vector, normal_vector), 0)
58      diffuse_color = diffuse_coefficient * dot_l_n
59
60      reflection_vector = 2 * normal_vector * dot_l_n -
            light_vector
61      dot_r_v = max(np.dot(reflection_vector, viewer_vector),
            0)
62      specular_color = specular_coefficient * (dot_r_v **
            shininess)
63
64      intensity = ambient_color + diffuse_color +
            specular_color
```

```
65      intensity = min(intensity, 1)
66
67      return (intensity, intensity, intensity)
68
69  def get_color(normal, light_source, viewer, model):
70      if model == 'lambert':
71          return _lambert_reflection(normal, light_source)
72      elif model == 'phong':
73          return _phong_reflection(normal, light_source, viewer
                )
74      elif model == 'torrance-sparrow':
75          return _torrance_sparrow_reflection(normal,
                light_source, viewer)
```

Листинг depth_map_info.json:

```
1  {
2    "name": "test.dat",
3    "light_source": {
4      "position": [1.0, 1.0, -1.0]
5    },
6    "viewer": {
7      "position": [0.0, 0.0, -500.0]
8    },
9    "reflection_model": "lambert",
10   "export_format": "ply",
11   "export_name": "result/output"
12 }
```