

Gebze Technical University
Computer Engineering

Computer Graphics(CSE 461)

Homework #1

Berkan AKIN

171044073

1)Requirements

Scene Description File

The required assignment involves reading data from XML file tags, performing necessary calculations with the read data, rendering, and creating an image. The program to be written should operate quickly and efficiently. The tags to be interpreted by the program are listed below.

The file will be in "xml" format, and you may use XML parsers.

Definitions of elements that can be included in the scene description:

maxraytracedepth: The maximum recursive ray tracing depth. Rays starting from the camera begin at depth 0.

background: The color (r, g, b) to be used for pixels when rays do not intersect with any objects.

camera: Defines the camera in the scene, including properties such as position, gaze direction, up vector, image plane, and image resolution.

ambientlight: The amount of light received by surfaces in shadow (r, g, b).

pointlight: A point light source defined by a position vector and an intensity vector.

triangularlight: A planar directional light source defined by a triangle.

material: Material properties of surfaces including ambient, diffuse, specular, mirror reflectance, and phong exponent values.

vertexdata: Coordinates of all vertices (vertex) in the scene.

mesh: Structures composed of triangular faces, each face defined by vertex indices.

2) Development

The program is written in three stages. Firstly, code has been developed to parse the XML file. Secondly, the ray tracing process is carried out. Thirdly, the generated image is output in PPM format.

1)XML Parser

The program begins by using an XML parser to retrieve XML data and store it in struct.

```

Kopya > C:\xmlParser\ > loadFromXml(const std::string&
1 #include "xmlParser.h"
2 #include <sstream>
3 #include <stdexcept>
4 #include "tinyxml2.h"
5 #include <array>
6
7 std::array<float, 3> crossProduct(const std::array<float, 3>& v1, const std::array<float, 3>& v2) {
8     std::array<float, 3> result;
9     result[0] = v1[1] * v2[2] - v1[2] * v2[1];
10    result[1] = v1[2] * v2[0] - v1[0] * v2[2];
11    result[2] = v1[0] * v2[1] - v1[1] * v2[0];
12    return result;
13 }
14
15 void parser::Scene::loadFromXml(const std::string& filepath) {
16     tinyxml2::XMLDocument file;
17     std::stringstream stream;
18
19     auto res = file.LoadFile(filepath.c_str());
20     if (res) {
21         throw std::runtime_error("Error: The xml file cannot be loaded.");
22     }
23
24     auto root = file.FirstChild();
25     if (!root) {
26         throw std::runtime_error("Error: Root is not found.");
27     }
28
29     // Get BackgroundColor
30     auto element = root->FirstChildElement("backgroundColor");
31     if (element) {
32         stream << element->GetText() << std::endl;
33     } else {
34         stream << "0 0 0" << std::endl;
35     }
36     stream >> background_color.x >> background_color.y >> background_color.z;
37     //std::cout<<"burada 2\n";
38
39     // Get MaxRecursionDepth
40     element = root->FirstChildElement("maxraytracedepth");
41     if (element) {
42         stream << element->GetText() << std::endl;
43     } else {
44         stream << "0" << std::endl;
45     }
46     stream >> max_recursion_depth;
47
48 Kopya > C:\xmlParser\ > parser > PointLight > position
1 #ifndef __XMLPARSER__
2 #define __XMLPARSER__
3
4 #include <limits>
5 #include <memory>
6 #include <string>
7 #include <vector>
8 #include "RayTracing.h"
9 #include "vector.h"
10
11 namespace parser {
12 namespace {
13
14 float Determinant(Vec3f a, Vec3f b, Vec3f c) {
15     // vectors are columns
16     return a.x * (b.y * c.z - c.y * b.z) + a.y * (c.x * b.z - b.x * c.z) +
17         a.z * (b.x * c.y - c.x * b.y);
18 }
19 } // namespace
20
21 struct Camera {
22     Vec3f position;
23     Vec3f gaze;
24     Vec3f up;
25     Vec4f near_plane;
26     float near_distance;
27     int image_width, image_height;
28     std::string image_name;
29 };
30
31 struct PointLight {
32     Vec3f position;
33     Vec3f intensity;
34 };
35
36 struct TriangularLight {
37     Vec3f vertex1;
38     Vec3f vertex2;
39     Vec3f vertex3;
40     Vec3f intensity;
41 };
42
43 struct Material {
44     Vec3f ambient;
45     Vec3f diffuse;
46     Vec3f specular;
47     Vec3f mirror;
48     float phong_exponent;
49 };

```

2) Ray Tracing

The retrieved data is processed to perform the necessary calculations, and is then prepared to be output as an image in PPM format.

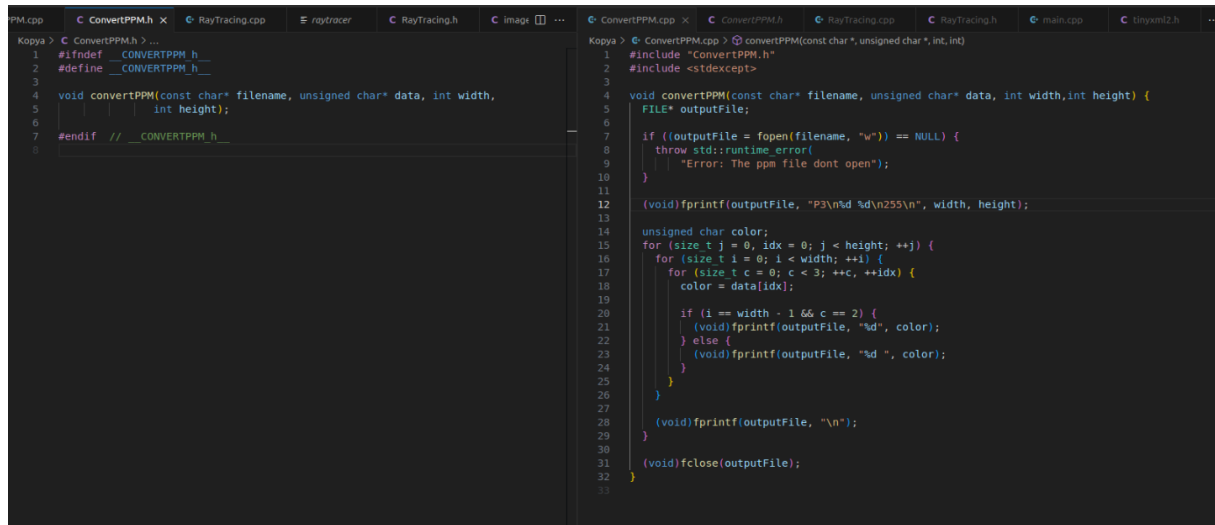
```

1 // RayTracing - 11.11.2024 - 11.11.2024 - 11.11.2024
2 #include "RayTracing.h"
3 #define _RAYTRACING_
4 #include "vector.h"
5
6 struct Ray {
7     Vec3f origin;
8     Vec3f direction;
9     bool is_shadow;
10 };
11
12 class Object {
13 public:
14     struct HitRecord {
15         int material_id;
16         float t;
17         Vec3f normal;
18         const Object* obj;
19     };
20
21     class BoundingBox {
22     public:
23         BoundingBox() {
24             min_corner(parser::Vec3f(kInf, kInf, kInf));
25             max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
26         }
27         BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
28             min_corner(min_c);
29             max_corner(max_c);
30         }
31
32         float DoesIntersect(const Ray& ray) const {
33             float tmax = -kInf;
34             float tmin = kInf;
35             const Vec3f origin = ray.origin;
36             const Vec3f direction = ray.direction;
37             for (int i = 0; i < 3; ++i) {
38                 if (fabs(direction[i]) < kEpsilon) continue;
39                 float tn = (min_corner[i] - origin[i]) / direction[i];
40                 float tf = (max_corner[i] - origin[i]) / direction[i];
41                 if (direction[i] < 0) std::swap(tn, tf);
42                 if (tn > tmax) tmax = tn;
43                 if (tf < tmin) tmin = tf;
44                 if (tmax < tmin) return kInf;
45             }
46             return tmax;
47         }
48
49         class BoundingBox {
50         public:
51             BoundingBox() {
52                 min_corner(parser::Vec3f(kInf, kInf, kInf));
53                 max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
54             }
55             BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
56                 min_corner(min_c);
57                 max_corner(max_c);
58             }
59
60             float DoesIntersect(const Ray& ray) const {
61                 float tmax = -kInf;
62                 float tmin = kInf;
63                 const Vec3f origin = ray.origin;
64                 const Vec3f direction = ray.direction;
65                 for (int i = 0; i < 3; ++i) {
66                     if (fabs(direction[i]) < kEpsilon) continue;
67                     float tn = (min_corner[i] - origin[i]) / direction[i];
68                     float tf = (max_corner[i] - origin[i]) / direction[i];
69                     if (direction[i] < 0) std::swap(tn, tf);
70                     if (tn > tmax) tmax = tn;
71                     if (tf < tmin) tmin = tf;
72                     if (tmax < tmin) return kInf;
73                 }
74                 return tmax;
75             }
76
77             class BoundingBox {
78             public:
79                 BoundingBox() {
80                     min_corner(parser::Vec3f(kInf, kInf, kInf));
81                     max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
82                 }
83                 BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
84                     min_corner(min_c);
85                     max_corner(max_c);
86                 }
87
88                 float DoesIntersect(const Ray& ray) const {
89                     float tmax = -kInf;
90                     float tmin = kInf;
91                     const Vec3f origin = ray.origin;
92                     const Vec3f direction = ray.direction;
93                     for (int i = 0; i < 3; ++i) {
94                         if (fabs(direction[i]) < kEpsilon) continue;
95                         float tn = (min_corner[i] - origin[i]) / direction[i];
96                         float tf = (max_corner[i] - origin[i]) / direction[i];
97                         if (direction[i] < 0) std::swap(tn, tf);
98                         if (tn > tmax) tmax = tn;
99                         if (tf < tmin) tmin = tf;
100                        if (tmax < tmin) return kInf;
101                    }
102                    return tmax;
103                }
104            };
105        };
106
107         class BoundingBox {
108         public:
109             BoundingBox() {
110                 min_corner(parser::Vec3f(kInf, kInf, kInf));
111                 max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
112             }
113             BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
114                 min_corner(min_c);
115                 max_corner(max_c);
116             }
117
118             float DoesIntersect(const Ray& ray) const {
119                 float tmax = -kInf;
120                 float tmin = kInf;
121                 const Vec3f origin = ray.origin;
122                 const Vec3f direction = ray.direction;
123                 for (int i = 0; i < 3; ++i) {
124                     if (fabs(direction[i]) < kEpsilon) continue;
125                     float tn = (min_corner[i] - origin[i]) / direction[i];
126                     float tf = (max_corner[i] - origin[i]) / direction[i];
127                     if (direction[i] < 0) std::swap(tn, tf);
128                     if (tn > tmax) tmax = tn;
129                     if (tf < tmin) tmin = tf;
130                     if (tmax < tmin) return kInf;
131                 }
132                 return tmax;
133             }
134         };
135     };
136
137     class BoundingBox {
138     public:
139         BoundingBox() {
140             min_corner(parser::Vec3f(kInf, kInf, kInf));
141             max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
142         }
143         BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
144             min_corner(min_c);
145             max_corner(max_c);
146         }
147
148         float DoesIntersect(const Ray& ray) const {
149             float tmax = -kInf;
150             float tmin = kInf;
151             const Vec3f origin = ray.origin;
152             const Vec3f direction = ray.direction;
153             for (int i = 0; i < 3; ++i) {
154                 if (fabs(direction[i]) < kEpsilon) continue;
155                 float tn = (min_corner[i] - origin[i]) / direction[i];
156                 float tf = (max_corner[i] - origin[i]) / direction[i];
157                 if (direction[i] < 0) std::swap(tn, tf);
158                 if (tn > tmax) tmax = tn;
159                 if (tf < tmin) tmin = tf;
160                 if (tmax < tmin) return kInf;
161             }
162             return tmax;
163         }
164     };
165
166     class BoundingBox {
167     public:
168         BoundingBox() {
169             min_corner(parser::Vec3f(kInf, kInf, kInf));
170             max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
171         }
172         BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
173             min_corner(min_c);
174             max_corner(max_c);
175         }
176
177         float DoesIntersect(const Ray& ray) const {
178             float tmax = -kInf;
179             float tmin = kInf;
180             const Vec3f origin = ray.origin;
181             const Vec3f direction = ray.direction;
182             for (int i = 0; i < 3; ++i) {
183                 if (fabs(direction[i]) < kEpsilon) continue;
184                 float tn = (min_corner[i] - origin[i]) / direction[i];
185                 float tf = (max_corner[i] - origin[i]) / direction[i];
186                 if (direction[i] < 0) std::swap(tn, tf);
187                 if (tn > tmax) tmax = tn;
188                 if (tf < tmin) tmin = tf;
189                 if (tmax < tmin) return kInf;
190             }
191             return tmax;
192         }
193     };
194
195     class BoundingBox {
196     public:
197         BoundingBox() {
198             min_corner(parser::Vec3f(kInf, kInf, kInf));
199             max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
200         }
201         BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
202             min_corner(min_c);
203             max_corner(max_c);
204         }
205
206         float DoesIntersect(const Ray& ray) const {
207             float tmax = -kInf;
208             float tmin = kInf;
209             const Vec3f origin = ray.origin;
210             const Vec3f direction = ray.direction;
211             for (int i = 0; i < 3; ++i) {
212                 if (fabs(direction[i]) < kEpsilon) continue;
213                 float tn = (min_corner[i] - origin[i]) / direction[i];
214                 float tf = (max_corner[i] - origin[i]) / direction[i];
215                 if (direction[i] < 0) std::swap(tn, tf);
216                 if (tn > tmax) tmax = tn;
217                 if (tf < tmin) tmin = tf;
218                 if (tmax < tmin) return kInf;
219             }
220             return tmax;
221         }
222     };
223
224     class BoundingBox {
225     public:
226         BoundingBox() {
227             min_corner(parser::Vec3f(kInf, kInf, kInf));
228             max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
229         }
230         BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
231             min_corner(min_c);
232             max_corner(max_c);
233         }
234
235         float DoesIntersect(const Ray& ray) const {
236             float tmax = -kInf;
237             float tmin = kInf;
238             const Vec3f origin = ray.origin;
239             const Vec3f direction = ray.direction;
240             for (int i = 0; i < 3; ++i) {
241                 if (fabs(direction[i]) < kEpsilon) continue;
242                 float tn = (min_corner[i] - origin[i]) / direction[i];
243                 float tf = (max_corner[i] - origin[i]) / direction[i];
244                 if (direction[i] < 0) std::swap(tn, tf);
245                 if (tn > tmax) tmax = tn;
246                 if (tf < tmin) tmin = tf;
247                 if (tmax < tmin) return kInf;
248             }
249             return tmax;
250         }
251     };
252
253     class BoundingBox {
254     public:
255         BoundingBox() {
256             min_corner(parser::Vec3f(kInf, kInf, kInf));
257             max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
258         }
259         BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
260             min_corner(min_c);
261             max_corner(max_c);
262         }
263
264         float DoesIntersect(const Ray& ray) const {
265             float tmax = -kInf;
266             float tmin = kInf;
267             const Vec3f origin = ray.origin;
268             const Vec3f direction = ray.direction;
269             for (int i = 0; i < 3; ++i) {
270                 if (fabs(direction[i]) < kEpsilon) continue;
271                 float tn = (min_corner[i] - origin[i]) / direction[i];
272                 float tf = (max_corner[i] - origin[i]) / direction[i];
273                 if (direction[i] < 0) std::swap(tn, tf);
274                 if (tn > tmax) tmax = tn;
275                 if (tf < tmin) tmin = tf;
276                 if (tmax < tmin) return kInf;
277             }
278             return tmax;
279         }
280     };
281
282     class BoundingBox {
283     public:
284         BoundingBox() {
285             min_corner(parser::Vec3f(kInf, kInf, kInf));
286             max_corner(parser::Vec3f(-kInf, -kInf, -kInf));
287         }
288         BoundingBox(const parser::Vec3f& min_c, const parser::Vec3f& max_c) {
289             min_corner(min_c);
290             max_corner(max_c);
291         }
292
293         float DoesIntersect(const Ray& ray) const {
294             float tmax = -kInf;
295             float tmin = kInf;
296             const Vec3f origin = ray.origin;
297             const Vec3f direction = ray.direction;
298             for (int i = 0; i < 3; ++i) {
299                 if (fabs(direction[i]) < kEpsilon) continue;
300                 float tn = (min_corner[i] - origin[i]) / direction[i];
301                 float tf = (max_corner[i] - origin[i]) / direction[i];
302                 if (direction[i] < 0) std::swap(tn, tf);
303                 if (tn > tmax) tmax = tn;
304                 if (tf < tmin) tmin = tf;
305                 if (tmax < tmin) return kInf;
306             }
307             return tmax;
308         }
309     };

```

3) Convert image PPM Format

The created image array has been converted to PPM format using the convert function.



```
PM.cpp  C ConvertPPM.h x  RayTracing.cpp  raytracer  C RayTracing.h  C Image  ...  ConvertPPM.cpp  ConvertPPM.h  RayTracing.cpp  RayTracing.h  main.cpp  tinyxml2.h ...

Kopya > C ConvertPPM.h > ...
1 #ifndef CONVERTPPM_H
2 #define CONVERTPPM_H
3
4 void convertPPM(const char* filename, unsigned char* data, int width,
5               int height);
6
7 #endif // CONVERTPPM_H
8

Kopya > C ConvertPPM.cpp > convertPPM(const char*, unsigned char*, int, int)
1 #include "ConvertPPM.h"
2 #include <stdexcept>
3
4 void convertPPM(const char* filename, unsigned char* data, int width, int height) {
5     FILE* outputFile;
6
7     if ((outputFile = fopen(filename, "w")) == NULL) {
8         throw std::runtime_error(
9             "Error: The ppm file dont open");
10    }
11
12    (void)fprintf(outputFile, "P3\n%d %d\n255\n", width, height);
13
14    unsigned char color;
15    for (size_t j = 0; j < height; ++j) {
16        for (size_t i = 0; i < width; ++i) {
17            for (size_t c = 0; c < 3; ++c, ++idx) {
18                color = data[idx];
19
20                if (i == width - 1 && c == 2) {
21                    (void)fprintf(outputFile, "%d", color);
22                } else {
23                    (void)fprintf(outputFile, "%d ", color);
24                }
25            }
26        }
27
28        (void)fprintf(outputFile, "\n");
29    }
30
31    (void)fclose(outputFile);
32 }
33
```

3)Test

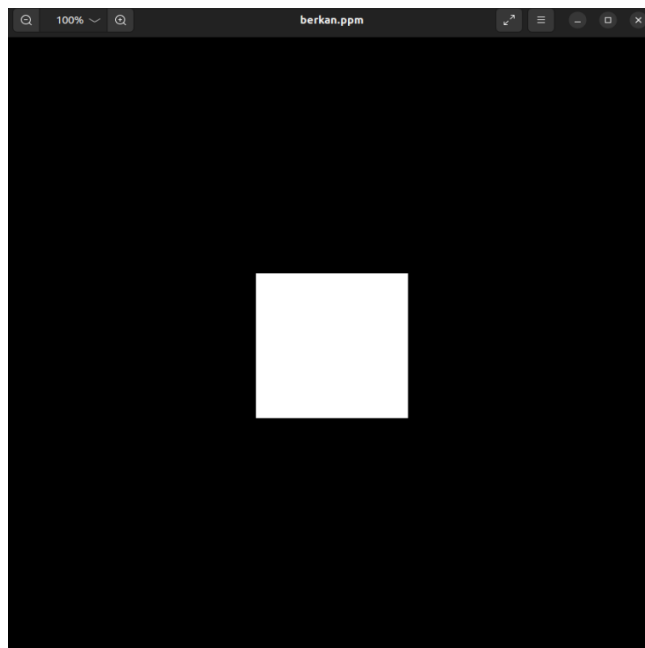
The developed program has been tested with various XML files. Successful results have been obtained from the tests. Examples of the tests conducted are provided below.

Example 1

Given input

```
Kopya > test.xml
1 <scene>
2   <maxraytracedepth>6</maxraytracedepth>
3   <backgroundColor>0 0 0</backgroundColor>
4
5   <camera>
6     <position>0 0 0</position>
7     <gaze>0 0 -1</gaze>
8     <up>0 1 0</up>
9     <nearPlane>-1 1 -1 1</nearPlane>
10    <neardistance>1</neardistance>
11    <imageresolution>800 800</imageresolution>
12  </camera>
13
14  <lights>
15    <ambientlight>25 25 25</ambientlight>
16    <pointlight id="1">
17      <position>0 0 0</position>
18      <intensity>1000 1000 1000</intensity>
19    </pointlight>
20    <triangularlight id="2">
21      <vertex1> 0 0 0</vertex1>
22      <vertex2> 1.2 0.5 0.5</vertex2>
23      <vertex3> 0.5 0.5 0.5</vertex3>
24      <intensity> 800 800 800</intensity>
25    </triangularlight>
26  </lights>
27
28  <materials>
29    <material id="1">
30      <ambient>1 1 1</ambient>
31      <diffuse>1 1 1</diffuse>
32      <specular>1 1 1</specular>
33      <mirrorreflectance>0 0 0</mirrorreflectance>
34      <phongexponent>1</phongexponent>
35    </material>
36  </materials>
37
38  <vertexdata>
39    -0.5 0.5 -2
40    -0.5 -0.5 -2
41    0.5 -0.5 -2
42    0.5 0.5 -2
43    0.75 0.75 -2
44    1 0.75 -2
45    0.875 1 -2
46  </vertexdata>
```

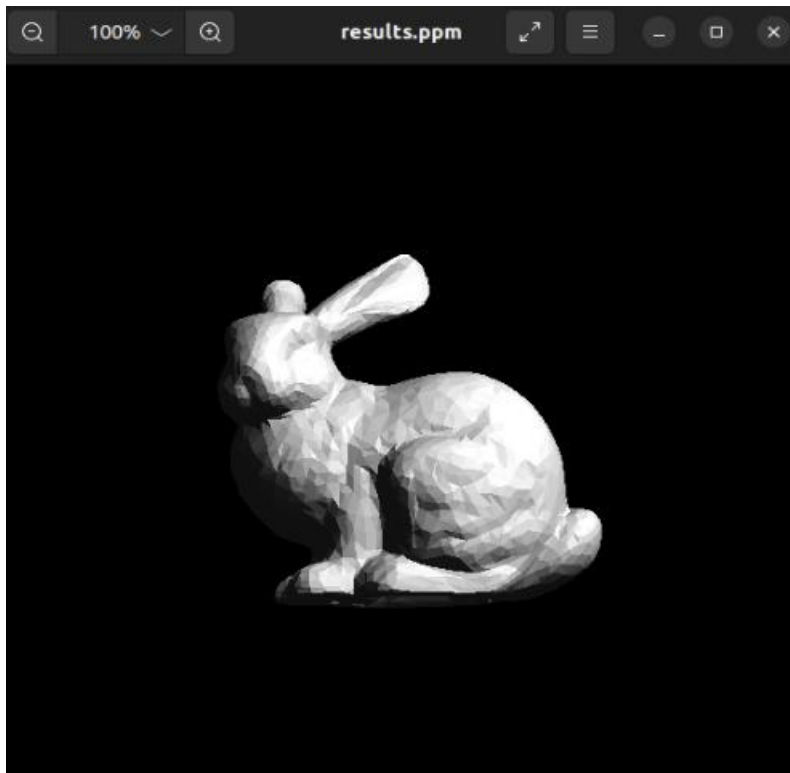
Output



Example 2

```
Kopya > test2.xml
1  <scene>
2    <backgroundColor>0 0 0</backgroundColor>
3
4
5
6    <maxraytracedepth>6</maxraytracedepth>
7
8
9    <camera id="1">
10     <position>-0.02 -0.05 1.5</position>
11     <gaze>0 0 -1</gaze>
12     <up>0 1 0</up>
13     <nearPlane>0.1 0.1 0.0 0.2</nearPlane>
14     <nearDistance>1</nearDistance>
15     <ImageResolution>512 512</ImageResolution>
16
17   </camera>
18
19
20
21   <lights>
22     <ambientlight>81 81 81</ambientlight>
23     <pointlight id="1">
24       <position>2 2 2 </position>
25       <intensity>3000 3000 3000</intensity>
26     </pointlight>
27
28     <triangularlight id="2">
29       <vertex1> 0 0 0 </vertex1>
30       <vertex2> 1.2 0.5 0.5 </vertex2>
31       <vertex3> 0.5 0.5 0.5 </vertex3>
32       <intensity> 800 800 800 </intensity>
33     </triangularlight>
34   </lights>
35
36   <materials>
37     <material id="1">
38       <ambient>0.2 0.2 0.2</ambient>
39       <diffuse>0.8 0.8 0.8</diffuse>
40       <specular>0.2 0.2 0.2</specular>
41       <mirrorreflectance>0 0 0</mirrorreflectance>
42       <phongexponent>3</phongexponent>
43     </material>
44   </materials>
45
46   <vertexdata>
47     -0.00341018 0.13032 0.0217544
48     -0.0017192 0.152501 0.0296561
```

Output



4) Compilation and Execution

You should run the makefile.

1) `./make`

You should run the created raytracer file by providing it with an XML file.

2) `./raytracer <xml file>`