

**Grafika komputerowa
Laboratorium**

Stanislau Antanovich & Mykola Sharonov



**WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI**
POLITECHNIKI RZESZOWSKIEJ

Spis treści

1	Budowa obiektu sterowanego	3
1.1	Opis zadania	3
1.2	Wymagania	3
1.3	Realizacja zadania	3
1.3.1	class <i>Wheel</i>	4
1.3.1.1	Opis działania	4
1.3.1.2	Plik <i>Wheel.h</i>	4
1.3.1.3	Plik <i>Wheel.cpp</i>	5
1.3.2	class <i>SideSciana</i>	8
1.3.2.1	Opis działania	8
1.3.2.2	Plik <i>SideSciana.h</i>	8
1.3.2.3	Plik <i>SideSciana.cpp</i>	9
1.3.3	class <i>Front</i>	11
1.3.3.1	Opis działania	11
1.3.3.2	Plik <i>Front.h</i>	11
1.3.3.3	Plik <i>Front.cpp</i>	11
1.3.4	class <i>Back</i>	13
1.3.4.1	Opis działania	13
1.3.4.2	Plik <i>Back.h</i>	13
1.3.4.3	Plik <i>Back.cpp</i>	14
2	Budowa otoczenia	17
2.1	Opis zadania	17
2.2	Wymagania	17
2.3	Realizacja zadania	17
2.3.1	class <i>Podloze</i>	17
2.3.1.1	Opis działania	18
2.3.1.2	Plik <i>Podloze.h</i>	18

3	Tekstutowanie	19
3.1	Opis zadania	19
3.2	Wymagania	19
3.3	Realizacja zadania	19
4	Sterowanie obiektem głównym	20
4.1	Opis zadania	20
4.2	Wymagania	20
4.3	Realizacja zadania	20

Rozdział 1

Budowa obiektu sterowanego

1.1 Opis zadania

Należy zbudować “robot rolniczy (łazik)” wykorzystując wyłącznie prymitywy bazujące na trójkącie. Obiekt ten będzie wykorzystywany na kolejnych zajęciach. W tworzonej grze komputerowej użytkownik będzie miał możliwość sterowania tym łazikiem.

1.2 Wymagania

Wymagania dotyczące budowy głośnego obiektu:

- Na ocenę 3: Obiekt złożony z co najmniej 10 brył elementarnych (walec, prostopadłościan, itp.) zbudowanych przy użyciu prymitywów bazujących na trójkącie.
- Na ocenę 4: Obiekt złożony z co najmniej 20 brył elementarnych (walec, prostopadłościan, itp.) zbudowanych przy użyciu prymitywów bazujących na trójkącie.
- Na ocenę 5: Obiekt złożony z co najmniej 25 brył elementarnych (walec, prostopadłościan, itp.) zbudowanych przy użyciu prymitywów bazujących na trójkącie oraz projekt napisany obiektowo w C++.

Możliwość zaimportowania łazika z programu graficznego (np. Blender) o budowie odpowiadającej co najmniej 25 bryłom elementarnym.

1.3 Realizacja zadania

Naszym “łazikiem” będzie występował zwykły samochód.



Rysunek 1.1: Łazik(samoshód)

1.3.1 class *Wheel*

1.3.1.1 Opis działania

Klasa *Wheel* odpowiada za rysowanie koła.

1.3.1.2 Plik *Wheel.h*

Plik *Wheel.h* deklaruje wszystkie zmienne oraz metody, które będą używane obiektami tej klasy.

```

1 #include "includes.h"
2
3 class Wheel
4 {
5 private:
6     float radius {};
7     float width {};
8     float posX {};
9     float posY {};
10    float posZ {};
11
12    void outerObject () const;
13    void crochet (const float) const;
14    void innerObject () const;
15    void protector () const;
16    void cuboid (const float , const float , const float , const float , const
        float) const;
17 public:
18     Wheel (const float , const float , const float , const float , const float);
19     void draw () const;
20 };

```

1.3.1.3 Plik *Wheel.cpp*

Plik *Wheel.cpp* zawiera inicjalizacje zmiennych oraz metod opisanych w pliku nagłówkowym *Wheel.h*.

```
1 #include "Wheel.h"
2
3 Wheel::Wheel(const float radius, const float width, const float posX, const
    float posY, const float posZ) : radius(radius), width(width), posX(posX)
    , posY(posY), posZ(posZ) {}
4
5
6
7 void Wheel::cuboid(const float angle, const float partAngle, const float
    posY, const float width, const float height) const {
8     glBegin(GL_TRIANGLE_STRIP);
9     for (float alpha = angle; alpha <= partAngle; alpha += GL_PI / 32){
10         float x1 = radius * cos(alpha);
11         float z1 = radius * sin(alpha);
12
13         float x2 = (radius + height) * cos(alpha);
14         float z2 = (radius + height) * sin(alpha);
15         glVertex3f(x1 + posX, z1 + posZ, posY);
16         glVertex3f(x2 + posX, z2 + posZ, posY);
17     }
18     glEnd();
19
20     glBegin(GL_TRIANGLE_STRIP);
21     for (float alpha = angle; alpha <= partAngle; alpha += GL_PI / 32){
22         float x1 = radius * cos(alpha);
23         float z1 = radius * sin(alpha);
24
25         float x2 = (radius + height) * cos(alpha);
26         float z2 = (radius + height) * sin(alpha);
27         glVertex3f(x1 + posX, z1 + posZ, posY + width);
28         glVertex3f(x2 + posX, z2 + posZ, posY + width);
29     }
30     glEnd();
31
32     glBegin(GL_TRIANGLE_STRIP);
33     for (float y = posY; y <= width + posY; y++){
34         float x1 = radius * cos(angle);
35         float z1 = radius * sin(angle);
36
37         float x2 = (radius + height) * cos(angle);
38         float z2 = (radius + height) * sin(angle);
39
40         glVertex3f(x1 + posX, z1 + posZ, y);
41         glVertex3f(x2 + posX, z2 + posZ, y);
42     }
43     glEnd();
44
45     glBegin(GL_TRIANGLE_STRIP);
46     for (float alpha = angle; alpha <= partAngle; alpha += GL_PI / 32){
47         float x2 = (radius + height) * cos(alpha);
48         float z2 = (radius + height) * sin(alpha);
49
50         glVertex3f(x2 + posX, z2 + posZ, posY);
51         glVertex3f(x2 + posX, z2 + posZ, posY + width);
52     }
53     glEnd();
```

```

54
55 glBegin(GL_TRIANGLE_STRIP);
56     for (float y = posY; y <= width + posY; y++){
57         float x1 = radius * cos(partAngle);
58         float z1 = radius * sin(partAngle);
59
60         float x2 = (radius + height) * cos(partAngle);
61         float z2 = (radius + height) * sin(partAngle);
62
63         glVertex3f(x1 + posX, z1 + posZ, y);
64         glVertex3f(x2 + posX, z2 + posZ, y);
65     }
66 glEnd();
67 }
68
69 void Wheel::outerObject() const {
70     glColor3f(0.0, 0.0, 0.0);
71
72     for (float y1 = posY; y1 < width + posY; y1 += 1.0f){
73         glBegin(GL_TRIANGLE_STRIP);
74         for (float alpha = 0.0f; alpha <= 2 * GL_PI + 1; alpha += GL_PI / 32){
75             float x1 = radius * cos(alpha);
76             float z1 = radius * sin(alpha);
77
78             glVertex3f(x1 + posX, z1 + this->posZ, y1);
79             glVertex3f(x1 + posX, z1 + this->posZ, y1 + 1.0f);
80         }
81         glEnd();
82     }
83     this->protector();
84 }
85
86 void Wheel::innerObject() const {
87     glColor3f(0.0f, 0.0f, 0.0f);
88
89     glBegin(GL_TRIANGLE_FAN);
90     glVertex3f(posX, posZ, posY);
91     for (float alpha = 0.0f; alpha <= 2 * GL_PI; alpha += GL_PI / 32) {
92         float x1 = 0.1 * radius * cos(alpha);
93         float z1 = 0.1 * radius * sin(alpha);
94
95         glVertex3f(posX + x1, z1 + posZ, posY + 10);
96     }
97     glEnd();
98
99     glBegin(GL_TRIANGLE_STRIP);
100     for (float alpha = 0.0; alpha <= 2 * GL_PI + 1; alpha += GL_PI / 32){
101         float x1 = radius * cos(alpha);
102         float z1 = radius * sin(alpha);
103
104         float x2 = (radius - 10.0) * cos(alpha);
105         float z2 = (radius - 10.0) * sin(alpha);
106
107         glVertex3f(x1 + posX, z1 + posZ, posY);
108         glVertex3f(x2 + posX, z2 + posZ, posY + 5.0f);
109     }
110     glEnd();
111
112     for (float y1 = posY + 5.0; y1 < width + posY; y1 += 1.0f){
113         glBegin(GL_TRIANGLE_STRIP);
114         for (float alpha = 0.0f; alpha <= 2 * GL_PI + 1; alpha += GL_PI / 32){
115             float x1 = (radius - 10.0) * cos(alpha);

```

```

116         float z1 = (radius - 10.0) * sin(alpha);
117
118         glVertex3f(x1 + posX, z1 + posZ, y1);
119         glVertex3f(x1 + posX, z1 + posZ, y1 + 1.0f);
120     }
121     glEnd();
122 }
123
124 glBegin(GL_TRIANGLE_STRIP);
125 glColor3f(0.0,0.0,0.0);
126 for (float alpha = 0.0; alpha <= 2 * GL_PI + 1; alpha += GL_PI / 32){
127     float x1 = radius * cos(alpha);
128     float z1 = radius * sin(alpha);
129
130     float x2 = (radius - 15.0) * cos(alpha);
131     float z2 = (radius - 15.0) * sin(alpha);
132
133     glVertex3f(x1 + posX, z1 + posZ, posY);
134     glVertex3f(x2 + posX, z2 + posZ, posY + 10.0f);
135 }
136 glEnd();
137
138 for (float y1 = posY + 10.0; y1 < width + posY; y1 += 1.0f){
139     glBegin(GL_TRIANGLE_STRIP);
140     for (float alpha = 0.0f; alpha <= 2 * GL_PI + 1; alpha += GL_PI / 32){
141         float x1 = (radius - 15.0) * cos(alpha);
142         float z1 = (radius - 15.0) * sin(alpha);
143
144         glVertex3f(x1 + posX, z1 + posZ, y1);
145         glVertex3f(x1 + posX, z1 + posZ, y1 + 1.0f);
146     }
147     glEnd();
148 }
149
150 for (float y1 = posY + 10.0; y1 < width + posY; y1 += 1.0f){
151     glBegin(GL_TRIANGLE_STRIP);
152     glColor3f(0.0f, 0.0f, 0.0f);
153     for (float alpha = 0.0f; alpha <= 2 * GL_PI + 1; alpha += GL_PI / 32){
154         float x1 = (radius - 25.0) * cos(alpha);
155         float z1 = (radius - 25.0) * sin(alpha);
156
157         glVertex3f(x1 + posX, z1 + posZ, y1);
158         glVertex3f(x1 + posX, z1 + posZ, y1 + 1.0f);
159     }
160     glEnd();
161 }
162
163 for (float alpha = 0.0; alpha <= 2 * GL_PI; alpha += GL_PI / 4){
164     this->crochet(alpha);
165 }
166 }
167
168 void Wheel::crochet(const float alpha) const{
169     glColor3f(0.0, 0.0, 0.0);
170
171     glBegin(GL_TRIANGLE_FAN);
172
173     float x1 = 0.1 * radius * cos(alpha);
174     float z1 = 0.1 * radius * sin(alpha);
175
176     glVertex3f(x1 + posX, z1 + posZ, posY + 10);
177

```



```

178 float xLeft = (radius - 15.0) * cos(alpha - GL_PI / 32);
179 float zLeft = (radius - 15.0) * sin(alpha - GL_PI / 32);
180
181 glVertex3f(xLeft + posX, zLeft + posZ, posY + 10);
182
183 float xRight = (radius - 15.0) * cos(alpha + GL_PI / 32);
184 float zRight = (radius - 15.0) * sin(alpha + GL_PI / 32);
185
186 glVertex3f(xRight + posX, zRight + posZ, posY + 10);
187
188 glEnd();
189 }
190
191 void Wheel::protector() const{
192     glColor3f(0.0, 0.0, 0.0);
193     const float height = 3.0f, width = 5.0f, length = GL_PI / 32, spaceAngle =
        GL_PI / 64;
194     bool pos = 0;
195
196     for (float y = this->posY ; y <= this->width + this->posY; y += width + (
        this->width - 5 * width) / 4){
197         for (float alpha = pos == 0 ? length / 2 : 0.0; alpha <= 2 * GL_PI;
            alpha += length + spaceAngle){
198             this->cuboid(alpha, alpha + length, y, width, height);
199         }
200         pos = !pos;
201     }
202 }
203
204
205 void Wheel::draw() const{
206     this->outerObject();
207     this->innerObject();
208 }

```

1.3.2 class *SideSciana*

1.3.2.1 Opis działania

Klasa *SideSciana* odpowiada za rysowanie ścian bocznych samochodu.

1.3.2.2 Plik *SideSciana.h*

Plik nagłówkowy *SideSciana.h* deklaruje wszystkie zmienne oraz metody, które będą używane obiektami tej klasy.

```

1 #include "includes.h"
2 class SideSciana{
3 private:
4     float length{};
5     float width{};
6     float height{};
7     float posX{};
8     float posY{};
9     float posZ{};
10    float holeScianaLength, scianaLength;
11    float radius = 20.0;

```

```

12
13 void holeSciana(const float , const float) const;
14 void sciana() const;
15 public:
16 SideSciana(const float length, const float width, const float heigth,
17             const float posX, const float posY, const float posZ);
18 void draw() const;
19 };

```

1.3.2.3 Plik *SideSciana.cpp*

Plik *SideSciana.cpp* zawiera inicjalizacje zmiennych oraz metod opisanych w pliku nagłówkowym *SideSciana.h*.

```

1 #include "SideSciana.h"
2
3 SideSciana::SideSciana(const float length, const float width, const float
4     heigth, const float posX, const float posY, const float posZ) : length(
5     length), width(width), heigth(heigth), posX(posX), posY(posY), posZ(posZ)
6     ) {
7
8     holeScianaLength = (this->length / 3) + 10;
9     scianaLength = this->length / 3;
10 }
11
12 void SideSciana::holeSciana(const float posX, const float length) const {
13     float x {}, z {};
14
15     glBegin(GL_TRIANGLE_FAN);
16     glColor3f(0.5f, 0.5f, 0.5f);
17     glVertex3f(posX, this->posZ + heigth, this->posY + width);
18     glVertex3f(posX, this->posZ, this->posY + width);
19     for (float alpha = 0.0; alpha <= GL_PI / 2; alpha += GL_PI / 128){
20         x = ((length - 2 * radius) / 2) + (radius - radius * cos(alpha));
21         z = radius * sin(alpha);
22         glVertex3f(x + posX, z + this->posZ, this->posY + width);
23     }
24     glVertex3f(x + posX, this->posZ + heigth, this->posY + width);
25     glEnd();
26
27     glBegin(GL_TRIANGLE_FAN);
28     glColor3f(0.5f, 0.5f, 0.5f);
29     glVertex3f(posX + length, this->posZ + heigth, this->posY + width);
30     glVertex3f(posX + x, this->posZ + heigth, this->posY + width);
31     for (float alpha = GL_PI / 2; alpha >= 0; alpha -= GL_PI / 128){
32         x = (length / 2) + cos(alpha) * radius;
33         z = radius * sin(alpha);
34         glVertex3f(posX + x, z + this->posZ, this->posY + width);
35     }
36     glVertex3f(posX + length, this->posZ, this->posY + width);
37     glEnd();
38
39     glBegin(GL_TRIANGLE_FAN);
40     glColor3f(0.5f, 0.5f, 0.5f);
41     glVertex3f(posX, this->posZ + heigth, this->posY);
42     glVertex3f(posX, this->posZ, this->posY);
43     for (float alpha = 0; alpha <= GL_PI / 2; alpha += GL_PI / 128){
44         x = ((length - 2 * radius) / 2) + (radius - radius * cos(alpha));
45         z = radius * sin(alpha);
46         glVertex3f(posX + x, this->posZ + z, this->posY);
47     }
48 }

```

```

43     }
44     glVertex3f(posX + x, this->posZ + height, this->posY);
45 glEnd();
46
47 glBegin(GL_TRIANGLE_FAN);
48 glColor3f(0.5f, 0.5f, 0.5f);
49 glVertex3f(posX + length, this->posZ + height, this->posY);
50 glVertex3f(posX + x, this->posZ + height, this->posY);
51 for (float alpha = GL_PI / 2; alpha >= 0; alpha -= GL_PI / 128){
52     x = (length / 2) + cos(alpha) * radius;
53     z = radius * sin(alpha);
54     glVertex3f(posX + x, this->posZ + z, this->posY);
55 }
56 glVertex3f(posX + length, this->posZ, this->posY);
57 glEnd();
58
59 for (float y = this->posY; y < this->posY + width; y += 10.0f){
60     glBegin(GL_TRIANGLE_STRIP);
61     for (float alpha = 0.0f; alpha <= GL_PI / 2; alpha += GL_PI / 128){
62         x = ((length - 2 * radius) / 2) + (radius - radius * cos(alpha));
63         z = radius * sin(alpha);
64         glColor3f(0.0f, 1.0f, 1.0f);
65         glVertex3f(posX + x, this->posZ + z, y);
66         glVertex3f(posX + x, this->posZ + z, y + 10.0f);
67     }
68     glEnd();
69 }
70
71 for (float y = this->posY; y < this->posY + width; y += 10.0f) {
72     glBegin(GL_TRIANGLE_STRIP);
73     for (float alpha = GL_PI / 2; alpha >= 0; alpha -= GL_PI / 128){
74         x = (length / 2) + radius * cos(alpha);
75         z = radius * sin(alpha);
76         glColor3f(0.0f, 1.0f, 1.0f);
77         glVertex3f(posX + x, this->posZ + z, y);
78         glVertex3f(posX + x, this->posZ + z, y + 10.0f);
79     }
80     glEnd();
81 }
82 }
83
84 void SideSciana::sciana() const{
85     for (float x = this->posX + holeScianaLength; x < this->posX +
86         scianaLength + holeScianaLength; x += 10.0){
87         glBegin(GL_TRIANGLE_STRIP);
88         for (float z = this->posZ; z <= this->posZ + height; z += 10.0){
89             glVertex3f(x, z, this->posY + width);
90             glVertex3f(x + 10.0, z, this->posY + width);
91         }
92         glEnd();
93     }
94
95     for (float x = this->posX + holeScianaLength; x < this->posX +
96         scianaLength + holeScianaLength; x += 10.0){
97         glBegin(GL_TRIANGLE_STRIP);
98         for (float z = this->posZ; z <= this->posZ + height; z += 10.0){
99             glVertex3f(x, z, this->posY);
100             glVertex3f(x + 10.0, z, this->posY);
101         }
102         glEnd();
103     }
104 }

```

```

103     for (float x = posX; x < posX + this->length + radius; x += 10.0f){
104         glBegin(GL_TRIANGLE_STRIP);
105         for (float y = this->posY; y <= this->posY + width; y += 10.0f){
106             glColor3f(0.5f, 0.5f, 0.5f);
107             glVertex3f(x, this->posZ + height, y);
108             glVertex3f(x + 10.0f, this->posZ + height, y);
109         }
110         glEnd();
111     }
112 }
113
114 void SideSciana::draw() const {
115     this->holeSciana(this->posX, holeScianaLength);
116     glColor3f(0.5, 0.5, 0.5);
117     this->sciana();
118     this->holeSciana(this->posX + scianaLength + holeScianaLength,
119                     holeScianaLength);
119 }

```

1.3.3 class *Front*

1.3.3.1 Opis działania

Klasa *Front* odpowiada za rysowanie przdniej ściany samochodu.

1.3.3.2 Plik *Front.h*

Plik nagłówkowy *Front.h* deklaruje wszystkie zmienne oraz metody, które będą używane obiektami tej klasy.

```

1  #include "includes.h"
2  class Front{
3  private:
4      float length{};
5      float width{};
6      float height{};
7      float posX{};
8      float posY{};
9      float posZ{};
10
11     void sciana(const float, const float, const float, const float) const;
12     void scinal(const float posX, const float posY, const float posZ, const
13                float angle, const float height) const;
14     void lightsaber(const float, const float, const float, const float) const;
15 public:
16     Front(const float, const float, const float, const float, const float,
17           const float);
18     void draw() const;
19 };

```

1.3.3.3 Plik *Front.cpp*

Plik *Front.cpp* zawiera inicjalizacje zmiennych oraz metod opisanych w pliku nagłówkowym *Front.h*.

```

1  #include "Front.h"
2
3  Front::Front(const float length, const float width, const float height,
4              const float posX, const float posY, const float posZ) : length(length),
5              width(width), height(height), posX(posX), posY(posY), posZ(posZ) {}
6
7  void Front::sciana(const float posX, const float posY, const float angle,
8                    const float length) const{
9      for (float z = this->posZ; z < height; z += 1.0){
10         glBegin(GL_TRIANGLE_STRIP);
11         for (float y = posY; y <= length + posY; y += 1.0){
12             glVertex3f(posX, z, y);
13             glVertex3f(posX, z + 1.0, y);
14         }
15         glEnd();
16     }
17 }
18
19 void Front::scinal(const float posX, const float posY, const float posZ,
20                   const float angle, const float height) const{
21     for (float z = posZ; z < posZ + height; z += 1.0){
22         glBegin(GL_TRIANGLE_STRIP);
23         for (float length = 0.0; length <= this->length / 2; length += 1.0){
24             float x = posX + length * cos(angle);
25             float y = posY + length * sin(angle);
26
27             glVertex3f(x, z, y);
28             glVertex3f(x, z + 1, y);
29         }
30         glEnd();
31     }
32 }
33
34 void Front::lightsaber(const float posX, const float posY, const float
35                        radius, const float angle) const{
36     float x, y, z;
37     glBegin(GL_TRIANGLE_FAN);
38     glVertex3f(posX, this->posZ + height - radius, posY + radius);
39     for (float alpha = 0.0; alpha <= 2 * GL_PI + 1; alpha += GL_PI / 128){
40         y = radius * sin(alpha);
41         z = radius * cos(alpha);
42         x = y * (cos(angle) / sin(angle));
43
44         glVertex3f(posX + x, this->posZ + height - z - radius, posY + y +
45                    radius);
46     }
47     glEnd();
48
49     for (float length = posX; length < posX + 3; length += 1.0){
50         glBegin(GL_TRIANGLE_STRIP);
51         for (float alpha = 0.0; alpha <= 2 * GL_PI + 1; alpha += GL_PI /
52            128){
53             y = radius * sin(alpha);
54             z = radius * cos(alpha);
55             x = y * (cos(angle) / sin(angle));
56
57             glVertex3f(x + length, this->posZ + height - z - radius, posY +
58                        y + radius);
59             glVertex3f(x + length + 1.0, this->posZ + height - z - radius,
60                        posY + y + radius);
61         }
62     }
63 }

```

```

53         glEnd();
54     }
55 }
56
57 void Front::draw() const{
58     glColor3f(1.0, 0.0, 0.0);
59
60     this->sciana1(this->posX, this->posY, this->posZ, GL_PI / 2.1, this->
        height);
61     this->sciana1(this->posX + (this->length / 2) * cos(GL_PI / 2.1), this->
        posY + (this->length / 2) * sin(GL_PI / 2.1), this->posZ, GL_PI /
        2.4, this->height);
62     glColor3f(1.0, 1.0, 0.0);
63
64     this->lightsaber(this->posX - 3.0, this->posY, 5, GL_PI / 2.1);
65     this->lightsaber(this->posX + 6.0, this->length - 5, 5, -GL_PI / 1.7);
66 }

```

1.3.4 class *Back*

1.3.4.1 Opis działania

Klasa *Back* odpowiada za rysowanie tylnej ściany samochodu.

1.3.4.2 Plik *Back.h*

Plik nagłówkowy *Back.h* deklaruje wszystkie zmienne oraz metody, które będą używane obiektami tej klasy.

```

1 #include "includes.h"
2 class Back{
3 private:
4     float length{};
5     float width{};
6     float height{};
7     float posX{};
8     float posY{};
9     float posZ{};
10
11     void scianaW(const float, const float, const float, const float, const
        float) const;
12     void scianaL(const float, const float, const float, const float, const
        float) const;
13     void rama(const float, const float, const float, const float, const float,
        const float) const;
14     void lightsaber(const float, const float, const float, const float, const
        float, const float) const;
15 public:
16     Back(const float, const float, const float, const float, const float,
        const float);
17     void draw() const;
18 };

```

1.3.4.3 Plik *Back.cpp*

Plik *Back.cpp* zawiera inicjalizacje zmiennych oraz metod opisanych w pliku nagłówkowym *Back.h*.

```
1 #include "Back.h"
2
3 Back::Back(const float length, const float width, const float height, const
   float posX, const float posY, const float posZ) : length(length), width(
   width), height(height), posX(posX), posY(posY), posZ(posZ){}
4
5 void Back::scianaW(const float width, const float height, const float posX,
   const float posY, const float posZ) const{
6     for (float z = posZ; z < posZ + height; z += 1.0){
7         glBegin(GL_TRIANGLE_STRIP);
8         for (float y = posY; y <= posY + width; y += 1.0){
9             glVertex3f(posX, z, y);
10            glVertex3f(posX, z + 1.0, y);
11        }
12        glEnd();
13    }
14 }
15
16 void Back::scianaL(const float length, const float height, const float posX,
   const float posY, const float posZ) const{
17     for (float z = posZ; z < posZ + height; z += 1.0){
18         glBegin(GL_TRIANGLE_STRIP);
19         for (float x = posX; x <= posX + length; x += 1.0){
20             glVertex3f(x, z, posY);
21             glVertex3f(x, z + 1.0, posY);
22         }
23         glEnd();
24     }
25 }
26
27 void Back::rama(const float length, const float width, const float height,
   const float posX, const float posY, const float posZ) const{
28     this->scianaW(width, height, posX, posY, posZ);
29     this->scianaL(length, height, posX, posY, posZ);
30     this->scianaL(length, height, posX, posY + width, posZ);
31
32     for (float x = posX + length - 1; x >= posX; x -= 1.0){
33         glBegin(GL_TRIANGLE_STRIP);
34         for (float y = posY; y <= this->posY; y += 1.0){
35             glVertex3f(x, posZ + height, y);
36             glVertex3f(x + 1.0, posZ + height, y);
37         }
38         glEnd();
39
40         glBegin(GL_TRIANGLE_STRIP);
41         for (float y = posY; y <= this->posY; y += 1.0){
42             glVertex3f(x, posZ, y);
43             glVertex3f(x + 1.0, posZ, y);
44         }
45         glEnd();
46
47         glBegin(GL_TRIANGLE_STRIP);
48         for (float y = posY + width; y >= this->posY + this->width; y -= 1.0){
49             glVertex3f(x, posZ + height, y);
50             glVertex3f(x + 1.0, posZ + height, y);
51     }
```

```

52     glEnd();
53
54     glBegin(GL_TRIANGLE_STRIP);
55     for (float y = posY + width; y >= this->posY + this->width; y -= 1.0){
56         glVertex3f(x, posZ + height, y);
57         glVertex3f(x + 1.0, posZ + height, y);
58     }
59     glEnd();
60 }
61
62 for (float y = this->posY; y <= this->posY + this->width; y += 1.0){
63     glBegin(GL_TRIANGLE_STRIP);
64     for (float x = posX; x <= this->posX; x += 1.0){
65         glVertex3f(x, posZ + height, y);
66         glVertex3f(x, posZ + height, y + 1.0);
67     }
68     glEnd();
69
70     glBegin(GL_TRIANGLE_STRIP);
71     for (float x = posX; x <= this->posX; x += 1.0){
72         glVertex3f(x, posZ, y);
73         glVertex3f(x, posZ, y + 1.0);
74     }
75     glEnd();
76 }
77
78 for (float z = posZ; z < posZ + height; z += 1.0){
79     glBegin(GL_TRIANGLE_STRIP);
80     glColor3f(1.0, 1.0, 0.0);
81     for (float y = posY; y <= this->posY; y += 1.0){
82         glVertex3f(posX + length, z, y);
83         glVertex3f(posX + length, z + 1.0, y);
84     }
85     glEnd();
86
87     glBegin(GL_TRIANGLE_STRIP);
88     for (float y = posY + width; y >= this->posY + this->width; y -= 1.0){
89         glVertex3f(posX + length, z, y);
90         glVertex3f(posX + length, z + 1.0, y);
91     }
92     glEnd();
93 }
94 }
95
96 void Back::lightsaber(const float length, const float width, const float
    height, const float posX, const float posY, const float posZ) const{
97     for (float z = posZ; z < posZ + height; z += 1.0){
98         glBegin(GL_TRIANGLE_STRIP);
99         glColor3f(1.0, 0.0, 0.0);
100         for (float y = posY; y <= posY + width; y += 1.0){
101             glVertex3f(posX, z, y);
102             glVertex3f(posX, z + 1.0, y);
103         }
104         glEnd();
105     }
106
107     for (float x = posX; x <= this->posX; x += 1.0){
108         glBegin(GL_TRIANGLE_STRIP);
109         for (float z = posZ; z <= posZ + height; z += 1.0){
110             glVertex3f(x, z, posY);
111             glVertex3f(x + 1.0, z, posY);
112         }

```



```

113     glEnd();
114
115     glBegin(GL_TRIANGLE_STRIP);
116     for (float z = posZ; z <= posZ + heigth; z += 1.0){
117         glVertex3f(x, z, posY + width);
118         glVertex3f(x + 1.0, z, posY + width);
119     }
120     glEnd();
121 }
122 }
123
124 void Back::draw() const{
125     glColor3f(0.5, 0.5, 0.5);
126     this->scianaW(this->width, this->heigth, this->posX, this->posY, this->
        posZ);
127     glColor3f(1.0, 1.0, 0.0);
128     this->rama(this->length, this->width + 4, 0.2 * this->heigth, this->posX -
        2, this->posY - 2, (this->posZ + this->heigth) * 0.6);
129     this->lightsaber(4, 5, 5, this->posX - 2, this->posY, this->posZ);
130     this->lightsaber(4, 5, 5, this->posX - 2, this->posY+75, this->posZ);
131 }

```

Rozdział 2

Budowa otoczenia

2.1 Opis zadania

Należy zbudować elementy otoczenia, w którym będzie poruszał się robot rolniczy wykorzystując wyłącznie prymitywy bazujące na trójkącie. Elementy te będą wykorzystywane na kolejnych zajęciach i będą powiązanie z fabułą gry.

2.2 Wymagania

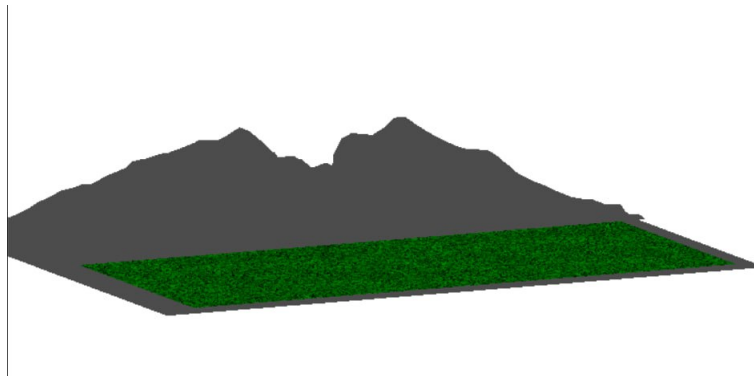
Wymagania dotyczące budowy otoczenia:

- Na ocenę 3: Przygotowanie otoczenia o podłożu płaskim oraz utworzenie dwóch obiektów dodatkowych (drzewo, bramka, budynek).
- Na ocenę 4: Przygotowanie otoczenia o podłożu nieregularnym (góra, stadion, wyboista ziemia) oraz utworzenie jednego obiektu dodatkowego.
- Na ocenę 5: Import otoczenia z programu graficznego (otoczenie o podłożu nieregularnym i minimum 1 obiekt dodatkowy).

2.3 Realizacja zadania

2.3.1 class Podloze

Klasa *Podloze* odpowiada za rysowanie otoczenia. Plik *Podloze.cpp* zawiera ponad 29 tys. linii kodu więc nie będzie umieszczony w sprawozdaniu.



Rysunek 2.1: Podłoże

2.3.1.1 Opis działania

Podłoże zostało eksportowane z programu *Blender*.

2.3.1.2 Plik *Podloze.h*

```
1 #pragma once
2 #include "includes.h"
3 class Podloze
4 {
5 public:
6     void draw();
7 };
```

Rozdział 3

Teksturowanie

3.1 Opis zadania

Należy dokonać teksturowania według przedstawionych poniżej kryteriów.

3.2 Wymagania

Wymagania dotyczące dodania teksurowania.

- Na ocenę 3: Teksturowanie obiektów otoczenia oraz utworzenie autorskiego rozwiązania sterowaniem kamerą.
- Na ocenę 4: Jak na ocenę 3 oraz teksturowanie powierzchni.
- Na ocenę 5: Jak na ocenę 4 oraz teksturowanie obiektu, który będzie sterowany (minimum 3 bryły).

3.3 Realizacja zadania

```
1  glColor3f(0.0, 1.0, 0.0);
2  glEnable(GL_TEXTURE_2D);
3  glBindTexture(GL_TEXTURE_2D, texture[1]);
4  glBegin(GL_QUADS);
5  glNormal3d(0, 0, 1);
6  glTexCoord2d(1.0, 1.0); glVertex3d(-100, 3.3, -196);
7  glTexCoord2d(0.0, 1.0); glVertex3d(78, 3.3, -196);
8  glTexCoord2d(0.0, 0.0); glVertex3d(78, 3.3, 196);
9  glTexCoord2d(1.0, 0.0); glVertex3d(-100, 3.3, 196);
10 glEnd();
11 glDisable(GL_TEXTURE_2D);
```

Rozdział 4

Sterowanie obiektem głównym

4.1 Opis zadania

Należy dokonać sterowanie obiektem głównym.

4.2 Wymagania

Wymagania dotyczące sterowania obiektem głównym.

- Na ocenę 3: Realizacja prostego sterowanie przód-tył i obrót wokół własnej osi.
- Na ocenę 4: Implementacja prostej fizyki sterowania (w przypadku łazika różnica prędkości na gąsienicach lub oś skrętna).
- Na ocenę 5: Jak na ocenę 4 oraz implementacja podstawowych zagadnień fizycznych np. pęd ciała.

4.3 Realizacja zadania

W pliku głównym *main.cpp* są 8 zmiennych odpowiadających za naciśnięcie klawisz.

```
1 bool keyWPressed{ false };  
2 bool keySPressed{ false };  
3 bool keyAPressed{ false };  
4 bool keyDPressed{ false };  
5 bool keyQPressed{ false };  
6 bool keyEPressed{ false };  
7 bool keyXPressed{ false };  
8 bool keyZPressed{ false };
```

Niżej w tym samym pliku znajduje się `switch\case`, który cały czas sprawdza czy klawisze są naciśnięte i w zależności od tego zmienne przyjmują inne wartości(`true` albo `false`).

```
1  case WM_KEYDOWN:{
2      switch (wParam) {
3          case 0x57: // W
4              keyWPressed = true;
5              break;
6          case 0x53: // S
7              keySPressed = true;
8              break;
9          case 0x41: // A
10             keyAPressed = true;
11             break;
12          case 0x44: // D
13             keyDPressed = true;
14             break;
15          case 0x51: // Q
16             keyQPressed = true;
17             break;
18          case 0x45: // E
19             keyEPressed = true;
20             break;
21          case 0x58: // X
22             keyXPressed = true;
23             break;
24          case 0x5A: // Z
25             keyZPressed = true;
26             break;
27          case VK_UP:
28             xRot -= 5.0 f;
29             break;
30          case VK_DOWN:
31             xRot += 5.0 f;
32             break;
33          case VK_LEFT:
34             yRot -= 5.0 f;
35             break;
36          case VK_RIGHT:
37             yRot += 5.0 f;
38             break;
39      }
40      xRot = (const int)xRot % 360;
41      yRot = (const int)yRot % 360;
42
43      InvalidateRect(hWnd, NULL, FALSE);
44  }
45  break;
46  case WM_KEYUP: {
47      switch (wParam) {
48          case 0x57: // W
49              keyWPressed = false;
50              break;
51          case 0x53: // S
52              keySPressed = false;
53              break;
54          case 0x41: // A
55              keyAPressed = false;
56              break;
57          case 0x44: // D
```

```

58         keyDPressed = false;
59         break;
60     case 0x51: // Q
61         keyQPressed = false;
62         break;
63     case 0x45: // E
64         keyEPressed = false;
65         break;
66     case 0x58: // X
67         keyXPressed = false;
68         break;
69     case 0x5A: // Z
70         keyZPressed = false;
71         break;
72     }
73     break;
74 }

```

W funkcji *RenderScene* znajduje się fragment kodu, który sprawdza stany zmiennych.

```

1  if (keyWPressed)
2      posX += 1.f;
3  if (keySPressed)
4      posX -= 1.f;
5  if (keyAPressed)
6      posZ -= 1.f;
7  if (keyDPressed)
8      posZ += 1.f;
9  if (keyQPressed)
10     posY += 1.f;
11  if (keyEPressed)
12     posY -= 1.f;
13  if (keyXPressed)
14     yAngleRotate += yAngleStep;
15  if (keyZPressed)
16     yAngleRotate -= yAngleStep;

```

W zależności od wartości zmiennej będą również zmienione zmienne: *posX*, *posY*, *posZ*. Wartości odpowiadają za pozycję samochodu na płaszczyźnie.