3D_Viewer_v1

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Ui Namespace Reference

Chapter 6

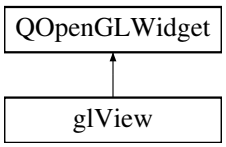
Class Documentation

6.1 glView Class Reference

Class for working with QWidget The main task of the class is to visualize the model.

`#include <glview.h>`

Inheritance diagram for glView:



Public Slots

- void open_obj ()

  Slot to open .obj file.

Signals

- void set_counts (int vertex_count, int line_count)

  Signal to transmit the number of vertices and the number of lines.

Public Member Functions

- glView (QWidget *parent=nullptr)

  Class constructor.
- ~glView ()

  Class destructor.
- void update_figure ()

  The widget redrawing.
- bool saveImage (const QString &fileName, const QString &format)

  Save file in bmp or jpeg format.

Public Attributes

- QString FileName = NULL

  The file name.
- float x_move

  The value to move the model along the x-axis.
- float y_move

  The value to move the model along the y-axis.
- float z_move

  The value to move the model along the z-axis.
- float x_rotate

  The value to rotate the model along the x-axis.
- float y_rotate

  The value to rotate the model along the y-axis.
- float z_rotate

  The value to rotate the model along the z-axis.
- float scale_change

  The value to scale the model.
- bool flag_solid_dotted_line
- int value_projection
- bool flag_projection = false
- int value_background_color
- bool flag_background_color
- int value_line_color
- int value_vertex_type
- int custom_vertex_size
- int custom_line_size
- int value_vertex_color

Private Member Functions

- void initializeGL () override

  function called at initialization glView.
- void resizeGL (int w, int h) override

  Setting the size of the output area.
- void paintGL () override

  The widget drawing.
- void draw_figure ()

  The model drawing.
- void set_projection ()

  Setting the projection of the model: parallel or central.
- void centering ()

  Place the model in the center of the area.
- void scaling ()

  Setting model size.
- void set_background_color ()

  Setting the background color.
- void set_line_color ()

  Setting the line color.
- void set_vertex_type ()

  Setting the type of verteces.

- void set_vertex_size ()

  Setting the size of verteces.
- void set_vertex_color ()

  Setting the color of verteces.
- void move_for_asix_x (double value)

  The moving the model along the x-axis.
- void move_for_asix_y (double value)

  The moving the model along the y-axis.
- void move_for_asix_z (double value)

  The moving the model along the z-axis.
- void rotate_for_asix_x (double value)

  The rotation the model along the x-axis.
- void rotate_for_asix_y (double value)

  The rotation the model along the y-axis.
- void rotate_for_asix_z (double value)

  The rotation the model along the z-axis.

Private Attributes

- struct info_figure figure

  The information about the model.
- bool flag_centering
- float centr_x

  The value of the center of the model along the x-axis.
- float centr_y

  The value of the center of the model along the y-axis.
- float centr_z

  The value of the center of the model along the z-axis.
- double scaleFactor_x
- double scaleFactor_y

  The coefficient for moving the model along the y-axis.
- double scaleFactor_z

  The coefficient for moving the model along the z-axis.

6.1.1 Detailed Description

Class for working with QWidget The main task of the class is to visualize the model.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 glView()

```
glView::glView (
            QWidget * parent = nullptr )
```

Class constructor.

Parameters

paren	the reference to parent object.

```
5                                : QOpenGLWidget(parent) {
6   x_move = 0;
7   y_move = 0;
8   z_move = 0;
9   x_rotate = 0;
10   y_rotate = 0;
11   z_rotate = 0;
12   scale_change = 0;
13   flag_solid_dotted_line = false;
14   flag_projection = false;
15   value_background_color = 1;
16   value_line_color = 1;
17   value_vertex_type = 1;
18   value_vertex_color = 1;
19   custom_vertex_size = 1;
20   custom_line_size = 1;
21   flag_background_color = true;
22   flag_centering = false;
23 }
```

6.1.2.2 ~glView()

```
glView::~glView ( )
```

Class destructor.

```
25 { free_matrix(&figure); }
```

6.1.3 Member Function Documentation

6.1.3.1 centering()

```
void glView::centering ( )  [private]
```

Place the model in the center of the area.

```
183                          {
184   centr_x = figure.x_min + (figure.x_max - figure.x_min) / 2;
185   centr_y = figure.y_min + (figure.y_max - figure.y_min) / 2;
186   centr_z = figure.z_min + (figure.z_max - figure.z_min) / 2;
187   move_model(figure.matrix_of_vertex_origin, 0, figure.number_of_vertex * 3,
188              -centr_x);
189   move_model(figure.matrix_of_vertex_origin, 1, figure.number_of_vertex * 3,
190              -centr_y);
191   move_model(figure.matrix_of_vertex_origin, 2, figure.number_of_vertex * 3,
192              -centr_z);
193   flag_centering = false;
194   figure.x_min -= centr_x;
195   figure.x_max -= centr_x;
196   figure.y_min -= centr_y;
197   figure.y_max -= centr_y;
198   figure.z_min -= centr_z;
199   figure.z_max -= centr_z;
200   double diagonal_x = std::sqrt(std::pow(figure.x_max - figure.x_min, 2));
201   double diagonal_y = std::sqrt(std::pow(figure.y_max - figure.y_min, 2));
202   double diagonal_z = std::sqrt(std::pow(figure.z_max - figure.z_min, 2));
203   scaleFactor_x = diagonal_x / 100.0;
204   scaleFactor_y = diagonal_y / 100.0;
205   scaleFactor_z = diagonal_z / 100.0;
206 }
```

6.1.3.2 draw_figure()

```
void glView::draw_figure ( )  [private]
```

The model drawing.

```
94                          {
95   if (figure.load_result == 1) {
96     glEnable(GL_DEPTH_TEST);
97     glDepthFunc(GL_LESS);
98
99     glVertexPointer(3, GL_FLOAT, 0, figure.matrix_of_vertex);
100     glEnableClientState(GL_VERTEX_ARRAY);
101
102     glLineStipple(1, 0x00FF);
103     glLineWidth(custom_line_size);
104     if (flag_solid_dotted_line) {
105       glEnable(GL_LINE_STIPPLE);
106     }
107     if (value_line_color) {
108       set_line_color();
109     }
110     glDrawElements(GL_LINES, figure.number_of_faces_all * 2, GL_UNSIGNED_INT,
111                    figure.matrix_of_index);
112     glDisable(GL_LINE_STIPPLE);
113
114     if (value_vertex_type != 1 && custom_vertex_size != 0) {
115       if (value_vertex_type == 2)
116         glEnable(GL_POINT_SMOOTH);
117       else
118         glDisable(GL_POINT_SMOOTH);
119       set_vertex_size();
120       set_vertex_color();
121       glDrawArrays(GL_POINTS, 0, figure.number_of_vertex);
122     }
123
124     glDisableClientState(GL_VERTEX_ARRAY);
125     glDisable(GL_DEPTH_TEST);
126   }
127 }
```

6.1.3.3 initializeGL()

```
void glView::initializeGL ( )  [override], [private]
```

function called at initialization glView.

```
27                          {
28   set_background_color();
29   glEnable(GL_DEPTH_TEST);
30 }
```

6.1.3.4 move_for_asix_x()

```
void glView::move_for_asix_x (
              double value )  [private]
```

The moving the model along the x-axis.

Parameters

value	value to move the model.

```
226                                     {
227   move_model(figure.matrix_of_vertex, 0, figure.number_of_vertex * 3,
```

```
228             value * scaleFactor_x);
229 }
```

6.1.3.5 move_for_asix_y()

```
void glView::move_for_asix_y (
             double value )  [private]
```

The moving the model along the y-axis.

Parameters

value	value to move the model.

```
231                                         {
232   move_model(figure.matrix_of_vertex, 1, figure.number_of_vertex * 3,
233             value * scaleFactor_y);
234 }
```

6.1.3.6 move_for_asix_z()

```
void glView::move_for_asix_z (
             double value )  [private]
```

The moving the model along the z-axis.

Parameters

value	value to move the model.

```
236                                         {
237   move_model(figure.matrix_of_vertex, 2, figure.number_of_vertex * 3,
238             value * scaleFactor_z);
239 }
```

6.1.3.7 open_obj

```
void glView::open_obj ( )  [slot]
```

Slot to open .obj file.

```
213                       {
214   if (!FileName.isEmpty()) {
215     free_matrix(&figure);
216     QByteArray ba = FileName.toLocal8Bit();
217     char* name_of_file = ba.data();
218     open_file(name_of_file, &figure);
219     flag_centering = true;
220     flag_projection = true;
221     update();
222   }
223   emit set_counts(figure.number_of_vertex, figure.number_of_faces_all);
224 }
```

6.1.3.8 paintGL()

```
void glView::paintGL ( )  [override], [private]
```

The widget drawing.

```
33                      {
34   if (flag_centering) {
35     centering();
36   }
37   if (flag_projection) {
38     set_projection();
39   }
40   if (flag_background_color) {
41     set_background_color();
42   }
43   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
44
45   for (int i = 0; i < figure.number_of_vertex * 3; ++i) {
46     figure.matrix_of_vertex[i] = figure.matrix_of_vertex_origin[i];
47   }
48   if (x_rotate) {
49     rotate_for_asix_x(x_rotate);
50   }
51   if (y_rotate) {
52     rotate_for_asix_y(y_rotate);
53   }
54   if (z_rotate) {
55     rotate_for_asix_z(z_rotate);
56   }
57
58   if (x_move) {
59     move_for_asix_x(x_move);
60   }
61   if (y_move) {
62     move_for_asix_y(y_move);
63   }
64   if (z_move) {
65     move_for_asix_z(z_move);
66   }
67
68   if (scale_change) {
69     scaling();
70   }
71
72   glMatrixMode(GL_MODELVIEW);
73   glLoadIdentity();
74
75   draw_figure();
76 }
```

6.1.3.9 resizeGL()

```
void glView::resizeGL (
              int w,
              int h )  [override], [private]
```

Setting the size of the output area.

Parameters

w	width output area
h	height output area

```
31 { glViewport(0, 0, w, h); }
```

6.1.3.10 rotate_for_asix_x()

```
void glView::rotate_for_asix_x (
            double value )  [private]
```

The rotation the model along the x-axis.

Parameters

value	value to rotate the model.

```
243                                         {
244   rotation_by_ox(figure.matrix_of_vertex, figure.number_of_vertex, value);
245 }
```

6.1.3.11 rotate_for_asix_y()

```
void glView::rotate_for_asix_y (
            double value )  [private]
```

The rotation the model along the y-axis.

Parameters

value	value to rotate the model

```
247                                         {
248   rotation_by_oy(figure.matrix_of_vertex, figure.number_of_vertex, value);
249 }
```

6.1.3.12 rotate_for_asix_z()

```
void glView::rotate_for_asix_z (
            double value )  [private]
```

The rotation the model along the z-axis.

Parameters

value	value to rotate the model.

```
251                                         {
252   rotation_by_oz(figure.matrix_of_vertex, figure.number_of_vertex, value);
253 }
```

6.1.3.13 saveImage()

```
bool glView::saveImage (
            const QString & fileName,
            const QString & format )
```

Save file in bmp or jpeg format.

Parameters

fileName	name of the file to save.
format	format of the file to save: bmp or jpeg.

Returns

true if the file was saved, false if the file could not be saved.

```
255                                                                    {
256   makeCurrent();  // setting the current OpenGL context
257   QSize size = this->size();
258   int width = size.width();
259   int height = size.height();
260   QOpenGLFramebufferObject fbo(width, height);  // creating a frame buffer
261   fbo.bind();                                   // making it current
262   paintGL();                                    // draw a scene into it
263   QImage image = fbo.toImage();  // getting an image from the frame buffer
264   return image.save(
265       fileName,
266       format.toStdString().c_str());  // saving the image to a file
267 }
```

6.1.3.14 scaling()

```
void glView::scaling ( )  [private]
```

Setting model size.

```
208                     {
209   scale_model(figure.matrix_of_vertex, figure.number_of_vertex * 3,
210               scale_change);
211 }
```

6.1.3.15 set_background_color()

```
void glView::set_background_color ( )  [private]
```

Setting the background color.

```
129                                 {
130   if (value_background_color == 1) {
131     glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
132   } else if (value_background_color == 2) {
133     glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
134   } else if (value_background_color == 3) {
135     glClearColor(1.0f, 0.0f, 0.0f, 1.0f);
136   } else if (value_background_color == 4) {
137     glClearColor(0.0f, 0.0f, 1.0f, 1.0f);
138   } else if (value_background_color == 5) {
139     glClearColor(0.0f, 1.0f, 0.0f, 1.0f);
140   }
141   flag_background_color = false;
142 }
```

6.1.3.16 set_counts

```
void glView::set_counts (
             int vertex_count,
             int line_count )  [signal]
```

Signal to transmit the number of vertices and the number of lines.

Parameters

vertex_count	The number of model vertices.
line_count	The number of model lines.

6.1.3.17 set_line_color()

```
void glView::set_line_color ( )  [private]
```

Setting the line color.

```
144                                {
145   if (value_line_color == 1) {
146     glColor3f(1.0f, 1.0f, 1.0f);
147   } else if (value_line_color == 2) {
148     glColor3f(0.0f, 0.0f, 0.0f);
149   } else if (value_line_color == 3) {
150     glColor3f(1.0f, 0.0f, 0.0f);
151   } else if (value_line_color == 4) {
152     glColor3f(0.0f, 0.0f, 1.0f);
153   } else if (value_line_color == 5) {
154     glColor3f(0.0f, 1.0f, 0.0f);
155   }
156 }
```

6.1.3.18 set_projection()

```
void glView::set_projection ( )  [private]
```

Setting the projection of the model: parallel or central.

```
78                                 {
79   glMatrixMode(GL_PROJECTION);
80   glLoadIdentity();
81
82   float view = 5 * (figure.z_max / (2 * tan(60.0 * M_PI / 180 / 2)));
83   if (value_projection) {
84     glOrtho(figure.x_min * 2.5, figure.x_max * 2.5, figure.y_min * 2.5,
85             figure.y_max * 2.5, view, 5000);
86   } else {
87     glFrustum(figure.x_min * 1.25, figure.x_max * 1.25, figure.y_min * 1.25,
88               figure.y_max * 1.25, view, 5000);
89   }
90   glTranslated(0, 0, -view * 2);
91   flag_projection = false;
92 }
```

6.1.3.19 set_vertex_color()

```
void glView::set_vertex_color ( )  [private]
```

Setting the color of verteces.

```
158                                  {
159   if (value_vertex_color == 1) {
160     glColor3f(0.0f, 0.0f, 0.0f);
161   } else if (value_vertex_color == 2) {
162     glColor3f(1.0f, 1.0f, 1.0f);
163   } else if (value_vertex_color == 3) {
164     glColor3f(1.0f, 0.0f, 0.0f);
165   } else if (value_vertex_color == 4) {
166     glColor3f(0.0f, 0.0f, 1.0f);
167   } else if (value_vertex_color == 5) {
168     glColor3f(0.0f, 1.0f, 0.0f);
169   }
170 }
```

6.1.3.20 set_vertex_size()

```
void glView::set_vertex_size ( )  [private]
```

Setting the size of verteces.

```
172 { glPointSize(custom_vertex_size); }
```

6.1.3.21 set_vertex_type()

```
void glView::set_vertex_type ( )  [private]
```

Setting the type of verteces.

```
174                                {
175   if (value_vertex_type == 2) {
176     glEnableClientState(GL_VERTEX_ARRAY);
177     glVertexPointer(3, GL_FLOAT, 0, figure.matrix_of_vertex_origin);
178     glDrawArrays(GL_POINTS, 0, figure.number_of_vertex * 3);
179     glDisableClientState(GL_VERTEX_ARRAY);
180   }
181 }
```

6.1.3.22 update_figure()

```
void glView::update_figure ( )
```

The widget redrawing.

```
241 { update(); }
```

6.1.4 Member Data Documentation

6.1.4.1 centr_x

```
float glView::centr_x  [private]
```

Initial value:

```
=
            0.0
```

The value of the center of the model along the x-axis.

6.1.4.2 centr_y

```
float glView::centr_y  [private]
```

Initial value:

```
=
          0.0
```

The value of the center of the model along the y-axis.

6.1.4.3 centr_z

```
float glView::centr_z  [private]
```

Initial value:

```
=
          0.0
```

The value of the center of the model along the z-axis.

6.1.4.4 custom_line_size

```
int glView::custom_line_size
```

The value for custom line size: 1 - min line size, 10 - max line size.

6.1.4.5 custom_vertex_size

```
int glView::custom_vertex_size
```

The value for custom vertex size: 0 - no vertex, 30 - max vertex size.

6.1.4.6 figure

```
struct info_figure glView::figure  [private]
```

Initial value:

```
= {
      0, 0, NULL, NULL, 0,      0,
      0, 0, 0,    0,    false, NULL}
```

The information about the model.

6.1.4.7 FileName

```
QString glView::FileName = NULL
```

The file name.

6.1.4.8 flag_background_color

```
bool glView::flag_background_color
```

The flag indicating that the background color needs to be set.

6.1.4.9 flag_centering

```
bool glView::flag_centering  [private]
```

The flag indicating that the model needs to be centered.

6.1.4.10 flag_projection

```
bool glView::flag_projection = false
```

The flag indicating that the projection matrix needs to be set.

6.1.4.11 flag_solid_dotted_line

```
bool glView::flag_solid_dotted_line
```

Flag for using dashed lines to draw lines, false - solid line, true - dotted line.

6.1.4.12 scale_change

```
float glView::scale_change
```

The value to scale the model.

6.1.4.13 scaleFactor_x

```
double glView::scaleFactor_x  [private]
```

The coefficient for moving the model along the x-axis.

6.1.4.14 scaleFactor_y

```
double glView::scaleFactor_y  [private]
```

The coefficient for moving the model along the y-axis.

6.1.4.15 scaleFactor_z

```
double glView::scaleFactor_z  [private]
```

The coefficient for moving the model along the z-axis.

6.1.4.16 value_background_color

```
int glView::value_background_color
```

The value for choose background color: 1 - black, 2 - white, 3 - red , 4 - blue, 5 - green.

6.1.4.17 value_line_color

```
int glView::value_line_color
```

The value for choose line color: 1 - white, 2 - black, 3 - red , 4 - blue, 5 - green.

6.1.4.18 value_projection

```
int glView::value_projection
```

The value for selection projection type: 0 - central projection, 1 - parallel projection.

6.1.4.19 value_vertex_color

```
int glView::value_vertex_color
```

The value for choose vertex color: 1 - black, 2

- white, 3 - red , 4 - blue, 5 - green.

6.1.4.20 value_vertex_type

```
int glView::value_vertex_type
```

The value for vertex type selection: 1 - no vertex, 2 - circle vertex , 3- square vertex.

6.1.4.21 x_move

```
float glView::x_move
```

The value to move the model along the x-axis.

6.1.4.22 x_rotate

```
float glView::x_rotate
```

The value to rotate the model along the x-axis.

6.1.4.23 y_move

```
float glView::y_move
```

The value to move the model along the y-axis.

6.1.4.24 y_rotate

```
float glView::y_rotate
```

The value to rotate the model along the y-axis.

6.1.4.25 z_move

```
float glView::z_move
```

The value to move the model along the z-axis.

6.1.4.26 z_rotate

```
float glView::z_rotate
```

The value to rotate the model along the z-axis.

The documentation for this class was generated from the following files:

- 3dviewer/glview.h
- 3dviewer/glview.cpp

6.2 info_figure Struct Reference

A structure containing information about a shape.

```
#include <parser.h>
```

Public Attributes

- int number_of_vertex
  The number of vertices.
- int number_of_faces_all
  The number of faces.
- float * matrix_of_vertex
- unsigned int * matrix_of_index
  The matrix of vertices.
- float x_max
  The maximum value on the X-axis.
- float x_min
  The minimum value on the X-axis.
- float y_max
  The maximum value on the Y-axis.
- float y_min
  The minimum value on the Y-axis.
- float z_max
  The maximum value on the Z-axis.
- float z_min
  The minimum value on the Z-axis.
- bool load_result
  The result of the download.
- float * matrix_of_vertex_origin
  The matrix of the original vertices.

6.2.1 Detailed Description

A structure containing information about a shape.

The "info_figure" structure contains information about the shape, including the number of vertices, the number of faces, the vertex and index matrices, as well as the minimum and maximum values for each of the axes and the matrix of the original vertices, as well as information about the success of opening the file.

6.2.2 Member Data Documentation

6.2.2.1 load_result

```
bool info_figure::load_result
```

The result of the download.

6.2.2.2 matrix_of_index

```
unsigned int* info_figure::matrix_of_index
```

The matrix of vertices.

6.2.2.3 matrix_of_vertex

```
float* info_figure::matrix_of_vertex
```

6.2.2.4 matrix_of_vertex_origin

```
float* info_figure::matrix_of_vertex_origin
```

The matrix of the original vertices.

6.2.2.5 number_of_faces_all

```
int info_figure::number_of_faces_all
```

The number of faces.

6.2.2.6 number_of_vertex

```
int info_figure::number_of_vertex
```

The number of vertices.

6.2.2.7 x_max

```
float info_figure::x_max
```

The maximum value on the X-axis.

6.2.2.8 x_min

```
float info_figure::x_min
```

The minimum value on the X-axis.

6.2.2.9 y_max

```
float info_figure::y_max
```

The maximum value on the Y-axis.

6.2.2.10 y_min

```
float info_figure::y_min
```

The minimum value on the Y-axis.

6.2.2.11 z_max

```
float info_figure::z_max
```

The maximum value on the Z-axis.

6.2.2.12 z_min

```
float info_figure::z_min
```

The minimum value on the Z-axis.

The documentation for this struct was generated from the following file:

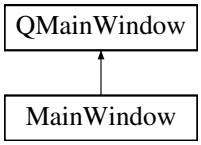
- 3dviewer/parser.h

6.3 MainWindow Class Reference

MainWindow class. It contains class member definitions such as constructor, destructor methods, slots, and signals. The class contains variables for saving and loading settings, signal mappers for matching actions to specific IDs, a timer, and image objects for creating animated GIFs. Also in this class, various slots are defined that are called during user interaction events with the interface, such as changing the values of spinboxes for moving, rotating and scaling objects, choosing the type of projection and line color, setting the background color, etc.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Public Slots

- void set_counts (int vertex_count, int line_count)

  Sets the values of the number of vertices and edges in the corresponding fields of the user interface.

Signals

- void openobj ()

  Signal to open .obj file.

Public Member Functions

- MainWindow (QWidget ∗parent=nullptr)

  Class constructor.
- ∼MainWindow ()

  Class destructor.
- void save_settings ()

  This feature saves application settings to a QSettings∗ settings. The code starts by saving the title of the main window, and then saves all other settings, which include settings for projection, line style, background color, line color, vertex color, vertex type, vertex size, and line size. The settings object is used to save the settings. This allows you to restore the previously saved settings the next time you start the application.
- void load_settings ()

  The function restores the previous settings of the application when it is launched, so that the user can continue working with the application exactly where he left off the last time.

Private Slots

- void on_actionopen_triggered ()

  This is a handler slot that is called when you click on the "Open" menu item in the main application window.
- void on_move_x_spinbox_valueChanged (double arg1)

  Changes the position of the object along the x-axis.
- void on_move_y_spinbox_valueChanged (double arg1)

  Changes the position of the object along the y-axis.
- void on_move_z_spinbox_valueChanged (double arg1)

  Changes the position of the object along the z-axis.
- void on_rotate_x_spinbox_valueChanged (double arg1)

  Rotate the object along the x-axis.
- void on_rotate_y_spinbox_valueChanged (double arg1)

  Rotate the object along the y-axis.
- void on_rotate_z_spinbox_valueChanged (double arg1)

  Rotate the object along the z-axis.
- void on_scaling_butt_valueChanged (double arg1)

  Changes the scale of the model.
- void on_solid_line_triggered ()

  Checks if the flag for drawing solid lines is set and updates the model.
- void on_dotted_line_triggered ()

  Checks if the flag for drawing dotted lines is set and updates the model.
- void on_parallel_projection_triggered ()

  Checks if the flag for drawing model in parallel projection is set and updates the model.
- void on_central_projection_triggered ()

  Checks if the flag for drawing model in central projection is set and updates the model.
- void action_background_triggered (int id)

  The function sets the flags for each background color and redraws the model with the selected color.
- void setupMenuActions_back_color ()

  Create QSignalMapper object and associates triggered() signals from background color selection menu items with it.
- void action_line_triggered (int id)

  The function sets the flags for each line color and redraws the model with the selected color.
- void setupMenuActions_line_color ()

  Create QSignalMapper object and associates triggered() signals from line color selection menu items with it.
- void setupMenuActions_vertex_type ()

  Create QSignalMapper object and associates triggered() signals from verteces type selection menu items with it.
- void action_vertex_triggered (int id)

  The function sets the flags for each verteces type and redraws the model with the selected type.
- void setupMenuActions_vertex_color ()

  Create QSignalMapper object and associates triggered() signals from verteces color selection menu items with it.
- void action_vertex_color_triggered (int id)

  The function sets the flags for each verteces color and redraws the model with the selected color.
- void on_action_bmp_triggered ()

  Is responsible for saving the image in BMP format.
- void on_action_jpeg_triggered ()

  Is responsible for saving the image in JPEG format.
- void on_start_screencast_clicked ()

  Is responsible for starting the screen recording in GIF format.
- void save_gif ()

  is responsible for save in GIF format.
- void on_custom_vertex_size_valueChanged (int value)

  Change verteces size.
- void on_custom_line_size_valueChanged (int value)

  Change line size.

Private Attributes

- Ui::MainWindow * ui
- QSettings * settings

  Object for storing application settings.
- QSignalMapper * signalMapper_back_color
- QSignalMapper * signalMapper_line_color
- QSignalMapper * signalMapper_vertex_type
- QSignalMapper * signalMapper_vertex_size
- QSignalMapper * signalMapper_vertex_color
- int time
- QString gif_Path

  Path to saved GIF file.
- QTimer * timer
- QGifImage * gif
- QImage image

  Object that is used to store the image.

6.3.1 Detailed Description

MainWindow class. It contains class member definitions such as constructor, destructor methods, slots, and signals. The class contains variables for saving and loading settings, signal mappers for matching actions to specific IDs, a timer, and image objects for creating animated GIFs. Also in this class, various slots are defined that are called during user interaction events with the interface, such as changing the values of spinboxes for moving, rotating and scaling objects, choosing the type of projection and line color, setting the background color, etc.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget * parent = nullptr )
```

Class constructor.

Parameters

parent	the reference to parent object.

```
7     : QMainWindow(parent), ui(new Ui::MainWindow) {
8   gif = NULL;
9   timer = NULL;
10   time = 0;
11   ui->setupUi(this);
12
13   settings = new QSettings("Channelle&Larraqan", "3d_viewer", this);
14   load_settings();
15
16   setupMenuActions_back_color();
17   setupMenuActions_line_color();
18   setupMenuActions_vertex_type();
19   setupMenuActions_vertex_color();
20
```

```
21   connect(this, &MainWindow::openobj, ui->widget, &glView::open_obj);
22   connect(ui->widget, &glView::set_counts, this, &MainWindow::set_counts);
23
24   connect(ui->solid_line, &QAction::triggered, this,
25           &MainWindow::on_solid_line_triggered);
26   connect(ui->dotted_line, &QAction::triggered, this,
27           &MainWindow::on_dotted_line_triggered);
28
29   connect(ui->parallel_projection, &QAction::triggered, this,
30           &MainWindow::on_parallel_projection_triggered);
31   connect(ui->central_projection, &QAction::triggered, this,
32           &MainWindow::on_central_projection_triggered);
33 }
```

6.3.2.2 ~MainWindow()

```
MainWindow::~MainWindow ( )
```

Class destructor.

```
321                          {
322   save_settings();
323   delete signalMapper_back_color;
324   delete signalMapper_line_color;
325   delete signalMapper_vertex_type;
326   delete signalMapper_vertex_color;
327   delete settings;
328   if (gif != NULL) {
329     delete gif;
330   }
331   if (timer != NULL) {
332     delete timer;
333   }
334   delete ui;
335 }
```

6.3.3 Member Function Documentation

6.3.3.1 action_background_triggered

```
void MainWindow::action_background_triggered (
             int id )  [private], [slot]
```

The function sets the flags for each background color and redraws the model with the selected color.

Parameters

id	value of background color.

```
85                                                  {
86   ui->background_black->setChecked(id == 1);
87   ui->background_white->setChecked(id == 2);
88   ui->background_red->setChecked(id == 3);
89   ui->background_blue->setChecked(id == 4);
90   ui->background_green->setChecked(id == 5);
91   ui->widget->value_background_color = id;
92   ui->widget->flag_background_color = true;
93   ui->widget->update_figure();
94 }
```

6.3.3.2 action_line_triggered

```
void MainWindow::action_line_triggered (
            int id )  [private], [slot]
```

The function sets the flags for each line color and redraws the model with the selected color.

Parameters

id	value of line color.

```
120                                          {
121   ui->line_white->setChecked(id == 1);
122   ui->line_black->setChecked(id == 2);
123   ui->line_red->setChecked(id == 3);
124   ui->line_blue->setChecked(id == 4);
125   ui->line_green->setChecked(id == 5);
126   ui->widget->value_line_color = id;
127   ui->widget->update_figure();
128 }
```

6.3.3.3 action_vertex_color_triggered

```
void MainWindow::action_vertex_color_triggered (
            int id )  [private], [slot]
```

The function sets the flags for each verteces color and redraws the model with the selected color.

Parameters

id	value of verteces color.

```
180                                                 {
181   ui->vertex_black->setChecked(id == 1);
182   ui->vertex_white->setChecked(id == 2);
183   ui->vertex_red->setChecked(id == 3);
184   ui->vertex_blue->setChecked(id == 4);
185   ui->vertex_green->setChecked(id == 5);
186   ui->widget->value_vertex_color = id;
187   ui->widget->update_figure();
188 }
```

6.3.3.4 action_vertex_triggered

```
void MainWindow::action_vertex_triggered (
            int id )  [private], [slot]
```

The function sets the flags for each verteces type and redraws the model with the selected type.

Parameters

id	value of verteces type.

```
148                                           {
149   ui->no_vertex->setChecked(id == 1);
150   ui->vertex_circle->setChecked(id == 2);
```

```
151   ui->vertex_square->setChecked(id == 3);
152   ui->widget->value_vertex_type = id;
153   ui->widget->update_figure();
154 }
```

6.3.3.5 load_settings()

```
void MainWindow::load_settings ( )
```

The function restores the previous settings of the application when it is launched, so that the user can continue working with the application exactly where he left off the last time.

```
380                                 {
381   setWindowTitle(settings->value("title", "3d_viewer").toString());
382   ui->central_projection->setChecked(
383       settings->value("central_projection", true).toBool());
384   ui->parallel_projection->setChecked(
385       settings->value("parallel_projection", false).toBool());
386   ui->widget->value_projection = settings->value("value_projection", 0).toInt();
387   ;
388   ui->widget->flag_projection = true;
389   ui->solid_line->setChecked(settings->value("solid_line", true).toBool());
390   ui->dotted_line->setChecked(settings->value("dotted_line", false).toBool());
391   ui->widget->flag_solid_dotted_line =
392       settings->value("flag_solid_dotted_line", false).toBool();
393   ui->background_black->setChecked(
394       settings->value("background_black", true).toBool());
395   ui->background_white->setChecked(
396       settings->value("background_white", false).toBool());
397   ui->background_red->setChecked(
398       settings->value("background_red", false).toBool());
399   ui->background_blue->setChecked(
400       settings->value("background_blue", false).toBool());
401   ui->background_green->setChecked(
402       settings->value("background_green", false).toBool());
403   ui->widget->value_background_color =
404       settings->value("value_background_color", 1).toInt();
405   ui->line_black->setChecked(settings->value("line_black", false).toBool());
406   ui->line_white->setChecked(settings->value("line_white", true).toBool());
407   ui->line_red->setChecked(settings->value("line_red", false).toBool());
408   ui->line_blue->setChecked(settings->value("line_blue", false).toBool());
409   ui->line_green->setChecked(settings->value("line_green", false).toBool());
410   ui->widget->value_line_color = settings->value("value_line_color", 1).toInt();
411   ui->vertex_black->setChecked(settings->value("vertex_black", false).toBool());
412   ui->vertex_white->setChecked(settings->value("vertex_white", true).toBool());
413   ui->vertex_red->setChecked(settings->value("vertex_red", false).toBool());
414   ui->vertex_blue->setChecked(settings->value("vertex_blue", false).toBool());
415   ui->vertex_green->setChecked(settings->value("vertex_green", false).toBool());
416   ui->widget->value_vertex_color =
417       settings->value("value_vertex_color", 1).toInt();
418   ui->no_vertex->setChecked(settings->value("no_vertex", true).toBool());
419   ui->vertex_circle->setChecked(
420       settings->value("vertex_circle", false).toBool());
421   ui->vertex_square->setChecked(
422       settings->value("vertex_square", false).toBool());
423   ui->widget->value_vertex_type =
424       settings->value("value_vertex_type", 1).toInt();
425   ui->widget->custom_vertex_size =
426       settings->value("custom_vertex_size", 1).toInt();
427   ui->custom_vertex_size->setValue(ui->widget->custom_vertex_size);
428   ui->widget->custom_line_size = settings->value("custom_line_size", 1).toInt();
429   ui->custom_line_size->setValue(ui->widget->custom_line_size);
430 }
```

6.3.3.6 on_action_bmp_triggered

```
void MainWindow::on_action_bmp_triggered ( )  [private], [slot]
```

Is responsible for saving the image in BMP format.

```
267                                     {
268   QString fileName = QFileDialog::getSaveFileName(this, tr("Save Image"), "",
```

```
269                                                     tr("BMP (*.bmp)"));
270   if (!fileName.isEmpty()) {
271     QString format = "BMP";
272     ui->widget->saveImage(fileName, format);
273   }
274 }
```

6.3.3.7 on_action_jpeg_triggered

```
void MainWindow::on_action_jpeg_triggered ( )  [private], [slot]
```

Is responsible for saving the image in JPEG format.

```
276                                             {
277   QString fileName = QFileDialog::getSaveFileName(this, tr("Save Image"), "",
278                                                     tr("JPEG (*.jpeg)"));
279   if (!fileName.isEmpty()) {
280     QString format = "JPEG";
281     ui->widget->saveImage(fileName, format);
282   }
283 }
```

6.3.3.8 on_actionopen_triggered

```
void MainWindow::on_actionopen_triggered ( )  [private], [slot]
```

This is a handler slot that is called when you click on the "Open" menu item in the main application window.

```
35                                          {
36   QString filePath = NULL;
37   filePath = QFileDialog::getOpenFileName(this, NULL, "/~", "OBJ (*.obj)");
38   if (!filePath.isNull()) {
39     ui->widget->FileName = filePath;
40     QFileInfo fileInfo(filePath);
41     QString file_Name = fileInfo.fileName();
42     ui->name_file->setText(file_Name);
43     ui->vertex_count->setText("0");
44     ui->line_count->setText("0");
45     ui->move_x_spinbox->setValue(0);
46     ui->move_y_spinbox->setValue(0);
47     ui->move_z_spinbox->setValue(0);
48     ui->rotate_x_spinbox->setValue(0);
49     ui->rotate_y_spinbox->setValue(0);
50     ui->rotate_z_spinbox->setValue(0);
51     ui->scaling_butt->setValue(1);
52     emit openobj();
53   }
54 }
```

6.3.3.9 on_central_projection_triggered

```
void MainWindow::on_central_projection_triggered ( )  [private], [slot]
```

Checks if the flag for drawing model in central projection is set and updates the model.

```
257                                                 {
258   if (!(ui->central_projection->isChecked())) {
259     ui->central_projection->setChecked(true);
260   }
261   ui->parallel_projection->setChecked(false);
262   ui->widget->value_projection = 1;
263   ui->widget->flag_projection = true;
264   ui->widget->update_figure();
265 }
```

6.3.3.10 on_custom_line_size_valueChanged

```
void MainWindow::on_custom_line_size_valueChanged (
            int value )  [private], [slot]
```

Change line size.

Parameters

value	line size.

```
316                                                     {
317   ui->widget->custom_line_size = value;
318   ui->widget->update_figure();
319 }
```

6.3.3.11 on_custom_vertex_size_valueChanged

```
void MainWindow::on_custom_vertex_size_valueChanged (
            int value )  [private], [slot]
```

Change verteces size.

Parameters

value	verteces size.

```
311                                                       {
312   ui->widget->custom_vertex_size = value;
313   ui->widget->update_figure();
314 }
```

6.3.3.12 on_dotted_line_triggered

```
void MainWindow::on_dotted_line_triggered ( )  [private], [slot]
```

Checks if the flag for drawing dotted lines is set and updates the model.

```
238                                     {
239   if (!(ui->dotted_line->isChecked())) {
240     ui->dotted_line->setChecked(true);
241   }
242   ui->solid_line->setChecked(false);
243   ui->widget->flag_solid_dotted_line = true;
244   ui->widget->update_figure();
245 }
```

6.3.3.13 on_move_x_spinbox_valueChanged

```
void MainWindow::on_move_x_spinbox_valueChanged (
            double arg1 )  [private], [slot]
```

Changes the position of the object along the x-axis.

Parameters

arg1	value to set the x-coordinate.

```
190                                                     {
191   ui->widget->x_move = arg;
192   ui->widget->update_figure();
193 }
```

6.3.3.14 on_move_y_spinbox_valueChanged

```
void MainWindow::on_move_y_spinbox_valueChanged (
            double arg1 )  [private], [slot]
```

Changes the position of the object along the y-axis.

Parameters

arg1	value to set the y-coordinate.

```
195                                                     {
196   ui->widget->y_move = arg;
197   ui->widget->update_figure();
198 }
```

6.3.3.15 on_move_z_spinbox_valueChanged

```
void MainWindow::on_move_z_spinbox_valueChanged (
            double arg1 )  [private], [slot]
```

Changes the position of the object along the z-axis.

Parameters

arg1	value to set the z-coordinate

```
200                                                     {
201   ui->widget->z_move = arg;
202   ui->widget->update_figure();
203 }
```

6.3.3.16 on_parallel_projection_triggered

```
void MainWindow::on_parallel_projection_triggered ( )  [private], [slot]
```

Checks if the flag for drawing model in parallel projection is set and updates the model.

```
247                                                     {
248   if (!(ui->parallel_projection->isChecked())) {
249     ui->parallel_projection->setChecked(true);
250   }
251   ui->central_projection->setChecked(false);
```

```
252   ui->widget->value_projection = 0;
253   ui->widget->flag_projection = true;
254   ui->widget->update_figure();
255 }
```

6.3.3.17 on_rotate_x_spinbox_valueChanged

```
void MainWindow::on_rotate_x_spinbox_valueChanged (
            double arg1 )  [private], [slot]
```

Rotate the object along the x-axis.

Parameters

arg1	value in degrees by which you want to rotate the model along the x-axis

```
209                                                     {
210   ui->widget->x_rotate = radians(-arg1);
211   ui->widget->update_figure();
212 }
```

6.3.3.18 on_rotate_y_spinbox_valueChanged

```
void MainWindow::on_rotate_y_spinbox_valueChanged (
            double arg1 )  [private], [slot]
```

Rotate the object along the y-axis.

Parameters

arg1	value in degrees by which you want to rotate the model along the y-axis

```
214                                                     {
215   ui->widget->y_rotate = radians(-arg1);
216   ui->widget->update_figure();
217 }
```

6.3.3.19 on_rotate_z_spinbox_valueChanged

```
void MainWindow::on_rotate_z_spinbox_valueChanged (
            double arg1 )  [private], [slot]
```

Rotate the object along the z-axis.

Parameters

arg1	value in degrees by which you want to rotate the model along the z-axis

```
219                                                     {
```

```
220   ui->widget->z_rotate = radians(-arg1);
221   ui->widget->update_figure();
222 }
```

6.3.3.20 on_scaling_butt_valueChanged

```
void MainWindow::on_scaling_butt_valueChanged (
             double arg1 )  [private], [slot]
```

Changes the scale of the model.

Parameters

arg1	value to change the scale of the model.

```
224                                                    {
225   ui->widget->scale_change = arg1;
226   ui->widget->update_figure();
227 }
```

6.3.3.21 on_solid_line_triggered

```
void MainWindow::on_solid_line_triggered ( )  [private], [slot]
```

Checks if the flag for drawing solid lines is set and updates the model.

```
229                                        {
230   if (!(ui->solid_line->isChecked())) {
231     ui->solid_line->setChecked(true);
232   }
233   ui->dotted_line->setChecked(false);
234   ui->widget->flag_solid_dotted_line = false;
235   ui->widget->update_figure();
236 }
```

6.3.3.22 on_start_screencast_clicked

```
void MainWindow::on_start_screencast_clicked ( )  [private], [slot]
```

Is responsible for starting the screen recording in GIF format.

```
285                                              {
286   if (time == 0) {
287     gif = new QGifImage;
288     gif->setDefaultDelay(100);
289     time = 0;
290     timer = new QTimer(this);
291     connect(timer, SIGNAL(timeout()), this, SLOT(save_gif()));
292     timer->start(100);
293   }
294 }
```

6.3.3.23 openobj

```
void MainWindow::openobj ( )  [signal]
```

Signal to open .obj file.

6.3.3.24 save_gif

```
void MainWindow::save_gif ( )  [private], [slot]
```

is responsible for save in GIF format.

```
296                             {
297   time++;
298   image = ui->widget->grab().toImage();
299   gif->addFrame(image, QPoint(0, 0));
300   if (time == 50) {
301     timer->stop();
302     time = 0;
303     disconnect(timer, SIGNAL(timeout()), this, SLOT(save_gif()));
304     gif_Path = QFileDialog::getSaveFileName(this, NULL, NULL, "GIF (*.gif)");
305     if (!gif_Path.isEmpty()) gif->save(gif_Path);
306     delete gif;
307     gif = NULL;
308   }
309 }
```

6.3.3.25 save_settings()

```
void MainWindow::save_settings ( )
```

This feature saves application settings to a QSettings∗ settings. The code starts by saving the title of the main window, and then saves all other settings, which include settings for projection, line style, background color, line color, vertex color, vertex type, vertex size, and line size. The settings object is used to save the settings. This allows you to restore the previously saved settings the next time you start the application.

```
336                               {
337   settings->setValue("title", windowTitle());
338
339   settings->setValue("central_projection", ui->central_projection->isChecked());
340   settings->setValue("parallel_projection",
341                      ui->parallel_projection->isChecked());
342   settings->setValue("value_projection", ui->widget->value_projection);
343
344   settings->setValue("solid_line", ui->solid_line->isChecked());
345   settings->setValue("dotted_line", ui->dotted_line->isChecked());
346   settings->setValue("flag_solid_dotted_line",
347                      ui->widget->flag_solid_dotted_line);
348
349   settings->setValue("background_black", ui->background_black->isChecked());
350   settings->setValue("background_white", ui->background_white->isChecked());
351   settings->setValue("background_red", ui->background_red->isChecked());
352   settings->setValue("background_blue", ui->background_blue->isChecked());
353   settings->setValue("background_green", ui->background_green->isChecked());
354   settings->setValue("value_background_color",
355                      ui->widget->value_background_color);
356
357   settings->setValue("line_black", ui->line_black->isChecked());
358   settings->setValue("line_white", ui->line_white->isChecked());
359   settings->setValue("line_red", ui->line_red->isChecked());
360   settings->setValue("line_blue", ui->line_blue->isChecked());
361   settings->setValue("line_green", ui->line_green->isChecked());
362   settings->setValue("value_line_color", ui->widget->value_line_color);
363
364   settings->setValue("vertex_black", ui->vertex_black->isChecked());
365   settings->setValue("vertex_white", ui->vertex_white->isChecked());
366   settings->setValue("vertex_red", ui->vertex_red->isChecked());
367   settings->setValue("vertex_blue", ui->vertex_blue->isChecked());
```

```
368   settings->setValue("vertex_green", ui->vertex_green->isChecked());
369   settings->setValue("value_vertex_color", ui->widget->value_vertex_color);
370
371   settings->setValue("no_vertex", ui->no_vertex->isChecked());
372   settings->setValue("vertex_circle", ui->vertex_circle->isChecked());
373   settings->setValue("vertex_square", ui->vertex_square->isChecked());
374   settings->setValue("value_vertex_type", ui->widget->value_vertex_type);
375
376   settings->setValue("custom_vertex_size", ui->widget->custom_vertex_size);
377   settings->setValue("custom_line_size", ui->widget->custom_line_size);
378 }
```

6.3.3.26 set_counts

```
void MainWindow::set_counts (
              int vertex_count,
              int line_count )  [slot]
```

Sets the values of the number of vertices and edges in the corresponding fields of the user interface.

Parameters

vertex_count	count of vertisices
line_count	count of lines

```
56                                                {
57   ui->vertex_count->setText(QString::number(vertex));
58   ui->line_count->setText(QString::number(lines));
59 }
```

6.3.3.27 setupMenuActions_back_color

```
void MainWindow::setupMenuActions_back_color ( )  [private], [slot]
```

Create QSignalMapper object and associates triggered() signals from background color selection menu items with it.

```
61                                                {
62   signalMapper_back_color = new QSignalMapper(this);
63
64   connect(ui->background_black, SIGNAL(triggered()), signalMapper_back_color,
65           SLOT(map()));
66   connect(ui->background_white, SIGNAL(triggered()), signalMapper_back_color,
67           SLOT(map()));
68   connect(ui->background_red, SIGNAL(triggered()), signalMapper_back_color,
69           SLOT(map()));
70   connect(ui->background_blue, SIGNAL(triggered()), signalMapper_back_color,
71           SLOT(map()));
72   connect(ui->background_green, SIGNAL(triggered()), signalMapper_back_color,
73           SLOT(map()));
74
75   signalMapper_back_color->setMapping(ui->background_black, 1);
76   signalMapper_back_color->setMapping(ui->background_white, 2);
77   signalMapper_back_color->setMapping(ui->background_red, 3);
78   signalMapper_back_color->setMapping(ui->background_blue, 4);
79   signalMapper_back_color->setMapping(ui->background_green, 5);
80
81   connect(signalMapper_back_color, SIGNAL(mapped(int)), this,
82           SLOT(action_background_triggered(int)));
83 }
```

6.3.3.28 setupMenuActions_line_color

```
void MainWindow::setupMenuActions_line_color ( )  [private], [slot]
```

Create QSignalMapper object and associates triggered() signals from line color selection menu items with it.

```
96                                          {
97   signalMapper_line_color = new QSignalMapper(this);
98
99   connect(ui->line_black, SIGNAL(triggered()), signalMapper_line_color,
100          SLOT(map()));
101  connect(ui->line_white, SIGNAL(triggered()), signalMapper_line_color,
102          SLOT(map()));
103  connect(ui->line_red, SIGNAL(triggered()), signalMapper_line_color,
104          SLOT(map()));
105  connect(ui->line_blue, SIGNAL(triggered()), signalMapper_line_color,
106          SLOT(map()));
107  connect(ui->line_green, SIGNAL(triggered()), signalMapper_line_color,
108          SLOT(map()));
109
110  signalMapper_line_color->setMapping(ui->line_black, 2);
111  signalMapper_line_color->setMapping(ui->line_white, 1);
112  signalMapper_line_color->setMapping(ui->line_red, 3);
113  signalMapper_line_color->setMapping(ui->line_blue, 4);
114  signalMapper_line_color->setMapping(ui->line_green, 5);
115
116  connect(signalMapper_line_color, SIGNAL(mapped(int)), this,
117          SLOT(action_line_triggered(int)));
118 }
```

6.3.3.29 setupMenuActions_vertex_color

```
void MainWindow::setupMenuActions_vertex_color ( )  [private], [slot]
```

Create QSignalMapper object and associates triggered() signals from verteces color selection menu items with it.

```
156                                           {
157   signalMapper_vertex_color = new QSignalMapper(this);
158
159   connect(ui->vertex_black, SIGNAL(triggered()), signalMapper_vertex_color,
160           SLOT(map()));
161   connect(ui->vertex_white, SIGNAL(triggered()), signalMapper_vertex_color,
162           SLOT(map()));
163   connect(ui->vertex_red, SIGNAL(triggered()), signalMapper_vertex_color,
164           SLOT(map()));
165   connect(ui->vertex_blue, SIGNAL(triggered()), signalMapper_vertex_color,
166           SLOT(map()));
167   connect(ui->vertex_green, SIGNAL(triggered()), signalMapper_vertex_color,
168           SLOT(map()));
169
170   signalMapper_vertex_color->setMapping(ui->vertex_black, 1);
171   signalMapper_vertex_color->setMapping(ui->vertex_white, 2);
172   signalMapper_vertex_color->setMapping(ui->vertex_red, 3);
173   signalMapper_vertex_color->setMapping(ui->vertex_blue, 4);
174   signalMapper_vertex_color->setMapping(ui->vertex_green, 5);
175
176   connect(signalMapper_vertex_color, SIGNAL(mapped(int)), this,
177           SLOT(action_vertex_color_triggered(int)));
178 }
```

6.3.3.30 setupMenuActions_vertex_type

```
void MainWindow::setupMenuActions_vertex_type ( )  [private], [slot]
```

Create QSignalMapper object and associates triggered() signals from verteces type selection menu items with it.

```
130                                           {
131   signalMapper_vertex_type = new QSignalMapper(this);
132
133   connect(ui->no_vertex, SIGNAL(triggered()), signalMapper_vertex_type,
```

```
134           SLOT(map()));
135   connect(ui->vertex_circle, SIGNAL(triggered()), signalMapper_vertex_type,
136           SLOT(map()));
137   connect(ui->vertex_square, SIGNAL(triggered()), signalMapper_vertex_type,
138           SLOT(map()));
139
140   signalMapper_vertex_type->setMapping(ui->no_vertex, 1);
141   signalMapper_vertex_type->setMapping(ui->vertex_circle, 2);
142   signalMapper_vertex_type->setMapping(ui->vertex_square, 3);
143
144   connect(signalMapper_vertex_type, SIGNAL(mapped(int)), this,
145           SLOT(action_vertex_triggered(int)));
146 }
```

6.3.4 Member Data Documentation

6.3.4.1 gif

```
QGifImage* MainWindow::gif  [private]
```

Pointer to an object of the QGifImage class, which is intended for creating, storing and displaying a GIF animation.

6.3.4.2 gif_Path

```
QString MainWindow::gif_Path  [private]
```

Path to saved GIF file.

6.3.4.3 image

```
QImage MainWindow::image  [private]
```

Object that is used to store the image.

6.3.4.4 settings

```
QSettings* MainWindow::settings  [private]
```

Object for storing application settings.

6.3.4.5 signalMapper_back_color

```
QSignalMapper* MainWindow::signalMapper_back_color  [private]
```

Pointer that is used to bind signals from the main window's background color picker widgets to a procedure that changes the background color.

6.3.4.6 signalMapper_line_color

```
QSignalMapper* MainWindow::signalMapper_line_color  [private]
```

Pointer that is used to bind signals from the main window's line color picker widgets to a procedure that changes the line color.

6.3.4.7 signalMapper_vertex_color

```
QSignalMapper* MainWindow::signalMapper_vertex_color  [private]
```

Pointer that is used to bind signals from the main window's vertices color picker widgets to a procedure that changes the vertices color.

6.3.4.8 signalMapper_vertex_size

```
QSignalMapper* MainWindow::signalMapper_vertex_size  [private]
```

Pointer that is used to bind signals from the main window's size of vertices to a procedure that changes the size of vertices.

6.3.4.9 signalMapper_vertex_type

```
QSignalMapper* MainWindow::signalMapper_vertex_type  [private]
```

Pointer that is used to bind signals from the main window's type of vertices to a procedure that changes the type of vertices.

6.3.4.10 time

```
int MainWindow::time  [private]
```

Variable to keep track of the number of frames added to the GIF animation.

6.3.4.11 timer

```
QTimer* MainWindow::timer  [private]
```

Value is used to generate a signal every 100 milliseconds (0.1 seconds) that calls the save_gif() slot. After 50 signal generations (5 seconds elapsed), the timer stops.

6.3.4.12 ui

```
Ui::MainWindow* MainWindow::ui  [private]
```

Pointer to a class generated from the user interface file mainwindow.ui using the Qt Designer tool.

The documentation for this class was generated from the following files:

- 3dviewer/mainwindow.h
- 3dviewer/mainwindow.cpp

Chapter 7

File Documentation

7.1 3dviewer/affine_transform.c File Reference

```
#include "affine_transform.h"
```

Functions

- void move_model (float *matrix_of_vertex, int axis, int number, float value)

  Shifts the model along the selected axis by the specified value.
- void rotation_by_ox (float *matrix_of_vertex, int number, float angle)

  Rotates the model around the OX axis by the specified angle.
- void rotation_by_oy (float *matrix_of_vertex, int number, float angle)

  Rotates the model around the OY axis by the specified angle.
- void rotation_by_oz (float *matrix_of_vertex, int number, float angle)

  Rotates the model around the OZ axis by the specified angle.
- void scale_model (float *matrix_of_vertex, int number, float value)

  Scales the model by the specified factor.

7.1.1 Function Documentation

7.1.1.1 move_model()

```
void move_model (
            float * matrix_of_vertex,
            int axis,
            int number,
            float value )
```

Shifts the model along the selected axis by the specified value.

Parameters

matrix_of_vertex	Matrix of model vertices
axis	Selected axis (0 - X, 1 - Y, 2 - Z)
number	Number of coordinates of model points
value	Shift value

```
3                                                                                {
4   for (int i = 0 + axis; i < number; i += 3) matrix_of_vertex[i] += value;
5 }
```

7.1.1.2 rotation_by_ox()

```
void rotation_by_ox (
              float * matrix_of_vertex,
              int number,
              float angle )
```

Rotates the model around the OX axis by the specified angle.

Parameters

matrix_of_vertex	Matrix of model vertices
number	Number of model vertices
angle	Angle of rotation in radians

```
7                                                                                {
8   for (int i = 0; i < number * 3; i += 3) {
9     float y = matrix_of_vertex[i + 1];
10     float z = matrix_of_vertex[i + 2];
11     matrix_of_vertex[i + 1] = y * cos(angle) - z * sin(angle);
12     matrix_of_vertex[i + 2] = y * sin(angle) + z * cos(angle);
13   }
14 }
```

7.1.1.3 rotation_by_oy()

```
void rotation_by_oy (
              float * matrix_of_vertex,
              int number,
              float angle )
```

Rotates the model around the OY axis by the specified angle.

Parameters

matrix_of_vertex	Matrix of model vertices
number	Number of model vertices
angle	Angle of rotation in radians

```
16                                                                               {
17   for (int i = 0; i < number * 3; i += 3) {
18     float x = matrix_of_vertex[i];
```

```
19     float z = matrix_of_vertex[i + 2];
20     matrix_of_vertex[i] = x * cos(angle) + z * sin(angle);
21     matrix_of_vertex[i + 2] = -x * sin(angle) + z * cos(angle);
22   }
23 }
```

7.1.1.4 rotation_by_oz()

```
void rotation_by_oz (
              float * matrix_of_vertex,
              int number,
              float angle )
```

Rotates the model around the OZ axis by the specified angle.

Parameters

matrix_of_vertex	Matrix of model vertices
number	Number of model vertices
angle	Angle of rotation in radians

```
25                                                                    {
26   for (int i = 0; i < number * 3; i += 3) {
27     float x = matrix_of_vertex[i];
28     float y = matrix_of_vertex[i + 1];
29     matrix_of_vertex[i] = x * cos(angle) - y * sin(angle);
30     matrix_of_vertex[i + 1] = x * sin(angle) + y * cos(angle);
31   }
32 }
```

7.1.1.5 scale_model()

```
void scale_model (
              float * matrix_of_vertex,
              int number,
              float value )
```

Scales the model by the specified factor.

Parameters

matrix_of_vertex	Matrix of model vertices
number	Number of coordinates of model points
value	Scaling factor

```
34                                                                  {
35   if (fabs(value) > 1e-8) {
36     for (int i = 0; i < number; i++) matrix_of_vertex[i] *= value;
37   }
38 }
```

7.2 3dviewer/affine_transform.h File Reference

```
#include <math.h>
#include <stdio.h>
```

Functions

- void move_model (float *matrix_of_vertex, int axis, int number, float value)

  Shifts the model along the selected axis by the specified value.
- void rotation_by_ox (float *matrix_of_vertex, int number, float angle)

  Rotates the model around the OX axis by the specified angle.
- void rotation_by_oy (float *matrix_of_vertex, int number, float angle)

  Rotates the model around the OY axis by the specified angle.
- void rotation_by_oz (float *matrix_of_vertex, int number, float angle)

  Rotates the model around the OZ axis by the specified angle.
- void scale_model (float *matrix_of_vertex, int number, float value)

  Scales the model by the specified factor.

7.2.1 Function Documentation

7.2.1.1 move_model()

```
void move_model (
            float * matrix_of_vertex,
            int axis,
            int number,
            float value )
```

Shifts the model along the selected axis by the specified value.

Parameters

matrix_of_vertex	Matrix of model vertices
axis	Selected axis (0 - X, 1 - Y, 2 - Z)
number	Number of coordinates of model points
value	Shift value

```
3                                                                                {
4   for (int i = 0 + axis; i < number; i += 3) matrix_of_vertex[i] += value;
5 }
```

7.2.1.2 rotation_by_ox()

```
void rotation_by_ox (
            float * matrix_of_vertex,
```

```
              int number,
              float angle )
```

Rotates the model around the OX axis by the specified angle.

Parameters

matrix_of_vertex	Matrix of model vertices
number	Number of model vertices
angle	Angle of rotation in radians

```
7                                                                    {
8    for (int i = 0; i < number * 3; i += 3) {
9      float y = matrix_of_vertex[i + 1];
10      float z = matrix_of_vertex[i + 2];
11      matrix_of_vertex[i + 1] = y * cos(angle) - z * sin(angle);
12      matrix_of_vertex[i + 2] = y * sin(angle) + z * cos(angle);
13    }
14 }
```

7.2.1.3 rotation_by_oy()

```
void rotation_by_oy (
              float * matrix_of_vertex,
              int number,
              float angle )
```

Rotates the model around the OY axis by the specified angle.

Parameters

matrix_of_vertex	Matrix of model vertices
number	Number of model vertices
angle	Angle of rotation in radians

```
16                                                                   {
17    for (int i = 0; i < number * 3; i += 3) {
18      float x = matrix_of_vertex[i];
19      float z = matrix_of_vertex[i + 2];
20      matrix_of_vertex[i] = x * cos(angle) + z * sin(angle);
21      matrix_of_vertex[i + 2] = -x * sin(angle) + z * cos(angle);
22    }
23 }
```

7.2.1.4 rotation_by_oz()

```
void rotation_by_oz (
              float * matrix_of_vertex,
              int number,
              float angle )
```

Rotates the model around the OZ axis by the specified angle.

Parameters

matrix_of_vertex	Matrix of model vertices
number	Number of model vertices
angle	Angle of rotation in radians

```
25                                                                    {
26   for (int i = 0; i < number * 3; i += 3) {
27     float x = matrix_of_vertex[i];
28     float y = matrix_of_vertex[i + 1];
29     matrix_of_vertex[i] = x * cos(angle) - y * sin(angle);
30     matrix_of_vertex[i + 1] = x * sin(angle) + y * cos(angle);
31   }
32 }
```

7.2.1.5 scale_model()

```
void scale_model (
             float * matrix_of_vertex,
             int number,
             float value )
```

Scales the model by the specified factor.

Parameters

matrix_of_vertex	Matrix of model vertices
number	Number of coordinates of model points
value	Scaling factor

```
34                                                                    {
35   if (fabs(value) > 1e-8) {
36     for (int i = 0; i < number; i++) matrix_of_vertex[i] *= value;
37   }
38 }
```

7.3 3dviewer/glview.cpp File Reference

```
#include "glview.h"
#include "parser.h"
```

7.4 3dviewer/glview.h File Reference

```
#include <QOpenGLFunctions>
#include <QOpenGLWidget>
#include <iostream>
#include <QWidget>
#include <QtOpenGL>
#include <QtWidgets/QWidget>
#include "parser.h"
#include "affine_transform.h"
```

Classes

- class glView

  Class for working with QWidget The main task of the class is to visualize the model.

7.5 3dviewer/mainwindow.cpp File Reference

```
#include "mainwindow.h"
#include "glview.h"
#include "ui_mainwindow.h"
```

Functions

- double radians (double degrees)

7.5.1 Function Documentation

7.5.1.1 radians()

```
double radians (
            double degrees )
205                                {
206   return degrees * 3.14159265358979323846 / 180.0;
207 }
```

7.6 3dviewer/mainwindow.h File Reference

```
#include <QColorDialog>
#include <QDebug>
#include <QFileDialog>
#include <QFileInfo>
#include <QImage>
#include <QMainWindow>
#include <QPoint>
#include <QSettings>
#include <QSignalMapper>
#include <QString>
#include <QTimer>
#include <cmath>
#include "qgifimage.h"
```

Classes

- class MainWindow

  MainWindow class. It contains class member definitions such as constructor, destructor methods, slots, and signals. The class contains variables for saving and loading settings, signal mappers for matching actions to specific IDs, a timer, and image objects for creating animated GIFs. Also in this class, various slots are defined that are called during user interaction events with the interface, such as changing the values of spinboxes for moving, rotating and scaling objects, choosing the type of projection and line color, setting the background color, etc.

Namespaces

- Ui

7.7 3dviewer/parser.c File Reference

```
#include "parser.h"
```

Functions

- int parser (char *name_of_file, struct info_figure *figure)

  Performs file parsing .obj with the geometric information of the shape and returns the result.
- int second_open (char *name_of_file, struct info_figure *figure)

  Reads information from a file .obj and writes it to the info_figure structure.
- void mem_alloc_matrices (struct info_figure *figure)

  Allocates memory for geometric information matrices of a shape.
- void connection_from_faces_to_edges (unsigned int temp_ind_in_str[], struct info_figure *figure, int *count_vertex_in_str, int *ind_now)

  Performs a connection between vertices and edges.
- void found_max_min (float x, float y, float z, struct info_figure *figure, int i)

  Finds the maximum and minimum values of the x, y, z coordinates and stores them in the info_figure structure.
- void free_matrix (struct info_figure *figure)

  Frees up the memory allocated for the matrices and resets the variables responsible for the size of the matrices.
- void open_file (char *filename, struct info_figure *figure)

  Runs a function that parses the shape data from the file passed as an argument and saves the error code to the structure.

7.7.1 Function Documentation

7.7.1.1 connection_from_faces_to_edges()

```
void connection_from_faces_to_edges (
            unsigned int temp_ind_in_str[],
            struct info_figure * figure,
            int * count_vertex_in_str,
            int * ind_now )
```

Performs a connection between vertices and edges.

Parameters

temp_ind_in_str	Temporary array of vertex indexes
figure	The structure that contains information about the shape
count_vertex_in_str	Number of vertices in a row
ind_now	The current index in the matrix_of_index array

```
104                                                                                 {
105   for (int j = 0; j < *count_vertex_in_str * 2; (*ind_now)++, j++) {
106     if (j == 0) {
107       figure->matrix_of_index[*ind_now] = temp_ind_in_str[j];
108       figure->matrix_of_index[*ind_now + 1] = temp_ind_in_str[j + 1];
109     } else if (j == *count_vertex_in_str - 1) {
110       figure->matrix_of_index[*ind_now + j] = temp_ind_in_str[j];
111       figure->matrix_of_index[*ind_now + j + 1] = temp_ind_in_str[j - j];
112     } else if (j > 0 && j < *count_vertex_in_str) {
113       figure->matrix_of_index[*ind_now + j] = temp_ind_in_str[j];
114       figure->matrix_of_index[*ind_now + j + 1] = temp_ind_in_str[j + 1];
115     }
116   }
117 }
```

7.7.1.2 found_max_min()

```
void found_max_min (
              float x,
              float y,
              float z,
              struct info_figure * figure,
              int i )
```

Finds the maximum and minimum values of the x, y, z coordinates and stores them in the info_figure structure.

Parameters

X	coordinate of the current vertex
Y	coordinate of the current vertex
Z	coordinate of the current vertex
figure	A pointer to the info_figure structure in which the maximum and minimum values will be stored
i	Index of the current vertex

```
120                          {
121   if (x > figure->x_max || i == 0) figure->x_max = x;
122   if (x < figure->x_min || i == 0) figure->x_min = x;
123   if (y > figure->y_max || i == 0) figure->y_max = y;
124   if (y < figure->y_min || i == 0) figure->y_min = y;
125   if (z > figure->z_max || i == 0) figure->z_max = z;
126   if (z < figure->z_min || i == 0) figure->z_min = z;
127 }
```

7.7.1.3 free_matrix()

```
void free_matrix (
              struct info_figure * figure )
```

Frees up the memory allocated for the matrices and resets the variables responsible for the size of the matrices.

Parameters

figureA	pointer to the info_figure structure whose memory needs to be freed

```
129                                             {
130   figure->number_of_faces_all = 0;
131   figure->number_of_vertex = 0;
132   free(figure->matrix_of_index);
133   free(figure->matrix_of_vertex);
134   free(figure->matrix_of_vertex_origin);
135 }
```

7.7.1.4 mem_alloc_matrices()

```
void mem_alloc_matrices (
            struct info_figure * figure )
```

Allocates memory for geometric information matrices of a shape.

Parameters

figure	The structure for which memory is allocated

```
93                                                  {
94   figure->matrix_of_vertex =
95       (float *)calloc(figure->number_of_vertex * 3, sizeof(float));
96   figure->matrix_of_vertex_origin =
97       (float *)calloc(figure->number_of_vertex * 3, sizeof(float));
98   figure->matrix_of_index = (unsigned int *)calloc(
99       figure->number_of_faces_all * 2, sizeof(unsigned int));
100 }
```

7.7.1.5 open_file()

```
void open_file (
            char * filename,
            struct info_figure * figure )
```

Runs a function that parses the shape data from the file passed as an argument and saves the error code to the structure.

Parameters

filename	Name of the file with data about the shape
figure	Pointer to the info_figure structure to which the read data will be written

```
137                                                 {
138   figure->load_result = parser(filename, figure);
139 }
```

7.7.1.6 parser()

```
int parser (
```

```
            char * name_of_file,
            struct info_figure * figure )
```

Performs file parsing .obj with the geometric information of the shape and returns the result.

Parameters

name_of_file	The name of the file containing geometric information
figure	The structure in which the geometric information of the shape is recorded

Returns

0 - failed to open the file, 1 - successfully read information from the file

```
3                                                                   {
4   int res = -1;
5   int flag_vertex = 0, flag_faces = 0;
6   FILE *file = NULL;
7   size_t len = 0;
8   ssize_t read = 0;
9   figure->number_of_faces_all = 0;
10   figure->number_of_vertex = 0;
11   file = fopen(name_of_file, "r");
12   if (file != NULL) {
13     char *lbuf = NULL;
14     while ((read = getline(&lbuf, &len, file)) != -1) {
15       if (lbuf[0] == 'v' && lbuf[1] == ' ') {
16         figure->number_of_vertex++;
17         flag_vertex = 1;
18
19       } else if (lbuf[0] == 'f' && lbuf[1] == ' ') {
20         for (int i = 1; lbuf[i] != '\n'; i++) {
21           if (lbuf[i] == ' ') {
22             figure->number_of_faces_all++;
23           }
24         }
25         flag_faces = 1;
26       }
27     }
28     fclose(file);
29     if (lbuf) free(lbuf);
30     res = second_open(name_of_file, figure);
31   } else {
32     res = 0;
33   }
34   if (res != 0 && flag_vertex > 0 && flag_faces > 0)
35     res = 1;
36   else
37     res = 0;
38   return res;
39 }
```

7.7.1.7 second_open()

```
int second_open (
            char * name_of_file,
            struct info_figure * figure )
```

Reads information from a file .obj and writes it to the info_figure structure.

Parameters

name_of_file	The name of the file containing geometric information
figure	The structure in which the geometric information of the shape is recorded

Returns

0 - failed to read information from the file, 1 - successfully read information from the file

```
41                                                                  {
42    int res = 0, flag_vertex = 0, flag_faces = 0;
43    FILE *file = NULL;
44    size_t len = 0;
45    ssize_t read = 0;
46    mem_alloc_matrices(figure);
47    file = fopen(name_of_file, "r");
48    int i = 0, count = 0, ind_now = 0, count_vertex_in_str = 0;
49    setlocale(LC_NUMERIC, "en_US.UTF-8");
50    char digits_str[64] = "\0", *lbuf = NULL;
51    unsigned int temp_ind_in_str[256];
52    for (; (read = getline(&lbuf, &len, file)) != -1; count++) {
53      if ((int)read > 1 && lbuf[0] == 'v' && lbuf[1] == ' ') {
54        sscanf(lbuf, "v %f %f %f", &figure->matrix_of_vertex_origin[i],
55               &figure->matrix_of_vertex_origin[i + 1],
56               &figure->matrix_of_vertex_origin[i + 2]);
57        found_max_min(figure->matrix_of_vertex_origin[i],
58                      figure->matrix_of_vertex_origin[i + 1],
59                      figure->matrix_of_vertex_origin[i + 2], figure, i);
60        i += 3;
61        flag_vertex = 1;
62      } else if (lbuf[0] == 'f' && lbuf[1] == ' ') {
63        count_vertex_in_str = 0;
64        for (int j = 1; j < (int)(read - 1); j++) {
65          if (lbuf[j] == ' ') {
66            j++;
67            memset(digits_str, 0, sizeof(digits_str));
68            for (int k = 0; lbuf[j] != '\n' && lbuf[j] != '/' && lbuf[j] != ' ';
69                 j++) {
70              if (lbuf[j] != '-') {
71                digits_str[k] = lbuf[j];
72                k++;
73              }
74            }
75            unsigned int temp = (unsigned int)(atoi(digits_str));
76            if (temp > 0) {
77              temp_ind_in_str[count_vertex_in_str] = temp - 1;
78              count_vertex_in_str++;
79            }
80          }
81        }
82        connection_from_faces_to_edges(temp_ind_in_str, figure,
83                                       &count_vertex_in_str, &ind_now);
84        flag_faces = 1;
85      }
86    }
87    fclose(file);
88    if (lbuf) free(lbuf);
89    if (flag_vertex > 0 && flag_faces > 0) res = 1;
90    return res;
91  }
```

7.8 3dviewer/parser.h File Reference

```
#include <getopt.h>
#include <locale.h>
#include <regex.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Classes

- struct info_figure

  A structure containing information about a shape.

Functions

- int parser (char *name_of_file, struct info_figure *figure)

  Performs file parsing .obj with the geometric information of the shape and returns the result.
- int second_open (char *name_of_file, struct info_figure *figure)

  Reads information from a file .obj and writes it to the info_figure structure.
- void connection_from_faces_to_edges (unsigned int temp_ind_in_str[ ], struct info_figure *figure, int *count_vertex_in_str, int *ind_now)

  Performs a connection between vertices and edges.
- void mem_alloc_matrices (struct info_figure *figure)

  Allocates memory for geometric information matrices of a shape.
- void found_max_min (float x, float y, float z, struct info_figure *figure, int i)

  Finds the maximum and minimum values of the x, y, z coordinates and stores them in the info_figure structure.
- void free_matrix (struct info_figure *figure)

  Frees up the memory allocated for the matrices and resets the variables responsible for the size of the matrices.
- void open_file (char *filename, struct info_figure *figure)

  Runs a function that parses the shape data from the file passed as an argument and saves the error code to the structure.

7.8.1 Function Documentation

7.8.1.1 connection_from_faces_to_edges()

```
void connection_from_faces_to_edges (
            unsigned int temp_ind_in_str[],
            struct info_figure * figure,
            int * count_vertex_in_str,
            int * ind_now )
```

Performs a connection between vertices and edges.

Parameters

temp_ind_in_str	Temporary array of vertex indexes
figure	The structure that contains information about the shape
count_vertex_in_str	Number of vertices in a row
ind_now	The current index in the matrix_of_index array

```
104                                                                    {
105   for (int j = 0; j < *count_vertex_in_str * 2; (*ind_now)++, j++) {
106     if (j == 0) {
107       figure->matrix_of_index[*ind_now] = temp_ind_in_str[j];
108       figure->matrix_of_index[*ind_now + 1] = temp_ind_in_str[j + 1];
109     } else if (j == *count_vertex_in_str - 1) {
110       figure->matrix_of_index[*ind_now + j] = temp_ind_in_str[j];
111       figure->matrix_of_index[*ind_now + j + 1] = temp_ind_in_str[j - j];
112     } else if (j > 0 && j < *count_vertex_in_str) {
113       figure->matrix_of_index[*ind_now + j] = temp_ind_in_str[j];
114       figure->matrix_of_index[*ind_now + j + 1] = temp_ind_in_str[j + 1];
115     }
116   }
117 }
```

7.8.1.2 found_max_min()

```
void found_max_min (
            float x,
            float y,
            float z,
            struct info_figure * figure,
            int i )
```

Finds the maximum and minimum values of the x, y, z coordinates and stores them in the info_figure structure.

Parameters

X	coordinate of the current vertex
Y	coordinate of the current vertex
Z	coordinate of the current vertex
figure	A pointer to the info_figure structure in which the maximum and minimum values will be stored
i	Index of the current vertex

```
120                         {
121   if (x > figure->x_max || i == 0) figure->x_max = x;
122   if (x < figure->x_min || i == 0) figure->x_min = x;
123   if (y > figure->y_max || i == 0) figure->y_max = y;
124   if (y < figure->y_min || i == 0) figure->y_min = y;
125   if (z > figure->z_max || i == 0) figure->z_max = z;
126   if (z < figure->z_min || i == 0) figure->z_min = z;
127 }
```

7.8.1.3 free_matrix()

```
void free_matrix (
            struct info_figure * figure )
```

Frees up the memory allocated for the matrices and resets the variables responsible for the size of the matrices.

Parameters

figureA	pointer to the info_figure structure whose memory needs to be freed

```
129                                         {
130   figure->number_of_faces_all = 0;
131   figure->number_of_vertex = 0;
132   free(figure->matrix_of_index);
133   free(figure->matrix_of_vertex);
134   free(figure->matrix_of_vertex_origin);
135 }
```

7.8.1.4 mem_alloc_matrices()

```
void mem_alloc_matrices (
            struct info_figure * figure )
```

Allocates memory for geometric information matrices of a shape.

Parameters

figure	The structure for which memory is allocated

```
93                                                  {
94   figure->matrix_of_vertex =
95       (float *)calloc(figure->number_of_vertex * 3, sizeof(float));
96   figure->matrix_of_vertex_origin =
97       (float *)calloc(figure->number_of_vertex * 3, sizeof(float));
98   figure->matrix_of_index = (unsigned int *)calloc(
99       figure->number_of_faces_all * 2, sizeof(unsigned int));
100 }
```

7.8.1.5 open_file()

```
void open_file (
              char * filename,
              struct info_figure * figure )
```

Runs a function that parses the shape data from the file passed as an argument and saves the error code to the structure.

Parameters

filename	Name of the file with data about the shape
figure	Pointer to the info_figure structure to which the read data will be written

```
137                                                     {
138   figure->load_result = parser(filename, figure);
139 }
```

7.8.1.6 parser()

```
int parser (
              char * name_of_file,
              struct info_figure * figure )
```

Performs file parsing .obj with the geometric information of the shape and returns the result.

Parameters

name_of_file	The name of the file containing geometric information
figure	The structure in which the geometric information of the shape is recorded

Returns

0 - failed to open the file, 1 - successfully read information from the file

```
3                                                   {
4   int res = -1;
5   int flag_vertex = 0, flag_faces = 0;
6   FILE *file = NULL;
7   size_t len = 0;
8   ssize_t read = 0;
```

```
9   figure->number_of_faces_all = 0;
10   figure->number_of_vertex = 0;
11   file = fopen(name_of_file, "r");
12   if (file != NULL) {
13     char *lbuf = NULL;
14     while ((read = getline(&lbuf, &len, file)) != -1) {
15       if (lbuf[0] == 'v' && lbuf[1] == ' ') {
16         figure->number_of_vertex++;
17         flag_vertex = 1;
18
19       } else if (lbuf[0] == 'f' && lbuf[1] == ' ') {
20         for (int i = 1; lbuf[i] != '\n'; i++) {
21           if (lbuf[i] == ' ') {
22             figure->number_of_faces_all++;
23           }
24         }
25         flag_faces = 1;
26       }
27     }
28     fclose(file);
29     if (lbuf) free(lbuf);
30     res = second_open(name_of_file, figure);
31   } else {
32     res = 0;
33   }
34   if (res != 0 && flag_vertex > 0 && flag_faces > 0)
35     res = 1;
36   else
37     res = 0;
38   return res;
39 }
```

7.8.1.7 second_open()

```
int second_open (
              char * name_of_file,
              struct info_figure * figure )
```

Reads information from a file .obj and writes it to the info_figure structure.

Parameters

name_of_file	The name of the file containing geometric information
figure	The structure in which the geometric information of the shape is recorded

Returns

0 - failed to read information from the file, 1 - successfully read information from the file

```
41                                                                {
42   int res = 0, flag_vertex = 0, flag_faces = 0;
43   FILE *file = NULL;
44   size_t len = 0;
45   ssize_t read = 0;
46   mem_alloc_matrices(figure);
47   file = fopen(name_of_file, "r");
48   int i = 0, count = 0, ind_now = 0, count_vertex_in_str = 0;
49   setlocale(LC_NUMERIC, "en_US.UTF-8");
50   char digits_str[64] = "\0", *lbuf = NULL;
51   unsigned int temp_ind_in_str[256];
52   for (; (read = getline(&lbuf, &len, file)) != -1; count++) {
53     if ((int)read > 1 && lbuf[0] == 'v' && lbuf[1] == ' ') {
54       sscanf(lbuf, "v %f %f %f", &figure->matrix_of_vertex_origin[i],
55              &figure->matrix_of_vertex_origin[i + 1],
56              &figure->matrix_of_vertex_origin[i + 2]);
57       found_max_min(figure->matrix_of_vertex_origin[i],
58                     figure->matrix_of_vertex_origin[i + 1],
59                     figure->matrix_of_vertex_origin[i + 2], figure, i);
60       i += 3;
61       flag_vertex = 1;
```

```
62      } else if (lbuf[0] == 'f' && lbuf[1] == ' ') {
63        count_vertex_in_str = 0;
64        for (int j = 1; j < (int)(read - 1); j++) {
65          if (lbuf[j] == ' ') {
66            j++;
67            memset(digits_str, 0, sizeof(digits_str));
68            for (int k = 0; lbuf[j] != '\n' && lbuf[j] != '/' && lbuf[j] != ' ';
69                 j++) {
70              if (lbuf[j] != '-') {
71                digits_str[k] = lbuf[j];
72                k++;
73              }
74            }
75            unsigned int temp = (unsigned int)(atoi(digits_str));
76            if (temp > 0) {
77              temp_ind_in_str[count_vertex_in_str] = temp - 1;
78              count_vertex_in_str++;
79            }
80          }
81        }
82        connection_from_faces_to_edges(temp_ind_in_str, figure,
83                                       &count_vertex_in_str, &ind_now);
84        flag_faces = 1;
85      }
86    }
87    fclose(file);
88    if (lbuf) free(lbuf);
89    if (flag_vertex > 0 && flag_faces > 0) res = 1;
90    return res;
91  }
```

Index