# svg-reader

0.1

Generated by Doxygen 1.9.1

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

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

# Chapter 2

# Class Index

## 2.1 Class List

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

# Chapter 3

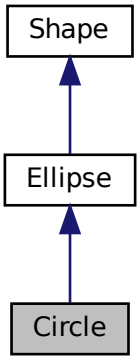
# Class Documentation

## 3.1 Circle Class Reference

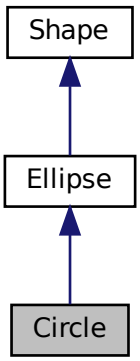
Represents a circle in 2D space.

`#include <Circle.hpp>`

Inheritance diagram for Circle:



Collaboration diagram for Circle:




## Public Member Functions

- Circle (float radius, const sf::Vector2f &center, sf::Color fill, sf::Color stroke, float stroke_thickness)
  *Constructs a Circle object.*

## Additional Inherited Members

### 3.1.1 Detailed Description

Represents a circle in 2D space.

The Circle class is derived from the Shape class and defines a circle with a specified radius, center, fill color, stroke color, and stroke thickness.

Definition at line 13 of file Circle.hpp.

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 Circle()

```
Circle::Circle (
            float radius,
            const sf::Vector2f & center,
            sf::Color fill,
            sf::Color stroke,
            float stroke_thickness )
```

Constructs a Circle object.

**Parameters**

| | |
|---|---|
| *radius* | The radius of the circle. |
| *center* | The center of the circle. |
| *fill* | Fill color of the circle. |
| *stroke* | Outline color of the circle. |
| *stroke_thickness* | Thickness of the circle outline. |

Definition at line 3 of file Circle.cpp.

```
5     : Ellipse(sf::Vector2f(radius, radius), center, fill, stroke,
6               stroke_thickness) {}
```

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

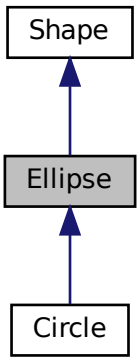
- src/graphics/Circle.hpp
- src/graphics/Circle.cpp

## 3.2 Ellipse Class Reference

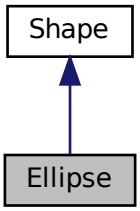
Represents an ellipse in 2D space.

`#include <Ellipse.hpp>`

Inheritance diagram for Ellipse:



Collaboration diagram for Ellipse:




## Public Member Functions

- Ellipse (const sf::Vector2f &radius, const sf::Vector2f &center, sf::Color fill, sf::Color stroke, float stroke_↩ thickness)

  *Constructs an Ellipse object.*
- virtual std::size_t getPointCount () const

  *Gets the total number of points representing the ellipse.*
- virtual sf::Vector2f getPoint (std::size_t index) const override

  *Gets the position of a point on the ellipse.*

## Protected Attributes

- const int SCALE

  *Scale factor for determining the number of points.*
- sf::Vector2f radius

  *Radii of the ellipse in the x and y directions.*

## Additional Inherited Members

### 3.2.1 Detailed Description

Represents an ellipse in 2D space.

The Ellipse class is derived from the Circle class and defines an ellipse with a variable radius in the x and y directions.

Definition at line 12 of file Ellipse.hpp.

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 Ellipse()

```
Ellipse::Ellipse (
            const sf::Vector2f & radius,
            const sf::Vector2f & center,
            sf::Color fill,
            sf::Color stroke,
            float stroke_thickness )
```

Constructs an Ellipse object.

**Parameters**

| | |
|---|---|
| *radius* | The radii of the ellipse in the x and y directions. |
| *center* | The center of the ellipse. |
| *fill* | Fill color of the ellipse. |
| *stroke* | Outline color of the ellipse. |
| *stroke_thickness* | Thickness of the ellipse outline. |

Definition at line 5 of file Ellipse.cpp.

```
7     : radius(radius) {
8     setPosition(center);
9     setFillColor(fill);
10     setOutlineColor(stroke);
11     setOutlineThickness(stroke_thickness);
12     update();
13 }
```

### 3.2.3 Member Function Documentation

#### 3.2.3.1 getPoint()

```
sf::Vector2f Ellipse::getPoint (
            std::size_t index ) const  [override], [virtual]
```

Gets the position of a point on the ellipse.

**Parameters**

| | |
|---|---|
| *index* | The index of the point. |

**Returns**

The position of the specified point on the ellipse.

Implements Shape.

Definition at line 17 of file Ellipse.cpp.

```
17                                          {
18     static const float pi = acos(-1);
```

```
19
20      float angle = index * 2 * pi / getPointCount() - pi / 2;
21      float x = std::cos(angle) * radius.x;
22      float y = std::sin(angle) * radius.y;
23
24      return sf::Vector2f(radius.x + x, radius.y + y);
25  }
```

#### 3.2.3.2 getPointCount()

```
std::size_t Ellipse::getPointCount ( ) const  [virtual]
```

Gets the total number of points representing the ellipse.

In this case, it returns a large number (SCALE) to approximate a smooth ellipse.

**Returns**

The number of points representing the ellipse.

Implements Shape.

Definition at line 15 of file Ellipse.cpp.

```
15 { return SCALE; }
```

### 3.2.4 Member Data Documentation

#### 3.2.4.1 SCALE

```
const int Ellipse::SCALE  [protected]
```

**Initial value:**

```
=
        100000
```

Scale factor for determining the number of points.

Definition at line 14 of file Ellipse.hpp.

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

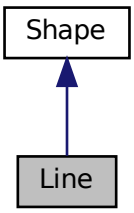
- src/graphics/Ellipse.hpp
- src/graphics/Ellipse.cpp

## 3.3 Line Class Reference

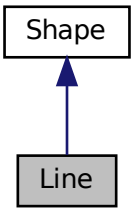
Represents a line in 2D space.

`#include <Line.hpp>`

Inheritance diagram for Line:




Collaboration diagram for Line:




### Public Member Functions

- Line (const sf::Vector2f &point1, const sf::Vector2f &point2, sf::Color stroke=sf::Color::White, float stroke_↩ width=1.f)

  *Constructs a Line object.*
- void setThickness (float thickness)

  *Sets the thickness of the line.*
- float getLength () const

  *Calculates and returns the length of the line.*
- virtual std::size_t getPointCount () const

  *Gets the total number of points representing the line.*
- virtual sf::Vector2f getPoint (std::size_t index) const

  *Gets the position of a point on the line.*

## Private Attributes

- sf::Vector2f direction

  *Direction of the line.*
- float thickness

  *Thickness of the line.*

## Additional Inherited Members

### 3.3.1 Detailed Description

Represents a line in 2D space.

The Line class is derived from the Shape class and defines a line segment with a specified direction and thickness.

Definition at line 14 of file Line.hpp.

### 3.3.2 Constructor & Destructor Documentation

#### 3.3.2.1 Line()

```
Line::Line (
            const sf::Vector2f & point1,
            const sf::Vector2f & point2,
            sf::Color stroke = sf::Color::White,
            float stroke_width = 1.f )
```

Constructs a Line object.

**Parameters**

| | |
|---|---|
| *point1* | The starting point of the line. |
| *point2* | The ending point of the line. |
| *stroke* | The color of the line (default is sf::Color::White). |
| *stroke_width* | The thickness of the line (default is 1.0). |

Definition at line 3 of file Line.cpp.

```
5     : direction(point2 - point1), thickness(stroke_width) {
6     setPosition(point1);
7     setThickness(stroke_width);
8     setFillColor(stroke);
9     update();
10 }
```

### 3.3.3 Member Function Documentation

#### 3.3.3.1 getLength()

```
float Line::getLength ( ) const
```

Calculates and returns the length of the line.

**Returns**

The length of the line.

Definition at line 14 of file Line.cpp.

```
14                              {
15     return std::sqrt(direction.x * direction.x + direction.y * direction.y);
16 }
```

#### 3.3.3.2 getPoint()

```
sf::Vector2f Line::getPoint (
             std::size_t index ) const  [virtual]
```

Gets the position of a point on the line.

**Parameters**

| | |
|---|---|
| *index* | The index of the point (0 for the starting point, 1 for the end point, 2 for the first additional point, 3 for the second additional point). |

**Returns**

The position of the specified point on the line.

Implements Shape.

Definition at line 20 of file Line.cpp.

```
20                                              {
21     sf::Vector2f unitDirection = direction / getLength();
22     sf::Vector2f unitPerpendicular(-unitDirection.y, unitDirection.x);
23
24     sf::Vector2f offset = (thickness / 2.f) * unitPerpendicular;
25
26     switch (index) {
27         default:
28         case 0:
29             return offset;
30         case 1:
31             return (direction + offset);
32         case 2:
33             return (direction - offset);
34         case 3:
35             return (-offset);
36     }
37 }
```

#### 3.3.3.3 getPointCount()

```
std::size_t Line::getPointCount ( ) const  [virtual]
```

Gets the total number of points representing the line.

In this case, it always returns 4 since a line is represented by 4 points (start, end, and two additional points for thickness).

**Returns**

The number of points representing the line.

Implements Shape.

Definition at line 18 of file Line.cpp.

```
18 { return 4; }
```

#### 3.3.3.4 setThickness()

```
void Line::setThickness (
            float thickness )
```

Sets the thickness of the line.

**Parameters**

| | |
|---|---|
| *thickness* | The new thickness of the line. |

Definition at line 12 of file Line.cpp.

```
12 { thickness = thickness; }
```

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

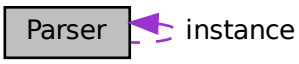
- src/graphics/Line.hpp
- src/graphics/Line.cpp

## 3.4 Parser Class Reference

Represents a parser for SVG files.

```
#include <Parser.hpp>
```

Collaboration diagram for Parser:




## Public Member Functions

- Parser (const Parser &)=delete
  *Deleted assignment operator to prevent copying of Parser instances.*
- std::string getAttribute (pugi::xml_node node, std::string name)
  *Get the Attribute object which is parsed from the XML file.*
- sf::Color parseColor (pugi::xml_node node, std::string name)
  *Parse the color from the XML file.*
- std::vector< sf::Vector2f > parsePoints (pugi::xml_node node)
  *Parse the points from the XML file.*
- void parseSVG ()
  *Parse the XML file.*
- void renderSVG (sf::RenderWindow &window)
  *Render the shapes on the window.*
- ~Parser ()
  *Destructor of Parser.*

## Static Public Member Functions

- static Parser * getInstance (const std::string &file_name)
  *Gets the singleton instance of Parser.*

## Private Member Functions

- Parser (const std::string &file_name)
  *Construct a new Parser object.*

## Private Attributes

- pugi::xml_node svg
  *The node of the SVG.*
- std::vector< Shape * > shapes
  *The vector of the shapes.*

## Static Private Attributes

- static Parser * instance = nullptr
  *The instance of the Parser.*

### 3.4.1 Detailed Description

Represents a parser for SVG files.

The parser class is responsible for parsing XML attributes (SVG files) and passing this information to create shapes and render them.

Definition at line 13 of file Parser.hpp.

### 3.4.2 Constructor & Destructor Documentation

#### 3.4.2.1 Parser() [1/2]

```
Parser::Parser (
            const Parser &  )  [delete]
```

Deleted assignment operator to prevent copying of Parser instances.

**Returns**

The Parser instance.

**Note**

This function is deleted because Parser is a singleton class.

#### 3.4.2.2 Parser() [2/2]

```
Parser::Parser (
            const std::string & file_name )  [private]
```

Construct a new Parser object.

**Parameters**

| | |
|---|---|
| *file_name* | The name of the file to be parsed. |

Definition at line 17 of file Parser.cpp.

```
17                                      {
18     pugi::xml_document doc;
19     pugi::xml_parse_result result = doc.load_file(file_name.c_str());
20     if (!result) EXIT_FAILURE;
21     svg = doc.child("svg");
22 }
```

### 3.4.3 Member Function Documentation

#### 3.4.3.1 getAttribute()

```
std::string Parser::getAttribute (
            pugi::xml_node node,
            std::string name )
```

Get the Attribute object which is parsed from the XML file.

**Parameters**

| | |
|---|---|
| *node* | The node of the XML file (pugi::xml_node is a typedef of pugixml) |
| *name* | The name of the attribute. |

**Returns**

The attribute which is parsed from the XML file.

**Note**

This function is private because it is only used by the Parser class.

Definition at line 24 of file Parser.cpp.

```
24                                                                  {
25     pugi::xml_attribute attr = node.attribute(name.c_str());
26     if (!attr) {
27         if (name == "fill")
28             return "black";
29         else if (name == "stroke")
30             return "none";
31         else if (name == "stroke-width" || name == "stroke-opacity" ||
32                  name == "fill-opacity" || name == "opacity")
33             return "1";
34         else if (name == "r" || name == "cx" || name == "cy" || name == "x" ||
35                  name == "y" || name == "width" || name == "height")
36             return "0";
37     }
38     return attr.value();
39 };
```

#### 3.4.3.2 getInstance()

```
Parser * Parser::getInstance (
            const std::string & file_name )  [static]
```

Gets the singleton instance of Parser.

**Parameters**

| | |
|---|---|
| *file_name* | The name of the SVG file to parse. |

**Returns**

The singleton instance of Parser.

**Note**

This function is thread-safe.

Definition at line 9 of file Parser.cpp.

```
9                                                                   {
10      if (instance == nullptr) {
11          instance = new Parser(file_name);
12          instance->parseSVG();
13      }
14      return instance;
15  }
```

#### 3.4.3.3 parseColor()

```
sf::Color Parser::parseColor (
            pugi::xml_node node,
            std::string name )
```

Parse the color from the XML file.

**Parameters**

| | |
|---|---|
| *node* | The node of the XML file (pugi::xml_node is a typedef of pugixml) |
| *name* | The name of the attribute. |

**Returns**

The color which is parsed from the XML file.

**Note**

This function is private because it is only used by the Parser class.

The color is represented by sf::Color (SFML)

The color is parsed from the XML file in the format of "rgb(r, g, b)".

Definition at line 41 of file Parser.cpp.

```
41                                                                  {
42      auto getRgbColor = [](std::string color) -> sf::Color {
43          int r, g, b;
44          float a = 1;
45          sscanf(color.c_str(), "rgb(%d,%d,%d,%f)", &r, &g, &b, &a);
46          return sf::Color(r, g, b, 255 * a);
47      };
48
49      auto getHexColor = [](std::string color) -> sf::Color {
50          std::stringstream ss;
51          ss << std::hex << color.substr(1, 2) << " " << color.substr(3, 2) << " "
52             << color.substr(5, 2);
53          int r, g, b;
54          ss >> r >> g >> b;
55          if (color.size() > 7) {
56              std::stringstream ss;
57              ss << std::hex << color.substr(7, 2);
```

```
58              int a;
59              ss » a;
60              return sf::Color(r, g, b, a);
61          }
62          return sf::Color(r, g, b, 255);
63      };
64
65      std::string color = getAttribute(node, name);
66      for (auto& c : color) c = tolower(c);
67      if (color == "none")
68          return sf::Color::Transparent;
69      else {
70          sf::Color result;
71          if (color[0] == '#') {
72              result = getHexColor(color);
73          } else if (color.find("rgb") == std::string::npos) {
74              auto color_code = color_map.find(color);
75              if (color_code == color_map.end()) exit(-1);
76              result = color_code->second;
77          } else
78              result = getRgbColor(color);
79
80          result.a = result.a / 255.f *
81                     std::stof(getAttribute(node, name + "-opacity")) *
82                     std::stof(getAttribute(node, "opacity")) * 255;
83          return result;
84      }
85  }
```

#### 3.4.3.4 parsePoints()

```
std::vector< sf::Vector2f > Parser::parsePoints (
              pugi::xml_node node )
```

Parse the points from the XML file.

**Parameters**

| | |
|---|---|
| *node* | The node of the XML file (pugi::xml_node is a typedef of pugixml) |

**Returns**

The points which are parsed from the XML file.

**Note**

This function is private because it is only used by the Parser class.

The points are represented by std::vector< sf::Vector2f > (SFML)

The points are parsed from the XML file in the format of "x1,y1 x2,y2 x3,y3 ...".

Definition at line 87 of file Parser.cpp.

```
87                                                          {
88      std::vector< sf::Vector2f > points;
89      std::string points_string = getAttribute(node, "points");
90      std::string point = "";
91      int pos = 0;
92      float x = 0, y = 0;
93      for (int i = 0; i < (int)points_string.size(); i++) {
94          if (points_string[i] == ' ') {
95              if (point.size() > 0) {
96                  pos = point.find(',');
97                  x = std::stof(point.substr(0, pos));
98                  y = std::stof(point.substr(pos + 1));
99                  points.push_back(sf::Vector2f(x, y));
```

```
100                 point.clear();
101             }
102         } else {
103             point += points_string[i];
104         }
105     }
106
107     if (point.size() > 0) {
108         pos = point.find(',');
109         x = std::stof(point.substr(0, pos));
110         y = std::stof(point.substr(pos + 1));
111         points.push_back(sf::Vector2f(x, y));
112     }
113
114     return points;
115 }
```

### 3.4.3.5 parseSVG()

```
void Parser::parseSVG ( )
```

Parse the XML file.

**Parameters**

| | |
|---|---|
| *window* | The window to render the shapes on. |

**Note**

This function is private because it is only used by the Parser class.

The shapes are rendered on the window by calling the draw() method of the Shape class.

The shapes are rendered on the window in the order of the XML file.

Definition at line 117 of file Parser.cpp.

```
117                     {
118     for (pugi::xml_node tool = svg.first_child(); tool;
119          tool = tool.next_sibling()) {
120         sf::Color stroke_color = parseColor(tool, "stroke");
121         sf::Color fill_color = parseColor(tool, "fill");
122
123         float stroke_width = std::stof(getAttribute(tool, "stroke-width"));
124
125         if (tool.name() == std::string("rect")) {
126             Rect* shape = new Rect(std::stof(getAttribute(tool, "width")),
127                                    std::stof(getAttribute(tool, "height")),
128                                    std::stof(getAttribute(tool, "x")),
129                                    std::stof(getAttribute(tool, "y")),
130                                    fill_color, stroke_color, stroke_width);
131             shapes.push_back(shape);
132         } else if (tool.name() == std::string("line")) {
133             Line* shape =
134                 new Line(sf::Vector2f(std::stof(getAttribute(tool, "x1")),
135                                       std::stof(getAttribute(tool, "y1"))),
136                          sf::Vector2f(std::stof(getAttribute(tool, "x2")),
137                                       std::stof(getAttribute(tool, "y2"))),
138                          stroke_color, stroke_width);
139             shapes.push_back(shape);
140         } else if (tool.name() == std::string("text")) {
141             float x = std::stof(getAttribute(tool, "x"));
142             float y = std::stof(getAttribute(tool, "y"));
143             float font_size = std::stof(getAttribute(tool, "font-size"));
144             sf::String text = tool.text().get();
145             Text* shape = new Text(sf::Vector2f(x, y - font_size), text,
146                                    fill_color, font_size);
147             shapes.push_back(shape);
148         } else if (tool.name() == std::string("circle")) {
149             float radius = std::stof(getAttribute(tool, "r"));
150             Circle* shape = new Circle(
```

```
151                 radius,
152                 sf::Vector2f(std::stof(getAttribute(tool, "cx")) - radius,
153                              std::stof(getAttribute(tool, "cy")) - radius),
154                 fill_color, stroke_color, stroke_width);
155             shapes.push_back(shape);
156         } else if (tool.name() == std::string("ellipse")) {
157             float radius_x = std::stof(getAttribute(tool, "rx"));
158             float radius_y = std::stof(getAttribute(tool, "ry"));
159             Ellipse* shape = new Ellipse(
160                 sf::Vector2f(radius_x, radius_y),
161                 sf::Vector2f(std::stof(getAttribute(tool, "cx")) - radius_x,
162                              std::stof(getAttribute(tool, "cy")) - radius_y),
163                 fill_color, stroke_color, stroke_width);
164             shapes.push_back(shape);
165         } else if (tool.name() == std::string("polygon")) {
166             Polygon* shape =
167                 new Polygon(fill_color, stroke_color, stroke_width);
168             std::vector< sf::Vector2f > points = parsePoints(tool);
169             for (auto point : points) {
170                 shape->addPoint(point);
171             }
172             shape->polygonUpdate();
173             shapes.push_back(shape);
174         } else if (tool.name() == std::string("polyline")) {
175             Polyline* shape =
176                 new Polyline(stroke_width, stroke_color, fill_color);
177             std::vector< sf::Vector2f > points = parsePoints(tool);
178             for (auto point : points) {
179                 shape->addPoint(point);
180             }
181             shape->polylineUpdate();
182             shapes.push_back(shape);
183         } else if (tool.name() == std::string("path")) {
184             /*
185
186             PATH
187
188             */
189         } else {
190             continue;
191         }
192     }
193 }
```

#### 3.4.3.6 renderSVG()

```
void Parser::renderSVG (
             sf::RenderWindow & window )
```

Render the shapes on the window.

**Parameters**

| | |
|---|---|
| *window* | The window to render the shapes on. |

**Note**

This function is private because it is only used by the Parser class.

The shapes are rendered on the window by calling the draw() method of the Shape class.

The shapes are rendered on the window in the order of the XML file.

Apply Polymorphism to render the shapes on the window.

Definition at line 195 of file Parser.cpp.

```
195                                         {
196     for (auto shape : shapes) {
197         shape->draw(window);
198     }
```

```
199 }
```

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

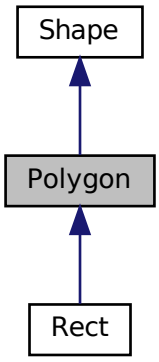
- src/Parser.hpp
- src/Parser.cpp

## 3.5 Polygon Class Reference

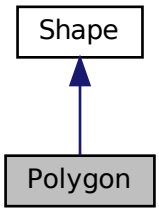
Represents a polygon in 2D space.

`#include <Polygon.hpp>`

Inheritance diagram for Polygon:




Collaboration diagram for Polygon:



## Public Member Functions

- Polygon (sf::Color fill=sf::Color::Transparent, sf::Color stroke=sf::Color::White, float stroke_thickness=0)

  *Constructs a Polygon object.*
- virtual std::size_t getPointCount () const

  *Gets the total number of vertices representing the polygon.*
- virtual sf::Vector2f getPoint (std::size_t index) const

  *Gets the position of a vertex in the polygon.*
- void addPoint (const sf::Vector2f &point)

  *Adds a vertex to the polygon.*
- void polygonUpdate ()

  *Updates the polygon.*

## Private Attributes

- std::vector< sf::Vector2f > points

  *Vertices of the polygon.*

## Additional Inherited Members

### 3.5.1 Detailed Description

Represents a polygon in 2D space.

The Polygon class is derived from the Shape class and defines a polygon with a variable number of vertices.

Definition at line 15 of file Polygon.hpp.

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 Polygon()

```
Polygon::Polygon (
            sf::Color fill = sf::Color::Transparent,
            sf::Color stroke = sf::Color::White,
            float stroke_thickness = 0 )
```

Constructs a Polygon object.

**Parameters**

| | |
|---|---|
| *fill* | Fill color of the polygon (default is sf::Color::Transparent). |
| *stroke* | Outline color of the polygon (default is sf::Color::White). |
| *stroke_thickness* | Thickness of the polygon outline (default is 0). |

Definition at line 3 of file Polygon.cpp.

```
3                                                                                {
4     setFillColor(fill);
5     setOutlineColor(stroke);
6     setOutlineThickness(stroke_thickness);
7 }
```

## 3.5.3 Member Function Documentation

### 3.5.3.1 addPoint()

```
void Polygon::addPoint (
            const sf::Vector2f & point )
```

Adds a vertex to the polygon.

**Parameters**

| | |
|---|---|
| *point* | The position of the vertex to be added. |

Definition at line 19 of file Polygon.cpp.

```
19 { points.push_back(point); }
```

### 3.5.3.2 getPoint()

```
sf::Vector2f Polygon::getPoint (
            std::size_t index ) const  [virtual]
```

Gets the position of a vertex in the polygon.

**Parameters**

| | |
|---|---|
| *index* | The index of the vertex. |

**Returns**

The position of the specified vertex in the polygon.

Implements Shape.

Reimplemented in Rect.

Definition at line 11 of file Polygon.cpp.

```
11                                                      {
12     if (index < points.size()) {
13         return points[index];
14     } else {
15         return points[index % points.size()];
16     }
17 }
```

#### 3.5.3.3 getPointCount()

```
std::size_t Polygon::getPointCount ( ) const  [virtual]
```

Gets the total number of vertices representing the polygon.

**Returns**

The number of vertices representing the polygon.

Implements Shape.

Reimplemented in Rect.

Definition at line 9 of file Polygon.cpp.

```
9 { return points.size(); }
```

#### 3.5.3.4 polygonUpdate()

```
void Polygon::polygonUpdate ( )
```

Updates the polygon.

This method is provided for consistency with other shapes but does not introduce any additional behavior for polygons.

Definition at line 21 of file Polygon.cpp.

```
21 { update(); }
```

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

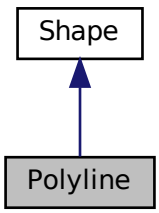
- src/graphics/Polygon.hpp
- src/graphics/Polygon.cpp

## 3.6 Polyline Class Reference

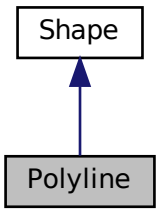
Represents a polyline in 2D space.

```
#include <Polyline.hpp>
```

Inheritance diagram for Polyline:



Collaboration diagram for Polyline:




## Public Member Functions

- Polyline (float stroke_Width=0, const sf::Color &stroke_color=sf::Color::White, const sf::Color &fill=sf::Color↩ ::Transparent)

  *Constructs a Polyline object.*
- void addPoint (const sf::Vector2f &point)

  *Adds a vertex to the polyline.*
- void draw (sf::RenderWindow &target, sf::RenderStates states=sf::RenderStates::Default) const

  *Draws the polyline on the specified render target.*
- sf::Vector2f getPoint (std::size_t index) const override

  *Gets the position of a vertex in the polyline.*
- std::size_t getPointCount () const override

  *Gets the total number of vertices representing the polyline.*
- void polylineUpdate ()

  *Updates the polyline.*

## Private Attributes

- std::vector< sf::Vector2f > points

  *Vertices of the polyline.*

## Additional Inherited Members

### 3.6.1 Detailed Description

Represents a polyline in 2D space.

The Polyline class is derived from the Shape class and defines a polyline with a variable number of vertices.

Definition at line 17 of file Polyline.hpp.

### 3.6.2 Constructor & Destructor Documentation

#### 3.6.2.1 Polyline()

```
Polyline::Polyline (
            float stroke_Width = 0,
            const sf::Color & stroke_color = sf::Color::White,
            const sf::Color & fill = sf::Color::Transparent )
```

Constructs a Polyline object.

**Parameters**

| | |
|---|---|
| *stroke_Width* | The stroke width of the polyline (default is 0). |
| *stroke_color* | The stroke color of the polyline (default is sf::Color::White). |
| *fill* | The fill color of the polyline (default is sf::Color::Transparent). |

Definition at line 145 of file Polyline.cpp.

```
146                                         {
147     setOutlineThickness(stroke_width);
148     setOutlineColor(stroke_color);
149     setFillColor(fill);
150 }
```

### 3.6.3 Member Function Documentation

#### 3.6.3.1 addPoint()

```
void Polyline::addPoint (
            const sf::Vector2f & point )
```

Adds a vertex to the polyline.

**Parameters**

| | |
|---|---|
| *point* | The position of the vertex to be added. |

Definition at line 152 of file Polyline.cpp.

```
152 { points.push_back(point); }
```

#### 3.6.3.2 draw()

```
void Polyline::draw (
            sf::RenderWindow & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [virtual]
```

Draws the polyline on the specified render target.

### Parameters

| | |
|---|---|
| *target* | The render target to draw on. |
| *states* | The render states to apply (default is sf::RenderStates::Default). |

Reimplemented from Shape.

Definition at line 164 of file Polyline.cpp.

```
164                                                                                 {
165     if (points.size() < 2) return;
166     sf::VertexArray lineStrip(sf::PrimitiveType::Quads);
167     sf::Vector2f p1a, p1b, p2a, p2b;
168     sf::Vector2f r_p1a, r_p1b, r_p2a, r_p2b;
169     for (std::size_t i = 1; i < points.size(); i++) {
170         sf::Vector2f p1 = points[i - 1];
171         sf::Vector2f p2 = points[i];
172 
173         sf::Vector2f delta = p2 - p1;
174         float length = std::sqrt(delta.x * delta.x + delta.y * delta.y);
175 
176         sf::Vector2f unitDirection = delta / length;
177 
178         sf::Vector2f perpendicularDirection(-unitDirection.y, unitDirection.x);
179 
180         float thickness = getOutlineThickness();
181         sf::Color stroke = getOutlineColor();
182 
183         p1a = p1 - perpendicularDirection * (thickness / 2.0f);
184         p1b = p1 + perpendicularDirection * (thickness / 2.0f);
185         p2a = p2 - perpendicularDirection * (thickness / 2.0f);
186         p2b = p2 + perpendicularDirection * (thickness / 2.0f);
187         if (i > 1) {
188             sf::VertexArray lS(sf::PrimitiveType::Quads);
189             if (isPerpendicular({points[i], points[i - 1]},
190                                 {points[i - 1], points[i - 2]})) {
191                 lS.append(sf::Vertex(
192                     findIntersection({r_p1a, r_p2a}, {p1a, p2a}), stroke));
193                 lS.append(sf::Vertex(
194                     findIntersection({r_p1a, r_p2a}, {p1b, p2b}), stroke));
195                 lS.append(sf::Vertex(
196                     findIntersection({r_p1b, r_p2b}, {p1b, p2b}), stroke));
197                 lS.append(sf::Vertex(
198                     findIntersection({r_p1b, r_p2b}, {p1a, p2a}), stroke));
199 
200             } else {
201                 lS.append(sf::Vertex(p1a, stroke));
202                 lS.append(sf::Vertex(r_p2a, stroke));
203                 lS.append(sf::Vertex(p1b, stroke));
204                 lS.append(sf::Vertex(r_p2b, stroke));
205             }
206             target.draw(lS);
207         }
208         r_p1a = p1a;
209         r_p1b = p1b;
210         r_p2a = p2a;
211         r_p2b = p2b;
212         lineStrip.append(sf::Vertex(p1a, stroke));
213         lineStrip.append(sf::Vertex(p1b, stroke));
214         lineStrip.append(sf::Vertex(p2b, stroke));
215         lineStrip.append(sf::Vertex(p2a, stroke));
216     }
217     std::vector< ClosedPolygon > cP = findClosedPolygons(points);
218     if (cP.size() > 0) {
219         sf::Color fillColor = getFillColor();
220         for (const ClosedPolygon& polygon : cP) {
221             if (polygon.cP.size() > 2) {  // Ensure it's a valid polygon
222                 sf::ConvexShape fillShape;
223                 fillShape.setFillColor(fillColor);
224                 fillShape.setOutlineThickness(0);
225                 fillShape.setPointCount(polygon.cP.size());
226 
227                 // Set the points of the shape based on the polygon
228                 for (std::size_t j = 0; j < polygon.cP.size(); j++) {
229                     fillShape.setPoint(j, polygon.cP[j]);
230                 }
231 
232                 target.draw(fillShape);
233             }
234         }
235         target.draw(lineStrip);
236     }
237 }
```

#### 3.6.3.3 getPoint()

```
sf::Vector2f Polyline::getPoint (
            std::size_t index ) const  [override], [virtual]
```

Gets the position of a vertex in the polyline.

**Parameters**

| | |
|---|---|
| *index* | The index of the vertex. |

**Returns**

The position of the specified vertex in the polyline.

Implements Shape.

Definition at line 156 of file Polyline.cpp.

```
156                                                 {
157     if (index < points.size()) {
158         return points[index];
159     }
160     return sf::Vector2f(0, 0);
161 }
```

#### 3.6.3.4 getPointCount()

```
std::size_t Polyline::getPointCount ( ) const  [override], [virtual]
```

Gets the total number of vertices representing the polyline.

**Returns**

The number of vertices representing the polyline.

Implements Shape.

Definition at line 163 of file Polyline.cpp.

```
163 { return points.size(); }
```

#### 3.6.3.5 polylineUpdate()

```
void Polyline::polylineUpdate ( )
```

Updates the polyline.

This method is provided for consistency with other shapes but does not introduce any additional behavior for polylines.

Definition at line 154 of file Polyline.cpp.

```
154 { update(); }
```

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

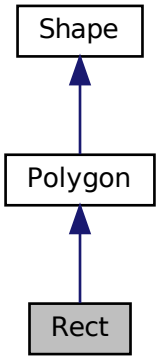
- src/graphics/Polyline.hpp
- src/graphics/Polyline.cpp

## 3.7 Rect Class Reference

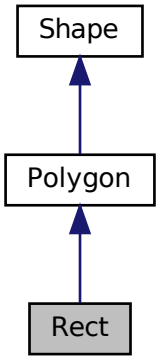
Represents a rectangle in 2D space.

`#include <Rect.hpp>`

Inheritance diagram for Rect:




Collaboration diagram for Rect:




### Public Member Functions

- Rect (float width, float height, float x, float y, sf::Color fill, sf::Color stroke, float stroke_thickness)

  *Constructs a Rect object.*
- virtual std::size_t getPointCount () const override

  *Gets the total number of vertices representing the rectangle.*
- virtual sf::Vector2f getPoint (std::size_t index) const override

  *Gets the position of a vertex in the rectangle.*

## Private Attributes

- float width

  *Width of the rectangle.*
- float height

  *Height of the rectangle.*
- sf::Vector2f rect_size

  *Size of the rectangle.*

## Additional Inherited Members

### 3.7.1 Detailed Description

Represents a rectangle in 2D space.

The Rect class is derived from the Polygon class and defines a rectangle with a specified width, height, position, fill color, stroke color, and stroke thickness.

Definition at line 13 of file Rect.hpp.

### 3.7.2 Constructor & Destructor Documentation

#### 3.7.2.1 Rect()

```
Rect::Rect (
            float width,
            float height,
            float x,
            float y,
            sf::Color fill,
            sf::Color stroke,
            float stroke_thickness )
```

Constructs a Rect object.

**Parameters**

| | |
|---|---|
| *width* | The width of the rectangle. |
| *height* | The height of the rectangle. |
| *x* | The X-coordinate of the position. |
| *y* | The Y-coordinate of the position. |
| *fill* | Fill color of the rectangle. |
| *stroke* | Outline color of the rectangle. |
| *stroke_thickness* | Thickness of the rectangle outline. |

Definition at line 3 of file Rect.cpp.

```
5     : Polygon(fill, stroke, stroke_thickness), width(width), height(height) {
6     addPoint(sf::Vector2f(0, 0));
7     addPoint(sf::Vector2f(width, 0));
8     addPoint(sf::Vector2f(width, height));
9     addPoint(sf::Vector2f(0, height));
10
11     setPosition(x, y);
12     update();
13 }
```

### 3.7.3 Member Function Documentation

#### 3.7.3.1 getPoint()

```
sf::Vector2f Rect::getPoint (
            std::size_t index ) const  [override], [virtual]
```

Gets the position of a vertex in the rectangle.

**Parameters**

| | |
|---|---|
| *index* | The index of the vertex (0 for top-left, 1 for top-right, 2 for bottom-right, 3 for bottom-left). |

**Returns**

The position of the specified vertex in the rectangle.

Reimplemented from Polygon.

Definition at line 19 of file Rect.cpp.

```
19                                              {
20     return Polygon::getPoint(index);  // Inherited from Polygon
21 }
```

#### 3.7.3.2 getPointCount()

```
std::size_t Rect::getPointCount ( ) const  [override], [virtual]
```

Gets the total number of vertices representing the rectangle.

In this case, it always returns 4 since a rectangle has four corners.

**Returns**

The number of vertices representing the rectangle.

Reimplemented from Polygon.

Definition at line 15 of file Rect.cpp.

```
15                                  {
16     return Polygon::getPointCount();  // Inherited from Polygon
17 }
```

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

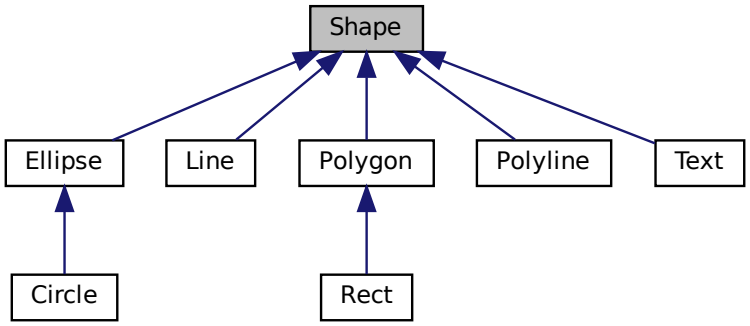
- src/graphics/Rect.hpp
- src/graphics/Rect.cpp

## 3.8 Shape Class Reference

Represents a shape in 2D space.

`#include <Shape.hpp>`

Inheritance diagram for Shape:




### Public Member Functions

- virtual ~Shape ()=default

  *Virtual constructor.*
- void setFillColor (const sf::Color &color)

  *Sets the fill color of the shape.*
- void setOutlineColor (const sf::Color &color)

  *Sets the outline color of the shape.*
- void setOutlineThickness (float thickness)

  *Sets the outline thickness of the shape.*
- const sf::Color & getFillColor () const

  *Gets the fill color of the shape.*
- const sf::Color & getOutlineColor () const

  *Gets the outline color of the shape.*
- float getOutlineThickness () const

  *Gets the outline thickness of the shape.*
- void setPosition (float x, float y)

  *Sets the position of the shape.*
- void setPosition (const sf::Vector2f &position)

  *Sets the position of the shape.*
- virtual std::size_t getPointCount () const =0

  *Virtual method: Get the number of point of the shape (for Polygon shape)*
- virtual sf::Vector2f getPoint (std::size_t index) const =0

  *Virtual method: Get the position of the point on the shape (for Polygon shape)*
- virtual void draw (sf::RenderWindow &target, sf::RenderStates states=sf::RenderStates::Default) const

  *Virtual method: Draw the shape on the specified render target.*
- const sf::Transform & getTransform () const

  *Gets the shape transform.*
- const sf::Transform & getInverseTransform () const

  *Gets the inverse shape transform.*

## Protected Member Functions

- Shape ()

  *Constructs a Shape object.*
- void update ()

  *Sets the texture of the shape.*

## Private Member Functions

- void updateFillColors ()

  *Updates the fill colors of the shape.*
- void updateOutline ()

  *Updates the outline of the shape.*
- void updateOutlineColors ()

  *Updates the outline colors of the shape.*

## Private Attributes

- const sf::Texture * texture

  *Texture of the shape.*
- sf::Color fill_color

  *Fill color.*
- sf::Color outline_color

  *Outline color.*
- float outline_thickness

  *Thickness of the shape's outline.*
- sf::VertexArray vertices

  *Vertex array containing the fill geometry.*
- sf::VertexArray outline_vertices

  *Vertex array containing the outline geometry.*
- sf::FloatRect inside_bounds

  *Bounding rectangle of the inside (fill)*
- sf::FloatRect bounds

  *Bounding rectangle of the outside (outline + fill)*
- sf::Vector2f origin

  *Origin of translation/rotation/scaling of the object.*
- sf::Vector2f position

  *Position of the object in the 2D world.*
- float rotation

  *Orientation of the object, in degrees.*
- sf::Vector2f scale

  *Scale of the object.*
- sf::Transform transform

  *Combined transformation of the object.*
- bool transform_need_update

  *Does the transform need to be recomputed?*
- sf::Transform inverse_transform

  *Combined transformation of the object.*
- bool inverse_transform_need_update

  *Same as transform but for inverse.*

### 3.8.1 Detailed Description

Represents a shape in 2D space.

**Note**

This class is abstract and cannot be instantiated.

This class is applied Abstract Factory design pattern and used as interface for other shapes.

Definition at line 16 of file Shape.hpp.

### 3.8.2 Constructor & Destructor Documentation

#### 3.8.2.1 Shape()

```
Shape::Shape ( )  [protected]
```

Constructs a Shape object.

**Note**

This constructor is protected because Shape is an abstract class that cannot be instantiated.

Definition at line 40 of file Shape.cpp.

```
41     : texture(nullptr), fill_color(255, 255, 255), outline_color(255, 255, 255),
42       outline_thickness(0), vertices(sf::TriangleFan),
43       outline_vertices(sf::TriangleStrip), inside_bounds(), bounds(),
44       origin(0, 0), position(0, 0), rotation(0), scale(1, 1), transform(),
45       transform_need_update(true), inverse_transform(),
46       inverse_transform_need_update(true) {}
```

### 3.8.3 Member Function Documentation

#### 3.8.3.1 draw()

```
void Shape::draw (
            sf::RenderWindow & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [virtual]
```

Virtual method: Draw the shape on the specified render target.

**Parameters**

| | |
|---|---|
| *target* | The render target (sf::Renderwindow is a typedef of SFML drawing window) |
| *states* | The render states to apply (default is sf::RenderStates::Default) |

Reimplemented in Text, and Polyline.

Definition at line 79 of file Shape.cpp.

```
79                                                                  {
80     states.transform *= getTransform();
81
82     // Render the inside
83     states.texture = texture;
84     target.draw(vertices, states);
85
86     // Render the outline
87     if (outline_thickness != 0) {
88         states.texture = nullptr;
89         target.draw(outline_vertices, states);
90     }
91 }
```

#### 3.8.3.2 getFillColor()

```
const sf::Color & Shape::getFillColor ( ) const
```

Gets the fill color of the shape.

**Returns**

The fill color of the shape.

**Note**

The default fill color is white.

Definition at line 23 of file Shape.cpp.

```
23 { return fill_color; }
```

#### 3.8.3.3 getInverseTransform()

```
const sf::Transform & Shape::getInverseTransform ( ) const
```

Gets the inverse shape transform.

**Returns**

The inverse shape transform (sf::Transform is a typedef of SFML)

**Note**

This function returns the inverse of the combined transform of the object.

Definition at line 173 of file Shape.cpp.

```
173                                                 {
174     // Recompute the inverse transform if needed
175     if (inverse_transform_need_update) {
176         inverse_transform = getTransform().getInverse();
177         inverse_transform_need_update = false;
178     }
179
180     return inverse_transform;
181 }
```

#### 3.8.3.4 getOutlineColor()

```
const sf::Color & Shape::getOutlineColor ( ) const
```

Gets the outline color of the shape.

**Returns**

The outline color of the shape.

**Note**

The default outline color is white.

Definition at line 30 of file Shape.cpp.

```
30 { return outline_color; }
```

#### 3.8.3.5 getOutlineThickness()

```
float Shape::getOutlineThickness ( ) const
```

Gets the outline thickness of the shape.

**Returns**

The outline thickness of the shape.

**Note**

The default outline thickness is 0.

Definition at line 38 of file Shape.cpp.

```
38 { return outline_thickness; }
```

#### 3.8.3.6 getPoint()

```
virtual sf::Vector2f Shape::getPoint (
            std::size_t index ) const  [pure virtual]
```

Virtual method: Get the position of the point on the shape (for Polygon shape)

**Parameters**

| | |
|---|---|
| *index* | The index of the point |

**Returns**

The position of the specified point on the shape

**Note**

The returned point is in local coordinates, that is, the shape's transforms (position, rotation, scale) are not taken into account.

The result is undefined if index is out of the valid range.

The number of points of the shape is defined by the concrete implementation.

The returned vector is constant, which means that you can't modify its coordinates when you retrieve it.

This is a pure virtual method, so it has to be implemented by the derived class.

Implemented in Rect, Polyline, Ellipse, Text, Polygon, and Line.

#### 3.8.3.7 getPointCount()

```
virtual std::size_t Shape::getPointCount ( ) const  [pure virtual]
```

Virtual method: Get the number of point of the shape (for Polygon shape)

**Returns**

The number of points of the shape

**Note**

This is a pure virtual method, so it has to be implemented by the derived class to define how the shape should be drawn.

Implemented in Rect, Polyline, Text, Polygon, Line, and Ellipse.

#### 3.8.3.8 getTransform()

```
const sf::Transform & Shape::getTransform ( ) const
```

Gets the shape transform.

**Returns**

The shape transform (sf::Transform is a typedef of SFML)

**Note**

This function returns the combined transform of the object.

The transform is a combination of the position, rotation, and scale of the object.

Definition at line 153 of file Shape.cpp.

```
153                                          {
154     // Recompute the combined transform if needed
155     if (transform_need_update) {
156         float angle = -rotation * acos(-1) / 180.f;
157         float cosine = std::cos(angle);
158         float sine = std::sin(angle);
159         float sxc = scale.x * cosine;
160         float syc = scale.y * cosine;
161         float sxs = scale.x * sine;
162         float sys = scale.y * sine;
163         float tx = -origin.x * sxc - origin.y * sys + position.x;
164         float ty = origin.x * sxs - origin.y * syc + position.y;
165 
166         transform = sf::Transform(sxc, sys, tx, -sxs, syc, ty, 0.f, 0.f, 1.f);
167         transform_need_update = false;
168     }
169 
170     return transform;
171 }
```

### 3.8.3.9 setFillColor()

```
void Shape::setFillColor (
              const sf::Color & color )
```

Sets the fill color of the shape.

**Parameters**

| | |
|---|---|
| *color* | The new fill color of the shape. |

Definition at line 18 of file Shape.cpp.

```
18                                           {
19     fill_color = color;
20     updateFillColors();
21 }
```

### 3.8.3.10 setOutlineColor()

```
void Shape::setOutlineColor (
              const sf::Color & color )
```

Sets the outline color of the shape.

**Parameters**

| | |
|---|---|
| *color* | The new outline color of the shape. |

Definition at line 25 of file Shape.cpp.

```
25                                                  {
26     outline_color = color;
27     updateOutlineColors();
28 }
```

#### 3.8.3.11 setOutlineThickness()

```
void Shape::setOutlineThickness (
             float thickness )
```

Sets the outline thickness of the shape.

**Parameters**

| | |
|---|---|
| *thickness* | The new outline thickness of the shape. |

**Note**

If the thickness is negative, the outline will be inside the shape. If the thickness is positive, the outline will be outside the shape. If the thickness is zero, no outline will be drawn.

The default outline thickness is 0.

The outline thickness cannot be greater than the radius of the shape.

Definition at line 32 of file Shape.cpp.

```
32                                                  {
33     outline_thickness = thickness;
34     update();  // recompute everything because the whole shape must be
35                // offset
36 }
```

#### 3.8.3.12 setPosition() [1/2]

```
void Shape::setPosition (
             const sf::Vector2f & position )
```

Sets the position of the shape.

**Parameters**

| | |
|---|---|
| *position* | The new position of the shape (sf::Vector2f is a typedef of coordination in SFML) |

**Note**

The default position of the shape is (0, 0).

The position of the shape is relative to its origin.

Definition at line 190 of file Shape.cpp.

```
190                                                 {
```

```
191     setPosition(position.x, position.y);
192 }
```

#### 3.8.3.13 setPosition() [2/2]

```
void Shape::setPosition (
            float x,
            float y )
```

Sets the position of the shape.

**Parameters**

| | |
|---|---|
| *x* | The x coordinate of the new position |
| *y* | The y coordinate of the new position |

**Note**

The default position of the shape is (0, 0).

The position of the shape is relative to its origin.

Definition at line 183 of file Shape.cpp.

```
183                                     {
184     position.x = x;
185     position.y = y;
186     transform_need_update = true;
187     inverse_transform_need_update = true;
188 }
```

#### 3.8.3.14 update()

```
void Shape::update ( )  [protected]
```

Sets the texture of the shape.

**Parameters**

| | |
|---|---|
| *texture* | The new texture of the shape |

**Note**

The texture is not copied, it is referenced by the shape.

The default texture is NULL.

Definition at line 48 of file Shape.cpp.

```
48                      {
49     // Get the total number of points of the shape
50     std::size_t count = getPointCount();
51     if (count < 3) {
```

```
52          vertices.resize(0);
53          outline_vertices.resize(0);
54          return;
55      }
56
57      vertices.resize(count + 2);  // + 2 for center and repeated first point
58
59      // Position
60      for (std::size_t i = 0; i < count; ++i)
61          vertices[i + 1].position = getPoint(i);
62      vertices[count + 1].position = vertices[1].position;
63
64      // Update the bounding rectangle
65      vertices[0] = vertices[1];  // so that the result of getBounds() is correct
66      inside_bounds = vertices.getBounds();
67
68      // Compute the center and make it the first vertex
69      vertices[0].position.x = inside_bounds.left + inside_bounds.width / 2;
70      vertices[0].position.y = inside_bounds.top + inside_bounds.height / 2;
71
72      // Color
73      updateFillColors();
74
75      // Outline
76      updateOutline();
77  }
```

#### 3.8.3.15 updateFillColors()

```
void Shape::updateFillColors ( )  [private]
```

Updates the fill colors of the shape.

**Note**

This method call the update() method.

Definition at line 93 of file Shape.cpp.

```
93                                  {
94      for (std::size_t i = 0; i < vertices.getVertexCount(); ++i)
95          vertices[i].color = fill_color;
96  }
```

#### 3.8.3.16 updateOutline()

```
void Shape::updateOutline ( )  [private]
```

Updates the outline of the shape.

**Note**

This method call the update() method.

Definition at line 98 of file Shape.cpp.

```
98                          {
99     // Return if there is no outline
100     if (outline_thickness == 0.f) {
101         outline_vertices.clear();
102         bounds = inside_bounds;
103         return;
104     }
105
106     std::size_t count = vertices.getVertexCount() - 2;
107     outline_vertices.resize((count + 1) * 2);
108
109     for (std::size_t i = 0; i < count; ++i) {
110         std::size_t index = i + 1;
111
112         // Get the two segments shared by the current point
113         sf::Vector2f p0 =
114             (i == 0) ? vertices[count].position : vertices[index - 1].position;
115         sf::Vector2f p1 = vertices[index].position;
116         sf::Vector2f p2 = vertices[index + 1].position;
117
118         // Compute their normal
119         sf::Vector2f n1 = computeNormal(p0, p1);
120         sf::Vector2f n2 = computeNormal(p1, p2);
121
122         // Make sure that the normals point towards the outside of the shape
123         // (this depends on the order in which the points were defined)
124         if (dotProduct(n1, vertices[0].position - p1) > 0) n1 = -n1;
125         if (dotProduct(n2, vertices[0].position - p1) > 0) n2 = -n2;
126
127         // Combine them to get the extrusion direction
128         float factor = 1.f + (n1.x * n2.x + n1.y * n2.y);
129         sf::Vector2f normal = (n1 + n2) / factor;
130
131         // Update the outline points
132         sf::Vector2f offset = normal * (outline_thickness / 2.f);
133         outline_vertices[i * 2 + 0].position = p1 - offset;
134         outline_vertices[i * 2 + 1].position = p1 + offset;
135     }
136
137     // Duplicate the first point at the end, to close the outline
138     outline_vertices[count * 2 + 0].position = outline_vertices[0].position;
139     outline_vertices[count * 2 + 1].position = outline_vertices[1].position;
140
141     // Update outline colors
142     updateOutlineColors();
143
144     // Update the shape's bounds
145     bounds = outline_vertices.getBounds();
146 }
```

### 3.8.3.17 updateOutlineColors()

```
void Shape::updateOutlineColors ( )  [private]
```

Updates the outline colors of the shape.

**Note**

This method call the update() method.

Definition at line 148 of file Shape.cpp.

```
148                                 {
149     for (std::size_t i = 0; i < outline_vertices.getVertexCount(); ++i)
150         outline_vertices[i].color = outline_color;
151 }
```

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

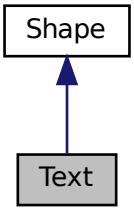
- src/graphics/Shape.hpp
- src/graphics/Shape.cpp

## 3.9 Text Class Reference

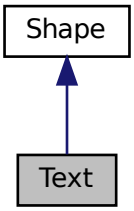
Represents text in 2D space.

`#include <Text.hpp>`

Inheritance diagram for Text:




Collaboration diagram for Text:




### Public Member Functions

- Text (sf::Vector2f pos, sf::String TEXT, sf::Color fill_color=sf::Color::Black, float font_size=1)

  *Constructs a Text object.*
- virtual std::size_t getPointCount () const

  *Gets the total number of points representing the text.*
- virtual sf::Vector2f getPoint (std::size_t index) const

  *Gets a dummy point for compatibility with Shape interface.*
- void draw (sf::RenderWindow &target, sf::RenderStates states=sf::RenderStates::Default) const

  *Draws the text on the specified render target.*

### Static Public Attributes

- static sf::Font font

  *Static font shared across all Text instances.*

## Private Attributes

- sf::Text text

  *Text element.*

## Additional Inherited Members

### 3.9.1 Detailed Description

Represents text in 2D space.

The Text class is derived from the Shape class and defines a text element with a specified position, string, fill color, and font size.

Definition at line 12 of file Text.hpp.

### 3.9.2 Constructor & Destructor Documentation

#### 3.9.2.1 Text()

```
Text::Text (
             sf::Vector2f pos,
             sf::String TEXT,
             sf::Color fill_color = sf::Color::Black,
             float font_size = 1 )
```

Constructs a Text object.

**Parameters**

| | |
|---|---|
| *pos* | The position of the text. |
| *TEXT* | The string of the text. |
| *fill_color* | The fill color of the text (default is sf::Color::Black). |
| *font_size* | The font size of the text (default is 1). |

Definition at line 7 of file Text.cpp.

```
8                                {
9      text.setPosition(pos);
10      text.setFont(font);
11      text.setCharacterSize(font_size);
12      text.setFillColor(fill_color);
13      text.setString(TEXT);
14  }
```

### 3.9.3 Member Function Documentation

#### 3.9.3.1 draw()

```
void Text::draw (
            sf::RenderWindow & target,
            sf::RenderStates states = sf::RenderStates::Default ) const  [virtual]
```

Draws the text on the specified render target.

**Parameters**

| | |
|---|---|
| *target* | The render target to draw on. |
| *states* | The render states to apply (default is sf::RenderStates::Default). |

Reimplemented from Shape.

Definition at line 21 of file Text.cpp.

```
21                                                          {
22     target.draw(this->text);
23 }
```

#### 3.9.3.2 getPoint()

```
sf::Vector2f Text::getPoint (
            std::size_t index ) const  [virtual]
```

Gets a dummy point for compatibility with Shape interface.

Since text is not represented by points, this method always returns (0, 0).

**Parameters**

| | |
|---|---|
| *index* | The index of the dummy point (ignored). |

**Returns**

A dummy point for compatibility.

Implements Shape.

Definition at line 17 of file Text.cpp.

```
17                                                  {
18     return sf::Vector2f(0, 0);
19 }
```

#### 3.9.3.3 getPointCount()

```
std::size_t Text::getPointCount ( ) const  [virtual]
```

Gets the total number of points representing the text.

Since text is not represented by points, this method always returns 0.

**Returns**

The number of points representing the text.

Implements Shape.

Definition at line 15 of file Text.cpp.

```
15 { return 0; }
```

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

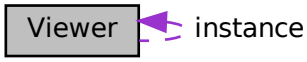
- src/graphics/Text.hpp
- src/graphics/Text.cpp

## 3.10 Viewer Class Reference

Represents a viewer for handling events and interactions with an SFML window.

```
#include <Viewer.hpp>
```

Collaboration diagram for Viewer:




### Public Member Functions

- void handleEvents (sf::Event event)

  *Handles SFML events.*
- void handleDragging ()

  *Handles dragging interaction.*
- void operator= (const Viewer &)=delete

  *Deleted assignment operator to prevent copying of Viewer instances.*
- Viewer (const Viewer &)=delete

  *Deleted copy constructor to prevent copying of Viewer instances.*

### Static Public Member Functions

- static Viewer * getInstance (sf::RenderWindow &Window, sf::View &View)

  *Gets the singleton instance of Viewer.*

### Private Member Functions

- Viewer (sf::RenderWindow &Window, sf::View &View)

  *Constructs a Viewer object.*
- void zoom (float factor)

  *Zooms the view by the specified factor.*
- void rotate (float angle)

  *Rotates the view by the specified angle.*
- void startDragging ()

  *Starts dragging the view.*
- void stopDragging ()

  *Stops dragging the view.*
- void moveView (const sf::Vector2f &offset)

  *Moves the view by the specified offset.*

### Private Attributes

- sf::RenderWindow & window

  *Reference to the SFML window.*
- sf::View & view

  *Reference to the SFML view.*
- sf::Vector2i last_mouse_position

  *Last recorded mouse position.*
- bool is_mouse_dragging

  *Flag indicating whether mouse dragging is active.*

### Static Private Attributes

- static Viewer * instance = nullptr

  *Singleton instance of Viewer.*

## 3.10.1 Detailed Description

Represents a viewer for handling events and interactions with an SFML window.

The Viewer class is responsible for handling events, such as zooming, rotating, and dragging, to interact with an SFML window and view.

Definition at line 13 of file Viewer.hpp.

## 3.10.2 Constructor & Destructor Documentation

### 3.10.2.1 Viewer()

```
Viewer::Viewer (
            sf::RenderWindow & Window,
            sf::View & View )  [private]
```

Constructs a Viewer object.

**Parameters**

| | |
|---|---|
| *Window* | The SFML window to associate with the viewer. |
| *View* | The SFML view to associate with the viewer. |

Definition at line 12 of file Viewer.cpp.

```
13     : window(Window), view(View) {}
```

### 3.10.3 Member Function Documentation

#### 3.10.3.1 getInstance()

```
Viewer * Viewer::getInstance (
            sf::RenderWindow & Window,
            sf::View & View )  [static]
```

Gets the singleton instance of Viewer.

**Parameters**

| | |
|---|---|
| *Window* | The SFML window to associate with the viewer. |
| *View* | The SFML view to associate with the viewer. |

**Returns**

The singleton instance of Viewer.

Definition at line 5 of file Viewer.cpp.

```
5                                                                    {
6     if (instance == nullptr) {
7         instance = new Viewer(Window, View);
8     }
9     return instance;
10 }
```

#### 3.10.3.2 handleEvents()

```
void Viewer::handleEvents (
            sf::Event event )
```

Handles SFML events.

**Parameters**

| | |
|---|---|
| *event* | The SFML event to handle. |

Definition at line 15 of file Viewer.cpp.

```
15                                      {
16     if (event.type == sf::Event::Closed) {
17         window.close();
18     }
19
20     // Zoom in by + (including '=' key)
21     if ((event.type == sf::Event::KeyPressed &&
22          event.key.code == sf::Keyboard::Add) ||
23         (event.type == sf::Event::KeyPressed &&
24          event.key.code == sf::Keyboard::Equal)) {
25         zoom(0.9f);
26     }
27
28     // Zoom out by - (including '-' key)
29     if ((event.type == sf::Event::KeyPressed &&
30          event.key.code == sf::Keyboard::Subtract) ||
31         (event.type == sf::Event::KeyPressed &&
32          event.key.code == sf::Keyboard::Hyphen)) {
33         zoom(1.1f);
34     }
35
36     // Rotate clockwise by 'R' key
37     if (event.type == sf::Event::KeyPressed &&
38         event.key.code == sf::Keyboard::R) {
39         rotate(90.0f);
40     }
41
42     // Rotate anti-clockwise by 'E' key
43     if (event.type == sf::Event::KeyPressed &&
44         event.key.code == sf::Keyboard::E) {
45         rotate(-90.0f);
46     }
47
48     // Zoom in/out with mouse scroll
49     if (event.type == sf::Event::MouseWheelScrolled) {
50         if (event.mouseWheelScroll.wheel == sf::Mouse::VerticalWheel) {
51             if (event.mouseWheelScroll.delta > 0) {
52                 zoom(0.9f);
53             } else {
54                 zoom(1.1f);
55             }
56         }
57     }
58
59     // Zoom in/out with touchpad (control pad)
60     if (event.type == sf::Event::TouchMoved) {
61         // Assuming that touchpad input scales the zoom based on movement
62         float delta = event.touch.y;
63         if (delta > 0) {
64             zoom(0.9f);
65         } else {
66             zoom(1.1f);
67         }
68     }
69
70     // Start dragging the left mouse button
71     if (event.type == sf::Event::MouseButtonPressed &&
72         event.mouseButton.button == sf::Mouse::Left) {
73         startDragging();
74     }
75
76     // Finish dragging the left mouse button
77     if (event.type == sf::Event::MouseButtonReleased &&
78         event.mouseButton.button == sf::Mouse::Left) {
79         stopDragging();
80     }
81 }
```

#### 3.10.3.3 moveView()

```
void Viewer::moveView (
            const sf::Vector2f & offset )  [private]
```

Moves the view by the specified offset.

**Parameters**

| | |
|---|---|
| *offset* | The offset by which to move the view. |

Definition at line 110 of file Viewer.cpp.

```
110                                          {
111     view.move(-offset);
112     window.setView(view);
113 }
```

#### 3.10.3.4 rotate()

```
void Viewer::rotate (
            float angle )  [private]
```

Rotates the view by the specified angle.

**Parameters**

| | |
|---|---|
| *angle* | The rotation angle. |

Definition at line 98 of file Viewer.cpp.

```
98                                {
99     view.rotate(angle);
100     window.setView(view);
101 }
```

#### 3.10.3.5 zoom()

```
void Viewer::zoom (
            float factor )  [private]
```

Zooms the view by the specified factor.

**Parameters**

| | |
|---|---|
| *factor* | The zoom factor. |

Definition at line 93 of file Viewer.cpp.

```
93                                {
94     view.zoom(factor);
95     window.setView(view);
96 }
```

### 3.10.4 Member Data Documentation

#### 3.10.4.1 is_mouse_dragging

```
bool Viewer::is_mouse_dragging  [private]
```

**Initial value:**

```
=
       false
```

Flag indicating whether mouse dragging is active.

Definition at line 60 of file Viewer.hpp.

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

- src/Viewer.hpp
- src/Viewer.cpp

# Index