|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 2**

|  |  |
| --- | --- |
| **Тема** Преобразование изображений: перенос, масштабирование, поворот,  комбинированные преобразования.  **Студент** Воякин А. Я.  **Группа ИУ**7-44Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель** Куров А.В. |  |

Москва.

2020 г.

**Цель работы:**

познакомиться с назначением, областями применения, сущностью операций преобразования; научиться разрабатывать программы, осуществляющие преобразования изображений на плоскости.

**Техническое задание:**

Должна быть разработана программа, реализующая операции  
преобразования:

1. переноса (перемещения) изображения;
2. масштабирования (увеличения или уменьшения размеров) изображения;
3. поворота изображения (изменение ориентации).

Выбор операции, задание параметров, определяющих каждую операцию, должно выполняться с помощью меню или с помощью системы кнопок и  
окон для ввода параметров. Должны быть реализованы функции вывода исходного изображения, текущего изображения, возврата к предыдущему изображению. Каждое преобразование должно применяться к текущему изображению.

**Индивидуальное задание:**



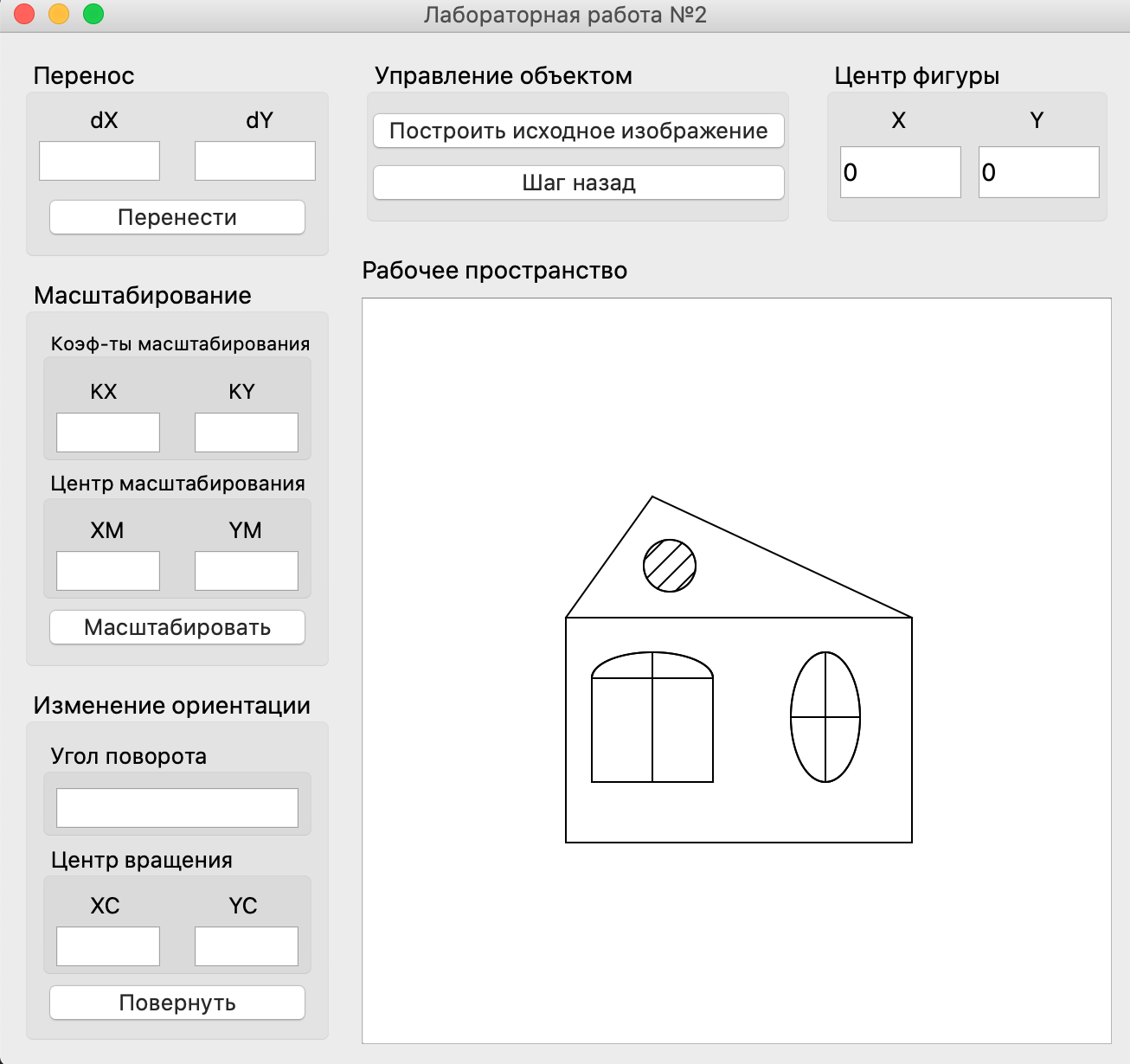
**Перенос изображения**

Для переноса точки из позиции с координатами (X,Y) в позицию с координатами (X1,Y1) надо к координате X добавить DX, а к координате Y - DY единиц.

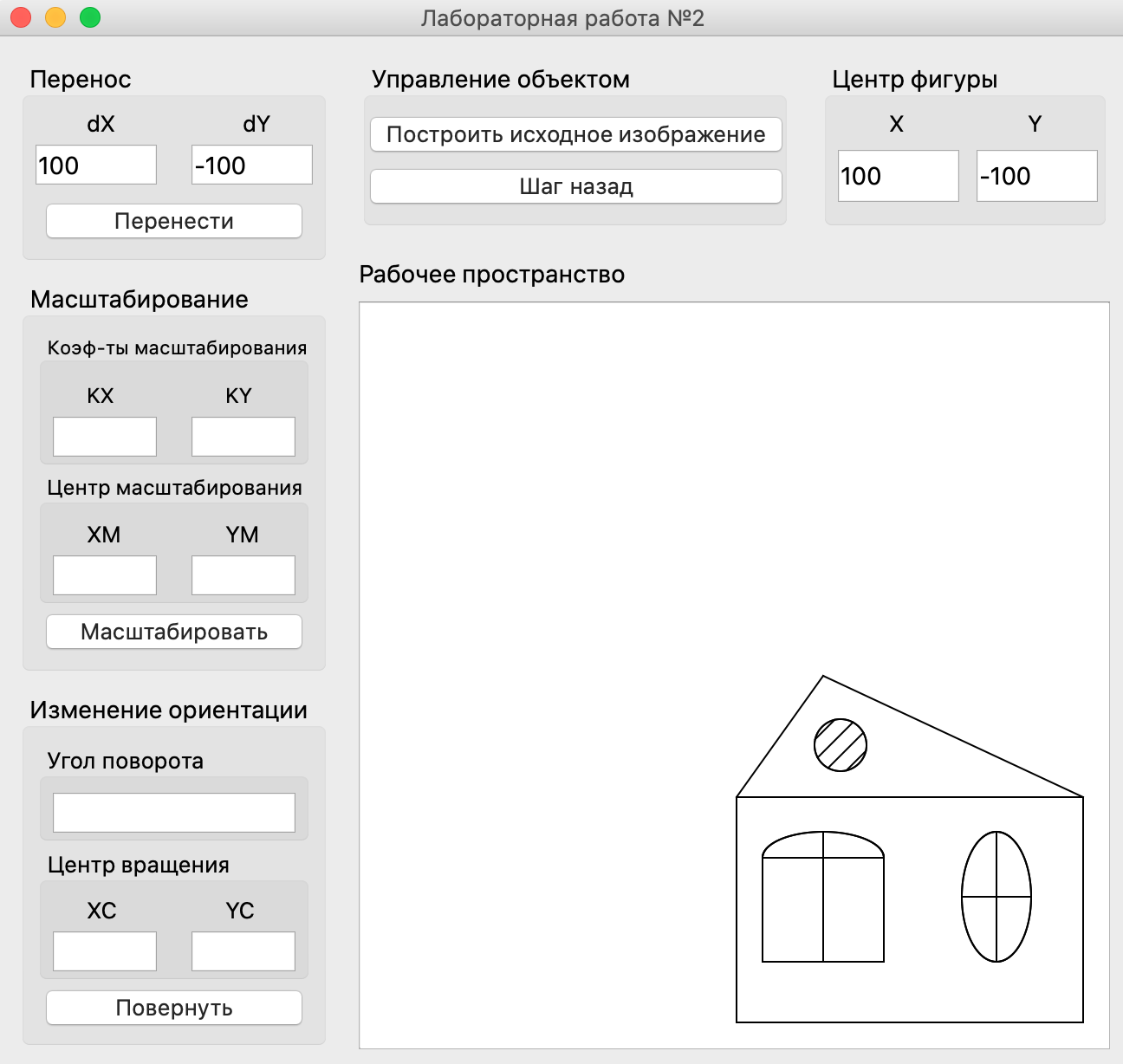
Положительное значение DX означает перемещение точки вправо по горизонтали, отрицательное - влево; положительное значение DY -перемещение вверх по вертикали, отрицательное - вниз.

**Пример перемещения изображения на 100 по горизонтали и на -100 по вертикали.**

Исходное изображение:



Преобразованное изображение:

****

**Масштабирование изображения**

Координаты промасштабированной точки определяются из следующих выражений:

где X, Y - координаты исходной точки;

X1, Y1 – координаты промасштабированной точки;

XM, YM - координаты центра масштабирования;

KX, KY - коэффициенты масштабирования.

При этом, если и , то рисунок увеличивается в размере и удаляется от начала координат; если и , то рисунок уменьшается в размерах и приближается к началу координат.

С помощью масштабирования можно растянуть или сжать изображение вдоль одной координатной оси, оставив его без изменения вдоль другой оси. Например, масштабируя квадрат с коэффициентами масштабирования KX = 1, KY = 2, получим прямоугольник, у которого большая сторона имеет вертикальное расположение.

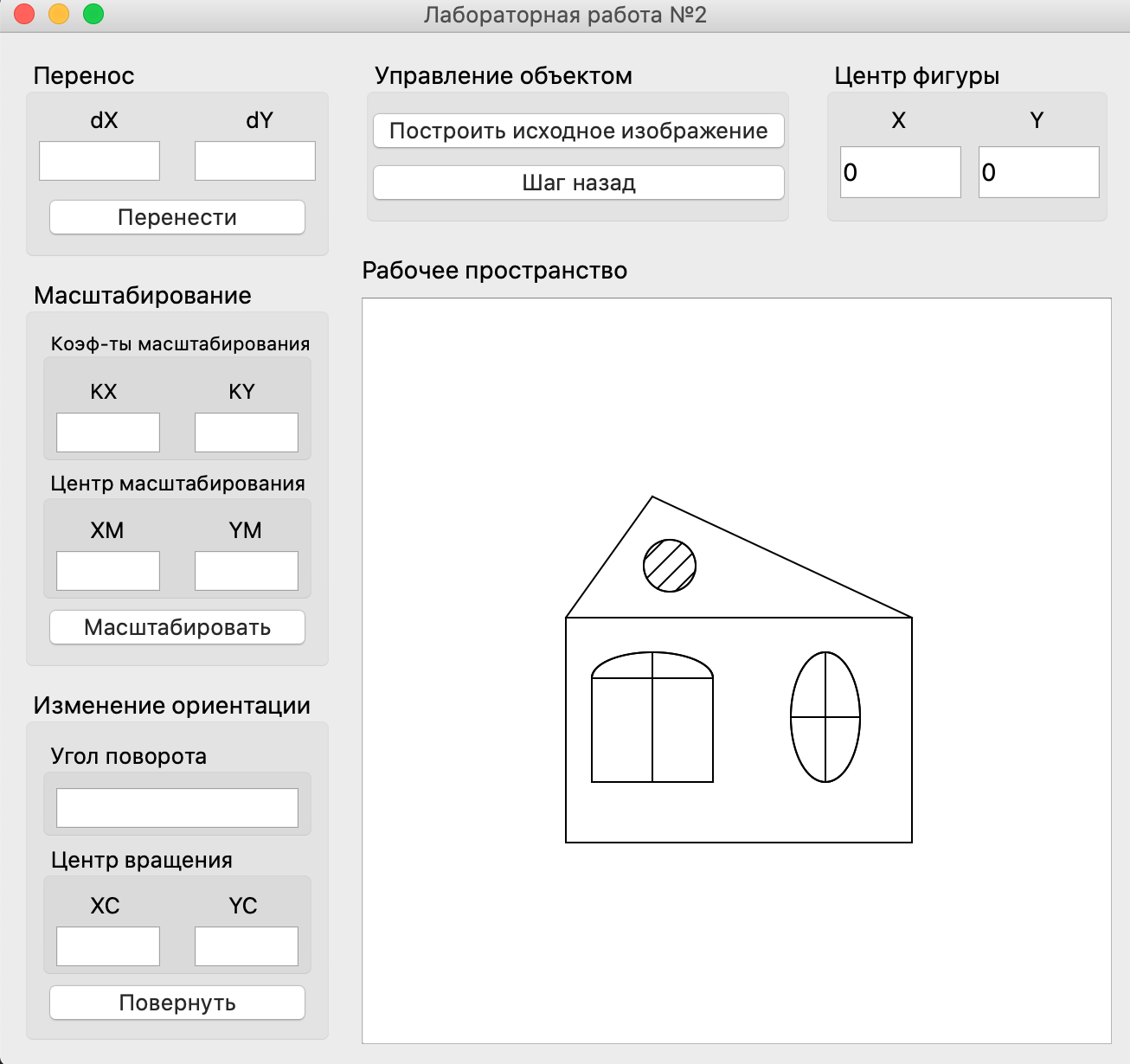
Неравномерное масштабирование окружности приводит к тому, что будет изображен эллипс. В этом случае необходимо рисовать эллипс, а не окружность. Из-за этого в целях общности целесообразно окружности вычерчивать процедурой рисования эллипса (задавая равные значения полуосей).

Коэффициенты масштабирования могут принимать любые действительные значения. При KX, KY>1 изображение увеличивается в размерах и удаляется от центра масштабирования, при KX, KY<1 изображение уменьшается в размерах и приближается к центру масштабирования.

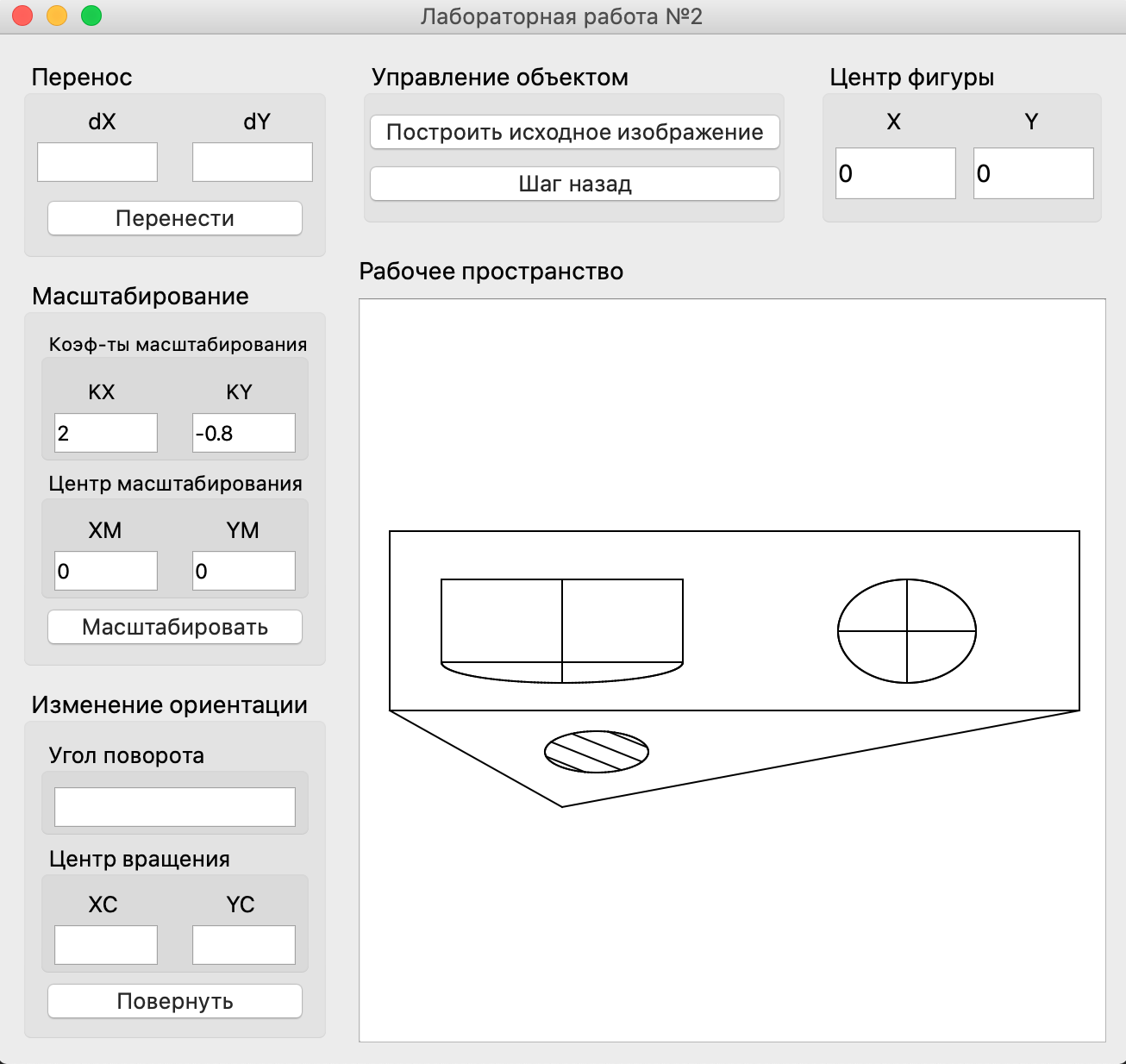
Если коэффициенты масштабирования отрицательные, то происходит отражение. При KX<0, KY>0 происходит осевое отражение относительно OY. При KX>0, KY<0 происходит осевое отражение относительно OX. При KX, KY<0 происходит отражение относительно центра симметрии.

**Пример масштабирования изображения при KX = 2, KY = -0.8 относительно центра фигуры.**

Исходное изображение:



Преобразованное изображение:

****

**Поворот изображения**

Поворот исходной точки А с координатами (X, Y) по дуге окружности с центром в точке С с координатами (Xc, Yc) на угол

Координаты (X1, Y1) повернутой точки могут быть записаны в следующем виде:

Для поворота всего рисунка прежде всего необходимо выбрать центр поворота и определить угол, на который необходимо повернуть исходное изображение. После этого в соответствии с формулами вычислить координаты всех точек рисунка и соединить вновь полученные точки линиями.

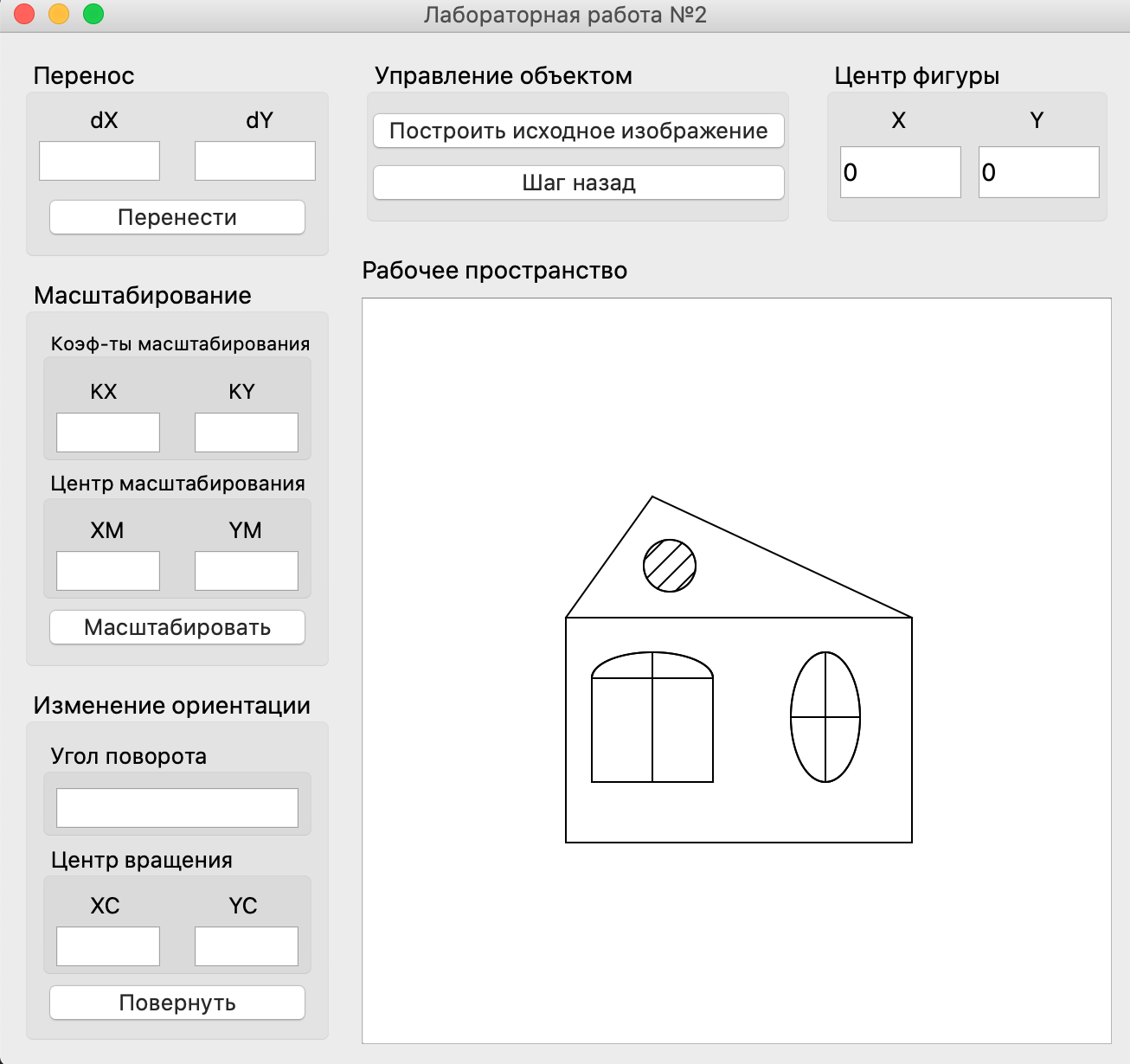
Центр поворота может быть расположен в любом месте экрана, а также за

пределами его границ.

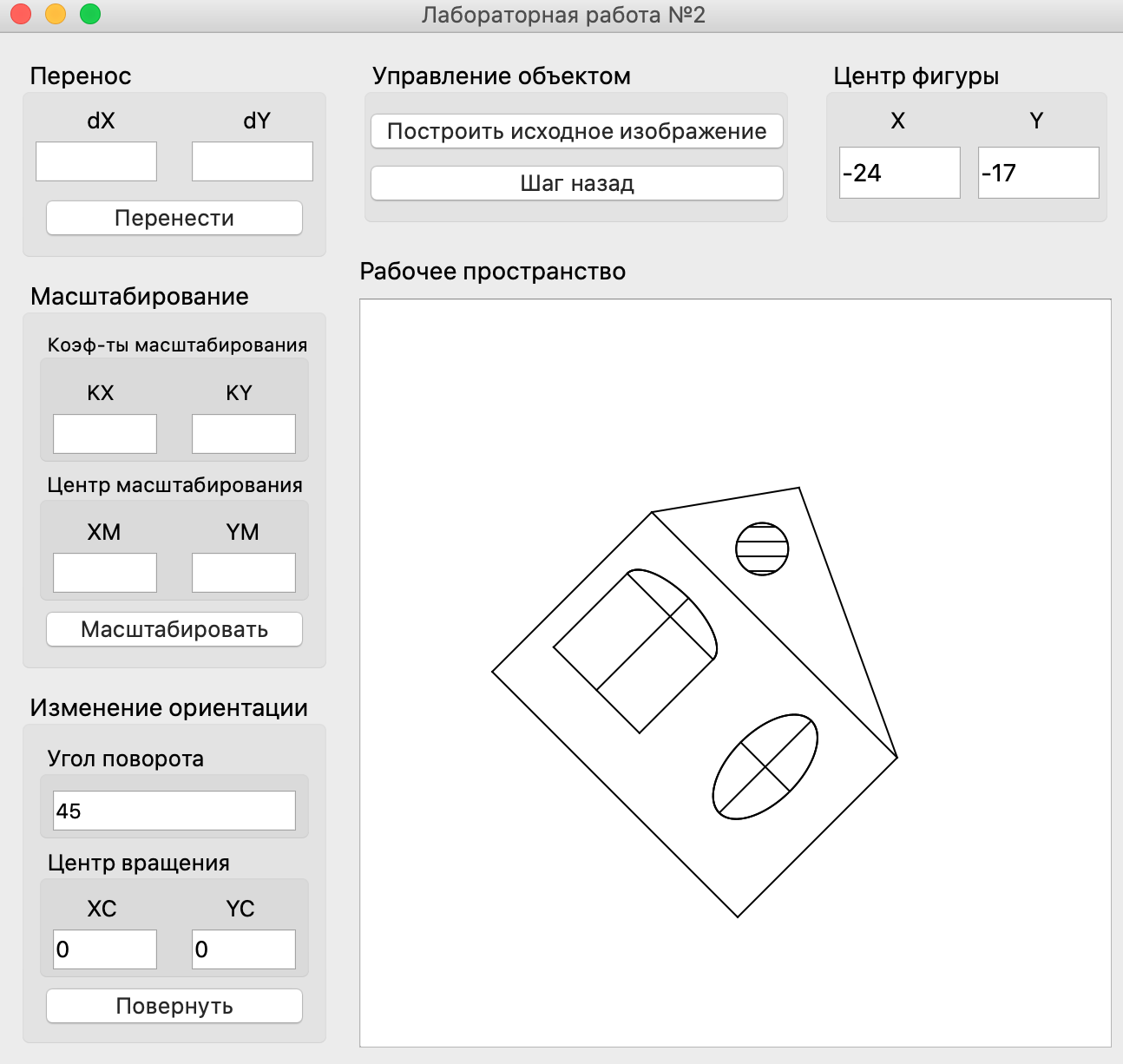
Угол поворота лежит обычно в пределах от 0 до 360, другие углы поворота также допустимы, однако поворот при этих углах эквивалентен повороту при углах из указанного диапазона.

**Пример поворота изображения на 45 градусов относительно центра фигуры.**

Исходное изображение:



Преобразованное изображение:

****

**Код программы**

**import** sys  
**from** copy **import** deepcopy  
**from** PyQt5 **import** QtWidgets, QtCore  
**from** math **import** sin, cos, pi, radians  
**import** design  
  
**class** Visual(QtWidgets.QMainWindow, design.Ui\_MainWindow):  
 **def** \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.graphicsView.scale(1, -1)  
 self.arr\_walls = [] *# Массив точек для стен дома.* self.arr\_l\_wind = [] *# Массив точек левого окна.* self.arr\_l\_ell = [] *# Массив точек левого эллипса.* self.arr\_r\_wind = [] *# Массив точек правого окна.* self.arr\_r\_ell = [] *# Массив точек правого эллипса.* self.arr\_u\_ell = [] *# Массив точек верхнего эллипса.* self.arr\_hatch = [] *# Массив штрихов у верхнего эллипса.  
  
 # Массивы для возврата на один шаг назад.* self.arr\_walls\_buff = []; self.arr\_l\_wind\_buff = []; self.arr\_l\_ell\_buff = []; self.arr\_r\_wind\_buff = []  
 self.arr\_r\_ell\_buff = []; self.arr\_u\_ell\_buff = []; self.arr\_hatch\_buff = []  
  
 *# Связи кнопок и функций.* self.pushButton\_draw\_base.clicked.connect(self.draw\_base)  
 self.pushButton\_back.clicked.connect(self.step\_back)  
 self.button\_move.clicked.connect(self.move\_func)  
 self.button\_scale.clicked.connect(self.scale\_func)  
 self.button\_rotate.clicked.connect(self.rotate\_func)  
  
 *# Вычисление начальных координат фигуры.* **def** init\_base\_data(self):  
 self.arr\_walls = [[-50, 100], [-100, 30], [-100, -100], [100, -100], [100, 30]]  
 self.arr\_l\_wind = [[-50, 10], [-85, -5], [-85, -65], [-50, -65], [-15, -65], [-15, -5], [-50, -5]]  
 self.arr\_r\_wind = [[50, 10], [30, -27.5], [50, -65], [70, -27.5], [50, -27.5]]  
 self.arr\_hatch = [[-54.6, 63.4], [-43.4, 74.6], [-53.2, 52.8], [-32.8, 73.2], [-47.2, 46.8], [-26.8, 67.2],  
 [-36.6, 45.4], [-25.4, 56.6]]  
 self.arr\_l\_ell.clear()  
 self.arr\_r\_ell.clear()  
 self.arr\_u\_ell.clear()  
 a = 35; b = 15; t = 0; xc = -50; yc = -5; h = 1 / 35  
 **while** t <= pi:  
 x = xc + a \* cos(t)  
 y = yc + b \* sin(t)  
 self.arr\_l\_ell.append([x, y])  
 t += h  
 a = 20; b = 37.5; t = 0; xc = 50; yc = -27.5; h = 1 / 37.5  
 **while** t <= 2 \* pi:  
 x = xc + a \* cos(t)  
 y = yc + b \* sin(t)  
 self.arr\_r\_ell.append([x, y])  
 t += h  
 a = 15; b = 15; t = 0; xc = -40; yc = 60; h = 1 / 15  
 **while** t <= 2 \* pi:  
 x = xc + a \* cos(t)  
 y = yc + b \* sin(t)  
 self.arr\_u\_ell.append([x, y])  
 t += h  
  
 *# Рисование фигуры.* **def** draw(self):  
 scene = QtWidgets.QGraphicsScene()  
 self.graphicsView.setScene(scene)  
 h = self.graphicsView.height()  
 w = self.graphicsView.width()  
 scene.setSceneRect(-w/2, -h/2, w - 2, h - 2)  
 **if** len(self.arr\_walls) == 0:  
 self.lineEdit\_center\_x.setText(**"0"**)  
 self.lineEdit\_center\_y.setText(**"0"**)  
 self.graphicsView.repaint()  
 **return** *# Рисование стен дома.* scene.addLine(self.arr\_walls[0][0], self.arr\_walls[0][1], self.arr\_walls[1][0], self.arr\_walls[1][1])  
 scene.addLine(self.arr\_walls[1][0], self.arr\_walls[1][1], self.arr\_walls[2][0], self.arr\_walls[2][1])  
 scene.addLine(self.arr\_walls[2][0], self.arr\_walls[2][1], self.arr\_walls[3][0], self.arr\_walls[3][1])  
 scene.addLine(self.arr\_walls[3][0], self.arr\_walls[3][1], self.arr\_walls[4][0], self.arr\_walls[4][1])  
 scene.addLine(self.arr\_walls[4][0], self.arr\_walls[4][1], self.arr\_walls[0][0], self.arr\_walls[0][1])  
 scene.addLine(self.arr\_walls[4][0], self.arr\_walls[4][1], self.arr\_walls[1][0], self.arr\_walls[1][1])  
  
 *# Рисование левого окна.* scene.addLine(self.arr\_l\_wind[1][0], self.arr\_l\_wind[1][1], self.arr\_l\_wind[2][0], self.arr\_l\_wind[2][1])  
 scene.addLine(self.arr\_l\_wind[2][0], self.arr\_l\_wind[2][1], self.arr\_l\_wind[4][0], self.arr\_l\_wind[4][1])  
 scene.addLine(self.arr\_l\_wind[4][0], self.arr\_l\_wind[4][1], self.arr\_l\_wind[5][0], self.arr\_l\_wind[5][1])  
 scene.addLine(self.arr\_l\_wind[5][0], self.arr\_l\_wind[5][1], self.arr\_l\_wind[1][0], self.arr\_l\_wind[1][1])  
 scene.addLine(self.arr\_l\_wind[0][0], self.arr\_l\_wind[0][1], self.arr\_l\_wind[3][0], self.arr\_l\_wind[3][1])  
 **for** i **in** range(len(self.arr\_l\_ell) - 1):  
 scene.addLine(self.arr\_l\_ell[i][0], self.arr\_l\_ell[i][1], self.arr\_l\_ell[i+1][0], self.arr\_l\_ell[i+1][1])  
  
 *# Рисование правого окна.* scene.addLine(self.arr\_r\_wind[0][0], self.arr\_r\_wind[0][1], self.arr\_r\_wind[2][0], self.arr\_r\_wind[2][1])  
 scene.addLine(self.arr\_r\_wind[1][0], self.arr\_r\_wind[1][1], self.arr\_r\_wind[3][0], self.arr\_r\_wind[3][1])  
 **for** i **in** range(len(self.arr\_r\_ell) - 1):  
 scene.addLine(self.arr\_r\_ell[i][0], self.arr\_r\_ell[i][1], self.arr\_r\_ell[i+1][0], self.arr\_r\_ell[i+1][1])  
  
 *# Рисование верхнего окна.* **for** i **in** range(len(self.arr\_u\_ell) - 1):  
 scene.addLine(self.arr\_u\_ell[i][0], self.arr\_u\_ell[i][1], self.arr\_u\_ell[i+1][0], self.arr\_u\_ell[i+1][1])  
 **for** i **in** range(0, 7, 2):  
 scene.addLine(self.arr\_hatch[i][0], self.arr\_hatch[i][1], self.arr\_hatch[i+1][0], self.arr\_hatch[i+1][1])  
  
 self.graphicsView.repaint()  
  
 *# Обновление информации о центре фигуры.* x = QtCore.QPointF.x(QtCore.QRectF.center(scene.itemsBoundingRect())) + 0.1  
 y = QtCore.QPointF.y(QtCore.QRectF.center(scene.itemsBoundingRect()))  
 self.lineEdit\_center\_x.setText(str(int(x)))  
 self.lineEdit\_center\_y.setText(str(int(y)))  
 self.repaint()  
  
 *# Копирование массивов для возврата на шаг назад.* **def** copy(self):  
 self.arr\_walls\_buff = deepcopy(self.arr\_walls)  
 self.arr\_l\_wind\_buff = deepcopy(self.arr\_l\_wind)  
 self.arr\_l\_ell\_buff = deepcopy(self.arr\_l\_ell)  
 self.arr\_r\_wind\_buff = deepcopy(self.arr\_r\_wind)  
 self.arr\_r\_ell\_buff = deepcopy(self.arr\_r\_ell)  
 self.arr\_u\_ell\_buff = deepcopy(self.arr\_u\_ell)  
 self.arr\_hatch\_buff = deepcopy(self.arr\_hatch)  
  
 *# Обработчик события рисования исходного изображения.* **def** draw\_base(self):  
 self.copy()  
 self.init\_base\_data()  
 self.draw()  
  
 *# Обработчик события возврата на шаг назад.* **def** step\_back(self):  
 self.arr\_walls\_buff, self.arr\_walls = self.arr\_walls, self.arr\_walls\_buff  
 self.arr\_l\_wind\_buff, self.arr\_l\_wind = self.arr\_l\_wind, self.arr\_l\_wind\_buff  
 self.arr\_l\_ell\_buff, self.arr\_l\_ell = self.arr\_l\_ell, self.arr\_l\_ell\_buff  
 self.arr\_r\_wind\_buff, self.arr\_r\_wind = self.arr\_r\_wind, self.arr\_r\_wind\_buff  
 self.arr\_r\_ell\_buff, self.arr\_r\_ell = self.arr\_r\_ell, self.arr\_r\_ell\_buff  
 self.arr\_u\_ell\_buff, self.arr\_u\_ell = self.arr\_u\_ell, self.arr\_u\_ell\_buff  
 self.arr\_hatch\_buff, self.arr\_hatch = self.arr\_hatch, self.arr\_hatch\_buff  
 self.draw()  
  
 *# Обработчик события перемещения фигуры.* **def** move\_func(self):  
 **try**:  
 dx = int(self.lineEdit\_dx.text())  
 dy = int(self.lineEdit\_dy.text())  
 **except**:  
 QtWidgets.QMessageBox.critical(self, **""**, **"Расстояние переноса должно быть целым числом."**)  
 **return** self.copy()  
 **for** i **in** range(len(self.arr\_walls)):  
 self.arr\_walls[i][0] += dx; self.arr\_walls[i][1] += dy  
 **for** i **in** range(len(self.arr\_l\_wind)):  
 self.arr\_l\_wind[i][0] += dx; self.arr\_l\_wind[i][1] += dy  
 **for** i **in** range(len(self.arr\_r\_wind)):  
 self.arr\_r\_wind[i][0] += dx; self.arr\_r\_wind[i][1] += dy  
 **for** i **in** range(len(self.arr\_hatch)):  
 self.arr\_hatch[i][0] += dx; self.arr\_hatch[i][1] += dy  
 **for** i **in** range(len(self.arr\_l\_ell)):  
 self.arr\_l\_ell[i][0] += dx; self.arr\_l\_ell[i][1] += dy  
 **for** i **in** range(len(self.arr\_r\_ell)):  
 self.arr\_r\_ell[i][0] += dx; self.arr\_r\_ell[i][1] += dy  
 **for** i **in** range(len(self.arr\_u\_ell)):  
 self.arr\_u\_ell[i][0] += dx; self.arr\_u\_ell[i][1] += dy  
 self.draw()  
  
 *# Обработчик события масштабирования фигуры.* **def** scale\_func(self):  
 **try**:  
 kx = float(self.lineEdit\_KX.text())  
 ky = float(self.lineEdit\_KY.text())  
 xm = float(self.lineEdit\_XM.text())  
 ym = float(self.lineEdit\_YM.text())  
 **except**:  
 QtWidgets.QMessageBox.critical(self, **""**, **"Коэффициенты масштабирования и координаты центра масштабирования должны быть целыми или вещественными числами"**)  
 **return** self.copy()  
 **for** i **in** range(len(self.arr\_walls)):  
 self.arr\_walls[i][0] = self.arr\_walls[i][0] \* kx + (1 - kx) \* xm  
 self.arr\_walls[i][1] = self.arr\_walls[i][1] \* ky + (1 - ky) \* ym  
 **for** i **in** range(len(self.arr\_l\_wind)):  
 self.arr\_l\_wind[i][0] = self.arr\_l\_wind[i][0] \* kx + (1 - kx) \* xm  
 self.arr\_l\_wind[i][1] = self.arr\_l\_wind[i][1] \* ky + (1 - ky) \* ym  
 **for** i **in** range(len(self.arr\_r\_wind)):  
 self.arr\_r\_wind[i][0] = self.arr\_r\_wind[i][0] \* kx + (1 - kx) \* xm  
 self.arr\_r\_wind[i][1] = self.arr\_r\_wind[i][1] \* ky + (1 - ky) \* ym  
 **for** i **in** range(len(self.arr\_hatch)):  
 self.arr\_hatch[i][0] = self.arr\_hatch[i][0] \* kx + (1 - kx) \* xm  
 self.arr\_hatch[i][1] = self.arr\_hatch[i][1] \* ky + (1 - ky) \* ym  
 **for** i **in** range(len(self.arr\_l\_ell)):  
 self.arr\_l\_ell[i][0] = self.arr\_l\_ell[i][0] \* kx + (1 - kx) \* xm  
 self.arr\_l\_ell[i][1] = self.arr\_l\_ell[i][1] \* ky + (1 - ky) \* ym  
 **for** i **in** range(len(self.arr\_r\_ell)):  
 self.arr\_r\_ell[i][0] = self.arr\_r\_ell[i][0] \* kx + (1 - kx) \* xm  
 self.arr\_r\_ell[i][1] = self.arr\_r\_ell[i][1] \* ky + (1 - ky) \* ym  
 **for** i **in** range(len(self.arr\_u\_ell)):  
 self.arr\_u\_ell[i][0] = self.arr\_u\_ell[i][0] \* kx + (1 - kx) \* xm  
 self.arr\_u\_ell[i][1] = self.arr\_u\_ell[i][1] \* ky + (1 - ky) \* ym  
 self.draw()  
  
 *# Обработчик события вращения фигуры.* **def** rotate\_func(self):  
 **try**:  
 deg = float(self.lineEdit\_dg.text())  
 xc = float(self.lineEdit\_XC.text())  
 yc = float(self.lineEdit\_YC.text())  
 **except**:  
 QtWidgets.QMessageBox.critical(self, **""**, **"Угол поворота и координаты центра поворота должны быть целыми или вещественными числами."**)  
 **return** self.copy()  
 SIN = sin(radians(deg))  
 COS = cos(radians(deg))  
 **for** i **in** range(len(self.arr\_walls)):  
 x1 = xc + (self.arr\_walls[i][0] - xc) \* COS + (self.arr\_walls[i][1] - yc) \* SIN  
 self.arr\_walls[i][1] = yc + (self.arr\_walls[i][1] - yc) \* COS - (self.arr\_walls[i][0] - xc) \* SIN  
 self.arr\_walls[i][0] = x1  
 **for** i **in** range(len(self.arr\_l\_wind)):  
 x1 = xc + (self.arr\_l\_wind[i][0] - xc) \* COS + (self.arr\_l\_wind[i][1] - yc) \* SIN  
 self.arr\_l\_wind[i][1] = yc + (self.arr\_l\_wind[i][1] - yc) \* COS - (self.arr\_l\_wind[i][0] - xc) \* SIN  
 self.arr\_l\_wind[i][0] = x1  
 **for** i **in** range(len(self.arr\_r\_wind)):  
 x1 = xc + (self.arr\_r\_wind[i][0] - xc) \* COS + (self.arr\_r\_wind[i][1] - yc) \* SIN  
 self.arr\_r\_wind[i][1] = yc + (self.arr\_r\_wind[i][1] - yc) \* COS - (self.arr\_r\_wind[i][0] - xc) \* SIN  
 self.arr\_r\_wind[i][0] = x1  
 **for** i **in** range(len(self.arr\_hatch)):  
 x1 = xc + (self.arr\_hatch[i][0] - xc) \* COS + (self.arr\_hatch[i][1] - yc) \* SIN  
 self.arr\_hatch[i][1] = yc + (self.arr\_hatch[i][1] - yc) \* COS - (self.arr\_hatch[i][0] - xc) \* SIN  
 self.arr\_hatch[i][0] = x1  
 **for** i **in** range(len(self.arr\_l\_ell)):  
 x1 = xc + (self.arr\_l\_ell[i][0] - xc) \* COS + (self.arr\_l\_ell[i][1] - yc) \* SIN  
 self.arr\_l\_ell[i][1] = yc + (self.arr\_l\_ell[i][1] - yc) \* COS - (self.arr\_l\_ell[i][0] - xc) \* SIN  
 self.arr\_l\_ell[i][0] = x1  
 **for** i **in** range(len(self.arr\_r\_ell)):  
 x1 = xc + (self.arr\_r\_ell[i][0] - xc) \* COS + (self.arr\_r\_ell[i][1] - yc) \* SIN  
 self.arr\_r\_ell[i][1] = yc + (self.arr\_r\_ell[i][1] - yc) \* COS - (self.arr\_r\_ell[i][0] - xc) \* SIN  
 self.arr\_r\_ell[i][0] = x1  
 **for** i **in** range(len(self.arr\_u\_ell)):  
 x1 = xc + (self.arr\_u\_ell[i][0] - xc) \* COS + (self.arr\_u\_ell[i][1] - yc) \* SIN  
 self.arr\_u\_ell[i][1] = yc + (self.arr\_u\_ell[i][1] - yc) \* COS - (self.arr\_u\_ell[i][0] - xc) \* SIN  
 self.arr\_u\_ell[i][0] = x1  
 self.draw()  
  
*# Точка входа.***def** main():  
 app = QtWidgets.QApplication(sys.argv)  
 window = Visual()  
 window.show()  
 app.exec\_()  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 main()