Министерство образования Республики Беларусь

Белорусский Национальный Технический Университет

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных

систем и технологий

**КУРСОВАЯ РАБОТА**

по дисциплине «**Разработка приложений в визуальных средах**»

на тему: «**Симуляция движения шестерён**» 1

Выполнил

студент гр.10701219 Колосов А.А.

Принял

доц. Гурский Н.Н.

Минск 2021

Белорусский национальный технический университет

Кафедра программного обеспечения информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту (работе)**

**по дисциплине «**Разработка приложений в визуальных средах»

Тема: «Симуляция движения шестерён»

**Исполнитель**: Колосов А.А. (фамилия, инициалы)

**Студент 2 курса 10701219 группы**

**Руководитель**: доц. Гурский Н.Н. (фамилия, инициалы)

Минск 2021

**Содержание**

[ВВЕДЕНИЕ 4](#_Toc72080328)

[1 МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ 5](#_Toc72080329)

[2 ОПИСАНИЕ ПРОГРАММЫ 6](#_Toc72080330)

[2.1 Структурная схема программы 6](#_Toc72080331)

[2.2 Описание разработанного класса 6](#_Toc72080332)

[2.3 Основные возможности программы 7](#_Toc72080333)

[3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 8](#_Toc72080334)

[4 МЕТОДИКА ИСПЫТАНИЙ 11](#_Toc72080335)

[ЗАКЛЮЧЕНИЕ 15](#_Toc72080336)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 16](#_Toc72080337)

# ВВЕДЕНИЕ

Трёхмерная графика (3D Graphics, Три измерения изображения, 3 Dimensions, рус. 3 измерения) — раздел компьютерной графики, совокупность приемов и инструментов (как программных, так и аппаратных), предназначенных для изображения объёмных объектов. Больше всего применяется для создания изображений на плоскости экрана или листа печатной продукции в архитектурной визуализации, кинематографе, телевидении, компьютерных играх, печатной продукции, а также в науке и промышленности.

Трёхмерное изображение на плоскости отличается от двумерного тем, что включает построение геометрической проекции трёхмерной модели сцены на плоскость (например, экран компьютера) с помощью специализированных программ. При этом модель может, как соответствовать объектам из реального мира (автомобили, здания, ураган, астероид), так и быть полностью абстрактной (проекция четырёхмерного фрактала).

Для получения трёхмерного изображения на плоскости требуются следующие шаги:

1. моделирование — создание трёхмерной математической модели сцены и объектов в ней.
2. рендеринг (визуализация) — построение проекции в соответствии с выбранной физической моделью.
3. вывод полученного изображения на устройство вывода — дисплей или принтер.

# 1 МАТЕМАТИЧЕСКАЯ ФОРМУЛИРОВКА ЗАДАЧИ

Зубчатое колесо или Шестерня – это важнейшая деталь, которая применяется в механизмах зубчатой передачи и выполняет основную функцию - передает вращательное движения между валами, при помощи зацепление с зубьями соседней шестерни. Выглядит шестерня как диск с конической или цилиндрической поверхностью на которой на равном расстоянии расположены зубья. В зубчатой передаче шестерней называют малое зубчатое колесо с небольшим количеством зубьев, а большое - зубчатым колесом. В случае применения пары шестерен с одинаковым количеством зубьев, ведущую называют шестерней, а ведомую – зубчатым колесом. Но чаще всего все зубчатые колеса и малые и большие называют шестернями (шестеренками).

# 2 ОПИСАНИЕ ПРОГРАММЫ

# 2.1 Структурная схема программы

Структурно программа состоит из главной формы и второстепенных форм, одна из которых служит для показа изображения, а вторая является оригинальной заставкой (см. рисунок 2.1).

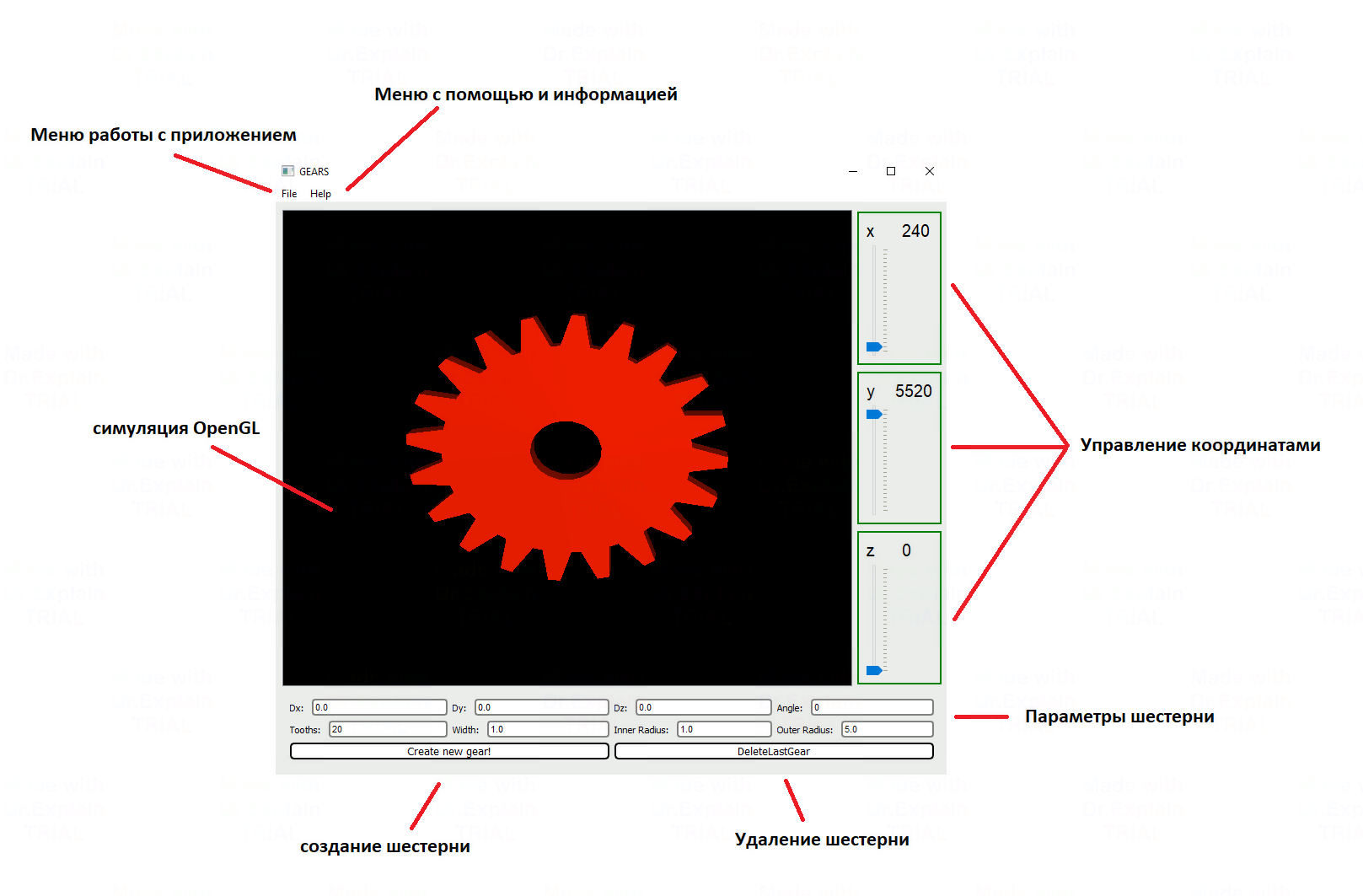


Рисунок 2.1 – Структурная схема связей модулей программы.

# 2.2 Описание разработанного класса

В результате анализа предметной области для её описания и моделирования был разработан класс статический класс GearMaker, инкапсулирующий поля, методы и свойства, предназначенные для создания точек шестерни.

На основе параметров, переданных пользователем, происходит расчёт точек шестерни и создаётся сам объект шестерни, содержащий в себе описание характеристик.

self.reflectance = reflectance – световые характеристики

self.innerRadius = innerRadius – внутренний радиус шестерни

self.outerRadius = outerRadius – внешний радиус шестерни

self.thickness = thickness – толщина шестерни

self.toothSize = toothSize – размер зубов

self.toothCount = toothCount – кол-во зубов

self.points = [] – точки для OpenGL

outputAngle = 0 – угол отрисовки

dx = 0

dy = 0 – координаты на осях x, y, z в величинах OpenGL

dz = 0

# 2.3 Основные возможности программы

Для взаимодействия с программой пользователю предоставлена главное окно MainGearsWindow, состоящая из форм UserParamsInput, AxisSlider, GLSimulationWidget. Функционал, который предоставляет форма:

* Поля для ввода величин шестерни.
* Вызов окна помощи.
* Выбор координат обзора

Пользователь может использовать кнопку «Create New Gear!», которая вызывает метод, создающий новую шестерню на основе данных пользователя

Пользователь может использовать кнопку «Delete last gear» которая вызывает метод, удаляющий последнюю шестерню.

При menu «Help» вызывается help-файл с руководством пользователя и описанием задачи.

# 3 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска приложения необходимо вызвать файл MainGearsWindow.py. После вызова приложения появится оригинальная заставка, изображённая на рисунке 3.1.

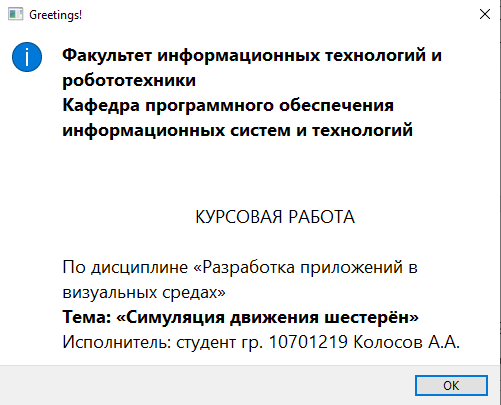


Рисунок 3.1 – Приветственное окно

После заставки появляется главное окно программы (см. рисунок 3.2).

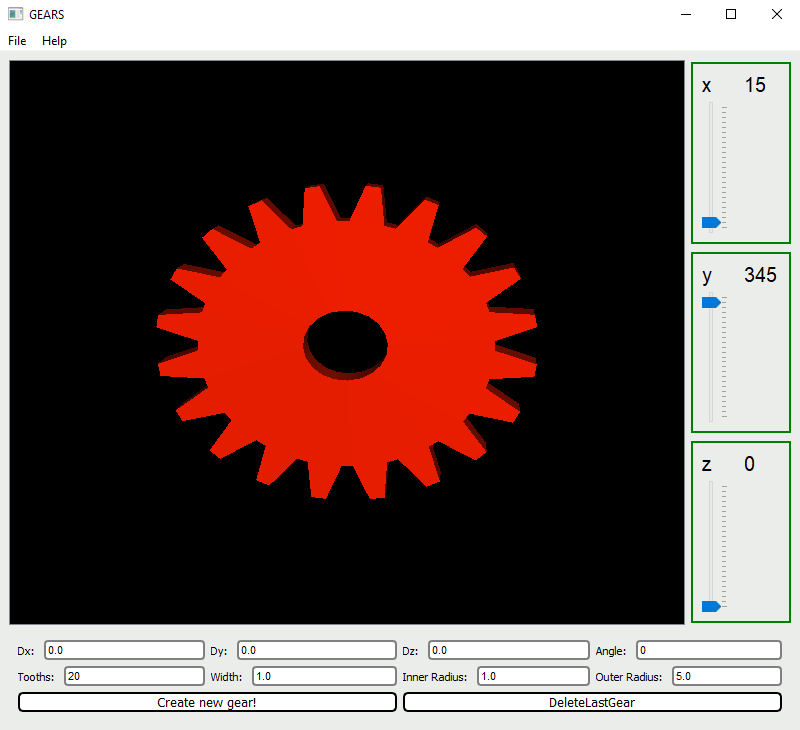


Рисунок 3.2 - Главное окно программы

Главная форма имеет:

* Главное меню (см. рисунок 3.3)
* Области ввода данных
* Управляющие кнопки (см. рисунок 3.8)



Рисунок 3.3 – Пункты главного меню

Пункт меню «О программе» выводит информацию о создателе программы.

Пункт меню «About» вызывает help-файл с небольшим руководством пользователя (см. рисунок 3.4)

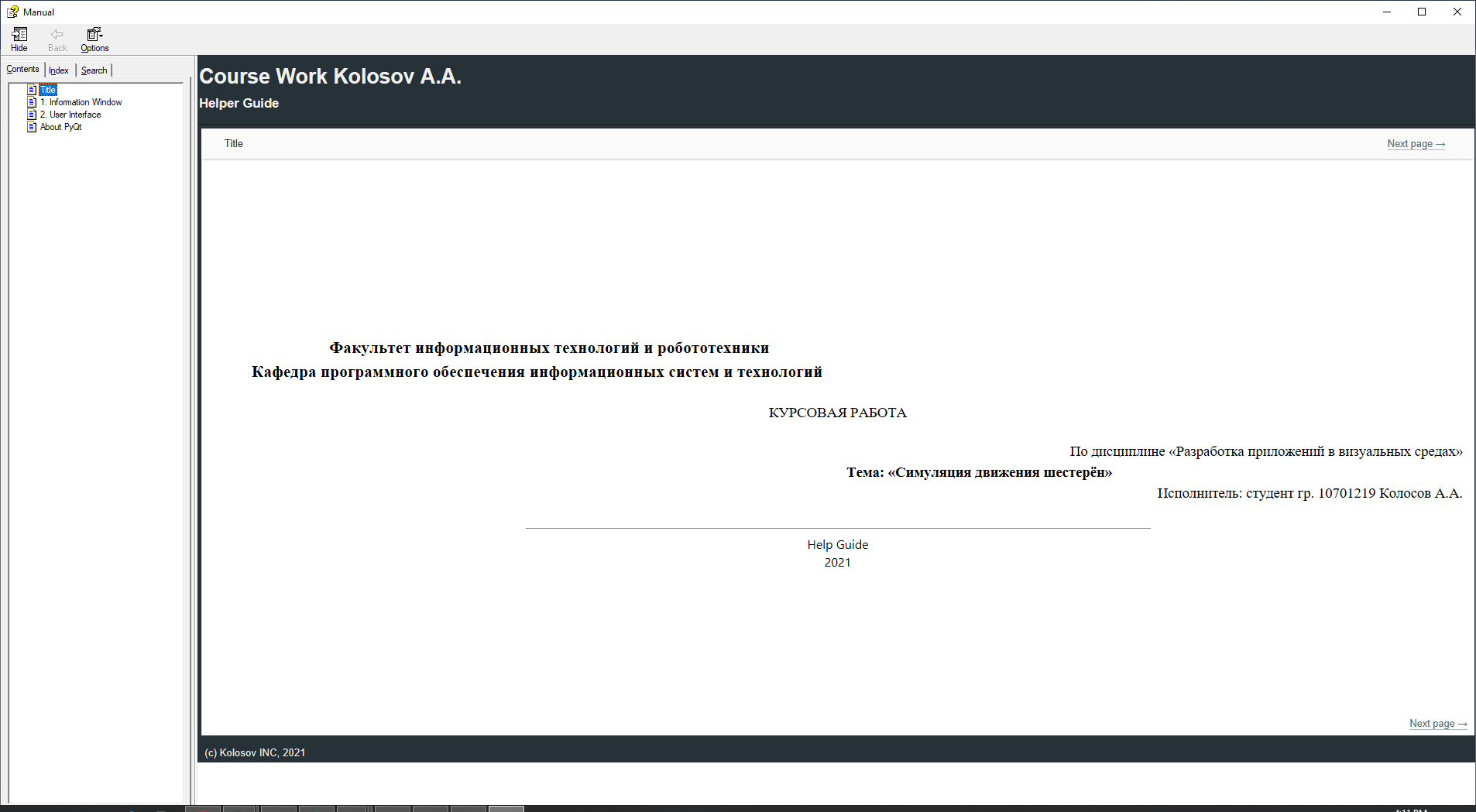


Рисунок 3.4 – Руководство пользователя, вызываемое клавишей «Помощь»

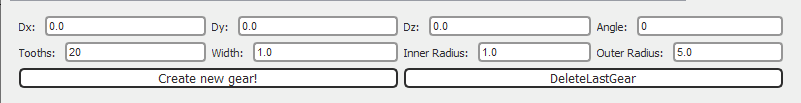


Рисунок 3.5 – управляющие кнопки

# 4 МЕТОДИКА ИСПЫТАНИЙ

Целью проведения испытаний является проверка работоспособности (надежности) программы при различных условиях её функционирования.

Программа должна обеспечивать корректность ввода исходных данных (путем осуществления соответствующих проверок и информирования пользователя о возникших неточностях в работе), а также получение непротиворечивого результата.

Для обеспечения нормальной работы программы требуется наличие необходимых динамических библиотек, а также приложений Helper.ch.

Первое испытание будет проведено при корректных данных (см. рисунок 4.1, 4.2).

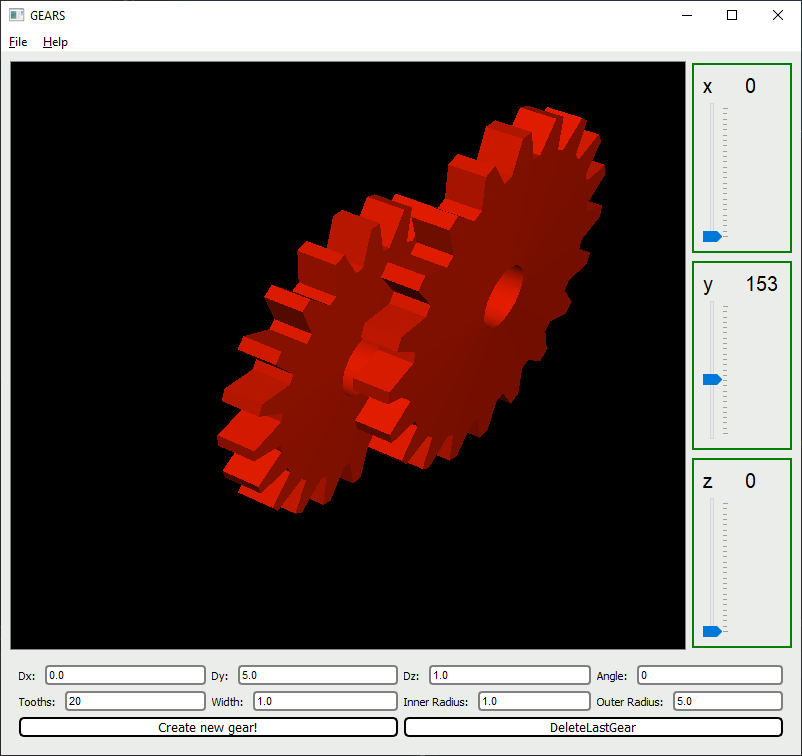


Рисунок 4.1 – Ввод корректных данных

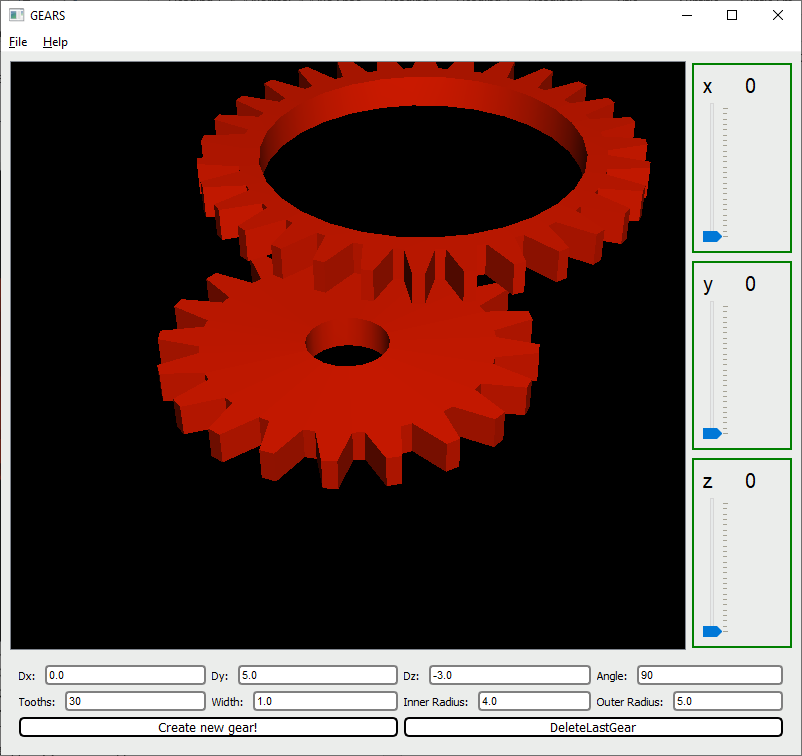


Рисунок 4.2 – Вывод второй формы в случае корректных данных.

Далее проведем проверку с некорректными параметрами, например, неверными параметрами, которые должен быть больше нуля (см. рисунок 4.3, 4.4).

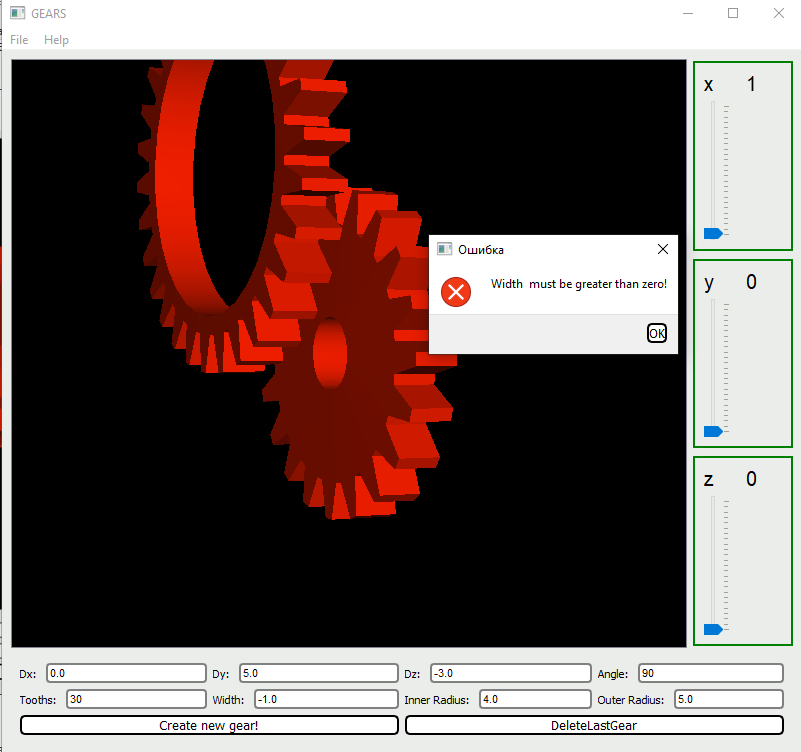


Рисунок 4.3 – Окно предупреждения о том, что ширина должна быть больше нуля.

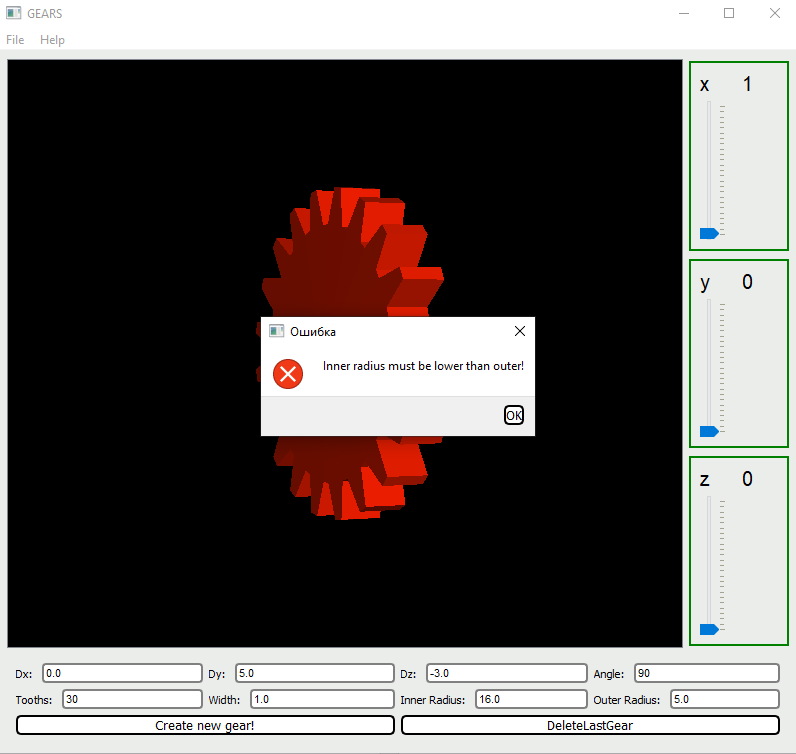


Рисунок 4.4 – Окно предупреждения о том, что внутренний радиус должен быть больше внешнего радиуса.

В результате тестирования приложения ошибок обнаружено не было. Следует считать, что в целом программа протестирована, отвечает поставленным требованием и работоспособна.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы был приобретен опыт разработки объектно-ориентированных программ, закреплен опыт, полученный при прохождении курса «Разработка приложений в визуальных средах», было изучено взаимодействие с библиотекой OpenGL При помощи языка программирования Python и библиотеке PyQt5 для разработки приложений с графическим интерфейсом.

Программа имеет удобный для пользователя интерфейс, различные формы вывода информации. Программа может использоваться для симуляций. Программа является расширяемой, что в бедующем позволит расширить её функционал.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Kessenich J., Sellers G., Shreiner D. - OpenGL Programming Guide. The Official Guide to Learning OpenGL – 2017
2. Информация по разработке - https://wikipedia.org/
3. Дополнительные сведения по языку – https://habr.com/
4. Дополнительные сведения по языку – <https://metanit.com/>
5. Пол Бэрри. Изучаем программирование на Python, стр.611
6. Эрик Мэтиз. Изучаем Python. Программирование игр, визуализация данных, веб-приложения, стр. 496
7. Лучано Рамальо. Python. К вершинам мастерства, 2015. - 770c
8. Даг Хеллман. Стандартная библиотека Python 3. Справочник с примерами, 2019. -1235 с.

**ПРИЛОЖЕНИЕ A**

**Файл MainGearsWindow.py**

import os

import sys

from PyQt5.QtCore import QSize, Qt

from PyQt5.QtWidgets import (QAction, QApplication, QLabel,

QMainWindow, QMessageBox, QScrollArea,

QSizePolicy, QWidget, QHBoxLayout, QVBoxLayout)

from Modules.GearMaker import GearMaker

from Widgets.GearParamsInput import GearParamsInput

from Widgets.GlSimulationWidget import GLWidget

from Widgets.AxisSlider import AxisSlider

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super(MainWindow, self).\_\_init\_\_()

centralWidget = QWidget()

self.setCentralWidget(centralWidget)

self.glWidget = GLWidget()

self.pixmapLabel = QLabel()

self.glWidgetArea = QScrollArea()

self.glWidgetArea.setWidget(self.glWidget)

self.glWidgetArea.setWidgetResizable(True)

self.glWidgetArea.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)

self.glWidgetArea.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOff)

self.glWidgetArea.setSizePolicy(QSizePolicy.Ignored,

QSizePolicy.Ignored)

self.glWidgetArea.setMinimumSize(50, 50)

xSlider = AxisSlider.createSlider(self.glWidget.xRotationChanged,

self.glWidget.setXRotation, "x")

ySlider = AxisSlider.createSlider(self.glWidget.yRotationChanged,

self.glWidget.setYRotation, "y")

zSlider = AxisSlider.createSlider(self.glWidget.zRotationChanged,

self.glWidget.setZRotation, "z")

self.createActions()

self.createMenus()

centralWidgetLayout = QHBoxLayout()

hbox1 = QHBoxLayout()

hbox1.addWidget(self.glWidgetArea)

hbox2 = QVBoxLayout()

hbox2.addSpacing(2)

hbox2.addWidget(xSlider)

hbox2.addSpacing(2)

hbox2.addWidget(ySlider)

hbox2.addSpacing(2)

hbox2.addWidget(zSlider)

hbox2.addSpacing(2)

centralWidgetLayout.addLayout(hbox1)

centralWidgetLayout.addLayout(hbox2)

verticalLayout = QVBoxLayout()

verticalLayout.addLayout(centralWidgetLayout)

self.paramsInput = GearParamsInput()

self.paramsInput.createPushButton.clicked.connect(self.createNewGear)

self.paramsInput.deleteLastGearPushButton.clicked.connect(self.deleteLastGear)

verticalLayout.addWidget(self.paramsInput)

centralWidget.setLayout(verticalLayout)

xSlider.slider.setValue(15 \* 16)

ySlider.slider.setValue(345 \* 16)

zSlider.slider.setValue(0 \* 16)

self.setWindowTitle("GEARS")

self.resize(800, 700)

self.showInfo()

def showInfo(self):

text = """<p style="font-size:18px;"><b>Факультет информационных технологий и робототехники

<br/>Кафедра программного обеспечения информационных систем и технологий</b>

<br/><br/><center>КУРСОВАЯ РАБОТА</center><br/>

По дисциплине «Разработка приложений в визуальных средах»

<br/><b>Тема: «Симуляция движения шестерён»</b><br/>

Исполнитель: студент гр. 10701219 Колосов А.А.</p>

"""

QMessageBox.information(self, "Greetings!", text, QMessageBox.Ok)

def createNewGear(self):

reflectance1 = (0.8, 0.1, 0.0, 1.0)

try:

toothAmount = int(self.paramsInput.toothAmount\_lineEdit.text())

innerRadius = float(self.paramsInput.innerRadius\_lineEdit.text())

outerRadius = float(self.paramsInput.outerRadius\_lineEdit.text())

width = float(self.paramsInput.width\_lineEdit.text())

if toothAmount < 5:

raise Exception("Tooth amount lower than 5!")

if innerRadius <= 0:

raise Exception("Inner radius must be greater than zero!")

if outerRadius <= 0:

raise Exception("Outer radius must be greater than zero!")

if width <= 0:

raise Exception("Width must be greater than zero!")

if innerRadius > outerRadius:

raise Exception("Inner radius must be lower than outer!")

gear = GearMaker.makeGear(

reflectance1,

innerRadius,

outerRadius,

width,

1.0,

toothAmount)

gear.dx = float(self.paramsInput.dx\_lineEdit.text())

gear.dy = float(self.paramsInput.dy\_lineEdit.text())

gear.dz = float(self.paramsInput.dz\_lineEdit.text())

gear.outputAngle = float(self.paramsInput.angle\_lineEdit.text())

self.glWidget.gears.append(gear)

self.update()

except Exception as ex:

QMessageBox.critical(self, "Ошибка ", str(ex), QMessageBox.Ok)

def deleteLastGear(self):

if len(self.glWidget.gears) > 0:

self.glWidget.gears.pop()

self.update()

def about(self):

os.system("Helper.chm")

def createActions(self):

self.exitAct = QAction("E&xit", self, shortcut="Ctrl+Q",

triggered=self.close)

self.aboutAct = QAction("&Help", self, triggered=self.about)

self.helpAct = QAction("&About", self, triggered=self.showInfo)

self.aboutQtAct = QAction("About &Qt", self,

triggered=QApplication.instance().aboutQt)

def createMenus(self):

self.fileMenu = self.menuBar().addMenu("&File")

self.fileMenu.addSeparator()

self.fileMenu.addAction(self.exitAct)

self.helpMenu = self.menuBar().addMenu("&Help")

self.helpMenu.addAction(self.aboutAct)

self.helpMenu.addAction(self.helpAct)

self.helpMenu.addAction(self.aboutQtAct)

def setPixmap(self, pixmap):

self.pixmapLabel.setPixmap(pixmap)

size = pixmap.size()

if size - QSize(1, 0) == self.pixmapLabelArea.maximumViewportSize():

size -= QSize(1, 0)

self.pixmapLabel.resize(size)

if \_\_name\_\_ == '\_\_main\_\_':

app = QApplication(sys.argv)

style = """ QLineEdit:hover{border-radius:4px;

border-color:rgb(0,128,0);

background-color:rgb(220,220,220);}

QLineEdit{background: white;

font:11px Arial;

border: 2px solid grey;

border-radius:4px;}

QMainWindow{background-color: rgb(235, 237, 235);}

QPushButton{

border-radius: 5px;

border: 2px solid black;

background: white;;

font: 13px Arial italic;

}

QPushButton:hover{

border-radius: 5px;

border-color: rgb(0,128,0);

background-color: rgb(210,210,210)}

QPushButton:pressed{

border-radius: 5px;

border-color: rgb(0,128,0);

background-color: rgb(240,240,240);

}

QGroupBox{

border: 2px solid green;

}

"""

mainWin = MainWindow()

mainWin.setStyleSheet(style)

mainWin.show()

sys.exit(app.exec\_())

**Файл AxisSlider.py**

from PyQt5.QtCore import Qt

from PyQt5.QtGui import QFont

from PyQt5.QtWidgets import QSlider, QLabel, QHBoxLayout, QVBoxLayout, QGroupBox

class AxisSlider(QGroupBox):

def \_\_init\_\_(self, axisName: str):

super().\_\_init\_\_()

self.textLabel = QLabel()

self.setFont(QFont("Arial", 15, 1))

self.textLabel.setText('0')

self.slider = QSlider(Qt.Vertical)

text = QLabel()

text.setText(axisName)

textHbox = QHBoxLayout()

textHbox.addWidget(text)

textHbox.addWidget(self.textLabel)

vbox = QVBoxLayout()

vbox.addLayout(textHbox)

vbox.addWidget(self.slider)

self.setMaximumWidth(100)

self.setLayout(vbox)

def valueChangedEvent(self, value):

self.textLabel.setText(str(value // 16))

self.update()

@staticmethod

def createSlider(changedSignal, setterSlot, axisName):

axisSlider = AxisSlider(axisName)

axisSlider.slider.setRange(0, 360 \* 16)

axisSlider.slider.setSingleStep(16)

axisSlider.slider.setPageStep(15 \* 16)

axisSlider.slider.setTickInterval(15 \* 16)

axisSlider.slider.setTickPosition(QSlider.TicksRight)

axisSlider.slider.valueChanged.connect(setterSlot)

axisSlider.slider.valueChanged.connect(axisSlider.valueChangedEvent)

changedSignal.connect(axisSlider.slider.setValue)

return axisSlider

**Файл GearParamsInput.py**

from PyQt5.QtWidgets import QWidget, QLineEdit, QLabel, QHBoxLayout, QPushButton, QVBoxLayout

class GearParamsInput(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

toothAmount\_hbox = QHBoxLayout()

self.toothAmount\_label = QLabel("Tooths: ")

self.toothAmount\_lineEdit = QLineEdit("20")

toothAmount\_hbox.addWidget(self.toothAmount\_label)

toothAmount\_hbox.addWidget(self.toothAmount\_lineEdit)

self.width\_label = QLabel("Width: ")

self.width\_lineEdit = QLineEdit("1.0")

width\_hbox = QHBoxLayout()

width\_hbox.addWidget(self.width\_label)

width\_hbox.addWidget(self.width\_lineEdit)

self.innerRadius\_label = QLabel("Inner Radius: ")

self.innerRadius\_lineEdit = QLineEdit("1.0")

innerRadius\_hbox = QHBoxLayout()

innerRadius\_hbox.addWidget(self.innerRadius\_label)

innerRadius\_hbox.addWidget(self.innerRadius\_lineEdit)

self.outerRadius\_label = QLabel("Outer Radius: ")

self.outerRadius\_lineEdit = QLineEdit("5.0")

outerRadius\_hbox = QHBoxLayout()

outerRadius\_hbox.addWidget(self.outerRadius\_label)

outerRadius\_hbox.addWidget(self.outerRadius\_lineEdit)

self.dx\_label = QLabel("Dx: ")

self.dx\_lineEdit = QLineEdit("0.0")

dx\_hbox = QHBoxLayout()

dx\_hbox.addWidget(self.dx\_label)

dx\_hbox.addWidget(self.dx\_lineEdit)

self.dy\_label = QLabel("Dy: ")

self.dy\_lineEdit = QLineEdit("0.0")

dy\_hbox = QHBoxLayout()

dy\_hbox.addWidget(self.dy\_label)

dy\_hbox.addWidget(self.dy\_lineEdit)

self.dz\_label = QLabel("Dz: ")

self.dz\_lineEdit = QLineEdit("0.0")

dz\_hbox = QHBoxLayout()

dz\_hbox.addWidget(self.dz\_label)

dz\_hbox.addWidget(self.dz\_lineEdit)

self.angle\_label = QLabel("Angle: ")

self.angle\_lineEdit = QLineEdit("0")

angle\_hbox = QHBoxLayout()

angle\_hbox.addWidget(self.angle\_label)

angle\_hbox.addWidget(self.angle\_lineEdit)

secondLevelHbox = QHBoxLayout()

secondLevelHbox.addLayout(dx\_hbox)

secondLevelHbox.addLayout(dy\_hbox)

secondLevelHbox.addLayout(dz\_hbox)

secondLevelHbox.addLayout(angle\_hbox)

firstLevelHbox = QHBoxLayout()

firstLevelHbox.addLayout(toothAmount\_hbox)

firstLevelHbox.addLayout(width\_hbox)

firstLevelHbox.addLayout(innerRadius\_hbox)

firstLevelHbox.addLayout(outerRadius\_hbox)

self.createPushButton = QPushButton("Create new gear!")

self.deleteLastGearPushButton = QPushButton("DeleteLastGear")

buttonsHbox = QHBoxLayout()

buttonsHbox.addWidget(self.createPushButton)

buttonsHbox.addWidget(self.deleteLastGearPushButton)

mainVerticalLayout = QVBoxLayout()

mainVerticalLayout.addLayout(secondLevelHbox)

mainVerticalLayout.addLayout(firstLevelHbox)

mainVerticalLayout.addLayout(buttonsHbox)

self.setLayout(mainVerticalLayout)

Файл GlSimulationWidget.py

import OpenGL.GL as gl

from PyQt5.QtCore import pyqtSignal, Qt, QTimer

from PyQt5.QtWidgets import (QOpenGLWidget)

from Modules.GearMaker import GearMaker

class GLWidget(QOpenGLWidget):

xRotationChanged = pyqtSignal(int)

yRotationChanged = pyqtSignal(int)

zRotationChanged = pyqtSignal(int)

def \_\_init\_\_(self, parent=None):

super(GLWidget, self).\_\_init\_\_(parent)

self.gears = []

self.gear1 = 0

self.gear2 = 0

self.gear3 = 0

self.xRot = 0

self.yRot = 0

self.zRot = 0

self.gear1Rot = 0

timer = QTimer(self)

timer.timeout.connect(self.advanceGears)

timer.start(20)

def setXRotation(self, angle):

self.normalizeAngle(angle)

if angle != self.xRot:

self.xRot = angle

self.xRotationChanged.emit(angle)

self.update()

def setYRotation(self, angle):

self.normalizeAngle(angle)

if angle != self.yRot:

self.yRot = angle

self.yRotationChanged.emit(angle)

self.update()

def setZRotation(self, angle):

self.normalizeAngle(angle)

if angle != self.zRot:

self.zRot = angle

self.zRotationChanged.emit(angle)

self.update()

def initializeGL(self):

lightPos = (5.0, 5.0, 10.0, 1.0)

reflectance1 = (0.8, 0.1, 0.0, 1.0)

gl.glLightfv(gl.GL\_LIGHT0, gl.GL\_POSITION, lightPos)

gl.glEnable(gl.GL\_LIGHTING)

gl.glEnable(gl.GL\_LIGHT0)

gl.glEnable(gl.GL\_DEPTH\_TEST)

self.gears.append(GearMaker.makeGear(reflectance1, 1.0, 4.0, 1.0, 1.0, 20))

gl.glEnable(gl.GL\_NORMALIZE)

gl.glClearColor(0.0, 0.0, 0.0, 1.0)

def paintGL(self):

gl.glClear(gl.GL\_COLOR\_BUFFER\_BIT | gl.GL\_DEPTH\_BUFFER\_BIT)

gl.glPushMatrix()

gl.glRotated(self.xRot / 16.0, 1.0, 1.0, 0.0)

gl.glRotated(self.yRot / 16.0, 0.0, 1.0, 0.0)

gl.glRotated(self.zRot / 16.0, 0.0, 0.0, 1.0)

for gear in self.gears:

self.drawGear(gear.points, gear.dx, gear.dy, gear.dz, self.gear1Rot / 16.0)

if gear.outputAngle > 0:

gl.glRotated(+gear.outputAngle, 1.0, 1.0, 0.0)

gl.glPopMatrix()

def resizeGL(self, width, height):

side = min(width, height)

if side < 0:

return

gl.glViewport((width - side) // 2, (height - side) // 2, side, side)

gl.glMatrixMode(gl.GL\_PROJECTION)

gl.glLoadIdentity()

gl.glFrustum(-1.0, +1.0, -1.0, 1.0, 5.0, 60.0)

gl.glMatrixMode(gl.GL\_MODELVIEW)

gl.glLoadIdentity()

gl.glTranslated(0.0, 0.0, -40.0)

def mousePressEvent(self, event):

self.lastPos = event.pos()

def mouseMoveEvent(self, event):

dx = event.x() - self.lastPos.x()

dy = event.y() - self.lastPos.y()

if event.buttons() & Qt.LeftButton:

self.setXRotation(self.xRot + 8 \* dy)

self.setYRotation(self.yRot + 8 \* dx)

elif event.buttons() & Qt.RightButton:

self.setXRotation(self.xRot + 8 \* dy)

self.setZRotation(self.zRot + 8 \* dx)

self.lastPos = event.pos()

def advanceGears(self):

self.gear1Rot += 2 \* 16

self.update()

def xRotation(self):

return self.xRot

def yRotation(self):

return self.yRot

def zRotation(self):

return self.zRot

@staticmethod

def drawGear(gear, dx, dy, dz, angle):

gl.glPushMatrix()

gl.glTranslated(dx, dy, dz)

gl.glRotated(angle, 0.0, 0.0, 1.0)

gl.glCallList(gear)

gl.glPopMatrix()

@staticmethod

def normalizeAngle(angle):

while angle < 0:

angle += 360 \* 16

while angle > 360 \* 16:

angle -= 360 \* 16

**Файл GearMaker.py**

import math

import OpenGL.GL as gl

from Models.Gear import Gear

class GearMaker:

@staticmethod

def makeGear(reflectance, innerRadius, outerRadius, thickness, toothSize, toothCount):

gear = Gear(reflectance, innerRadius, outerRadius, thickness, toothSize, toothCount)

result\_list = gl.glGenLists(1)

gl.glNewList(result\_list, gl.GL\_COMPILE)

gl.glMaterialfv(gl.GL\_FRONT, gl.GL\_AMBIENT\_AND\_DIFFUSE,

reflectance)

r0 = innerRadius

r1 = outerRadius - toothSize / 2.0

r2 = outerRadius + toothSize / 2.0

delta = (2.0 \* math.pi / toothCount) / 4.0

z = thickness / 2.0

gl.glShadeModel(gl.GL\_FLAT)

for i in range(2):

if i == 0:

sign = +1.0

else:

sign = -1.0

gl.glNormal3d(0.0, 0.0, sign)

gl.glBegin(gl.GL\_QUAD\_STRIP)

for j in range(toothCount + 1):

angle = 2.0 \* math.pi \* j / toothCount

gl.glVertex3d(r0 \* math.cos(angle), r0 \* math.sin(angle), sign \* z)

gl.glVertex3d(r1 \* math.cos(angle), r1 \* math.sin(angle), sign \* z)

gl.glVertex3d(r0 \* math.cos(angle), r0 \* math.sin(angle), sign \* z)

gl.glVertex3d(r1 \* math.cos(angle + 3 \* delta), r1 \* math.sin(angle + 3 \* delta), sign \* z)

gl.glEnd()

gl.glBegin(gl.GL\_QUADS)

for j in range(toothCount):

angle = 2.0 \* math.pi \* j / toothCount

gl.glVertex3d(r1 \* math.cos(angle), r1 \* math.sin(angle), sign \* z)

gl.glVertex3d(r2 \* math.cos(angle + delta), r2 \* math.sin(angle + delta), sign \* z)

gl.glVertex3d(r2 \* math.cos(angle + 2 \* delta), r2 \* math.sin(angle + 2 \* delta), sign \* z)

gl.glVertex3d(r1 \* math.cos(angle + 3 \* delta), r1 \* math.sin(angle + 3 \* delta), sign \* z)

gl.glEnd()

gl.glBegin(gl.GL\_QUAD\_STRIP)

for i in range(toothCount):

for j in range(2):

angle = 2.0 \* math.pi \* (i + (j / 2.0)) / toothCount

s1 = r1

s2 = r2

if j == 1:

s1, s2 = s2, s1

gl.glNormal3d(math.cos(angle), math.sin(angle), 0.0)

gl.glVertex3d(s1 \* math.cos(angle), s1 \* math.sin(angle), +z)

gl.glVertex3d(s1 \* math.cos(angle), s1 \* math.sin(angle), -z)

gl.glNormal3d(s2 \* math.sin(angle + delta) - s1 \* math.sin(angle),

s1 \* math.cos(angle) - s2 \* math.cos(angle + delta), 0.0)

gl.glVertex3d(s2 \* math.cos(angle + delta), s2 \* math.sin(angle + delta), +z)

gl.glVertex3d(s2 \* math.cos(angle + delta), s2 \* math.sin(angle + delta), -z)

gl.glVertex3d(r1, 0.0, +z)

gl.glVertex3d(r1, 0.0, -z)

gl.glEnd()

gl.glShadeModel(gl.GL\_SMOOTH)

gl.glBegin(gl.GL\_QUAD\_STRIP)

for i in range(toothCount + 1):

angle = i \* 2.0 \* math.pi / toothCount

gl.glNormal3d(-math.cos(angle), -math.sin(angle), 0.0)

gl.glVertex3d(r0 \* math.cos(angle), r0 \* math.sin(angle), +z)

gl.glVertex3d(r0 \* math.cos(angle), r0 \* math.sin(angle), -z)

gl.glEnd()

gl.glEndList()

gear.points = result\_list

return gear

Файл Gear.py

class Gear:

outputAngle = 0

dx = 0

dy = 0

dz = 0

def \_\_init\_\_(self, reflectance, innerRadius, outerRadius, thickness, toothSize, toothCount):

self.reflectance = reflectance

self.innerRadius = innerRadius

self.outerRadius = outerRadius

self.thickness = thickness

self.toothSize = toothSize

self.toothCount = toothCount

self.points = []