**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 3**

Тема: Основы построения фотореалистичных изображений

Студент: Попова Наталья Сергеевна

Группа: 08-305

Преподаватель: Чернышов Л.Н.

Дата: 01.11.22

Оценка:

Москва, 2022

1. **Постановка задачи**

Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность

аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и

удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света.

Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

**Вариант №15:** Сектор эллипсоида.

.

1. **Описание программы**

**Язык программирования**: Python

**Используемые библиотеки**: tkinter, math, numpy

**Используемая среда программирования**: PyCharm

Программа строит сектор эллипсоида в зависимости от переданных на вход значений a, b, c; окрашивает грани фигуры в зависимости от координаты точки освещения. Также в ней можно поворачивать фигуры вдоль осей OX, OY, OZ; менять коэффициенты аппроксимации и отражения; сохранять файл в формате XML и открывать файлы в этом же формате.

Функции getPlane, getVector, getCosPlaneVector отвечают за определение цвета грани: нужно взять вектор от источника света до центра фигуры и найти косинус между нормалью к плоскости и этим вектором, чем больше косинус, тем светлее грань.

Функции getPoints, getFaces находят соответственно точки сектора эллипсоида по формуле эллипсоида с заданными a, b, c и грани по точкам.

За повороты фигуры отвечают функции turnX, turnY, turnZ, turn.

Функции faceVision и showFace раскрашивают грань, в зависимости от того, видно ли ее и если да, то закрашивают ее в более светлый или более темный цвет (см выше).

Функции saveParams и openParams сохраняют и открывают файлы в формате XML.

1. **Набор тестов**

**Тест №1**

Точность аппроксимации = 19

Коэффициент отражения = 1

Координаты освещения: X = 240 , Y = -90 , Z = 100

Параметры эллипсоида: A = 200 , B = 150 , C = 160

Повороты: X = 122 , Y = 30, Z = 180

**Тест №2**

Точность аппроксимации = 19

Коэффициент отражения = 1

Координаты освещения: X = 56 , Y = 47, Z = 81

Параметры эллипсоида: A = 200 , B = 150, C = 160

Повороты: X = 122, Y = 30, Z = 180

**Тест №3**

Точность аппроксимации = 43

Коэффициент отражения = 0.63

Координаты освещения: X = 56, Y = 47, Z = 81

Параметры эллипсоида: A = 200, B = 150, C = 160

Повороты: X = 128.2, Y = 116, Z = 167.8

**Тест №4**

Точность аппроксимации = 11

Коэффициент отражения = 0.59

Координаты освещения: X = 15, Y = -1, Z = 290

Параметры эллипсоида: A = 261.2, B = 89.8, C = 294.4

Повороты: X = 180, Y = 228.6, Z = 0

**Тест №5**

Точность аппроксимации = 20

Коэффициент отражения = 0.81

Координаты освещения: X = 66, Y = 18, Z = 127

Параметры эллипсоида: A = 261.2, B = 89.8, C = 294.4

Повороты: X = 180, Y = 228.8, Z = 0

**Тест №6**

Со значениями предыдущего тестапопробуем сохранить файл в формате XML, а затем поменять параметры. Как итог, файл был сохранен в папку «загрузки».

**Тест №7**

Теперь попробуем открыть этот же файл. Как показано на снимках экрана, параметры поменялись и стали такими же, как в тесте №5, следовательно, открылся нужный файл.

1. **Результаты выполнения тестов**

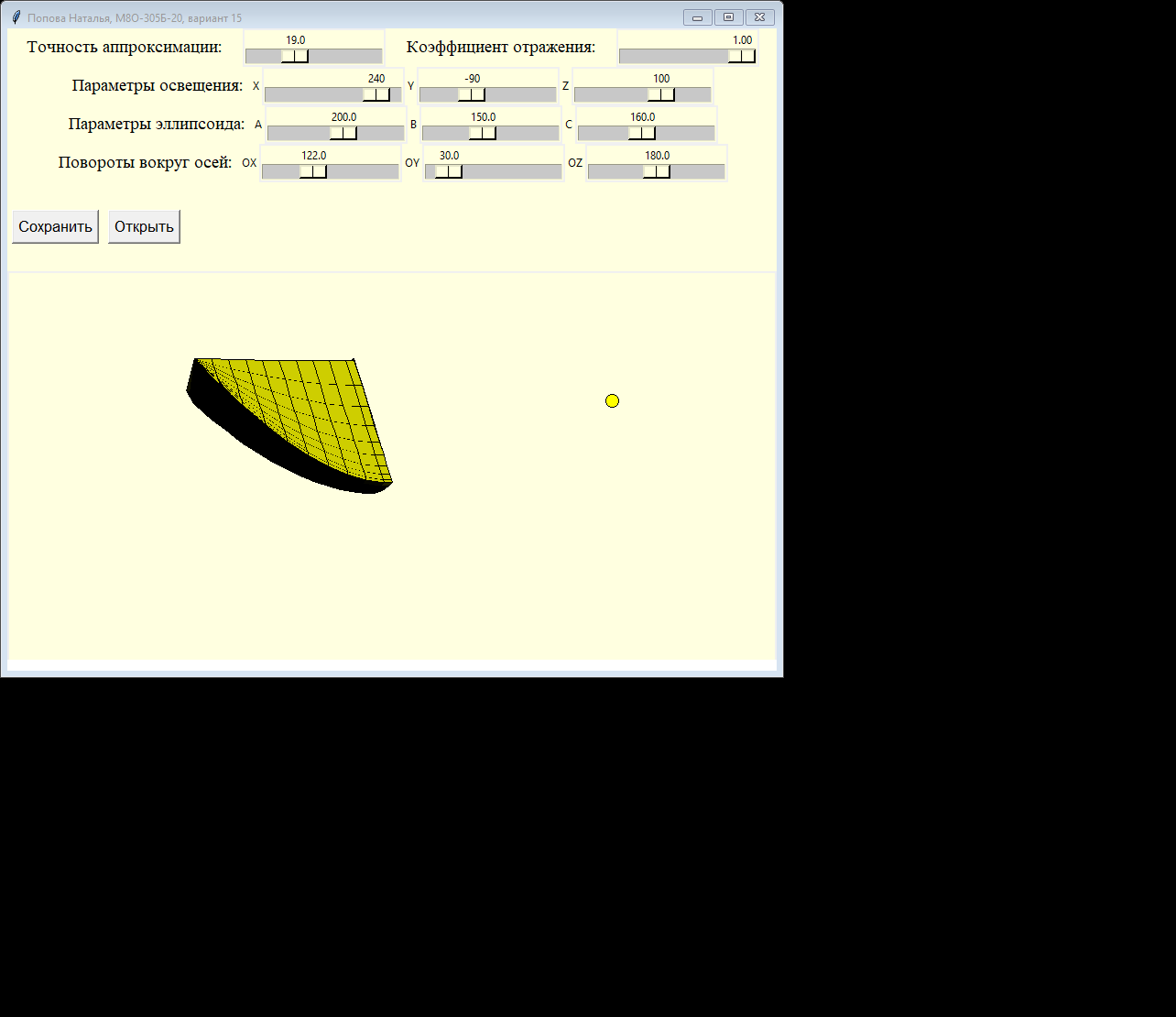


Рисунок 1 - Тест №1

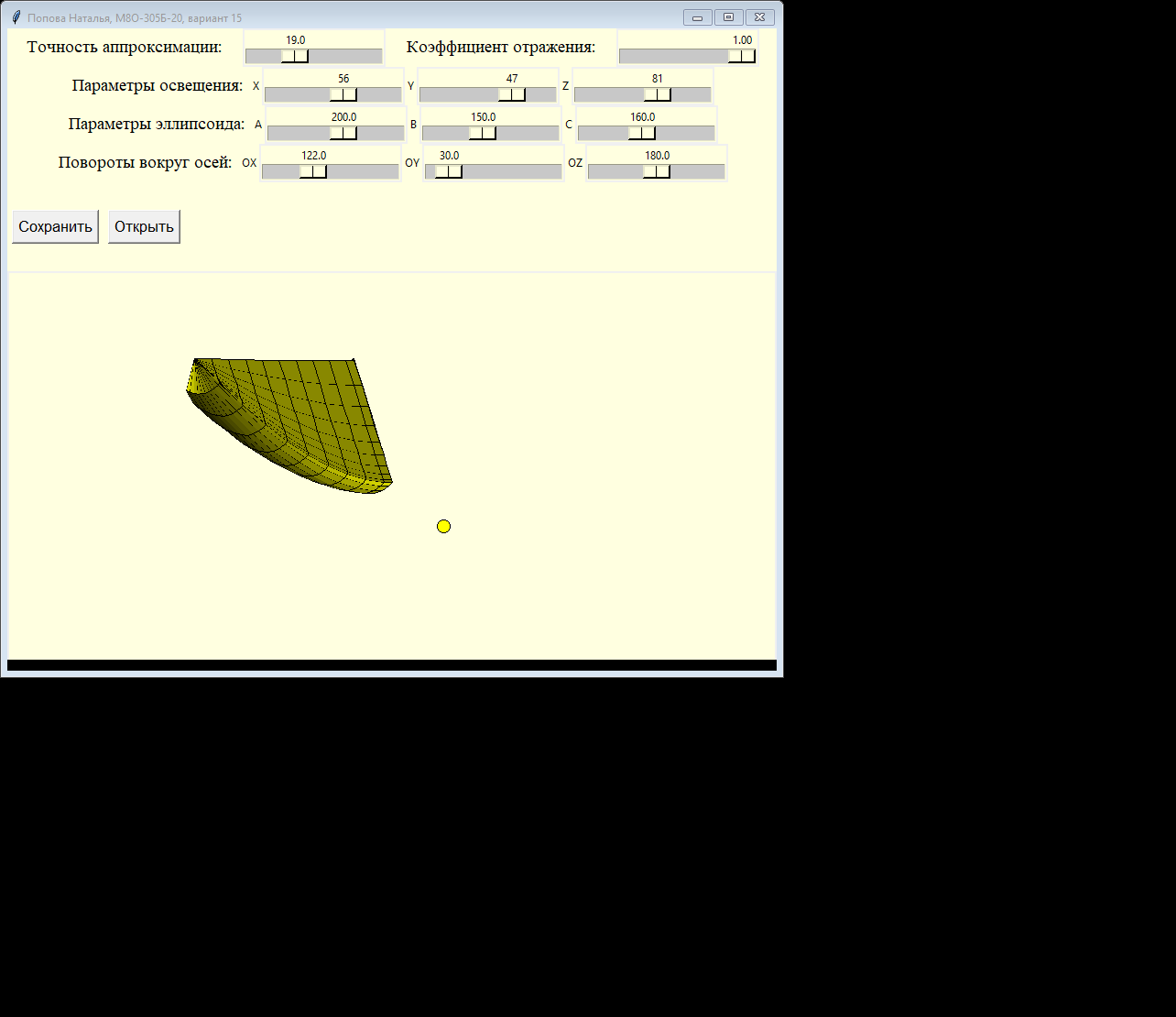


Рисунок 2 - Тест №2

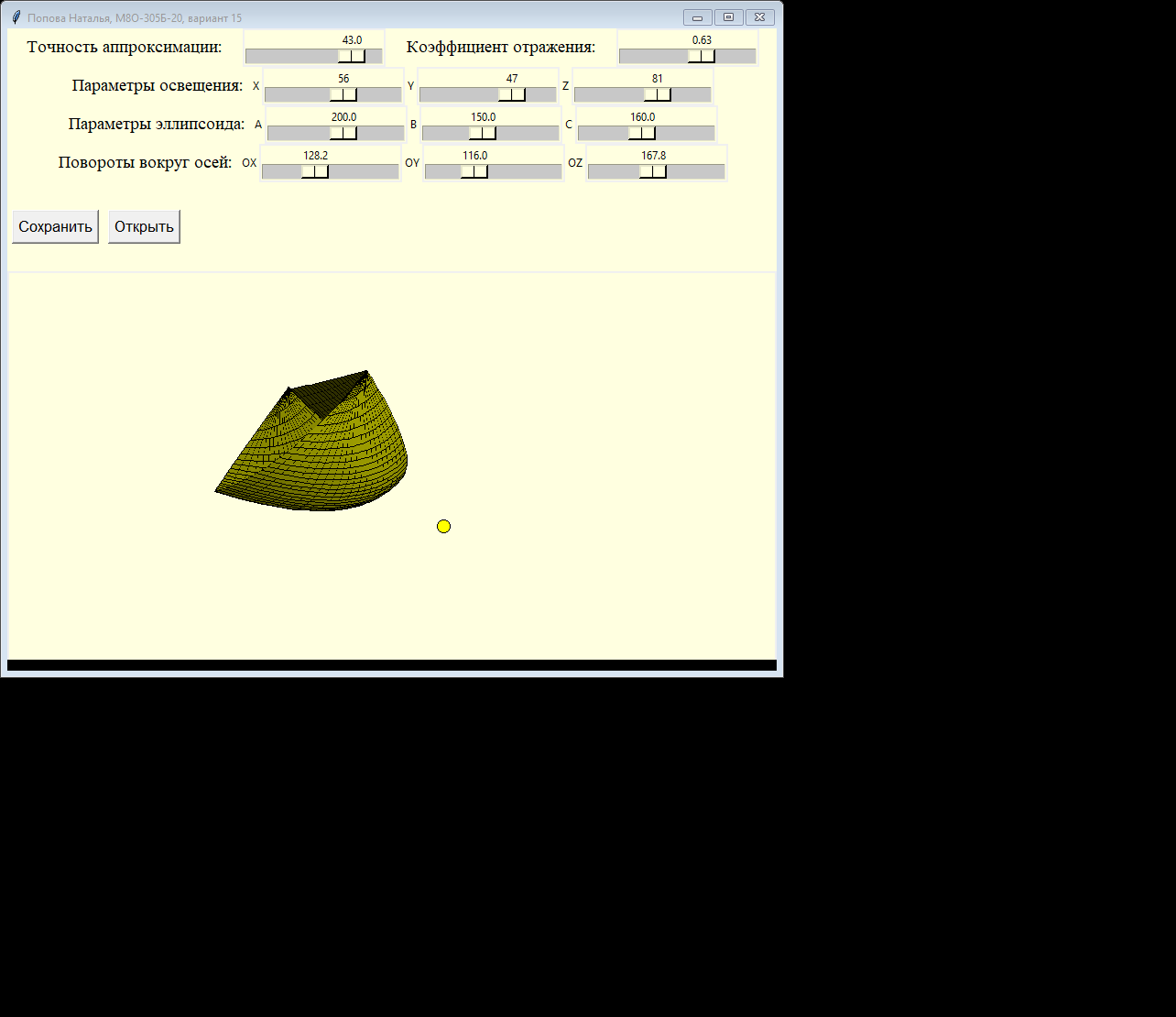


Рисунок 3 - Тест №3

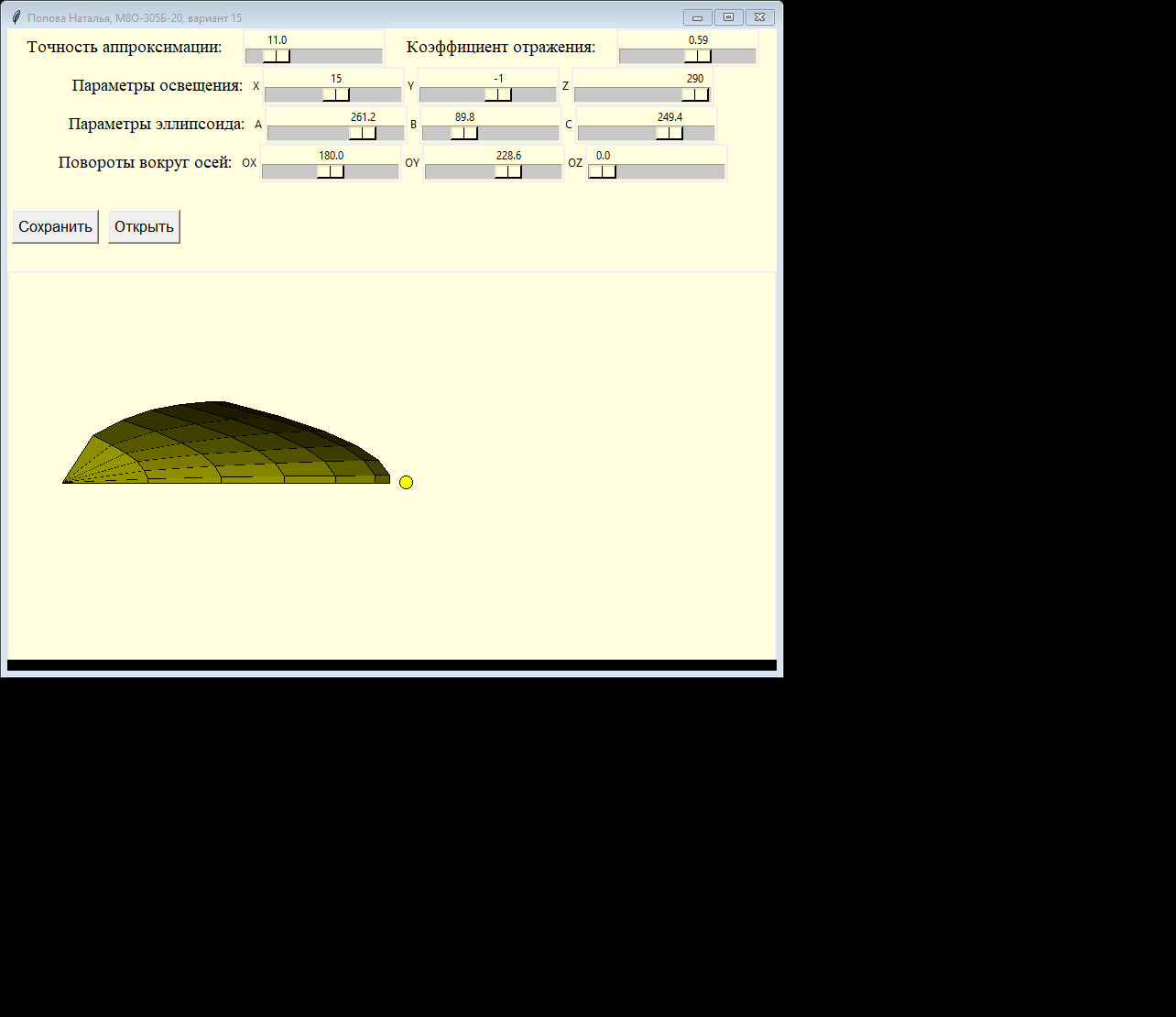


Рисунок 4 - Тест №4

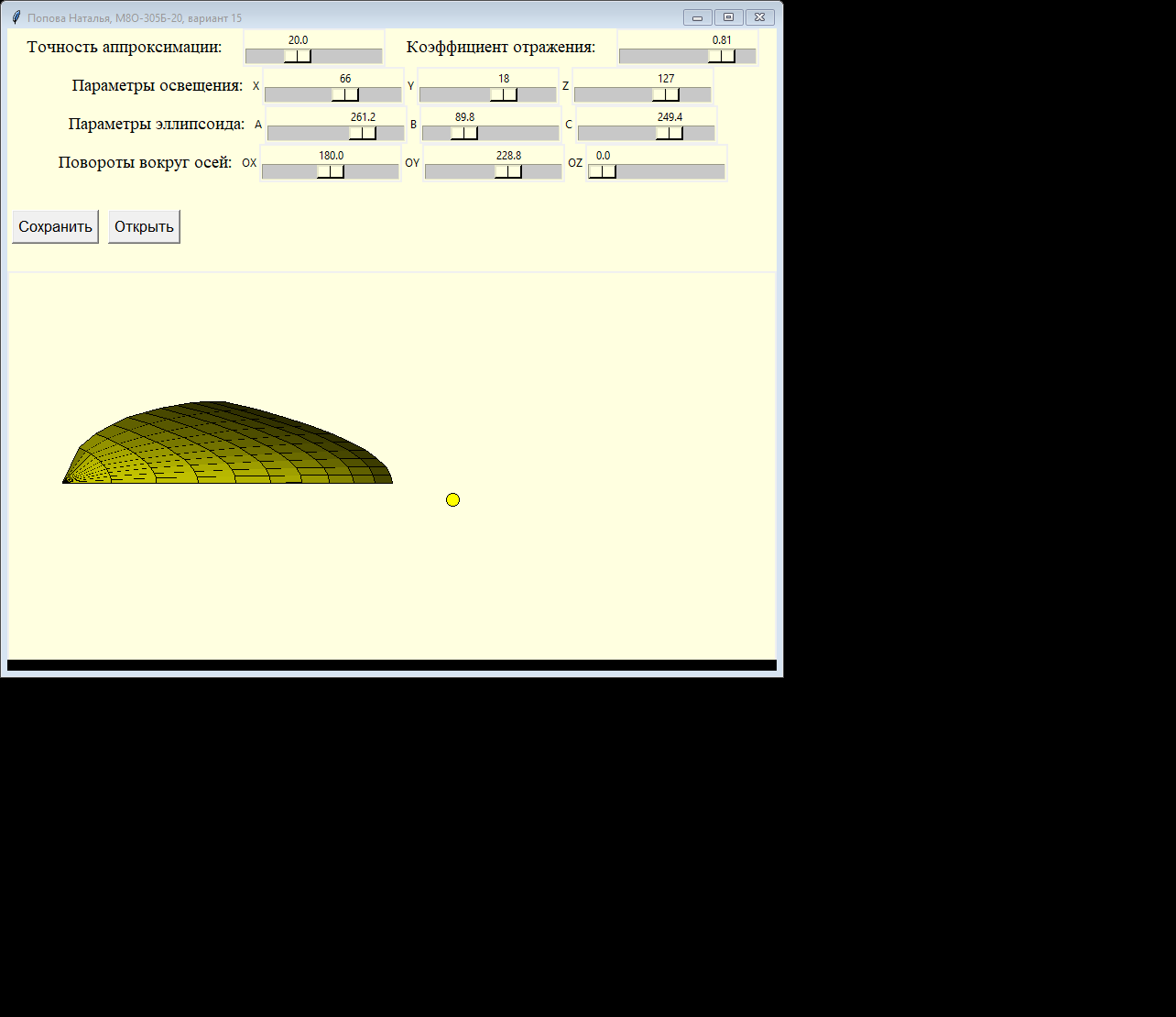


Рисунок 5 - Тест №5

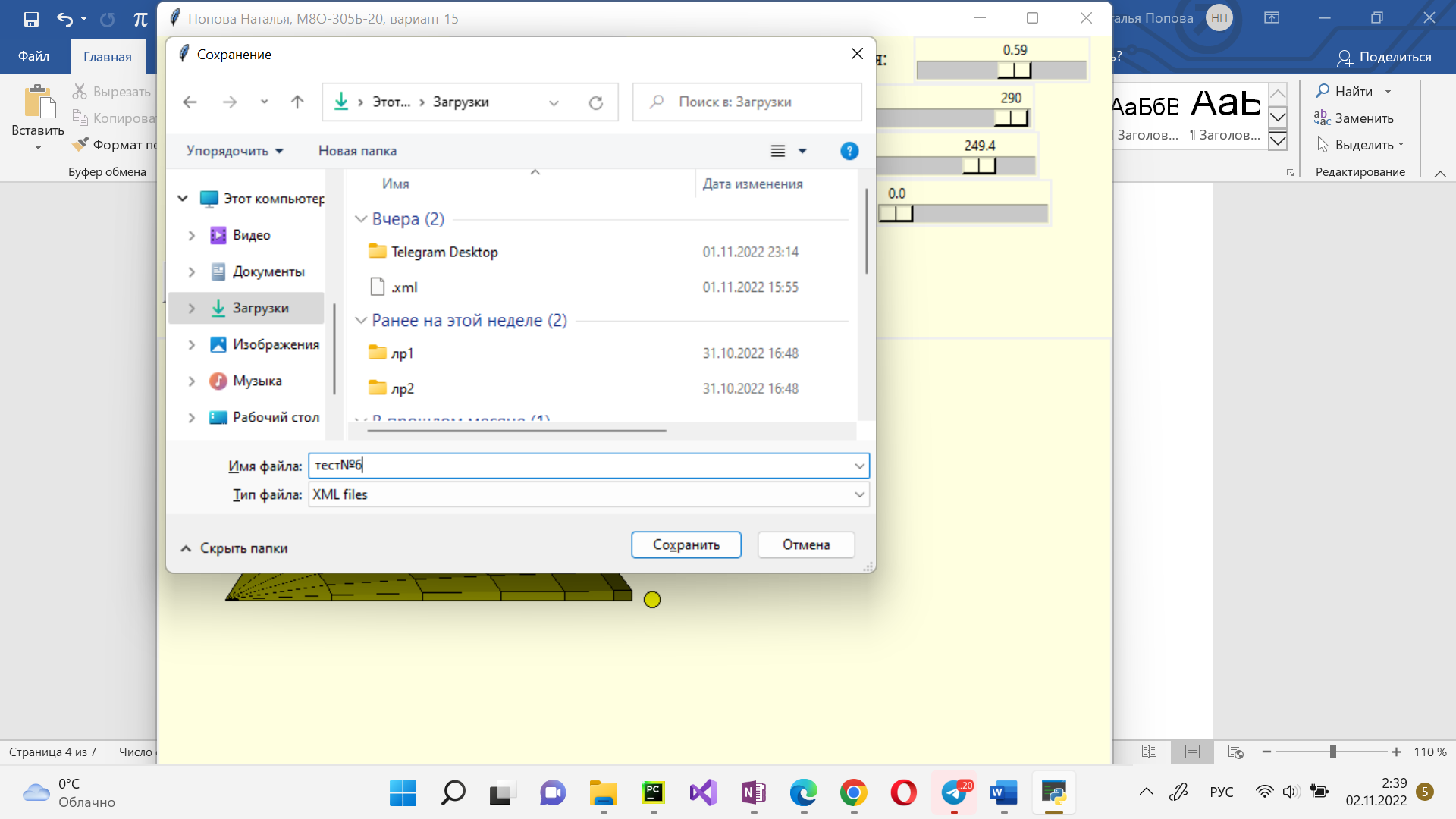


Рисунок 6 - Тест №6

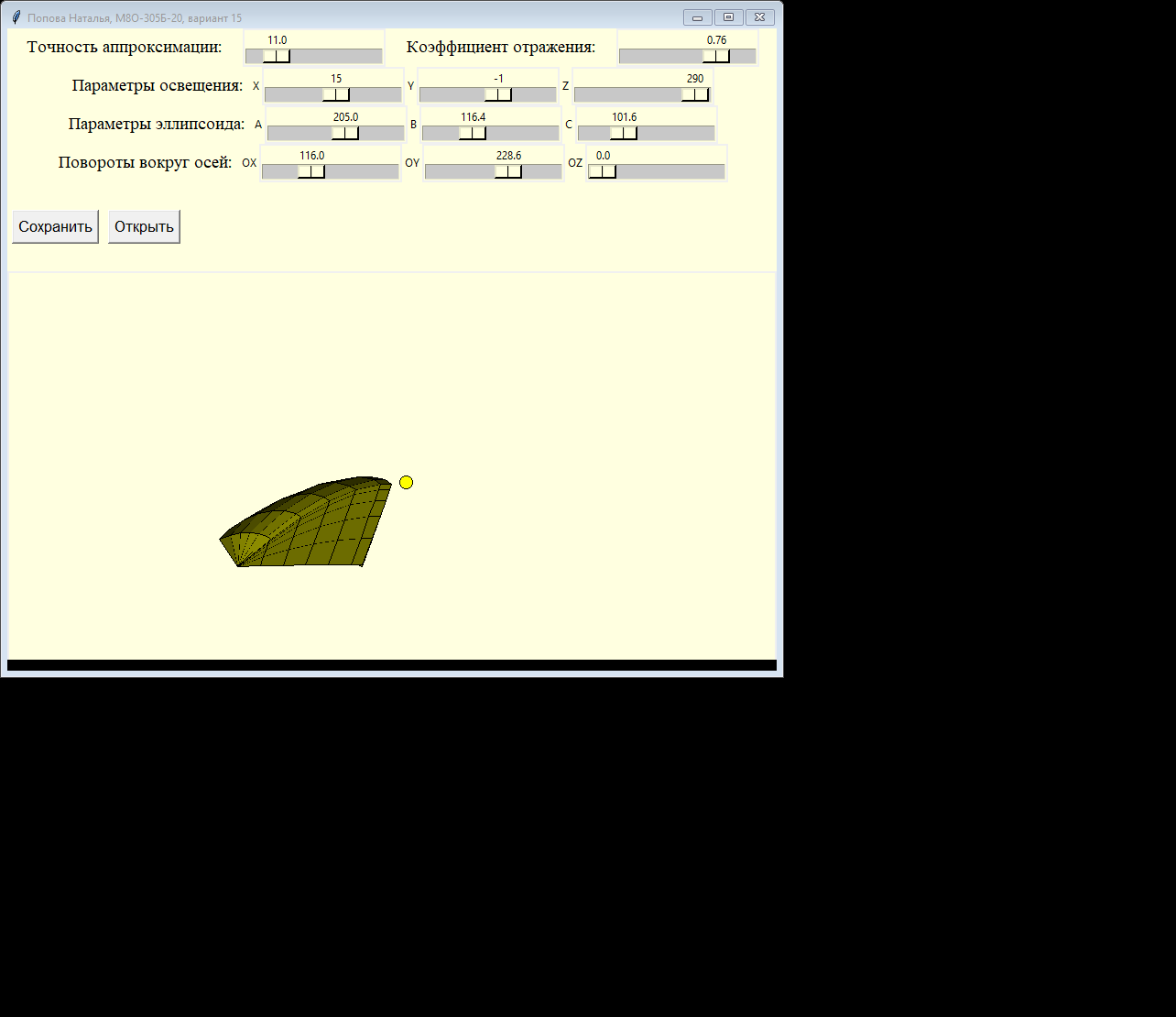


Рисунок 7 - Тест №6

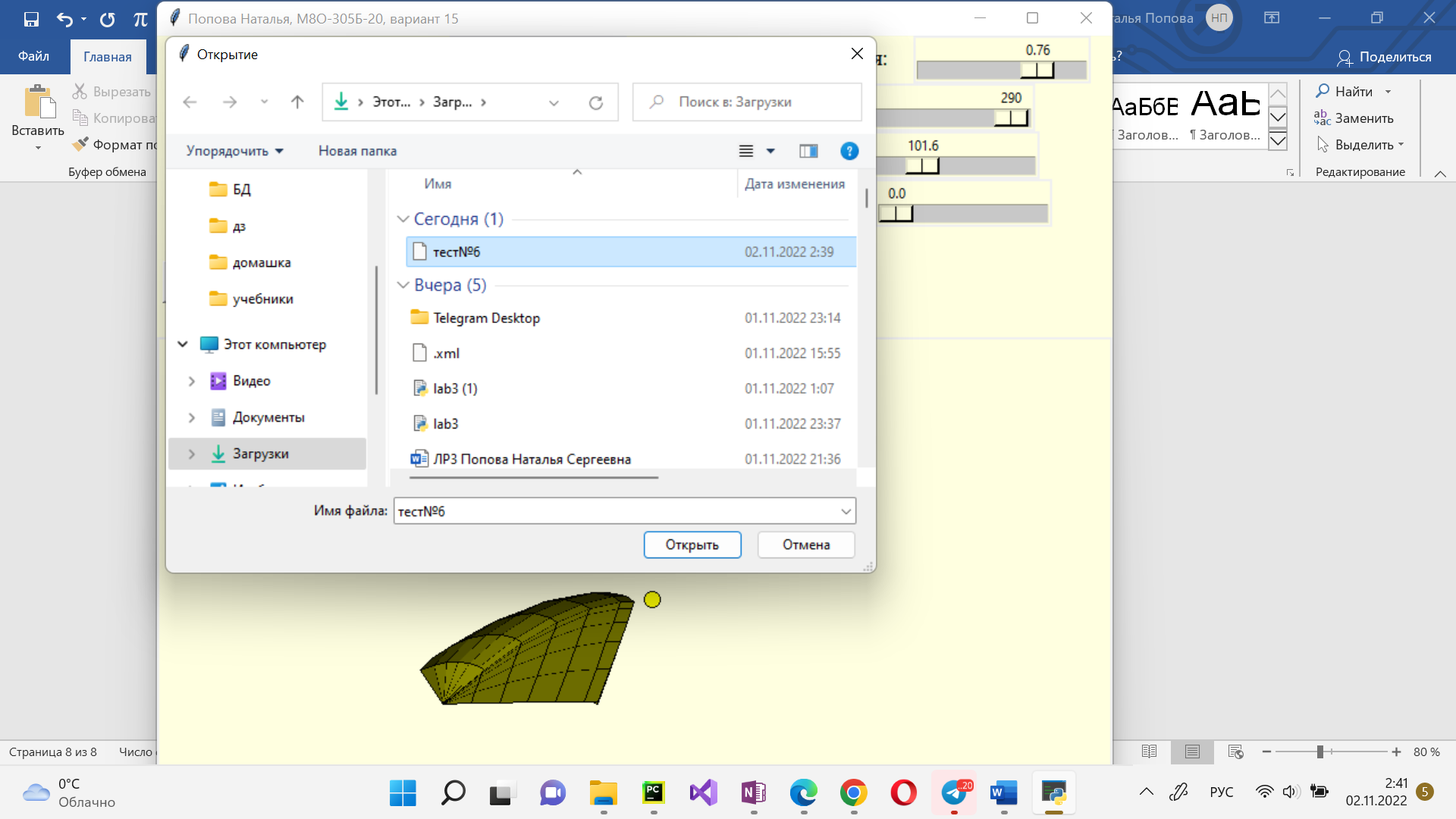


Рисунок 8 - Тест №7

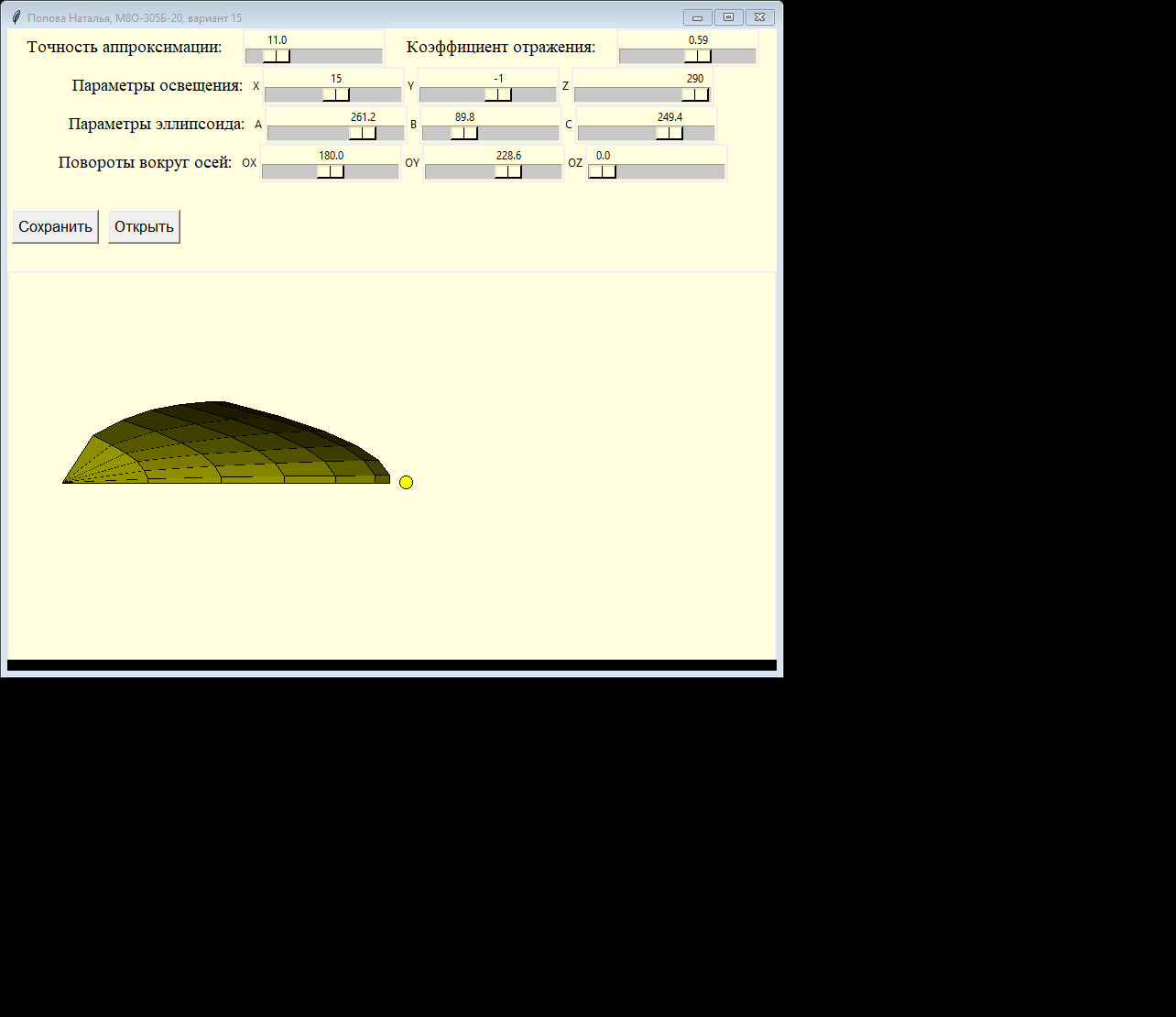


Рисунок 9 - Тест №7

1. **Листинг программы**

*# Попова Наталья М8О-305Б-20, Вариант №15: сектор эллипсоида***from** tkinter **import** \*  
**from** tkinter **import** filedialog **as** fd  
**import** xml.etree.ElementTree **as** xml  
**import** xml.dom.minidom **as** minidom  
**from** math **import** radians, sqrt, sin, cos  
**from** numpy **import** angle  
  
a, b, c = 200, 150, 160  
xS, yS, zS = 240, -90, 100  
reflection = 1  
p0 = [250, 200, 0]  
  
**def** getPlane(i1, j1, i2, j2, i3, j3):  
 x1 = points[i1][j1][0]  
 y1 = points[i1][j1][1]  
 z1 = points[i1][j1][2]  
 x2 = points[i2][j2][0]  
 y2 = points[i2][j2][1]  
 z2 = points[i2][j2][2]  
 x3 = points[i3][j3][0]  
 y3 = points[i3][j3][1]  
 z3 = points[i3][j3][2]  
 A = y1\*z2 - y1\*z3 - y2\*z1 + y2\*z3 + y3\*z1 - y3\*z2  
 B = x1\*z3 - x1\*z2 - x2\*z3 - x3\*z1 + x3\*z2 + x2\*z1  
 C = x1\*y2 - x1\*y3 - x2\*y1 + x2\*y3 + x3\*y1 - x3\*y2  
 **return** [A, B, C]  
  
**def** getVector(x1, y1, z1, x2, y2, z2):  
 **return** [x2 - x1, y2 - y1, z2 - z1]  
  
**def** getCosPlaneVector(vector, plane):  
 a = vector[0]\*plane[0] + vector[1]\*plane[1] + vector[2]\*plane[2]  
 b = sqrt((vector[0]\*vector[0] + vector[1]\*vector[1] + vector[2]\*vector[2])\*(plane[0]\*plane[0] + plane[1]\*plane[1] +  
 plane[2]\*plane[2]))  
 **return** 0 **if** b == 0 **else** a/b  
  
**def** getPoints(a, b, c, precision):  
 points = []  
 maxX = 100000 \* a / 100001  
 stepX = 2 \* maxX / precision  
 x = -maxX  
 **while** x <= (maxX + maxX/100000):  
 maxY = b\*sqrt(1 - (x/a)\*(x/a))  
 maxY = 100 \* maxY / 101  
 stepY = 2 \* maxY / precision  
 currentPoints = []  
 y = -maxY  
 **while** y <= (maxY + maxY/100):  
 z = c\*((x/a)\*(x/a) + (y/b)\*(y/b))  
 **if** z < 0:  
 **if** x<0:  
 **if** y<0:  
 currentPoints.append([0, 0, 0])  
 **else**:  
 currentPoints.append([0, y, 0])  
 **else**:  
 currentPoints.append([x, y, 0])  
 **elif** x<0:  
 **if** y<0:  
 currentPoints.append([0, 0, z])  
 **else**:  
 currentPoints.append([0, y, z])  
 **elif** (y<0):  
 currentPoints.append([x, 0, z])  
 **else**:  
 currentPoints.append([x, y, z])  
 y += stepY  
 points.append(currentPoints)  
 x += stepX  
 lowPoints = []  
 **for** i **in** range(0, len(points)):  
 currentPoints = []  
 **for** j **in** range(len(points[i]) - 1, -1, -1):  
 currentPoints.append([points[i][j][0], points[i][j][1], points[i][j][2]])  
 lowPoints.append(currentPoints)  
 **for** i **in** range(0, len(points)):  
 points[i] = points[i] + lowPoints[i]  
 **return** points  
  
**def** getFaces(points):  
 k = len(points)  
 n = len(points[0])  
 faces = []  
 **for** i **in** range(0, k - 1):  
 **for** j **in** range(0, n - 1):  
 temp = [[i, j], [i+1, j], [i+1, j+1], [i, j+1]]  
 faces.append(temp)  
 faces.append([[i, n-1], [i+1, n-1], [i+1, 0], [i, 0]])  
 **return** faces  
  
startPoints = getPoints(a, b, c, 50)  
points = list(startPoints)  
faces = getFaces(points)  
  
**def** initImage():  
 **for** circle **in** range(0, len(points)):  
 **for** point **in** range(0, len(points[circle])):  
 **for** i **in** range(0, len(points[circle][point])):  
 points[circle][point][i] = startPoints[circle][point][i]  
  
**def** turnX(fi, i, j):  
 y = points[i][j][1] \* cos(fi) + points[i][j][2] \* sin(fi)  
 z = -points[i][j][1] \* sin(fi) + points[i][j][2] \* cos(fi)  
 points[i][j][1] = y  
 points[i][j][2] = z  
  
**def** turnY(fi, i, j):  
 x = points[i][j][0] \* cos(fi) - points[i][j][2] \* sin(fi)  
 z = points[i][j][0] \* sin(fi) + points[i][j][2] \* cos(fi)  
 points[i][j][0] = x  
 points[i][j][2] = z  
  
**def** turnZ(fi, i, j):  
 x = points[i][j][0] \* cos(fi) - points[i][j][1] \* sin(fi)  
 y = points[i][j][0] \* sin(fi) + points[i][j][1] \* cos(fi)  
 points[i][j][0] = x  
 points[i][j][1] = y  
  
**def** turn(ax, ay, az):  
 **for** i **in** range(0, len(points)):  
 **for** j **in** range(0, len(points[i])):  
 turnX(ax, i, j)  
 **for** i **in** range(0, len(points)):  
 **for** j **in** range(0, len(points[i])):  
 turnY(ay, i, j)  
 **for** i **in** range(0, len(points)):  
 **for** j **in** range(0, len(points[i])):  
 turnZ(az, i, j)  
  
**def** faceVision(i1, j1, i2, j2, i3, j3):  
 x1 = points[i1][j1][0]  
 y1 = points[i1][j1][1]  
 x2 = points[i2][j2][0]  
 y2 = points[i2][j2][1]  
 x3 = points[i3][j3][0]  
 y3 = points[i3][j3][1]  
 **return** (x1-x2)\*(y2-y3) - (x2-x3)\*(y1-y2) >= 0  
  
**def** showFace(face):  
 **if** (**not** faceVision(face[0][0], face[0][1], face[1][0], face[1][1], face[2][0], face[2][1])):  
 **return** pointsPoly = []  
 **for** i **in** range(1, len(face)):  
 circle1 = face[i-1][0]  
 point1 = face[i-1][1]  
 circle2 = face[i][0]  
 point2 = face[i][1]  
 p1 = points[circle1][point1]  
 p2 = points[circle2][point2]  
 canvas.create\_line(p0[0] + p1[0], p0[1] + p1[1], p0[0] + p2[0], p0[1] + p2[1], width=1)  
 pointsPoly.append(p0[0] + p1[0])  
 pointsPoly.append(p0[1] + p1[1])  
 pointsPoly.append(p0[0] + p2[0])  
 pointsPoly.append(p0[1] + p2[1])  
 plane = getPlane(face[0][0], face[0][1], face[1][0], face[1][1], face[2][0], face[2][1])  
 vector = getVector(0, 0, 0, xS, yS, zS)  
 koef = getCosPlaneVector(vector, plane)  
 koef = 0 **if** koef < 0 **else** koef  
 canvas.create\_polygon(pointsPoly, fill=(**'#%02x%02x%02x'** % (int(255\*reflection\*koef), int(255\*reflection\*koef),  
 int(0\*reflection\*koef))))  
 circle1 = face[0][0]  
 point1 = face[0][1]  
 circle2 = face[-1][0]  
 point2 = face[-1][1]  
 p1 = points[circle1][point1]  
 p2 = points[circle2][point2]  
 canvas.create\_line(p0[0] + p1[0], p0[1] + p1[1], p0[0] + p2[0], p0[1] + p2[1], width=0.5)  
  
**def** showBody():  
 **for** i **in** faces:  
 showFace(i)  
 canvas.create\_oval(p0[0] + xS - 7, p0[1] + yS - 7, p0[0] + xS + 7, p0[1] + yS + 7, fill=**'yellow'**)  
  
**def** paramsChanged(ev):  
 canvas.delete(**"all"**)  
 **global** startPoints, points, p0, faces, reflection, xS, yS, zS  
 xS = float(scaleXS.get())  
 yS = float(scaleYS.get())  
 zS = float(scaleZS.get())  
 reflection = float(scaleReflection.get())  
 precision = float(scalePrecision.get())  
 w = window.winfo\_width()  
 h = window.winfo\_height()  
 p0[0] = w/2  
 p0[1] = h/2 - 120  
 a = float(scaleA.get())  
 b = float(scaleB.get())  
 c = float(scaleC.get())  
 startPoints = getPoints(a, b, c, precision)  
 points = list(startPoints)  
 faces = getFaces(points)  
 rx = float(scaleX.get())  
 ry = float(scaleY.get())  
 rz = float(scaleZ.get())  
 ax = radians(rx)  
 ay = radians(ry)  
 az = radians(rz)  
 initImage()  
 turn(ax, ay, az)  
 showBody()  
  
**def** saveParams():  
 file\_name = fd.asksaveasfilename(  
 filetypes=[(**"XML files"**, **"\*.xml"**)])  
 rx = str(scaleX.get())  
 ry = str(scaleY.get())  
 rz = str(scaleZ.get())  
 xS = str(scaleXS.get())  
 yS = str(scaleYS.get())  
 zS = str(scaleZS.get())  
 reflection = str(scaleReflection.get())  
 precision = str(scalePrecision.get())  
 a = str(scaleA.get())  
 b = str(scaleB.get())  
 c = str(scaleC.get())  
 root = xml.Element(**"params"**)  
 paraboloidParams = xml.SubElement(root, **"paraboloidParams"**)  
 angles = xml.SubElement(root, **"angles"**)  
 sourceCoords = xml.SubElement(root, **"sourceCoords"**)  
 otherParams = xml.SubElement(root, **"otherParams"**)  
 aXml = xml.SubElement(paraboloidParams, **"a"**)  
 aXml.text = a  
 bXml = xml.SubElement(paraboloidParams, **"b"**)  
 bXml.text = b  
 cXml = xml.SubElement(paraboloidParams, **"c"**)  
 cXml.text = c  
 ax = xml.SubElement(angles, **"ax"**)  
 ax.text = rx  
 ay = xml.SubElement(angles, **"ay"**)  
 ay.text = ry  
 az = xml.SubElement(angles, **"az"**)  
 az.text = rz  
 x = xml.SubElement(sourceCoords, **"x"**)  
 x.text = xS  
 y = xml.SubElement(sourceCoords, **"y"**)  
 y.text = yS  
 z = xml.SubElement(sourceCoords, **"z"**)  
 z.text = zS  
 reflectionXml = xml.SubElement(otherParams, **"reflection"**)  
 reflectionXml.text = reflection  
 precisionXml = xml.SubElement(otherParams, **"precision"**)  
 precisionXml.text = precision  
 tree = xml.ElementTree(root)  
 tree.write(file\_name + **".xml"**)  
  
**def** openParams(): *# восстановление данных* file\_name = fd.askopenfilename()  
 doc = minidom.parse(file\_name)  
 paraboloidParams = doc.getElementsByTagName(**"paraboloidParams"**)[0]  
 angles = doc.getElementsByTagName(**"angles"**)[0]  
 sourceCoords = doc.getElementsByTagName(**"sourceCoords"**)[0]  
 otherParams = doc.getElementsByTagName(**"otherParams"**)[0]  
 scaleX.set(float(angles.childNodes[0].firstChild.nodeValue))  
 scaleY.set(float(angles.childNodes[1].firstChild.nodeValue))  
 scaleZ.set(float(angles.childNodes[2].firstChild.nodeValue))  
 scaleXS.set(float(sourceCoords.childNodes[0].firstChild.nodeValue))  
 scaleYS.set(float(sourceCoords.childNodes[1].firstChild.nodeValue))  
 scaleZS.set(float(sourceCoords.childNodes[2].firstChild.nodeValue))  
 scaleReflection.set(float(otherParams.childNodes[0].firstChild.nodeValue))  
 scalePrecision.set(float(otherParams.childNodes[1].firstChild.nodeValue))  
 scaleA.set(float(paraboloidParams.childNodes[0].firstChild.nodeValue))  
 scaleB.set(float(paraboloidParams.childNodes[1].firstChild.nodeValue))  
 scaleC.set(float(paraboloidParams.childNodes[2].firstChild.nodeValue))  
  
window = Tk() *# создание и заполнение окна*window.title(**"Попова Наталья, М8О-305Б-20, вариант 15"**)  
window.configure(bg = **'lightyellow'**)  
window.geometry(**'840x755'**)  
precisionFrame = Frame(window,bg = **'lightyellow'**)  
precisionFrame.pack()  
coordsFrame = Frame(window,bg = **'lightyellow'**)  
coordsFrame.pack()  
topFrame = Frame(window,bg = **'lightyellow'**)  
topFrame.pack()  
bottomFrame = Frame(window,bg = **'lightyellow'**)  
bottomFrame.pack()  
  
labelPrecision = Label(precisionFrame, font=**'Times 14'**, text=**"Точность аппроксимации:"**,bg = **'lightyellow'**)  
labelPrecision.pack(side=LEFT, padx=10)  
  
scalePrecision = Scale(precisionFrame, from\_=4, to=50, orient=HORIZONTAL, length=150, resolution=0.2, command=paramsChanged,bg = **'lightyellow'**)  
scalePrecision.pack(side=LEFT, padx=10)  
scalePrecision.set(15)  
  
labelPrecision = Label(precisionFrame, font=**'Times 14'**, text=**"Коэффициент отражения:"**,bg = **'lightyellow'**)  
labelPrecision.pack(side=LEFT, padx=10)  
  
scaleReflection = Scale(precisionFrame, from\_=0, to=1, orient=HORIZONTAL, length=150, resolution=0.01, command=paramsChanged,bg = **'lightyellow'**)  
scaleReflection.pack(side=RIGHT, padx=10)  
scaleReflection.set(1)  
  
labelPrecision = Label(coordsFrame, font=**'Times 14'**, text=**"Параметры освещения: "**, bg = **'lightyellow'**)  
labelPrecision.pack(side=LEFT)  
  
labelXS = Label(coordsFrame, text=**"X"**, bg = **'lightyellow'**)  
labelXS.pack(side=LEFT)  
scaleXS = Scale(coordsFrame, from\_=-300, to=300, orient=HORIZONTAL, length=150,  
 resolution=1, command=paramsChanged,bg = **'lightyellow'**)  
scaleXS.pack(side=LEFT)  
scaleXS.set(240)  
  
labelYS = Label(coordsFrame, text=**"Y"**, bg = **'lightyellow'**)  
labelYS.pack(side=LEFT)  
scaleYS = Scale(coordsFrame, from\_=-220, to=150, orient=HORIZONTAL, length=150,  
 resolution=1, command=paramsChanged,bg = **'lightyellow'**)  
scaleYS.pack(side=LEFT)  
scaleYS.set(-90)  
  
labelZS = Label(coordsFrame, text=**"Z"**,bg = **'lightyellow'**)  
labelZS.pack(side=LEFT)  
scaleZS = Scale(coordsFrame, from\_=-300, to=300, orient=HORIZONTAL, length=150,  
 resolution=1, command=paramsChanged,bg = **'lightyellow'**)  
scaleZS.pack(side=LEFT)  
scaleZS.set(100)  
  
labelX = Label(topFrame, font=**'Times 14'**, text=**'Параметры эллипсоида: '**, bg = **'lightyellow'**)  
labelX.pack(side=LEFT)  
  
labelA = Label(topFrame, text=**"A"**,bg = **'lightyellow'**)  
labelA.pack(side=LEFT)  
scaleA = Scale(topFrame, from\_=1, to=350, orient=HORIZONTAL, length=150,  
 resolution=0.2, command=paramsChanged,bg = **'lightyellow'**)  
scaleA.pack(side=LEFT)  
scaleA.set(200)  
  
labelB = Label(topFrame, text=**"B"**,bg = **'lightyellow'**)  
labelB.pack(side=LEFT)  
scaleB = Scale(topFrame, from\_=1, to=350, orient=HORIZONTAL, length=150,  
 resolution=0.2, command=paramsChanged,bg = **'lightyellow'**)  
scaleB.pack(side=LEFT)  
scaleB.set(150)  
  
labelC = Label(topFrame, text=**"C"**,bg = **'lightyellow'**)  
labelC.pack(side=LEFT)  
scaleC = Scale(topFrame, from\_=1, to=350, orient=HORIZONTAL, length=150,  
 resolution=0.2, command=paramsChanged,bg = **'lightyellow'**)  
scaleC.pack(side=LEFT)  
scaleC.set(160)  
  
labelX = Label(bottomFrame, font=**'Times 14'**, text=**'Повороты вокруг осей: '**,bg = **'lightyellow'**)  
labelX.pack(side=LEFT)  
  
labelX = Label(bottomFrame, text=**'ОX'**,bg = **'lightyellow'**)  
labelX.pack(side=LEFT)  
scaleX = Scale(bottomFrame, from\_=0, to=360, orient=HORIZONTAL, length=150,  
 resolution=0.2, command=paramsChanged,bg = **'lightyellow'**)  
scaleX.pack(side=LEFT)  
scaleX.set(80)  
  
labelY = Label(bottomFrame, text=**'ОY'**,bg = **'lightyellow'**)  
labelY.pack(side=LEFT)  
scaleY = Scale(bottomFrame, from\_=0, to=360, orient=HORIZONTAL, length=150,  
 resolution=0.2, command=paramsChanged,bg = **'lightyellow'**)  
scaleY.pack(side=LEFT)  
scaleY.set(30)  
  
labelZ = Label(bottomFrame, text=**'ОZ'**,bg = **'lightyellow'**)  
labelZ.pack(side=LEFT)  
scaleZ = Scale(bottomFrame, from\_=0, to=360, orient=HORIZONTAL, length=150,  
 resolution=0.2, command=paramsChanged,bg = **'lightyellow'**)  
scaleZ.pack(side=LEFT)  
scaleZ.set(180)  
  
buttonsFrame = Frame(window, bg = **'lightyellow'**)  
buttonsFrame.pack(fill=BOTH)  
saveButton = Button(buttonsFrame, font=**'Canvas 13'**, text=**"Сохранить"**, command=saveParams)  
saveButton.pack(side=LEFT, ipady=2, pady=30, padx=5)  
openButton = Button(buttonsFrame, font=**'Canvas 13'**, text=**"Открыть"**, command=openParams)  
openButton.pack(side=LEFT, ipady=2, pady=30, padx=5)  
  
  
canvas = Canvas(window, width=2000, height=2000, bg = **'lightyellow'**)  
canvas.pack(fill=BOTH)  
window.bind(**'<Configure>'**, paramsChanged)  
showBody()  
  
window.mainloop()

**ЛИТЕРАТУРА**

1. Документация к библиотеке tkinter [Электронный ресурс].

URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения 31.10.22)

1. Текстовые файлы. URL: <https://metanit.com/python/tutorial/4.2.php> (дата обращения 30.10.22)
2. Аппроксимация 3D форм. URL: <http://gwyddion.net/documentation/user-guide-ru/fitting-3d-shapes.html> (дата обращения 30.10.22)