Идеи по реализации вращения фигуры в трёхмерном пространстве)

Прежде всего надо разделить (в голове) две вещи:

- процесс движения фигуры в пространстве (с использованием нужных структур данных, пересчётом координат);
- отображение этого процесса на плоском экране компьютера.

Ниже обе части описаны подробнее.

1. Запишите матрицу A поворота точки с координатами (x_1, y_1) на плоскости вокруг начала координат.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

   Элементы полученной матрицы это коэффициенты перед x_1 и y_1 в выражениях:

$$x_2 = a_{11} \cdot x_1 + a_{12} \cdot y_1$$
$$y_2 = a_{21} \cdot x_1 + a_{22} \cdot y_1$$

2. Что поменяется, если вам нужен поворот вокруг точки с координатами (p, q)?
3. Напишите программу, которая будет вращать точку, например, так: ссылка.

   Удобно реализовать для этого функцию `rotate`, как метод класса `Point`.
4. Опишите подходящий класс для хранения грани (`Face`). Например, можно хранить набор вершин, перечисленных в определённом порядке обхода границы (по или против часовой стрелки). Реализуйте функцию `rotate`, как метод этого класса, используя `rotate` из предыдущего пункта.
5. Проверьте, что на плоскости всё работает (например, многоугольник вращается вокруг начала координат или какой-то другой точки). Примерно так: ссылка.
6. Добавим $3d$. Теперь можно вращать вокруг трёх осей (в трёх плоскостях: $x = const, y = const, z = const$).

   Задайте углы вращения вокруг всех трёх осей $\alpha_x, \alpha_y, \alpha_z$. Напишите для каждого из трёх углов свою матрицу поворота (в каждой будут только 4 ненулевых элемента, похожие на те, что было в матрице поворота на плоскости).

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$$

   Чтобы вращать свою фигуру вокруг нескольких осей, надо умножить вектор с координатами точки на соответствующие матрицы поворота. Можно получить матрицу всего преобразования, перемножив все три матрицы поворота.
7. Попробуйте покрутить плоскую фигуру (грань) в пространстве. Получится примерно так: ссылка.

   В примере грань освещается и интенсивность цвета зависит от угла между вектором наблюдатель-фигура и вектором нормали к грани (для этого нам надо, чтобы точки, определяющие грань всегда перечислялись в одном и том же порядке). Освещение можно пока не делать.
8. Опишите подходящий класс для хранения многогранника и напишите для него метод `rotate`.
9. Чтобы правильно показывать многогранник в пространстве, надо сперва осознать, что вы показываете его проекцию на плоскость. Кроме того, надо научиться определять — видна грань из точки наблюдателя или нет. Тут вам поможет угол между вектором нормали к грани и вектором из «центра» фигуры к точке наблюдателя.

   Вот видео освещаемого вращающегося усечённого куба: ссылка.

Вот так выглядит код с точки зрения tkinter-рисования:

```
1  from tkinter import *
2
3  # Here are all classes, functions, etc...
4
5
6  def move():
7      global viewPoint                    # viewPoint is the point where we looking from
8      w.delete(ALL)                       # delete all from the screen
9      T.rotate(ax, ay, az)                # rotate all polyhedron T (i.e. get new coords)
10     for face in T.faces:                # show all visible faces of polyhedron T
11         if face.visible(viewPoint):     # visible() is a method of Face class
12             coords = []
13             for p in face.pts:          # Move figure to the center of the screen
14                 coords += [p.x + WIDTH / 2, p.y + HEIGHT / 2]
15             w.create_polygon(*coords, fill=face.getColor(viewPoint))
16     root.after(10, move)                # make a cycle over each 10ms
17
18
19 WIDTH = 700
20 HEIGHT = 700
21 viewPoint = Point(1, 1, 1000)          # view point is high above Oxy plane
22
23 root = Tk()
24 w = Canvas(root, width=WIDTH, height=HEIGHT) # create canvas
25 w.configure(background='#113311')         # some settings
26 w.pack()    # make all objects visible using one of geometry managers
27 move()      # start move
28 mainloop()  # this function starts all and just wait for events (user or programming)
```

Функция `move()` связывает вашу геометрическую составляющую с `tkinter`-рисованием и состоит из четырёх частей:

- удаляет всё нарисованное с экрана (8 строка, вызов функции из библиотеки `tkinter`);
- пересчитывает координаты вершин (строка 9, вызов метода класса, описывающего многогранник);
- собирает координаты вершин видимых граней многогранника и сохраняет их в массиве `coords` (строки 10-14)
- рисование многогранника (15 строка — вызов функции из библиотеки `tkinter`);
- помещает в очередь событий информацию о том, что функцию `move()` надо запустить через 10 миллисекунд (строка 16).

Как нарисовать многоугольник в `tkinter` написано, например, здесь.

Затем создаётся и подготавливается `tkinter`-окно, canvas («холст») и вызывается функция `move()`. Результат этого вызова — нарисованый первый кадр и создание записи о новом запуске функции `move()` через указаное время (т.е. она сама себя ставит в очередь, организуя таким образом последовательность вызовов).

Последняя строчка, `mainloop()`, это запуск основного цикла `tkinter`-программы, которая обрабатывает очередь событий.