Научная статья
УДК 004.42
EDN SLOQZB

Создание процедурной анимации

Илья Юйцзоевич Ефременко[1], обучающийся
Олеся Александровна Пустовая[2], кандидат сельскохозяйственных наук, доцент
[1] Средняя общеобразовательная школа № 1
Амурская область, Ивановка, Россия
[2] Дальневосточный государственный аграрный университет
Амурская область, Благовещенск, Россия
[1] efrtvtnko@gmail.com, [2] pus14@yandex.ru

Аннотация. Показан процесс создания модели движения и описания тел со сложной математической структурой. Модель предназначена для отображения и отрисовки и выполнена при помощи языка программирования Python.

Ключевые слова: процедурная анимация, модуль, программный код, отрисовка, программирование, Python

Для цитирования: Ефременко И. Ю., Пустовая О. А. Создание процедурной анимации // Актуальные вопросы энергетики в АПК : материалы всерос. (нац.) науч.-практ. конф. (Благовещенск, 19 декабря 2024 г.). Благовещенск : Дальневосточный ГАУ, 2025. С. 262–268.

Original article

Creating a procedural animation

Ilya Yu. Efremenko[1], School Student
Olesya A. Pustovaya, Candidate of Agricultural Sciences, Associate Professor
[1] Secondary School No. 1, Amur region, Ivanovka, Russia
[2] Far Eastern State Agrarian University, Amur region, Blagoveshchensk, Russia
[1] efrtvtnko@gmail.com, [2] pus14@yandex.ru

Abstract. The process of creating a motion model and describing bodies with a complex mathematical structure is shown. The model is designed for display and rendering and is executed using the Python programming language.

Keywords: procedural animation, module, program code, rendering, programming, Python

For citation: Efremenko I. Yu., Pustovaya O. A. Creating a procedural animation. Proceedings from Current issues of energy in the agro-industrial complex: Vserossiiskaya (natsional'naya) nauchno-prakticheskaya konferentsiya. (PP. 262–268), Blagoveshchensk, Dal'nevostochnyi gosudarstvennyi agrarnyi universitet, 2025 (in Russ.).

С развитием человечества технологический прогресс не стоит на месте. Программы также начинают усложняться и требуют внедрения плавающих алгоритмов работы, одним из которых является процедурная анимация. Их использование обусловлено развитием высокоуровневых языков программирования, в частности Python.

Нами предложено использовать этот язык для создания объектов. Обозначим основные этапы программирования поведения объекта.

Подготовка к созданию. Устанавливается язык программирования Python и среда разработки (PyCharm). Создается корневая папка main.py. Добавляется библиотека Pygame через терминал в проект с использованием команды pip install pygame. Пишется код в файле main.py. Именно в этой папке будет конечное использование написанного модуля, являющегося результатом программирования. Он должен быть удобным и понятным для использования.

Создается папка для кода модуля BodyElements.py. В ней пишутся необходимые объекты и их методы, требуемые для создания, моделирования и отображения объектов с процедурной анимацией.

Этапы проектирования модуля:

1. *Создается класс объекта материальной точки – математической окружности, которая будет выступать в качестве сегмента для тела и ножек отображаемого персонажа.* Назовем этот класс Segment. Задание параметров и полей окружности показано на рисунке 1.

```
class Segment:
    def __init__(self, x, y, r): # Передаваемые значения (аргументы)
        self.x = x  # x, y – координаты окружности-сегмента
        self.y = y
        self.radius = r  # Радиус окружности

        self.vector_x = 0  # Направление-поворот окружности(оси)
        self.vector_y = -1

        self.points = [(-r, 0), (r, 0)]  # Точки, лежащие на окружности с учетом ее
направления
```

Рисунок 1 – Создание класса Segment

Для задания поворота осей окружности на угол α используем теорему Пифагора. Контрольные точки показаны на рисунке 2.

$$c = \text{sqrt}\ (a*a + b*b)\ \#$$

Сторона c: self.vector_x = c/a; self.vector_y = c/b

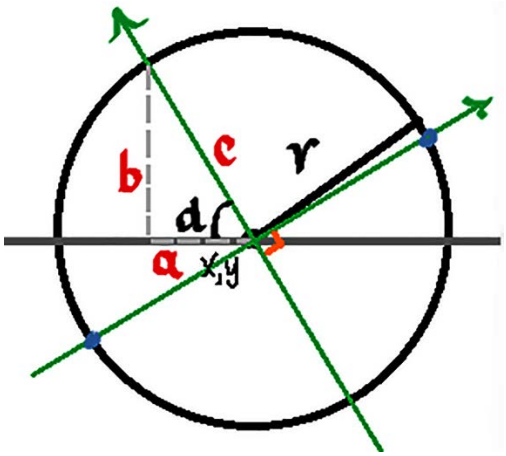

Рисунок 2 – Сегмент

2. *Добавляем классу Segment методы создания новых точек на окружность и получения координат точек окружности с учетом ее поворота* (рис. 3). Теперь есть модули для работы с точками на окружности с учетом направления окружности.

```
# Метод для получение произвольной точки, на направление осей
окружности
def get_pos(self, x, y):
    x_res = self.x + y * self.vector_x + x * -self.vector_y
    y_res = self.y + y * self.vector_y + x * self.vector_x
    return x_res, y_res

# Метод для добавления и сохранения точки на окружность
def add_point(self, pos):
        self.points.append(pos)

# Метод для получения координат сохраненных точек окружности
def get_points_pos(self, index):
    pos = self.get_pos(self.points[index][0], self.points[index][1])
    return pos[0], pos[1]
```

Рисунок 3 – Добавление методов создания новых точек на окружности и получения их координат

3. *Создается класс для объекта ноги-лапы и название Leg* (рис. 4). Этот класс будет отвечать за моделирование структуры и свойств ног или лапок будущих объектов (рис. 5).

```
class Leg:
    def __init__(self, body, x, y, stap_x, stap_y, range_pos, l, r, distance):
        self.segment = body # Сегмент, на котором будет закреплена нога
        self.x = x  # Координаты прикрепления ноги к сегменту
        self.y = y
        self.correct_x = stap_x  # Место, куда нужно будет наступать ноге
        self.correct_y = stap_y
        self.stap_x = stap_x  # Место, где в реальном времени находится
нога
        self.stap_y = stap_y
        self.range_pos = range_pos  # Допустимый радиус отклонения ноги
от места наступления
        self.parts_distance = distance  # Расстояние между частями ноги
        self.parts = []  # Части ноги: объекты класса Segment
        self.l = l  #
        for i in range(l):  # Наполнение ноги
            self.parts.append(Segment(0 + i * 100, i * 10, r))
        p = self.segment.get_pos(stap_x, stap_y)
        self.stap_to(p[0], p[1])
        self.parts[self.l - 1].y = x
        self.parts[self.l - 1].y = y
```

Рисунок 4 – Создание класса Leg

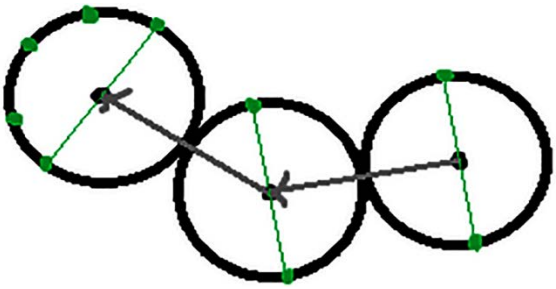

Рисунок 5 – Объект Leg, состоящий из трех сегментов (сегменты – класс Segment)

У ноги есть две части – основание и место наступления. У нее есть метод *update()*, обновляющий положение всех сегментов, из которых состоит нога. Нога – это последовательность окружных сегментов, находящихся на определенном расстоянии друг от друга. Направление каждого сегмента направлено в сторону предыдущего: **def update(self):**.

4. *Добавление в класс ноги необходимых функций и методов* (рис. 6):

```
def update(self):
    #Проверяем находится ли стопа в нужных пределах,если нет, то делаем
шаг
    self.update_stap_pos()
    #Обновляем расстояние сегментов ноги
    for i in range(self.l - 1, 1, -1):
        a = self.parts[i].x - self.parts[i - 1].x
        b = self.parts[i].y - self.parts[i - 1].y
        c = sqrt(a*a + b*b)
        vector_x = 1
        vector_y = 1
        # Если расстояние от сегмента до предыдущего сегмента больше
заданого, то расстояние сегмента меняется на нужное от предыдщего
сегмента.
        if fabs(self.parts_distance - c) > 1:
            if c != 0:
                vector_x = a / c  # Вычислем направление в сторону предыдущей
окружности
                vector  y = b / c
            else:
                vector_x = 0
                vector_y = 0
            # Устанавливаем направление для сегмента
            self.parts[i].vector_x = -vector_x
            self.parts[i].vector  y = -vector  y
            # Изменяем расстояние межу сегментами на нужное
            self.parts[i - 1].x = self.parts[i].x - self.parts_distance * vector_x
            self.parts[i - 1].y = self.parts[i].y - self.parts_distance * vector_y
    for i in range(1, self.l):
        a = self.parts[i - 1].x - self.parts[i].x
        b = self.parts[i - 1].y - self.parts[i].y
        c = sqrt(a * a + b * b)
        vector_x = 1
        vector_y = 1
         # Если расстояние от сегмента до предыдущего сегмента больше
заданого, то расстояние сегмента меняется на нужное от предыдщего
сегмента.
        if fabs(self.parts_distance - c) > 1:
            if c != 0:
                vector_x = a / c
                vector_y = b / c
            else:
                vector  x = 0
                vector_y = 0
            # Изменяем расстояние межу сегментами на нужное
            self.parts[i].x = self.parts[i - 1].x - self.parts_distance * vector_x
            self.parts[i].y = self.parts[i - 1].y - self.parts_distance * vector_y
# Меняем положение основания ноги таким образом, чтобы оно было на
заданной координате на окружности
    pos = self.segment.get_pos(self.x, self.y)    self.head_to(pos[0], pos[1])
```

Рисунок 6 – Добавление в класс ноги необходимых функций и методов

def stap_to(self, x, y): метод для наступления стопы ноги в определенное место (он просто перемещает стопу ноги в определенную координату);

def update_stap_pos(self): метод для обновления места наступления (делает проверку, находится ли стопа ноги в допускаемом пределе от места, куда нужно будет поставить стопу; если условие не удовлетворяется, то стопа перемещается в нужное место);

def head_to(self, x, y): устанавливает основание ноги на определенную координату.

5. *В методе update() проверяются два раза сегменты ноги*: сначала сегменты, начиная с основания ноги; затем снова проверяются сегменты, но уже, начиная со стопы.

Отображение объекта ноги:

1. *Отрисовка ноги будет осуществляться путем получения точек для отрисовки.* Для того, чтобы отобразить ногу, используются точки, которые уже заданы на окружности сегментов ноги, а затем их последовательно соединяют линией и закрашивают полигоном, чтобы получился целостный объект (рис. 7).

```
self.draw_point = []  # Точки для отрисовки
for i in range(self.l * 2 + 3):
    self.draw_point.append([self.x, self.y])
self.parts[0].add_point([-r + r / 3,
                         r - r / 3])
self.parts[0].add_point([0, r])
self.parts[0].add_point([r - r / 3,
                         r - r / 3])

self.color_in = (79, 122, 112)  # цвет для заполнения ноги в формате RGB
self.color_out = (246, 254, 255) # цвет для обводки ноги в формате RGB
```

Рисунок 7 – Получение точек для отрисовки ноги

2. *Производится написание метода для поддержания актуальных координат этих точек:* **update_draw_points()**.

3. *Пишется конечный метод для отображения ноги* (заполнение ноги определенным цветом, а также ее обводка) (рис. 8).

```
def draw(self, sc):  # Метод для отображения ноги по точкам и ее обводка
    pg.draw.polygon(sc, self.color_in, self.draw_point)
    pg.draw.lines(sc, self.color_out, True, self.draw_point, 3)
```

Рисунок 8 – Разработка конечного метода для отображения ноги

Заключение. *В результате разработки модуля удалось создать удобный и понятный программный код, позволяющий быстро, эффективно, а также плавно передавать движения процедурных анимаций любого объекта. В данном случае был создан объект «Тритон» для демонстрации модуля.*

Статья поступила в редакцию 12.12.2024; одобрена после рецензирования 24.12.2024; принята к публикации 04.02.2025.
The article was submitted 12.12.2024; approved after reviewing 24.12.2024; accepted for publication 04.02.2025.