МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего профессионального образования

«Дальневосточный федеральный университет»

Отчет по дисциплине «Проект по компьютерной графике»

Работу выполнил:

Поповкин Артемий Андреевич

Студент группы:

Б9122-02.03.01сцт

Владивосток – 2023 г.

**Введение**

В данном отчете приведены результаты работы по дисциплине «Проект по компьютерной графике».

Целью работы стала поэтапная реализация собственного графического движка с технологией **Raycast**.

Для выполнения поставленной задачи были рассмотрены основные принципы объектно-ориентированного программирования (ООП).

**Используемые средства при разработке**

Для реализации графического движка мной был выбран язык Python, так как он является полностью объектно-ориентированным, предоставляет множество удобных и интуитивно понятных структур данных, а также имеет лёгкий для использования синтаксис.

Используемой средой разработки был выбран PyCharm, так как в нём можно легко подключать нужные пакеты, необходимые для реализации некоторых функций.

**Используемые библиотеки**

В процессе разработки обнаружилась необходимость написать библиотеки для различных «структурных» объектов, которые были в дальнейшем использованы для реализации полноценных фигур и нужных для движка классов.

Ниже перечислены классы, атрибуты и методы, которые было необходимо реализовать для начала работы над движком:

1. **Point**  
   Точка.
   1. Атрибут **coords[x, y, z]** — координаты в 3-х мерном пространстве
   2. Метод **distance(other)** — Дистанция между двумя точками.
   3. Перегрузка математических операторов (сложение / вычитание точек, умножения/деления точки на число (масштабирование).
2. **Vector**Радиус-вектор.
   1. Атрибут **point** — конечная точка радиус-вектора.
   2. Метод **len()** — Вычисление длины вектора **.**
   3. Метод **normalize()** — Нормализация длины вектора.
   4. Метод **rotate(x\_angle, y\_angle, z\_angle)** —Поворот вектора в различных проекциях. Так, **x\_angle** — поворот в плоскости YZ и т. д.
   5. Скалярное и векторное произведения векторов.
   6. Перегрузка математических операторов (сложение / вычитание векторов, умножения/деления вектора на число (масштабирование).
3. **VectorSpace**Система координат в трёхмерном пространстве. Относительно неё строится сцена.
   1. Атрибут **init\_pt** — точка начала координат.
   2. Атрибут **basis[Vector, Vector, Vector]** — базис векторного пространства.
4. **Object**Класс для описания объектов в трёхмерном пространстве.
   1. Атрибут **pos** — местоположение объекта в трёхмерном пространстве. Определяется точкой.
   2. Атрибут **normal** — нормаль объекта. Определяется вектором.
   3. Метод **contains(pt)** — метод, проверяющий содержание точки на поверхности объекта

**Реализация движка**

Результатом реализации движка стала возможность создавать собственные сцены в трёхмерном декартовом пространстве, в которых можно размещать различные объекты. Эти объекты написаны на библиотеках, представленных в предыдущих пунктах.

Также была реализована система событий, с помощью которых различные объекты сцены, в том числе и игрок, могут передвигаться.

Ещё был реализован метод **Raycast** для визуализации сцен в ASCII графике, о котором стоит рассказать чуть подробнее, ибо он является ключевым компонентом в работе камеры.

**Raycast** работает следующим образом: из камеры к каждому пикселю отправляются лучи. Затем каждый из лучей проверяется на пересечение с объектами в сцене. Если одно или несколько пересечений было обнаружено, то определяется расстояние до ближайшего объекта. Необходимо это для того, чтобы определить, какой символ использовать для отображения глубины.

При реализации **Raycast** была встречена проблема с так называемым «рыбьим глазом», т. е. объекты отображались искажённо, непропорционально.

Для исправления этой проблемы было принято решение проецировать лучи на касательную к сфере. Это позволяет избежать искажений перспективы и правильно отображать объекты на экране.

Для работы с объектами и для наполнения сцен, на основе вышеописанных библиотек было реализовано несколько технических классов, использующихся только для вычисления различных необходимых параметров, а также 4 объекта, являющихся непосредственно участниками сцен:

1. **Camera**Камера, способная получать информацию об окружающем мире.  
   Имеет 2 вида: **Spectator** и **Player**. **Spectator** не имеет коллизии и может спокойно проходить через объекты, когда как **Player** этого делать не может.
   1. Атрибут **pos** — Положение камеры.
   2. Атрибут **look\_dir** — Направление камеры.
   3. Атрибуты **fov и vfov** — Горизонтальный и вертикальный углы обзора.
   4. Атрибут **draw\_dist** —Дистанция прорисовки.
   5. Метод **send\_rays()** — Отправка лучей к каждому пикселю экрана.
   6. Метод **rotate(x\_angle, y\_angle, z\_angle)** — Поворот камеры.
2. **Map**  
   Список объектов в сцене. Используется для нахождения пересечений луча с объектами.
   1. Атрибут **obj\_list[Object]** — сам список объектов.
   2. Метод **append(objects)** — добавление объектов в список.
3. **Ray**Луч. Используется для камеры.
   1. Атрибут **inpt** — начальная точка.
   2. Атрибут **dir** — направление луча.
   3. Метод **intersect(mapping)** — Определение пересечения луча с объектами из списка объектов. Если есть пересечение, определяет расстояние.
4. **Canvas**Канва для отображения трехмерных объектов, используя заданный объект карты и камеру.
   1. Атрибут **map** —карта объектов.
   2. Атрибут **cam** — камера.
   3. Метод **update**() — возвращает матрицу расстояний для всех лучей, отправленных с помощью метода **send\_rays()** у **Camera**. Если луч не пересёк ни один объект, или объект оказался вне дальности прорисовки, то для него расстоянием будет None.
5. **Console**  
   Класс, предназначенный для отрисовки объектов на экране консоли.
   1. Атрибут **symbols()** — список символов, отвечающих за отрисовку изображений, где первый символ отвечает за самое большое расстояние, а последний — за самое ближнее.
   2. Метод **draw()** — посимвольно отображает объекты на экране консоли, в соответствии с матрицей расстояний, полученной с помощью метода **update()** у класса **Canvas**.
6. **Parameters** и его различные наследники для разных объектов.  
   Класс, содержащий в себе основные параметры различных объектов.  
   В зависимости от объектов, класс может содержать различные атрибуты, но есть два общих атрибута для любого объекта.
   1. Атрибут **pos** — позиция объекта. Определена точкой.
   2. Атрибут **normal** — нормаль объекта. Определена вектором.
   3. Метод **move(move\_to: Point)** —передвижение на координаты точки. Передвижение происходит через суммирование начальных координат и координат точки **move\_to**
   4. Метод **scaling(value)** — сжатие или расширение объекта.
   5. Метод **rotate(x\_angle, y\_angle, z\_angle)** — поворот объекта.
7. **Plane**Плоскость.
   1. Атрибут **pr** — параметры плоскости. Содержит **pos** и **normal**.
   2. Метод **contains(pt: Point, delta)** — определение, находится ли точка в плоскости.
   3. Метод **intersect(ray)** — проверка на пересечение луча с плоскостью.
8. **BoundedPlane**  
   Ограниченная плоскость.
   1. Атрибут **pr** — параметры ограниченной плоскости. Содержит **pos**, **normal**, **u**, **v**, **du**, **dv**, где **u** и **v** —векторы ширины и высоты, а **du** и **dv** — ширина и высота соответственно.
   2. Метод **contains(pt: Point, delta)** — определение, находится ли точка в плоскости.
   3. Метод **intersect(ray)** — проверка на пересечение луча с плоскостью.
9. **Sphere**Сфера.
   1. Атрибут **pr** — параметры сферы. Содержит **pos**, **normal**, **radius**.
   2. Метод **contains(pt: Point, delta)** — определение, находится ли точка на поверхности сферы.
   3. Метод **intersect(ray)** — проверка на пересечение луча со сферой.
10. **Cube**  
    Куб.
    1. Атрибут **pr** — параметры куба. Содержит **pos**, **limit**, **normal**, **u**, **v**, **edges[BoundedPlane]**, где **limit** — половина длины ребра куба, edges — массив рёбер граней.
    2. Метод **contains(pt: Point, delta)** — определение, находится ли точка на поверхности куба.
    3. Метод **intersect(ray)** — проверка на пересечение луча с кубом.

Помимо различных геометрических объектов была реализована упомянутая мной ранее система событий, позволяющая участникам сцен перемещаться по ней, вращаться и изменять свой размер.

Используется она, как ясно из названия, во время оперирования различными событиями, такими как движения объектов и нажатия клавиш на клавиатуре.

Система событий работает следующим образом:

Есть множество событий, в котором содержатся названия различных событий.

К каждому из этих событий можно добавить несколько уникальных функций так, что при выполнении этого события каждая из этих функций будет выполняться.

У класса, отвечающего за систему событий в данной программе, есть следующие атрибуты и методы:

1. **Events**  
   Множество событий.
   1. Атрибут **event\_data** — само множество событий.
   2. Метод **add(event: string)** — добавление событий по названию.
   3. Метод **handle(event: string, function)** — добавление функции к событию.
   4. Метод **remove(event: string, function)** — удаление функции из события.
   5. Метод **trigger(event, args)** — Вызывает функции, принадлежащие событию **event** с аргументами **args**.

В завершении презентации остановимся на связующем звене всего движка — функции **launch()**

В ней обрабатываются все поступаемые от пользователя команды, такие как различные сочетания клавиш и повороты мышкой.

**Вывод**

В результате выполнения этой курсовой работы я научился применять знания по аналитической геометрии в реализации сложных конструкций: выстраивать объекты в пространстве, находить пересечения между объектами и выпускаемыми из камеры лучами для рисования в консоли.

Ещё были получены базовые навыки работы в сфере объектно-ориентированного программирования.

Теперь у меня есть понимание того, как нужно выстраивать линии наследований атрибутов и методов, а также базовые навыки ведения документации.