# Лабораторные работы по дисциплине «Аналитическая геометрия и компъютерная графика»

П.А. Максимов

Данный документ представляет из себя набросок заданий к каждой лабораторной работе

# Содержание

# Введение

Лабораторные работы по аналитической геометрии реализуются в формате программирования с использованием парадигм ООП для создания собственного 3D движка на основе технологии RayCast.

Каждая работа включает в себя реализацию определенных классов, способных реализовывать ту или иную спецификацию движка, для получения конечного продукта, готового к использованию.

# Лабораторная работа №1

Первая работа заключается в реализации базовых абстрактных классов для определения системы координат в трехмерном пространстве и абстрактных объектов внутри этой системы, задаваемых с помощью уравнений. К базовым классам относятся – точка (`Point`), вектор (`Vector`), система координат относительно экрана (`CoordinateSystem`), камера (`Camera`), объект (`Object`).

Каждый класс представляет из себя определенный набор свойств и методов:

## Point

`Point` – точка в трехмерном пространстве:

- $x, y, z$ – координаты в трехмерном пространстве;
- `distance(Point)` – метод поиска расстояния между двумя точками;
- Перегрузка операторов сложения/вычитания точек, умножения/деления точки на число (масштабирование).

## Vector

`Vector` – наследник класса `Point`, дополнительно включает:

- `length()` – метод определения длины вектора;
- `scalarProduct(Vector)` – метод, определяющий скалярное произведение;
- `vectorProduct(Vector)` – метод, определяющий векторное произведение;

- Перегрузка отдельных операторов для вышенаписанных методов (`len`, умножение для скалярного произведения векторов, и степенное выражение ($\wedge$, $**$) для векторного произведения).

### CoordinateSystem

`CoordinateSystem` – система координат в трехмерном пространстве. Будет использована для определения единиц измерения в контексте канваса (для отображения на экране):

- `unitLength` – длина единичного вектора на экране.

### Camera

`Camera` – объект камеры, способной получать информацию об окружающем мире:

- `position` – объект класса `Point` – местоположение в трехмерном пространстве;
- `lookAt` – объект класса `Vector` – направление просмотра камеры;
- `FOV` – угол обзора камеры (в радианах);
- `drawDistance` – дистанция отрисовки лучей.

### Object

`Object` – абстрактный пустой родительский класс для описания объектов в трехмерном пространстве:

- `position` – объект класса `Point` – местоположение в пространстве;
- `rotation` – информация о повороте объекта;
- `contains(Point)` – метод, проверяющий содержание точки на поверхности объекта.

## Лабораторная работа №2

Вторая лабораторная работа заключается в реализации абстрактных объектов с параметрами, и их взаимодействия между точками пространства.

### Объекты реализации

1. Реализовать класс `Parameters` для определения параметров объекта. В качестве свойств – коэффициенты уравнения, в качестве методов – методы преобразования коэффициентов путем поворота, перемещения и/или масштабирования;

2. Для объекта (`Object`) реализовать метод-заглушку определения ближайшей точки из набора;

3. Для объекта реализовать метод-заглушку `intersect(Point, Vector)` для поиска точки пересечения прямой, задаваемой через точку и направление;

4. Для объекта ввести свойство `parameters`, чтобы передавать в него параметры объекта (для составления уравнений);

5. Для камеры (`Camera`) реализовать метод определения минимального расстояния до объекта через метод `intersect` у объекта, в зависимости еще и от свойства `drawDistance`;

6. Для камеры определить свойство количества разбиений экрана на блоки, для того, чтобы затем по ним рассчитывать угол разброса лучей;

7. На основе класса объекта реализовать класс плоскости (`Plane`), с наследованием всех свойств, и перегрузкой метода `contains` и `intersect` в зависимости от уравнения плоскости;

8. Для плоскости задать наследника класса `Parameters` – `ParametersPlane`, который содержит коэффициенты $A, B, C, D$ уравнения плоскости в качестве свойств, и умеет их преобразовывать в зависимости от операций поворота, перемещения и масштабирования.

## Лабораторная работа №3

В третьей лабораторной работе реализуются классы карты (`Map`) и луча (`Ray`), базовые классы отрисовки, и уже затем реализуется вертикальная отрисовка в консоли. Требуется реализовать классы `Canvas` и `Console` – наследник класса `Canvas`. Так же требуется реализовать для камеры методы отправки горизонтальных лучей по всему углу обзора, чтобы получить информацию о расстоянии, чтобы отрисовать потом в классе отрисовки.

Вводим понятие карты (`Map`), состоящей из объектов подклассов `Object` (на данный момент уже будут готовы `Plane`), и `Ray` – луч, имеющий начальную точку и направление.

## Map

- `objects` – массив объектов класса `Object` и его наследников;
- `append(Object)` – метод добавление объекта на карту (в массив объектов).

## Ray

- `position` – точка (`Point`) начала луча;
- `direction` – направление луча (`Vector`);
- `intersect(Map)` – метод поиска ближайшей точки пересечения с объектами на карте.

## Object

1. Для объекта переписать метод-заглушку `intersect(Point, Vector)` на `intersect(Ray)`, для поиска точки пересечения прямой, задаваемой через точку и направление;

## Canvas

`Canvas` – абстрактный класс отрисовки на "несуществующем"экране, базовый класс:

- `map` – объект класса `Map` – карта;
- `camera` – объект класса `Camera`;
- `coordsystem` – объект класса `CoordinateSystem` – система координат для класса отрисовки с заданным параметром длины единичного вектора для конкретного класса канваса;
- `draw()` – метод отрисовки проекции карты `map` на камеру `camera` относительно текущего формата отрисовки.

### Console

`Console(Canvas)` – консольный метод отрисовки, наследует базовый класс отрисовки `Canvas`:

1. Требуется перегрузить метод `draw` для отрисовки конкретно в консоль;
2. Ввести свойство градиента символов для отрисовки в зависимости от дальности объекта на экране.

### Camera

1. Добавить метод `sendRays(Map)`, который просчитывает углы и отправляет лучи `Ray` в зависимости от карты и обзора камеры через их метод `intersect` (пока только отправлять лучи горизонтально).

Так же требуется реализовать класс `Sphere` – наследник класса `Object`, реализующий сферу в трехмерном пространстве. Для нее реализовать класс `ParametersSphere` для работы с уравнением сферы/эллипсоида.

Затем можно реализовать `Main`-класс движка с отрисовкой в консоли – базовый движок рисования объектов с вертикальным разбиением камеры на блоки будет готов.

## Лабораторная работа №4

Четвертая работа заключается в реализации отправки вертикальных лучей, реализации поворотов объектов и камеры с помощью углов Эйлера (`Angle`), и исправления таких особенностей проекций трехмерных объектов на двухмерную плоскость камеры, как «рыбий глаз». Помимо этого, требуется реализовать классы игроков, способных перемещаться по карте свободно (`Spectator/FreeCamera`), и с коллизией (`Player`):

### Camera

1. Ввести свойство `VFOV` – вертикальный угол отрисовки;
2. В метод `sendRays` добавить отправку вертикальных лучей – разбиение экрана на «блоки»;
3. В методе `sendRays` для лучей просчитывать расстояния до объектов со включением угла отклонения луча от центра камеры (исправление особенности «рыбий глаз»).

### Angle

`Angle(Point)` – класс угла поворота, наследуется от класса `Point`:

1. Перегрузить операторы преобразования координат (сумма/произведение, и тд) по кольцу вычетов $2\pi$;
2. Здесь $x, y, z$ – соответствующие углы поворота в плоскостях $YZ, XZ, XY$.

### Spectator/FreeCamera и Player

`Spectator/FreeCamera` и `Player` – классы управления игроком, содержат свойства:

- `camera` – объект класса `Camera`, позволит отрисовывать вид из глаз персонажа;
- `position` – положение игрока в пространстве, будет переписывать свойства положения камеры;
- `lookAt` – вектор направления просмотра, будет переписывать свойства просмотра камеры.

### Описание действия камеры игрока

Камера для объекта может и отсутствовать, если у игрока не требуется отображать вид в камеру.

Объекты классов `Spectator/FreeCamera` и `Player` умеют перемещаться в пространстве. Требуется реализовать методы перемещения вперед/назад и по сторонам (`strafe` – движение без поворота камеры в стороны влево/вправо, и `rotate` – поворот камеры по сторонам). Объекты класса `Spectator/FreeCamera` могут проходить сквозь объекты, в то время как `Player`, помимо всего рассмотренного, должен еще иметь свойство коллизии, и представлять из своей колизионной модели цилиндр ограниченной высоты (в идеале – капсулу), который при столкновении с объектами будет идти со скоростью, зависимой от угла движения и направления касательной к объекту). Пока что нижняя часть не играет никакой роли, коллизия только с боковыми обьектами.

## Лабораторная работа №5

Пятая лабораторная работа заключается в реализации простейшей системы событий, введения рисования в графический канвас, преобразо-

вания углов с помощью кватернионов, и формирования отчета о проделанной работе.

## Описание системы событий

Система событий (`Event/Trigger`) должна представлять из себя систему обмена информацией между элементами движка. `Event` должен уметь создавать прослушивающие системы «облака» событий, в то время, как `Trigger` должен отправлять информацию о совершенном событии в это облако. Прослушивание событий представляет собой бесконечный цикл, в котором ставится условие на появление события заданного имени в облаке событий.

## Event

Класс `Event` содержит:

- `add(name)` – метод добавления события «`name`» в пространство имён событий
- `handle(name, callfunction)` – метод установки обработчика на событие name функции `callfunction` нескольких аргументов. Это значит, что если событие получило триггер события `name`, то в системе сразу же вызывается функция `callfunction` с аргументами, переданными триггером. Принцип работы – создает в себе бесконечный цикл, имеющий условие исполнения функции, и условие выхода – окончание существования программы или удаление системы событий.

## Trigger

Класс `Trigger` содержит:

- `trigger(name, *arguments)` – метод отправки триггера события «`name`» в пространство событий с набором каких либо аргументов. Функции, которые поймали событие (`handler`'ы), вызывают свои функции `callfunction` именно с этим набором аргументов. Аргументы представляют из себя список передаваемых значений, которые будут использоваться в обработчике.

## Quaternion

Класс `Quaternion` описывает все действия, связанные с кватернионами. Базовый рабочий класс кватернионов предоставляется (как класс чисел, со всеми возможными операциями), задача заключается в реализации углов поворота класса `Angle` на основе нового класса. Теперь поворот должен содержать два параметра, вместо трех:

- `axis` – объект класса `Vector`, описывает направление оси, вокруг которой будет производиться поворот;
- `angle` – угол поворота (в радианной мере) вокруг оси `axis`.

## Описание задания

В качестве задания, с целью оптимизации движка, реализовать на основе `Event-Trigger` отрисовку изображения только по достижению условия отрисовки, а не бесконечно, как это реализуется по умолчанию. Например: игрок переместился вперед – отрисовать новую картинку.

Перевести все известные повороты из углов Эйлера в кватернионы.

Реализовать графический канвас, на основе реализации аналогичного из консоли, только теперь в попиксельном и цветном формате (изначально можно в оттенках серого).

Сформировать отчет о проделанной работе, и работоспособности проекта, реализовать презентацию проекта, и продемонстрировать его окончательную работу.