ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт прикладной математики и механики

Кафедра теоретической механики

**КУРСОВОЙ ПРОЕКТ**

по учебной дисциплине

**«Программирование»**

на тему:

**МоДелирование Динамической Системы частиц**

**Выполнили:**

студенты 1 курса дневного отделения

**Абрамов Игорь Алексеевич,**

**Нарядчиков Александр Владимирович**

обучающиеся по профессиональной образовательной программе подготовки бакалавров

«01.03.03 – Механика и математическое моделирование»

**Научный руководитель:**

**Панченко Артем Юрьевич**

Санкт - Петербург

2016

Оглавление

[**Введение** 3](#_Toc453926615)

[**Цель** 4](#_Toc453926616)

[**Задачи** 4](#_Toc453926617)

[**Математическая составляющая** 5](#_Toc453926618)

[**Класс векторов** 5](#_Toc453926619)

[**Класс кватернионов** 5](#_Toc453926620)

[**Класс базиса** 5](#_Toc453926621)

[**Использование математической библиотеки в программе** 6](#_Toc453926622)

[**Графический интерфейс** 7](#_Toc453926623)

[**Шейдера** 7](#_Toc453926624)

[**Буферы** 8](#_Toc453926625)

[**Анимация** 8](#_Toc453926626)

[**Система управления камерой** 9](#_Toc453926627)

[**Перспективы развития проекта** 10](#_Toc453926628)

[**Список литературы** 11](#_Toc453926629)

[**Ссылки** 11](#_Toc453926630)

# **Введение**

Для решения многих задач термодинамики требуется нахождение не только макропараметров системы, но и микропараметров отдельных частиц, из которых она состоит. Решение этой задачи сводится к нахождению координат и скоростей молекул в каждый момент времени с помощью исходных данных о начальных координатах и скоростях, а также типе движения и взаимодействия частиц между собой. Написание программы для вычисления этих данных с помощью методов численного интегрирования не составит особого труда, но, так как количество молекул воздуха в 1 см3 при нормальных условиях составляет порядка 1019, то отладка подобных программ и нахождение в них ошибок затрудняется большим объёмом вычисляемых данных. Кроме того, по полученным данным не всегда понятно, по какой траектории движется та или иная частица, а также провзаимодействовала ли она с стенками сосуда или с другими частицами. Именно поэтому для решения подобных задач приобретают принципиальную значимость системы для моделирования динамики процессов, происходящих в жидкостях и газах. Существует большое количество программных пакетов, моделирующих обтекание газом или жидкостью того или иного объекта, но практически нет таких, которые бы позволили смоделировать внутренние процессы в данной среде, задаваемые пользовательскими уравнениями. После краткого анализа существующих программных пакетов было принято решение написать собственную программу для графического моделирования динамики системы множества частиц, которая позволит пользователю загружать файлы специального формата с расчётами скоростей и координат молекул, а результатом выполнения программы станет визуальное представление модели, позволяющее перемещать камеру для обзора сосуда со всех сторон и просматривать положения частиц в выбранные моменты без произведения расчётов в режиме реального времени.

# **Цель**

Создание программы, визуализирующей динамическое развитие системы множества частиц.

# **Задачи**

* Создание математической библиотеки для работы с векторами, кватернионами и базисами;
* Создание класса анимации;
* Создание собственного компилятора шейдеров GLSL;
* Написание системы управления камерой с помощью мыши и клавиатуры;
* Отправление буфера координат и цветов точек на видеокарту.

# **Математическая составляющая**

Для реализации возможности управления камерой с помощью мыши и клавиатуры, а также для упрощения работы с положениями и скоростями молекул была написана математическая библиотека MTH.H. В её состав входят класс векторов, класс кватернионов и класс базисов пространства.

## **Класс векторов**

Для удобства дальнейшей работы с программой для векторов были перегружены все необходимые операции, такие как сложение и произведение, умножение на число, нормализация и инвертирование векторов, а также обращение к составляющим вектора по индексу. Класс написан с использованием шаблонов, поэтому при инициализации вектора в программе можно выбрать тип данных его составляющих.

vec<double> LastPos;

Кроме того, для удобства дальнейшей работы с базисами были написаны специальные функции поворота вектора вокруг заданной оси на определённый угол или просто с использованием кватерниона поворота.

vec Rotate(vec pivot, Type angle);

vec Rotate(quat<Type> rotation\_q);

## **Класс кватернионов**

Кватернионы предоставляют удобное математическое обозначение положения и вращения объектов в пространстве. В сравнении с углами Эйлера, кватернионы позволяют проще комбинировать вращения, а также избежать проблемы, связанной с невозможностью поворота вокруг оси, независимо от совершённого вращения по другим осям, известной как “Gimbal Lock”. Для работы с кватернионами были перегружены операции умножения на вектор и на кватернион, нормализации и инвертирования кватернионов. Класс также использует шаблон для удобства изменения типов данных.

quat<double> q;

**Класс базиса**

Для удобства ориентации в пространстве, а также инициализации дополнительных камер (при необходимости) был написан класс базиса, который хранит в себе информацию о координате расположения базиса в пространстве относительно базиса объекта, и направлении трёх его осей: Right, Up и Direction. В данном классе реализованы функции поворота базиса относительно оси на определённый угол и с помощью кватерниона вращения.

void Rotate(vec<Type> V, Type alpha); void Rotate(quat<Type> rotation\_q);

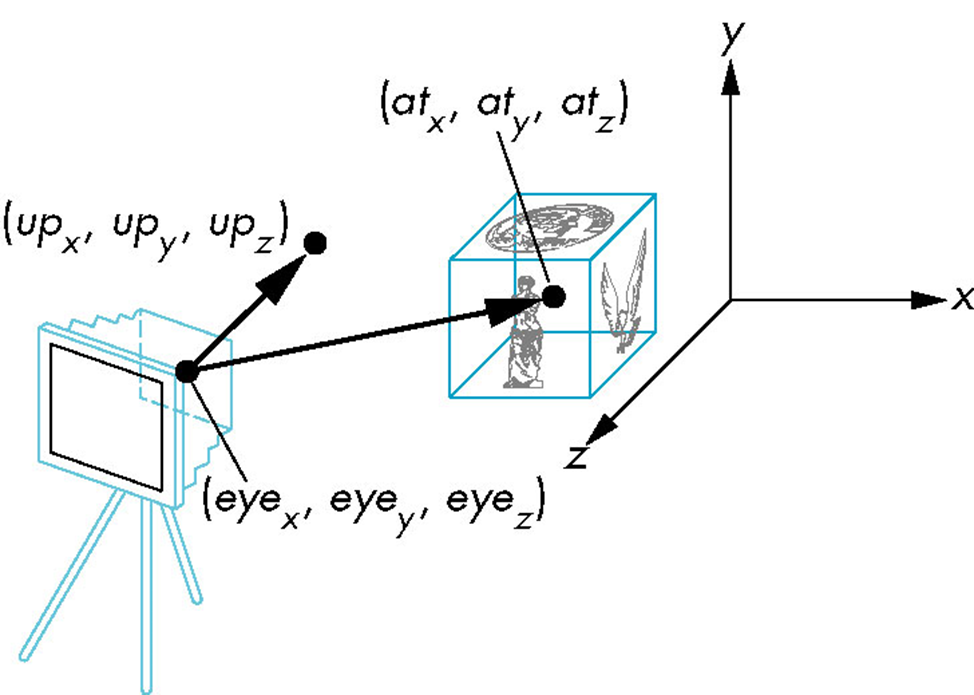
## **Использование математической библиотеки в программе**

Математическая библиотека используется в программе для позиционирования и поворота камеры в пространстве. В классе Instance, находящемся в файле ANIM.H инициализированы два базиса: Eye – базис камеры, Center – базис сосуда. Они необходимы для дальнейшего позиционирования камеры в определённую точку с помощью функции GluLookAt. Она принимает на вход параметры векторов, указанные на изображении 1.

gluLookAt(Instance.Eye.Pos.X, Instance.Eye.Pos.Y, Instance.Eye.Pos.Z,

Instance.Center.Pos.X, Instance.Center.Pos.Y, Instance.Center.Pos.Z,

Instance.Center.Up.X, Instance.Center.Up.Y, Instance.Center.Up.Z);



Изображение 1 – расположение объектов и необходимые для функции gluLookAt данные.

Далее в специальных функциях OpenGl, отвечающих за получение данных с мыши и клавиатуры в зависимости от нажатых клавиш осуществляется поворот базиса Eye в пространстве на соответствующие углы, в зависимости от длительности нажатия клавиатуры. Кроме того, с помощью мыши можно изменить расстояние до объекта, сохраняемое в переменной Instance.Radius.

# **Графический интерфейс**

Программа написана с использованием библиотеки OpenGL и языка высокого уровня для программирования шейдеров GLSL (OpenGL Shading Language) версии 1.30.10.

* Создания собственной системы анимации и “стека” для объектов;
* Написание камеры на кватернионах;
* Работа с буферами точек (частиц) на шейдерах.

## **Шейдера**

Было использовано 2 типа шейдеров: вершинный и фрагментный. Вершинный шейдер оперирует данными, связанными с координатами точек в пространстве и с их цветом.

#version 130

in vec4 pos;

in vec4 color;

out vec4 color\_from\_vshader;

void main( void )

{

gl\_PointSize = 1.0;

gl\_Position = pos;

color\_from\_vshader = color;

gl\_Position = gl\_ModelViewProjectionMatrix \* pos;

}

Фрагментный шейдер работает с фрагментами растрового изображения — обрабатывает данные, связанные с пикселями (цвет и глубина). Фрагментный шейдер используется на последней стадии графического конвейера для формирования фрагмента изображения. Также был создан свой компилятор для шейдеров GLSL.

#version 130

in vec4 color\_from\_vshader;

out vec4 out\_color;

void main( void )

{

out\_color = color\_from\_vshader;

}

## **Буферы**

Vertex Buffer Object (VBO) – средство OpenGL, позволяющее загружать определенные данные в память GPU. Например, сообщить GPU координаты вершин и цвета.

Vertex Arrays Object (VAO) – средство OpenGL, которое сообщает, какую часть VBO следует использовать в последующих командах. Таким образом, один VAO по разным индексам может хранить координаты вершин и их цвета. Переключившись на нужный VAO мы можем эффективно обращаться к данным, на которые он «указывает», используя только индексы.

// Create a vertex array object

GLuint vao;

// Create a Vector Buffer Object that will store the vertices on video memory

GLuint vbo;

// Use a Vertex Array Object

glGenVertexArrays(1, &vao);

glBindVertexArray(vao);

glGenBuffers(1, &vbo);

// Allocate space and upload the data from CPU to GPU

glBindBuffer(GL\_ARRAY\_BUFFER, vbo);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(random\_position) + sizeof(colors), NULL, GL\_STATIC\_DRAW);

// Transfer the vertex positions

glBufferSubData(GL\_ARRAY\_BUFFER, 0, sizeof(random\_position), random\_position);

// Transfer the vertex colors

glBufferSubData(GL\_ARRAY\_BUFFER, sizeof(random\_position), sizeof(colors), colors);

...

// Draw Array

glBindVertexArray(vao);

glDrawArrays(GL\_POINTS, 0, (SIZE / 3));

## **Анимация**

Создан свой класс анимации типа Singleton.

Singleton – порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр класса с глобальной точкой доступа, то есть запрещается создание нескольких экземпляров того же типа.

Создан свой “стек” для хранения множества объектов.

# **Система управления камерой**

При нажатии правой кнопки мыши и перемещении указателя курсора камера вращается с зафиксированным радиусом относительно сосуда вокруг правого и верхнего векторов камеры пропорционально смещению указателя по оси y и x соответственно. При зажатии левой кнопки мыши и перемещении указателя курсора изменяется расстояние от центра сосуда до объекта пропорционально смещению указателя по оси y. Поворот вокруг базисных векторов сосуда осуществляется с помощью клавиш “A” и “D” (“против часовой” и “по часовой” вокруг вектора Up соответственно), “W” и “S” (“против часовой” и “по часовой” вокруг вектора Right соответственно). По нажатию кнопки “X” происходит обновление сцены и загрузка нового кадра. Кнопка “P” включает и выключает режим паузы. При нажатии “F” включается режим полного экрана.

# **Перспективы развития проекта**

1. Использование других способов поворота базисных векторов для выявления лучшего из них.
2. Улучшение плавности поворота камеры с помощью задания скоростных коэффициентов, добавление инерции при повороте.
3. Улучшение пользовательского интерфейса, добавление меню и элементов управления.
4. Получение характеристик отдельной частицы в данный момент времени путём запуска луча.
5. Переход на более новые графические библиотеки (Vulkan API).
6. Доработка системы работы с файлами.

# **Список литературы**

1. OpenGL Development Cookbook by Muhammad Mobeen Movania
2. OpenGL 4 Shading Language Cookbook - Second Edition by David Wolff

# **Ссылки**

1. <https://www.opengl.org/>
2. <https://www.opengl.org/resources/libraries/glut/>
3. <https://www.opengl.org/documentation/glsl/>
4. <http://glew.sourceforge.net/>
5. <https://habrahabr.ru/post/255005/>
6. <http://www.rossprogrammproduct.com/translations/Matrix%20and%20Quaternion%20FAQ.htm#Q47>
7. <http://www.gamedev.ru/code/articles/?id=4215>

# **Приложение**

Изображения 1 - 3 – скриншоты работы программы.



