Оглавление

[ВВЕДЕНИЕ 3](#_Toc326168083)

[Обзор БПЛА типа мультикоптер 8](#_Toc326168084)

[Виртуальный полигон 10](#_Toc326168085)

[Роторы 12](#_Toc326168086)

[6-DOF тело 14](#_Toc326168087)

[Модель потоков газа в замкнутых пространствах 16](#_Toc326168088)

[Численное решение уравнений Навье-Стокса 18](#_Toc326168089)

[Адвекция 23](#_Toc326168090)

[Диффузия и вязкость 25](#_Toc326168091)

[Решение уравнений Пуассона 25](#_Toc326168092)

[Начальные и граничные условия 27](#_Toc326168093)

[Реализация ВП 28](#_Toc326168094)

[CUDA реализация Навье-Стокса 28](#_Toc326168095)

[Бортовое оборудование 35](#_Toc326168096)

[Гироскоп 35](#_Toc326168097)

[Акселерометр 36](#_Toc326168098)

[Сонар 37](#_Toc326168099)

[Магнитометр 37](#_Toc326168100)

[Барометр 38](#_Toc326168101)

[Система глобального позиционирования 39](#_Toc326168102)

[Камеры 39](#_Toc326168103)

[Средства анализа данных 40](#_Toc326168104)

[Система оптического захвата движений 40](#_Toc326168105)

[Система управления 46](#_Toc326168106)

[ЗАКЛЮЧЕНИЕ 47](#_Toc326168107)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 48](#_Toc326168108)

# ВВЕДЕНИЕ

Актуальность

В современном мире все большее распространение получают беспилотные летательные аппараты. Беспилотный летательный аппарат (БПЛА) — летательный аппарат многоразового использования без экипажа на борту.

Такие БПЛА применяются, наиболее часто, в качестве робототехнического средства, способного выполнять технологические операции в опасных для человека зонах (инженерная, радиационная, химическая и биологическая разведка). Также возможно осуществление контроля трубопроводов на газовых, химических и нефтяных магистралях в целях предупреждения аварий, охраны лесов и рыбоохраны, ледовой разведки, наблюдение за движением на дорогах. Так, к примеру, на рисунке 1 представлен гексакоптер, который недавно проходил испытания в ГИБДД ГУ МВД России по Московской области, с его помощью можно искать угнанные автомобили в местах недоступных для сотрудников полиции. Также существую разного рода соревнования, в которых БПЛА должны самостоятельно выполнить ряд заданий, к примеру, существует международный конкурс летающих роботов, созданный Робертом Михельсоном. Данный конкурс был создан для того чтобы стимулировать создание небольших, но очень умных БПЛА, способных автономно выполнять сложные задачи.



Рис. 1. Гексакоптер используемый в ГИБДД ГУ МВД России по Московской области.

Возможности изучения поведения БПЛА в экстремальных ситуациях экспериментальными методами сильно ограничено, как в связи с дороговизной самого аппарата, так и высокой стоимостью ошибки, которая может возникнуть в ходе эксперимента. Поэтому для этих целей разумно использовать компьютерные эксперименты в реальном времени. Для интерпретации его результатов привлекаются технологии виртуальной реальности, обеспечивающие «погружение» исследователя в моделируемое явление с возможностью всестороннего наблюдения и анализа воспроизводимых закономерностей реального мира. В свою очередь, это стимулирует развитие нового класса проблемно-ориентированных программных комплексов для проведения вычислительного эксперимента – виртуальных полигонов (ВП) для поддержки принятия решений в различных областях науки и промышленности. Процесс проектирования и разработки ВП требует совокупного учета особенностей методов компьютерного моделирования в конкретной предметной области и соответствующих возможностей технологий виртуальной реальности, включая специфику аппаратной реализации. Это достигается путем адаптации математических моделей для формирования предметно-зависимых визуальных динамических сцен с высоким уровнем реалистичности и достоверности.

Мотивацией для создания виртуального полигона послужило следующее:

* Наличие реального объекта и/или объектов в должном количестве – цена на подобные БПЛА может сильно варьироваться в зависимости от характеристик аппарата, но все же остается достаточно высокой.
* Износ оборудования
* Стоимость ошибки на реальном объекте может быть весьма высока
* Эксперимент на реальном объекте требует времени на подготовку и развертывание эксперимента, а также на приведение объектов в исходное состояние на каждой итерации

Цель и задачи исследования

* Анализ математических моделей динамики четырехроторных летательных аппаратов.
* Разработка метода численного моделирования динамики групп четырехроторных летательных аппаратов с учетом взаимного влияния.
* Проектирование, разработка и отладка программно-аппаратного комплекса виртуального полигона для исследования динамики групп четырехроторных летательных аппаратов с учетом взаимного влияния.

Область исследования

Теория управления, гидромеханика, теория динамики твердого тела, имитационное моделирование.

Предметом исследования является технология создания виртуального полигона применительно к задачам моделирования динамики групп четырехроторных летательных аппаратов.

Теоретическая и методологическая основа исследования включает в себя методы вычислительной гидромеханики, теории динамики твердого тела, имитационного моделирования, обработки изображений и научной визуализации.

Научная новизна исследования

Использование метода прямого моделирования динамики БПЛА с учетом аэродинамического взаимодействия с другими БПЛА и элементами замкнутых помещений в реальном масштабе времени, моделирование бортового оборудования, позволяющего проводить разнообразные эксперименты для отработки и совершенствования алгоритмов управления БПЛА.

Практическую значимость исследования составляет

Программно-аппаратный комплекс виртуального полигона QuadroX-DS, в состав которого входят сам программный комплекс, комплект оборудования для захвата движения и четырехроторный БПЛА.

Апробация результатов исследования

Изложенные в диссертации результаты обсуждались на одной IV международной научной конферернции MEDIAS-2012 (2012 г., Лимасол, Кипр).

Объем и структура работы

# Обзор БПЛА типа мультикоптер

Мультикоптер, как правило, состоит из следующих объектов:

* Рама — является несущей конструкцией, на которой монтируются остальные компоненты БПЛА, а также дополнительное оборудование, такое как фотокамеры, различные измерители, полезные грузы и т.д.. Также выполняет функцию защиты хрупких компонент в случае столкновения или падения БПЛА. В некоторых случаях, рама позволяет варьировать длину плеча.
* Двигатели – количество двигателей должно быть четным, при этом одна половина двигателей должна крутиться в противоположном направлении относительно другой половины, это необходимо для того чтобы взаимно погашать крутящий момент, который возникает при вращении роторов.
* Набор сенсоров:
  + Акселерометр – позволяет определять скорость перемещения и ориентацию БПЛА;
  + Гироскоп – позволяет определять вращение БПЛА в пространстве;
  + Сонар – позволяет определять расстояние до объектов, к примеру, полов, стен, потолка;
  + Барометр – позволяет определять высоту, на которой находится БПЛА;
  + Магнитометр – позволяет определять направление четырех сторон света, так же может быть использован для поиска электромагнитных аномалий.
  + GPS – позволяет определять глобальные координаты БПЛА, к примеру, если ЛА предназначен для поиска людей в лесу или на льдинах
  + Контроллер — выполняет функцию стабилизации БПЛА, приема и передачи управляющих сигналов, телеметрии и прочих данных.
  + Аккумулятор (LiPo)

На рисунке 2 представлен квадрокоптер, на котором проводились все эксперименты.



Рис. 2. Квадрокоптер, на котором происходили тесты.

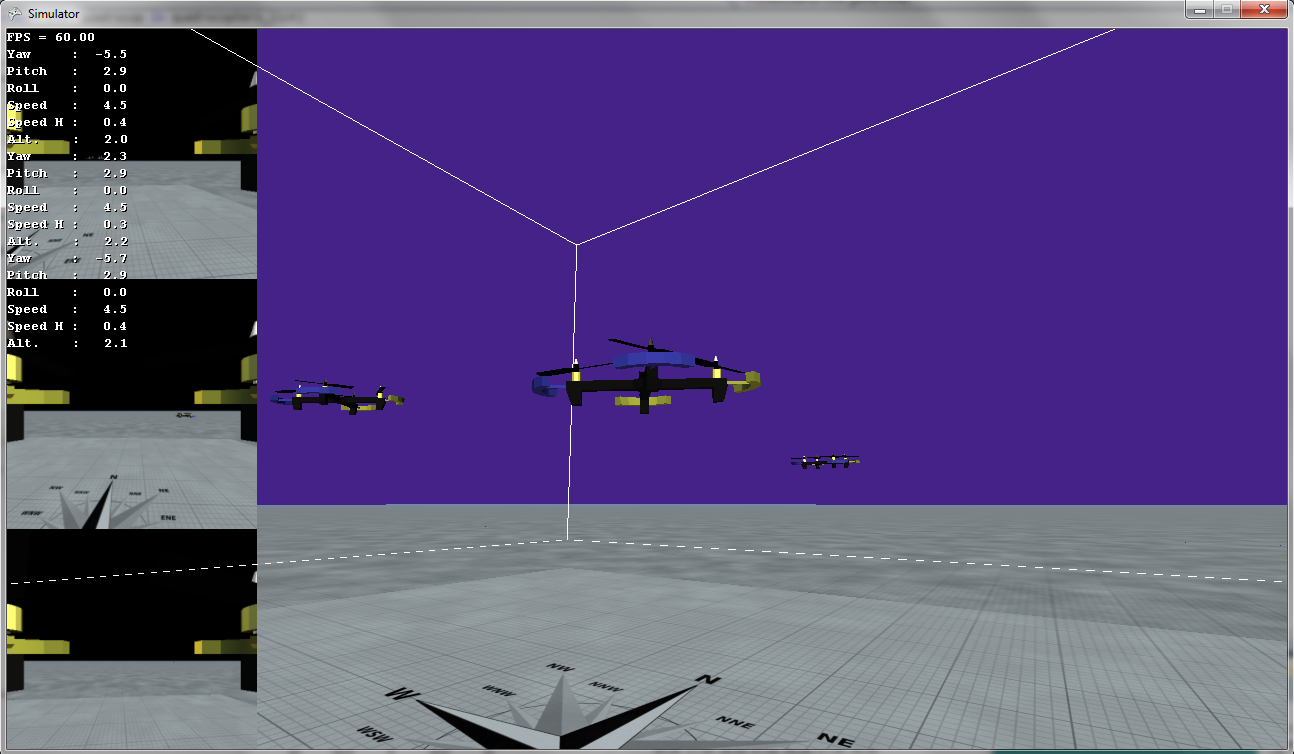
# Виртуальный полигон

На основе поставленных задач к виртуальному полигону можно предъявить следующие требования:

* Качественное воспроизведение аэродинамических эффектов взаимодействия групп БПЛА и окружения;
* Моделирование инерциальных, барометрических и магнитометрических датчиков;
* Синтез изображений формируемых камерами БПЛА;
* Варьирование параметров БПЛА с целью поиска оптимальной конфигурации:
  + Длина плеча;
  + Двигатели + ESC (по таблицам) ;
  + Пропеллеры и т.д. (по таблицам + опт. связь c пакетами CFD) ;
* Выбор окружения для моделирования;
* Сопряжение с реальным объектом БПЛА;
* Расчет в реальном масштабе времени.

Архитектура виртуального полигона представлена на рис. 3

Рис. 3



# Роторы

Расчет аэродинамических свойств винта является сложной численной задачей. Для упрощения используется приблизительный подход, основанный на знании его предельных характеристик и закону их изменения. Сила тяги винта возникает в результате действия аэродинамической силы на элемент лопасти винта при его вращении Разложив эту силу на две составляющие, параллельную оси вращения и параллельную плоскости вращения, получим подъмную силу и силу сопротивления вращению.

Суммируя силу тяги отдельных элементов лопасти винта и приложив ее к оси вращения, получим силу тяги винта *Y*.

(3.1)

где *ρ* — плотность воздуха, *Сy*— безразмерный коэффициент подъемной силы, который в общем случае зависит от формы аэродинамической поверхности и режима обтекания, определяется эксприментально, r — радиус винта

где N — количество лопастей, r — радиус винта, h — ширина лопасти (предполагается, что ширина лопасти неизменна), ω — частота вращения, а Γ — характеристический коэффициент для данного винта в данных условиях..

Пусть известно, что заданный воздушный винт при заданной частоте вращения ротора двигателя *ωmax* создает подъемную силу *Ymax*. Тогда коэффициент *Γ* определяется как:

(3.2)

Определим скорость воздуха проходящего через плоскость винта, предполагая, что воздух несжимаем. Согласно второму закону Ньютона:

(3.3)

(3.4)

где p — импульс воздушного потока, *u —* скорость воздушного потока, *m* — масса воздуха, разгоняемая до скорости *u* за время *t*.

Так как за время *t* через плоскость винта проходит масса воздуха равная:

(3.5)

то, скорость для воздушных масс можно выразить следующим образом:

(3.6) (3.7) (3.8)

Данное выражение можно использовать в дальнейшем для формирования потоков воздушных масс в замкнутых помещениях. Следует также отметить, что в случае попадания винта в поток воздуха тяга винта падает. Для аппроксимации падения силы тяги, будем считать, что тяга винта квадратично убывает по мере возрастания скорости потока:

(3.9)

где *k* — определяется экспериментально. В случае, если поток воздуха не параллелен оси винта, берется проекция скорости.

# 6-DOF тело

Для построения модели распределения сил и моментов, действующих на квадрокоптер, летательный аппарат рассматривается как твердое тело с 6-ю степенями свободы. Введем параметры, описывающие положение летательного аппарата в пространстве. Для этого необходимо выбрать локальную систему координат. За начало системы локальных координат примем центр тяжести квадрокоптера, а оси расположим так, чтобы ось  была направлена между первым и четвертым двигателем квадрокоптера, ось  ­­- влево, ось  - вверх (см. рис.).

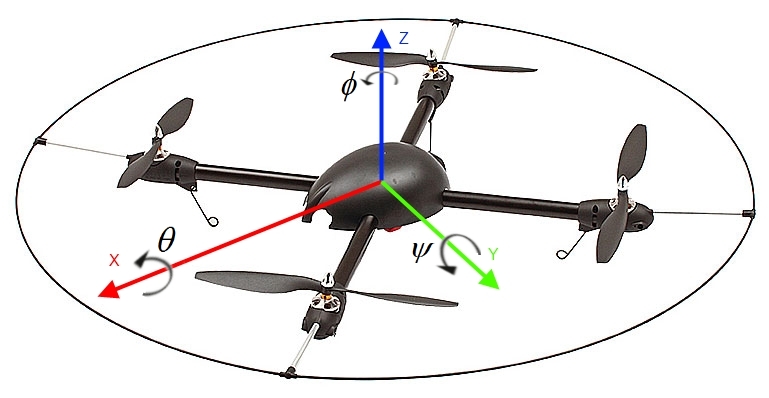


Рис.

Положение летательного аппарата в пространстве однозначно определяется кортежем из вектора положения центра тяжести и вектора вращения: , где , , где, в свою очередь , ,  - углы крена, тангажа и рысканья, соответственно, , ,  - глобальное положение центра тяжести летательного аппарата, а , ,  - орты глобальной системы координат.

Выпишем второй закон Ньютона:

 (4.1)

 (4.2)

где  – масса летательного аппарата,  – тензор инерции квадрокоптера.

# Модель потоков газа в замкнутых пространствах

Роторы летательного аппарата создают сильные потоки воздуха, которые могут влиять на полет других БПЛА или же приводить к тому, что когда ЛА подлетает достаточно близко к стене или другому крупному объекту возникает так называемый эффект подсасывания, вследствие которого ЛА начинает притягивать к этому объекту.

Для моделирования потоков газа в замкнутых пространствах обычно используют следующие методы:

* Методы решёточных уравнений Больцмана (англ. Lattice Boltzmann methods, LBM) — класс методов вычислительной гидродинамики для моделирования жидкостей. В отличие от многих других методов, метод LBM не решает уравнения Навье — Стокса, а моделирует поток ньютоновской жидкости с помощью дискретного кинетического уравнения Больцмана. Столкновения зачастую учитываются с помощью модели Батнагара — Гросса — Крука. Методы решёточных уравнений Больцмана удобны благодаря их концептуальной и вычислительной простоте, их использование ограничено малыми скоростями и тем, что методы решёточных уравнений Больцмана обладают неустойчивым поведением на границе подвижных тел.
* Гидродинамика сглаженных частиц (англ. Smoothed Particle Hydrodynamics, SPH) — вычислительный метод для симуляции жидкостей и газов. Используется во многих областях исследований, включая астрофизику, баллистику, вулканологию и океанографию. Метод гидродинамики сглаженных частиц является не-сеточным (англ. mesh-free) лагранжевым методом (то есть координаты движутся вместе с жидкостью), и разрешающая способность метода может быть легко отрегулирована относительно переменных, таких как плотность.
* Прямое численное моделирование (англ. DNS (Direct Numerical Simulation)) — метод основан на численном решении системы уравнений Навье-Стокса и позволяет моделировать в общем случае движение вязких сжимаемых газов с учётом химических реакций, притом как для ламинарных, так и турбулентных случаев. DNS предъявляет высокие требования к вычислительным ресурсам.

Для использования в виртуальном полигоне было решено реализовать прямое численное моделирование, в связи с его простотой реализации на GPU. Уравнения Навье-Стокса — система дифференциальных уравнений в частных производных, описывающая движение вязкой ньютоновской жидкости. Уравнения Навье-Стокса являются одними из важнейших в гидродинамике и применяются в математическом моделировании многих природных явлений и технических задач.

## Численное решение уравнений Навье-Стокса

Система уравнений Навье-Стокса состоит из двух уравнений:

* уравнения движения,
* уравнения неразрывности.

В векторном виде для несжимаемой жидкости они записываются следующим образом:

 (6.1)

 (6.2)

где  векторное поле скорости,  постоянная плотность газа,  скалярное поле давления,  вязкость газа и  представляет любую внешнюю силу, которая может воздействовать на газ, к примеру, винты летательного аппарата.

Для решения численным методом операторы градиента, дивергенции и Лапласа необходимо представить в конечно-разностной форме:

* Градиент 



* Дивергенция 



* Оператор Лапласа :



При этом если расчет ведется на сетке и все стороны ячеек этой сетки равны, то конечно-разностную форму оператора Лапласа можно упростить до:

 (6.3)

Уравнения Навье-Стокса можно решить аналитически только для нескольких простых физических конфигураций. Однако чтобы решить их можно использовать численные интеграционные методы. Для этого, как и любой другой алгоритм, необходимо разбить решение уравнений Навье-Стокса на несколько простых шагов. Для решения уравнений Навье-Стокса используется метод, основанный на технике “стабильных жидкостей” описанный в [Stam99]. Для начала необходимо привести уравнения к форме более подходящей для численного решения. Уравнения Навье-Стокса это три уравнения, которые мы можем решить для величин ,  и . Любой вектор  может быть разложен на несколько базовых составляющих, суммой которых будет . К примеру, векторы в декартовом пространстве обычно представляют, как три длины вдоль каждой оси: . Такой вектор так же может быть записан как , где ,  и  единичные базисные векторы, выровненные по осям декартовой сетки.

Так же как мы можем разложить вектор на сумму векторов, мы можем разложить векторное поле на сумму векторных полей. Пусть  будет областью пространства или как в нашем случае плоскостью, на котором определена наша жидкость. Пусть у этой области будут мягкие (т.е. дифференцируемые) границы  с направлением нормали . Согласно [Chorin and Marsden 1993] мы можем использовать теорему разложения Гельмгольца-Ходжа. Векторное поле  на  может быть однозначно разложено в форму

, (6.4)

где  имеет нулевую дивергенцию и параллельно ; так что  на .

Доказательство данной теоремы можно найти в [Chorin and Marsden 1993]. Эта теорема говорит о том, что любое векторное поле может быть разложено на сумму двух других векторных полей: бездивергентное векторное поле и градиент скалярного поля. Данная теория также говорит, что бездивергентное векторное поле обращается в ноль на границе.

Решение уравнений Навье-Стокса включает три вычисления для обновления скорости на каждом временном шаге: адвекция, диффузия и применение силы. Результатом будет новое поле скорости  с ненулевой дивергенцией. Но уравнение неразрывности требует, чтобы в конце каждого временного шага поле скорости было бездивергентным. Теорема разложения Гельмгольца-Ходжа говорит, что дивергенцию поля скорости можно скорректировать, если из него вычесть градиент результирующего поля давления:

. (6.5)

Теорема также ведет к методу расчета поля давления. Если применить оператор дивергенции к обеим сторонам уравнения (6.4), то можно получить:

. (6.6)

Но так как уравнение (6.2) обеспечивает соблюдение того, что , уравнение (6.6) упрощается до следующего вида:

, (6.7)

Что является уравнением Пуассона для расчета давления жидкости, так же это уравнение иногда называют уравнением давления-Пуассона. Это означает, что после того как будет рассчитано поле скорости с ненулевой дивергенцией , можно решить уравнение (6.7) для давления  и затем используя  и  рассчитать новое бездивергентное поле  используя уравнение (6.5).

Теперь необходимо рассчитать , чтобы это сделать мы можем использовать теорему разложения Гельмгольца-Ходжа, чтобы определить оператор проекции , который проецирует векторное поле  на его бездивергентную компоненту . Если мы применим  к уравнению (6.4), мы получим:

.

Но согласно определению , . Следовательно, . Благодаря этому можно упростить уравнения Навье-Стокса.

Вначале, применим оператор проекции к обеим сторонам уравнения (6.1):

.

Так как  бездивергентно, так же как и производная на левой стороне, то . И так же, , так что элемент давления выпадает. В итоге получается следующее уравнение:

. (6.8)

Данное уравнение символически инкапсулирует весь алгоритм симуляции потока жидкости. Сначала вычисляем то, что находиться в скобках с правой стороны. Слева на право, мы рассчитываем адвекцию, диффузию и приложенные силы. Выполнение этих трех шагов позволяет нам в результате получить дивергентное поле скоростей , к которому мы применяем оператор проекции, чтобы получить в итоге новое бездивергентное поле . Чтобы это сделать необходимо решить уравнение (6.7) для давления , а потом вычесть градиент  из , как показано в уравнении (6.5).

В типичной реализации, разные компоненты не вычисляются и н суммируются вместе, как в уравнении (6.8). Вместо этого, уравнение решается следующим образом: каждая компонента это шаг, на вход которого поступает поле и в результате на выходе получается новое поле. Мы можем определить оператор , который эквивалентен решению уравнения (6.8) за единичный временной шаг. Этот оператор определяется как последовательность операторов адвекции (), диффузии (), приложенных сил () и проекции ():

. (6.9)

Таким образом, алгоритм на каждом шаге симуляции можно выразить следующим образом , операторы выполняются справа налево; сначала адвекция, потом диффузия, приложенные силы и проекция. Здесь время опущено для большей ясности, но на практике, время шага должно быть использовано при расчетах каждого оператора.

### Адвекция

Адвекция это процесс, благодаря которому скорость газа транспортирует себя и другие величины в газе. Чтобы рассчитать адвекцию величины, мы должны обновлять эту величину в каждой ячейке сетки. Потому что мы вычисляем, как величина движется вдоль поля скоростей, можно представить, что каждая ячейка сетки это частица. Первым желанием рассчитать результат адвекции может быть попытка обновлять сетку также как и систему частиц. Просто двигать позицию  каждой частицы вперед по направлению скорости и дистанция, которую он пройдет за время :

.

Что является Эйлеровским методом; это простой метод для явной (или прямой) интеграции обыкновенных дифференциальных уравнений.

У данного подхода есть две проблемы. Первая заключается в том, что симуляция, которая использует прямые методы для адвекции, нестабильна для больших временных шагов, такая симуляция может “взорваться” если величина  будет больше чем размер ячейки сетки. Вторая проблема заключается в особенностях реализации данного метода на GPU. В случае если вся симуляция происходит на пиксельных шейдерах, то такие шейдеры не могут изменить позицию пикселя, в который они записывают результат. Метод прямой интеграции требует возможность “двигать” частицы, что невозможно осуществить на текущих графических процессорах.

Решение состоит в том, чтобы инвертировать проблему и использовать неявный метод, предложенный в [Stam99]. Вместо того чтобы осуществлять процесс адвекции путем расчета перемещения частицы за текущий временной шаг, мы можем отследить траекторию частицы из каждой ячейки сетки обратно во времени до текущей позиции и копировать значения данной позиции в стартовую ячейку сетки. Чтобы обновить значение  (это может быть скорость, плотность, температура или любая другая величина, переносимая в газе) можно использовать следующее уравнение:

. (6.10)

Данный метод можно легко реализовать на GPU, при этом как было показано в [Stam99] данный метод стабилен для произвольных временных шагов и скоростей. Рисунок 4 иллюстрирует вычисление адвекции для ячейки, которая выделена зеленым кругом. Отслеживание поля скорости обратно во времени приводит к точке x. Четыре значения ближайших к x ячеек (соединенных зеленым квадратом) билинейно интерполируются, и результат записывается в начальную ячейку сетки.

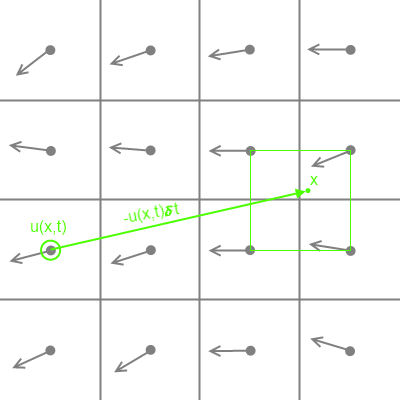


Рис. 4.

### Диффузия и вязкость

Вязкие жидкости имеют определенное сопротивление потоку, что приводит к диффузии (или рассеиванию) скорости. Частное дифференциальное уравнение для диффузии вследствие вязкости выглядит следующим образом:

. (6.11)

Наиболее явным подходом для решения данного уравнения является формулировка явной, дискретной формы для того чтобы разработать простой алгоритм:

.

В этом уравнении  дискретная форма оператора Лапласа (6.3). Как и явный Эйлеровский метод для расчета адвекции, этот метод нестабилен для больших значений  и . В [Stam99] для решения уравнения (6.11) предлагается неявная формулировка уравнения (6.11):

, (6.12)

где  это единичная матрица. Данная формулировка стабильна для произвольных значений временного шага и скорости. Данное уравнение является уравнением Пуассона для скорости. Это уравнение, как и уравнение (6.7) можно решить с помощью итерационного метода релаксации.

### Решение уравнений Пуассона

Необходимо решить два уравнения Пуассона: уравнение для расчета давления (6.7) и уравнение для расчета диффузии скорости вследствие вязкости (6.12). Для решения уравнений Пуассона мы воспользуемся итерационной техникой, которая начинает с аппроксимированного решения и улучшает его на каждой итерации.

Уравнение Пуассона это матричное уравнение следующего вида , где  это вектор значений, для которых мы решаем уравнения ( или  в нашем случае),  это вектор констант, а  это матрица. В нашем случае  неявно представляет собой оператор Лапласа. Итерационная техника, которую мы используем, начинается с первоначального «предположения» для решения  и с каждым шагом  мы получаем улучшенное решение , здесь индекс указывает на номер итерации. Простейшая итерационная техника называется методом итераций Якоби.

Более сложные методы, такие как сопряженные градиенты и многосеточные методы, сходятся быстрее, но мы используем итерации Якоби из-за его простоты и легкости его реализации.

Уравнения (6.7) и (6.12) выглядят по-разному, но оба могут быть дискретизированы, используя уравнение (6.3) и приведены к следующей форме:

, (6.13)

где  и  это константы. Значения , ,  и  разные для двух уравнений. В уравнении Пуассона для давления,  представляет ,  представляет , , и . Для уравнения диффузии за счет вязкости оба  и  представляют , , и .

Чтобы решить уравнения (6.7) и (6.12) мы просто запускаем несколько итераций, в которых мы решаем уравнение (6.13) для каждой ячейки сетки, используя результаты предыдущей итерации как входные данные. Из-за того что итерации Якоби медленно сходятся, необходимо выполнить много итераций. Итерации Якоби достаточно дешевы для выполнения на GPU, так что можно выполнить множество итераций за очень короткий промежуток времени.

### Начальные и граничные условия

Проблема любого дифференциального уравнения определенного на конечной области заключается в том что необходимо определить граничные условия для того чтобы оно было корректным. Граничные условия определяют, как мы будем вычислять значения на границах области, в которой происходит симуляция. Так же для расчета развития потока в течение времени, мы должны знать его начальное состояние. Для нашей симуляции потоков газа, мы полагаем, что газ изначально имел нулевую скорость и нулевое давление по всему пространству. Граничные условия требуют чуть более пристального внимания.

В течение каждого временного шага, мы решаем уравнения для двух величин: скорость и давление, и нам необходимы граничные условия для каждого из них. Благодаря тому, что наш газ рассчитывается на прямоугольной сетке, мы можем представить, что это газ в “коробке”, и он не может выйти за границы этой “коробки”. Для скорости мы используем условие прилипания, которое определяет, что скорость становится нулевой на границах. Корректное решение уравнения Пуассона для давления требует чистые граничные условия Неймана: , это означает, что на границах скорость изменения давления в направлении нормали равна нулю.

# Реализация ВП

## CUDA реализация Навье-Стокса

Моделирование потоков воздуха происходит в трехмерной декартовой сетке, размер которой может варьироваться.

Итак, согласно уравнению (6.9) для решения уравнений Навье-Стокса необходимо рассчитать адвекцию, диффузию, добавить внешние силы и выполнить проекцию.

Пример расчета адвекции представлен в листинге 7.1. При этом благодаря тому, что мы рассматриваем массивы компонент скоростей как трехмерные текстуры, мы получаем результат, который уже был билинейно интерполирован.

Листинг 7.1. Пример функции для расчета адвекции:

\_\_global\_\_ void advect ( float \*velx, float \*vely, float \*velz, float dt,

int nx, int ny, int nz,

float dx, float dy, float dz )

{

// cmpute voxel indices

int const x = blockIdx.x \* blockDim.x + threadIdx.x;

int const y = blockIdx.y \* blockDim.y + threadIdx.y;

int const z = blockIdx.z \* blockDim.z + threadIdx.z;

int const i = x + y\*nx + z\*nx\*ny;

float u = ((float)x + 0.5f) / (float)nx;

float v = ((float)y + 0.5f) / (float)ny;

float w = ((float)z + 0.5f) / (float)nz;

float vx = tex3D( tex\_velocity\_x, u, v, w );

float vy = tex3D( tex\_velocity\_y, u, v, w );

float vz = tex3D( tex\_velocity\_z, u, v, w );

velx[i] = tex3D( tex\_velocity\_x, u - vx \* dt, v - vy \* dt, w - vz \* dt );

vely[i] = tex3D( tex\_velocity\_y, u - vx \* dt, v - vy \* dt, w - vz \* dt );

velz[i] = tex3D( tex\_velocity\_z, u - vx \* dt, v - vy \* dt, w - vz \* dt );

}

Листинг 7.2. Пример функции для расчета границы:

\_\_global\_\_ void boundry ( float \*velx, float \*vely, float \*velz, float \*pressure, int \*bounds, int nx, int ny, int nz )

{

// cmpute voxel indices

int const x = blockIdx.x \* blockDim.x + threadIdx.x;

int const y = blockIdx.y \* blockDim.y + threadIdx.y;

int const z = blockIdx.z \* blockDim.z + threadIdx.z;

int const i = x + y\*nx + z\*nx\*ny;

int remap = bounds[i];

if (remap>=0) {

velx[i] = -velx [ remap ];

vely[i] = -vely [ remap ];

velz[i] = -velz [ remap ];

pressure[i] = pressure [ remap ];

}

}

Листинг 7.3. Пример функции для расчета уравнения Пуассона для вязкости:

int N = 10;

float visc = 0.073f;

float alpha = dx\*dx / visc / dt;

float beta = 6 + alpha;

for (int i=0; i<N; i++)

{

jacobi<<<grid\_size\_3d, block\_size\_3d>>>( d\_velocity\_x, d\_velocity\_y, d\_velocity\_z,

nx, ny, nz, alpha, beta );

boundry<<<grid\_size\_3d, block\_size\_3d>>>( d\_velocity\_x, d\_velocity\_y, d\_velocity\_z, d\_pressure, d\_boundry, nx, ny, nz );

}

\_\_global\_\_ void jacobi ( float \*velx, float \*vely, float \*velz,

int nx, int ny, int nz,

float alpha, float beta )

{

// cmpute voxel indices

int const x = blockIdx.x \* blockDim.x + threadIdx.x;

int const y = blockIdx.y \* blockDim.y + threadIdx.y;

int const z = blockIdx.z \* blockDim.z + threadIdx.z;

int const i = x + y\*nx + z\*nx\*ny;

float du = 1.0 / (float)nx;

float dv = 1.0 / (float)ny;

float dw = 1.0 / (float)nz;

float u = ((float)x + 0.5f) / (float)nx;

float v = ((float)y + 0.5f) / (float)ny;

float w = ((float)z + 0.5f) / (float)nz;

velx[i] = tex3D( tex\_velocity\_x, u+du, v, w )

+ tex3D( tex\_velocity\_x, u-du, v, w )

+ tex3D( tex\_velocity\_x, u, v+dv, w )

+ tex3D( tex\_velocity\_x, u, v-dv, w )

+ tex3D( tex\_velocity\_x, u, v, w+dw )

+ tex3D( tex\_velocity\_x, u, v, w-dw )

+ tex3D( tex\_velocity\_x, u, v, w ) \* alpha;

vely[i] = tex3D( tex\_velocity\_y, u+du, v, w )

+ tex3D( tex\_velocity\_y, u-du, v, w )

+ tex3D( tex\_velocity\_y, u, v+dv, w )

+ tex3D( tex\_velocity\_y, u, v-dv, w )

+ tex3D( tex\_velocity\_y, u, v, w+dw )

+ tex3D( tex\_velocity\_y, u, v, w-dw )

+ tex3D( tex\_velocity\_y, u, v, w ) \* alpha;

velz[i] = tex3D( tex\_velocity\_z, u+du, v, w )

+ tex3D( tex\_velocity\_z, u-du, v, w )

+ tex3D( tex\_velocity\_z, u, v+dv, w )

+ tex3D( tex\_velocity\_z, u, v-dv, w )

+ tex3D( tex\_velocity\_z, u, v, w+dw )

+ tex3D( tex\_velocity\_z, u, v, w-dw )

+ tex3D( tex\_velocity\_z, u, v, w ) \* alpha;

velx[i] /= beta;

vely[i] /= beta;

velz[i] /= beta;

}

Листинг 7.4. Пример функции, которая добавляет внешние силы в систему:

\_\_global\_\_ void rotor ( float \*velx, float \*vely, float \*velz, int nx, int ny, int nz, float sx,

float sy, float sz, int px, int py, int pz )

{

// cmpute voxel indices

int const x = blockIdx.x \* blockDim.x + threadIdx.x;

int const y = blockIdx.y \* blockDim.y + threadIdx.y;

int const z = blockIdx.z \* blockDim.z + threadIdx.z;

int const i = x + y\*nx + z\*nx\*ny;

float3 v = make\_float3(0,0,0);

if ( x >= px-1 && x <= px+1 )

if ( y >= py-1 && y <= py+1 )

if ( z >= pz-1 && z <= pz+1 )

v = make\_float3(sx,sy,sz);

velx[i] += v.x;

vely[i] += v.y;

velz[i] += v.z;

}

Листинг 7.5. Пример функции для расчета уравнения Пуассона для давления:

divergence<<<grid\_size\_3d, block\_size\_3d>>> ( d\_divergence, nx, ny, nz, dx, dy, dz );

alpha = -dx\*dx;

beta = 6;

for (int i=0; i<N; i++)

{

jacobi2<<<grid\_size\_3d, block\_size\_3d>>>( d\_pressure, nx, ny, nz, alpha, beta );

boundry<<<grid\_size\_3d, block\_size\_3d>>>( d\_velocity\_x, d\_velocity\_y, d\_velocity\_z, d\_pressure, d\_boundry, nx, ny, nz );

}

\_\_global\_\_ void jacobi2 ( float \*pressure, int nx, int ny, int nz, float alpha, float beta )

{

// cmpute voxel indices

int const x = blockIdx.x \* blockDim.x + threadIdx.x;

int const y = blockIdx.y \* blockDim.y + threadIdx.y;

int const z = blockIdx.z \* blockDim.z + threadIdx.z;

int const i = x + y\*nx + z\*nx\*ny;

float du = 1.0 / (float)nx;

float dv = 1.0 / (float)ny;

float dw = 1.0 / (float)nz;

float u = ((float)x + 0.5f) / (float)nx;

float v = ((float)y + 0.5f) / (float)ny;

float w = ((float)z + 0.5f) / (float)nz;

pressure[i] = tex3D( tex\_pressure, u+du, v, w )

+ tex3D( tex\_pressure, u-du, v, w )

+ tex3D( tex\_pressure, u, v+dv, w )

+ tex3D( tex\_pressure, u, v-dv, w )

+ tex3D( tex\_pressure, u, v, w+dw )

+ tex3D( tex\_pressure, u, v, w-dw )

+ tex3D( tex\_divergence, u, v, w ) \* alpha;

pressure[i] /= beta;

}

Листинг 7.6. Пример функции для расчета дивергенции:

\_\_global\_\_ void divergence ( float \*div, int nx, int ny, int nz,

float dx, float dy, float dz )

{

// cmpute voxel indices

int const x = blockIdx.x \* blockDim.x + threadIdx.x;

int const y = blockIdx.y \* blockDim.y + threadIdx.y;

int const z = blockIdx.z \* blockDim.z + threadIdx.z;

int const i = x + y\*nx + z\*nx\*ny;

float du = 1.0 / (float)nx;

float dv = 1.0 / (float)ny;

float dw = 1.0 / (float)nz;

float u = ((float)x + 0.5f) / (float)nx;

float v = ((float)y + 0.5f) / (float)ny;

float w = ((float)z + 0.5f) / (float)nz;

div[i]= (tex3D(tex\_velocity\_x,u+du,v,w)-tex3D(tex\_velocity\_x,u-du,v,w))/2.0f/dx

+ (tex3D(tex\_velocity\_y,u,v+dv,w)-tex3D(tex\_velocity\_y,u,v-dv,w))/2.0f/dy

+ (tex3D(tex\_velocity\_z,u,v,w+dw)-tex3D(tex\_velocity\_z,u,v,w-dw))/2.0f/dz;

}

Листинг 7.7. Пример функции для расчета градиента:

\_\_global\_\_ void gradient ( float \*velx, float \*vely, float \*velz, int nx, int ny, int nz, float dx, float dy, float dz )

{

int const x = blockIdx.x \* blockDim.x + threadIdx.x;

int const y = blockIdx.y \* blockDim.y + threadIdx.y;

int const z = blockIdx.z \* blockDim.z + threadIdx.z;

int const i = x + y\*nx + z\*nx\*ny;

float du = 1.0 / (float)nx;

float dv = 1.0 / (float)ny;

float dw = 1.0 / (float)nz;

float u = ((float)x + 0.5f) / (float)nx;

float v = ((float)y + 0.5f) / (float)ny;

float w = ((float)z + 0.5f) / (float)nz;

float grad\_x = ( tex3D( tex\_pressure, u+du, v, w ) - tex3D( tex\_pressure, u-du, v, w ) ) / 2.0f / dx;

float grad\_y = ( tex3D( tex\_pressure, u, v+dv, w ) - tex3D( tex\_pressure, u, v-dv, w ) ) / 2.0f / dy;

float grad\_z = ( tex3D( tex\_pressure, u, v, w+dw ) - tex3D( tex\_pressure, u, v, w-dw ) ) / 2.0f / dz;

velx[i] -= grad\_x;

vely[i] -= grad\_y;

velz[i] -= grad\_z;

}

Результат представлен на рис.

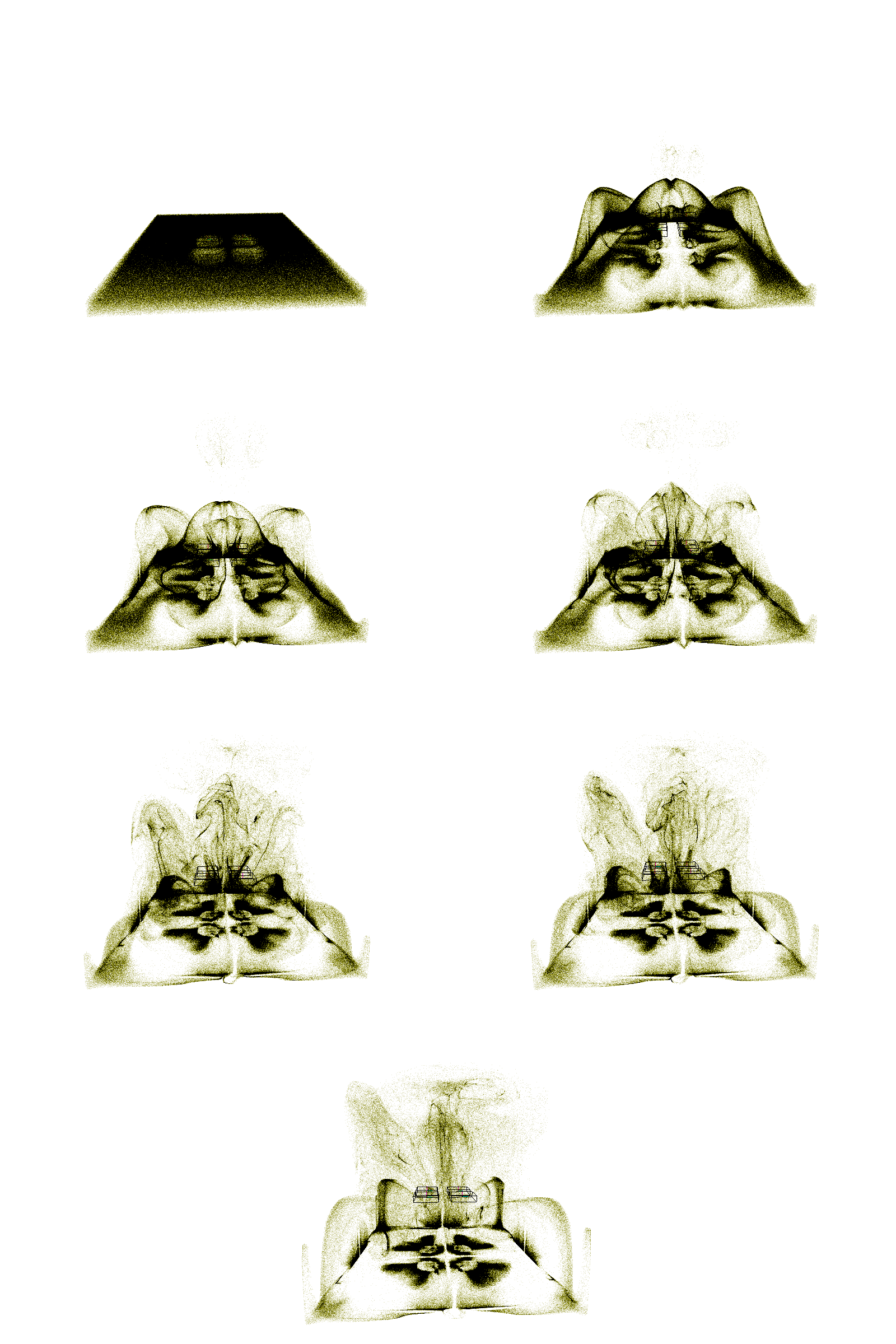


Рис. 5.

## Бортовое оборудование

### Гироскоп

В качестве объекта для моделирования был использован реальный трех осевой гироскоп Grove - 3-axis Gyro основанный на чипе ITG3200 (см. рис.)

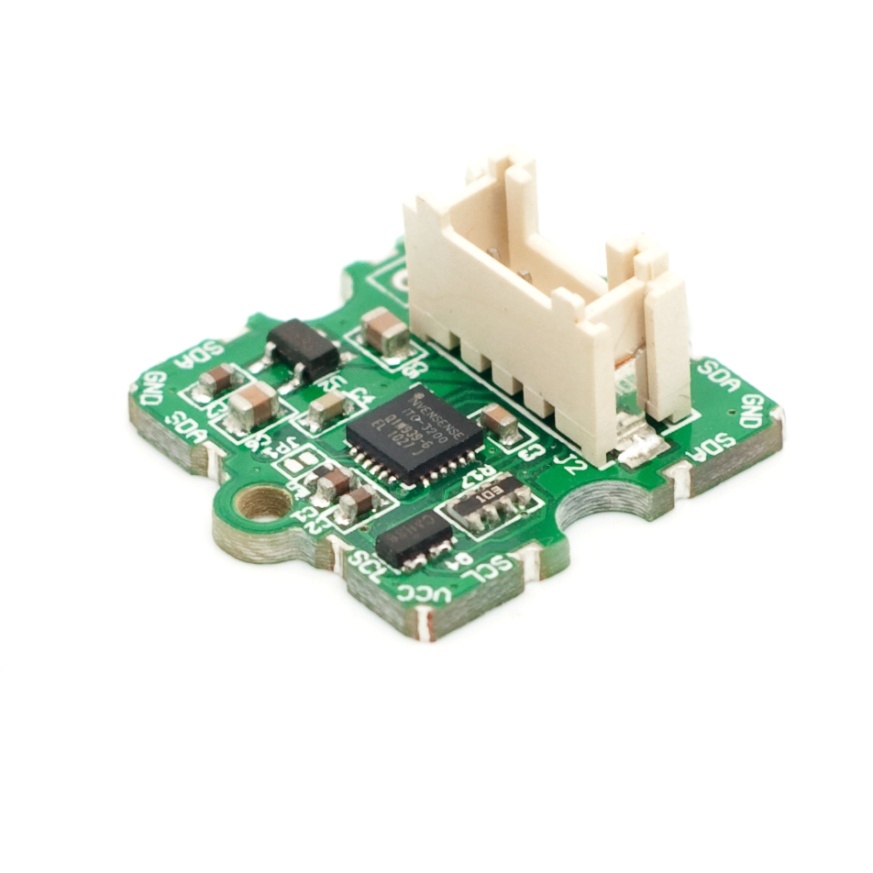


Рис. 6. Трех осевой гироскоп Grove - 3-axis Gyro

Данный модуль предоставляет высокоскоростное подключение по шине I2C на частоте до 400 кГц. Для дискретизации данных поступающих с гироскопа используются три аналого-цифровых преобразователя с разрядностью шестнадцать бит. Диапазон измерений составляет  градусов в секунду. Частоту дискретизации можно регулировать путем изменения значений двух регистров. Первый такой регистр располагается по адресу 21 и называется делитель частоты дискретизации, с его помощью можно установить частоту дискретизации, которую можно определить по следующей формуле:

,

где  может быть равно либо 1 кГц, либо 8 кГц, значение задается в самом регистре. Второй регистр располагается по адресу 22 и с его помощью можно задать  и задать конфигурацию цифрового фильтра низких частот.

### Акселерометр

В качестве реального объекта для моделирования был использован трех осевой акселерометр Grove – I2C 3-axis Accelerometer на базе чипа MMA7660FC.



Рис. 7.

### Сонар

В качестве реального объекта для моделирования был использован сонар производства компании seeedstudio.



Рис. 8.

### Магнитометр

В качестве реального объекта для моделирования был использован Grove - 3-axis Compass.

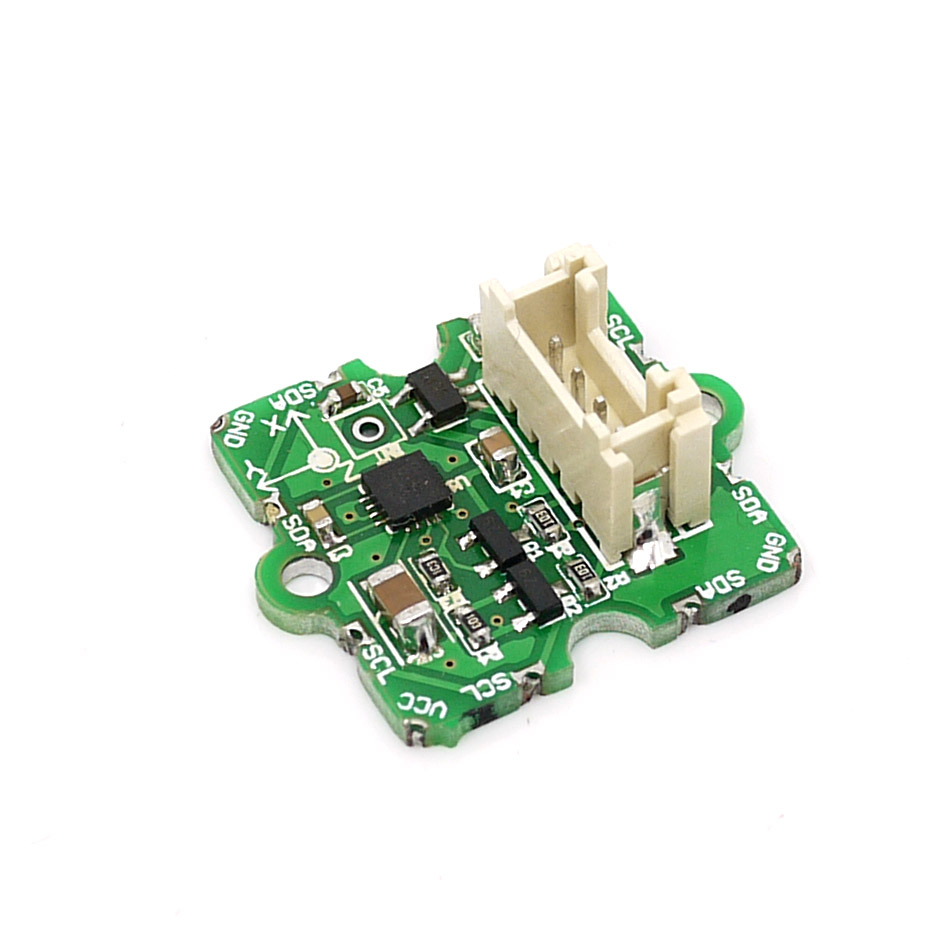


Рис. 9.

### Барометр

В качестве реального объекта для моделирования был использован Grove - Barometer Sensor.

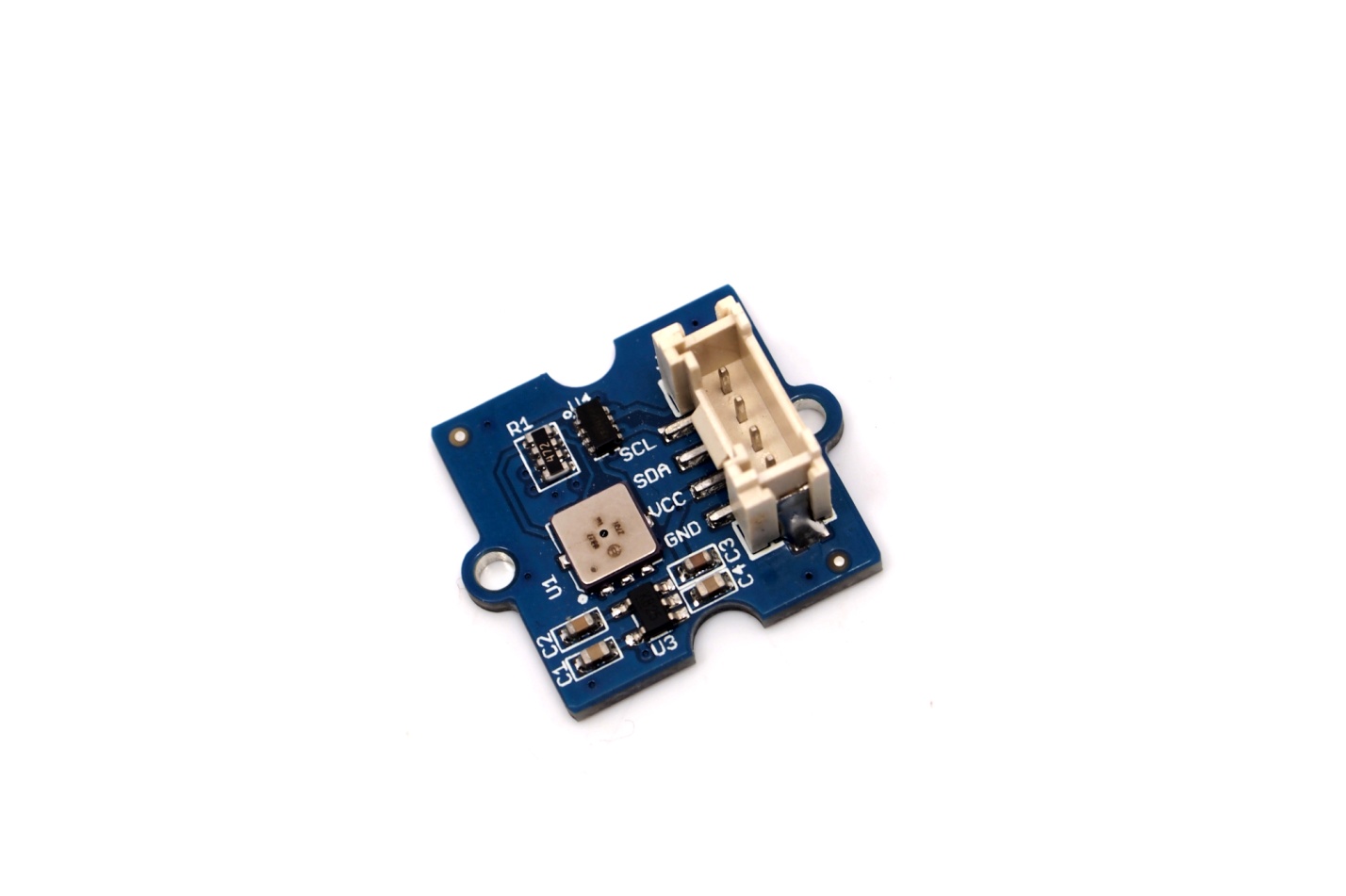


Рис. 10.

### Система глобального позиционирования

В качестве реального объекта для моделирования был использован модуль Grove – GPS.



Рис. 11.

## Камеры

На данный момент камера для БПЛА не выбрана, однако сформулированы требования к имитационной модели бортовой камеры. Модель должна воспроизводить следующие эффекты:

* Помехи и шум
* Блики в линзах
* HDR
* Задержки видеопотока
* Разные фокусные расстояния и различные оптические искажения

Данные эффекты реализуются путем пост-обработки синтезируемого изображения.

## Моделирование бортового оборудования

Для отработки алгоритмов управления БПЛА необходимо реалистичное моделирование бортового оборудования. С учетом их разрешающей способности, частоты дискретизации и уровня шума.

Интерфейс класса объединяющего модели всего представленного выше бортового оборудования представлен в листинге.

Листинг 8.1.

public class OnBoardDevices

{

public OnBoardDevices ( Quadrocopter qc );

public void Update ( GameTime gameTime );

public void UpdateCamera ()

public int CameraUpdateRate { get; set; }

public Rectangle CameraImageSize { get; set; }

public float CameNoiseLevel { get; set; }

public int GyroUpdateRate { get; set; }

public int GyroDataBits { get; set; }

public int GyroNoiseLevel { get; set; }

public int AccelUpdateRate { get; set; }

public int AccelDataBits { get; set; }

public int AccelNoiseLevel { get; set; }

public int CompassUpdateRate { get; set; }

public int CompassDataBits { get; set; }

public int CompassNoiseLevel { get; set; }

public int BarometerUpdateRate { get; set; }

public int BarometerDataBits { get; set; }

public int BarometerNoiseLevel { get; set; }

public int SonarUpdateRate { get; set; }

public int SonarDataBits { get; set; }

public int SonarNoiseLevel { get; set; }

public Vector3 SonarDirection { get; set; }

public int GpsUpdateRate { get; set; }

public int GpsDataBits { get; set; }

public int GpsNoiseLevel { get; set; }

public Vector3 GetGyroData ();

public Vector3 GetAccelData ();

public Vector3 GetCompassData ();

public float GetBarometerData ();

public float GetSonarData ();

public Vector2 GetGpsData ();

public Texture2D GetCameraImage();

public void SetupITG3200 ();

public void SetupMMA7660FC ();

public void SetupSeeduinoSonar ();

public void SetupGroveBarometer ();

public void SetupGroveCompass ();

public void SetupGroveGPS ();

}

Данный класс предоставляет гибкие возможности по настройке параметров моделируемого бортового оборудования, а также реализованы функции для настройки под конкретные модели устройств.

## Средства анализа данных

Для вывода графиков в реальном времени используется Microsoft Chart.

### Система оптического захвата движений

Вследствие того что инерциальные сенсоры со временем накапливают ошибку появляется необходимость во внешней стационарной системе, которая бы не была подвержена подобной проблеме. Такая система может пригодиться в следующих случаях:

* Отладка бортовой системы управления
* Автоматическая посадка на базе
* Калибровка

Для таких целей очень хорошо подходит оптическая система захвата движения. Как правило, оптическая система захвата движения состоит из нескольких камер, это могут быть как обычные RGB камеры, так и инфракрасные. Эти камеры отслеживают положение в пространстве специальных маркеров, маркеры могут быть активными, т.е. излучать свет или же пассивными, т.е. отражать свет, в этом случае может понадобиться специальная система подсветки.

В программном комплексе QuadroX-DS реализована поддержка системы оптического захвата движений, производства фирмы Vicon. Данная система состоит из восьми камер типа Vicon Bonita 3 с инфракрасной подсветкой (см. рис.), набора светоотражающих маркеров и программы Vicon Tracker. Камеры имеют следующие характеристики:

* частота смены кадров – 240 кадров в секунду
* максимальное время выдержки – 0.5 мс.
* Разрешение – 640х480
* Интерфейс – Gigabit Ethernet, RJ45
* Задержка системы – 2 мс.
* Точность – 1 мм.
* Фокусное расстояние – 4-12 мм.
* Диапазон фокусировки – от 0.3 м. до бесконечности



Рис. 12. Камера Vicon Bonita 3.

Сами камеры работают только с программой Vicon Tracker. Для того чтобы отследить положение маркера необходимо чтобы его было видно как минимум на двух камерах.

Для того чтобы начать работать с системой её необходимо развернуть. Вначале расставить камеры так чтобы они в полной мере покрывали объем пространства, в котором необходимо отслеживать объекты, т.е. в любой точке пространства маркер, установленный на объекте, должен быть виден как минимум с двух камер. Однако даже при выполнении этого минимального условия могут возникнуть проблемы, т.к. сам отслеживаемый объект может закрывать маркеры. В этом случае может помочь увеличение количества камер (их количество практически никак не ограниченно и существуют системы, в которых общее количество камер превышает сорок штук) или же можно изменить количество и положения маркеров, располагаемых на объекте. После того как все камеры были расставлены систему захвата движения необходимо откалибровать. Калибровка системы происходит в несколько этапов в программе Vicon Tracker. На первом этапе необходимо откалибровать камеры путем изменения фокусного расстояния, фокальную плоскость и диафрагму, все эти параметры необходимо настроить так чтобы маркеры, помещенные в отслеживаемое пространство, имели четкую границу, не мерцали и желательно имели размер, не превышающий три на три пикселя. Такой небольшой размер (три на три пикселя) обуславливается тем, что два маркера могут перекрываться и тогда камера может распознать два таких маркера как один, поэтому чем меньше размер маркеров, тем меньше шансов на то, что они будут перекрываться. Однако это может привести к тому, что маркеры, которые находятся слишком далеко от камеры, начнут “мерцать”. На втором этапе необходимо убрать все маркеры из сцены, так чтобы ни одного маркера не было видно ни в одной из камер. На этом этапе происходит “маскирование” тех областей, которые либо сами являются источником инфракрасного излучения, например инфракрасная подсветка, установленная на камерах, либо его отражают, к примеру, пол на который падает луч света. Такие области будут “замаскированы” и те маркеры, которые будут попадать в эти области, никак учитываться не будут. На третьем этапе происходит определение положения камер относительно друг друга, а так же происходит их взаимная калибровка. Для того чтобы провести подобную калибровку системе нужен объект размеры которого ей известны. Для этих целей в комплекте с системой идет специальное приспособление под названием “wand” (см. рис.)



Рис. 13. Wand, приспособление, необходимое для калибровки системы оптического захвата движений.

Расстояния между маркерами строго фиксированы и известны системе. Благодаря этому, если две камеры или более видят эти маркеры, то они могут определить не только расстояние до них, но и расстояние до других камер. Сам процесс калибровки происходит следующим образом, человек, который производит калибровку, “размахивает” данным приспособлением вначале в самом центре отслеживаемого пространства, а затем на его периферии. Система запоминает кадры с камер, на которых было видно данное приспособление и когда для каждой камеры наберется достаточное количество кадров (не меньше тысячи), система начнет расчет положения камер. Когда расчет будет закончен, можно переходить к последнему шагу. На последнем, четвертом, шаге нужно определить положение и вращение начала координат, для этого необходимо снова использовать “wand”, его позиция и вращение и будут задавать начало координат. После калибровки системы можно приступать к работе с ней. Развертывание системы и ее калибровка занимает около часа. Работать с системой можно только внутри помещений, потому что любая поверхность, которая отражает падающий на нее солнечный свет, будет хорошо видна в камере, тем самым мешая поиску маркеров.

Для начала работы с реальным объектом необходимо установить на него несколько маркеров. Если необходимо отслеживать несколько реальных объектов, то маркеры необходимо установить уникально для каждого объекта, в таком случае на основе расстояний между маркерами система сможет распознать какой именно это объект. При этом если один или несколько маркеров были закрыты и ни одна из камер их не видит, то их позиция будет восстановлена, если видно хотя бы три маркера.

Расположение маркеров на реальном объекте и их отображение в программе Vicon Tracker представлены на рисунках и соответственно.



Рис.

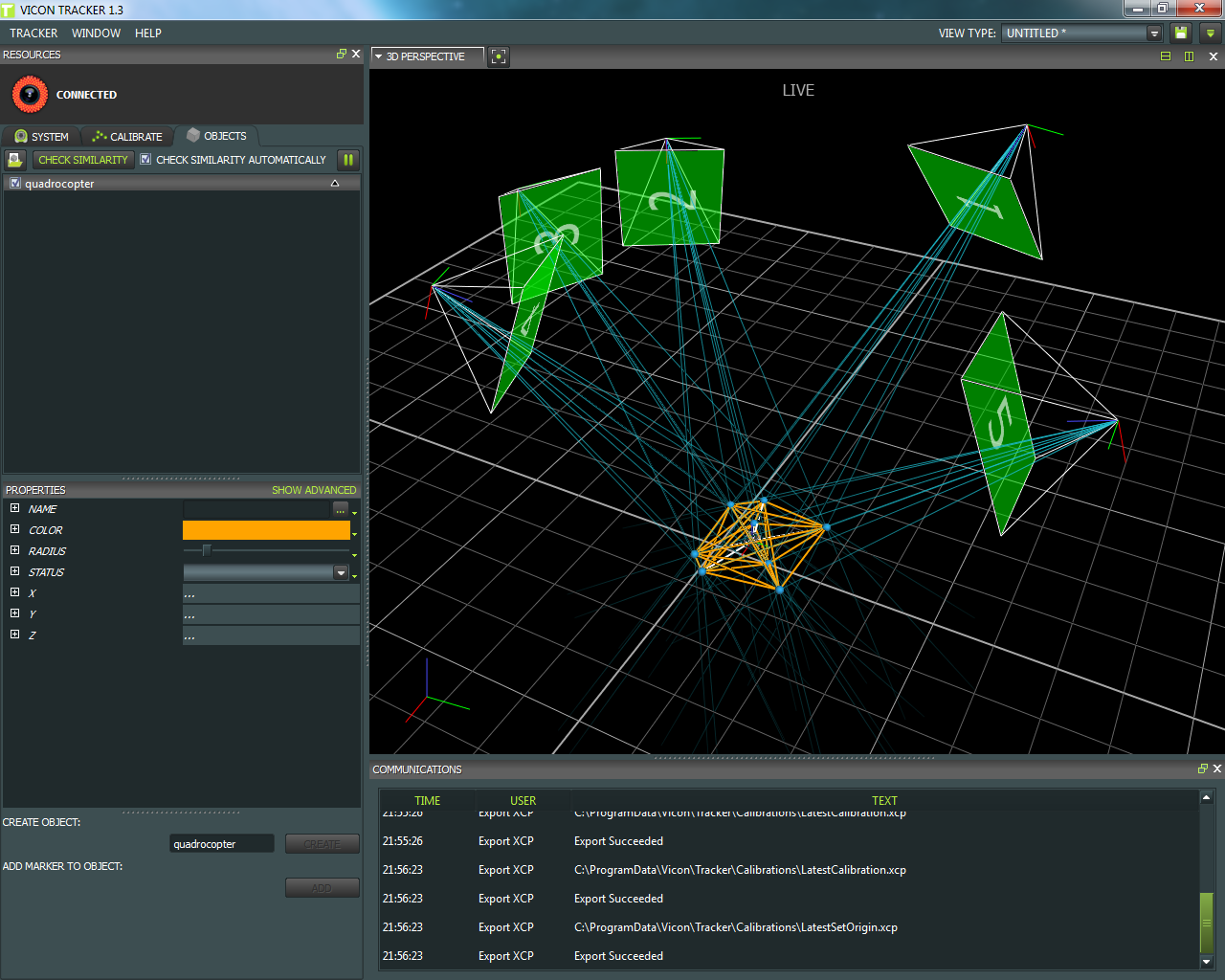


Рис.

Сама программа Vicon Tracker работает как сервер, т.е. в любой момент времени машина подключенная к сети в которой работает Vicon Tracker может запросить по имени объекта его положение и вращение. Связь с Vicon Tracker осуществляется с помощью SDK, который поставляет компания Vicon.

# Система управления

Система управления обеспечивает возможность как ручного управления квадрокоптером, так и автоматического. Причем управлять можно не только виртуальным объектом, но и реальным.

Система управления обеспечивает двунаправленное взаимодействие

# ЗАКЛЮЧЕНИЕ

* Разработана архитектура ВП для изучения динамики 4-х роторных БПЛА в реальном масштабе времени
* Спроектирован и частично разработан программно-аппаратный комплекс ВП QuadroX-DS
* Реализован механизм мониторинга динамики БПЛА посредством:
  + Телеметрии сенсоров и результатов работы бортового оборудования
  + Системы оптического захвата движения

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Chorin A., Marsden J. A. Mathematical Introduction to Fluid Mechanics. – NY: Springer, 1993. – 168 p.

2. Clancy, L.J., Aerodynamics. – NY.: Shroff Publishers, 2006. – 638 p.

3. Huang H., Hoffmann G. M., Aerodynamics and Control of Autonomous Quadrotor Helicopters in Aggressive Maneuvering. – NJ.: IEEE Press Piscataway. Proceedings of the 2009 IEEE international conference on Robotics and Automation. – pp. 2408-2413

4. James F. Roberts, Timothy S. Stirling, Quadrotor Using Minimal Sensing For Autonomous Indoor Flight. – T.: EMAV,2007. – 8 p.

5. Qimi Jiang, Daniel Mellinger, Christine Kappeyne, Vijay Kumar. Analysis and Synthesis of Multi-Rotor Aerial Vehicles. – W.: IDETC/CIE, 2011. – 10 p.

6. Stam, J. Stable Fluids. – NY.:ACM. SIGGRAPH 99 Conference Proceedings, 1999. – pp. 121-128.

7. Висленев Б. В., Кузьменко Д. В. Теория авиации. – М.: Гос. военное изд-во наркомата обороны Союза ССР, 1939. – 380 стр.