Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка программного обеспечения для генерации круговых волн на поверхности жидкости»

Студент ИУ7-53Б (Группа) _____ (Подпись, дата) Шимшир Э. О. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Филлипов М. В. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

Введение

Компьютерная графика — это совокупность методов и способов преобразования информации в графическое представление при помощи ЭВМ. Ее применение находит широкий спектр приложений, от создания удивительных визуальных эффектов в компьютерных играх до моделирования сложных трехмерных объектов. Одной из интересных задач в области компьютерной графики является визуализация и моделирование волн на поверхности жидкости, особенно круговых волн, вызванных падением капли.

Генерация круговых волн на поверхности жидкости при падении капли имеет большое практическое значение в различных областях, включая компьютерные игры, анимацию, визуальные эффекты и моделирование. Точное воссоздание реалистичного движения волн на поверхности жидкости является сложной задачей, требующей разработки эффективных алгоритмов и методов моделирования.

Целью данного курсового проекта является разработка программного обеспечения, позволяющего моделировать генерацию круговых волн на поверхности жидкости.

Постановка задачи

Чтобы достигнуть поставленой в курсовом проекте цели, требуется решить следующие задачи.

— произвести анализ существующих алгоритмов компьютерной графики;

— выбрать наиболее подходящие алгоритмы для достижения поставленой цели;

— выбрать средства реализации программного обеспечения;

— разработать программное обеспечение и реализовать выбранные алгоритмы и структуры данных;

— провести замеры временных характеристик разработанного программного обеспечения.

1 Аналитический раздел

В данном разделе проводится анализ существующих алгоритмов построения изображений и выбор подходящих алгоритмов для решения задачи.

1.1 Формализация объектов сцены

Сцена состоит из следующих объектов:

— поверхность жидкости — трехмерная модель, представляющая собой полигональную сетку, состоящую из связанных между собой плоских треугольников;

— раковина — трехмерная модель, которая содержит в себе поверхность жидкости;

— капля — трехмерная модель, возбуждающая волны на поверхности жидкости;

— источник света — вектор направления света;

— камера — характеризуется своим положением и направлением просмотра.

1.2 Анализ способов описания трехмерных моделей

В компьютерной графике существуют три основных типа моделей для описания трехмерных объектов: каркасная, поверхностная и твердотельная модели. Они предоставляют различные способы представления объектов и позволяют достичь правильного отображения их формы и размеров на сцене.

1.2.1 Каркасная модель

Каркасная модель —- в трехмерной графике описывает совокупность вершин и ребер, которая показывает форму многогранного объекта. Это моделирование самого низкого уровня и имеет ряд серьезных ограничений, большинство из которых возникает из-за недостатка информации о гранях, которые заключены между линиями, и невозможности выделить внутреннюю и внешнюю область изображения твердого объемного тела. Однако каркасная

модель требует меньше памяти и вполне пригодна для решения задач относящихся к простым. Основным недостатком каркасной модели является то, что модель не всегда однозначно передает информацию о форме объекта [1].

1.2.2 Поверхностная модель

Поверхностная модель — в трехмерной графике определяется в терминах точек, линий и поверхностей. При построении поверхностной модели предполагается, что технические объекты ограничены поверхностями, которые отделяют их от окружающей среды. Недостатком поверхностной модели является отсутствие информации о том, с какой стороны находится поверхности материала [1].

1.2.3 Твердотельная модель

Твердотельная модель — содержит информацию о том, каким образом материал расположен с той или иной стороны. В модели необходимо указать направление внутренней нормали.

1.2.4 Выбор способа описания модели

Для решения поставленной задачи наилучшим образом подойдет поверхностная модель.

Поверхностная модель задается полигональной сеткой. Полигональная сетка характеризуется совокупностью вершин, ребер и граней, определяющих форму объекта в трехмерном пространстве.

Изначально для представления поверхности жидкости используется сеточная модель, состоящая из массива точек (вершин сетки), которые разбиваются на треугольники. Для вычисления координат точек поверхности жидкости на каждом шаге времени необходимо использовать волновое уравнение. Это позволяет быстро определить новые координаты точек сетки и обновить их значения.

1.3 Анализ алгоритмов удаления невидимых поверхностей

Удаление невидимых линий и поверхностей – одна из самых сложных задач в графике. Алгоритмы удаления невидимых линий и поверхностей определяют, какие линии, поверхности или объемы видимы или невидимы для наблюдателя, находящегося в определенной точке пространства.

1.3.1 Алгоритм Робертса

Алгоритм Робертса работает в объектном пространстве только с выпуклыми телами. Если тело не является выпуклым, то его предварительно нужно разбить на выпуклые составляющие [2].

Этот алгоритм выполняется в 4 этапа:

— Подготовка исходных данных – составление матрицы тела для каждого тела сцены;

— Удаление ребер, экранируемых самим телом;

— Удаление ребер, экранируемых другими телами;

— Удаление линий пересечения тел, экранируемых самими телами и другими телами, связанными отношением протыкания.

Преимуществом алгоритма является высокая точность вычислений, но он позволяет работать только с выпуклыми телами и имеет ограничение вычислительной сложности, которая возрастает пропорционально квадрату числа объектов.

1.3.2 Алгоритм Варнока

Алгоритм Варнока работает в пространстве изображения и позволяет определить, какие грани или части граней объектов сцены видимы, а какие нет. Алгоритм прелагает разбиение области изображения на более мелкие окна, и для каждого подобного окна определяются связанные с ней многоугольники. Полигоны, видимость которых можно определить, изображаются на сцене.

Алгоритм работает рекурсивно, это является его главным недостатком.

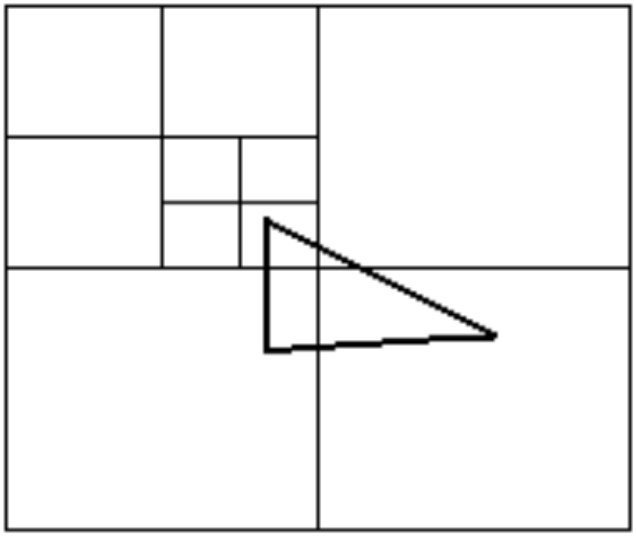

Рисунок 1 – Пример разбиения в алгоритме Варнока

1.3.3 Алгоритм, использующий Z-буфер

Алгоритм, основанный на использовании Z-буфера, является простым и широко используемым инструментом.

В данном алгоритме используется два буфера: буфер кадра и Z-буфер. Буфер кадра используется для заполнения атрибутов каждого пикселя в пространстве изображения. Z-буфер – отдельный буфер глубины, используемый для запоминания глубины каждого видимого пикселя в пространстве изображения [2].

В начале работы значения Z-буфера устанавливаются на минимальные, а в буфере кадра размещаются атрибуты фона. В процессе работы каждый новый пиксель сравнивается со значениями в Z-буфере. Если новый пиксель ближе к наблюдателю, чем предыдущий, он заносится в буфер кадра и в Z-буфер.

Основным преимуществом работы алгоритма является простота реализации. Трудоемкость алгоритма увеличивается линейно в зависимости от количества объектов на сцене. Требуется большой объем памяти.

1.3.4 Алгоритм обратной трассировки лучей

Алгоритм имеет такое название, потому что эффективнее с точки зрения вычислений отслеживать пути лучей в обратном направлении, то есть от наблюдателя к объекту. Наблюдатель видит объект посредством испускаемого источником света, который падает на этот объект и согласно законам оптики

некоторым путем доходит до глаза наблюдателя.

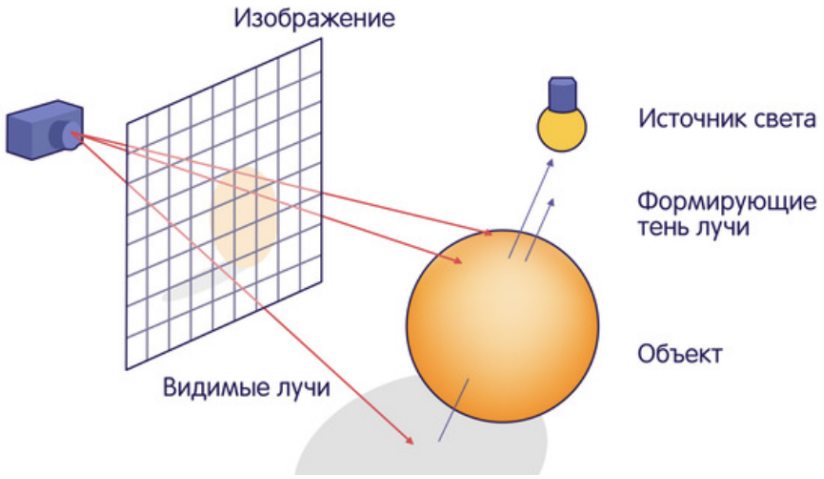

Рисунок 2 – Пример работы обратной трассировки лучей

Преимуществами являются возможность использования алгоритма в параллельных вычислительных системах и высокая реалистичность получаемого изображения, но необходимо большое количество вычислений.

1.3.5 Выбор алгоритма удаления невидимых поверхностей

Поверхность жидкости аппроксимируется треугольниками. Для визуализации поверхности жидкости за основу был взят алгоритм построчного сканирования, использующий Z-буфер.

1.4 Анализ алгоритмов закраски

Методы закраски используются для затенения полигонов модели в условиях некоторой сцены, имеющей источник освещения. Учитывая взаимное положение рассматриваемого полигона и источника света, можно найти уровень освещенности.

1.4.1 Простая закраска

Вся грань закрашивается одним уровнем интенсивности, который высчитывается по закону Ламберта. При данной закраске все плоскости будут закрашены однотонно.

Рисунок 3 – Пример простой закраски

1.4.2 Закраска по Гуро

Метод Гуро устраняет дискретность изменения интенсивности и создает иллюзию гладкой криволинейной поверхности. Он основан на интерполяции интенсивности.

Рисунок 4 – Пример закраски по Гуро

1.4.3 Закраска по Фонгу

Закраска Фонга по своей идее похожа на закраску Гуро, но ее отличие состоит в том, что в методе Гуро по всем точкам полигона интерполируется значения интенсивностей, а в методе Фонга - вектора нормалей, и с их помощью для каждой точки находится значение интенсивности.

Рисунок 5 – Пример закраски по Фонгу

1.4.4 Выбор алгоритма закраски

Алгоритм закраски Фонга дает наиболее реалистичное изображение, в частности зеркальных бликов. В курсовом проекте будет использоваться метод закраски Фонга и метод закраски Гуро.

1.5 Анализ моделей освещения

Выделяют две основные модели освещения: модель Ламберта и модель Фонга.

1.5.1 Модель освещения Ламберта

Модель Ламберта моделирует идеальное диффузное освещение. Освещенность в точке определяется только плотностью света в точке поверхности, а она линейно зависит от косинуса угла падения. При этом положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениям Модель Ламберта является одной из самых простых моделей освещения.

Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта) [3].

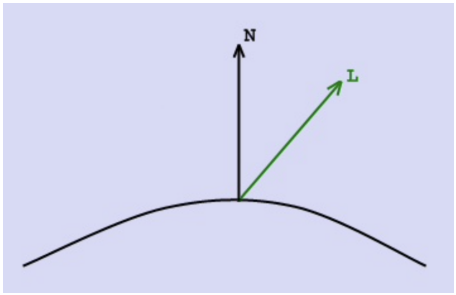

Рисунок 6 – Модель освещения Ламберта

Все векторы берутся единичными. Тогда косинус угла между ними совпадает со скалярным произведением. Формула расчета интенсивности имеет следующий вид:

$$I = I_0 \cdot k_d \cdot \cos(\vec{L}, \vec{N}) \cdot I_d = I_0 \cdot K_d \cdot (\vec{L}, \vec{N}) \cdot I_d \quad (1)$$

Где I — результирующая интенсивность света в точке; I_0 — интенсивность источника; k_d — коэффициент диффузного освещения; $\vec{L}$ — вектор от точки до источника; $\vec{N}$— вектор нормали в точке; I_d — мощность рассеянного освещения.

1.5.2 Модель освещения Фонга

Модель представляет собой комбинацию диффузной составляющей и зеркальной составляющей. Работает таким образом, что кроме равномер-

ного освещения на материале может также появиться блик. Отраженная составляющая в точке зависит от того, насколько близки направления от рассматриваемой точки на точку взгляда и отраженного луча. Местонахождение блика на объекте, освещенном по модели Фонга, определяется из закона равенства углов падения и отражения. Если наблюдатель находится вблизи углов отражения, яркость соответствующей точки повышается [3].

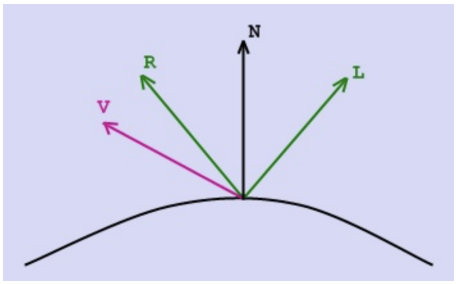

Рисунок 7 – Модель освещения Фонга

Для модели Фонга освещение в точке вычисляется по следующей формуле:

$$I = I_a + I_d + I_s \tag{2}$$

Где I — результирующая интенсивность света в точке; I_a — фоновая составляющая; I_d — рассеянная составляющая; I_s — зеркальная составляющая;

Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части. Таким образом отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча. Это можно выразить следующей формулой [3].

Формула для расчета интенсивности для модели Фонга имеет вид:

$$I = K_a \cdot I_a + I_0 \cdot K_d \cdot (\vec{L}, \vec{N}) \cdot I_d + I_0 \cdot K_s \cdot (\vec{R}, \vec{V})^{\alpha} \cdot I_s \tag{3}$$

Где I — результирующая интенсивность света в точке; K_a — коэффициент фонового освещения; I_a — интенсивность фонового освещения; I_0 — интенсивность источника; K_d — коэффициент диффузного освещения; $\vec{L}$ — вектор от точки до источника; $\vec{N}$ — вектор нормали в точке; I_d — интенсивность диффузного освещения; K_s — коэффициент зеркального освещения;

$\vec{R}$ — вектор отраженного луча; $\vec{V}$ — вектор от точки до наблюдателя; α — коэффициент блеска; I_s — интенсивность зеркального освещения.

1.5.3 Выбор модели освещения

Наилучшим решением для поставленной задачи будет остановться на модели Фонга при использовании закраски по Фонгу. Модель освещения Ламберта будет использоваться вместе с закраской по Гуро.

Вывод

В данном разделе проведен анализ существующих алгоритмов построения изображений и выбор подходящих алгоритмов для решения задачи.

2 Конструкторская часть

В данном разделе представлены требования к программному обеспечению, рассмотрены структуры данных, алгоритмы и математические уравнения, выбранные для построения сцены.

2.1 Требование к программному обеспечению

Программа должна обладать следующим функционалом:

— изменение положения камеры и направления её взгляда;

— изменения характеристик волны;

— изменения характеристик капли;

— изменение алгоритма закраски;

— возможность отрисовки полигонов модели.

2.2 Описание структур данных

Сцена представляет собой массив моделей, объект камеры и объект источника освещения.

1) Модель включает в себя следующие данные:
   — массив вершин фигуры;
   — массив полигонов фигуры;
   — массив векторов нормалей к вершинам;
   — цвет поверхности;
   — матрица аффинных преобразований.
2) Камера содержит:
   — положение в пространстве;
   — значения углов тангажа и рыскания;
   — направление взгляда и верха;
3) Источник освещения характеризуется вектором распространения лучей света или положением в пространстве.

2.3 Общий алгоритм построения изображения

Алгоритм генерации изображения представлен на рисунке 8. На вход подаются характеристики моделей, камеры, источника освещения. На выход выдается результат построения буфера кадра сцены.

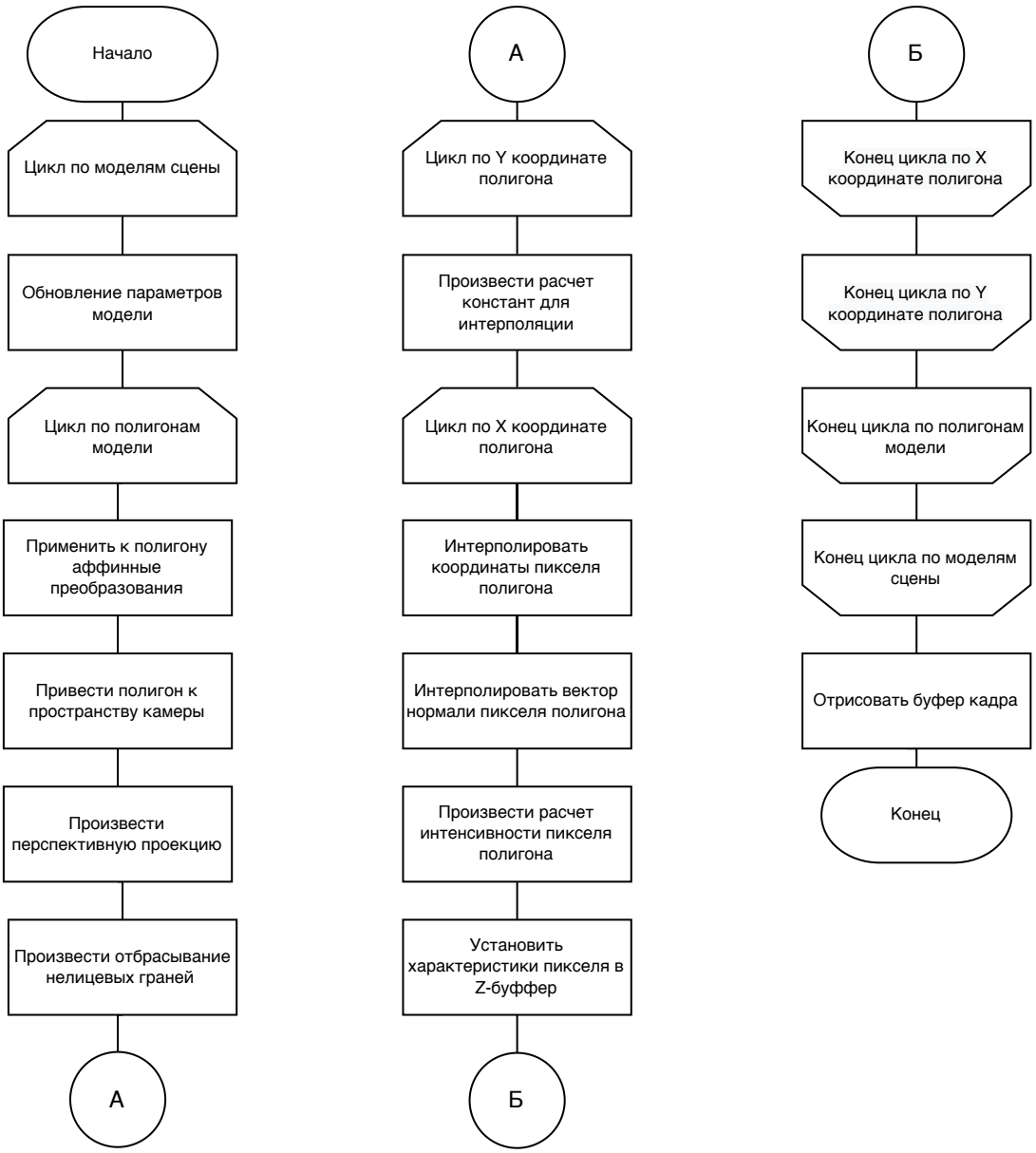

Рисунок 8 – Общая схема алгоритма синтеза изображения

2.4 Приведение к пространству камеры

Для перемещения по сцене используется камера, задаваемая точкой положения в пространстве и собственной системой координат, которая состоит из трех ортогональных векторов.

Обозначим:

— Point (P) — положение камеры;

— forward (F) — вектор взгляда;

— Up (U) — вектор вверх;

— Right (R) — вектор вправо.

Для перехода в пространство камеры выполняется в два этапа, указанные далее.

1) Перенос полигона в отрицательную стороны от камеры на расстояние Point с помощью матрицы переноса [4]:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Px & -Py & -Pz & 1 \end{pmatrix} \tag{1}$$

2) Преобразование полигона к системе координат камеры при помощи матрицы поворота [4]:

$$\begin{pmatrix} Rx & Ux & Fx & 0 \\ Ry & Uy & Fy & 0 \\ Rz & Uz & Fz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

Управление камеры производится с помощью изменения углом Эйлера. Обозначим α — угол поворота вокруг оси ОУ (тангаж) и β — угол поворота вокруг оси ОХ (рыскание). Тогда координаты вектора направления камеры

можно вычислить по следующим формулам:

$$F_x = cos(\alpha) \cdot cos(\beta) \tag{3}$$

$$F_y = sin(\alpha) \tag{4}$$

$$F_z = cos(\alpha) \cdot sin(\beta) \tag{5}$$

2.5 Невидимые грани

С помощью отбрасывания нелицевых граней моделей при построении изображения можно существенно сократить временные затраты, так как невидимые полигоны не будут растеризоваться. Для определения видимости грани требуется использовать формулу:

$$(\overrightarrow{N}, \overrightarrow{V}) = \begin{cases} \geq 0, \text{ если грань невидима} \\ < 0, \text{ если грань видима} \end{cases}, \tag{6}$$

где $\overrightarrow{N}$ — вектор внешней нормали к грани модели, $\overrightarrow{V}$ — вектор от камеры до любой точки грани.

2.6 Алгоритм Z-буфера

Для растеризации треугольного полигона необходимо найти глубину каждой его точки. Далее нужно произвести сравнение значения глубины точки со значением глубины из Z-буфера. Если глубина пикселя меньше, значит он лежит ближе к камере и должен быть растеризован, Z-буфер заносится значение глубины пиксела, и происходит вычисление интенсивности пикселя, значение которого заностится в буфер кадра. Полная схема алгоритма Z-буфера представлена на рисунке 9.

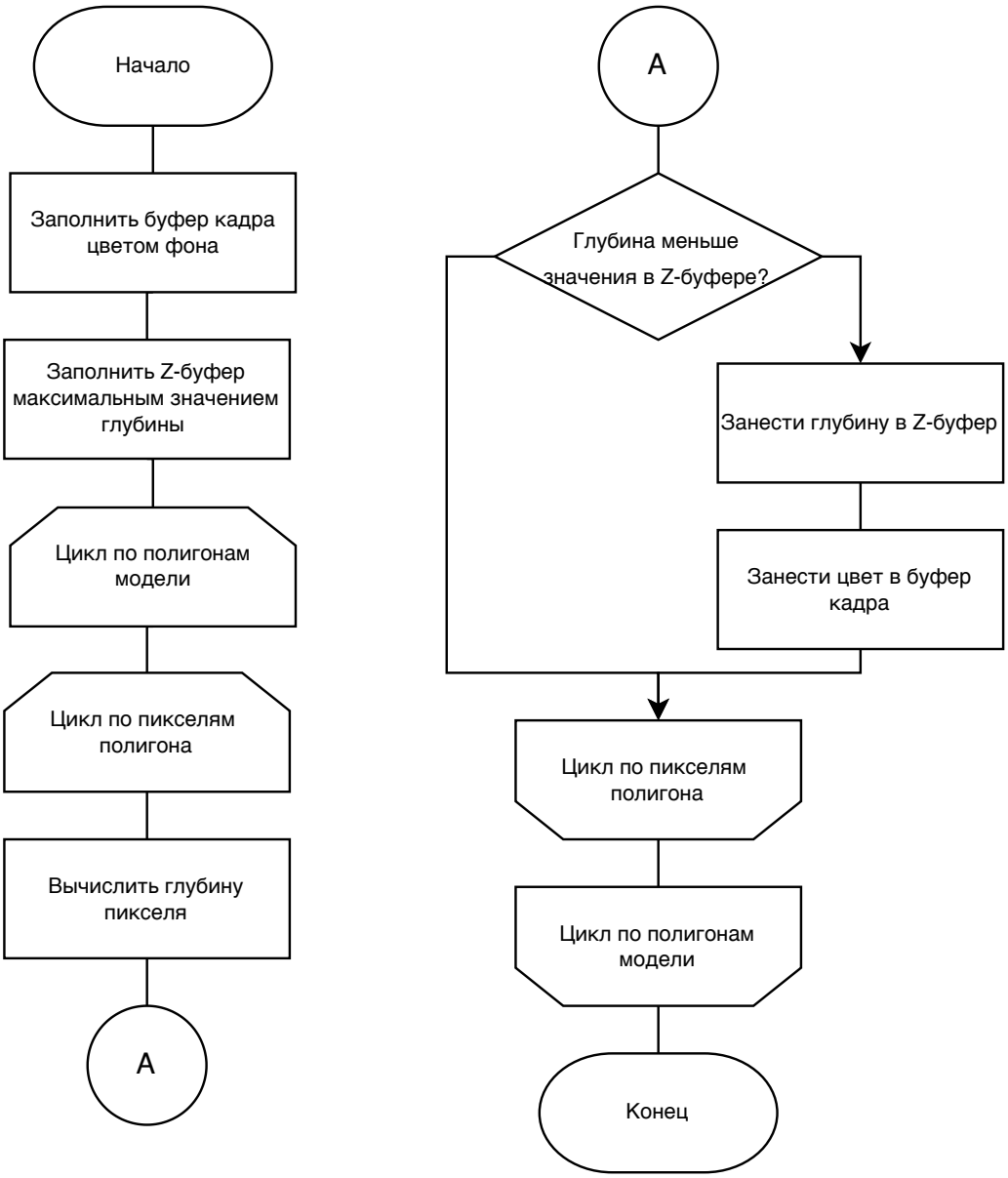

Рисунок 9 – Схема алгоритма Z-буфера

2.7 Описание уравнения бегущей волны

Бегущие волны моделируются свободными гармоническими колебаниями. Для перехода в нормализированное пространство, в котором значение функции лежит в пределах от 0 до 1, выполняется преобразование:

$$f(x) = \frac{\sin x + 1}{2} \tag{7}$$

С целью получения реалистичного изображения волн необходимо принять во внимание тот факт, что волны могут иметь большую крутизну и остроту пиков [5].

$$f(x) = \frac{\sin x + 1}{2} \tag{8}$$

Необходимо также учитывать направление волны. Ее моделирование происходит в двумерном поле высот, поэтому требуется определять движение волн в обоих направлениях. Вектор направления должен быть параллельным плоскости невозмущенной поверхности жидкости, т. е. иметь координату z, равную нулю. [5].

$$S = Dir(x, y) * Pos(x, y) \tag{9}$$

где $Dir(x, y)$ — вектор направления волны, $Pos(x, y)$ — вектор координат точки.

Учет частоты происходит в формуле в соответствии с известным соотношением определения частоты f:

$$f = \frac{2\pi}{\lambda} \tag{10}$$

где λ — длина волны.

Тогда получим:

$$S = Dir(x, y) * Pos(x, y)f \tag{11}$$

Для учета скорости волны необходимо определить фазовую постоянную в соответствии с выражением [5].

$$\psi = vf = \frac{2v\pi}{\lambda} \tag{12}$$

Окончательное выражение для вычисления координаты точки z на поверхности жидкости в зависимости от ее координат x, y и времени имеет следующий вид:

$$f(x, y, t) = A\left(\frac{\sin S + 1}{2}\right)^k \tag{13}$$

где $S = Dir(x, y) * Pos(x, y)f + t\psi$

2.8 Описание круговых волн

Учесть этот эффект можно с помощью переопределения вектора направления в каждой точке поверхности и изменения координаты точки смещением начала координат в точку центра круговой волны [5].

$$D(x, y) = \left(\frac{(x, y) - C}{|(x, y) - C|} \right), \tag{14}$$

где C — точка начала распространения круговой волны; X, Y — координаты рассматриваемой точки.

Измененные значения подставляются в формулу бегущей волны.

2.9 Описание вычисления нормалей

Используя уравнения поверхностей, определим нормали в любой ее точке. Уравнение нормали к поверхности в точке (x, y, z):

$$N(x, y, z) = NB(x, y, z) \otimes NT(x, y, z) \tag{15}$$

где $NB(x, y, z)$ и $NT(x, y, z)$ — частные производные по x и y;

$$NB(x, y, z) = \frac{\partial f(x, y, t)}{\partial x} \tag{16}$$

$$NT(x, y, z) = \frac{\partial f(x, y, t)}{\partial y} \tag{17}$$

где $f(x, y, t)$ — итоговое уравнение поверхности;

$$\frac{\partial f(x, y, t)}{\partial x} = 0.5 Dir_x f A \left(\frac{\sin S + 1}{2} \right)^{k-1} \cos S, \tag{18}$$

$$\frac{\partial f(x, y, t)}{\partial y} = 0.5 Dir_y f A \left(\frac{\sin S + 1}{2} \right)^{k-1} \cos S. \tag{19}$$

Задавая векторы нормалей проекциями на координатные оси, получим [5].

$$NB(x,y,z) = \left(\frac{\partial x}{\partial x}, \frac{\partial y}{\partial x}, \frac{\partial f(x,y,t)}{\partial x}\right) = \left(1, 0, \frac{\partial f(x,y,t)}{\partial x}\right), \tag{20}$$

$$NT(x,y,z) = \left(\frac{\partial x}{\partial y}, \frac{\partial y}{\partial y}, \frac{\partial f(x,y,t)}{\partial y}\right) = \left(0, 1, \frac{\partial f(x,y,t)}{\partial y}\right). \tag{21}$$

Вывод

В данном разделе были представлены требования к программному обеспечению, рассмотрены структуры данных, алгоритмы и математические уравнения, выбранные для построения сцены.

3 Технологическая часть

В данной части рассматривается выбор средств реализации, описывается структура классов программы и приводится интерфейс программного обеспечения.

3.1 Технологическая часть

Для написания данного курсового проекта был выбран язык Python [6].

Для разработки интерфейса и работы с пикселями изображения был выбран фреймворк Qt [7].

Используемые инструменты обладают полным функционалом для разработки, профилирования и отладки необходимой программы, а также создания графического пользовательского интерфейса.

3.2 Структура программы

Разработанная программа состоит из следующих классов. Базовые математические классы:

— Mat4x4 – класс матриц;

— Vector3D – класс векторов трехмерного пространства.

Классы для работы с моделями:

— Model – базовый класс моделей;

— Wave – класс модели поверхности жидкости.

Вспомогательные классы:

— Camera – класс камеры с возможностью перемещения по сцене;

— ZBuffer – класс z-буффера;

— RebderWidget – класс интерфейса;

— EngineBase – класс для отрисовки треугольников;

— Engine3D – класс для отрисовки моделей.

На рисунке 10 представлена схема разработанных классов.

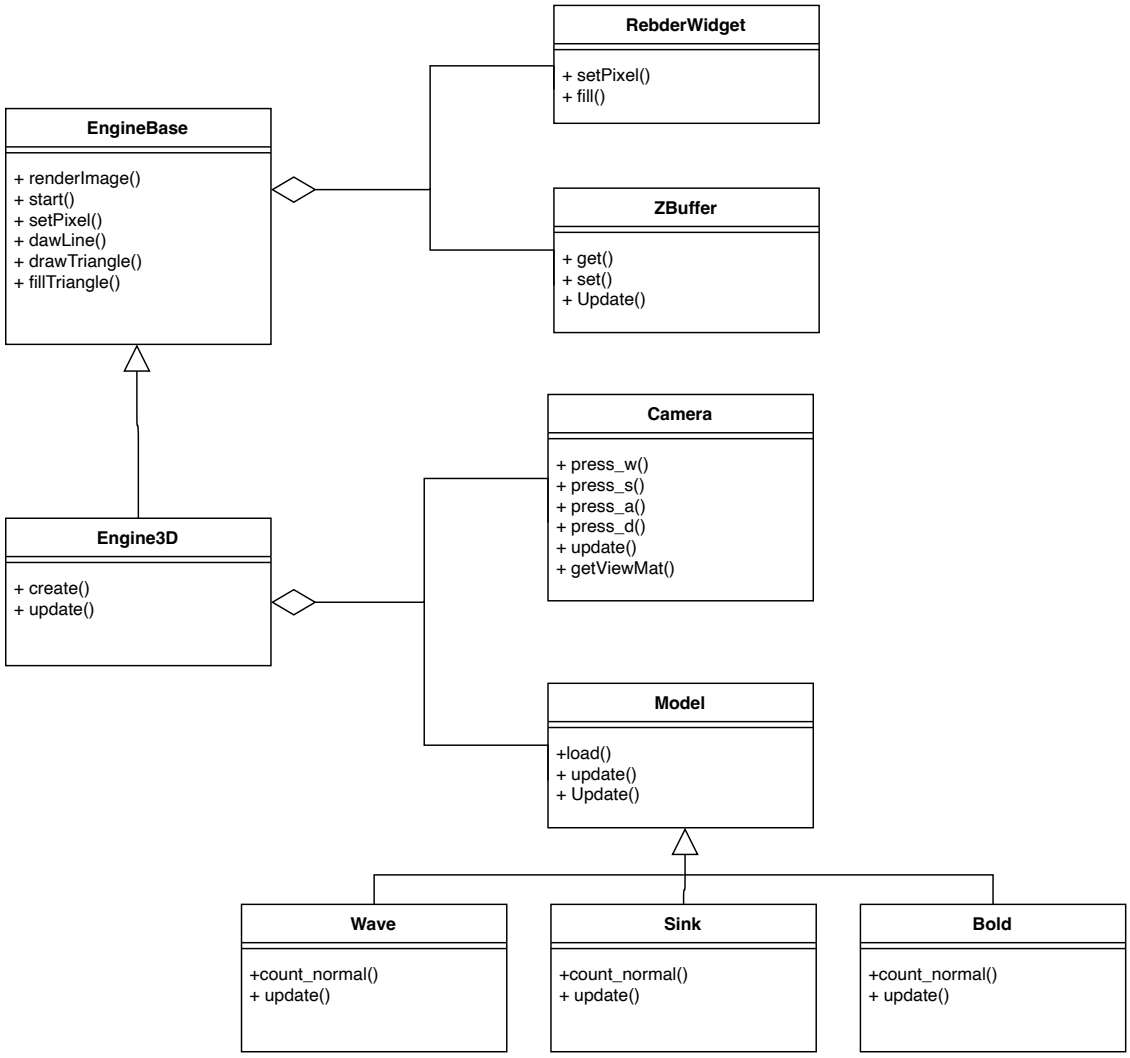

Рисунок 10 – Схема классов программы

3.3 Реализация алгоритмов

На листингах 1 – 4 приведены реализации алгоритмов

Листинг 1 – Алгоритм построения полигональной сетки

```
    def gen_poligon(self)
        start = -0.85
        stop = 0.85
        num = 11
        step = (stop - start) / (num - 1)
        x = [start + i * step for i in range(num)]

        z_def = 0

        for i in range((len(x) - 1)):
            for j in range((len(x) - 1)):
                mid = num // 2
                if i < mid and j < mid or i >= mid and j >= mid:
                    t1 = Triangle(Vec3D(x[i], x[j], z_def),
                                  Vec3D(x[i], x[j + 1], z_def),
                                  Vec3D(x[i + 1], x[j], z_def))
                    self.append(t1)

                    t2 = Triangle(Vec3D(x[i + 1], x[j], z_def),
                                  Vec3D(x[i], x[j + 1], z_def),
                                  Vec3D(x[i + 1], x[j + 1],
                                     z_def))
                    self.append(t2)
                else:
                    t1 = Triangle(Vec3D(x[i], x[j], z_def),
                                  Vec3D(x[i + 1], x[j + 1],
                                     z_def),
                                  Vec3D(x[i + 1], x[j], z_def),
                                  )
                    self.append(t1)

                    t2 = Triangle(Vec3D(x[i], x[j], z_def),
                                  Vec3D(x[i], x[j + 1], z_def),
                                  Vec3D(x[i + 1], x[j + 1],
                                     z_def),
                                  )
                    self.append(t2)
```

Листинг 2 – Алгоритм пересчета нормалей модели в каждый момент времени

```
    def update(self):
        self.time += 1 / self.fps

        for tri in self.tris:
            for p in tri.p:
                p.z = 0
                direction = Vec3D.sub(p, self.c)
                direction.normalize()
                s = Vec3D.dp(direction, p) * self.frequency -
                   self.time * self.fi
                p.z = self.amplitude * ((math.sin(s) + 1) * 0.5)
                   ** self.k

        self.count_normal()

    def count_normal(self):
        for tri in self.tris:
            for i in range(3):
                direction = Vec3D.sub(tri.p[i], self.c)
                direction.normalize()
                s = Vec3D.dp(direction, tri.p[i]) *
                   self.frequency - self.time * self.fi

                df_dx = 0.5 * self.k * direction.x *
                   self.frequency * self.amplitude * \
                        ((math.sin(s) + 1) * 0.5) ** (self.k -
                           1) * math.cos(s)

                df_dy = 0.5 * self.k * direction.y *
                   self.frequency * self.amplitude * \
                        ((math.sin(s) + 1) * 0.5) ** (self.k -
                           1) * math.cos(s)

                tri.n[i] = Vec3D(df_dx, -df_dy, -1).normalize()
```

Листинг 3 – Реализация Z-буффера

```
class ZBuffer:
    def __init__(self, width, height):
        self.width = width
        self.height = height

        self.array = [float('+inf')] * width * height

    def get(self, x, y):
        if 0 <= x < self.width and 0 <= y < self.height:
            return self.array[x + y * self.width]

        return float('+inf')

    def set(self, x, y, value):
        if 0 <= x < self.width and 0 <= y < self.height:
            self.array[x + y * self.width] = value
```

Листинг 4 – Модель освещения Фонга

```
light_dir = Vec3D.sub(light_pos, p)
light_dir.normalize()

eye_dir = Vec3D.sub(self.camera.point, p)
eye_dir.normalize()

reflect_dir = Vec3D.sub(Vec3D.mul_value(n, 2 * Vec3D.dp(n,
   light_dir)), light_dir).mul(-1)
reflect_dir.normalize()

kd = 0.75
ks = 0.25
spect_alpha = 8

ambient = 0.25
diffuse = Vec3D.dp(n, light_dir)
spect = Vec3D.dp(reflect_dir, eye_dir) ** spect_alpha
intensity = ks * spect + kd * diffuse + kd * ambient
```

Листинг 5 – Алгоритм закраски по Фонгу (интерполяция нормалей)

```
def fill_triangle_fong(self, rp0, rp1, rp2, rn0, rn1, rn2, red,
   green, blue):
    total_height = p2.y - p0.y
    for i in range(total_height):
        second_half = i > p1.y - p0.y or p1.y == p0.y
        segment_height = p2.y - p1.y if second_half else p1.y -
           p0.y
        alpha = i / total_height
        beta = (i - (p1.y - p0.y)) / segment_height if
           second_half else i / segment_height
        a = Vec3D.add(p0, Vec3D.sub(p2, p0).mul(alpha))
        u = Vec3D.dist(p0, a) / Vec3D.dist(p2, p0)
        n_a = Vec3D.add(Vec3D.mul_value(n0, 1 - u),
           Vec3D.mul_value(n2, u)).normalize()
        b = Vec3D.add(p1, Vec3D.sub(p2, p1).mul(beta)) if
           second_half else Vec3D.add(p0, Vec3D.sub(p1,
           p0).mul(beta))
        w = Vec3D.dist(p1, b) / Vec3D.dist(p2, p1) if
           second_half else Vec3D.dist(p0, b) / Vec3D.dist(p1,
           p0)
        n_b = Vec3D.add(Vec3D.mul_value(n1, 1 - w),
           Vec3D.mul_value(n2, w)).normalize() if second_half
           else Vec3D.add(Vec3D.mul_value(n0, 1 - w),
           Vec3D.mul_value(n1, w)).normalize()
        if a.x > b.x:
            a, b = b, a
            n_a, n_b = n_b, n_a
        for j in range(int(a.x), int(b.x) + 1):
            phi = 1 if b.x == a.x else (j - a.x) / (b.x - a.x)
            p = Vec3D.add(a, Vec3D.sub(b, a).mul(phi))
            p.int()
            n = n_a.copy()
            if not Vec3D.equal(a, b):
                t = Vec3D.dist(a, p) / Vec3D.dist(b, a)
                n = Vec3D.add(Vec3D.mul_value(n_a, 1 - t),
                   Vec3D.mul_value(n_b, t)).normalize()
```

Листинг 6 – Алгоритм закраски по Гуро (интерполяция интенсивностей)

```
def fill_triangle_guro(self, rp0, rp1, rp2, rn0, rn1, rn2, red,
   green, blue):
    total_height = p2.y - p0.y

    for i in range(total_height):
        second_half = i > p1.y - p0.y or p1.y == p0.y
        segment_height = p2.y - p1.y if second_half else p1.y -
           p0.y
        alpha = i / total_height
        beta = (i - (p1.y - p0.y)) / segment_height if
           second_half else i / segment_height

        a = Vec3D.add(p0, Vec3D.sub(p2, p0).mul(alpha))
        a.int()

        ia = i0 + (i2 - i0) * alpha

        b = Vec3D.add(p1, Vec3D.sub(p2, p1).mul(beta)) if
           second_half else \
            Vec3D.add(p0, Vec3D.sub(p1, p0).mul(beta))
        b.int()

        ib = i1 + (i2 - i1) * beta if second_half else \
            i0 + (i1 - i0) * beta

        if a.x > b.x:
            a, b = b, a
            ia, ib = ib, ia

        for j in range(int(a.x), int(b.x) + 1):
            phi = 1 if b.x == a.x else (j - a.x) / (b.x - a.x)

            p = Vec3D.add(a, Vec3D.sub(b, a).mul(phi))
            p.int()

            ip = ia + (ib - ia) * phi
```

3.4 Интерфейс программного обеспечения

При запуске программы в левой части определен виджет сцены, в правой части интерфейса определены разделы для настройки сцены (рисунок 11).

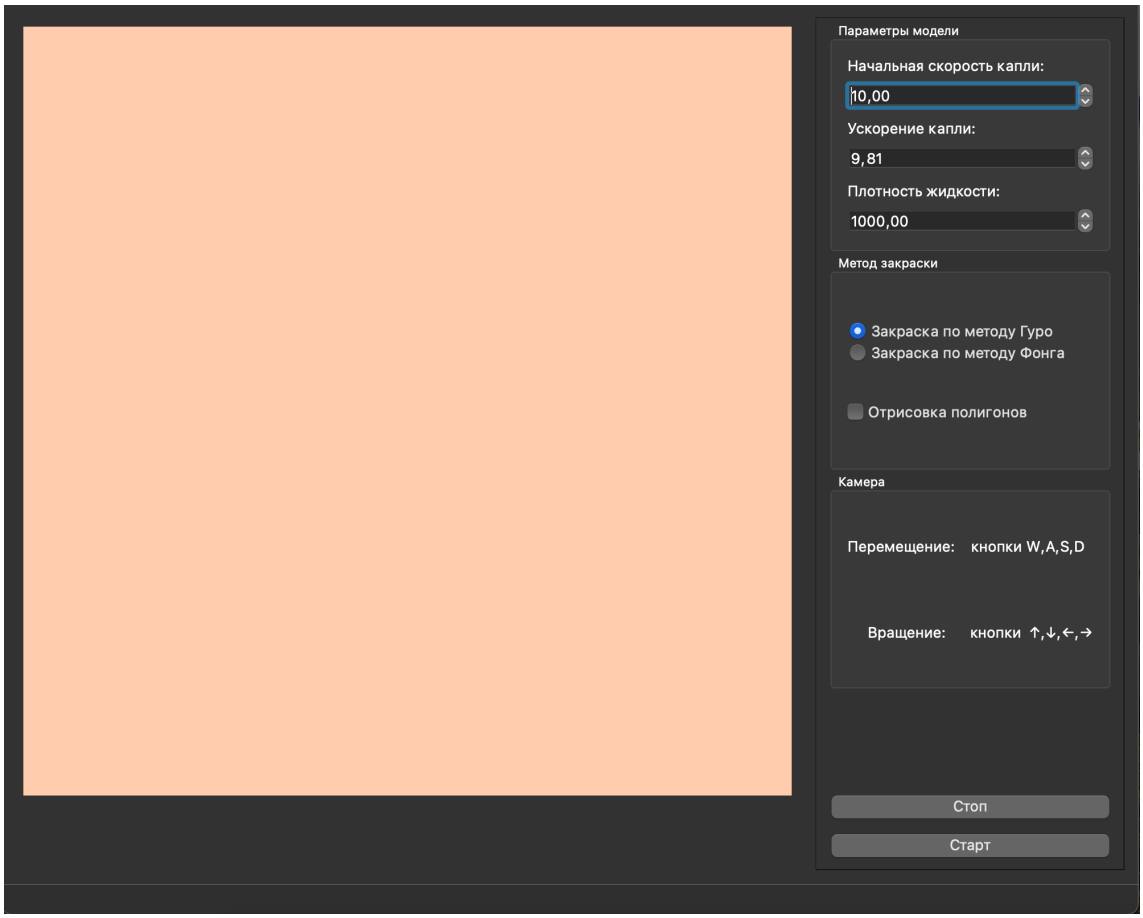

Рисунок 11 – Графический интерфейс программы

Для создания модели пользователю необходимо выбрать параметры начальной скорости капли, её ускорения и плотности жидкости в разделе «Параметры модели» (рисунок 12).

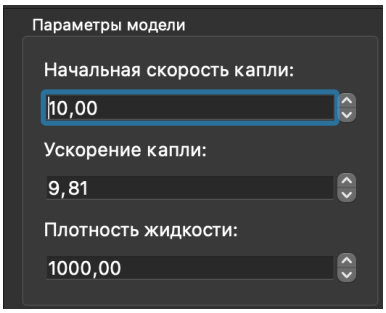

Рисунок 12 – Параметры модели

Далее пользователю необходимо выбрать метод закраски модели (Фонг, Гуро) и указать необходимость вывода полигонов в разделе «Метод закраски» (рисунок 13).

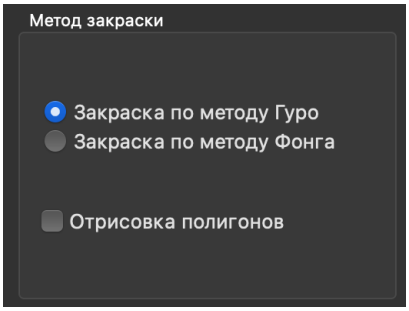

Рисунок 13 – Метод закраски

Управление камерой описано в разделе «Камера» (рисунок 14).

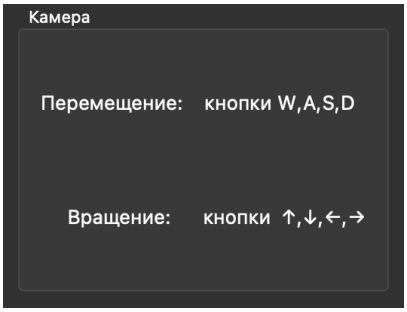

Рисунок 14 – Камера

Для создания модели на сцене необходимо нажать кнопку «Старт». Меню с настройками модели будет заблокировано, камера станет доступна для изменения своего расположения. Для изменения параметров модели или метода закраски необходимо нажать кнопку «Стоп».

На рисунке 15 приведен пример работы программы.

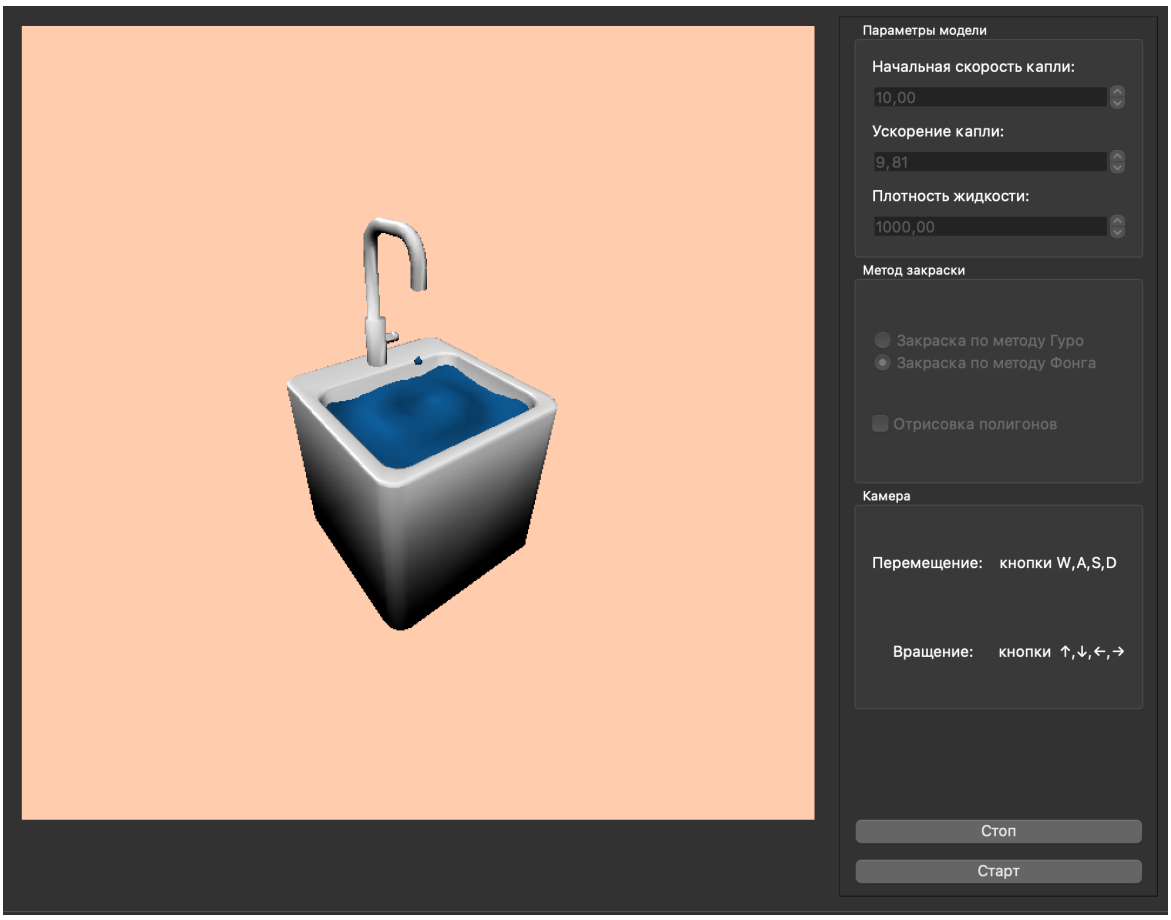

Рисунок 15 – Тестовая сцена

Вывод

В данном разделе были выбраны средства реализации, описаны структуры классов программы, описаны модули, а также рассмотрен интерфейс программы.

4 Исследовательская часть

4.1 Постановка исследования

Целью исследования является определение времени пересчета полигонов поверхности жидкости и времени закраски поверхности жидкости по методам Гуро и Фонга от количества полигонов.

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

— операционная система MacOS Ventura 13.5.2;

— 16 ГБ оперативной памяти;

— процессор 2,6 ГГц 6-ядерный Intel Core i7 [8].

Во время тестирования устройство было подключено к сети электропитания, нагружено приложениями окружения и самой системой тестирования.

4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи функции $process\_time()$ из библиотеки $time$ [9] языка $Python$ [6]. Данная функция возвращает текущее значение системного процессорного времени в секундах.

Замеры времени для каждого количества полигонов проводились 20 раз. В качестве результата взято среднее время работы алгоритма.

В таблице 1 приведены результаты замера времени пересчета полигонов поверхности жидкости (в микросекундах).

Таблица 1 – Результаты измерений времени пересчета полигонов поверхности жидкости

Количество полигонов, штук	Время, микросекунды
8	213
32	785
72	1 753
128	3 112
200	4 843
288	6 988
392	9 506
512	12 387
648	15 489
800	18 840
968	22 818
1 152	26 544
1 352	30 861
1 568	35 217

По таблице 1 был построен график на рисунке 16

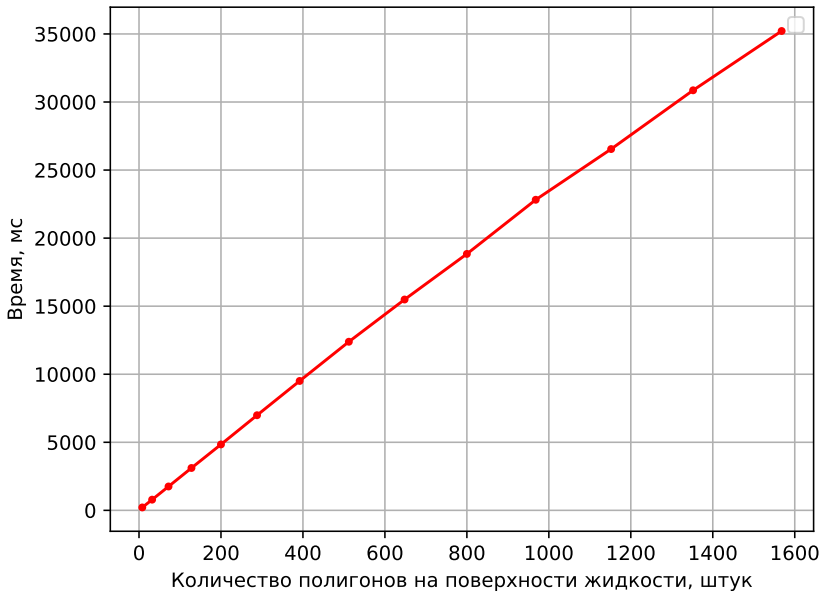

Рисунок 16 – График зависимости измерений времени пересчета полигонов поверхности жидкости от количества полигонов

Как видно из графика, время пересчета полигонов поверхности жидкости линейно зависит от количества полигонов.

В таблице 2 приведены результаты замера времени закраски поверхности жидкости по методам Гуро и Фонга (в микросекундах).

Таблица 2 – Результаты измерений времени закраски поверхности жидкости по методам Гуро и Фонга

Количество полигонов, штук	Время, микросекунды	
	Гуро	Фонг
8	134 711	920 098
32	158 717	963 961
72	184 150	1 011 580
128	206 442	1 072 216
200	250 857	1 135 146
288	300 670	1 245 661
392	369 820	1 356 393
512	440 931	1 477 853
648	523 032	1 574 341
800	604 238	1 722 877
968	704 506	1 871 794
1 152	812 945	2 009 417
1 352	935 350	2 162 179
1 568	1 054 775	2 308 075

По таблице 2 был построен график на рисунке 17

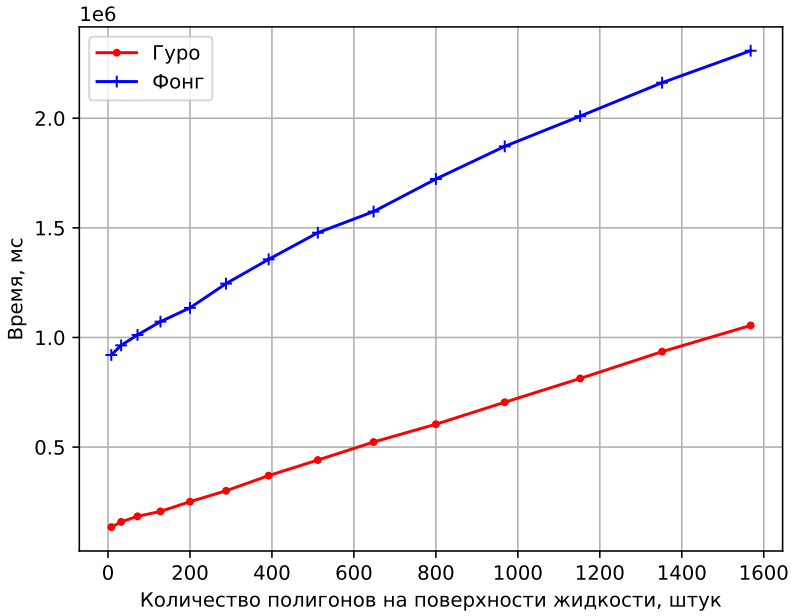

Рисунок 17 – График зависимости измерений времени закраски поверхности жидкости по методам Гуро и Фонга от количества полигонов

Как видно из графика, время закраски поверхности жидкости по методам Гуро и Фонга линейно зависит от количества полигонов, при этом метод закраски по Фонгу более чем в 2 раза медленне закраски по методу Гуро. Это обусловлено тем, что интенствность каждой точки полигона в методе Фонга рассчитывается отдельно по вектору номали и вектору направления источника освещения в данной точке полигона. В закраске по методу Гуро данная операция производится только в вершинах полгона, далее полученные значения интенствности интерполируются.

Вывод

В данном разделе были исследованы результаты измерения времени пересчета полигонов поверхности жидкости и времени закраски поверхности жидкости по методам Гуро и Фонга от количества полигонов.

Заключение

В ходе выполнения курсовой работы поставленная цель была достигнута: было разработано программное обеспечение, позволяющее моделировать генерацию круговых волн на поверхности жидкости.

Для достижение цели были выполнены следующие задачи:

— был произведен анализ существующих алгоритмов компьютерной графики;

— были выбраны наиболее подходящие алгоритмы для достижения поставленой цели;

— были выбраны средства реализации программного обеспечения;

— было разработано программное обеспечение и реализовать выбранные алгоритмы и структуры данных;

— были проведены замеры временных характеристик разработанного программного обеспечения.

Реализованную программу можно усовершенствовать добавив возможность сложения нескольких волн для наблюдения интеференции.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методы трехмерного моделирования [Электронный ресурс]. —. — Режим доступа: https://studopedia.ru/19_307536_metodi-trehmernogo-modelirovaniya-karkasnoe-modelirovanie-poverhnostnoe-tverdotelnoe-modelirovanie-tipi-poverhnostey-chto-predstavlyayut-s-soboy-trehmernie-ob-ekti.html (дата обращения: 10.7.2023).
2. *Роджерс Д.* Алгоритмические основы машинной графики. — 1989.
3. Простые модели освещения [Электронный ресурс]. —. — Режим доступа: https://cgraph.ru/node/435 (дата обращения: 18.7.2023).
4. Placing a Camera: the LookAt Function [Электронный ресурс]. — Режим доступа: https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/lookat-function/framing-lookat-function.html (дата обращения: 18.9.2023).
5. *Куров А. В.* Моделирование волн на поверхности жидкости. — МГТУ им. Н.Э. Баумана, 2013.
6. Welcome to Python [Электронный ресурс]. — Режим доступа: https://www.python.org (дата обращения: 18.9.2023).
7. All Qt Documentation [Электронный ресурс]. — Режим доступа: https://doc.qt.io/all-topics.html (дата обращения: 17.9.2023).
8. Intel [Электронный ресурс]. — Режим доступа: https://www.intel.com/content/www/us/en/products/details/processors/core/i7.html (дата обращения: 18.9.2023).
9. time — Time access and conversions [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 19.9.2023).