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 КУРСОВОМУ ПРОЕКТУ
НА ТЕМУ:
«Программа построения реалистичного изображения сцены, состоящей из трехмерных объектов»

Студент ИУ7-55Б _____ Клименко А. К.
(Группа) (Подпись, дата) (Фамилия И. О.)

Руководитель курсовой работы _____ Майков К. А.
(Подпись, дата) (Фамилия И. О.)

2021 г.

Содержание

Введение

В современном мире всё более и более востребованным становится использование компьютерной графики в разных сферах деятельности. Машинная графика используется в компьютерном моделировании, компьютерных играх и системах автоматизированного проектирования [1]. В связи с нарастающей потребностью к реалистичному изображению трехмерных объектов, появляется необходимость создавать программное обеспечение, способное учитывать широкий спектр оптических явлений при отрисовке изображения [2].

В компьютерной графике алгоритмы создания реалистичного изображения требуют особых затрат ресурсов, тратится много времени и памяти для получения результата. Затраты времени на синтез изображения не всегда позволяют удобно и эффективно подобрать параметры отрисовки.

Целью данной работы является создание программного обеспечения, которое позволило бы синтезировать изображение трехмерной сцены с настраиваемыми параметрами ее объектов в реальном времени и сменой режимов быстродействия.

Для достижения поставленной цели необходимо решить следующие задачи:

- исследовать подходы к синтезу реалистичных изображений;
- сравнить существующие решения;
- описать структуру разрабатываемого ПО;
- описать используемые при разработке ПО алгоритмы;
- определить средства программной реализации;

- реализовать алгоритмы отрисовки сцены;
- протестировать разработанное ПО;
- сравнить производительность реализованных алгоритмов на основе разных сцен.

1 Аналитический раздел

В данном разделе проводится исследование подходов к синтезу реалистичных изображений, осуществляется описание сцены, трехмерных объектов в сцене, проводится сравнительный анализ различных алгоритмов и обзор существующих решений.

1.1 Описание сцены

В компьютерной графике сцена является неотъемлемым компонентом, содержащим в себе всю информацию, необходимую для синтеза изображения. Поэтому определение ее структуры является одним из важнейших и первостепенных шагов в создании программы.

В реализовываемом программном продукте сцена будет представлена структурой, содержащей в себе коллекцию из объектов и коллекцию из источников освещения. Размеры этих коллекций будут ограничены лишь объемом оперативной памяти.

1.2 Описание моделей трехмерных объектов в сцене

Существует множество способов представления трехмерных объектов в сцене. Различным алгоритмам отрисовки требуется разная информация о самом объекте.

1.2.1 Функциональное моделирование

Одним из простейших способов задания трехмерных объектов можно назвать функциональное моделирование. Возможно несколько вариан-

тов.

1. Поверхность такого объекта задается системой уравнений:

$$\begin{cases} x = x(u, v) \\ y = y(u, v) \\ z = z(u, v) \end{cases}, \tag{1.1}$$

   в этом случае каждая точка на поверхности объекта из плоской системы координат (u, v) преобразуется в мировое пространство (x, y, z).

2. Внутренний объем такого объекта выражается неравенством:

$$f(x, y, z) > 0. \tag{1.2}$$

   Данное неравенство справедливо для всех точек внутри объекта.

   Особенности данного представления:

- функциональное моделирование позволяет получить максимальный уровень детализации при отрисовке, наилучшую сглаженность формы объекта;
- достаточно сложная форма может иметь довольно простое параметрическое представление, из чего следует меньший расход памяти при хранении представления объекта;
- трансформации над объектом производятся очень просто. Достаточно применить преобразования к описывающим объект уравнениям или неравенствам;
- наложение текстур в случае с внутренне-определёнными объектами является затруднительным;
- процесс отыскания параметрического представления невыпуклых многогранников может быть затруднительным.

1.2.2 Полигональная сетка

Полигональная сетка состоит из набора отдельных полигонов, которые в свою очередь являются многоугольниками, ребра которых соединяют вершины сетки. [3]. Структура полигональной модели представлена ниже, на рисунке 1.1

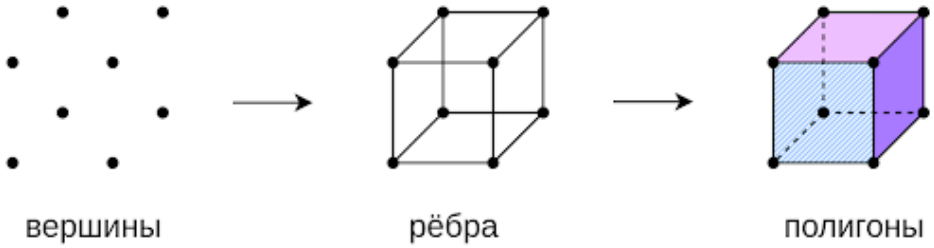

Рис. 1.1: Структура полигональной модели.

Полигональная сетка является аппроксимацией поверхности представляемого объекта. От её плотности, т.е. от количества полигонов на единицу объема, зависит качество изображения объекта.

Особенности данного представления:

- просто создать топологию поверхности;
- преобразования над объектом сводятся к преобразованию полигональной сетки, что в свою очередь раскладывается на независимые преобразования над отдельными полигонами;
- наложение текстур на поверхность является несложной процедурой;
- повышение плотности сетки приводит к увеличению объема затрачиваемой памяти для ее хранения.

1.2.3 Воксельная сетка

Воксел – это элемент объема. Воксельная модель разбивает весь объем трехмерного изображения на ячейки – вокселы, создавая трехмерный растр [4].

1.2.4 Вывод

Для решения поставленных задач лучше всего подходит метод с использованием полигональной сетки для представления поверхностей объектов, так как он рассчитан на широкий круг задач моделирования, позволяет легко модифицировать положение объекта в пространстве, а также прост в реализации.

1.3 Методы отсечения невидимых линий

Для интерактивного режима очень важно быстро получить набросок будущего изображения, для этого подойдут методы, которые не учитывают оптические явления не влияющие на форму объектов в пространстве изображения. Данное упрощение в значительной степени позволит сократить время на синтез изображения.

1.3.1 Алгоритм Варнока

Алгоритм Варнока работает по принципу “Разделяй и властвуй” и разбивает окно на подокна в случаях, когда его отрисовка нетривиальна. Такие разбиения могут продолжаться до тех пор, пока размер окна не станет равным единице растеризации или пока в окне не останется всего одной простой для отрисовки фигуры.

В алгоритме Варнока и его вариациях делается попытка извлечь преимущество из того факта, что большие области изображения однородны.

1.3.2 Алгоритм Вейлера-Азертона

Алгоритм Вейлера-Азертона позволяет выполнить отсечение невидимой части объекта перед его отрисовкой по контурам загораживающих его объектов. Этот прием позволяет не делать лишнюю работу при отрисовке частично видимых объектов.

1.3.3 Алгоритм с использованием Z-буфера

Алгоритм с использованием Z-буфера базируется на том, что вместо того, чтобы определять очередность в которой должны отрисовываться объекты, для каждого пикселя при отрисовке сохраняется дополнительный атрибут - глубина, использование которого гарантирует корректное отображение пикселя, соответствующего наиболее близкому объекту.

1.3.4 Вывод

Для программной реализации отображения сцены на экран в интерактивном режиме было решено использовать алгоритм Z-буфера.

1.4 Обзор методов построения реалистичного изображения

Рассмотрим методы постоения реалистичного изображения, основанные на физической модели распространения света.

1.4.1 Трассировка лучей

Трассировка лучей – алгоритм основанный на отслеживании пути, преодолеваемом лучом света от камеры через сцену, и расчете отражения, преломления, поглощения света при пересечении с объектами сцены [5].

Луч света исходит из источника света, причем при пересечении с поверхностью и может произойти поглощение, отражение или преломление. После этого отраженные лучи могут пересечься с другими поверхностя-

ми. Часть из таких лучей попадает в глаз наблюдателю и формируется изображение.

Основное ограничение метода – его производительность. Рекурсивная природа и большое количество лучей приводят к тому, что синтез изображения может занимать до нескольких часов [6].

1.4.2 Метод радиосити

Метод радиосити основан на теории теплоизучения, ибо полагается на расчеты количества световой энергии, переходящей между двумя поверхностями. Для каждой поверхности рассчитывается энергия, покидающая поверхность [7].

Отраженная энергия рассчитывается на основе форм факторов – долей от общей излучающей площади первой поверхности, которая перекрывается второй поверхностью.

1.4.3 Вывод

Для получение реалистичного изображения в реализуемом программном продукте будет использован алгоритм трассировки лучей. Данный выбор обусловлен простотой реализации без использования графических процессоров. Также этот метод позволяет синтезировать реалистичные изображения с тенями.

2 Конструкторский раздел

В данном разделе описаны структура разрабатываемого ПО, используемые структуры данных, а также приведены схемы алгоритмов используемых при отрисовке изображения.

В данном разделе описаны требования, которым должно удовлетворять ПО, структура разрабатываемого ПО, используемые структуры данных, а также приведены схемы алгоритмов используемых при отрисовке изображения.

2.1 Требования к ПО

Разрабатываемое ПО должно предоставлять следующий функционал:

- загрузка заготовленных полигональных моделей из файлов;
- добавление в сцену объектов из заготовленного набора примитивов;
- независимое трансформирование положения объектов в сцене;
- навигация по сцене с помощью перемещения и поворота главной камеры;
- просмотр и изменение специфических свойств объектов в сцене;
- сохранение текущего изображения сцены в файл на диске;
- сохранение созданной сцены в файл на диске;
- загрузка готовой сцены из файла.

2.2 Структура ПО

2.2.1 Схема доменов

Программное обеспечение будет состоять из трех доменов: прикладного, архитектурного и интерфейсного. Прикладной домен будет отвечать за процесс визуализации сцены, интерфейсный - за взаимодействие с пользователем, а архитектурный послужит связующим звеном между двумя вышеупомянутыми доменами.

На рисунке 2.1 представлена схема доменов.

Рис. 2.1: Схема доменов

2.2.2 Прикладной домен

Для прикладного домена были подготовлены информационная модель и схема иерархии по данным.

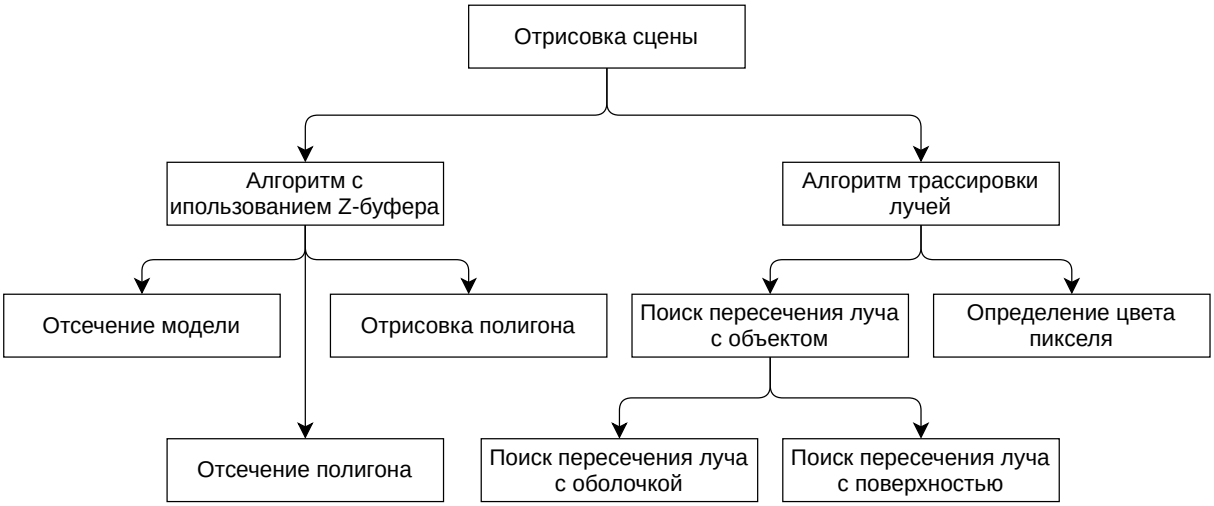

Рис. 2.2: Информационная модель прикладного домена

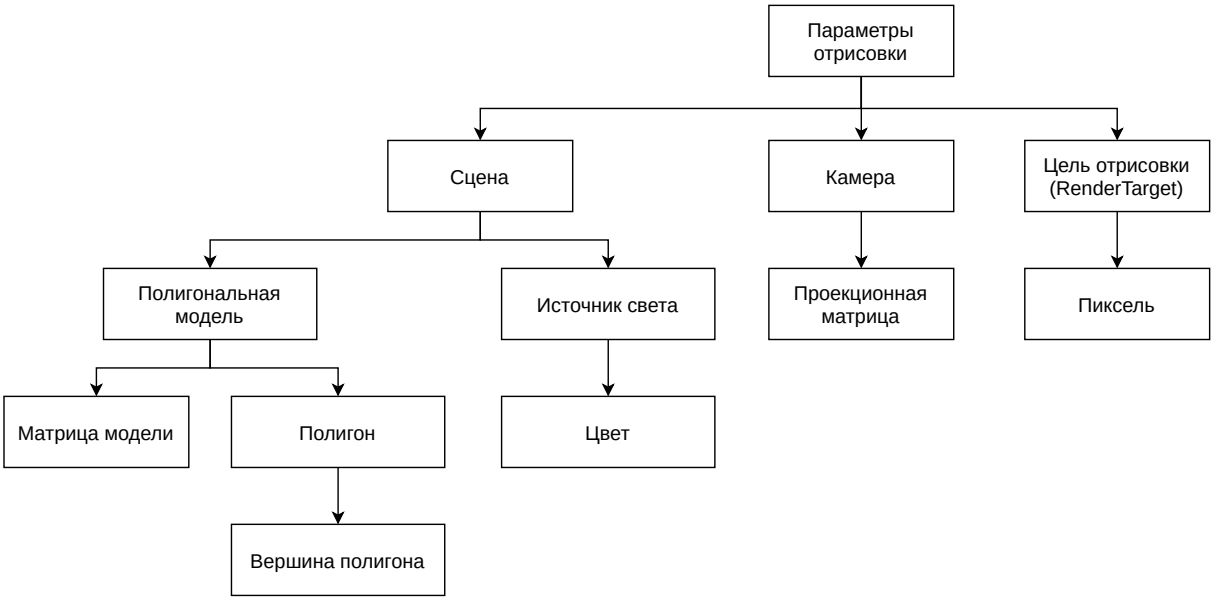

Рис. 2.3: Иерархия прикладного домена по данным

2.2.3 Архитектурный домен

В приложении А на рисунке 4.2 приведена иерархия классов для архитектурного домена.

2.2.4 Процесс синтеза изображения

Ниже на рисунках 2.4–2.7 представлена IDEF0 диаграмма процесса синтеза изображения сцены.

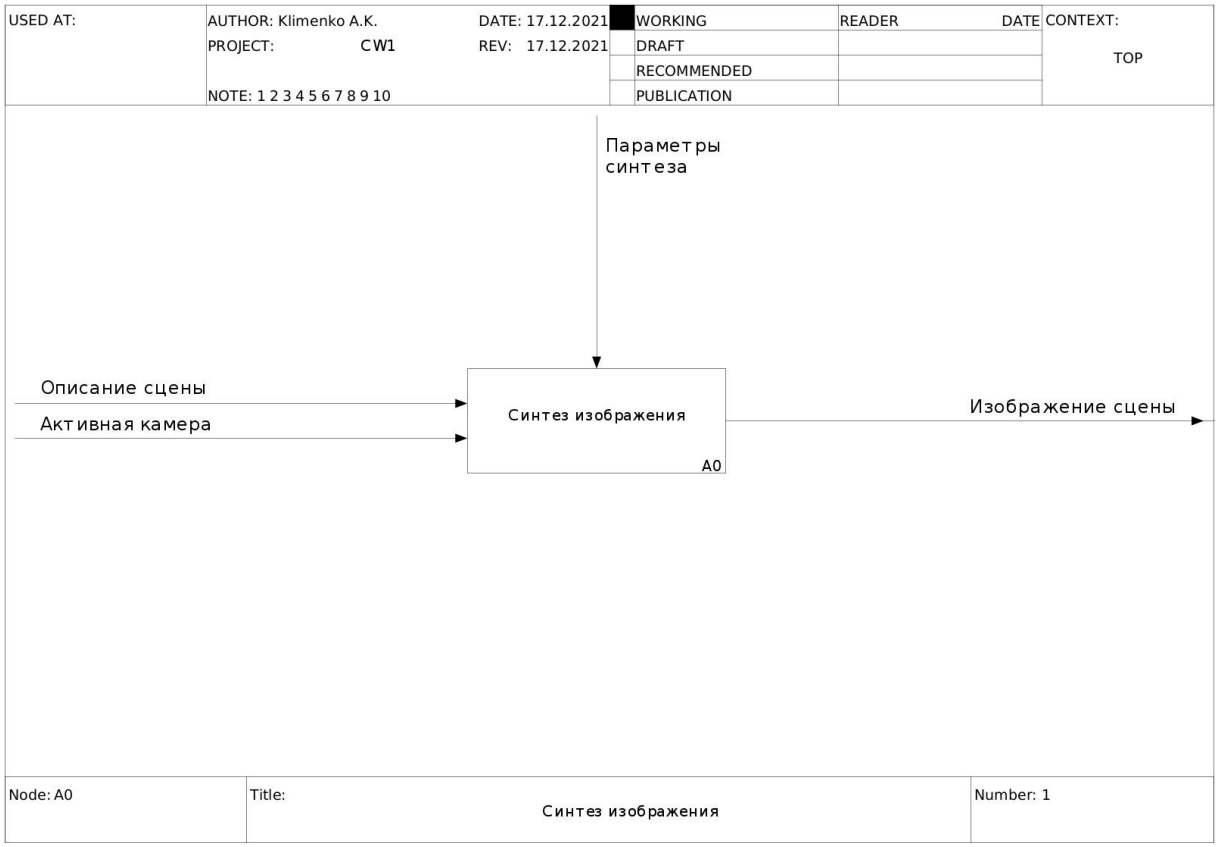

Рис. 2.4: IDEF0 диаграмма процесса синтеза изображения (часть 1)

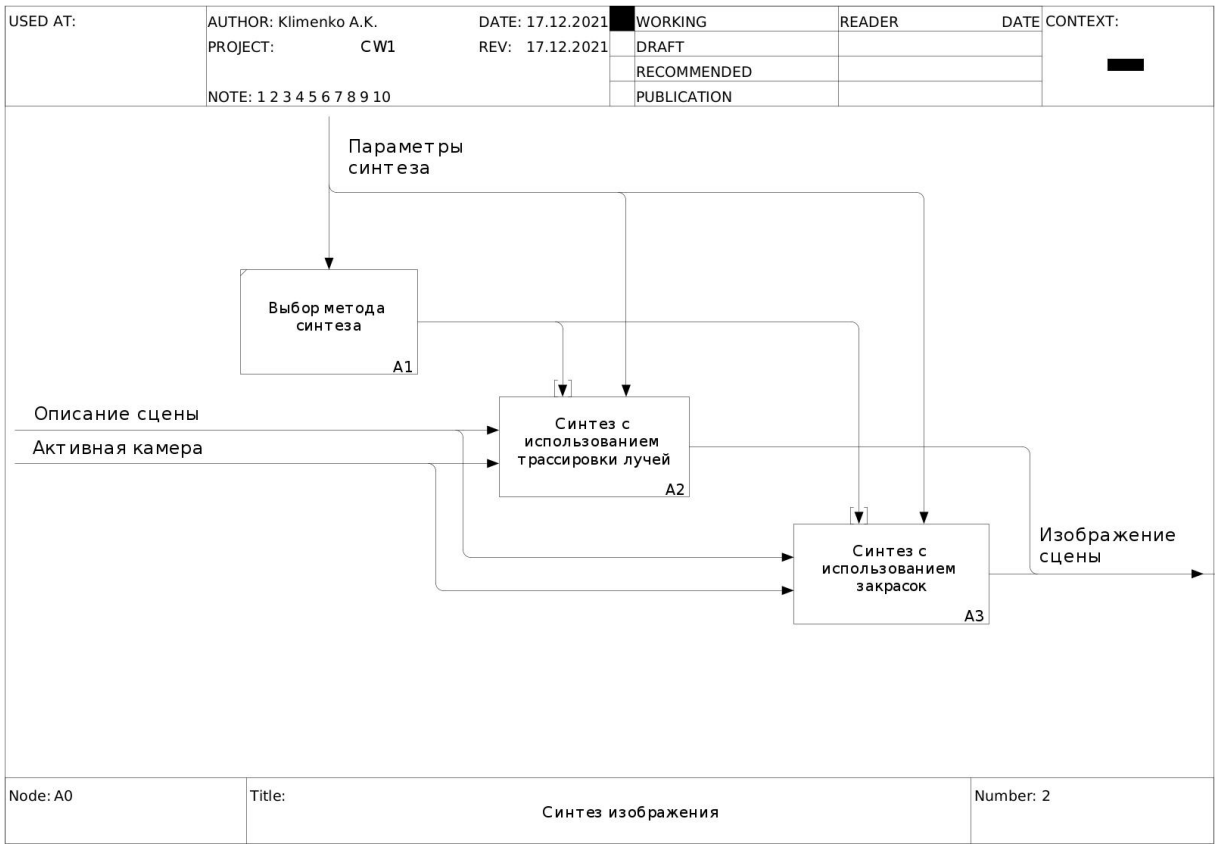

Рис. 2.5: IDEF0 диаграмма процесса синтеза изображения (часть 2)

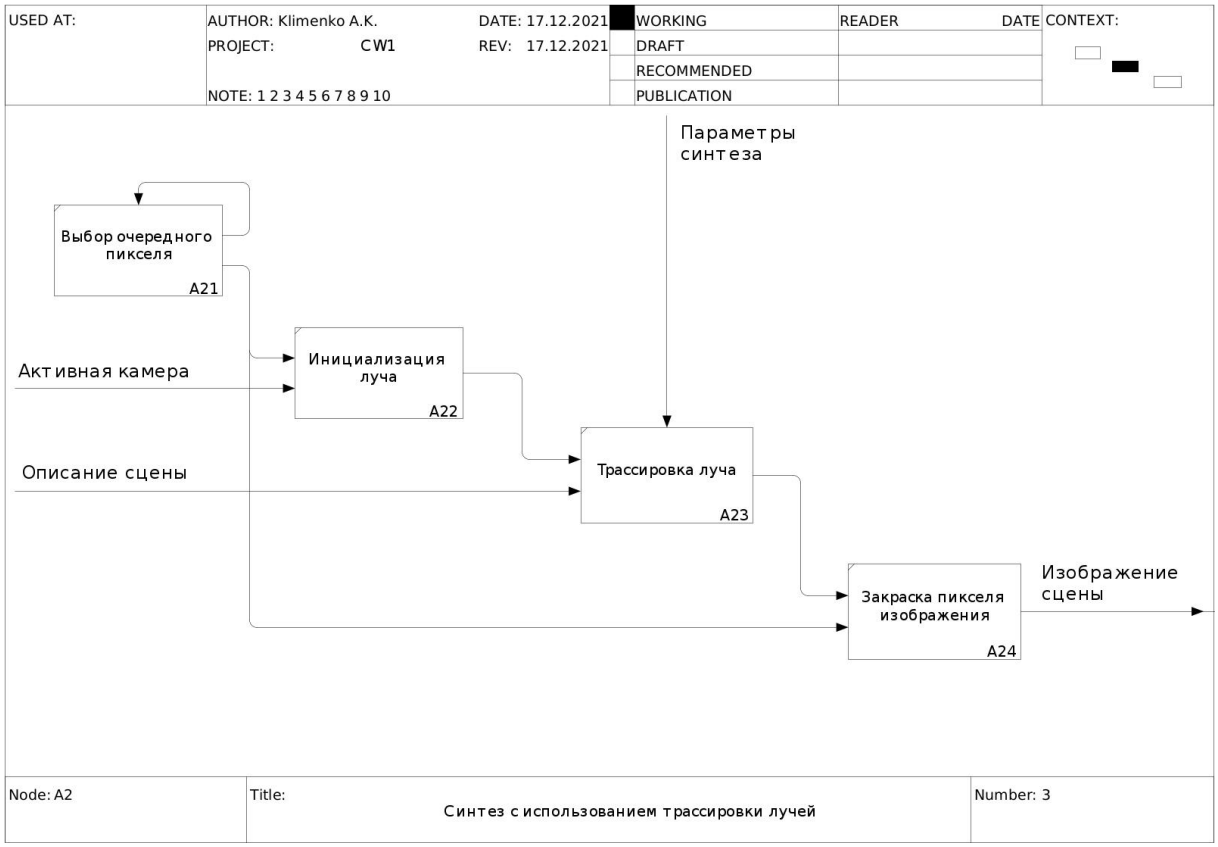

Рис. 2.6: IDEF0 диаграмма процесса синтеза изображения (часть 3)

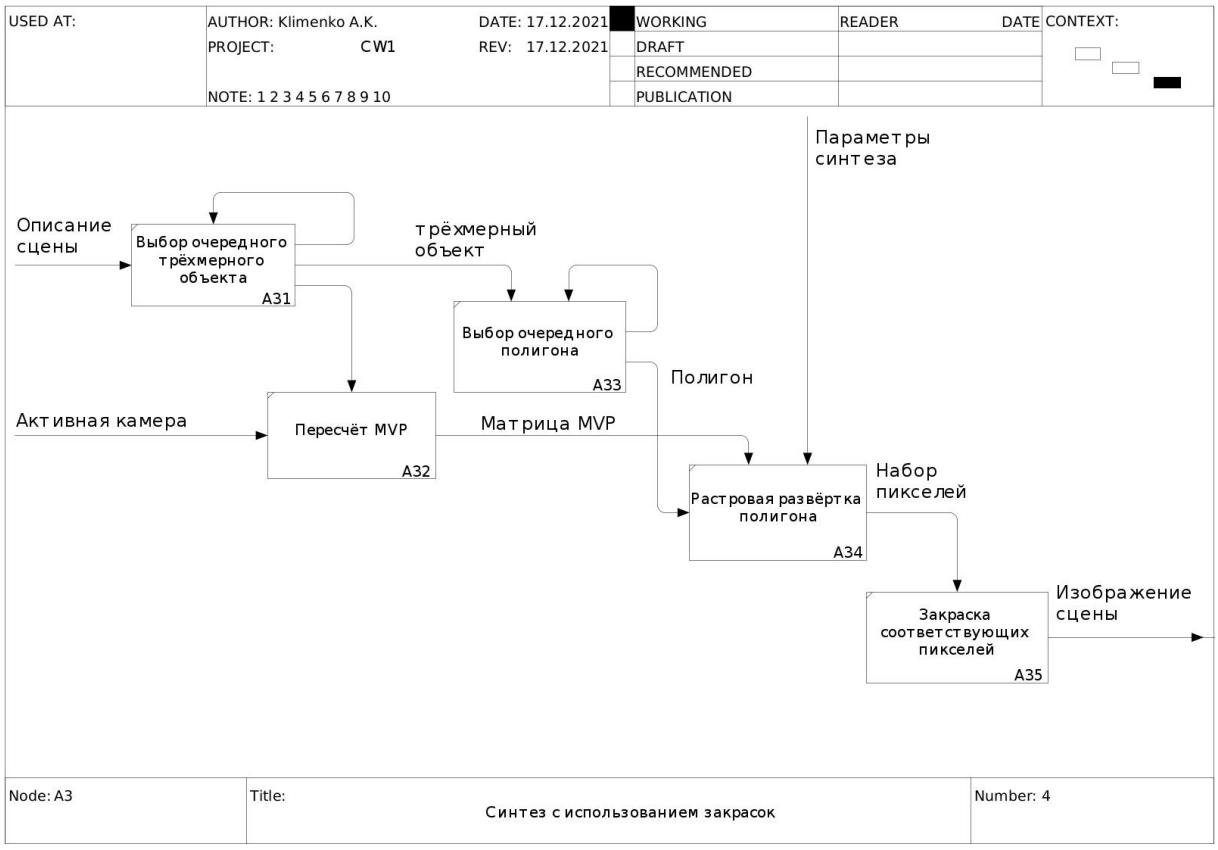

Рис. 2.7: IDEF0 диаграмма процесса синтеза изображения (часть 4)

2.3 Схемы алгоритмов

В данной секции представлены 2 алгоритма синтеза изображения. Первый использует закраски полигонов, а второй – метод трассировки лучей.

2.3.1 Быстрая отрисовка

На рисунках 2.8–2.9 представлен алгоритм синтеза изображения, использующий простые закраски.

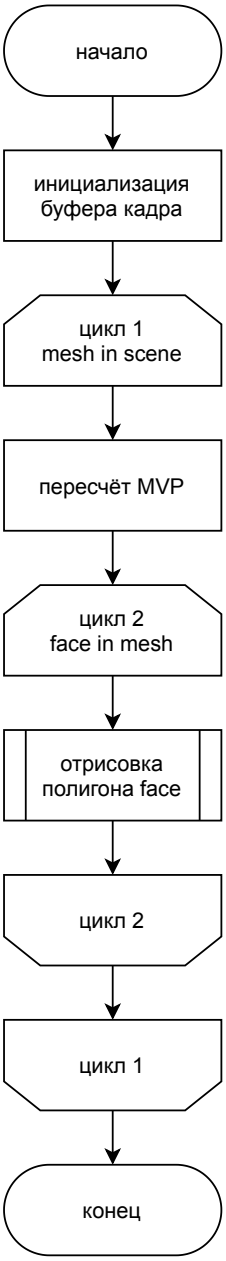

Рис. 2.8: Алгоритм, использующий простую закраску (часть 1)

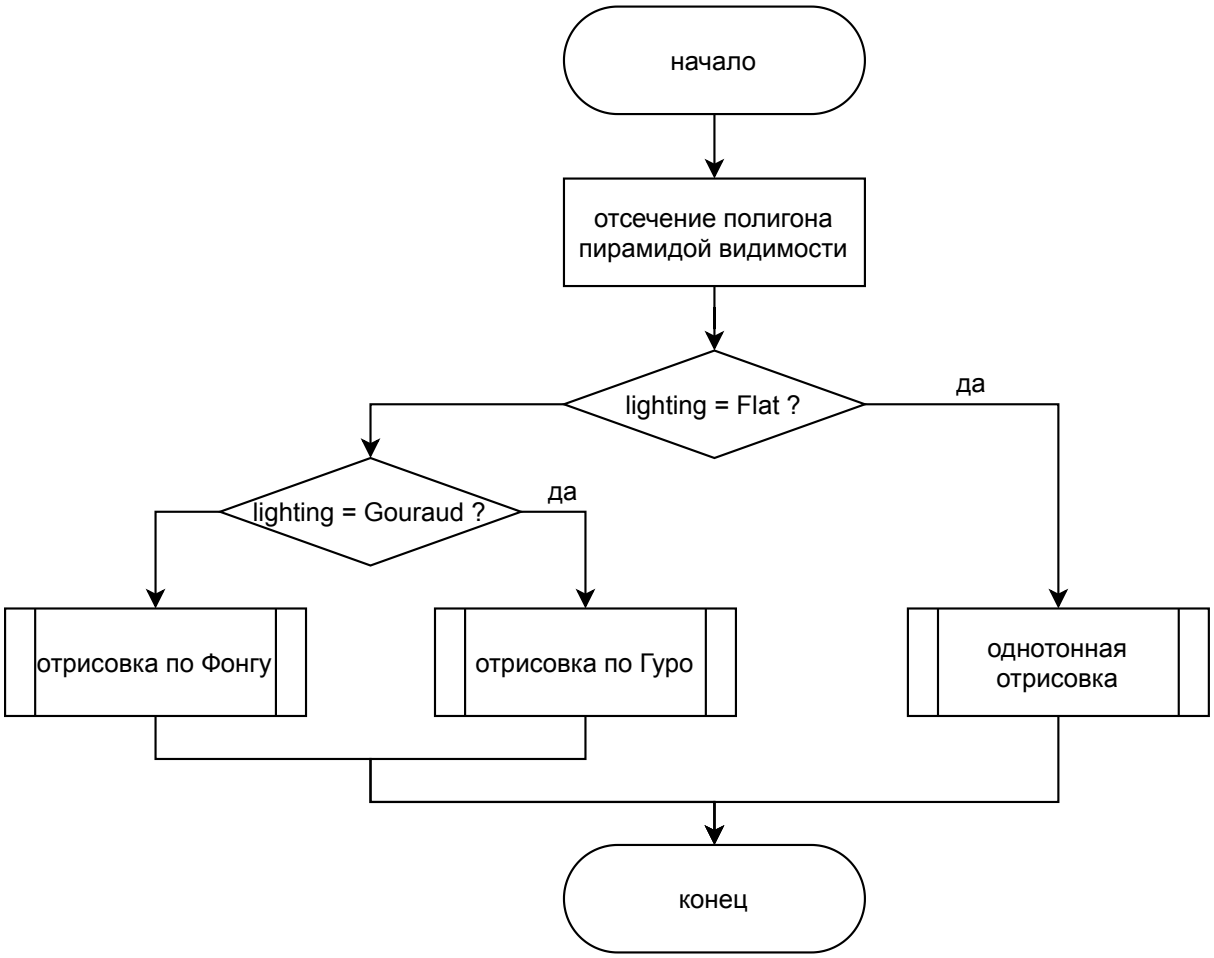

Рис. 2.9: Алгоритм, использующий простую закраску (часть 2)

2.3.2 Реалистичная отрисовка

На рисунке 2.10 представлена схема алгоритма трассировки лучей.

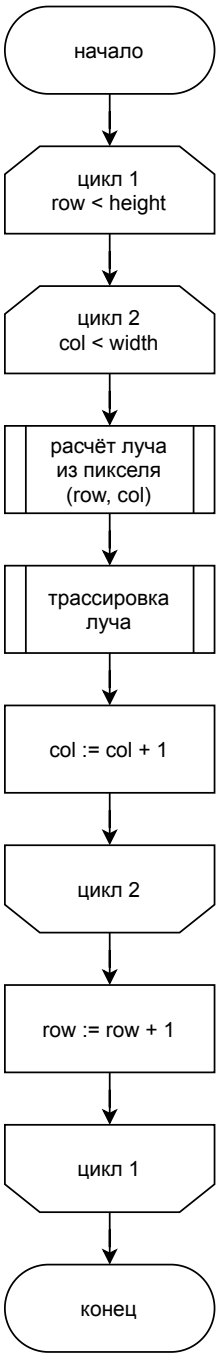

Рис. 2.10: Алгоритм, использующий трассировку лучей

2.4 Структуры данных

В данной подсекции описываются основные структуры данных, необходмые для решения задачи синтеза изображения.

2.4.1 Сцена

В рассматриваемой задаче синтеза изображения сцена является элементом самого верхнего уровня, в полной мере содержащим информацию об объектах, находящихся на сцене, а также информацию об используемом освещении.

Данная структура данных будет использоваться прежде всего для отображения объектов, поэтому основным требованием, предъявляемым к ней, будет эфективная выборка объектов сцены. Из этого можно заключить, что наиболее выгодным с точки зрения производительности будет выбор массива в качестве контейнера для объектов сцены и источников освещения.

2.4.2 Трёхмерное тело

Рассматривая задачу синтеза изображения, любое трёхмерный объект можно разложить на две составляющие – форму тела (его геометрию), и оптические свойства поверхности материала, из которого изготовлен объект.

Выбор структуры трёхмерного тела должен выполняться в соответствии с выбранными алгоритмами решения поставленной задачи. В следствии чего было принято решение об использовании массива треугольников в качестве описания геометрии объекта.

Оптические свойства поверхности объекта будут описаны в отдельной структуре – материал. Она будет включать в себя такие параметры, как: цвет фонового освещения, цвет направленного освещения, цвет бликов. Также данная структура будет содержать следующие параметры для синтеза реалистичного изображения: коэффициенты прозрачности, отражния и преломления, а также показатель преломления среды.

2.4.3 Источник освещения

В постановке задачи синтеза изображения выделены следующие типы источников освещения: фоновый, направленный и точечный. Далее каждый тип будет рассмотрен в отдельности.

Фоновое освещение. Фоновая освещённость объектов в сцене не зависит от положения источника света в пространстве, а зависит только от цвета освещения и его интенсивности.

Направленное освещение. Для данного типа освещения необходимо хранить его цвет, интенсивность и направление.

Точечное освещение. Данный тип освещения регулируется положением источника в пространстве, радиусом освещения и коэффициентами затухания.

3 Технологический раздел

Данный раздел посвящен выбору средств для программной реализации разработанного метода, описанию основных моментов разработки программного обеспечения.

3.1 Инструменты разработки

Для написания программы был выбран язык программирования C++[8] по следующим причинам:

- C++ имеет статическую типизацию, является компилируемым языком программирования;
- он позволяет работать с памятью напрямую, а также контролировать все вызовы к менеджеру памяти;
- является объекто-ориентированным языком, но также позволяет писать программы в структурном стиле;
- имеет мощный механизм шаблонов.

Для раработки графического интерфейса будет использована библиотека Qt[9], так как данная библиотека имеет открытый исходный код, а также является платформо-независимой.

В качестве среды разработки будет использована «CLion»[10] по следующим причинам:

- данная среда разработки является кросплатформенной;
- она имеет встроенный редактор с функцией интеллектуального автодополнения кода;

- она имеет удобный в использовании многофункциональный отладчик.

3.2 Листинги реализаций алгоритмов

В данном подразделе будут представлены реализации алгоритмов синтеза изображения.

На листинге 3.1 представлена реализация алгоритма трассировки лучей.

Листинг 3.1: Реализация алгоритма трассировки лучей

```
static std::pair<double, Color> traceRay(Ray ray, const Scene& scene, const Camera&
    camera, int depthLeft) {
  Mesh* mesh;
  Face* face;
  double distance;

  if (depthLeft > 0 && findIntersectionPoint(mesh, face, distance, ray, scene)) {
    Vec position = ray_at(ray, distance);
    Vec view = normalized(position - camera.eye);
    Mat view_mat = inverse(camera.model_mat);
    Vec normal = get_normal_at(mesh->model_mat * *face, position);

    Color colorReflected = Colors::black;
    Color colorTransmitted = Colors::black;
    Color colorLighted = Colors::black;

    if (mesh->material.coefReflection > 0) {
      Vec reflected = ray.direction - 2 * dot(ray.direction, normal) * normal;
      auto traced = traceRay(make_ray(position + 0.01 * reflected, reflected), scene,
          camera, depthLeft - 1);
      if (traced.first < std::numeric_limits<double>::infinity())
        colorReflected += mesh->material.coefReflection * traced.second;
    }
    if (mesh->material.coefRefraction > 0) {
      double d = dot(-ray.direction, normal);
      if (d > 0) {
        double theta = std::acos(d);
        Vec m = (normal * std::cos(theta) + ray.direction) / std::sin(theta);
        Vec refracted = normalized(m * std::sin(theta) - normal * std::cos(theta));
        auto traced = traceRay(make_ray(position + 0.01 * refracted, refracted),
            scene, camera, depthLeft - 1);
        if (traced.first < std::numeric_limits<double>::infinity())
          colorTransmitted += mesh->material.coefRefraction * traced.second;
      }
```

```
32      }
33
34      for (size_t i = 0; i < scene.lightList.size; i++) {
35        Light light;
36
37        if (!get(scene.lightList, i, light))
38          return {std::numeric_limits<double>::infinity(), Colors::black};
39        else if (!light.visible)
40          continue;
41        else {
42          if (light.type == LightType::Ambient) {
43            colorLighted += light.intensity * light.color * mesh->material.ambientColor;
44          }
45          else if (light.type == LightType::Directional) {
46            if (dot(normal, -light.direction) > 0) {
47              auto traced = traceRay(make_ray(position - 0.1 * light.direction,
                    -light.direction), scene, camera, depthLeft - 1);
48              if (traced.first == std::numeric_limits<double>::infinity()) {
49                colorLighted += dot(normal, -light.direction) * light.intensity *
                      light.color;
50              }
51            }
52          }
53          else if (light.type == LightType::Point) {
54            auto traced = traceRay(make_ray(position, normalized(light.position -
                  position)), scene, camera, depthLeft - 1);
55            colorLighted += traced.second;
56          }
57        }
58      }
59
60      double op = mesh->material.opacity;
61      Color color = op * colorLighted + (1 - op) * colorTransmitted + colorReflected;
62
63      return {distance, color};
64    }
65
66    Color color = Colors::black;
67
68    for (size_t i = 0; i < scene.lightList.size; i++) {
69      Light light{};
70      get(scene.lightList, i, light);
71
72      if (!light.visible)
73        continue;
74      else if (light.type == LightType::Ambient)
75        color += light.intensity * light.color;
76      else if (light.type == LightType::Directional) {
```

```
77      double val = dot(ray.direction, -light.direction);
78      if (val > 0)
79        color += val * light.intensity * light.color;
80    }
81  }
82
83  return {std::numeric_limits<double>::infinity(), color};
84 }
```

Листинг 3.2 содержит реализацию алгоритма сплошной закраски полигона.

Листинг 3.2: Реализация алгоритма сплошной закраски полигона

```
1  void Core::renderFlat(RenderTarget& renderTarget, ZBuffer& zbuffer, RenderRegion
       region, ColorComputeFn colorComputeFn) {
2    double xLeft = region.xStartLeft;
3    double xRight = region.xStartRight;
4
5    double zLeft = region.zStartLeft;
6    double zRight = region.zStartRight;
7
8    Pixel color = to_pixel(colorComputeFn(region.center, region.normal,
         region.meshPtr->material));
9
10   for (int y = region.yStart; y <= region.yEnd; y++) {
11     double z = zLeft;
12     double dz = (zRight - zLeft) / (xRight - xLeft + 1);
13
14     if (0 <= y && y < renderTarget.height) {
15       for (auto x = (int)xLeft; x < xRight; x++) {
16         if (0 <= x && x < renderTarget.width)
17           updatePixel(renderTarget, zbuffer, y, x, 1.0 / z, color);
18
19         z += dz;
20       }
21     }
22
23     xLeft += region.dxLeft;
24     xRight += region.dxRight;
25
26     zLeft += region.dzLeft;
27     zRight += region.dzRight;
28   }
29 }
```

4 Исследовательский раздел

В данном разделе представлены эксперименты для проведения сравнительного анализа быстродействия реализуемых алгоритмов синтеза изображения.

4.1 Постановка эксперимента

В эксперименте исследуется зависимость времени выполнения алгоритмов синтеза изобржения в зависимости от количества объектов сцены при двух и пяти различных источниках освещения.

При проведении эксперимента все объекты сцены будут расположены в поле зрения камеры, чтобы исключить случаи, когда реальная отрисовка объекта не выполняется.

Количество объектов сцены будет варьироваться от 1 по 7 с шагом 2.

Для каждого случая будет рассчитано время синтеза изображения алгоритмом предобработки (быстрая отрисовка) и алгоритмом трассировки лучей.

Полученный результат не усредняется.

4.2 Технические характеристики

Ниже приведены характеристики устройства, на котором будет производиться эксперимент:

- операционная система: Pop!_OS 21.04 [11] Linux 64-bit;
- оперативная память: 8Gb;
- процессор: Intel Core i7-8750H @ 12x 4.1 GHz.

4.3 Результаты эксперимента

Ниже в таблице 4.1 представлены результаты эксперимента.

Таблица 4.1: Время работы алгоритмов при различных параметрах сцены

кол-во объектов в сцене	кол-во источников освещения	время закраски	время трассировки
1	2	10.3 мс	10.5 с
3	2	12.2 мс	25.7 с
5	2	15.4 мс	107 с
7	2	31.5 мс	174 с
1	5	10.3 мс	14.8 с
3	5	13.7 мс	62.1 с
5	5	15.9 мс	334 с
7	5	35.8 мс	1520 с

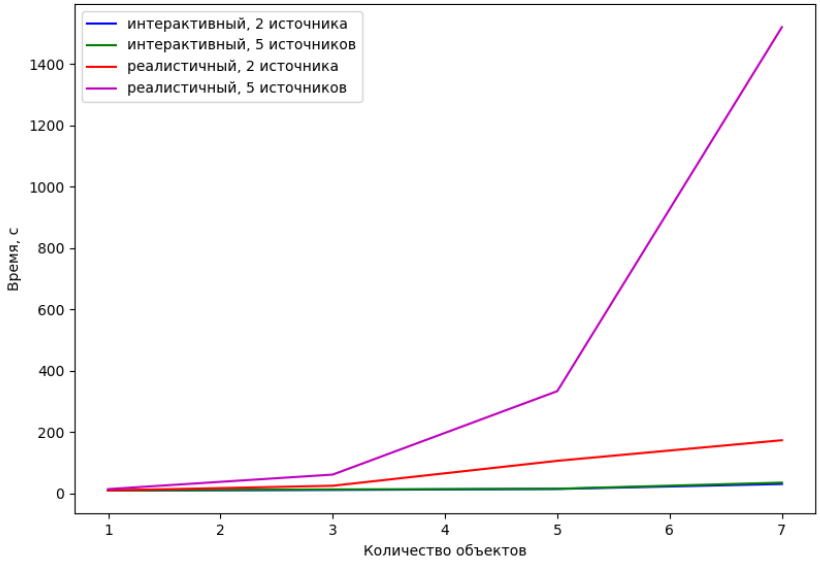

Рис. 4.1: Время работы алгоритмов при различных параметрах сцены

Как видно из рисунка 4.1, время затрачиваемое на отрисовку с использованием трассировки лучей в тысячу раз превосходит время затрачиваемое быстрым алгоритмом.

4.4 Вывод

В данном разделе был поставлен эксперимент по сравнению производительности интерактивного и реалистичного режимов синтеза изображения в разработанном ПО при различных параметрах сцены.

Количество источников освещения незначительно влияет на время при быстрой закраске, при методе трассировки лучей на порядок увеличивает время синтеза изображения.

По результатам эксперимента было выявлено, что интерактивный режим выгодно использовать при настройке параметров сцены для дальнейшего синтеза реалистичного изображения, время получение которого в тысячи раз превышает время отрисовки в интерактивном режиме.

Заключение

В результате выполнения курсовой работы были исследованы основные подходы к синтезу реалистичных изображений, описана структура разрабатываемого программного обеспечения, а также описаны разрабатываемые алгоритмы и структуры данных. Был разработан программный продукт, ключевой особенностью которого является возможность интерактивной смены параметров сцены для последующего синтеза реалистичного изображения. Интерфейс программного продукта представлен в приложении Б на рисунках 4.3–4.5.

Программа реализована таким образом, что пользователь может добавлять и изменять объекты, редактировать освещение, изменять положение камер в режиме реального времени, наблюдая при этом результат быстрого но менее реалистичного синтеза изображения. Пользователь может выбирать между различными алгоритмами отрисовки и менять метод освещения.

В ходе выполнения исследовательской части было установлено, что интерактивный режим рендеринга мало зависит от количества источников освещения и синтезирует изображение в среднем в 10000 раз быстрее (при возрастании количества объектов нелинейно возрастает время, затрачиваемое на синтез изображения).

Список литературы

[1] Волков В. Я., Мясоедова Н. В. Решение позиционных задач средствами компьютерной графики // ОНВ. 2006. №2 (35). (дата обращения: 11.08.2021). https://cyberleninka.ru/article/n/reshenie-pozitsionnyh-zadach-sredstvami-kompyuternoy-grafiki.

[2] Доморацкий Е.П., Байбикова Т.Н. О формализации понятий и терминологии при анализе изображений объектов // Вестник МФЮА. 2013. №4. (дата обращения: 11.08.2021). https://cyberleninka.ru/article/n/o-formalizatsii-ponyatiy-i-terminologii-pri-analize-izobrazheniy-obektov.

[3] Colin smith, on vertex-vertex meshes and their use in geometric and biological modeling [Электронный ресурс] // algorithmic botany. (Дата обращения: 13.08.2021). http://algorithmicbotany.org/papers/smithco.dis2006.pdf.

[4] Симонов Евгений Николаевич, Аврамов Денис Витальевич, Аврамов Максим Витальевич Метод объемного рендеринга для визуализации трехмерных данных в рентгеновской компьютерной томографии // Вестник ЮУрГУ. Серия: Компьютерные технологии, управление, радиоэлектроника. 2016. №4. (дата обращения: 14.08.2021). https://cyberleninka.ru/article/n/metod-obemnogo-renderinga-dlya-vizualizatsii-trehmernyh-dannyh-v-rentgenovskoy-kompyuternoy-tomografii.

[5] Arthur Appel. *Some Techniques for Shading Machine Renderings of Solid*. SJCC, 1968, P. 27–45.

[6] Попов Андрей Сергеевич Обзор методов реалистичной визуализации и постановка задачи исследования // Радиоэлектроника

и информатика. 2006. №4. (дата обращения: 17.12.2021). https://cyberleninka.ru/article/n/obzor-metodov-realistichnoy-vizualizatsii-i-postanovka-zadachi-issledovaniya.

[7] Donald P. Greenberg Cindy M. Goral, Kenneth E. Torrance and Bennett Battaile. *Modeling the Interaction of Light Between Diffuse Surfaces*. Proceedings of ACM SIGGRAPH, 1984, P. 213–222.

[8] C++ standard. https://isocpp.org/. (дата обращения: 24.11.2021).

[9] One framework. one codebase. any platform. https://www.qt.io/. (дата обращения: 24.11.2021).

[10] Clion – кросс-платформенная ide для c и c++. https://www.jetbrains.com/ru-ru/clion/. (дата обращения: 24.11.2021).

[11] Pop!_os. https://pop.system76.com/. (дата обращения: 01.12.2021).

Приложение А

Приложение содержит UML диаграмму классов архитектурного домена.

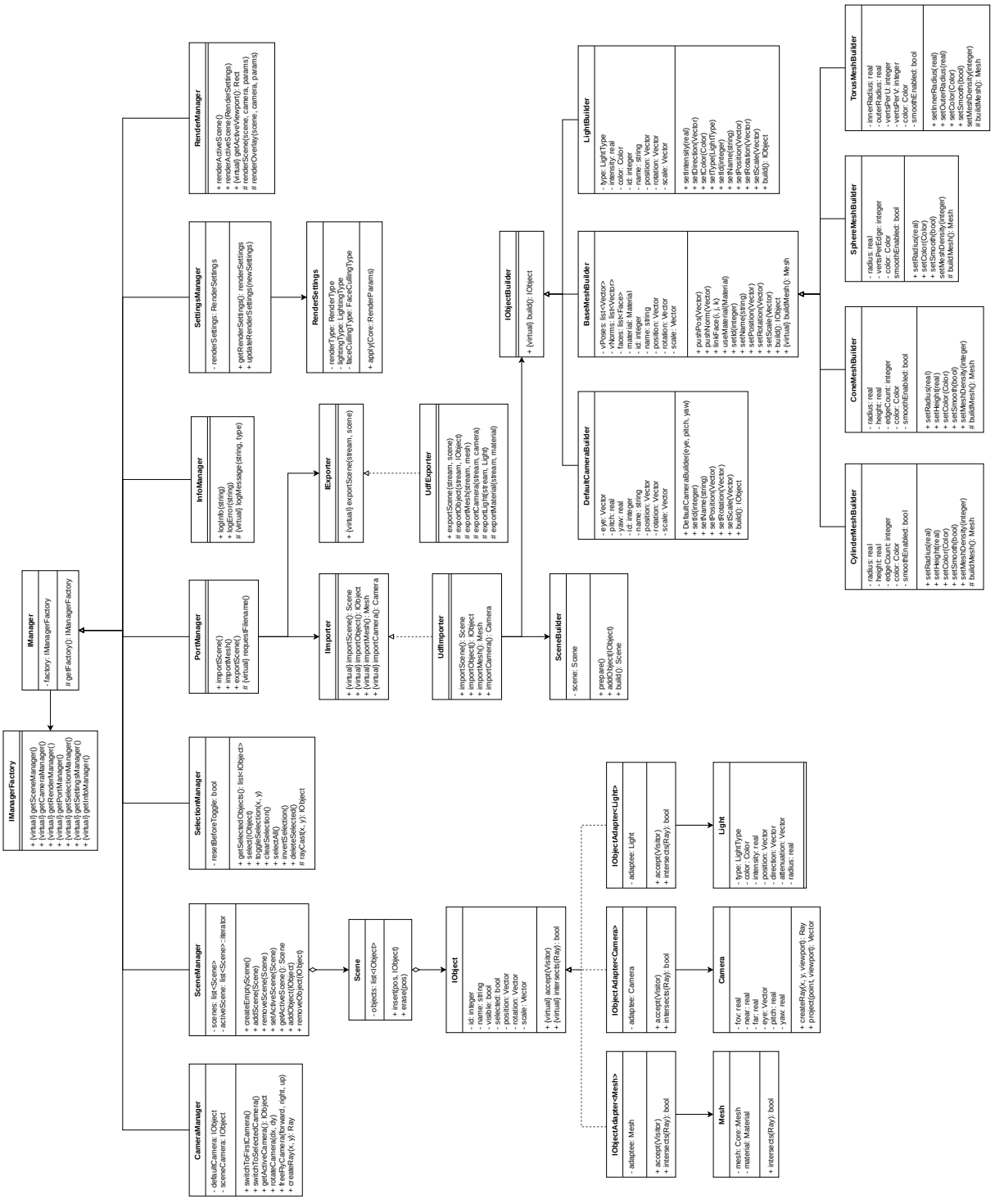

Рис. 4.2: UML диаграмма классов архитектурного домена

Приложение Б

Приложение содержит скриншоты интерфейса программного продукта.

Рис. 4.3: Интерфейс программы

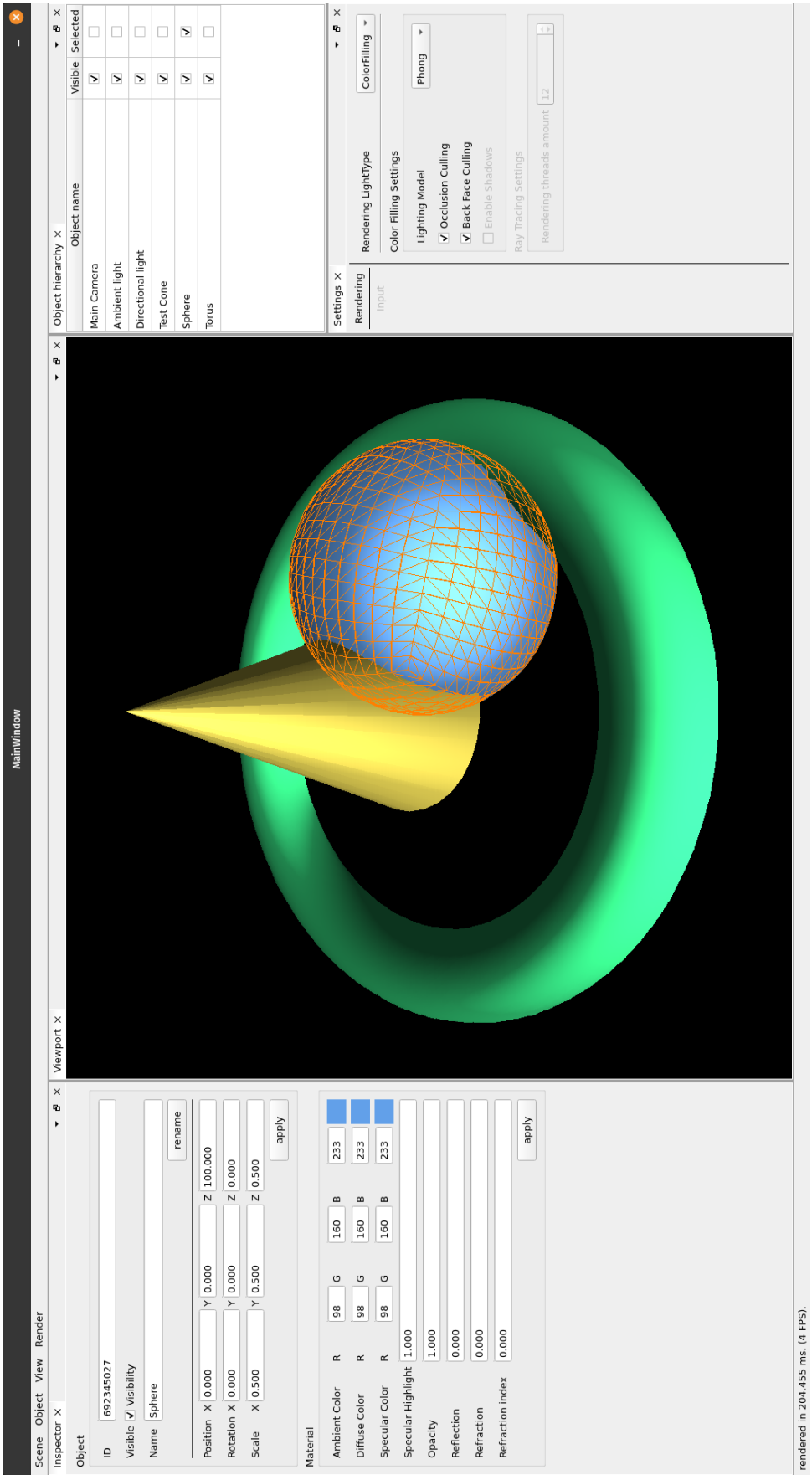

Рис. 4.4: Редактирование параметров объекта сцены

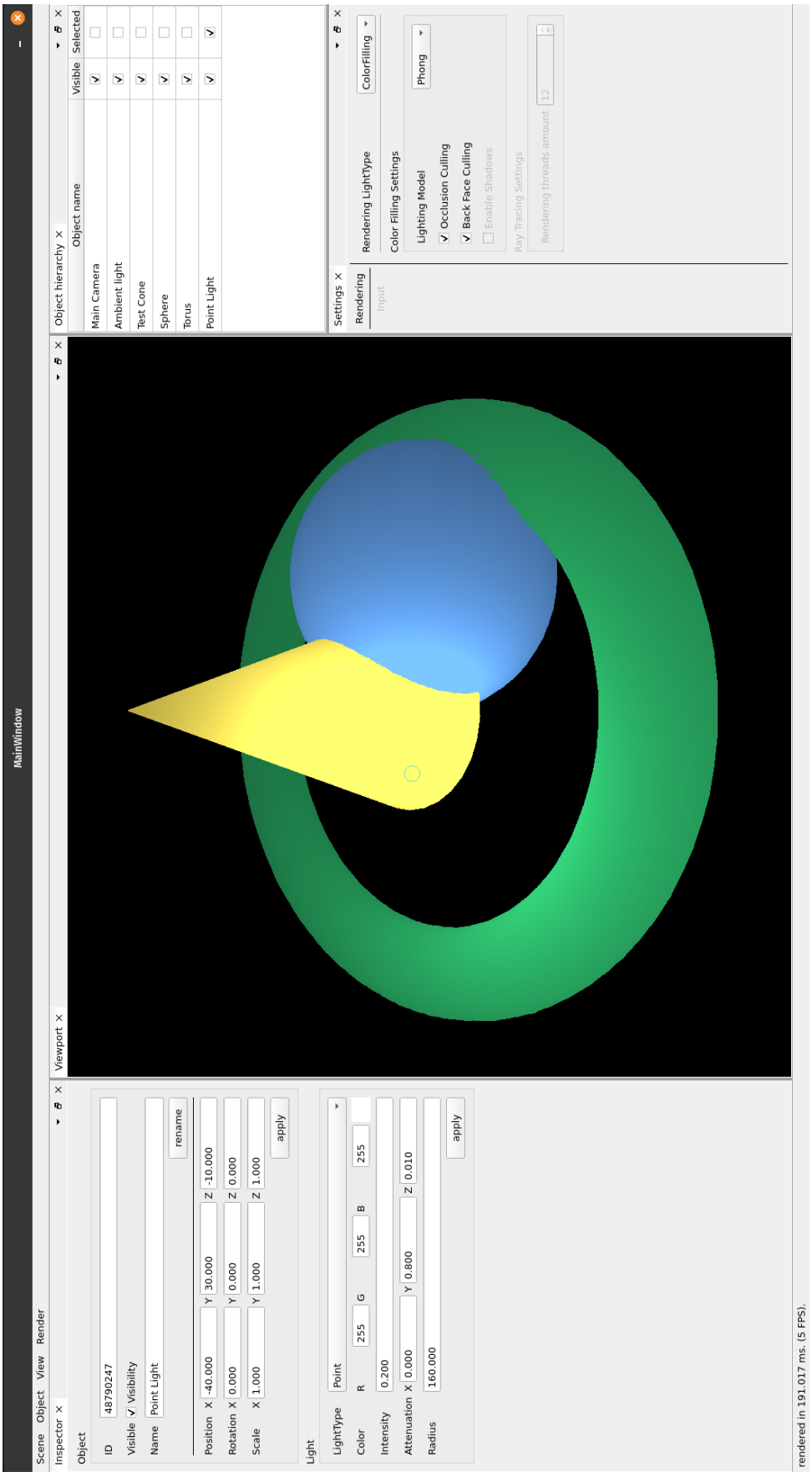

Рис. 4.5: Смена направленного освещения на точечное