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:

«Визуализация модели цветка»

Студент ИУ7-56Б (Группа) _____ (Подпись, дата) Жаворонкова А. А. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Куров А. В. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка программного обеспечения для создания реалистичного изображения цветка. Программа должна предоставлять возможность изменения положения камеры и источника освещения.

Для достижения поставленной цели требуется решить следующие задачи:

— выделить объекты сцены и выбрать модель их представления;

— проанализировать алгоритмы визуализации трехмерной сцены, при необходимости рассмотреть модификации, обосновать выбор конкретного алгоритма;

— реализовать выбранные алгоритмы;

— спроектировать архитектуру и графический интерфейс программы;

— реализовать программное обеспечение для визуализации модели цветка;

— исследовать зависимость скорости генерации кадра от шага полигональной сетки.

1 Аналитический раздел

1.1 Формализация объектов синтезируемой сцены

Сцена состоит из следующих объектов:

1. Ограничивающая плоскость — расположена параллельно плоскости OXZ;

2. Цветок — расположен на ограничивающей плоскости. В нем можно выделить следующие составляющие:

   2.1. Стеблевая часть:

   a) Цветоножка — длинный изогнутый цилиндр, описываемый следующими уравнениями:

$$\begin{cases} x = 0.6\cos t + \frac{1}{5}\sin z, \\ y = 0.6\sin t, \\ z = z. \end{cases}$$

   где $t \in [0; 2\pi), z \in [0; 20]$. Здесь z определяет высоту стебля. Выбор интервала основывался на размерах остальных частей цветка.

   b) Цветоложе — часть эллипсоида, задаваемая уравнением:

$$z = \frac{(x - \frac{1}{4})^2}{2} + \frac{y^2}{2}$$

   при $x \in [-2.25; 2.75], y \in [-2.5; 2.5]$. Интервалы для x и y были выбраны таким образом, чтобы часть эллипсоида имела характерные для цветоложа углубления.

   c) Лист — поверхность, ограниченная парой кривых, описываемых следующими уравнениями:

$$\begin{cases} x = t, \\ y = \frac{1}{3}t^2, \\ z = \frac{2}{t+1} + 3t. \end{cases}$$

где $t \in [0; 5.2]$.

Вторая кривая получается в результате поворота первой кривой на угол $\phi = \frac{-\pi}{3}$. Полученные уравнения, задающие вторую кривую:

$$\begin{cases} x = t\cos\frac{-\pi}{3} - \frac{1}{3}t^2\sin\frac{-\pi}{3}, \\ y = t\sin\frac{-\pi}{3} + \frac{1}{3}t^2\cos\frac{-\pi}{3}, \\ z = \frac{-2}{t-1} - 3t. \end{cases}$$

где $t \in [-5.2; 0]$.

Описанные выше кривые пересекаются в точках:

$$A(0; 0; 2), B(3\sqrt{3}; 9; \frac{2}{3\sqrt{3}} + 9\sqrt{3})$$

2.2. Листовая часть — совокупность поверхностей, образующих непосредственно лепестки цветка. Один лепесток описывается системой:

$$\begin{cases} z = \frac{1}{2}x^2 + y \\ x^2 + \frac{1}{5}y^2 \leq 3 \end{cases}$$

3. Источник освещения — начальное положение источника указывается по умолчанию, но пользователь может его изменить;

4. Камера — начальное положение источника указывается по умолчанию, но пользователь может его изменить.

1.2 Выбор способа определения моделей

Модели могут задаваться в следующих формах [**models**]:

— **Каркасная модель.** В данной модели задается информация о вершинах и ребрах объекта. Это одна из простейших форм задания модели. Основная проблема отображения объектов с помощью каркасной модели заключается в том, что модель не всегда однозначно передает представление о форме объекта.

— **Поверхностная модель.** Данный тип модели часто используется в компьютерной графике. Поверхность может описываться аналитически,

либо задаваться другим способом. Недостатком поверхностной модели является отсутствие информации о том, с какой стороны поверхности находится материал.

— **Твердотельная модель.** Данная форма задания модели отличается от поверхностной формы тем, что в объемных моделях к информации о поверхностях добавляется информация о том, с какой стороны расположен материал. Это можно сделать путем указания направления внутренней нормали.

При рассмотрении объектов было сказано, что они представляют собой поверхности. Поэтому для решения поставленной задачи была выбрана поверхностная форма модели.

1.3 Выбор алгоритма удаления невидимых ребер и поверхностей

Алгоритмы удаления невидимых линий и поверхностей служат для определения поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства. Решать поставленную задачу удаления можно как в объектном пространстве (в мировой системе координат), так и в пространстве изображения (в экранных координатах).

Рассмотрим алгоритмы для удаления невидимых ребер и поверхностей.

1.3.1 Алгоритм Робертса

Алгоритм работает в объектном пространстве. Требуется, чтобы все изображаемые тела были выпуклыми.

Алгоритм работает в 3 этапа: подготовка исходных данных, удаление ребер, экранируемых самим телом, удаление ребер, экранируемых другими телами.

Этап 1. Подготовка исходных данных.

Необходимо сформировать матрицы тел, которые будут представлять выпуклые твердые тела. В такой матрице каждый столбец содержит коэффициенты одной плоскости. При этом точки, лежащие внутри тела, дают положительное скалярное произведение с каждым столбцом матрицы.

Этап 2. Удаление ребер, экранируемых самим телом.

На данном этапе используется вектор направления взгляда:

$$E = [0; 0; -1; 0]$$

При умножении вектора E на матрицу тела отрицательные компоненты полученного вектора будут соответствовать задним граням. Если объект на сцене один, то работа алгоритма завершается на данном этапе.

Этап 3. Удаление ребер, экранируемых другими телами.

Необходимо провести луч из произвольной точки анализируемого отрезка в точку наблюдения. Если луч проходит через тело, то точка невидима, а луч расположен с положительной стороны от каждой грани тела.

Преимущества [1]:

— Точность вычислений благодаря тому, что алгоритм работает в объектном пространстве;

— Использование математически простых и точных методов.

Недостатки [1]:

— Возможность работать только с выпуклыми объектами;

— Сложность алгоритма $O(n^2)$, где n — количество объектов сцены.

1.3.2 Алгоритм, использующий z-буфер

Алгоритм работает в пространстве изображения. Основная идея: поиск по x и y наибольшего значения функции $z(x, y)$ [2].

Используются два буфера:

— Буфер кадра, используемый для запоминания интенсивности каждого пикселя;

— Z-буфер — буфер глубины, используемый для запоминания координаты z (глубины каждого видимого пикселя).

В начале работы алгоритма буфер кадра заполнен фоновым значением интенсивности или цвета, а z-буфер – минимальным значением координаты z. Также удаляются нелицевые грани, если это целесообразно.

Затем каждый многоугольник преобразовывается в растровую форму в произвольном порядке. Для каждого пикселя в многоугольнике вычисляется его глубина и записывается в z-буфер, если она больше хранящегося значения.

Преимущества:

— Алгоритм делает тривиальной визуализацию пересечений сложных поверхностей;

— Сцены могут быть любой сложности;

— Сложность алгоритма $O(n)$, где n — количество объектов сцены;

— Экономия вычислительного времени, так как элементы сцены не сортируются.

Недостатки:

— Большой объем требуемой памяти;

— Трудоемкость устранения лестничного эффекта.

1.3.3 Алгоритм обратной трассировки лучей

Алгоритм работает в пространстве изображения.

Наблюдатель видит объект благодаря испускаемому неким источником свету, который падает на этот объект и каким-либо образом доходит до наблюдателя: отразившись от поверхности, преломившись или пройдя через нее. Так как немногие из лучей, выпущенных источником, доходят до наблюдателя, то целесообразно трассировать (отслеживать) лучи в обратном направлении – от наблюдателя к объекту [2].

Предполагается, что сцена уже преобразована в пространство изображения. Каждый луч, исходящий от наблюдателя, проходит через центр пикселя на растре до сцены. Траектория каждого луча отслеживается, чтобы определить, какие именно объекты сцены, пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта сцены с каждым лучом. Если луч пересекает объект, то определяются все возможные точки пересечения луча и объекта. Можно получить большое количество пересечений, если рассматривать много объектов. Эти пересечения упорядочиваются по глубине. Пересечение с максимальным значением координаты z представляет видимую

поверхность для данного пикселя. Атрибуты этого объекта используются для определения характеристик пикселя.

Если точка зрения находится не в бесконечности, предполагается, что наблюдатель по-прежнему находится на положительной полуоси z. Картинная плоскость, перпендикулярна оси z. Задача состоит в построении одноточечной центральной проекции на картинную плоскость.

Преимущества:

— Высокая реалистичность получаемого изображения;

— Простота модификации при работе с несколькими источниками освещения, реализации различных оптических явлений;

— Алгоритм не требует дополнительных вычислений для нахождения теней.

Недостатки:

— Большая трудоемкость вычислений.

1.3.4 Алгоритм Варнока

Алгоритм работает в пространстве изображения.

В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое подокна не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения. В последнем случае информация, содержащаяся в окне, усредняется, и результат изображается с одинаковой интенсивностью или цветом [2].

Устранение лестничного эффекта можно реализовать, доведя процесс разбиения до размеров, меньших, чем разрешение экрана на один пиксель, и усредняя атрибуты подпикселей, чтобы определить атрибуты самих пикселей.

Преимущества:

— Эффективность для простых сцен;

— Простота устранения лестничного эффекта.

Недостатки:

— Неэффективность при большом количестве объектов.

Вывод

Сравнение описанных выше алгоритмов представлено таблицей 1.1.

Таблица 1.1 – Результаты замеров времени для произвольных массивов

	Алгоритм Робертса	Алгоритм, использующий z-буфер	Алгоритм обратной трассировки лучей	Алгоритм Варнока
Сложность алгоритма (N — количество граней, C — количество пикселей)	$O(N^2)$	$O(CN)$	$O(CN)$	$O(CN)$
Эффективность для сцен с большим количеством объектов	Низкая	Высокая	Низкая	Средняя
Пространство работы алгоритма	Объектное пространство	Пространство изображений	Пространство изображений	Пространство изображений
Сложность реализации	Высокая	Низкая	Средняя	Средняя

Таким образом, для решения данной задачи был выбран алгоритм, использующий z-буфер, так как он отвечает заявленным требованиям при постановке задачи.

1.4 Выбор алгоритма построения теней

Поскольку в качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм, использующий z-буфер, для построения теней будет использована его модификация [3].

Строится сцена из точки наблюдения, совпадающей с источником. Зна-

чения z для этого вида хранятся в отдельном теневом z-буфере. Значения интенсивности не рассматриваются.

Затем сцена строится из точки, в которой находится наблюдатель. При обработке каждой поверхности или многоугольника его глубина в каждом пикселе сравнивается с глубиной в z-буфере наблюдателя. Если поверхность видима, то значения (x, y, z) из вида наблюдателя линейно преобразуются в значения (x', y', z') на виде из источника. Для того чтобы проверить, видимо ли значение z' из положения источника, оно сравнивается со значением теневого z-буфера при x', y'. Если оно видимо, то оно отображается в буфер кадра в точке x, y без изменений. Если нет, то точка находится в тени и изображается согласно соответствующему правилу расчета интенсивности с учетом затенения, а значение в z-буфере наблюдателя заменяется на z' [4][5].

1.5 Анализ методов закрашивания

Существует несколько методов закрашивания. Рассмотрим некоторые из них.

1.5.1 Простая закраска

Вся грань закрашивается одним уровнем интенсивности. Используется минимальное количество вычислений, но снижается качество получаемого изображения [6].

Используется при выполнении трех условий:

1. Предполагается, что источник находится в бесконечности;
2. Предполагается, что наблюдатель находится в бесконечности;
3. Закрашиваемая грань является реально существующей, а не полученной в результате аппроксимации поверхности.

Недостатком является возникновение ребер. При закраске каждой грани со своей интенсивностью граница между ними становится видна, и возникают ребра [7].

1.5.2 Закраска по Гуро

Закраска по Гуро выполняет сглаживание на основе биполярной интерполяции интенсивности [8].

Вводится понятие нормали к вершине, на основе которой вычисляется интенсивность каждой вершины и выполняется первая интерполяция вдоль ребер. Вторая интерполяция выполняется при вычислении интенсивности пикселей, расположенных на сканирующей строке. Качество изображения улучшится. Граница между двумя гранями визуально сгладится.

Закраска по Гуро не предусматривает учет кривизны поверхности. При применении закраски по Гуро возможно получение плоского изображения, когда углы, образованные гранями, одинаковые.

Закраска по Гуро хорошо сочетается с диффузной составляющей поверхности (матовой).

1.5.3 Закраска по Фонгу

Основная идея закраски по Фонгу: интерполировать нормали, а не интенсивности, как в закраске по Гуро.

От точки к точке в пределах грани нормали изменяются, учитывается криволинейный характер поверхности. Изображение получается более качественное, но трудоёмкость закраски по Фонгу будет выше.

Закраска по Фонгу хорошо сочетается с зеркальной составляющей: моделирует блики, возникающие при зеркальном отражении [9].

Вывод

Результаты сравнения алгоритмов закраски представлены таблицей 1.2

Таблица 1.2 – Сравнение алгоритмов закраски

	Простая закраска	Закраска по Гуро	Закраска по Фонгу
Реалистичность получаемого изображения	Низкая	Средняя	Высокая
Эффективность для сцен с большим количеством объектов	Высокая	Средняя	Низкая
Сочетаемость с диффузной составляющей поверхности	Нет	Да	Нет

Для данной задачи был выбран алгоритм закраски по Гуро, так как он отвечает заявленным требованиям при постановке задачи.

Вывод

В данном разделе были формализованы объекты синтезируемой сцены, проведен обзор предметной области: рассмотрены существующие методы удаления невидимых линий и поверхностей, методы закрашивания и методы удаления теней. Из рассмотренных методов были выбраны алгоритмы для решения поставленной задачи.

В качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм, использующий Z-буфер. В качестве алгоритма закрашивания была выбрана закраска по Гуро. В качестве алгоритма построения теней была выбрана модификация алгоритма, использующего Z-буфер.

2 Конструкторский раздел

2.1 Общий алгоритм решения поставленной задачи

Структура программы представлена на рисунке 2.1 в виде IDF0-диаграммы.

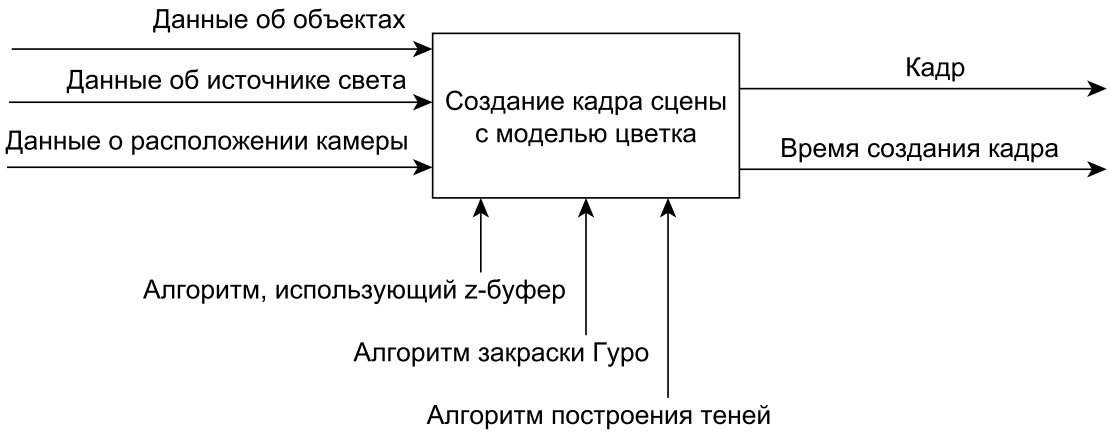

Рисунок 2.1 – Структура программы

В соответствии с рисунком 2.1 входными данными для разрабатываемой программы будут являться данные об объектах, об источнике света, о расположении камеры. В программе будут реализованы следующие алгоритмы: модифицированный алгоритм, использующий z-буфер, алгоритм закраски Гуро. В результате работы программы будет получен кадр сцены, содержащий модель цветка, а также время создания одного кадра.

2.2 Алгоритм, использующий z-буфер

Схема алгоритма, использующего z-буфер, представлена на рисунке 2.2

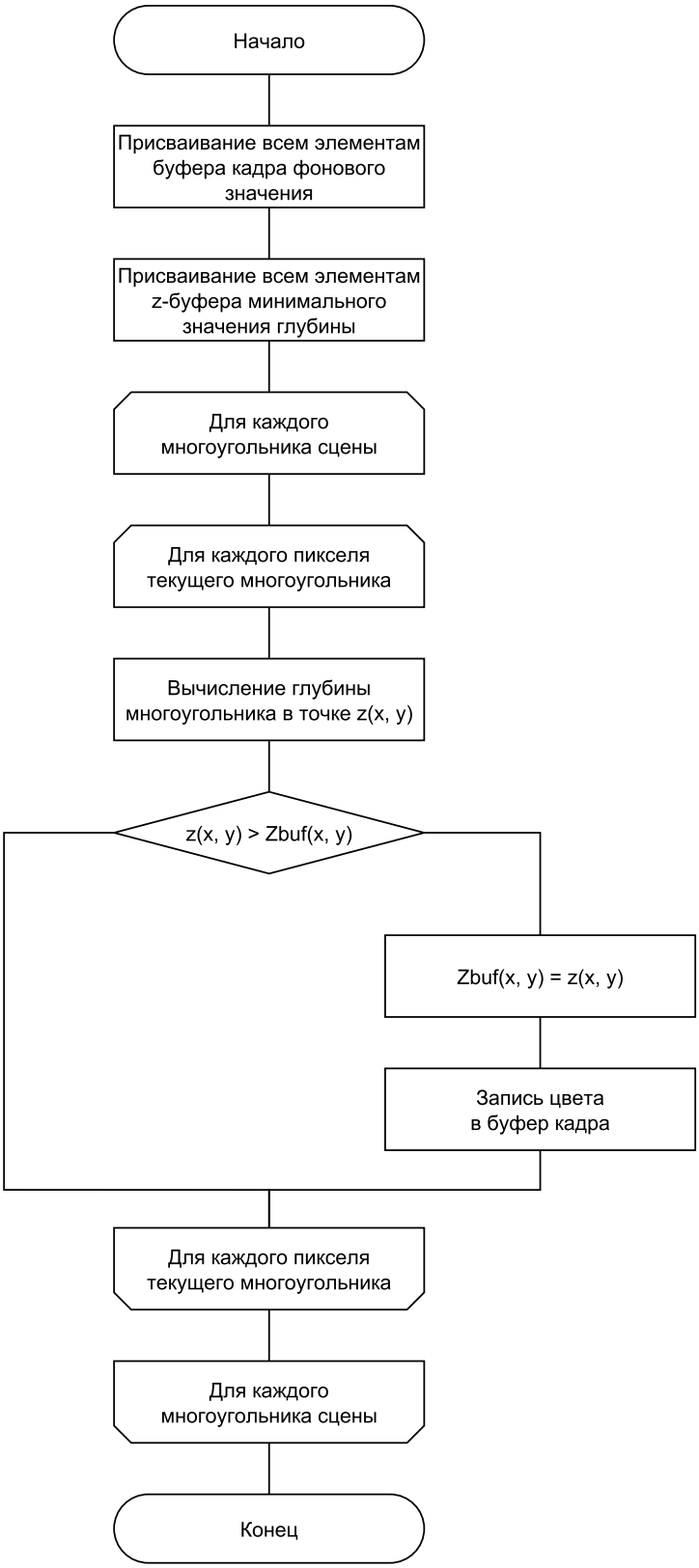

Рисунок 2.2 – Схема алгоритма, использующего z-буфер

2.3 Модифицированный алгоритм, использующий z-буфер

Схема модифицированного алгоритма, использующего z-буфер, представлена на рисунках 2.3–2.4

Рисунок 2.3 – Схема модифицированного алгоритма, использующего z-буфер (часть 1)

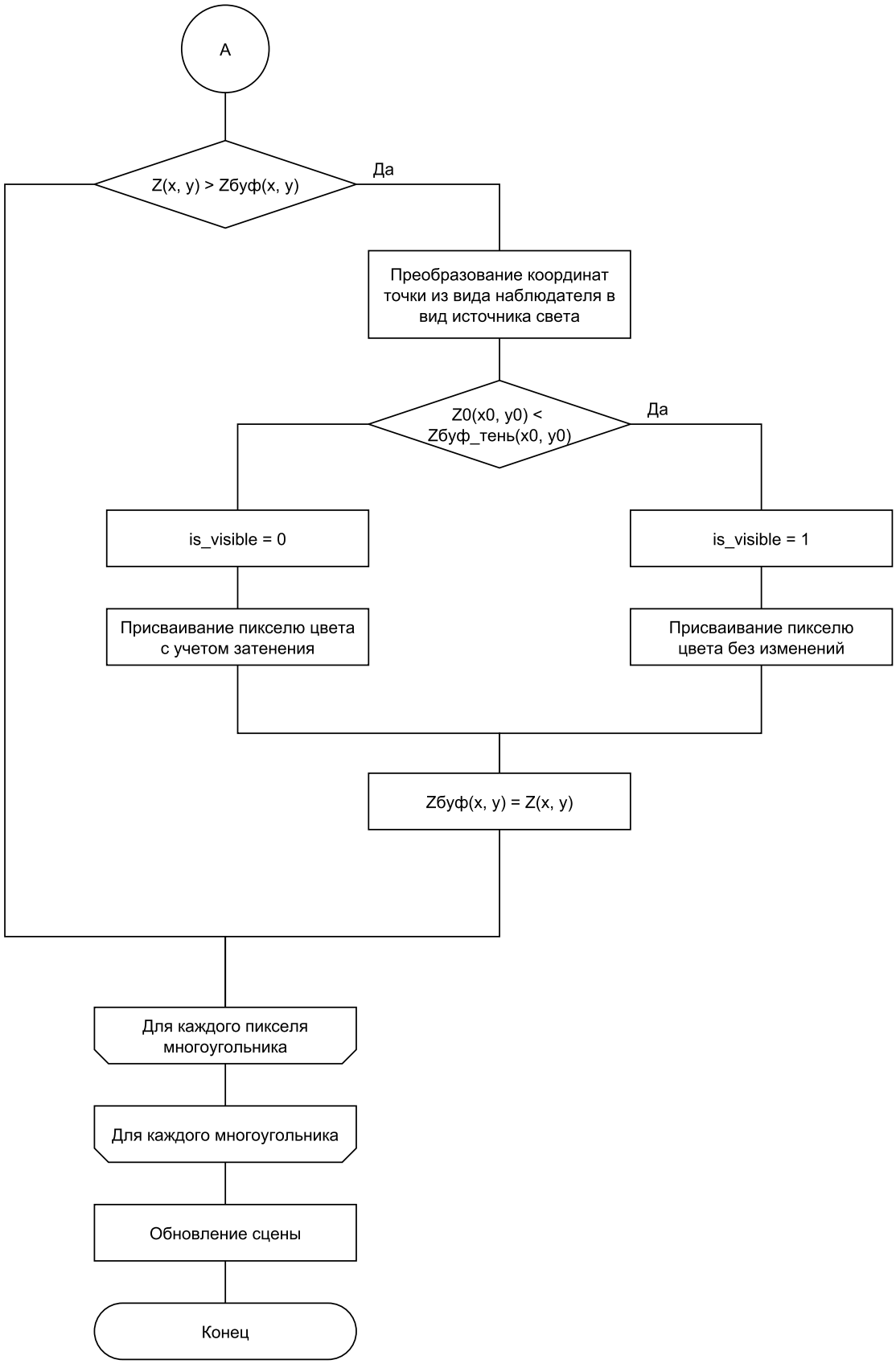

Рисунок 2.4 – Схема модифицированного алгоритма, использующего z-буфер (часть 2)

2.4 Алгоритм закраски Гуро

В алгоритме закраски Гуро сначала определяется интенсивность в вершинах, потом вдоль ребер вычисляется интенсивность соответствующего пикселя. Схема алгоритма представлена на рисунке 2.5.

Рисунок 2.5 – Схема алгоритма закраски Гуро

Этот метод хорошо сочетается с алгоритмом, использующим z-буфер. Для каждой сканирующей строки определяются ее точки пересечения с ребрами. В этих точках интенсивность вычисляется с помощью линейной интерполяции интенсивностей в вершинах ребра. Затем для всех пикселей, находящихся внутри многоугольника и лежащих на сканирующей строке, аналогично вычисляется интенсивность.

2.5 Схема алгоритма генерации одного кадра изображения

На рисунках 2.6–2.7 представлена схема алгоритма генерации одного кадра изображения, объединяющего в себе модифицированный алгоритм Z-буфера и алгоритм закраски по Гуро.

Рисунок 2.6 – Схема алгоритма генерации одного кадра изображения (часть 1)

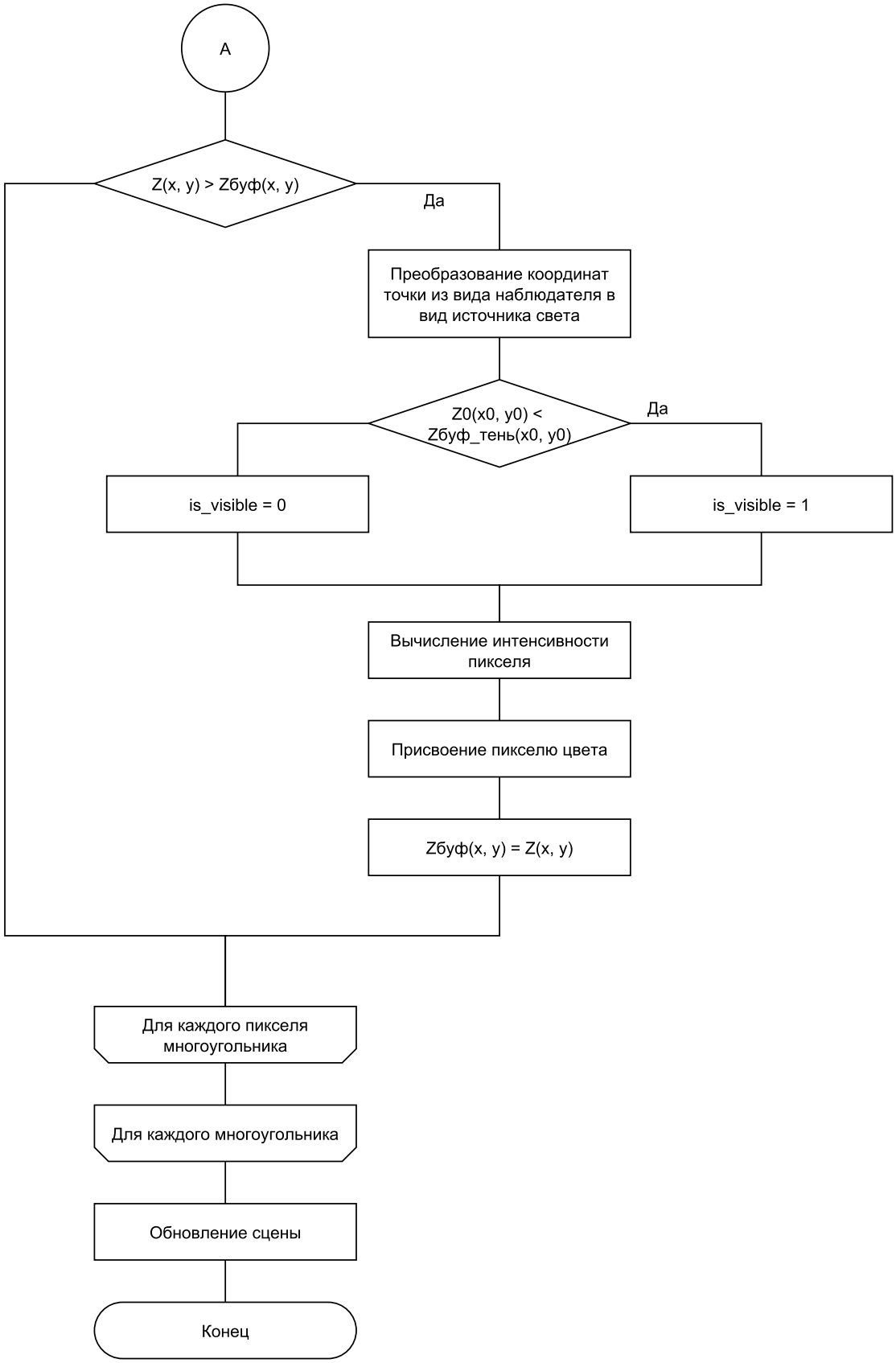

Рисунок 2.7 – Схема алгоритма генерации одного кадра изображения (часть 2)

2.6 Описание используемых структур данных

На рисунке 2.8 представлены классы основных объектов сцены, и показаны их атрибуты.

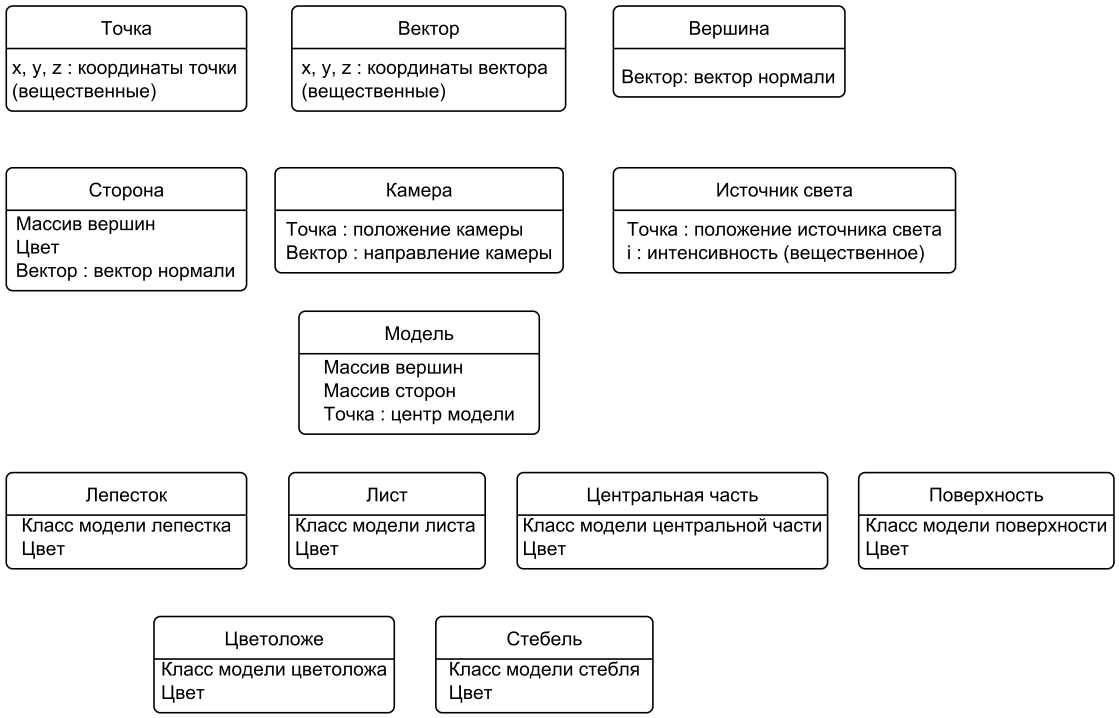

Рисунок 2.8 – Классы основных объектов сцены

Основные использованные типы и структуры данных:

— тип *double* для координат точки, вектора, а также для интенсивности источника;

— класс *Вектор* для вектора нормали, направления камеры;

— массив объектов класса *Вершина* для использования в классах *Сторона* и *Модель*;

— класс *Точка* для положения камеры, источника света, а также для центра модели;

— массив объектов класса *Сторона* для использования в классе *Модель*;

— тип *Qrgb* для цвета.

Вывод

В данном разделе был представлен общий алгоритм решения поставленной задачи в виде диаграммы IDF0 0 уровня, схемы алгоритмов использующего z-буфер, его модификации для построения теней и закраски Гуро. Также был описан алгоритм генерации одного кадра изображения.

3 Технологический раздел

3.1 Средства реализации

В качестве языка для разработки программы был выбран язык программирования C++. Данный выбор основан на следующих аспектах:

— В стандартной библиотеке языка присутствует поддержка всех структур данных, выбранных по результатам проектирования;

— Средствами языка можно реализовать все алгоритмы, выбранные в результате проектирования;

— C++ обладает высокой вычислительной производительностью, что очень важно для выполнения поставленной задачи;

— Статическая типизация позволит устранять ошибки на стадии компиляции;

— Доступность учебной литературы.

В качестве среды разработки был выбран QtCreator. Данный выбор обусловлен следующими факторами:

— Данная среда разработки предоставляет удобную графическую библиотеку;

— Позволяет работать с графическим интерфейсом;

— Является бесплатной.

3.2 Разработка используемых классов

На рисунке А.1 представлена схема взаимодействия основных объектов сцены и показаны их составляющие.

Используются следующие основные классы:

— Point — класс точки в трехмерном пространстве;

— Vector — класс вектора в трехмерном пространстве;

— Vertex — класс вершины;

— Side — класс грани;

— Model — класс модели;

— Camera — класс камеры;

— LightSource — класс источника света;

— Receptacle — класс цветоложа цветка;

— Leaf — класс листа цветка;

— Petal — класс лепестка цветка;

— Stem — класс стебля цветка;

— Center — класс центральной части цветка;

— Surface — класс ограничивающей поверхности.

3.3 Разработка интерфейса

В связи с тем, что у пользователя должна быть возможность перемещать камеру и источник света, в интерфейсе необходимы кнопки, при нажатии на которые будет происходить соответствующее движение. Для камеры необходимы кнопки для перемещения по всем трем осям, а также кнопки для вращения вокруг них. Для источника света достаточно только кнопок для перемещения, поскольку в данном случае сцена не изменится от его поворота.

Для перемещения вдоль оси x в положительном и отрицательном направлениях предусмотрены кнопки «Вправо» и «Влево» соответственно, вдоль y — «Вверх», «Вниз», вдоль z — «Ближе», «Дальше». Для поворота вокруг оси x на положительный и отрицательный угол предусмотрены кнопки «Вниз» и «Вверх» соответственно, вокруг y — «Вправо», «Влево».

В результате пользователю предоставляется интерфейс, показанный на рисунке 3.1. На панели справа расположены кнопки для перемещения и вращения камеры, перемещения источника света. В нижней части находятся кнопки, позволяющие сменить время суток: день, ночь.

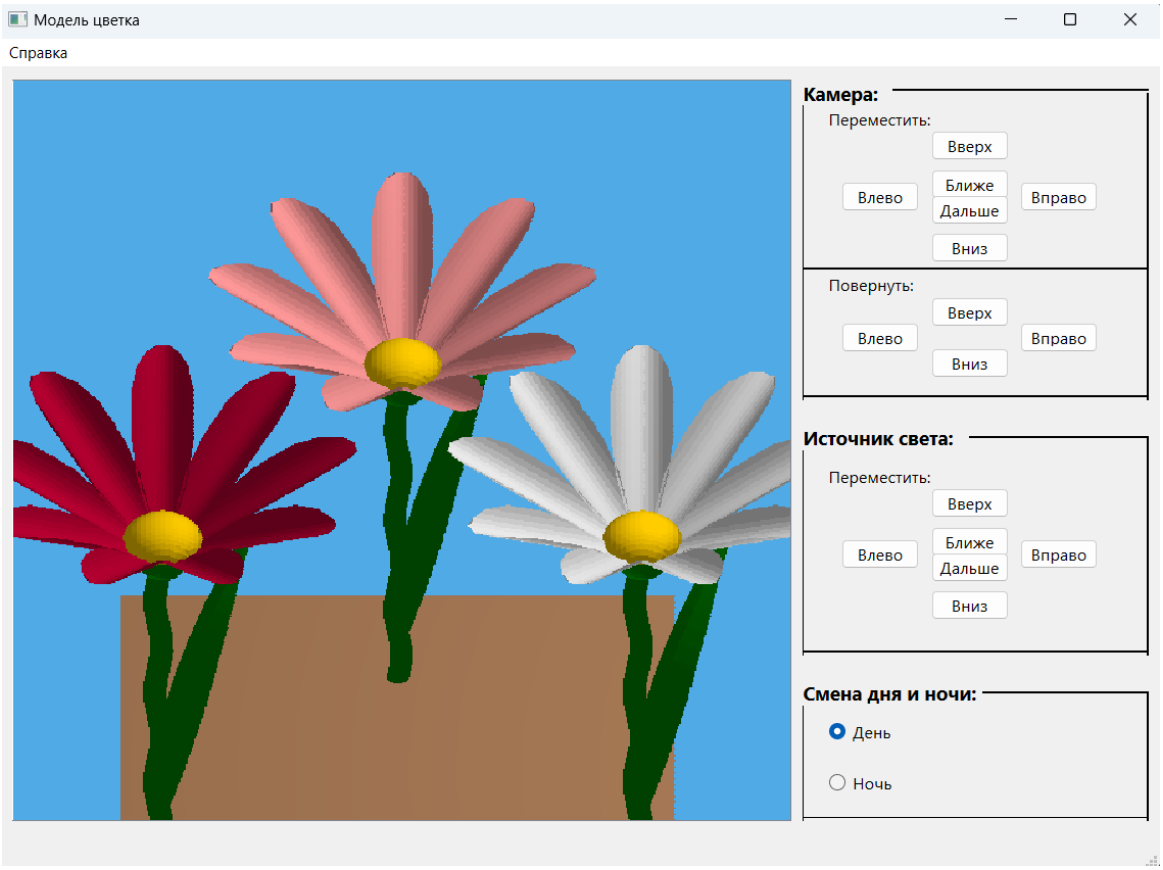

Рисунок 3.1 – Интерфейс программного обеспечения

При нажатии на кнопку «Ночь» лепестки модели цветка начнут вращаться к центру, имитируя закрытие цветка, начнет затемняться фон, а также снижаться интенсивность источника. При нажатии на кнопку «День» будут происходить обратные действия: раскрытие лепестков, фон будет становиться светлее, увеличение интенсивности источника. Нажатие на кнопки «День» и «Ночь» возможно, если модель цветка не находится в процессе смены времени суток.

В верхнем левом углу расположено выпадающее меню, в котором находятся кнопки «Горячие клавиши» и «О программе». При нажатии на кнопку «Горячие клавиши» пользователю будет предоставлена таблица, связывающая клавиши клавиатуры и кнопки интерфейса (Рисунок 3.2).

Рисунок 3.2 – Горячие клавиши программного обеспечения

Вывод

В данном разделе было разработано программное обеспечение для визуализации модели цветка. Был выбран язык программирования и среда разработки, а также описан интерфейс, предоставляемый пользователю.

4 Исследовательский раздел

В связи с тем, что все объекты сцены задаются с использованием полигональной сетки, возникает вопрос о том, как зависит время генерации одного кадра изображения от величины шага полигональной сетки.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование представлены далее:

— операционная система: Windows 11, x64;

— оперативная память: 8 Гб;

— процессор: AMD Ryzen 5 5500U с видеокартой Radeon Graphics 2.10 ГГц.

Во время исследования ноутбук был нагружен только встроенными приложениями окружения.

4.2 Цель исследования

Целью исследования является определение зависимости скорости генерации одного кадра изображения от шага полигональной сетки.

Исследование проводится при смене времени суток с значения «День» на «Ночь». В качестве результирующего значения времени генерации одного кадра изображения берется среднее значение. Шаг полигональной сетки принимает следующие значения: $\{0.025, 0.050, 0.100, 0.250, 0.500, 0.750\}$.

4.3 Результаты исследования

Результаты исследования представлены в таблице 4.1.

Таблица 4.1 – Результаты замеров времени

Шаг сетки	Время генерации одного кадра, мс
0.025	50 585
0.050	5 468
0.100	1 964
0.250	259
0.500	119
0.750	103

На рисунке 4.1 приведен график зависимости времени генерации одного кадра изображения от шага полигональной сетки.

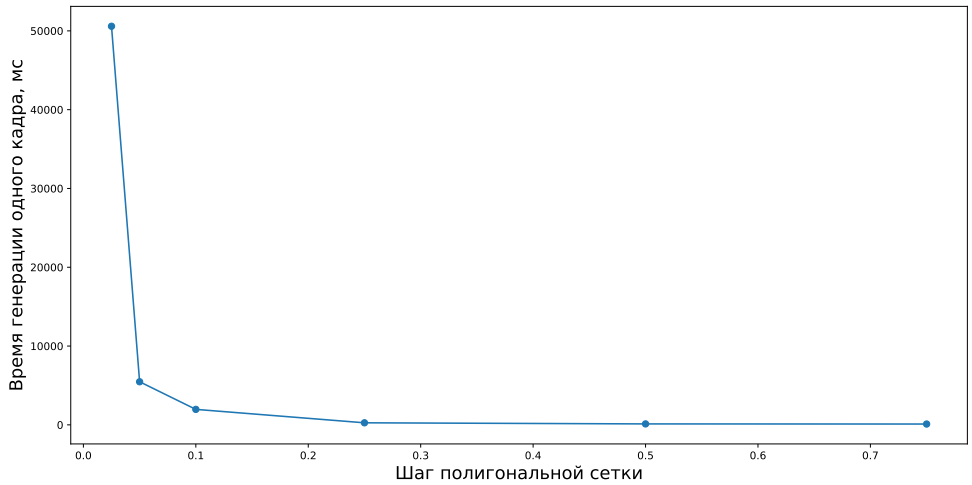

Рисунок 4.1 – Визуализация результатов исследования

На рисунках 4.2 – 4.4 представлены получаемые изображения при разном шаге полигональной сетки: 0.025, 0.250 и 0.750 соответственно.

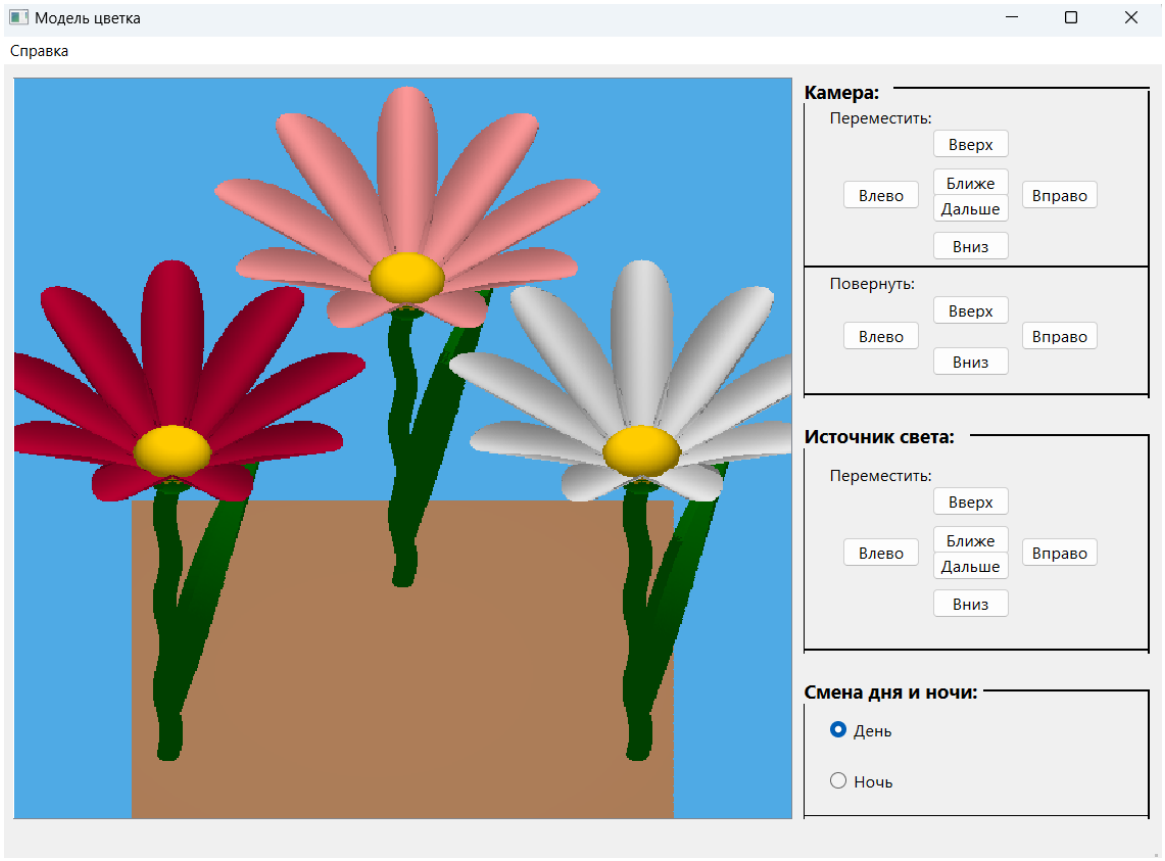

Рисунок 4.2 – Изображение, получаемое при шаге полигональной сетки 0.025

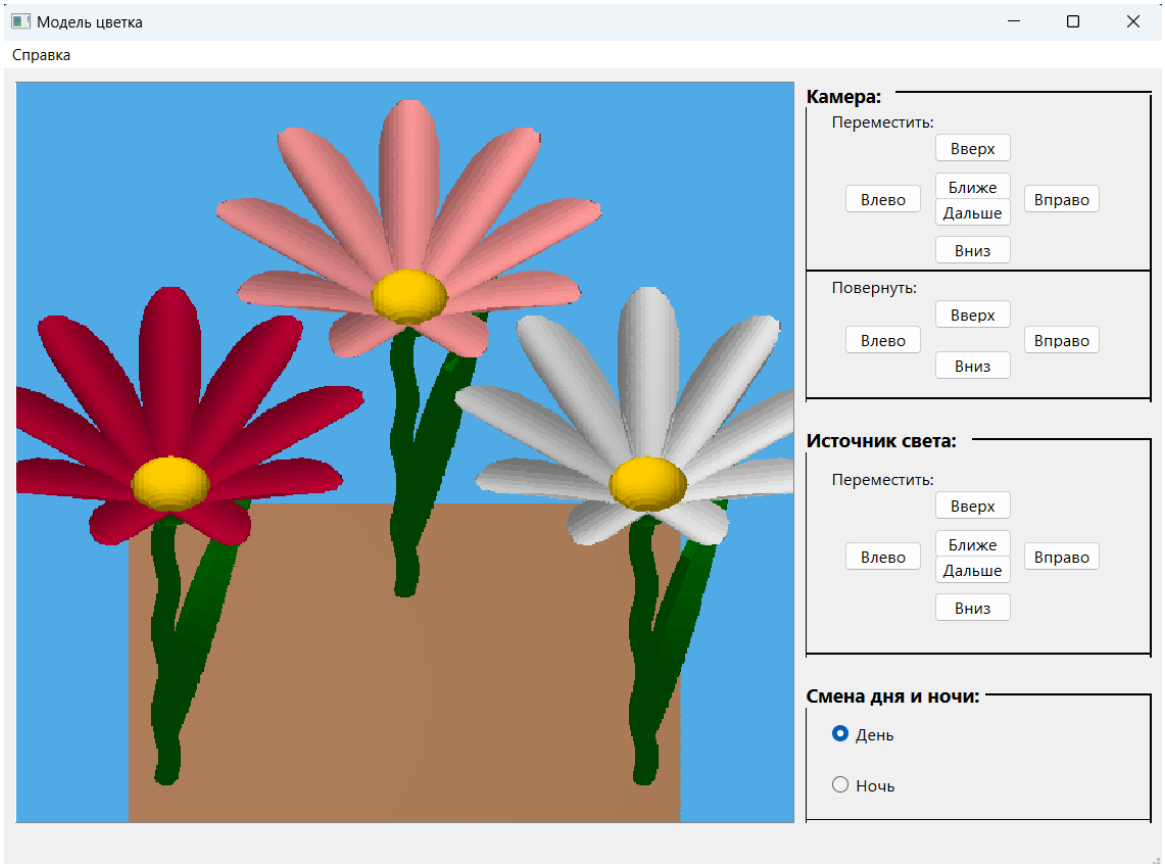

Рисунок 4.3 – Изображение, получаемое при шаге полигональной сетки 0.250

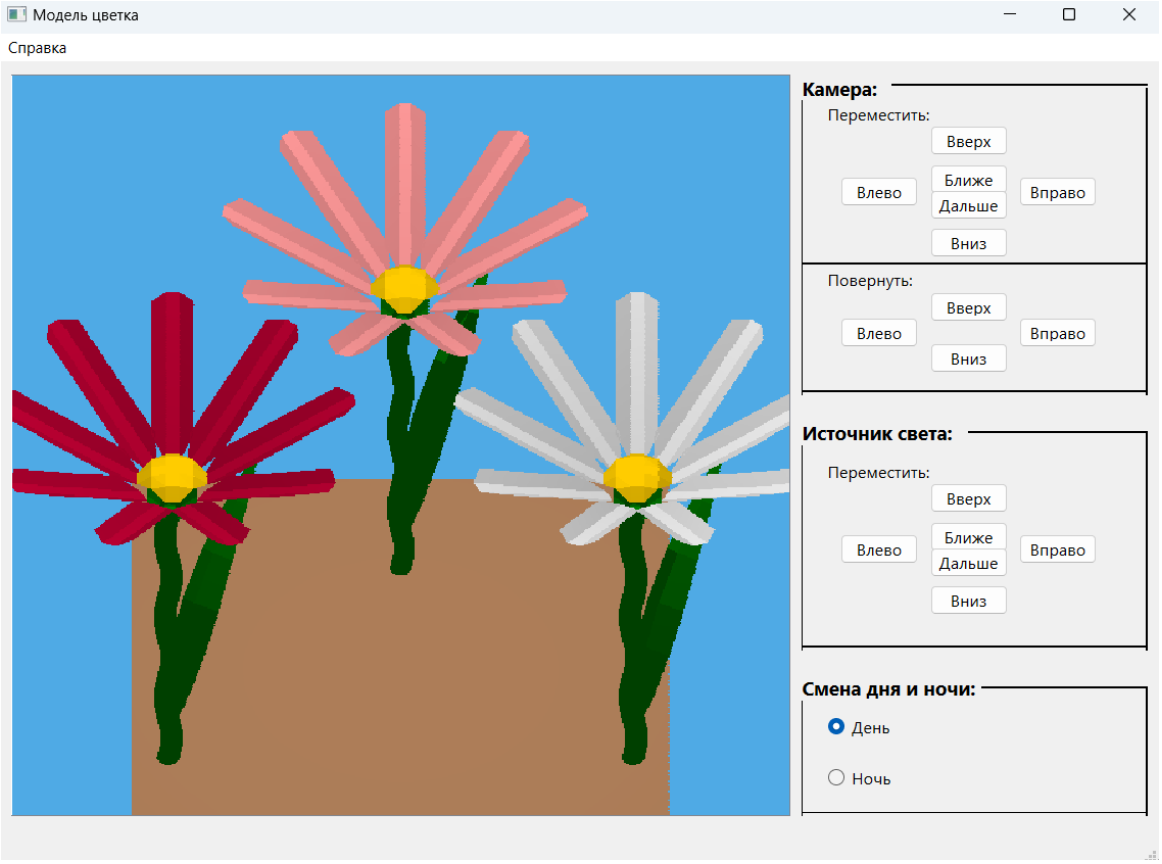

Рисунок 4.4 – Изображение, получаемое при шаге полигональной сетки 0.750

Вывод

На основании результатов исследования можно сделать вывод, что при уменьшении шага полигональной сетки возрастает время генерации одного кадра изображения. Так например, при уменьшении шага с 0.250 до 0.100 время генерации возросло в ≈ 7.6 раза, а при уменьшении с 0.100 до 0.050 — в ≈ 2.8 раза. Сравнивая рисунки 4.2 – 4.4 можно заметить, что при увеличении шага полигональной сетки лепестки приобретают более ровную, прямую форму, центральная часть становится менее круглой, появляются грани. Кроме того, снижается плавность смены оттенков при переходе от тени к свету (наиболее заметно на центральной части цветка). Все эти факторы влияют на реалистичность изображения: при увеличении шага полигональной сетки снижается реалистичность получаемого изображения.

Таким образом, оптимальным значением шага полигональной сетки является 0.250, поскольку такой шаг обеспечивает достаточную реалистичность и время генерации одного кадра изображения.

ЗАКЛЮЧЕНИЕ

Цель, которая была поставлена в начале курсовой работы была достигнута: разработано программное обеспечение для создания реалистичного изображения цветка.

В ходе выполнения были решены все задачи:

— выделены объекты сцены и выбрана модель их представления;

— проанализированы алгоритмы визуализации трехмерной сцены, рассмотрены модификации, обоснован выбор конкретного алгоритма;

— спроектирована архитектура и графический интерфейс программы;

— реализованы выбранные ранее алгоритмы;

— реализовано программное обеспечение для визуализации модели цветка;

— исследована зависимость скорости генерации кадра от шага полигональной сетки.

В результате исследования был выбран шаг полигональной сетки, обеспечивающий достаточную реалистичность и время генерации одного кадра изображения — 0.250.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Роджерс Д.* Алгоритмические основы машинной графики: Пер. с англ. — М.: Мир, 1989.
2. *Куров А. В.* Курс лекций по дисциплине «Компьютерная графика». — 2023.
3. *Польский С. В.* Компьютерная графика: учебн.-методич. Пособие. — М.: ГОУ ВПО МГУЛ, 2008.
4. Тени. Модификация алгоритма с z - буфером. [Электронный ресурс]. — URL: `https://studfile.net/preview/8656499/page:5/` (дата обращения: 18.10.2023).
5. *Романюк А. Н.*, *Куринный М. В.* АЛГОРИТМЫ ПОСТРОЕНИЯ ТЕНЕЙ. — 2000.
6. Модели закраски [Электронный ресурс]. — URL: `https://studfile.net/preview/6010005/page:40/` (дата обращения: 14.10.2023).
7. Алгоритмы закраски [Электронный ресурс]. — URL: `https://studbooks.net/2248060/informatika/odnotonnaya_zakraska_metod_graneniya` (дата обращения: 14.10.2023).
8. Алгоритмы закраски [Электронный ресурс]. — URL: `https://studbooks.net/2248060/informatika/odnotonnaya_zakraska_metod_graneniya` (дата обращения: 14.10.2023).
9. НОУ ИНТУИТ | Алгоритмические основы современной компьютерной графики. Лекция 9: Закрашивание. Рендеринг полигональных моделей [Электронный ресурс]. — URL: `https://intuit.ru/studies/courses/70/70/lecture/2108?page=2` (дата обращения: 18.10.2023).

ПРИЛОЖЕНИЕ А

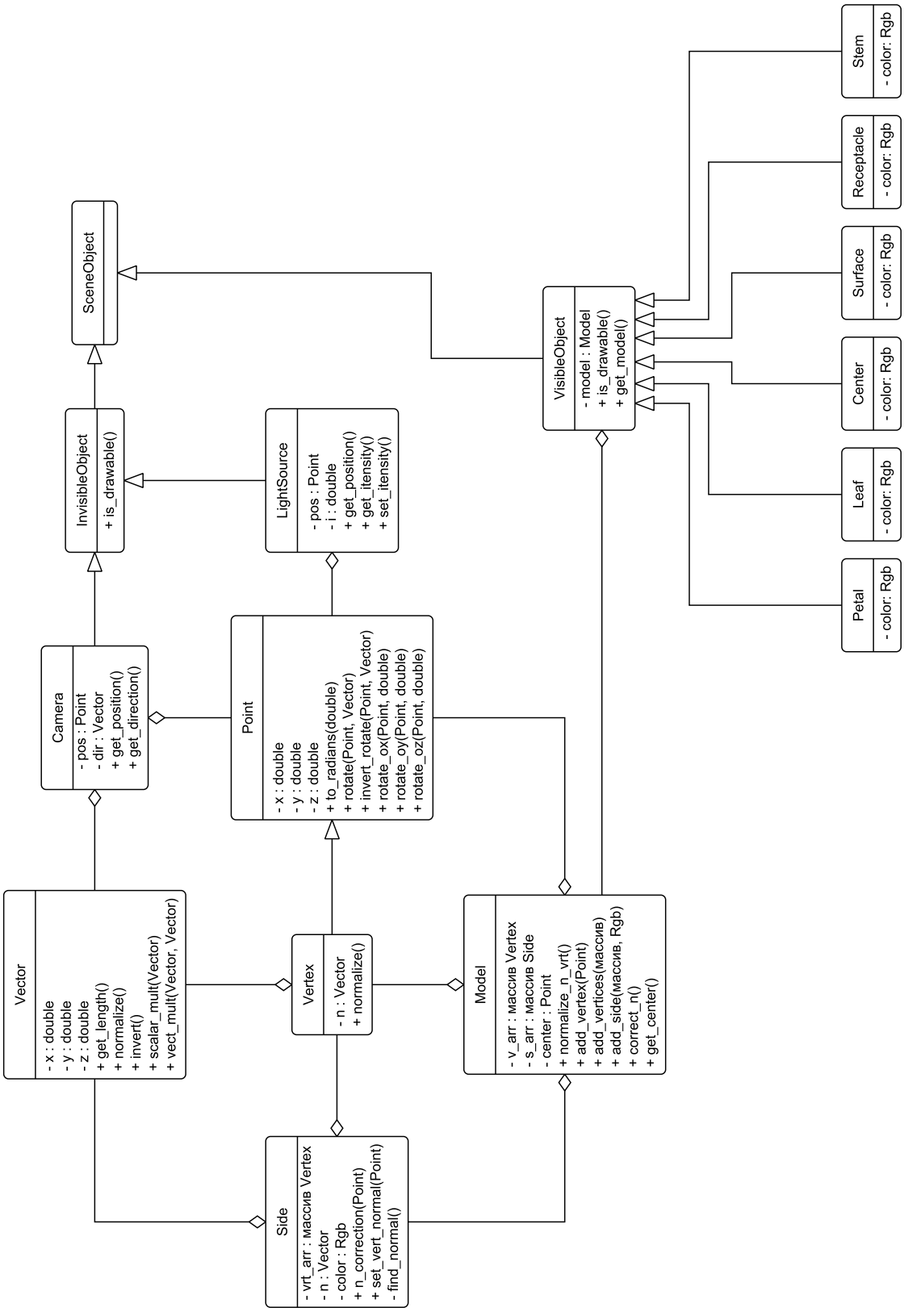

Рисунок А.1 – Схема взаимодействия основных объектов сцены

ПРИЛОЖЕНИЕ Б

Ниже приведены реализации классов основных объектов сцены.

Рисунок Б.1 – Реализация класса точки

```
1  class Point
2  {
3  public:
4      double x, y, z;
5
6      Point();
7      Point(double x, double y, double z);
8      Point(const Point& other);
9
10     virtual ~Point();
11
12     void operator =(const Point& other);
13
14     double to_radians(double angle);
15     void rotate(const Point& center, const Vector& angles);
16     void invert_rotate(const Point &center, const Vector &angles);
17
18     void rotate_ox(const Point& center, double k);
19     void rotate_oy(const Point& center, double k);
20     void rotate_oz(const Point& center, double k);
21 };
22
23 Point::Point() :
24     x(0), y(0), z(0) {}
25
26 Point::~Point() {}
27
28 Point::Point(double data_x, double data_y, double data_z) :
29     x(data_x), y(data_y), z(data_z) {}
30
31 Point::Point(const Point& other) :
32     x(other.x), y(other.y), z(other.z) {}
33
34 void Point::operator=(const Point &other)
35 {
36     this->x = other.x;
37     this->y = other.y;
38     this->z = other.z;
39 }
40
41 double Point::to_radians(double angle)
42 {
```

```
43      return angle * PI / 180;
44  }
45
46  void Point::rotate(const Point &center, const Vector &angles)
47  {
48      rotate_ox(center, angles.x);
49      rotate_oy(center, angles.y);
50      rotate_oz(center, angles.z);
51  }
52
53  void Point::invert_rotate(const Point &center, const Vector &angles)
54  {
55      rotate_oz(center, angles.z);
56      rotate_oy(center, angles.y);
57      rotate_ox(center, angles.x);
58  }
59
60  void Point::rotate_ox(const Point &center, double angle)
61  {
62      double y_temp, z_temp;
63
64      y_temp = center.y + (this->y - center.y) * cos(angle) + (this->z -
            center.z) * sin(angle);
65      z_temp = center.z + (this->z - center.z) * cos(angle) - (this->y -
            center.y) * sin(angle);
66
67      this->y = y_temp;
68      this->z = z_temp;
69  }
70
71  void Point::rotate_oy(const Point &center, double angle)
72  {
73      double x_temp, z_temp;
74
75      x_temp = center.x + (this->x - center.x) * cos(angle) - (this->z -
            center.z) * sin(angle);
76      z_temp = center.z + (this->z - center.z) * cos(angle) + (this->x -
            center.x) * sin(angle);
77
78      this->x = x_temp;
79      this->z = z_temp;
80  }
81
82  void Point::rotate_oz(const Point &center, double angle)
83  {
84      double x_temp, y_temp;
85
86      x_temp = center.x + (this->x - center.x) * cos(angle) + (this->y -
```

```
        center.y) * sin(angle);
87      y_temp = center.y + (this->y - center.y) * cos(angle) - (this->x -
            center.x) * sin(angle);
88
89      this->x = x_temp;
90      this->y = y_temp;
91  }
```

Рисунок Б.2 – Реализация класса вектора

```
1   class Vector
2   {
3   public:
4       double x, y, z;
5
6       Vector();
7       Vector(double data_x, double data_y, double data_z);
8       Vector(const Vector& other);
9       Vector(const Point& begin_pnt, const Point& end_pnt);
10
11      virtual ~Vector();
12
13      double get_length() const;
14      void normalize();
15      void invert();
16      double scalar_mult(const Vector& other);
17      Vector vect_mul(const Vector& v1, const Vector& v2) const;
18
19      Vector operator +(const Vector& other);
20      void operator +=(const Vector& other);
21      void operator=(const Vector& other);
22  };
23
24  //constructors and destructor
25  Vector::Vector(): x(0), y(0), z(0) {}
26
27  Vector::~Vector() {}
28
29  Vector::Vector(double data_x, double data_y, double data_z) :
30      x(data_x), y(data_y), z(data_z) {}
31
32  Vector::Vector(const Vector& other) :
33      x(other.x), y(other.y), z(other.z) {}
34
35  Vector::Vector(const Point &begin_pnt, const Point &end_pnt)
36  {
37      x = end_pnt.x - begin_pnt.x;
38      y = end_pnt.y - begin_pnt.y;
39      z = end_pnt.z - begin_pnt.z;
```

```
40 }
41
42
43 //methonds
44 double Vector::get_length() const
45 {
46     return sqrt(x * x + y * y + z * z);
47 }
48
49 void Vector::normalize()
50 {
51     double len = this->get_length();
52
53     if (len < EPS)
54         return;
55 //        throw error::InvalidOperation(__FILE__, typeid (*this).name(),
      __LINE__ - 1);
56
57     x /= len;
58     y /= len;
59     z /= len;
60 }
61
62 void Vector::invert()
63 {
64     x = -x;
65     y = -y;
66     z = -z;
67 }
68
69 double Vector::scalar_mult(const Vector &other)
70 {
71     return this->x * other.x + this->y * other.y + this->z * other.z;
72 }
73
74 Vector Vector::vect_mul(const Vector &v1, const Vector &v2) const
75 {
76     Vector result;
77
78     result.x = v1.y * v2.z - v1.z * v2.y;
79     result.y = v1.z * v2.x - v1.x * v2.z;
80     result.z = v1.x * v2.y - v1.y * v2.x;
81
82     return result;
83 }
84
85
86 //operators
```

```
87  Vector Vector::operator+(const Vector &other)
88  {
89      Vector result;
90
91      result.x = this->x + other.x;
92      result.y = this->y + other.y;
93      result.z = this->z + other.z;
94
95      return result;
96  }
97
98  void Vector::operator+=(const Vector &other)
99  {
100     this->x += other.x;
101     this->y += other.y;
102     this->z += other.z;
103 }
104
105 void Vector::operator=(const Vector &other)
106 {
107     this->x = other.x;
108     this->y = other.y;
109     this->z = other.z;
110 }
```

Рисунок Б.3 – Реализация класса вершины

```
1   class Vertex : public Point
2   {
3   public:
4       Vector n;
5
6       Vertex();
7       Vertex(double data_x, double data_y, double data_z);
8       Vertex(const Point& other);
9       Vertex(const Vertex& other);
10
11      virtual ~Vertex();
12
13      Vertex operator =(Vertex& other);
14      bool operator==(const Vertex& other);
15      void normalize();
16  };
17
18  Vertex::Vertex() {}
19
20  Vertex::~Vertex() {}
21
22  Vertex::Vertex(double data_x, double data_y, double data_z) :
```

```
23      Point(data_x, data_y, data_z) {}
24
25  Vertex::Vertex(const Point& other) :
26      Point(other) {}
27
28  Vertex::Vertex(const Vertex& other) :
29      Point(other), n(other.n) {}
30
31
32  bool Vertex::operator==(const Vertex &other)
33  {
34      if (fabs(this->x - other.x) > EPS)  return false;
35      if (fabs(this->y - other.y) > EPS)  return false;
36      if (fabs(this->z - other.z) > EPS)  return false;
37
38      return true;
39  }
40
41  Vertex Vertex::operator =(Vertex &other)
42  {
43      x = other.x;
44      y = other.y;
45      z = other.z;
46      n = other.n;
47  }
48
49  void Vertex::normalize()
50  {
51      n.normalize();
52  }
```

Рисунок Б.4 – Реализация класса стороны

```
1   class Side
2   {
3   public:
4       vector<shared_ptr<Vertex>> vertex_arr;
5       Vector n;
6       QRgb color;
7
8       Side();
9       Side(const Side& other);
10      Side(vector<shared_ptr<Vertex>> vertex_arr, const Point& control_p,
            QRgb color);
11
12      virtual ~Side();
13
14      void n_correction(const Point& control_p);
15      void set_vert_normal(const Point& control_p);
```

```
16
17 private:
18     void _find_normal();
19 };
20
21 Side::Side() {}
22
23 Side::~Side() {}
24
25 Side::Side(const Side &other)
26 {
27     vertex_arr.clear();
28     for (auto& vertex : other.vertex_arr)
29         vertex_arr.push_back(make_shared<Vertex>(*vertex));
30     n = other.n;
31     color = other.color;
32 }
33
34 Side::Side(vector<shared_ptr<Vertex>> v_arr, const Point& control_p, QRgb
       data_color)
35 {
36     if (v_arr.size() < 2)
37         throw error::DegenerateSide(__FILE__, typeid (*this).name(),
               __LINE__ - 1, v_arr.size());
38
39     color = data_color;
40     vertex_arr = v_arr;
41
42     _find_normal();
43     n_correction(control_p);
44
45     for (auto vertex : vertex_arr)
46         vertex->n += this->n;
47 }
48
49
50 void Side::n_correction(const Point &control_p)
51 {
52     Vector temp(control_p, *vertex_arr[0]);
53
54     if (this->n.scalar_mult(temp) < 0)
55         n.invert();
56 }
57
58 void Side::set_vert_normal(const Point& control_p)
59 {
60     _find_normal();
61     n_correction(control_p);
```

```
62
63      for (auto vertex : vertex_arr)
64          vertex->n += this->n;
65  }
66
67
68  void Side::_find_normal()
69  {
70      Vertex p1 = *vertex_arr[0];
71      Vertex p2 = *vertex_arr[1];
72      Vertex p3 = *vertex_arr[2];
73
74      n.x = (p2.y - p1.y)*(p3.z - p1.z) - (p3.y - p1.y)*(p2.z - p1.z);
75      n.y = (p3.x - p1.x)*(p2.z - p1.z) - (p2.x - p1.x)*(p3.z - p1.z);
76      n.z = (p2.x - p1.x)*(p3.y - p1.y) - (p3.x - p1.x)*(p2.y - p1.y);
77
78      n.normalize();
79  }
```

Рисунок Б.5 – Реализация класса камеры

```
1   class Camera : public InvisibleObject
2   {
3   public:
4       Camera();
5       Camera(const Point& position);
6       Camera(const Point& position, const Vector& direction);
7       explicit Camera(const Camera& other);
8       virtual ~Camera();
9
10      Point& get_position();
11      Vector& get_direction();
12
13      const Point& get_position() const;
14      const Vector& get_direction() const;
15
16      void operator=(const Camera& other);
17
18      virtual void accept(ObjectVisitor &visitor);
19      virtual SceneObject* clone();
20
21  private:
22      Point _pos;
23      Vector _dir;
24  };
25
26  //constructors and destructor
27  Camera::Camera() :
28      _pos(Point(10, 10, 100)) {}
```

```
29
30 Camera::Camera(const Point& position) :
31     _pos(position) {}
32
33 Camera::Camera(const Point& position, const Vector& direction) :
34     _pos(position), _dir(direction) {}
35
36 Camera::Camera(const Camera& other) :
37     _pos(other._pos), _dir(other._dir) {}
38
39 Camera::~Camera() {}
40
41
42 //methods
43 Point& Camera::get_position() {     return _pos;    }
44
45 Vector& Camera::get_direction() {   return _dir;    }
46
47 const Point& Camera::get_position() const {     return _pos;    }
48
49 const Vector& Camera::get_direction() const {   return _dir;    }
50
51 void Camera::operator=(const Camera &other)
52 {
53     this->_pos = other._pos;
54     this->_dir = other._dir;
55 }
56
57
58 void Camera::accept(ObjectVisitor& visitor)
59 {
60     visitor.visit(*this);
61 }
62
63 SceneObject *Camera::clone()
64 {
65     return (new Camera(*this));
66 }
```

Рисунок Б.6 – Реализация класса источника света

```
1 class LightSource : public InvisibleObject
2 {
3 public:
4     LightSource();
5     LightSource(const Point& position);
6     LightSource(const Point& position, double itensity);
7     explicit LightSource(const LightSource& other);
8
```

```
9       virtual ~LightSource();
10
11      Point& get_position();
12      double get_itensity();
13      void set_itensity(double itensity);
14
15      const Point& get_position() const;
16
17      void operator =(const LightSource& other);
18
19      virtual void accept(ObjectVisitor&);
20      virtual SceneObject* clone();
21
22  private:
23      Point _pos;
24      double _i;
25  };
26
27  //constructors and destructor
28  LightSource::LightSource() {}
29
30  LightSource::LightSource(const Point& position) :
31      _pos(position) {}
32
33  LightSource::LightSource(const Point& position, double itensity) :
34      _pos(position), _i(itensity) {}
35
36  LightSource::LightSource(const LightSource& other) :
37      _pos(other._pos), _i(other._i) {}
38
39  LightSource::~LightSource() {}
40
41
42  //methonds
43  Point& LightSource::get_position() {    return _pos;    }
44
45  double LightSource::get_itensity() {    return _i;  }
46
47  void LightSource::set_itensity(double itensity) {   _i = itensity;  }
48
49  const Point& LightSource::get_position() const {    return _pos;    }
50
51  void LightSource::operator =(const LightSource& other)
52  {
53      this->_pos = other._pos;
54      this->_i = other._i;
55  }
56
```

```
57 void LightSource::accept(ObjectVisitor& visitor)
58 {
59     visitor.visit(*this);
60 }
61
62 SceneObject *LightSource::clone()
63 {
64     return new LightSource(*this);
65 }
```

Рисунок Б.7 – Реализация класса модели

```
1  class Model
2  {
3  public:
4      vector<shared_ptr<Vertex>> v_arr;
5      vector<shared_ptr<Side>> s_arr;
6
7      Model();
8      Model(const char *filename, QRgb color, double delta_x, double delta_y);
9      Model(const vector<Point>& p_arr);
10     explicit Model(const Model& other);
11
12     virtual ~Model();
13
14     void normalize_n_vrt();
15     void add_vertex(const Point& pnt);
16     void add_vertices(const vector<Point> &p_arr);
17     void add_side(std::initializer_list<size_t> ind_arr, QRgb color);
18     void add_side(vector<size_t> ind_arr, QRgb color);
19
20     void correct_n();
21     Point &get_center();
22
23     void operator=(const Model& other);
24
25 protected:
26     Point _center;
27
28 private:
29     void _add_side(vector<shared_ptr<Vertex>> vertex_arr, QRgb color);
30     shared_ptr<Side> _makeSide(FILE *f, QRgb color, double delta_x, double
           delta_y);
31     shared_ptr<Point> _readPoint(FILE *f, double delta_x, double delta_y);
32 };
33
34 //constructors and destructor
35 Model::Model() {}
36
```

```
37 Model::Model(const vector<Point>& p_arr)
38 {
39     this->add_vertices(p_arr);
40 
41     _center.x = 0;
42     _center.y = 0;
43     _center.z = 0;
44 
45     for (Point pnt : p_arr)
46     {
47         _center.x += pnt.x;
48         _center.y += pnt.y;
49         _center.z += pnt.z;
50     }
51 
52     _center.x /= p_arr.size();
53     _center.y /= p_arr.size();
54     _center.z /= p_arr.size();
55 }
56 
57 Model::Model(const Model& other)
58 {
59     this->_center = other._center;
60     this->v_arr.clear();
61     this->s_arr.clear();
62     for (auto &vertex : other.v_arr)
63         v_arr.push_back(make_shared<Vertex>(*vertex));
64     for (auto &side : other.s_arr)
65         s_arr.push_back(make_shared<Side>(*side));
66 }
67 
68 Model::Model(const char *filename, QRgb color, double delta_x, double
        delta_y)
69 {
70     FILE *f = fopen(filename, "r");
71 
72     shared_ptr<Side> s;
73     while ((s = _makeSide(f, color, delta_x, delta_y)) != nullptr)
74     {
75         _add_side(s->vertex_arr, s->color);
76     }
77 
78     fclose(f);
79 
80     _center.x = 0;
81     _center.y = 0;
82     _center.z = 0;
83 
```

```
84      for (shared_ptr<Vertex> pnt : v_arr)
85      {
86          _center.x += pnt->x;
87          _center.y += pnt->y;
88          _center.z += pnt->z;
89      }
90
91      _center.x /= v_arr.size();
92      _center.y /= v_arr.size();
93      _center.z /= v_arr.size();
94  }
95
96  Model::~Model() {}
97
98
99  //methods
100 shared_ptr<Point> Model::_readPoint(FILE *f, double delta_x, double delta_y)
101 {
102     double x, y, z;
103     shared_ptr<Point> p = nullptr;
104     if (fscanf(f, "%lf %lf %lf", &x, &y, &z) == 3)
105         p = make_shared<Point>(x * 5 + delta_x * 5, y * 5 + delta_y * 5, z
            * 5);
106
107     return p;
108 }
109
110
111 shared_ptr<Side> Model::_makeSide(FILE *f, QRgb color, double delta_x,
    double delta_y)
112 {
113     shared_ptr<Side> s = nullptr;
114
115     shared_ptr<Point> p1 = _readPoint(f, delta_x, delta_y), p2 =
        _readPoint(f, delta_x, delta_y), p3 = _readPoint(f, delta_x,
        delta_y);
116     if (p1 && p2 && p3)
117     {
118         vector<shared_ptr<Vertex>> v;
119         shared_ptr<Vertex> v1(new Vertex(*p1));
120         v.push_back(v1);
121         shared_ptr<Vertex> v2(new Vertex(*p2));
122         v.push_back(v2);
123         shared_ptr<Vertex> v3(new Vertex(*p3));
124         v.push_back(v3);
125
126         add_vertex(*p1);
127         add_vertex(*p2);
```

```
128         add_vertex(*p3);
129         s = make_shared<Side>(v, *p1, color);
130     }
131 
132     return s;
133 }
134 
135 void Model::normalize_n_vrt()
136 {
137     for (auto vertex : v_arr)
138         vertex->n.normalize();
139 }
140 
141 void Model::correct_n()
142 {
143     for (auto vertex : v_arr)
144     {
145         vertex->n.x = 0;
146         vertex->n.y = 0;
147         vertex->n.z = 0;
148     }
149 
150     for (auto side : s_arr)
151         side->set_vert_normal(_center);
152 
153     normalize_n_vrt();
154 }
155 
156 void Model::_add_side(vector<shared_ptr<Vertex>> vertex_arr, QRgb color)
157 {
158     shared_ptr<Side> new_side(new Side(vertex_arr, _center, color));
159     s_arr.push_back(new_side);
160 }
161 
162 void Model::add_vertex(const Point &pnt)
163 {
164     shared_ptr<Vertex> new_vertex(new Vertex(pnt));
165     v_arr.push_back(new_vertex);
166 }
167 
168 void Model::add_vertices(const vector<Point> &p_arr)
169 {
170     for (Point point : p_arr)
171         add_vertex(point);
172 }
173 
174 void Model::add_side(std::initializer_list<size_t> ind_arr, QRgb color)
175 {
```

```
176     vector<shared_ptr<Vertex>> new_side;
177
178     for (size_t i : ind_arr)
179     {
180         if (i >= v_arr.size())
181             throw error::WrongIndex(__FILE__, typeid (*this).name(),
                    __LINE__ - 1);
182
183         new_side.push_back(v_arr[i]);
184     }
185
186     _add_side(new_side, color);
187 }
188
189 void Model::add_side(vector<size_t> ind_arr, QRgb color)
190 {
191     vector<shared_ptr<Vertex>> new_side;
192
193     for (auto i : ind_arr)
194     {
195         if (i >= v_arr.size())
196             throw error::WrongIndex(__FILE__, typeid (*this).name(),
                    __LINE__ - 1);
197
198         new_side.push_back(v_arr[i]);
199     }
200
201     _add_side(new_side, color);
202 }
203
204 Point& Model::get_center()
205 {
206     return _center;
207 }
208
209 void Model::operator=(const Model &other)
210 {
211     this->_center = other._center;
212     this->v_arr.clear();
213     this->s_arr.clear();
214     for (auto &vertex : other.v_arr)
215         v_arr.push_back(make_shared<Vertex>(*vertex));
216     for (auto &side : other.s_arr)
217         s_arr.push_back(make_shared<Side>(*side));
218 }
```

Рисунок Б.8 – Реализация класса лепестка

```
1 class Petal : public VisibleObject
```

```
2  {
3  public:
4      Petal(const char *color, int i, double delta_x, double delta_y);
5      explicit Petal(const Petal& other);
6
7      virtual ~Petal();
8
9      virtual void accept(ObjectVisitor& visitor);
10     virtual SceneObject* clone();
11
12     void rotate(Point &cent, Vector &dir, Point fl_cent, double angle);
13
14 private:
15     QRgb _color;
16 };
17
18 Petal::Petal(const char* color, int i, double delta_x, double delta_y)
19 {
20     _color = QColor(color).rgba();
21     char filename[FILENAME_MAX] =
           "C:/Users/Honor/Desktop/cw/program/petal_grid_";
22     filename[strlen(filename)] = (char) i + '0';
23     strncat(filename, ".txt", 4);
24     _model = make_shared<Model>(filename, _color, delta_x, delta_y);
25 }
26
27 Petal::Petal(const Petal& other) : VisibleObject(other)
28 {
29     _color = other._color;
30 }
31
32 Petal::~Petal() {}
33
34
35 //methods
36 void Petal::accept(ObjectVisitor &visitor)
37 {
38     visitor.visit(*this);
39 }
40
41 SceneObject* Petal::clone()
42 {
43     return new Petal(*this);
44 }
45
46 void Petal::rotate(Point &cent, Vector &dir, Point fl_cent, double angle)
47 {
48     Rotate(dir, cent).execute(*(this->get_model()));
```

```
49      Rotate(Vector(0, 0, angle), fl_cent).execute(cent);
50  }
```

Рисунок Б.9 – Реализация класса листа

```
1   class Leaf : public VisibleObject
2   {
3   public:
4       Leaf(double delta_x, double delta_y);
5       explicit Leaf(const Leaf& other);
6
7       virtual ~Leaf();
8
9       virtual void accept(ObjectVisitor& visitor);
10      virtual SceneObject* clone();
11
12  private:
13      QRgb _color;
14  };
15
16  Leaf::Leaf(double delta_x, double delta_y)
17  {
18      _color = QColor(Qt::darkGreen).rgba();
19
20      _model =
            make_shared<Model>("C:/Users/Honor/Desktop/cw/program/Leaf_grid.txt",
            _color, delta_x, delta_y);
21
22  }
23
24  Leaf::Leaf(const Leaf& other) : VisibleObject(other)
25  {
26      _color = other._color;
27  }
28
29  Leaf::~Leaf() {}
30
31
32  //methods
33  void Leaf::accept(ObjectVisitor &visitor)
34  {
35      visitor.visit(*this);
36  }
37
38  SceneObject* Leaf::clone()
39  {
40      return new Leaf(*this);
41  }
```

Рисунок Б.10 – Реализация класса центральной части

```
class Center : public VisibleObject
{
public:
    Center(double delta_x, double delta_y);
    explicit Center(const Center& other);

    virtual ~Center();

    virtual void accept(ObjectVisitor& visitor);
    virtual SceneObject* clone();

private:
    QRgb _color;
};

Center::Center(double delta_x, double delta_y)
{
    _color = QColor("#FFCC00").rgba();

    _model =
       make_shared<Model>("C:/Users/Honor/Desktop/cw/program/Center_grid.txt",
       _color, delta_x, delta_y);

}

Center::Center(const Center& other) : VisibleObject(other)
{
    _color = other._color;
}

Center::~Center() {}


//methods
void Center::accept(ObjectVisitor &visitor)
{
    visitor.visit(*this);
}

SceneObject* Center::clone()
{
    return new Center(*this);
}
```

Рисунок Б.11 – Реализация класса ограничивающей поверхности

```
class Surface : public VisibleObject
{
```

```
3  public:
4      Surface(double delta_x, double delta_y);
5      explicit Surface(const Surface& other);
6
7      virtual ~Surface();
8
9      virtual void accept(ObjectVisitor& visitor);
10     virtual SceneObject* clone();
11
12 private:
13     QRgb _color;
14 };
15
16 Surface::Surface(double delta_x, double delta_y)
17 {
18     _color = QColor("#B6845D").rgba();
19
20     _model =
           make_shared<Model>("C:/Users/Honor/Desktop/cw/program/surface_grid.txt",
           _color, delta_x, delta_y);
21
22 }
23
24 Surface::Surface(const Surface& other) : VisibleObject(other)
25 {
26     _color = other._color;
27 }
28
29 Surface::~Surface() {}
30
31
32 //methods
33 void Surface::accept(ObjectVisitor &visitor)
34 {
35     visitor.visit(*this);
36 }
37
38 SceneObject* Surface::clone()
39 {
40     return new Surface(*this);
41 }
```

Рисунок Б.12 – Реализация класса цветоложа

```
1  class Receptacle : public VisibleObject
2  {
3  public:
4      Receptacle(double delta_x, double delta_y);
5      explicit Receptacle(const Receptacle& other);
```

```
6
7       virtual ~Receptacle();
8
9       virtual void accept(ObjectVisitor& visitor);
10      virtual SceneObject* clone();
11
12  private:
13      QRgb _color;
14  };
15
16  Receptacle::Receptacle(double delta_x, double delta_y)
17  {
18      _color = QColor(Qt::darkGreen).rgba();
19
20      _model =
            make_shared<Model>("C:/Users/Honor/Desktop/cw/program/receptacle_grid.txt"
            _color, delta_x, delta_y);
21
22  }
23
24  Receptacle::Receptacle(const Receptacle& other) : VisibleObject(other)
25  {
26      _color = other._color;
27  }
28
29  Receptacle::~Receptacle() {}
30
31
32  //methods
33  void Receptacle::accept(ObjectVisitor &visitor)
34  {
35      visitor.visit(*this);
36  }
37
38  SceneObject* Receptacle::clone()
39  {
40      return new Receptacle(*this);
41  }
```

Рисунок Б.13 – Реализация класса стебля

```
1   class Stem : public VisibleObject
2   {
3   public:
4       Stem(double delta_x, double delta_y);
5       explicit Stem(const Stem& other);
6
7       virtual ~Stem();
8
```

```
9       virtual void accept(ObjectVisitor& visitor);
10      virtual SceneObject* clone();
11
12  private:
13      QRgb _color;
14  };
15
16  Stem::Stem(double delta_x, double delta_y)
17  {
18      _color = QColor(Qt::darkGreen).rgba();
19      _model =
            make_shared<Model>("C:/Users/Honor/Desktop/cw/program/Stem_grid.txt",
            _color, delta_x, delta_y);
20
21  }
22
23  Stem::Stem(const Stem& other) : VisibleObject(other)
24  {
25      _color = other._color;
26  }
27
28  Stem::~Stem() {}
29
30
31  //methods
32  void Stem::accept(ObjectVisitor &visitor)
33  {
34      visitor.visit(*this);
35  }
36
37  SceneObject* Stem::clone()
38  {
39      return new Stem(*this);
40  }
```

ПРИЛОЖЕНИЕ В

Ниже приведены реализации классов, отвечающих за визуализацию, трансформацию, смену времени суток.

Рисунок В.1 – Реализация классов, отвечающих за визуализацию, трансформацию, смену времени суток

```
1  class SceneManager
2  {
3  public:
4      SceneManager(weak_ptr<Scene> scene);
5      virtual ~SceneManager() = 0;
6
7      virtual void execute() = 0;
8
9  protected:
10     weak_ptr<Scene> _scene;
11 };
12
13
14 class InitDrawManager : public SceneManager
15 {
16 public:
17     InitDrawManager(weak_ptr<Scene> scene, shared_ptr<Drawer> draw);
18     virtual ~InitDrawManager();
19
20     virtual void execute();
21
22 private:
23     shared_ptr<Drawer> _draw;
24 };
25
26
27 class DrawManager : public SceneManager
28 {
29 public:
30     DrawManager(weak_ptr<Scene> scene);
31     virtual ~DrawManager();
32
33     virtual void execute();
34 };
35
36 class DrawNightManager : public SceneManager
37 {
38 public:
39     DrawNightManager(weak_ptr<Scene> scene);
```

```
40      virtual ~DrawNightManager();
41
42      virtual void execute();
43  };
44
45  class DrawDayManager : public SceneManager
46  {
47  public:
48      DrawDayManager(weak_ptr<Scene> scene);
49      virtual ~DrawDayManager();
50
51      virtual void execute();
52  };
53
54  class NightManager : public SceneManager
55  {
56  public:
57      NightManager(weak_ptr<Scene> scene);
58      virtual ~NightManager();
59
60      virtual void execute();
61  };
62
63  class DayManager : public SceneManager
64  {
65  public:
66      DayManager(weak_ptr<Scene> scene);
67      virtual ~DayManager();
68
69      virtual void execute();
70  };
71
72
73  class TransformManager : public SceneManager
74  {
75  public:
76      TransformManager(weak_ptr<Scene> scene,
77                       shared_ptr<Transformator> transf);
78      TransformManager(weak_ptr<Scene> scene, weak_ptr<Petal> p,
79                       shared_ptr<Transformator> transf);
80      virtual ~TransformManager();
81
82      virtual void execute();
83
84      void camera_execute();
85      void light_execute();
86      void petal_execute();
87
```

```
88  private:
89      shared_ptr<Transformator> _transf;
90      shared_ptr<Petal> _p;
91  };
92
93  SceneManager::SceneManager(weak_ptr<Scene> scene) :
94      _scene(scene) {}
95
96  SceneManager::~SceneManager() {}
97
98
99  InitDrawManager::InitDrawManager(weak_ptr<Scene> scene, shared_ptr<Drawer>
        draw) :
100     SceneManager(scene), _draw(draw) {}
101
102 InitDrawManager::~InitDrawManager() {}
103
104 void InitDrawManager::execute()
105 {
106     if (_scene.expired())
107         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
                - 1);
108
109     _scene.lock()->set_drawer(_draw);
110 }
111
112
113 DrawManager::DrawManager(weak_ptr<Scene> scene) :
114     SceneManager(scene) {}
115
116 DrawManager::~DrawManager() {}
117
118 void DrawManager::execute()
119 {
120     if (_scene.expired())
121         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
                - 1);
122
123     shared_ptr<Scene> scene = _scene.lock();
124
125     shared_ptr<Visualizer> visual(new Visualizer());
126
127     visual->set_camera(*scene->get_camera());
128     visual->set_light(*scene->get_light());
129     visual->set_draw(scene->get_drawer());
130     visual->clear(0);
131
132     shared_ptr<ObjectVisitor> visitor(new DrawVisitor(visual));
```

```
133
134     for (auto object : *_scene.lock())
135         object->accept(*visitor);
136
137     visual->show_scene();
138 }
139
140 DrawNightManager::DrawNightManager(weak_ptr<Scene> scene) :
141     SceneManager(scene) {}
142
143 DrawNightManager::~DrawNightManager() {}
144
145 void DrawNightManager::execute()
146 {
147     if (_scene.expired())
148         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
               - 1);
149
150     shared_ptr<Scene> scene = _scene.lock();
151
152     shared_ptr<Visualizer> visual(new Visualizer());
153
154     visual->set_camera(*scene->get_camera());
155
156     (scene->get_light())->set_itensity((scene->get_light())->get_itensity()
          - 3);
157     visual->set_light(*scene->get_light());
158
159     visual->set_draw(scene->get_drawer());
160     visual->clear(1);
161
162     shared_ptr<ObjectVisitor> visitor(new DrawVisitor(visual));
163
164     for (auto object : *_scene.lock())
165         object->accept(*visitor);
166
167     visual->show_scene();
168 }
169
170 DrawDayManager::DrawDayManager(weak_ptr<Scene> scene) :
171     SceneManager(scene) {}
172
173 DrawDayManager::~DrawDayManager() {}
174
175 void DrawDayManager::execute()
176 {
177     if (_scene.expired())
178         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
```

```
            - 1);
179
180     shared_ptr<Scene> scene = _scene.lock();
181
182     shared_ptr<Visualizer> visual(new Visualizer());
183
184     visual->set_camera(*scene->get_camera());
185
186     (scene->get_light())->set_itensity((scene->get_light())->get_itensity()
            + 3);
187     visual->set_light(*scene->get_light());
188
189     visual->set_draw(scene->get_drawer());
190     visual->clear(-1);
191
192     shared_ptr<ObjectVisitor> visitor(new DrawVisitor(visual));
193
194     for (auto object : *_scene.lock())
195         object->accept(*visitor);
196
197     visual->show_scene();
198 }
199
200 NightManager::NightManager(weak_ptr<Scene> scene) :
201     SceneManager(scene) {}
202
203 NightManager::~NightManager() {}
204
205 void NightManager::execute()
206 {
207     if (_scene.expired())
208         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
                - 1);
209     shared_ptr<Scene> scene = _scene.lock();
210
211     double delta_x[3] = {0, 13, -13};
212     double delta_y[3] = {8, -7, -7};
213
214     // rotate petals
215     double delta = PI / 1500;
216
217     for (int j = 0; j < 3; ++j)
218     {
219         Vector dir(-5.5 * delta, 0, 0);
220         Point cent(0.122 * 5 + delta_x[j] * 5, 2.3 * 5 + delta_y[j] * 5,
                21.184 * 5);
221         Point fl_cent(1.25 + delta_x[j] * 5, -1 + delta_y[j] * 5, 101.167);
222         for (int i = 0; i < 9; ++i)
```

```
223         {
224             if (i == 1)
225             {
226                 dir.x = -4.5 * delta;
227                 dir.y = -3 * delta;
228             }
229             else if (i == 2)
230             {
231                 dir.x = -1.5 * delta;
232                 dir.y = -4.8 * delta;
233             }
234             else if (i == 3)
235             {
236                 dir.x = 2 * delta;
237                 dir.y = -4.5 * delta;
238             }
239             else if (i == 4)
240             {
241                 dir.x = 4.5 * delta;
242                 dir.y = -1.8 * delta;
243             }
244             else if (i == 5)
245             {
246                 dir.x = 4.5 * delta;
247                 dir.y = 1.8 * delta;
248             }
249             else if (i == 6)
250             {
251                 dir.x = 2 * delta;
252                 dir.y = 4.5 * delta;
253             }
254             else if (i == 7)
255             {
256                 dir.x = -1.5 * delta;
257                 dir.y = 4.8 * delta;
258             }
259             else if (i == 8)
260             {
261                 dir.x = -4.5 * delta;
262                 dir.y = 3 * delta;
263             }
264             scene->_petal_arr[j * 9 + i]->rotate(cent, dir, fl_cent, -2 *
                    PI / 9);
265         }
266     }
267 }
268
269 DayManager::DayManager(weak_ptr<Scene> scene) :
```

```
270     SceneManager(scene) {}
271
272 DayManager::~DayManager() {}
273
274 void DayManager::execute()
275 {
276     if (_scene.expired())
277         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
                - 1);
278     shared_ptr<Scene> scene = _scene.lock();
279
280     double delta_x[3] = {0, 13, -13};
281     double delta_y[3] = {8, -7, -7};
282
283     // rotate petals
284     double delta = PI / 1500;
285
286     for (int j = 0; j < 3; ++j)
287     {
288         Vector dir(5.5 * delta, 0, 0);
289         Point cent(0.122 * 5 + delta_x[j] * 5, 2.3 * 5 + delta_y[j] * 5,
                21.184 * 5);
290         Point fl_cent(1.25 + delta_x[j] * 5, -1 + delta_y[j] * 5, 101.167);
291         for (int i = 0; i < 9; ++i)
292         {
293             if (i == 1)
294             {
295                 dir.x = 4.5 * delta;
296                 dir.y = 3 * delta;
297             }
298             else if (i == 2)
299             {
300                 dir.x = 1.5 * delta;
301                 dir.y = 4.8 * delta;
302             }
303             else if (i == 3)
304             {
305                 dir.x = -2 * delta;
306                 dir.y = 4.5 * delta;
307             }
308             else if (i == 4)
309             {
310                 dir.x = -4.5 * delta;
311                 dir.y = 1.8 * delta;
312             }
313             else if (i == 5)
314             {
315                 dir.x = -4.5 * delta;
```

```
316                 dir.y = -1.8 * delta;
317             }
318             else if (i == 6)
319             {
320                 dir.x = -2 * delta;
321                 dir.y = -4.5 * delta;
322             }
323             else if (i == 7)
324             {
325                 dir.x = 1.5 * delta;
326                 dir.y = -4.8 * delta;
327             }
328             else if (i == 8)
329             {
330                 dir.x = 4.5 * delta;
331                 dir.y = -3 * delta;
332             }
333             scene->_petal_arr[j * 9 + i]->rotate(cent, dir, fl_cent, -2 *
                    PI / 9);
334         }
335     }
336 }
337
338
339 TransformManager::TransformManager(weak_ptr<Scene> scene,
        shared_ptr<Transformator> transf) :
340     SceneManager(scene), _transf(transf) {}
341
342 TransformManager::TransformManager(weak_ptr<Scene> scene, weak_ptr<Petal>
        p, shared_ptr<Transformator> transf) :
343     SceneManager(scene), _p(p), _transf(transf) {}
344
345 TransformManager::~TransformManager() {}
346
347 void TransformManager::execute()
348 {
349     if (_scene.expired())
350         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
                - 1);
351 }
352
353 void TransformManager::camera_execute()
354 {
355     if (_scene.expired())
356         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
                - 1);
357
358     _transf->rotate(_scene.lock()->get_camera()->get_direction());
```

```
359     _transf->transform(*_scene.lock()->get_camera());
360 }
361
362 void TransformManager::light_execute()
363 {
364     if (_scene.expired())
365         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
                - 1);
366
367     _transf->transform(_scene.lock()->get_light()->get_position());
368 }
369
370 void TransformManager::petal_execute()
371 {
372     if (_scene.expired())
373         throw error::SceneExpired(__FILE__, typeid (*this).name(), __LINE__
                - 1);
374
375     _transf->transform(*(_p->get_model()));
376 }
```