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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Программное обеспечение для визуализации трёхмерной модели материнской платы»

Студент **ИУ7И-54Б** _____ **Рохас М.А.**

Руководитель _____ **Быстрицкая А.Ю.**

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время одной из главных задач компьютерной графики является построение трехмерных изображений. Огромную роль играют программы, позволяющие моделировать 3D объекты, так как прежде чем что-нибудь выпустить в производство, необходимо понять, как будет выглядеть конечный продукт и какой будет последовательность его сборки.

В данной работе было решено создать программу-конструктор, основанную на всем известном конструкторе LEGO, которая позволяла бы создавать трехмерные фигуры из предложенных деталей Лего.

Цель работы — разработать программу моделирования детского конструктора LEGO. В программе задан набор элементов конструктора, которым пользователь может манипулировать. Из данных элементов можно собирать различные конструкции, выставляя одни элементы на другие. В программе предусмотрено управление положением сцены для предоставления возможности осматривать ее с разных сторон. Также можно установить источник света.

Для реализации поставленной цели следует решить следующие задачи:

— изучить модели представления объектов и выбрать подходящую;

— выбрать алгоритмы для отображения объектов на сцене;

— выбрать механизм размещения объектов (при работе с конструктором обычно подразумевается, что размещение объектов происходит так или иначе упорядоченным способом);

— выбрать язык программирования и среды разработки;

— реализовать выбранные алгоритмы отображения объектов;

— реализовать программное обеспечение с пользовательским интерфейсом, позволяющее собирать конструкции из блоков LEGO.

1 Аналитический раздел

1.1 Модель представления объектов сцены

В программе имеется набор трехмерных объектов, для которых необходимо определить модель их представления.

Сцена — часть плоскости в форме пластинки LEGO, которая окрашена в серый цвет. Сцену можно вращать, приближать и удалять от глаз наблюдателя. Также есть возможность двигать сцену в пространстве (перемещаться по ней).

Трехмерные объекты — набор элементов конструктора LEGO, которые пользователь может расставлять по сетке сцены и состыковывать с другими элементами конструктора. Каждый элемент представляет собой множество точек, которые соединены отрезками. Заданный набор элементов конструктора не может быть изменен. У пользователя есть возможность перемещать данные элементы по сетке сцены, добавлять и удалять их.

На рисунках 1.1–1.5 представлены примеры доступных деталей LEGO.

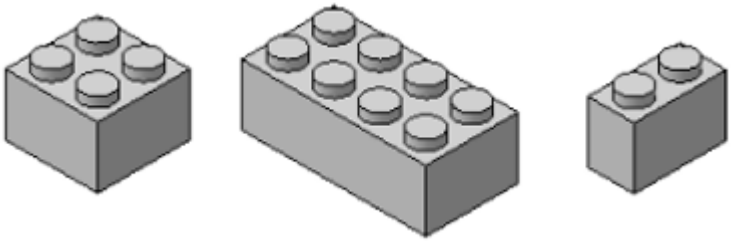

Рисунок 1.1 – Примеры кирпичиков

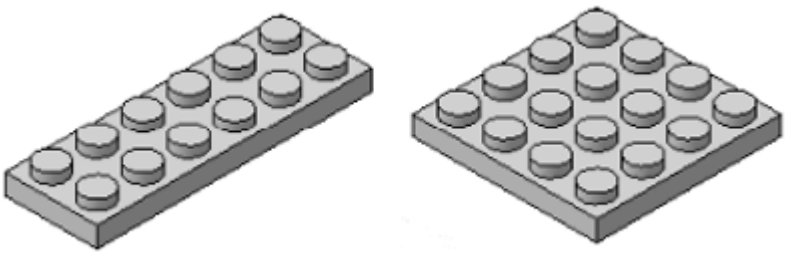

Рисунок 1.2 – Примеры пластинок

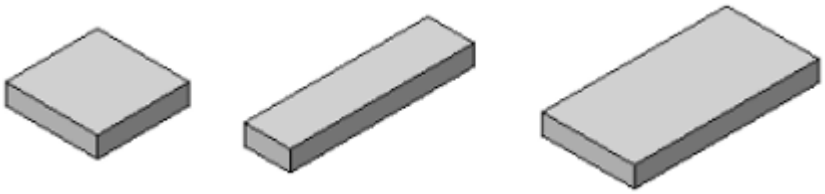

Рисунок 1.3 – Примеры плиток

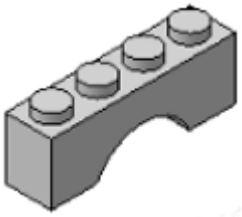

Рисунок 1.4 – Пример арки

Рисунок 1.5 – Пример цилиндра

Источник света — точечный объект в пространстве на некотором расстоянии от сцены, расположение которого задается с помощью значений углов по осям X и Y относительно наблюдателя. Лучи света распространяются от источника параллельно в сторону сцены в некотором ограниченном секторе.

Имеется три варианта отображения объекта.

— *Каркасная модель* — задается информация о вершинах и рёбрах объектов. Это простейший вид моделей, так как задается минимум информации. Однако данный вид представления объектов не всегда корректно передает форму объекта.

— *Поверхностная модель* — такая модель позволяет описывать и манипулировать поверхностями и кривыми на модели. Все поверхности могут быть аппроксимированы многогранниками. Недостаток данного вида модели заключается в том, что отсутствует информация, о том, с какой стороны поверхности находится материал.

— *Твердотельная модель* — помимо информации о поверхности добавляется информация о том, где расположен материал. Чаще всего это делают путём указания направления внутренней нормали.

В случае конструктора лучше всего использовать поверхностную модель, так как в данном случае не важно, из какого материала сделан элемент конструктора, а каркасная модель не всегда целиком передает форму объекта.

1.2 Способ задания поверхности

Существует несколько вариантов задания поверхностной модели.

— *Аналитический способ* — заключается в том, что для получения поверхности нужно дополнительно вычислять функцию, зависящую от параметра.

— *Полигональная сетка* — заключается в том, что информация о модели хранится в виде совокупности вершин, ребер и граней.

Из двух представленных вариантов наиболее оптимальным является использование полигональной сетки, так как такой вариант задания поверхности позволит более быстро выполнять операции над объектами.

1.3 Способ хранения полигональной модели

Далее необходимо выбрать, как именно хранить такую сетку. Существует несколько вариантов, как это можно реализовать.

— *Вершинное представление* (хранится информация о вершинах, которые в свою очередь указывают на другие вершины, с которыми они соединены).

— *Список граней* (объект представляется как множество граней и вершин, любая грань состоит минимум из трёх вершин).

— *Таблица углов* (веер треугольников).

Для хранения полигональной сетки будет использоваться список граней.

1.4 Алгоритмы построения изображения

Для того, чтобы выбрать подходящий алгоритм построения изображения, необходимо осуществить краткий обзор известных алгоритмов и осуществить выбор наиболее подходящего для решения поставленной задачи.

1.4.1 Алгоритм Робертса

Алгоритм Робертса [1] представляет собой первое известное решение задачи об удалении невидимых линий. Это метод, работающий в объектном пространстве. В соответствии с алгоритмом, прежде всего из каждого тела удаляются те ребра или грани, которые перекрываются самим телом. Затем

каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, перекрываются этими телами.

Преимущества данного алгоритма в том, что математические методы, используемые в нем, просты и точны. Более поздние реализации алгоритма, например, использующие предварительную сортировку вдоль оси z, демонстрируют почти линейную зависимость от числа объектов.

Минус этого алгоритма в том, что вычислительная трудоемкость алгоритма Робертса растет теоретически, как квадрат числа объектов. Реализация оптимизированных алгоритмов весьма сложна.

Недостатки алгоритма:

— вычислительная трудоёмкость (теоретически она растёт прямо пропорционально квадрату количества объектов сцены);

— существуют трудности на этапе подготовки информации об объектах сцены;

— возможность работы только с выпуклыми объектами.

В качестве преимущества можно отметить высокую точность вычислений.

1.4.2 Алгоритм Варнока

Алгоритм Варнока [1] работает в пространстве изображений. В основу алгоритма положен принцип «разделяй и властвуй», состоящий в разбиении области рисунка на более мелкие подобласти (окна). Для каждой подобласти (окна) определяются связанные с ней многоугольники и те из них, видимость которых определить «легко», изображаются на экране. В противном же случае разбиение повторяется, и для каждой из вновь полученных подобластей рекурсивно применяется процедура принятия решения. Предполагается, что с уменьшением размеров области ее перекрывает все меньшее и меньшее количество многоугольников. Считается, что в пределе будут получены области, содержащие не более одного многоугольника, и решение будет принято достаточно просто. Если же в процессе разбиения будут оставаться области,

содержащие не один многоугольник, то следует продолжать процесс разбиения до тех пор, пока размер области не станет совпадать с одним пикселом. В этом случае для полученного пикселя необходимо вычислить глубину (значение координаты z) каждого многоугольника и визуализировать тот из них, у которого максимальное значение этой координаты.

1.4.3 Алгоритм обратной трассировки лучей

Данный алгоритм работает в пространстве изображения. Основная идея заключается в том, что для каждого пиксела на дисплее проводится прямой луч от наблюдателя до элемента сцены. Первое пересечение используется для определения цвета пиксела как функции пересекаемой поверхности элемента. Также проводятся вторичные лучи от точек пересечения до разных источников света для определения освещённости пиксела. Если эти лучи блокируются объектом, то данная точка находится в тени, которую отбрасывает рассматриваемый источник света. Иначе источник света влияет на освещение. Совокупность всех вторичных лучей, которые достигают источника света, определяет качество освещения, которое попадает на объект сцены. Также для более реалистичного изображения необходимо проводить лучи отражения и лучи преломления.

Преимуществом этого метода является реалистичное изображение объекта, построенное по физическим законам. А недостатком — большая трудоемкость вычислений.

1.4.4 Алгоритм, использующий z-буффер

Данный алгоритм удаления невидимых поверхностей является одним из простейших. Этот алгоритм работает в пространстве изображения. Здесь обобщается идея о буфере кадра. Буфер кадра используется для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения. Наряду с буфером кадра вводится z-буфер, представляющий собой специальный буфер глубины, в котором запоминаются координаты z (глубина) каждого видимого пикселя в пространстве изображения. В процессе работы

глубина (значение координаты z) каждого нового пикселя, который надо занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксель расположен ближе к наблюдателю, чем пиксел, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра. Помимо этого производится корректировка z-буфера: в него заносится глубина нового пикселя. Если же глубина (значение координаты z) нового пикселя меньше, чем хранящегося в буфере, то никаких действий производить не надо. В сущности, алгоритм для каждой точки (x, y) находит наибольшее значение функции $Z(x, y)$.

Этот алгоритм несмотря на свою простоту позволяет удалять сложные поверхности и позволяет визуализировать пересечения таких поверхностей. Сцены могут быть произвольной сложности, а поскольку размеры изображения ограничены размером экрана дисплея, то трудоемкость алгоритма имеет линейную зависимость от числа рассматриваемых поверхностей. Элементы сцены заносятся в буфер кадра в произвольном порядке, поэтому в данном алгоритме не тратится время на выполнение сортировок, необходимых в других алгоритмах.

В рамках данной задачи нас интересует вычислительная сложность данного алгоритма.

Для каждого из $m \cdot n$ пикселей в буфере кадра размера $m \times n$ мы находим ближайший из k полигонов. Исходя из данных соображений, получаем nmk вычислений глубины. Таким образом временная сложность алгоритма — $O(nmk)$. Вдобавок, время работы алгоритма практически не зависит от числа многоугольников, так как с увеличением их числа уменьшается размер, поэтому для каждого пикселя k можно считать некоторой константой.

Преимущества алгоритма:

— относительно простая реализация;

— небольшая вычислительная трудоемкость;

— отсутствие сортировки объектов по параметру глубины, вследствие чего происходит экономия времени;

— возможность небольшой модификации алгоритма для работы с тенями.

Недостатки алгоритма:

— использование двух буферов, как следствие большое использование памяти;

— для работы с прозрачными объектами необходима модификация алгоритма.

1.4.5 Выбор алгоритма построения теней

При выборе алгоритма построения теней следует обратить внимание на простоту реализации алгоритма и время, необходимое для его написания. Если в качестве алгоритма удаления невидимых ребер и поверхностей использовать алгоритм обратной трассировки лучей, то тени уже будут построены в результате работы данного алгоритма (построение теней происходит во время выполнения алгоритма, т.к. пиксел затемняется в случае, когда испускаемый луч попадает на объект, но не попадает на источник света). Поскольку стало ясно, что алгоритм обратной трассировки использовать не следует, можно попробовать модифицировать алгоритм z-буфера путём добавления вычисления теневого буфера.

Благодаря такой модификации не потребуется писать много дополнительного кода (а лишь немного изменить уже написанный), да и модифицировать уже реализованный алгоритм удобнее, чем использовать что-то новое.

1.4.6 Итоговый выбор алгоритмов

Оценив все изложенные выше алгоритмы, я пришел к выводу, что самой подходящей комбинацией будет z-буфер и его дальнейшая модификация для построения теней. Z-буфер выбран потому, что его достаточно просто реализовать, он достаточно устойчив к входным данным, оценка сложности позволяет рассчитывать на работу этого алгоритма в реальном времени, что является одним из основных требований к данной задаче.

1.5 Выбор механизма размещения объектов

Необходимо дать пользователю возможность точно располагать объекты на сцене. Для этого используется следующий механизм:

— объекты можно добавлять только в клетки деталей LEGO, кроме деталей плиток — на них нельзя ставить другие объекты;

— если пользователь указывает координату для добавления нового объекта, а на этой клетке уже стоит деталь, то новая поставится на уже существующую;

— нельзя удалять и перемещать элементы, к которым прикреплены другие детали.

2 Конструкторский раздел

2.1 Алгоритм, использующий z-буфер

Для реализации в программе был выбран алгоритм, использующий z-буфер, так как этот алгоритм способен обеспечить необходимую скорость работы.

Z-буфер является одним из простейших алгоритмов удаления невидимых поверхностей. Работает этот алгоритм в пространстве изображения. Идея z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пикселя в пространстве изображения, z-буфер — это отдельный буфер глубины, используемый для запоминания координаты z (глубины) каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение z каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по x и y наибольшего значения функции $Z(x, y)$.

Алгоритм (рисунок 2.1):

1) заполнение буфера кадра фоновый значением интенсивности (цвета);

2) заполнение z-буфера минимальным значением Z;

3) преобразование каждого многоугольника в растровую форму в произвольном порядке;

4) вычисление для каждого пикселя с координатами (x, y), принадлежащего многоугольнику, его глубины $Z(x, y)$;

5) сравнение глубины $Z(x, y)$ со значением $Z_{\text{буф}}(x, y)$, которое хранится в z-буфере для пикселя с теми же координатами $(x,\ y)$;

6) запись атрибута очередного многоугольника в буфер кадра и замена $Z_{\text{буф}}(x, y)$ на значение $Z(x, y)$, если $Z(x, y) > Z_{\text{буф}}(x, y)$.

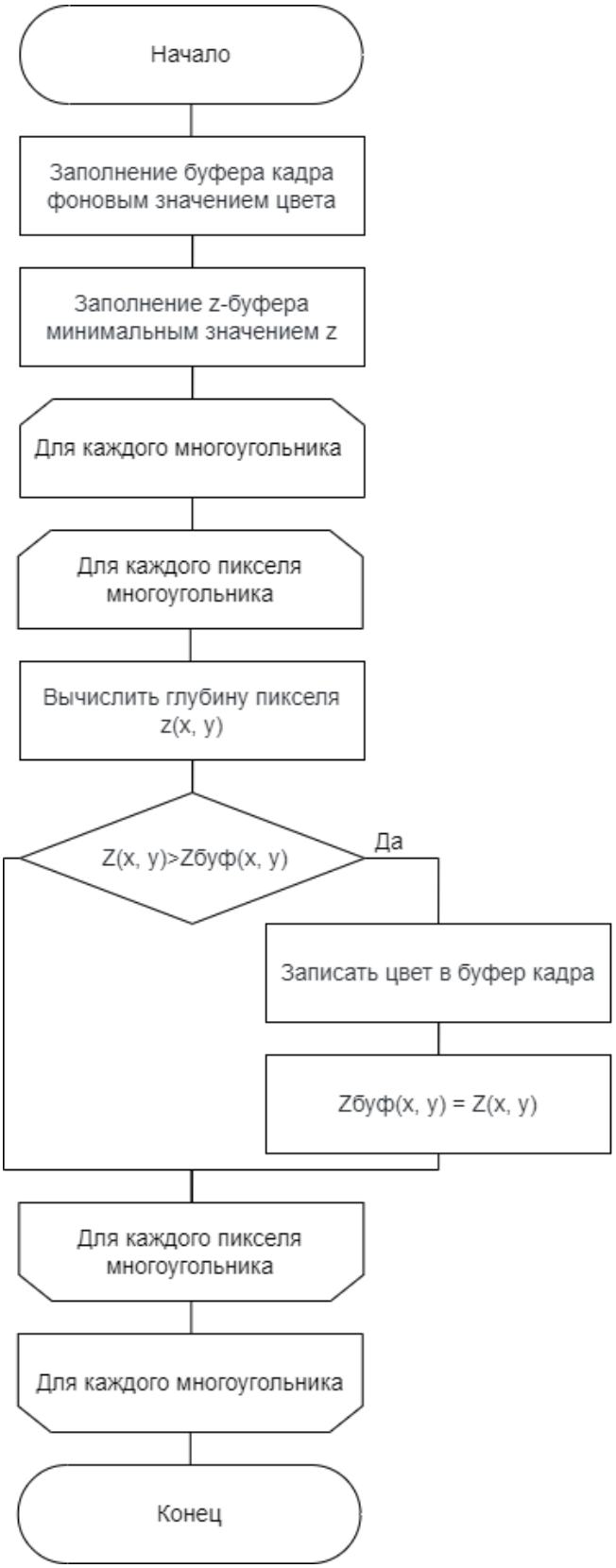

Рисунок 2.1 – Схема алгоритма z-буфера

Как уже было отмечено, главное преимущество алгоритма — его про-

стота. Так как габариты пространства изображения фиксированы, оценка вычис-лительной трудоемкости алгоритма не более чем линейна. Поскольку элементы сцены или картинки можно заносить в буфер кадра или в z-буфер в произвольном порядке, их не нужно предварительно сортировать по приоритету глубины. Поэтому экономится вычислительное время, затрачиваемое на сортировку по глубине.

2.2 Модификация алгоритма, использующего z-буфер, для построения теней

Модификация состоит из двух этапов.

1) Строится сцена из точки, совпадающей с источником. Значения z для этого ракурса хранятся в отдельном теневом z-буфере.

2) Строится сцена 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'.

На рисунках 2.2–2.3 представлен модифицированный алгоритм z-буфера для построения теней.

Рисунок 2.2 – Схема модифицированного алгоритма z-буфера для отображения теней (ч. 1)

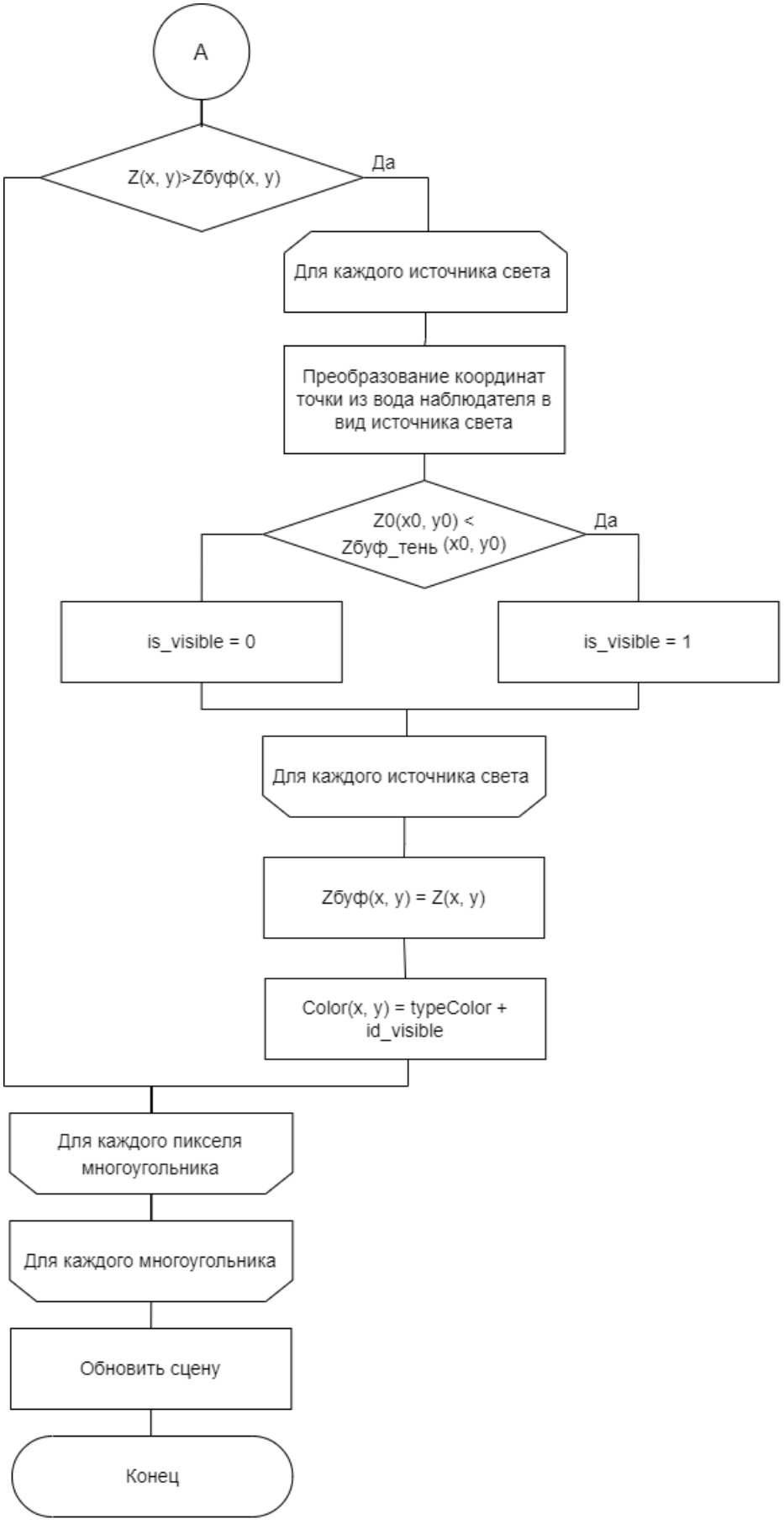

Рисунок 2.3 – Схема модифицированного алгоритма z-буфера для отображения теней (ч. 2)

2.3 Используемые типы и структуры данных для представления объектов

В таблице 2.1 представлены объекты, способ их представления в программе и типы и структуры данных для их описания.

Таблица 2.1 – Используемые типы и структуры данных для представлений объектов

Данные	Представление	Тип
Точка в пространстве	Координаты точки по трем осям	Список типа *double*
Грань	Номера задействованных вершин	Список типа *int*
Полигональная модель	Набор задействованных вершин и граней	Списки типа $Vertex$ (вершина) и $Facet$ (грань)
Источник света	Углы по двум осям	Тип *int*

2.4 Описание используемых классов

В результате разработки программы были реализованы классы, представленные ниже.

— $Dot3D$ — класс точки в пространстве, который хранит ее кординаты.

— $Vertex$ — класс для описания вершины. Хранит координаты самой вершины и список граней, в которых она находится.

— $Facet$ — класс задающий грань с помощью списка ее вершин.

— $PolygonModel$ — класс полигональной модели, хранящий в себе списки вершин и граней модели, а также информацию о размерах модели, ее типе и положении в сетке сцены.

— $Light$ — класс, описывающий источник света и хранящий информацию о положении источника, а также информацию о теневой карте.

— $SceneInf$ — класс, хранящий информацию о текущем состоянии сцены: размеры платформы, список добавленных моделей (деталей LEGO) и источников света, а также информацию о «заполненности» по высоте каждой клетки платформы.

— $AddBrick$, $AddPlate$, $AddTile$, $AddArc$, $AddCylinder$, $AddLight$ — классы интерфейсов добавления соответствующих объектов на сцену.

— $ObjectMove$, $ObjectChange$, $ObjectDelete$ — классы интерфейсов перемещения и удаления объектов.

— $Drawer$ — класс, который необходим для отрисовки сцены, он хранит буферы глубины и кадра.

— $Facade$ — класс для связи взаимодействия пользователя с программой. Хранит объекты класса $SceneInf$ и $Drawer$.

2.5 Структура программного комплекса

Программа должна обеспечить выполнение следующих функций:

— создание платформы для построения конструкций заданного размера;

— добавление на сцену деталей LEGO заданного размера (если для данной детали предусмотрен ввод длины и ширины);

— удаление и перемещение деталей LEGO, на которых не стоят другие элементы;

— перемещение, поворот и масштабирование платформы с находящимися на ней деталями;

— добавление и удаление источников света.

Таким образом в интерфейсе должны присутствовать следующие разделы.

1) Модуль для отрисовки сцены:
   - виджет отображения сцены
   - кнопка добавления сцены;
   - окно ввода размеров сцены;
   - кнопка «вид сверху».
2) Модуль для работы со сценой:
   - кнопки перемещения;
   - кнопки поворота;
   - кнопки масштабирования;
3) Модуль для работы с объектами сцены:
   - список доступных объектов (детали LEGO и источник света);
   - кнопки добавления, удаления и перемещения;
   - окна ввода координат и размеров объектов.

3 Технологический раздел

3.1 Выбор языка программирования и среды разработки

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

— C++ — объектно-ориентированный язык, а именно такая методология программирования была выбрана для разработки программы;

— статическая типизация, позволяющая находить большое число ошибок на стадии компиляции;

— изучение курса ООП в 4 семестре и написание лабораторной работы на схожую с текущей задачей тему.

В качестве среды разработки была выбрана программа *QTCreator*, так как в ней есть возможность быстрого создания интерфейса с помощью расширения *QTDesign*.

3.2 Интерфейс пользователя

На рисунках 3.1–3.11 представлен интерфейс окон программы.

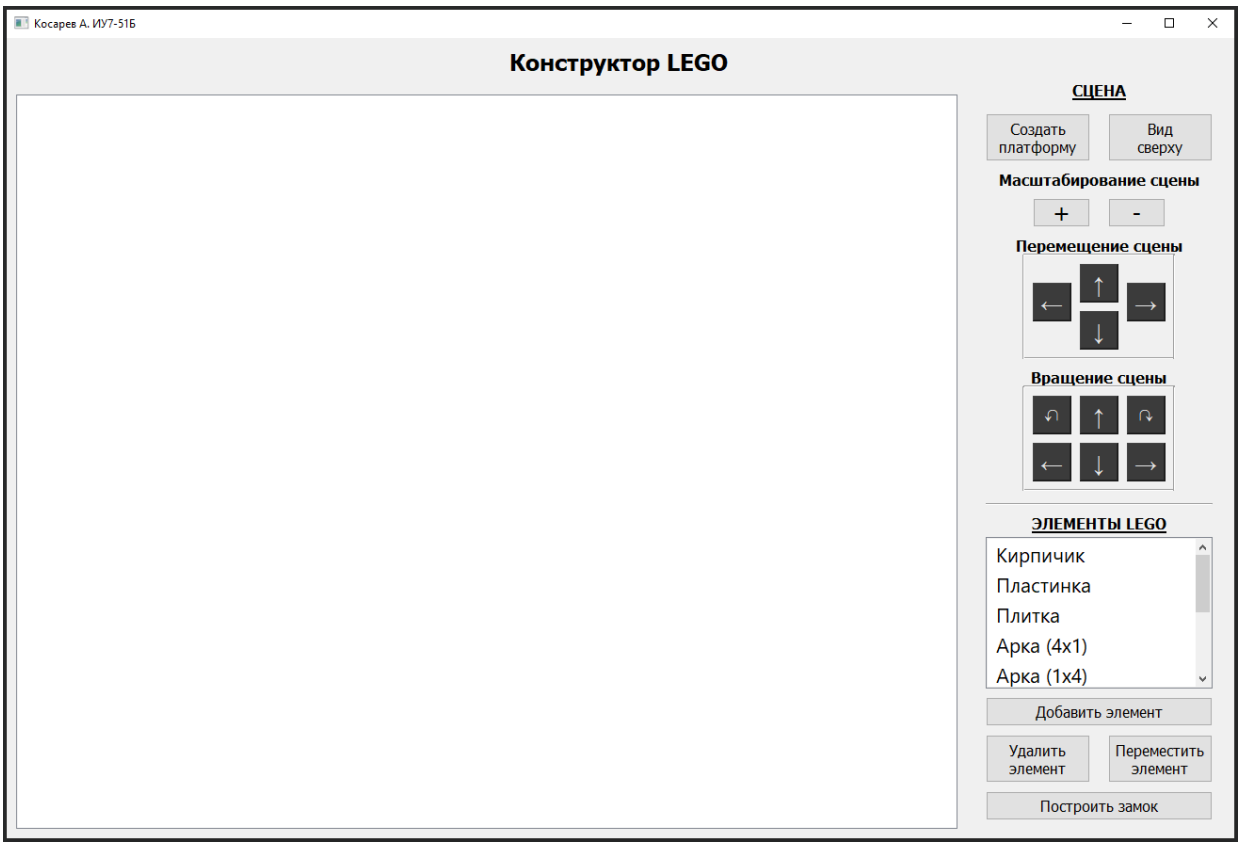

Рисунок 3.1 – Основное окно программы

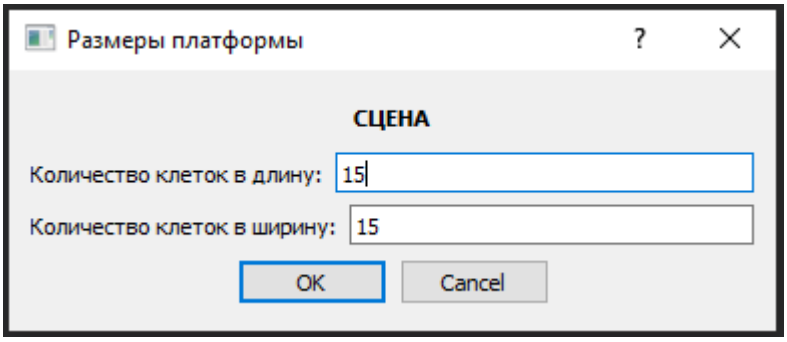

Рисунок 3.2 – Окно добавления платформы

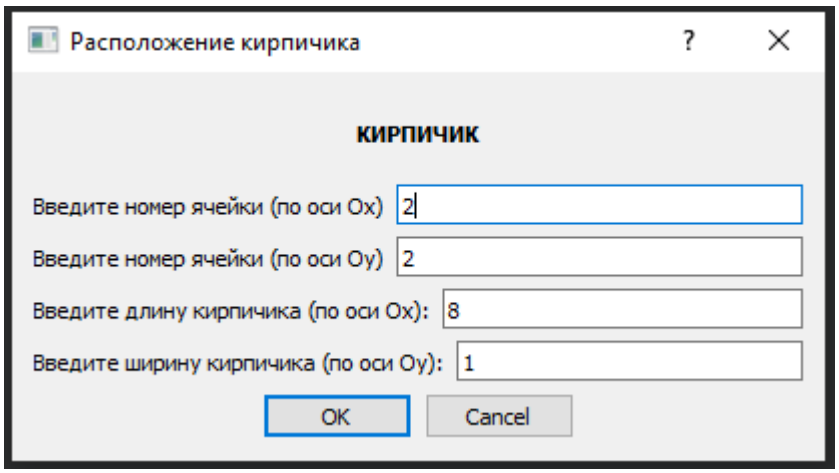

Рисунок 3.3 – Окно добавления кирпичика

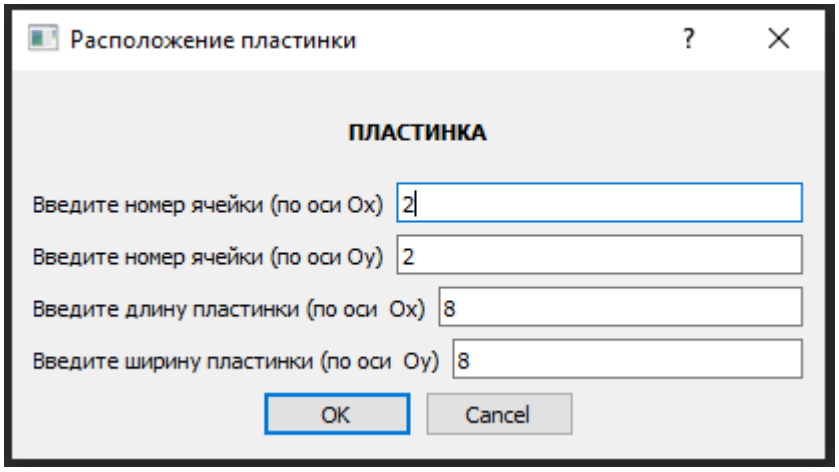

Рисунок 3.4 – Окно добавления пластинки

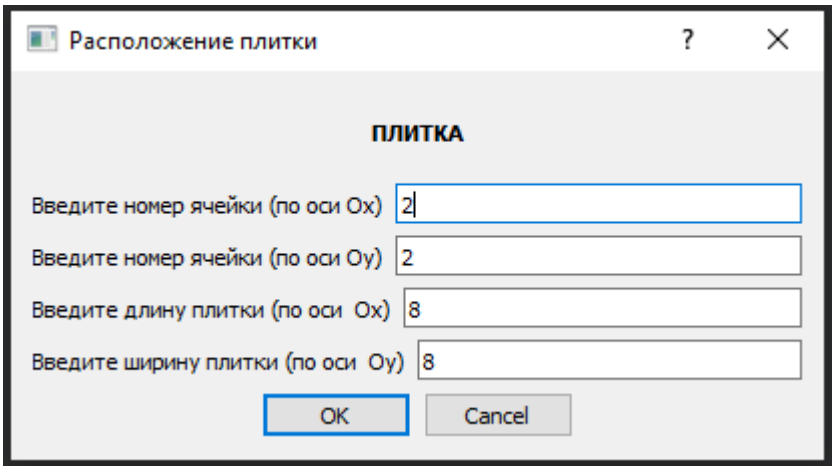

Рисунок 3.5 – Окно добавления плитки

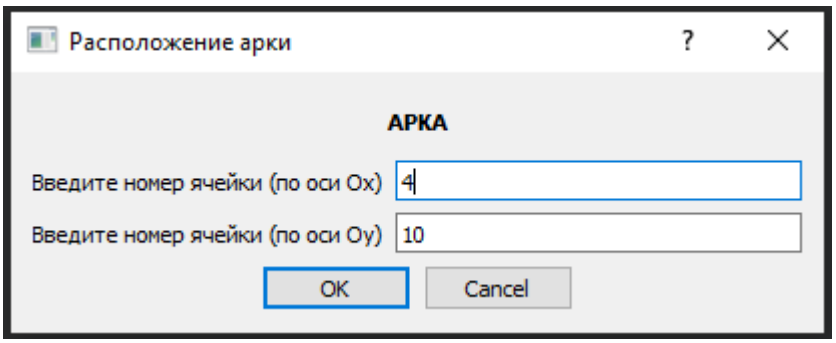

Рисунок 3.6 – Окно добавления арки

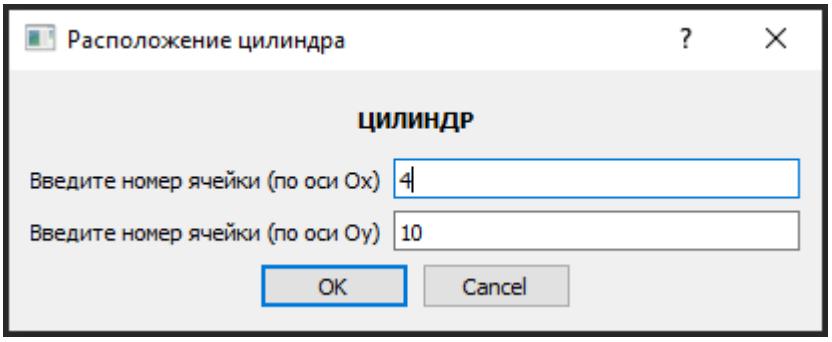

Рисунок 3.7 – Окно добавления цилиндра

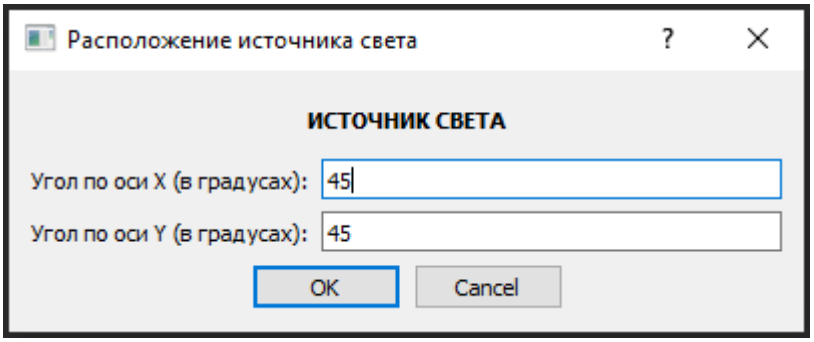

Рисунок 3.8 – Окно добавления источника света

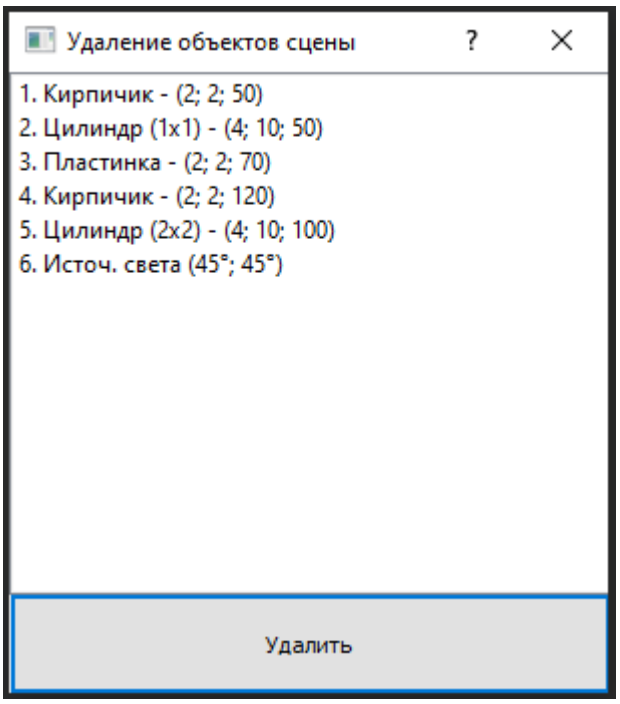

Рисунок 3.9 – Окно удаления деталей

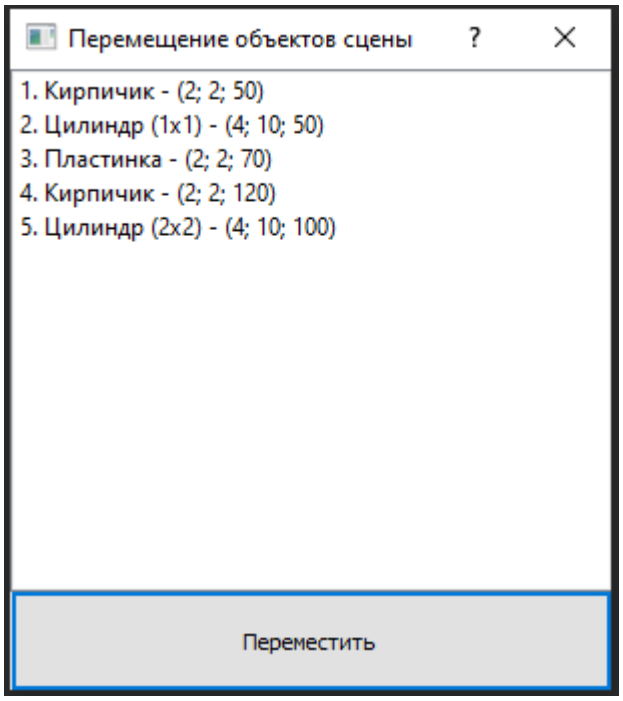

Рисунок 3.10 – Окно перемещения деталей

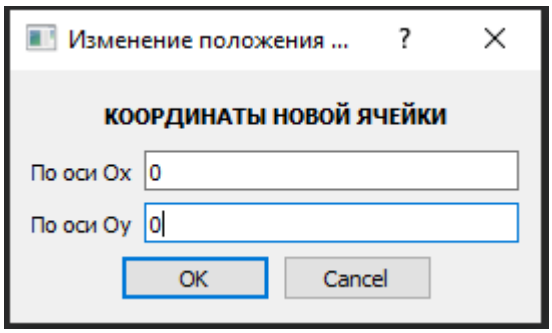

Рисунок 3.11 – Окно ввода новых координат

3.3 Порядок работы

Ниже приведены некоторые важные аспекты работы программы:

— для того чтобы начать строить конструкции из деталей LEGO необходимо создать платформу, задав ее размеры;

— номера ячеек, которые указываются в качестве данных о расположении детали, говорят о том, куда будет помещен верхний левый угол детали (если смотреть относительно начального положения камеры);

— если при добавлении очередной детали LEGO указать такую позицию, при которой она попадает сверху на другую, добавляемая деталь будет поставлена сверху на уже существующую;

— нельзя удалять и перемещать те элементы, над которыми находятся другие детали;

— на деталь «плитка» нельзя ставить блоки LEGO;

— если при добавлении детали было указано такое ее положение или размеры, при котором она вылезает за границы платформы, то эта деталь не будет размещена на сцене.

4 Исследовательский раздел

4.1 Технические характеристики устройства

Ниже представлены характеристики компьютера, на котором проводилось тестирование программы:

— операционная система Windows 10 Домашняя 21H2 [7];

— оперативная память 16 Гб;

— процессор Intel(R) Core(TM) i7-10870H CPU @ 2.20 ГГц [8].

Во время работы ноутбук был подключен к сети электропитания. Из программного обеспечения была запущена среда разработки *QTCreator* [9] и браузер *Chrome*.

Загруженность компонентов:

— процессор – 16 %;

— оперативная память – 60 %.

4.2 Примеры использования программы

В качестве примера работы программы приведен процесс постройки здания (рисунки 4.1–4.4).

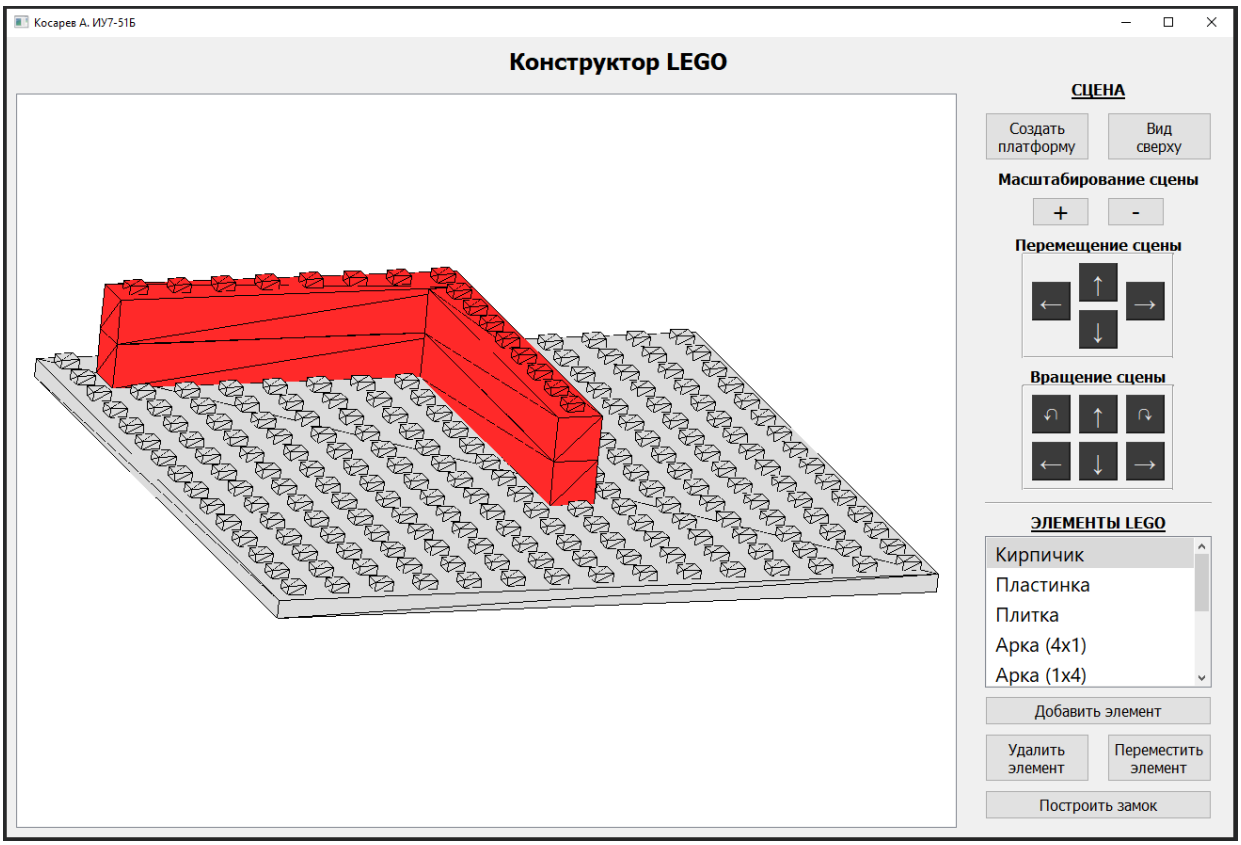

Рисунок 4.1 – Пример работы программы (ч.1)

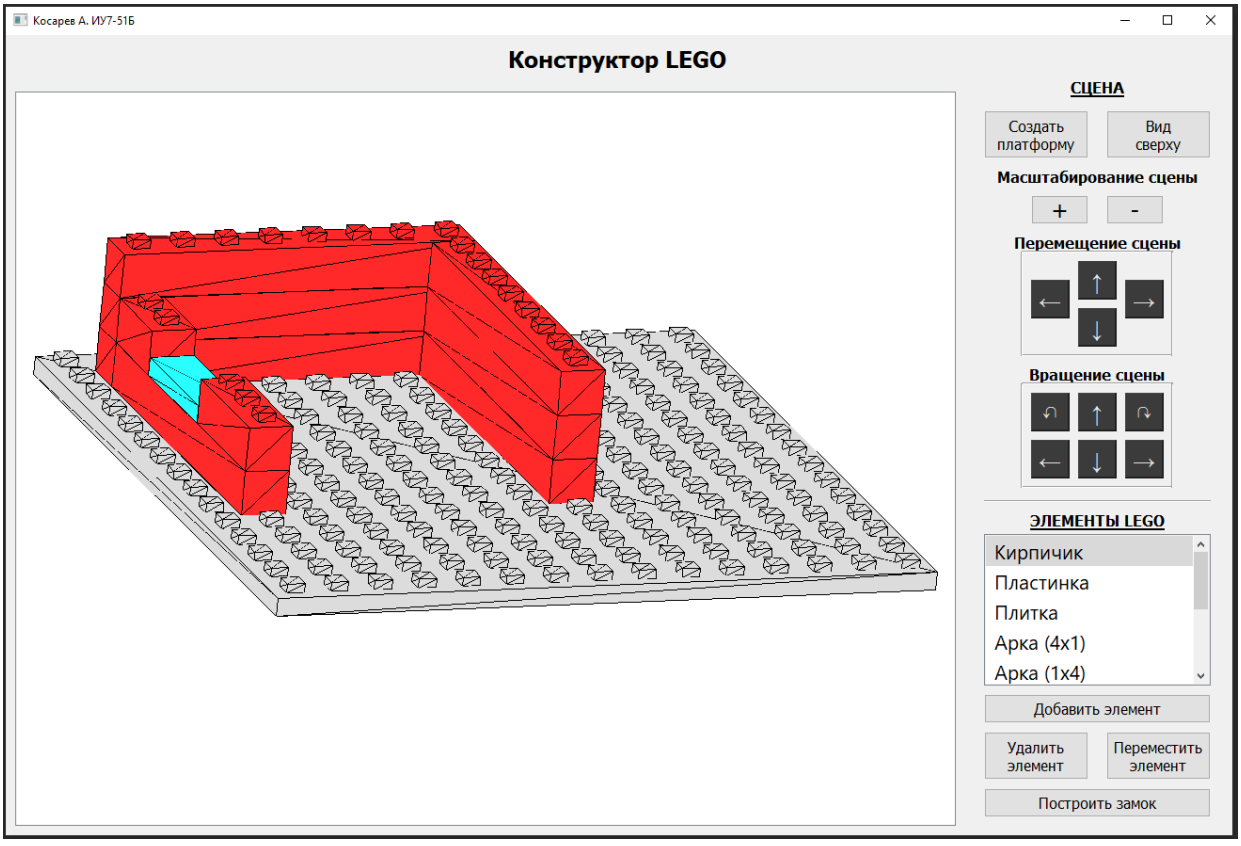

Рисунок 4.2 – Пример работы программы (ч.2)

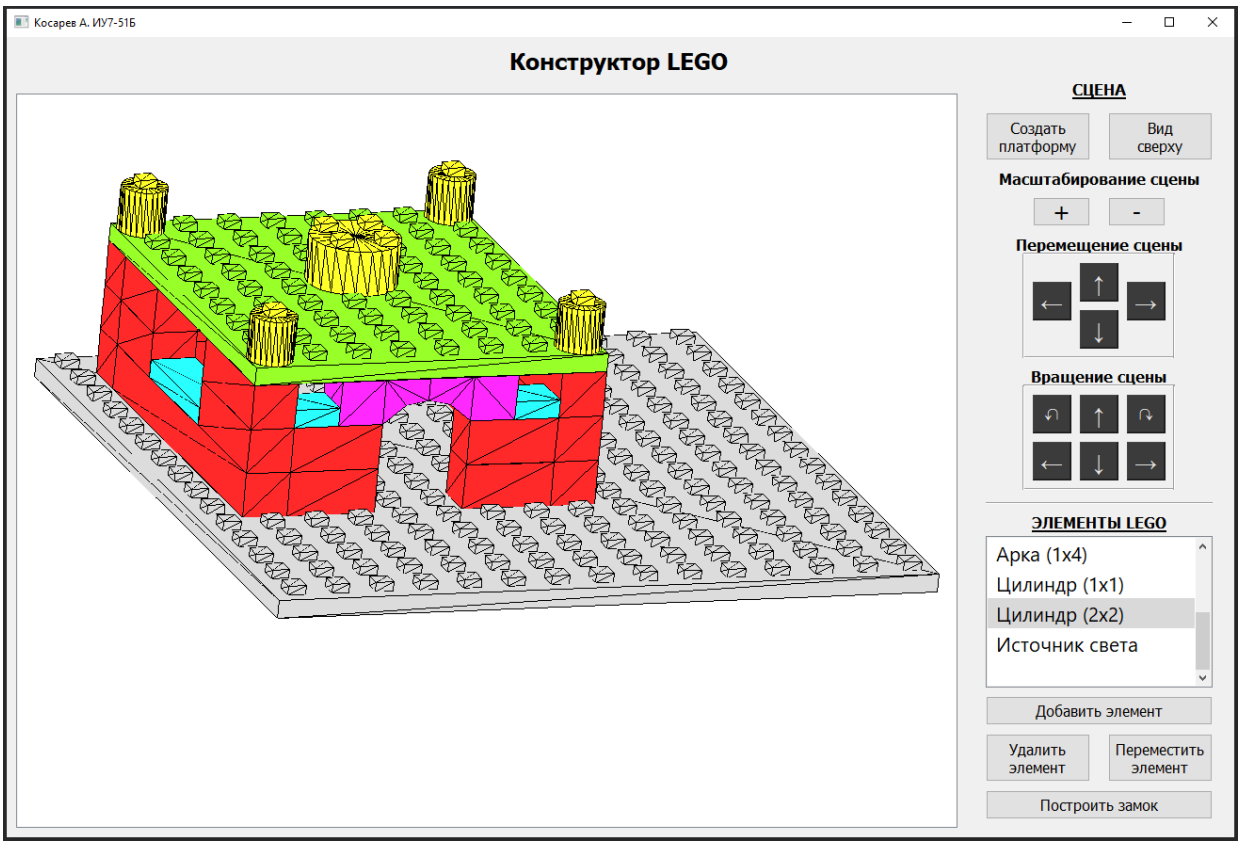

Рисунок 4.3 – Пример работы программы (ч.3)

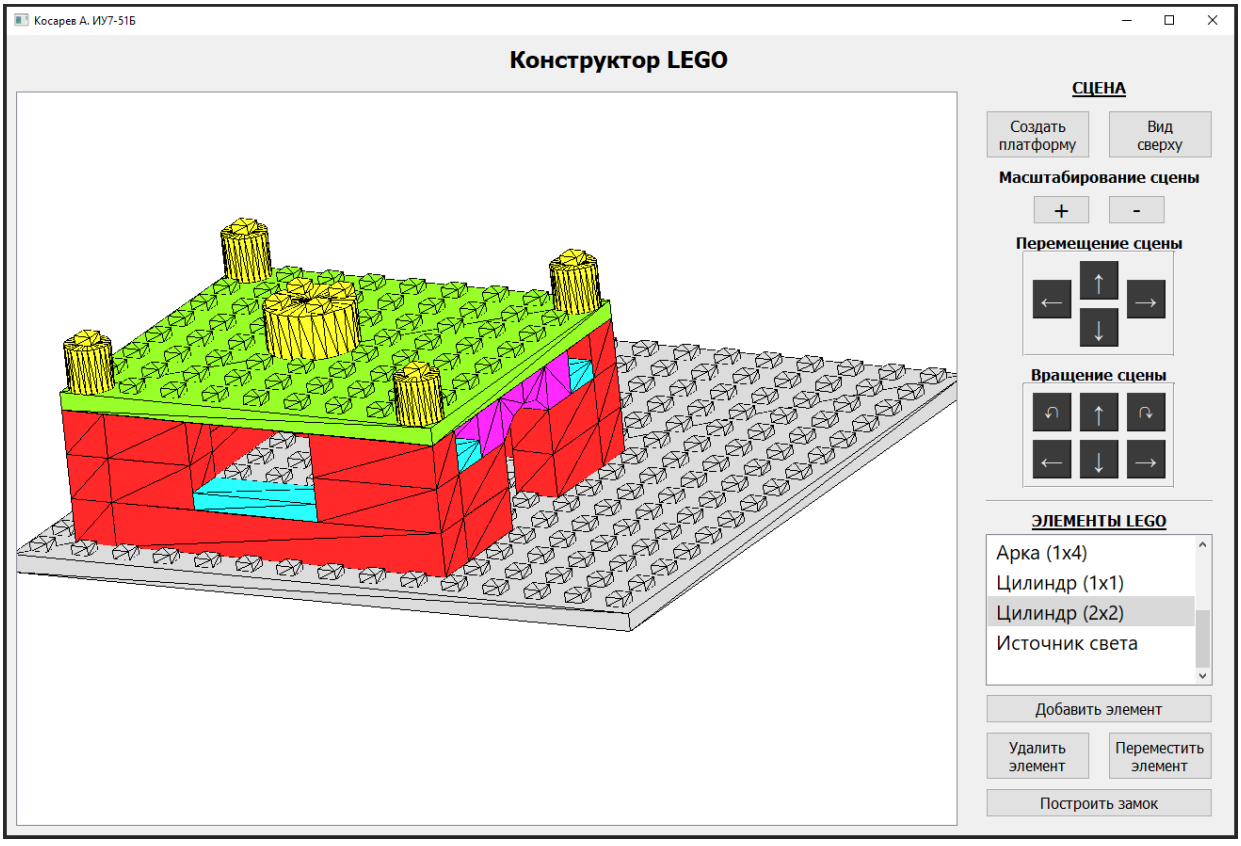

Рисунок 4.4 – Пример работы программы (ч.4)

На рисунках 4.5–4.6 представлена сцена с тем же зданием, но с добавленным источником света.

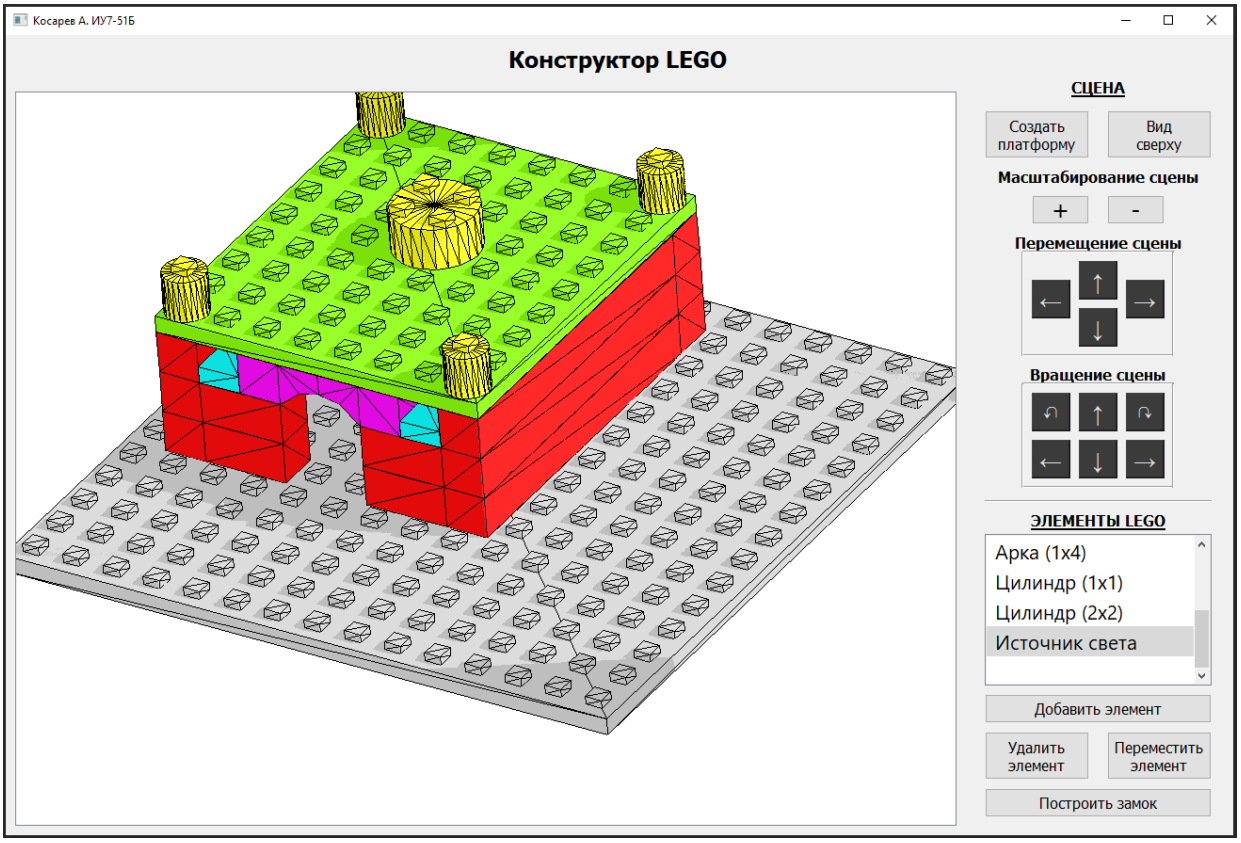

Рисунок 4.5 – Пример работы программы с источником света (ч.1)

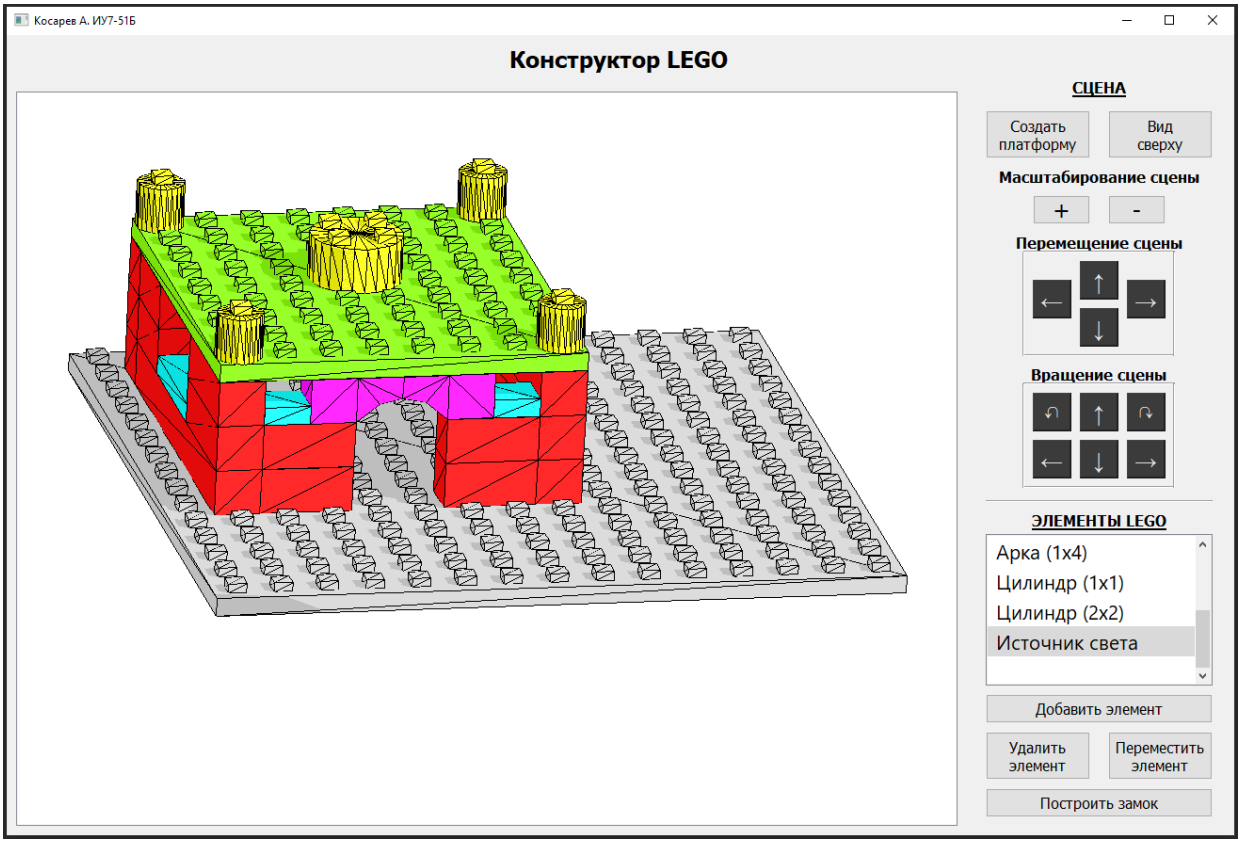

Рисунок 4.6 – Пример работы программы с источником света (ч.2)

4.3 Исследование времени работы алгоритма с z-буфером

В качестве исследования были проведены замеры времени работы обычного алгоритма с z-буфером и его модификации для построения теней в зависимости от количества деталей на платформе. Для сравнения времени работы двух алгоритмов использовались сцены с одинаковым расположением деталей (кирпичиков 2×2).

4.3.1 Замеры времени

Для подсчета затрачиваемого времени работы была использована функция *time_since_epoch()* [5] и библиотека *chrono* [6]. Возвращаемое значение функции *time_since_epoch()* — наносекунды, которое было переведено в миллисекунды.

На рисунках 4.7–4.8 представлены графики зависимости времени работы соответствующих алгоритмов от количества деталей LEGO на сцене, а на рисунке 4.9 — их сравнение.

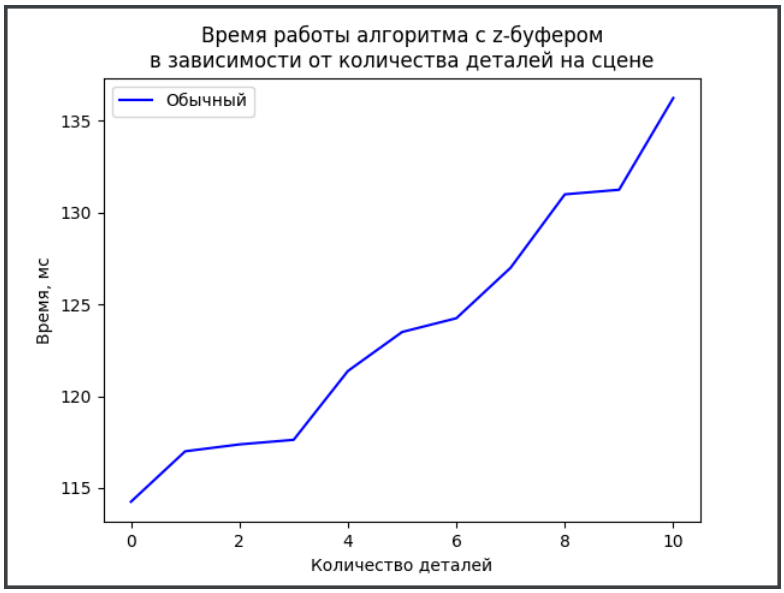

Рисунок 4.7 – Время работы алгоритма с z-буфером

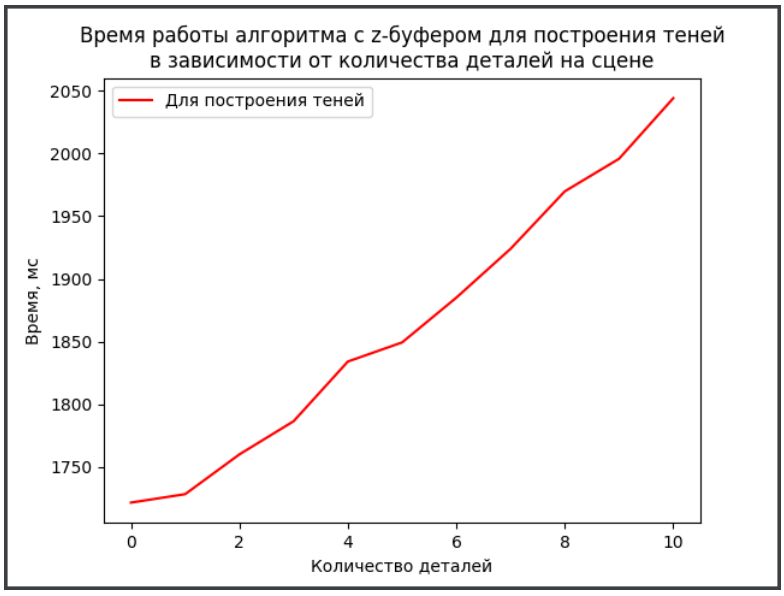

Рисунок 4.8 – Время работы алгоритма с z-буфером для построения теней

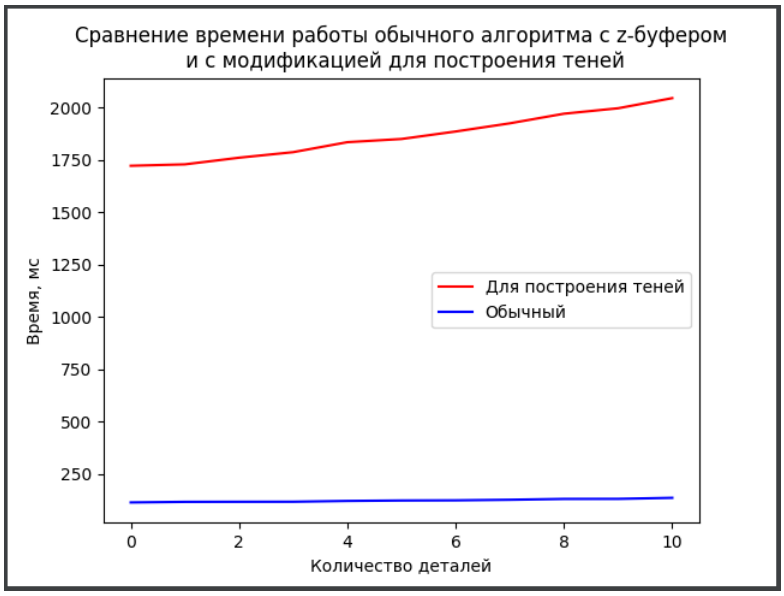

Рисунок 4.9 – Сравнение двух алгоритмов с z-буфером

4.3.2 Вывод

В результате исследования замеров времени работы алгоритмов выделены следующие аспекты:

— алгоритм с z-буфером для построения теней в среднем работает в 15 раз медленнее, чем обычная его версия;

— прирост времени работы алгоритма для построения теней при добавлении очередной детали LEGO больше в 15 раз, чем у алгоритма без модификаций.

ЗАКЛЮЧЕНИЕ

Цель, которая была поставлена в начале курсовой работы, была достигнута: разработана программа моделирования детского конструктора LEGO.

Решены все поставленные задачи:

— изучены модели представления объектов и выбрана подходящая;

— выбраны алгоритмы для отображения объектов на сцене;

— выбран механизм размещения объектов;

— выбран язык программирования и среды разработки;

— реализованы выбранные алгоритмы отображения объектов;

— реализовано программное обеспечение с пользовательским интерфейсом, позволяющее собирать конструкции из блоков LEGO.

В качестве исследования были проведены замеры времени работы обычного алгоритма с z-буфером и его модификации для построения теней в зависимости от количества деталей на платформе. Исходя из результатов, можно сделать следующие выводы:

— алгоритм с z-буфером для построения теней в среднем работает в 15 раз медленнее, чем обычная его версия;

— прирост времени работы алгоритма для построения теней при добавлении очередной детали LEGO больше в 15 раз, чем у алгоритма без модификаций.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роджерс Д., Адамс Дж. Математические основы машинной графики. — М.: Мир, 2001.

2. Роджерс Д. Алгоритмические основы машинной графики. — М.: Мир, 1989.

3. Вельтмандер П.В. Учебное пособие «Основные алгоритмы компьютерной графики». —- Новосибирск:Новосибирский государственный университет, 1997.

4. Programming Languages —- C++ [Электронный ресурс]. – URL: `https://isocpp.org/files/papers/N4860.pdf` (дата обращения: 10.09.2022)

5. time_since_epoch documentation [Электронный ресурс]. – URL: `https://en.cppreference.com/w/cpp/chrono/time_point/time_since_epoch` (дата обращения: 07.10.2022)

6. time_since_epoch documentation [Электронный ресурс]. – URL: `https://en.cppreference.com/w/cpp/chrono` (дата обращения: 07.10.2022)

7. Windows 10 Home [Электронный ресурс]. – URL: `https://www.microsoft.com/en-us/d/windows-10-home/d76qx4bznwk4?activetab=pivot:overviewtab` (дата обращения: 23.11.2022)

8. Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz [Электронный ресурс]. – URL: `https://ark.intel.com/content/www/ru/ru/ark/products/208018/intel-core-i710870h-processor-16m-cache-up-to-5-00-ghz.html` (дата обращения: 23.11.2022)

9. Qt Creator Manual [Электронный ресурс]. – URL: `https://doc.qt.io/qtcreator/` (дата обращения: 23.11.2022)