



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» (ИУ)

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

## **ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

Студент

**Чепиго Дарья Станиславовна**

Группа

**ИУ7-44Б**

Тип практики

**технологическая**

Название предприятия

**НУК ИУ МГТУ им. Н. Э. Баумана**

Студент

\_\_\_\_\_  
*подпись, дата*

**Чепиго Д.С**

*фамилия, и.о.*

Руководитель практики

\_\_\_\_\_  
*подпись, дата*

**Кузов А. В.**

*фамилия, и.о.*

Оценка \_\_\_\_\_

2022 г.

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)

---

---

УТВЕРЖДАЮ  
Заведующий кафедрой ИУ7  
\_\_\_\_\_ Рудаков И. В.  
«30» июня 2022 г.

**З А Д А Н И Е**  
**на прохождение производственной практики**  
**(технологическая практика)**

Студент 2 курса группы ИУ7-44Б

**Чепиги Дарья Станиславовна**

в период

**с 30.06.2022 г. по 20.07.2022 г.**

*Предприятие:*

**НУК ИУ МГТУ им. Н. Э. Баумана**

*Руководитель практики от кафедры:*

**Куров А. В.**

*Задание:*

1. Изучить документацию инструментов для автоматизации и поддержки процессов непрерывной интеграции и непрерывного развертывания программного обеспечения в процессе разработки: Qt Test, Gitlab CI/CD, Docker.
2. Собрать материалы по применению средств и методов автоматизации процессов непрерывной интеграции и непрерывного развертывания программного обеспечения в процессе разработки.
3. Получить практические навыки автоматизации процессов интеграции и развёртывания программного обеспечения в процессе разработки.

Дата выдачи задания «30» июня 2022 г.

Руководитель практики от кафедры \_\_\_\_\_/ **Куров А. В.** /

Студент \_\_\_\_\_/ **Чепиги Д.С.** /

# Содержание

<b>Введение</b>	<b>3</b>
<b>1 Знакомство с документацией</b>	<b>4</b>
1.1 Gitlab CI/CD . . . . .	4
1.2 Docker . . . . .	6
1.3 QT . . . . .	8
1.4 QT Test . . . . .	9
1.5 QT Widgets . . . . .	9
1.6 FFmpeg . . . . .	10
<b>2 Проектирование и реализация ПО</b>	<b>11</b>
2.1 Формат входных и выходных данных и обоснование выбора . .	11
2.2 Реализуемые алгоритмы . . . . .	13
2.2.1 Алгоритм, использующий Z-буфер . . . . .	13
2.2.2 Модель освещения Фонга . . . . .	15
2.3 Сценарий Gitlab CI/CD . . . . .	17
2.4 Docker . . . . .	22
2.5 Модульное тестирование . . . . .	25
2.6 Управление из командной строки . . . . .	26
2.7 Пример работы программы . . . . .	27
<b>Заключение</b>	<b>30</b>
<b>Литература</b>	<b>31</b>

# Введение

Цель практики – изучение и практическое применение инструментов для автоматизации и поддержки процессов непрерывной интеграции и непрерывного развертывания программного обеспечения в процессе разработки.

В рамках реализации проекта должны быть решены следующие задачи:

- изучить документацию Gitlab CI/CD, Docker, Qt, Qt Test, Qt Widgets, FFmpeg;
- создать программу, принимающую данные о сцене и создающую изображение с помощью алгоритма, использующего Z-буфер;
- реализовать управление программой из командной строки;
- создать сценарий `gitlab-ci.yml` автоматизации сборки, тестирования и получения данных будущего исследования;
- выбрать готовые образы `docker` для задач из сценария;
- создать три сцены для демонстрации работоспособности всего конвейера;
- создать модульный тест для демонстрации работоспособности системы;
- создать один сценарий исследования зависимости времени выполнения программы от количества полигонов на сцене.

# 1 Знакомство с документацией

В данном разделе представлены ссылки и краткое содержание документации, которая была изучена в рамках практики.

## 1.1 Gitlab CI/CD

Была изучена официальная документация Gitlab CI/CD [1].

Gitlab CI/CD – это инструмент для разработки программного обеспечения с использованием непрерывных методологий:

- непрерывная интеграция (Continuous Integration);
- непрерывная доставка (Continuous Delivery);
- непрерывное развертывание (Continuous Deployment).

Чтобы использовать GitLab CI/CD необходим код приложения, размещенный в репозитории Git, и файл `.gitlab-ci.yml`, находящийся в корне репозитория и содержащий конфигурацию CI/CD.

Используемые концепции и их описание:

- Файл `.gitlab-ci.yml`

Используя данный файл GitLab Runner запускает конвейеры и сценарии, определенные в заданиях.

Были изучены такие ключевые слова, как:

- `image`(docker образ);
- `stage`(стадии конвейера);
- `before_script`(команды, выполняемые перед основным скриптом);
- `script`(команды основного скрипта);
- `artifacts`(данные, полученные в результате заданий);
- `needs`(зависимости между заданиями).

### Листинг 1.1 – Пример файла .gitlab-ci.yml

```
1 graph:
2   stage: result_graph
3   image: python
4   before_script:
5     - pip install matplotlib
6   script:
7     - python3 result_graph.py
8   artifacts:
9     paths:
10    - chicago_develop/result.png
11    expire_in: 1 day
12  needs:
13    - research
```

- Конвейеры(pipelines)

Конвейеры или сборочные линии содержат стадии(stages), которые определяют задания и когда их следует запускать. Различают стадии сборки, тестирования, организации и выпуска.

- Задания(jobs)

Настройка конвейера начинается с заданий. Задания являются фундаментальным элементом файла gitlab-ci.yml.

Задания:

- определены с ограничениями, указывающими, при каких условиях они должны выполняться;
- элементы верхнего уровня с произвольным именем должны содержать секцию script;
- не ограничены в количестве.

### Листинг 1.2 – Пример задания

```
1 debug-build:
2   stage: build_debug
3   script: make
```

- Артефакты задания(job artifacts)

Артефакты задания – архив файлов и каталогов, которые могут выпускать задания. Их можно загружать с помощью пользовательского интерфейса GitLab или API.

Листинг 1.3 – Пример создания артефактов задания

```
1 generate-image:
2 script: ./app.exe -image
3 artifacts:
4   paths:
5   - /pictures/image.png
6   expire_in: 7 days
```

В этом примере задание с именем generate-image запускает приложение app.exe, которое генерирует изображение. Ключевое слово paths определяет, какие файлы следует добавлять в артефакты задания. Все пути к файлам и каталогам относятся к репозиторию, в котором было создано задание. Ключевое слово expire\_in определяет, как долго GitLab хранит артефакты задания.

## 1.2 Docker

Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации.

Была изучена официальная документация Docker [2]:

- Создание Dockerfile

Листинг 1.4 – Dockerfile

```
1 # syntax=docker/dockerfile:1
2 FROM python:3.7 - alpine
3 WORKDIR /code
4 ENV FLASK_APP=app.py
5 ENV FLASK_RUN_HOST=0.0.0.0
6 RUN apk add --no-cache gcc musl-dev linux-headers
7 COPY requirements.txt requirements.txt
8 RUN pip install -r requirements.txt
9 EXPOSE 5000
10 COPY . .
11 CMD ["flask", "run"]
```

В этом примере докер:

- создаёт образ, начиная с образа Python 3.7;
- устанавливает рабочий каталог/code;
- задаёт переменные среды, используемые командой flask;
- устанавливает gcc и другие зависимости;
- копирует requirements.txt и устанавливает зависимости Python;
- добавляет метаданные к образу, чтобы описать, что контейнер прослушивает порт 5000;
- копирует текущий каталог . в проекте в рабочий каталог . в образе;
- устанавливает для контейнера команду по умолчанию flask run.

- Определение служб в Compose file

Листинг 1.5 – Compose file

```
1 version: "3.9"
2 services:
3   web:
4     build: .
5     ports:
6       - "8000:5000"
7   redis:
8     image: "redis:alpine"
```

Compose file определяет две службы: web и redits. Веб-служба использует образ, созданный из файла Dockerfile в текущем каталоге. Затем он привязывает контейнер и хост-машину к открытому порту 8000. В этом примере службы используется порт по умолчанию для веб-сервера Flask 5000. Служба redis использует общедоступный образ Redis, извлеченный из реестра Docker Hub.

- Сборка и запуск приложения с помощью Compose

Из каталога проекта было запущено приложение с помощью docker compose up.

- Docker Hub

Были изучены создание, удаление, клонирование репозиторий в Docker Hub [3].



## 1.3 QT

Были изучены интерфейсы основных классов библиотеки Qt [4], необходимые для работы с компьютерной графикой:

- QImage

Класс QImage обеспечивает аппаратно-независимое представление изображения. QImage разработан и оптимизирован для ввода-вывода, а также для прямого доступа к пикселям и манипулирования ими. Также позволяет сохранять изображения в формате PNG.

- QPixmap

Класс QPixmap разработан и оптимизирован для отображения изображения на экране, которое можно использовать в качестве устройства рисования.

- QVector3D

Класс QVector3D представляет вектор или вершину в трехмерном пространстве.

Помимо конструкторов создания были изучены и использованы методы:

- `dotProduct(QVector3D v1, QVector3D v2)` – скалярное произведение векторов `v1` и `v2`;
- `normalize()` – нормирование вектора;
- `toVector4D()` – четырехмерная форма трехмерного вектора с нулевой координатой `w`;
- `setX(float x)` – установка координаты `x`;
- `setY(float y)` – установка координаты `y`;
- `setZ(float z)` – установка координаты `z`.

- QVector4D

Класс QVector4D представляет вектор или вершину в четырехмерном пространстве. Был использован для матриц преобразований в трехмерном пространстве.

- QColor

Класс QColor предоставляет цвета на основе значений RGB, HSV или CMYK. QColor был использован в программе для создания и изменения цвета на основе значений RGB.

## 1.4 QT Test

Qt Test [5] – это фреймворк для модульного тестирования приложений и библиотек на основе Qt. Qt Test предоставляет расширения для тестирования графических пользовательских интерфейсов.

Были изучен и использован макрос QCOMPARE(actual, expected). Он сравнивает фактическое значение с ожидаемым значением с помощью оператора равенства. Если фактическое и ожидаемое совпадают – выполнение продолжается. Иначе ошибка записывается в журнал тестирования, и тестовая функция возвращается без попыток каких-либо последующих проверок.

## 1.5 QT Widgets

Qt Widgets [6] предоставляет набор элементов пользовательского интерфейса для создания классических пользовательских интерфейсов.

Были изучены основные классы модуля:

- QApplication

Класс QApplication управляет потоком управления и основными настройками приложения с графическим интерфейсом. Для любого приложения с графическим интерфейсом, использующего Qt, существует ровно один объект QApplication, независимо от того, имеет ли приложение 0, 1, 2 или более окон в любой момент времени.

- QMainWindow

Класс QMainWindow предоставляет главное окно приложения. Главное окно обеспечивает основу для создания пользовательского интерфейса приложения.

- QGraphicsScene

Класс QGraphicsScene предоставляет поверхность для управления большим количеством 2D-графических элементов. Он используется вместе с QGraphicsView для визуализации графических элементов, таких как линии, прямоугольники, текст или даже пользовательские элементы, на 2D-поверхности.

- QGraphicsView

Класс QGraphicsView предоставляет виджет для отображения содержимого. QGraphicsView визуализирует содержимое QGraphicsScene в прокручиваемом окне просмотра.

- QPushButton

Виджет, представляющий обычную кнопку. Если произошло событие clicked, управление передаётся обработчику этого события.

## 1.6 FFmpeg

Была изучена официальная документация утилиты FFmpeg [7]. Она позволяет записывать, конвертировать и передавать цифровые аудио- и видео-записи в различных форматах.

Листинг 1.6 – Пример создания MP4 видеофильма из PNG изображений

```
1 ffmpeg -r 5 -f image2 -s 1920x1080 -i film%d.png -crf 1 -pix_fmt yuv420p  
   film.mp4
```

- -r – установка частоты кадров в секунду;
- -f – установка имени файла форматом film%02d.png;
- -s – установка размера кадра;
- -i – ввод имени входного файла;
- -crf – фактор постоянного оценивания (Constant Rate Factor);
- -pix\_fmt – установка формата пикселей.

## 2 Проектирование и реализация ПО

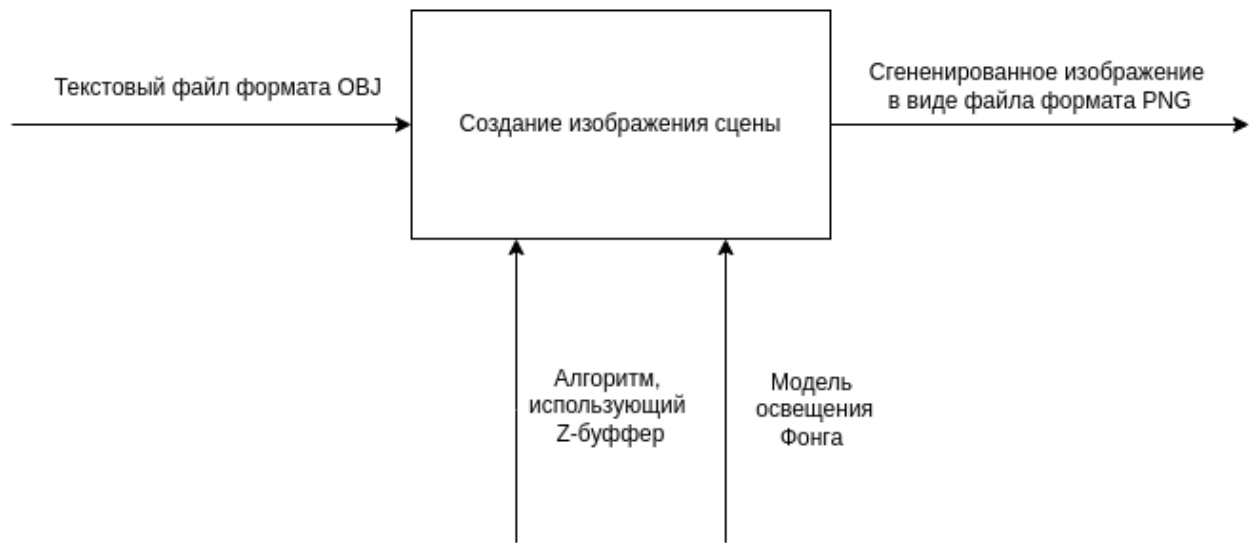


Рисунок 2.1 – IDEF0 диаграмма работы программы

### 2.1 Формат входных и выходных данных и обоснование выбора

Для представления входных данных был выбран текстовый файл формата OBJ – формат описания геометрии.

Преимущества формата:

- открытый формат файла, что позволяет его редактировать, в отличие от бинарных форматов;
- занимает первое место по количеству моделей в рейтинге, указанным в статье [8];
- не привязан к определенной программе, поэтому может быть экспортирован/импортирован во многие программы.

## Листинг 2.1 – Пример .obj файла

```
1 # comment
2 o figure
3
4 v 10 20 30
5 v 40 50 100
6 v -10 20 40
7
8 f 1 2 3
9
10 Ka 255 200 180
```

- # – комментарий
- o – название фигуры
- v – параметры вершины
- f – определение поверхности
- Ka – внешний цвет поверхности(RGB)

Также в файле могут быть заданы vn(нормали), vt(текстуры вершины), g(группировка объектов), usemtl(параметры материала, цветовых характеристик и прозрачности).

Из всех возможных форматов изображений для выходных данных (BMP, GIF, JPG, JPEG, PNG, PBM, PGM, PPM, XBM, XPM), которая предоставляет библиотека Qt, был выбран формат PNG, так как:

- формат PNG – графический, что необходимо для создания фильма с помощью утилиты FFmpeg [6];
- формат PNG является платформонезависимым, в отличие от формата BMP;
- изображения в формате PNG более качественные, чем изображения в формате JPG по метрике типа PSN [9].

## 2.2 Реализуемые алгоритмы

### 2.2.1 Алгоритм, использующий Z-буфер

Данный алгоритм работает в пространстве изображения [10]. Используется два буфера:

- буфер кадра, в котором хранятся атрибуты каждого пикселя в пространстве изображения;
- z-буфер, куда помещается информация о координате  $z$  для каждого пикселя.

Первоначально в z-буфере находятся минимально возможные значения  $z$ , а в буфере кадра располагаются пиксели, описывающие фон. Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра. В процессе подсчета глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий, то он заносится в буфер кадра и происходит корректировка z-буфера.

Для решения задачи вычисления глубины  $z$  каждый многоугольник описывается уравнением  $ax + by + zc + d = 0$ . При  $c = 0$  многоугольник для наблюдателя вырождается в линию.

В программе была использована модификация указанного алгоритма путем добавления вычисления теневого z-буфера из точки наблюдения, совпадающей с источником света.

На рисунке 2.2 изображена схема модифицированного алгоритма z-буфера.

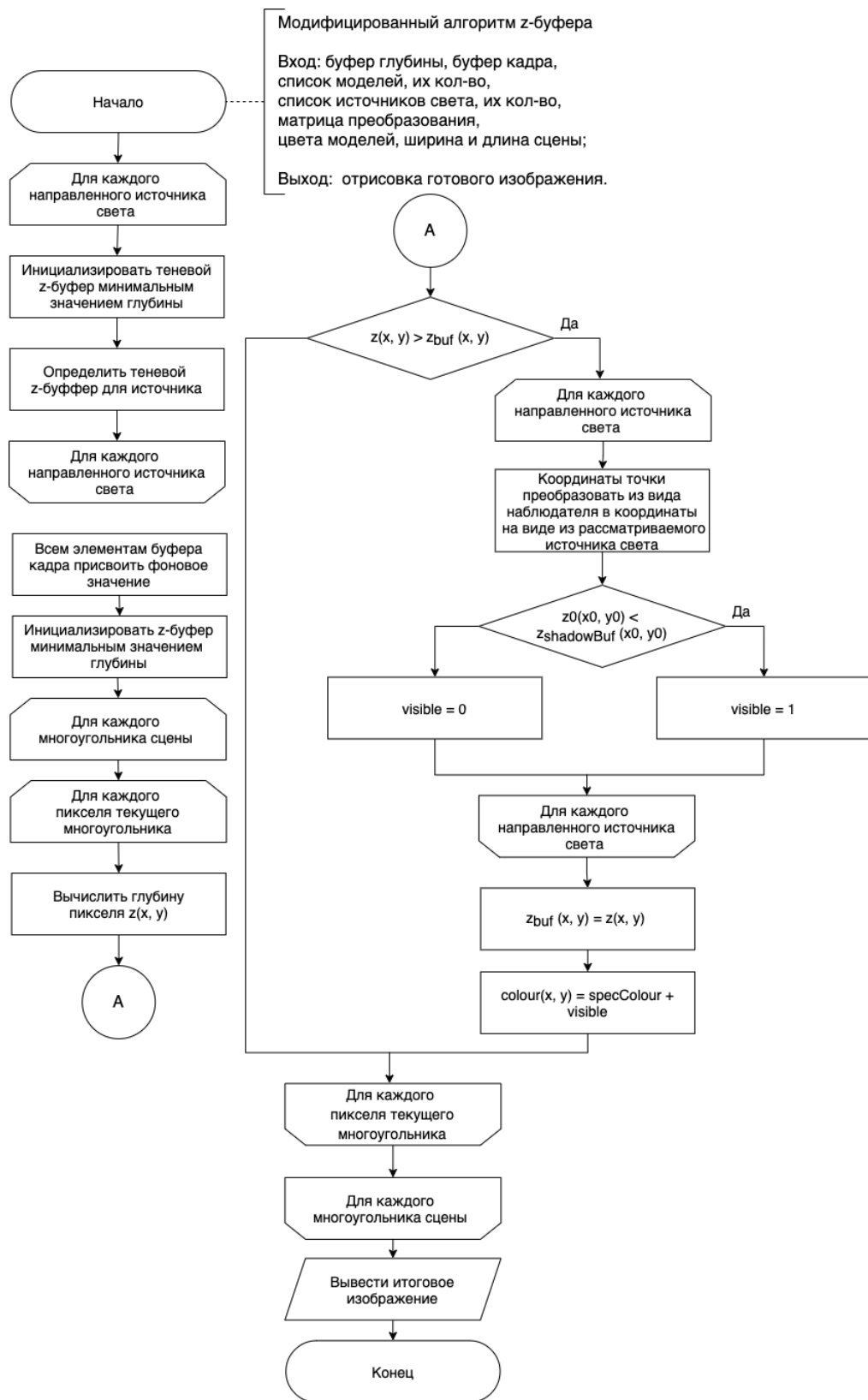


Рисунок 2.2 – модифицированный алгоритма z-буфера

## 2.2.2 Модель освещения Фонга

Модель Фонга – классическая модель освещения [11]. Пусть заданы точечный источник света, расположенный в некоторой точке, поверхность, которая будет освещаться и наблюдатель. Каждая точка поверхности имеет свои координаты и в ней определена нормаль к поверхности. Её освещенность складывается из трех компонент:

- фоновое освещение (ambient);
- рассеянный свет (diffuse);
- бликовая составляющая (specular).

Фоновое освещение это постоянная в каждой точке величина надбавки к освещению. Вычисляется фоновая составляющая освещения по формуле 2.1:

$$I_a = k_a i_a \quad (2.1)$$

$I_a$  - фоновая составляющая освещенности в точке

$k_a$  – свойство материала воспринимать фоновое освещение

$i_a$  – мощность фонового освещения

Рассеянный свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только нормаль к поверхности и направление на источник света. Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта):

$$I_d = k_d \cos(\vec{L}, \vec{N}) i_d = k_d (\vec{L} \cdot \vec{N}) i_d, \quad (2.2)$$

$I_d$  – рассеянная составляющая освещенности в точке

$k_d$  – свойство материала воспринимать рассеянное освещение

$i_d$  – мощность рассеянного освещения

$\vec{L}$  – направление из точки на источник

$\vec{N}$  - вектор нормали в точке.



Зеркальный свет при попадании на поверхность подчиняется следующему закону: «Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части». То есть отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча. Это можно выразить формулой 2.3:

$$I_s = k_s \cos^\alpha (\vec{R}, \vec{V}) i_s = k_s (\vec{R} \cdot \vec{V})^\alpha i_s, \quad (2.3)$$

$I_s$  – зеркальная составляющая освещенности в точке,

$k_s$  – коэффициент зеркального отражения,

$i_d$  – мощность зеркального освещения,

$\vec{R}$  – направление отраженного луча,

$\vec{V}$  – направление на наблюдателя,

$\alpha$  – коэффициент блеска, свойство материала.

Тогда интенсивность света подсчитывается формулой 2.4:

$$I_s = I_a + I_d + I_s = k_a i_a + k_d (\vec{L} \cdot \vec{N}) i_d + k_s (\vec{R} \cdot \vec{V})^\alpha i_s \quad (2.4)$$

Пример работы модели Фонга:

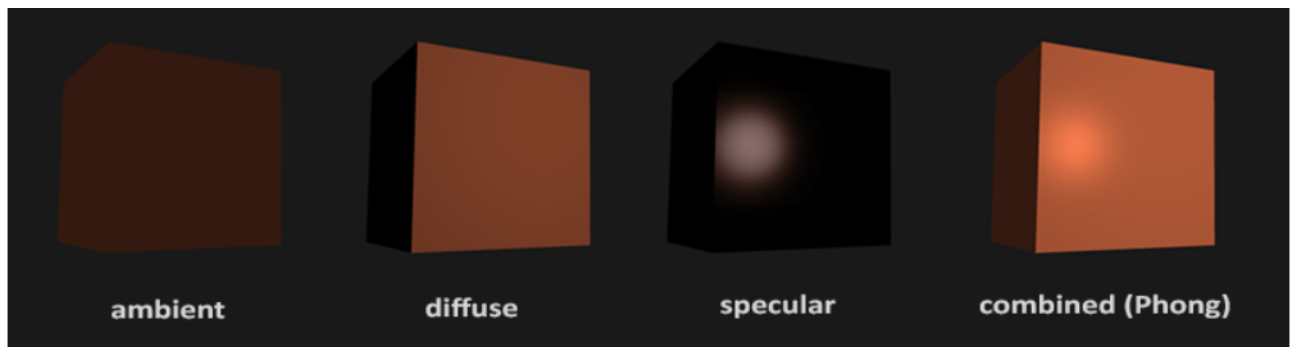


Рисунок 2.3 – Модель освещения Фонга

## 2.3 Сценарий Gitlab CI/CD

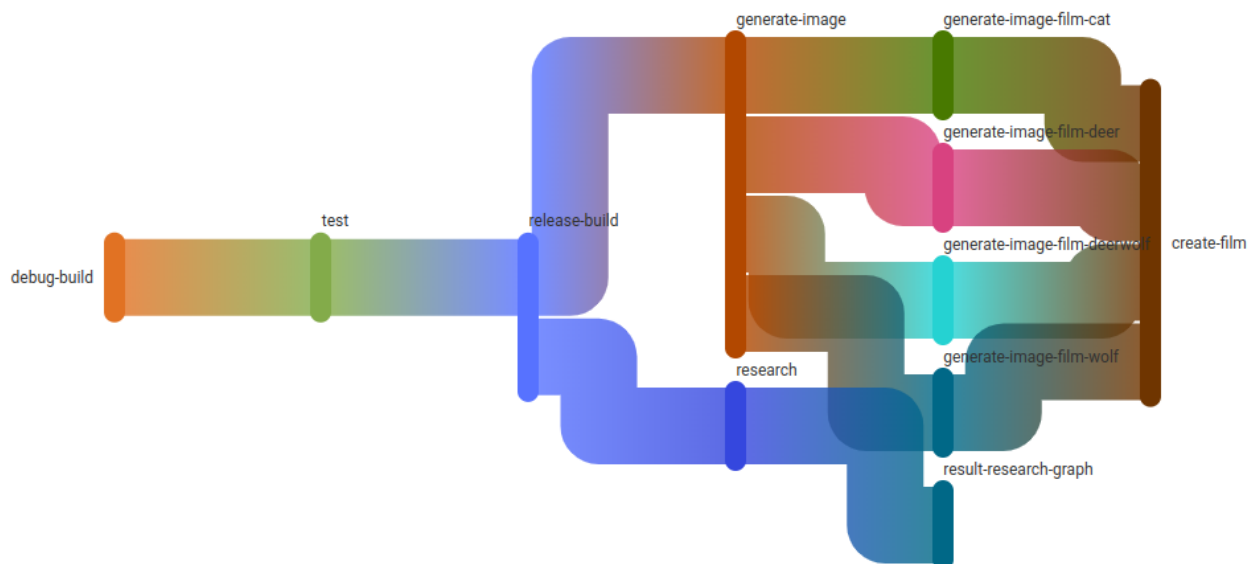


Рисунок 2.4 – Сценарий Gitlab CI/CD

Сценарий состоит из семи стадий:

- **build\_debug** – сборка отладочной версии программы.
  - **debug-build** – задание команд для сборки отладочной версии программы.
- **testing** – модульное тестирование программы.
  - **unit-test** – модульное тестирование.
- **build\_release** – сборка версии программы на выпуск.
  - **release-build** – задание команд для сборки версии программы на выпуск
- **generate\_image** – получение изображения сцены.
  - **generate-image** – получение изображении указанной сцены.

- `generate_image_film` – генерация нескольких изображений изменяемой сцены.
  - `generate-image-film-deer`
  - `generate-image-film-catwolf`
  - `generate-image-film-trees`
- `create_film` – создание фильма из полученных изображений сцен на предыдущей стадии.
  - `create-film` – задание команд для создания фильма на основе изображений из прошлой стадии.
- `research` – исследование временных характеристик программы.
  - `research` – замер времени работы программы при разном количестве полигонов на сцене.
  - `result-research-graph` – преобразование данных и построение графика зависимости времени работы программы от количества полигонов на сцене.

## Листинг 2.2 – `.gitlab-ci.yml` файл

```

1 image: darkmattercoder/qt-build:latest
2
3 stages:
4   - build_debug
5   - testing
6   - build_release
7   - generate_image
8   - generate_image_film
9   - create_film
10  - research
11
12 debug-build:
13   stage: build_debug
14   script:
15     - cd chicago_develop
16     - qmake -project QT+=widgets QT+=testlib chicago_develop.pro
17     - qmake
18     - make
19   artifacts:
20     paths:

```

```

21     - chicago_develop/chicago_develop
22     expire_in: 1 day
23
24 unit-test:
25     stage: testing
26     script:
27         - export QT_QPA_PLATFORM=offscreen
28         - cd chicago_develop
29         - ./chicago_develop -test
30     needs:
31         - debug-build
32
33 release-build:
34     stage: build_release
35     script:
36         - cd chicago_develop
37         - qmake -project QT+=widgets QT+=testlib chicago_develop.pro
38         - qmake
39         - make
40     artifacts:
41         paths:
42             - chicago_develop/chicago_develop
43         expire_in: 1 day
44     needs:
45         - unit-test
46
47
48 generate-image:
49     stage: generate_image
50     script:
51         - export QT_QPA_PLATFORM=offscreen
52         - cd chicago_develop
53         - ./chicago_develop -scene ./scenes/scene.obj -image deer.png
54     artifacts:
55         paths:
56             - chicago_develop/deer.png
57         expire_in: 7 days
58     needs:
59         - release-build
60
61
62 generate-image-film-deer:
63     stage: generate_image_film
64     script:
65         - export QT_QPA_PLATFORM=offscreen
66         - cd chicago_develop
67         - mkdir film
68         - ./chicago_develop -scene ./scenes/deer.obj -film 0

```

```

69     artifacts:
70         paths:
71             - chicago_develop/film/film*.png
72         expire_in: 7 days
73     needs:
74         - release-build
75         - generate-image
76
77 generate-image-film-catwolf:
78     stage: generate_image_film
79     script:
80         - export QT_QPA_PLATFORM=offscreen
81         - cd chicago_develop
82         - mkdir film
83         - ./chicago_develop -scene ./scenes/catwolf.obj -film 20
84     artifacts:
85         paths:
86             - chicago_develop/film/film*.png
87         expire_in: 7 days
88     needs:
89         - release-build
90         - generate-image
91
92 generate-image-film-trees:
93     stage: generate_image_film
94     script:
95         - export QT_QPA_PLATFORM=offscreen
96         - cd chicago_develop
97         - mkdir film
98         - ./chicago_develop -scene ./scenes/trees.obj -film 40
99     artifacts:
100         paths:
101             - chicago_develop/film/film*.png
102         expire_in: 7 days
103     needs:
104         - release-build
105         - generate-image
106
107
108 create-film:
109     image:
110         name: jrottenberg/ffmpeg
111         entrypoint: [""]
112     stage: create_film
113     script:
114         - cd chicago_develop/film
115         - ffmpeg -r 5 -f image2 -s 1920x1080 -i film%d.png -crf 1 -pix_fmt yuv420p
           film.mp4

```

```
116 artifacts:
117     paths:
118         - chicago_develop/film/film.mp4
119     expire_in: 7 days
120 needs:
121     - release-build
122     - generate-image
123     - generate-image-film-deer
124     - generate-image-film-catwolf
125     - generate-image-film-trees
126
127 research-time:
128     stage: research
129     script:
130         - cd chicago_develop
131         - export QT_QPA_PLATFORM=offscreen
132         - bash research.sh
133     artifacts:
134         paths:
135             - chicago_develop/result.txt
136         expire_in: 1 day
137     needs:
138         - release-build
139
140 result-research-graph:
141     stage: research
142     image: python
143     before_script:
144         - pip install matplotlib
145     script:
146         - cd chicago_develop
147         - python3 result_graph.py
148     artifacts:
149         paths:
150             - chicago_develop/result.png
151         expire_in: 1 day
152     needs:
153         - research-time
```

## 2.4 Docker

Для различных заданий сценария Gitlab CI/CD были использованы различные образы Docker с DockerHub [3]:

- darkmattercoder/qt-build:latest

Окружение сборки и запуска приложения QT. Тег latest предоставляет последнюю выпущенную qt версию для среды сборки.

Стадии, использующие этот образ: все задания стадии build\_debug, testing, build\_release, generate\_image, generate\_image\_film, задание research-time стадии research

Зависимости этого образа:

- ca-certificates;
- sudo (для возможности изменять контейнер от имени администратора);
- curl;
- python
- gperf;
- bison;
- flex;
- build-essential;
- pkg-config;
- libgl1-mesa-dev;
- libicu-dev;
- firebird-dev;
- libmysqlclient-dev;
- libpq-dev;
- bc;
- git;

- зависимости `xcb`;
- `bash`;
- `libdbus-1-dev` (для Qt версии 5.14.0 и выше);
- `libnss3-dev` (для Qt версии 5.14.0 и выше).

- `jrottenberg/ffmpeg`

Минималистичный контейнер Docker с утилитой FFmpeg, которая компилируется из файлов исходного кода.

Стадии, использующие этот образ: `create_film`.

Зависимости этого образа:

- `automake`;
- `bzip2`;
- `cmake`;
- `diffutils`;
- `expat-devel`;
- `gcc`;
- `git`;
- `gperf`;
- `libtool`;
- `make`;
- `nasm`;
- `perl`;
- `python3`;
- `openssl-devel`;
- `tar`;
- `yasm`;
- `which`;
- `zlib-devel`.



- python

Официальный образ Docker, поддерживаемый одноименным сообществом. Содержит только минимальное количество пакетов, необходимых для запуска python.

Стадии, использующие этот образ: задание result-research-graph стадии research.

Зависимости этого образа:

- dpkg-dev;
- gcc;
- gnupg dirmngr;
- libbluetooth-dev;
- libbz2-dev;
- libc6-dev;
- libexpat1-dev;
- libffi-dev;
- libgdbm-dev;
- liblzma-dev;
- libncursesw5-dev;
- libreadline-dev;
- libsqlite3-dev;
- libssl-dev;
- make;
- tk-dev;
- uuid-dev;
- wget;
- xz-utils;
- zlib1g-dev.

## 2.5 Модульное тестирование

Для модульного тестирования был использован фреймворк QT Test, описанный в пункте 1.4.

Чтобы создать тест, необходимо создать подкласс QObject с названием Test\$, где \$- имя тестируемого класса.

Листинг 2.3 – Создание класса TestTriangle

```
1 class TestTriangle : public QObject
2 {
3     Q_OBJECT
4
5     public:
6         explicit TestTriangle(QObject *parent = nullptr);
7
8     private slots:
9         void test_findZ();
10 };
```

В приватных слотах класса создаются тесты, которые используют макросы фреймворка QT Test.

Листинг 2.4 – Модульный тест

```
1 void TestTriangle::test_findZ()
2 {
3     Triangle polygon(QVector3D(0, 0, 10), QVector3D(10, 10, 10), QVector3D(10,
4         0, 10));
5
6     double z = polygon.findZ(QVector3D(6, 6, 0));
7
8     QCOMPARE(z, 10);
9 }
```

Чтобы добавить модульный тест, необходимо создать ещё один приватный слот в классе Test\$.

## 2.6 Управление из командной строки

Для автоматизации процессов в программе реализовано управление из командной строки:

- ключ «-scene», после которого идет название OBJ файла, используется для загрузки сцены из указанного файла;
- ключ «-image», после которого идет название PNG файла, используется для генерации изображения и сохранение его в файл с указанным названием;
- ключ «-film» используется для генерации PNG изображений поворота сцены для фильма;
- ключ «-research» используется для замера времени отрисовки сцены;
- ключ «-test» используется для модульного тестирования.

## 2.7 Пример работы программы



Рисунок 2.5 – Олень. 1508 полигонов

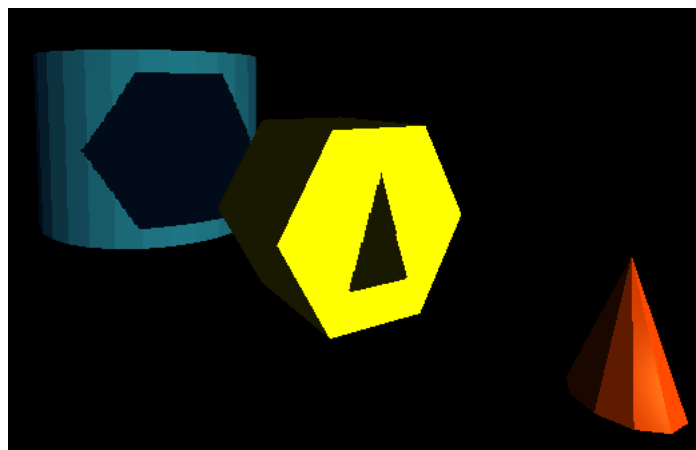


Рисунок 2.6 – Сцена с тенями. 300 полигонов

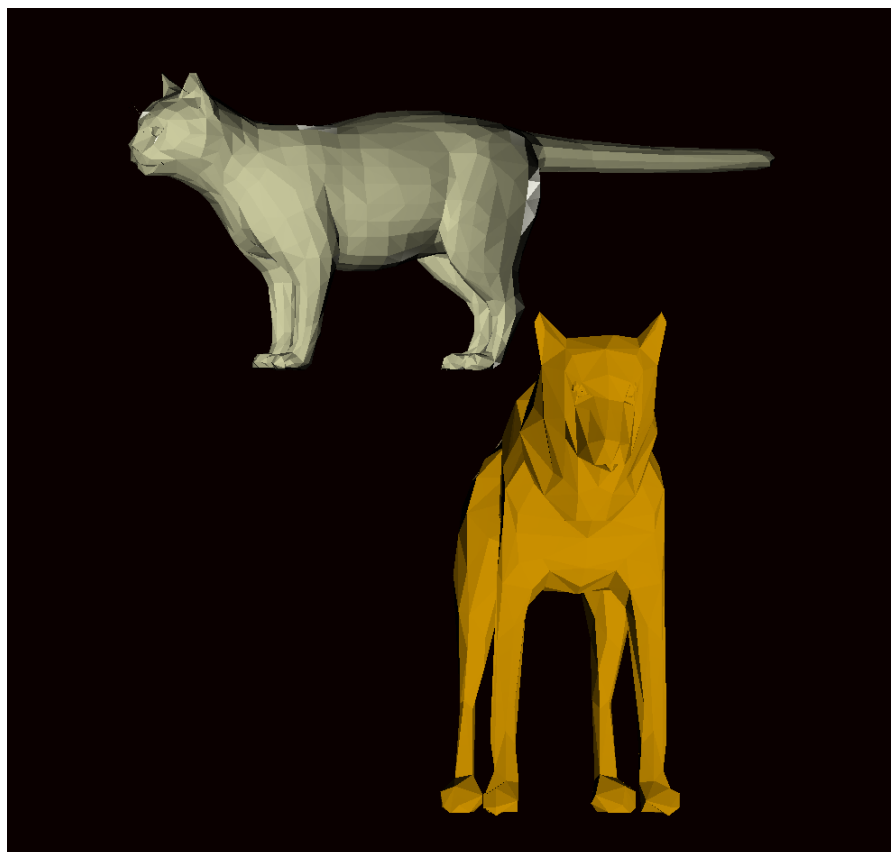


Рисунок 2.7 – Кот и волк. 3038 полигонов

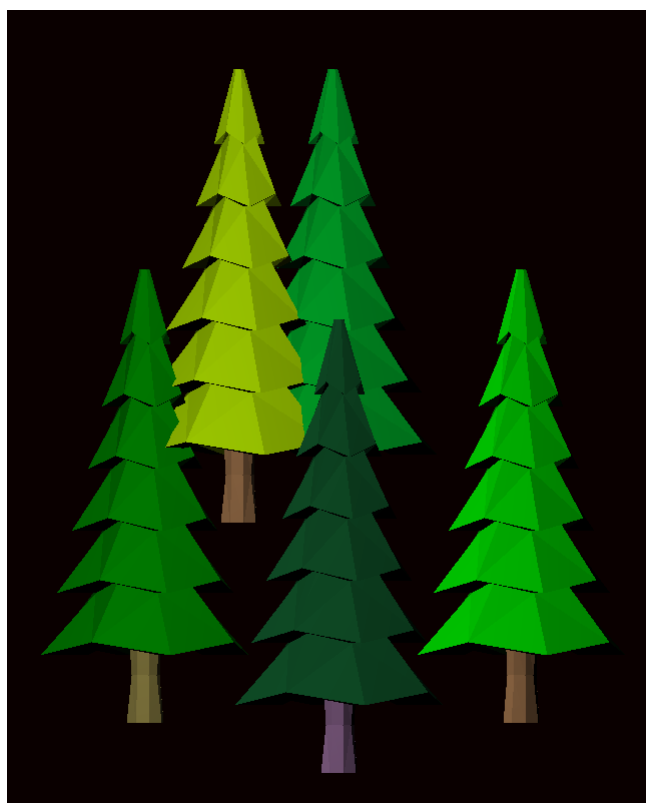


Рисунок 2.8 – Деревья. 1240 полигонов

Было проведено исследование временных характеристик программы от числа полигонов на сцене. Из полученных данных был построен график, изображенный на рисунке 2.9:

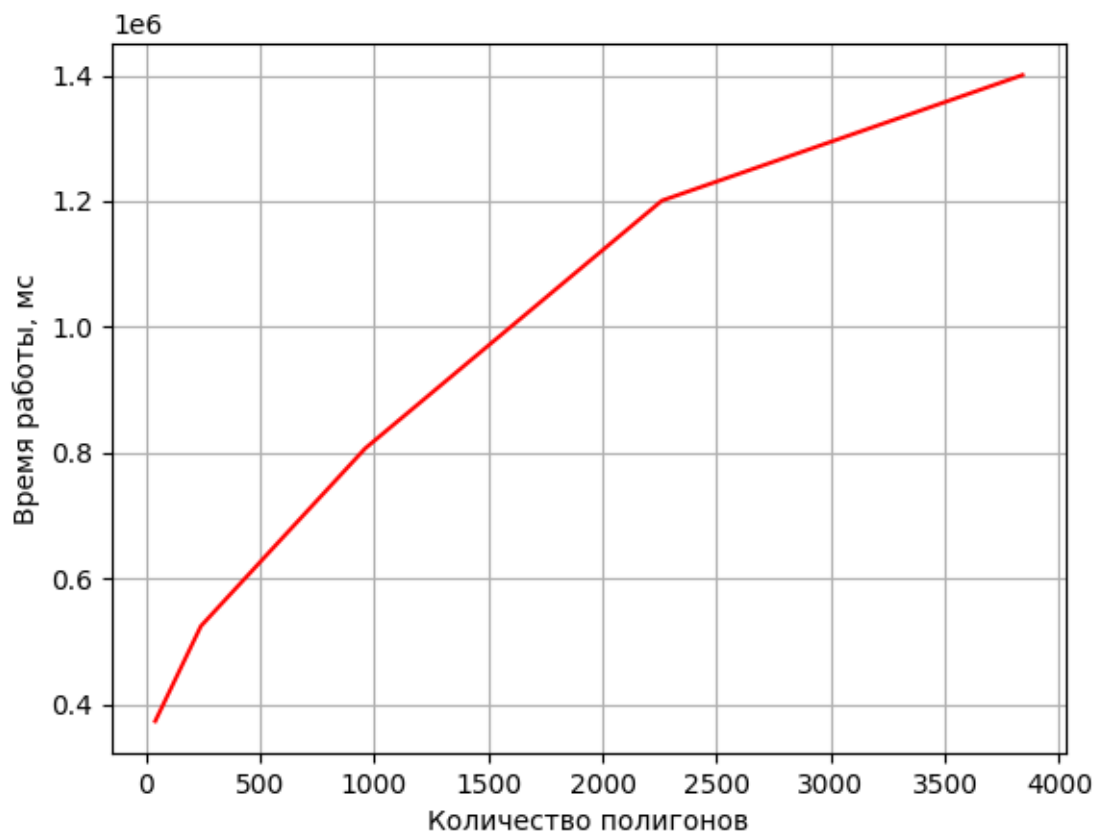


Рисунок 2.9 – Зависимость времени работы программы от количества полигонов

По итогам исследования можно сказать, что время работы алгоритма линейно увеличивается при увеличении числа полигонов на сцене.

# Заключение

В ходе выполнения практики все поставленные цели и задачи были выполнены:

- изучена документация Gitlab CI/CD, Docker, Qt, Qt Test, Qt Widgets, FFmpeg;
- создана программа, принимающая данные о сцене и создающая изображение, с помощью алгоритма, использующего Z-буфер;
- реализовано управление программой из командной строки;
- создан сценарий автоматизации сборки, тестирования и получения данных для будущего исследования
- проведено исследование зависимости времени работы программы от количества полигонов на сцене;
- созданы три сцены для демонстрации работоспособности конвейера;
- сгенерирован видеофильм на основе изображений сцен;
- созданы семь стадий для демонстрации работоспособности всего конвейера;
- создан модульный тест для демонстрации работоспособности системы.

Проделанная работа помогла получить практические навыки автоматизации процессов интеграции и развёртывания программного обеспечения в процессе разработки, а также закрепить полученные навыки в области компьютерной графики.

# Литература

- [1] Документация по Gitlab CI/CD [Электронный ресурс]. Режим доступа: <https://docs.gitlab.com/ee/ci>.(дата обращения: 06.09.2022)
- [2] Документация по Docker [Электронный ресурс]. Режим доступа: <https://docs.docker.com>.(дата обращения: 06.09.2022)
- [3] Открытый репозиторий образов контейнеров, поддерживаемый Docker Inc.[Электронный ресурс]. Режим доступа: <https://hub.docker.com>(дата обращения: 06.09.2022)
- [4] Документация по библиотеке Qt [Электронный ресурс]. Режим доступа: <https://doc.qt.io>(дата обращения: 06.09.2022)
- [5] Документация по библиотеке Qt Test [Электронный ресурс]. Режим доступа: <https://doc.qt.io/qt-6/qtest-index.html>(дата обращения: 06.09.2022)
- [6] Документация по библиотеке Qt Widgets [Электронный ресурс]. Режим доступа: <https://doc.qt.io/qt-6/qtwidgets-index.html>(дата обращения: 06.09.2022)
- [7] Документация по утилите FFmpeg [Электронный ресурс]. Режим доступа: <https://ffmpeg.org/documentation.html>(дата обращения: 06.09.2022)
- [8] An Overview of 3D Data Content, File Formats and Viewers [Электронный ресурс]. Режим доступа: <http://isda.ncsa.illinois.edu/peter/publications/techreports/2008/NCSA-ISDA-2008-002.pdf>(дата обращения: 06.09.2022)
- [9] Performance Evaluation of Secrete Image Steganography Techniques Using Least Significant Bit (LSB) Method [Электронный ресурс]. Режим доступа: <http://www.ijcstjournal.org/volume-6/issue-2/IJCST-V6I2P30.pdf>(дата обращения: 06.09.2022)
- [10] Rogers David, Adams J., Matematical Elements for Computer Graphics [Электронный ресурс] // Аннаполис, United States Naval Academy. 1989.



Режим доступа: [https://itslearningakarmazyan.files.wordpress.com/2015/08/rodzhers\\_adams.pdf](https://itslearningakarmazyan.files.wordpress.com/2015/08/rodzhers_adams.pdf). (дата обращения: 06.09.2022)

- [11] Модель Фонга. [Электронный ресурс]. Режим доступа: [https://compgraphics.info/3D/lighting/phong\\_reflection\\_model.php](https://compgraphics.info/3D/lighting/phong_reflection_model.php) (дата обращения: 06.09.2022)