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 _____ **Чепиго Д.С**
подпись, дата *фамилия, и.о.*

Руководитель практики _____ **Куров А. В.**
подпись, дата *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
Чепиго 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 _____/ **Чепиго Д.С.** /

Содержание

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

```
graph:
  stage: result_graph
  image: python
  before_script:
    - pip install matplotlib
  script:
    - python3 result_graph.py
  artifacts:
    paths:
    - chicago_develop/result.png
    expire_in: 1 day
  needs:
  - 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

```
debug-build:
stage: build_debug
  script: make
```

- Артефакты задания(job artifacts)

  Артфекаты 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

```
generate-image:
script: ./app.exe -image
artifacts:
  paths:
  - /pictures/image.png
  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

```
# syntax=docker/dockerfile:1
FROM python:3.7 - alpine
WORKDIR /code
ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0
RUN apk add --no-cache gcc musl-dev linux-headers
COPY requirements.txt requirements.txt
RUN pipinstall -r requirements.txt
EXPOSE 5000
COPY . .
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

```
version: "3.9"
services:
  web:
    build: .
    ports:
      -"8000:5000"
  redis:
    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 репозиториев 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 видеозаписи 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

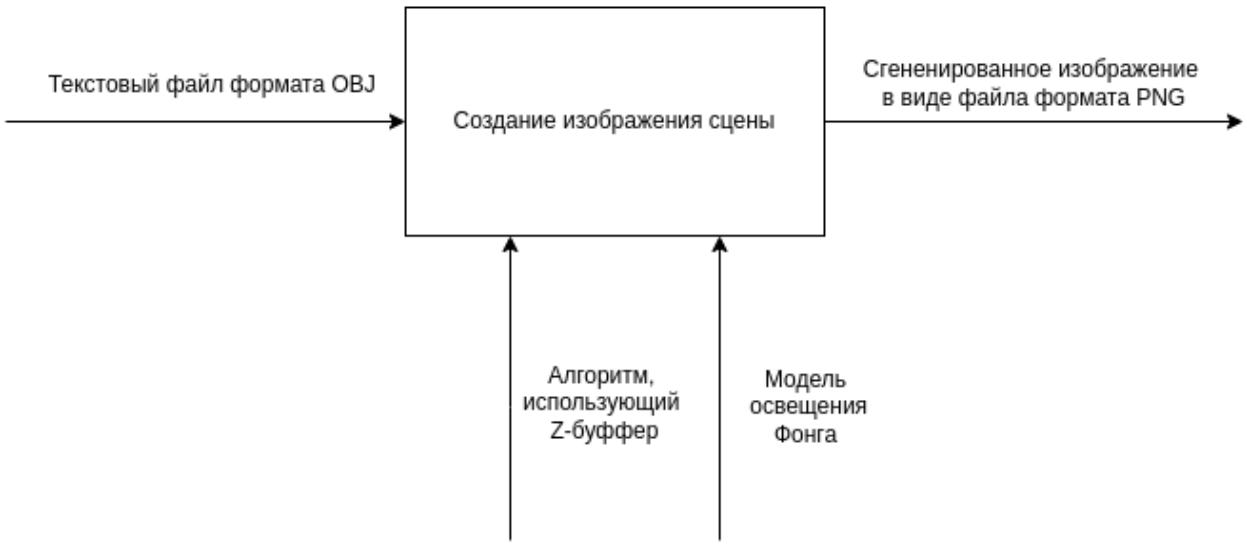

Рисунок 2.1 – IDEF0 диаграмма работы программы

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

```
# comment
o figure

v 10 20 30
v 40 50 100
v -10 20 40

f 1 2 3

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 использованна 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.

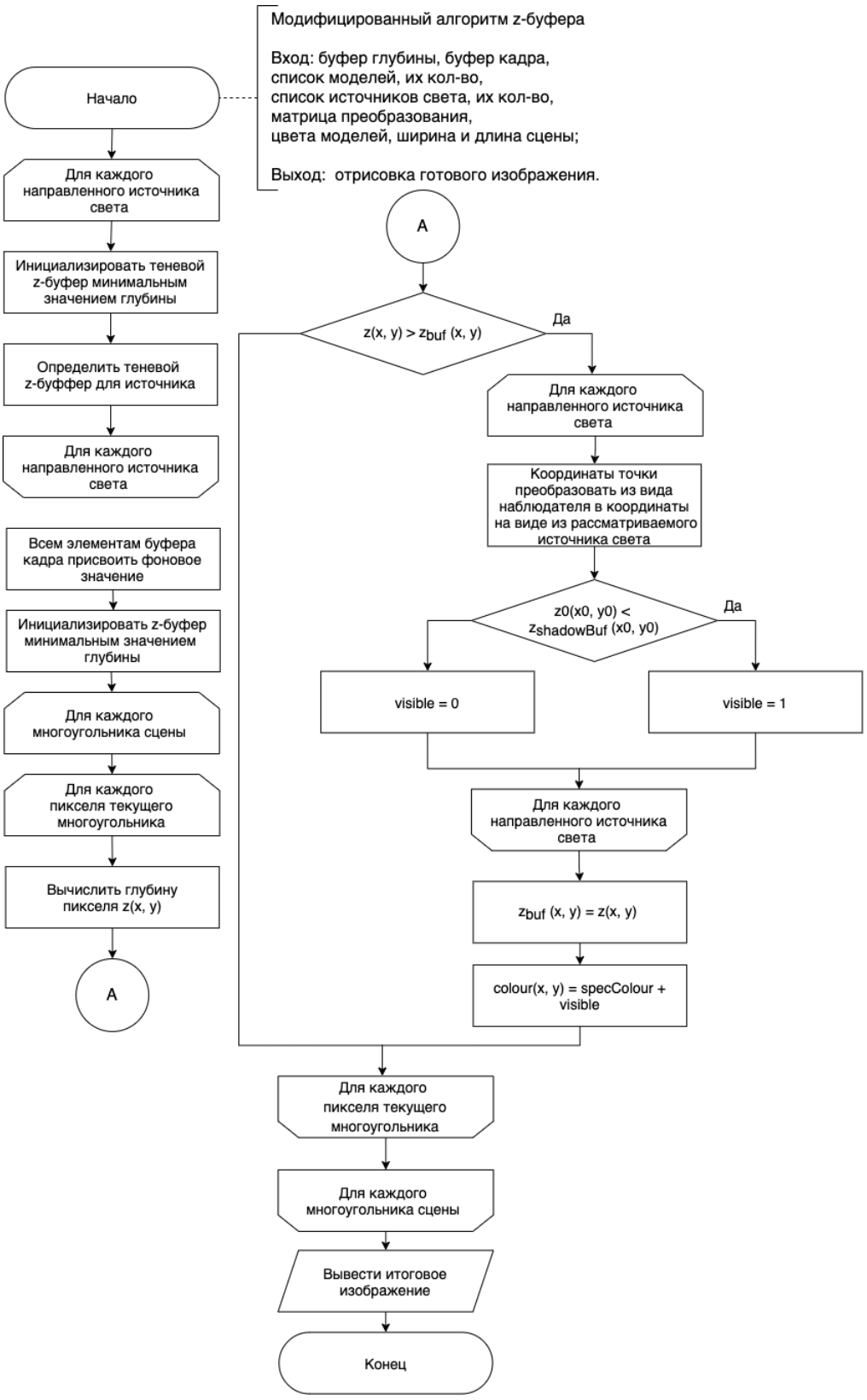

Рисунок 2.2 – модифицированный алгоритма z-буфера

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:

$$I_a = k_a i_a \tag{2.1}$$

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):

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

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:

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

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,
α - коэффициент блеска, свойство материала.

Тогда интенсивность света подсчитывается формулой 2.4:

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

Пример работы модели Фонга:

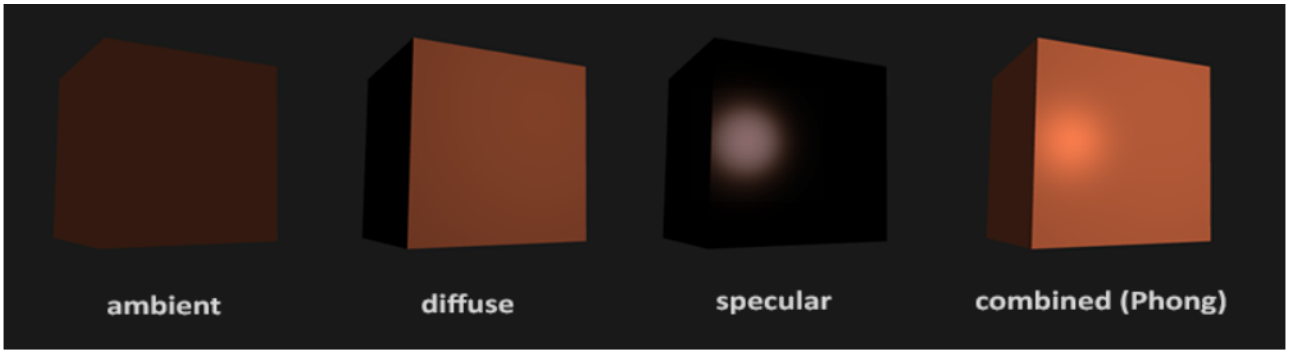

Рисунок 2.3 – Модель освещения Фонга

2.3 Сценарий Gitlab CI/CD

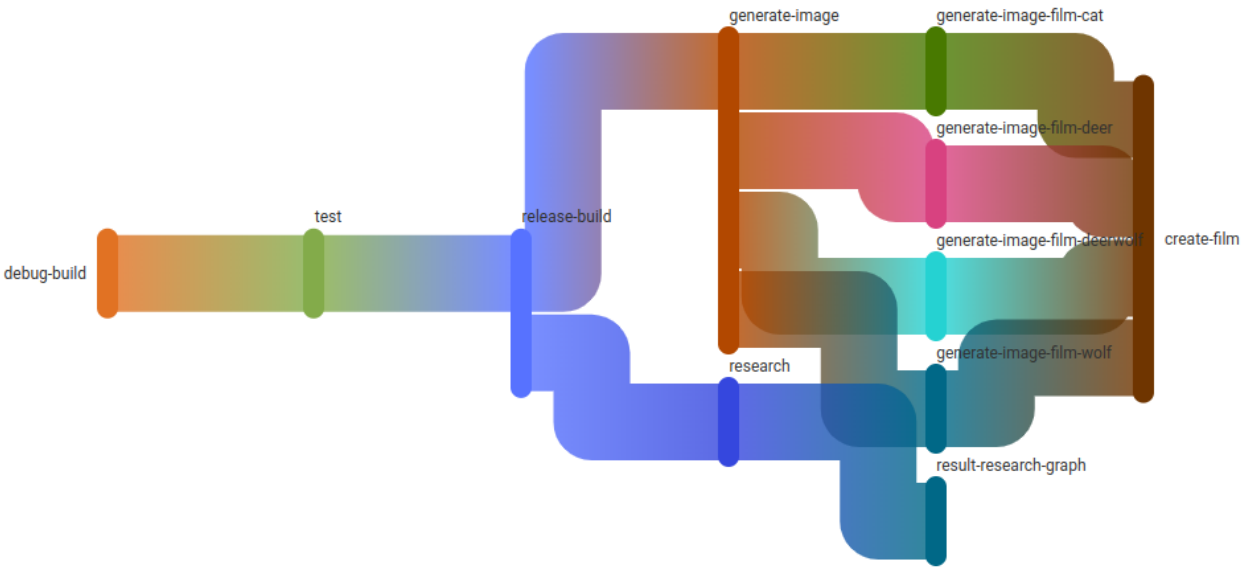

Рисунок 2.4 – Сценарий Gitlab CI/CD

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

```
image: darkmattercoder/qt-build:latest

stages:
  - build_debug
  - testing
  - build_release
  - generate_image
  - generate_image_film
  - create_film
  - research

debug-build:
  stage: build_debug
  script:
    - cd chicago_develop
    - qmake -project QT+=widgets QT+=testlib chicago_develop.pro
    - qmake
    - make
  artifacts:
    paths:
```

```
      - chicago_develop/chicago_develop
    expire_in: 1 day

unit-test:
  stage: testing
  script:
    - export QT_QPA_PLATFORM=offscreen
    - cd chicago_develop
    - ./chicago_develop -test
  needs:
    - debug-build

release-build:
  stage: build_release
  script:
    - cd chicago_develop
    - qmake -project QT+=widgets QT+=testlib chicago_develop.pro
    - qmake
    - make
  artifacts:
    paths:
      - chicago_develop/chicago_develop
    expire_in: 1 day
  needs:
    - unit-test


generate-image:
  stage: generate_image
  script:
    - export QT_QPA_PLATFORM=offscreen
    - cd chicago_develop
    - ./chicago_develop -scene ./scenes/scene.obj -image deer.png
  artifacts:
    paths:
      - chicago_develop/deer.png
    expire_in: 7 days
  needs:
    - release-build


generate-image-film-deer:
  stage: generate_image_film
  script:
    - export QT_QPA_PLATFORM=offscreen
    - cd chicago_develop
    - mkdir film
    - ./chicago_develop -scene ./scenes/deer.obj -film 0
```

```
  artifacts:
    paths:
      - chicago_develop/film/film*.png
    expire_in: 7 days
  needs:
    - release-build
    - generate-image

generate-image-film-catwolf:
  stage: generate_image_film
  script:
    - export QT_QPA_PLATFORM=offscreen
    - cd chicago_develop
    - mkdir film
    - ./chicago_develop -scene ./scenes/catwolf.obj -film 20
  artifacts:
    paths:
      - chicago_develop/film/film*.png
    expire_in: 7 days
  needs:
    - release-build
    - generate-image

generate-image-film-trees:
  stage: generate_image_film
  script:
    - export QT_QPA_PLATFORM=offscreen
    - cd chicago_develop
    - mkdir film
    - ./chicago_develop -scene ./scenes/trees.obj -film 40
  artifacts:
    paths:
      - chicago_develop/film/film*.png
    expire_in: 7 days
  needs:
    - release-build
    - generate-image


create-film:
  image:
    name: jrottenberg/ffmpeg
    entrypoint: [""]
  stage: create_film
  script:
    - cd chicago_develop/film
    - ffmpeg -r 5 -f image2 -s 1920x1080 -i film%d.png -crf 1 -pix_fmt yuv420p
        film.mp4
```

```
  artifacts:
    paths:
      - chicago_develop/film/film.mp4
    expire_in: 7 days
  needs:
    - release-build
    - generate-image
    - generate-image-film-deer
    - generate-image-film-catwolf
    - generate-image-film-trees

research-time:
  stage: research
  script:
    - cd chicago_develop
    - export QT_QPA_PLATFORM=offscreen
    - bash research.sh
  artifacts:
    paths:
      - chicago_develop/result.txt
    expire_in: 1 day
  needs:
    - release-build

result-research-graph:
  stage: research
  image: python
  before_script:
    - pip install matplotlib
  script:
    - cd chicago_develop
    - python3 result_graph.py
  artifacts:
    paths:
      - chicago_develop/result.png
    expire_in: 1 day
  needs:
    - 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

  Мнималистичный 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

```
class TestTriangle : public QObject
{
    Q_OBJECT

    public:
        explicit TestTriangle(QObject *parent = nullptr);

    private slots:
        void test_findZ();
};
```

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

Листинг 2.4 – Модулный тест

```
void TestTriangle::test_findZ()
{
    Triangle polygon(QVector3D(0, 0, 10), QVector3D(10, 10, 10), QVector3D(10,
        0, 10));

    double z = polygon.findZ(QVector3D(6, 6, 0));

    QCOMPARE(z, 10);
}
```

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

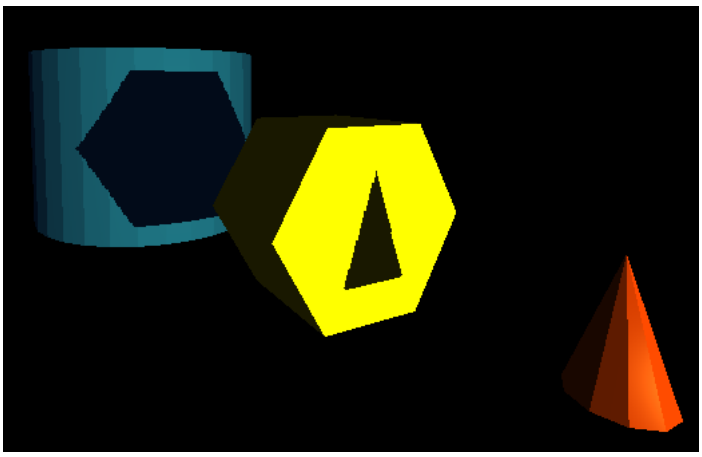

Рисунок 2.6 – Сцена с тенями. 300 полигонов

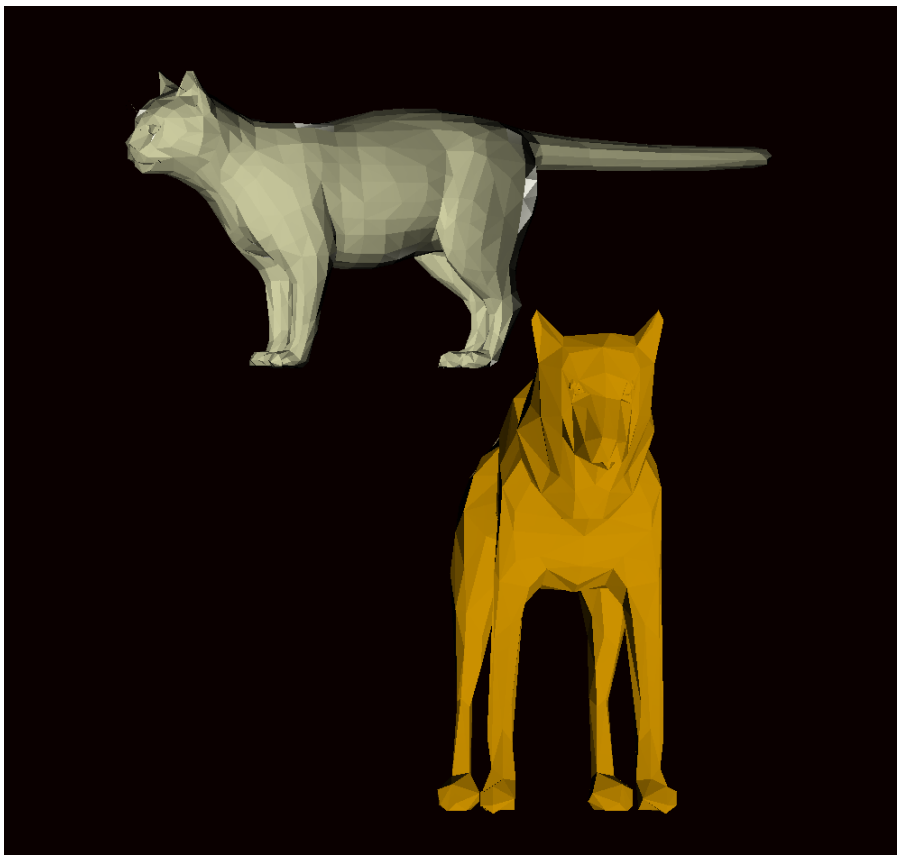

Рисунок 2.7 – Кот и волк. 3038 полигонов

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:

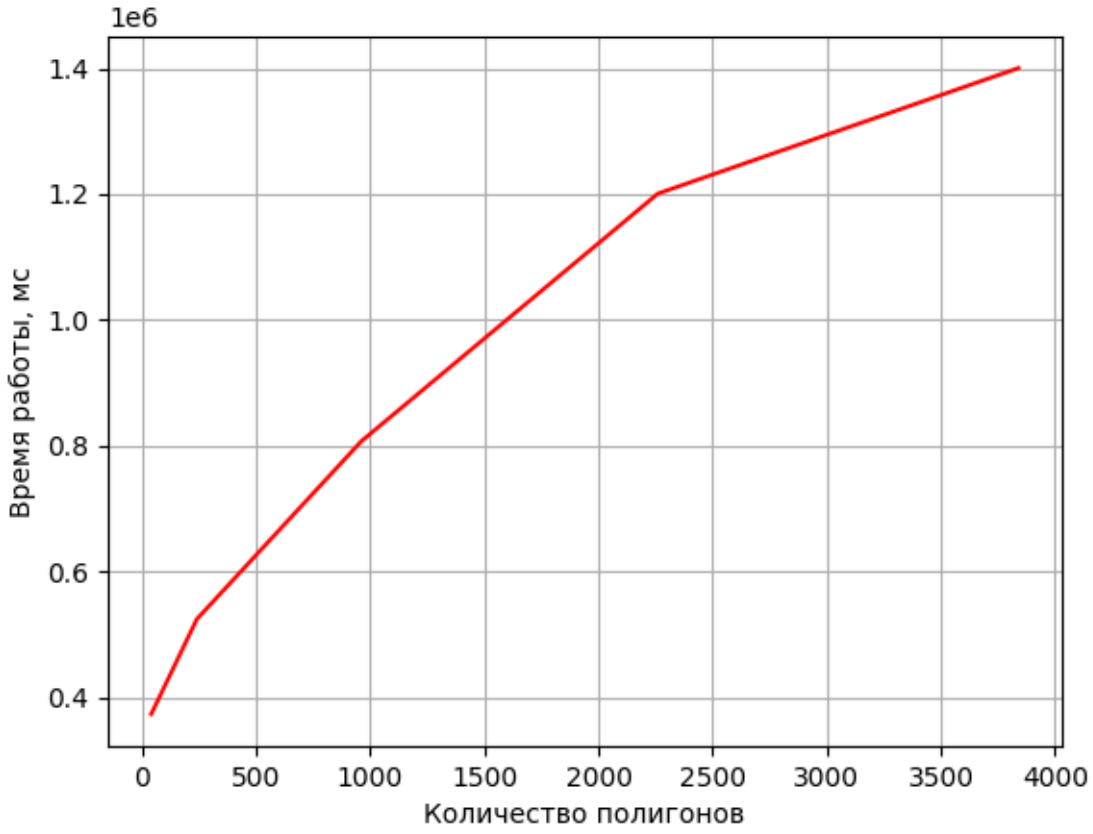

Рисунок 2.9 – Зависимость времени работы программы от количества полигонов

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/qttest-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.(дата обращения: 06.09.2022)

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