Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Здесь пишем тему»

Студент _____ _____ _____
(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель _____ _____
(Подпись, дата) (И.О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современная компьютерная графика активно используется в различных сферах, включая разработку игр, создание анимации, визуализацию научных данных и моделирование. В основе многих графических технологий лежат преобразования объектов, которые позволяют менять их положение, форму и размеры. Однако простых операций, таких как поворот, масштабирование и перемещение, недостаточно для создания сложных визуальных эффектов. Здесь на помощь приходят более сложные методы, например, морфинг — плавное преобразование одной формы в другую.

Данная работа посвящена изучению и реализации морфинга трёхмерных объектов, на примере перехода между кубом и сферой, сделанных на лабораторных по компьютерной графике. Выбор темы обусловлен желанием изучить алгоритмы морфинга и их применение для анимации. Это исследование дополняет базовые знания, полученные в ходе изучения курса компьютерной графики, и позволяет расширить практический опыт работы с такими инструментами, как OpenGL и Python.

Целью данной работы является изучение и реализация алгоритма морфинга 3D-объектов с использованием современных технологий.

Для достижения цели поставлены следующие задачи:

1. Изучить теоретические основы морфинга и его применение.
2. Реализовать базовые формы (куб и сферу) с использованием треугольников (GL_TRIANGLE_STRIP), что обеспечит удобство отображения и высокую производительность.
3. Разработать алгоритм плавного перехода между объектами с использованием шейдеров.
4. Обеспечить взаимодействие с пользователем, позволяя изменять параметры морфинга в реальном времени.
5. Провести экспериментальное исследование работы программы, оценив её производительность и визуальное качество.

Работа организована следующим образом: в первой главе рассматриваются теоретические основы морфинга, включая его области применения и методы. Вторая глава посвящена описанию архитектуры программы и реализации алгоритма.

В третьей главе представлены результаты тестирования, а также анализ качества и производительности программы. Заключение содержит выводы и направления дальнейших исследований.

1 Теоретические основы морфинга

1.1 Понятие морфинга и его место в компьютерной графике

Морфинг (от англ. morphing — «превращение») представляет собой процесс плавного перехода одной формы объекта в другую. Этот метод широко используется в компьютерной графике для создания визуальных эффектов в анимации, фильмах, играх и других мультимедийных приложениях. Морфинг включает интерполяцию геометрических и визуальных характеристик объектов, таких как вершины, текстуры и освещение, что позволяет добиться реалистичного изменения формы [1].

В источнике [2] сказано, что одним из первых примеров использования морфинга в кино стал фильм "Вилли Вонка и шоколадная фабрика"(1971), где морфинг был использован для создания спецэффектов. С тех пор технология морфинга значительно эволюционировала, и сегодня она используется в самых разных областях, от медицины до видеоигр.

В основе морфинга лежат два ключевых этапа:

— определение соответствия между вершинами начального и конечного объектов;

— вычисление промежуточных состояний объектов через интерполяцию.

1.2 Историческое развитие морфинга

История морфинга начинается с ранних экспериментов в области компьютерной графики в 1970-х и 1980-х годах. В это время исследователи начали разрабатывать алгоритмы для плавного перехода одной формы в другую. Одним из первых значительных достижений в этой области стала работа Томаса Бера и Скотта Уэйверли, которые представили метод морфинга, основанный на интерполяции ключевых точек [3].

В 1990-х годах морфинг начал активно использоваться в кинематографе. Фильмы, такие как "Терминатор 2: Судный день"(1991) и "Маска"(1994), продемонстрировали возможности морфинга для создания спецэффектов. В этих филь-

мах морфинг использовался для создания эффекта превращения одного персонажа в другого, что стало возможным благодаря развитию компьютерных технологий и улучшению алгоритмов интерполяции.

Морфинг также находит применение в области научной визуализации, где он используется для отображения динамических процессов, таких как рост кристаллов или изменения в биологических структурах. В медицине морфинг позволяет визуализировать изменения в анатомических структурах, например, рост опухолей или изменения в костной ткани [4]. В архитектуре морфинг используется для моделирования изменений в зданиях и сооружениях, что позволяет архитекторам визуализировать изменения в форме зданий и изучать их влияние на окружающую среду.

1.3 Математические основы морфинга

Морфинг основан на математических методах интерполяции, которые позволяют плавно изменять форму объекта. Как сказано в [5], основные методы интерполяции включают:

— Линейную интерполяцию: простейший метод, при котором промежуточные точки вычисляются как среднее значение между начальной и конечной точками. Формула линейной интерполяции выглядит следующим образом: $P(t) = (1 - t) * P_0 + t * P_1$, где P_0 и P_1 — начальная и конечная точки, а t — параметр интерполяции, изменяющийся от 0 до 1.

— Кубическую интерполяцию: более сложный метод, который учитывает не только начальные и конечные точки, но и их производные, что позволяет добиться более плавного перехода. Формула кубической интерполяции выглядит следующим образом: $P(t) = a*t^3+b*t^2+c*t+d$, где a, b, c, d — коэффициенты, определяемые на основе начальных и конечных точек и их производных.

— Сплайновую интерполяцию: метод, использующий сплайны для создания гладких кривых, которые проходят через заданные точки.

Кроме того, в [6] сказано, что существуют и другие методы интерполяции, такие как бикубическая интерполяция и интерполяция с использованием радиальных базисных функций. Эти методы позволяют добиться еще более плавных и реалистичных переходов между формами объектов.

1.4 Алгоритмы морфинга

Алгоритмы морфинга можно разделить на несколько категорий в зависимости от используемых методов интерполяции и подходов к реализации. Основные категории, согласно [7], включают:

— **Геометрический морфинг:** этот метод фокусируется на интерполяции геометрических характеристик объектов, таких как вершины и грани. Геометрический морфинг широко используется в компьютерной графике для создания анимаций и спецэффектов.

— **Текстурный морфинг:** этот метод фокусируется на интерполяции текстурных характеристик объектов, таких как цвет и текстура. Текстурный морфинг используется для создания реалистичных переходов между текстурами объектов.

— **Гибридный морфинг:** этот метод комбинирует геометрический и текстурный морфинг для создания более сложных и реалистичных переходов. Гибридный морфинг используется в кинематографе и видеоиграх для создания спецэффектов и анимаций.

1.5 Применение морфинга

Морфинг нашёл своё применение в различных областях:

— **Кинематограф:** создание спецэффектов, например, превращение одного персонажа в другого. Примером может служить фильм "Терминатор 2: Судный день"(1991), где морфинг был использован для создания эффекта превращения робота T-1000.

— **Научная визуализация:** плавное отображение изменения данных, таких как модели молекул или результаты симуляций. Морфинг позволяет визуализировать динамические процессы, такие как рост кристаллов или изменения в биологических структурах.

— **Игровая индустрия:** анимация персонажей, трансформация объектов и переходы между формами. В играх морфинг используется для создания реалистичных анимаций, таких как превращение персонажа в другого или изменение формы объектов.

— **Дизайн:** моделирование прототипов и изучение переходов между концепциями. Морфинг позволяет дизайнерам визуализировать изменения в форме объектов и изучать их влияние на конечный продукт.
— **Медицина:** визуализация изменений в анатомических структурах, например, рост опухолей или изменения в костной ткани. Морфинг позволяет врачам наблюдать за динамикой изменений в организме пациента и планировать лечение.
— **Архитектура:** моделирование изменений в зданиях и сооружениях, например, визуализация строительных проектов. Морфинг позволяет архитекторам визуализировать изменения в форме зданий и изучать их влияние на окружающую среду.
— **Образование:** использование морфинга для создания наглядных учебных материалов, таких как анимации и симуляции, которые помогают студентам лучше понять сложные концепции.
— **Реклама:** создание эффектных визуальных эффектов для рекламных роликов и презентаций, что позволяет привлечь внимание зрителей и сделать рекламу более запоминающейся.
— **Искусственный интеллект:** использование морфинга для создания реалистичных анимаций в виртуальных ассистентах и роботах, что позволяет улучшить взаимодействие с пользователями.

1.6 Методы морфинга трёхмерных объектов

Существует несколько подходов к реализации морфинга в трёхмерной графике:

1. **Линейный морфинг.** Этот метод предполагает линейную интерполяцию позиций вершин начального и конечного объектов. Преимущество метода в его простоте, однако он может создавать визуально некорректные результаты, если структуры объектов сильно различаются.
2. **Нелинейный морфинг.** В данном подходе интерполяция может быть модифицирована функциями, учитывающими форму объектов. Это позволяет создать более реалистичные переходы, но увеличивает сложность реализации. Одним из примеров нелинейного морфинга является использование кубической интерполяции.

3. **Морфинг с использованием шейдеров.** Этот метод позволяет выполнять вычисления на графическом процессоре (GPU), что значительно ускоряет обработку и позволяет добиться плавности анимации даже при сложных преобразованиях.

Листинг 1: Реализация морфинга в вершинном шейдере

```
uniform float morphFactor;
in vec3 cubePosition;
in vec3 spherePosition;

void main() {
    vec3 morphedPosition = mix(cubePosition, spherePosition, morphFactor);
    gl_Position = projection * view * model * vec4(morphedPosition, 1.0);
}
```

4. **Морфинг с использованием физических моделей.** Этот метод использует физические законы для моделирования переходов между объектами, что позволяет создать более реалистичные и естественные анимации.
   Пример использования физических моделей для морфинга включает в себя моделирование деформаций объектов под воздействием внешних сил. Это позволяет создать более реалистичные анимации, такие как превращение одного объекта в другой под воздействием гравитации или других физических сил.

1.7 Особенности морфинга куба и сферы

Куб и сфера представляют собой объекты с разной структурой и количеством вершин. Куб состоит из шести граней, каждая из которых образована четырьмя вершинами. Сфера, напротив, имеет гладкую поверхность, которая аппроксимируется большим количеством треугольников.

В данной работе для удобства отображения и повышения производительности оба объекта описываются набором треугольников, упорядоченных с использованием `GL_TRIANGLE_STRIP`. Такой подход позволяет упростить обработку данных и повысить производительность за счёт эффективного использования графического процессора.

Для реализации морфинга между кубом и сферой важно обеспечить одинаковое количество вершин и корректное их соответствие. Это достигается путём

равномерного разбиения поверхностей объектов и использования алгоритмов для поиска ближайших точек.

1.8 Оптимизация вычислений

Для улучшения производительности морфинга можно использовать различные методы оптимизации:

- **Использование GPU:** выполнение вычислений на графическом процессоре позволяет значительно ускорить обработку данных.
- **Оптимизация геометрии:** уменьшение количества вершин и использование более эффективных структур данных для хранения геометрии объектов.
- **Параллельные вычисления:** использование многопоточности и параллельных вычислений для ускорения обработки данных.
- **Алгоритмы сжатия:** использование алгоритмов сжатия для уменьшения объема данных, что позволяет ускорить передачу и обработку данных.
- **Кэширование:** использование кэширования для хранения промежуточных результатов, что позволяет избежать повторных вычислений и ускорить процесс морфинга.
- **Оптимизация шейдеров:** использование оптимизированных шейдеров для ускорения вычислений на GPU. Это включает в себя использование более эффективных алгоритмов и структур данных в шейдерах.

1.9 Примеры использования морфинга в реальных проектах

Морфинг находит широкое применение в реальных проектах, таких как создание спецэффектов в кино, анимация персонажей в видеоиграх и визуализация данных в научных исследованиях. Например, в фильме "Аватар"(2009) морфинг был использован для создания реалистичных анимаций персонажей и окружающей среды, что показано в [8]. В видеоигре "The Legend of Zelda: Breath of the

Wild"(2017) морфинг был использован для создания плавных переходов между различными формами объектов и персонажей.

1.10 Будущее морфинга

С развитием технологий морфинг продолжает эволюционировать и находить новые области применения. В будущем можно ожидать улучшения качества и реалистичности морфинга за счет использования более сложных математических моделей и алгоритмов. Также можно ожидать развития технологий виртуальной и дополненной реальности, где морфинг будет играть ключевую роль в создании реалистичных и интерактивных визуальных эффектов.

2 Описание реализации программы

2.1 Выбор алгоритма

Для реализации морфинга между кубом и сферой был выбран алгоритм линейной интерполяции. Этот алгоритм позволяет плавно переходить от одной формы к другой, используя параметр `morphFactor`, который изменяется от 0 до 1. Реализация морфинга осуществляется в вершинном шейдере, что позволяет эффективно использовать возможности GPU для параллельных вычислений. В вершинном шейдере выполняется интерполяция позиций вершин между начальной (куб) и конечной (сфера) формами. Это обеспечивает высокую производительность и плавность анимации, так как все вычисления выполняются на GPU, что значительно снижает нагрузку на центральный процессор (CPU).

2.2 Использование алгоритма

Алгоритм морфинга между кубом и сферой включает несколько ключевых этапов, начиная с генерации вершин и заканчивая их интерполяцией в вершинном шейдере. Подробное описание использования алгоритма представлено ниже.

2.2.1 Генерация вершин сферы

Первым шагом является генерация вершин сферы. Поверхность сферы аппроксимируется сеткой широт и долгот, что обеспечивает гладкость поверхности. Количество вершин сферы определяется параметрами `lat_segments` (количество сегментов широты) и `lon_segments` (количество сегментов долготы).

2.2.2 Адаптация вершин куба

Исходя из количества полученных вершин сферы, необходимо адаптировать количество вершин куба для обеспечения совместимости с морфингом. Этот процесс включает несколько шагов:

1. **Разбиение граней куба на сетку точек.** Каждая грань куба разбивается на равномерную сетку точек. Количество точек на каждой грани определяется параметром `divisions`, который указывает количество отрезков,

на которые делится ребро куба. Общее количество вершин в кубе после этого шага можно посчитать по формуле: $number\ of\ vertices = 2 * (divisions + 1)^2 - 2 * (divisions + 1)$

2. **Удаление полос точек.** Если количество вершин после первого шага превышает необходимое количество, удаляются одни или несколько полос точек на каждой грани. Точки удаляются полосами для корректной отрисовки треугольников в дальнейшем. Параметр `divisions` подбирается таким образом, чтобы минимизировать разницу между необходимым количеством точек и количеством точек после удаления полос.
3. **Добавление точек.** Если после удаления полос точек все еще недостаточно, добавляются дополнительные точки. Точки добавляются равномерно в грани. Если в грани нужно добавить три или более точек, то точки добавляются дублированием различных треугольников. Если осталось добавить одну или две точки, то дублируется последняя точка в грани. Это сделано для правильного поиска нормалей в дальнейшем и соответственно корректного отображения куба с текстурой.

2.2.3 Вычисление нормалей

После генерации вершин куба и сферы вычисляются нормали для каждой точки. Нормали для сферы вычисляются просто как нормализованные радиус-векторы для точек сферы. Нормали для куба вычисляются на основе его геометрии.

2.2.4 Вычисление текстурных координат

Далее вычисляются текстурные координаты для куба и сферы. Эти координаты также будут меняться с помощью морфинга.

2.2.5 Передача данных в вершинный шейдер

Все найденные данные (вершины, нормали, текстурные координаты) передаются в вершинный шейдер с использованием библиотеки OpenGL. В вершинном шейдере выполняется интерполяция этих данных между начальной (куб) и конечной (сфера) формами. Переменная для интерполяции `morphFactor` вычисляется в

программе с учетом течения времени и изменяется в зависимости от направления преобразования.

2.2.6 Отображение и взаимодействие

С помощью библиотеки OpenGL происходит отображение полученных фигур. Пользователь может взаимодействовать с программой в реальном времени, управляя различными параметрами морфинга и отображения объектов с помощью клавиатуры и мыши.

2.3 Обоснование выбранных технологий

Для реализации комплекса программ были выбраны следующие языки программирования, технологии и сторонние библиотеки:

- **Python:** Язык программирования, обеспечивающий высокую производительность и удобство разработки.
- **OpenGL:** Библиотека для работы с 3D-графикой, позволяющая использовать возможности GPU для рендеринга.
- **GLFW:** Библиотека для создания окон и обработки событий.
- **NumPy:** Библиотека для работы с массивами данных, обеспечивающая высокую производительность вычислений.

2.4 Общая структура проекта

Архитектура проекта разделена на несколько модулей, каждый из которых отвечает за свою функциональность:

- `main.py` — основной модуль программы, содержащий логику запуска приложения, настройки шейдеров и обработки морфинга.
- `callbacks.py` — обработка событий, таких как нажатия клавиш, движение мыши и прокрутка колеса.
- `utils.py` — вспомогательные функции для работы с матрицами, текстурами и шейдерами.
- `shape.py` — генерация трёхмерных объектов (куб и сфера).
- `window_manager.py` — управление окном и настройками OpenGL.

— `fragment_shader.frag` — фрагментный шейдер.

— `vertex_shader.vert` — вершинный шейдер.

Подобная структура обеспечивает модульность и удобство разработки, позволяя независимо модифицировать и тестировать отдельные компоненты программы.

2.5 Форматы представления данных

Входные данные представляют собой количество количество сегментов широты и количество сегментов долготы. Выходные данные включают в себя промежуточные и окончательные результаты морфинга, которые отображаются на экране.

2.6 Структуры данных для внутреннего представления

Для внутреннего представления данных используются массивы вершин и нормалей, которые хранятся в формате `numpy.array`. Это позволяет эффективно работать с данными и передавать их на GPU для рендеринга.

2.7 Оценка теоретической сложности алгоритмов

Теоретическая сложность реализуемых алгоритмов оценивается следующим образом:

Генерация вершин куба и сферы: O(n), где n — количество вершин. Морфинг: O(1) для каждой вершины, так как интерполяция выполняется за константное время.

2.8 Генерация объектов

Основой работы программы является генерация трёхмерных объектов: куба и сферы. Оба объекта строятся с использованием треугольников (`GL_TRIANGLE_STRIP`), что упрощает их рендеринг и повышает производительность.

2.8.1 Класс CUBE

Куб представлен классом CUBE, который создаёт его вершины и нормали. Для обеспечения совместимости с морфингом количество точек на поверхности куба адаптируется к числу вершин сферы. За это отвечает метод _subdivide_face, показанный в листинге 4, который разбивает каждую грань куба на равномерную сетку точек в нужном порядке. В качестве входного параметра метод принимает total_expected_points, которое указывает общее ожидаемое количество точек на всём кубе. Упрощенный вариант класса представлен в листинге 2.

Листинг 2: Код генерации куба

```
class CUBE(Shape):
    def __init__(self, total_expected_points):
        super().__init__()
        self.vertices = np.array([...])  # координаты 8 вершин
        self.faces = [...]               # описание граней
        self._subdivide_face(total_expected_points)  # разбиение куба на
        ↪  нужное количество вершин
        self._update_normals()  # обновление нормалей

    def _custom_shuffle(self, arr, a, b):
        # Метод для переставления точек в нужном порядке на отрезке [a,
        ↪  b]
        # Вызывается только в _subdivide_face

        return shuffled_arr

    def _update_normals(self):
        # Метод для обновления нормалей
```

2.8.2 Класс SPHERE

Сфера создаётся классом SPHERE, в котором поверхность аппроксимируется сеткой широт и долгот. Использование триангуляции для генерации вершин обеспечивает гладкость поверхности. Пример кода представлен в листинге 3.

Листинг 3: Код генерации сферы

```
class SPHERE(Shape):
    def __init__(self, lat_segments, lon_segments, radius=0.5):
        super().__init__()
        self.lat_segments = lat_segments
        self.lon_segments = lon_segments
        self.radius = radius
        self.generate_sphere()  # Генерация точек сферы с учетом порядка
        ↪  для однозначного отображения с помощью GL_TRIANGLE_STRIP
```

2.9 Реализация морфинга

Морфинг реализован с использованием шейдеров, которые обрабатывают данные о вершинах объектов на графическом процессоре. Логика морфинга основывается на интерполяции позиций и нормалей вершин между начальной (куб) и конечной (сфера) формами.

На уровне программы интерполяция контролируется переменной `morphFactor`, изменяющейся от 0 (куб) до 1 (сфера). Код обработки переменной морфинга приведён в 5.

На GPU выполняются вычисления по смешиванию данных вершин, как показано в листинге 6.

2.10 Работа с событиями

Программа поддерживает взаимодействие с пользователем. Основные события включают:

— Нажатие клавиши `M` для включения/выключения морфинга.

— Нажатие клавиши `F` для переключение между отображением каркаса и заполненной модели.

— Нажатие клавиши `T` для включения/выключения текстуры.

— Нажатие ↑ / ↓ для увеличения/уменьшения объекта.

— Управление скоростью морфинга клавишами $+$ и $-$.

— Нажатие клавиши `Space` для сброса настроек.

— Нажатие клавиши `Escape` для закрытия окна.

Листинг 4: Код метода _subdivide_face класса CUBE

```
    def _subdivide_face(self, total_expected_points):
        # Разбиение каждой грани на одинаковое количество точек
        points = []
        for k, face in enumerate(self.faces):
            # v1, v2, v3, v4 - вершины грани
            for i in range(divisions):
                for j in range(divisions + 1):
                    p1 = v1 + (v2 - v1) * (j / divisions)
                    p2 = v4 + (v3 - v4) * (j / divisions)
                    point1 = p1 + (p2 - p1) * (i / divisions)
                    point2 = p1 + (p2 - p1) * ((i + 1) / divisions)

                    # Добавляем поочерёдно точки двух полос для получения
                    ↪ треугольников
                    points.append(point1)
                    points.append(point2)

        # Если точек больше, чем нужно, то удаляем одинаковое количество
        ↪ линий с каждой грани
        if len(points) > total_expected_points:
            ...

        # Добавляем недостающее количество точек
        missing_points = total_expected_points - len(points)  #
        ↪ Количество нехватающих точек
        num_faces = 6
        face_size = len(points) // num_faces  # Массив точек всегда можно
        ↪ разделить на равные 6 частей (грани)
        for face_index in range(num_faces - 1, -1, -1):
            end = (face_index + 1) * face_size if face_index < num_faces
            ↪ - 1 else len(points)
            # Вычисляем, сколько точек добавить в текущую грань
            points_to_add = missing_points // num_faces
            if face_index < missing_points % num_faces:
                points_to_add += 1  # Распределяем остаток равномерно

            i = end - 3
            while points_to_add >= 3:  # Добавляем повторением
            ↪ треугольников
                point = points[i:i + 3]
                for j in range(2, -1, -1):
                    points.insert(i, point[j])
                points_to_add -= 3
                i -= 3
                end += 3

        self.vertices = np.array(points, dtype=np.float32)
```

Листинг 5: Алгоритм управления фактором морфинга

```python
def morph_factor(t, last_time, state):
    EPSILON = 1e-6  # Погрешность для проверки границ
    PAUSE_TIME = 500  # Время задержки на границах в миллисекундах

    time = glutGet(GLUT_ELAPSED_TIME)
    if state.get('morph') == True:
        current_time = time  # Текущее время
        # Проверка: задерживаем t на границах
        if t >= 1.0 - EPSILON:
            t = 1.0  # Для правильного отображения
            if current_time - last_time > PAUSE_TIME:
                if state.get('direction') > 0:
                    state['direction'] *= -1  # Меняем направление движения
                last_time = current_time  # Обновляем время последней паузы
                t += 0.001 * state.get('direction')
        elif t <= 0.0 + EPSILON:
            t = 0.0  # Для правильного отображения
            if current_time - last_time > PAUSE_TIME:
                if state.get('direction') < 0:
                    state['direction'] *= -1  # Меняем направление движения
                last_time = current_time  # Обновляем время последней паузы
                t += 0.001 * state.get('direction')
        else:
            # t движется непрерывно внутри диапазона (0, 1)
            t += 0.001 * state.get('direction')
            last_time = current_time

    return t, last_time
```

Листинг 6: Реализация морфинга в вершинном шейдере

```glsl
uniform float morphFactor;
in vec3 cubePosition;
in vec3 spherePosition;

void main() {
    vec3 morphedPosition = mix(cubePosition, spherePosition, morphFactor);
    gl_Position = projection * view * model * vec4(morphedPosition, 1.0);
}
```

— Прокрутка колеса мыши для изменения масштаба объекта.

— Перемещение мыши для вращения объекта.

Логика обработки событий реализована в модуле `callbacks.py`, пример кода приведён в листингах 7 и 8.

Листинг 7: Обработка событий клавиатуры

```
def key_callback(window, key, scancode, action, mods, state):
    if action == glfw.PRESS:
        if key == glfw.KEY_M:
            state['morph'] = not state['morph']
```

Листинг 8: Обработка событий мыши

```
def mouse_button_callback(window, button, action, mods, state):
    if button == glfw.MOUSE_BUTTON_LEFT:
        if action == glfw.PRESS:
            state['mouse_pressed'] = True
        elif action == glfw.RELEASE:
            state['mouse_pressed'] = False

def cursor_position_callback(window, xpos, ypos, state):
    """
    Обработка движения курсора.
    """
    if state.get('mouse_pressed', False):
        dx = xpos - state['last_mouse_pos'][0]
        dy = ypos - state['last_mouse_pos'][1]
        state['rotation_angle_x'] += dy * 0.2  # Скорость вращения
        state['rotation_angle_y'] -= dx * 0.2
    state['last_mouse_pos'] = [xpos, ypos]

def scroll_callback(window, xoffset, yoffset, state):
    """
    Обработка прокрутки колеса.
    """
    if xoffset > 0:
        state['size'] -= yoffset / 10
    else:
        state['size'] += yoffset / 10
```

2.11 Руководство администратора

2.11.1 Процедура инсталляции и деинсталляции

1. **Инсталляция:**
   - Установите Python и необходимые библиотеки: `pip install numpy glfw PyOpenGL`

— Скачайте исходный код программы и разместите его в удобной директории.

2. **Деинсталляция:**
   — Удалите директорию с исходным кодом программы.
   — При необходимости удалите установленные библиотеки: `pip uninstall numpy glfw PyOpenGL`

2.11.2 Параметры запуска из командной строки

Программа запускается командой: `python main.py -s`

2.11.3 Требования к аппаратному и системному программному обеспечению

— Операционная система: Windows, macOS, Linux.
— Графический процессор с поддержкой OpenGL 3.3.
— Python 3.x.

2.12 Руководство пользователя

2.12.1 Описание графического интерфейса

Программа предоставляет окно с рендерингом 3D-объектов. Пользователь может взаимодействовать с объектами с помощью клавиатуры и мыши.

2.12.2 Перечень сообщений об ошибках

— **Ошибка инициализации GLFW:** Программа не может инициализировать GLFW.
— **Ошибка создания окна:** Программа не может создать окно.
— **Ошибка нехватки вершин:** $lat\_segments * (lon\_segments + 1)$ должно быть больше либо равно 16 для корректного отображения фигур.

3 Экспериментальная часть

3.1 Демонстрация работы программы

Разработанная программа успешно реализует морфинг между кубом и сферой с использованием трёхмерных объектов, описанных треугольниками (`GL_TRIANGLE_STRIP`). На рисунке 1 показаны промежуточные этапы морфинга, включая начальную форму (куб), конечную форму (сфера) и несколько состояний между ними.

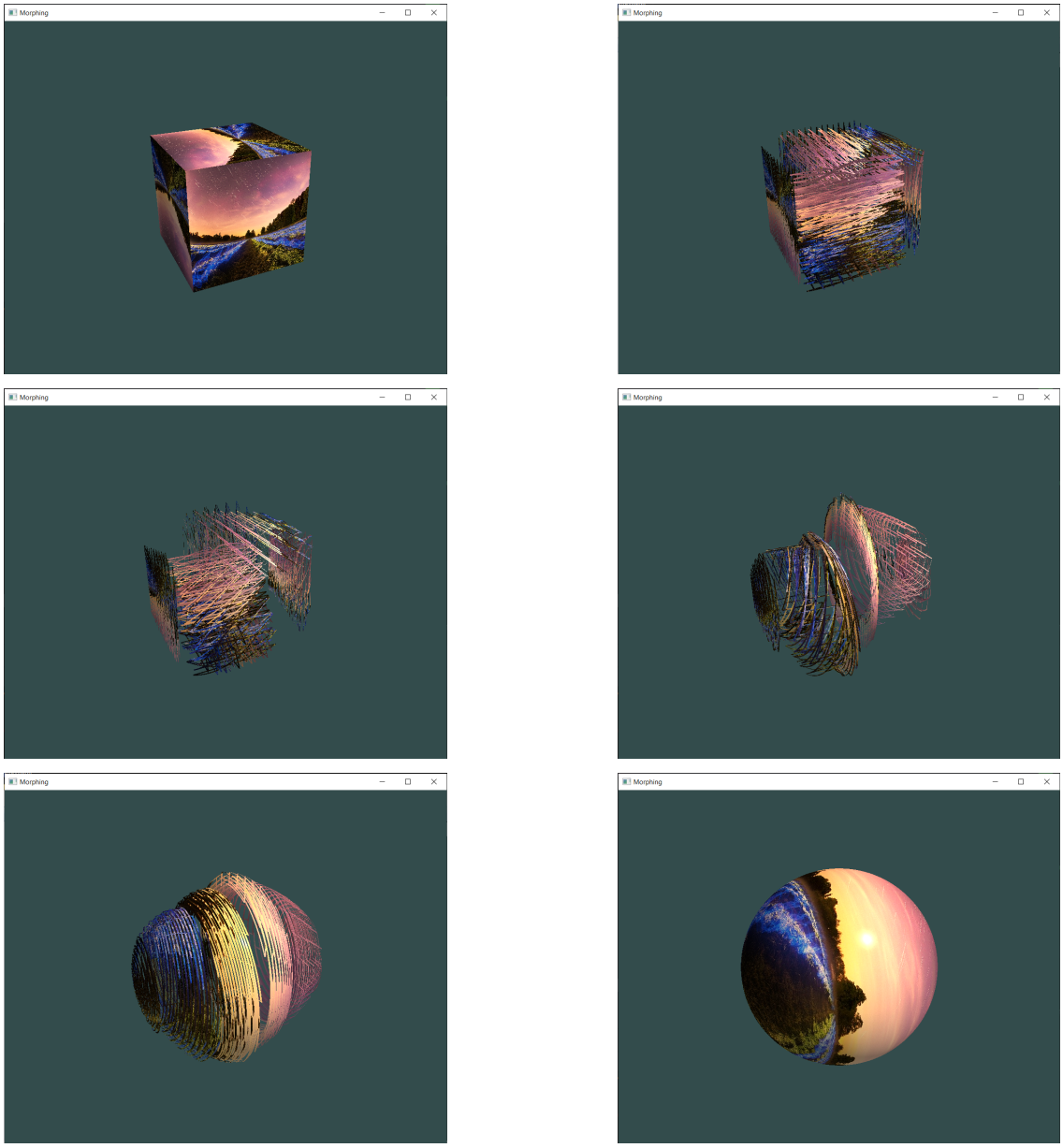


Рисунок 1 — Демонстрация работы программы: промежуточные этапы морфинга.

Пользователь может взаимодействовать с программой в реальном времени, управляя следующими параметрами:

— включение и отключение морфинга (нажатие клавиши `M`);
— вращение объекта с помощью движения мыши;
— изменение масштаба прокруткой колеса мыши;
— переключение между отображением каркаса и заполненной модели (нажатие клавиши `F`);
— управление скоростью морфинга клавишами `+` и `-`.

3.2 Визуальная оценка качества

Для оценки визуального качества морфинга были проведены тесты с различным количеством сегментов сферы. Результаты показали, что при увеличении числа сегментов улучшается плавность переходов, однако возрастают требования к производительности при создании вершин куба и сферы. Оптимальным было выбрано значение 200 полос широты и 249 сегментов долготы, что обеспечивает баланс между качеством изображения и скоростью работы программы.

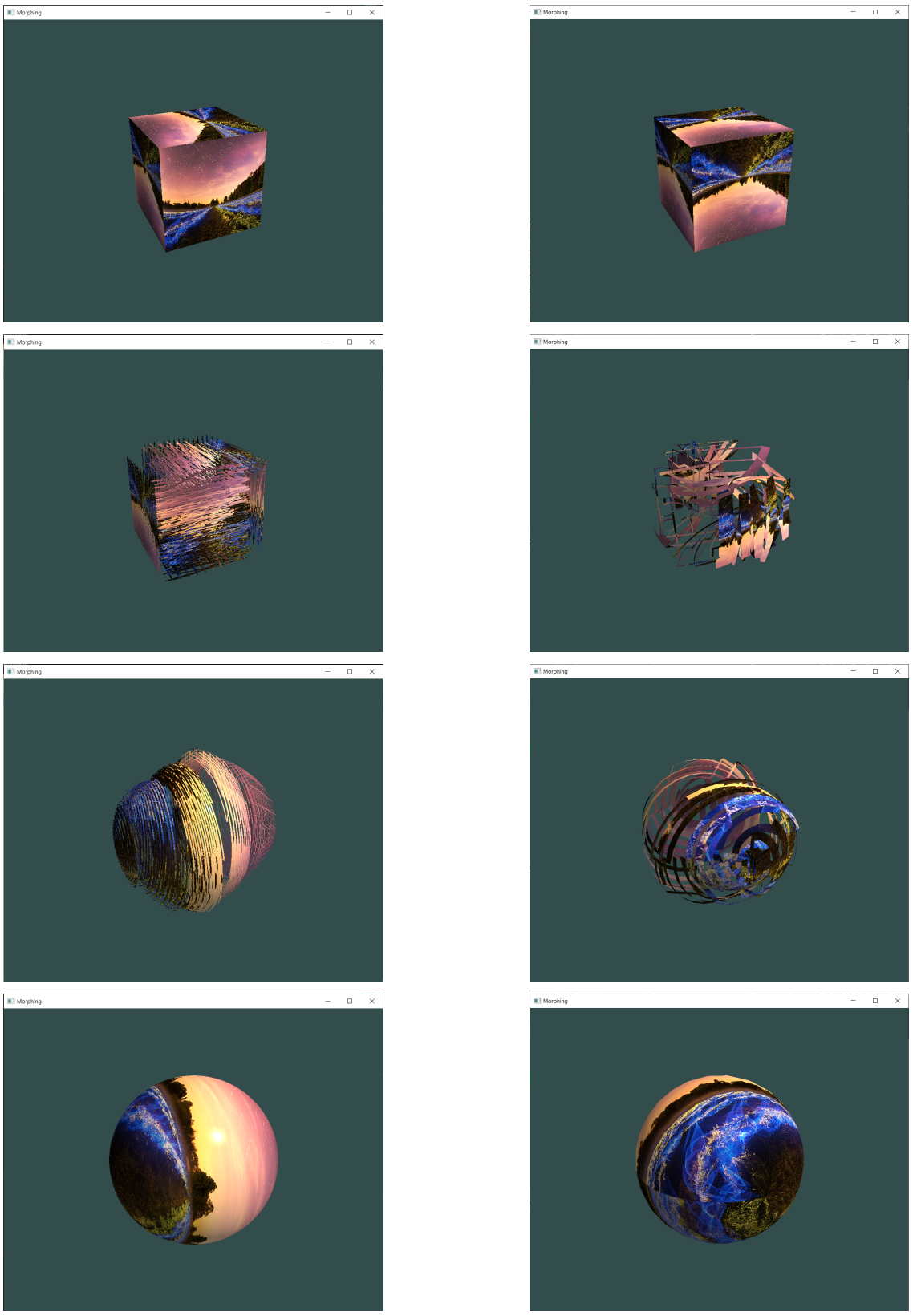
На рисунке 2 приведено сравнение объектов с различным уровнем детализации.

3.3 Оценка производительности

Производительность программы была протестирована на нескольких конфигурациях компьютеров, включая системы с различными характеристиками графических процессоров. Важными метриками производительности стали:

— частота кадров (FPS);
— задержка при изменении формы;
— загрузка GPU при разных уровнях детализации объектов.

На рисунке 3 показана зависимость частоты кадров от количества вершин в моделях. Видно, что даже при значительном увеличении числа вершин (до 100,000) программа демонстрирует стабильную производительность благодаря оптимизированной обработке на GPU.



100000 вершин 1800 вершин

Рисунок 2 — Сравнение уровня детализации объектов при разных параметрах.

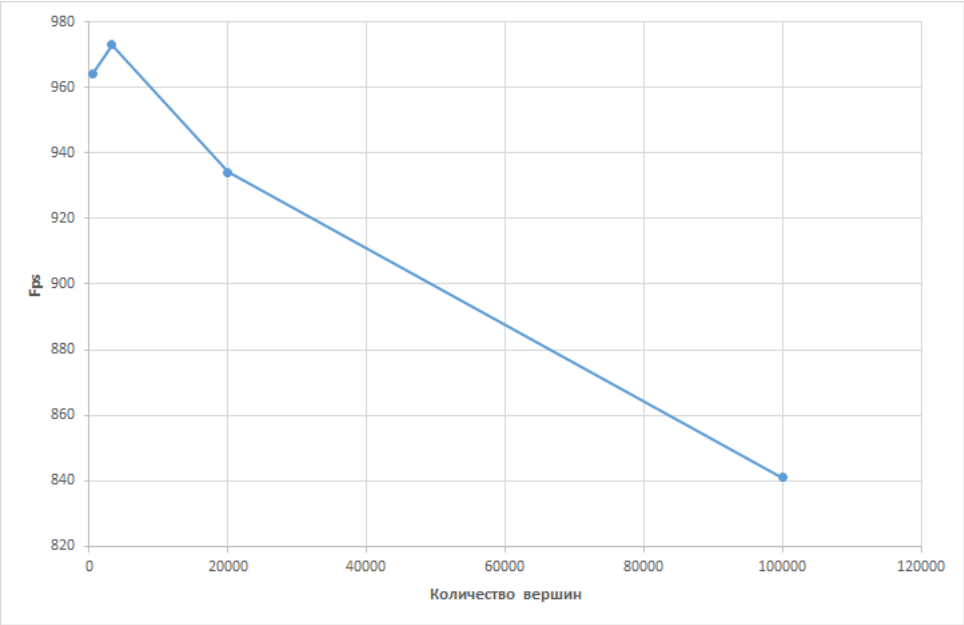


Рисунок 3 — График зависимости частоты кадров от числа вершин.

3.4 Анализ результатов

Результаты экспериментов показали, что разработанная программа успешно реализует морфинг между кубом и сферой, обеспечивая плавные и реалистичные переходы. Визуальная оценка качества морфинга показала, что при увеличении числа сегментов сферы улучшается плавность переходов, однако возрастают требования к производительности. Оптимальное значение в 200 полос широты и 249 сегментов долготы было выбрано также для более быстрого запуска программы.

Производительность программы была протестирована на нескольких конфигурациях компьютеров, и результаты показали, что программа демонстрирует стабильную производительность даже при значительном увеличении числа вершин благодаря оптимизированной обработке на GPU.

3.5 Обсуждение возможных улучшений

Несмотря на успешную реализацию морфинга, существуют возможности для улучшения программы. Одним из таких улучшений может быть использование более сложных математических моделей для интерполяции, таких как бикубическая интерполяция или интерполяция с использованием радиальных базисных

функций. Это позволит добиться еще более плавных и реалистичных переходов между формами объектов.

Другим возможным улучшением может быть использование адаптивных сеток для динамического изменения разрешения геометрии объектов в зависимости от сложности сцены. Это позволит оптимизировать вычисления и улучшить производительность программы при работе с объектами различной сложности.

Также можно рассмотреть возможность использования параллельных вычислений и многопоточности для ускорения обработки данных. Это позволит улучшить производительность программы и обеспечить более плавную анимацию даже при сложных преобразованиях.

ЗАКЛЮЧЕНИЕ

В ходе данной работы был изучен и реализован алгоритм морфинга трёхмерных объектов на примере перехода от куба к сфере. Проект позволил углубиться в основы работы с 3D-графикой и изучить применение современных инструментов, таких как OpenGL, для создания анимационных эффектов.

Основные результаты работы:

— Изучены теоретические основы морфинга, включая его применение в различных областях компьютерной графики.

— Реализованы трёхмерные объекты (куб и сфера) с использованием треугольников (`GL_TRIANGLE_STRIP`) для повышения производительности рендеринга.

— Разработан алгоритм морфинга, обеспечивающий плавный переход между объектами, и реализован на уровне GPU с помощью шейдеров.

— Обеспечено интерактивное управление параметрами морфинга, включая изменение скорости, вращение и масштабирование объектов.

— Проведены эксперименты, продемонстрировавшие стабильную производительность программы и высокое визуальное качество при оптимальном количестве сегментов.

Разработанная программа имеет несколько ограничений, которые могут быть устранены в дальнейшем:

1. Добавление поддержки более сложных форм для морфинга, например, моделей, импортированных из внешних файлов.
2. Реализация нелинейного морфинга для более реалистичных преобразований.
3. Использование текстур высокого разрешения для улучшения визуального восприятия.
4. Оптимизация кода для работы на устройствах с ограниченными ресурсами.

Работа над этим проектом позволила не только закрепить теоретические знания в области компьютерной графики, но и получить практические навыки в программировании 3D-анимации. Предложенный подход может быть использован

в дальнейшем для создания более сложных визуальных эффектов и интерактивных приложений.

Таким образом, поставленные цели и задачи были выполнены, что подтверждает успешность проведённого исследования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Foley J. D., Hughes J. F., Dam A. van. Computer Graphics: Principles and Practice. — Boston : Addison-Wesley Professional, 2013.
2. Goulekas K. E. Visual Effects in a Digital World: A Comprehensive Glossary of over 7000 Visual Effects Terms. — San Francisco : Morgan Kaufmann, 2001.
3. Ryan D. History of Computer Graphics: DLR Associates Series. — Scotts Valley, CA : CreateSpace Independent Publishing Platform, 2011.
4. Haque H., Hassaniem A.-E., Nakajima M. Generation of Missing Medical Slices Using Morphing Technology // IEICE Transactions on Information & Systems. — 2000. — Июль. — Т. E83—D, № 8. — С. 1400—1407.
5. Vince J. Mathematics for Computer Graphics. — Fifth Edition. — Breinton, UK : Springer-Verlag London Ltd., 2017.
6. Гонсалес Р. и Вудс Р. Цифровая обработка изображений. — 3-е издание. — Москва : Williams Publishing House, 2012.
7. Mukundan R. Advanced Methods in Computer Graphics: With examples in OpenGL. — London, Dordrecht, Heidelberg, New York : Springer London Ltd., 2012.
8. Bergan R. The Film Book: A Complete Guide to the World of Film. — New York, New York : DK Publishing, 2011.

ПРИЛОЖЕНИЕ