



ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки

09.03.02 «Информационные системы и технологии»

Отчёт по практике «Преддипломная»

Выполнил(а): студент(ка) группы 211-7212

Крылов Д.А.

(Фамилия И.О.)

Дата, подпись 07.05.25
(дата)


(подпись)

Проверил: _____

Дата, подпись _____
(дата) (подпись)

Москва
2025

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
Список использованных сокращений и обозначений	3
Введение.....	4
Основная часть.....	6
Глава 2 Конструктивная глава	6
Глава 3 Технологическая глава	23
Приложение.....	44
Список литературы:	46

Список использованных сокращений и обозначений

НИР – научно-исследовательская работа

ИТ – информационные технологии

ПО – программное обеспечение

GPU – графический процессор

ИПС – информационно-программной система

Введение

Развитие цифровых технологий требует создания эффективных инструментов для работы с графическими объектами, обеспечивающих функциональность и удобство. Существующие 2D-редакторы обладают значительным набором возможностей, однако интеграция 3D-элементов часто осуществляется с учетом ограничений, что делает эти инструменты неудобными для пользователей, привыкших работать исключительно с двухмерной графикой.

Большинство профессиональных 3D-редакторов требуют значительных вычислительных ресурсов, что ограничивает их доступность для пользователей с менее мощными устройствами. Это создает потребность в разработке нового программного обеспечения, которое объединяет возможности 2D-редактирования и базовые функции 3D-моделирования, обеспечивая при этом высокую производительность на различных уровнях вычислительных мощностей.

Современные потребности дизайнеров, художников и специалистов анимационной индустрии требуют наличия интуитивно понятного и гибкого инструмента, обеспечивающего работу с различными кистями, слоями и базовыми 3D-операциями. Это позволит эффективно работать с графическими объектами без необходимости использования ресурсоемких профессиональных пакетов.

Актуальность темы обусловлена растущей потребностью в удобных и универсальных инструментах для графического редактирования, сочетающих функционал двухмерного и трехмерного рисования. Существующие решения, хотя и предлагают широкий спектр возможностей для работы с 3D, зачастую оказываются неудобными для пользователей, привыкших к 2D-редактированию.

Основные сложности при разработке связаны с подходом к управлению программой, поскольку традиционные интерфейсы 3D-редакторов могут быть недостаточно удобны для пользователей, имеющих опыт работы исключительно с 2D-графикой. Это накладывает дополнительные требования к разработке интуитивного управления, позволяющего легко интегрировать 3D-элементы в процесс рисования без необходимости глубоко осваивать трехмерное моделирование.

Также важно оптимизировать программное обеспечение таким образом, чтобы оно могло эффективно работать на устройствах с разными уровнями производительности, обеспечивая плавную работу и отзывчивый интерфейс.

Основная часть

Глава 2 Конструктивная глава

2.1. Целесообразность разработки, семантические и информационно-логические модели

Веб-приложение нового поколения строит свою архитектуру на единой модели предметной области, объединяющей семантику данных и их информационно-логические связи. Семантическая модель описывает ключевые элементы — трёхмерные объекты с их атрибутами (название, тип, геометрия, координаты), камеры и источники света, а также возможные аннотации, привязанные к объектам.

Информационно-логическая модель развивает эту схему, задавая форматы хранения (координаты как числа с плавающей точкой, текстовые аннотации в JSON), протоколы передачи (REST-API для загрузки сцен, WebSocket для синхронизации), а также правила связи между коллекциями данных (многие-к-одному для аннотаций и объектов, один-ко-многим для сцены и её элементов).

Такой подход позволяет автоматизировать наполнение и обновление сцены, снижая ручную рутинную работу и повышая точность при создании сложных многослойных композиций.

Объединение описания сущностей и детального описания потоков данных ускоряет процесс проектирования, упрощает масштабирование функционала и гарантирует высокую производительность на различных аппаратных платформах.

Согласно Watt [12], эффективные графические системы должны минимизировать сложность перехода от 2D к 3D, предоставляя интуитивно понятные инструменты для художников. Пользователи, знакомые с 2D-редакторами, получают интуитивно понятный интерфейс для базовых 3D-операций: достаточно выбрать объект, настроить его атрибуты и увидеть мгновенный результат в сцене, без необходимости глубоко погружаться в нюансы 3D-программирования.

2.2. Архитектурные и технологические особенности реализации

Разрабатываемое графическое веб-приложение ориентировано на клиентскую реализацию с акцентом на модульность, расширяемость и кроссплатформенность.

Базовая архитектура предусматривает начальную реализацию инструментов двумерной графики и элементарных операций с трёхмерными объектами. В качестве среды исполнения выбраны современные веб-браузеры (например, Google Chrome, Mozilla Firefox, Яндекс.Браузер), что обеспечивает переносимость и снижение зависимости от платформы.

Архитектура спроектирована с расчётом на последовательное расширение функциональности, включая поддержку интерактивного интерфейса и адаптацию к различным устройствам ввода. Расширение возможностей планируется в рамках следующих итераций разработки, с постепенным переходом к настольной версии редактора.

В качестве методологической основы при проектировании архитектуры используется подход, изложенный в труде Роджерса и Адамса [14]. Авторы подчёркивают важность декомпозиции программной системы на независимые модули с минимальной связанностью и чёткой внутренней логикой.

Такое структурирование позволяет повысить устойчивость приложения к изменениям, упростить тестирование отдельных компонентов и обеспечить гибкость масштабирования.

Особое внимание в рамках первой версии уделяется обеспечению кроссплатформенности за счёт использования веб-технологий. Приложение разрабатывается с прицелом на работу в современных браузерах (включая Google Chrome и Яндекс.Браузер), что позволяет достичь широкой совместимости без необходимости установки клиентского ПО. Такой подход снижает порог входа для пользователей и упрощает распространение обновлений.

Производительность интерфейса обеспечивается за счёт использования аппаратного ускорения рендеринга (через OpenGL/WebGL), что позволяет сократить задержки при отображении и трансформации графических объектов. В данном контексте под интерактивным временем отклика понимается задержка, не превышающая порог, воспринимаемый пользователем как мгновенный отклик (в среднем до 100 мс). Данный показатель критичен для обеспечения комфортного взаимодействия, особенно при работе с многослойными сценами и в условиях ограниченных вычислительных ресурсов.

Дополнительно учитываются особенности взаимодействия с различными устройствами ввода. В текущей версии основное взаимодействие осуществляется с использованием мыши и клавиатуры, что обеспечивает точность и контроль при работе с графическими элементами. При этом архитектура интерфейса спроектирована с возможностью последующего расширения функциональности — предусмотрен потенциал для адаптации под сенсорные экраны и стилусы.

Такой подход соответствует принципам универсального дизайна и формирует базу для обеспечения согласованного пользовательского опыта на различных платформах (рис. 8, 9, 10).

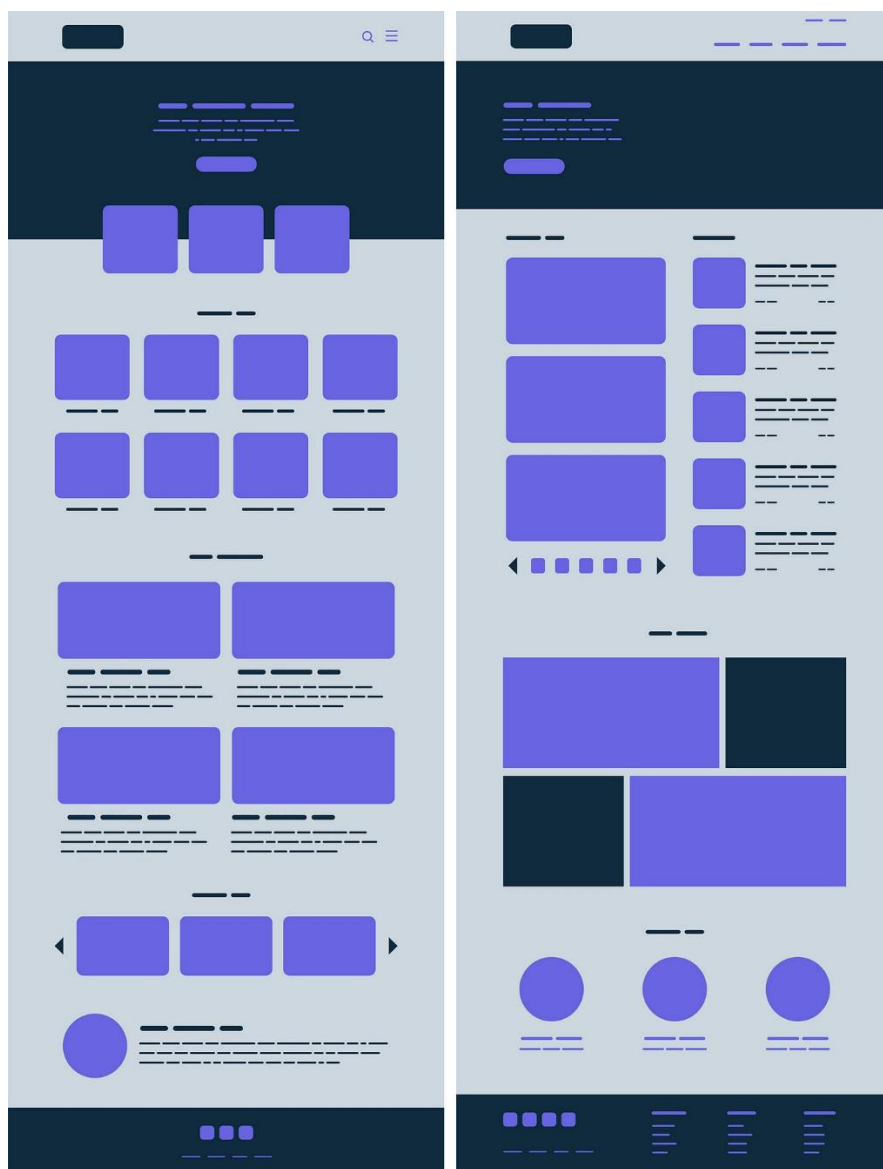


рис.8 “Пример Гештальт-принципов в дизайне интерфейсов”



рис.9 “Пример интерфейса в модуле Рисования”

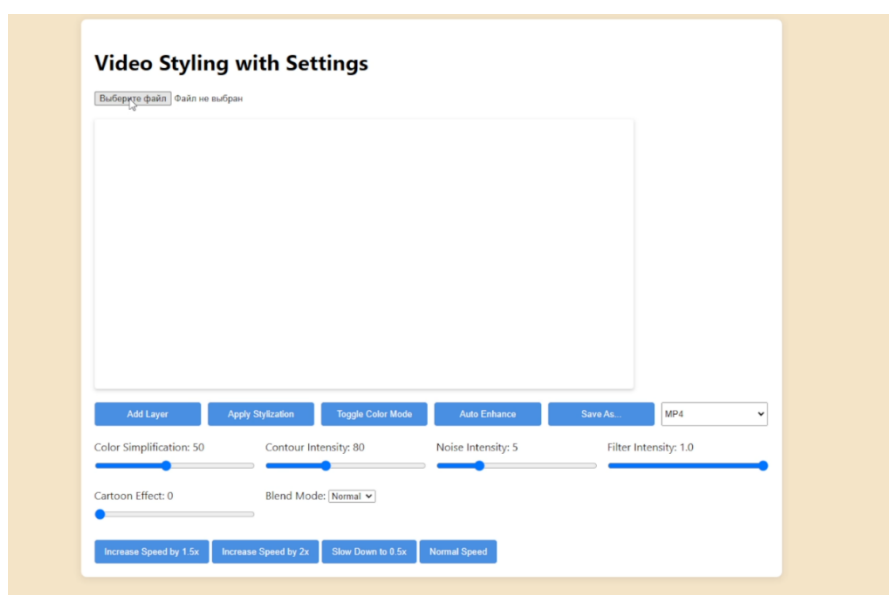


рис.10 “Пример интерфейса в модуле Видеоредактора alpha-version”

2.3. Функциональные требования

Разрабатываемый проект ориентирован на обеспечение высокой гибкости в обработке графического контента и эффективное управление многослойными сценами с возможностью взаимодействия с 3D-объектами.

1. Базовая функциональность двумерной графики

- Инструментарий для создания и редактирования графических примитивов: линия, прямоугольник, эллипс, многоугольник, произвольные кривые.
- Настройка параметров объектов: цвет, толщина, прозрачность.
- Свободное рисование с использованием кистей, инструментов заливки и ластика.
- Управление цветовой палитрой; поддержка градиентных заливок.
- Поддержка разнообразных форматов.
- Реализация механизма отмены и возвращения действий.

2. Элементы трёхмерной визуализации

- Создание и отображение базовых трёхмерных примитивов: куб, сфера, плоскость.
- Базовые трансформации: перемещение, масштабирование, вращение объектов.
- Регулировка параметров освещения и перспективной проекции.
- Импорт 3D-моделей в формате FBX.
- Привязка двумерных элементов к 3D-объектам в рамках сцены.

3. Многослойная структура сцены

- Неограниченное количество слоёв с возможностью управления порядком отображения, прозрачностью, блокировкой и скрытием.
- Организация иерархии объектов и группировка элементов.

4. Пользовательский интерфейс

- Адаптивная структура интерфейса с возможностью корректной работы на ПК, ноутбуках и устройствах с сенсорным вводом.
- Поддержка взаимодействия через клавиатуру и мышь.

5. Производительность и устойчивость

- Поддержка интерфейсного отклика с задержкой не более 100 мс при активной работе с множеством слоёв.
- Использование многопоточности и аппаратного ускорения на базе WebGL(рис. 11).
- Автоматическое сохранение данных и восстановление сессии.
- Возможность автономной работы без постоянного сетевого подключения.

рис.11 “Пример работы WebGL с компьютером”



6. Архитектурные свойства и расширяемость

- Модульная архитектура с логическим разделением между графическим движком, пользовательским интерфейсом и подсистемой управления данными.
- Поддержка подключения внешних плагинов.
- Возможность расширения функциональности через API и интеграции с внешними библиотеками.
- Планируемая интеграция поддержки скриптов на Python через соответствующие интерфейсы.

7. Дополнительные возможности

- Поддержка взаимодействия с графическими планшетами находится в стадии разработки.
- Рассматривается внедрение механизмов учёта силы нажатия и угла наклона пера в целях повышения точности и выразительности визуального редактирования.

2.4. Нефункциональные требования

Нефункциональные требования определяют качественные характеристики разрабатываемой системы, необходимые для её надёжной, удобной и эффективной эксплуатации в условиях реального применения. Эти требования охватывают производительность, надёжность, масштабируемость, совместимость, безопасность и удобство использования.

1. Производительность

- Время отклика интерфейса: не более 100 мс при стандартных действиях (рисование, переключение инструментов, перемещение объектов).
- Обработка сцен: до 50 слоёв и 1000 объектов без снижения производительности на системе с рекомендованной конфигурацией.
- Рендеринг: частота обновления интерфейса не ниже 60 FPS на устройствах с поддержкой WebGL.
- Загрузка проекта: не более 3 секунд для типовых файлов объёмом до 50 МБ.

2. Надёжность и устойчивость

- **Стабильность:** система должна сохранять работоспособность в течение не менее 8 часов непрерывной работы без сбоев.
- **Обработка ошибок:** предусмотрена защита от критических сбоев и автоматическое восстановление последнего сохранения после аварийного завершения работы.
- **Автосохранение:** периодическое сохранение текущего состояния проекта (по умолчанию каждые 5 минут, с возможностью настройки).

3. Масштабируемость

- **Программная архитектура** должна поддерживать расширение функционала без полной переработки существующего кода (plug-in система).
- Возможность добавления новых модулей, форматов данных, инструментов, фильтров и функций через внешние библиотеки и API.
- Горизонтальное масштабирование серверной части при использовании сетевых функций (в перспективе)

4. Совместимость и кроссплатформенность

- **Браузеры:** корректная работа в последних версиях Google Chrome, Mozilla Firefox, Microsoft Edge и Safari.
- **Мобильные устройства:** адаптация под Android 10+ и iOS 13+ (через веб-версию).
- **Разрешения экрана:** интерфейс корректно отображается на дисплеях от 1024×768 до 4K (3840×2160).

5. Безопасность

- **Конфиденциальность данных:** локальное хранение пользовательских данных без передачи третьим лицам.
- **Защита от внешних воздействий:** базовая проверка входных данных, защита от XSS/CSRF-уязвимостей в веб-версии.
- **Обновления:** система должна регулярно получать обновления безопасности, особенно при использовании open-source компонентов.
- **Резервное копирование:** поддержка экспорта резервной копии проекта в зашифрованном виде.

6. Удобство использования (usability)

- **Кривая обучения:** освоение основных функций должно занимать не более 30 минут при наличии пользовательской инструкции.
- **Локализация:** поддержка многоязычного интерфейса (русский, английский, опционально другие).
- **Поддержка пользовательских тем:** возможность выбора цветовой схемы интерфейса (светлая, тёмная).
- **Помощь и обучение:** встроенные подсказки, интерактивные туториалы, документация и FAQ.

7. Лицензирование и открытость

- **Модель распространения:** open-source (лицензия MIT/GPL), исходный код доступен на GitHub.
- **Сообщество:** возможность участия пользователей в разработке, предложение улучшений и доработка модулей.
- **Поддержка:** осуществляется через систему тикетов, GitHub и дискуссионные форумы.

2.5. Общее описание алгоритма решения задачи

Алгоритм функционирования системы описывает последовательность этапов, обеспечивающих выполнение ключевых функций интерактивного веб-приложения для визуализации.

Он формирует логическую основу для обработки пользовательских данных, отображения информации и взаимодействия с графическим интерфейсом, закладывая фундамент для последующей программной реализации.

На первом этапе происходит инициализация системы: загружается пользовательский интерфейс, конфигурационные параметры, а также базовые компоненты сцены, включая рендерер, камеру, освещение и начальные объекты. Настраивается графическое окружение, и система подготавливается к пользовательскому вводу.

Следующий этап связан с формированием сцены и индивидуальной настройкой инструментов взаимодействия. Особое внимание уделяется конфигурации параметров кисти: пользователю предоставляется возможность выбрать форму, размер, силу давления, прозрачность и тип воздействия (например, окрашивание, сглаживание или деформация). Такие параметры могут меняться в процессе работы, позволяя добиться более точного и выразительного результата при взаимодействии с 2D/3D-поверхностями сцены.

Сама работа пользователя организована по событийной модели. Каждое действие — перемещение курсора, нажатие, изменение параметров кисти — вызывает соответствующее обновление состояния.

Сцена перерисовывается в режиме реального времени, что обеспечивает непрерывную визуальную обратную связь.

Завершающий этап алгоритма включает сериализацию текущего состояния проекта, экспорт данных в подходящем формате и завершение пользовательской сессии. При необходимости данные могут быть переданы на сервер или сохранены локально.

Таким образом, алгоритм охватывает весь жизненный цикл работы приложения: от запуска и настройки до интерактивного взаимодействия и финального сохранения. Система ориентирована на точную и гибкую работу с параметрами кисти, что особенно важно при создании визуального контента, требующего индивидуального подхода к каждому элементу сцены.

В детальном обзоре [3] представлены основные этапы работы системы, действия пользователя и соответствующие реакции системы, включая инициализацию, настройку параметров, интерактивное взаимодействие с объектами сцены, обновление отображения и управление проектом.

Выводы

На начальном этапе разработки проводился детальный анализ целевых ориентиров проекта, с фокусом на создание универсального инструмента для визуального редактирования, сочетающего возможности двумерной и базовой трёхмерной графики в рамках одного пользовательского интерфейса.

В качестве принципиального архитектурного решения была заложена концепция единого поля редактирования, обеспечивающего когерентность взаимодействия в различных режимах визуализации.

Такое решение позволило устранить традиционные ограничения, связанные с разделением рабочих пространств для различных типов графических данных, и тем самым повысить гибкость и адаптивность системы.

Разработка функциональных и нефункциональных требований производилась в строгом соответствии с поставленными целями.

Среди ключевых параметров, определяющих поведение системы, были выделены: обеспечение высокой отзывчивости интерфейса в условиях динамической нагрузки, совместимость с технологиями WebGL для рендеринга трёхмерной графики, поддержка иерархической структуры объектов (включая возможность создания слоёв), а также способность к последовательной модификации и расширению существующих компонентов без нарушения общей целостности проекта.

Особое внимание было уделено механизму инкрементального обновления компонентов — в рамках которого каждое последующее изменение не только сохраняет ранее заданные свойства объекта, но и предоставляет расширенные опции взаимодействия с ним. Такой подход реализует принцип «редактируемости без разрушения», что критически важно для креативных задач, требующих множественных итераций.

Следующим этапом стала разработка концептуальных моделей, формализующих внутренние процессы обработки графических данных. Эти модели учитывали необходимость обеспечения синхронной поддержки как 2D-, так и 3D-объектов, что потребовало внедрения гибкой системы типов и механизмов трансформации контента внутри единой среды.

Архитектура приложения была спроектирована таким образом, чтобы обеспечить масштабируемость и возможность дальнейшего внедрения специализированных модулей анализа, рендеринга и пользовательского взаимодействия.

Глава 3 Технологическая глава

3.1. Обоснование архитектурных решений и выбора технологий визуализации

Формирование требований к модулю визуализации информационно-программной системы (ИПС) обусловлено необходимостью реализации интерактивного трёхмерного представления элементов предметной области с поддержкой операций редактирования, аннотирования, трансформации и аналитического взаимодействия. Указанный компонент должен быть функционально и архитектурно интегрирован с подсистемами хранения, логики интерфейса и модулями пользовательского взаимодействия.

С учётом специфики предметной области, визуализирующий модуль представляет собой механизм отображения и манипуляции множеством объектов и их связей. Объекты включают геометрические примитивы, составные модели и пространственные зависимости, сопровождаемые метаинформацией (аннотациями и состояниями). Модуль обеспечивает выполнение следующих ключевых задач:

- отрисовку сцены с учётом иерархической структуры объектов;
- манипулирование параметрами объектов и виртуальной камеры;
- отображение пользовательских аннотаций и подсказок;
- генерацию визуальной обратной связи при редактировании данных.

Для реализации данных функций в качестве основы выбран движок визуализации **Three.js**, построенный на WebGL, обеспечивающий требуемый баланс между производительностью, гибкостью и совместимостью с актуальными веб-технологиями. Применение указанного решения обеспечивает достижение заявленных в разделе 2.4 параметров производительности, включая рендеринг до 1000 объектов в сцене с частотой не менее 60 кадров в секунду.

Технические и архитектурные аргументы в пользу выбора Three.js:

- Поддержка низкоуровневого API (шейдеры, буферы) и высокоуровневых абстракций (Mesh, Light, Group), позволяющая реализовать адаптивные уровни детализации;
- Совместимость с популярными форматами 3D-данных (GLTF, FBX, OBJ), включая поддержку материалов PBR;
- Развитая документация и устойчивая экосистема, способствующие снижению стоимости сопровождения и порога вхождения;
- Наличие интеграционного слоя с React посредством `@react-three/fiber` и вспомогательных утилит (`react-three-drei`), поддерживающих декларативную модель сборки;
- Встроенные механизмы оптимизации визуализации: LOD, instancing, frustum culling, динамическая загрузка ассетов.

Архитектурная интеграция и организационные принципы:

- Инкапсуляция графического движка в Canvas-компонент React с управлением жизненным циклом через хуки;
- Модульная декомпозиция сцены на независимые функциональные единицы (объекты, источники освещения, управляющие элементы), реализуемые как изолированные компоненты;
- Реализация пользовательского взаимодействия через raycasting, drag'n'drop и событийную модель React;
- Централизация состояния сцены с использованием глобального хранилища (Zustand / Redux Toolkit), обеспечивающего интеграцию с другими модулями ИПС;
- Получение входных данных от серверной части (через REST или GraphQL), формирование выходных сериализованных представлений сцены и логов взаимодействия.

Выбор технологий и языковых средств:

- Языки реализации: JavaScript / TypeScript;
- Применяемые фреймворки и библиотеки: React, Three.js, @react-three/fiber, Zustand;
- При необходимости серверного рендеринга (SSR): использование Next.js;
- Архитектура: компонентно-ориентированная, с разделением визуальных, логических и сервисных слоёв;
- Возможность расширения функциональности за счёт плагинов, сторонних библиотек и API-интерфейсов.

Дополнительные технологические аспекты:

- Локальное кэширование ресурсов на клиенте с целью ускорения повторного доступа;
- Использование OffscreenCanvas и Web Workers для распределения вычислительной нагрузки;
- Наличие fallback-механизмов при невозможности использования WebGL (например, отображение заглушек).

Обоснование реализуемости:

Выбранный технологический стек обладает полной совместимостью с современными браузерами и поддерживает кроссплатформенную работу (включая мобильные устройства). Используемые программные компоненты являются зрелыми, активно развивающимися и протестированными в реальных условиях. Архитектура визуализирующего модуля допускает масштабирование, расширение функциональности и обеспечивает требуемые показатели надёжности, отклика и совместимости.

3.2. Детальное устройство веб-приложения

1.2.1. Технологические процессы обработки информации

В модуле **Draw** реализован событийно-ориентированный механизм обработки пользовательского ввода. При нажатии и движении мыши над канвой (canvas) генерируются события, которые преобразуются в 3D-координаты на плоскости рисования с помощью лучевого трассирования (Raycaster). При событии mousedown проверяется наличие как минимум одного слоя и вычисляется начальная точка линии (startLine), при mousemove при активном режиме рисования (isDrawing) вычисляются новые точки.

Каждое действие рисования сохраняется в виде структурированного объекта { start, end, brushType, currentLayerIndex } в массив strokes для возможности отмены/возврата. На основе этих данных создаются геометрические примитивы: при выборе «обычной» кисти строится кривая между точками, при распылителе – набор точек (PointCloud), акварелью и масляной кистью – концентрические окружности (многоугольники с прозрачностью). Эти объекты добавляются в сцену scene библиотеки Three.js.

Для поддержки отмены/возврата реализованы два стека: при **Undo** последний элемент из strokes перемещается в redoStack и канвас перерисовывается методом redrawCanvas(), который удаляет текущие объекты сцены и воспроизводит все оставшиеся штрихи; при **Redo** – обратно. Слои реализованы как массив имен layers и соответствующий объект layerObjects, хранящий графические объекты по слоям. При добавлении/удалении слоя модифицируется этот массив, а при удалении конкретного слоя все связанные с ним объекты убираются из сцены через scene.remove().

Переключение между режимами **2D** и **3D** реализовано через кнопку. При переключении система анимированно перемещает камеру (camera) из позиции «сверху» (режим 2D) к наклонной позиции (режим 3D) с помощью библиотеки




Anime.js. Одновременно меняется режим управления камерой (OrbitControls): в 2D режиме вращение отключено, в 3D режиме по нажатию Shift+правой кнопки мыши включается поворот сцены. Пользователь видит индикатор режима («Рисование» или «Вращение камеры»), который обновляется при нажатии/отпуске клавиши Shift.

Вспомогательный модуль **Video** выполняет обработку видеопотока кадр за кадром. После загрузки видеофайла каждый кадр выводится на элемент `<canvas>` и анализируется построчно: пиксели подвергаются квантизации цвета (уменьшение числа тонов по ползунку «Color Simplification»), двоичной фильтрации контуров (по пороговому значению «Contour Intensity»), наложению шума (ползунок «Noise Level») и эффекту «мультифильтра» (ползунок «Cartoon Effect»). Затем цветовое пространство может переключаться в градации серого и обратно. Для каждого слоя наложения (создается кнопкой «Add Layer») используется свой контекст canvas с настраиваемыми непрозрачностью и режимом смешивания (blend mode) – это позволяет создавать композитные изображения. Готовый кадр рисуется с учётом этих параметров, после чего запускается следующий кадр через `requestAnimationFrame`, обеспечивая непрерывную потоковую обработку. Таким образом, технологический процесс в модуле Video – это циклический сбор кадра, его pixel-level фильтрация и обновление экрана.

1.2.2. Описание интерфейса взаимодействия

Интерфейс **Draw** состоит из двух основных областей: **панели инструментов (toolbar)** и **области рисования (canvas)**. В панели инструментов расположены кнопки выбора типа кисти и её настроек. Кнопка «Кисть» открывает выпадающее меню выбора формы кисти (круглая, квадратная, спрей, акварель, масло, ластик).

Есть кнопка «Настройки», при нажатии которой раскрывается меню с ползунками и полями ввода для параметров кисти: толщина (диапазон 1–20), цвет (палитра), непрозрачность (0–100%), яркость и контрастность (диапазоны –100...100). Кнопка «Переключить на 3D» меняет режим работы между двумерным и трёхмерным. Слева панели отображается индикатор режима («Режим: Рисование» или «Вращение камеры»), видимый в 3D-режиме при нажатом Shift.

Ниже панели инструментов располагается **панель слоёв (layerPanel)**. В ней кнопка «Добавить слой» создаёт новый слой с уникальным именем («Слой 1», «Слой 2» и т.д.), а «Удалить слой» убирает активный слой. Кнопки Undo () и Redo () находятся рядом и позволяют откатить/вернуть последнее действие. Под ними выводится список слоёв (`<div class="layer-list">`), динамически обновляемый функцией `updateLayerList()`: каждый элемент списка содержит название слоя и кнопку «» для удаления конкретного слоя. Активный слой становится последним созданным.

Основная область приложения – канвас с идентификатором `unifiedCanvas`, занимающий весь доступный экран. Здесь реализован WebGL-рендерер Three.js, который в 2D-режиме показывает плоскость рисования, а в 3D-режиме – наклонную проекцию с возможностью поворота. Подключены библиотеки: `three.js` (WebGL), `OrbitControls` (для вращения), `three-mesh-line` (для отрисовки линий), `Anime.js` (для анимации перехода камеры).

Модуль **Video** содержит веб-страницу с формой загрузки видео (кнопка файлового ввода) и панелью эффектов. В верхней части – заголовок и поле `<input type="file" accept="video/*">` для выбора видео. Ниже – контейнер с элементами `<video>` (скрыт по умолчанию) и `<canvas>` для вывода обработанного видео. Далее горизонтальная панель содержит кнопки управления: «Add Layer» (добавление слоя наложения), «Apply Stylization»




(применение эффектов к кадрам), «Toggle Color Mode» (переключение цвет/ч/б), «Auto Enhance» (автоналадка ползунков), «Save As...» (сохранение), выпадающий список формата (MP4/GIF) и элементы управления скоростью воспроизведения. Ниже идут группы ползунков настроек: Color Simplification, Contour Intensity, Noise Level, Filter Intensity, Cartoon Effect и выпадающий список Blend Mode – все они связаны с соответствующими переменными скрипта. В самом низу – панель управления скоростью (кнопки для изменения playback rate). Слева находится <div id="layerContainer">, куда динамически добавляются элементы управления для каждого слоя (ползунков opacity, список blend mode, кнопка Remove). Интерфейс модульный, все элементы подписаны на русском языке или понятны по стандартным иконкам, обеспечивая понятное взаимодействие пользователя с системой.

1.2.3. Руководство пользователя

Пользовательское руководство описывает шаги работы с приложением. Для модуля **Draw** основные действия следующие:

1. **Добавление слоя:** Нажмите «Добавить слой». В списке слоёв появится «Слой 1». Повторное нажатие создаст «Слой 2» и т.д. Активным всегда считается последний созданный.
2. **Выбор инструмента (кисти):** Нажмите кнопку «Кисть», выберите в выпадающем меню нужный тип (круглая, квадратная, спрей, акварель, масло или ластик). Название активной кисти отобразится на кнопке.
3. **Настройка кисти:** Нажмите «Настройки», при необходимости измените параметры: передвиньте ползунок «Толщина кисти», выберите цвет в палитре, отрегулируйте непрозрачность, яркость и контраст. Изменения применяются сразу к следующему мазку.
4. **Рисование на холсте:** Переместите курсор на область холста и нажмите левую кнопку мыши. Ведите мышь, чтобы рисовать линию. Если вы переместите мышь без нажатой кнопки, ничего не рисуется.

Чтобы стереть часть рисунка, выберите инструмент «Ластик» и проведите по области.

5. **Переключение в 3D-режим:** Нажмите «Переключить на 3D». Камера плавно изменит угол обзора. Теперь можно удерживать клавишу Shift и правой кнопкой мыши поворачивать камеру вокруг модели. При этом на экране появится надпись «Режим: Вращение камеры» оранжевого цвета. Отпустите Shift, чтобы вернуться к рисованию и увидеть «Режим: Рисование» зелёным.
6. **Работа с Undo/Redo:** Если совершено нежелательное действие, нажмите кнопку Undo () – последнее действие отменится. Для возврата действия используйте Redo (). Обратите внимание: Undo возвращает только нарисованные штрихи, но не отменяет удаление или добавление слоёв.
7. **Удаление слоя:** В списке слоёв нажмите на кнопку «» рядом с названием слоя. Все объекты этого слоя исчезнут. Останется предыдущий слой (или будет создан новый, если слоев не осталось).
8. **Сохранение работы:** В текущей версии (0.52 beta) приложение не содержит отдельной функции сохранения изображения. Для сохранения рисунка можно экспортировать сцену программным способом. На релизе планируется доработать данную функцию с соответствующей кнопкой.

Для модуля **Video** последовательность действий другая:

1. **Загрузка видео:** Нажмите на поле загрузки файлов и выберите видео в формате MP4 (или другом, поддерживаемом браузером). Видео начнёт проигрываться (однократно или в цикле).

2. **Добавление слоя наложения:** Нажмите «Add Layer». Появится новый элемент управления в блоке слоёв: ползунок непрозрачности и выпадающий список Blend Mode. Повторное нажатие создаст новый независимый слой.
3. **Настройка эффектов:** Передвиньте ползунки «Color Simplification», «Contour Intensity», «Noise Level», «Filter Intensity», «Cartoon Effect» в нужные положения (цифры отразятся рядом). Выберите режим смешивания в «Blend Mode». При желании нажмите «Auto Enhance» для автоматической установки фильтров.
4. **Применение стилизации:** Нажмите «Apply Stylization». Канвас станет видимым, и видео будет рисоваться с учётом выбранных параметров. Ожидайте обработки; внизу в «Speed Controls» можно изменять скорость (1×, 1.5×, 2× или 0.5×).
5. **Сохранение результата:** Выберите формат («MP4» или «GIF») в выпадающем списке справа. Нажмите «Save As...». Приложение иницирует сохранение: в случае MP4 – возможно, загрузит готовый видеофайл, в случае GIF – сгенерирует и даст скачать анимацию.

1.2.4. Руководство администратора

Инструкции по установке и обслуживанию системы предназначены для тех, кто разворачивает и поддерживает приложение. Данное решение представляет собой веб-приложение без серверной части и может работать в любом современном браузере. Основные требования:

- **Браузер:** современный браузер (Chrome, Firefox, Edge, Safari) с поддержкой WebGL и ES6-модулей JavaScript. Приложение использует Three.js и Anime.js через CDN, поэтому рекомендуется обеспечить интернет-соединение для доступа к библиотекам. В случае отсутствия сети можно скачать эти библиотеки локально и подключить из файлов.

- **Сервер:** для запуска приложения достаточно статического веб-сервера (Apache, Nginx, локальный http-server на Node.js, Python SimpleHTTPServer и т.п.). Файлы можно также открыть локально (file://), но некоторые браузеры могут блокировать ES-модули в таком режиме. Рекомендуется запуск через http/https протокол.
- **Структура файлов:** Распакуйте предоставленный архив, в каталоге program/Draw и program/Video находятся соответствующие страницы приложения. Убедитесь, что HTML, CSS и JS файлы лежат рядом (без изменения путей). В Draw/index.html и Video/index.html прописаны относительные пути к скриптам и стилям. Права доступа на файлы должны позволять чтение.
- **Зависимости:** Пакет package.json в модуле Draw не содержит дополнительных зависимостей, все сторонние библиотеки загружаются из внешних источников. Если требуется оффлайн-версия, можно установить зависимости локально через npm. Однако для работы приложения запустить npm install и npm start не нужно – просто поместите файлы на сервер.
- **Персональные настройки:** При необходимости можно изменить стили (CSS) в файлах styles.css, скорректировать начальный размер канвы (по умолчанию подгоняется под окно браузера), а также отредактировать перечень кистей или слоёв в исходном коде. Любые изменения скриптов должны согласовываться с логикой обновления интерфейса (например, не забыть вызвать updateLayerList() при изменении массива слоёв).
- **Логи и отладка:** В консоле разработчика браузера выводятся сообщения о состоянии («Инициализация завершена» и т.д.) и возможные ошибки. Администратор должен контролировать отсутствие JavaScript ошибок, обеспечивать совместимость версий браузера. Для отладки можно использовать встроенный режим разработчика.

Администратор также следит за ресурсами: WebGL-рендеринг потребляет GPU, поэтому на слабых машинах рекомендуется не рисовать слишком толстые линии или большое количество спрайтов одновременно. При развертывании на публичном сервере следует позаботиться о защите страниц (например, HTTPS) и установке ограничений на размер обрабатываемых видео (чтобы избежать перегрузки сервера, если обработка выполняется на стороне клиента). Регулярно проверяйте наличие обновлений библиотек и браузерных движков для поддержания корректной работы.

1.2.6. Разработка информационных моделей отображения текущей информации

Информационные модели определяют, как данные, полученные или генерируемые системой, отображаются пользователю и хранятся внутри приложения. В **Draw** используются следующие модели:

- **Модель слоёв:** массив `layers` (типа `Array<String>`), содержащий имена слоёв («Слой 1», «Слой 2» и т.д.). Этот массив является источником данных для интерфейса – список `<div class="layer-item">` генерируется из него. Индекс активного слоя хранится в `currentLayerIndex`.
- **Модель штрихов (рисунков):** массив `strokes` (типа `Array<Object>`), каждый элемент которого хранит информацию о начальной и конечной точке линии, типе кисти и слое (`{ start: {x,y,z}, end: {x,y,z}, brushType, currentLayerIndex }`). Эта модель содержит всю историю рисования и используется для Undo/Redo и повторного построения сцены.
- **Модель объектов сцены:** объект `layerObjects` (`{ [layerIndex]: Array<THREE.Object3D> }`), где каждому слою соответствует список объектов Three.js (линий, точек, кругов). При удалении слоя все объекты из `layerObjects[layerIndex]` удаляются из `scene`, и запись удаляется из модели. Эта модель обеспечивает согласованность между логической моделью слоёв и визуальной сценой.

- **Модель параметров кисти:** объект `brushSettings`, в котором хранятся настройки для каждого типа кисти (толщина, цвет, непрозрачность, радиус и т.д.). При выборе типа кисти данные из `brushSettings[currentBrush]` используются в процедуре рисования, и эти же данные изменяются при работе ползунков.
- **Модель состояния камеры и режима:** глобальная переменная `currentMode` со значениями «2D» или «3D», а также объект `cameraPositions`, задающий позиции камеры для каждого режима. Эти модели обеспечивают логику отображения и управление видом сцены.

Во **Video** модели таковы:

- **Модель каналов видео:** объект `<video>` как источник, и `<canvas>` как представление. Текущий кадр читается через `getImageData` и модифицируется согласно фильтрам.
- **Модель слоёв видео:** массив `layers` (либо отдельных `canvas`-слоёв), каждый элемент которого содержит канвас, 2D-контекст, значения `opacity` и `blendMode`. Эти данные определяют, как наложить несколько слоёв графики – аналогично слоям в графических редакторах. UI-элементы (ползунки `opacity` и селектор `blend mode`) синхронизированы с этими объектами.
- **Параметры фильтров:** отдельные переменные, связанные с ползунками (например, `colorSimplification.value`, `contourThreshold.value` и т.д.), служат моделью для текущих значений эффектов.

Все перечисленные модели используются системой для обновления экрана: при изменении любой модели (добавление слоя, изменение цвета, смещение ползунка) вызываются функции, которые генерируют новое изображение на канвасе на основе актуальных данных. Модели хранят информацию о состоянии системы в реальном времени и позволяют восстановить интерфейс после перерисовки.

1.2.7. Разработка графа взаимодействия пользователя с системой

Граф взаимодействия описывает состояния системы и переходы между ними при действиях пользователя. Основные **состояния** приложения Draw:

- **Состояние «Ожидание» (Idle):** приложение загружено, но рисование не выполняется (mouse up, не рисуется). Камера на прежней позиции.
- **Состояние «Рисование» (Drawing):** пользователь нажал левую кнопку мыши и ведёт курсор — создаётся новый штрих на активном слое.
- **Состояние «Ожидание 2D» и «Ожидание 3D»:** различаются положением камеры и активностью индикатора. Из них происходит переход в рисование при клике.
- **Состояние «Вращение камеры»:** только в 3D-режиме при удержании Shift и нажатии правой кнопки мыши — происходит поворот сцены.
- **Состояние «Настройки кисти»:** когда открыт выпадающий список параметров кисти, приложение ожидает ввода новых значений.
- **Состояние «Видео процесс» (для модуля Video):** при нажатии «Apply Stylization» запускается цикл обработки кадров, далее — режим воспроизведения/обработки.

Переходы и события:

- Переход в «Рисование» из «Ожидания» — при событии mousedown по канве (левая кнопка). Если слоёв нет, отображается предупреждение и переход не происходит.
- Возврат в «Ожидание» — при отпускании кнопки (mouseup) или при уходе курсора за границы канвы (mouseleave). В коде это stopDrawing().
- Переключение 2D↔3D — по нажатию кнопки «Переключить на 3D». Это действие ведёт к изменению currentMode, анимации камеры и отображению/скрытию индикатора.




- Переход в «Вращение камеры» — при удерживании клавиши Shift (keydown) в режиме 3D и нажатии правой кнопки (contextmenu) переопределён на startCameraRotation()). Вход в это состояние включает controls.enabled = true для OrbitControls.
- Выход из «Вращение камеры» — при отпуске Shift (keyup), приводя к controls.enabled = false.
- Добавление/удаление слоёв — в любое время через кнопки «Добавить/Удалить слой», которые изменяют массив layers. Эти действия не создают новые основные состояния, но порождают переходы в состояние «Ожидание» (после удаления, когда активный слой изменяется).
- Undo/Redo — вызывают функции undo()/redo(), которые переходят системе в состояние «Перерисовка сцены»: сцена очищается, выполняется новый цикл рендеринга согласно спискам strokes. Пользователь при этом всё ещё остаётся в «Ожидании» (рисование не активно).

Аналогично, модуль Video имеет свои состояния: загрузка видео, ожидание параметров, обработка, сохранение. Переход в режим «Сохранение файла» происходит при нажатии «Save As...».

Таким образом, граф взаимодействия представляет собой конечный автомат с узлами-состояниями приложения и ребрами, соответствующими событиям (кликам, нажатию клавиш) пользователя. Такой граф помогает формализовать пользовательские сценарии и убедиться, что все логические переходы корректно обработаны.

1.2.8. Эргономические требования к отображению информации

Интерфейс разработан с учётом эргономики и удобства: все элементы управления расположены логично и интуитивно доступны. Кнопки инструментов сгруппированы в единую панель; выпадающие меню и ползунки имеют подписи на русском языке, обеспечивая понятность. Для кистей используются наглядные названия («Круглая», «Ластик» и т.д.), благодаря чему пользователь сразу понимает эффект инструмента. Все текстовые метки (например, «Толщина кисти:», «Непрозрачность:», «Режим: Рисование») имеют контрастный цвет шрифта на светлом фоне, что повышает читаемость.

Цветовая схема сведена к минимуму отвлекающих элементов: фон приложения – светлый нейтральный (#f4e4c8), чтобы кнопки и панель выглядели контрастно. Активная область рисования (canvas) имеет белый фон (`scene.background = 0xffffffff`), что имитирует чистый лист. Индикация состояний (к примеру, оранжевый фон при «Вращении камеры») чётко выделяет активные режимы. Кнопки имеют достаточный размер для клика (минимум 32×32 пикселя) и некоторые содержат графические иконки (, , ) , что ускоряет восприятие без чтения текста.

Минимальный размер окна приложения не задан жёстко — канвас масштабируется по окну (`renderer.setSize(window.innerWidth, window.innerHeight)`), обеспечивая максимальное пространство для рисования. При изменении размера окна в реальном времени вызывается обработчик `onWindowResize()`, который меняет проекцию камеры и разрешение рендера, сохраняя корректность отображения.

Для ползунков (ranges) значения показаны рядом (элемент ``), что позволяет сразу видеть численную величину параметра. Все интерактивные элементы (кнопки, слайдеры, селекторы) снабжены стандартными HTML-

атрибутами `label for="..."` или подписью, что улучшает доступность (а также эргономично для тех, кто использует экранные чтецы).

Таким образом, эргономически интерфейс обеспечивает быструю ориентацию: новому пользователю не нужно долго изучать меню, т.к. названия и иконки однозначно указывают на функции. Расположение элементов минимизирует необходимость лишнего движения мышью: основные инструменты верхней панели находятся рядом, слои – слева, сам холст занимает большую центральную площадь для работы. В совокупности эти меры повышают скорость работы и снижают нагрузку на пользователя.

1.2.10. Технические решения для правильной эксплуатации программного изделия

Для надёжной работы и максимальной производительности приложения реализованы следующие технические решения:

- **Кроссбраузерность и стандарты:** Приложение написано с использованием стандартных технологий веб (HTML5, CSS, JavaScript ES6). Это гарантирует работу в современных браузерах без плагинов. Используются CDN-подключения для популярных библиотек (Three.js, Anime.js), что упрощает поддержку.
- **WebGL-ускорение:** Рендеринг рисования осуществляется через WebGL (Three.js). Это позволяет использовать графический ускоритель для плавного отображения даже при большом количестве объектов. При добавлении сложных фигур или повышении уровня сглаживания задействуется GPU, снижая нагрузку на CPU.
- **Адаптивная разметка:** Область рисования (canvas) автоматически масштабируется под размер окна, благодаря обработчику `onWindowResize()`. Это обеспечивает корректную работу на экранах разного размера.

- Все UI-элементы стилизованы с помощью CSS (файл `styles.css`), что позволяет легко изменять оформление без изменения логики.
- **Обработка ошибок:** В коде есть проверки критических условий (например, предупреждение если пользователь пытается рисовать без слоёв). При возможных исключениях (напр. нет WebGL) приложение может упасть, поэтому целесообразно проверять поддержку WebGL перед запуском или информировать пользователя о необходимости сменить браузер. Для модульности ошибки работы с видеообработкой также учитываются (проверяется наличие `video.src` перед началом).
- **Управление ресурсами:** При удалении слоя объекты немедленно удаляются из сцены (`scene.remove()`), что освобождает графическую память. Аналогично при Undo/Redo старые объекты очищаются перед перерисовкой, чтобы предотвратить нарастание памяти. Такой подход предотвращает утечки памяти в долгосрочной работе.
- **Асинхронность и оптимизация:** Использование `requestAnimationFrame` для цикла анимации рисования и видеообработки обеспечивает синхронизацию с частотой обновления экрана, даёт плавность и экономию ресурсов (в отличие от `setInterval`). Фильтрация видео ведётся построчно в JavaScript, но если производительность низкая, можно рассмотреть WebAssembly или шейдеры WebGL для ускорения (рекомендация для будущих версий).
- **Серверные настройки:** Поскольку приложение статическое, для веб-сервера важны правильные MIME-типы (например, `.js` как `application/javascript`, `.html` как `text/html`), а также активный HTTPS (для доступа к некоторым функциям браузера). В HTML нет чувствительных данных, но рекомендуется включать заголовки безопасности (Content Security Policy).

В совокупности эти решения повышают устойчивость приложения к ошибкам, делают его отзывчивым и удобным при эксплуатации на различных конфигурациях. Пользователь получает высокое качество отрисовки при минимальных задержках ввода, а администратор может легко обновлять библиотеки и внедрять изменения за счёт модульной структуры (функции разделены по файлам: canvas.js, toolbar.js, layers.js).

1.2.11. Разработка рекомендаций по эксплуатации изделия

Для эффективной и долгосрочной эксплуатации рекомендовано следующее:

- **Регулярно сохраняйте результаты:** В текущей версии нет встроенной функции сохранения чертежа в файл. Пользователю следует заранее позаботиться о сохранении промежуточных результатов (например, скриншотом или экспортом сцены). Для видео рекомендуется работать с короткими клипами и сохранять итоговые файлы после обработки каждого.
- **Предпочтительные платформы:** Приложение оптимизировано для настольных браузеров. Работа на мобильных устройствах менее комфортна: из-за ограничений WebGL может снижаться производительность. Рекомендуется использовать современные компьютеры с включённым аппаратным ускорением графики. Убедитесь, что в настройках браузера включён WebGL.
- **Ресурсозатраты:** При длительном рисовании с многочисленными сложными штрихами может расти использование памяти браузера. При замедлении работы полезно очистить канву и слои, создать новый документ. Для видео – избегайте загрузки слишком больших по размеру (High Definition), поскольку обработка в браузере может замедлить систему.

- **Обновление программного обеспечения:** Для стабильности рекомендуется использовать актуальные версии браузеров и библиотек (three.js регулярно обновляется).

При обновлении библиотеки проверяйте совместимость с текущим кодом (интерфейс объектов three.js иногда меняется).

- **Аппаратные рекомендации:** Для плавной работы в режиме 3D и при сложных эффектовых обработках видео желательно иметь дискретную видеокарту. При отсутствии ускорения на встроенной графике возможно падение FPS. Закрывайте другие ресурсоёмкие приложения при длительной работе с графикой.
- **Технические перерывы:** Если приложение запущено непрерывно (особенно видео с высокой скоростью обработки), рекомендуется делать паузы, чтобы браузер не переполнял память. В случае зависания страницы – обновите её, предварительно сохранив данные.
- **Безопасность данных:** Поскольку данные не отправляются на сервер, вся обработка локальна. Однако при скачивании библиотек рекомендуется доверять проверенным CDN. Периодически проверяйте целостность файлов приложения (например, антивирусом), особенно если приложение развёрнуто на общем сервере.
- **Обучающие материалы:** Для новых пользователей следует ознакомиться с базовыми понятиями работы слоями и эффектами. Подробное руководство пользователя (пункт 1.2.3) должно быть доступно вместе с приложением. При обучении рекомендуется практиковаться с простыми примерами: начать с 2D-режима, попрактиковаться в рисовании простых фигур, затем плавно переходить к настройке кистей и работе в 3D.
- **Тестирование на стабильность:** Перед деплоем или важной демонстрацией проверьте работу на нескольких конфигурациях: разных

браузерах и устройствах. Также протестируйте крайние случаи: удаление всех слоёв, отмену всех действий и работу с нулевыми значениями параметров.

Выводы

В данной главе была рассмотрена технологическая сторона разработки информационно-программного средства, охватывающая процессы обработки данных, взаимодействие пользователя с системой и специфику реализации ключевых компонентов. Описание охватило как инструментальные, так и архитектурные аспекты, что позволило представить проект с точки зрения его внутренней организации и пользовательского опыта.

Особое внимание было уделено модулю **Draw**, реализующему основные функции редактирования 2D/3D-графики. Рассмотрены механизмы работы с холстом, структура слоёв, настройка кистей, управление сценой и обработка пользовательского ввода. Интерфейс разработан с ориентацией на интерактивность и эргономику, обеспечивая минимальные усилия со стороны пользователя для выполнения сложных визуальных операций.

Модуль **Video**, дополняющий систему, предоставляет средства потоковой стилизации и редактирования видео с применением фильтров и масок, а также экспорт результата. Его интеграция демонстрирует расширяемость архитектуры и возможность масштабирования продукта.

Таким образом, проведённый анализ и реализация подтверждают соответствие технологических решений поставленным задачам, обеспечивая стабильную, отзывчивую и адаптивную основу для работы редактора в веб-среде.

Приложение

Таблица 3 – детальный алгоритм решения задачи [3]

Этап	Действие пользователя	Реакция системы / Описание процесса
1. Инициализация	Открытие веб-приложения	Загрузка интерфейса, инициализация WebGL-контекста, создание сцены, камеры, освещения и рендерера.
2. Настройка кисти	Выбор инструмента (кисти)	Отображение панели параметров кисти: форма, радиус, интенсивность, тип воздействия.
	Изменение параметров кисти	Сохранение пользовательских настроек, подготовка к использованию на сцене.
3. Взаимодействие со сценой	Наведение курсора на объект / область сцены	Определение позиции взаимодействия, подготовка данных для применения кисти.
	Применение кисти (нажатие/движение)	Модификация данных сцены (например, изменение цвета, нормалей, высот и т.д.). Перерисовка сцены.
	Динамическая корректировка параметров во время работы	Немедленное применение новых параметров без перезагрузки сцены.
4. Обновление отображения	Завершение кистевого действия или перемещения	Автоматическая перерисовка затронутой области сцены.

5. Управление проектом	Сохранение сцены	Сериализация текущего состояния (объекты, параметры кисти, конфигурации), экспорт или отправка на сервер.
	Загрузка сохранённого проекта	Десериализация данных и восстановление сцены и параметров интерфейса.
6. Завершение работы	Выход пользователя из приложения	Очистка ресурсов, сохранение состояния при необходимости, завершение рендеринга.

Список литературы:

1. Foley, J. D., Van Dam, A., Feiner, S. K., Hughes, J. F. «Computer Graphics: Principles and Practice». Addison-Wesley, 1990. [5]
2. Akenine-Möller, T., Haines, E., Hoffman, N. «Real-Time Rendering». CRC Press, 2018. [17]
3. Kivistö, J. «Innovative Approaches in Digital Art: Bridging 2D and 3D». Journal of Digital Creativity, 2019. [32]
4. Shreiner, D., Sellers, G., Kessenich, J., Licea-Kane, B. «OpenGL Programming Guide: The Official Guide to Learning OpenGL». Addison-Wesley, 2013. [21]
5. Williams, R. «The Art of 3D Computer Animation and Effects». Wiley, 2012. [29]
6. Rogers, D. F., & Adams, J. A. (2013). Computer Graphics: Principles and Practice (3rd ed.). Addison-Wesley. [14]
7. Watt, J. D. (2000). 3D Computer Graphics (3rd ed.). Addison-Wesley. [12]

