**ТИТУЛЬНЫЙ ЛИСТ**

**ЗАДАНИЕ(ТЗ)**

**РЕФЕРАТ**

Дипломная работа на тему «Программный каркас для создания спрайтовой анимации на HTML5» решает проблему сложности создания интерактивных приложений с двумерной графикой с применением 2d-context HTML5. Исходя из поставленной задачи, в работе подробно рассмотрена область компьютерной графики в целом и проблема сложности использования 2d-context в частности. В работе предлагается метод решения упомянутой проблемы с помощью создания специализированного каркаса.

Работа представляет интерес для специалистов, работающих в области создания мультимедийных приложений(# игр), а также для всех, кому интересна данная тематика. Работа содержит 2 рисунков и 2 приложения. Общий объем работы составляет 51 страницу. Структура работы представлена списком определений, введением, тремя главами, заключением, списком литературы, а также приложениями.

# СОДЕРЖАНИЕ

Оглавление

[СОДЕРЖАНИЕ 4](#_Toc356088311)

[ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 6](#_Toc356088312)

[ВВЕДЕНИЕ 7](#_Toc356088313)

[1. ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ И ПОСТАНОВКА ЗАДАЧИ 8](#_Toc356088314)

[1.1. Введение в создание мультимедийных веб-приложений 8](#_Toc356088315)

[1.1.1. Мультимедийное веб-приложение 8](#_Toc356088316)

[1.1.2. Широко используемые технологии, библиотеки и каркасы 9](#_Toc356088317)

[1.2. Технологии HTML5 12](#_Toc356088318)

[1.2.1. 2d-context 12](#_Toc356088319)

[1.3. Сложности разработки с помощью технологии 2d-context HTML5 13](#_Toc356088320)

[1.4. Постановка задачи 14](#_Toc356088321)

[2. ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНОГО КАРКАСА 15](#_Toc356088322)

[2.1. Обзор существующих решений 15](#_Toc356088323)

[2.2. Требования к программному каркасу 16](#_Toc356088324)

[2.3. Выбор сторонних вспомогательных каркасов\библиотек 17](#_Toc356088325)

[2.4. Используемые алгоритмы 18](#_Toc356088326)

[2.5. Архитектура каркаса 19](#_Toc356088327)

[2.5.1. Парадигма программирования 19](#_Toc356088328)

[2.5.2. UML – диаграммы 20](#_Toc356088329)

[2.5.2.1. Диаграмма классов 20](#_Toc356088330)

[2.5.3. Используемые шаблоны проектирования 22](#_Toc356088331)

[3. ПРОЦЕСС КОДИРОВАНИЯ 24](#_Toc356088332)

[3.1. Средства разработки 24](#_Toc356088333)

[3.1.1. Интегрированная среда разработки 24](#_Toc356088334)

[3.1.2. Браузер 26](#_Toc356088335)

[3.1.3. Отладчик 27](#_Toc356088336)

[3.1.4. Профайлер 28](#_Toc356088337)

[3.1.5. Система контроля версий 29](#_Toc356088338)

[3.2. Принятые стандарты кодирования 30](#_Toc356088339)

[3.3. Интеграция классов каркаса 31](#_Toc356088340)

[ЗАКЛЮЧЕНИЕ 32](#_Toc356088341)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc356088342)

[ПРИЛОЖЕНИЯ 34](#_Toc356088343)

# ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

# ВВЕДЕНИЕ

Еще недавно самым популярным средством создания интерактивных графических веб-приложений, насыщенных анимацией и трехмерной графикой, была технология Flash от фирмы Adobe[10], что требовало, от разработчика, покупки среды разработки Flash Professional и изучения разработчиком языка программирования Acton Script, а так же требовало от пользователя установки в браузер средства для воспроизведения(Flash Player) Flash-роликов. Долгое время Flash был, по сути единственной технологией для создания игр, мультфильмов, интерактивны веб-приложений. Для всех основных платформ существовал и поддерживался Flash-player, но с развитием открытой технологии HTML5, постепенно, Adobe отказалось от поддержки Flash-player для операционных систем семейства Linux[11], а затем и от поддержки для мобильных платформ [12], так же ряд крупных фирм отказались от поддержки Flash:

• Apple в пользу HTML5 для iPhone, iPod touch и iPad;

• Microsoft в пользу Silverlight для Windows Phone 7 и HTML5 для Windows 8;

• Oracle в пользу JavaFX.

С появлением поддержки нового, предоставляющего схожие с Flash графические возможности, но еще не готового стандарта HTML(HTML5), ведущими производителями браузерных ядер (WebKit (Chrome), Gecko(FireFox), Presto(Opera)), перед разработчиком интерактивных веб-приложений встает необходимость использования графических возможностей HTML5.

В данной работе решается проблема сложности разработки графических веб-приложений с помощью 2d-context HTML5.

Что такое программный каркас?

# ВВЕДЕНИЕ В ПРЕДМЕТНУЮ ОБЛАСТЬ И ПОСТАНОВКА ЗАДАЧИ

(Архитектура веб-приложения)

# Введение в создание мультимедийных веб-приложений

# Мультимедийное веб-приложение

# Широко используемые технологии, библиотеки и каркасы

На данный момент существует ряд технологий позволяющих вести разработку графических веб-приложений, приведем наиболее известные технолгии:

1. **Flash** – мультимедийная платформа компании Adobe для создания веб-приложений или мультимедийных презентаций. Широко используется для создания рекламных баннеров, анимации, игр, а также воспроизведения на веб-страницах видео- и аудиозаписей.

Платформа включает в себя ряд средств разработки, прежде всего en:Adobe Flash Professional и Adobe Flash Builder (ранее Adobe Flex Builder); а также программу для воспроизведения flash-контента — Adobe Flash Player;

1. **2d-context** – это контекст тега <canvas> предоставляющего интерфейс для создания двумерной растровой графики. 2d-context позволяет манипулировать изображением на плоской двумерной системе координат с центром в левом верхнем углу экрана. Интерфейс представлен набором методов и свойств, определяющих графические примитивы их внешний вид и аффинные преобразования над экранной плоскостью. Данная технология реализует непосредственный режим графики, то есть программист должен сам заботится о перерисовки каждого кадра, реализации необходимых графических алгоритмов и т.д.

Данный контекст можно считать простой альтернативой технологии Flash и вероятно будет подходить для создания мультимедийных веб-приложений, игр, анимированных интерфейсов и т.д.;

1. **Webgl-context** – Это контекст тега <canvas> предоставляющего интерфейс для создания трехмерной графики. Интерфейс является производным от OpenGL ® ES 2.0 и имеет схожие возможности, включая работу с вершинными и пиксельными шейдерами. Webgl-context позволяет задействовать вычислительные мощности видеокарты, что может подвергать пользователя риску, через открытие доступа к привилегированному режиму видеокарты и оборудования[13];
2. **SVG** – язык описания векторных изображений, построенный на языке XML;
3. **Манипуляции DOM-моделью HTML документа** – С самого появления языка HTML, HTML документ состоял из набора тегов определявших внешний вид и назначение различных элементов на странице. Манипуляция тегами, т.е. структурными элементами DOM–модели, можно рассматривать как возможность создания спрайтовой анимации. Этот метод мало применим для достаточно сложной игры из-за «подрагивания» или «мелькания» при достаточно частом обновления страницы. Манипуляции с DOM-моделью не предназначены для частых динамичных изменений.

Для сравнения 2d-context и SVG, можно использовать тест приведенный по ссылке[14]: тест состоял в отслеживании числа кадров в секунду (FPS) при изменении количества объектов (движущиеся с максимальной скоростью квадраты) на экране при использовании 2d-context и при использовании SVG. Результаты теста на ОС Windows 7, приведены на рисунке1.

****

**Рис. 1. Сравнительный тест технологий 2d-context и SVG**

Как можно видеть на графике (Рисунок 1), 2d-contex более производителен при большом количестве объектов и так как основными задачами, решаемыми с помощью графических возможностей HTML5 и Flash, чаще всего являются реализация спрайтовой анимация и работа с изображениями, можно сделать вывод, что для замены технологии Flash больше всего подходит 2d-context ведь он, как и Flash, реализует работу с растровой графикой и позволяет работать с большим числом объектов. Очевидно, что выбирая между 2d-context и SVG для реализации приложения, которое должно генерировать динамичное изображение состоящее из большого количества объектов, следует выбрать 2d-context.

# Технологии HTML5

# 2d-context

2d-context – это контекст тега <canvas> предоставляющего интерфейс для создания двумерной растровой графики. 2d-context позволяет манипулировать изображением на плоской двумерной системе координат с центром в левом верхнем углу экрана. Интерфейс представлен набором методов и свойств, определяющих графические примитивы их внешний вид и аффинные преобразования над экранной плоскостью. Данная технология реализует непосредственный режим графики, то есть программист должен сам заботится о перерисовки каждого кадра, реализации необходимых графических алгоритмов и т.д.

Данный контекст можно считать простой альтернативой технологии Flash и вероятно будет подходить для создания мультимедийных веб-приложений, игр, анимированных интерфейсов и т.д.

# Сложности разработки с помощью технологии 2d-context HTML5

Для оценки проблем и сложностей разработки с помощью 2d-context необходимо ознакомится с самим процессом разработки.

Для использования любой технологии или инструмента не обходимо ознакомится со спецификацией, из прочтения которой можно выяснить возможности, требования и концепции использования или технологии. Для технологии 2d-context спецификация доступна по адресу [5].

Из прочтения

Использование 2d-context влечет за собой следующие трудности:

1. Неготовый стандарт. Стандарт в стадии тестирования – W3C объявил о планах, согласно которым окончательная версия стандарта HTML5 будет утверждена лишь к 2014 году[6];

2. Отсутствие визуальных сред, вроде Flash Professional CS6;

3. Слабое развитие специализированных каркасов, вызванные, скорее всего, незавершенностью стандарта;

4. Разработка без использования специализированного каркаса требует большой объем кода для реализации несложной графики.

Но тем не менее, не смотря на трудности, HTML5 перспективная технология, т.к. не зависит от какой то одной компании, имеет открытый стандарт и рекомендуется такими компаниями как Apple и Microsoft.

# Постановка задачи

Постоянное совершенствование браузеров, появление поддержки браузерами более производительных графических технологий, а так же рост аппаратных возможностей дает разработчикам возможность создавать более динамичные мультимедийные веб-приложения.

Появление технологии 2d-context HTML5 позволяет создавать динамичные мультимедийные веб-приложения, но разработка без специализированного программного каркаса требует много времени на разработку и более подвержено ошибкам в силу объема и сложности реализуемого кода. В связи с данной проблемой возникает потребность в специализированном программном каркасе, реализующем базовую функциональность графического приложения.

Для решения поставлено задачи необходимо:

* Ознакомится с программным интерфейсом 2d-context;
* Выявить требования к каркасу;
* Изучить необходимые алгоритмы и техники;
* Спроектировать и реализовать программный каркас.

# ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ ПРОГРАММНОГО КАРКАСА

# Обзор существующих решений

В ходе анализа существующих решений и анализа процесса создания мультимедийных приложений было выявлено:

В мультимедиа приложении, интерактивность достигается за счет взаимодействия пользователя с отдельными элементами на экране (# кнопки, поя ввода и т.д. ), при этом элементы представлены набором графических примитивов. Можно сделать вывод, что для изменения внешнего вида элемента необходимо изменять свойства каждого примитива входящего в состав элемента, то есть для разработчика важно иметь возможность представлять элемент интерфейса с помощью совокупности графических примитивов, возможность управлять внешними видом совокупности примитивов как одной сущности.

Так как 2d-context представляет так называемый непосредственный графический режим[]

Анализ игровых приложений, а также личный опыт создания, подталкивают к выводу, что каркас должен обеспечивать различные оптимизации с точки зрения производительности, для оценки качества оптимизации можно использовать такой параметр как FPS (#число кадров в секунду).

Программный каркас это инструмент программиста, то есть он должен быть удобен программисту, а значит должен обладать интуитивным, лаконичным интерфейсом.

Необходимо учитывать, что размер каркаса влияет на общее время загрузки и старта приложения, а так как при создании достаточно универсального каркаса, в разных приложениях могут быть задействованы далеко не все возможности, необходимо реализовать модульности каркаса. Должна быть возможности собирать каркас для конкретного приложения, из необходимых для конкретного приложения, модулей.

# Требования к программному каркасу

# Выбор сторонних вспомогательных каркасов\библиотек

# Используемые алгоритмы

В каркасе были реализованы такие алгоритмы:

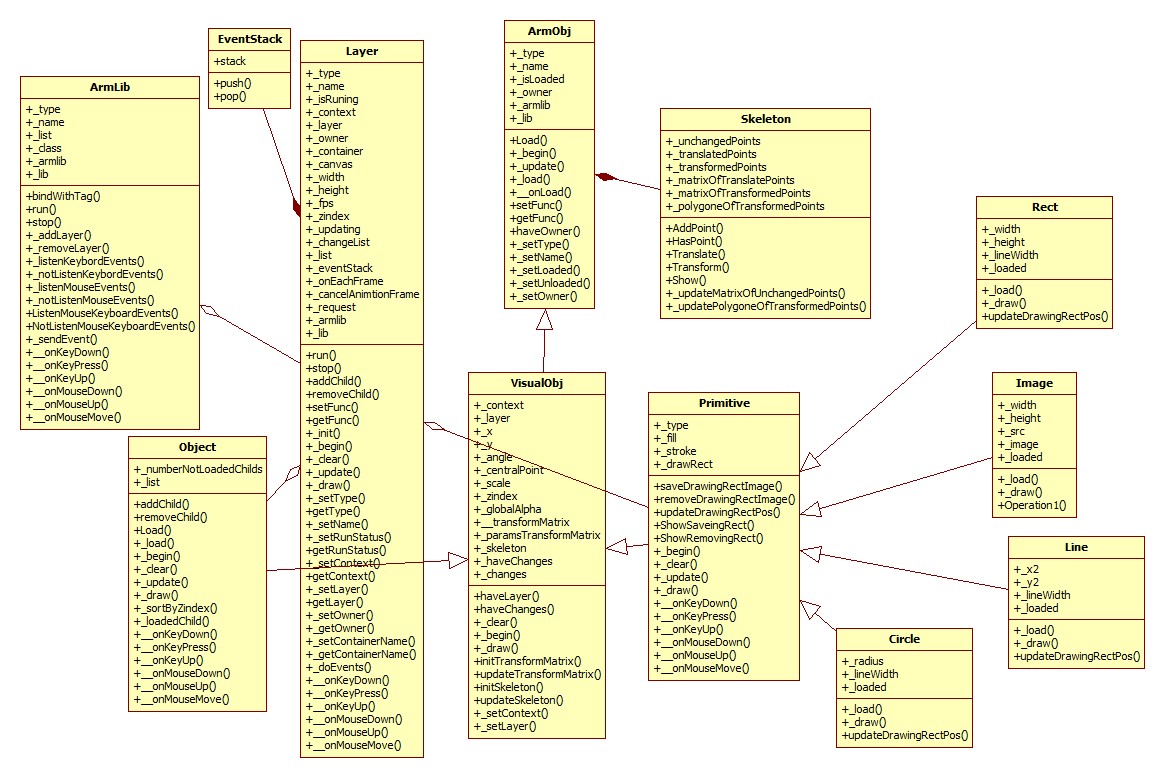
# Архитектура каркаса

# Парадигма программирования

# UML – диаграммы

# Диаграмма классов

Перед началом разработки приложения была создана диаграмма классов (Рисунок 2).

****

**Рисунок 2 – UML диаграмма классов**

Данная диаграмма (Рисунок 2) отображает отношения между классами каркаса. Класс ArmLib, является менеджером каркаса, т.е. через его методы происходит управление всем каркасом. Можно видеть, что класс Layer и Object агрегирует в себе объекты классов Object и классы производные от класса Primitive. Данное отношение между классами позволяет ввести абстракцию объектов, т.е. структурные части приложения представлены не просто набором классов, функций и т.д., а набором объектов классов Object, Image, Circle, Line, Rect и производными от этих классов. Данный подход позволяет уменьшить сопряжение[15] и сделать структурные части каркаса более выраженными. Классы Rect, Line, Circle, Text, Image описывают графические примитивы, с помощью которых описываются визуальные объекты приложения.

# Используемые шаблоны проектирования

В ходе разработки сложного программного обеспечения важно создавать качественный, готовый к сопровождению другими программистами, код. Одной из методик уменьшения сложности сопровождения, является использование, так называемых, шаблонов проектирования программного обеспечения.

Шаблоны проектирования описывают повторимые архитектурные конструкции, представляющие собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях.

Шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться. Использование широко известных шаблонов, делает код более «читаемым».

В каркасе были использованы ряд шаблонов, а именно:

**Порождающие шаблоны:**

• «Одиночка», данный шаблон использован в классе ArmLib, для гарантирования существования только одного объекта класса ArmLib;

• «Фасад», данный шаблон использован в классах графических примитивов: Rect, Line, Circle, Image. В данных классах данный шаблон дает возможность более удобно работать с API контекста рисования HTML5;

• «Компоновщик», данный шаблон использован в классах ArmLib, Layer и Object для создания древовидной структуры отношений примитивов и объектов сцены.

**Фундаментальные шаблоны**

• «Делегирование», данный шаблон является базовым для приложения созданного в объектно-ориентированном стиле. Данный шаблон используется почти во всех классах каркаса.

**Поведенческие шаблоны**

• «Наблюдатель», данный шаблон используется в классах ArmLib, Layer и Object для оповещения объектов класса Layer в классе ArmLib, объектов класса Object, Image, Rect, Line в класах Object и Layer о событиях клавиатуры и мыши.

# ПРОЦЕСС КОДИРОВАНИЯ

# Средства разработки

# Интегрированная среда разработки

Инструменты

«По некоторым оценкам до 40% рабочего времени программист тратит

на редактирование исходного кода» [16]

Среда разработки является инструментом призванным упрощать разработку программного продукта, за счет интеграции таких инструментов как: текстовый редактор, компилятор, средства оптимизации сборки, система контроля версий и т.д, в одном продукте.

Выбор среды разработки, является критичным как с точки зрения времени выполнения проекта, так и с точки зрения качества проекта. Грамотный выбор подходящей среды разработки может позволять: уменьшать время кодирования и количество ошибок, за счет:

• Подсветки синтаксиса используемого языка программирования;

• Наглядном представлении файлов проекта;

• Интеграции с системами контроля версий;

• Интеграция с системами тестирования;

• Использовании различных плагинов, например jsLint и т.д.

Для разработки была выбрана интегрированная среда разработки Netbeans. Выбор Netbeans обусловлен:

• Стоимостью, среда бесплатна;

• Большим количеством пользователей, в случае возникновения проблем проще найти решение;

• Поддержкой синтаксиса языка JavaScript;

• Поддержка распределенной системы контроля версий Git.

• Возможность использовать утилиту jsLint в виде плагина, что позволяет использовать данную утилиту совместно со средой разработки.

# Браузер

В силу заложенных требований к кросбраузерности, каркас тестировался на трех основных браузерах:

* Chrome 26.0.1410.64;
* Opera 12.13;
* FireFox 18.0.1.

В первую очередь тестирование проводилось на браузере Chrome из-за наличия встроенного отладчика и профилировщика.

# Отладчик

Для отладки кода использовался встроенный отладчик браузера Chrome

# Профайлер

# Система контроля версий

В процессе разработки была использована распределенная система контроля версий для снижения рисков потери выполненной работы и упрощения процесса работы с разными версиями кода.

# Принятые стандарты кодирования

# Интеграция классов каркаса

Интеграция классов системы это процесс

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

# ПРИЛОЖЕНИЯ

**ТЗ**

**Руководство программиста (ГОСТ 19.504)**