|  |
| --- |
| **МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**  Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  **«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  **ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»** |

Институт кибернетики  
Направление – информатика и вычислительная техника  
Кафедра автоматизации и компьютерных систем

**ПРОГРАММНЫЙ КАРКАС ДЛЯ СОЗДАНИЯ СПРАЙТОВОЙ АНИМАЦИИ НА HTML5**

**Выпускная квалификационная работа**

**(на соискание квалификации бакалавр)**

Студент гр. 8В94 ИК ТПУ  А.С. Лизин

Руководитель

Аспирант каф. АиКС ИК ТПУ Ф.Е. Татарский

**Допустить к защите:**

Заведующий кафедрой АиКС ИК ТПУ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| (Подпись) |  | (Дата) |  | (Фамилия И.О.) |

ТОМСК – 2013 г.

**МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

|  |  |  |
| --- | --- | --- |
| Кафедра АиКС ИК ТПУ  **УТВЕРЖДАЮ** | | |
| Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
|  | подпись | И.О.Фамилия |
| «\_\_\_\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г. | |

**З А Д А Н И Е**

**на выполнение выпускной квалификационной работы**

Студенту гр. 8В94 Лизину Александру Сергеевичу

1. **Тема выпускной квалификационной работы** «Программный каркас для создания спрайтовой анимации на html5»,утверждена приказом от \_\_\_\_\_\_\_\_\_ № \_\_\_\_\_\_\_
2. **Срок сдачи студентом готовой работы:** 11 июня 2013 г.
3. **Исходные данные к работе:**
   1. Ознакомится с программным интерфейсом технологи 2d-context HTML5.
   2. Выявить требования к каркасу.
   3. Изучить необходимые алгоритмы и техники.
   4. Спроектировать и реализовать программный каркас.
4. **Содержание расчётно-пояснительной записки (перечень вопросов, подлежащих разработке):**
   1. Процесс создания веб-приложений.
   2. Программный интерфейс технологии 2d-context HTML5.
   3. Проблема создания мультимедийных веб-приложений с использованием 2d-context HTML5.
   4. Архитектура программного каркаса.
5. **Дата выдачи задания на выполнение выпускной квалификационной работы:**

20 января 2013 г.

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата, И.О.Фамилия)

#### Задание принял к исполнению (студент)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата, И.О.Фамилия)

### 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

[РЕФЕРАТ 4](#_Toc357033976)

[СОДЕРЖАНИЕ 5](#_Toc357033977)

[ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 7](#_Toc357033978)

[ВВЕДЕНИЕ 9](#_Toc357033979)

[1. МУЛЬТИМЕДИЙНОЕ ВЕБ-ПРИЛОЖЕНИЯ 10](#_Toc357033980)

[1.1. Современное веб-приложение 10](#_Toc357033981)

[1.2. Архитектура веб-приложения 11](#_Toc357033982)

[1.3. Широко используемые клиентские технологии 12](#_Toc357033983)

[1.4. Технология 2d-context HTML5 14](#_Toc357033984)

[1.5. Сложности разработки с помощью технологии 2d-context HTML5 17](#_Toc357033985)

[1.6. Постановка задачи 18](#_Toc357033986)

[2. АРХИТЕКТУРА ПРОГРАММНОГО КАРКАСА 19](#_Toc357033987)

[2.1. Обзор существующих решений 19](#_Toc357033988)

[2.2. Требования к программному каркасу 20](#_Toc357033989)

[2.2.1. Парадигма программирования 20](#_Toc357033990)

[2.3. Используемые алгоритмы 20](#_Toc357033991)

[2.4. Диаграмма классов 20](#_Toc357033992)

[2.5. Используемые шаблоны проектирования 21](#_Toc357033993)

[2.6. Принятые стандарты кодирования 22](#_Toc357033994)

[2.7. Интеграция классов каркаса 22](#_Toc357033995)

[2.8. Тестирование 22](#_Toc357033996)

[2.9. Используемые утилиты 22](#_Toc357033997)

[2.10. Средства разработки 22](#_Toc357033998)

[2.10.1. Интегрированная среда разработки 22](#_Toc357033999)

[2.10.2. Браузер 23](#_Toc357034000)

[2.10.3. Отладчик 24](#_Toc357034001)

[2.10.4. Система контроля версий 24](#_Toc357034002)

[2.10.5. Профайлер 24](#_Toc357034003)

[ЗАКЛЮЧЕНИЕ 25](#_Toc357034004)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26](#_Toc357034005)

[ПРИЛОЖЕНИЯ 27](#_Toc357034006)

[ТЗ 27](#_Toc357034007)

[Руководство программиста (ГОСТ 19.504) 27](#_Toc357034008)

### ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

**RIA** (от англ. Rich Internet application) — веб-приложение, доступное через Интернет, насыщенное функциональностью традиционных настольных приложений, которое предоставляется либо уникальной спецификой браузера, либо через плагин, либо путём «песочницы» (виртуальной машины).

**API** (англ. application programming interface) — набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах. Используется программистами для написания всевозможных приложений.

**SVG** (от англ. Scalable Vector Graphics) — язык разметки масштабируемой векторной графики, созданный W3C и входящий в подмножество расширяемого языка разметки XML, предназначен для описания двумерной векторной и смешанной векторно/растровой графики в формате XML.

**DOM** (от англ. Document Object Model) — не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

**OpenGL ES** — подмножество графического интерфейса OpenGL, разработанное специально для встраиваемых систем — мобильных телефонов, карманных компьютеров, игровых консолей. OpenGL ES определяется и продвигается консорциумом Khronos Group, в который входят производители программного и аппаратного обеспечения, заинтересованные в открытом API для графики и мультимедиа.

**Шейдер** – программа, которая используется в трёхмерной графике для определения окончательных параметров изображения или объекта.

**Программный каркас –**

**Трехмерная графика реального времени –**

### ВВЕДЕНИЕ

Использование насыщенных веб-приложений позволяет создавать приложения с функциональностью сравнимой с традиционными приложениями, а по таким параметрам как сохранность личных данных и системные требования даже выгодно превосходить традиционные приложения. Такая технология как 2d-context HTML5 позволяет создавать графические приложения с двумерной графикой сравнимой с традиционными графическими приложениями.

Традиционно разработчик заинтересован упростить процесс создания, увеличить надежность, а так же сократить время разработки. Распространенным способом реализации этого является создание программного каркаса. Программный каркас это программное обеспечение, облегчающее разработку и объединение частей программного проекта.

Технология 2d-context в силу относительной новизны, а так же незаконченной спецификации еще не располагает богатым набором развитых программных каркасов. В связи, с чем встает необходимость создании специализированного каркаса.

Реализованный каркас упрощает работу программиста за счет:

1. Предоставления готовых, согласованных реализации частей мультимедийнго приложения использующего 2d-context HTML5.
2. Упрощения взаимодействия с интерфейсом 2d-context.
3. Модульности каркаса.

### МУЛЬТИМЕДИЙНОЕ ВЕБ-ПРИЛОЖЕНИЯ

### Современное веб-приложение

С ростом возможностей современных веб-технологий, веб-приложение становится серьезной альтернативой для традиционных настольных приложений. Это связано с эволюцией браузеров, увеличением скорости и качества интернет соединения у среднего пользователя, а так же с появлением новых веб-технологий. Для исполнения веб-приложение нуждается только в наличии современного браузера, что позволяет считать веб-приложение кросплатформенным. Данный факт делает целевой платформой большинство современных вычислительных устройств, включая мобильные телефоны и даже некоторые виды бытовой техники. Хорошей оценкой возможностей мультимедийных веб-приложений могут послужить игры, так для большинства игр требуется достаточно большая производительность платформы, что включает в себя не только аппаратные, но и программные возможности. Программной платформой для веб-приложений, в первую очередь, служат браузеры и технологии доступные для программиста при их использовании. На данный момент с помощью таких технологий как Flash и HTML5 существует возможность разработки веб-приложений с трехмерной графикой реального времени. Это очень важный показатель программных возможностей платформы, так как для поддержки трехмерной графики реального времени требуется частота обновления изображения порядка 60 кадров в секунду с просчетом всей экранной сцены примерно за 16 миллисекунд. Это стало возможно благодаря появлению поддержки данными технологиями вычислительных возможностей видеокарт. Из данного факта можно сделать вывод: **Современной мультимединое веб-приложение может служить альтернативой настольным приложениям.**

### Архитектура веб-приложения

Веб-приложение представляет клиент-серверное приложение, в котором клиентом является браузер, а сервером – веб-сервер. Логика приложения распределена между клиентской и серверной частями. Данные используемые приложением хранятся на сервере и используются приложением по запросу.

Клиентская часть приложения реализует пользовательский интерфейс, обрабатывает действия пользователя, формирует запросы к серверной части и обрабатывает его ответы.

Серверная часть получает, обрабатывает и отправляет запросы клиентской части либо по протоколу http, либо с помощью сокетов.

В зависимости от того какая часть приложения выполняется на стороне клиента, клиентскую часть называют толстым или тонким клиентом.

Толстым клиентом называют клиентскую часть приложения, в которой выполняется большая часть логики приложения, а серверная сторона занимается в основном передачей и хранением данных.

Тонкий клиент по аналогии с толстым, имеет не равную функциональную нагрузку, но большая часть работы происходит на серверной стороне.

Большинство современных веб-приложений имеют толстые клиенты.

Клиентская часть реализуется с помощью таких технологий как:

Интерфейс:

* HTML\ XHTML
* CSS

Логика приложения и независимый от браузера интерфейс:

* Adobe Flash, Adobe Flex
* Java
* JavaScript
* Silverlight

Серверная часть может быть реализована практический с помощью любого языка программирования. Из данной главы можно сделать вывод:

**Веб-приложение состоит из клиентской и серверной частей между которыми разделена логика приложения. Существует множество технологий для реализации клиентской и серверной части.**

### Широко используемые клиентские технологии

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

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 HTML5** – 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.

### Технология 2d-context HTML5

2d-context HTML5 – 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.

Спецификация на данную технологию создается W3C, но на сегодняшний день спецификация еще не дописана.

Ниже приведен программный интерфейс доступный для использования с помощью языка программирования JavaScript:

**Методы сохранения и восстановления состояния контекста:**

* void save();
* void restore();

**Методы трансформации сцены**

* void scale( double x, double y);
* void rotate( double angle);
* void translate( double x, double y);
* void transform( double a, double b, double c, double d, double e, double f);
* void setTransform( double a, double b, double c, double d, double e, double f);

**Атрибут характеризующий уровень прозрачности**

* double globalAlpha;

**Атрубут задающий тип пересечения примитивов**

* DOMString globalCompositeOperation;

**Атрибут устанавливающий размытие изображения**

* boolean imageSmoothingEnabled;

**Атрибуты характеризующие внешний вид контуров и заливки**

* DOMString or CanvasGradient or CanvasPattern strokeStyle;
* DOMString or CanvasGradient or CanvasPattern fillStyle;

**Метод создающий линейный градиент**

* CanvasGradient createLinearGradient(double x0, double y0, double x1, double y1);

**Метод создающий радиальный градиент**

* CanvasGradient createRadialGradient(double x0, double y0, double r0, double x1, double y1, double r1);

**Атрибуты устанавливающие смещение тени по оси x и y**

* double shadowOffsetX;
* double shadowOffsetY;

**Атрибут характеризующий степень размытия тени**

* double shadowBlur;

**Атрибут характеризующий цвет тени**

* DOMString shadowColor;

**Метод стирающий область**

* void clearRect( double x, double y, double w, double h);

**Метод заполняющий область цветом**

* void fillRect( double x, double y, double w, double h);

**Метод рисующий рамку вокруг области**

* void strokeRect( double x, double y, double w, double h);

**Метод начала графического пути**

* void beginPath();

**Метод закраски пути**

* void fill(optional CanvasWindingRule w = "nonzero");
* void fill(Path path);

**Метод закраски контуров в пути**

* void stroke();
* void stroke(Path path);

**Метод рисующий рамку вокруг переданного элемента**

* void drawSystemFocusRing(Element element);
* void drawSystemFocusRing(Path path, Element element);

**Метод рисующий рамку вокруг выбранного элемента**

* boolean drawCustomFocusRing(Element element);
* boolean drawCustomFocusRing(Path path, Element element);

**Метод обрезающий вывод пути по контору**

* void clip(optional CanvasWindingRule w = "nonzero");
* void clip(Path path);

**Метод проверки принадлежности точки пути**

* boolean isPointInPath( double x, double y, optional CanvasWindingRule w = "nonzero");
* boolean isPointInPath(Path path, double x, double y);

**Методы выводящий текст с заданными парамтетрами**

* void fillText(DOMString text, double x, double y, optional double maxWidth);
* void strokeText(DOMString text, double x, double y, optional double maxWidth);

**Метод прорисовки изображении**

* void drawImage((HTMLImageElement or HTMLCanvasElement or HTMLVideoElement) image, double dx, double dy);
* void drawImage((HTMLImageElement or HTMLCanvasElement or HTMLVideoElement) image, double dx, double dy, double dw, double dh);
* void drawImage((HTMLImageElement or HTMLCanvasElement or HTMLVideoElement) image, double sx, double sy, double sw, double sh, double dx, double dy, double dw, double dh);
* void addHitRegion(HitRegionOptions options);
* void removeHitRegion(HitRegionOptions options);

**Метод создающий массив пикселей**

* ImageData createImageData( double sw, double sh);
* ImageData createImageData(ImageData imagedata);

**Метод получения массива пикселей с экрана**

* ImageData getImageData(double sx, double sy, double sw, double sh);

**Метод вывода на экран массива пикселей**

* void putImageData(ImageData imagedata, double dx, double dy, double dirtyX, double dirtyY, double dirtyWidth, double dirtyHeight);
* void putImageData(ImageData imagedata, double dx, double dy);

**Атрубуты с информацией о параметрах щрифта для вывод на экран**

* DOMString font;
* DOMString textAlign;
* DOMString textBaseline;

**Метод закрывающий путь**

* void closePath();

**Метод устанавливающий начальную точку для вывода линий**

* void moveTo( double x, double y);

**Метод проводящий линию**

* void lineTo( double x, double y);

**Методы рисующие кривые**

* void quadraticCurveTo( double cpx, double cpy, double x, double y);
* void bezierCurveTo( double cp1x, double cp1y, double cp2x, double cp2y, double x, double y);

**Метод рисующий дугу по точкам и радиусу**

* void arcTo( double x1, double y1, double x2, double y2, double radius);

**Метод рисующий прямоугольник**

* void rect( double x, double y, double w, double h);

**Метод рисующий дугу**

* void arc( double x, double y, double radius, double startAngle, double endAngle, optional boolean anticlockwise = false);

**Метод рисующий элипс**

* void ellipse( double x, double y, double radiusX, double radiusY, double rotation, double startAngle, double endAngle, boolean anticlockwise);

Как можно видеть 2d-context предоставляет простейшие методы для создания двумерного изображения. Методы и атрибуты 2d-context можно условно разделить на:

1. Методы работы с путями.
2. Методы и атрибуты для визуализации текста.
3. Методы для визуализации графических примитивов:
   1. Прямоугольник.
   2. Изображение.
   3. Линия.
   4. ломаная линия.
   5. Дуга.
   6. кривые Безье.
   7. Эллипс.
4. Методы для работы с массивом пикселей.
5. Методы сохранения и восстановления контекста.
6. Метод определения принадлежности точки пути.

Можно сделать вывом: **Методы и атрибуты 2d-context позволяют программировать изображение, но не позволяют работать с отдельными частями экрана как с объектами, для создания анимации требуют перерисовки экрана для каждого изменения, описание несложных сцен требуют достаточно большое количество кода.**

### 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.

### АРХИТЕКТУРА 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.

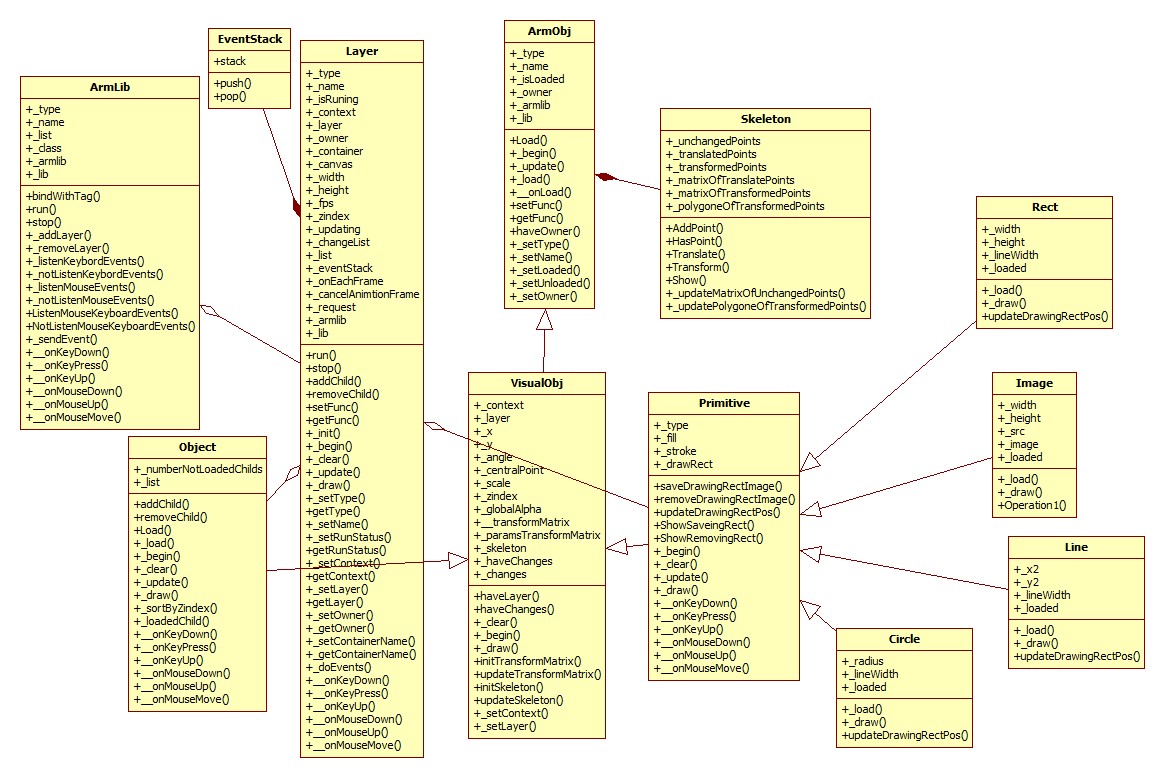
### Требования к программному каркасу

### Парадигма программирования

### Используемые алгоритмы

### 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.

### Принятые стандарты кодирования

### Интеграция классов каркаса

### Тестирование

### Используемые утилиты

### Средства разработки

### Интегрированная среда разработки

«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

### Система контроля версий

### Профайлер

### ЗАКЛЮЧЕНИЕ

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

### ПРИЛОЖЕНИЯ

### ТЗ

### Руководство программиста (ГОСТ 19.504)