|  |
| --- |
| **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](#_Toc357897334)

[СОДЕРЖАНИЕ 5](#_Toc357897335)

[ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 7](#_Toc357897336)

[ВВЕДЕНИЕ 9](#_Toc357897337)

[1. МУЛЬТИМЕДИЙНОЕ ВЕБ-ПРИЛОЖЕНИЯ 10](#_Toc357897338)

[1.1. Современное веб-приложение 10](#_Toc357897339)

[1.2. Архитектура веб-приложения 11](#_Toc357897340)

[1.3. Широко используемые клиентские технологии 12](#_Toc357897341)

[1.4. Технология 2d-context HTML5 14](#_Toc357897342)

[1.5. Сложности разработки с помощью технологии 2d-context HTML5 19](#_Toc357897343)

[1.6. Постановка задачи 20](#_Toc357897344)

[2. АРХИТЕКТУРА ПРОГРАММНОГО КАРКАСА 22](#_Toc357897345)

[2.1. Обзор существующих решений 22](#_Toc357897346)

[2.2. Требования к программному каркасу 23](#_Toc357897347)

[2.2.1. Парадигма программирования 23](#_Toc357897348)

[2.3. Используемые алгоритмы 23](#_Toc357897349)

[2.4. Диаграмма классов 23](#_Toc357897350)

[2.5. Используемые шаблоны проектирования 24](#_Toc357897351)

[2.6. Принятые стандарты кодирования 25](#_Toc357897352)

[2.7. Интеграция классов каркаса 27](#_Toc357897353)

[2.8. Используемые утилиты и библиотеки 28](#_Toc357897354)

[2.9. Средства разработки 29](#_Toc357897355)

[2.9.1. Интегрированная среда разработки 29](#_Toc357897356)

[2.9.2. Браузер 30](#_Toc357897357)

[2.9.3. Отладчик и профайлер 30](#_Toc357897358)

[2.9.4. Система контроля версий 31](#_Toc357897359)

[ЗАКЛЮЧЕНИЕ 32](#_Toc357897360)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc357897361)

[ПРИЛОЖЕНИЯ 34](#_Toc357897362)

[ТЗ 34](#_Toc357897363)

[Руководство программиста (ГОСТ 19.504) 34](#_Toc357897364)

### 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 –**

**BOM –**

**UNIX формат –**

**CamelCase нотация –**

### 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-ПРИЛОЖЕНИЕ

### 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.

**Вывод: Современной мультимединое веб-приложение может служить альтернативой настольным графическим приложениям, таким как игры.**

### 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.

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

### 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.

**Вывод: Существует достаточно обширный выбор графических технологий для веб-приложений.**

### 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);

**Вывод: Программный интерфейс 2d-context предоставляет минимально необходимые методы для создания двумерного статического изображения. Методом направленных на реализацию анимации интерфейс не содержит.**

### Сложности разработки с помощью 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 необходимо ознакомится со спецификацией, из прочтения которой можно выяснить возможности, требования и концепции использования. Для технологии 2d-context спецификация доступна по адресу [5].

В ходе анализа спецификации 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 заметить, что методы и атрибуты 2d-context позволяют:

* Программировать изображение, но не позволяют работать с отдельными частями экрана как с объектами.
* Для создания анимации требуются перерисовка экрана для изменения изображения. Это вызвано отсутствием методов для создания анимации.
* Описание несложных сцен требуют достаточно большое количество кода. Это вызвано тем фактом, что изображение создается с помощью минимального набора графических примитивов.

В ходе проверки заявленных методов в основных браузерах, было замеченно, что браузеры поддерживают спецификацию не в равной степени, так же ряд методов не имею реализации. Вероятно, это было вызвано не законченностью стандарта.

**Можно сделать вывод, что использование 2d-context влечет за собой следующие трудности:**

1. **Нет возможности работать с частями изображения как с самостоятельными объектами.**
2. **Для создания анимации требуются перерисовка экрана для изменения изображения.**
3. **Описание несложных сцен требуют достаточно большое количество кода.**
4. **Неготовый стандарт. Стандарт в стадии тестирования – W3C объявил о планах, согласно которым окончательная версия стандарта HTML5 будет утверждена лишь к 2014 году[6].**
5. **Отсутствие визуальных сред, вроде Flash Professional CS6.**
6. **Слабое развитие специализированных каркасов, вызванные, скорее всего, незавершенностью стандарта.**

### 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:

* Подробнее 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

Перед разработкой каркаса был совершен обзор существующих свободно распространяемым каркасов и библиотек для работы с 2d-context. Обзор был совершен для трех каркасов: jCanvaScript, KineticJS, LibCanvas.

1. jCanvaScript – Свободно распространяемая библиотека доступная по адресу [20] по лицензии MIT или GPL версии 2.
2. KineticJS – Свободно распространяемая библиотека доступная по адресу [21] по лицензии MIT или GPL версии 2.
3. LibCanvas – Свободно распространяемая библиотека доступная по адресу [22] по лицензии MIT или GPL.

Результаты обзора представлены в таблице 1.

**Таблица 1 – Возможности существующих решений**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Графический цикл | Слои | Объектное представление примитивов | Call-back функции для событий объектов | Работа с мышю и клавиатурой | классы обертки для примитивов | Загрузка мультимедиа-зависимостей объектов | Приоритеты очереди прорисовки | Модульность |
| jCanvaScript | + | + | + | -\+ | + | + | - | + | - |
| KineticJS | + | + | + | + | + | + | - | + | + |
| LibCanvas | + | + | + | + | + | + | - | + | - |

### Требования к программному каркасу

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.

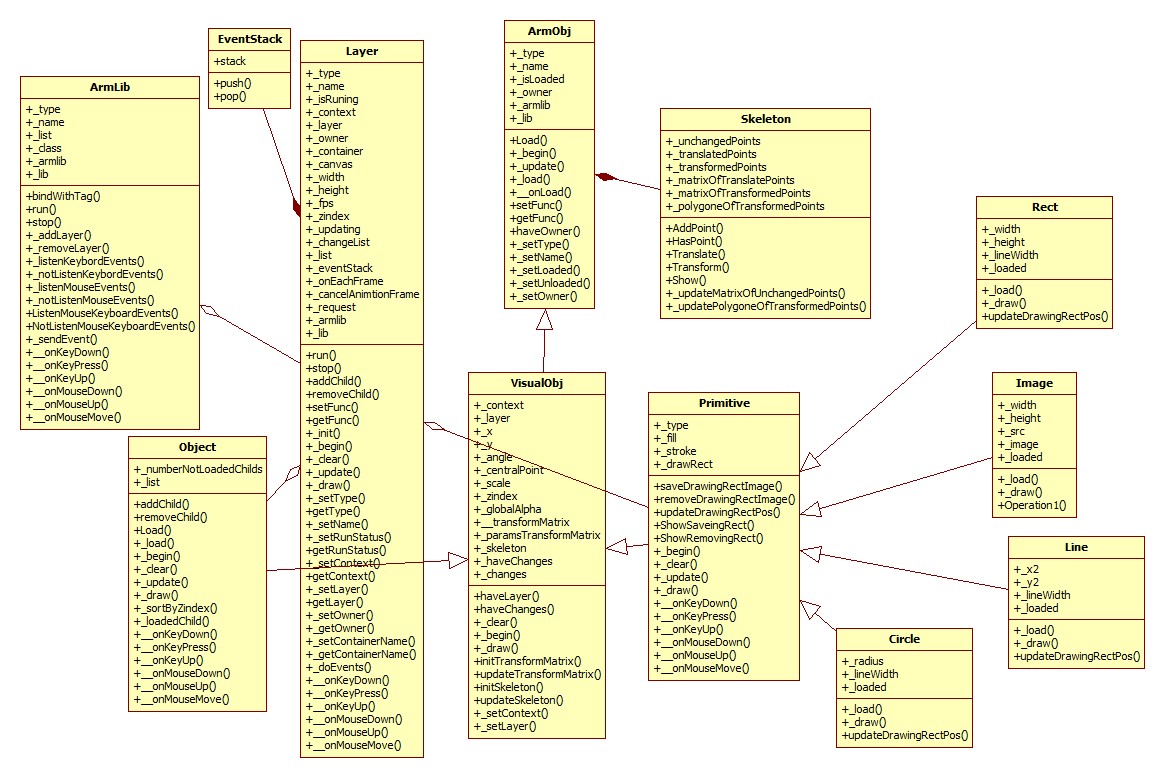
Необходимо учитывать, что размер каркаса влияет на общее время загрузки и старта приложения, а так как при создании достаточно универсального каркаса, в разных приложениях могут быть задействованы далеко не все возможности, необходимо реализовать модульности каркаса. Должна быть возможности собирать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

### 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.

**Вывод: Использование шаблонов проектирования позволило стандартизировать некоторые части каркаса, что может упростить работу с исходным кодом другим разработчиками.**

### Принятые стандарты кодирования

Общепринятым мнением и в частности мнением А. Александреску и Г. Саттера [17] признанно, что принятие стандарта кодирования положительно сказывается на процессе разработки:

«Хорошие стандарты кодирования могут принести не малую выгоду с различных точек зрения.

* *Повышение качества кода.* Работа в соответствии со стандартом приводит к однотипному решению одинаковых задач, что повышает ясноть кода и упрощает его сопровождение.
* *Повышение скорости разработки*. Разработчику не приходится решать все задачи и принимать решения «с нуля»»
* *Повышение уровня взаимодействия в команде*. Наличие стандарта позволяет уменьшить разногласия в команде и устранить ненужные дебаты по мелким вопросам, облегчает понимание и поддержку чужого кода членами команды.
* *Согласованность в работе*. При использовании стандарта разарботчики направляют свои услилия в верном направлении, на решение действительно важных задач.» [17].

Исходя из выше озвученных доводов для разработки был сформирован стандарт кодирования, изложенный в таблице 2.

**Таблица 2. Принятый стандарт кодирования**

|  |  |
| --- | --- |
| Объект стандартизации | Стандарт |
| Кодировка текста программы | UTF-8 без BOM |
| Символ перевода строки | LF (UNIX формат) |
| Пробелы | Операторы и аргументы функций отделяются одним пробелом. |
| Именование идентификаторов (переменные, называния функций, классы) | * CamelCase нотация * Имена классов с большой буквы * Имена открытых методов и переменных с маленькой буквы * Имена закрытых методов и переменных с нижнего подчеркивания. |
| Блоки и отступы | Скобка начала блока на той же строке, что оператор владелец блока. Операторы в блоке сдвинуты от начала блока на один символ <Tab>. |
| Структура директорий | Каждый класс расположен в отдельном файле. Файлы классов-наследников располагаются на директорию глубже класса родителя. |

### Интеграция классов каркаса

Термин «интеграция» обозначает операцию по объединению отдельных программных компонентов в функционирующую систему.

Выбранный способ интеграции определяет порядок кодирования и объединения компонентов.

Удачно выбранный способ интеграции может дать: [18]

* Возможность выявить принципиальные ошибки в архитектуре приложения при минимальном количестве закодированных компонентов системы.
* Упростить диагностику дефектов.
* Уменьшить число возможных ошибок.
* Уменьшение общего времени разработки.
* Лучшее качество кода.

Для интеграции классов каркаса была выбрана так называемая инкрементная функционально-ориентированная интеграция.

Инкрементная функционально-ориентированная интеграция – это вид инкрементной интеграции, при котором программный продукт собирается постепенным наращиванием системы с помощью добавления отдельных частей реализующих отдельные функции системы. В рамках каркаса интеграция шла через реализацию функций каркаса в порядке:

* Классы Layer и EventStack.
* ArmLib.
* Классы ArmObj и Skeleton.
* Класс VisualObj.
* Класс Object и Primitive.
* Класс Image.
* Класс Rect, Line, Circle.

**Вывод: Осознанный выбор способа интеграции позволил реализовывать каркас с помощью ряда итерации, в ходе которых постепенно наращивалась функциональность каркаса, что давало возможность проверить удачность реализованной за итерацию части.**

### Используемые утилиты и библиотеки

В ходе разработки была использована утилита jsLint и библиотека общего назначения Gizmo [19].

Утилита jsLint проверяет код на наличие синтаксических и широко распространенных семантических ошибок. Возможность подсветки семантических ошибок намного ускорило процесс кодирования из-за возможности быстро исключать многие ошибки данного класса с помощью данной утилиты. Была использована версия jsLint в виде плагина для среды разработки Netbeans, что являлось более удобным вариантом использования, чем запуск утилиты как консольного приложения.

Библиотека Gizmo была реализована в ходе кодирования каркаса. Данная библиотека была создана для хранения функций, чей функционал напрямую не относится к функциональности реализуемого каркаса, но необходим для его реализации. Примером таких функции может служить реализация обертки имитирующей синтаксис ООП для языка C++.

**Вывод: Использование утилиты для анализа кода помогло упростить процесс кодирования, а написание своей библиотеки общего назначения позволило повысить количество повторного использования уже реализованных решений и накопить набор потенциально полезных в других проектах, функций.**

### 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.

**Вывод: Использование интегрированной среды разработки дало возможность сосредоточиться на процессе кодирования, что вероятно уменьшило количество возможных ошибок и улучшило качество кода.**

### 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.

**Вывод: выделение трех целевых браузеров позволило оптимизировать каркаса только к наиболее востребованным браузерам.**

### Отладчик и профайлер

В ходе разработки использовался встроенные отладчик и профайлер браузера Chrome. Это было вызвано наличием сразу ряда инструментов в данном браузере.

**Вывод: использование отладчика и профайлера позволило выявлять ошибки и оценивать производительность отдельных участков кода, что позволило повысить производительность каркаса**

### Система контроля версий

В проекте была использована распределенная система контроля версий Git и бесплатный сервис для хостинга репозиториев под управлением Git – GitHub. Это позволило обезопасить проект от возможной порчи диска и утраты файлов проекта, упростило возможность переключатся между версиями кода. Выбор распределенной, а не централизованной системы контроля версий позволил не зависеть от наличия доступа в интернет.

**Вывод: Использование системы контроля версий уменьшило риск потери исходного кода проекта и упростило процесс кодирования.**

### ЗАКЛЮЧЕНИЕ

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Отличия от html4

https://rawgithub.com/whatwg/html-differences/master/Overview.html#apis

Спецификации html5

http://www.whatwg.org/specs/web-apps/current-work/multipage/

http://www.w3.org/TR/html5/

Application cache

[4] http://www.whatwg.org/specs/web-apps/current-work/multipage/offline.html#application-cache

1. \*3\* [5] 2d-context specification http://www.w3.org/TR/2010/WD-2dcontext-20101019/
2. [6] webgl-context specification http://www.khronos.org/registry/webgl/specs/latest/
3. X[7] История в браузерах – http://www.whatwg.org/specs/web-apps/current-work/multipage/history.html#history
4. X[8] Хранилища на стороне клиента – http://www.whatwg.org/specs/web-apps/current-work/multipage/webstorage.html#dom-localstorage
5. X[9] W3C – http://www.w3.org/
6. X[10] Adobe – http://www.adobe.com/
7. X[11] Adobe отказалась от поддержки flash-плеера для Linux – http://blogs.adobe.com/flashplayer/2012/02/adobe-and-google-partnering-for-flash-player-on-linux.html
8. X[12] Adobe отказалась от поддержки flash-плеера для мобильных платформ – http://blogs.adobe.com/flashplayer/2012/06/flash-player-and-android-update.html
9. \*1\* [13] WEBGL - A NEW DIMENSION FOR BROWSER EXPLOITATION – http://www.contextis.com/research/blog/webgl-new-dimension-browser-exploitation/
10. \*2\* [14] SVG vs Canvas Performance: Joel Oughton. — 03.04.2011 [Электронный ресурс]. — URL: http://joeloughton.com/blog/web-applications/svg-vs-canvas-performance/ (дата обращения: 04.03.2013)
11. [15] Стив Макконнелл - Совершенный код, 2-е издание (мастер-класс) – 2010
12. \*14\*[16] Стив Макконнелл - Совершенный код, 2-е издание (мастер-класс) – 2010, с 695.
13. \*9\*[17] Александреску А., Саттер Г. Стандарты программирования на C++. 101 правило и рекомендация, с. 10
14. [18] Стив Макконнелл - Совершенный код, 2-е издание (мастер-класс) – 2010, с 674.
15. [19] <http://github.com/sogimu/Gizmo.git>
16. \*12\* <http://www.jslint.com/>
17. \*13\* [20] <http://github.com/sogimu/Gizmo.git>
18. \*4\* [21] <http://jcscript.com/>
19. \*5\*[22] <http://kineticjs.com/>
20. \*6\*[23] <https://github.com/theshock/libcanvas>
21. \*10\*[24] Стив Макконнелл - Совершенный код, 2-е издание (мастер-класс) – 2010, с 678.
22. \*8\*[25] Никулин Е.А. – Компьютерная геометрия и алгоритмы машинной графики – 2005, с.34
23. \*7\*[26] <http://jquery.com/>
24. \*11\*[27] Стив Макконнелл - Совершенный код, 2-е издание (мастер-класс) – 2010, с 674.
25. \*15\* http://gtihub.com/

### ПРИЛОЖЕНИЯ

### ТЗ

### Руководство программиста (ГОСТ 19.504)