МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ имени В. И. ВЕРНАДСКОГО»
(ФГАОУ ВО «КФУ им. В. И. Вернадского»)
Таврическая академия (структурное подразделение)
Факультет математики и информатики
Кафедра информатики

Долгий Виктор Сергеевич

Построение динамической среды с использованием bitmap эффектов(Gaussian Blur и Radial Blur)

Выпускная квалификационная работа

Обучающегося 4 курса

Направления подготовки 01.03.02. Прикладная математика и информатика

Форма обучения очная

Научный руководитель
доцент кафедры информатики,
кандидат физико-математических наук В. Ф. Блыщик

К ЗАЩИТЕ ДОПУСКАЮ:

Заведующий кафедрой,
Кандидат физико-математических наук,
доцент Л. И. Руденко

Симферополь, 2021

Аннотация

Долгий В. С. Построение динамической среды с использованием bitmap эффектов (Gaussian Blur и Radial Blur) : выпускная квалификационная работа по направлению подготовки 01.03.02 Прикладная математика и информатика / В. С. Долгий– Симферополь : КФУ им. В. И. Вернадского, 2021. – 34 с.

Работа посвящена поэтапному исследованию работы и построению динамической среды, включающей в себя bitmap эффектов: Smoke Shader, Gaussin Blur и Radial Blur. Рассмотрены основы создания динамической среды на JavaScript, графическая библиотека pixl.js, технология WebGL, шейдер дыма и использованные размытия.

Ключевые слова: графика, JavaScript, Radial Blur, Gaussian Blur, WebGL, bitmap эффекты.

Страниц – 35, иллюстраций – 25, библиографических источников – 6

Введение 4
1. Теоретические основы технологий веб-программирования 5
1.1. Основные понятия и принципы работы JavaScript, WebGL, pixi.js, bitmap и шейдеров 5
1.1.1 Язык программирования JavaScript 5
1.1.2 Графическая библиотека Pixi.js 6
1.1.3 Технология WebGL 9
1.1.4 Понятия шейдера. Vertex Shader и Fragment Shader. 10
1.2 Гауссовское и радиальное размытия (Gaussian Blur and Radial Blur). 13
1.3. Другой пример практической реализации данной темы. 19
2. Разработка демонстрационного примера динамической среды 23
2.1. Начало работы , подключение библиотеки
PIXI.JS и локального сервера. 23
2.2. Эффект шейдера дыма 24
2.3 Подключение Гауссовского и радиального фильтров размытия с элементами рендеринга сцены 27
Заключение 34
Список использованных источников 35

Введение

В современном представление Интернета сайты выглядят красиво, благодаря продвинутой технологии графического анимирования веб-страниц. Хорошо подобранные оформление и анимация сайта это залог успеха любого предприятия или компании. Чтобы получить качественную анимацию нужно задействовать язык программирования JavaScript и его библиотеки, шейдеры и другие средства компьютерной графики.

Целью выпускной квалификационной работы является создание анимационного обучающего примера динамической среды с использованием bitmap эффектов на языке программирования JavaScript при помощи библиотеки pixl.js и технологии WebGL.

Анимационным проектом будет веб-страница с активным окном, внутри которого находиться динамическая среда с фоновым изображением на которое влияют эффекты самой среды и фильтров Гауссовского и радиального размытия.

Актуальность изучения данной темы является её новизна и современность по нескольким причинами:

1. Большое спрос на анимированные веб-страниц в сети Интернет.

2. Востребованность и большое количество возможностей языка программирования JavaScript.

3. Актуальность и поддержка технологии WebGL большим количеством браузеров.

Основными задачами выпускной квалификационной работы являются:

– изучение основных понятий, используемых при выполнении практической части.

– подробное изложение теоретического материала и подхода создания динамической среды.

– разработка анимационного обучающего примера динамической среды

В первом разделе работы излагается основная теоретическая часть работы с объяснением определений и демонстрируется другие существующий пример реализации текущей темы.

Во втором разделе описывается сам процесс создания практического примера и присутствует частичный листинг кода проекта для более полной картины подхода

В заключении подведены итоги и получены результаты выпускной квалификационной работы.

1. Теоретические основы технологий веб-программирования

1.1. Основные понятия и принципы работы JavaScript, WebGL, pixi.js, bitmap и шейдеров

1.1.1 Язык программирования JavaScript

JavaScript — сценарный, или скриптовый язык. Скрипт является программным кодом — набором инструкций, которым не нужна дополнительная обработка (например, компиляция) перед запуском. Чаще всего этот язык программирования используется для разработки веб-страниц. Его главным преимуществом является изначальное наличие базовых элементов, таких как:

– структуры данных

– алгоритмы

– объектно-ориентированную модель

Плюсы и минусы JavaScript:

– Широкая распространенность. Буквально каждый браузер и каждая операционная система поддерживает этот язык. Без проблем запускается на стационарном компьютере и мобильном устройстве. Не требует разработки проекта отдельно под каждую платформу[5].

– JavaScript-приложение не требует установки на компьютер пользователя. Скрипты запускаются и выполняются непосредственно внутри браузера.

– Взаимодействие с Office. Как правило некоторые языки нужно редактировать только в определенной среде программирования. Но изменять код JS можно в редакторах Office, что позволит не загружать дополнительное программное обеспечение[3].

– Язык высокого уровня. Это означает, что существует уже определенный набор команд, с помощью которых можно написать

код. Не нужно прописывать действия на машинном коде. Язык высокого уровня упрощает работу программисту, но при этом не сужает спектр его возможностей.

– Быстрый для пользователя. Преимущество скорости JS получает благодаря тому, что код не компилируется на стороне клиента, ведь скрипт выполняется в браузере пользователя. Это значительно уменьшает нагрузку на сервер по сравнению с веб-приложениями, написанными на других языках.

– В JavaScript не поможет компилятор. Невозможно заранее узнать, работает ли программа до того момента пока не дойдёт до строчки с ошибкой, где лишь небольшая опечатка. Тогда выполнение программы просто остановится. Здесь компилятор не поможет узнать где ошибка или как можно оптимизировать код.[4]

– Необходимость использование сторонних библиотек. В языке не имеется поддержки работы с файлами и потоками ввода и вывода. Поэтому приходится применять фреймворки и библиотеки, написанные другими разработчиками.

– Низкая безопасность. В сети Интернет с легкостью скачиваются исходники для популярных скриптов JS. Поэтому невозможно гарантировать что JS - приложение не может быть взломано. Еще одной проблемой является то, что код всегда выполняется на стороне пользователя, а не на сервере где можно увидеть поломку кода.

– Повсеместное использование. Это значит, если выйдет язык программирования, который будет решать те же задачи лучше и с меньшим количеством ошибок, то программы на JS быстро устареют. Попытки создать такой язык становятся успешнее, к примеру Google Dart, находящийся всё ближе и ближе к тому чтобы заменить JavaScript.

1.1.2 Графическая библиотека Pixi.js

Pixi.js - это библиотека рендеринга 2D-спрайтов, которая помогает отображать, анимировать и управлять интерактивной графикой, чтобы было легче создавать игры и приложения с использованием JavaScript и других технологий HTML5. Он имеет разумный, лаконичный API и включает в огромное количество полезных функций, таких как поддержка атласов текстур и

обеспечение оптимизированной системы для анимирования спрайтов (интерактивных изображений). Он также дает вам полный график сцен, так что вы можете создавать иерархии вложенных спрайтов, а также позволяет прикреплять мышь и сенсорные события непосредственно к спрайтам.[6]

Библиотека pixi.js используется во множестве различных графических и не только проектов. Она не является архитектурным или физическим движком, это самостоятельный фреймворк для качественного рендеринга двумерной графики, раскрывающий все возможности WebGL. Это позволяет создавать как простые анимационные сцены, так и полноценные компьютерные видеоигры, запускаемые на обычном браузере(Рис.1.1- 1.3).

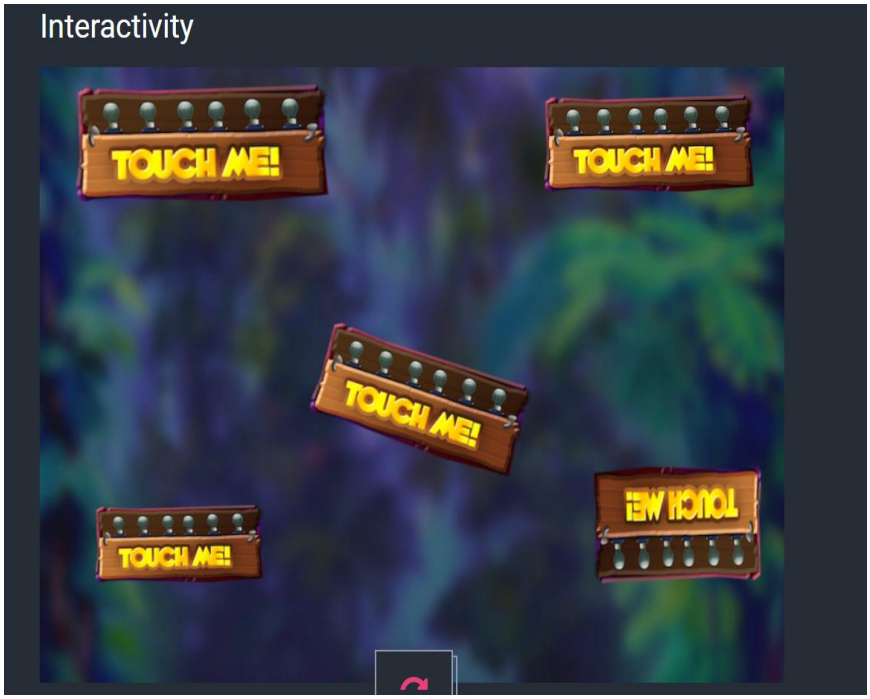

Рис 1.1. Сцена с интерактивными объектами на pixi.js

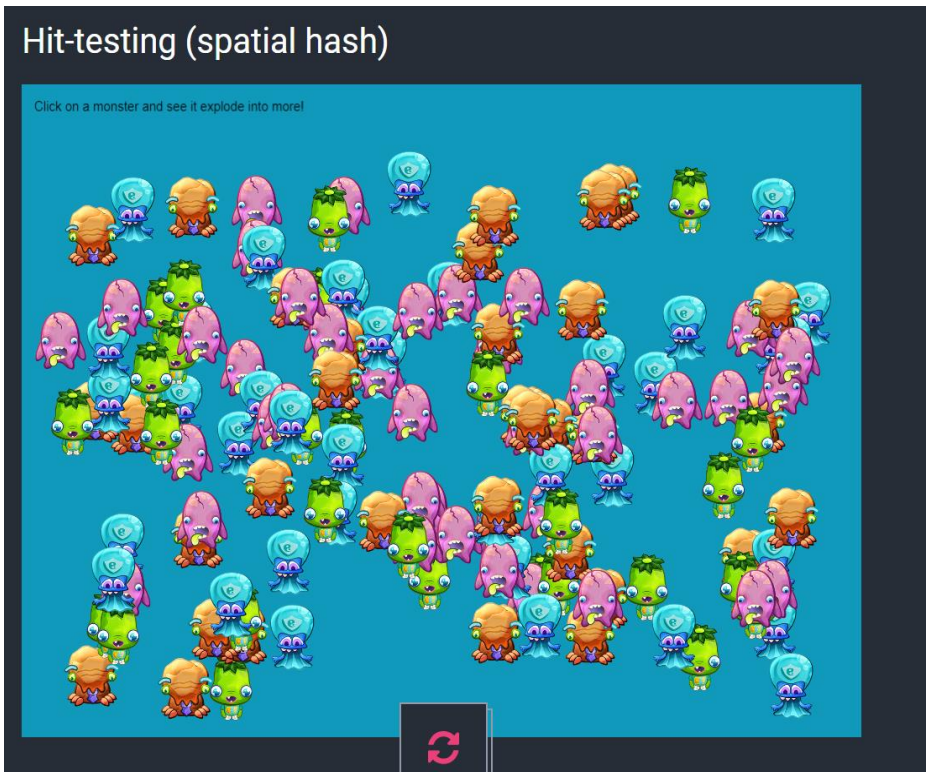

Рис 1.2. Canvas с объектами, количество которых увеличивается по нажатию клавиши.

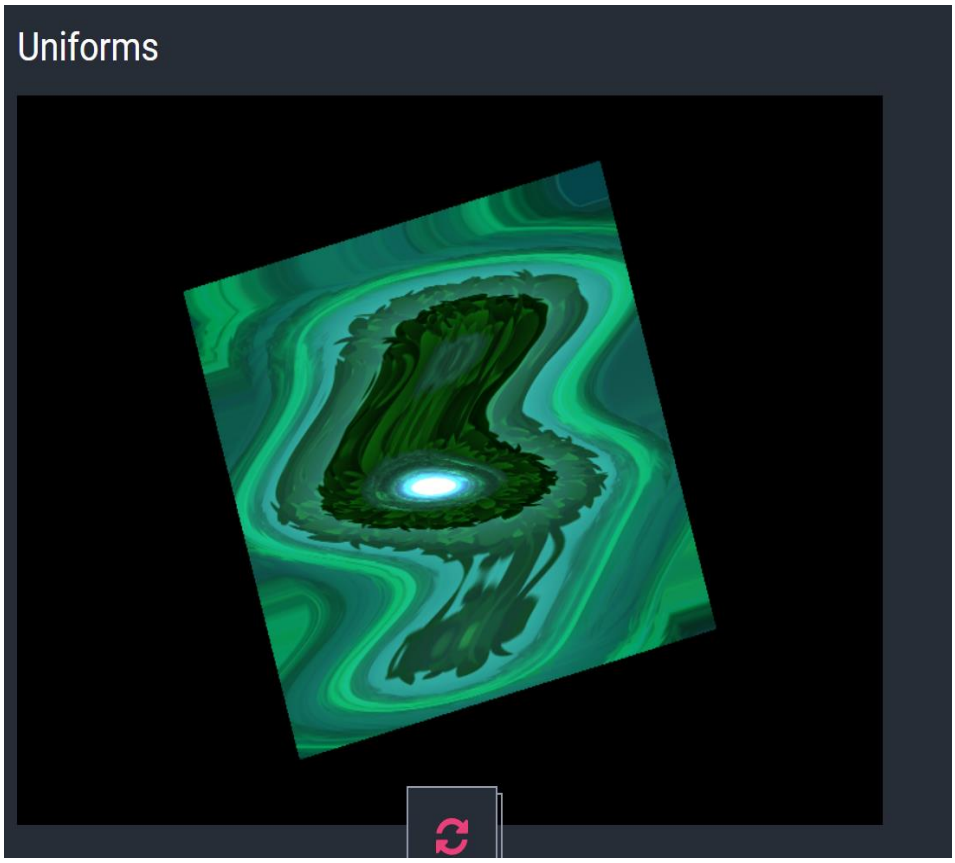

Рис 1.3. Круговое вращение квадрата с анимацией внутри него

1.1.3 Технология WebGL

WebGL является технологией, придуманной на основе OpenGL ES 2.0 и предназначенной для рисования и отображения интерактивной 2D- и 3D-графики в веб-браузерах. При этом для работы с этой технологией не нужно использовать другие плагины или библиотеки. Работа веб- приложений с использованием WebGL основана на коде JavaScript, а некоторые элементы кода - шейдеры исполняются прямо на графических процессорах видеокарт, с помощью чего программисты могут получить доступ к большему количеству ресурсов компьютера, а также увеличить его быстродействие. Если при создании приложений работающих с 2d и 3d-графикой часто возникает проблема с ограничением разработки только для одной платформы, то в технологии WebGL главным ограничениием является возможность или невозможность воспроизведения браузером такой графики. А сами веб-приложения, созданные на этой платформе будут везде доступны любом устройстве, независимо от операционной системы.[1]

Как работает WebGL? С момента запуска страницы до полного отображения содержимого проходит ряд инструкций. Во-первых подгружаются все использующиеся ресурсы: шейдеры, скрипты, отвечающие за обработку данных, текстуры и прочее. Затем инициализируется WebGL, что означает , что браузеру сообщается что отрисовка графических элементов будет происходить при помощи этой технологии. После чего в видеокарте обновляются uniform (глобальные) переменные, которые она использует для отрисовки изображений. Это значит, что любой новый кадр будет рисоваться заново. При помощи математических функций и имеющихся данных рассчитываются данные (координаты) трех вершин отображаемых фигур, находящиеся в трех измерениях (x, y, z) между 1 и -1, независящих от размера холста. Из этих вершин собирается массив данных или матрица, с помощью которой будет рисоваться изображение. Затем каждую вершину из этого массива нужно поместить в двумерную плоскость с учетом искажения глубины, то есть в зависимости от отдаления разных точек от пользователя, эти точки тоже изменят свое положение, что в двух других осях они имеют одинаковые координаты. Далее с помощью Vertex Shader происходит расстановка вершин. После чего

происходит растрирование изображения, то есть заполняются все пиксели находящимися между данными вершинами. Заполнение использует графические примитивы такие как: точки, линии, треугольники. Затем Fragment Shader заполняет каждый пиксель фигуры или изображения цветом, получаемые от значения переменной или текстуры. Цвет может от изменения параметров или освещения.

Преимущества использования WebGL:

– Поддержка этой технологии множеством браузеров. Отсутствие зависимости от платформы.

– Использование языка JavaScript, который достаточно распространен в мире программирования.

– Автоматизированное управление памятью. В отличие от OpenGL в WebGL не нужно лишних действий для выделения и очистки памяти.

– Поскольку WebGL для рендеринга графики использует графический процессор на видеокарте (GPU), то для этой технологии характерна высокая производительность, которая сравнима с производительностью нативных приложений.

1.1.4 Понятия шейдера. Vertex Shader и Fragment Shader

Шейдером в широком смысле называется программа для визуального определения поверхности объекта. Это может быть описание освещения, текстуризации, постобработки и т.п. Шейдеры выросли из работ Кука (Cook's shade trees) и Перлина (Perlin’s pixel stream language). Наиболее известны шейдеры RenderMan Shading Language. Программируемые шейдеры были впервые представлены в RenderMan компании Pixar, там определены несколько типов шейдеров: light source shaders, surface shaders, и другие. Эти шейдеры в большинстве своём выполняются программно при помощи универсальных процессоров так как у них отсутствует полная аппаратная реализация. В дальнейшем, многие ученые-программисты изобретали языки, имевшие схожесть с RenderMan языком, но те были нужны для аппаратного ускорения: система PixelFlow (Olano и Lastra), Quake Shader Language и другие. Шейдеры RenderMan разбивались на некоторое количество шагов, которые

комбинировались во фреймбуфере. Позднее появились языки, которые мы видим аппаратно ускоренными в DirectX и OpenGL. Так шейдеры были адаптированы для графических приложений работающих в реальном времени.[2]

Vertex Shaders (Вершинные Шейдеры) — это программы, выполняемые видеопроцессорами, которые проводят математические операции с вершинами (vertex, из них состоят 3D объекты в играх), они предоставляют возможность исполнять алгоритмы связанные с изменением параметров вершин и их освещения (T&L — Transform & Lighting). Каждая вершина представляется некоторым количеством переменных, например, положение вершины в 3D пространстве определяется координатами: x, y и z. Вершинные шейдеры, в зависимости от алгоритмов, изменяют данные в процессе своей работы, к примеру, производя вычисления и записывая новые координаты и/или цвет. То есть, входные данные вершинного шейдера — данные об одной вершине геометрической модели, которая в данный момент обрабатывается. Обычно это координаты в пространстве, нормаль, компоненты цвета и текстурные координаты. Результирующие данные выполняемой программы служат входными для дальнейшей части конвейера, растеризатор делает линейную интерполяцию входных данных для поверхности треугольника и для каждого пиксел исполняет соответствующий пиксельный шейдер.

Fragment Shader (Фрагментный Шейдер)-это программа, которая обрабатывает фрагмент, созданный растеризацией в набор цветов и одно значение глубины.

Выходные данные этого шейдера - это значение глубины, возможное значение трафарета (не измененное шейдером фрагментов) и ноль или более значений цвета, которые могут быть записаны в буферы в текущих буферах.

Фрагментный шейдер принимает один фрагмент в качестве входных данных и производят один фрагмент в качестве выходных данных.

Этот шейдер технически является необязательной частью шейдерной обработки. Если не использовать шейдер, то выходные результаты будут не особо значительными. Однако значения глубины и трафарета для выходного фрагмента будут такими же , что

и для входных данных.

Во время рендеринга очень удобно когда единственным результатом является глубина фрагмента и можно использовать эту глубину, вычисленную системой. Такой рендеринг только по глубине нужен для операций с помощью которых происходит отображение теней, а также оптимизация глубины.

В отличие от любого другого шейдера, фрагментный шейдер может создавать случайные производные. Из этого следует что они используют большее количество функций текстуризации.

Шейдер также имеет доступ к команде discard. При выполнении этой команды выходные значения фрагмента отбрасываются, что значит что фрагмент не переходит к следующим шагам конвейера, поэтому все выходные данные шейдера пропадают. Хотя выполнение шейдера фрагментов технически остановлено отбрасыванием, в реальных системах оно может продолжаться и дальше. Такие системы требуются для остановки работы хранилища изображений, атомарного счетчика и буфера хранения шейдеров записи объектов, выданных после сброса (такие операции выполняются до работы сброса, как и ожидалось).

Входные данные для шейдера производятся системой из прошлых операций с функцией, которая является фиксированной, и интерполируются по всей поверхности примитива.

Входные данные которые определил пользователь будут интерполироваться в соответствии с квалификаторами интерполяции, которые были объявлены для входных переменных, объявленных этим шейдером фрагментов. Входные переменные шейдера фрагментов должны быть объявлены так чтобы было согласование интерфейса между шагами шейдера. В частности, между этим этапом и последним шагом шейдера обработки вершин в объекте программ.

Bitmap в цифровых изображениях — матрица, хранящая значения элементов изображения (пикселов). При отображении информации на экране дисплея (мониторе) одному элементу изображения (пикселу) может соответствовать один или более битов памяти. При этом обеспечивается высокая гибкость в отображении текстовой и графической информации[1].

1.2 Гауссовское и радиальное размытия (Gaussian Blur and Radial Blur)

Гауссовское размытие (Gaussian Blur) — метод размытия изображения при помощи функции Гаусса.

Формула функции Гаусса:

$$G(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

x, y – координаты, σ – среднеквадратичное отклонение нормального распределения.

Значения этого распределения используются для получения матрицы сверстки, которая применяется к начальному изображению (Рис. 1.4, 1.6). Новое значение каждого пикселя становится средневзвешенным значению окрестности этого пикселя. Значение начального пикселя становится самым большим (по весу пикселя), а веса соседних пикселей становятся средневзвешенными, что приводит к размытию изображения(Рис. 1.5, 1.7).

Теоретически функция Гаусса в каждой точке изображения будет отличной от нуля, а это означает, что в вычислении значения для каждого пикселя участвует всё изображение. На практике, при вычислении дискретного приближения функции Гаусса, пиксели на расстоянии более 3σ оказывают на него достаточно малое влияние, поэтому могут считаться фактически нулевыми. Таким образом, значения пикселей вне этого диапазона можно игнорировать. Обычно программе обработки изображений требуется лишь вычислить матрицу с размерами $\lceil 6\sigma \times 6\sigma \rceil$ (где $\lceil . \rceil$ — функция округления в большую сторону), чтобы гарантировать результат, достаточно близкий к результату, полученному с помощью полного распределения Гаусса.

Помимо круговой симметрии, размытие по Гауссу может применяться к двумерному изображению как два независимых одномерных вычисления, и поэтому оно является разделяемым фильтром. Это означает, что эффект от применения двумерной матрицы также может быть достигнут путём применения серии одномерных матриц в горизонтальном направлении с последующим повторением процесса в вертикальном направлении. С вычислительной точки зрения это полезное свойство, так как расчёт может быть выполнен за время, $O(\omega_{kernel}\omega_{image}\ h_{image}) +$

$O\left(h_{kernel}\omega_{image}h_{image}\right)$ где h — высота, а ω — ширина, в отличие от $O(\omega_{kernel}\ h_{kernel}\ \omega_{image}\ h_{image})$ при использовании неразделимого ядра.

Применение последовательных размытий по Гауссу к изображению имеет тот же эффект, что и применение одного большего размытия по Гауссу, радиус которого является квадратным корнем из суммы квадратов фактически применённых радиусов размытия. Например, применение последовательных размытий по Гауссу с радиусами 6 и 8 даёт те же результаты, что и применение одного размытия по Гауссу с радиусом 10, поскольку $\sqrt{6^2 + 8^2} = 10$. Из-за этой взаимосвязи время обработки не может быть сэкономлено путём имитации размытия по Гауссу последовательными более мелкими размытиями — необходимое время будет по крайней мере таким же, как и при выполнении одного большого размытия.

Рис 1.4. Начальное изображение

Рис 1.5. Изображение размытое по Гауссу

Рис 1.6. Начальное изображение

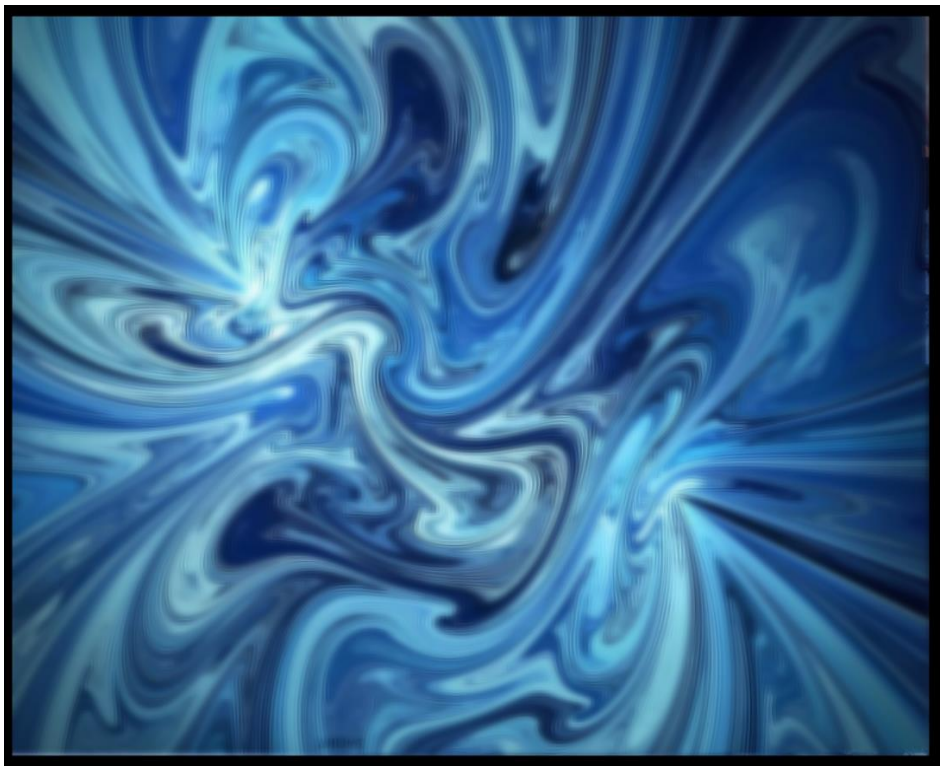

Рис 1.7. Изображение размытое по Гауссу

Эффект размытия по Гауссу обычно создаётся путём свёртки изображения ядром КИХ с использованием значений функции Гаусса.

На практике лучше всего использовать свойство разделимости размытия по Гауссу, выполняя процесс за два прохода. На первом проходе используется одномерное ядро для размытия изображения только в горизонтальном или вертикальном направлении. Во втором проходе то же одномерное ядро используется для размытия в другом направлении. Результирующий эффект такой же, как при свёртке с двумерным ядром за один проход, но при этом требуется меньше вычислений.

Дискретность обычно достигается путём выбора дискретных точек, обычно в позициях, соответствующих центральным точкам каждого пикселя. Это позволяет снизить вычислительные затраты, но для очень маленьких ядер фильтров точечная выборка функции Гаусса с очень маленьким количеством выборок приводит к большой ошибке.

В этих случаях точность поддерживается (с небольшими вычислительными затратами) путем интегрирования функции Гаусса по площади каждого пикселя.

При преобразовании непрерывных значений функции Гаусса в дискретные значения, необходимые для ядра, сумма значений будет отличаться от 1. Это приведёт к потемнению или осветлению изображения. Чтобы исправить этот эффект, значения можно нормализовать, разделив каждый элемент в ядре на сумму всех элементов.

Эффективность КИХ снижается для высоких значений σ. Существуют альтернативы КИХ-фильтру. К ним относятся очень быстрое множественное размытие по рамке, быстрый и точный детектор границ Дерише, «стековое размытие» на основе размытии по рамке и многое другое.

Радиальное размытие (Radial Blur) - способ изменения исходного изображения (Рис.1.8, 1.10), который смазывает каждый пиксель, двигаясь по концентрическим круговым линиям от исходной центральной точки, из-за чего создается ощущения кругового размытия (Рис. 1.9, 1.11).

Рис 1.8. Исходное изображение

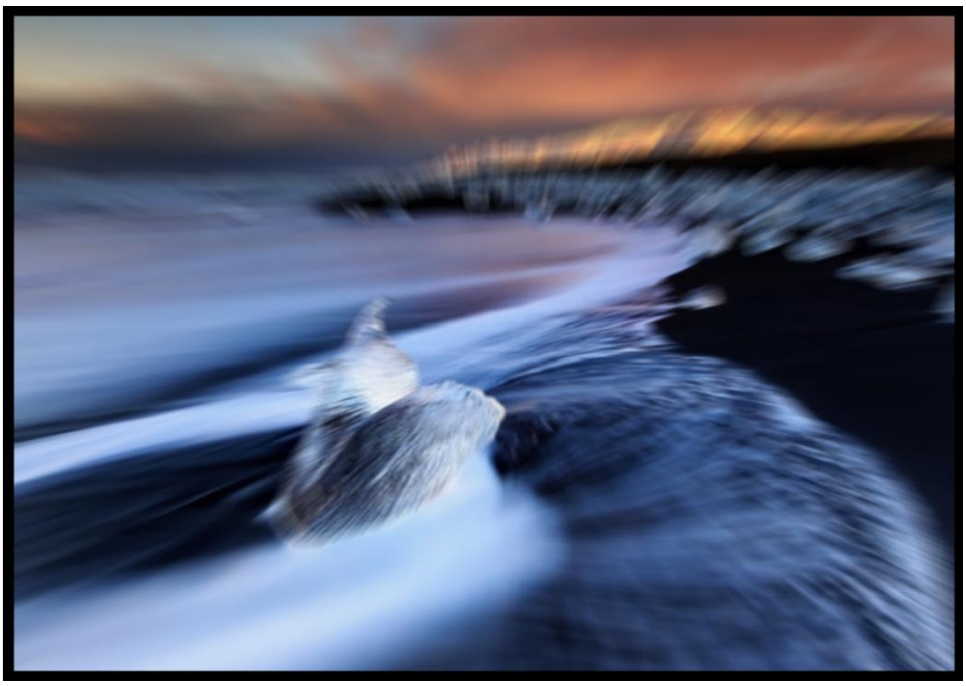

Рис 9. Радиальное размытие изображения

Рис.1.10. Исходное изображение

Рис 1.11. Радиальное размытие изображения

1.3. Другой пример практической реализации данной темы

Размытие изображений это конечно же хорошо, но есть кое-что покрупнее и получше. Динамическая среда в виде аквариума с анимированными объектами (рыбами). Такое может предложить зарубежный сайт и по совместительству база открытых исходников openbase.com. Сам анимационный пример является лишь дополнением к сайту и прекрасным наглядным помощником при выборе фильтра для изображений или динамической среды (Рис.1.12).

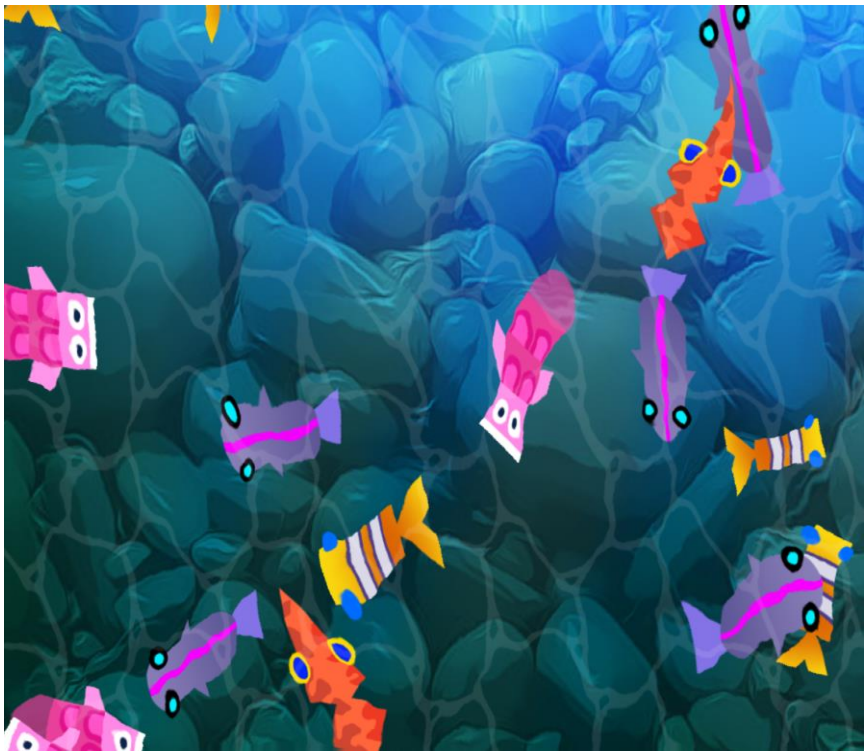

Рис.1.12. Аквариум Openbase

Рядом с активным окном присутствует список различных эффектов размытия (Рис.1.13), которые могут быть применимы к данной анимационной сцене. Можно включить одновременно несколько эффектов, остановить анимацию, включить и отключить рендеринг сцены (Рис.1.14). При выборе любого фильтра изображения предоставляется набор параметров, которые может изменить пользователь (Рис.1.15).

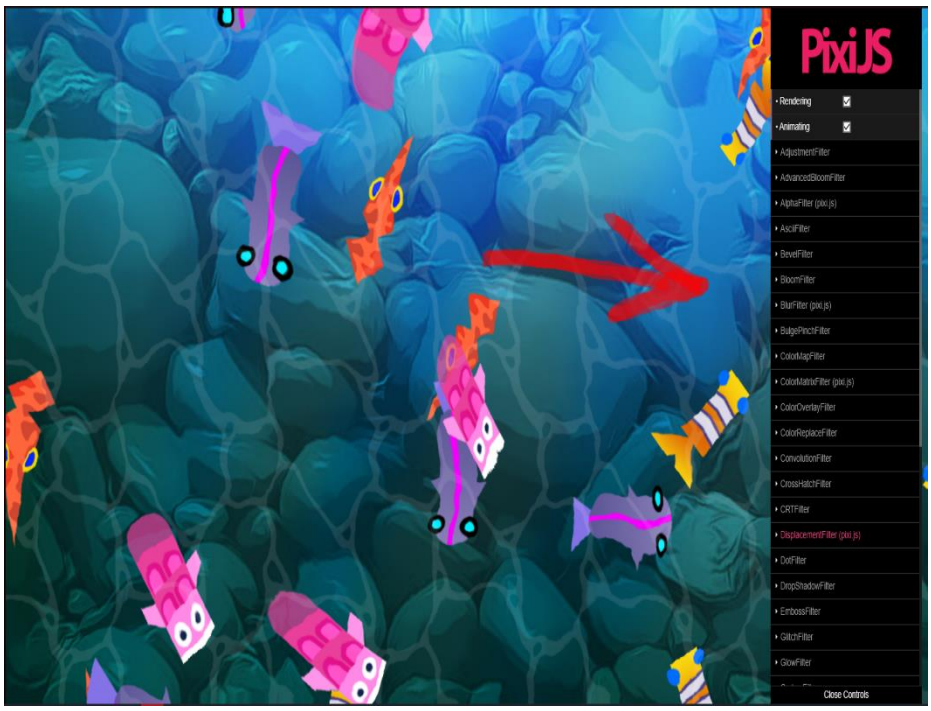

Рис.1.13 Список различных эффектов аквариума

Рис.1.14 Одновременно наложенные фильтры радиального размытия и рефлексии изображения

Рис.1.15 Регулируемые параметры для каждого фильтра

Данный анимационный пример имеет огромный спектр функций. Помимо фильтров размытия имеются эффект теней и контуров для движущихся объектов , а также эффект зума и сепии.

В основном же сам сайт является неким аналогом GitHub, но для тех, кто использует JavaScript и его библиотек. Сама база исходников в этом веб-проекте имеет возможность регистрации собственного аккаунта, оценивания алгоритмов и исходников добавленных другими пользователями и также присутствуют окна статистики использования и скачивания библиотек и исходников JavaScript (Рис.1.16).

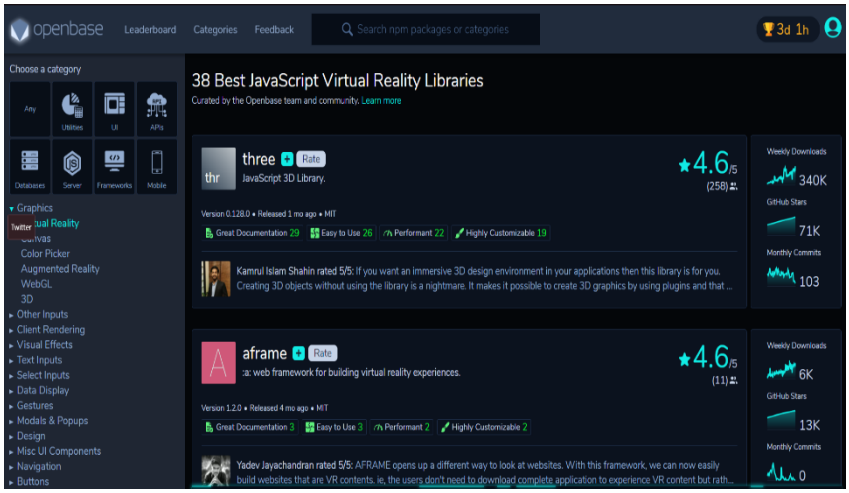

Рис.1.16 Основная часть Openbase

2. Разработка демонстрационного примера динамической среды.

2.1. Начало работы , подключение библиотеки PIXI.JS и локального сервера.

Несмотря на то, что веб-программисты часто используют такие локальные сервера как OpenServer, Denver и т.д. Было решено использовать широко известный редактор компании Microsoft под названием Visual Studio Code. Его преимуществом является так называемый маркет дополнений к редактору, в нем можно скачать и установить прямо в него IDE для множества известных языков программирования. Из этих богатств местного маркета было взято лишь несколько нужных и важных вещей таких как поддержка JavaScript, HTML5 и CSS и конечно же самая удобная фишка редактора – встроенный локальный сервер (Рис. 2.1).

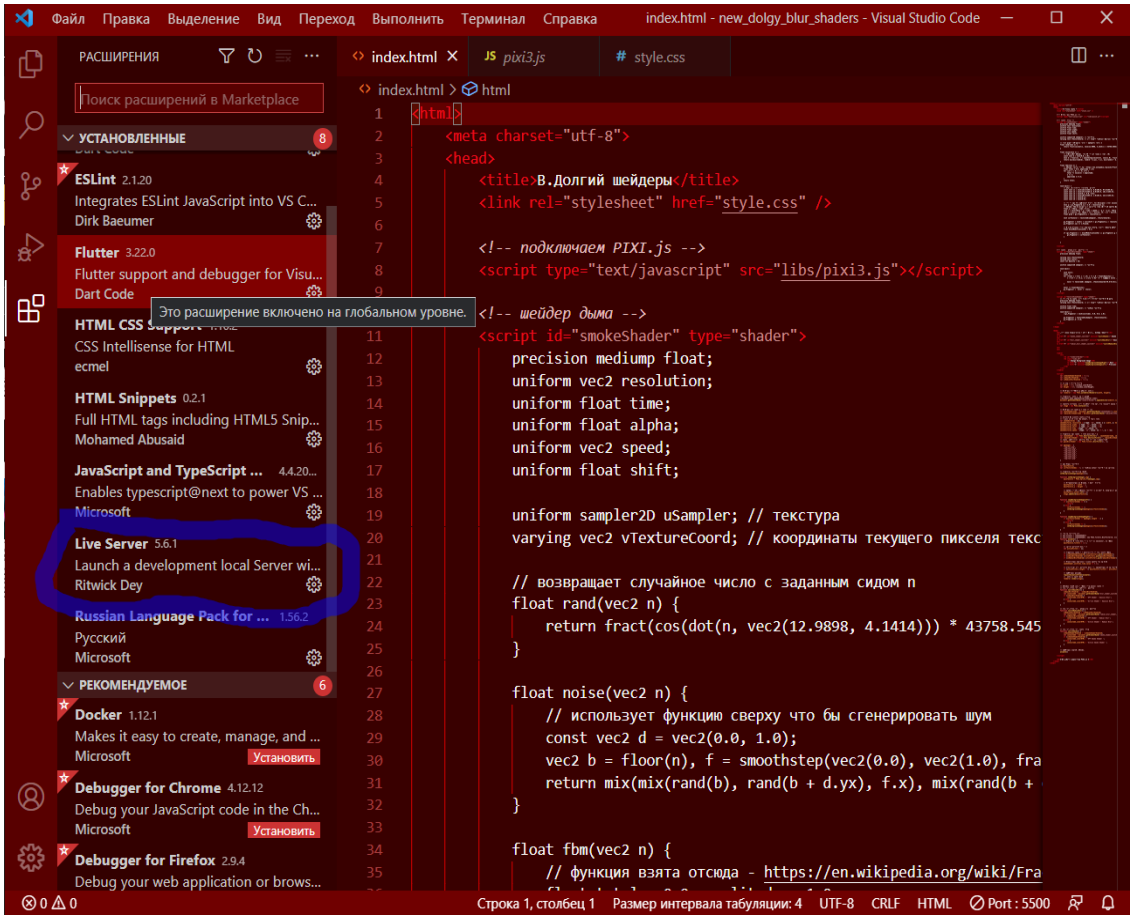

Рис 2.1. Live Server внутри редактора Visual Studio Code

Далее происходит подключение библиотеки pixi.js. Загрузка библиотеки с официального сайта разработчика и ссылка на

библиотеку в основном файле проекта (index.html). После чего создание небольшого макета будущей веб-страницы: активное анимационное окно canvas размером 850 x 600 и несколько активных кнопок.

2.2. Эффект шейдера дыма

В описанном выше примере динамической средой выступал аквариум с рыбками с возможностью наложения эффектов на анимацию. Здесь будет присутствовать теоретически тоже самое , но в меньшем масштабе. Это будет начальным примером для обучения основам. Заменой динамической среды послужит визуализация эффекта дыма, накладываемого на изображение. Эффект дыма отображен в программном коде под видом шейдера(См. Листинг 2.1).

```
  <!-- шейдер дыма -->
<script id="smokeShader" type="shader">
precision mediump float;uniform vec2 resolution;
uniform float time;
uniform float alpha;
uniform vec2 speed;
uniform float shift;
uniform sampler2D uSampler; // текстура
varying vec2 vTextureCoord; // координаты текущего пикселя тек
стуры
 // возвращает случайное число с заданным сидом n
float rand(vec2 n) {
return fract(cos(dot(n, vec2(12.9898, 4.1414))) * 43758.5453);
                  }
float noise(vec2 n) {
// использует функцию сверху что бы сгенерировать шум
const vec2 d = vec2(0.0, 1.0);
vec2 b = floor(n), f = smoothstep(vec2(0.0), vec2(1.0), fract(
n))
return mix(mix(rand(b), rand(b + d.yx), f.x), mix(rand(b + d.x
y), rand(b + d.yy), f.x), f.y);
                  }
float fbm(vec2 n)
float total = 0.0, amplitude = 1.0;
 for (int i = 0; i < 4; i++) {
total += noise(n) * amplitude;
n += n;
amplitude *= 0.5;
                     }
```

```
return total;
                }
void main() {
const vec3 c1 = vec3(126.0/255.0, 0.0/255.0, 97.0/255.0);
const vec3 c2 = vec3(173.0/255.0, 0.0/255.0, 161.4/255.0);
const vec3 c3 = vec3(0.2, 0.0, 0.0);
const vec3 c4 = vec3(164.0/255.0, 1.0/255.0, 214.4/255.0);
const vec3 c5 = vec3(0.1);
const vec3 c6 = vec3(0.9);
// это то как дым наполняет холст, для понимания можно изменит
ь 3.0 на 1.0
 vec2 p = gl_FragCoord.xy * 3.0 / resolution.xx;
// Функция fbm принимает p в качестве сида (поэтому каждый пик
сель выглядит по
разному) и время (поэтому он смещается со временем)
float q = fbm(p - time * 0.1);
vec2 r = vec2(fbm(p + q + time * speed.x - p.x - p.y), fbm(p +
 q - time * speed.y));
vec3 c = mix(c1, c2, fbm(p + r)) + mix(c3, c4, r.x) - mix(c5,
c6, r.y);
float grad = gl_FragCoord.y / resolution.y;
vec4 curTexColor = texture2D(uSampler, vTextureCoord);

gl_FragColor = vec4(c * cos(shift * gl_FragCoord.y / resolutio
n.y) * 0.9, 1.0);
gl_FragColor.xyz *= 1.0-grad;
// при превышении этого значения цвета, вместо эффекта будет п
оказываться текстура
float blackModificationVal = 0.3;
if (gl_FragColor.r > blackModificationVal || gl_FragColor.g >
blackModificationVal || gl_FragColor.b > blackModificationVal)

{
gl_FragColor = curTexColor;
                    }
            }
</script>
```

Листинг 2.1 Шейдер дыма

В самом шейдере указываются значения через uniform – переменные для эффекта дыма, которые влияют на исходное изображение (Рис.2.2). В главной функции имеются основные расчеты, которые определяют как дым должен заполнять холст (Рис.2.3).

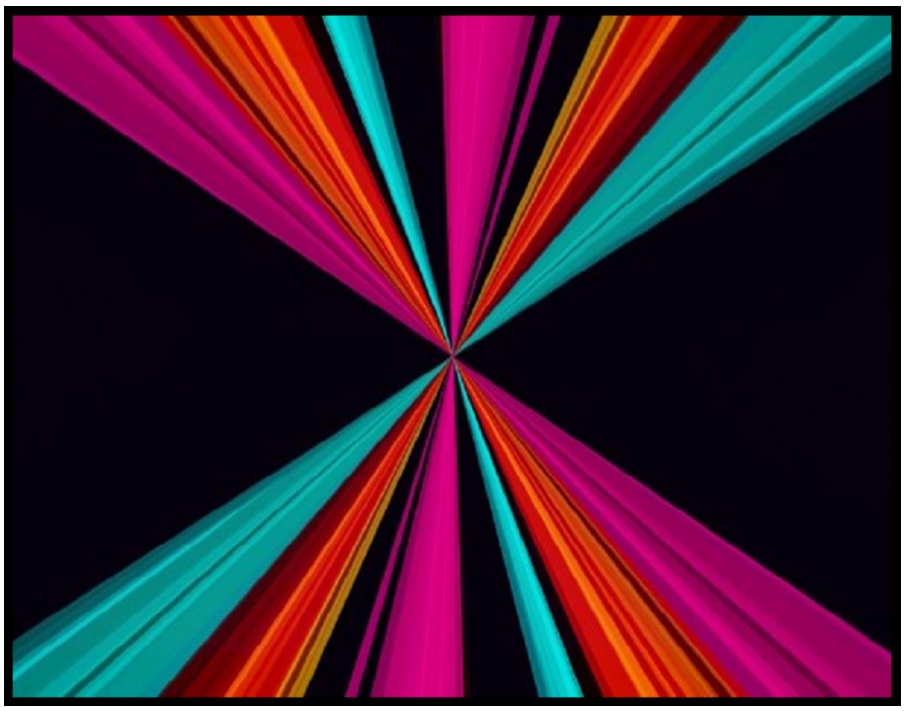

Рис 2.2. Исходное изображение

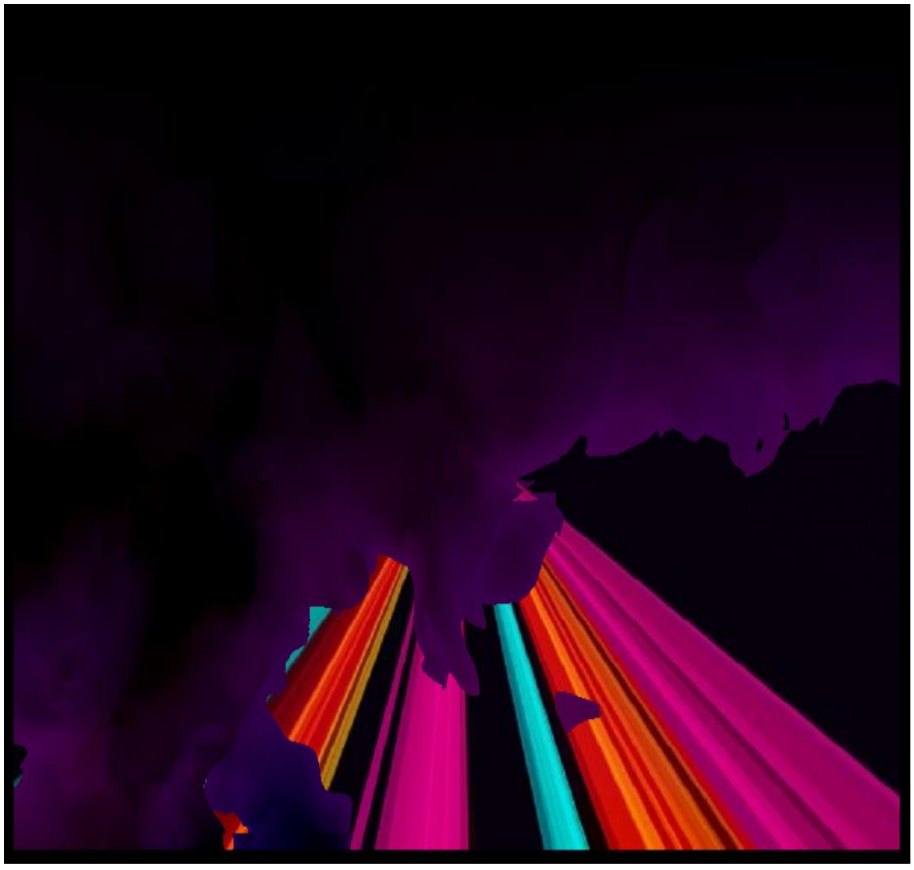

Рис 2.3. Заполненный дымом холст

При добавлении возможности переключения заднего фона пришлось пожертвовать качеством детализации в угоду нормального распределения ресурсов ПК при воспроизведении анимации.

2.3 Подключение Гауссовского и радиального фильтров размытия с элементами рендеринга сцены

Чтобы хоть как-то иметь схожесть с аквариумом на Openbase нужно добавить фильтры размытия. Гауссовское размытие является уже встроенным в библиотеку. Эффект радиального размытия также отображен в виде шейдера (См. Листинг 2.2).

```
      <!-- шейдер радиального размытия -->
<script id="radialBlurShader" type="shader">
precision mediump float;
ёvarying vec2 vTextureCoord;
varying vec4 vColor;
const int Quality = 16;
uniform sampler2D uSampler; //текстура
void main()
                {
vec4 Color;
float v;
for (float i = 0.0; i < 1.0; i += 1.0 / float(Quality)) {
v = 0.9 + i * 0.1; // конвертирует "i" в диапазон [0.9] .. [1.0]
Color += texture2D( uSampler, vTextureCoord*v+0.5-0.5*v);
                      }
Color /= float(Quality);
gl_FragColor =  Color *  vColor;
                }
          </script>
```

Листинг 2.2 Шейдер радиального размытия

Гауссовское размытие переведено на активную кнопку веб-страницы с помощью ссылки на библиотеку. Проверим работоспособность Гауссовского и радиального размытия на холсте (Рис. 2.4 -2.6).

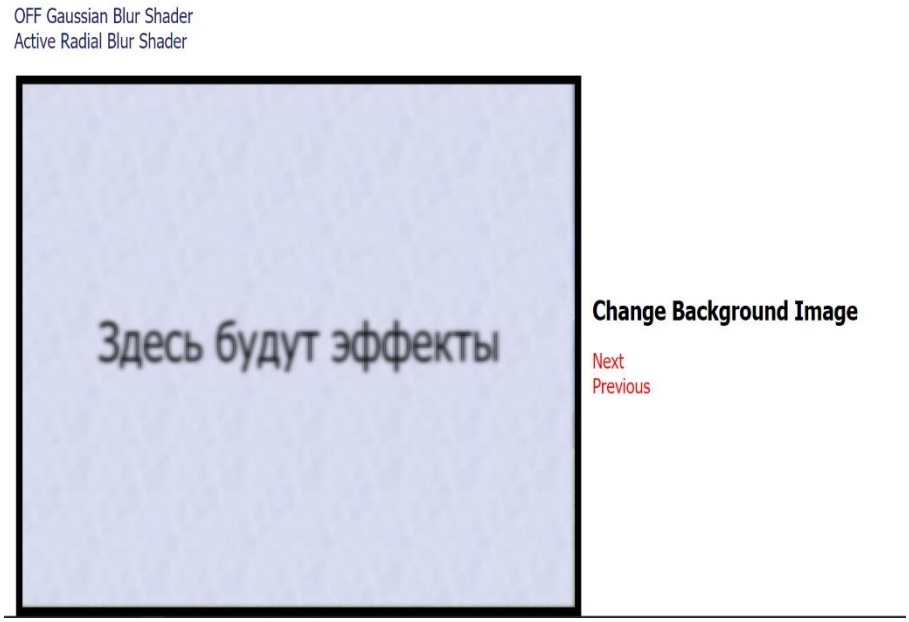

Рис 2.4. Гауссовское размытие

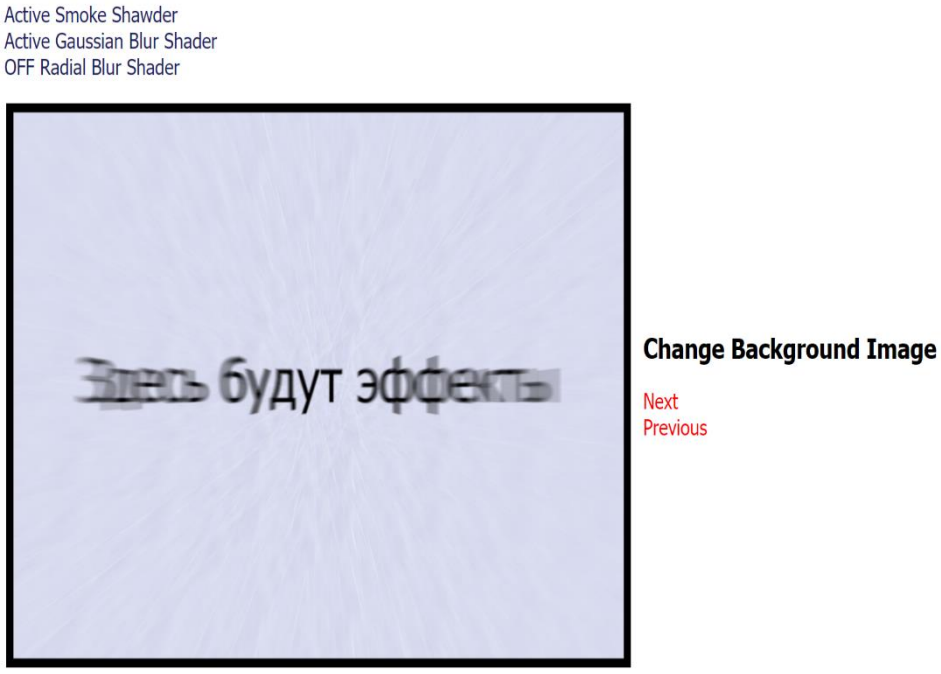

Рис 2.5. Радиальное размытие

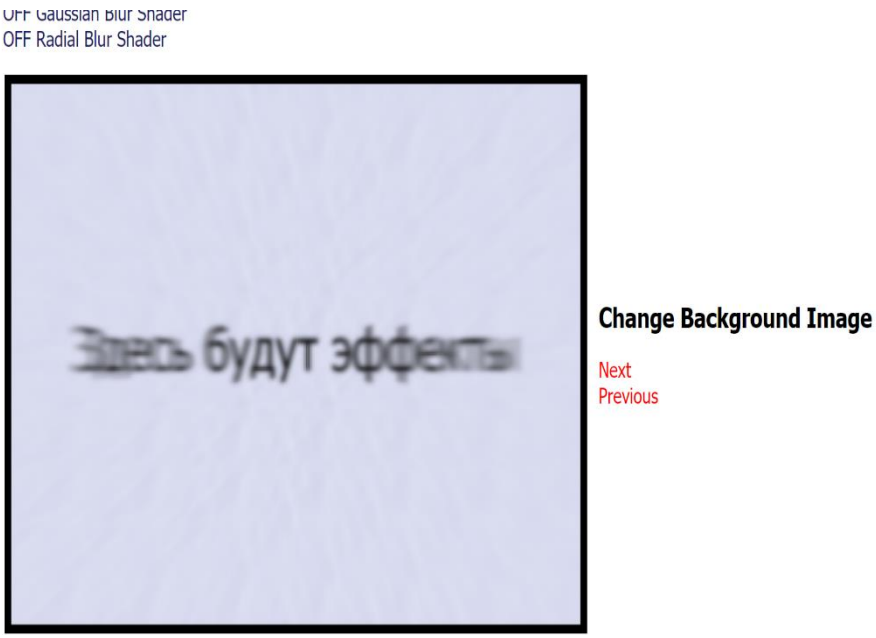

Рис 2.6. Оба размытие одновременно

Далее выставляется параметры холста, ширина и высота. Затем получаем, доступный браузеру, рендеринг и выводим на веб-страницу(См. Листинг 2.3).

```
 var width = 870; //window.innerWidth;
var height = 620; //window.innerHeight;
var renderer = new PIXI.autoDetectRenderer(width, height);
//document.body.appendChild(renderer.view);
document.getElementById("rendererHolder").appendChild(renderer
.view)
```

Листинг. 2.3. Рендеринг веб-страницы

После того как все элементы сцены собраны в главный контейнер, передаем глобальные переменные шейдера в рендер под видеокарту и включаем эффекты размытия в виде фильтров библиотеки. Затем подгружаем текстуры, в роли которых выступает набор меняющихся фоновых картинок и задаем их расположение (См. Листинг.2.4).

```
var stage = new PIXI.Container();
var                    smokeShaderCode                    =
document.getElementById("smokeShader").innerHTML;
var                  radialBlurShaderCode                 =
document.getElementById("radialBlurShader").innerHTML;
var smokeUniforms = {};
smokeUniforms.resolution = { type: 'v2', value: { x: width, y:
height}};
```

```
smokeUniforms.alpha = { type: '1f', value: 1.0};
smokeUniforms.shift = { type: '1f', value: 1.6};
smokeUniforms.time = {type: '1f',value: 0};
smokeUniforms.speed = {type: 'v2', value: {x: 0.7, y: 0.4}};
var smokeShader = new PIXI.AbstractFilter('', smokeShaderCode,
smokeUniforms);
var    radialBlurShader    =    new    PIXI.AbstractFilter('',
radialBlurShaderCode);
var gausBlurShader = new PIXI.filters.BlurFilter(8, 1);
var bgImages = [
                    "img/tex.png",
                    "img/tex2.png",
                    "img/tex3.png",
                    "img/tex4.png",
                    "img/tex5.png",
                    "img/tex6.png",
                    "img/tex7.png"
                ]
// загружаем текстуру
var mainTexture;
var curTextureIndex = 0; // текущий индекс текстуры из массива

// добавляем текстуру на сцену
setBackgroundImage(bgImages[0]);

// устанавливаем её позицию в центр холста
mainTexture.x = width / 2;
mainTexture.y = height / 2;

// задаём точку крепления текстуры к её центру, изначально
берется левый верхний угол
mainTexture.anchor.set(0.5);
stage.addChild(mainTexture);
                }

function swapBackgroundImageLeft() {
if (curTextureIndex == 0) {
return;
}else {
curTextureIndex -= 1;
setBackgroundImage(bgImages[curTextureIndex]);
                    }
                }

function swapBackgroundImageRight() {
if (curTextureIndex == bgImages.length - 1) {
return;
} else {
```

```
curTextureIndex += 1;
setBackgroundImage(bgImages[curTextureIndex]);
                    }
                }
```

Листинг 2.4 Работа с текстурами

Следующим этапом будет выставление шейдеров, в зависимости включения или выключения опций активных кнопок, и самой функциональности кнопок (См. Листинг 2.5).

```
// добавляем шейдеры в зависимости от включенных опций
if (isSmokeShaderEnabled) activeFilters.push(smokeShader);
if (isGausBlurEnabled) activeFilters.push(gausBlurShader);
if (isRadialBlurEnabled) activeFilters.push(radialBlurShader);

// увеличиваем значение времени каждый тик на 0.01
smokeShader.uniforms.time.value += 0.01;
// если у нас есть активные фильтры, накладываем их на текстуру
if (activeFilters.length > 0) mainTexture.filters =
activeFilters;

// запускаем анимацию
requestAnimationFrame(animate);
// рендерим нашу сцену
renderer.render(stage);
}

// функции снизу меняют опции отображения шейдеров
// включить/выключить гауссово размытие
function switchGausBlur() {
isGausBlurEnabled = !isGausBlurEnabled;
let                    connectedA                    =
document.getElementById('blur_shader_switcher');
if (isGausBlurEnabled) {
connectedA.innerHTML = 'OFF Gaussian Blur Shader';
} else {
connectedA.innerHTML = 'Active Gaussian Blur Shader';
                    }
                }
// включить/выключить радиальное размытие
function switchRadialBlur() {
isRadialBlurEnabled = !isRadialBlurEnabled;
let                    connectedA                    =
document.getElementById('radial_blur_shader_switcher');
if (isRadialBlurEnabled) {
connectedA.innerHTML = 'OFF Radial Blur Shader';
} else {
```

```
connectedA.innerHTML = 'Active Radial Blur Shader';
                    }
                }
// включить/выключить шейдер дыма
function switchSmoke() {
isSmokeShaderEnabled = !isSmokeShaderEnabled;
let                         connectedA                          =
document.getElementById('smoke_shader_switcher');
f (isSmokeShaderEnabled) {
connectedA.innerHTML = 'OFF Smoke Shader';
} else {
connectedA.innerHTML = 'Active Smoke Shader';
                    }
                }               }
```

Листинг 2.5 Функциональность и работа активных кнопок

Теперь посмотрим на работу финальной версии анимационного примера. Эффекты фильтров размытия могут работать могут по отдельности или одновременно интегрироваться с текстурой и динамической средой, при этом не используя слишком большое количество ресурсов компьютера(Рис. 2.7-2.9).

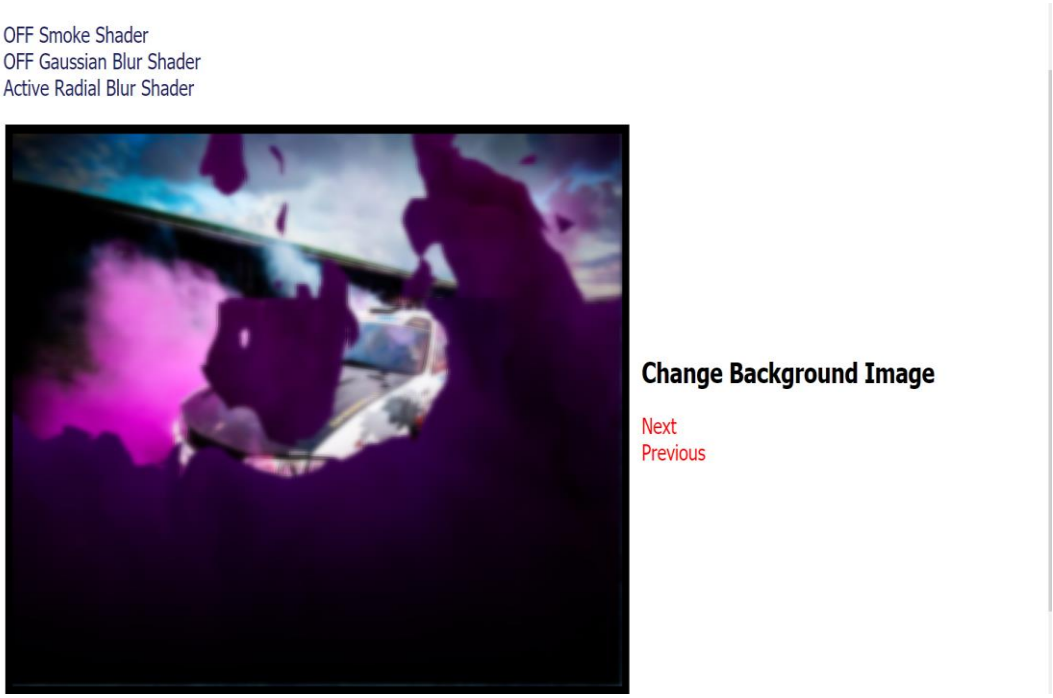

Рис 2.7. Гауссовское размытие во взаимодействии с динамической средой

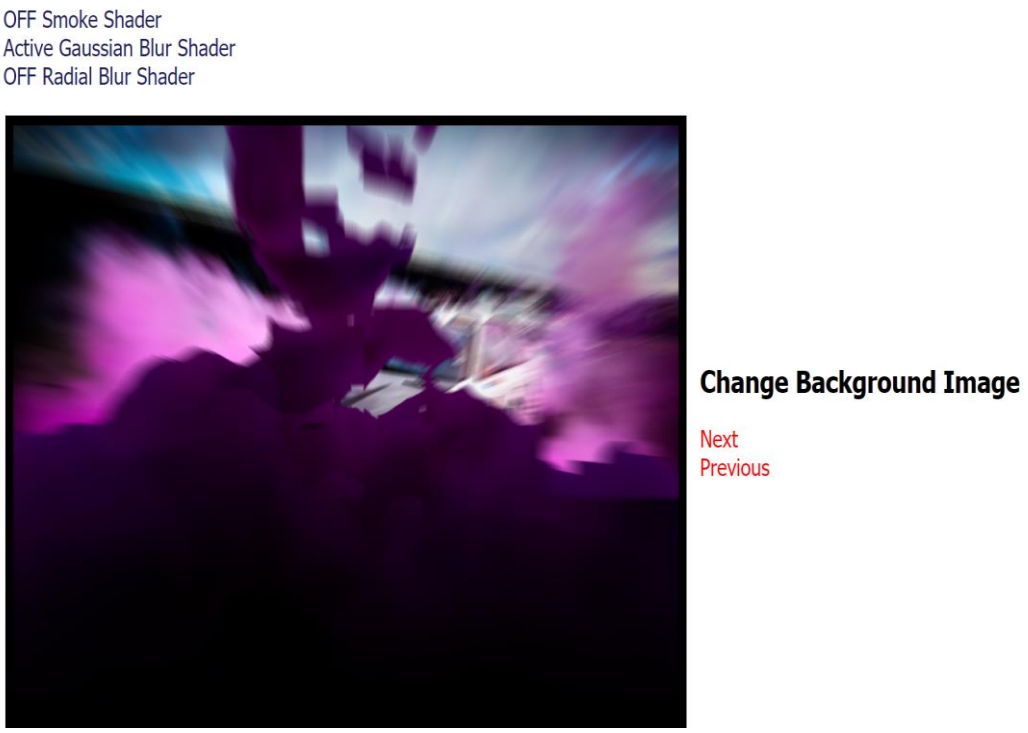

Рис 2.8. Радиальное размытие во взаимодействии с динамической средой

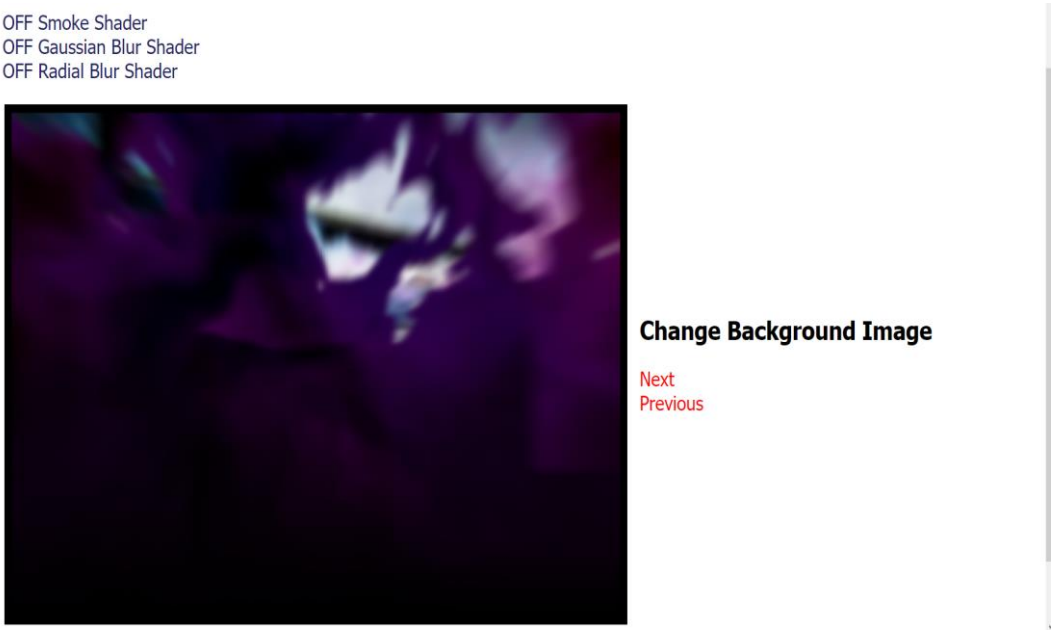

Рис 2.9. Гауссовское размытие во взаимодействии с динамической средой

Практический пример выпускной квалификационной работы готов и полностью работоспособен и интерактивен. Эффекты размытия и динамическая среда деактивируются по нажатию активных кнопок на данной веб-странице. Функциональные клавиши меняют свое состояние в зависимости от режима работы шейдеров. Проект может использоваться в качестве обучающего примера или как часть дизайна страницы в сети Интернет.

Заключение

В работе рассмотрен способ построения динамической среды при помощи использования bitmap эффектов. Изложены основные понятия языка JavaScript и его библиотеки Pixi.js, технологии WebGL, а также эффектов Гауссовского и радиального размытия. Описаны принципы работы, преимущества и недостатки используемых основных и вспомогательных средств разработки. Приведены иллюстрирующие наглядные примеры графических возможностей выбранных инструментов.

Для построения динамической среды взаимодействующей с bitmap эффектами требуется изучить основы и графический инструментарий скриптового языка программирования JavaScript и что собой представляет его библиотека Pixi.js и как работают Вершинные и Фрагментные шейдеры, задействующие технологию WebGL для рисования объектов за счёт мощностей графического процессора видеокарты. Сравнить и изучить как работают и изменяют изображение эффекты фильтров Гауссовского и радиального размытия.

Разработан технический и наглядный пример демонстрирующий работу графических технологий веб – программирования при построении динамической среды, влияющей на изображения за счет визуализации физического явления и наложения эффектов замыливания мелких частей самого изображения.

Реферативная часть работы и самостоятельно разработанный интерактивный пример предназначен для использования в учебном процессе при изучении темы «Графическое построение динамической среды».

Список использованных источников

1.WebGL: Программирование трехмерной графики / К. Мацуда [и др.]/ Пер. с англ. Киселев А.Н. – М.: ДМК Пресс, 2015 – 494 с.: ил. . – Текст : электронный // ЭБС «ЛитРес: Библиотека» – URL: https://www.litres.ru/koichi-macuda/webgl-programmirovanie-trehmernoy-grafiki-10002663/ (Дата обращения 03.01.2021) – Режим доступа: по подписке.

2. Бесплатный онлайн курс по WebGL и Three.js // Academiait.ru – URL: https://academiait.ru/course/webgl-i-three-js-videokurs/ (Дата обращения 07.02.2021)) – Режим доступа: бесплатно.

3. Курс лекций по JavaScript // Intuit.ru – URL: https://www.intuit.ru/studies/courses/1978/465/lecture/20681/ (Дата обращения 13.03.2021)) – Режим доступа: пробная версия.

4. Изучаем программирование на JavaScript / Фримен Э.[и др.] / Пер. с англ. ООО Издательство «Питер» 2015 – 640 с.: ил –Текст : электронный // ЭБС «ЛитРес: Библиотека» – URL: https://www.litres.ru/elizabet-robson/izuchaem-programmirovanie-na-javascript-9523650/ (Дата обращения 09.01.2021) – Режим доступа: по подписке.

5. Как устроен JavaScript / Крокфорд Д. / Пер. с англ. Н. Вильчинский – Питер 2019 – 304 с. –Текст : электронный // ЭБС «ЛитРес: Библиотека» – URL: https://www.litres.ru/duglas-krokford/kak-ustroen-javascript-48613309/ (Дата обращения 09.12.2020) – Режим доступа: по подписке.

6. Официальная документация библиотеки Pixi.js. – Текст : электронный // Официальный сайт разработчиков библиотеки Pixi.js – URL: https://www.pixijs.com/tutorials (Дата обращения 15.02.2021) - – Режим доступа: в открытом доступе.