Заключение. Создание игр – это сложный и многогранный процесс, требующий координации усилий множества специалистов. Каждый этап важен и влияет на конечный результат.

Успех игры зависит не только от хорошей идеи, но и от тщательной реализации всех аспектов: дизайна, программирования, графики, звука и тестирования. Благодаря слаженной работе команды и вниманию к деталям, на свет появляются шедевры, которые захватывают сердца миллионов игроков по всему миру.

Список использованных источников:

1. Казакова Н.Ю. Этап тестирования игрового проекта в гейм-дизайне как виде проектной деятельности / Н.Ю. Казакова // Академическая наука-проблемы и достижения. – 2016. – С. 6–10.
2. Жукова М.С. Особенности создания десктопных 2D игр в жанре test / М.С. Жукова // Наука россии 3. – 2025. – С. 17.
3. Кадыров П.Р. Оптимизация процесса создания игровых ассетов при помощи использования нейронных сетей / П.Р. Кадыров // Вестник науки. – 2024. – Т. 5. – № 12 (81). – С. 688–697.

ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ WEBGL-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ МЕТОДОВ ЛИНЕЙНОЙ АЛГЕБРЫ

И.С. Потапов[a], студент гр. 17В41,
Научные руководители: Воробьев А.В., к.т.н., доц., Гиль Л.Б., к.пед.н., доц.
Юргинский технологический институт (филиал)
Национального исследовательского Томского политехнического университета
652055, Кемеровская обл., г. Юрга, ул. Ленинградская, 26
E-mail: [a]daisy40@inbox.ru

Аннотация: В данной статье рассматриваются методы применения линейной алгебры для улучшения WebGL-приложений. К рассматриваемым методам относятся: снижение количества матричных операций, совершенствование геометрических преобразований, а также использование разреженных матриц в 3D-моделировании.

Ключевые слова: WebGL, линейная алгебра, матрицы, разреженные матрицы, геометрические преобразования, 3D-моделирование.

Abstract: This article explores methods for applying linear algebra to enhance WebGL applications. The methods considered include the number of matrix operations, improving geometric transformations, and utilizing sparse matrices in 3D modeling.

Keywords: WebGL, linear algebra, matrices, sparse matrices, geometric transformations, 3D modeling.

WebGL – API для языка программирования JavaScript, оно предназначено для отображения 2D и 3D графики[1]. Его популярность объясняется универсальной работой на любой платформе и возможностью создания трёхмерной графики прямо в браузере без использования дополнительных расширений [2]. 3D-графика сама по себе является очень трудоёмкой задачей, которая требует сложных вычислений, что не каждое устройство способно осилить.

Большинство операций WebGL основываются на применении принципов линейной алгебры. Как и в языках программирования, в математике в решении задач можно пользоваться разными методами, одни методы могут быть рациональнее других. Это можно отнести и к сфере 3D-моделирования в браузере, в которой от выбранного подхода зависит количество вычислений, требуемых устройству для отображения графики.

Вычисления можно свести к минимуму, грамотно выбирая подходы в линейной алгебре и комбинируя это с WebGL.

К основным понятиям линейной алгебры, применяемым в WebGL, относятся матрицы, векторы и преобразования. Назначение матриц – это изменение масштаба, вращение и проекция объекта.

Конвейер рендеринга WebGL использует матрицы в шейдерах вершин, чтобы превратить координаты вершин из пространства объекта в пространство отсечения.

Все вершины умножаются на модально-видовую матрицу и проекционную, таким образом от этих матричных операций прямо зависит производительность рендеринга.

Рассматривая матричные операции, их можно оптимизировать несколькими способами, в том числе допускается пользоваться сразу несколькими методами одновременно.

Первый метод – это снижение количества умножений матриц. Различные сложные преобразования можно заменить на более простые.

Предварительно вычислив и объединив несколько матриц в одну, можно сократить количество умножений матриц на каждый кадр.

Второй метод подразумевает использование библиотек линейной алгебры с готовыми, улучшенными идеями для быстрых вычислений. К ним можно отнести библиотеку glMatix, данная библиотека имеет встроенные оптимизированные функции не только для матриц, но и для векторов. Используя glMatrix, можно избежать многих типичных ошибок, а также повысить производительность.

Следующий метод – использование Single-Precision Floating-Point. Говоря простым языком, компьютеры хранят числа с разной точностью. Double (Double-Precision Floating-Point) означает очень точную линейку, с помощью которой можно измерить очень маленькие доли миллиметра. Float (Single-Precision Floating-Point) – более простая линейка с менее точным измерением. Для Float требуется меньше памяти и вычислений в сравнении с Double.

Рассмотрим способ разложения матриц. При использовании матрицы преобразования её можно сегментировать на более простые матрицы. К примеру, матрицу вращения можно представить в виде операции произведения самой матрицы вращения вокруг осей X, Y и Z. Способ полезен при вычислении обратных матриц.

Рассматривая применение оптимизации, можно проиллюстрировать это на примерах, представленных на рисунке 1. Чем плох первый способ? В нём все преобразования применяются внутри одной функции, последовательно. На языке математики это значит, что мы умножаем матрицы в заданном коде порядке, сначала S (матрица масштабирования) умножаем на R (матрица вращения), и дальше R умножаем на T (матрица трансляции), в результате получаем итоговую матрицу модели M. Проблема заключается в том, что каждый раз при вызове draw Cube Bad данная последовательность умножений происходит заново, даже если S и T остаются неизменными.

Что касается второго примера, статические преобразования, такие как изменение масштаба и трансляция, созданы заранее в static Model Matrix. На языке математики это значит, что математическая модель в виде произведения S и T вычисляется только один раз. Дальше, в каждом последующем вызове функции draw Cube Good, переменная произведения умножается на матрицу вращения. Таким образом, мы отделили статические элементы от динамических, чтобы каждый кадр обновлялось то, что действительно изменяется.

```
// ---------------------- КАК НЕ НАДО ----------------------
function drawCubeBad(t, r, s) {
   const m = mat4.create();
   mat4.translate(m, m, t);
   mat4.rotate(m, m, r, [0, 1, 0]);
   mat4.scale(m, m, s);
   gl.uniformMatrix4fv(gl.getUniformLocation(gl.program, "uModelMatrix"), false, m);
   gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
   gl.drawArrays(gl.TRIANGLES, 0, cubeVertexCount);
}

// ---------------------- КАК НАДО ----------------------
let staticModelMatrix = mat4.create();
let translationVector = [1, 0, -5];
let scaleVector       = [0.5, 0.5, 0.5];
mat4.translate(staticModelMatrix, staticModelMatrix, translationVector);
mat4.scale(staticModelMatrix, staticModelMatrix, scaleVector);
const uModelMatrixLocation = gl.getUniformLocation(gl.program, "uModelMatrix");

function drawCubeGood(r) {
   const rotationMatrix = mat4.create();
   mat4.rotate(rotationMatrix, rotationMatrix, r, [0, 1, 0]);
   const finalModelMatrix = mat4.create();
   mat4.multiply(finalModelMatrix, staticModelMatrix, rotationMatrix);
   gl.uniformMatrix4fv(uModelMatrixLocation, false, finalModelMatrix);
   gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
   gl.drawArrays(gl.TRIANGLES, 0, cubeVertexCount);
}
```

Рис. 1. Пример плохого и хорошего кода

Резюмировать пример с плохой и хорошей матрицами можно тем, что draw Cube Good лучше, так как он использует свойства ассоциативности умножения матриц, статические преобразования композируются предварительно, соответственно выполняется меньшее количество операций.

Эта оптимизация основывается на понимании того, какие части преобразования должны быть неизменными, и использовании этого для уменьшения вычислений устройством.

Ещё один практический пример (см. рис. 2) – то, как библиотеки могут упростить работу с матрицами. Пример «Без библиотеки» наглядно демонстрирует, что объявить матрицу-то мы объявили, но для дальнейших операций нужно писать свой код. Для этого потребуется знать о том, как эффективно производить те или иные операции, что может сказаться на долгом анализе выбираемого метода, но и самостоятельно реализовать алгоритмы, которые подвержены ошибкам и требующие оптимизации. В то время как библиотека предоставляет возможность сфокусироваться на логике приложения, а не низкоуровневых деталях реализации операций с матрицами. В библиотеке нам предоставляются такие возможности, как: создание матрицы, преобразование, преобразование векторов, для которых используются готовые функции.

```
// ------------ БЕЗ БИБЛИОТЕКИ ------------
// Ручное создание матрицы
let matrix = [
   2, 0, 0,
   0, 1, 0,
   0, 5, 1
];
// (Для дальнейших операций нужно писать свой код...)
console.log(matrix);

// ------------ С БИБЛИОТЕКОЙ ------------
import { mat3, vec2 } from 'gl-matrix';

// Создание матрицы
let matrix = mat3.create();

// Инициализация матрицы масштабирования и сдвига
mat3.fromTranslation(matrix, vec2.fromValues(0, 5)); // Сначала сдвиг
mat3.scale(matrix, matrix, vec2.fromValues(2, 1));   // Затем масштабирование

// Пример применения матрицы к вектору
let point = vec2.fromValues(1, 2);
let transformedPoint = vec2.create();
vec2.transformMat3(transformedPoint, point, matrix);
```

Рис. 2. Пример без использования библиотеки и с библиотекой

Подводя итоги, использование методов линейной алгебры в программировании способно значительно повысить производительность WebGL-приложений, а также повысить их эффективность.

Список использованных источников:

1. Осин А.В. Обзор методов идентификации пользователя на основе цифровых отпечатков / А.В. Осин, Ю.В. Мурашко // Труды учебных заведений связи. – 2023. – № 5. – URL: https://cyberleninka.ru/article/n/obzor-metodov-identifikatsii-polzovatelya-na-osnove-tsifrovyh-otpechatkov (дата обращения: 11.02.2025).

2. Любахинец А.А. Интеграция реалистичных ии-аватаров в веб-сайты для улучшения пользовательского взаимодействия / А.А. Любахинец // Universum: технические науки. – 2024. – № 12 (129). – URL: https://cyberleninka.ru/article/n/integratsiya-realistichnyh-ii-avatarov-v-veb-sayty-dlya-uluchsheniya-polzovatelskogo-vzaimodeystviya (дата обращения: 11.02.2025).