Лабораторная работа №5

«Работа с цветом и фрагментные шейдеры»

Оглавление

Изменение цвета (varying-переменные)

Рассмотрим случай, когда цвет каждой точки на сцене может отличаться от остальных. Пусть, например, требуется нарисовать три точки – красного, синего и зеленого цвета. Для этого, добавим в массив данных о вершинах еще и цвета точек. После сохранения цвета в буферном объекте, можно присвоить значение цвета специально созданной переменной-атрибуту в вершинном шейдере.

Мы уже знаем, что атрибутами, такими как цвет, управляет фрагментный шейдер. Поэтому передав цвет точки в вершинный шейдер через переменную-атрибут, мы не сможем сразу присвоить значение цвета переменной `gl_FragColor`, потому что эта переменная доступна только в фрагментном шейдере. Соответственно, нам нужно найти какой-то способ взаимодействий с фрагментным шейдером, чтобы передать ему информацию о цвете, предварительно переданную в вершинный шейдер.

Ранее для передачи информации о цвете во фрагментный шейдер использовалась `uniform`-переменная; однако, из-за того, что значение переменной является «общим для всех» (то есть, не изменяется), ее нельзя использовать для передачи разных цветов. Здесь необходимо использовать новый для нас способ передачи данных во фрагментный шейдер — посредством `varying`-переменной, представляющей механизм передачи между вершинным и фрагментным шейдерами.

В WebGL, когда `varying`-переменные, объявленные во фрагментном шейдере, своими именами и типами совпадают с переменными, объявленными в вершинном шейдере, значения, присваиваемые в вершинном шейдере автоматически передаются во фрагментный шейдер.

Поэтому объявим новую `varying`-переменную в вершинном шейдере, которая будет использоваться для передачи цвета фрагментному шейдеру. В объявлениях `varying`-переменных можно использовать только вещественные типы (и связанные с ними типы `vec2`, `vec3`, `vec4`,

`mat2`, `mat3` и `mat4`). В данном случае требуется выбрать тот тип, который соответствует типу данных о цвете. Присвоим ей значение переменной-атрибута, в которой загружается информация о цвете из буферного объекта.

Для получения присвоенных данных во фрагментном шейдере нужно в нем объявить ту же самую переменную с тем же именем и того же типа, как в вершинном шейдере.

Задание для самостоятельной работы №1

Нужно выполнить только **один** вариант, соответствующий вашему номеру в журнале. Если ваш номер в журнале превышает максимальный вариант, то вариант задания определяется как остаток от деления первого значения на второе, к которому прибавляется единица. Полученное значение также увеличивается на единицу. Например, если ваш номер в журнале равен 15, а максимальный вариант равен 12, то вариант задания определяется по формуле: 15%13 + 1 = 3.

Вариант 1. Продемонстрируйте работу программы в случае, когда в **одном** массиве данных вершин значения хранятся в следующем порядке: координаты точки, размер точки, цвет точки.

Вариант 2. Продемонстрируйте работу программы в случае, когда в **одном** массиве данных вершин значения хранятся в следующем порядке: координаты точки, цвет точки, размер точки.

Вариант 3. Продемонстрируйте работу программы в случае, когда в **одном** массиве данных вершин значения хранятся в следующем порядке: размер точки, координаты точки, цвет точки.

Вариант 4. Продемонстрируйте работу программы в случае, когда в **одном** массиве данных вершин значения хранятся в следующем порядке: размер точки, цвет точки, координаты точки.

Вариант 5. Продемонстрируйте работу программы в случае, когда в **одном** массиве данных вершин значения хранятся в следующем порядке: цвет точки, координаты точки, размер точки.

Вариант 6. Продемонстрируйте работу программы в случае, когда в **одном** массиве данных вершин значения хранятся в следующем порядке: цвет точки, размер точки, координаты точки.

Вариант 7. Продемонстрируйте работу программы в случае, когда заданы три массива данных вершин: массив координат точек, массив размеров точек, массив цветов точек. Запишите их в один буфер в следующем порядке: массив координат точек, массив размеров точек, массив цветов точек.

Вариант 8. Продемонстрируйте работу программы в случае, когда заданы три массива данных вершин: массив координат точек, массив размеров точек, массив цветов точек. Запишите их в один буфер в следующем порядке: массив координат точек, массив цветов точек, массив размеров точек.

Вариант 9. Продемонстрируйте работу программы в случае, когда заданы три массива данных вершин: массив координат точек, массив размеров точек, массив цветов точек. Запишите их в один буфер в следующем порядке: массив размеров точек, массив координат точек, массив цветов точек.

Вариант 10. Продемонстрируйте работу программы в случае, когда заданы три массива данных вершин: массив координат точек, массив размеров точек, массив цветов точек. Запишите их в один буфер в следующем порядке: массив размеров точек, массив цветов точек, массив координат точек.

Вариант 11. Продемонстрируйте работу программы в случае, когда заданы три массива данных вершин: массив цветов точек, массив координат точек, массив размеров точек. Запишите их в один буфер в следующем порядке: массив цветов точек, массив координат точек, массив размеров точек.

Вариант 12. Продемонстрируйте работу программы в случае, когда заданы три массива данных вершин: массив координат точек, массив размеров точек, массив цветов точек. Запишите их в один буфер в следующем порядке: массив цветов точек, массив размеров точек, массив координат точек.

Эксперименты с примером программы

Давайте изменим первый аргумент в вызове `gl.drawArrays()` на `gl.TRIANGLES` и посмотрим, что из этого получится.

Изменение значения всего одного параметра произвело потрясающий эффект — три цветные точки превратились в градиентный треугольник с плавными переходами цветов между красным, зеленым и синим в углах. Давайте разберемся, как это получилось.

Между вершинным и фрагментным шейдерами выполняются две операции:

1) **операция сборки геометрической фигуры:** на этом этапе из указанных координат вершин выполняется сборка геометрической фигуры; тип фигуры определяет первый аргумент метода `gl.drawArrays();`
2) **операция растеризации:** на этом этапе собранная геометрическая фигура преобразуется в массив фрагментов, составляющих эту фигуру.

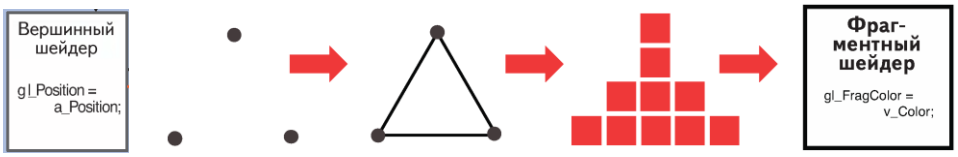

Рис. 1. Координаты вершин, идентификация треугольника по координатам вершин, растеризация и выполнение фрагментного шейдера

Хотя на этом рисунке показано всего 10 фрагментов, фактическое их число определяется площадью отображаемого треугольника.

По завершении этапа растеризации, для обработки каждого созданного фрагмента вызывается фрагментный шейдер. Все фрагменты передаются фрагментному шейдеру по одному, друг за другом, и для каждого фрагмента шейдер устанавливает цвет и записывает его в буфер цвета. Значение, присвоенное `varying`-переменной в вершинном шейдере, интерполируется на этапе растеризации соответственно позициям фрагментов. Соответственно, значения, фактически передаваемые фрагментному шейдеру, отличаются для разных фрагментов. Это объясняет суть спецификатора `varying` (изменчивый, меняющийся).

Когда фрагментный шейдер завершит обработку последнего фрагмента, в окне браузера появится получившееся изображение.

Задание для самостоятельной работы №2

Визуализируйте градиентный треугольник с плавными переходами цветов между красным, зеленым и синим в углах.

Давайте убедимся, что фрагментный шейдер действительно вызывается для каждого фрагмента и устанавливает его цвет, исходя из местоположения этого фрагмента. Каждый фрагмент, сгенерированный на этапе растеризации, имеет определенные координаты, передаваемые фрагментному шейдеру при вызове. Эти координаты доступны через встроенную переменную:

```
vec4 gl_FragCoord
```

Первый и второй элементы массива — координаты фрагмента в системе координат, связанной с элементом `<canvas>` (система координат окна), где начало координат находится в левом нижнем углу, ось y направлена вверх.

Задание для самостоятельной работы №3 (необязательное)

Чтобы убедиться, что фрагментный шейдер действительно вызывается для каждого фрагмента, измените реализацию фрагментного шейдера в программе, как показано ниже:

```
gl_FragColor = vec4(gl_FragCoord.x/u_Width, 0.0,
                    gl_FragCoord.y/u_Height, 1.0);
```

Здесь цвет каждого фрагмента вычисляется исходя из координат этого фрагмента в элементе `<canvas>`. Значение составляющей цвета в WebGL может изменяться в диапазоне от 0.0 до 1.0, поэтому чтобы получить значение цвета, нужно разделить координату на соответствующий ей размер элемента `<canvas>` (в нашем случае, на 400 пикселей). Для получения этих значений рекомендуется использовать свойства `gl.drawingBufferWidth` и `gl.drawingBufferHeight`.

С помощью данного фрагментного шейдера нарисуйте прямоугольник, заполняющий все окно. Где находятся максимальные и минимальные значения координат `gl_FragCoord.x` и `gl_FragCoord.y`?

Создание круглой точки

До сих пор мы рисовали точку не «круглой», а «квадратной», что намного проще. Чтобы нарисовать «круглую» точку, требуется скруглить углы у «квадратной» точки. Эту операцию можно выполнить на этапе растеризации, который выполняется между вызовами вершинного и фрагментного шейдеров. На этапе растеризации генерируется прямоугольник, состоящий из множества фрагментов, каждый из которых передается фрагментному шейдеру. Если нарисовать эти фрагменты «как есть», получится прямоугольник. Поэтому нам нужно всего лишь изменить фрагментный шейдер, который будет рисовать только фрагменты, попадающие внутрь окружности, как показано на рис. 2.

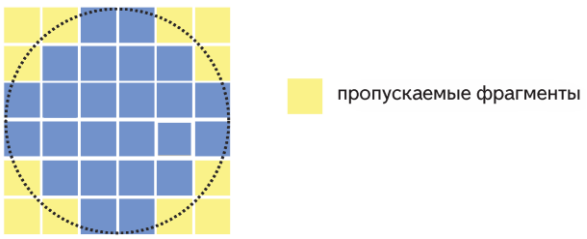

Рис. 2. Пропуск фрагментов с целью преобразовать прямоугольник в круг

Для этого требуется знать позиции всех фрагментов, созданных на этапе растеризации для примитива «точка». Положение фрагмента в пределах нарисованной точки можно определить с помощью переменной `gl_PointCoord` типа `vec2`. Переменная `gl_PointCoord` хранит

координаты каждого фрагмента в диапазоне от (0.0, 0.0) до (1.0, 1.0), как показано на рис. 3. Чтобы закруглить углы, нужно просто пропустить фрагменты, выходящие за окружность с центром в точке (0.5, 0.5) и с радиусом 0.5. Выполнить пропуск фрагментов можно с помощью инструкции `discard`.

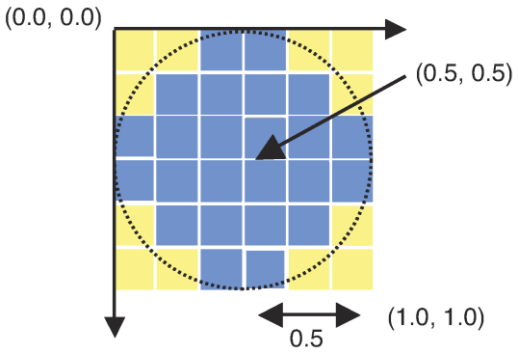

Рис. 3. Координаты переменной `gl_PointCoord`

Соответственно, чтобы закруглить квадратную точку, нужно выполнить следующее:

1. Вычислить расстояние от центра (0.5, 0.5) до каждого фрагмента.
2. Нарисовать только те фрагменты, расстояние до которых меньше 0.5.

Для вычисления расстояний в языке шейдеров реализована специальная функция `distance(point1, point2)`, принимающая два векторных аргумента.

Задание для самостоятельной работы №4 (необязательное)

Модифицируйте программу, чтобы она рисовала круглые точки.

Альфа-смешивание

Значение альфа-канала управляет прозрачностью рисуемых объектов. Если установить значение альфа-канала равным 0.5, объект станет полупрозрачным и через него станут частично видимы другие объекты. По мере приближения значения альфа-канала к 0.0, находящиеся позади объекты будут видимы все отчетливее.

Как реализовать альфа-смешивание

Чтобы активировать функцию альфа-смешивания и включить ее в работу, необходимо выполнить следующие два шага.

1. Активировать функцию альфа-смешивания:

```
gl.enable(gl.BLEND);
```

2. Указать, как должна действовать функция смешивания:

```
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
```

Как должна действовать функция смешивания

В смешивании участвуют два цвета: цвет, который будет подмешиваться (исходный цвет) и цвет, в который будет осуществляться подмешивание (целевой цвет). Когда рисуется одна фигура поверх другой, цвет уже нарисованной фигуры является целевым цветом, а цвет фигуры, рисуемой поверх, является исходным цветом.

`gl.blendFunc(src_factor, dst_factor)`
Определяет, как должно выполняться смешивание исходного цвета с целевым.

Результирующий цвет вычисляется по следующей формуле:

цвет(RGB) = <исходный цвет> × src_factor + <целевой цвет> × dst_factor

Параметры:	
src_factor	Определяет коэффициент для исходного цвета (см. табл. 1).
dst_factor	Определяет коэффициент для целевого цвета (см. табл. 1).

Возвращаемое значение: нет

Табл. 1. Возможные значения параметров src_factor и dst_factor

Константа	Множитель для R	Множитель для G	Множитель для B
gl.SRC_COLOR	Rs	Gs	Bs
gl.ONE_MINUS_SRC_COLOR	(1 – Rs)	(1 – Gs)	(1 – Bs)
gl.DST_COLOR	Rd	Gd	Bd
gl.ONE_MINUS_DST_COLOR	(1 – Rd)	(1 – Gd)	(1 – Bd)
gl.SRC_ALPHA	As	As	As
gl.ONE_MINUS_SRC_ALPHA	(1 – As)	(1 – As)	(1 – As)
gl.DST_ALPHA	Ad	Ad	Ad
gl.ONE_MINUS_DST_ALPHA	(1 – Ad)	(1 – Ad)	(1 – Ad)
gl.SRC_ALPHA_SATURATE	min(As, Ad)	min(As, Ad)	min(As, Ad)

Здесь (Rs, Gs, Bs, As) — это исходный цвет, а (Rd, Gd, Bd, Ad) — целевой цвет.

Например, если исходный цвет — полупрозрачный зеленый (0.0, 1.0, 0.0, 0.4) и целевой цвет — желтый (1.0, 1.0, 0.0, 1.0), параметр src_factor получит значение альфа-канала 0.4, а параметр dst_factor получит значение (1 - 0.4) = 0.6. Вычисления показаны на рис. 5:

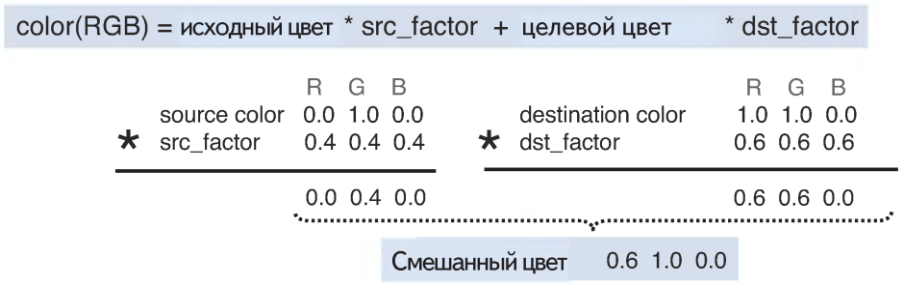

Рис. 4. Вычисления для gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA)

На практике также часто используется кумулятивное смешивание. Результат такого смешивания получается более ярким, чем оригинал. Этот прием можно использовать, например, для достижения эффекта подсветки от взрыва. Реализовать его можно с помощью, например, такой настройки функции смешивания:

```
gl.blendFunc(gl.ONE, gl.SRC_ALPHA);
```

Задание для самостоятельной работы №5 (необязательное)

1. Создать программу, которая рисует 2 накладывающихся друг на друга треугольника. Для рисования нужно использовать примитив `gl.TRIANGLES` и 6 точек — вершин. Для каждой тройки вершин укажите свой цвет и значение 0.4 для альфа-канала.
2. Поэкспериментируйте с другими допустимыми значениями параметров `src_factor` и `dst_factor`
3. Наложите полностью 2 треугольника друг на друга. Используйте кумулятивное смешивание для достижения эффекта подсветки от взрыва.