Лабораторная работа №3

«Рисование точки по щелчку мышью»

Оглавление

Введение

Программа из лабораторной работы №2 способна передавать координаты точки в вершинный шейдер из программного кода на JavaScript. Однако координаты все еще жестко «зашиты» в код.

В этой лабораторной работе мы добавим еще гибкости и реализуем передачу в шейдер координат точки, где был выполнен щелчок мышью. На рис. 1 показан скриншот работы программы.

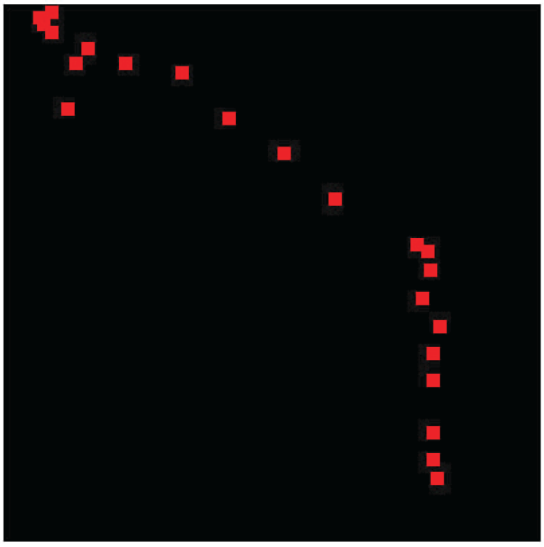

Рис. 1. Результат работы программы

Регистрация обработчиков событий

Обработчик события – это асинхронная функция обратного вызова, обрабатывающая события ввода от пользователя, такие как щелчки мышью или нажатия клавиш на клавиатуре. Возможность определения обработчиков позволяет создавать динамические веб-страницы и изменять их содержимое в соответствии с вводом пользователя. Чтобы задействовать обработчик, его нужно зарегистрировать (то есть, сообщить системе, что она должна вызывать функцию-обработчик при появлении указанного события). Элемент `<canvas>` поддерживает специальные свойства для регистрации обработчиков, которые и используются в данном примере.

Например, чтобы получить возможность обрабатывать события щелчков мышью, нужно присвоить свойству `onmousedown` элемента `<canvas>` ссылку на функцию, которая будет обрабатывать щелчки, как показано ниже. Для регистрации обработчика используется определение анонимной функции, то есть функции, не имеющей имени, что очень удобно, когда требуется определить функцию, не требующую уникального имени.

Например, в следующей строке программного кода определяется переменная `thanks`:

```
const thanks = function(){ alert(' Thanks a million!'); }
```

Эту переменную можно вызвать как функцию:

```
thanks(); // Выведет: 'Thanks a million!'
```

Как видите, переменная `thanks` может выступать в роли функции. Эти строки можно переписать иначе:

```
function thanks() { alert('Thanks a million!'); }
thanks(); // Выведет:'Thanks a million!'
```

С какой целью здесь используется анонимная функция? Дело в том, что для рисования точки нам нужны три константы: `gl`, `canvas` и `a_Position`. Эти три переменные являются локальными и определяются в функции `main()`. Но, когда возникает событие щелчка, браузер автоматически вызовет функцию, зарегистрированную в свойстве `onmousedown` элемента `<canvas>` с единственным предопределенным параметром – объектом события, содержащем информацию о щелчке. Обычно в программах на JavaScript сначала объявляется функция, а затем она регистрируется как обработчик события:

```
function mousedown(ev) { // Обработчик события: принимает один
                         // параметр 'ev'
...
}
...
canvas.onmousedown = mousedown; // Зарегистрировать 'mousedown'
как обработчик
```

Однако, такая функция не сможет получить доступ к локальным переменным `gl`, `canvas` и `a_Position`. Анонимная функция решает эту проблему, так как будучи объявленной в области видимости этих переменных, может обращаться к ним:

```
canvas.onmousedown = function(ev) {
click(ev,gl,canvas,a_Position); };
```

В этом случае, когда возникает событие щелчка мышью, сначала вызывается анонимная функция `function(ev)`, которая затем выполняет вызов и передает локальные переменные, объявленные в `main()`. Такой прием позволяет избежать необходимости использовать глобальные переменные.

Обработка события щелчка мышью.

Функция `click()` выполняет следующие действия:

1. Получает координаты указателя мыши в момент щелчка и сохраняет их в массиве.

2. Очищает `<canvas>`.

3. Для каждой пары координат в массиве рисует точку.

Координаты указателя мыши в момент щелчка хранятся в объекте события, который передается браузером в аргументе `ev`. Извлечь координаты из объекта `ev` можно обратившись к свойствам `ev.clientX` и `ev.clientY`. Однако эти координаты нельзя использовать непосредственно по двум причинам:

1. Координаты соответствуют положению указателя мыши в клиентской области окна браузера, а не в элементе `<canvas>` (см. рис. 2).

Рис. 2. Система координат клиентской области окна браузера и координаты элемента `<canvas>`

2. Система координат элемента `<canvas>` отличается от системы координат WebGL (см. рис. 3) – начало системы координат и направление оси *y* не совпадают.

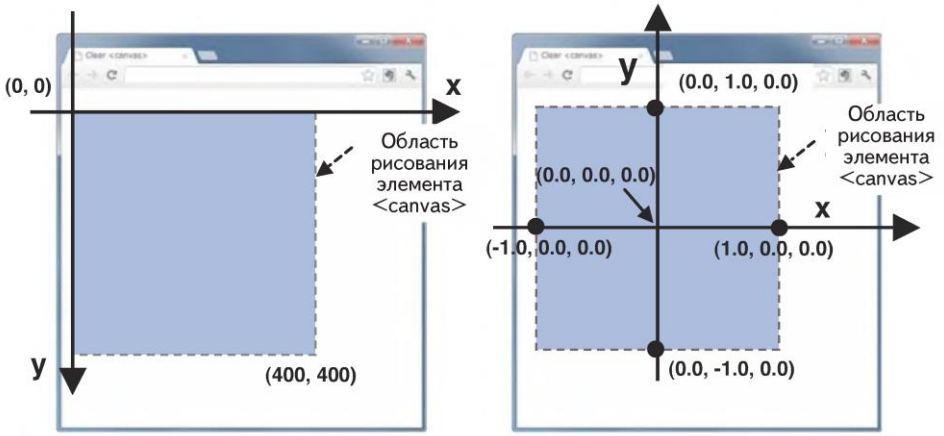

Рис. 3. Система координат элемента `<canvas>` (слева) и система координат WebGL в элементе `<canvas>` (справа)

Прежде всего нужно преобразовать координаты из системы координат клиентской области окна браузера в систему координат элемента <canvas>, а затем – в систему координат WebGL.

Значения rect.left и rect.top – это координаты верхнего левого угла <canvas> в клиентской области окна браузера (см. рис. 2). То есть, выражение (x - rect.left) и выражение (y - rect.top) смещают начало координат в позицию верхнего левого угла элемента <canvas>.

Далее нам нужно преобразовать координаты в элементе <canvas> в систему координат WebGL, как показано на рис. 3. Для этого требуется определить координаты центра элемента <canvas>. Получить высоту и ширину элемента <canvas> можно с помощью свойств canvas.height (в данном случае имеет значение 400) и canvas.width (так же имеет значение 400). Таким образом, центр элемента <canvas> будет иметь координаты (canvas.width/2, canvas.height/2).

Далее нам нужно реализовать это преобразование путем смещения начала координат в центр элемента <canvas>, где находится начало системы координат WebGL. Необходимое преобразование выполняется с помощью выражений ((x - rect.left) - canvas.width/2) и (canvas.height/2 - (y - rect.top)).

Наконец, как показано на рис. 3, диапазон значений по оси *x* в элементе <canvas> изменяется от 0 до canvas.width (400), а диапазон значений по оси *y* – от 0 до canvas.height (400). Но, так как диапазон значений по осям координат в WebGL изменяется от -1.0 до 1.0, на последнем шаге преобразования системы координат <canvas> в систему координат WebGL необходимо разделить координату *x* на canvas.width/2, а координату *y* – на canvas.height/2.

Преобразованные координаты указателя мыши сохраняются в массиве g_points, с помощью метода push(), который добавляет данные в конец массива.

Каждый раз, когда возникает событие щелчка, координаты указателя мыши добавляются в конец массива, как показано на рис. 4. (Длина массива при этом автоматически увеличивается.) Нумерация элементов массива начинается с 0, поэтому первый элемент доступен как g_points[0].

Рис. 4. Содержимое массива g_points

Далее выполняется очистка элемента <canvas>. После этого инструкция for последовательно переписывает координаты из массива g_points в attribute-переменную a_Position вершинного шейдера. Затем gl.drawArrays() рисует точку.

Так же, как было в программе 2.js, для передачи координат точки в attribute-переменную a_Position используется метод gl.vertexAttrib3f().

Массив g_points хранит координаты *x* и *y* указателя мыши в момент щелчков, как показано на рис. 4. То есть, если g_points[i] хранит координату *x*, то g_points[i+1] хранит координату *y*, поэтому переменная i цикла for увеличивается на 2.

Теперь, когда все готово к рисованию точки, осталось только нарисовать ее, что и делается вызовом `gl.drawArrays()`.

Задание для самостоятельной работы №1

Сейчас координаты *x* и *y* хранятся в массиве `g_points` по отдельности. Требуется сохранять их вместе, в виде массива. В этом случае в каждом элементе массива `g_points` будет сохраняться новый, двухэлементный массив `[x,y]`.

Извлечь отдельные координаты из массива можно следующим образом: сначала из массива извлекается элемент с парой координат, а так как сам элемент так же является массивом с парой координат (*x*, *y*), то, чтобы получить их, достаточно извлечь первый и второй элементы из этого массива. Благодаря этому приему можно хранить координаты *x* и *y* вместе, что упростит программу и повысит ее читаемость.

Изменение цвета точки

Как изменить цвет точки, мы узнали в лабораторной работе №2. Там мы изменяли сам фрагментный шейдер, подставляя выбранное значение цвета. Рассмотрим теперь вопрос задания цвета каждой точки из программного кода на JavaScript. Ранее мы уже передавали координаты точки из программы JavaScript в вершинный шейдер. Разница лишь в том, что здесь нам нужно будет передавать данные во фрагментный шейдер, а не в вершинный.

Для передачи данных во фрагментный шейдер можно использовать `uniform`-переменные (объявленные со спецификатором `uniform`) и реализуя те же шаги, что и при использовании `attribute`-переменных, только на этот раз целью является фрагментный шейдер, а не вершинный:

1. Объявить `uniform`-переменную во фрагментном шейдере.
2. Присвоить встроенной переменой `gl_FragColor` значение `uniform`-переменной.
3. Присвоить данные `uniform`-переменной.

`uniform`-переменные

`attribute`-переменные используются для передачи данных из программного кода на JavaScript в вершинные шейдеры. К сожалению, `attribute`-переменные доступны только вершинным шейдерам, а во фрагментных шейдерах следует использовать `uniform`-переменные. Существует также альтернативный механизм – `varying`-переменные, мы рассмотрим его в одной из следующих лабораторных работ.

`uniform`-переменные служат для передачи «одинаковых» («uniform»), не изменяющихся данных в вершинный или фрагментный шейдеры.

Прежде чем задействовать `uniform`-переменную, ее необходимо объявить. Объявление должно следовать стандартному шаблону (рис. 5):

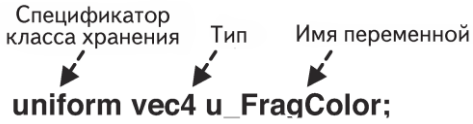

Рис. 5. Объявление `uniform`-переменной

В нашем примере программы uniform-переменная u_FragColor получила свое имя по переменной gl_FragColor, потому что значение этой uniform-переменной будет присваиваться переменной gl_FragColor. Префикс u_ в имени u_FragColor является частью наших соглашений об именовании и указывает, что данная переменная является uniform-переменной. Тип переменной u_FragColor должен соответствовать типу переменной gl_FragColor.

С помощью спецификатора точности (precision) определяется диапазон и точность представления значений переменными. Наш выбор mediump (средняя точность) в целом безопасен, поскольку существуют комбинации GPU/браузер, которые могут вообще не поддерживать highp (наивысшую точность).

Далее выполняется присваивание значения uniform-переменной u_FragColor переменной gl_FragColor, что вызывает окрашивание рисуемой точки в переданный цвет. Передача цвета через uniform-переменную напоминает использование attribute-переменных – нужно получить ссылку на переменную и затем в программе JavaScript записывать данные по этой ссылке.

Получение ссылки на uniform-переменную

Получить ссылку на uniform-переменную можно с помощью метода:

`gl.getUniformLocation(program, name)`
Возвращает ссылку на uniform-переменную, определяемую параметром name.

Параметры:

program	объект программы, хранящий вершинный и фрагментный шейдеры;
name	определяет имя uniform-переменной, ссылку на которую требуется получить.

Возвращаемое значение:

непустое значение	Ссылка на указанную uniform-переменную
null	Указанная uniform-переменная не найдена или ее имя начинается с зарезервированного префикса gl_ или webgl_.

Назначение и параметры этого метода полностью совпадают с назначением и параметрами метода gl.getAttribLocation(). Но, если запрошенная uniform-переменная не существует или имя начинается с зарезервированного префикса, этот метод возвращает null, а не –1. По этой причине возвращаемое значение следует проверять, сравнивая его со значением null. В логическом контексте значение null в языке JavaScript интерпретируется как false, поэтому для проверки результата можно использовать оператор "!".

Присваивание значения uniform-переменной

После получения ссылки на uniform-переменную, ей можно присвоить значение с помощью метода gl.uniform4f(). Он имеет то же назначение и параметры, что и методы gl.vertexAttrib[1234]f().

Семейство методов `gl.uniform()`

Используются для присвоения значения `uniform`-переменной.

`gl.uniform1f(location, v0)`
`gl.uniform2f(location, v0, v1)`
`gl.uniform3f(location, v0, v1, v2)`
`gl.uniform4f(location, v0, v1, v2, v3)`

Параметры:

`location`	Ссылка на `uniform`-переменную, которой требуется присвоить указанное значение.
`v0`, `v1`, `v2` и `v3`	Значения, которые должны быть присвоены первому, второму, третьему и четвертому элементам `uniform`-переменной.

Возвращаемое значение: нет

Задание для самостоятельной работы №2

Напишите программу, которая рисует точки цветом, зависящим от их координат в элементе `<canvas>`. В первом квадранте точки должны быть нарисованы красным цветом; в третьем квадранте – зеленым; в остальных квадрантах – белым (см. рис. 6).

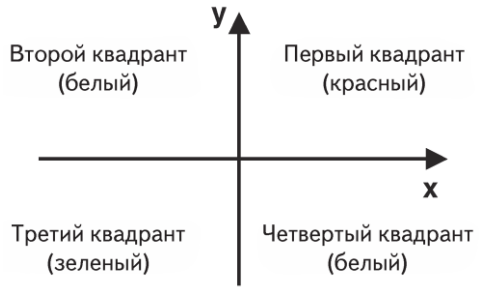

Рис. 6. Нумерация квадрантов в системе координат и соответствующие им цвета

Цвет точки должен сохраняться в массив `g_colors`. Далее программа должна выполнять обход точек в цикле и записывать соответствующее значение цвета в `uniform`-переменную `u_FragColor`.