Лабораторная работа №4

«Рисование множества точек»

Оглавление

Рисование множества точек

В предыдущей лабораторной работе мы изучили пример программы, которая рисует множество точек, соответствующих щелчкам мышью. Координаты щелчков сохранялись в массиве JavaScript (`g_points[]`). Чтобы нарисовать несколько точек, в ней использовался цикл, выполнявший обход массива и рисовавший точки по очереди, передавая в шейдер каждую вершину по отдельности с помощью вызова метода `gl.drawArrays()`.

Очевидно, что это решение можно использовать, только для рисования не связанных между собой точек. Для создания фигур, состоящих из нескольких вершин, таких как треугольники, прямоугольники и кубы, необходим способ одновременной передачи множества вершин в вершинный шейдер.

Система WebGL предусматривает удобный способ передачи множества вершин с использованием, так называемого буферного объекта. Буферный объект (Vertex Buffer Object (VBO)) — это область памяти в системе WebGL, где можно хранить множество вершин. Он используется и как область для сборки фигуры, и как средство передачи сразу нескольких вершин в вершинный шейдер.

В качестве примера одновременной работы с несколькими вершинами рассмотрим программу `4.js`, которая рисует три красные точки (затем по ним можно будет построить, например, треугольник).

Новый шаг, связанный с определением координат вершин, подготовки буферного объекта, сохранением множества вершин в буферном объекте и затем передачи его вершинному шейдеру, реализован в виде вызова функции `initVertexBuffers()`. Возвращаемое значение функции — число вершин, которые нужно нарисовать. В случае ошибки функция возвращает отрицательное значение.

Операция рисования выполняется единственным вызовом `gl.drawArrays()`, за исключением того, что в третьем аргументе передается значение `n` вместо `1`.

Использование буферных объектов

Итак, буферный объект, предоставляемый системой WebGL, является областью памяти, где хранятся вершины, которые требуется нарисовать. Применение буферного объекта позволяет передать в вершинный шейдер сразу несколько вершин, используя при этом единственную переменную-атрибут.

Чтобы посредством буферного объекта передать в вершинный шейдер массив данных, необходимо выполнить пять шагов:

1. Создать буферный объект (`gl.createBuffer()`).

2. Связать созданный буферный объект с текущим буферным объектом указанного типа (`gl.bindBuffer()`).

3. Записать данные в буферный объект (`gl.bufferData()`).

4. Связать буферный объект с переменной-атрибутом (`gl.vertexAttribPointer()`).

5. Разрешить присваивание (`gl.enableVertexAttribArray()`).

Создание буферного объекта (`gl.createBuffer()`)

Прежде, чем задействовать буферный объект, его нужно создать. Создается буфер вызовом WebGL-метода `gl.createBuffer()`:

`gl.createBuffer()`
Создает буферный объект

Возвращаемое значение:	
непустое	ссылка на вновь созданный буферный объект
`null`	признак ошибки создания буферного объекта

Соответствующий ему метод `gl.deleteBuffer()` удаляет буферный объект, созданный вызовом `gl.createBuffer()`.

`gl.deleteBuffer(buffer)`
Удаляет буферный объект, заданный параметром `buffer`

Параметры:	
`buffer`	ссылка на буферный объект, подлежащий удалению

Возвращаемое значение: нет

Связывание буферного объекта с текущим буферным объектом указанного типа (gl.bindBuffer())

Второй шаг после создания буферного объекта – связывание созданного буферного объекта с текущим буферным объектом указанного типа с помощью функции `gl.bindBuffer()`. Из текущего буферного объекта вершинный шейдер выбирает информацию о вершинах. Тип сообщает системе WebGL, какие данные находятся в буферном объекте, что обеспечивает правильную их обработку.

`gl.bindBuffer(target, buffer)`
Связывает буферный объект `buffer` с текущим и указывает его тип `target`

Параметры:		
`target`	Параметр типа `target` может иметь одно из следующих значений:	
	`gl.ARRAY_BUFFER`	указывает, что буферный объект содержит информацию о вершинах
	`gl.ELEMENT_ARRAY_BUFFER`	указывает, что буферный объект содержит значения индексов, ссылающихся на информацию о вершинах
`buffer`	Ссылка на буферный объект, предварительно созданный вызовом `gl.createBuffer()` Если в этом параметре передать `null`, связь буфера с текущим буфером типа `target` будет разорвана	

Возвращаемое значение: нет.

После считывания данных из текущего буферного объекта, связанного с созданным буферным объектом, хорошей практикой является его отвязывание. Для этого требуется вызвать следующую инструкцию:

```
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

Запись данных в буферный объект (gl.bufferData())

На третьем шаге выделяется память для буферного объекта и в него записываются данные. Для этого используется метод `gl.bufferData()`:

`gl.bufferData(target, data, usage)`
Выделяет память для буферного объекта и записывает данные `data` в буферный объект, тип которого определяется параметром `target`

Параметры:		
`target`	`gl.ARRAY_BUFFER` или `gl.ELEMENT_ARRAY_BUFFER`	
`data`	Данные для записи в буферный объект (типизированный массив)	
`usage`	Подсказка о том, как программа собирается использовать данные в буферном объекте. Эта подсказка помогает системе WebGL оптимизировать производительность, но не является страховкой от ошибок в вашей программе	
	`gl.STATIC_DRAW`	данные в буферном объекте будут определены один раз и использованы многократно для рисования фигур
	`gl.DYNAMIC_DRAW`	данные в буферном объекте будут определены многократно и использованы для рисования фигур так же многократно

Возвращаемое значение: нет

В программе `4.js` для передачи данных в вершинный шейдер, методу `gl.bufferData()` передается специальный массив `vertices`. Сам массив создается с помощью оператора `new` и данных в формате: <координата *x* и координата *y* первой вершины>, <координата *x* и координата *y* второй вершины>, и так далее. Вместо обычного JavaScript-объекта `Array` используется объект `Float32Array`. Это обусловлено тем, что стандартный массив в JavaScript является универсальной структурой данных, способной хранить числовые данные и строки, и она не оптимизирована для хранения больших объемов данных одного типа, таких как вершины. Чтобы решить эту проблему, в язык были добавлены типизированные массивы, такие как `Float32Array`.

Как и обычные массивы JavaScript, типизированные массивы так же имеют собственные методы, свойства и константы. Обратите внимание, что в отличие от стандартного объекта `Array`, типизированные массивы не поддерживают методы `push()` и `pop()`.

Типизированные массивы, как и стандартные, создаются с помощью оператора `new`, которому передаются исходные данные для массива. Например, чтобы создать массив вершин типа `Float32Array`, можно передать оператору `new` литерал массива `[0.0, 0.5, -0.5, -0.5, 0.5, -0.5]`. Обратите внимание, что оператор `new` является единственным способом создания типизированных массивов. В отличие от объекта `Array`, типизированные массивы не поддерживают создание с помощью оператора `[]`.

Настройка передачи информации из буферного объекта в переменную-атрибут вершинного шейдера (`gl.vertexAttribPointer()`)

В предыдущих лабораторных работах присваивание значений переменным-атрибутам выполнялось с помощью семейства методов `gl.vertexAttrib[1234]f()`. Однако эти методы могут использоваться только для присваивания единственного значения. Теперь требуется присвоить массив значений – координат вершин.

Для решения этой задачи можно использовать метод `gl.vertexAttribPointer()`, осуществляющий присваивание буферного объекта (в действительности – ссылки, или дескриптора, на буферный объект) переменной-атрибуту:

`gl.vertexAttribPointer(location, size, type, normalized, stride, offset)`
Присваивает буферный объект типа `gl.ARRAY_BUFFER` переменной-атрибуту `location`

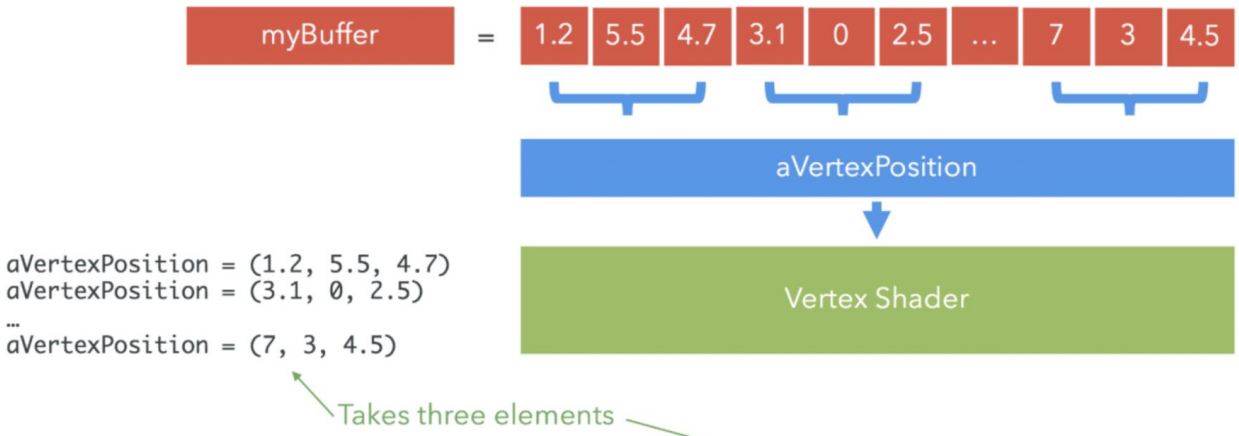
Параметры:	
`location`	Определяет переменную-атрибут, которой будет выполнено присваивание
`size`	Определяет число компонентов на вершину в буферном объекте (допустимыми являются значения от 1 до 4). Если значение параметра меньше числа элементов, требуемых переменной-атрибутом, отсутствующие компоненты автоматически получат значения по умолчанию, как указывалось в описании семейства методов `gl.vertexAttrib[1234]f()`. Например, если в параметре передать значение `1`, второй и третий компоненты получат значение `0.0`, а четвертый элемент – значение `1.0`. В программе `4.js` второй параметр в вызове метода `gl.vertexAttribPointer()` имеет значение `2`, так как для каждой вершины указываются только координаты *x* и *y*. Таким образом, для каждого прогона вершинного шейдера, компонентам *z* и *w* в переменной `a_Position`

	автоматически присваиваются значения `0.0` и `1.0`, потому что эта переменная требует наличия четырех компонентов (`vec4`), а ей передаются только два.
`type`	Определяет формат данных. Для вещественных чисел типа `Float32Array` параметр должен принимать значение `gl.FLOAT`.
`normalized`	Флаг, который позволяет нормализовать данные. Для типа `float` он не используется. Если значение равно `true`, беззнаковые целые отображаются в диапазон [0.0, 1.0], а целые со знаком – в диапазон [–1.0, 1.0]. Например, для типа `gl.UNSIGNED_BYTE` отображение в диапазон [0.0, 1.0] осуществляется с помощью деления на 255.
`stride`	Определяет длину шага (в байтах) для извлечения всей информации об одной вершине, то есть, число байтов между одинаковыми элементами данных разных вершин.
`offset`	Определяет смещение (в байтах) от начала буферного объекта, где хранятся данные для вершин. Если данные хранятся, начиная с самого начала буфера, в этом параметре следует передать значение `0`.

Возвращаемое значение: нет

Pointing an Attribute to the Currently Bound VBO

```
1. gl.bindBuffer(gl.ARRAY_BUFFER, myBuffer);
```




```
2. gl.vertexAttribPointer(aVertexPosition, 3, gl.FLOAT, false, 0, 0);

3. gl.enableVertexArrayAttrib(aVertexPosition);

4. gl.bindBuffer(gl.ARRAY_BUFFER, null);
```

После выполнения четвертого шага, подготовка буферного объекта к использованию практически закончена. Пятый и последний шаг – разрешение присваивания буферного объекта переменной-атрибуту.

Разрешение присваивания переменной-атрибуту (`gl.enableVertexAttribArray()`)

Чтобы сделать буферный объект доступным для вершинного шейдера, необходимо разрешить его присваивание переменной-атрибуту вызовом метода `gl.enableVertexAttribArray()`:

```
gl.enableVertexAttribArray(location)
```

Разрешает присваивание буферного объекта переменной-атрибуту, определяемой параметром `location`.

Параметры:	
`location`	Определяет переменную-атрибут.

Возвращаемое значение: нет

Запретить присваивание можно с помощью вызова метода `gl.disableVertexAttribArray()`:

`gl.disableVertexAttribArray(location)`
Запрещает присваивание буферного объекта переменной-атрибуту, определяемой параметром `location`.

Параметры:	
`location`	Определяет переменную-атрибут.

Возвращаемое значение: нет

Использование объектов массивов вершин

Объекты массива вершин (VAO) позволяют хранить всю информацию о привязке вершин/индексов для набора буферов в одном, легко управляемом объекте. То есть состояние атрибутов, какие буферы использовать для каждого атрибута и как извлекать данные из этих буферов, собираются в VAO.

VAO всегда следует использовать, поскольку они значительно сокращают время рендеринга. Если не использовать VAO, все данные атрибутов находятся в глобальном состоянии WebGL, что означает, что вызов таких функций, как `gl.vertexAttribPointer`, `gl.enableVertexAttribArray` и `gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, buffer)`, манипулируют глобальным состоянием. Это приводит к потере производительности, поскольку перед любым вызовом отрисовки нам нужно будет настроить все атрибуты вершин и индексы.

VAO позволяет настроить все атрибуты во время инициализации и просто привяжем настроенные данные при рендеринге, что обеспечит гораздо лучшую производительность.

Теперь осталось лишь нарисовать точки. Как и в предыдущих лабораторных работах, эта операция выполняется вызовом `gl.drawArrays`, но, так как выполняется рисование сразу нескольких точек, необходимо задействовать второй и третий параметры этого метода.

Второй и третий параметры метода `gl.drawArrays()`

Давайте познакомимся поближе с методом `gl.drawArrays()`:

`gl.drawArrays(mode, first, count)`
Выполняет вершинный шейдер, чтобы нарисовать фигуры, определяемые параметром `mode`.

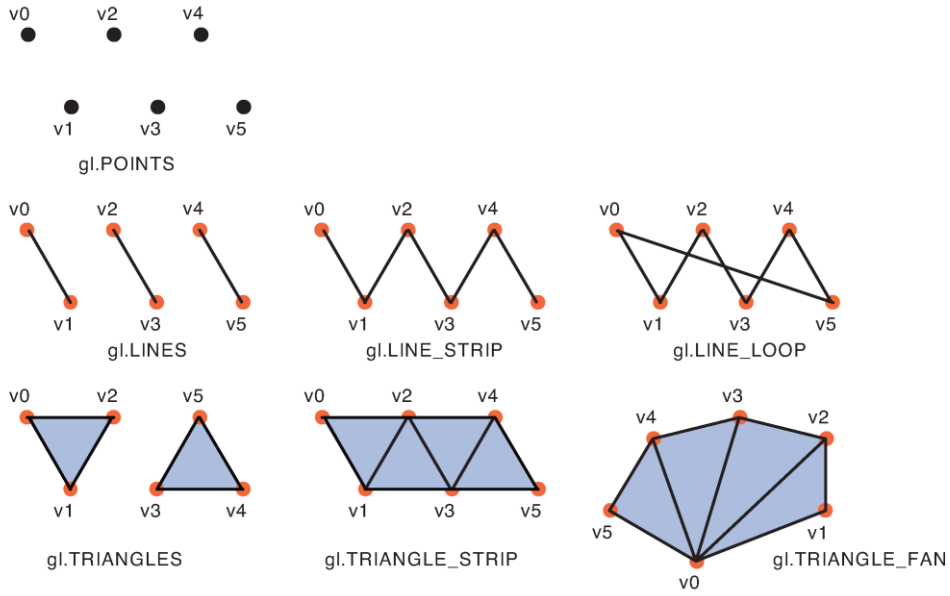


Рис. 1. Простые фигуры, поддерживаемые в WebGL

Параметры:	
`mode`	Определяет тип фигуры. Допустимыми значениями являются следующие константы: `gl.POINTS`, `gl.LINES`, `gl.LINE_STRIP`, `gl.LINE_LOOP`, `gl.TRIANGLES`, `gl.TRIANGLE_STRIP` и `gl.TRIANGLE_FAN`
`first`	Определяет номер вершины, с которой должно начинаться рисование (целое число)
`count`	Определяет количество вершин (целое число).

Как и в предыдущих лабораторных работах, так как мы рисуем всего лишь три точки, в первом параметре по-прежнему передается значение `gl.POINTS`. Во втором параметре передается `0`, потому что информация о вершинах располагается с самого начала буфера. В третьем параметре `count` передается `3`, потому что требуется нарисовать три точки. Выполняя эту функцию, программа фактически выполняет вершинный шейдер `count` раз (3 раза), последовательно передавая ему координаты вершин, хранящиеся в буферном объекте, через переменную-атрибут.

Эксперименты с примером программы

Поэкспериментируем с программой, чтобы лучше понять, как действует `gl.drawArrays()`, и попробуем поиграть со вторым и третьим его параметрами.

Сначала передадим `1` в третьем параметре `count`, вместо переменной `n` (имеющей значение 3). В этом случае вершинный шейдер будет выполнен только один раз и нарисована будет только одна точка — первая вершина в буферном объекте.

Если передать `1` во втором параметре, нарисована будет только вторая точка. Это объясняется тем, что передав `1` во втором параметре, мы сообщили WebGL начать рисование со второй вершины и нарисовать только одну вершину. То есть, на экране снова появится только одна точка, но на этот раз, соответствующая второй вершине.

Этот эксперимент позволяет быстро понять назначение параметров `first` и `count`. Но что произойдет, если изменить первый параметр `mode`?

Внесем в программу следующие изменения: значение `gl.POINTS` первого параметра в вызове метода `gl.drawArrays()` заменим значением `gl.TRIANGLES`. В данном случае три

вершины в буферном объекте больше не являются отдельными точками, а превращаются в три вершины треугольника.

Манипулируя значением первого параметра `mode` метода `gl.drawArrays()` можно рисовать разные фигуры. Доступны семь разных типов фигур: `gl.POINTS`, `gl.LINES`, `gl.LINE_STRIP`, `gl.LINE_LOOP`, `gl.TRIANGLES`, `gl.TRIANGLE_STRIP` и `gl.TRIANGLE_FAN`. Обратите внимание, что параметр, определяющий размер точки в вершинном шейдере `gl_PointSize = 10.0`, нужен только для случая рисования точек, в остальных случаях эту строчку можно закомментировать и на изображении это никак не скажется.

Рисование прямоугольника

Давайте теперь попробуем таким же способом, каким мы рисовали треугольник, нарисовать прямоугольник (рис. 2). WebGL не может рисовать четырехугольники непосредственно, поэтому нужно разбить прямоугольник на два треугольника (v0, v1, v2) и (v2, v1, v3), и затем нарисовать их, определив шесть вершин и передав в параметре `mode` значение `gl.TRIANGLES`.

Значение `gl.TRIANGLE_STRIP` позволяет нарисовать прямоугольник указав только четыре вершины.

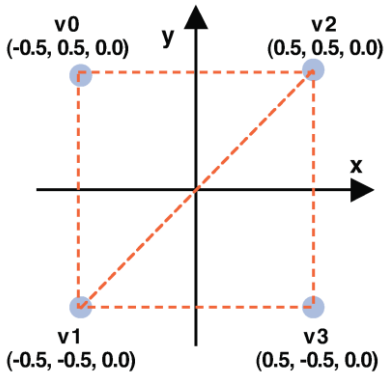


Рис. 2. Четыре координаты, необходимые для рисования прямоугольника

В программу нужно добавить координаты дополнительной вершины. Обратите внимание, в каком порядке следуют вершины, если он будет иной, команда рисования будет работать неправильно:

```
const vertices = new Float32Array([
-0.5, 0.5, -0.5, -0.5, 0.5, 0.5, 0.5, -0.5
]);
```

Поскольку теперь у нас четыре вершины, необходимо также изменить число вершин:

```
const n = 4; // Число вершин
```

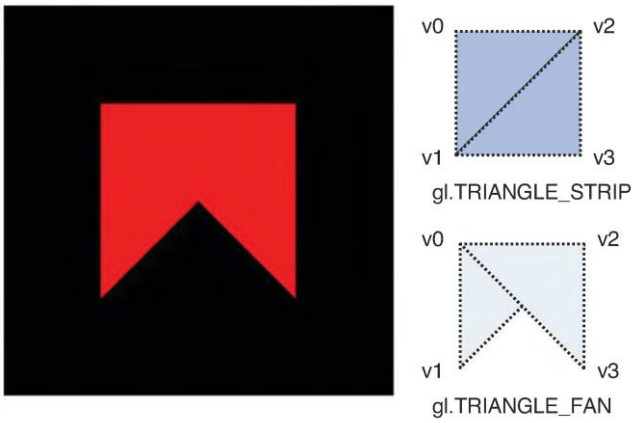
Затем нужно изменить вызов функции `gl.drawArrays()` как показано ниже, чтобы программа нарисовала прямоугольник:

```
gl.drawArrays( gl.TRIANGLE_STRIP , 0, n);
```

Эксперименты с примером программы

Теперь, когда мы получили представление о том, как использовать значение `gl.TRIANGLE_STRIP`, попробуем заменить его значением `gl.TRIANGLE_FAN`.

На этот раз получилась фигура, напоминающая флажок:



Взгляните на порядок следования вершин в программе и расположение треугольников рисуемых методом `gl.drawArrays()`, когда он получает параметр `mode` со значением `gl.TRIANGLE_FAN`. Это объясняет, почему получилась фигура, напоминающая флажок.

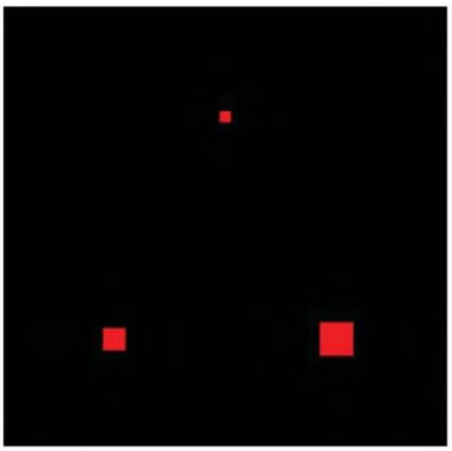
Задание для самостоятельной работы №1

Вернемся к программе, рисующей три точки. Сейчас благодаря строчке в вершинном шейдере

```
gl_PointSize = 10.0;
```

все точки имеют фиксированный размер, равный `10.0`.

По аналогии с передачей координат в вершинный шейдер с помощью буферных объектов, организуйте управление размерами точек из программы на JavaScript, и нарисуйте три точки разных размеров: `10.0`, `20.0` и `30.0`, соответственно:



Для этого реализуйте следующие шаги:

1. Создайте еще один буферный объект.

2. Свяжите созданный буферный объект с текущим буферным объектом указанного типа.

3. Запишите координаты в буферный объект.

4. Присвойте ссылку на буферный объект переменной-атрибуту.

5. Разрешите присваивание.

Передача разнородных данных в одном буферном массиве

Создание разных буферных объектов для разнородных данных часто используется в случае, если данные уже хранятся в разных массивах. Однако, WebGL позволяет хранить координаты вершин и размеры вместе, и поддерживает механизмы доступа к разнотипным данным. Например, координаты и размеры можно сгруппировать, как показано ниже, и такой прием часто называется перемежением (interleaving). С точки зрения производительности, этот способ является более предпочтительным, поскольку все необходимые данные считываются непрерывным образом из одной области памяти.

```
const verticesSizes = new Float32Array([
    // Координаты вершин и размеры точек
    0.0, 0.5, 10.0, // Первая точка
    -0.5, -0.5, 20.0, // Вторая точка
    0.5, -0.5, 30.0 // Третья точка
]);
```

После сохранения разнотипной информации о вершинах в общем буферном объекте возникает потребность в механизме доступа к разным элементам данных. Для этого можно использовать пятый (`stride`) и шестой (`offset`) аргументы метода `gl.vertexAttribPointer()`:

```
const FSIZE = verticesSizes.BYTES_PER_ELEMENT;
gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, FSIZE*3, 0);
gl.vertexAttribPointer(a_PointSize, 1, gl.FLOAT, false, FSIZE*3, FSIZE*2);
```

В переменной `FSIZE` передается размер (число байтов) одного элемента массива `verticeSizes`. Размер (число байтов) элемента типизированного массива можно получить с помощью свойства `BYTES_PER_ELEMENT`.

Параметр `stride` определяет число байтов, занимаемых данными для одной вершины в буферном объекте.

В предыдущих примерах, где в буфере сохранялась лишь однотипная информация (координаты), значение параметра `stride` фактически было не нужно (оно всегда должно было совпадать со значением параметра `size`), поэтому в него можно было передавать значение `0`. Однако теперь в одном и том же буфере хранятся и координаты, и размеры точек (рис. 3), поэтому теперь параметры `stride` и `size` будут иметь разные значения.

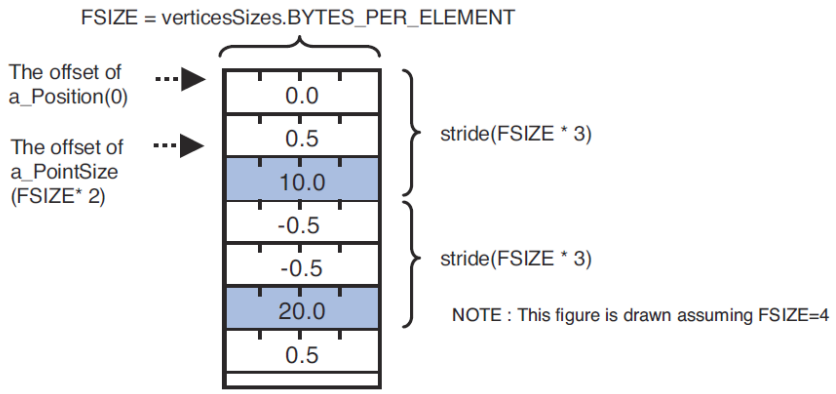


Рис. 3. Параметры `stride` и `offset`

Внутри каждой группы данных имеется три элемента (две координаты и один размер), поэтому размер шага (параметр `stride`) нужно установить равным трем размерам одного элемента в группе (то есть, `3*FSIZE` [число байтов в одном элементе массива `Floats32Array`]).

Параметр `offset` определяет расстояние (смещение) до первого элемента, который должен использоваться в этом вызове. Так как координаты вершин находятся в начале массива `verticesSize`, смещение равно 0.

То же самое делается для передачи размера точки. При этом мы используем тот же самый буфер, что перед этим использовался для передачи координат вершины. Чтобы сделать это возможным, мы задействуем шестой аргумент `offset`, в котором указываем смещение интересующих нас данных в буфере (в данном случае — размер точки). Первые два элемента массива – это координаты вершины, соответственно смещение должно быть равно `2*FSIZE`.

Задание для самостоятельной работы №2

Измените порядок хранения информации в массиве данных вершин – пусть сначала в нем содержится информация о размере точки, а затем о ее координатах. Настройте правильно параметры `stride` и `offset` для передачи данных из массива в шейдер. Продемонстрируйте результат работы программы.

Объединение информации из нескольких массивов в один буферный объект

В WebGL существует возможность формирования одного буферного объекта, объединяющего информацию из различных массивов. Например, для массива координат и массива размеров точек можно создавать не два отдельных буферных объекта, как мы делали ранее, а только один. Этот способ также обладает высокой производительностью, и при этом он позволяет не объединять разнородную информацию в одном массиве на этапе ее подготовки.

Для работы этого механизма нужно, как и ранее, вызвать метод `gl.bufferData()`, у которого в качестве второго параметра нужно указать не сам передаваемый в буферный объект массив, а только размер создаваемого буферного объекта:

`gl.bufferData(target, size, usage)`
Выделяет память для буферного объекта, тип которого определяется параметром `target`

Параметры:	
`target`	`gl.ARRAY_BUFFER` или `gl.ELEMENT_ARRAY_BUFFER`

`size`	Размер буферного объекта в байтах	
`usage`	Подсказка о том, как программа собирается использовать данные в буферном объекте. Эта подсказка помогает системе WebGL оптимизировать производительность, но не является страховкой от ошибок в вашей программе	
	`gl.STATIC_DRAW`	данные в буферном объекте будут определены один раз и использованы многократно для рисования фигур
	`gl.DYNAMIC_DRAW`	данные в буферном объекте будут определены многократно и использованы для рисования фигур так же многократно

Возвращаемое значение: нет

После определения размера буферного объекта, можно заполнять его данными вершин. Для этого используется функция `gl.bufferSubData()`. Для объединения информации используется параметр смещения `offset`, указывающий место начала записи данных из очередного массива.

`gl.bufferSubData(target, offset, data)`
Записывает данные `data` в буферный объект, тип которого определяется параметром `target`

Параметры:	
`target`	`gl.ARRAY_BUFFER` или `gl.ELEMENT_ARRAY_BUFFER`
`offset`	Определяет индекс элемента в буферном объекте, указывающий начальное местоположение, куда будут записываться данные.
`data`	Данные для записи в буферный объект (типизированный массив)

Возвращаемое значение: нет

Задание для самостоятельной работы №3

Вновь вернитесь к программе, использующей два разных массива для координат и размеров точек. Настройте передачу информации из них в один буферный объект с помощью методов `gl.bufferData()` и `gl.bufferSubData()`. Выполните правильную настройку параметров методов `gl.vertexAttribPointer()`. Продемонстрируйте результат работы программы.