Лабораторная работа №7

«Трехмерное наблюдение»

Рисование каркасной модели куба

Рисование трехмерных объектов начнем с рисования единичного куба в виде каркасной модели, как показано на рис. 1. Ребра куба должны быть параллельны координатным осям. Определим в каждой вершине куба свой цвет, чтобы с его помощью можно было однозначно определять различные грани куба и их ориентацию.

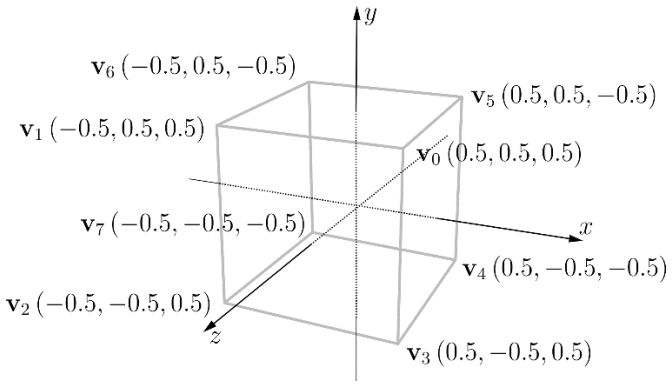

Рис. 1. Каркасный вид единичного куба

Для рисования каркасной модели нужно использовать один из следующих режимов рисования: `gl.LINES`, `gl.LINE_STRIP` или `gl.LINE_LOOP`. Будем пока использовать уже известную нам команду `gl.drawArrays()`.

Самый простой способ — рисование каждого ребра с помощью примитива `gl.LINES`. Тогда чтобы его нарисовать потребуется определить две вершины. Так как число ребер куба равно 12, общее число вершин, которое потребуется определить, составляет 2 × 12 = 24.

Однако есть возможность применить более экономный подход. Например, ребра, образующие переднюю и заднюю грань куба можно сформировать с помощью примитива `gl.LINE_LOOP`, а ребра боковых граней — с помощью примитива `gl.LINES`. Так как в режиме `gl.LINE_LOOP` ребра, образующие грань куба, можно нарисовать, определив всего 4 вершины, в конечном итоге придется определить 2 × 4 + 2 × 4 = 16 вершин. Однако нам потребуется вызвать отдельно функцию рисования для ребер передней и задней грани и отдельно для рисования ребер боковых граней. То есть, каждый из описанных подходов имеет свои недостатки, и не является идеальным.

Задание для самостоятельной работы

С помощью настройки положения камеры попытаться получить виды передней и задней грани. Отчего на самом деле сейчас зависит видимость грани? Ответив на этот вопрос, получить виды передней и задней грани.

Правильная обработка объектов переднего и заднего плана

Как мы только что убедились, по умолчанию, чтобы ускорить процесс рисования, WebGL рисует объекты в порядке следования вершин в буферном объекте.

Для решения этой проблемы в WebGL требуется задействовать возможности буфера глубины (Z-буфера), как показано на рис. 2.

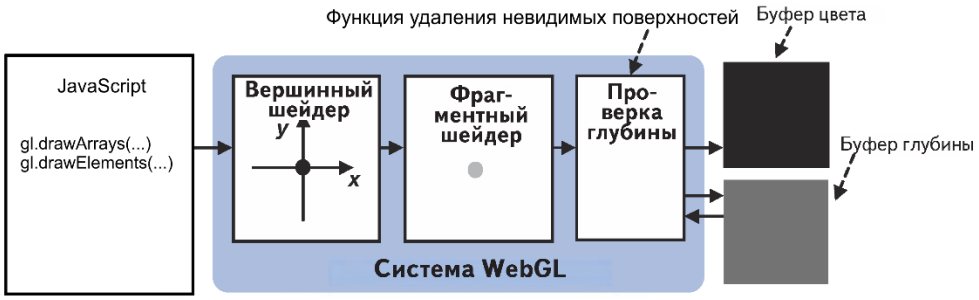

Рис. 2. Удаление невидимых поверхностей с помощью буфера глубины

Для использования буфера глубины нужно выполнить два следующих шага:

1. Активизировать буфер глубины:

```
gl.enable(gl.DEPTH_TEST);
```

2. Перед каждой перерисовкой выполнять очистку буфера глубины:

```
gl.clear(gl.DEPTH_BUFFER_BIT);
```

Для одновременной очистки буферов цвета и глубины можно использовать команду:

```
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

Задание для самостоятельной работы

С помощью настройки положения камеры и включенного Z-буфера получить виды передней и задней грани. Зависит ли теперь видимость грани от порядка вершин в буферном объекте? Проверить свой ответ на практике.

Что теперь не так с отображением передней и задней грани?

Использование матриц модели, вида и проекции

Для настройки трехмерного наблюдения используются две матрицы: матрица, определяющая тип проекции и границы видимого объема (матрица проекции), и матрица, определяющая настройки камеры (матрица вида). Так как тип проекции и границы видимого объема определяются в системе координат наблюдения после определения настроек камеры, то вычисления должны выполняться в следующем порядке:

```
<матрица проекции> × <матрица вида> × <координаты вершины>
```

В случае, если требуется провести еще и геометрические преобразования над изображаемой фигурой (например, преобразования перемещения, поворота или масштабирования) с помощью матрицы модели, то окончательные координаты вершин вычисляются по формуле:

```
<матрица проекции> × <матрица вида> × <матрица модели> × <координаты вершины>
```

Перемножение матриц проекции, вида и модели можно выполнять в вершинном шейдере при вычислении координат каждой вершины, однако данная реализация оказывается не самой эффективной, что будет особенно заметно при большом количестве вершин. Действительно, поскольку результат произведения этих матриц будет идентичен для всех вершин, его можно вычислить заранее один раз в коде на JavaScript и передать в вершинный шейдер уже готовый

результат – единственную матрицу. Эту матрицу можно назвать **матрицей модели вида проекции (model view projection matrix)** и обозначить, например, с использованием сокращения **mvp**.

Задание для самостоятельной работы

С помощью настроек ортогональной проекции и положения камеры получить вид **передней**, **задней**, **верхней** и **боковой** грани, а также **изометрический** и **аксонометрический** виды.

Задание для самостоятельной работы

С помощью перспективной проекции, используя **и функцию** `perspective` **и (в другой программе) функцию** `frustum` получить вид куба с **1**,**2** и **3** главными точками схождения.

Рисование объектов с использованием индексов и координат вершин

До сих пор для рисования мы использовали функцию `gl.drawArrays()`. Однако WebGL поддерживает альтернативный подход. С помощью функции `gl.drawElements()` можно рисовать фигуры не на основе непосредственно вершин каждого примитива, а на основе ссылок, т.е. индексов вершин в буферном объекте. Благодаря этому требуется определять лишь минимально необходимое число вершин и данных в них.

`gl.drawElements(mode, count, type, offset)`
Выполняет вершинный шейдер и рисует геометрическую фигуру с помощью примитивов типа `mode`, используя данные из индексного буферного объекта.

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`
`count`	Определяет количество **индексов**, участвующих в операции рисования (целое число).
`type`	Определяет тип индексов: `gl.UNSIGNED_BYTE` или `gl.UNSIGNED_SHORT`.
`offset`	Определяет смещение в массиве индексов в байтах, откуда следует начать рисование

Типу `gl.UNSIGNED_BYTE` в JavaScript соответствует типизированный массив `Uint8Array`, а типу `gl.UNSIGNED_SHORT` — типизированный массив `Uint16Array`.

Задание для самостоятельной работы

Письменно на бумаге составить таблицы вершин, ребер и граней для единичного куба. Программно создать массивы с подготовленной информацией.

Настройка индексных буферных объектов

Для работы функции `gl.drawElements()` нужно определить еще один буферный объект, который называется индексным. Создание индексного буферного объекта и запись в него данных из типизированного массива ничем не отличается от аналогичных операций для буферного объекта, в котором хранятся данные вершин, кроме указания типа, который для индексных буферных объектов имеет значение gl.ELEMENT_ARRAY_BUFFER.

Поскольку индексы не обрабатываются вершинными шейдерами, то для индексных буферных объектов не нужно вызывать функции `gl.vertexAttribPointer()` и `gl.enableVertexAttribArray()`.

На рис.3 показан пример использования информации о вершинах и индексах, записанной в буферные объекты

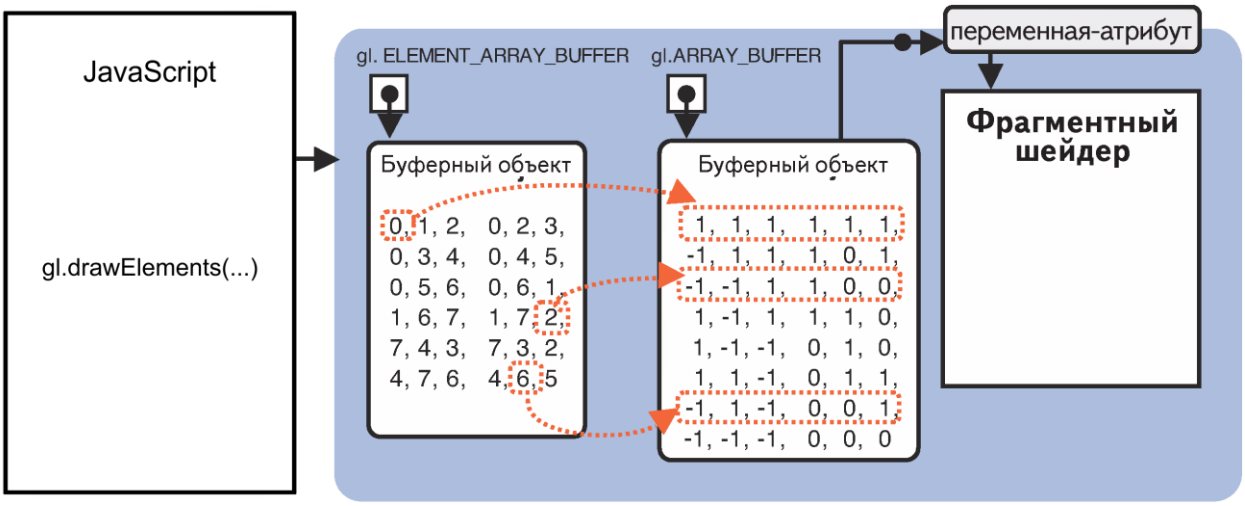

Рис. 3. Использование буферных объектов с данными о вершинах и индексах

Функция `gl.drawElements()` извлечет индексы из индексного буферного объекта и по ним отыщет информацию о вершинах в соответствующем буферном объекте. Затем эта информация будет передана в переменные-атрибуты вершинного шейдера. Эта процедура повторится для каждого индекса, после чего будет выполнено рисование фигуры. При таком подходе есть возможность многократно использовать одну и ту же информацию о вершинах, благодаря тому, что она извлекается с помощью индексов. Несмотря на то, что функция `gl.drawElements()` позволяет экономить память за счет многократного использования информации о вершинах, процедура рисования требует дополнительных вычислительных ресурсов для поиска информации о вершинах по значениям индексов.

Задание для самостоятельной работы

С помощью функции `gl.drawElements()` построить **каркасную** и **твердотельную** модель куба.