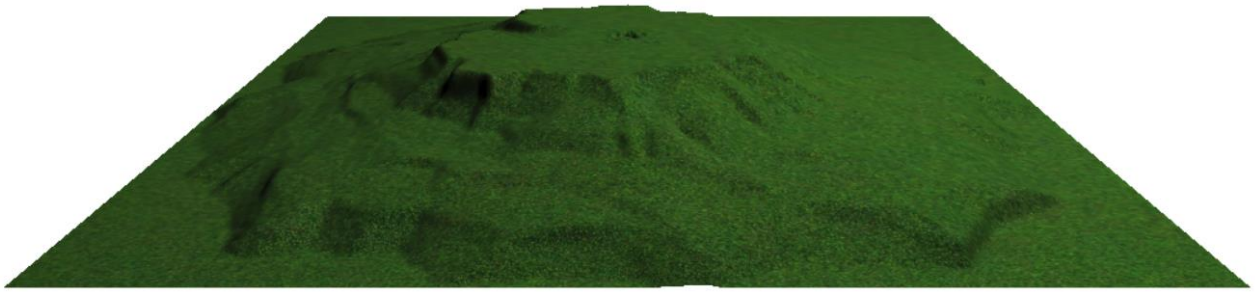
Лабораторная работа №1: Вершины, индексы, цвета и текстуры

Цель:

Целью данной лабораторной работы является получение базовых навыков разработки приложений трёхмерной графики с использованием библиотеки Three.js



Справка: саму библиотеку **three.js** можно получить по адресу: http://threejs.org/

Примеры исходного кода написаны для версии **r125** и могут иметь некоторые отличия от более поздних, или ранних версий.

В качестве среды разработки может быть использован любой редактор текста, однако, в рамках курса, предполагается использовать IDE Visual Studio Code: https://code.visualstudio.com/

Подробнее о установке и создании проекта в **VSCode** можно узнать из файла **Instalation.pdf**

Запуск проектов предлагается осуществлять в браузере Google Chrome или любом другом, поддерживающем WebGL 2.0.

Часть 1: Минимальное приложение

После создания файлов HTML, JavaScript и копирования файла библиотеки three.js, ваш проект будет готов к началу выполнения лабораторной работы.

Минимальный проект, как правило, состоит из двух частей, html страница (index.html) и связанный с ней script файл (обычно main.js).

Минимально необходимая html страница выглядит следующим образом:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        <!--объект веб страницы, в котором будет отображаться графика -->
        <div id="container"></div>
        <!-- Подключение библиотеки ThreeJS -->
        <script src="js/libs/three.min.js"></script>

        <!-- Подключение скрипта с графической программой -->
        <script src="js/main.js"></script>

    </body>
</html>
```

Важно следить за правильностью путей до библиотеки и скрипт файла.

Стоит отдельно выделить три действия:

- создание `div` элемента с `id = container`. В дальнейшем, этот элемент будет использован в качестве области отображения графического изображения.
- подключение библиотеки `three.min.js`. В данном файле, содержится описание функций для работы с графическим API WebGL.
- подключение пользовательского скрипта. Предполагается, что в нём содержится исходный код вашего проекта, отвечающий за создание, загрузку, перемещение и рендеринг графических объектов.

Минимально необходимый script файл состоит из следующих блоков:

Глобальные переменные:

```
// Ссылка на элемент веб страницы в котором будет отображаться графика
var container;

// Переменные "камера", "сцена" и "отрисовщик"
var camera, scene, renderer;
```

В самом начале js файла находятся объявления переменных, доступных в любой функции.

Вызов функций инициализации и обновления/отрисовки кадра:

```
// Функция инициализации камеры, отрисовщика, объектов сцены и т.д.
init();

// Обновление данных по таймеру браузера
animate();
```

Первая функция будет создавать и настраивать объекты, вторая будет изменять их параметры и выводить результат на экран.

Функция инициализации:

```
// В этой функции можно добавлять объекты и выполнять их первичную настройку
function init()
{
    // Получение ссылки на элемент html страницы
    container = document.getElementById( 'container' );
    // Создание "сцены"
    scene = new THREE.Scene();

    // Установка параметров камеры
    // 45 - угол обзора
    // window.innerWidth / window.innerHeight - соотношение сторон
    // 1 - 4000 - ближняя и дальняя плоскости отсечения
    camera = new THREE.PerspectiveCamera(
                                45, window.innerWidth / window.innerHeight, 1, 4000 );

    // Установка позиции камеры
    camera.position.set(5, 5, 5);

    // Установка точки, на которую камера будет смотреть
    camera.lookAt(new THREE.Vector3( 0,  0.0, 0));

    // Создание отрисовщика
    renderer = new THREE.WebGLRenderer( { antialias: false } );
    renderer.setSize( window.innerWidth, window.innerHeight );
    // Закрашивание экрана синим цветом, заданным в 16ричной системе
    renderer.setClearColor( 0x000000ff, 1);

    container.appendChild( renderer.domElement );

    // Добавление функции обработки события изменения размеров окна
    window.addEventListener( 'resize', onWindowResize, false );
}
```

Функция инициализации содержит, как минимум, связывание элемента веб страницы с графическим API, создание “сцены”, создание и установка камеры, создание и настройка рендера.

Так же, в этой функции будут выполняться любые действия, которые нужно выполнить только один раз за время работы приложения.

Например:

- загрузка моделей
- создание и настройка источника освещения

Обработка события изменения размеров окна:

```
function onWindowResize()
{
    // Изменение соотношения сторон для виртуальной камеры
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    // Изменение соотношения сторон рендера
    renderer.setSize( window.innerWidth, window.innerHeight );
}
```

В случае изменения размеров окна вывода, необходимо изменить параметры виртуальной камеры.

Функции изменения параметров объектов и рисования кадра:

```
// В этой функции можно изменять параметры объектов и обрабатывать действия пользователя
function animate()
{
    // Добавление функции на вызов, при перерисовки браузером страницы
    requestAnimationFrame( animate );

    render();
}

function render()
{
    // Рисование кадра
    renderer.render( scene, camera );
}
```

Основное назначение данного кода описано в комментариях и не требует подробного разъяснения.

Задание:

реализовать минимальное приложение three.js При запуске, в окне браузера, должна открываться html страница, залитая синим цветом.

Часть 2: Вершины, индексы, цвет

При использовании библиотеки three.js, работа с вершинами и индексами осуществляется через класс геометрии.

Например, **создание треугольника** будет выглядеть следующим образом:

```
var vertices = []; // Объявление массива для хранения вершин
var faces = [];    // Объявление массива для хранения индексов

var geometry = new THREE.BufferGeometry();// Создание структуры для хранения геометрии

vertices.push(1.0, 0.0, 3.0); // Добавление координат первой вершины в массив вершин
vertices.push(1.0, 3.0, 0.0); // Добавление координат второй вершины в массив вершин
vertices.push(3.0, 0.0, 1.0); // Добавление координат третьей вершины в массив вершин

faces.push(0, 1, 2); // Добавление индексов (порядок соединения вершин) в массив индексов
```

```
//Добавление вершин и индексов в геометрию
geometry.setAttribute( 'position', new THREE.Float32BufferAttribute( vertices, 3 ) );
geometry.setIndex( faces );
```

Добавление цветов осуществляется так же при помощи объекта геометрии. Цвета назначаются для каждой вершины отдельно:

```
var colors = [];   // Объявление массива для хранения цветов вершин

colors.push(0.8, 0.0, 0.0); // Добавление цвета для первой вершины (красный)
colors.push(0.0, 0.8, 0.0); // Добавление цвета для второй вершины (зелёный)
colors.push(0.0, 0.0, 0.8); // Добавление цвета для третьей вершины (синий)
//Добавление цветов вершин в геометрию
geometry.setAttribute( 'color', new THREE.Float32BufferAttribute( colors, 3 ) );
```

Обратите внимание, что цвета задаются в формате RGB. Каждый цветовой канал задаётся значением в диапазоне от 0 до 1.

В библиотеке three.js, графический объект помимо вершин, индексов и цвета имеет **настройки отображения**, которые хранятся в специальной структуре данных **“материал”**:

```
var triangleMaterial = new THREE.MeshBasicMaterial({
    vertexColors:THREE.VertexColors,
    wireframe: false,
    side:THREE.DoubleSide
});
```

В данном примере, используется определение источника цвета объекта (указаны цвета вершин), тип заливки объекта (заливать объект полностью) и режим отрисовки объекта (будут прорисованы обе стороны объекта). Опция отрисовки `wireframe`, позволяет отображать объект в виде полигональной сетки.

Финальным этапом является **создание объекта**, на основе геометрии и материала, и добавление его в сцену:

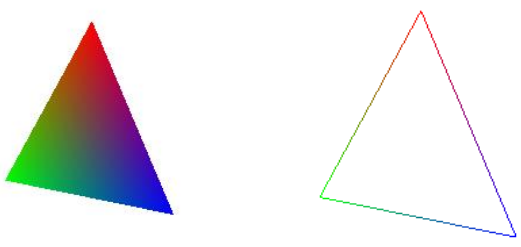
```
// Создание объекта и установка его в определённую позицию
var triangleMesh = new THREE.Mesh(geometry, triangleMaterial);
triangleMesh.position.set(0.0, 0.0, 0.0);

// Добавление объекта в сцену
scene.add(triangleMesh);
```

Следует помнить две вещи:

- только объекты могут быть добавлены сцену
- объекты могут появится на экране только после добавления в сцену

После выполнения этих действий, при запуске проекта, в окне браузера должно получиться следующее изображение (слева `wireframe: false`, справа `wireframe: true`):

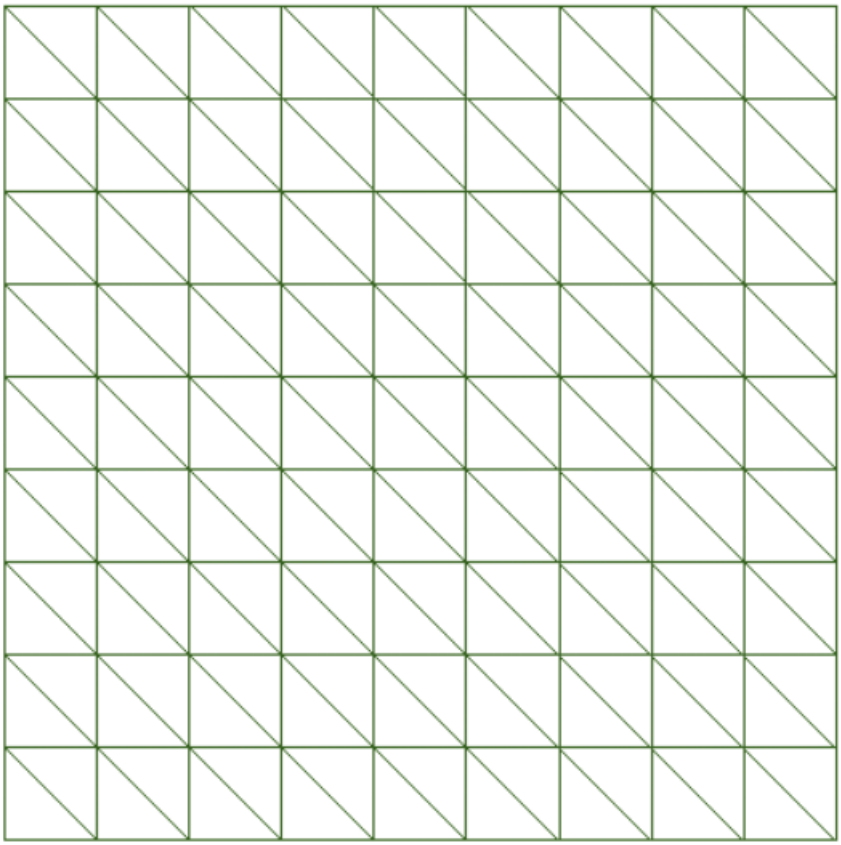


Задание:

разработать программу, отображающую на экран монитора регулярную полигональную сетку заданного размера. Размер задаётся в блоке объявления глобальных переменных в виде константы.

Все расчёты должны выполняться автоматически (вывести формулы).

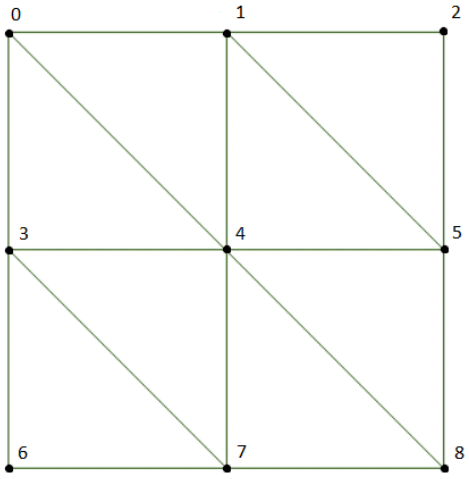
Пример регулярной сетки размером 10x10 вершин:



Для того, чтобы сгенерировать регулярную сетку, необходимо:

- рассчитать, сколько понадобится всего вершин
- расположить (задать координаты вершин в плоскости X|Z) вершины строками по N в каждой
- рассчитать, сколько в сетке должно быть треугольников
- составить алгоритм перечисления вершин, в порядке их соединения в треугольники

Пример для N = 3:



Общее число вершин = 9.

Координаты вершин: 0 - (0, 0, 0), 1 - (1, 0, 0), 2 - (2, 0, 0), 3 - (0, 0, 1), 4 - (1, 0, 1), 5 - (2, 0, 1), 6 - (0, 0, 2), 7 - (1, 0, 2), 8 - (2, 0, 2).

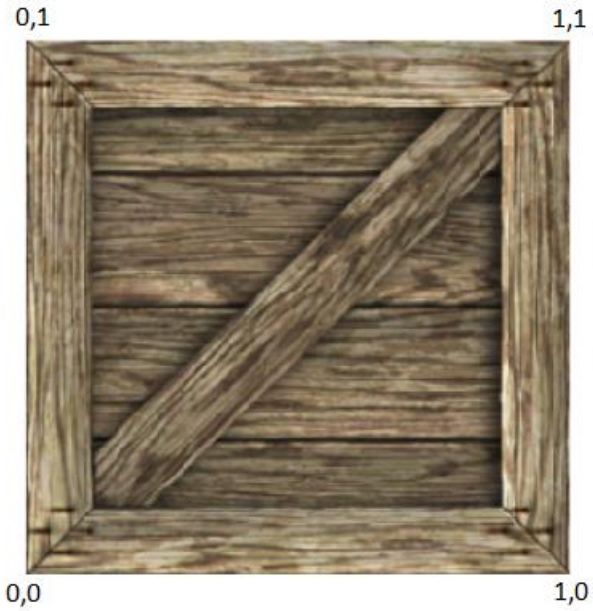
Общее число треугольников = 8.

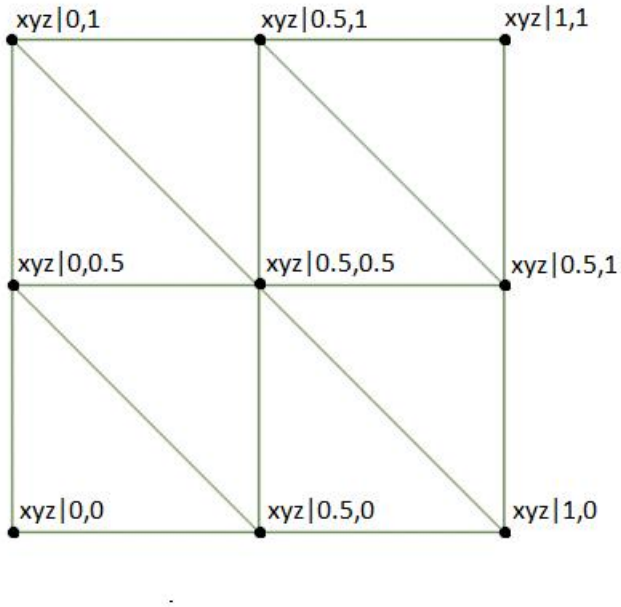
Порядок объединения вершин: [0, 1, 4], [0, 4, 3], [1, 2, 5], [1, 5, 4], [3, 4, 7], [3, 7, 6], [4, 5, 8], [4, 8, 7].

Часть 3: Текстуры

Под **текстурой**, как правило, понимается растровое изображение. Все текстуры имеют текстурные координаты в диапазоне от 0 до 1 по обеим осям. **Текстурированием**, называется процесс наложения текстуры на объект.

Для того, чтобы наложить текстуру на объект, необходимо для каждой вершины объекта указать соответствующие ей текстурные координаты:





Соответственно, в приведённом выше примере, текстура будет наложена на регулярную сетку полностью. Если задать диапазон текстурных координат больше единицы, текстура будет повторена, если меньше, наложена будет только часть текстуры.

Задание текстурных координат осуществляется так же при помощи объекта геометрия и, для прямоугольника, выглядит следующим образом:

```
var uvs = []; // Массив для хранения текстурных координат

uvs.push(0, 0); // Добавление текстурных координат для левой верхней вершины
uvs.push(1, 0); // Добавление текстурных координат для правой верхней вершины
uvs.push(1, 1); // Добавление текстурных координат для правой нижней вершины
uvs.push(0, 1); // Добавление текстурных координат для левой нижней вершины

//Добавление текстурных координат в геометрию
geometry.setAttribute( 'uv', new THREE.Float32BufferAttribute( uvs, 2 ) );
```

Добавление текстурных координат происходит в порядке добавления треугольников. То есть, первая добавленная тройка текстурных координат будет приписана первому добавленному треугольнику, вторая второму и т.д.

Непосредственно **загрузка текстуры** осуществляется следующим образом:

```
// Загрузка текстуры yachik.jpg из папки pics
var tex = new THREE.TextureLoader().load( 'img/yachik.jpg' );
```

Установка текстуры в материале выглядит следующим образом:

```
var mat = new THREE.MeshBasicMaterial({
    // Источник цвета - текстура
    map: tex,
    wireframe: false,
    side: THREE.DoubleSide
});
```

Текстурированный подобным образом четырёхугольник может выглядеть так:

При необходимости, можно задать **метод наложения текстуры**, например, повторение:

```
// Режим повторения текстуры
tex.wrapS = tex.wrapT = THREE.RepeatWrapping;
// Повторить текстуру 10x10 раз
tex.repeat.set( 10, 10 );
```

Задание:

модифицировать программу из прошлого раздела таким образом, чтобы для регулярной сетки рассчитывались текстурные координаты. Текстурные координаты рассчитываются по тому же принципу что и индексы треугольников.

Итоговое задание:

Разработать программу осуществляющую построение и визуализацию трёхмерного ландшафта по карте высот. Под картой высот, подразумевается grayscale растровое изображение, тёмные участки которого обозначают низины ландшафта, а светлые – возвышенности. Визуализируемая модель должна быть текстурирована.

Пример кода функций, для **чтения пикселей изображения**:

```
// Глобальная переменная для хранения карты высот
var imagedata;
…
var canvas = document.createElement('canvas');
var context = canvas.getContext('2d');
var img = new Image();

img.onload = function()
{
    canvas.width = img.width;
    canvas.height = img.height;
    context.drawImage(img, 0, 0 );
    imagedata = context.getImageData(0, 0, img.width, img.height);

    // Пользовательская функция генерации ландшафта
    CreateTerrain();
}

// Загрузка изображения с картой высот
img.src = 'pics/plateau.jpg';
```

Следует помнить, что загрузка изображений в JavaScript происходит асинхронно, поэтому функция генерации трёхмерного ландшафта должна быть вызвана по событию img.onload. (в данном примере, функция CreateTerrain();)

Получить **данные о высоте в конкретной точке** можно при помощи функции:

```
function getPixel( imagedata, x, y )
{
    var position = ( x + imagedata.width * y ) * 4, data = imagedata.data;
    return data[ position ];;
}
```

Пример использования функции:

```
//получение цвета пикселя в десятом столбце десятой строки изображения
var h = getPixel( imagedata, 10, 10 );
```

Так же, при выполнении задания, может быть использован встроенный механизм расчёта освещения, включающий в себя три этапа.

Добавление **источника освещения**:

```
//создание точечного источника освещения заданного цвета
var spotlight = new THREE.PointLight(0xffffff);
```

```
//установка позиции источника освещения
spotlight.position.set(100, 100, 100);
//добавление источника в сцену
scene.add(spotlight);
```

Для того, чтобы рассчитать освещение, необходимо иметь нормали объекта, а так же, назначить материал с поддержкой одной из моделей освещения.

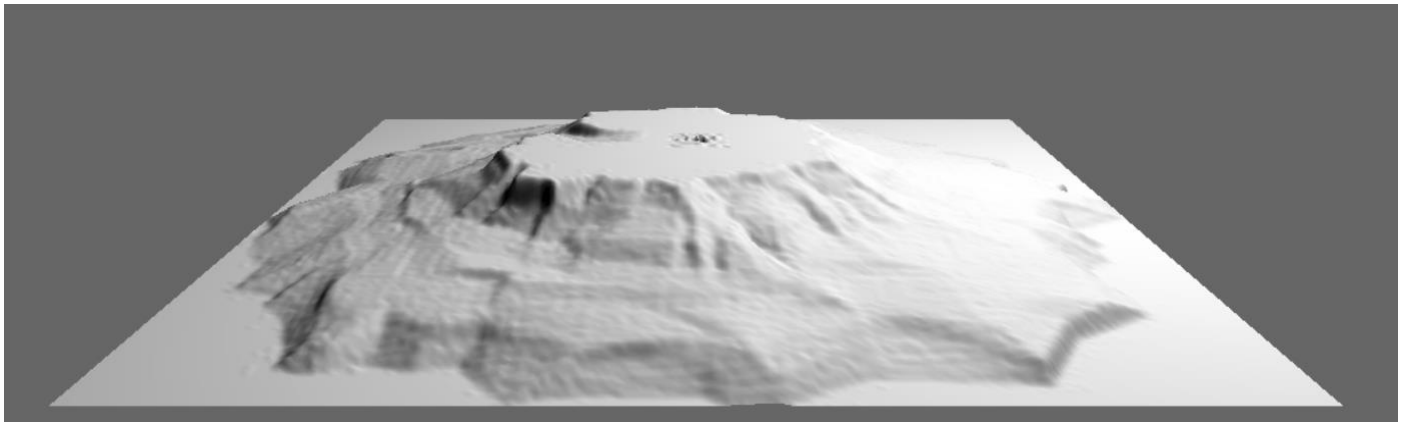
Расчёт **нормалей** для геометрии осуществляется следующим образом:

```
geometry.computeFaceNormals();
geometry.computeVertexNormals();
```

Использование **материала**, поддерживающего расчёт **освещения**:

```
var mat = new THREE.MeshLambertMaterial({
    map:tex,
    wireframe: false,
    side:THREE.DoubleSide
});
```

Предполагаемый результат до наложения текстуры:



После наложения текстуры: