МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ
по дисциплине «3D Компьютерная графика»
Тема: 3D трансформации

Студент гр. 5304 Лянгузов А.А.

Преподаватель Герасимова Т.В.

Санкт-Петербург
2019

Введение

Цель работы: представить 3D сцену и в ней один или несколько объектов из вашего задания (стараться представить все).

Освоить использование стандартных матричных преобразований над этими объектами и осуществить:

- просмотр трансформаций: через lookAt или эквивалентные модельные трансформации;
- трансформацию проекции: через perspective и ortho;

моделирование трансформации rotate, translate, scale и матричный стек.

Ход работы

В данной работе реализовано рисование параллелепипедов для сцены, вращение и масштабирование самой сцены, а также расширены возможности драйвера web-gl.

Шаг1. Редактирование фигур

Для начала, необходимо вместо простых двумерных фигур, задать точки для трехмерных. Поскольку на финальной сцене много повторяющихся примитивов, целесообразно сделать обертки, генерирующие точки для того или иного примитива. В финальной сцене много кубов, а также есть шахматная доска, представляющая собой параллелепипед.
Параллелепипед имеет 6 граней, каждую из которых можно представить как композицию из двух треугольников. Таким образом, задача рисования параллелепипеда сводится к рисованию последовательности треугольников.
Существуют разные способы рисования фигур на базе треугольников:

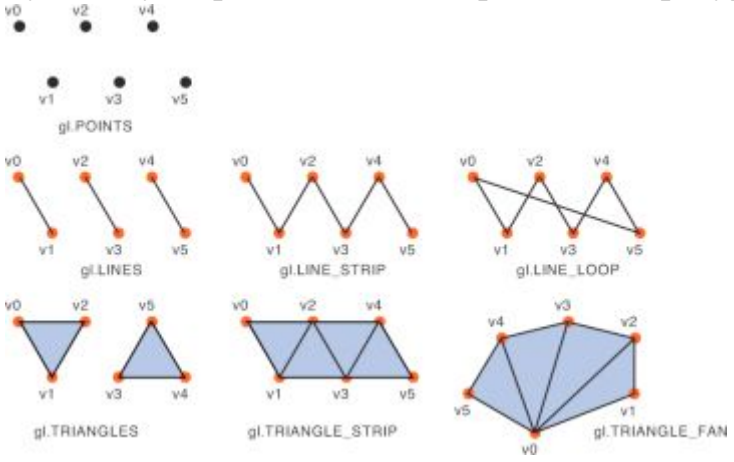

Рисунок 1 - Способы рисования по точкам в WebGL.

В данной работе будет использоваться TRIANGLE_FAN.
Итак, используя всю полученную информацию, был написан класс, инициализирующий массив вершин для рисования параллелепипеда:

```
class Parallelepiped {
	static build(position, sizeX, sizeY, sizeZ, color, angle = 0,
rotationAxis = COORDINATE_AXISES.Y) {
	return [
		new Figure(position,
			[
			new Vertex(-sizeX/2,  -sizeY/2,  -sizeZ/2, color),
			new Vertex(sizeX/2,  -sizeY/2,  -sizeZ/2, color),
			new Vertex(sizeX/2,  sizeY/2,  -sizeZ/2, color),
			new Vertex(-sizeX/2,  sizeY/2,  -sizeZ/2, color),

			new Vertex(-sizeX/2,  sizeY/2,  sizeZ/2, color),
			new Vertex(-sizeX/2,  -sizeY/2,  sizeZ/2, color),
			new Vertex(sizeX/2,  -sizeY/2,  sizeZ/2, color),
			new Vertex(sizeX/2,  -sizeY/2,  -sizeZ/2, color),
			],
			DRAWING_TYPE.FAN,
			angle,
			rotationAxis
		),

		new Figure(position,
			[
			new Vertex(sizeX/2,  sizeY/2,  sizeZ/2, color),
			new Vertex(sizeX/2,  -sizeY/2,  -sizeZ/2, color),
			new Vertex(sizeX/2,  sizeY/2,  -sizeZ/2, color),
			new Vertex(-sizeX/2,  sizeY/2,  -sizeZ/2, color),
			new Vertex(-sizeX/2,  sizeY/2,  sizeZ/2, color),
			new Vertex(-sizeX/2,  -sizeY/2,  sizeZ/2, color),
			new Vertex(sizeX/2,  -sizeY/2,  sizeZ/2, color),
			new Vertex(sizeX/2,  -sizeY/2,  -sizeZ/2, color),
			],
			DRAWING_TYPE.FAN,
			angle,
			rotationAxis
		),
	]
	}
}
```

Куб же, в свою очередь, является просто разновидностью параллелепипеда, поэтому код для рисования его вершин значительно проще:

```
class Cube {
	static build(position, size, color, angle = 0, rotationAxis =
COORDINATE_AXISES.Y) {
	return Parallelepiped.build(position, size, size, size, color,
angle, rotationAxis);
	}
}
```

После создания примитивов, требуется внести изменения в драйвер web-gl, чтобы он мог использовать различные типы рисования по точкам при рисовании фигуры. Для этого изменяем функцию рисования фигуры на холсте следующим образом:

```
/**
 * Рисует фигуру на холсте.
 * @param {Figure} figure - фигура.
 */
drawFigure: function(figure, verticesBuffer, colorBuffer) {
	this._setCurrentPosition(initialPosition);

	this._mvMatrixPush();
	mat4.multiply(mvMatrix, sceneRotationMatrix);

	this._setCurrentArrayBuffer(verticesBuffer);
	gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
	 Position.size(), gl.FLOAT, false, 0, 0);

	this._setCurrentArrayBuffer(colorBuffer);
	gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
	Color.size(), gl.FLOAT, false, 0, 0);

	this._setMatrixUniforms();

	switch (figure.getDrawingType()) {
		case DRAWING_TYPE.STRIP:
			gl.drawArrays(gl.TRIANGLE_STRIP, 0,
			figure.getVerticesCount());
			break;
		case DRAWING_TYPE.FAN:
```

```
                gl.drawArrays(gl.TRIANGLE_FAN, 0,
                figure.getVerticesCount());
                break;
            case DRAWING_TYPE.TRIANGLES:
                gl.drawArrays(gl.TRIANGLES, 0,
                figure.getVerticesCount());
                break;
        }


        this._mvMatrixPop();

        this._resetCurrentBuffer();
        this._setCurrentPositionToZero();
},
```

Далее, необходимо внести изменения в репозиторий:

```
/**
 * Репозиторий для всех объектов на сцене.
 */
let SHAPES_REPOSITORY = {
        data: [],

        init: function() {
        SHAPES_REPOSITORY.data = [
                // begin board
                ...Parallelepiped.build(new Position(0, 0, 0), 50, 5, 50,
new Color(0, 0, 0, 1)),
                // end board

                // begin cubes
                ...Cube.build(new Position(20, 5, 0.0), 5, new Color(1.0,
0.0, 0.0, 1.0)),
                ...Cube.build(new Position(20, 9, 0.0), 3, new Color(0.0,
0.0, 1.0, 1.0), 60, COORDINATE_AXISES.Y),
                ...Cube.build(new Position(-20, 4.0, 15.0), 3, new
Color(0.0, 1.0, 0.0, 1.0), 30, COORDINATE_AXISES.X),
                ...Cube.build(new Position(-10, 4.0, 15.0), 3, new
Color(0.0, 1.0, 0.0, 1.0), 120, COORDINATE_AXISES.Z),
                // end cubes
```

```
        ];
        }
}
```

Шаг 2. Вращение сцены

Для того, чтобы внесенные изменения были хоть сколько-нибудь заметны, требуется добавить возможность вращения сцены. Для этого правим головной файл приложения:

```
function init(canvas_node_id) {
        WEBGL_DRIVER.init(canvas_node_id);
        SHAPES_REPOSITORY.init();
}

function update() {
        requestAnimFrame(update);

        WEBGL_DRIVER.resetScene();

        SHAPES_REPOSITORY.data.forEach((figure) => {
        figure.draw();
        });

        animate();
}

let lastTime = 0;
function animate() {
        let timeNow = new Date().getTime();
        if (lastTime != 0) {
          let elapsed = timeNow - lastTime;
        }
        lastTime = timeNow;
}

function main(canvas_node_id) {
        init(canvas_node_id);
        update();
        setupUI();
}
```

```
function setupUI() {
      let container = document.getElementById('vertexColors');
      container.innerHTML = '';

      createVertexColorNode = function(figure, figureIndex) {
      let vertexIndex = 0;
      figure.getVertices().forEach((vertex) => {
            let wrapper = document.createElement('div');
                  let input = document.createElement('input');
                  input.setAttribute('id',
`vertex_${figureIndex}_${vertexIndex}`);
                  input.value = String(vertex.getColorCode());
                  input.addEventListener('keydown', function(e) {
                  if (e.keyCode === 13) {
                        colorChanged(e.target.id, this.value);
                  }
                  });
                  wrapper.appendChild(input);
            container.appendChild(wrapper);
            vertexIndex++;
      });
      }

      let i = 0;
      SHAPES_REPOSITORY.data.forEach((figure) => {
      createVertexColorNode(figure, i);
      i++;
      });
}

function colorChanged(node_id, value) {
      let meta = node_id.split('_');

      let figureIndex = meta[1];
      let vertexIndex = meta[2];

      meta = value.split(',');
      let r = meta[0];
      let g = meta[1];
      let b = meta[2];
      let a = meta[3];
      SHAPES_REPOSITORY.data[figureIndex].vertices[vertexIndex].color =
new Color(r, g, b, a);
```

```
        update();
}
```

Результат работы программы:

Перезагрузив, веб-страницу, получаем следующий результат.

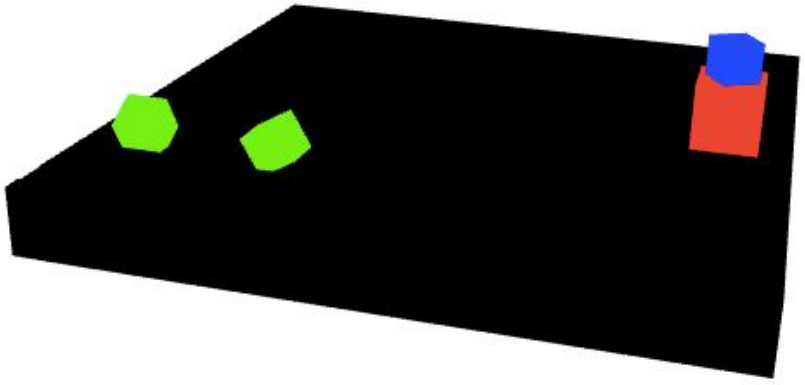

Рисунок 2 - Результат работы программы

Как можно видеть на скриншоте, кубы имеют разный цвет, различное положение на доске и повернуты под разными углами вокруг разных осей координат.

Выводы

В ходе лабораторной работы были созданы функции для рисования параллелепипедов, доработан драйвер для возможности использования различных режимов рисования по точкам. Также освоены матричные преобразования над объектами.

Исходный код

Файл index.html

```
<html>

<head>
<title>3D-graphics</title>
<meta http-equiv="content-type" content="text/html;
charset=ISO-8859-1">
<link rel="stylesheet" href="./app.css" />

<script type="text/javascript"
src="./libs/glMatrix-0.9.5.min.js"></script>
<script type="text/javascript" src="./libs/webgl-utils.js"></script>

<script id="shader-fs" type="x-shader/x-fragment">
      precision mediump float;

      varying vec4 vColor;

      void main(void) {
      gl_FragColor = vColor;
      }
</script>

<script id="shader-vs" type="x-shader/x-vertex">
      attribute vec3 aVertexPosition;
      attribute vec4 aVertexColor;

      uniform mat4 uMVMatrix;
      uniform mat4 uPMatrix;

      varying vec4 vColor;

      void main(void) {
      gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
      vColor = aVertexColor;
      }
</script>
```

```
<script type="text/javascript" src="./models/figure.js"></script>
<script type="text/javascript" src="./models/vertex.js"></script>
<script type="text/javascript" src="./models/position.js"></script>
<script type="text/javascript" src="./models/color.js"></script>

<script type="text/javascript" src="./models/shapes/cube.js"></script>
<script type="text/javascript"
src="./models/shapes/parallelepiped.js"></script>

<script type="text/javascript"
src="./models/shapes_repository.js"></script>
<script type="text/javascript" src="./models/webgl-driver.js"></script>
<script type="text/javascript" src="./app.js"></script>
<script type="text/javascript" src="./utils.js"></script>


</head>

<body onload="main('canvas-field');">
     <div id="container">
     <canvas id="canvas-field" width="800px" height="600px"></canvas>
     </div>
     <div id="vertexColors" style="display: none;">
     </div>
</body>

</html>
```

Файл figure.js

```
/**
 * Фигура - базовый примитив.
 */
class Figure {
     /**
     * Конструктор класса.
     * @param {Position} positionFromZero - позиция фигуры
относительно нуля.
     * @param {Array<Vertex>} vertices - массив координат
вершин.
     * @param {DRAWING_TYPE} drawingType - тип отрисовки.
```

```
     * @param {number} angle - угол поворота фигуры.
     * @param {COORDINATE_AXISES} rotationAxis - ось вдоль
которой требуется осуществить поворот.
     */
     constructor(positionFromZero, vertices, /*normalMatrix,
textureCoords, indexMatrix,*/ drawingType, angle, rotationAxis) {
     this.vertices = Figure.rotate(vertices, UTILS.degToRad(angle),
rotationAxis);
     this.vertices = Figure.moveTo(this.vertices, positionFromZero);
     //this.normalMatrix = normalMatrix;
     //this.textureCoords = textureCoords;
     //this.indexMatrix = indexMatrix;
     this.angle = UTILS.degToRad(angle);
     this.drawingType = drawingType;
     }

     /**
     * Возвращает вершины фигуры.
     */
     getVertices() {
     return this.vertices;
     }

     /**
     * Возвращает количество вершин фигуры.
     */
     getVerticesCount() {
     return this.vertices.length;
     }

     /**
     * Возвращает угол поворота фигуры.
     */
     getAngle() {
     return this.angle;
     }

     /**
     * Возвращает тип отрисовки фигуры.
     */
     getDrawingType() {
     return this.drawingType;
     }
```

```
	/**
	* Рисует фигуру на холсте, который
использует драйвер.
	*/
	draw() {
	let vertexBuffer = WEBGL_DRIVER.initBuffer(this.vertices,
BUFFER_TYPE.vertex);
	let colorBuffer = WEBGL_DRIVER.initBuffer(this.vertices,
BUFFER_TYPE.color);
	//let textureBuffer = WEBGL_DRIVER.initBuffer(this.textureCoords,
BUFFER_TYPE.texture);
	//let indexBuffer = WEBGL_DRIVER.initBuffer(this.indices,
BUFFER_TYPE.index);
	//let normalBuffer = WEBGL_DRIVER.initBuffer(this.normales,
BUFFER_TYPE.normal);

	WEBGL_DRIVER.drawFigure(this, vertexBuffer, colorBuffer);
	}

	/**
	* Возвращает вершины фигуры в виде
одномерного массива.
	*/
	static joinVerticesPositions(vertices) {
	let result = [];

	for(let vertex of vertices) {
		result.push(...vertex.getCoords());
	}

	return result;
	}

	static joinVerticesColors(vertices) {
	let result = [];

	for(let vertex of vertices) {
		result.push(...vertex.getColorCode());
	}

	return result;
	}
```

```
	static moveTo(vertices, position) {
	let result = [];

	for(let vertex of vertices) {
		let coords = vertex.getPosition();

		result.push(new Vertex(
			coords.getX() + position.getX(),
			coords.getY() + position.getY(),
			coords.getZ() + position.getZ(),
			vertex.getColor()
		));
	}

	return result;
	}

	static rotate(vertices, angle, rotationAxis) {
	let result = [];

	for(let vertex of vertices) {
		let coords = vertex.getPosition();
		let newCoords = {};

		switch(rotationAxis) {
			case COORDINATE_AXISES.X:
			newCoords = new Position(
				coords.getX(),
				coords.getY() * Math.cos(angle) - coords.getZ() *
Math.sin(angle),
				coords.getY() * Math.sin(angle) + coords.getZ() *
Math.cos(angle)
			);
			break;
			case COORDINATE_AXISES.Y:
			newCoords = new Position(
				coords.getX() * Math.cos(angle) - coords.getZ() *
Math.sin(angle),
				coords.getY(),
				coords.getX() * Math.sin(angle) + coords.getZ() *
Math.cos(angle)
			);
```

```
                break;
                case COORDINATE_AXISES.Z:
                newCoords = new Position(
                      coords.getX() * Math.cos(angle) - coords.getY() *
Math.sin(angle),
                      coords.getX() * Math.sin(angle) + coords.getY() *
Math.cos(angle),
                      coords.getZ(),
                );
                break;
                default:
                newCoords = coords;
          }

          result.push(new Vertex(
                newCoords.getX(),
                newCoords.getY(),
                newCoords.getZ(),
                vertex.getColor()
          ));
     }

     return result;
     }
}
```

Файл webgl-driver.js

```
// Контекст WebGL.
let gl;

// Шейдеры.
let shaderProgram;

// Модельно-видовая матрица.
let mvMatrix = mat4.create();
let mvMatrixStack = [];

// Проекционная матрица.
let pMatrix = mat4.create();

// Временный буфер.
let tmpBuffer = undefined;
```

```
let BUFFER_TYPE = {vertex : 0, color: 1, texture: 2, index: 3, normal:
4};

let DRAWING_TYPE = {STRIP: 0, FAN: 1, TRIANGLES: 2};

let COORDINATE_AXISES = {X: 0, Y: 1, Z: 2};

let mouseDown = false;
let lastMouseX = null;
let lastMouseY = null;

let sceneRotationMatrix = mat4.create();
mat4.identity(sceneRotationMatrix);

// Позиция начала отрисовки
let initialPosition = new Position(0, 0, -100);

/**
 * Драйвер WebGL.
 */
let WEBGL_DRIVER = {
     // public

     /**
     * Инициализирует драйвер.
     * @param {string} canvas_id - идентификатор canvas на
странице.
     */
     init: function(canvas_id) {
     let canvas = document.getElementById(canvas_id);
     WEBGL_DRIVER._initContext(canvas);
     WEBGL_DRIVER._initShaders();

     this.resetScene();

     canvas.onmousedown = WEBGL_DRIVER._onMouseDown;
     document.onmouseup = WEBGL_DRIVER._onMouseUp;
     document.onmousemove = WEBGL_DRIVER._onMouseMove;
     },

     resetScene() {
     gl.clearColor(1.0, 1.0, 1.0, 1.0);
```

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

        gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
        gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
        mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1,
1000.0, pMatrix);
        mat4.identity(mvMatrix);
        },

        /**
        * Инициализирует вершинный буфер.
        * @param {Array<any>} vertices - массив вершин.
        */
        initBuffer: function(vertices, type) {
        let buffer = this._createEmptyBuffer();

        this._setCurrentArrayBuffer(buffer);

        switch(type) {
             case BUFFER_TYPE.vertex:
                  this._fillVertexBuffer(vertices);
                  break;
             case BUFFER_TYPE.color:
                  this._fillColorBuffer(vertices);
                  break;
        }

        this._resetCurrentBuffer();

        return buffer;
        },

        /**
        * Рисует фигуру на холсте.
        * @param {Figure} figure - фигура.
        */
        drawFigure: function(figure, verticesBuffer, colorBuffer) {
        this._setCurrentPosition(initialPosition);

        this._mvMatrixPush();
        mat4.multiply(mvMatrix, sceneRotationMatrix);

        this._setCurrentArrayBuffer(verticesBuffer);
```

```
      gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
Position.size(), gl.FLOAT, false, 0, 0);

      this._setCurrentArrayBuffer(colorBuffer);
      gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
Color.size(), gl.FLOAT, false, 0, 0);

      this._setMatrixUniforms();

      switch (figure.getDrawingType()) {
            case DRAWING_TYPE.STRIP:
                  gl.drawArrays(gl.TRIANGLE_STRIP, 0,
figure.getVerticesCount());
                  break;
            case DRAWING_TYPE.FAN:
                  gl.drawArrays(gl.TRIANGLE_FAN, 0,
figure.getVerticesCount());
                  break;
            case DRAWING_TYPE.TRIANGLES:
                  gl.drawArrays(gl.TRIANGLES, 0,
figure.getVerticesCount());
                  break;
      }


      this._mvMatrixPop();

      this._resetCurrentBuffer();
      this._setCurrentPositionToZero();
      },

      //private

      /**
      * Инициализирует контекст WebGL.
      * @param {string} canvas - идентификатор canvas на
странице.
      */
      _initContext: function(canvas) {
      try {
            gl = canvas.getContext("experimental-webgl");
            gl.viewportWidth = canvas.width;
            gl.viewportHeight = canvas.height;
```

```
        } catch (e) {
        }
        if (!gl) {
             alert("Could not initialise WebGL, sorry :-(");
        }
        },

        /**
        * Инициализирует шейдеры.
        */
        _initShaders: function() {
        let fragmentShader = WEBGL_DRIVER._getShader(gl, "shader-fs");
        let vertexShader = WEBGL_DRIVER._getShader(gl, "shader-vs");

        shaderProgram = gl.createProgram();
        gl.attachShader(shaderProgram, vertexShader);
        gl.attachShader(shaderProgram, fragmentShader);
        gl.linkProgram(shaderProgram);

        if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
             alert("Could not initialise shaders");
        }

        gl.useProgram(shaderProgram);

        shaderProgram.vertexPositionAttribute =
gl.getAttribLocation(shaderProgram, "aVertexPosition");
        gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

        shaderProgram.vertexColorAttribute =
gl.getAttribLocation(shaderProgram, "aVertexColor");
        gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

        shaderProgram.pMatrixUniform =
gl.getUniformLocation(shaderProgram, "uPMatrix");
        shaderProgram.mvMatrixUniform =
gl.getUniformLocation(shaderProgram, "uMVMatrix");
        },

        /**
        * Инициализирует шейдер.
        * @param gl - контекст WebGL.
        * @param id - идентификатор тэга script,
```

```
содержащего шейдер.
     * @returns объект шейдера.
     */
     _getShader: function(gl, id) {
     let shaderScript = document.getElementById(id);
     if (!shaderScript) {
          return null;
     }

     let str = "";
     let k = shaderScript.firstChild;
     while (k) {
          if (k.nodeType == 3) {
               str += k.textContent;
          }
          k = k.nextSibling;
     }

     let shader;
     if (shaderScript.type == "x-shader/x-fragment") {
          shader = gl.createShader(gl.FRAGMENT_SHADER);
     } else if (shaderScript.type == "x-shader/x-vertex") {
          shader = gl.createShader(gl.VERTEX_SHADER);
     } else {
          return null;
     }

     gl.shaderSource(shader, str);
     gl.compileShader(shader);

     if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
          alert(gl.getShaderInfoLog(shader));
          return null;
     }

     return shader;
     },

     /**
     * Создает пустой буфер.
     */
     _createEmptyBuffer: function() {
     return gl.createBuffer();
```

```
    },

    /**
    * Устанавливает буфер в качестве
активного.
    * @param {WebGLBuffer} buffer - буфер.
    */
    _setCurrentArrayBuffer: function(buffer) {
    gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    },

    /**
    * Заполняет буфер данными о вершинах.
    * @param {Array<Vertex>} vertices - вершины.
    */
    _fillVertexBuffer: function(vertices) {
    gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(Figure.joinVerticesPositions(vertices)), gl.STATIC_DRAW);
    },

    /**
    * Заполняет буфер данными о цвете.
    * @param {Array<Color>} vertices - вершины.
    */
    _fillColorBuffer: function(vertices) {
    gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(Figure.joinVerticesColors(vertices)), gl.STATIC_DRAW);
    },

    /**
    * Сбрасывает текущий буфер.
    * Устанавливается временный буфер.
    */
    _resetCurrentBuffer: function() {
    if(tmpBuffer === undefined) {
        tmpBuffer = this._createEmptyBuffer();
    }
    this._setCurrentArrayBuffer(tmpBuffer);
    },

    /**
    * Устанавливает позицию для рисования.
    * @param {Position} positionFromZero - позиция
```

```
относительно нуля.
     */
     _setCurrentPosition(positionFromZero) {
     this._setCurrentPositionToZero();
     mat4.translate(mvMatrix, positionFromZero.getCoords());
     },

     /**
     * Перемещает позицию для рисования в
нулевые координаты.
     */
     _setCurrentPositionToZero() {
     mat4.identity(mvMatrix);
     },

     /**
     * Инициализирует поля программы
шейдеров (uniform-переменные) объектами JS,
     * соответствующими матрице проекции и
матрице модели.
     */
     _setMatrixUniforms: function () {
     gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
     gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false,
mvMatrix);
     },

     _mvMatrixPush: function() {
     let copy = mat4.create();
     mat4.set(mvMatrix, copy);
     mvMatrixStack.push(copy);
     },

     _mvMatrixPop: function() {
     if (mvMatrixStack.length == 0) {
          throw "Matrix stack is empty";
     }
     mvMatrix = mvMatrixStack.pop();
     },

     _onMouseDown: function (event) {
     mouseDown = true;
     lastMouseX = event.clientX;
```

```
        lastMouseY = event.clientY;
        },

        _onMouseUp: function (event) {
        mouseDown = false;
        },


        _onMouseMove: function (event) {
        if (!mouseDown) {
             return;
        }

        let newX = event.clientX;
        let newY = event.clientY;
        let deltaX = newX - lastMouseX;

        let newRotationMatrix = mat4.create();
        mat4.identity(newRotationMatrix);
        mat4.rotate(newRotationMatrix, UTILS.degToRad(deltaX / 10), [0, 1,
0]);
        let deltaY = newY - lastMouseY;
        mat4.rotate(newRotationMatrix, UTILS.degToRad(deltaY / 10), [1, 0,
0]);
        mat4.multiply(newRotationMatrix, sceneRotationMatrix,
sceneRotationMatrix);
        lastMouseX = newX
        lastMouseY = newY;
        }
}
```