# 

|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **АЭРОКОСМИЧЕСКИЙ**

КАФЕДРА **ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И МАТЕМАТИЧЕСКАЯ ФИЗИКА**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **02.03.01 МАТЕМАТИКА И КОМПЬЮТЕРНЫЕ НАУКИ**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 7 |

**Название: Трёхмерное наблюдение**

**Дисциплина: Компьютерная геометрия**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | АК3-51Б |  |  | Щербаков Н. С. |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Захаров А. А. |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2021

**Задание 1**

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

**Решение**

Код файла lab\_07.html:

<!DOCTYPE html>

<html>

  <head>

    <meta charset="utf-8">

    <title>Лабораторная работа №7</title>

  </head>

  <body onload="main()">

    <canvas id="webgl" width="400" height="400">

    Please use a browser that supports "canvas"

    </canvas>

    <script src="lib/webgl-utils.js"></script>

    <script src="lib/webgl-debug.js"></script>

    <script src="lib/cuon-utils.js"></script>

    <script src="lib/dat.gui.js"></script>

    <script src="lib/gl-matrix.js"></script>

    <script src="lab\_07\_1.js"></script>

  </body>

</html>

Код файла lab\_07\_1.js:

"use strict";

const VSHADER\_SOURCE =

  '#version 100\n' +

  'precision mediump float;\n' +

  'attribute vec4 a\_Position;\n' +

  'varying vec4 v\_FragColor;\n' +

  'attribute vec4 a\_Color;\n' +

  'uniform mat4 u\_MVP;\n' +

  'void main() {\n' +

  '  gl\_Position = u\_MVP \* a\_Position;\n' +

  '  gl\_PointSize = 10.0;\n' +

  '  v\_FragColor = a\_Color;\n'+

  '}\n';

const FSHADER\_SOURCE =

  'precision mediump float;\n' +

  'varying vec4 v\_FragColor;\n' +

  'void main() {\n' +

  '  gl\_FragColor = v\_FragColor;\n' +

  '}\n';

const {mat2, mat3, mat4, vec2, vec3, vec4} = glMatrix;

function main() {

  const canvas = document.getElementById('webgl');

  const gl = getWebGLContext(canvas);

  if (!gl) {

    console.log('Failed to get the rendering context for WebGL');

    return;

  }

  if (!initShaders(gl, VSHADER\_SOURCE, FSHADER\_SOURCE)) {

    console.log('Failed to intialize shaders.');

    return;

  }

  const n = initVertexBuffers(gl);

  if (n < 0) {

    console.log('Failed to set the positions of the vertices');

    return;

  }

  gl.clearColor(0.0, 0.0, 0.0, 1.0);

  const controls = {

    view: 'axonometry'

  };

  const gui = new dat.GUI();

  const guiCamera = gui.addFolder('camera');

  const projection = {

      perspective: "Orthographic",

      switchCamera: function () {

          if (this.perspective == "Perspective") {

              this.perspective = "Orthographic";

              guiCamera.remove(view);

              controls.view = 'axonometry';

              view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

                  console.log(e);

                  controls.view = e;

              });

          } else {

              this.perspective = "Perspective";

              guiCamera.remove(view);

              controls.view = '3-point';

              view = guiCamera.add(controls, 'view', ['1-point', '2-point', '3-point']).onChange(function (e) {

                  console.log(e);

                  controls.view = e;

              });

          }

      }

  };

  guiCamera.add(projection, 'switchCamera');

  guiCamera.add(projection, 'perspective').listen();

  let view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

      console.log(e);

      controls.view = e;

  });

  let eye = vec3.create();

  function render() {

    switch (controls.view) {

        case 'left':

        break;

        case 'right':

        break;

        case 'top':

        break;

        case 'bottom':

        break;

        case 'front':

          vec3.set(eye, 0.0, 0.0, 0.5)

        break;

        case 'back':

          vec3.set(eye, 0.0, 0.0, -0.5)

        break;

        case 'isometry':

        break;

        case 'axonometry':

        break;

        case '1-point':

        break;

        case '2-point':

        break;

        case '3-point':

        break;

    }

    const u\_MVP = gl.getUniformLocation(gl.program, 'u\_MVP');

    if (u\_MVP < 0) {

      console.log('Failed to get the storage location of u\_MVP');

      return -1;

    }

    let viewMatrix = mat4.create();

    mat4.lookAt(viewMatrix, eye, [0.0, 0.0, 0.0], [0.0, 1.0, 0.0]);

    gl.clear(gl.COLOR\_BUFFER\_BIT);

    gl.uniformMatrix4fv(u\_MVP, 0, viewMatrix);

    gl.drawArrays(gl.LINES, 0, n);

    requestAnimationFrame(render);

  }

  render();

}

function initVertexBuffers(gl) {

  const vertices = new Float32Array([

  0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, -0.4, 0.4,

  -0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, 0.4, 0.4,

  0.4, 0.4, 0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, 0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, 0.4,

  -0.4, -0.4, -0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, -0.4,

  -0.4, -0.4, -0.4,

  0.4, -0.4, -0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,]);

  const Color = new Float32Array([

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0

  ]);

  const n = vertices.length;

  const vertexBuffer = gl.createBuffer();

  if (!vertexBuffer) {

    console.log('Failed to create the buffer object');

    return -1;

  }

  gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

  var FSIZE = vertices.BYTES\_PER\_ELEMENT;

  gl.bufferData(gl.ARRAY\_BUFFER, 6 \* n \* FSIZE, gl.STATIC\_DRAW);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 0, vertices);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 3 \* n \* FSIZE, Color);

  const a\_Position = gl.getAttribLocation(gl.program, 'a\_Position');

  if (a\_Position < 0) {

    console.log('Failed to get the storage location of a\_Position');

    return -1;

  }

  const a\_Color = gl.getAttribLocation(gl.program, 'a\_Color');

  if (!a\_Color) {

    console.log('Failed to get the storage location of a\_Color');

    return;

  }

  gl.vertexAttribPointer(a\_Position, 3, gl.FLOAT, false, 0, 0);

  gl.vertexAttribPointer(a\_Color, 3, gl.FLOAT, false, 0, FSIZE \* 3 \* n);

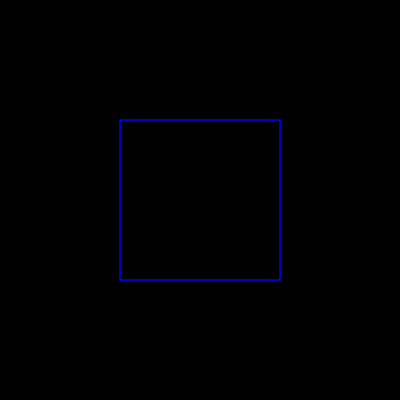
  gl.enableVertexAttribArray(a\_Position);

  gl.enableVertexAttribArray(a\_Color);

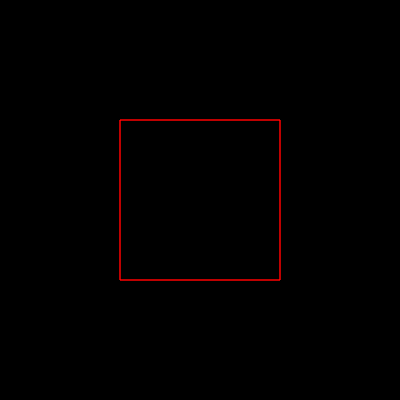
  return n;

}

Результат выполнения программы имеет вид:



Несмотря на попытки получить виды передней и задней граней куба, всегда видна только задняя грань куба. Это происходит из-за того, что задняя грань рисуется последней. В результате происхоит наложение задней грани на переднюю. Чтобы получить вид передней грани, достатотчно поменять последовательность рисования граней, т е. передняя грань должна рисоватся последней. Вид передней грани куба:



**Задание 2**

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

Что теперь не так с отображением передней и задней граней куба?

**Решение**

Код файла lab\_07\_2.js:

"use strict";

const VSHADER\_SOURCE =

  '#version 100\n' +

  'precision mediump float;\n' +

  'attribute vec4 a\_Position;\n' +

  'varying vec4 v\_FragColor;\n' +

  'attribute vec4 a\_Color;\n' +

  'uniform mat4 u\_MVP;\n' +

  'void main() {\n' +

  '  gl\_Position = u\_MVP \* a\_Position;\n' +

  '  gl\_PointSize = 10.0;\n' +

  '  v\_FragColor = a\_Color;\n'+

  '}\n';

const FSHADER\_SOURCE =

  'precision mediump float;\n' +

  'varying vec4 v\_FragColor;\n' +

  'void main() {\n' +

  '  gl\_FragColor = v\_FragColor;\n' +

  '}\n';

const {mat2, mat3, mat4, vec2, vec3, vec4} = glMatrix;

function main() {

  const canvas = document.getElementById('webgl');

  const gl = getWebGLContext(canvas);

  if (!gl) {

    console.log('Failed to get the rendering context for WebGL');

    return;

  }

  if (!initShaders(gl, VSHADER\_SOURCE, FSHADER\_SOURCE)) {

    console.log('Failed to intialize shaders.');

    return;

  }

  const n = initVertexBuffers(gl);

  if (n < 0) {

    console.log('Failed to set the positions of the vertices');

    return;

  }

  gl.clearColor(0.0, 0.0, 0.0, 1.0);

  gl.enable(gl.DEPTH\_TEST);

  const controls = {

    view: 'axonometry'

  };

  const gui = new dat.GUI();

  const guiCamera = gui.addFolder('camera');

  const projection = {

      perspective: "Orthographic",

      switchCamera: function () {

          if (this.perspective == "Perspective") {

              this.perspective = "Orthographic";

              guiCamera.remove(view);

              controls.view = 'axonometry';

              view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

                  console.log(e);

                  controls.view = e;

              });

          } else {

              this.perspective = "Perspective";

              guiCamera.remove(view);

              controls.view = '3-point';

              view = guiCamera.add(controls, 'view', ['1-point', '2-point', '3-point']).onChange(function (e) {

                  console.log(e);

                  controls.view = e;

              });

          }

      }

  };

  guiCamera.add(projection, 'switchCamera');

  guiCamera.add(projection, 'perspective').listen();

  let view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

      console.log(e);

      controls.view = e;

  });

  let eye = vec3.create();

  function render() {

    switch (controls.view) {

        case 'left':

        break;

        case 'right':

        break;

        case 'top':

        break;

        case 'bottom':

        break;

        case 'front':

          vec3.set(eye, 0.0, 0.0, 0.5)

        break;

        case 'back':

          vec3.set(eye, 0.0, 0.0, -0.5)

        break;

        case 'isometry':

        break;

        case 'axonometry':

        break;

        case '1-point':

        break;

        case '2-point':

        break;

        case '3-point':

        break;

    }

    const u\_MVP = gl.getUniformLocation(gl.program, 'u\_MVP');

    if (u\_MVP < 0) {

      console.log('Failed to get the storage location of u\_MVP');

      return -1;

    }

    let viewMatrix = mat4.create();

    mat4.lookAt(viewMatrix, eye, [0.0, 0.0, 0.0], [0.0,  1.0, 0.0]);

    gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

    gl.uniformMatrix4fv(u\_MVP, 0, viewMatrix);

    gl.drawArrays(gl.LINES, 0, n);

    requestAnimationFrame(render);

  }

  render();

}

function initVertexBuffers(gl) {

  const vertices = new Float32Array([

  0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, -0.4, 0.4,

  -0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, 0.4, 0.4,

  0.4, 0.4, 0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, 0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, 0.4,

  -0.4, -0.4, -0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, -0.4,

  -0.4, -0.4, -0.4,

  0.4, -0.4, -0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,]);

  const Color = new Float32Array([

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0

  ]);

  const n = vertices.length;

  const vertexBuffer = gl.createBuffer();

  if (!vertexBuffer) {

    console.log('Failed to create the buffer object');

    return -1;

  }

  gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

  var FSIZE = vertices.BYTES\_PER\_ELEMENT;

  gl.bufferData(gl.ARRAY\_BUFFER, 6 \* n \* FSIZE, gl.STATIC\_DRAW);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 0, vertices);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 3 \* n \* FSIZE, Color);

  const a\_Position = gl.getAttribLocation(gl.program, 'a\_Position');

  if (a\_Position < 0) {

    console.log('Failed to get the storage location of a\_Position');

    return -1;

  }

  const a\_Color = gl.getAttribLocation(gl.program, 'a\_Color');

  if (!a\_Color) {

    console.log('Failed to get the storage location of a\_Color');

    return;

  }

  gl.vertexAttribPointer(a\_Position, 3, gl.FLOAT, false, 0, 0);

  gl.vertexAttribPointer(a\_Color, 3, gl.FLOAT, false, 0, FSIZE \* 3 \* n);

  gl.enableVertexAttribArray(a\_Position);

  gl.enableVertexAttribArray(a\_Color);

  return n;

}

Теперь, после настройки положения камеры и включения Z-буфера, видимость граней не зависит от порядка вершин в буферном объекте. Однако, пытаясь получить вид передней грани, можно видеть только заднюю грань, и наоборот.

**Задание 3**

1. С помощью настроек ортогональной проекции и положения камеры получить вид передней грани куба.

2. С помощью настроек ортогональной проекции и положения камеры получить вид задней грани куба.

3. С помощью настроек ортогональной проекции и положения камеры получить вид верхней грани куба.

4. С помощью настроек ортогональной проекции и положения камеры получить вид боковой грани куба.

5. С помощью настроек ортогональной проекции и положения камеры изометрический вид куба.

6. С помощью настроек ортогональной проекции и положения камеры получить аксонометрический вид куба.

7. С помощью настройки объема наблюдения, сделайте панорамное отображение куба и отображение куба крупным планом.

**Решение**

Код файла lab\_07\_3.js:

"use strict";

const VSHADER\_SOURCE =

  '#version 100\n' +

  'precision mediump float;\n' +

  'attribute vec4 a\_Position;\n' +

  'varying vec4 v\_FragColor;\n' +

  'attribute vec4 a\_Color;\n' +

  'uniform mat4 u\_MVP;\n' +

  'void main() {\n' +

  '  gl\_Position = u\_MVP \* a\_Position;\n' +

  '  gl\_PointSize = 10.0;\n' +

  '  v\_FragColor = a\_Color;\n'+

  '}\n';

const FSHADER\_SOURCE =

  'precision mediump float;\n' +

  'varying vec4 v\_FragColor;\n' +

  'void main() {\n' +

  '  gl\_FragColor = v\_FragColor;\n' +

  '}\n';

const {mat2, mat3, mat4, vec2, vec3, vec4} = glMatrix;

function main() {

  const canvas = document.getElementById('webgl');

  const gl = getWebGLContext(canvas);

  if (!gl) {

    console.log('Failed to get the rendering context for WebGL');

    return;

  }

  if (!initShaders(gl, VSHADER\_SOURCE, FSHADER\_SOURCE)) {

    console.log('Failed to intialize shaders.');

    return;

  }

  const n = initVertexBuffers(gl);

  if (n < 0) {

    console.log('Failed to set the positions of the vertices');

    return;

  }

  gl.clearColor(0.0, 0.0, 0.0, 1.0);

  gl.enable(gl.DEPTH\_TEST);

  const controls = {

    view: 'axonometry'

  };

  const gui = new dat.GUI();

  const guiCamera = gui.addFolder('camera');

  let projectionMatrix = mat4.create();

  const projection = {

      perspective: "Orthographic",

      switchCamera: function () {

          if (this.perspective == "Perspective") {

            mat4.ortho(projectionMatrix, -1.0, 1.0, -1.0, 1.0, 0.0, 3.0);

            //mat4.ortho(projectionMatrix, -5.0, 5.0, -5.0, 5.0, 0.0, 3.0);

            //mat4.ortho(projectionMatrix, -0.6, 0.6, -0.6, 0.6, 0.0, 3.0);

            this.perspective = "Orthographic";

            guiCamera.remove(view);

            controls.view = 'axonometry';

            view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

              console.log(e);

              controls.view = e;

            });

          } else {

              this.perspective = "Perspective";

              guiCamera.remove(view);

              controls.view = '3-point';

              view = guiCamera.add(controls, 'view', ['1-point', '2-point', '3-point']).onChange(function (e) {

                  console.log(e);

                  controls.view = e;

              });

          }

      }

  };

  guiCamera.add(projection, 'switchCamera');

  guiCamera.add(projection, 'perspective').listen();

  let view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

      console.log(e);

      controls.view = e;

  });

  let eye = vec3.create();

  let up = vec3.create();

  function render() {

    switch (controls.view) {

        case 'left':

          vec3.set(eye, -0.5, 0.0, 0.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'right':

          vec3.set(eye, 0.5, 0.0, 0.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'top':

          vec3.set(eye, 0.0, 0.5, 0.0); vec3.set(up, 0.0, 0.0, -1.0)

        break;

        case 'bottom':

          vec3.set(eye, 0.0, -0.5, 0.0); vec3.set(up, 0.0, 0.0, 1.0)

        break;

        case 'front':

          vec3.set(eye, 0.0, 0.0, 0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'back':

          vec3.set(eye, 0.0, 0.0, -0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'isometry':

          vec3.set(eye, 0.5, 0.5, 0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'axonometry':

          vec3.set(eye, 0.5, 0.7, 0.9); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case '1-point':

        break;

        case '2-point':

        break;

        case '3-point':

        break;

    }

    const u\_MVP = gl.getUniformLocation(gl.program, 'u\_MVP');

    if (u\_MVP < 0) {

      console.log('Failed to get the storage location of u\_MVP');

      return -1;

    }

    let MVP\_Matrix = mat4.create();

    let viewMatrix = mat4.create();

    mat4.lookAt(viewMatrix, eye, [0.0, 0.0, 0.0], up);

    mat4.multiply(MVP\_Matrix, projectionMatrix, viewMatrix);

    gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

    gl.uniformMatrix4fv(u\_MVP, 0, MVP\_Matrix);

    gl.drawArrays(gl.LINES, 0, n);

    requestAnimationFrame(render);

  }

  render();

}

function initVertexBuffers(gl) {

  const vertices = new Float32Array([

  0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, -0.4, 0.4,

  -0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, 0.4, 0.4,

  0.4, 0.4, 0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, 0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, 0.4,

  -0.4, -0.4, -0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, -0.4,

  -0.4, -0.4, -0.4,

  0.4, -0.4, -0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,]);

  const Color = new Float32Array([

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0

  ]);

  const n = vertices.length;

  const vertexBuffer = gl.createBuffer();

  if (!vertexBuffer) {

    console.log('Failed to create the buffer object');

    return -1;

  }

  gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

  var FSIZE = vertices.BYTES\_PER\_ELEMENT;

  gl.bufferData(gl.ARRAY\_BUFFER, 6 \* n \* FSIZE, gl.STATIC\_DRAW);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 0, vertices);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 3 \* n \* FSIZE, Color);

  const a\_Position = gl.getAttribLocation(gl.program, 'a\_Position');

  if (a\_Position < 0) {

    console.log('Failed to get the storage location of a\_Position');

    return -1;

  }

  const a\_Color = gl.getAttribLocation(gl.program, 'a\_Color');

  if (!a\_Color) {

    console.log('Failed to get the storage location of a\_Color');

    return;

  }

  gl.vertexAttribPointer(a\_Position, 3, gl.FLOAT, false, 0, 0);

  gl.vertexAttribPointer(a\_Color, 3, gl.FLOAT, false, 0, FSIZE \* 3 \* n);

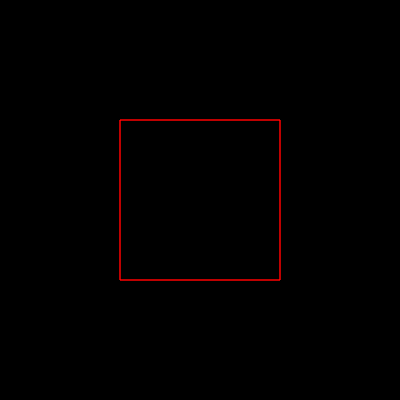
  gl.enableVertexAttribArray(a\_Position);

  gl.enableVertexAttribArray(a\_Color);

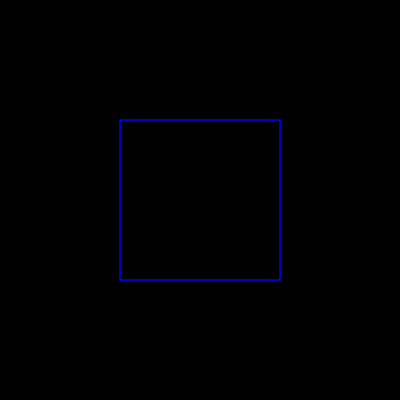
  return n;

}

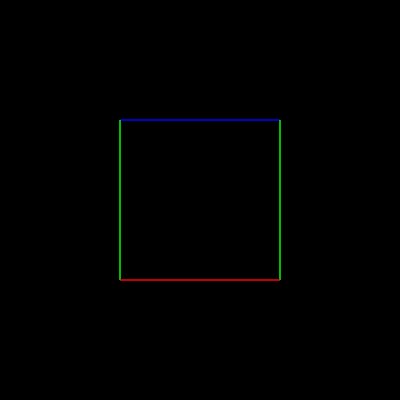
Вид передней грани куба:



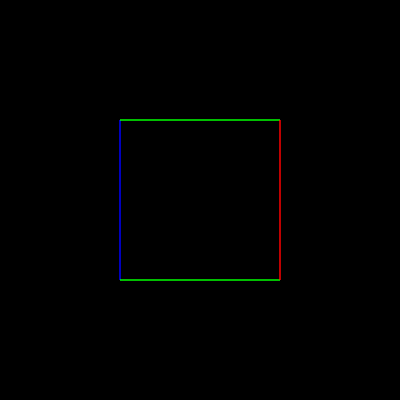
Вид задней грани куба:



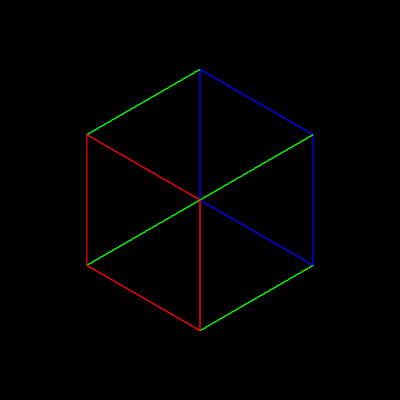
Вид верхней грани куба:



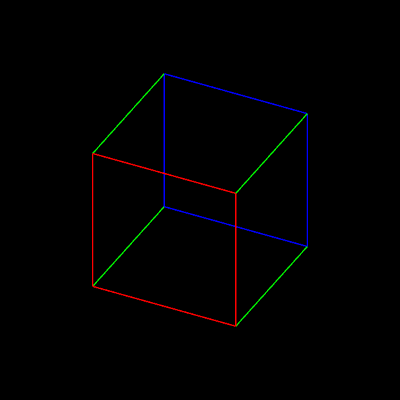
Вид боковой грани куба:



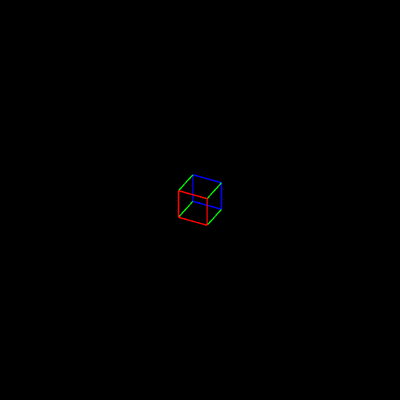
Изометрический вид куба:

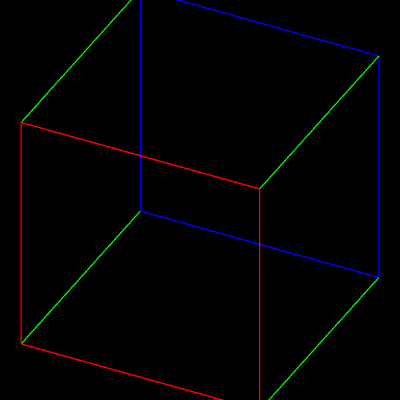


Аксонометрический вид куба:



Панорамное отображение куба и отображение куба крупным планом:





**Задание 4**

1. С помощью перспективной проекции, используя функцию frustum, получить вид куба с 1 главной точкой схождения.

2. С помощью перспективной проекции, используя функцию frustum, получить вид куба с 2 главными точками схождения.

3. С помощью перспективной проекции, используя функцию frustum, получить вид куба с 3 главными точками схождения.

4. С помощью перспективной проекции, используя функцию perspective, получить вид куба с 1 главной точкой схождения.

5. С помощью перспективной проекции, используя функцию perspective, получить вид куба с 2 главными точками схождения.

6. С помощью перспективной проекции, используя функцию perspective, получить вид куба с 3 главными точками схождения.

7. С помощью настройки объема наблюдения и настроек положения камеры, усильте и, наоборот, уменьшите эффект перспективы.

**Решение**

Код файла lab\_07\_4.js:

"use strict";

const VSHADER\_SOURCE =

  '#version 100\n' +

  'precision mediump float;\n' +

  'attribute vec4 a\_Position;\n' +

  'varying vec4 v\_FragColor;\n' +

  'attribute vec4 a\_Color;\n' +

  'uniform mat4 u\_MVP;\n' +

  'void main() {\n' +

  '  gl\_Position = u\_MVP \* a\_Position;\n' +

  '  gl\_PointSize = 10.0;\n' +

  '  v\_FragColor = a\_Color;\n'+

  '}\n';

const FSHADER\_SOURCE =

  'precision mediump float;\n' +

  'varying vec4 v\_FragColor;\n' +

  'void main() {\n' +

  '  gl\_FragColor = v\_FragColor;\n' +

  '}\n';

const {mat2, mat3, mat4, vec2, vec3, vec4} = glMatrix;

function main() {

  const canvas = document.getElementById('webgl');

  const gl = getWebGLContext(canvas);

  if (!gl) {

    console.log('Failed to get the rendering context for WebGL');

    return;

  }

  if (!initShaders(gl, VSHADER\_SOURCE, FSHADER\_SOURCE)) {

    console.log('Failed to intialize shaders.');

    return;

  }

  const n = initVertexBuffers(gl);

  if (n < 0) {

    console.log('Failed to set the positions of the vertices');

    return;

  }

  gl.clearColor(0.0, 0.0, 0.0, 1.0);

  gl.enable(gl.DEPTH\_TEST);

  const controls = {

    view: 'axonometry'

  };

  const gui = new dat.GUI();

  const guiCamera = gui.addFolder('camera');

  let projectionMatrix = mat4.create();

  const projection = {

      perspective: "Orthographic",

      switchCamera: function () {

          if (this.perspective == "Perspective") {

            mat4.ortho(projectionMatrix, -1.0, 1.0, -1.0, 1.0, 0.0, 3.0);

            //mat4.ortho(projectionMatrix, -5.0, 5.0, -5.0, 5.0, 0.0, 3.0);

            //mat4.ortho(projectionMatrix, -0.6, 0.6, -0.6, 0.6, 0.0, 3.0);

            this.perspective = "Orthographic";

            guiCamera.remove(view);

            controls.view = 'axonometry';

            view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

              console.log(e);

              controls.view = e;

            });

          } else {

              mat4.frustum(projectionMatrix, -1, 1.3, -1.3, 1, 1, 6);

              //mat4.perspective(projectionMatrix, 60 \* Math.PI / 180.0, 1 , 1, 6);

              //mat4.perspective(projectionMatrix, 135 \* Math.PI / 180.0, 1 , 0.1, 6)

              //mat4.perspective(projectionMatrix, 20 \* Math.PI / 180.0, 1 , 1, 12)

              this.perspective = "Perspective";

              guiCamera.remove(view);

              controls.view = '3-point';

              view = guiCamera.add(controls, 'view', ['1-point', '2-point', '3-point']).onChange(function (e) {

                  console.log(e);

                  controls.view = e;

              });

          }

      }

  };

  guiCamera.add(projection, 'switchCamera');

  guiCamera.add(projection, 'perspective').listen();

  let view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

      console.log(e);

      controls.view = e;

  });

  let eye = vec3.create();

  let up = vec3.create();

  function render() {

    switch (controls.view) {

        case 'left':

          vec3.set(eye, -0.5, 0.0, 0.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'right':

          vec3.set(eye, 0.5, 0.0, 0.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'top':

          vec3.set(eye, 0.0, 0.5, 0.0); vec3.set(up, 0.0, 0.0, -1.0)

        break;

        case 'bottom':

          vec3.set(eye, 0.0, -0.5, 0.0); vec3.set(up, 0.0, 0.0, 1.0)

        break;

        case 'front':

          vec3.set(eye, 0.0, 0.0, 0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'back':

          vec3.set(eye, 0.0, 0.0, -0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'isometry':

          vec3.set(eye, 0.5, 0.5, 0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'axonometry':

          vec3.set(eye, 0.5, 0.7, 0.9); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case '1-point':

          vec3.set(eye, 0.0, 0.0, 1.5); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 0.0, 0.0, 0.9); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 0.0, 0.0, 8.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case '2-point':

          vec3.set(eye, 1.2, 0.0, 1.2); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 0.6, 0.0, 0.6); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 6.0, 0.0, 6.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case '3-point':

          vec3.set(eye, 1.0, 1.0, 1.0); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 0.5, 0.5, 0.5); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 5.0, 5.0, 5.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

    }

    const u\_MVP = gl.getUniformLocation(gl.program, 'u\_MVP');

    if (u\_MVP < 0) {

      console.log('Failed to get the storage location of u\_MVP');

      return -1;

    }

    let MVP\_Matrix = mat4.create();

    let viewMatrix = mat4.create();

    mat4.lookAt(viewMatrix, eye, [0.0, 0.0, 0.0], up);

    mat4.multiply(MVP\_Matrix, projectionMatrix, viewMatrix);

    gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

    gl.uniformMatrix4fv(u\_MVP, 0, MVP\_Matrix);

    gl.drawArrays(gl.LINES, 0, n);

    requestAnimationFrame(render);

  }

  render();

}

function initVertexBuffers(gl) {

  const vertices = new Float32Array([

  0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, -0.4, 0.4,

  -0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, 0.4, 0.4,

  0.4, 0.4, 0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, 0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, 0.4,

  -0.4, -0.4, -0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, -0.4,

  -0.4, -0.4, -0.4,

  0.4, -0.4, -0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,]);

  const Color = new Float32Array([

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  1.0, 0.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0

  ]);

  const n = vertices.length;

  const vertexBuffer = gl.createBuffer();

  if (!vertexBuffer) {

    console.log('Failed to create the buffer object');

    return -1;

  }

  gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

  var FSIZE = vertices.BYTES\_PER\_ELEMENT;

  gl.bufferData(gl.ARRAY\_BUFFER, 6 \* n \* FSIZE, gl.STATIC\_DRAW);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 0, vertices);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 3 \* n \* FSIZE, Color);

  const a\_Position = gl.getAttribLocation(gl.program, 'a\_Position');

  if (a\_Position < 0) {

    console.log('Failed to get the storage location of a\_Position');

    return -1;

  }

  const a\_Color = gl.getAttribLocation(gl.program, 'a\_Color');

  if (!a\_Color) {

    console.log('Failed to get the storage location of a\_Color');

    return;

  }

  gl.vertexAttribPointer(a\_Position, 3, gl.FLOAT, false, 0, 0);

  gl.vertexAttribPointer(a\_Color, 3, gl.FLOAT, false, 0, FSIZE \* 3 \* n);

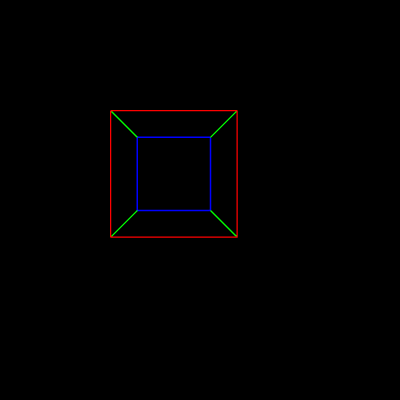
  gl.enableVertexAttribArray(a\_Position);

  gl.enableVertexAttribArray(a\_Color);

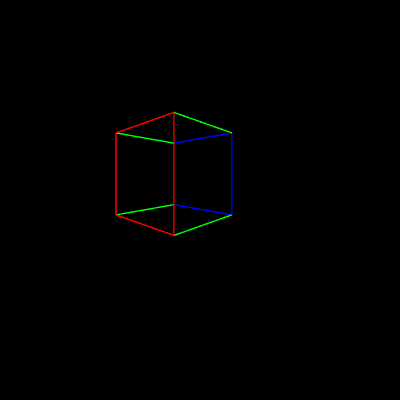
  return n;

}

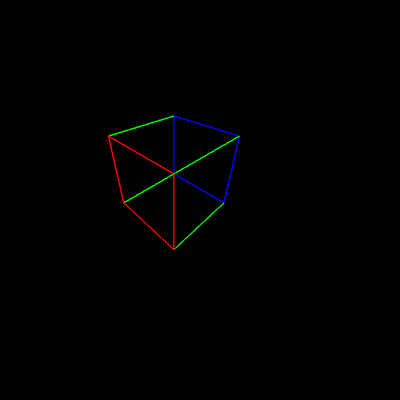
Вид куба с 1 главной точкой схождение, полученный с помощью функции frustum:



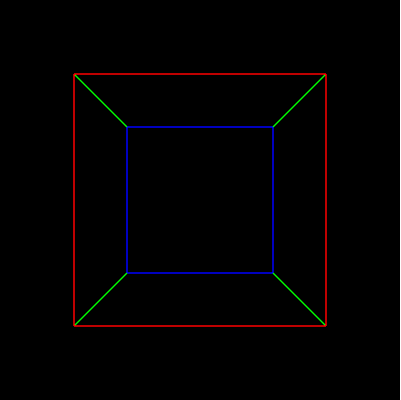
Вид куба с 2 главными точками схождение, полученный с помощью функции frustum:



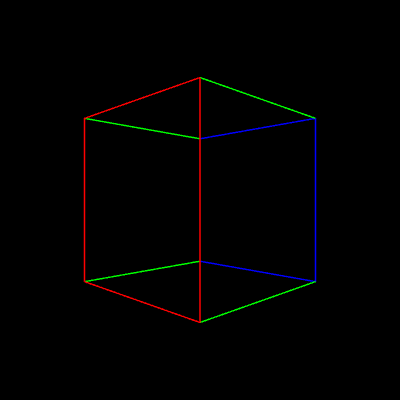
Вид куба с 3 главными точками схождение, полученный с помощью функции frustum:



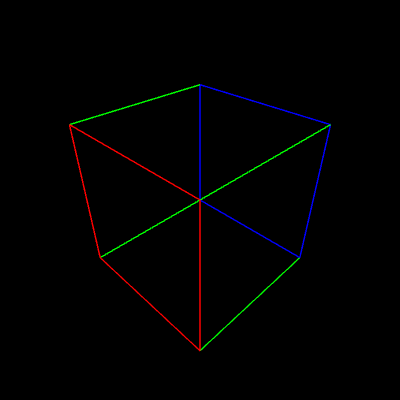
Вид куба с 1 главной точкой схождение, полученный с помощью функции perspective:



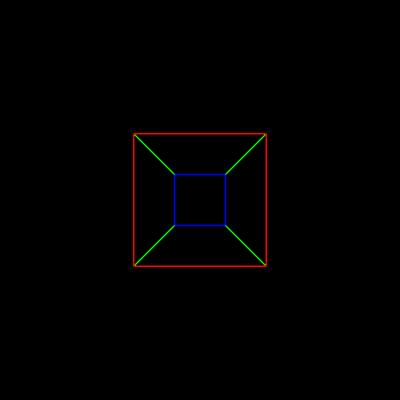
Вид куба с 2 главными точками схождение, полученный с помощью функции perspective:

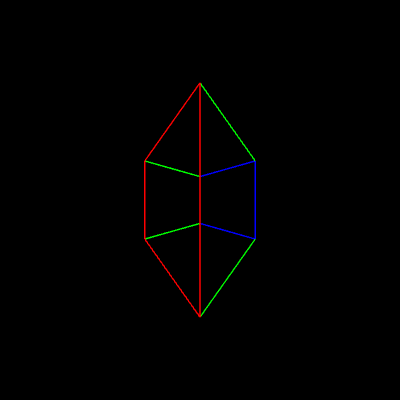


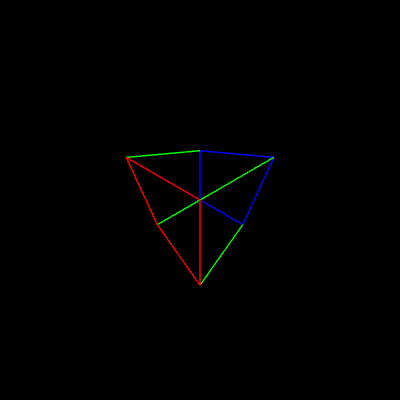
Вид куба с 3 главными точками схождение, полученный с помощью функции perspective:



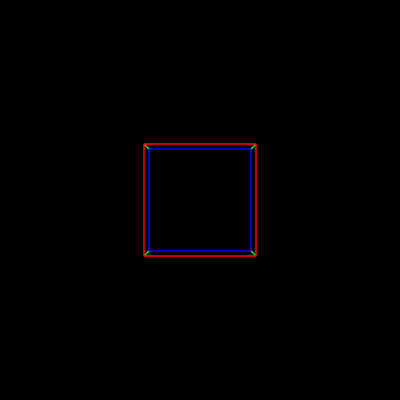
Усилиенный эффект перспективы:

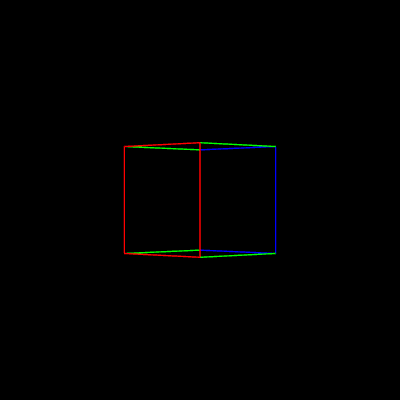


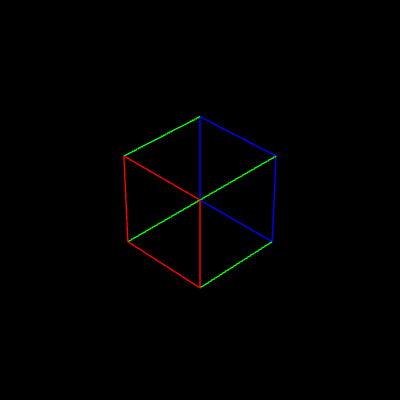




Уменьшеный эффект перспективы:







**Задание 5**

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

**Решение**

Составим таблицы вершин, ребер и граней.

|  |  |
| --- | --- |
| Таблица вершин | |
| Номер вершины | Координаты вершины |
| 0 | 0.4, 0.4, 0.4 |
| 1 | -0.4, 0.4, 0.4 |
| 2 | -0.4, -0.4, 0.4 |
| 3 | 0.4, -0.4, 0.4 |
| 4 | 0.4, -0.4, -0.4 |
| 5 | 0.4, 0.4, -0.4 |
| 6 | -0.4, 0.4, -0.4 |
| 7 | -0.4, -0.4, -0.4 |

|  |  |
| --- | --- |
| Таблица ребер | |
| Номер ребра | Номера вершин |
| 0 | 0, 1 |
| 1 | 1, 2 |
| 2 | 2, 3 |
| 3 | 3, 0 |
| 4 | 0, 5 |
| 5 | 1, 6 |
| 6 | 2, 7 |
| 7 | 3, 4 |
| 8 | 5, 6 |
| 9 | 6, 7 |
| 10 | 7, 4 |
| 11 | 4, 5 |

|  |  |
| --- | --- |
| Таблица граней | |
| Номер грани | Номера вершин |
| 0 | 0, 1, 2 |
| 1 | 0, 2, 3 |
| 2 | 0, 3, 4 |
| 3 | 0, 4, 5 |
| 4 | 0, 5, 6 |
| 5 | 0, 6, 1 |
| 6 | 7, 6, 5 |
| 7 | 7, 5, 4 |
| 8 | 7, 2, 1 |
| 9 | 7, 1, 6 |
| 10 | 7, 4, 3 |
| 11 | 7, 3, 2 |

Создадим соответсвующие массивы.

  const vertices = new Float32Array([

  0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, -0.4]);

  const edges = new Uint8Array([

  0, 1,

  1, 2,

  2, 3,

  3, 0,

  0, 5,

  1, 6,

  2, 7,

  3, 4,

  5, 6,

  6, 7,

  7, 4,

  4, 5]);

  const facets = new Uint8Array([

  0, 1, 2,

  0, 2, 3,

  0, 3, 4,

  0, 4, 5,

  0, 5, 6,

  0, 6, 1,

  7, 6, 5,

  7, 5, 4,

  7, 2, 1,

  7, 1, 6,

  7, 4, 3,

  7, 3, 2]);

**Задание 6**

1. С помощью функции gl.drawElements() построить каркасную модель куба.

2. С помощью функции gl.drawElements() построить твердотельную модель куба.

**Решение**

Код файла lab\_07\_6.js:

"use strict";

const VSHADER\_SOURCE =

  '#version 100\n' +

  'precision mediump float;\n' +

  'attribute vec4 a\_Position;\n' +

  'varying vec4 v\_FragColor;\n' +

  'attribute vec4 a\_Color;\n' +

  'uniform mat4 u\_MVP;\n' +

  'void main() {\n' +

  '  gl\_Position = u\_MVP \* a\_Position;\n' +

  '  gl\_PointSize = 10.0;\n' +

  '  v\_FragColor = a\_Color;\n'+

  '}\n';

const FSHADER\_SOURCE =

  'precision mediump float;\n' +

  'varying vec4 v\_FragColor;\n' +

  'void main() {\n' +

  '  gl\_FragColor = v\_FragColor;\n' +

  '}\n';

const {mat2, mat3, mat4, vec2, vec3, vec4} = glMatrix;

function main() {

  const canvas = document.getElementById('webgl');

  const gl = getWebGLContext(canvas);

  if (!gl) {

    console.log('Failed to get the rendering context for WebGL');

    return;

  }

  if (!initShaders(gl, VSHADER\_SOURCE, FSHADER\_SOURCE)) {

    console.log('Failed to intialize shaders.');

    return;

  }

  const n = initVertexBuffers(gl);

  if (n < 0) {

    console.log('Failed to set the positions of the vertices');

    return;

  }

  gl.clearColor(0.0, 0.0, 0.0, 1.0);

  gl.enable(gl.DEPTH\_TEST);

  const controls = {

    view: 'axonometry'

  };

  const gui = new dat.GUI();

  const guiCamera = gui.addFolder('camera');

  let projectionMatrix = mat4.create();

  const projection = {

      perspective: "Orthographic",

      switchCamera: function () {

          if (this.perspective == "Perspective") {

            mat4.ortho(projectionMatrix, -1.0, 1.0, -1.0, 1.0, 0.0, 3.0);

            //mat4.ortho(projectionMatrix, -5.0, 5.0, -5.0, 5.0, 0.0, 3.0);

            //mat4.ortho(projectionMatrix, -0.6, 0.6, -0.6, 0.6, 0.0, 3.0);

            this.perspective = "Orthographic";

            guiCamera.remove(view);

            controls.view = 'axonometry';

            view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

              console.log(e);

              controls.view = e;

            });

          } else {

              //mat4.frustum(projectionMatrix, -1, 1.3, -1.3, 1, 1, 6);

              mat4.perspective(projectionMatrix, 60 \* Math.PI / 180.0, 1 , 1, 6);

              //mat4.perspective(projectionMatrix, 135 \* Math.PI / 180.0, 1 , 0.1, 6)

              //mat4.perspective(projectionMatrix, 20 \* Math.PI / 180.0, 1 , 1, 12)

              this.perspective = "Perspective";

              guiCamera.remove(view);

              controls.view = '3-point';

              view = guiCamera.add(controls, 'view', ['1-point', '2-point', '3-point']).onChange(function (e) {

                  console.log(e);

                  controls.view = e;

              });

          }

      }

  };

  guiCamera.add(projection, 'switchCamera');

  guiCamera.add(projection, 'perspective').listen();

  let view = guiCamera.add(controls, 'view', ['left', 'right', 'top', 'bottom', 'front', 'back', 'isometry', 'axonometry']).onChange(function (e) {

      console.log(e);

      controls.view = e;

  });

  let eye = vec3.create();

  let up = vec3.create();

  function render() {

    switch (controls.view) {

        case 'left':

          vec3.set(eye, -0.5, 0.0, 0.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'right':

          vec3.set(eye, 0.5, 0.0, 0.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'top':

          vec3.set(eye, 0.0, 0.5, 0.0); vec3.set(up, 0.0, 0.0, -1.0)

        break;

        case 'bottom':

          vec3.set(eye, 0.0, -0.5, 0.0); vec3.set(up, 0.0, 0.0, 1.0)

        break;

        case 'front':

          vec3.set(eye, 0.0, 0.0, 0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'back':

          vec3.set(eye, 0.0, 0.0, -0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'isometry':

          vec3.set(eye, 0.5, 0.5, 0.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case 'axonometry':

          vec3.set(eye, 0.5, 0.7, 0.9); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case '1-point':

          vec3.set(eye, 0.0, 0.0, 1.5); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 0.0, 0.0, 0.9); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 0.0, 0.0, 8.5); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case '2-point':

          vec3.set(eye, 1.2, 0.0, 1.2); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 0.6, 0.0, 0.6); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 6.0, 0.0, 6.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

        case '3-point':

          vec3.set(eye, 1.0, 1.0, 1.0); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 0.5, 0.5, 0.5); vec3.set(up, 0.0, 1.0, 0.0)

          //vec3.set(eye, 5.0, 5.0, 5.0); vec3.set(up, 0.0, 1.0, 0.0)

        break;

    }

    const u\_MVP = gl.getUniformLocation(gl.program, 'u\_MVP');

    if (u\_MVP < 0) {

      console.log('Failed to get the storage location of u\_MVP');

      return -1;

    }

    let MVP\_Matrix = mat4.create();

    let viewMatrix = mat4.create();

    mat4.lookAt(viewMatrix, eye, [0.0, 0.0, 0.0], up);

    mat4.multiply(MVP\_Matrix, projectionMatrix, viewMatrix);

    gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

    gl.uniformMatrix4fv(u\_MVP, 0, MVP\_Matrix);

    gl.drawElements(gl.LINES, n, gl.UNSIGNED\_BYTE, 0);

    //gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED\_BYTE, 0);

    requestAnimationFrame(render);

  }

  render();

}

function initVertexBuffers(gl) {

  const vertices = new Float32Array([

  0.4, 0.4, 0.4,

  -0.4, 0.4, 0.4,

  -0.4, -0.4, 0.4,

  0.4, -0.4, 0.4,

  0.4, -0.4, -0.4,

  0.4, 0.4, -0.4,

  -0.4, 0.4, -0.4,

  -0.4, -0.4, -0.4]);

  const edges = new Uint8Array([

  0, 1,

  1, 2,

  2, 3,

  3, 0,

  0, 5,

  1, 6,

  2, 7,

  3, 4,

  5, 6,

  6, 7,

  7, 4,

  4, 5]);

  const facets = new Uint8Array([

  0, 1, 2,

  0, 2, 3,

  0, 3, 4,

  0, 4, 5,

  0, 5, 6,

  0, 6, 1,

  7, 6, 5,

  7, 5, 4,

  7, 2, 1,

  7, 1, 6,

  7, 4, 3,

  7, 3, 2]);

  const Color = new Float32Array([

  1.0, 0.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  1.0, 0.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 1.0, 0.0,

  0.0, 0.0, 1.0,

  0.0, 0.0, 1.0]);

  const n = edges.length;

  //const n = facets.length;

  const vertexBuffer = gl.createBuffer();

  if (!vertexBuffer) {

    console.log('Failed to create the buffer object');

    return -1;

  }

  gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

  var FSIZE = vertices.BYTES\_PER\_ELEMENT;

  gl.bufferData(gl.ARRAY\_BUFFER, 2 \* vertices.length \* FSIZE, gl.STATIC\_DRAW);

  gl.bufferSubData(gl.ARRAY\_BUFFER, 0, vertices);

  gl.bufferSubData(gl.ARRAY\_BUFFER, vertices.length \* FSIZE, Color);

  const indexBuffer = gl.createBuffer();

  if (!indexBuffer) {

    console.log('Failed to create the buffer object');

    return -1;

  }

  gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, indexBuffer);

  gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, edges, gl.STATIC\_DRAW);

  //gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, facets, gl.STATIC\_DRAW);

  const a\_Position = gl.getAttribLocation(gl.program, 'a\_Position');

  if (a\_Position < 0) {

    console.log('Failed to get the storage location of a\_Position');

    return -1;

  }

  const a\_Color = gl.getAttribLocation(gl.program, 'a\_Color');

  if (!a\_Color) {

    console.log('Failed to get the storage location of a\_Color');

    return;

  }

  gl.vertexAttribPointer(a\_Position, 3, gl.FLOAT, false, 0, 0);

  gl.vertexAttribPointer(a\_Color, 3, gl.FLOAT, false, 0, FSIZE \* vertices.length);

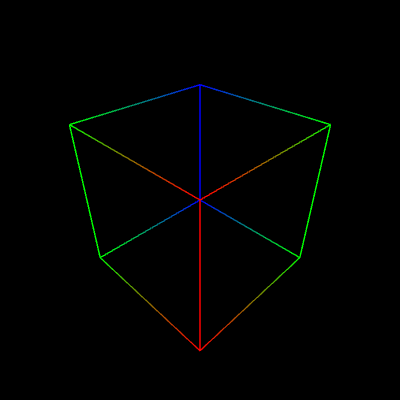
  gl.enableVertexAttribArray(a\_Position);

  gl.enableVertexAttribArray(a\_Color);

  return n;

}

Каркасная модель куба:



Твердотельная модель куба:

