**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа №2**

Тема: Каркасная визуализация выпуклого многогранника. Удаление невидимых линий.

Студент: Алексеев Владислав Евгеньевич

Группа: М8О-306Б-19

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Разработать формат представления многогранника и процедуру его каркасной отрисовки в ортографической и изометрической проекциях. Обеспечить удаление невидимых линий и возможность пространственных поворотов и масштабирования многогранника. Обеспечить автоматическое центрирование и изменение размеров изображения при изменении размеров окна.

**Вариант многогранника**: Шестигранная прямая правильная призма.

1. **Описание программы**

Программа состоит из окна для взаимодействия с многогранником, из конваса для его визуализации и для вращения многогранника.

1. **Набор тестов**

1. Изначальный вид.

2. Вращение многогранника.

3. Увеличение масштаба.

4. Уменьшение масштаба.

1. **Результаты выполнения тестов**

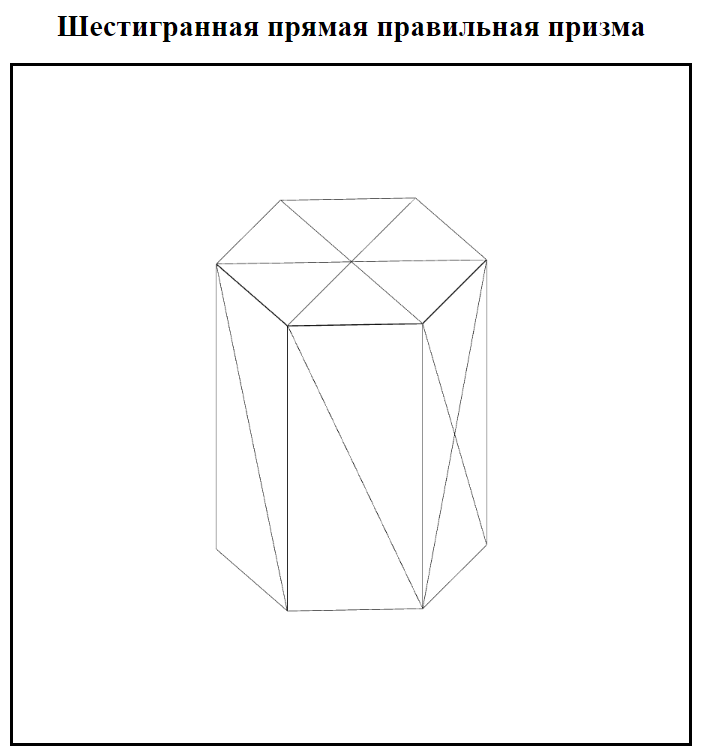
1. Изначальный вид.  


Рис. 4.1 Результат открытия окна

2. Вращение многогранника.

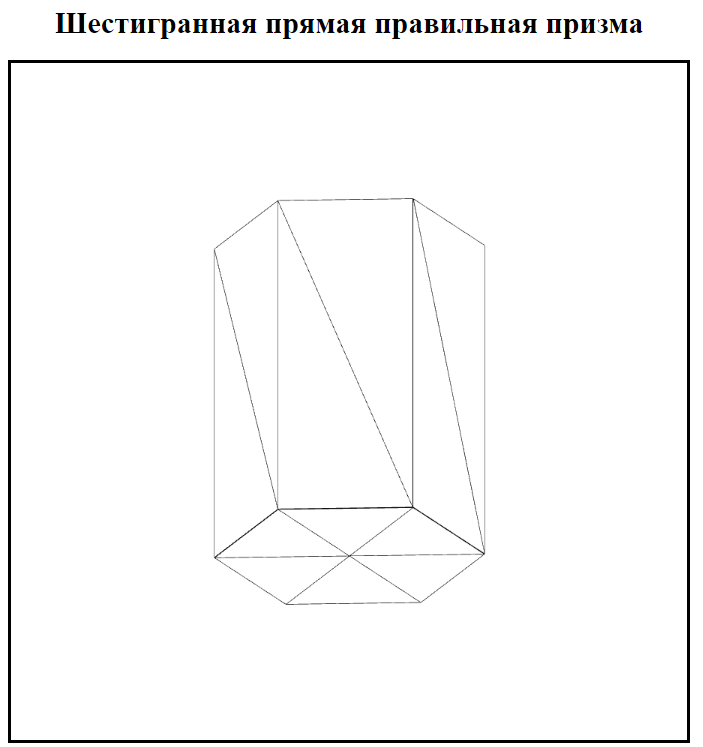


Рис. 4.2 Результат вращения многогранника

3. Увеличение масштаба.

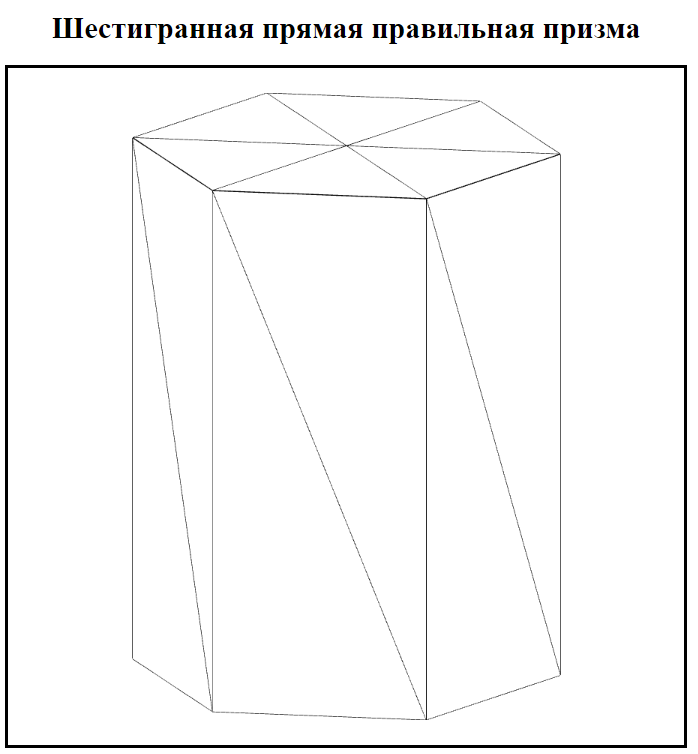


Рис. 4.3 Результат увеличения масштаба

4. Уменьшение масштаба.

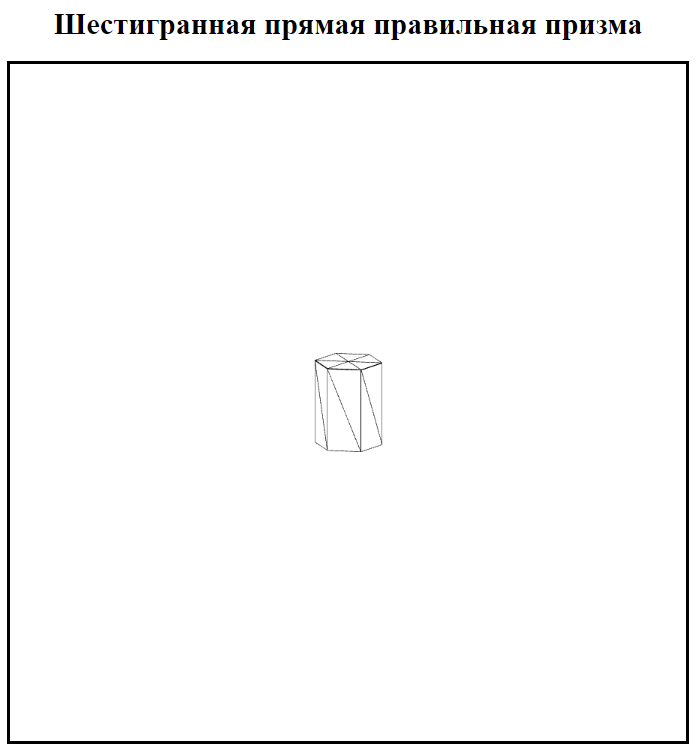


Рис. 4.4 Результат уменьшения масштаба

1. **Листинг программы**

**index.html**

<!doctype html>

<html lang = "ru">

<head>

<meta charset = "utf-8"/>

<title></title>

<link rel = "stylesheet" href = "style.css"/>

</head>

<body>

<div class = "parent">

<div class = "child">

<p>Лабораторная работа №2

<br>Вариант №9

<br>Алексеев Владислав

<br>Группа М8О-306Б-19

</p>

</div>

<div class = "child" id = "canvas">

<h1 id = "header">Шестигранная прямая правильная призма</h1>

<canvas id = "prism" height = "900" width = "900"></canvas>

</div>

</div>

<script id = "vertex-shader-2d" type = "x-shader/x-vertex">

attribute vec4 a\_position;

uniform mat4 u\_matrix;

uniform vec3 u\_Color;

varying vec3 v\_Color;

void main() {

v\_Color = u\_Color;

gl\_Position = u\_matrix\*a\_position;

}

</script>

<script id = "fragment-shader-2d" type = "x-shader/x-fragment">

precision mediump float;

uniform vec4 u\_FragColor;

varying vec3 v\_Color;

void main() {

gl\_FragColor = vec4(v\_Color,1.0);

}

</script>

<script type = "text/javascript" src = "solution.js"></script>

</body>

</html>

**style.css**

#header {

text-align: center;

}

#prism {

display: block;

margin: 0 auto;

border: 5px solid black;

}

input {

margin: 0 auto;

display: block;

}

.child {

font-size:20px;

display: inline-block;

}

#canvas {

margin: 0 auto;

display: block;

}

**solution.js**

function main() {

let canvas = document.getElementById("prism");

let gl = canvas.getContext("webgl");

let program = createProgram(gl);

let positionLocation = gl.getAttribLocation(program, "a\_position");

let matrixLocation = gl.getUniformLocation(program, "u\_matrix");

let colorLocation = gl.getUniformLocation(program,'u\_Color');

gl.enable(gl.DEPTH\_TEST);

let positionBuffer = gl.createBuffer();

let rotation = [-Math.PI/2, 0, 0];

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

setVertex(gl);

let triangleBuffer= gl.createBuffer();

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

setGeometry(gl);

let drag = false;

let old\_x; let old\_y;

let rot\_x = 45.; let rot\_y = 12.;

let scale = 1;

let mouseDownHandler = function (e) { // Обработчик нажатия мыши

drag = true;

old\_x = e.pageX;

old\_y = e.pageY;

e.preventDefault();

}

let mouseUpHandler = function (e) { // Обработчик мыши вверх

drag = false;

e.preventDefault();

}

let mouseMoveHandler = function (e) { // Обработчик перемещения мыши

if (!drag) {

return false;

};

let newX = old\_x - e.pageX;

let newY = old\_y - e.pageY;

rot\_x += newX \* 2 \* Math.PI / canvas.width;

rot\_y += newY \* 2 \* Math.PI / canvas.height;

old\_x = e.pageX;

old\_y = e.pageY;

e.preventDefault();

}

let wheelHandler = function (e) { // Масштабирование

scale += e.deltaY/1000;

e.preventDefault();

}

canvas.addEventListener("mousedown", mouseDownHandler, false); // Мышь вниз

canvas.addEventListener("mouseup", mouseUpHandler, false); // Мышь вверх

canvas.addEventListener("mouseout", mouseUpHandler, false); // Вывод

canvas.addEventListener("mousemove", mouseMoveHandler, false); // Движение мыши

canvas.addEventListener('wheel',wheelHandler, false); // Колёсико

let drawScene = function (time) { // Прорисовка

rotation=[rot\_y, rot\_x, 0];

let matrix = m4.identity();

matrix = m4.scale(matrix, scale, scale, scale);

matrix = m4.xRotate(matrix, rotation[0]);

matrix = m4.yRotate(matrix, rotation[1]);

matrix = m4.zRotate(matrix, rotation[2]);

resize(gl.canvas);

gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

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

gl.useProgram(program);

gl.enableVertexAttribArray(positionLocation);

gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 0, 0);

gl.uniform3f(colorLocation, 1, 1, 1);

gl.uniformMatrix4fv(matrixLocation, false, matrix);

gl.drawElements(gl.TRIANGLES, 72, gl.UNSIGNED\_SHORT, 0 );

gl.uniform3f(colorLocation, 0.0, 0.0, 0.0);

gl.drawElements(gl.LINE\_STRIP, 72, gl.UNSIGNED\_SHORT, 0);

gl.flush();

window.requestAnimationFrame(drawScene);

}

drawScene(0);

}

function createShader(gl, type, source) {

let shader = gl.createShader(type);

gl.shaderSource(shader, source);

gl.compileShader(shader);

let success = gl.getShaderParameter(shader, gl.COMPILE\_STATUS);

if (success) {

return shader;

}

console.log(gl.getShaderInfoLog(shader));

gl.deleteShader(shader);

alert("Не удалось создать шейдер");

}

function createProgram(gl) {

let program = gl.createProgram();

let vertexShaderSource = document.querySelector("#vertex-shader-2d").text;

let fragmentShaderSource = document.querySelector("#fragment-shader-2d").text;

let vertexShader = createShader(gl, gl.VERTEX\_SHADER, vertexShaderSource);

let fragmentShader = createShader(gl, gl.FRAGMENT\_SHADER, fragmentShaderSource);

gl.attachShader(program, vertexShader);

gl.attachShader(program, fragmentShader);

gl.linkProgram(program);

if (gl.getProgramParameter(program, gl.LINK\_STATUS)) {

return program;

}

console.log(gl.getProgramInfoLog(program));

gl.deleteProgram(program);

alert("Не удалось связать Шейдеры");

}

function setVertex(gl) { // Установление вершин

let array = new Float32Array([

0,-0.5, 0, // 0

0.4, -0.5, 0, // 1

0.4 \* Math.cos(Math.PI/3), -0.5, 0.4 \* Math.sin(Math.PI/3), // 2

-0.4, -0.5, 0, // 3

-0.4 \* Math.cos(Math.PI/3), -0.5, 0.4 \* Math.sin(Math.PI/3), // 4

0.4 \* Math.cos(Math.PI/3), -0.5, -0.4 \* Math.sin(Math.PI/3), // 5

-0.4 \* Math.cos(Math.PI/3), -0.5, -0.4 \* Math.sin(Math.PI/3), // 6

0, 0.5, 0, // 7

0.4, 0.5, 0, // 8

0.4 \* Math.cos(Math.PI/3), 0.5, 0.4 \* Math.sin(Math.PI/3), // 9

-0.4, 0.5, 0, // 10

-0.4 \* Math.cos(Math.PI/3), 0.5, 0.4 \* Math.sin(Math.PI/3), // 11

0.4 \* Math.cos(Math.PI/3), 0.5, -0.4 \* Math.sin(Math.PI/3), // 12

-0.4 \* Math.cos(Math.PI/3), 0.5, -0.4 \* Math.sin(Math.PI/3), // 13

]);

gl.bufferData(gl.ARRAY\_BUFFER, array, gl.STATIC\_DRAW);

}

function setGeometry(gl) {

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array([

// Нижнее основание

0, 2, 1,

2, 4, 0,

4, 3, 0,

6, 3, 0,

6, 5, 0,

5, 1, 0,

// Боковые грани

1, 2, 8,

8, 2, 9,

2, 4, 9,

9, 4, 11,

4, 3, 11,

11, 3, 10,

3, 6, 10,

10, 13, 6,

6, 5, 13,

13, 12, 5,

5, 1, 12,

12, 8, 1,

// Верхняя грань

9, 8, 7,

9, 11, 7,

11, 10, 7,

13, 10, 7,

13, 12, 7,

12, 8, 7,

]), gl.STATIC\_DRAW);

}

function resize(canvas) {

// Получаем размер canvas

let displayWidth = canvas.clientWidth;

let displayHeight = canvas.clientHeight;

// Проверяем, отличается ли размер canvas

if (canvas.width != displayWidth || canvas.height != displayHeight) {

// Подгоняем размер буфера отрисовки под размер HTML-элемента

canvas.width = displayWidth;

canvas.height = displayHeight;

}

}

let m4 = {

xRotate: function(m, angleInRadians) {

return m4.multiply(m, m4.xRotation(angleInRadians));

},

yRotate: function(m, angleInRadians) {

return m4.multiply(m, m4.yRotation(angleInRadians));

},

zRotate: function(m, angleInRadians) {

return m4.multiply(m, m4.zRotation(angleInRadians));

},

translate: function(m, tx, ty, tz) {

return m4.multiply(m, m4.translation(tx, ty, tz));

},

scale: function(m, sx, sy, sz) {

return m4.multiply(m, m4.scaling(sx, sy, sz));

},

xRotation: function(angleInRadians) {

let c = Math.cos(angleInRadians);

let s = Math.sin(angleInRadians);

return [

1, 0, 0, 0,

0, c, s, 0,

0, -s, c, 0,

0, 0, 0, 1,

];

},

scaling: function(sx, sy, sz) {

return [

sx, 0, 0, 0,

0, sy, 0, 0,

0, 0, sz, 0,

0, 0, 0, 1,

];

},

translation: function(tx, ty, tz) {

return [

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

tx, ty, tz, 1,

];

},

yRotation: function(angleInRadians) {

var c = Math.cos(angleInRadians);

var s = Math.sin(angleInRadians);

return [

c, 0, -s, 0,

0, 1, 0, 0,

s, 0, c, 0,

0, 0, 0, 1,

];

},

zRotation: function(angleInRadians) {

var c = Math.cos(angleInRadians);

var s = Math.sin(angleInRadians);

return [

c, s, 0, 0,

-s, c, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1,

];

},

multiply: function(a, b) {

let a00 = a[0 \* 4 + 0];

let a01 = a[0 \* 4 + 1];

let a02 = a[0 \* 4 + 2];

let a03 = a[0 \* 4 + 3];

let a10 = a[1 \* 4 + 0];

let a11 = a[1 \* 4 + 1];

let a12 = a[1 \* 4 + 2];

let a13 = a[1 \* 4 + 3];

let a20 = a[2 \* 4 + 0];

let a21 = a[2 \* 4 + 1];

let a22 = a[2 \* 4 + 2];

let a23 = a[2 \* 4 + 3];

let a30 = a[3 \* 4 + 0];

let a31 = a[3 \* 4 + 1];

let a32 = a[3 \* 4 + 2];

let a33 = a[3 \* 4 + 3];

let b00 = b[0 \* 4 + 0];

let b01 = b[0 \* 4 + 1];

let b02 = b[0 \* 4 + 2];

let b03 = b[0 \* 4 + 3];

let b10 = b[1 \* 4 + 0];

let b11 = b[1 \* 4 + 1];

let b12 = b[1 \* 4 + 2];

let b13 = b[1 \* 4 + 3];

let b20 = b[2 \* 4 + 0];

let b21 = b[2 \* 4 + 1];

let b22 = b[2 \* 4 + 2];

let b23 = b[2 \* 4 + 3];

let b30 = b[3 \* 4 + 0];

let b31 = b[3 \* 4 + 1];

let b32 = b[3 \* 4 + 2];

let b33 = b[3 \* 4 + 3];

return [

b00 \* a00 + b01 \* a10 + b02 \* a20 + b03 \* a30,

b00 \* a01 + b01 \* a11 + b02 \* a21 + b03 \* a31,

b00 \* a02 + b01 \* a12 + b02 \* a22 + b03 \* a32,

b00 \* a03 + b01 \* a13 + b02 \* a23 + b03 \* a33,

b10 \* a00 + b11 \* a10 + b12 \* a20 + b13 \* a30,

b10 \* a01 + b11 \* a11 + b12 \* a21 + b13 \* a31,

b10 \* a02 + b11 \* a12 + b12 \* a22 + b13 \* a32,

b10 \* a03 + b11 \* a13 + b12 \* a23 + b13 \* a33,

b20 \* a00 + b21 \* a10 + b22 \* a20 + b23 \* a30,

b20 \* a01 + b21 \* a11 + b22 \* a21 + b23 \* a31,

b20 \* a02 + b21 \* a12 + b22 \* a22 + b23 \* a32,

b20 \* a03 + b21 \* a13 + b22 \* a23 + b23 \* a33,

b30 \* a00 + b31 \* a10 + b32 \* a20 + b33 \* a30,

b30 \* a01 + b31 \* a11 + b32 \* a21 + b33 \* a31,

b30 \* a02 + b31 \* a12 + b32 \* a22 + b33 \* a32,

b30 \* a03 + b31 \* a13 + b32 \* a23 + b33 \* a33,

];

},

identity: function () {

return [

1, 0, 0, 0,

0, 1, 0, 0,

0, 0, 1, 0,

0, 0, 0, 1,

]

}

};

main()

**6. Выводы**

Выполнив данную лабораторную работу, я научился пользоваться HTML, запускать графический интерфейс, перерисовывать его в зависимости от изменения окна и отрисовывать график по массиву точек.

**ЛИТЕРАТУРА**

1. Работа VS code с HTML[Электронный ресурс]URL: [https://blog.altuninvv.ru/программирование/visual-studio-code/98-настройка-vscode-для-работы-с-html](https://blog.altuninvv.ru/%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/visual-studio-code/98-%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B9%D0%BA%D0%B0-vscode-%D0%B4%D0%BB%D1%8F-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B-%D1%81-html) (Дата обращения: 04.10.2021).

2. Документация HTML[Электронный ресурс]URL: <https://htmlacademy.ru/blog/boost/tutorial/canvas-chart> (Дата обращения: 04.10.2021).