**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Компьютерная графика»

**Лабораторная работа № 3**

Тема: Основы построения фотореалистичных изображений

Студент: Алексеев Владислав Евгеньевич

Группа: М8О-306Б-19

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Используя результаты Л.Р.№2, аппроксимировать заданное тело выпуклым многогранником. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света.

Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант многогранника: Прямой эллиптический цилиндр.

1. **Описание программы**

Программа состоит из окна для взаимодействия с многогранником, из canvas для визуализации, вращения многогранника и изменения некоторых его параметров.

1. **Набор тестов**

1. Изначальный вид.

2. Вращение многогранника.

3. Увеличение точности аппроксимации.

4. Увеличение коэффициента диффузного отражения рассеянного света.

5. Увеличение коэффициента диффузного отражения падающего света.

1. **Результаты выполнения тестов**

1. Изначальный вид.  


Рис. 4.1 Результат открытия окна

2. Вращение многогранника.



Рис. 4.2 Результат вращения многогранника

3. Увеличение точности аппроксимации.



Рис. 4.3 Результат увеличение точности аппроксимации

4. Увеличение коэффициента диффузного отражения рассеянного света.



Рис. 4.4 Результат Увеличение коэффициента диффузного отражения рассеянного света

5. Увеличение коэффициента диффузного отражения падающего света.



Рис. 4.5 Результат Увеличение коэффициента диффузного падающего света

1. **Листинг программы**

**main.html**

<!-- Алексеев В.Е. М8О-306Б-19

Вариант: Прямой эллиптический цилиндр. -->

<!DOCTYPE html>

<html>

<head>

<meta charset = "utf-8">

<title>Elliptical cylinder</title>

<link rel="stylesheet" href = "styles.css">

</head>

<body>

<div class = 'canvas-wrapper'>

<canvas id = "sol\_canvas" width = "1000" height = "1000"></canvas>

</div>

<div class = "note">

<div>

<label for = "approx">Точность аппроксимации</label>

<div class = 'range'>

<input type = "range" id = "approx" name = "volume"

min = "6" max = "50" step = "1" value = "6">

</div>

<label for = "KSLI">Коэффициент диффузного отражения рассеянного света (Ka)</label>

<div class = 'range'>

<input type = "range" id = "KSLI" name = "volume"

min = "0" max = "1" step = "0.01" value = "0.15">

</div>

<label for = "KSFI">Коэффициент диффузного отражения падающего света (Kd)</label>

<div class = 'range'>

<input type = "range" id = "KSFI" name = "volume"

min = "0" max = "1" step = "0.01" value = "0.15">

</div>

</div>

</div>

<script src = "vector.js"></script>

<script src = "solution.js"></script>

</body>

</html>

**style.css**

body {

background-color: white;

padding-top: 50px;

display: flex;

align-items: center;

justify-content: space-around;

}

.canvas-wrapper {

position: relative;

top: 200px;

left: 100px;

}

canvas {

display: block;

background-color: white;

}

.range {

margin: 10px 0 20px;

}

.range input {

width: 100%;

}

.note {

color: black;

font-family: sans-serif;

font-size: 30px;

border: 5px solid black;

width: 500px;

height: 500px;

padding: 50px;

box-sizing: border-box;

display: flex;

text-align: center;

align-items: center;

}

**vector.js**

'use strict';

class Vector {

constructor(x, y, z) {

this[0] = x;

this[1] = y;

this[2] = z;

}

cross(other) {

return new Vector(this[1] \* other[2] - this[2] \* other[1],

this[0] \* other[2] - this[2] \* other[0],

this[0] \* other [1] - this[1] \* other[0]);

}

add(other) {

return new Vector(this[0] + other[0], this[1] + other[1], this[2] + other[2]);

}

scale(par) {

return new Vector( par \* this[0], par \* this[1], par \* this[2]);

}

subtract(other) {

return this.add(other.scale(-1));

}

rotateY(fi) {

var x = this[0], y = this[1], z = this[2];

return new Vector( Math.cos(fi) \* x - Math.sin(fi) \* z, y,

Math.sin(fi) \* x + Math.cos(fi) \* z);

}

rotateX(fi) {

var x = this[0], y = this[1], z = this[2];

return new Vector(x, Math.cos(fi) \* y + Math.sin(fi) \* z,

Math.cos(fi) \* z - Math.sin(fi) \* y);

}

scalar(obser) {

return (obser[0] \* this[0] + obser[1] \* this[1] + obser[2] \* this[2]);

}

}

solution.js

var SCALE = 2; // Коэффициент масштабирования

var MIN\_SCALE = 0.8; // Минимальный коэффициент масштабирования

var MAX\_SCALE = 5; // Максимальный коэффициент масштабирования

const OBSERVER = new Vector(0, 0, 4); // Вектор наблюдателя для удаления невидимых линий

const OBS = new Vector(0, 0, 1); // Единичный вектор наблюдателя для света

const LIGHT = new Vector(1, 1, 1); // Вектор источника света

const A = 2, B = 1, H = 6; // Параметры эллипса

const canvas = document.getElementById("sol\_canvas");

const approx = document.getElementById("approx");

const KSLI = document.getElementById("KSLI");

const KSFI = document.getElementById("KSFI");

const ctx = canvas.getContext("2d");

const canvasWidth = 600;

const canvasHeight = 600;

let NUM = approx.value; // Коэффициент аппроксимации

let Ka = KSLI.value; // Коэффициент диффузного отражения рассеянного света

let Kd = KSFI.value; // Коэффициент диффузного отражения падающего света

let bases = []; // Массив треугольников основания эллиптического цилиндра

let sideFaces = []; // Массив треугольников боковых сторон эллипт.цилиндра

let lastTriangles = []; // Массив для сохранения положения фигур при вращении

// Считывание изменения координат экрана при вращении

canvas.onmousedown = function (e) {

const x = e.clientX, y = e.clientY;

canvas.onmousemove = function (e) {

const moveX = e.clientX, moveY = e.clientY;

const thetaY = (moveX - x) / 100;

const thetaX = (moveY - y) / 100;

rotate(thetaX, thetaY);

}

};

canvas.onmouseup = function (e) {

canvas.onmousemove = null;

bases = lastTriangles;

};

// Реакция на изменение коэффициента аппроксимации

approx.onchange = function (e) {

NUM = approx.value;

initGeom();

render();

}

// Реакция на изменение коэффициента диффузного отражения рассеянного света

KSLI.onchange = function (e) {

Ka = KSLI.value;

render();

}

// Реакция на изменение коэффициента диффузного отражения падающего света

KSFI.onchange = function (e) {

Kd = KSFI.value;

render();

}

// Изменение масштаба с помощью колесика мыши

canvas.addEventListener('wheel', function (e) {

if (e.deltaY < 0) {

if (SCALE > MIN\_SCALE) {

SCALE -= 0.1;

console.log(SCALE)

render();

}

}

if (e.deltaY > 0) {

if (SCALE < MAX\_SCALE) {

SCALE += 0.1;

console.log(SCALE)

render();

}

}

});

// Корни через уравнение эллипса для разделения его на равные части

function fplus(x) {

return Math.sqrt((1 - (x \* x) / (A \* A)) \* B \* B);

}

function fminus(x) {

return -Math.sqrt((1 - (x \* x) / (A \* A)) \* B \* B);

}

function initGeom() {

bases = [];

// Аппроксимируем выпуклое тело треугольниками

var step = 2 \* A / NUM;

// Лицевые треугольники

var z = H / 2;

var prevX = -A, prevY = fplus(prevX);

for (var x = -A + step; Math.floor(x) < A; x += step) {

var y = fminus(x);

bases.push([new Vector(0, 0, z), new Vector(prevX, prevY, z), new Vector(x, y, z)]); // ++++

bases.push([new Vector(x, y, -z), new Vector(x, y, z), new Vector(prevX, prevY, z)]);

prevX = x;

prevY = y;

console.log('done')

}

var x = A, y = fminus(x);

bases.push([new Vector(0, 0, z), new Vector(prevX, prevY, z), new Vector(x, y, z)]);

bases.push([new Vector(x, y, -z), new Vector(x, y, z), new Vector(prevX, prevY, z)]);

prevX = -A, prevY = fminus(prevX);

for (var x = -A + step; x < A; x += step) {

var y = fplus(x);

bases.push([new Vector(x, y, z), new Vector(prevX, prevY, z), new Vector(0, 0, z)]); // ++++

bases.push([new Vector(prevX, prevY, -z), new Vector(prevX, prevY, z), new Vector(x, y, z)]);

prevX = x;

prevY = y;

}

var x = A, y = fplus(x);

bases.push([new Vector(x, y, z), new Vector(prevX, prevY, z), new Vector(0, 0, z)]); // ++++

bases.push([new Vector(prevX, prevY, -z), new Vector(prevX, prevY, z), new Vector(x, y, z)]);

// Задаем треуголники

z = -H / 2;

var prevX = -A, prevY = fplus(prevX);

for (var x = -A + step; x < A; x += step) {

var y = fplus(x);

bases.push([new Vector(x, y, z), new Vector(0, 0, z), new Vector(prevX, prevY, z)]); // ++++

bases.push([new Vector(x, y, z), new Vector(prevX, prevY, z), new Vector(x, y, -z)]);

prevX = x;

prevY = y;

}

var x = A, y = fplus(x);

bases.push([new Vector(x, y, z), new Vector(0, 0, z), new Vector(prevX, prevY, z)]); // ++++

bases.push([new Vector(x, y, z), new Vector(prevX, prevY, z), new Vector(x, y, -z)]);

prevX = -A, prevY = fminus(prevX);

for (var x = -A + step; x < A; x += step) {

var y = fminus(x);

bases.push([new Vector(prevX, prevY, z), new Vector(0, 0, z), new Vector(x, y, z)]); // ++++

bases.push([new Vector(prevX, prevY, -z), new Vector(prevX, prevY, z), new Vector(x, y, z)]);

prevX = x;

prevY = y;

}

var x = A, y = fminus(x);

bases.push([new Vector(prevX, prevY, z), new Vector(0, 0, z), new Vector(x, y, z)]); // ++++

bases.push([new Vector(prevX, prevY, -z), new Vector(prevX, prevY, z), new Vector(x, y, z)]);

}

function perspectiveProject(point) {

const x = point[0], y = point[1], z = point[2];

return new Vector(x / (z + 6), y / (z + 6), z);

}

function project(point) {

const perspectivePoint = perspectiveProject(point);

const x = perspectivePoint[0], y = perspectivePoint[1], z = perspectivePoint[2];

return new Vector(

canvasWidth \* (x + SCALE) / (SCALE + SCALE),

canvasHeight \* (1 - (y + SCALE) / (SCALE + SCALE)), z

);

}

function renderPoint(point) {

const projectedPoint = project(point);

const x = projectedPoint[0], y = projectedPoint[1];

ctx.beginPath();

ctx.moveTo(x, y);

ctx.lineTo(x + 1, y + 1);

ctx.lineWidth = 4;

ctx.strokeStyle = 'white';

ctx.stroke();

}

function renderTriangle(triangle) {

projectedTriangle = triangle.map(project);

const a = projectedTriangle[0], b = projectedTriangle[1], c = projectedTriangle[2];

const v2 = a.subtract(b);

const v1 = c.subtract(b);

const norm = v1.cross(v2);

const location = norm.scalar(OBSERVER);

if (location > 0) {

const normMod = Math.sqrt(norm[0] \* norm[0] + norm[1] \* norm[1] + norm[2] \* norm[2]);

const normE = norm.scale(1 / normMod); // Единичный вектор нормали

const LightMod = Math.sqrt(LIGHT[0] \* LIGHT[0] + LIGHT[1] \* LIGHT[1] + LIGHT[2] \* LIGHT[2]);

const LightE = LIGHT.scale(1 / LightMod); // Единичный вектор источника света

const LxN = LightE.scalar(normE); // Скалярное произведение вектора света на нормаль

const tmpV = normE.scale(2 \* LxN);

const refl = LIGHT.subtract(tmpV); // Получение отражающего вектора

const cos = refl.scalar(OBS); // Косинус угла

// Уравнение модели освещения

const d = 0, K = 1, Ia = 1, Ii = 10, Ks = 10;

Ip = Ia \* Ka + Ii \* (Kd \* LxN + Ks \* cos \* cos) / (d + K);

color = 255 / Ip; // Получение нужного оттенка для закрашивания

ctx.fillStyle = 'rgb(' + 0 + ',' + color + ',' + color + ')';

ctx.strokeStyle = 'rgb(' + 0 + ',' + color + ',' + color + ')';

ctx.beginPath();

ctx.moveTo(a[0], a[1]);

ctx.lineTo(b[0], b[1]);

ctx.lineTo(c[0], c[1]);

ctx.lineTo(a[0], a[1]);

ctx.stroke();

ctx.fill();

ctx.stroke();

}

}

function render() {

ctx.fillStyle = 'white';

ctx.fillRect(0, 0, canvasWidth, canvasHeight);

bases.forEach((triangle) => {

const color = 'blue';

renderTriangle(triangle);

});

}

function rotate(thetaX, thetaY) {

ctx.fillStyle = 'white';

ctx.fillRect(0, 0, canvasWidth, canvasHeight);

lastTriangles = [];

bases.forEach((triangle, idx) => {

const rotatedTriangle = triangle.map((point) => {

point = point.rotateX(thetaX);

point = point.rotateY(thetaY);

return point;

});

const color = 'white';

renderTriangle(rotatedTriangle, color);

lastTriangles.push(rotatedTriangle);

});

}

initGeom();

render();

1. **Выводы**

Выполнив данную лабораторную работу, я закрепил знания по использованию HTML и запуску графический интерфейс, перерисовыванию его в зависимости от изменения окна и отрисовки многогранника.

**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: 15.10.2021).

2. Освещение и свет [Электронный ресурс]URL: <https://studizba.com/lectures/10-informatika-i-programmirovanie/305-kompyuternaya-grafika/4058-7-osveschenie-i-svet.html> (Дата обращения: 17.10.2021).