|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 10**

|  |  |
| --- | --- |
| **Тема «Алгоритма Плавающего горизонта построения трехмерных поверхностей»**  **Дисциплина Компьютерная графика**  **Студент Кузин Антон**  **Группа ИУ7-42Б**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель Куров А.В.** |  |

Москва.

2020 г.**Цель:**

Изучение и программная реализация алгоритма Плавающего горизонта построения трехмерных поверхностей.

**Техническое задание:**

Должна быть разработана программа, позволяющая осуществлять ввод пределов и шага изменения координат x, z, выбора уравнения поверхности из заранее сформированного списка, построение поверхности. Должен быть обеспечен поворот изображения (поверхности) вокруг каждой из трех координатных осей. Система координат должна быт неподвижной. Выполнить масштабирование для обеспечения размещения исходного изображения целиком в пределах поля вывода

Список уравнений поверхностей задается в отдельном модуле.

**Теоретические сведения:**

Алгоритм плавающего горизонта чаще всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде F(x, y, z) = 0.

Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путём пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат x, y или z. Благодаря чему функция F (x, y, z) = 0 сводится к последовательности кривых, лежащих в каждой из плоскостей, например к последовательности y = f(x, z), где z = const.

Далее полученные кривые можно спроектировать на плоскость XY и изобразить видимые части каждой кривой. Изображение строится, начиная с кривой, полученной в ближайшем к наблюдателю сечении.

Для определения видимости участка кривой формируется два массива, содержащих значения ординат, а именно – верхний горизонт, который формируется из участков с наибольшим значением ординат, и нижний, формирующийся из участков с наименьшим значением ординат. И если на текущей плоскости значение ординаты при заданном x больше (меньше) предыдущего значения, хранящегося в верхнем (нижнем) горизонте соответственно, то текущая кривая видима в этой точке.

При определении точки пересечения мы аппроксимируем кривую отрезком. В связи с чем точку пересечения можно найти, используя формуды

m = (yn - yk) / (xn - xk)

y = m(x-xk)+yk

x = xk – (dx \* (yтек – yпред)/(dyтек – dyпред))

И далее ориентироваться на такое правило, если текущая точка не видима, а предыдущая видима, то изображаем прямую от предыдущей точки до найденной точки пересечения, а если текущая видима, а предыдущая нет, то изображаем прямую от точки пересечения до текущей точки.

**Практическая часть:**

#include "alg.h"

#include <iostream>

#include <math.h>

void horizon(int x1, int y1, int x2, int y2, QVector<int> &top, QVector<int> &bot, QPainter \*painter)

{

if (x2 < x1)

{

qSwap(x1, x2);

qSwap(y1, y2);

}

if (x2 == x1)

{

top[x2] = qMax(top[x2], y2);

bot[x2] = qMin(bot[x2], y2);

painter->drawLine(x1, y1, x2, y2);

}

else

{

int x\_prev = x1;

int y\_prev = y1;

float m = (y2 - y1) / (float)(x2 - x1);

for (int x = x1; x <= x2; x++)

{

int y = qRound(m \* (x - x1) + y1);

top[x] = qMax(top[x], y);

bot[x] = qMin(bot[x], y);

painter->drawLine(x\_prev, y\_prev, x, y);

}

}

}

int visible(int x, int y, QVector<int> top, QVector<int> bot)

{

if (y < top[x] && y > bot[x])

return 0;

else if (y >= top[x])

return 1;

return -1;

}

void processEdge(int x, int y, int &xEdge, int &yEdge, QVector<int> &top, QVector<int> &bot, QPainter \*painter)

{

if (xEdge != -1)

horizon(xEdge, yEdge, x, y, top, bot, painter);

xEdge = x;

yEdge = y;

}

void getIntersection(int x1, int y1, int x2, int y2, QVector<int> horizon, int &xi, int &yi)

{

xi = x1;

yi = y1;

int dx = x2 - x1;

int dy = y2 - y1;

int dh = horizon[x2] - horizon[x1];

float m = dh / (float)dx;

if (!dx)

{

xi = x2;

yi = horizon[x2];

}

else if (y1 == horizon[x1] && y2 == horizon[x2])

{

xi = x1;

yi = y1;

}

else

{

xi = x1 - qRound(dx \* (y1 - horizon[x1]) / (float)(dy - dh));

yi = qRound((xi - x1) \* m + y1);

}

}

void rotate(float &a, float &b, float teta)

{

teta \*= M\_PI / 180.;

float t = a;

float c = std::cos(teta);

float s = std::sin(teta);

a = c \* a - s \* b;

b = c \* b + s \* t;

}

void transform(float x, float y, float z, angles teta, int &res\_x, int &res\_y)

{

float coef = 50;

int xc = 360;

int yc = 360;

rotate(y, z, teta.x);

rotate(x, z, teta.y);

rotate(x, y, teta.z);

res\_x = qRound(x \* coef + xc);

res\_y = qRound(y \* coef + yc);

}

void floatingHorizon(range xr, range zr, angles teta, float(\*f)(float, float), QPainter \*painter)

{

QVector<int> top(720, 0);

QVector<int> bot(720, 720);

int x\_left = -1, y\_left = -1, x\_right = -1, y\_right = -1;

int xp, yp;

for (float z = zr.e; z > zr.s; z -= zr.d)

{

float y\_p = f(xr.s, z);

transform(xr.s, y\_p, z, teta, xp, yp);

processEdge(xp, yp, x\_left, y\_left, top, bot, painter);

int pFlag = visible(xp, yp, top, bot);

for (float x = xr.s; x < xr.e; x += xr.d)

{

int xc = 0, yc = 0;

int xi = 0, yi = 0;

y\_p = f(x, z);

transform(x, y\_p, z, teta, xc, yc);

int tFlag = visible(xc, yc, top, bot);

if (tFlag == pFlag){

if (tFlag)

horizon(xp, yp, xc, yc, top, bot, painter);

}

else if (!tFlag)

{

if (pFlag)

getIntersection(xp, yp, xc, yc, top, xi, yi);

else

getIntersection(xp, yp, xc, yc, bot, xi, yi);

}

else if (tFlag == 1)

{

getIntersection(xp, yp, xc, yc, top, xi, yi);

horizon(xp, yp, xi, yi, top, bot, painter);

if (pFlag)

{

getIntersection(xp, yp, xc, yc, bot, xi, yi);

horizon(xp, xp, xi, yi, top, bot, painter);

}

}

else

{

getIntersection(xp, yp, xc, yc, bot, xi, yi);

horizon(xp, yp, xi, yi, top, bot, painter);

if (pFlag)

{

getIntersection(xp, yp, xc, yc, top, xi, yi);

horizon(xp, yp, xi, yi, top, bot, painter);

}

}

pFlag = tFlag;

xp = xc;

yp = yc;

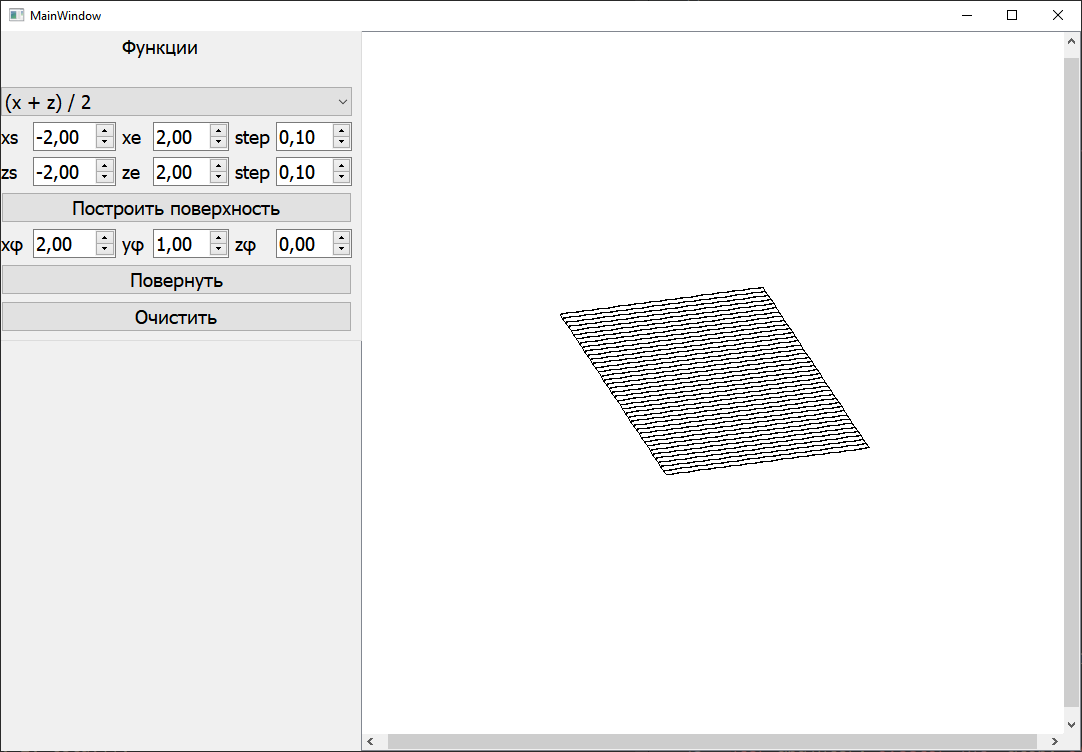
}

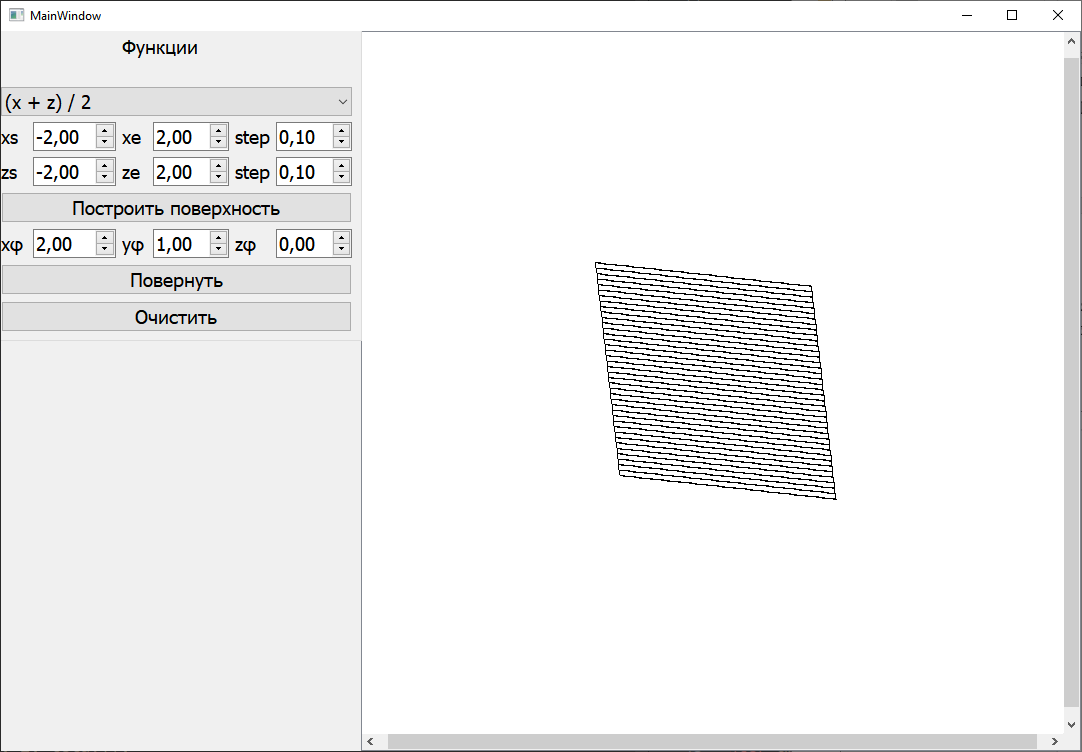
processEdge(xp, yp, x\_right, y\_right, top, bot, painter);

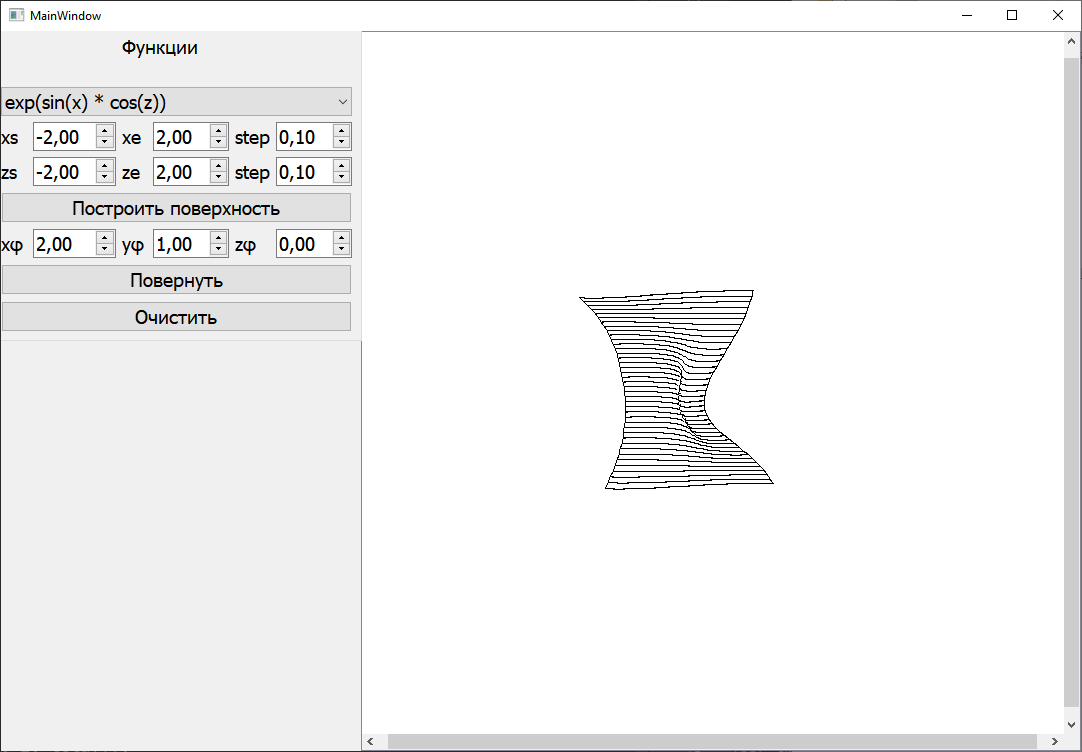
}

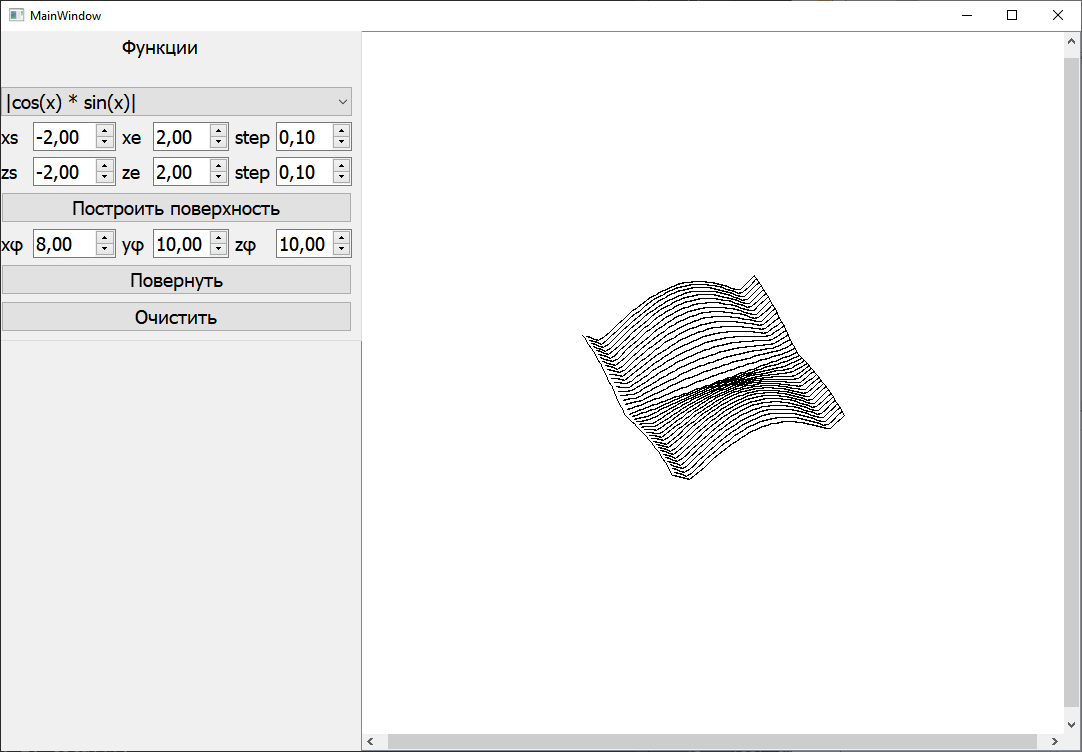
}

**Пример работы:**









**Вывод**

В ходе выполнения лабораторной работы был реализован и изучен алгоритм плавающего горизонта и проверена его работа.