|  |  |
| --- | --- |
| **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 \_ РЕАЛИЗАЦИЯ АЛГОРИТМА ПЛАВАЮЩЕГО ГОРИЗОНТА ПОСТРОЕНИЯ ТРЕХМЕРНЫХ ПОВЕРХНОСТЕЙ**  **.**  **Студент \_Чаушев Александър Красимиров\_\_\_\_\_\_\_\_**  **Группа \_ИУ7-46Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_Куров А. В.\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:** Изучение и программная реализация алгоритма Плавающего горизонта построения трехмерных поверхностей.

**Входные данные:** Список уравнений поверхносте, которые задаются в отдельном модуле.

**Выходные данные:** Пользовательское меню, содержащее поля ввода и холст с конечным изображением.

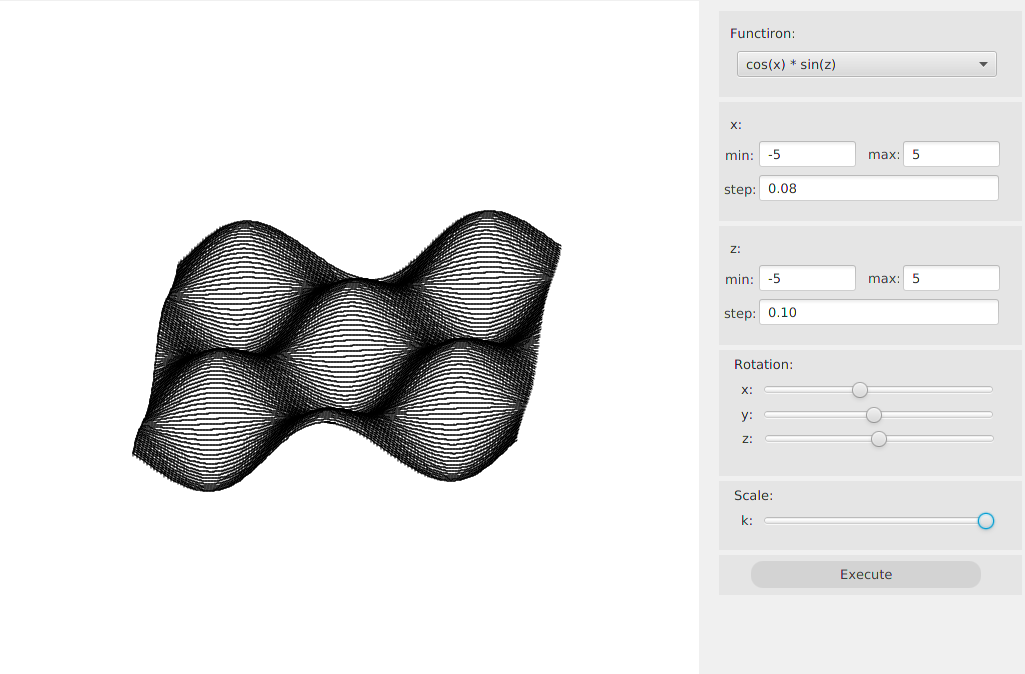
**Результат:** Должна быть разработана программа, позволяющая осуществлять ввод пределов и шага изменения координат x, z, выбора уравнения поверхности из заранее сформированного списка, построение поверхности. Должен быть обеспечен поворот изображения (поверхности) вокруг каждой из трех координатных осей. Система координат должна быт неподвижной.

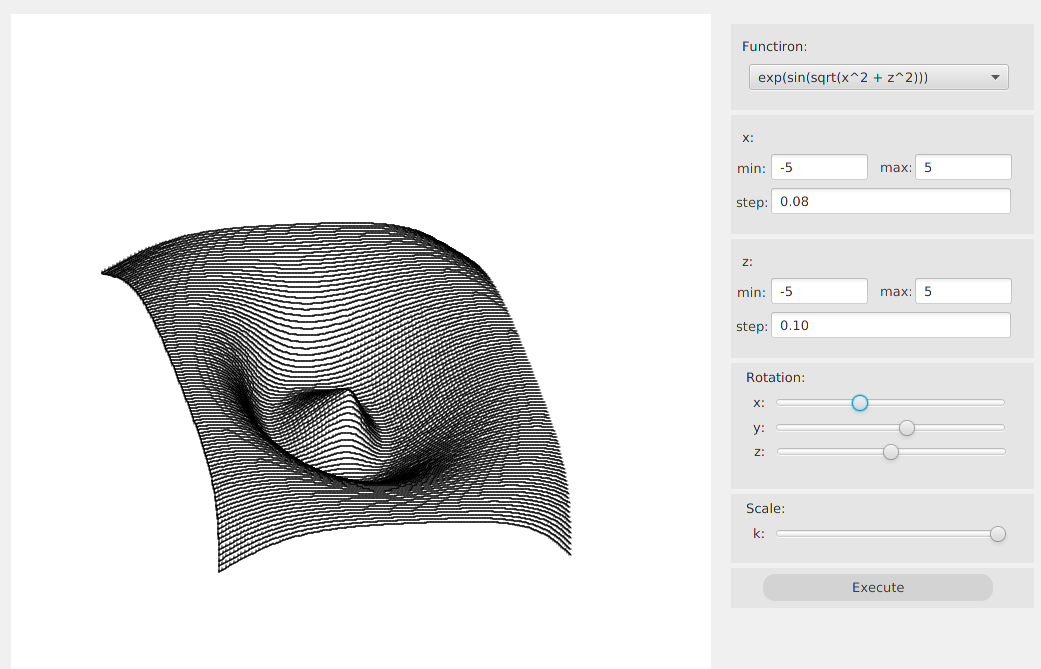
**Ошибочные ситуации:** Программа прекращается, если хотя бы один из входных данных не корректен.

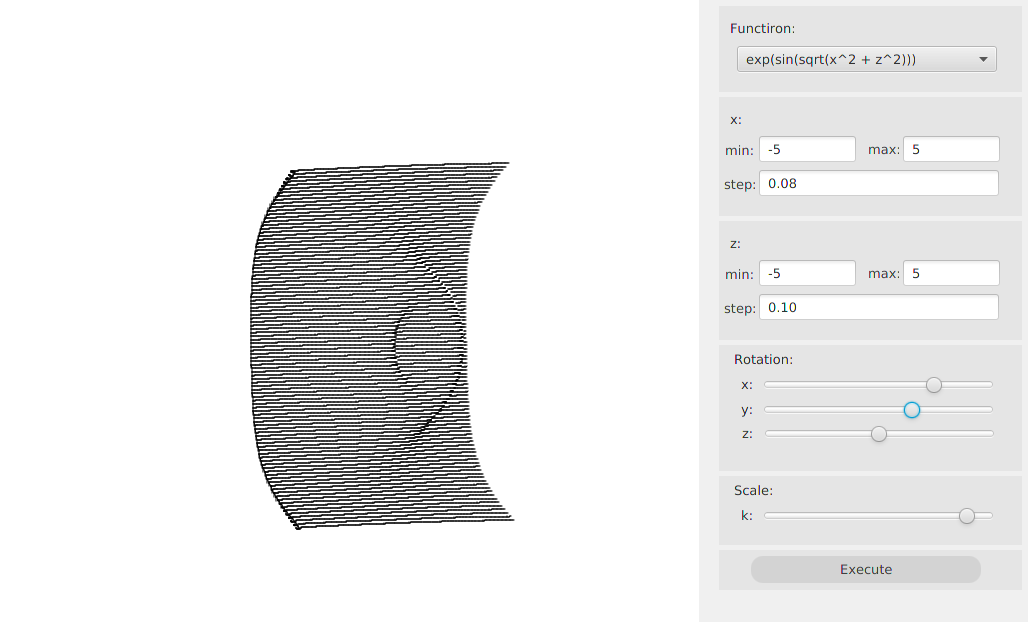
**Теоретическая**

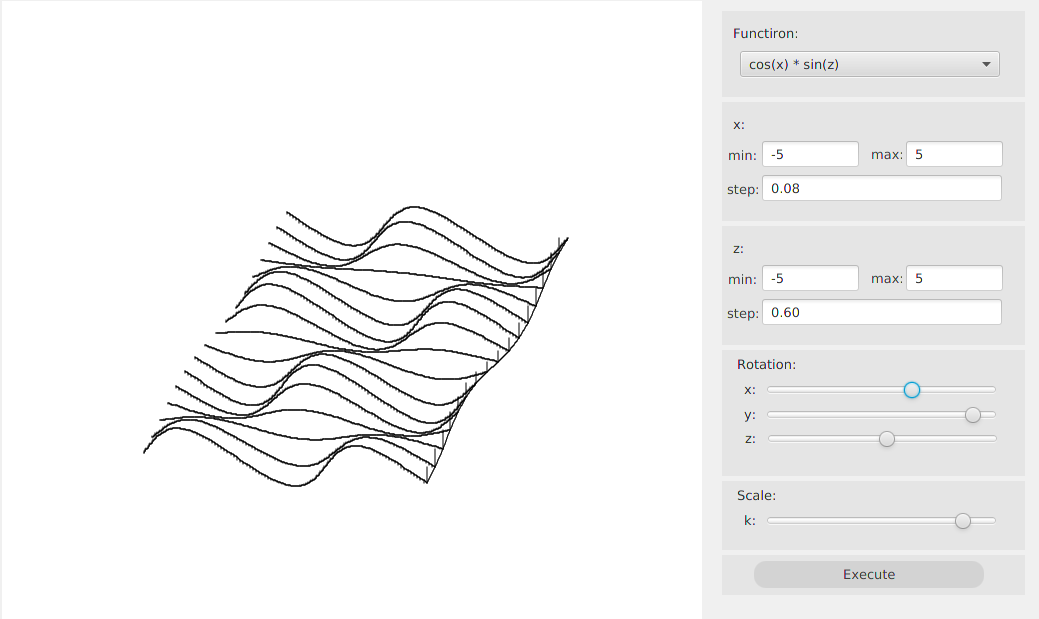
Этот алгоритм предназначен для изображения поверхностей, задаваемых неявным уравнением F(x,y,z)=0. В общем случае считается, что наблюдатель расположен на оси З в бесконечности. Ближайшая к наблюдателю кривая видима всегда. Далее в каждом сечении имеем уравнение y = f(x, z=const). Для очередной кривой вычисляем Yj(x\_i) – она будет видима, если она расположена выше самой верхней точки кривой j-1, или ниже самой нижней её точки. 𝑦𝑗(𝑥𝑖)>𝑦𝑚𝑎𝑥(𝑥𝑖)или𝑦𝑗(𝑥𝑖)<𝑦𝑚𝑖𝑛(𝑥𝑖). Таким образом, нижний горизонт – массив минимальных значений функции, верхний – массив максимальных значений. Если мы можем вычислить значение функции в каждой точке, то алгоритм достаточно просто выполняется. Очередная точка текущей кривой является видимой, если её ордината больше текущего максимума (по предыдущим кривым) или меньше текущего минимума.Вычисления могут производиться не с шагом=1 по оХ. Если очередная точка является невидимой, то она не соединяется с предыдущей. Наоборот, если видима, то соединяется. Для упрощения вычислений можно использовать линейную аппроксимации.

**Пример работы:**

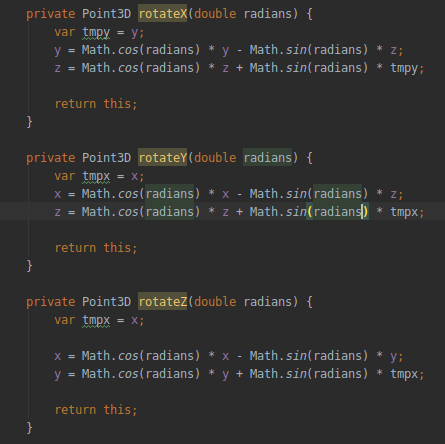




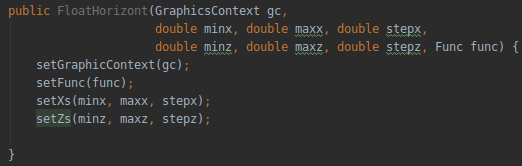




**Код программы:**



**Конструктор** FloatHorizont



Метод findIntersection

public Point2D findIntersection(Point2D prev, Point2D cur, LinkedHashMap<Integer, Integer> hor) {

Integer res\_x = prev.getY(), res\_y = prev.getY();

Integer arr\_y1 = hor.getOrDefault(prev.getX(), null);

Integer arr\_y2 = hor.getOrDefault(cur.getX(), null);

if (arr\_y1 == null || arr\_y2 == null) {

return null;

}

int dx = cur.getX() - prev.getX();

int dy\_cur = cur.getY() - prev.getY();

int dy\_prev = arr\_y2 - arr\_y1;

if (dx == 0)

{

res\_x = cur.getX();

res\_y = hor.getOrDefault(res\_x, null);

if (res\_y == null) {

return null;

}

}

else if (prev.getY() == arr\_y1 && cur.getY() == arr\_y2)

{

res\_x = prev.getX();

res\_y = prev.getY();

}

else

{

double m = dy\_cur / (double)(dx);

if (dy\_cur != dy\_prev)

{

res\_x = prev.getX() - (int)Math.round(((prev.getY() - arr\_y1) \* dx / (double)(dy\_cur - dy\_prev)));

res\_y = (int)Math.round((res\_x - prev.getX()) \* m + prev.getY());

}

}

return new Point2D(res\_x, res\_y);

}

Метoд draw()

public void draw() {

clean(gc);

boolean firstCurve = true;

// <x, y>

var topCoords = new LinkedHashMap<Integer, Integer>();

var botCoords = new LinkedHashMap<Integer, Integer>();

for (int i = -(int)(width / 2); i < width / 2; i++) {

topCoords.put(i, - (int)height / 2);

botCoords.put(i, (int)height / 2);

}

// Для каждой секущей плосковти

for (var z = maxz; z >= minz; z -= stepz) {

boolean firstPoint = true;

// 2. Для каждой точки, расположеной на кривой, полученой

// в текущей секущей плоскости

for(var x = minx; x <= maxx; x += stepx) {

var y = func.applyFunc(x, z);

// 1. Обработка левое баковое ребро

if (firstPoint && !firstCurve) {

drawHorizont(

to2D(transform(new Point3D(x, y, z))),

to2D(transform(new Point3D(x, func.applyFunc(x, z + stepz), z + stepz))),

topCoords, botCoords);

}

if (!firstPoint) {

Point2D curr = to2D(transform(new Point3D(x, y, z)));

Point2D prev = to2D(transform(new Point3D(x - stepx,

func.applyFunc(x - stepx, z), z)));

int visCurr = isVisible(curr, topCoords, botCoords);

int visPrev = isVisible(prev, topCoords, botCoords);

if (visCurr != 0 && visPrev != 0) {

// if both are visible

drawHorizont(prev, curr, topCoords, botCoords);

} else if (visCurr == 0) {

// if only prev is visible

if (visPrev == 1) {

var intersection = findIntersection(prev, curr, topCoords);

drawHorizont(prev, intersection, topCoords, botCoords);

} else if (visPrev == -1) {

var intersection = findIntersection(prev, curr, botCoords);

drawHorizont(prev, intersection, topCoords, botCoords);

}

} else {

// if only curr is visible

if (visCurr == 1) {

var intersection = findIntersection(prev, curr, topCoords);

drawHorizont(intersection, curr, topCoords, botCoords);

} else if (visCurr == -1) {

var intersection = findIntersection(prev, curr, botCoords);

drawHorizont(intersection, curr, topCoords, botCoords);

}

// if both are unvisible continue

}

}

firstPoint = false;

}

// 2.7 Обработать правое баковое ребро

if (!firstCurve) {

//var prev = new Point3D(maxx, func.applyFunc(maxx, z - stepz), z - stepz);

//var curr = new Point3D(maxx, func.applyFunc(maxx, z), z);

//drawHorizont(to2D(transform(prev)), to2D(transform(curr)), topCoords, botCoords);

}

firstCurve = false;

}

}

Метод DrawHorizont()

private void drawHorizont(Point2D from, Point2D to,

LinkedHashMap<Integer, Integer> top, LinkedHashMap<Integer, Integer> bot) {

if (from == null || to == null)

return;

if (from.getX() > to.getX()){

Point2D tmp= from;

from = to;

to = tmp;

}

int dx = to.getX() - from.getX();

int dy = to.getY() - from.getY();

if (dx == 0)

{

top.put(from.getX(), Math.max(from.getY(), top.get(from.getX())));

bot.put(from.getX(), Math.min(from.getY(), bot.get(from.getX())));

strokeLine(from, to);

}

else {

Point2D prev = new Point2D(from.getX(), to.getY());

double m = (dy) / (double) (dx);

for (int x = from.getX(); x <= to.getX(); x++) {

int y = (int)Math.round(m \* (x - from.getX()) + from.getY());

top.put(x, Math.max(top.get(x), y));

bot.put(x, Math.min(bot.get(x), y));

Point2D cur = new Point2D(x, y);

strokeLine(prev, cur);

prev = cur;

}

}

}

Метод is Visible()

private int isVisible(Point2D p,

HashMap<Integer, Integer> top, HashMap<Integer, Integer> bot) {

var topY = top.getOrDefault(p.getX(), null);

var botY = bot.getOrDefault(p.getX(), null);

var y = p.getY();

var x = p.getX();

if(topY == null || botY == null)

return 0;

if (y < top.get(x) && y > bot.get(x))

return 0;

if (y >= top.get(x))

return 1;

return -1;

}