|  |  |
| --- | --- |
| 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**  Алгоритм плавающего горизонта.  **Студент** Козлов М. А.  **Группа** ИУ7-45Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **Преподаватель \_**Куров А. В.**\_\_\_\_\_** |  |

Москва.

2020 г.

**Цель работы:** 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.

**Теоретический материал.**

Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объёмов, которые видимы или невидимы для наблюдателя, находящийся в заданной точке пространства.

Из-за сложности задачи и разных требований на качество и скорость решения задачи, нет единого алгоритма решения.

Для моделирования процессов в реальном времени нужны быстрые алгоритмы, для машинной мультипликации нужны алгоритмы, которые могут генерировать сложные реалистичные изображения.

Решать задачу можно в объектном пространстве или экранном пространстве.

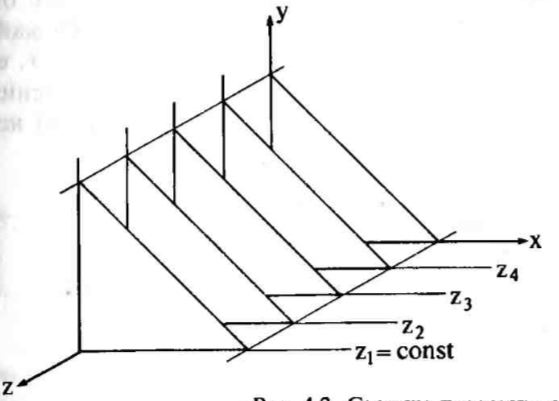
В объектном пространстве вещественные координаты, поэтому точность решения в объектном пространстве выше, чем в пространстве экранном, где используются целые величины.

Алгоритм плавающего горизонта.

Предназначен для построения поверхностей, заданных неявным уравнением

Задача решается в экранном пространстве.

Главная идея данного алгоритма – сведение трёхмерной задачи к двумерной путём пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющие постоянные значения координат x, y или z.



При пересечении поверхности плоскостью , получается кривая, описываемая уравнением

Алгоритм пересекает поверхность плоскостями z = const, (они упорядочены по удалению от наблюдателя). Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней.  
Удаление невидимых линий заключается в следующем:

Формируется два горизонта верхний и нижний.  
Верхний горизонт состоит из участков кривых с наибольшим значением y при каждом значении x. Нижний горизонт формируется из участков кривых с наименьшим значением y каждом значении x.

Если на текущей плоскости в точке x, значение y на кривой больше верхнего горизонта, или ниже нижнего горизонта, то точка видима и соответствующий горизонт изменяется.

Вычислять значение функции в каждой точке иногда слишком трудозатратно, поэтому чтобы снизить эти затраты можно увеличить шаг квантования. А изображаемые участки кривых аппроксимировать прямыми.

Если очередная точка невидима, мы не соединяем ее с предыдущей точкой, а если видима - соединяем с предыдущей точкой. В первом случае возникает участок, который является видимым, но не изображен, во втором невидимый участок, который изображен.

Для решения этой проблемы необходимо найти точку пересечения двух кривых. В первом случае изображаем участок от предыдущей точки до найденной точки пересечения, во втором - изображаем участок от найденной точки пересечения до текущей точки.

Если текущая точка кривой невидима, а предыдущая - видима, то надо изобразить участок кривой от предыдущей точки до найденной точки пересечения.

Если текущая точка кривой видима, а предыдущая - невидима, то изобразить участок кривой от найденной точки пересечения до текущей точки.

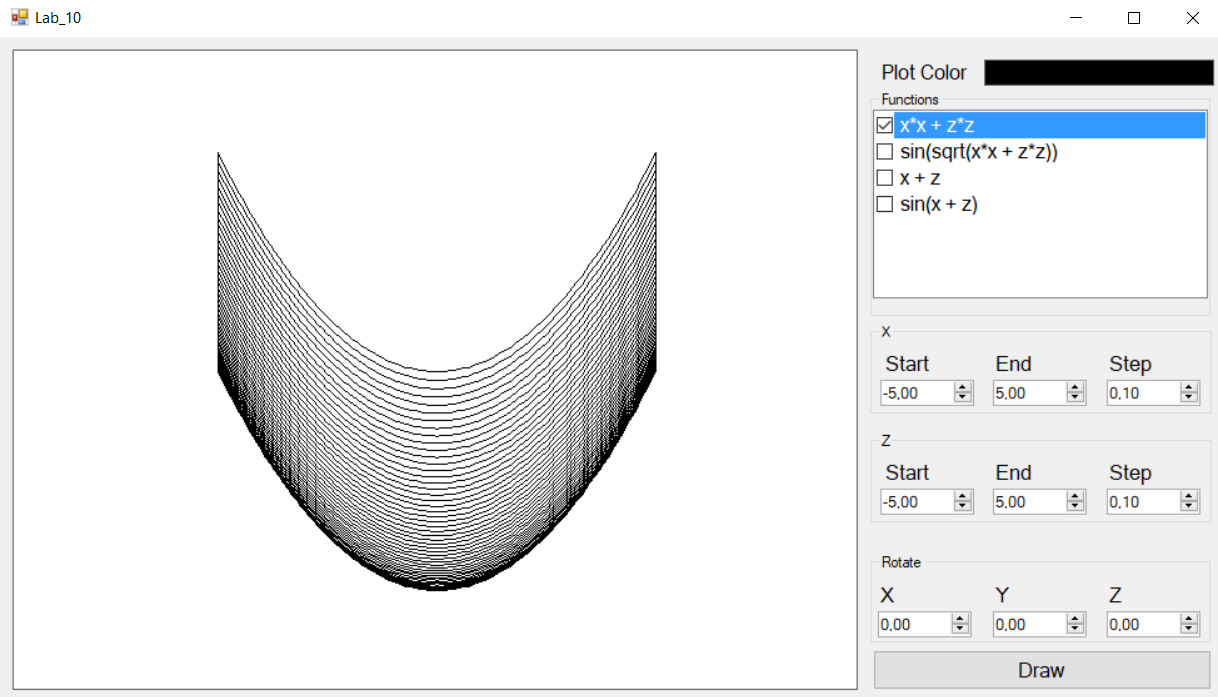
Качество изображения улучшится, если начальные и конечные точки каждой кривой соединить отрезками. Ребро, соединяющее начальные точки всех кривых называется левым боковым ребром. Соединяющее конечные точки - правым боковым ребром.

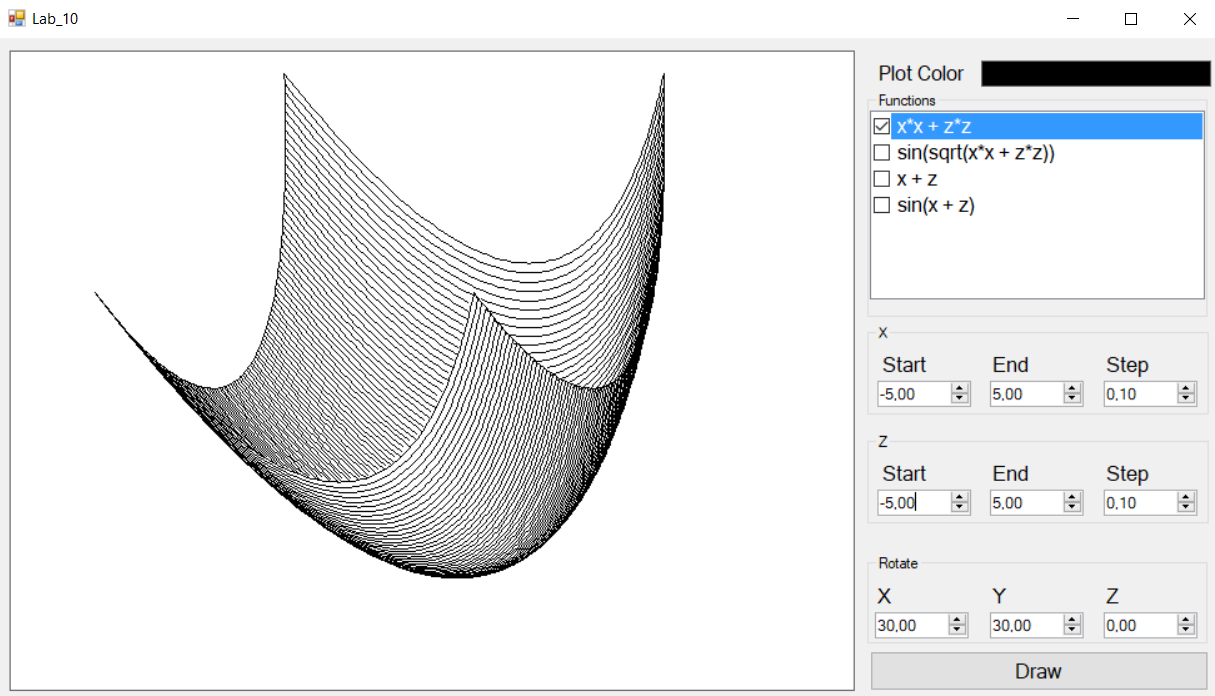
Недостаток алгоритма:

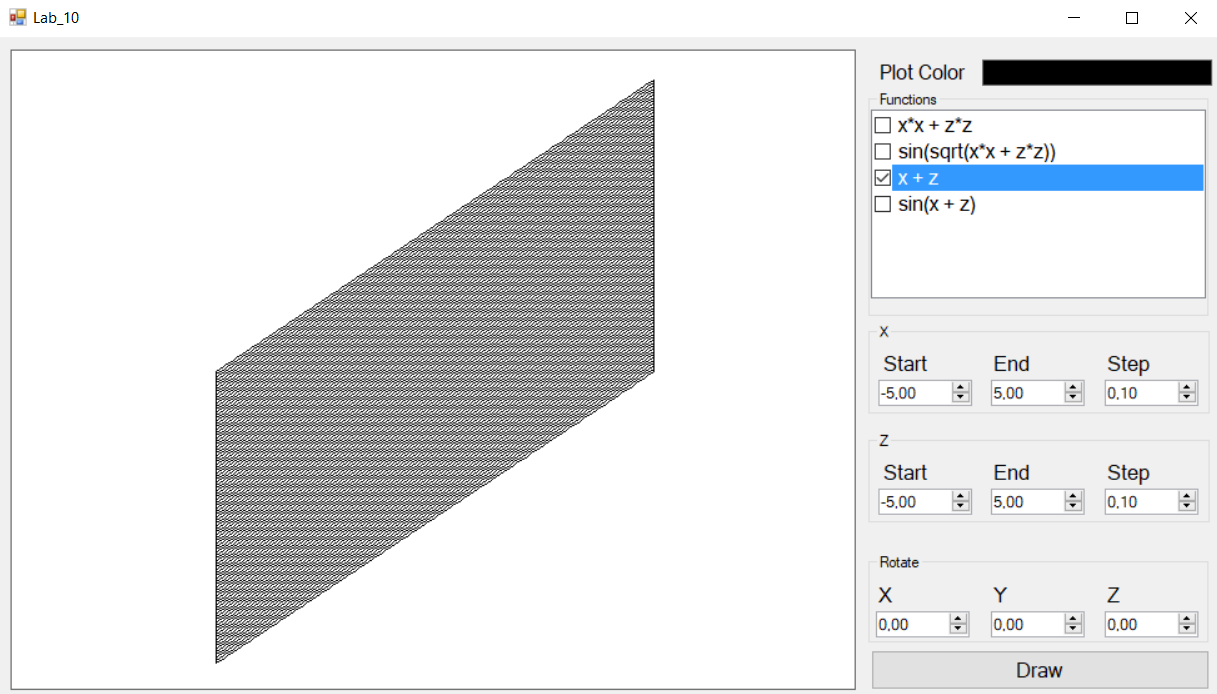
Если функция изменяется очень быстро, то возможны потери пиков между шагами квантования.

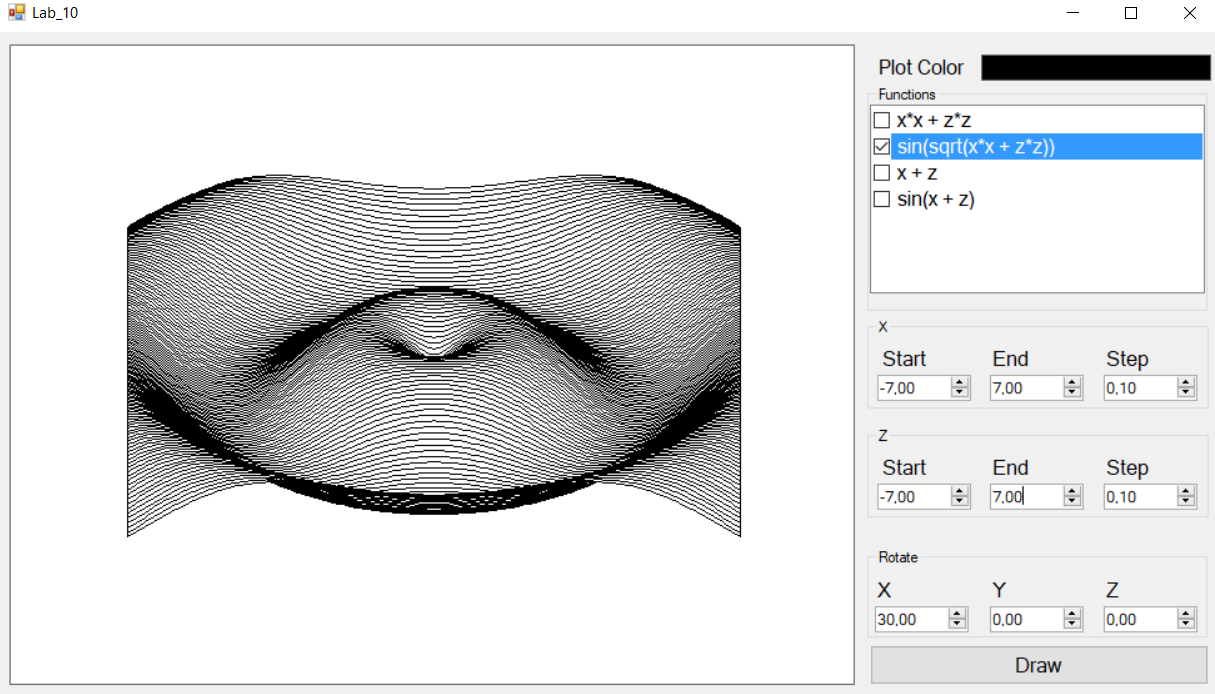
**Примеры работы программы.**

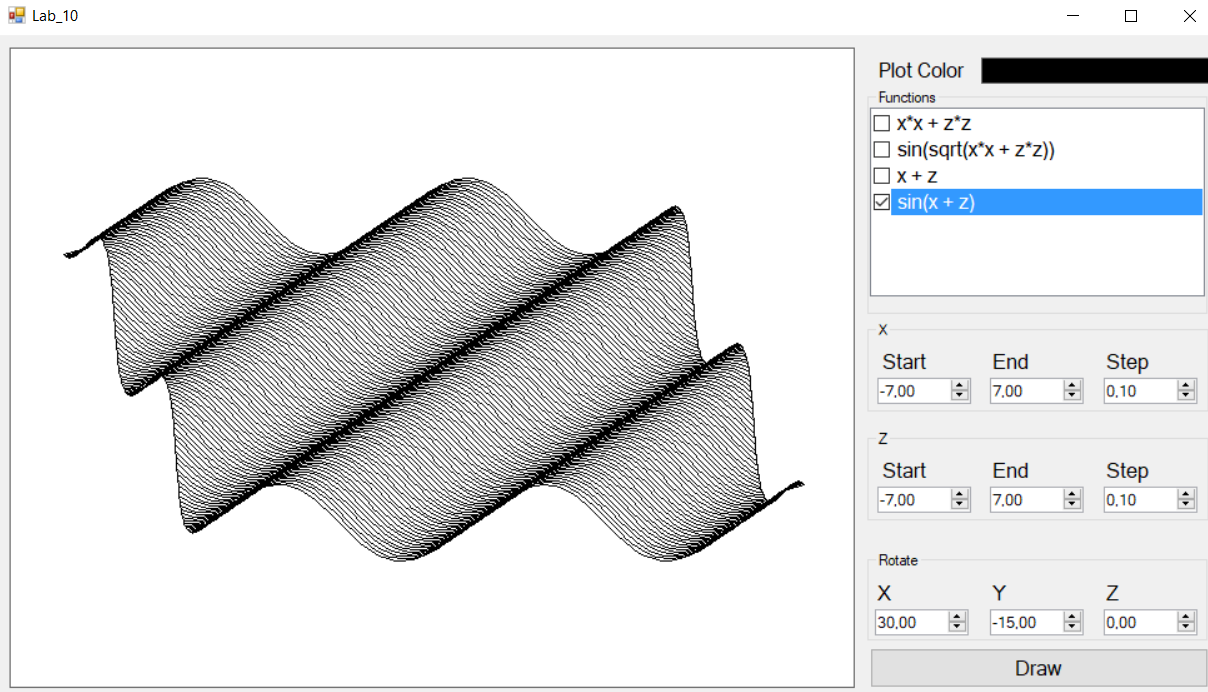
\*ось y направлена вверх

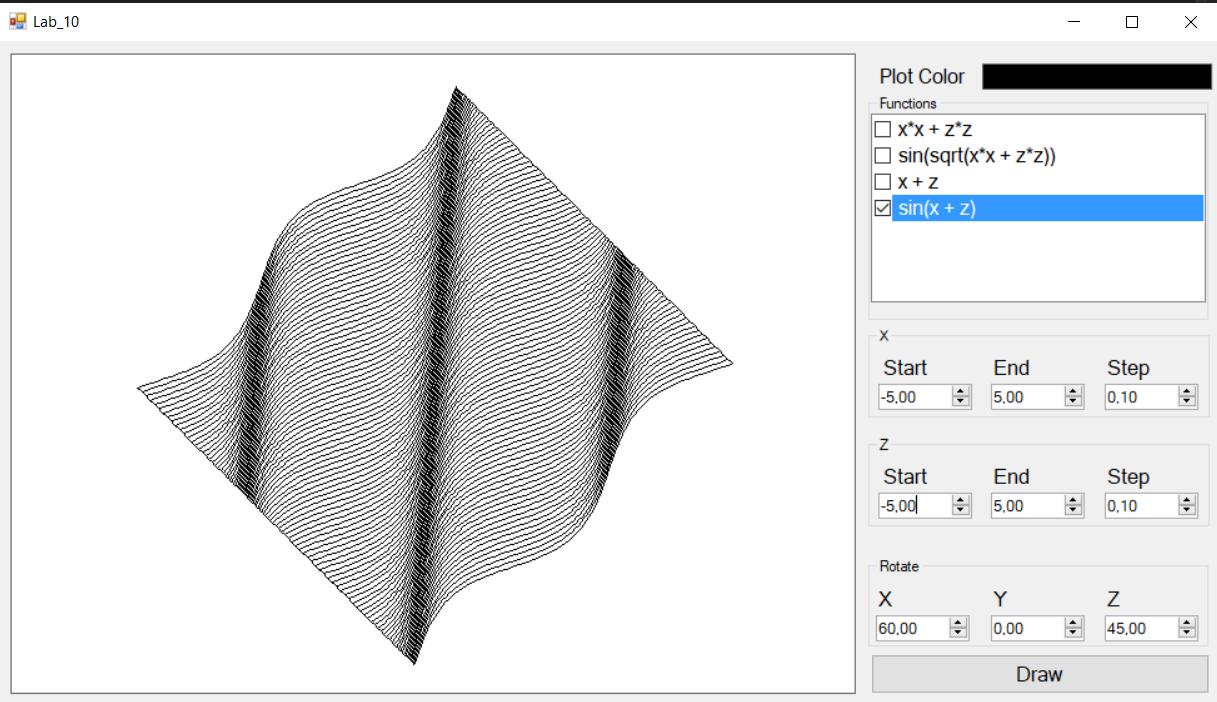












**Исходный код.**

private void ResetArray()

{

for (int i = 0; i < Up.Length; i++)

{

Down[i] = screenSize.Height;

Up[i] = 0;

}

}

public void Draw(Func<double, double, double> func,

double dx, double dy, double dz,

double zStart, double zEnd, double zStep,

double xStart, double xEnd, double xStep,

Graphics painter, Pen pen)

{

int xLeft = -1;

int yLeft = -1;

int xRight = -1;

int yRight = -1;

int xPrev = 0;

int yPrev = 0;

int k = 35;

ResetArray();

// вычисление функции y(x, z=const), начиная с ближайшей к наблюдателю

for (double z = zEnd; z > zStart; z -= zStep)

{

double yp = func(xStart, z);

// xPrev = xStart

// видовое преобразование

transform(xStart, yp, z, k, dx, dy, dz, ref xPrev, ref yPrev);

ProcessEdge(xPrev, yPrev, ref xLeft, ref yLeft, painter, pen);

int Pflag = Visible(xPrev, yPrev);

for (double x = xStart; x <= xEnd; x += xStep)

{

int xCur = 0, yCur = 0;

int xi = 0, yi = 0;

yp = func(x, z);

transform(x, yp, z, k, dx, dy, dz, ref xCur, ref yCur);

int Tflag = Visible(xCur, yCur);

if (Tflag == Pflag)

{

if (Pflag != 0)

{

horizon(xPrev, yPrev, xCur, yCur, painter, pen);

}

}

else if (Tflag == 0)

{

if (Pflag == 1)

GetIntersection(xPrev, yPrev, xCur, yCur, Up, ref xi, ref yi);

else

GetIntersection(xPrev, yPrev, xCur, yCur, Down, ref xi, ref yi);

horizon(xPrev, yPrev, xi, yi, painter, pen);

}

else if (Tflag == 1)

{

if (Pflag == 0)

{

GetIntersection(xPrev, yPrev, xCur, yCur, Up, ref xi, ref yi);

horizon(xPrev, yPrev, xi, yi, painter, pen);

}

else

{

GetIntersection(xPrev, yPrev, xCur, yCur, Up, ref xi, ref yi);

horizon(xPrev, yPrev, xi, yi, painter, pen);

GetIntersection(xPrev, yPrev, xCur, yCur, Down, ref xi, ref yi);

horizon(xi, yi, xCur, yCur, painter, pen);

}

}

else

{

if (Pflag == 0)

{

GetIntersection(xPrev, yPrev, xCur, yCur, Down, ref xi, ref yi);

horizon(xPrev, yPrev, xi, yi, painter, pen);

}

else

{

GetIntersection(xPrev, yPrev, xCur, yCur, Up, ref xi, ref yi);

horizon(xPrev, yPrev, xi, yi, painter, pen);

GetIntersection(xPrev, yPrev, xCur, yCur, Down, ref xi, ref yi);

horizon(xi, yi, xCur, yCur, painter, pen);

}

}

Pflag = Tflag;

xPrev = xCur;

yPrev = yCur;

}

ProcessEdge(xPrev, yPrev, ref xRight, ref yRight, painter, pen);

}

}

static void Swap<T>(ref T a, ref T b)

{

T temp = a;

a = b;

b = temp;

}

void transform(double x, double y, double z, double k, double dx, double dy, double dz, ref int resX, ref int resY)

{

rotateX(ref y, ref z, dx);

rotateY(ref x, ref z, dy);

rotateZ(ref x, ref y, dz);

resX = (int)(Math.Round(x \* k + screenSize.Width / 2));

resY = (int)(Math.Round(-y \* k + screenSize.Height / 2));

}

void GetIntersection(int x1, int y1, int x2, int y2, int[] horizon, ref int xi, ref int yi)

{

int deltaX = x2 - x1;

int deltaY = y2 - y1;

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

if (deltaX == 0)

{

xi = x2;

yi = horizon[x2];

}

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

{

xi = x1;

yi = y1;

}

else

{

double m = deltaY / (double)(deltaX);

xi = x1 - (int)(Math.Round(deltaX \* (y1 - horizon[x1]) / (double)(deltaY - deltaYHorizon)));

yi = (int)(Math.Round((xi - x1) \* m + y1));

}

}

void horizon(int x1, int y1, int x2, int y2, Graphics painter, Pen pen)

{

if (x2 < 0 || x2 > screenSize.Width - 1)

return;

if (x1 < 0 || x1 > screenSize.Width - 1)

return;

if (x2 - x1 == 0)

{

Up[x2] = Math.Max(Up[x2], y2);

Down[x2] = Math.Min(Down[x2], y2);

}

else

{

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

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

{

int y = (int)(Math.Round(m \* (x - x1) + y1));

Up[x] = Math.Max(Up[x], y);

Down[x] = Math.Min(Down[x], y);

}

}

painter.DrawLine(pen, x1, y1, x2, y2);

}

// Обработка ребер

void ProcessEdge(int x, int y, ref int xEdge, ref int yEdge, Graphics painter, Pen pen)

{

// xEdge == -1, когда создаётся первая кривая

if (xEdge != -1)

horizon(xEdge, yEdge, x, y, painter, pen);

xEdge = x;

yEdge = y;

}

// Проверяет видимость текущей точки

int Visible(int x, int y)

{

if (y < Up[x] && y > Down[x])

return 0; // невидима

if (y >= Up[x])

return 1; // видима, выше верхнего горизонта

return -1; // видима, ниже нижнего горизонта

}

// Преобразование координат

void rotateX(ref double y, ref double z, double dx)

{

double buf = y;

double cosX = Math.Cos(dx);

double sinX = Math.Sin(dx);

y = cosX \* y - sinX \* z;

z = sinX \* buf + cosX \* z;

}

void rotateY(ref double x, ref double z, double dy)

{

double buf = x;

double cosY = Math.Cos(dy);

double sinY = Math.Sin(dy);

x = cosY \* x - sinY \* z;

z = sinY \* buf + cosY \* z;

}

void rotateZ(ref double x, ref double y, double dz)

{

double buf = x;

double cosZ = Math.Cos(dz);

double sinZ = Math.Sin(dz);

x = cosZ \* x - sinZ \* y;

y = sinZ \* buf + cosZ \* y;

}