

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Дисциплина « Компьютерная графика»**

**Лабораторная работа №10**

**Работу выполнил:**

студент группы ИУ7-43Б

Сукочева А.

**Работу проверил:**

Куров А. В.

2020 г

**Цель работы:**

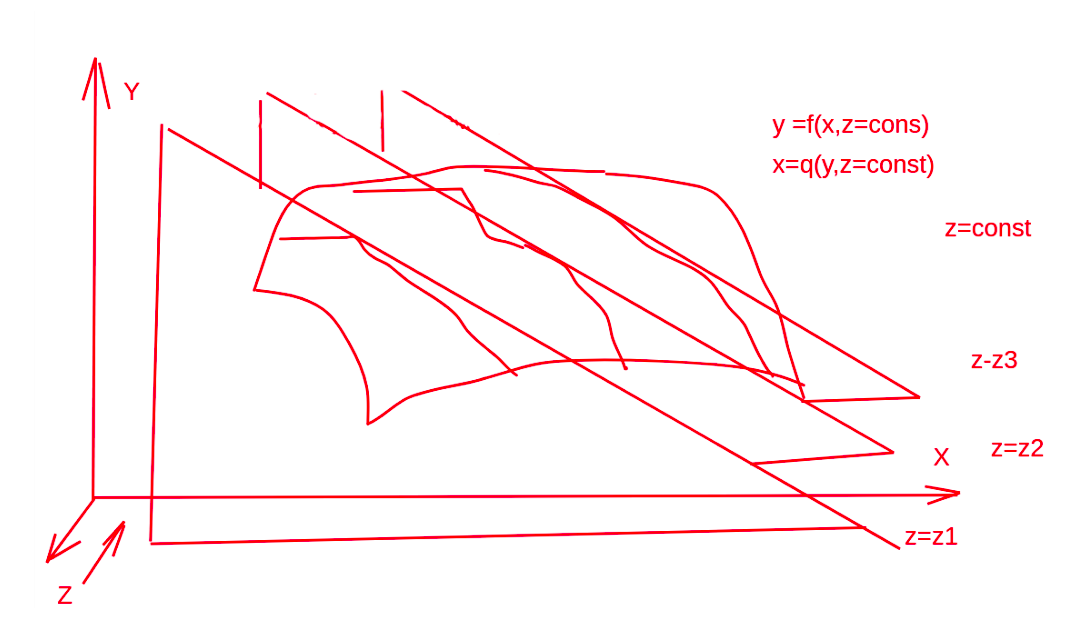
Изучение и программная реализация алгоритма Плавающего горизонта построения трехмерных поверхностей.

**Результат:**

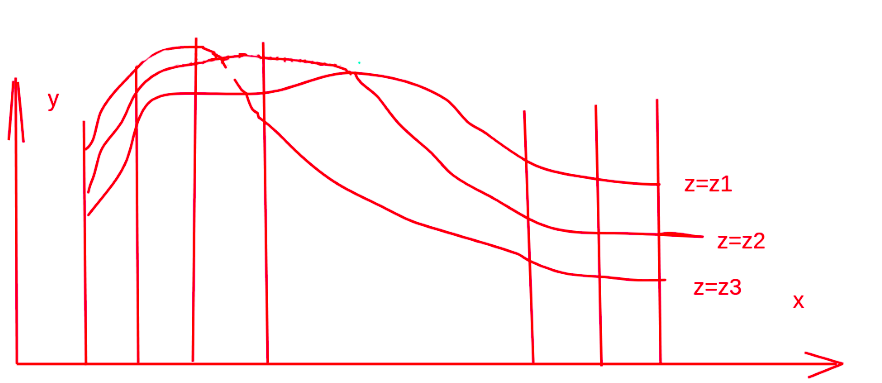
Должна быть разработана программа, позволяющая осуществлять ввод пределов и шага изменения координат x, z, выбора уравнения поверхности из заранее сформированного списка, построение поверхности. Должен быть обеспечен поворот изображения (поверхности) вокруг каждой из трех координатных осей. Выполнить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

**Теория:**

Алгоритм Плавающего горизонта позволяет построить нам изображение трехмерной поверхности, задаваемых неявным уравнением F(x,y,z) = 0. Решает задачу в экранной системе координат. Считается, что наблюдатель расположен на положительной полуоси z и смотрит в сторону начала координат. Идея алгоритма состоит в том, что мы рассматриваем ряд секущих плоскостей, которыми мы рассекаем поверхность. Секущие плоскости мы расставляем с некоторым шагом, перпендикулярно оси z. В сечение мы будем получать некоторые кривые. Секущие плоскости имеют уравнения z = const. Уравнение каждой полученной кривой мы можем записать y = f(x, z=const) или x=q(y, z=const).



Далее полученные кривые необходимо спроецировать на координатную плоскость х0y и отобразить видимые участки каждой кривой. Кривая, полученная в ближайшем к наблюдателю сечении будет полностью видима. Также кривая, полученная во втором сечении будет являться полностью видимой, потому что она может проходить либо над первой кривой, либо под первой кривой, в обоих случаях она полностью видимая. Далее, если мы вычисляем значение функции с шагом = 1 нужно будет определять видимость каждой точки кривой. Либо, если мы с большим шагом вычисляем значение функции, то мы должны определять видимость очередного участка кривой (сегмента).



Точка очередного сечения видима, если она находится выше самой верхней кривой или ниже самой нижней кривой. Т.е. y(x) > ymax(x) или y(x)<ymin(x), где ymax(x) и ymin(x) значение максимальных и минимальных ординат ранее рассмотренных кривых в данной кривой. Если точка очередной кривой находится между самой нижней кривой и самой верхней кривой, то она невидима.

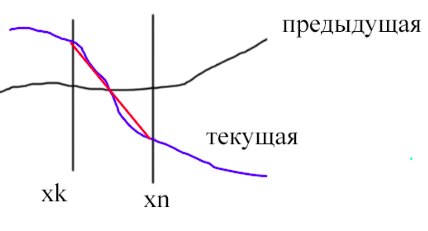
Верхний горизонт представляет собой совокупность видимых участков кривых с наибольшим значением ординат. Верхний горизонт - это массив максимальных значений ординат точек кривых.

Нижний горизонт представляет собой совокупность видимых участков кривых с наименьшим значением ординат. Нижний горизонт - это массив минимальных значений ординат точек кривых.

Плавающий горизонт - потому что от шага к шагу верхний горизонт поднимается, а нижний опускается.

Суть алгоритма сводится к тому, что на каждом шаге нам нужно вычислять текущее значение ординаты точки расположенной на очередной кривой и сравнивать с текущим максимумом и минимумом и, в зависимости от полученного результата сравнения, изменять или не изменять максимум или минимум. Т.е. поддерживать значение горизонта.

Рассмотрим, как решать задачу, если мы решили увеличить шаг изменения аргумента. С целью упрощения решения задачи и снижения трудозатрат аппроксимируем участки двух кривых отрезками прямых. Точка пересечения находится как точка пересечения двух прямых



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

y = m(x-xk)+yk

yпр=mпр(x-xk)+yпрk

yтек=mтек(x-xk)+yтекk

mпр(x-xk)+yпрk = mтек(x-xk)+yтекk

x(mпр-mтек) =xk(mпр-mтек)+(yтек - yпрк)

x = xk+(yтекk - yпрk) / (mпр-mтек)

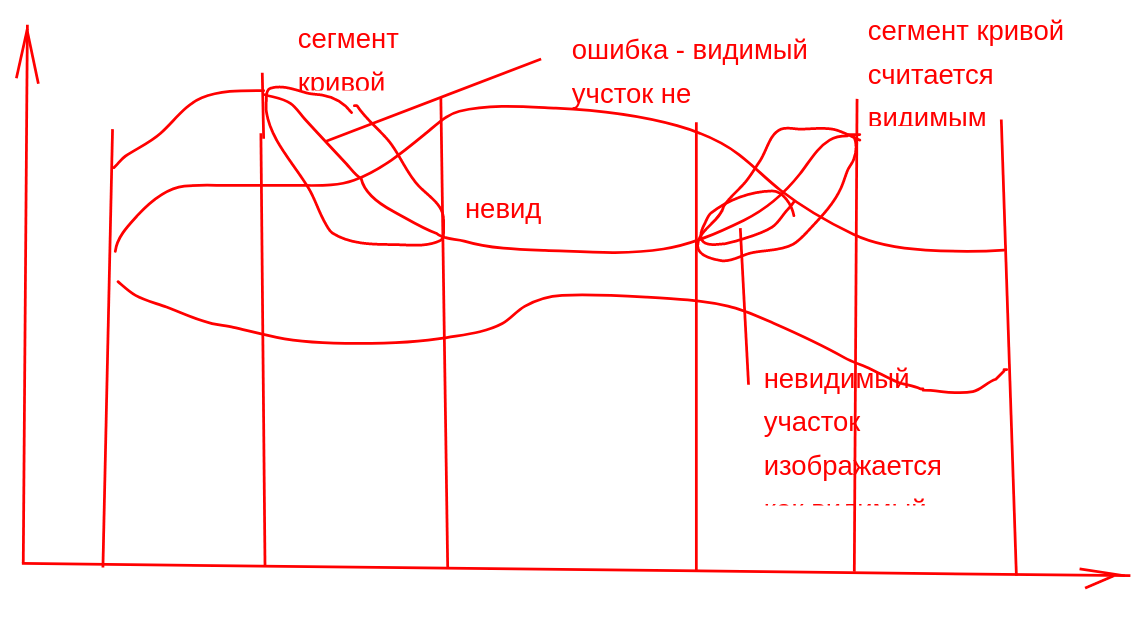
x = xk-(yтекk - yпрk) / (mтек-mпр)

Подставляем тангенс.

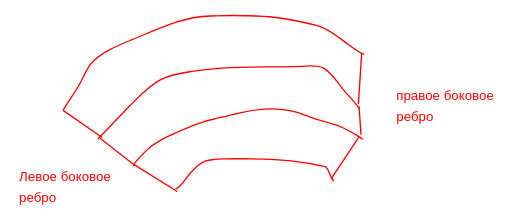
x=xk-(yтекk-yпрk)/(dyтек-dyпр)/dx

x=xk-dx(yтекk-yпрk)/(dyтек-dyпр)

Если видимость сегмента изменилась и текущая точка является невидимой, то мы изображаем участок кривой от предыдущей точки, до найденной точки пересечения. Если же текущая точка видима, то нужно изобразить участок кривой от пересечения до текущей точки.



Также нужно добавить боковые ребра (Соединить первые точки каждой кривой между собой прямыми отрезками и аналогично последнии):



Исходные данные:

Нужно задать поверхность.

Задать пределы изменения по координате z и шаг с которым мы расставляем секущие плоскости.

Задать пределы изменения по координате x и шаг изменения.

**Программа:**

**# Подпрограмма, определяющая видимость точки.**

**# flag:**

**# 0 - невидима.**

**# 1 - выше верхнего.**

**# -1 - ниже нижнего.**

**def Visible(x, y): # Visible point**

**global top, bottom**

**# Если точка, ниже нижнего горизонта (или на нем)**

**# То она видима.**

**if y <= bottom[x]:**

**return -1**

**# Если точка выше верхнего горизонта (или на нем)**

**# То она видима.**

**if y >= top[x]:**

**return 1**

**# Иначе она невидима.**

**return 0**

**# Подпрограмма вычисляет пересечение с горизонтом.**

**def Intersection(x1, y1, x2, y2, arr, canvas\_class):**

**dx = x2 - x1**

**dyc = y2 - y1**

**dyp = arr[x2] - arr[x1]**

**if dx == 0:**

**xi = x2**

**yi = arr[x2]**

**return xi, yi**

**if y1 == arr[x1] and y2 == arr[x2]:**

**return x1, y1**

**m = dyc / dx**

**xi = x1 - round(dx \* (y1 - arr[x1]) / (dyc - dyp))**

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

**return xi, yi**

**# Подпрограмма заполнения массивов горизонтов между x1 и x2**

**# На основе линейной интерполяции.**

**def Horizon(x1, y1, x2, y2, canvas\_class):**

**global top, bottom**

**# Проверка вертикальности наклона.**

**if (x2 - x1 == 0):**

**top[x2] = max(top[x2], y2)**

**bottom[x2] = min(bottom[x2], y2)**

**return**

**# Иначе вычисляем наклон.**

**m = (y2 - y1) / (x2 - x1)**

**# Движемся по x с шагом 1, чтобы заполнить**

**# Массивы от x1 до x2.**

**for x in range(x1, x2 + 1):**

**y = round(m \* (x - x1) + y1)**

**top[x] = max(top[x], y)**

**bottom[x] = min(bottom[x], y)**

**# Функция обработки и обновления точек бокового ребра**

**def Side(x, y, xe, ye,canvas\_class):**

**if (xe != -1):**

**# Если кривая не первая**

**canvas\_class.draw\_line([xe, ye], [x, y])**

**Horizon(xe, ye, x, y, canvas\_class)**

**xe = x**

**ye = y**

**return xe, ye**

**def FloatHorizon(borders\_x, x\_step, borders\_z, z\_step, canvas\_class, f, angles):**

**global top, bottom**

**# Инициализируем начальными значениями массивы горизонтов.**

**top = [0] \* WIDTH # Верхний.**

**bottom = [HEIGHT] \* WIDTH # Нижний.**

**x\_start = borders\_x[0]**

**x\_end = borders\_x[1]**

**z\_start = borders\_z[0]**

**z\_end = borders\_z[1]**

**x\_left, y\_left = -1, -1**

**x\_right, y\_right = -1, -1**

**z = z\_end**

**while z >= z\_start - z\_step / 2:**

**x\_prev = x\_start**

**y\_prev = f(x\_start, z)**

**x\_prev, y\_prev = transform(x\_prev,y\_prev, z, angles)**

**flag\_prev = Visible(x\_prev, y\_prev)**

**x\_left, y\_left = Side(x\_prev, y\_prev, x\_left, y\_left, canvas\_class)**

**x = x\_start**

**while x <= x\_end + x\_step / 2:**

**y\_curr = f(x, z)**

**x\_curr, y\_curr = transform(x, y\_curr, z, angles)**

**# Проверка видимости текущей точки.**

**flag\_curr = Visible(x\_curr, y\_curr)**

**# Равенство флагов означает, что обе точки находятся**

**# Либо выше верхнего горизонта, либо ниже нижнего,**

**# Либо обе невидимы.**

**if flag\_curr == flag\_prev:**

**# Если текущая вершина выше верхнего горизонта**

**# Или ниже нижнего (Предыдущая такая же)**

**if flag\_curr != 0:**

**# Значит отображаем отрезок от предыдущей до текущей.**

**canvas\_class.draw\_line([x\_prev, y\_prev], [x\_curr, y\_curr])**

**Horizon(x\_prev, y\_prev, x\_curr, y\_curr, canvas\_class)**

**# flag\_curr == 0 означает, что и flag\_prev == 0,**

**# А значит часть от flag\_curr до flag\_prev невидима. Ничего не делаем.**

**else:**

**# Если видимость изменилась, то**

**# Вычисляем пересечение.**

**if flag\_curr == 0:**

**if flag\_prev == 1:**

**# Сегмент "входит" в верхний горизонт.**

**# Ищем пересечение с верхним горизонтом.**

**xi, yi = Intersection(x\_prev, y\_prev, x\_curr, y\_curr, top, canvas\_class)**

**else: # flag\_prev == -1 (flag\_prev нулю (0) не может быть равен, т.к. мы обработали это выше).**

**# Сегмент "входит" в нижний горизонт.**

**# Ищем пересечение с нижним горизонтом.**

**xi, yi = Intersection(x\_prev, y\_prev, x\_curr, y\_curr, bottom, canvas\_class)**

**# Отображаем сегмент, от предыдущий точки, до пересечения.**

**canvas\_class.draw\_line([x\_prev, y\_prev], [xi, yi])**

**Horizon(x\_prev, y\_prev, xi, yi, canvas\_class)**

**else:**

**if flag\_curr == 1:**

**if flag\_prev == 0:**

**# Сегмент "выходит" из верхнего горизонта.**

**# Ищем пересечение с верхним горизонтом.**

**xi, yi = Intersection(x\_prev, y\_prev, x\_curr, y\_curr, top, canvas\_class)**

**# Отображаем сегмент от пересечения до текущей точки.**

**canvas\_class.draw\_line([xi, yi], [x\_curr, y\_curr])**

**Horizon(xi, yi, x\_curr, y\_curr, canvas\_class)**

**else: # flag\_prev == -1**

**# Сегмент начинается с точки, ниже нижнего горизонта**

**# И заканчивается в точке выше верхнего горизонта.**

**# Нужно искать 2 пересечения.**

**# Первое пересечение с нижним горизонтом.**

**xi, yi = Intersection(x\_prev, y\_prev, x\_curr, y\_curr, bottom, canvas\_class)**

**# Отображаем сегмент от предыдущей то пересечения.**

**canvas\_class.draw\_line([x\_prev, y\_prev], [xi, yi])**

**Horizon(x\_prev, y\_prev, xi, yi, canvas\_class)**

**# Второе пересечение с верхним горизонтом.**

**xi, yi = Intersection(x\_prev, y\_prev, x\_curr, y\_curr, top, canvas\_class)**

**# Отображаем сегмент от пересечения до текущей.**

**canvas\_class.draw\_line([xi, yi], [x\_curr, y\_curr])**

**Horizon(xi, yi, x\_curr, y\_curr, canvas\_class)**

**else: # flag\_curr == -1**

**if flag\_prev == 0:**

**# Сегмент "выходит" из нижнего горизонта.**

**# Ищем пересечение с нижним горизонтом.**

**xi, yi = Intersection(x\_prev, y\_prev, x\_curr, y\_curr, bottom, canvas\_class)**

**canvas\_class.draw\_line([xi, yi], [x\_curr, y\_curr])**

**Horizon(xi, yi, x\_curr, y\_curr, canvas\_class)**

**else:**

**# Сегмент начинается с точки, выше верхнего горизонта**

**# И заканчивается в точке ниже нижнего горизонта.**

**# Нужно искать 2 пересечения.**

**# Первое пересечение с верхним горизонтом.**

**xi, yi = Intersection(x\_prev, y\_prev, x\_curr, y\_curr, top, canvas\_class)**

**# Отображаем сегмент от предыдущей до пересечения.**

**canvas\_class.draw\_line([x\_prev, y\_prev], [xi, yi])**

**Horizon(x\_prev, y\_prev, xi, yi, canvas\_class)**

**# Ищем второе пересечение с нижним горизонтом.**

**xi, yi = Intersection(x\_prev, y\_prev, x\_curr, y\_curr, bottom, canvas\_class)**

**# Отображаем сегмент от пересечения до текущей.**

**canvas\_class.draw\_line([xi, yi], [x\_curr, y\_curr])**

**Horizon(xi, yi, x\_curr, y\_curr, canvas\_class)**

**x\_prev, y\_prev = x\_curr, y\_curr**

**flag\_prev = flag\_curr**

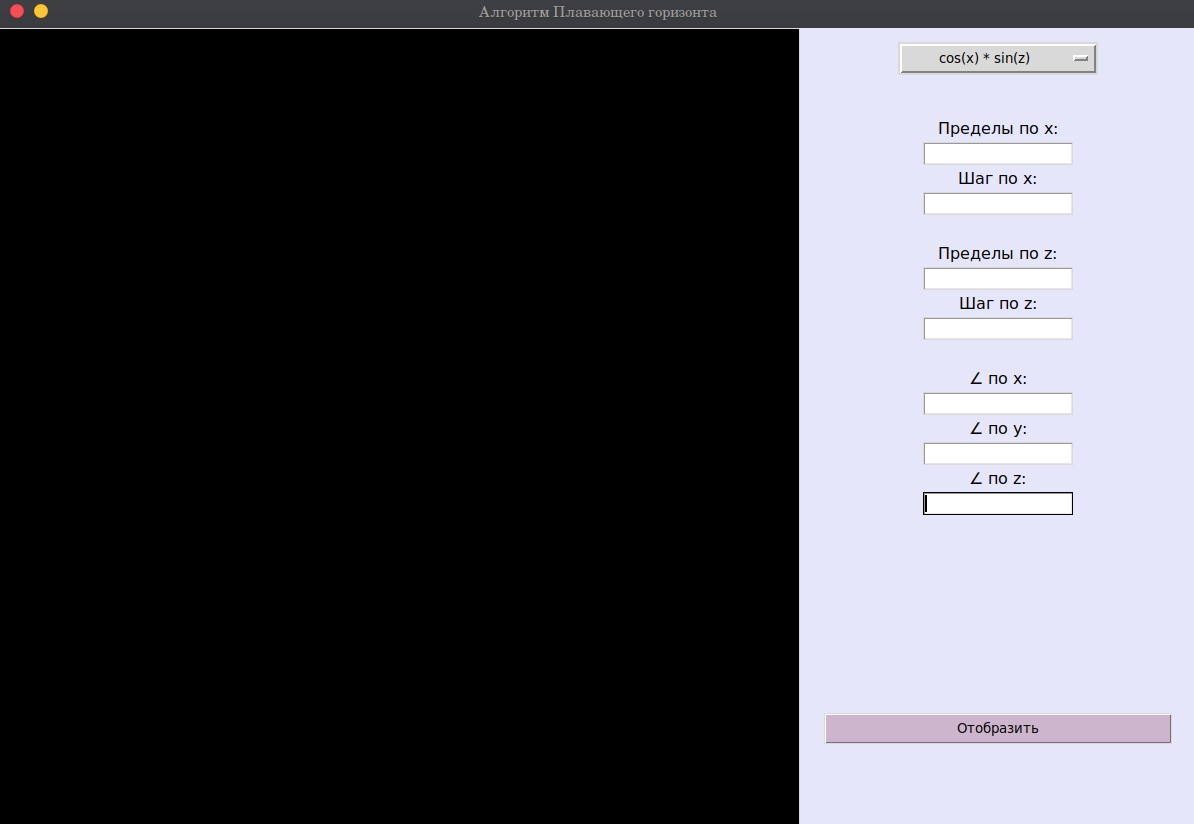
**x += x\_step**

**x\_right, y\_right = Side(x\_prev, y\_prev, x\_right, y\_right,canvas\_class)**

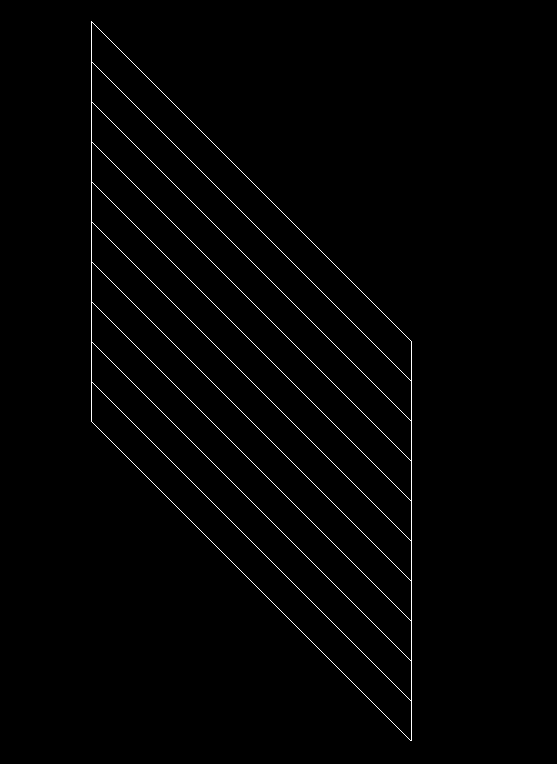
**z -= z\_step**

**Результат:**

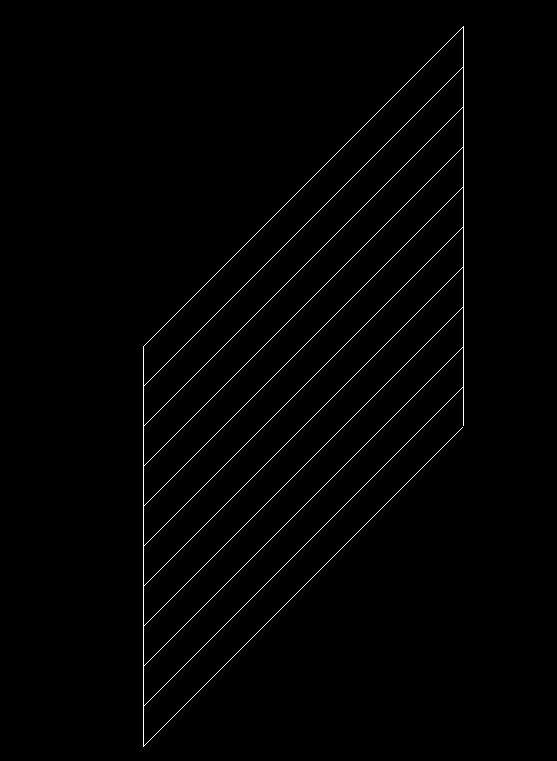
Интерфейс:



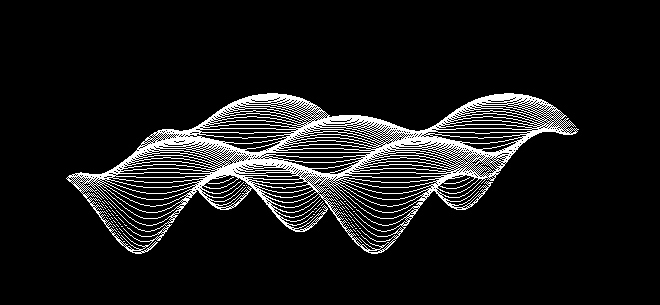
y = z + x



Повернули вокруг x на 180°



y = exp(cos(x) \* sin(z))



y = cos(x) \* sin(z)

