|  |  |
| --- | --- |
| **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 (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 10 |

**Название:**

Построение трехмерных поверхностей алгоритмом плавающего горизонта

**Дисциплина:** Компьютерная графика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-43Б |  | 08.06.2020 | С.С. Кононенко |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | А.В. Куров |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

**1. Цель работы**

Изучение и программная реализация алгоритма плавающего горизонта построения трехмерных поверхностей.

**2. 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.

**3. Теоретический материал**

**Алгоритм плавающего горизонта** – построение поверхностей заданных неявным уравнением вида *F(x, y, z) = 0*.

Заданная поверхность отсекается плоскостями, перпендикулярными оси Z. Наблюдатель распологается на положительной координате оси Z и смотрит в сторону начала координат. Уравнение плоскостей – Z = const. Следует заметить, что в данном случае задача решается в пространстве изображений.

Этапы алгоритма:

1. Рассматриваемая поверхность рассекается плоскостями, перпендикулярными плоскостями, перпендикулярными оси Z. В каждом сечении получается кривая, описываемая уравнением   
   **y = f(x, z=const)** или **x = q(y, z=const)**.
2. Полученные кривые проецируются на плоскость Z и изобразить видимые части каждой кривой. Изображение стоится начиная с кривой, полученной в ближайшем к наблюдателю сечении. Кривая, полученная в сечении ближайшей плоскостью, является видимой. Вторая кривая расположена либо выше первой, либо ниже первой, следовательно, она тоже будет видима. Начиная с третьей кривой решается задача определения видимости точек кривой.

Решение задачи определения видимости кривой:

1. Определяются границы построения поверхности: сначала определяются границы по X, затем – по Z. Шаг по Х в лучшем случае = 1 пиксел.
2. Изображаются первые две кривые, полученные сечением.
3. Изображаются остальные кривые. Участок кривой будет видимым, если он располагается выше верхней кривой или ниже нижней кривой.

Работа алгоритма сводится к вычислению значения у(х, z=const) и сравнению y(x, z=const) > Ymax(x)*.* Если y(x, z=const) > Ymax, то Ymax = y(x, z=const). В противном случае надо проверить y(x, z=const) < Ymin, если условие выполняется, то Ymin(x) = y(x, z=const) – процедура поддержания горизонта. Точка текущей кривой невидима, если она находится между двумя горизонтами.

Если видимость точек одинакова, то кривая между ними либо видима полностью, либо полностью невидима. Однако если точки имеют разные видимости, то можно чертить участок кривой исходя из видимости первой точки. Однако у такого решения есть большой минус: где-то будут изображаться в действительности невидимые участки, или же не будут изображаться видимые участки.

Для решения находится точка пересечения горизонта и текущей кривой и изображается видимый участок. С учетом того, что dx мало и функция гладкая, то функция аппроксимируется отрезком.

При повороте можно столкнуться с проблемой: отдельные прямые не будут восприниматься как поверхность. Качество изображения улучшится, если начальные точки кривых соединить отрезками. Эти отрезки называются левым и правым боковыми ребрами.

**Алгоритм:**

1. Обработать левое боковое ребро. Если точка является первой точкой кривой и лежит на первой кривой, то запомнить ее (обозначим за P). Если точка принадлежит не первой кривой, то соединить eе с точкой P и запомнить в качестве точки p.
2. Для каждой точки кривой сделать следующие действия:
   1. Определить видимость точки;
   2. Если видимость кривой изменилась, то найти точку пересечения кривой с горизонтом.
   3. Если текущий сегмент кривой видим, то изобразить его полностью.
   4. Если видимость изменилась, то если текущая точка невидима, то изобразить участок кривой от предыдущей точки до точки пересечия; если текущая точка видима, то изобразить участок кривой от точки пересечения до текущей точки.
   5. Заполнить массивы верхнего и нижнего горизонта
3. Обработать правое боковое ребро.

**4. Реализация алгоритма**

**def** **predraw\_horizon**(root, fdot, sdot, uphor, lowhor):

**if** fdot[**0**] > sdot[**0**]:

fdot, sdot = sdot, fdot

dx = sdot[**0**] - fdot[**0**]

dy = sdot[**1**] - fdot[**1**]

l = None

**if** dx > dy:

l = dx

**else**:

l = dy

dx /= l

dy /= l

x = fdot[**0**]

y = fdot[**1**]

**for** \_ **in** range(int(l) + **1**):

x = int(round(x))

**if** **not** root.draw\_dot(x, y, uphor, lowhor):

**return**

x += dx

y += dy

**def** **draw\_horizon**(root, func, uphor, lowhor, start, end, step, z):

**def** **f**(x): **return** func(x, z)

prev = None

**for** x **in** arange(start, end + step, step):

cur = trans\_dot(root, [x, f(x), z])

**if** prev:

predraw\_horizon(root, prev, cur, uphor, lowhor)

prev = cur

**def** **fhorizon**(root):

root.reset()

func\_ind = root.funclst.curselection()[**0**]

func = funcs.funcs[root.funclst.get(func\_ind)]

uphor = [**0** **for** \_ **in** range(root.canv\_width)]

lowhor = [root.canv\_height **for** \_ **in** range(root.canv\_width)]

**for** z **in** arange(root.z\_from, root.z\_to + root.z\_step, root.z\_step):

draw\_horizon(root, func, uphor, lowhor, root.x\_from, root.x\_to, root.x\_step, z)

Реализация вспомогательных функций (операции над точками):

**def** **rotate\_mat**(root, mat):

res\_mat = [

[**0**, **0**, **0**, **0**],

[**0**, **0**, **0**, **0**],

[**0**, **0**, **0**, **0**],

[**0**, **0**, **0**, **0**]

]

**for** i **in** range(**4**):

**for** j **in** range(**4**):

**for** k **in** range(**4**):

res\_mat[i][j] += root.mat[i][k] \* mat[k][j]

root.mat = res\_mat

**def** **trans\_dot**(root, dot):

dot.append(**1**)

res\_dot = [**0**, **0**, **0**, **0**]

**for** i **in** range(**4**):

**for** j **in** range(**4**):

res\_dot[i] += dot[j] \* root.mat[j][i]

**for** i **in** range(**3**):

res\_dot[i] \*= root.scale

res\_dot[**0**] += root.canv\_width / **2**

res\_dot[**1**] += root.canv\_height / **2**

**return** res\_dot[:**3**]

**def** **xrotate**(root):

val = float(root.xrotatesb.get()) / **180** \* pi

mat = [

[**1**, **0**, **0**, **0**],

[**0**, cos(val), sin(val), **0**],

[**0**, -sin(val), cos(val), **0**],

[**0**, **0**, **0**, **1**]

]

rotate\_mat(root, mat)

**def** **yrotate**(root):

val = float(root.yrotatesb.get()) / **180** \* pi

mat = [

[cos(val), **0**, -sin(val), **0**],

[**0**, **1**, **0**, **0**],

[sin(val), **0**, cos(val), **0**],

[**0**, **0**, **0**, **1**]

]

rotate\_mat(root, mat)

**def** **zrotate**(root):

val = float(root.yrotatesb.get()) / **180** \* pi

mat = [

[cos(val), sin(val), **0**, **0**],

[-sin(val), cos(val), **0**, **0**],

[**0**, **0**, **1**, **0**],

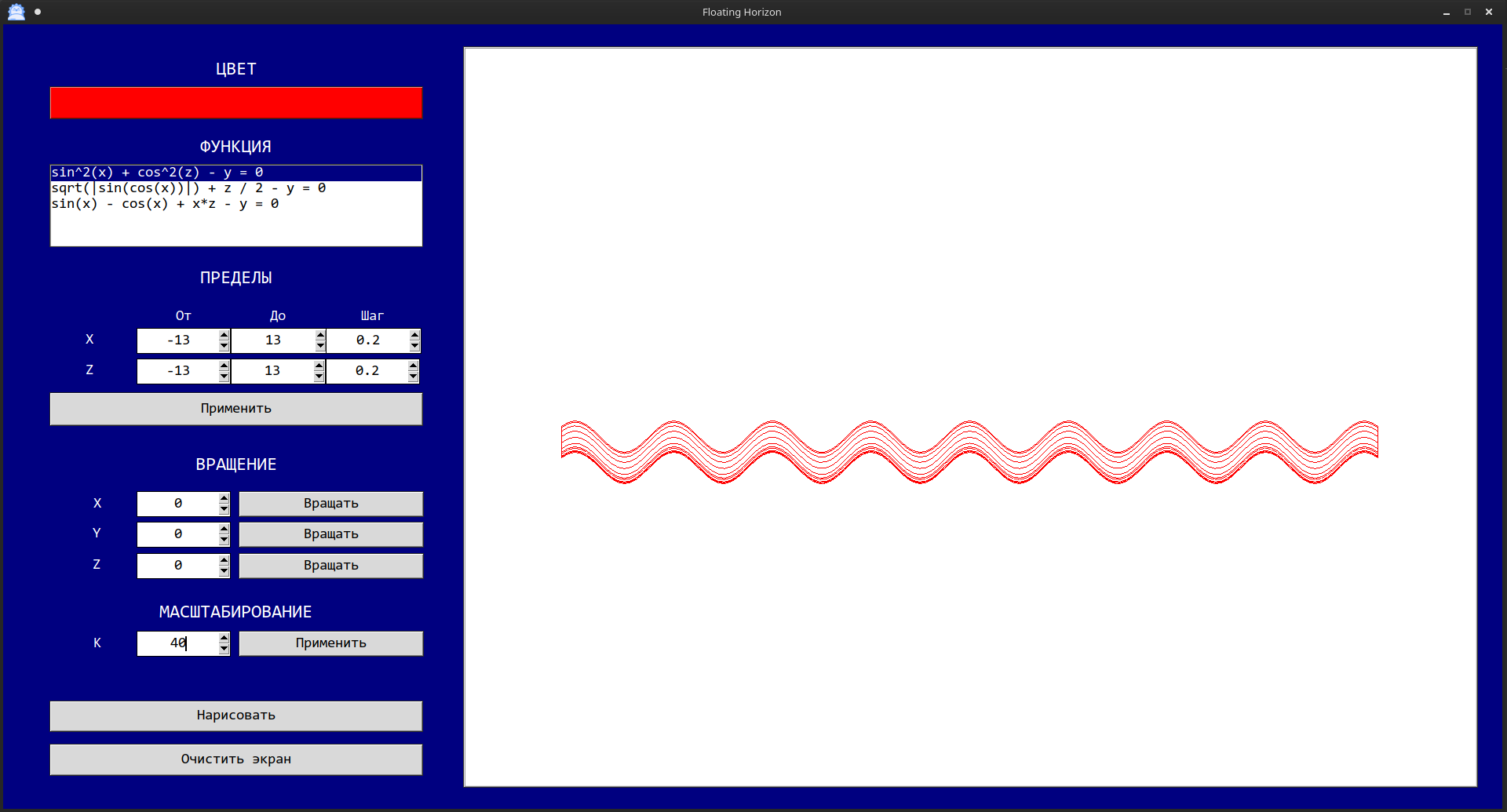
[**0**, **0**, **0**, **1**]

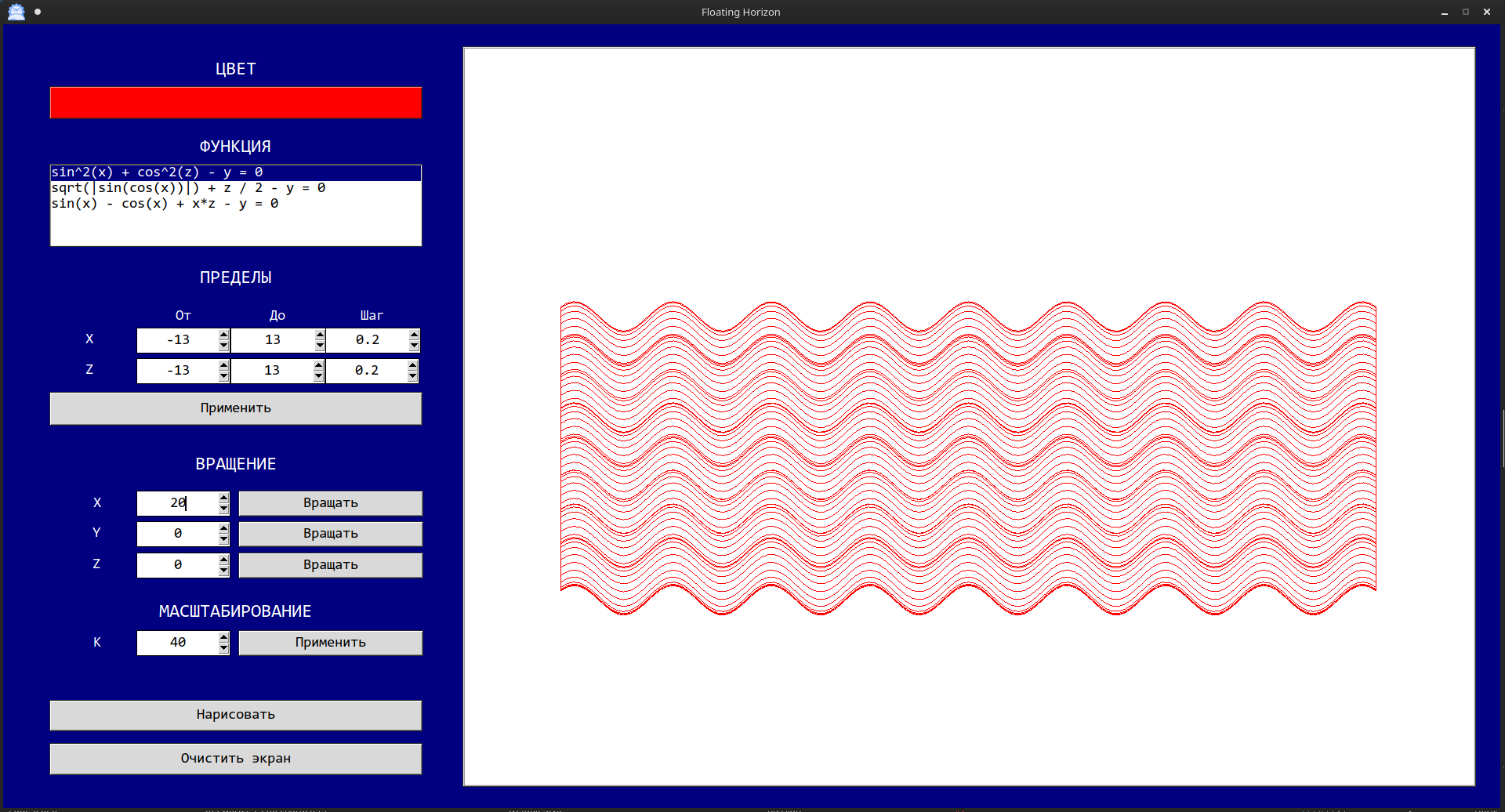
]

rotate\_mat(root, mat)

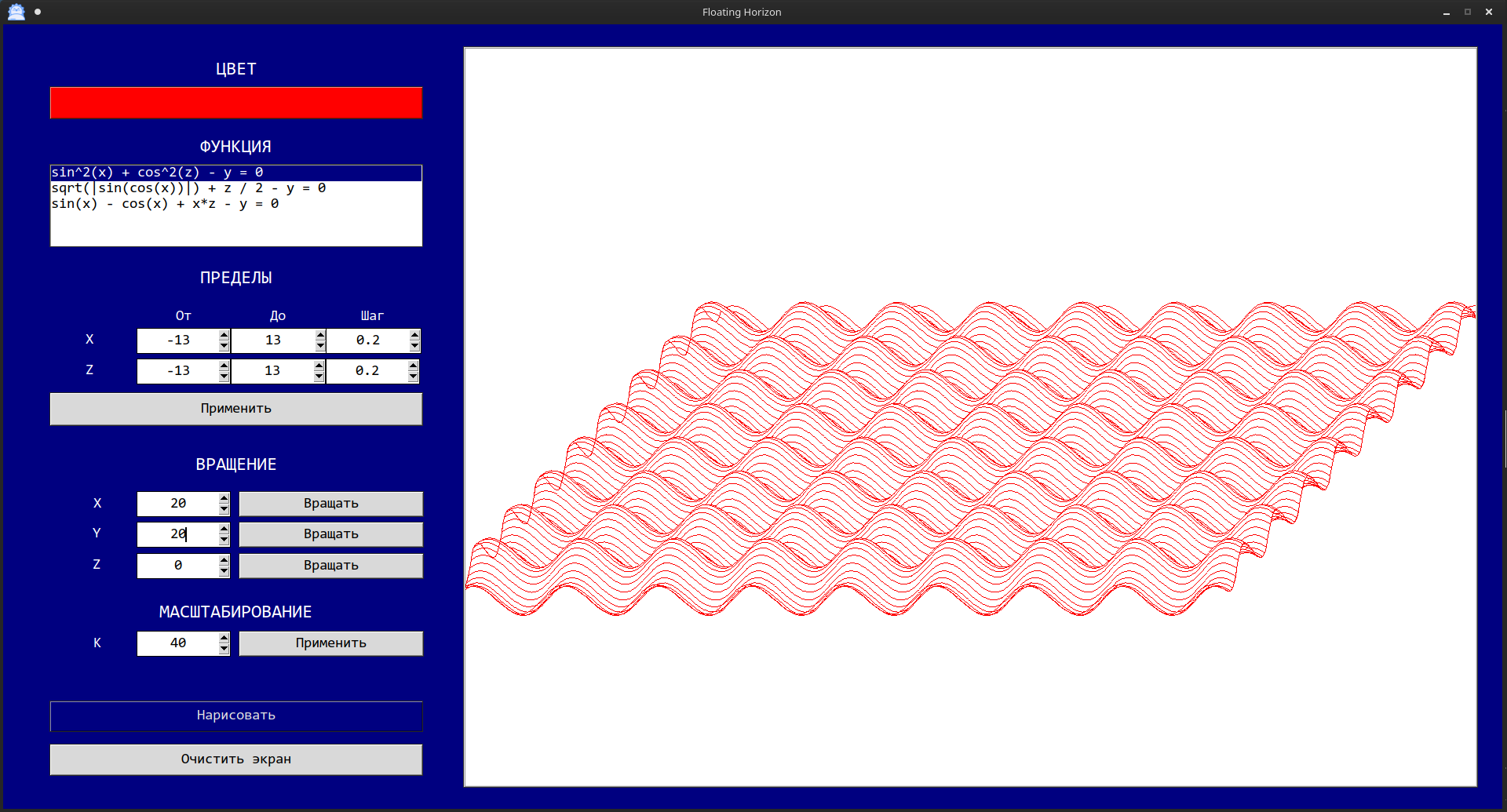
**5. Демонстрация работы программы**

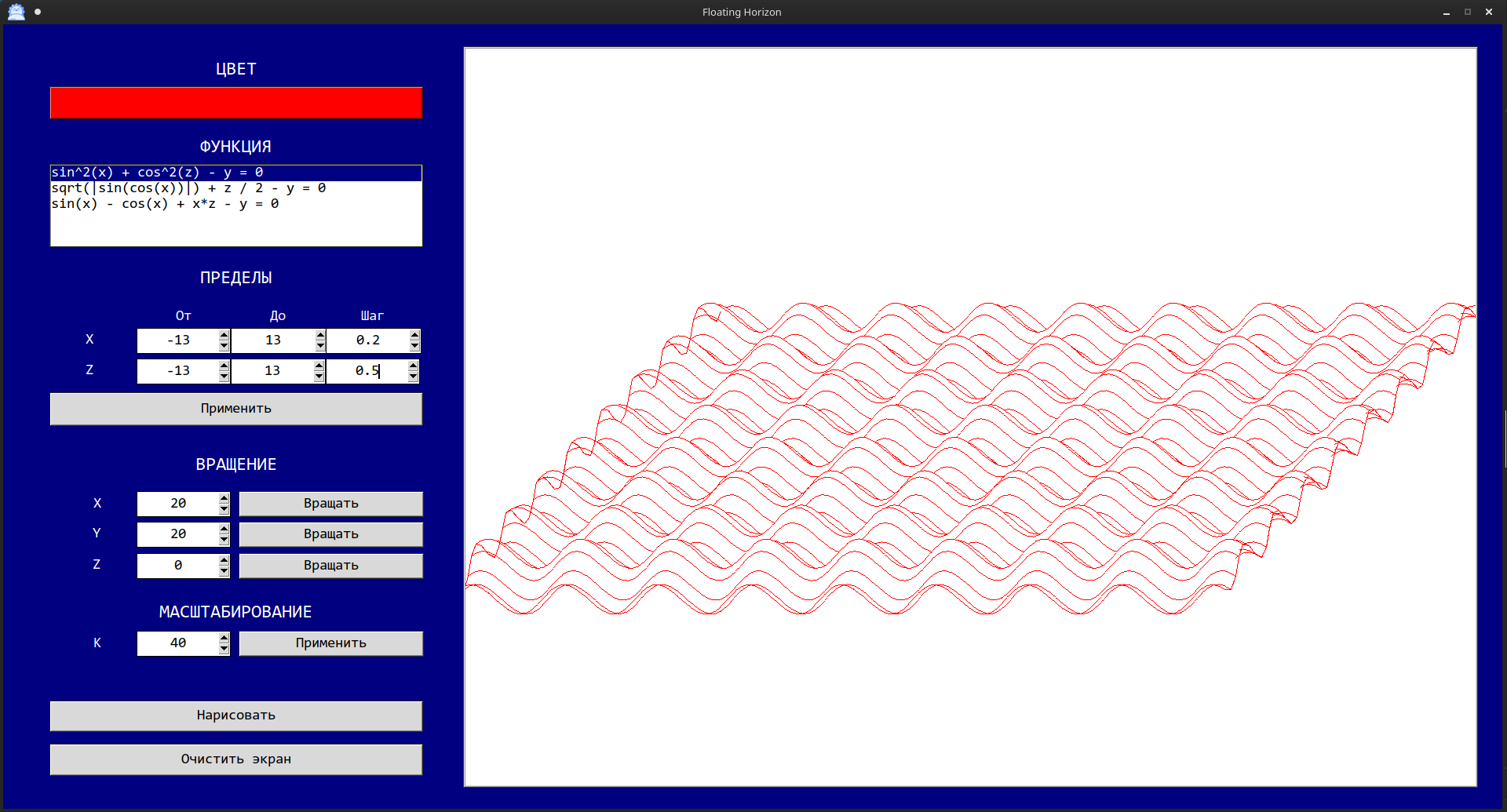
Без вращения, функция sin2(x) + cos2(x) — y = 0:

Вращение относительно оси X:

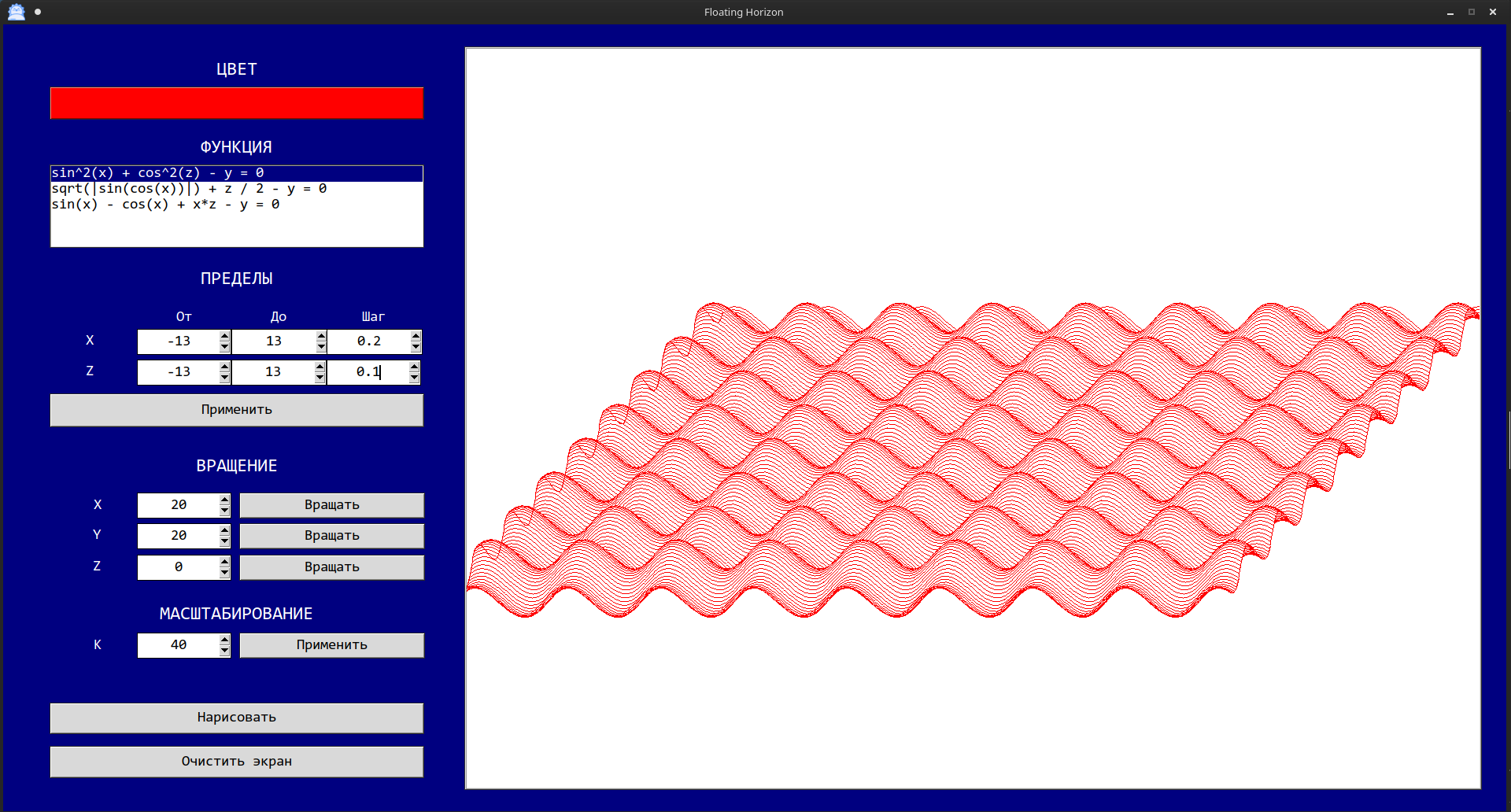


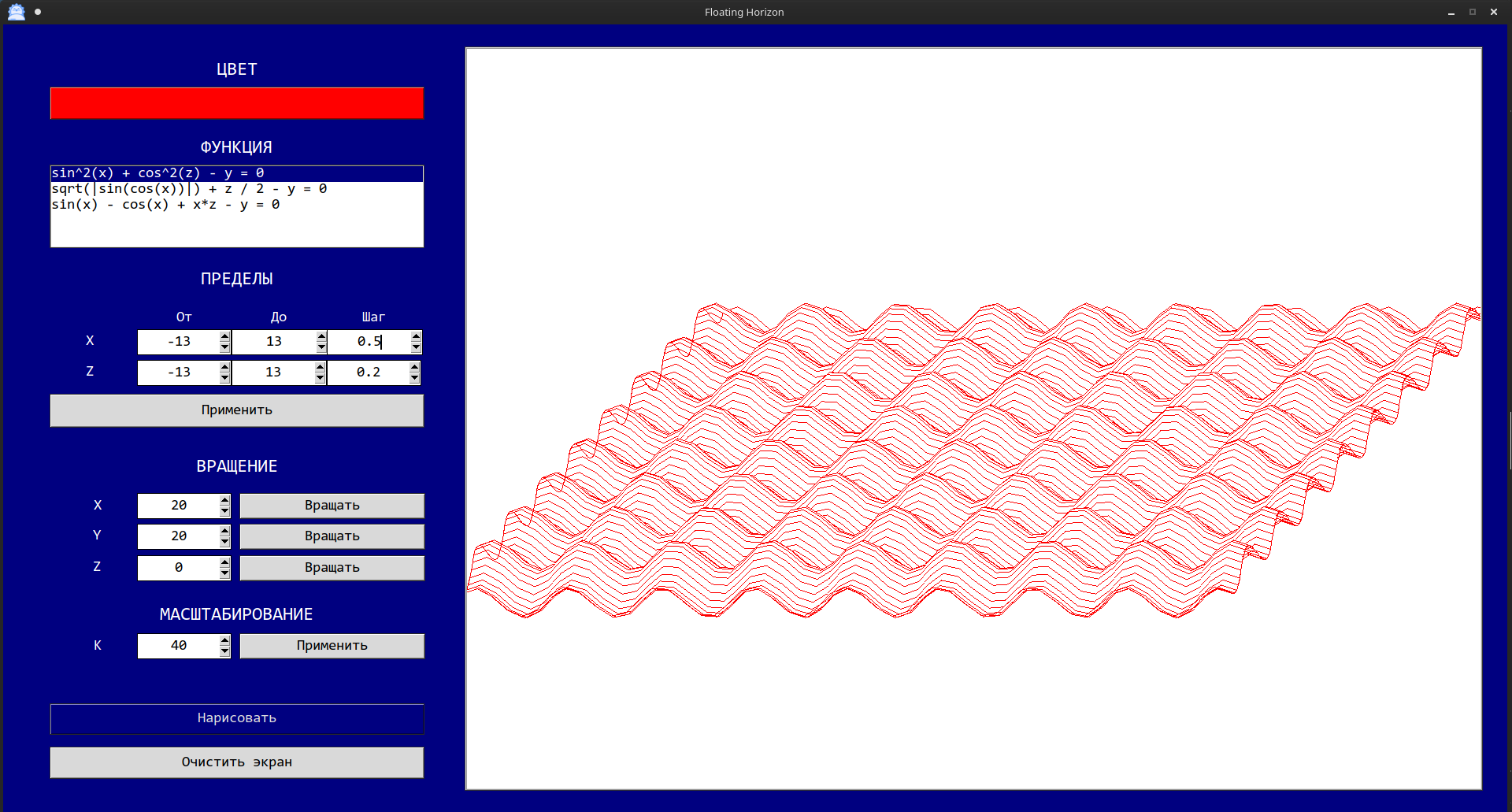
Вращение относительно осей X и Y:

Влияние шага по Z на итоговое изображение (увеличение шага):



Влияние шага по Z на итоговое изображение (уменьшение шага):

Влияние шага по X на итоговое изображение (увеличение шага):



Влияние шага по X на итоговое изображение (уменьшение шага):

