**МГТУ им. Н.Э. Баумана**

**КОМПЬЮТЕРНАЯ ГРАФИКА**

**Лабораторный практикум №10**

***Студент: Нгуен Фыок Санг***

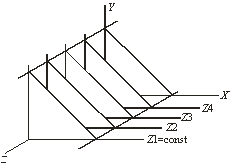
***Группa: ИУ7И-46Б***

***Преподаватель: Куров А. В.***

2020

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

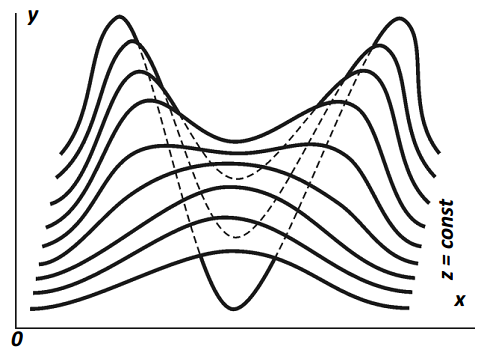
1. **Алгоритм плавающего горизонта:**
   * Алгоритм плавающего горизонта чаше всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде:
   * Главная идея заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат *х*, *у* или *z*.



* + При задании секущих плоскостей постоянными значениями ***z*** поверхность представляется совокупностью кривых, которые описываются функциями y = f(x, z), x = q(y, z), z = const, т.е. y = f(x), x = q(y).
  + Плоскости ***z = const***  сортируют по степени их удаленности от точки наблюдения. Затем поочередно для каждой плоскости, начиная с ближней, строится лежащая на ней кривая.
  + При конкретном значении ***х*** соответствующее значение ***у*** больше или, наоборот, меньше, чем значения ***y*** для всех предыдущих кривых при том же значении ***х***, то текущая кривая полагается видимой в данной точке; в противном случае она считается невидимой.
  + Для хранения максимальных и минимальных значений ***у*** при каждом значении ***х*** используются два массива чисел: ***массив верхнего горизонта*** и ***массив нижнего горизонта.***
    1. точкa видимой, если при заданном значении ***x*** значение ***у*** либо больше соответствующего значения в массиве верхнего горизонта, либо меньше аналогичного значения в массиве нижнего горизонта.

**y(x,z) > ymax(x) либо y(x,z) < ymin(x)**

* + 1. в противном случае она считается невидимой



* + Поочередно для каждой из равноотстоящих друг от друга плоскостей ***z = const***, начиная с ближней от точки наблюдения:

1. Обработать левое боковое ребро:
   1. если точка является первой точкой на первой кривой, то запомнить ее в качестве ***P*** и закончить обработку;
   2. в противном случае:
      1. соединит ее с P, и запомнит в качестве P
2. для каждой точки текущей кривой:
   1. определить видимость точки:
      1. если y(x) > ymax(x) или y(x) < ymin(x), то точка видима,
      2. в противном случае, точка невидима.
   2. Если видимость точки изменилась, то найти точку пересечения кривой с горизонтом
   3. Если текущий сегмент кривой видим, то изобразить его полностю
   4. Если видимость изменилась, то:
      1. Если текущая точка не видима, то изобразить участок кривой от предыдущей точки до точки пересечения
      2. Если текущая точка видима, то изобразить участок кривой от точки пересечения до текущей точки
   5. Заполнить массивы верхнего и нижнего горизонтов
3. Обработать правое боковое ребро:
   1. если точка является последней точкой на первой кривой, то запомнить в качестве ***Q***;
   2. в противном случае: соединит ее с Q и запомнит в качестве Q

**def float\_horizon(max\_width, max\_hight, x\_min, x\_max, x\_step, z\_min, z\_max, z\_step, tx, ty, tz, func, image):**

*# инициализация массивов горизонтов*

**top = {x: 0 for x in range(1, int(max\_width) + 1)}**

**bottom = {x: max\_hight for x in range(1, int(max\_width) + 1)}**

z = z\_max

# Поочередно для каждой из равноотстоящих друг от друга плоскостей z = const,

# начиная с ближней от точки наблюдения

**while z >= z\_min:**

***# Обработать левое боковое ребро***

**x = x\_min**

**y = func(x, z)**

**x, y, z\_tmp = tranform(x, y, z, tx, ty, tz)**

# если точка является первой точкой на первой кривой

**if fabs(z - z\_max)< eps:**

# запомнить ее в качестве P

**P = QPoint(x, y)**

**else:**

# соединит ее с P, и запомнит в качестве P

**top, bottom = add\_line(P.x(), P.y(), x, y, top, bottom, image)**

**P = QPoint(x,y)**

***# для каждой точки текущей кривой:***

**x = x\_min**

**while x <= x\_max:**

**y = func(x, z)**

**x\_curr, y\_curr, z\_curr = tranform(x, y, z, tx, ty, tz)**

**p\_curr = QPoint(x\_curr, y\_curr)**

# определить видимость точки

**isVis\_curr= isVisable(p\_curr, top, bottom)**

**if fabs(x - x\_min) > eps: # x <> x\_min**

# Если видимость точки изменилас

**if (isVis\_curr \* isVis\_prev == 0 and isVis\_curr + isVis\_prev != 0):**

# найти точку пересечения кривой с горизонтом

**if (isVis\_curr + isVis\_prev == 1):**

**I = intersection(p\_prev, p\_curr, top[p\_prev.x()], top[p\_curr.x()])**

**else:**

**I = intersection(p\_prev, p\_curr, bottom[p\_prev.x()], bottom[p\_curr.x()])**

# Если текущая точка не видима

**if (isVis\_curr == 0):**

# изобразить участок кривой от предыдущей точки до точки пересечения, Заполнить массивы верхнего и нижнего горизонтов

**top, bottom = add\_line(p\_prev.x(), p\_prev.y(), I.x(), I.y(), top, bottom, image)**

# Если текущая точка видима

**else**:

# изобразить участок кривой от точки пересечения до текущей точки, Заполнить массивы верхнего и нижнего горизонтов

**top, bottom = add\_line(I.x(), I.y(), p\_curr.x(), p\_curr.y(), top, bottom, image)**

**elif isVis\_curr \* isVis\_prev != 0:**

# то изобразить его полностю, Заполнить массивы верхнего и нижнего горизонтов

**top, bottom = add\_line(p\_prev.x(), p\_prev.y(), p\_curr.x(), p\_curr.y(), top, bottom, image)**

**p\_prev = p\_curr**

**isVis\_prev = isVis\_curr**

**x += x\_step**

***# Обработать правое боковое ребро***

# если точка является последней точкой на первой кривой

**if fabs(z - z\_max) < eps:**

# запомнить в качестве Q

**Q = p\_curr**

**else:**

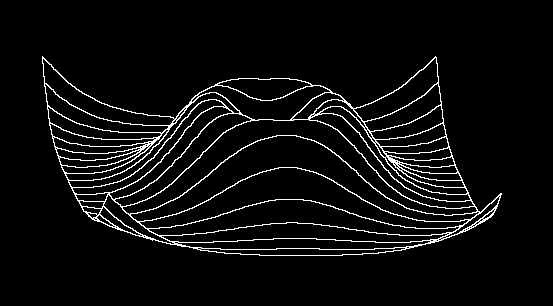
# соединит ее с Q и запомнит в качестве Q

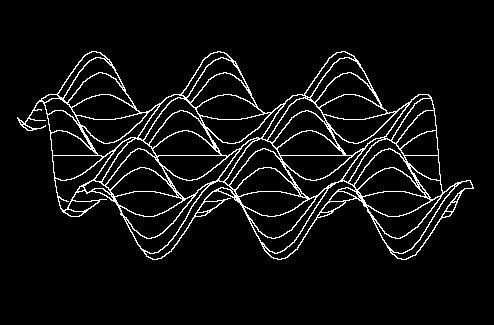
**top, bottom = add\_line(Q.x(), Q.y(), p\_curr.x(), p\_curr.y(), top, bottom, image)**

**Q = p\_curr**

**z -= z\_step**

**return image**

****

****