Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра «САПР»

Лабораторная работа №2

по дисциплине «Компьютерная графика»

Метод Плавающего горизонта

Выполнил: студент гр. −31 qwinmen

Проверил: С. А.

Тамбов,

***Цели и задачи****.*

Реализовать в коде алгоритм плавающего горизонта.

***Решение задачи****.*

Алгоритм плавающего горизонта можно отнести к классу алгоритмов, работающих в пространстве изображения. Алгоритм плавающего горизонта чаше всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде *F(x, у, z) =* 0. Подобные функции возникают во многих приложениях в математике, технике, естественных науках и других дисциплинах [1].

Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат *х*, *у* или *z*.

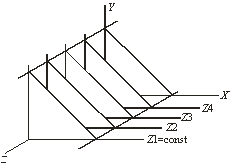
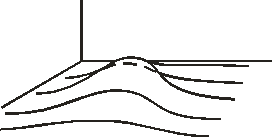
На рис. 1. приведен пример, где указанные параллельные плоскости определяются постоянными значениями *z*. Функция *F(x,у,z)* = 0 сводится к последовательности кривых, лежащих в каждой из этих параллельных плоскостей, например к последовательности *у*=*f(x,z)* или *х=g(у,z)*, где *z* постоянно на каждой из заданных параллельных плоскостей.

Рисунок Секущие плоскости с постоянной координатой.

Итак, поверхность теперь складывается из последовательности кривых, лежащих в каждой из этих плоскостей, как показано на рис. 2.

Здесь предполагается, что полученные кривые являются однозначными функциями независимых переменных. Если спроецировать полученные кривые на плоскость *z* = 0, как показано на рис.3., то сразу становится ясна идея алгоритма удаления невидимых участков исходной поверхности.

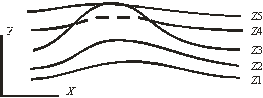


Рисунок 2 Секущие плоскости с постоянной координатой.

Алгоритм сначала упорядочивает плоскости *z =* const по возрастанию расстояния до них от точки наблюдения. Затем для каждой плоскости, начиная с ближайшей к точке наблюдения, строится кривая, лежащая на ней, т. е. для каждого значения координаты *х* в пространстве изображения определяется соответствующее значение *y*. Алгоритм удаления невидимой линии заключается в следующем.

Рисунок 3 Проекция кривых на плоскость z = 0.

Если на текущей плоскости при некотором заданном значении *x* соответствующее значение *у* на кривой больше значения *y* для всех предыдущих кривых при этом значении *x*, то текущая кривая видима в этой точке; в противном случае она невидима.

Невидимые участки показаны пунктиром на рис. 3. Реализация данного алгоритма достаточно проста. Для хранения максимальных значений *y* при каждом значении *x* используется массив, длина которого равна числу различимых точек (разрешению) по оси *x* в пространстве изображения. Значения, хранящиеся в этом массиве, представляют собой текущие значения "горизонта". Поэтому по мере рисования каждой очередной кривой этот горизонт "всплывает". Фактически этот алгоритм удаления невидимых линий работает каждый раз с одной линией.

Алгоритм работает очень хорошо до тех пор, пока какая-нибудь очередная кривая не окажется ниже самой первой из кривых, как показано на рис. 4., а.

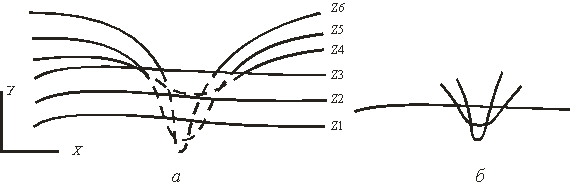
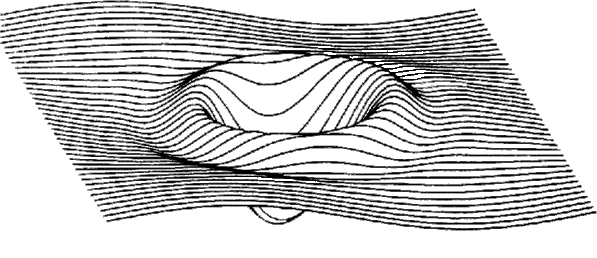
Подобные кривые, естественно, видимы и представляют собой нижнюю сторону исходной поверхности, однако алгоритм будет считать их невидимыми. Нижняя сторона поверхности делается видимой, если модифицировать этот алгоритм, включив в него нижний горизонт, который опускается вниз по ходу работы алгоритма. Это реализуется при помощи второго массива, длина которого равна числу различимых точек по оси *x* в пространстве изображения. Этот массив содержит наименьшие значения *y* для каждого значения *x*. Алгоритм теперь становится таким: если на текущей плоскости при некотором заданном значении *x* соответствующее значение *y* на кривой больше максимума или меньше минимума по *y* для всех предыдущих кривых при этом значении *x*, то текущая кривая видима. В противном случае она невидима.

Рисунок 4 Обработка нижней стороны поверхности.

Полученный результат показан на рис. 4., б.

В изложенном алгоритме предполагается, что значение функции, т. е. *y*, известно для каждого значения *x* в пространстве изображения. Однако если для каждого значения *х* нельзя указать (вычислить) соответствующее ему значение y, то невозможно поддерживать массивы верхнего и нижнего плавающих горизонтов. В таком случае используется линейная интерполяция значений *y* между известными значениями для того, чтобы заполнить массивы верхнего и нижнего плавающих горизонтов.

На рис. 5. показан типичный результат работы алгоритма плавающего горизонта для функции

*y* = (1/5)sin *x* cos *z* – (3/2) cos (7α/4) e(-α), где α = (*x* - )2+(*z*- )2, в интервале (0, 2).

Рисунок 5 Результат работы алгоритма плавающего горизонта.

***Исходный код:***

namespace Шляпа

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

OGlControl.InitializeContexts();

}

/// <summary>

/// Инициализация GL

/// </summary>

private void InitGL()

{

// инициализация Glut

Glut.glutInit();

Glut.glutInitDisplayMode(Glut.GLUT\_RGB | Glut.GLUT\_DOUBLE | Glut.GLUT\_DEPTH);

}

private void Form1\_Load(object sender, EventArgs e)

{

InitGL();

ResizeGL();

}

private static void Red(double[] верхнийГоризонт, double z, int xDist, double mashtab, double[] грань)

{

#region

double k = 0;

for (; k < z; k += 0.004)

{

double index = 0;

for (int j = 0; j < xDist; j++)//идем по [0]--Х-->1024

{

double tmp = (F(j\*0.04, k) \* mashtab);

if (tmp < грань[j])

{

// Gl.glEdgeFlag(Gl.GL\_FALSE);

}

if (tmp > верхнийГоризонт[j])

{

верхнийГоризонт[j] = tmp;

Gl.glColor3d(1, 0, 0);

Gl.glVertex2d(index, верхнийГоризонт[j]);

}

index += 0.3;

}

}

#endregion

}

/// <summary>

/// y = F(x,z) [0..2\*pi]

/// </summary>

private static double F(double x, double z)

{

return 0.2 \* Math.Sin(x) \* Math.Cos(z) -

1.5 \* Math.Cos((7.0 \* (Math.Pow(x - Math.PI, 2.0) + Math.Pow(z - Math.PI, 2.0))) / 6.0)\*

Math.Exp(-(Math.Pow(x - Math.PI, 2.0) + Math.Pow(z - Math.PI, 2.0)));

}

static void Blue(double[] нижнийГоризонт, double z, int xDist, double mashtab, double[] грань)

{

double k = 0.04;

for (; k < z; k += 0.004)

{

double index = 0;

for (int j = 0; j < xDist; j++)

{

double tmp = (F(j \* 0.04, k) \* mashtab);

if(tmp < грань[j])

{

//Gl.glEdgeFlag(Gl.GL\_FALSE);

}

if (tmp < нижнийГоризонт[j])

{

нижнийГоризонт[j] = tmp;

Gl.glColor3d(0, 0, 1);

Gl.glVertex2d(index, нижнийГоризонт[j]);

}

index += 0.3;

}

}

}

/// <summary>

/// Перерисовка компонента GL

/// </summary>

private void ResizeGL()

{

Gl.glViewport(0, 0, OGlControl.Width, OGlControl.Height);

Gl.glLoadIdentity();

Gl.glColor3f(0, 0, 0);//устанавл цвет отрисовки геометрии

// помещаем состояние матрицы в стек матриц

Gl.glPushMatrix();

// устанавливаем размер точек, равный 5 пикселям

Gl.glPointSize(1);

// выполняем перемещение в прострастве по осям X и Y

Gl.glTranslated(0, 8, 0);

var Style = Gl.GL\_LINES;

//---------------------------------------------------------------

int xDist = 148;//Screen.PrimaryScreen.Bounds.Width;

double mashtab = -30;// частоты

double k;

double z = (2 \* Math.PI);// частотаДескрита;// 6.2831/k

double[] верхнийГоризонт = new double[xDist],

нижнийГоризонт = new double[xDist], грань = new double[xDist];

for (int j = 0; j < xDist; j++)

верхнийГоризонт[j] = (F(j \* 0.04, 0) \* mashtab);//

нижнийГоризонт = верхнийГоризонт;

грань = верхнийГоризонт;

Gl.glColor3d(1, 0, 0);

double index = 0;

Gl.glBegin(Gl.GL\_LINE\_STRIP);

foreach (double pointF in верхнийГоризонт)

{

Gl.glVertex2d(index, pointF);

index += 0.3;

}

Gl.glEnd();

//---------------------------------------------------------------

Gl.glBegin(Gl.GL\_POINTS);//Gl.GL\_LINES//Gl.GL\_POINTS//Gl.GL\_LINE\_STRIP

Red(верхнийГоризонт, z, xDist, mashtab, грань);

Gl.glEnd();

Gl.glBegin(Gl.GL\_POINTS);//Gl.GL\_LINES//Gl.GL\_POINTS//Gl.GL\_LINE\_STRIP

Blue(нижнийГоризонт, z, xDist, mashtab, грань);

Gl.glEnd();

// возвращаем матрицу из стека

Gl.glPopMatrix();

// дожидаемся завершения визуализации кадра

Gl.glFlush();

// сигнал для обновление элемента реализующего визуализацию.

OGlControl.Invalidate();//обновить отображаемый кадр

}

private void Form1\_Resize(object sender, EventArgs e)

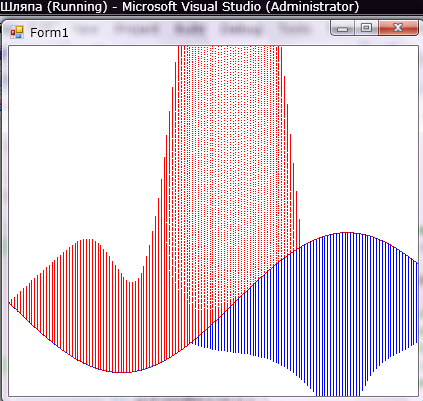
{

ResizeGL();

}

}

}

***Конечный результат:***

Список источников:

1. Интернет ресурс httр://cоmрgraрh.tрu.ru/hоrizоnt.htm