Курсовая работа

по дисциплине "Компьютерная геометрия и графика"

на тему: Построение трехмерной модели вазы

Содержание

1. Постановка задачи

2. Описание используемых алгоритмов

* 1. Построение перспективной проекции
  2. Алгоритм удаления невидимых линий и поверхностей

2.3 Алгоритм получения изменений формы и движения объекта

3. Описание программы

* 1. Обобщенная структурная диаграмма программы
  2. Введенные типы данных и их предназначение
  3. Введенные основные переменные и их предназначение
  4. Текстовое описание основных процедур и функций и их блок-схемы
  5. Алгоритм взаимодействия процедур

4. Тестирование программы

# Список литературы

# Приложение

проекция алгоритм диаграмма переменная

1. Постановка задачи

Целью данной работы является построение трехмерной модели вазы. Моделируемая сцена представляет собой поверхность, образованную полигонами. Координатная модель представляет собой набор координат нескольких окружностей различных радиусов, центром лежащими на вертикальной оси вазы и координаты центра дна вазы.

Для каркасного изображения вазы можно рисовать сетку параллелей и меридианов. Для этого удобно воспользоваться известными формулами параметрического описания. Координаты точек поверхности вазы определяются как функции от двух переменных (параметров) — высоты (h) и долготы (l).

х = R sin l

у = R cos l (1)

z = H h,

где R — радиус соответствующей параллели, l — долгота (от -180° до +180° или от 0° до 360°),h — высота (изменяется от -0,5 до +0,5), Н—высота вазы.

Параллель – это линия, состоящая из точек из точек с постоянной широтой. Меридиан — это линия, представляющая точки с постоянной долготой. В каркасной модели вазы меридианы - это криволинейная линия.

Ваза в данной работе состоит из 30 меридианов и 10 параллелей. Сначала вычисляются мировые координаты вершин стенок вазы. В данной модели их 300. После этого рассчитываются номера вершин полигонов для стенок вазы. Аналогичным образом вычисляются мировые координаты вершины дна вазы и номера полигонов дна вазы.

Далее производится видовое преобразование координат точек плоскостей, то есть выполняется преобразование мировых координат (x, y, z) в экранные координаты (X,Y) с добавлением перспективной проекции.

Для удаления невидимых линий используется Z буфер, в котором сортируются полигоны по удалению от плоскости экрана. Рассчитывается средняя координата Z для каждого полигона и в соответствии с её значением полигоны выводятся на экран, начиная с самых удалённых и заканчивая ближайшими. Тем самым и обеспечивается перекрытие ближними полигонами дальних и отсечение невидимых частей модели.

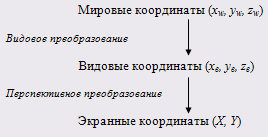
Для расчёта освещённости полигона рассчитывается координаты вектора нормали к нему и, исходя из направления вектора нормали, задаётся цвет заливки для полигона.

2. Описание используемых алгоритмов

2.1 Построение перспективной проекции

Точки в двухмерном и трехмерном пространствах представляются координатами (X, Y) и (х, у, z) соответственно. При необходимости получения перспективной проекции задается большое количество точек P(x, у, z), принадлежащих объекту, для которых предстоит вычислить координаты точек изображения Р'(Х, Y) на картинке. Для этого нужно только преобразовать координаты точки Р из мировых координат (х, у, z) в экранные координаты (X, Y) ее центральной проекции Р'. Предположим, что экран расположен между объектом и глазом Е. Для каждой точки Р объекта прямая линия РЕ пересекает экран в точке Р'.

Это отображение удобно выполнять в два этапа. Первый этап - видовое преобразованием - точка Р остается на своем месте, но система мировых координат переходит в систему видовых координат. Второй этап - перспективное преобразование. Это точное преобразование точки Р в точку Р' объединенное с переходом из системы трехмерных видовых координат в систему двухмерных экранных координат:



Для выполнения видовых преобразований должны быть заданы точка наблюдения, совпадающая с глазом, и объект. Будет удобно, если начало ее координат располагается где-то вблизи центра объекта, поскольку объект наблюдается в направлении от Е к О. Пусть точка наблюдения Е будет задана в сферических координатах по отношению к мировым координатам. То есть мировые координаты (точки Е) могут быть вычислены по формулам:

xe=  sin cos )

ye= sin  sin (2)

ze=  cos 

Обозначения сферических координат схематически изображены на рисунке 2.

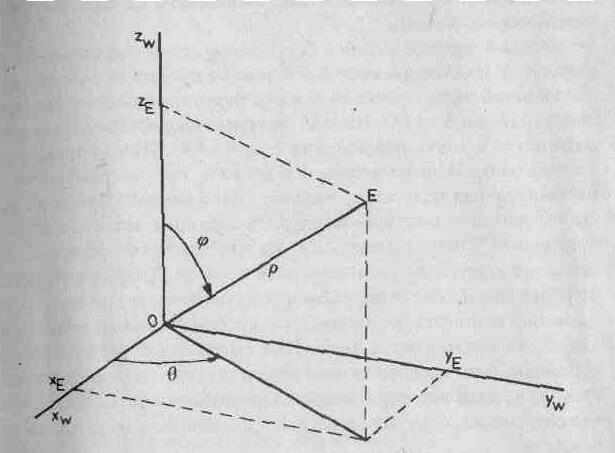


Рисунок 1 - Сферические координаты точки наблюдения Е

Видовое преобразование может быть записано в форме

[хe ye ze 1]=[хw уw zw 1]V, (3)

где V - матрица видового преобразования размерами 4х4.

Матрица V, полученная в процессе видового преобразования, выглядит следующим образом:

(4)



Сейчас можно использовать видовые координаты хe и уe просто игнорируя координату ze для получения ортогональной проекции. Каждая точка Р объекта проецируется в точку Р' проведением прямой линии из точки Р перпендикулярно плоскости, определяемой осями х и у. Эту проекцию можно также считать перспективной картинкой, которая была бы получена при удалении точки наблюдения в бесконечность. Параллельные линии остаются параллельными и на картинке, полученной при ортогональном проецировании.

Теперь для построения перспективной проекции рассмотрим непосредственно перспективное преобразование.

На рисунке 2 выбрана точка Q, видовые координаты которой равны (0, 0, d) для некоторого положительного числа d. Плоскость z = d определяет экран, который будет использоваться следующим образом. Экран — это плоскость, проходящая через точку Q и перпендикулярная оси z. Экранные координаты определяются привязкой начала к точке Q, а оси Х и Y имеют такие же направления, как оси х и у соответственно. Для каждой точки объекта Р точка изображения Р' определяется как точка пересечения прямой линии РЕ и экрана. Чтобы упростить рисунок, будем считать, что точка Р имеет нулевую у-координату. Но все последующие уравнения для вычисления ее у-координаты также пригодны и для любых других значений координаты X. На рисунке треугольники EPR и EP'Q подобны. Следовательно,

(5)



Отсюда будем иметь

(6)



X = dx/Z

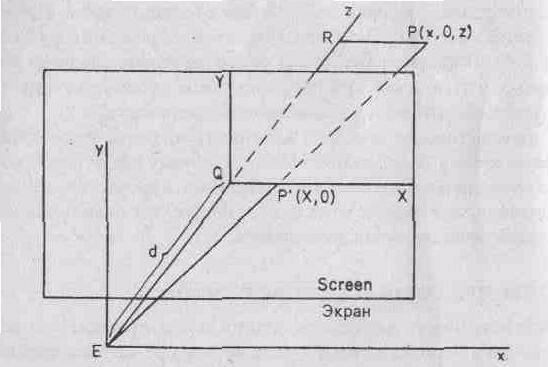


Рисунок 2 - Экран и видовые координаты

Аналогично можем получить:

Y = dy/Z. (7)

Ранее было введено предположение, что точка О начала системы мировых координат примерно совпадает с центром объекта. Поскольку ось z видовой системы координат совпадает с прямой линией ЕО, которая пересекает экран в точке Q, то начало Q системы экранных координат будет находиться в центре изображения. Если бы мы потребовали, чтобы это начало координат располагалось в нижнем левом углу экрана, а размеры экрана составляли 2с1 по горизонтали и 2с2 по вертикали, то нужно дополнить формулы вычисления X и Y:

Х = d x/z + с1

Y = d y/z + c2 (8)

2.2 Алгоритм удаления невидимых линий и поверхностей

Для изображения поверхности вазы с удалением невидимых точек в аксонометрической проекции можно воспользоваться таким свойством: видимыми являются точки с неотрицательным значением координаты Z в системе видовых координат. При этом центр видовых координат (X,Y,Z) совпадает с центром сферы, плоскость X0Y является плоскостью проецирования, а ось Z направлена на камеру (наблюдателя).

Сортировка граней по глубине

При вращении объекта наблюдается интересный артефакт – грани рисуются в одном и том же порядке, поэтому в какой-то момент времени объект отображается в необычном ракурсе. Для устранения этого эффекта применяется сортировка граней по глубине, так называемый "алгоритм художника".

Если для каждой грани вычислить координаты средней точки, то в общем случае, чем меньше координата z этой точки, тем больше вероятность что эта грань не будет перекрыта другой при отображении на экране (предполагая, что наблюдатель находится где-то на отрицательной полуоси z). Вариантов сортировки может быть несколько – можно сортировать грани в массиве polygons, можно сортировать указатели на грани в дополнительном массиве.

"Алгоритм художника" отличается чрезмерной простотой реализации, и, как следствие, недостаточно эффективен. В самом деле, если 3D сцена состоит из большого количества объектов, в среднем половина полигонов будет не видна в текущем кадре. А значит, их отображение на экране и не требуется. Недостаток "алгоритма художника" состоит в том, что отображаются все полигоны. Для сокращения числа треугольников можно вычислить для каждой грани вектор нормали, что позволит отбросить грани, повернутые от нас.

Определение нормали к поверхности

Нормаль к поверхности представляет ее локальную кривизну, а следовательно, и направление зеркального отражения. Если известно аналитическое описание поверхности, то нормаль вычисляется непосредственно. Но для многих поверхностей бывает задана лишь их полигональная аппроксимация. Зная уравнение каждой грани, можно найти направление внешней нормали.

Во многих алгоритмах удаления невидимых линий и поверхностей используются только ребра или вершины, поэтому, для того чтобы объединить их с моделью освещения, необходимо знать приближенное значение нормали на ребрах и в вершинах. Пусть заданы уравнения плоскостей полигональных граней, тогда нормаль к их общей вершине равна среднему значению нормалей ко всем многоугольникам, сходящимся в этой вершине.

Если уравнения плоскостей не заданы, то нормаль к вершине можно определить, усредняя векторные произведения всех ребер, пересекающихся в вершине.

Следует обратить внимание на то, что необходимы только внешние нормали. Кроме того, если полученный вектор не нормируется, то его величина зависит от количества и площади конкретных многоугольников, а также от количества и длины конкретных ребер. Сильнее проявляется влияние многоугольников с большей площадью и более длинных ребер.

Когда нормаль к поверхности используется для определения интенсивности и для изображения объекта или сцены выполняется перспективное преобразование, то нормаль следует вычислять до перспективного деления. В противном случае направление нормали будет искажено, а это приведет к тому, что интенсивность, задаваемая моделью освещения, будет определена неправильно.

В данной работе для каждой грани при инициализации объекта находиться вектор нормали. Далее этот вектор приводится к единичной длине, так как для определения косинуса угла между векторами проще использовать единичные векторы. Для этого нужно определить длину вектора и разделить на нее каждую координату.

Его координаты сохраняются в структуре соответствующей грани. Кроме того, при каждом повороте сцены векторы нормали нужно пересчитывать.

Координаты вектора нормали в общем виде:

nx = A.y ⋅(B.z - C.z) + B.y ⋅(C.z - A.z) + C.y ⋅(A.z - B.z)

ny = A.z ⋅(B.x - C.x) + B.z ⋅(C.x - A.x) + C.z ⋅(A.x - B.x) (9)

nz = A.x ⋅(B.y - C.y) + B.x ⋅(C.y - A.y) + C.x ⋅(A.y - B.y)

Отсюда невидимость грани определяется следующим образом:

По углу между направлением взгляда и нормалью можно определить повернута грань к нам или от нас. Если направление взгляда совпадает с вектором (0, 0, 1), то косинус угла между векторами равен nz (координата z нормали), и если nz < 0 – грань видна.

## 2.3 Алгоритм получения изменений формы и движения объекта

Вращение объекта в разные стороны, в том числе и вокруг своей оси осуществляется с помощью изменения координат точки наблюдения (ρ, θ, ψ)

Для повышения скорости создания изображения часто используется следующая технология – объект помещается в память, а затем копируется в нужное место экрана. Этот процесс можно разбить на следующие этапы:

1. закрашивание картинкой фона первой поверхности в памяти
2. отрисовка изображения в памяти поверх фона
3. копирование изображения из памяти на экран
4. вычисление новых координат объектов
5. снова п. 1

# 3. Описание программы

# 3.1 Обобщенная структурная диаграмма программы



Рисунок 3 - Обобщенная структурная диаграмма программы

Программа условно разделена на блоки, каждый из которых выполняют свои собственные функции. В начале программы идут название модуля и модули, использующие данный модуль. Далее в объявлении типов, постоянных, переменных описываются два новых типа записи, постоянные, а также процедуры и переменные, которые применяются в данной программе.

Процедура начальной установки инициализирует переменные требуемыми значениями, устанавливает количество полигонов, начальное положение, заполняет массив точек исходными значениями.

Процедура преобразования отвечает за перевод мировых координат в видовые, а потом из видовых в экранные.

Процедура рисования вызывает последовательно блок преобразования координат и процедуру составления полигонов из точек. Затем выполняет рисование полигонов в промежуточном буфере и совмещение результата с буфером отображения.

Процедура составления полигонов выполняет преобразования координат в экранные, составление полигонов из массива точек и сортировку полученных полигонов в Z буфере с учётом их расположения по глубине сцены.

Блок вращения меняет значения углов обзора в соответствии с нажатиями на клавиши позиционирования и вызывает процедуру отрисовки.

3.2 Введенные типы данных и их предназначение

При реализации программы были определены некоторые специализированные типы данных (таблица 1).

Таблица 1 - Типы данных.

|  |  |  |
| --- | --- | --- |
| Тип | Значение | Предназначение |
| T3DPoint | record | Содержит 3 координаты точки в трехмерном пространстве, координаты точки на экране, номер точки и нормаль к этой точке имеющая три координаты. |
| TPolygon | record | Содержит указатели на вершины полигона, то есть содержит переменные типа P3DPoint. нормаль к полигону, номер полигона, цвет и среднюю точку. |

3.3 Введенные основные переменные и их предназначение

При реализации программы были определены некоторые переменные (таблица 2).

Таблица 1 - Переменные

|  |  |  |
| --- | --- | --- |
| Переменная | Значение | Предназначение |
| w,v1 | array [1.. nPoint] of T3DPoint | мировые, видовые координаты вершин |
| v | array [1.. nPoint] of TPoint | экранные координаты точек |
| polygons | array [1.. nTPolygon] of TPolygon | массив объектов фигуры |
| teta | extended | Отвечает за угол поворота относительно оси Y. Имеет фиксированное значение teta:=pi/9. |
| phi | extended | Отвечает за угол поворота относительно оси X. Имеет фиксированное значение phi:=pi\*4/3. |
| buf | TBitmap | Вспомогательная поверхность в ней создается движение сферы |
| blink\_buf | TBitmap | Вспомогательная поверхность для быстрого закрашивания фоном (хранит в себе фон изображения) |
| A, B, C, D | переменные типа word типа TPolygon (record) | Определяют номера вершин четырехугольника в полигоне. Диапазон значений - 0.. nTPolygon |
| ro | real | Определяет расстояние от объекта наблюдения до фигуры. Имеет фиксированное значение |

3.4 Текстовое описание основных процедур и функций и их блок-схемы

Процедура FormCreate

Задаются углы поворота, радиусы паралеллей, их количество и высота вазы. Вычисляются видовые координаты вершин вазы (пересечения параллелей и меридианов). Рассчитывается количество полигонов. Создаётся фон типа TBitmap, на котором будет рисоваться фигура (рисунок 4).



Процедура FormKeyDown

В этой процедуре отслеживаются нажатые клавиши с последующим выполнений соответствующих команд.

При помощи процедуры KeyDown задается угол поворота объекта. При нажатии клавиш Up, Down, Left, Right объект поворачивается в заданном направлении (рисунок 5).



Рисунок 5 – Блок-схема процедуры FormKeyDown

Процедура Sort

Для каждого полигона высчитывается вектор нормали и приводится к единичной форме; сортировка массива полигонов по минимальным значениям средних координат (рисунок6).



Рисунок 6 – Блок-схема процедуры Sort

Процедура ViewTransformation

В этой процедуре вычисляются видовые и экранные координаты точек, задаются полигоны. В процедуре мировые координаты переводятся в видовые, в свою очередь которые преобразовываются в экранные координаты точек (рисунок7).



Рисунок 7 – Блок-схема процедуры ViewTransformation

Процедура Draw

Поочередная прорисовка полигонов, то есть который грань дальше от нас прорисовывается первой и т.д (рисунок 8).



Рисунок 8 – Блок-схема процедуры Draw

Функция tone

Функция, задающая плоскость цвета.

В зависимости от угла между направлением взгляда и нормалью цвет должен изменяться. Значение координаты nz нормали для видимой грани изменяется в диапазоне [-1, 0), цвет грани задан в виде RGB компонентов. И, следовательно, для получения цвета грани нужно умножить каждую компоненту на абсолютное значение nz. Для получения компонент цвета воспользовались функциями GetRValue, GetGValue, GetBValue.

3.5 Алгоритм взаимодействия процедур



Рисунок 9 – Алгоритм взаимодействия процедур

4. Тестирование программы

Интерфейс программы представляет собой форму, на которой представлена ваза. Форма имеет фон, для лучшего восприятия.

После запуска программы на экране не наблюдается ничего. Для наблюдения эффектов предлагается использовать следующие кнопки клавиатуры: ВНИЗ: Поворот вокруг оси x вниз; ВЛЕВО: Поворот вокруг оси y влево; ВВЕРХ: Поворот вокруг оси x вверх; ВПРАВО: Поворот вокруг оси y вправо.

Вид окна программы представлен на рисунке 10.

При нажатии на клавиши "влево", "вправо", "вверх", "вниз" происходит соответствующее перемещение фигуры.



Рисунок 10 - Вид окна программы

В ходе тестирования программа работала стабильно, не вызывала появления сообщений об ошибках.

Список литературы

1. Порев В.Н. Компьютерная графика – СПб.: БХВ – Петербург, 2002. – 432 с.: ил.
2. Шикин А.В., Боресков А.В. Компьютерная графика. Полигональные модели. – М.: ДИАЛОГ – МИФИ, 2001. – 464с.
3. Л. Аммерал Принципы программирования в машинной графике. Пер. с англ. – М.: "Сол Систем", 1992. – 224 с.: ил.

Приложение

Листинг программы

unit prog;

interface

uses

Windows,Messages,SysUtils,Variants,Classes,Graphics,Controls,Forms,

Dialogs,StdCtrls,Math;

type

TForm1 = class(TForm)

procedure KeyDown(Sender:TObject;var Key:Word;Shift:TShiftState);

procedure FormCreate(Sender:TObject);

procedure FormPaint(Sender:TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

T3DPoint=record

x,y,z:extended

end;

TPolygon=record

A,B,C,D:word;

clr:TColor;

end;

const

step=30;//количество точек на одной параллели

nPOINT=step\*10+1;

nPOLYGON=step\*10+step;

col1=255+255\*$100+204\*$10000; //цвет стенок(полигонов) вазы

col2=209+154\*$100+65\*$10000; //цвет дна вазы

var

Form1:TForm1;

Buf,Blinc\_Buf:TBitMap;

polygons:array[1..nPolygon] of Tpolygon;

w,v1:array[1..nPOINT] of T3DPoint;//мировые (world),видовые (view) координаты вершин

v:array[1..nPOINT] of TPoint;//экранные (screen) координаты вершин

S:array[1..nPOLYGON] of extended;

n:array[1..nPOLYGON] of T3DPoint; //массив нормалей

teta,phi,d,ro,r:real;

implementation

{$R \*.dfm}

function tone(clr:TColor;nz:extended):TColor; //плоскость цвета

begin

tone:=rgb(round(nz\*GetRValue(clr)),

round(nz\*GetGValue(clr)),

round(nz\*GetBValue(clr)))

end;

procedure ViewTransformation;

var i: integer;

begin

for i:=1 to nPOINT Do

begin

v1[i].x:=Round(w[i].x\*(-sin(teta))+w[i].y\*(cos(teta)));

v1[i].y:=Round(w[i].x\*(-cos(phi)\*cos(teta))-w[i].y\*(cos(phi)\*sin(teta))+

w[i].z\*( sin(phi)));

v1[i].z:=Round(w[i].x\*(-sin(phi)\*cos(teta))-w[i].y\*( sin(phi)\*sin(teta))-

w[i].z\*(cos(phi)))+ro;

v[i].x:=Round(Form1.ClientWidth div 2+v1[i].x );

v[i].y:=Round(Form1.ClientHeight div 2+v1[i].y);

end;

end;

procedure Sort;

var

i:integer;

begin

for i:=1 to nPOLYGON do

begin

s[i]:=(v1[polygons[i].a].z+v1[polygons[i].b].z+v1[polygons[i].c].z)/3;

//координаты вектора нормали

n[i].x:=v1[polygons[i].a].y\*(v1[polygons[i].b].z-v1[polygons[i].c].z)+

v1[polygons[i].b].y\*(v1[polygons[i].c].z-v1[polygons[i].a].z)+

v1[polygons[i].c].y\*(v1[polygons[i].a].z-v1[polygons[i].b].z);

n[i].y:=v1[polygons[i].a].z\*(v1[polygons[i].b].x-v1[polygons[i].c].x)+

v1[polygons[i].b].z\*(v1[polygons[i].c].x-v1[polygons[i].a].x)+

v1[polygons[i].c].z\*(v1[polygons[i].a].x-v1[polygons[i].b].x);

n[i].z:=v1[polygons[i].a].x\*(v1[polygons[i].b].y-v1[polygons[i].c].y)+

v1[polygons[i].b].x\*(v1[polygons[i].c].y-v1[polygons[i].a].y)+

v1[polygons[i].c].x\*(v1[polygons[i].a].y-v1[polygons[i].b].y);

if (sqrt(sqr(n[i].x)+sqr(n[i].y)+sqr(n[i].z)))<>0 then

n[i].z:=n[i].z/(sqrt(sqr(n[i].x)+sqr(n[i].y)+sqr(n[i].z)))

else n[i].z:=0;

end;

end;

procedure Draw;

var

j,i1,i:integer;

f: real;

begin

Sort;

f:=0;

buf.Canvas.Draw(0,0,blinc\_buf); //рисуем в основном буфере фон

for i1:=1 to nPOLYGON do

begin

//Опред.невидимости грани (слегка затеняем внутреннюю поверхность)

if (n[i1].z>0) then n[i1].z:=n[i1].z\*0.60;

end;

for i1:=1 to nPOLYGON do

begin

for i := 1 to nPOLYGON do

if s[i]>f then begin j:=i;f:=s[i];end;

with polygons[j] do

begin

Buf.Canvas.Brush.Color:=tone(clr,ABS(n[j].z)); //цвет полигона

Buf.Canvas.Pen.Color:=tone(clr,ABS(n[j].z\*0.96));//цвет границ полигонов

Buf.Canvas.Polygon([v[A],v[B],v[C],v[D]]); //прорисовка полигона

end;

s[j]:=0;

f:=0;

end;

Form1.Canvas.Draw(0,0,buf); //прорисовываем буфер на экране(форме)

end;

procedure TForm1.KeyDown(Sender:TObject;var Key:Word;Shift:TShiftState);

begin

CASE KEY of

VK\_UP: phi:=phi+pi\*0.05;

VK\_DOWN: phi:=phi-pi\*0.05;

VK\_LEFT: teta:=teta+pi\*0.03;

VK\_RIGHT: teta:=teta-pi\*0.03;

end;

ViewTransformation;

Draw;

end;

procedure TForm1.FormCreate(Sender: TObject);

var

B,L,H,nn:integer;

dL:Real;

const

Rz:array[0..9] of integer = (50,75,90,94,88,74,54,42,40,46);//радиусы параллелей

begin

H:=250; // высота вазы

d:=200; //масштаб

ro:=500; //перспектива

teta:=pi/9; //угол поворота

phi:=pi\*4/3; //угол поворота

// вершины вазы

for B:=0 to 9 do

begin

for L:=0 to step-1 do

begin

dL:=L\*Pi\*(360/step)/180;

w[B\*step+L+1].x:=Rz[B]\*sin(dL); //Вычисление мировых координат

w[B\*step+L+1].y:=Rz[B]\*cos(dL);

w[B\*step+L+1].z:=H/10\*B-H/2;

end;

end;

// полигоны вазы

nn:=1;

for B:=1 to 9 do

begin

for L:=0 to step-2 do

begin

polygons[nn].A:=(B-1)\*step+L+1;

polygons[nn].B:=(B-1)\*step+L+2;

polygons[nn].C:=B\*step+L+2;

polygons[nn].D:=B\*step+L+1;

polygons[nn].clr:=col1;

nn:=nn+1;

end;

polygons[nn].A:=B\*step;

polygons[nn].B:=(B-1)\*step+1;

polygons[nn].C:=B\*step+1;

polygons[nn].D:=(B+1)\*step;

polygons[nn].clr:=col1;

nn:=nn+1;

end;

// вершина дна вазы

w[nPOINT].x:=0;w[nPOINT].y:=0;w[nPOINT].z:=-H/2;

// полигоны дна вазы

for L:=0 to step-2 do

begin

polygons[L+nPOINT].A:=L+2;

polygons[L+nPOINT].B:=L+1;

polygons[L+nPOINT].C:=nPOINT;

polygons[L+nPOINT].D:=nPOINT;

polygons[L+nPOINT].clr:=col2;

end;

polygons[nPOLYGON].A:=1;

polygons[nPOLYGON].B:=step;

polygons[nPOLYGON].C:=nPOINT;

polygons[nPOLYGON].D:=nPOINT;

polygons[nPOLYGON].clr:=col2;

// буфер

buf:=TBitmap.Create;

buf.Width:=Form1.ClientWidth;

buf.Height:=Form1.ClientHeight;

// фон

blinc\_buf:=TBitmap.Create;

blinc\_buf.Width:=Form1.ClientWidth;

blinc\_buf.Height:= Form1.ClientHeight;

blinc\_buf.Canvas.Rectangle(0,0,Form1.ClientWidth,Form1.ClientHeight);

blinc\_buf.LoadFromFile('./background.bmp');

end;

procedure TForm1.FormPaint(Sender: TObject);

begin

// ViewTransformation;

// Draw;

end;

end.