**Оглавление**

[Main.pas 3](#_Toc359188818)

[Модуль TCameraClass 6](#_Toc359188819)

[Модуль TObjectClass 9](#_Toc359188820)

[Модуль TSceneClass. 13](#_Toc359188821)

[Модуль TRobertClass 19](#_Toc359188822)

Идея построения сцены заключается в перемещении всех объектов относительно точки обзора, если в предыдущей задаче точка обзора была внутри треугольника то здесь точку обзора можно менять.

# Main.pas

Управляющий (основной) модуль программы

**uses Crt,GraphABC,TCameraClass,TSceneClass;**

Подключаем описанные модуль к основному модулю также для возможности чтения нажатия клавиш подключаем crt

**var** Раздел описания переменных

**Scene:TScene;** Scene переменная типа Scene которая описана в модуле TSceneClass”

**Camera:TCamera;** Camera переменная типа TCamera “которая описана в м. TCameraClass”

**key:char;** Её мы используем для хранения кода нажатой клавиш

**begin** начало работы нашей программы

**Scene:=TScene.Create(clWhite,clRed,1500,1000);** Т.к переменная Scene пустая нам необходимо придать ей начальное значение для этого мы пользуемся конструктором класса «constructor» Класса. Входящие параметры конструктора являются 1 свет фона, 2 цвет линий из которых рисуется объект, 3и4 устанавливают высоту и ширину окна.

**Camera:=TCamera.Create;** Аналогично вызывает конструктор для класса Camera только без вводных параметров.

**Camera.SetupMonitor(-150,200);** Вызывает метод (метод – это термин использующийся в Объектно ориентированном программировании (ООП)) в данном случае метод вызывается для Класса TCamera копией которой является переменная Camera в ней и описан эта процедура. 1 параметр является значение переменной Zkp вторым Zk где Zpr- Это размещение Экрана в трёхмерной области. Zk точка от куда мы смотрим на нашу сцену )

**Scene.Position\_Scene(10,150,150,150,150);** Как Вы видите по названию данная процедура создаёт объекты в необходимых нам координатах а точнее пирамида в Центре и по кругу размещаются кубы. Процедура имеет 5 вводных параметров:

1. Кол-во объектов которые будут размещены в в сцене кубы+одна пирамида по условию задачи нам нужно число 5 т.к 4 куба +1 пирамида в центе.
2. 2, 3- высота и ширина кубов
3. 4, 5-высота и ширина пирамиды

**// Scene.CreateSingleObject('Cube',100,100);** //Рyramid And Сube

Вот эта процедура используется для создания 1 пирамиды или 1 куба в центре нашего экрана. Это только для последней задачи.

**Scene.MSObject[1].center;**

**Camera.X:=Round(Scene.MSObject[1].TcenterX);**

**Camera.Y:=Round(Scene.MSObject[1].TcenterY);**

**Camera.Z:=Round(Scene.MSObject[1].TcenterZ);**

Верхние четыре строки нужны но не обязательны они просто ставят точку куда мы смотри в центр первого объекта которым является пирамида. Желательно оставлять для удобства

**//Static:**

**Camera.SetupSingleCamera( 60, 'Z');**

**Scene.Flight(Camera);**

**Camera.SetupSingleCamera( 120, 'Y');**

**Scene.Flight(Camera);**

**//Static end;**

Используется для получения статической картинки всё так же как и в предыдущей задачи поворот на 2 угла. Сначала на 60 по Z потом на 120 по Y.Но учтите вызов метода Camera.SetupSingleCamera только придаёт некоторым переменным необходимые значения само же перемещение и прорисовка объектов выполняется уже в методе Flight «Полёт»

**Camera.step:=5;**

Используется для уточнения шага с которым мы будем вращать объекты

**Camera.Setting\_interfase(10,Scene.GCHeight-30);**

Этот метод устанавливает положение вспомогательной информации на экране А точнее положение точки обзора.

**While(True)Do Begin** Начало бесконечного цикла))

**key:=readkey();** переменной key присваиваем значение нажатой нами клавиши

**case key of** если нажатая клавиша есть в списке то что то произойдёт

**'q': Camera.SetupSingleCamera( 5, 'X');**

**'a': Camera.SetupSingleCamera(-5, 'X');**

**'w': Camera.SetupSingleCamera( 5,'Y');**

**'s': Camera.SetupSingleCamera(-5,'Y');**

**'e': Camera.SetupSingleCamera( 5,'Z');**

**'d': Camera.SetupSingleCamera(-5,'Z');**

Самое время описать что же всё таки нажимать q,a –вращение по оси X, за исключением что первая буква вращает в одну строну а другая в противоположную.

**'z': begin**

**window(0,0,20,20);**

**textout(10,10,' ');**

**textout(10,25,' ');**

**writeln('Введите новые координаты точки обзора:');**

**Readln(Camera.x,Camera.y,Camera.z);**

**clrscr;**

**end;**

Нажатие кнопки “Z” приведёт нас в меню для смены точки осмотра произойдёт это так:

1-процедура window() она описана в crt. Создаёт небольшое окошко в правом верхнем углу.textout нарисует почистит место для вывода меню Дважды.write выведет приветствие и скажет что же нам делать) clrscr очистит наше окошко)) и Всё будет чистенько

Учтите здесь мы только придаёт значения переменным которые описаны в классе Camera больше мы ничего тут не делаем. Так Фикция активности. Как вы догадались переменные в классе Camera X,Y,Z- они устанавливают точку куда мы смотрим.

**chr(72): begin**

**Camera.Zk:=Camera.Zk+5;**

**Camera.SetupSingleCamera( 0,'0');**

**Scene.Flight(Camera);**

**end;**

Это нажатие работаем только в том случае если вы нажали стрелочку вверх код которой символизирует число 72 смотрите таблицу кодов клавиш. При нажатии стрелочки увеличивается значение переменной Zk которая находится в классе TCamera тут же вызывает метод из класса Cemera с названием SetupSinglrCemera в ней мы устанавливаем значение угла =0 и значение оси тоже 0 только для того что бы выполнить перерисовку объектов сцены. Действие этой кнопки не будет видно только до тех пор пока мы не нажмём кнопку «2» о ней скажу ниже.

Метод Flight выполняет перерисовку сцены

**chr(80): begin Camera.Zk:=Camera.Zk-5;Camera.SetupSingleCamera( 0,'0');Scene.Flight(Camera);end;**

Аналогичная процедура только в этом случаем мы нажимаем кнопку стрелочка вниз. И значение переменной уменьшается

**'1': begin**

**Scene.Screening:=not(Scene.Screening);**

**Scene.DrawMassivRebock(false);**

**Camera.SetupSingleCamera( 0,'0');**

**Scene.Flight(Camera);**

**end;**

Нажатие кнопки «1» изменит значение переменной Screening «Экранирование» если она истина то будет включена процедура которая вкл\откл экранирование. Как Скажем позже.

**'2': begin**

**Scene.perspect:=not(Scene.perspect);**

**Camera.SetupSingleCamera( 0,'0');**

**Scene.Flight(Camera);**

**end;**

А Вот наша долгожданная кнопка «2» она работает аналогичным образом как экранирование.

**chr(32): Scene.Flight(Camera);**

код 32 вызывает метод перерисовки все1 сцены с настроенными переменными которые входят в переменной Camera Она и посылается в качестве вводного параметра в процедуру полёт

**end;** конец Case

**end;** конец Цикла

т.к цикл будет работать пока истина истина то тут мы никогда не будем)

**end.** Конец программы

# Модуль TCameraClass

В этой главе нам предстоит создать совой модуль в котором будет размещён пользовательский тип данных именуемый class. Так Же вы увидите как можно перенести массивную часть кода в другой файл и подключать его при необходимости к вашей задаче. Такой подход значительно упрощает поиск необходимых процедур и переменных.

**unit TCameraClass;**

Название нашего модуля именно это мы и должны писать при подключении его к программе

**Interface** тип интерфейса, необходимо прописать)) является сессией интерфейса модуля она пустая но без неё не запуститься)

**uses GraphABC;**

Вот подключаем графический модуль) он нужен в этом модуле только для того чтобы выводить информацию о точке куда мы смотрим а точнее процедура textout описана именно в этом модуле.

**implementation**

Аналогичный парамент как и Interface без него модуль тоже не Фурыкает.

Смотрим ниже там мы видим type TClass = class читается так. Создаём пользовательский тип данных именуемый TClass типа class именно class даёт возможность расширить использование нашего типа. А Точнее можно писать процедурки)) именуемые методами.

**type TCamera = class**

**X,Y,Z:integer;** Хранят значение точки обзора.

**XRot,YRot,ZRot:boolean;** Флаги осей из них 0 или 1 истина всегда)

**Step:integer; //**шаг с которым вы вращаем нашу сцену это именно та переменная которую мы вызывали в Main.pas она только имела вид Camera.step первое слово Camera говорила что она находится в классе TCamera.

**Access:boolean;** Если она истина то мы отображаем подсказки для пользователя. Это информация о точке обзора.

**Zk,Zpr:real;** Точка от куда смотрим в перспективе и где экран расположен

**Private** указываем тип доступа к переменным расположенным ниже если Приват то означает что мы их можем использовать только внутри нашего класса дальше они не уйдут их просто не получится вызвать.

**st:string;**

**X0,Y0:integer;**

Эти переменные чисто для вывод интерфейста камеры)

**Public**

Тип доступа к переменным Публичный виден по всюду и всякому.

**constructor Create;**

первое слово конструктор он при его вызове переменной типа TCamera при его вызове п переменной появится полная копия данного класса Camera в котором и будут доступны наши переменные Camera:=TCamera.Create; вот так вот вызывается конструктор который и сделает копию класса а заодно и присвоит некоторым переменным начальные значения как я сделал ниже.

**begin**

**x:=250;**

**y:=250;**

Начальное значение центра экрана будет уместно если экран 500х500 в противном случае если тот кусок кода который ставит точку в середину первого объекта не был удалён то они пере запишутся на нормальный центр экрана

**Zk:=-90;**

**Zpr:=100;**

Начальное положение экрана и точки обзора Подобрал наугад если честно работает да и ладно. Можете поменять значения может лучше обзор найдёте.

**z:=0;**

а эта переменная нужна для .. да не Нужна она просто надо её на всякий случай обнулить так как она играет очень важную роль при перспективизации.

**Access:=true;**

О Ней я говарил выше просто теперь она истина)

**Step:=5;** Тоже говорил шаг 5.

**XRot:=True;**

**YRot:=False;**

**ZRot:=False;**

Изначально доступен поворот по оси Х

**end;** конец конструтора.

Теперь описывает методы.

**procedure Setting\_interfase(X,Y:integer);**

**begin**

**X0:=X;Y0:=Y;**

**end;**

Согласен процедура тупая но всё же очень удобная целых 2 строчки экономит))

**procedure SetupMonitor(Z1,Z2:real); begin Zk:=Z1; Zpr:=Z2; end;**

Устанавливаем значение переменны экрана и точки обзора

**procedure SetupSingleCamera(Step\_loka:integer;Line:String);**

**begin**

**textout(x0,y0,' ');**

**St:=' X='+IntToStr(X)+' Y='+IntToStr(Y)+' Z='+IntToStr(Z);**

**textout(x0,y0-15,St);**

**if (Line='X') then begin**

**XRot:=True;**

**YRot:=False;**

**ZRot:=False;**

**if (Access and (Step\_loka>0) )then textout(X0,Y0,' OX');**

**if (Access and (Step\_loka<0) ) then textout(X0,Y0,'-OX');**

**end;**

**if (Line='Y') then begin**

**XRot:=False;**

**YRot:=True;**

**ZRot:=False;**

**if (Access and (Step\_loka>0) ) then textout(X0,Y0,' OY');**

**if (Access and (Step\_loka<0) ) then textout(X0,Y0,'-OY');**

**end;**

**if (Line='Z') then begin**

**XRot:=False;**

**YRot:=False;**

**ZRot:=True;**

**if (Access and (Step\_loka>0) ) then textout(X0,Y0,' OZ');**

**if (Access and (Step\_loka<0) ) then textout(X0,Y0,'-OZ');**

**end;**

**if (Line='0') then begin**

**XRot:=False;**

**YRot:=False;**

**ZRot:=False;**

**if (Access and (Step\_loka>0) ) then textout(X0,Y0,' OZ');**

**if (Access and (Step\_loka<0) ) then textout(X0,Y0,'-OZ');**

**end;**

**Step:=Step\_loka;**

**end;**

**end;//Конец класса Камеры**

Вот тут я скажу о процедуре SetupSingleCamera она упрощает работу с тремя переменными Xrot, Yrot,Zrit в зависимости что написано то тот набор значений флагов. Из них как я говорил раньше 0 ил 1 должен быть истиным. Это на тот случай чтобы 2 переменные подряд никогда небыли истины.

Textout(X0,Y0,’ ‘) 2 первых координаты куда выводить текст то что выводим.

**end.**

# Модуль TObjectClass

В этом модуле мы будем описывать наши объекты которые мы будем размещать по сцене.

Большая часть обозначений вам известна из предыдущего модуля так. Что мы это упустим.

**unit TObjectClass;**

**interface**

**uses GraphABC,TRobertClass;**

**implementation**

**type Matrix = array[1..100,1..100]of Real;**

Тип Matrix тоже самое что двемерный массив array[1..100,1..100] of Real;

Внимание!!

Вот тут мы для начало мы опишем всё то что нам пригодится в верхнем случае в классе TCamera мы описывали процедуры внутри класса тут же мы сделаем будем это делать в разных местах. Просто так удобнее.

**type**

**TObject = class**

**MatrixObject:Matrix;** переменная типа Matrix

**BMatrixObject:Matrix;** переменная типа Matrix

**Size:integer;** хранит кол-во вершин в объекте

**Robert:TRobert;** Экранирование оно вам не нужно) Так Для красоты. Но Всё же оно тоже описано в модуле и так же подключается к нашей программе удалять нельзя только потому что рисуется из этого класса ну как рисуется так как рисование всё ещё происходит с помощью плоскостей в классе Роберт и находится тот массив флагов которые говорят что нам рисовать а что не рисовать Ну это будет ниже заметно).

**TcenterX,TcenterY,TcenterZ:real;**

Переменные хранящие значение центра Объекта. Вы их видели в Main.pas.

**BuferDraw:boolean;**

Вот это вожно! Если включена т.е истина то рисование объекта будет произвидиться их переменной MatrixObject иначе BMatrixObject. Ну всё это тоже ниже.

**constructor Create(ObjectType:String;Width,Height:Real);**

Надеюсь что такое конструктор говорить не надо скажу лишь о вводных параметрах.

Дословно тип объекта строковая переменная что напишете то и будет создано)) Ну Конечно то что есть в описании)).

**procedure Center;**

Находит центр объекта и сохраняет его в переменные TCenterX..Y..Z

**procedure Draw(Color:integer);**

Рисует этот объект цветом Color.

**procedure Rect(v1,v2,v3,v4:byte);**

Рисует Прямоугольник по вершинам.

**procedure Treg(v1,v2,v3:byte);**

Рисует Треугольник по вершинам т.к у нас может быть только треугольник и Прямоугольник именно эти плоскости мы и будем рисовать

**procedure Add(S1:Matrix);begin BMatrixObject:=S1;end;**

Она тупа но идеально) Присваивает значение буферной матрице) из вводного параметра.

**procedure App(S1:Matrix);begin MatrixObject:=S1; end;**

Тоже самое) но Тольок оригинальной а не буферной.

**end;**

Теперь настал тот момент когда мы сможем описать наши процедуры. Для того чтобы описать процедуру написанную в классе необходимо указать первым тип что это процедура или Функция ну а может конструктор. Procedure, function, constructor.

Второ1 параметр имя класса От куда мы описываем процедуры потом ставится точка и имя процедуры с параметрами параметры должны полностью совпадать с теми что написаны выше. В противном случае будет ошибка.

**constructor TObject.Create(ObjectType:String;Width,Height:Real);**

**var T,W:real;**

**i:integer;**

**begin**

**T:=Height;**

**W:=Width;**

**if (ObjectType='Cube') then begin**

**Size:=8;** Ко-во вершин.

Координаты расположены изначально начале координат если не понятно то возьмите листок ручку и нарисуйте 2 мерную матрицу.

Ладно написал сам).

Первые четы точки в 0 остальные четыре аналогичны но подняты на высоту Т.

**MatrixObject[1,1]:=0; MatrixObject[2,1]:=Width; MatrixObject[3,1]:=Width; MatrixObject[4,1]:=0;**

**MatrixObject[1,2]:=0; MatrixObject[2,2]:=0; MatrixObject[3,2]:=Width; MatrixObject[4,2]:=Width;**

**MatrixObject[1,3]:=0; MatrixObject[2,3]:=0; MatrixObject[3,3]:=0; MatrixObject[4,3]:=0;**

**MatrixObject[1,4]:=1; MatrixObject[2,4]:=1; MatrixObject[3,4]:=1; MatrixObject[4,4]:=1;**

**MatrixObject[5,1]:=0; MatrixObject[6,1]:=Width; MatrixObject[7,1]:=Width; MatrixObject[8,1]:=0;**

**MatrixObject[5,2]:=0; MatrixObject[6,2]:=0; MatrixObject[7,2]:=Width; MatrixObject[8,2]:=Width;**

**MatrixObject[5,3]:=T; MatrixObject[6,3]:=T; MatrixObject[7,3]:=T; MatrixObject[8,3]:=T;**

**MatrixObject[5,4]:=1; MatrixObject[6,4]:=1; MatrixObject[7,4]:=1; MatrixObject[8,4]:=1;**

**end;**

**if (ObjectType='Ðyramid') then begin**

**Size:=4;**

**MatrixObject[1,1]:=0; MatrixObject[2,1]:=W; MatrixObject[3,1]:=-W; MatrixObject[4,1]:=Round((MatrixObject[1,1]+MatrixObject[2,1]+MatrixObject[3,1])/4);**

**MatrixObject[1,2]:=W; MatrixObject[2,2]:=-W; MatrixObject[3,2]:=-W; MatrixObject[4,2]:=Round((MatrixObject[1,2]+MatrixObject[2,2]+MatrixObject[3,2])/4);**

**MatrixObject[1,3]:=0; MatrixObject[2,3]:=0; MatrixObject[3,3]:=0; MatrixObject[4,3]:=T;**

**MatrixObject[1,4]:=1; MatrixObject[2,4]:=1; MatrixObject[3,4]:=1; MatrixObject[4,4]:=1;**

**end;**

**Robert:= TRobert.Create(Size);**

А тут спрятан конструктор экранирования.

**end;**

**procedure TObject.Center;**

**var i:integer;**

**begin**

**for i:=1 to Size do TcenterX:=TcenterX+MatrixObject[i,1];**

**for i:=1 to Size do TcenterY:=TcenterY+MatrixObject[i,2];**

**for i:=1 to Size do TcenterZ:=TcenterZ+MatrixObject[i,3];**

**TCenterX:=(TcenterX/Size);**

**TCenterY:=(TCenterY/Size);**

**TCenterZ:=(TCenterZ/Size);**

**end;**

Находит центр масс объекта это очевидно.. Тем более вы это рассказывали.

**procedure TObject.Rect(v1,v2,v3,v4:byte);**

**var x1,y1,x2,y2,x3,y3,x4,y4:real;**

**begin**

**if BuferDraw then begin**

**x1:=BMatrixObject[v1,1]; y1:=BMatrixObject[v1,2];**

**x2:=BMatrixObject[v2,1]; y2:=BMatrixObject[v2,2];**

**x3:=BMatrixObject[v3,1]; y3:=BMatrixObject[v3,2];**

**x4:=BMatrixObject[v4,1]; y4:=BMatrixObject[v4,2];**

**end else begin**

**x1:=MatrixObject[v1,1]; y1:=MatrixObject[v1,2];**

**x2:=MatrixObject[v2,1]; y2:=MatrixObject[v2,2];**

**x3:=MatrixObject[v3,1]; y3:=MatrixObject[v3,2];**

**x4:=MatrixObject[v4,1]; y4:=MatrixObject[v4,2];**

**end;**

**Line(Round(x1),Round(y1),Round(x2),Round(y2));**

**Line(Round(x2),Round(y2),Round(x3),Round(y3));**

**Line(Round(x3),Round(y3),Round(x4),Round(y4));**

**Line(Round(x4),Round(y4),Round(x1),Round(y1));**

**end;**

Суть в Чём процедура рисует прямоугольник по вершинам, и смотрит по какой именно из 2-х матриц рисовать.

**procedure TObject.Treg(v1,v2,v3:byte);**

**var x1,y1,x2,y2,x3,y3:real;**

**begin**

**if BuferDraw then begin**

**x1:=BMatrixObject[v1,1]; y1:=BMatrixObject[v1,2];**

**x2:=BMatrixObject[v2,1]; y2:=BMatrixObject[v2,2];**

**x3:=BMatrixObject[v3,1]; y3:=BMatrixObject[v3,2];**

**end else begin**

**x1:=MatrixObject[v1,1]; y1:=MatrixObject[v1,2];**

**x2:=MatrixObject[v2,1]; y2:=MatrixObject[v2,2];**

**x3:=MatrixObject[v3,1]; y3:=MatrixObject[v3,2];**

**end;**

**Line(Round(x1),Round(y1),Round(x2),Round(y2));**

**Line(Round(x2),Round(y2),Round(x3),Round(y3));**

**Line(Round(x3),Round(y3),Round(x1),Round(y1));**

**end;**

Бла бла бла.. аналогично)

Так Вот в Процедуре линии мы округляем потому что в паскале процедура линия она имеет она описана как Line(x1,y1,z2,y2:integer) вводный тип integer а мы пользуемся переменными типа real вот Rount() как раз и округлит тип Real в тип Integer.

**procedure TObject.Draw(Color:integer);**

**begin**

**SetPenColor(Color);**

**if (Size=8) then begin**

**if (not(Robert.GDraw[1])) then Rect(1,2,3,4);**

**if (not(Robert.GDraw[2])) then Rect(2,3,7,6);**

**if (not(Robert.GDraw[3])) then Rect(6,7,8,5);**

**if (not(Robert.GDraw[4])) then Rect(5,8,4,1);**

**if (not(Robert.GDraw[5])) then Rect(1,2,6,5);**

**if (not(Robert.GDraw[6])) then Rect(4,3,7,8);**

**end;**

**if (Size=4) then begin**

**if (not(Robert.GDraw[1])) then Treg(1,2,3);**

**if (not(Robert.GDraw[2])) then Treg(1,3,4);**

**if (not(Robert.GDraw[3])) then Treg(1,2,4);**

**if (not(Robert.GDraw[4])) then Treg(3,2,4);**

**end;**

**end;**

В Зависимости от того сколько вершин у объектов то и надо рисовать. Если 8 следовательно это куб. Если 4 то пирамида вот если значение массива GDraw который описан в классе TRobert истина то рисуем плоскость.

**end.**

Вот на этом Класс TObject Закончен но его использование будет описано ниже. В Последнем нашем классе TScene.

# Модуль TSceneClass.

**unit TSceneClass;**

**interface**

**uses GraphABC,TObjectClass,TCameraClass,TRobertClass;**

**implementation**

**type Matrix = array[1..100,1..100]of Real;**

**const M\_PI=pi;**

**type**

**TScene = class**

**GMove,GRotation,GRersp:Matrix;**

**GMBufer,GMBufer2,GMBufer3:Matrix;**

**Size:integer;** Хранит ко-во вершин объекта с которым мы работаем.

**Screening,perspect:boolean;**

Они включают или выключают из кода перспективу и экранирование в зависимости от их состояния истины они или ложны.

**ObjectColor:integer;**

**SceneColor:integer;**

Хранят цвета объекта и сцены.

**GSizeMatrix:integer;**

Размер матрици той с которой будем работать. На всякий случай и она тоже нужна.

**GPosIndexObject:integer;**

Вот эта переменная поистине важна она хранит в себе ко-во объектов на сцене.

**GCWidth,GCHeight:integer;**

Они хранят размер нашего окна.

**MSObject:array[1..100] of TObject;**

Тут мы используем наш тип данных как TObject и создаём массив этого типа данных где каждый элемент массива будет копией класса TObject и будет содержать в себе все значения переменных и массивов.

**constructor Create(Color,ColorObj,ClientWidth,ClientHeight:integer);**

Конструктор сцены.1 цвет фона, 2 цвет объекта, ширина высота окна. Его мы вызывали в Main.pas. Scene:=TScene(…);

**procedure Create\_GRotation(a:Real;X,Y,Z:boolean);**

**procedure Create\_GMove(dx,dy,dz:real);**

**procedure Create\_GRersp(Zk,Zpr,Z:real);**

Они создают матрицы каждая свою.

**procedure CreateSingleObject(Name:String;Widtch,Heigcht:integer);**

Создаёт один объект в нашей сцене. Он будет первым и единственным в массиве объектов.

**procedure Position\_Scene(Index,CHeigh,CWidth,CenteHight,CenteWidtch:integer);**

А Вот и она. Я уже говорил о ней. Повторяться не будем.

**procedure Flight(Camera:TCamera);**

Перерисовка сцены исходя из переменных которые лежат в Camera послали весь класс камеры в качестве вводного параметра процедуре.

**procedure DrawMassivRebock(Flag:boolean);**

Присвоит массиву который отвечает за рисование плоскостей необходимое нам значение.

**procedure Mult(a,b:Matrix);**

Перемножит a\*b=MBufer.

**end;**

Теперь более подробно…

**constructor TScene.Create(Color,ColorObj,ClientWidth,ClientHeight:integer);**

**begin**

**Clearwindow(Color);**

Очищает экран цветом color.

**SceneColor:=Color;**

**ObjectColor:=ColorObj;**

Присваиваем значение переменным описанным в классе.

**setwindowsize(ClientWidth,ClientHeight);**

Размер нашего окна для рисования.

**GCWidth:= ClientWidth;**

**GCHeight:=ClientHeight;**

Сохраняем размер нашего экрана в переменных класса.

**end;**

**procedure TScene.CreateSingleObject(Name:String;Widtch,Heigcht:integer);**

**var x0,y0:integer;**

**begin**

**GPosIndexObject:=1;**

Так как Кол-во объектов 1 то и переменная принимает значение 1 ну всё же как ни как процедура называется Создание одного объекта.

**x0:=Round(GCWidth/2);**

**y0:=Round(GCHeight/2);**

Центр Экрана. Очевидно. Делим высоту и ширину на 2.

**MSObject[1]:=TObject.Create(Name,Widtch,Heigcht);**

Первому элементу массива и единственному присваиваем значение класса TObject и заодно массиву MatrixObject значение куба или пирамиды стоящей в начале координат. См. Конструктор в классе TObject.

**Size:=MSObject[1].Size;**

Пере сохраняем размер объекта в переменную класса Scene.

**MSObject[1].center;**

Вызываем метод который найдёт центр нашего объекта и запише его в переменные TCenter<X,Y,Z>

**Create\_GMove(x0-MSObject[1].TcenterX,y0-MSObject[1].TcenterY,0);**

Создание матрицы перемещения Ну вы помните.

**Mult(MSObject[1].MatrixObject,GMove);**

Перемножаем нашу матрицу с координатами которые находятся в начале на середину экрана.

**MSObject[1].App(GMBufer);**

И На последок применяем изменения так как результат перемножения двух матриц хранится в MBufer. Теперь он записался процедурой App в MatrixObject

**end;**

**procedure TScene.DrawMassivRebock(Flag:boolean);**

**var i,j:byte;**

**begin**

**for i:=1 to GPosIndexObject do**

**for j:=1 to 10 do**

**MSObject[i].Robert.GDraw[j]:=flag;**

**end;**  проход по всем элементам сцены так как для каждого элемента есть свой массив рисования **MSObject[i].Robert.GDraw[j]:=flag;**  вот так он выглядит.

Эта процедура должна разместить наши объекты по сцене. НУ Она это и сделает рисовать тут не будет просто расположит координаты в нужных точках. Она работает аналогичным образом как и **CreateSingleObject**  за тем исключением что ко-во элементов равно вводному параметру. И их не 1 и первый объект у нас пирамида так что изначальный цикл идёт со 2 а в конце на первое место ставится пирамида Хотя не точно по Центру но это и не важно. Вот И Всё. А и Объекты кубы в Нашем Случае расставляются про кругу.

**Procedure TScene.Position\_Scene(Index,CHeigh,CWidth,CenteHight,CenteWidtch:integer);**

**var i,j,k,x0,y0,x1,y1:integer;**

**Step,Angl,R:real;**

**begin**

**GPosIndexObject:=Index;**

**x0:=Round(GCWidth/2);**

**y0:=Round(GCHeight/2);**

**if Index<>1 then begin**

**R:=GCHeight/2-100;**

Радиус я Взял такой Радиус круга имеется в виду.

**Step:=360/(Index-1);**

Шаг 360 делить на ко-во объектов минус 1 так как пирамида в центре экрана.

360 так как круг.

**for i:=2 to Index do begin**

**MSObject[i]:=TObject.Create('Cube',CHeigh,CWidth);**

**Size:=MSObject[i].Size;**

**x1:=Round(R\*cos(angl\*M\_PI/180))+x0;**

**y1:=Round(R\*sin(angl\*M\_PI/180))+y0;**

R-Радиус xo,yo-центр экрана. X1,y1 –полученные координаты

**angl:=angl+step;**

Накапливаем шаг.

**Create\_GMove(x1,y1,0);**

Создание матрицы перемещения

**Mult(MSObject[i].MatrixObject,GMove);**

**MSObject[i].App(GMBufer);**

Бля бла.. теперь она на месте идём к следующему..

**end; end;**

**//**

**MSObject[1]:=TObject.Create('Ðyramid',CenteHight,CenteWidtch);**

**Size:=MSObject[1].Size;**

**Create\_GMove(x0+MSObject[1].MatrixObject[4,1],y0+MSObject[1].MatrixObject[4,2],0);**

**Mult(MSObject[1].MatrixObject,GMove);**

**MSObject[1].App(GMBufer);**

**end;** Конец.

**//===============================**

Последнее что я вам расскажу так как всё остальное Очевидно.

Итак процедура полёт она выполняем перерисовку сцены вы её уже неоднократно видели в Main.pas входной параметр переменная Camera типа TCamera который описан в модуле TCameraClass значит как только мы послали в неё Camera мы передали все значения переменных класса TCamera вот от Туда она их и берут. Только не забывайте мы работаем копиями классов сам класс нужен только для вызова конструктора после чего все копируется в переменную с таким же типом. Вот Как Же Происходит и с массивом Объектов так в каждой ячейке массива будет копия класса в каждом своя. Так на всякий.

**procedure TScene.Flight(Camera:TCamera);**

**var i,k:integer;**

**begin**

**Create\_GRotation(Camera.Step,Camera.XRot,Camera.YRot,Camera.ZRot);**

Создание матрицы поворота.

**for i:=1 to GPosIndexObject do begin**

цикл с 1 по ко-во Объектов в Сцене.

**MSObject[i].Draw(SceneColor);**

Рисуем итый объект.

**Size:=MSObject[i].Size;**

Сохраняем размер так как Mult работает только с Size см Процедуру Mult.

**Create\_GMove(-Camera.x,-Camera.y,-Camera.z);**

Создаёт матрицу перемещения.

**Mult(MSObject[i].MatrixObject,GMove);**

Перемножает наша матрицу тем самым буферная матрица оказывается в начале координат.

**MSObject[i].Add(GMBufer);**

Сохранили.

**Mult(MSObject[i].BMatrixObject,GRotation);**

**GMBufer3:=GMBufer;**

Повернули и сохранили. Сохранили положение матрицы которая повёрнута и в начале координат то что нам нужно))

**Create\_GMove(Camera.x,Camera.y,Camera.z);**

**Mult(GMBufer,GMove);**

**MSObject[i].App(GMBufer);**

Тут Мы Возвращаем матрицу в начало и записываем её как основную.

Суть выше сделанного в том чтобы сохранить в Буфер 3 значение необходимой матрице так как перспектива делается в начале координат, а если просто облёт то получится что просто буфер 3 мы не будем трогать будем просто в него сохранять.

**if Screening then MSObject[i].Robert.RVisibleGran(GMBufer2,0,0,-10,0);**

Тут мы смотрим включено ли экранирование это опистим.. Не Важно

**if perspect then begin** Если перспектива вкл.

**for k:=1 to Size do begin**

**Create\_GRersp(Camera.Zk,Camera.Zpr, GMBufer3[k,3]);**

**Mult(GMBufer3,GRersp);**

**GMBufer2[k,1]:=GMBufer[k,1];**

**GMBufer2[k,2]:=GMBufer[k,2];**

**GMBufer2[k,3]:=GMBufer[k,3];**

**GMBufer2[k,4]:=GMBufer[k,4];**

**end;**

Эта штука.. Делает перспективную картинку а точнее берёт каждую высоту объекта сжимает его на получившийся коэффициент и сохраняет необходимый столбец в свободный буфер2.

**Mult(GMBufer2,GMove);**

Буфер 2 перемещаем в место от куда мы изначально взяли объект

**MSObject[i].Add(GMBufer);**

Добавляем значение в Класс в переменную BMatrixObject в буферную переменную

**MSObject[i].BuferDraw:=true;**

И Говорим что рисовать мы будем из Буфера

В итоге у нас в переменной **MSObject[i]** Класса TObject MatrixObject хранится объект который не изменён перспективой но повёрнут а в переменной BMatrixObject находится та перспективная проекция которая и будет рисоваться. Благодаря переменной BuferDraw.

**end else begin**

если перспектива выкл. То Рисуем по основной матрице.

**MSObject[i].BuferDraw:=false;**

**end;**

**MSObject[i].Draw(ObjectColor);**

Рисуем наш объект. Внимание!!! Вот Только тут мы рисуем итый объект! И так будет с каждый объектом в сцене!

**end;**

**end;**

**procedure TScene.Mult(a,b:Matrix);** a\*b=GMBufer

**var bv:real;**

**i,j,l:integer;**

**begin**

**for i:=1 to Size do begin**

**for j:=1 to 4 do begin**

**bv:=0;**

**for l:=1 to 4 do bv:=bv+a[i,l]\*b[l,j];**

**GMBufer[i,j]:=bv;**

**end;end;**

**end;**

Ниже описаны создание матриц! Как Выводятся матрицы смотрите лекции только перспективы может не быть.. Ну Там Всё простою.

**procedure TScene.Create\_GRersp(Zk,Zpr,Z:real);**

**var P:real;**

**begin**

**if (Zk-Z<>0) then begin**

**P:=(Zk-Zpr)/(Zk-Z);**

**GSizeMatrix:=4;**

**GRersp[1,1]:=P; GRersp[2,1]:=0; GRersp[3,1]:=0; GRersp[4,1]:=0;**

**GRersp[1,2]:=0; GRersp[2,2]:=P; GRersp[3,2]:=0; GRersp[4,2]:=0;**

**GRersp[1,3]:=0; GRersp[2,3]:=0; GRersp[3,3]:=1; GRersp[4,3]:=-Zpr;**

**GRersp[1,4]:=0; GRersp[2,4]:=0; GRersp[3,4]:=0; GRersp[4,4]:=1; end;**

**end;**

**procedure TScene.Create\_GMove(dx,dy,dz:real);**

**begin**

**GSizeMatrix:=4;**

**GMove[1,1]:=1; GMove[2,1]:=0; GMove[3,1]:=0; GMove[4,1]:=dx;**

**GMove[1,2]:=0; GMove[2,2]:=1; GMove[3,2]:=0; GMove[4,2]:=dy;**

**GMove[1,3]:=0; GMove[2,3]:=0; GMove[3,3]:=1; GMove[4,3]:=dz;**

**GMove[1,4]:=0; GMove[2,4]:=0; GMove[3,4]:=0; GMove[4,4]:=1;**

**end;**

**procedure TScene.Create\_GRotation(a:Real;X,Y,Z:boolean);**

**begin**

**GSizeMatrix:=4;**

**if (Y) then begin**

**GRotation[1,1]:=1; GRotation[2,1]:=0; GRotation[3,1]:=0; GRotation[4,1]:=0;**

**GRotation[1,2]:=0; GRotation[2,2]:=cos(a\*M\_PI/180); GRotation[3,2]:=sin(a\*M\_PI/180); GRotation[4,2]:=0;**

**GRotation[1,3]:=0; GRotation[2,3]:=-sin(a\*M\_PI/180);GRotation[3,3]:=cos(a\*M\_PI/180); GRotation[4,3]:=0;**

**GRotation[1,4]:=0; GRotation[2,4]:=0; GRotation[3,4]:=0; GRotation[4,4]:=1;**

**end;**

**if (X) then begin**

**GRotation[1,1]:=cos(a\*M\_PI/180); GRotation[2,1]:=0; GRotation[3,1]:=sin(a\*M\_PI/180); GRotation[4,1]:=0;**

**GRotation[1,2]:=0; GRotation[2,2]:=1; GRotation[3,2]:=0; GRotation[4,2]:=0;**

**GRotation[1,3]:=-sin(a\*M\_PI/180); GRotation[2,3]:=0; GRotation[3,3]:=cos(a\*M\_PI/180); GRotation[4,3]:=0;**

**GRotation[1,4]:=0; GRotation[2,4]:=0; GRotation[3,4]:=0; GRotation[4,4]:=1;**

**end;**

**if (Z) then begin**

**GRotation[1,1]:=cos(a\*M\_PI/180); GRotation[2,1]:=-sin(a\*M\_PI/180); GRotation[3,1]:=0; GRotation[4,1]:=0;**

**GRotation[1,2]:=sin(a\*M\_PI/180); GRotation[2,2]:=cos(a\*M\_PI/180); GRotation[3,2]:=0; GRotation[4,2]:=0;**

**GRotation[1,3]:=0; GRotation[2,3]:=0; GRotation[3,3]:=1; GRotation[4,3]:=0;**

**GRotation[1,4]:=0; GRotation[2,4]:=0; GRotation[3,4]:=0; GRotation[4,4]:=1;**

**end;**

**if (not(Z) and not(Y) and not(X)) then begin**

**GRotation[1,1]:=1; GRotation[2,1]:=0; GRotation[3,1]:=0; GRotation[4,1]:=0;**

**GRotation[1,2]:=0; GRotation[2,2]:=1; GRotation[3,2]:=0; GRotation[4,2]:=0;**

**GRotation[1,3]:=0; GRotation[2,3]:=0; GRotation[3,3]:=1; GRotation[4,3]:=0;**

**GRotation[1,4]:=0; GRotation[2,4]:=0; GRotation[3,4]:=0; GRotation[4,4]:=1;**

**end;**

**end;**

**end.**

# Модуль TRobertClass

Данный необходим для экранирования объектов и содержит массив флагов по которым смотрится видимость плоскостей для последних двух задач это не важно так что описание этого класса мы опустим,

**unit TRobertClass;**

**interface**

**uses GraphABC;**

**type Matrix = array[1..100,1..100]of Real;**

**implementation**

**type**

**TRobert = class**

**Size:integer;**

**GDraw:array[1..100] of boolean;**

Вот этот массив который у нас контролирует рисование плоскостей.

**Plos:array[1..10,1..3] of integer;**

**V:array[1..100,1..100] of integer;**

**Result:array[1..10] of real;**

**SizePlos:integer;**

Сохраняет кол-во плоскостей.

**constructor Create(TSize:integer);**

**procedure Add\_to\_Matrix(a:matrix;v1,v2,v3,NumSt:integer);**

**procedure RVisibleGran(outMAtrix:MAtrix;X,Y,Z,P:integer);**

**end;**

**constructor TRobert.Create(TSize:integer);**

**begin**

**Size:=TSize;**

**if (Size=8) then**

**begin SizePlos:=6;**

**Plos[1,1]:=1; Plos[1,2]:=2; Plos[1,3]:=3;**

**Plos[2,1]:=2; Plos[2,2]:=3; Plos[2,3]:=7;**

**Plos[3,1]:=6; Plos[3,2]:=7; Plos[3,3]:=8;**

**Plos[4,1]:=5; Plos[4,2]:=8; Plos[4,3]:=4;**

**Plos[5,1]:=1; Plos[5,2]:=2; Plos[5,3]:=6;**

**Plos[6,1]:=4; Plos[6,2]:=3; Plos[6,3]:=7;**

**end;**

**if Size=4 then begin SizePlos:=4;**

**Plos[1,1]:=1; Plos[1,2]:=2; Plos[1,3]:=3;**

**Plos[2,1]:=1; Plos[2,2]:=3; Plos[2,3]:=4;**

**Plos[3,1]:=1; Plos[3,2]:=2; Plos[3,3]:=4;**

**Plos[4,1]:=3; Plos[4,2]:=2; Plos[4,3]:=4;**

**end;**

**end;**

устанавливает кол-во плоскостей и номера вершин.

**procedure TRobert.RVisibleGran(outMAtrix:MAtrix;X,Y,Z,P:integer);**

**var i,j,k,xc,yc,zc:integer;**

**bv:real;**

**Look,center:array[1..4] of integer;**

**begin**

**for i:=1 to SizePlos do Add\_to\_Matrix(outMAtrix,Plos[i,1],Plos[i,2],Plos[i,3],i);**

**Look[1]:=X; Look[2]:=Y; Look[3]:=Z; Look[4]:=P;**

**for i:=1 to Size do xc:=round(xc+outMAtrix[i,1]);**

**for i:=1 to Size do yc:=round(yc+outMAtrix[i,2]);**

**for i:=1 to Size do zc:=round(zc+outMAtrix[i,3]);**

**xc:=Round(xc/Size);**

**yc:=Round(yc/Size);**

**zc:=Round(zc/Size);**

**center[1]:=xc; center[2]:=yc; center[3]:=zc; center[4]:=1;**

**for i:=1 to SizePlos do begin**

**bv:=0;**

**for j:=1 to 4 do bv:=bv+V[i,j]\*center[j];**

**if (bv<0) then for k:=1 to 4 do V[i,k]:=V[i,k]\*(-1);**

**end;**

**for i:=1 to SizePlos do begin**

**bv:=0;**

**for j:=1 to 4 do bv:=bv+V[i,j]\*Look[j];**

**if(bv>0) then begin GDraw[i]:=true; end else GDraw[i]:=false;**

**end;**

**end;**

Реализация алгоритма Робетртсона(см.Инет)

**procedure TRobert.Add\_to\_Matrix(a:Matrix;v1,v2,v3,NumSt:integer);**

**var x1,x2,x3,y1,y2,y3,z1,z2,z3:real;**

**begin**

**x1:=a[v1,1]; y1:=a[v1,2]; z1:=a[v1,3];**

**x2:=a[v2,1]; y2:=a[v2,2]; z2:=a[v2,3];**

**x3:=a[v3,1]; y3:=a[v3,2]; z3:=a[v3,3];**

**V[NumSt,1] :=Round((y2-y1)\*(z3-z1)-(z2-z1)\*(y3-y1)); // ' A**

**V[NumSt,2] :=Round((z2-z1)\*(x3-x1)-(x2-x1)\*(z3-z1)); // ' B**

**V[NumSt,3] :=Round((x2-x1)\*(y3-y1)-(y2-y1)\*(x3-x1)); // ' C**

**V[NumSt,4] :=-Round((V[NumSt,1]\*x1+V[NumSt,2]\*y1+V[NumSt,3]\*z1));**

**end;**

Помогает создать матрицу объекта.

**end.**