

Пензенский государственный университет
Политехнический институт
Кафедра «Технологии машиностроения»

Курсовой проект по курсу
программированию задач трёхмерной графики

Выполнил: студент группы 16ММм1
Сухов В.Н.

Проверил: к.т.н. доцент
отлично  Нестеров С.А.

Пенза 2017

Содержание

Введение

1. Основной текст Lazarus.

2. Файл описания

2.1.Задание

2.2.Общий алгоритм работы

2.3.Основные функции процедуры

2.4.Скриншоты

Литература

Введение

OpenGL является одним из самых популярных прикладных программных интерфейсов (API – Application Programming Interface) для разработки приложений в области двумерной и трехмерной графики.

Стандарт OpenGL (Open Graphics Library – открытая графическая библиотека) был разработан и утвержден в 1992 году ведущими фирмами в области разработки программного обеспечения как эффективный аппаратно-независимый интерфейс, пригодный для реализации на различных платформах. Основой стандарта стала библиотека IRIS GL, разработанная фирмой Silicon Graphics Inc.

На сегодняшний день графическая система OpenGL поддерживается большинством производителей аппаратных и программных платформ. Эта система доступна тем, кто работает в среде Windows, пользователям компьютеров Apple. Свободно распространяемые коды системы Mesa (пакет API на базе OpenGL) можно компилировать в большинстве операционных систем, в том числе в Linux.

Характерными особенностями OpenGL, которые обеспечили распространение и развитие этого графического стандарта, являются:

Стабильность. Дополнения и изменения в стандарте реализуются таким образом, чтобы сохранить совместимость с разработанным ранее программным обеспечением.

Надежность и переносимость. Приложения, использующие OpenGL, гарантируют одинаковый визуальный результат вне зависимости от типа используемой операционной системы и организации отображения информации. Кроме того, эти приложения могут выполняться как на персональных компьютерах, так и на рабочих станциях и суперкомпьютерах.

Легкость применения. Стандарт OpenGL имеет продуманную структуру и интуитивно понятный интерфейс, что позволяет с меньшими затратами создавать эффективные приложения, содержащие меньше строк кода, чем с использованием других графических библиотек. Необходимые функции для обеспечения совместимости с различным оборудованием реализованы на уровне библиотеки и значительно упрощают разработку приложений.

Наличие хорошего базового пакета для работы с трехмерными приложениями упрощает понимание студентами ключевых тем курса компьютерной графики – моделирование трехмерных объектов, закрашивание, текстурирование, анимацию и т.д. Широкие функциональные возможности OpenGL служат хорошим фундаментом для изложения теоретических и практических аспектов предмета.

1.ОСНОВНОЙ ТЕКСТ Lazarus.

```
{{
*****
*****
***
*
*
* This source is free software; you can
redistribute it and/or modify *
* it under the terms of the GNU General
Public License as published by *
* the Free Software Foundation; either
version 2 of the License, or *
* (at your option) any later version.
*
*
* This code is distributed in the hope that
it will be useful, but *
* WITHOUT ANY WARRANTY; without
even the implied warranty of *
* MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the
GNU *
* General Public License for more details.
*
*
* A copy of the GNU General Public
License is available on the World *
* Wide Web at
<http://www.gnu.org/copyleft/gpl.html>.
You can also *
* obtain it by writing to the Free Software
Foundation, *
* Inc., 59 Temple Place - Suite 330,
Boston, MA 02111-1307, USA. *
*
*****
*****
***

written 2001 by Satan

}
unit ExampleForm;

{$mode objfpc} {$H+}

interface

uses
  Classes, SysUtils, FileUtil, LCLProc,
  Forms, LResources, Buttons,
  StdCtrls, Dialogs, Graphics, IntfGraphics,
  GL, FPImage, OpenGLContext;

const
  GL_CLAMP_TO_EDGE = $812F;

type
  TglTexture = class
  public
    Width,Height: longint;
    Data : pointer;
    destructor Destroy; override;
  end;

type
  { TExampleForm }

  TExampleForm = class(TForm)
    OpenGLControl1: TOpenGLControl;
    ExitButton1: TButton;
    LightingButton1: TButton;
    BlendButton1: TButton;
    MoveCubeButton1: TButton;
    MoveBackgroundButton1: TButton;
    RotateZButton1: TButton;
    RotateZButton2: TButton;
    HintLabel1: TLabel;
    procedure IdleFunc(Sender: TObject; var
Done: Boolean);
    procedure FormResize(Sender: TObject);
    procedure ExitButton1Click(Sender:
TObject);
    procedure LightingButton1Click(Sender:
TObject);
    procedure BlendButton1Click(Sender:
TObject);
    procedure
MoveCubeButton1Click(Sender: TObject);
    procedure
MoveBackgroundButton1Click(Sender:
TObject);
    procedure RotateZButton1Click(Sender:
TObject);
    procedure RotateZButton2Click(Sender:
TObject);
```

```

    procedure OpenGLControl1Paint(Sender:
TObject);
    procedure
OpenGLControl1Resize(Sender: TObject);
    public
        constructor Create(TheOwner:
TComponent); override;
        destructor Destroy; override;
        procedure LoadTextures;
    private
        AreaInitialized: boolean;
        FrameCount: integer;
        LastFrameTicks: integer;
    end;

TParticle = class
    x, y, z: GLfloat;
    vx, vy, vz: GLfloat;
    life: single;
end;

TParticleEngine = class
    xspawn: GLfloat;
    Particle: array [1..2001] of TParticle;
    procedure MoveParticles;
    procedure DrawParticles;
    procedure Start;
    public
        constructor Create;
        destructor Destroy; override;
    private
        procedure RespawnParticle(i: integer);
    end;

var AnExampleForm: TExampleForm;
    front, left1: GLuint;
    rx, ry, rz, rrx, rry, da, rrz: single;
    LightAmbient : array [0..3] of GLfloat;
    checked, blended, lighted,
ParticleBlended, MoveCube,
MoveBackground: boolean;
    textures : array [0..2] of GLuint; //
Storage For 3 Textures
    MyglTextures : array [0..2] of
TglTexture;
    lightamb, lightdif, lightpos, light2pos,
light2dif,
    light3pos, light3dif, light4pos, light4dif,
fogcolor: array [0..3] of GLfloat;
    ParticleEngine: TParticleEngine;
    ParticleList, CubeList, BackList: GLuint;

```

```

var direction: boolean;
    timer: single;
    LastMsecs: integer;

function LoadFileToMemStream(const
Filename: string): TMemoryStream;
function
LoadglTexImage2DFromPNG(PNGFilename: string;
    Image: TglTexture): boolean;

implementation

function LoadFileToMemStream(const
Filename: string): TMemoryStream;
var FileStream: TFileStream;
begin
    Result:=TMemoryStream.Create;
    try

        FileStream:=TFileStream.Create(UTF8ToSy
s(Filename), fmOpenRead);
        try

            Result.CopyFrom(FileStream,FileStream.Si
ze);
            Result.Position:=0;
        finally
            FileStream.Free;
        end;
    except
        Result.Free;
        Result:=nil;
    end;
end;

function
LoadglTexImage2DFromPNG(PNGFilename: string; Image: TglTexture
): boolean;
var
    png: TPortableNetworkGraphic;
    IntfImg: TLazIntfImage;
    y: Integer;
    x: Integer;
    c: TFPColor;
    p: PByte;
begin
    Result:=false;
    png:=TPortableNetworkGraphic.Create;
    IntfImg:=nil;
    try

```

```

png.LoadFromFile(PNGFilename);
IntfImg:=png.CreateIntfImage;
Image.Width:=IntfImg.Width;
Image.Height:=IntfImg.Height;

GetMem(Image.Data,Image.Width*Image.Height * 3);
p:=PByte(Image.Data);
for y:=0 to IntfImg.Height-1 do begin
  for x:=0 to IntfImg.Width-1 do begin
    c:=IntfImg.Colors[x,y];
    p^:=c.red shr 8;
    inc(p);
    p^:=c.green shr 8;
    inc(p);
    p^:=c.blue shr 8;
    inc(p);
  end;
end;
finally
  png.Free;
  IntfImg.Free;
end;
Result:=true;
end;

{ TExampleForm }

constructor
TExampleForm.Create(TheOwner:
TComponent);
begin
  inherited CreateNew(TheOwner);
  da:=0.1;
  if LazarusResources.Find(ClassName)=nil
  then begin
    SetBounds((Screen.Width-800) div 2,
(Screen.Height-600) div 2,800,600);
    Caption:='LCL example for the
TOpenGLControl';

    Application.OnIdle:=@IdleFunc;
    OnResize:=@FormResize;
    blended:=false;
    lighted:=false;
    ParticleEngine:=TParticleEngine.Create;

    ExitButton1:=TButton.Create(Self);
    with ExitButton1 do begin
      Name:='ExitButton1';
      Parent:=Self;
      SetBounds(320,10,80,25);

```

```

Caption:='Exit';
OnClick:=@ExitButton1Click;
end;

```

```

LightingButton1:=TButton.Create(Self);
with LightingButton1 do begin
  Name:='LightingButton1';
  Parent:=Self;
  SetBounds(220,0,80,25);
  Caption:='Lighting';
  OnClick:=@LightingButton1Click;
end;

```

```

BlendButton1:=TButton.Create(Self);
with BlendButton1 do begin
  Name:='BlendButton1';
  Parent:=Self;
  SetBounds(220,0,80,25);
  Caption:='Blending';
  OnClick:=@BlendButton1Click;
end;

```

```

MoveCubeButton1:=TButton.Create(Self);
with MoveCubeButton1 do begin
  Name:='MoveCubeButton1';
  Parent:=Self;
  SetBounds(320,10,80,25);
  Caption:='Переместить Куб';
  Checked:=false;
  OnClick:=@MoveCubeButton1Click;
end;

```

```

MoveBackgroundButton1:=TButton.Create(
Self);
with MoveBackgroundButton1 do begin
  Name:='MoveBackgroundButton1';
  Parent:=Self;
  SetBounds(320,10,80,25);
  Caption:='Move Back';
  Checked:=false;

```

```

OnClick:=@MoveBackgroundButton1Click
;
end;

```

```

RotateZButton1:=TButton.Create(Self);
with RotateZButton1 do begin
  Name:='RotateZButton1';
  Parent:=Self;
  SetBounds(320,10,80,25);

```

```

Caption:='P. Respawn';
Checked:=false;
OnClick:=@RotateZButton1Click;
end;

RotateZButton2:=TButton.Create(Self);
with RotateZButton2 do begin
  Name:='RotateZButton2';
  Parent:=Self;
  SetBounds(320,10,80,25);
  Caption:='P. Blending';
  Checked:=false;
  OnClick:=@RotateZButton2Click;
end;

HintLabel1:=TLabel.Create(Self);
with HintLabel1 do begin
  Name:='HintLabel1';
  Parent:=Self;
  SetBounds(0,0,280,50);
  Caption:='Demo';
end;

// resize the components first, because the
opengl context needs some time to setup
FormResize(Self);

OpenGLControl1:=TOpenGLControl.Create
(Self);
with OpenGLControl1 do begin
  Name:='OpenGLControl1';
  Parent:=Self;
  SetBounds(10,90,380,200);
  OnPaint:=@OpenGLControl1Paint;
  OnResize:=@OpenGLControl1Resize;
end;

end;

LoadTextures;

// now resize
FormResize(Self);
end;

destructor TExampleForm.Destroy;
var i: integer;
begin
  for i:=0 to 2 do begin
    Textures[i]:=0;
    FreeAndNil(MyglTextures[i]);

```

```

end;
FreeAndNil(ParticleEngine);

inherited Destroy;
end;

procedure TExampleForm.LoadTextures;

  procedure LoadglTexture(Filename:string;
Image:TglTexture);
  begin
    Filename:=ExpandFileNameUTF8(Filename
);
    if not
LoadglTexImage2DFromPNG(Filename,Im
age) then begin
      MessageDlg('File not found',
        'Image file not found: '+Filename,
        mtError,[mbOk],0);
      raise Exception.Create('Image file not
found: '+Filename);
    end;
  end;

var
  i: Integer;
begin
  for i:=0 to 2 do begin
    Textures[i]:=0;
    MyglTextures[i]:=TglTexture.Create;
  end;
  {loading the texture and setting its
parameters}

  LoadglTexture('data/particle.png',MyglTextu
res[0]);

  LoadglTexture('data/texture2.png',MyglText
ures[1]);

  LoadglTexture('data/texture3.png',MyglText
ures[2]);
end;

//
-----
-----
//          Particle Engine
//
-----

```

```

-----

constructor TParticleEngine.Create;
var i: integer;
begin
    for i:=1 to 2001 do
        Particle[i]:=TParticle.Create;
        xspawn:=0;
    end;

    destructor TParticleEngine.Destroy;
    var i: integer;
    begin
        for i:=1 to 2001 do
            FreeAndNil(Particle[i]);
            inherited Destroy;
        end;

    procedure TParticleEngine.DrawParticles;
    var i: integer;
    begin
        for i:=1 to 2001 do begin
            glPushMatrix;
            glTranslatef(Particle[i].x, Particle[i].y,
                Particle[i].z);
            glCallList(ParticleList);
            glPopMatrix;
        end;
    end;

    procedure
        TParticleEngine.RespawnParticle(i: integer);
    begin
        if (xspawn>2) and (direction=true) then
            direction:=false;
        if (xspawn<-2) and (direction=false) then
            direction:=true;
        if direction then
            xspawn:=xspawn+0.0002*(timer/10)
        else
            xspawn:=xspawn-0.0002*(timer/10);
        Particle[i].x:=xspawn;
        Particle[i].y:=-0.5;
        Particle[i].z:=0;
        Particle[i].vx:=-
            0.005+GLfloat(random(2000))/200000;

        Particle[i].vy:=0.035+GLfloat(random(750)
            )/100000;
        Particle[i].vz:=-
            0.005+GLfloat(random(2000))/200000;

```

```

Particle[i].life:=GLfloat(random(1250))/100
0+1;
end;

procedure TParticleEngine.MoveParticles;
var i: integer;
begin
    for i:=1 to 2001 do begin
        if Particle[i].life>0 then begin
            Particle[i].life:=Particle[i].life-
                0.01*(timer/10);

            Particle[i].x:=Particle[i].x+Particle[i].vx*(ti
                mer/10);

            Particle[i].vy:=Particle[i].vy-
                0.00035*(timer/10); // gravity

            Particle[i].y:=Particle[i].y+Particle[i].vy*(ti
                mer/10);

            Particle[i].z:=Particle[i].z+Particle[i].vz*(ti
                mer/10);
            end else begin
                RespawnParticle(i);
            end;
        end;
    end;

    procedure TParticleEngine.Start;
    var i: integer;
    begin
        for i:=1 to 2001 do begin
            RespawnParticle(i);
        end;
    end;

    //
    -----
    -----
    // ++++++
    ++++++
    ++++++
    //
    -----
    -----

    procedure TExampleForm.IdleFunc(Sender:
        TObject; var Done: Boolean);
    begin
        OpenGLControl1.Invalidate;

```

```

//OpenGLControl1Paint(Self);
Done:=false; // tell lcl to handle messages
and return immediatly
end;

//
-----
-----
//          Buttons
//
-----
-----

procedure
TExampleForm.LightingButton1Click(Sender: TObject);
begin
  if lighted then glDisable(GL_LIGHTING)
else glEnable(GL_LIGHTING);
  lighted:=not lighted;
  OpenGLControl1.Invalidate;// not need
end;

procedure
TExampleForm.BlendButton1Click(Sender: TObject);
begin
  blended:=not blended;
  OpenGLControl1.Invalidate;
end;

procedure
TExampleForm.MoveCubeButton1Click(Sender: TObject);
begin
  MoveCube:=not MoveCube;
  OpenGLControl1.Invalidate;
end;

procedure
TExampleForm.MoveBackgroundButton1Click(Sender: TObject);
begin
  MoveBackground:=not MoveBackground;
  OpenGLControl1.Invalidate;
end;

procedure
TExampleForm.RotateZButton1Click(Sender: TObject);
begin
  ParticleEngine.Start;

```

```

OpenGLControl1.Invalidate;
end;

procedure
TExampleForm.RotateZButton2Click(Sender: TObject);
begin
  ParticleBlended:=not ParticleBlended;
  OpenGLControl1.Invalidate;
end;

//
-----
-----
// ++++++
+++++
+++++
//
-----
-----

procedure
TExampleForm.FormResize(Sender: TObject);
begin
  if OpenGLControl1<>nil then
    OpenGLControl1.SetBounds(10, 30,
Width-120, Height-40);
    ExitButton1.SetBounds(Width-90, 5, 80,
25);
    LightingButton1.SetBounds(Width-90,
180, 80, 25);
    BlendButton1.SetBounds(Width-90, 210,
80, 25);
    MoveCubeButton1.SetBounds(Width-90,
50, 80, 25);

MoveBackgroundButton1.SetBounds(Width
-90, 80, 80, 25);
    RotateZButton1.SetBounds(Width-90, 115,
80, 25);
    RotateZButton2.SetBounds(Width-90, 145,
80, 25);
    HintLabel1.SetBounds(10, 0, 80, 25);
end;

procedure
TExampleForm.ExitButton1Click(Sender: TObject);
begin
  Close;
end;

```

```

procedure
TExampleForm.OpenGLControl1Paint(Sender: TObject);

```

```

    procedure myInit;
    begin
        {init lighting variables}
        {ambient color}
        lightamb[0]:=0.5;
        lightamb[1]:=0.5;
        lightamb[2]:=0.5;
        lightamb[3]:=1.0;
        {diffuse color}
        lightdif[0]:=0.8;
        lightdif[1]:=0.0;
        lightdif[2]:=0.0;
        lightdif[3]:=1.0;
        {diffuse position}
        lightpos[0]:=0.0;
        lightpos[1]:=0.0;
        lightpos[2]:=3.0;
        lightpos[3]:=1.0;
        {diffuse 2 color}
        light2dif[0]:=0.0;
        light2dif[1]:=0.8;
        light2dif[2]:=0.0;
        light2dif[3]:=1.0;
        {diffuse 2 position}
        light2pos[0]:=3.0;
        light2pos[1]:=0.0;
        light2pos[2]:=3.0;
        light2pos[3]:=1.0;
        {diffuse 3 color}
        light3dif[0]:=0.0;
        light3dif[1]:=0.0;
        light3dif[2]:=0.8;
        light3dif[3]:=1.0;
        {diffuse 3 position}
        light3pos[0]:=-3.0;
        light3pos[1]:=0.0;
        light3pos[2]:=0.0;
        light3pos[3]:=1.0;
        {fog color}

```

```

        fogcolor[0]:=0.5;
        fogcolor[1]:=0.5;
        fogcolor[2]:=0.5;
        fogcolor[3]:=1.0;

```

```

    end;

```

```

const GLInitialized: boolean = false;

```

```

procedure InitGL;

```

```

var
    i: Integer;
begin
    if GLInitialized then exit;
    GLInitialized:=true;
    {setting lighting conditions}

```

```

    glLightfv(GL_LIGHT0, GL_AMBIENT, lightamb);

```

```

    glLightfv(GL_LIGHT1, GL_AMBIENT, lightamb);

```

```

    glLightfv(GL_LIGHT2, GL_DIFFUSE, lightdif);

```

```

    glLightfv(GL_LIGHT2, GL_POSITION, lightpos);

```

```

    glLightfv(GL_LIGHT3, GL_DIFFUSE, light2dif);

```

```

    glLightfv(GL_LIGHT3, GL_POSITION, light2pos);

```

```

    glLightfv(GL_LIGHT4, GL_POSITION, light3pos);

```

```

    glLightfv(GL_LIGHT4, GL_DIFFUSE, light3dif);

```

```

    glEnable(GL_LIGHT0);

```

```

    glEnable(GL_LIGHT1);

```

```

    glEnable(GL_LIGHT2);

```

```

    glEnable(GL_LIGHT3);

```

```

    glEnable(GL_LIGHT4);

```

```

    glGenTextures(3, @textures[0]);

```

```

    for i:=0 to 2 do begin

```

```

        glBindTexture(GL_TEXTURE_2D, textures[i]);

```

```

    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);

```

```

    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

```

```

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

glTexImage2D(GL_TEXTURE_2D, 0, 3, MyglTextures[i].Width, MyglTextures[i].Height, 0, GL_RGB, GL_UNSIGNED_BYTE, MyglTextures[i].Data);
end;

glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
{instead of GL_MODULATE you can try GL_DECAL or GL_BLEND}
glEnable(GL_TEXTURE_2D); // enables 2d textures
glClearColor(0.0, 0.0, 0.0, 1.0); // sets background color
glClearDepth(1.0);
glDepthFunc(GL_LEQUAL); // the type of depth test to do
glEnable(GL_DEPTH_TEST); // enables depth testing
glShadeModel(GL_SMOOTH); // enables smooth color shading
{blending}
glColor4f(1.0, 1.0, 1.0, 0.5); // Full Brightness, 50% Alpha ( NEW )
glBlendFunc(GL_SRC_ALPHA, GL_ONE);

glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
{}

glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);

glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST);

glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

// creating display lists

//ParticleList:=glGenLists(1);
glNewList(ParticleList, GL_COMPILE);

```

```

glBindTexture(GL_TEXTURE_2D, textures[0]);
glBegin(GL_TRIANGLE_STRIP);
glNormal3f( 0.0, 0.0, 1.0);
glTexCoord2f( 1.0, 1.0);
glVertex3f(+0.025, +0.025, 0);
glTexCoord2f( 0.0, 1.0); glVertex3f(-0.025, +0.025, 0);
glTexCoord2f( 1.0, 0.0);
glVertex3f(+0.025, -0.025, 0);
glTexCoord2f( 0.0, 0.0); glVertex3f(-0.025, -0.025, 0);
glEnd;
glEndList;

//BackList:=ParticleList+1; //Верхний
кyб
glNewList(BackList, GL_COMPILE);
glBindTexture(GL_TEXTURE_2D, textures[2]);
glBegin(GL_QUADS);
{Front Face}
glNormal3f( 0.0, 0.0, 1.0);
glTexCoord2f( 1.0, 1.0);
glVertex3f( 2.5, 2.5, 2.5);
glTexCoord2f( 0.0, 1.0); glVertex3f(-2.5, 2.5, 2.5);
glTexCoord2f( 0.0, 0.0); glVertex3f(-2.5, -2.5, 2.5);
glTexCoord2f( 1.0, 0.0);
glVertex3f( 2.5, -2.5, 2.5);
{Back Face}
glNormal3f( 0.0, 0.0, -1.0);
glTexCoord2f( 0.0, 1.0);
glVertex3f( 2.5, 2.5, -2.5);
glTexCoord2f( 0.0, 0.0);
glVertex3f( 2.5, -2.5, -2.5);
glTexCoord2f( 1.0, 0.0); glVertex3f(-2.5, -2.5, -2.5);
glTexCoord2f( 1.0, 1.0); glVertex3f(-2.5, 2.5, -2.5);
{Left Face}
glNormal3f(-1.0, 0.0, 0.0);
glTexCoord2f( 1.0, 1.0); glVertex3f(-2.5, 2.5, 2.5);
glTexCoord2f( 0.0, 1.0); glVertex3f(-2.5, 2.5, -2.5);
glTexCoord2f( 0.0, 0.0); glVertex3f(-2.5, -2.5, -2.5);
glTexCoord2f( 1.0, 0.0); glVertex3f(-2.5, -2.5, 2.5);
{Right Face}

```

```

    glNormal3f( 1.0, 0.0, 0.0);
    glTexCoord2f( 1.0, 1.0);
    glVertex3f( 2.5, 2.5,-2.5);
    glTexCoord2f( 0.0, 1.0);
    glVertex3f( 2.5, 2.5, 2.5);
    glTexCoord2f( 0.0, 0.0);
    glVertex3f( 2.5,-2.5, 2.5);
    glTexCoord2f( 1.0, 0.0);
    glVertex3f( 2.5,-2.5,-2.5);
    {Top Face}
    glNormal3f( 0.0, 1.0, 0.0);
    glTexCoord2f( 1.0, 1.0);
    glVertex3f( 2.5, 2.5,-2.5);
    glTexCoord2f( 0.0, 1.0);    glVertex3f(-
2.5, 2.5,-2.5);
    glTexCoord2f( 0.0, 0.0);    glVertex3f(-
2.5, 2.5, 2.5);
    glTexCoord2f( 1.0, 0.0);
    glVertex3f( 2.5, 2.5, 2.5);
    {Bottom Face}
    glNormal3f( 0.0,-1.0, 0.0);
    glTexCoord2f( 1.0, 1.0);    glVertex3f(-
2.5,-2.5,-2.5);
    glTexCoord2f( 0.0, 1.0);
    glVertex3f( 2.5,-2.5,-2.5);
    glTexCoord2f( 0.0, 0.0);
    glVertex3f( 2.5,-2.5, 2.5);
    glTexCoord2f( 1.0, 0.0);    glVertex3f(-
2.5,-2.5, 2.5);

    glEnd;
    glEndList;

```

```

CubeList:=BackList+1; // Маленький
Куб
glNewList(CubeList, GL_COMPILE);
//glBindTexture(GL_TEXTURE_2D,
textures[1]);
glBegin(GL_QUADS);
{Front Face}
glNormal3f( 0.0, 0.0, 1.0);
glTexCoord2f( 1.0, 1.0);
glVertex3f( 0.5, 0.5, 0.5);
glTexCoord2f( 1.0, 1.0);
glVertex3f( 0, 0.65, 0.5);
glTexCoord2f( 0.0, 1.0);    glVertex3f(-
0.5, 0.5, 0.5);
glTexCoord2f( 0.0, 1.0);    glVertex3f(-
0.65, 0, 0.5);

```

```

//glTexCoord2f( 0.0, 0.0);    glVertex3f(-
0.5,-0.5, 0.5);

```

```

//glTexCoord2f( 1.0, 0.0);
glVertex3f( 0.5,-0.5, 0.5);

```

```

{Back Face}
//glNormal3f( 0.0, 0.0,-1.0);
//glTexCoord2f( 0.0, 1.0);
glVertex3f( 0.1, 0.1,-0.1);
// glTexCoord2f( 0.0, 0.0);
glVertex3f( 0.1,-0.1,-0.1);
// glTexCoord2f( 1.0, 0.0);    glVertex3f(-
0.1,-0.1,-0.1);
// glTexCoord2f( 1.0, 1.0);    glVertex3f(-
0.1, 0.1,-0.1);
glEnd;
glBindTexture(GL_TEXTURE_2D,
textures[1]);
glBegin(GL_QUADS);
{Left Face}
// glNormal3f(-1.0, 0.0, 0.0);
// glTexCoord2f( 1.0, 1.0);    glVertex3f(-
0.1, 0.1, 0.1);
// glTexCoord2f( 0.0, 1.0);    glVertex3f(-
0.1, 0.1,-0.1);
// glTexCoord2f( 0.0, 0.0);    glVertex3f(-
0.1,-0.1,-0.1);
// glTexCoord2f( 1.0, 0.0);    glVertex3f(-
0.1,-0.1, 0.1);
{Right Face}
// glNormal3f( 0.1, 0.0, 0.0);
// glTexCoord2f( 0.1, 0.1);
glVertex3f( 0.1, 0.1,-0.1);
// glTexCoord2f( 0.0, 0.1);
glVertex3f( 0.1, 0.1, 0.1);
// glTexCoord2f( 0.0, 0.0);
glVertex3f( 0.1,-0.1, 0.1);
// glTexCoord2f( 0.1, 0.0);    glVertex3f(
0.0,-0.0,-0.1);
glEnd;
glBindTexture(GL_TEXTURE_2D,
textures[2]);
glBegin(GL_QUADS);
{Top Face}

```

```

        // glNormal3f( 0.0, 1.0, 0.0);
        // /glTexCoord2f( 1.0, 1.0);    glVertex3f(
0.1, 0.1,-0.1);
        //glTexCoord2f( 0.0, 1.0);    glVertex3f(-
0.1, 0.1,-0.1);
        //glTexCoord2f( 0.0, 0.0);    glVertex3f(-
0.1, 0.1, 0.1);
        //glTexCoord2f( 1.0, 0.0);
glVertex3f( 0.1, 0.1, 0.1);
        {Bottom Face}
        //glNormal3f( 0.0,-1.0, 0.0);
        //glTexCoord2f( 1.0, 1.0);    glVertex3f(-
0.1,-0.1,-0.1);
        //glTexCoord2f( 0.0, 1.0);
glVertex3f( 0.1,-0.1,-0.1);
        //glTexCoord2f( 0.0, 0.0);
glVertex3f( 0.1,-0.1, 0.1);
        // glTexCoord2f( 1.0, 0.0);    glVertex3f(-
0.1,-0.1, 0.1);
        glEnd;
glEndList;

end;

var
    CurTime: TDateTime;
    MSecs: integer;
begin
    inc(FrameCount);

inc(LastFrameTicks,OpenGLControl1.Frame
eDiffTimeInMSecs);
    if (LastFrameTicks>=1000) then begin

DebugLn(['TExampleForm.OpenGLControl
1Paint Frames per second: ',FrameCount]);
        dec(LastFrameTicks,1000);
        FrameCount:=0;
    end;

    if OpenGLControl1.MakeCurrent then
    begin
        if not AreaInitialized then begin
            myInit;
            InitGL;
            glMatrixMode (GL_PROJECTION);    {
prepare for and then }
            glLoadIdentity ();                { define the
projection }
            glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
{ transformation }
            glMatrixMode (GL_MODELVIEW);

```

```

{ back to modelview matrix }
        glViewport (0, 0,
OpenGLControl1.Width,
OpenGLControl1.Height);
                                { define the
viewport }
        AreaInitialized:=true;
        end;

        CurTime:=Now;
        MSecs:=round(CurTime*86400*1000)
mod 1000;
        if MSecs<0 then MSecs:=1000+MSecs;
        timer:=msecs-LastMsecs;
        if timer<0 then timer:=1000+timer;
        LastMsecs:=MSecs;

        ParticleEngine.MoveParticles;

        glClear(GL_COLOR_BUFFER_BIT or
GL_DEPTH_BUFFER_BIT);
        glLoadIdentity;            { clear the
matrix }
        glTranslatef (0.0, 0.0,-3.0); // -2.5);
{ viewing transformation }
        {rotate}

        glPushMatrix;

        if MoveBackground then begin
            rrx:=rrx-0.6*(timer/10);
            rry:=rry-0.5*(timer/10);
            rrz:=rrz-0.3*(timer/10);
        end;

        glRotatef(rrx,1.0,0.0,0.0);
        glRotatef(rry,0.0,1.0,0.0);
        glRotatef(rrz,0.0,0.0,1.0);

        // draw background
        if blended then begin
            glEnable(GL_BLEND);
            glDisable(GL_DEPTH_TEST);
        end;
        glCallList(BackList);

        glPopMatrix;

        glPushMatrix;

        if MoveCube then begin
            rx:=rx+0.1*(timer/10);

```

```

// ry:=ry+0.25*(timer/10);
// rz:=rz+0.8*(timer/10);

If ry<-2 then da:=0.1;
If ry>2 then da:=-0.1;
ry:=ry+da;

end;

// glRotatef(rx,1.0,0.0,0.0);
// glRotatef(ry,0.0,1.0,0.0);
// glRotatef(rz,0.0,0.0,1.0);

glTranslatef (0.0, ry,-3.0); // -2.5);
{ viewing transformation }

// draw cube
glCallList(CubeList);
if blended then begin
    glDisable(GL_BLEND);
    glEnable(GL_DEPTH_TEST);
end;

glPopMatrix;

if ParticleBlended then
glEnable(GL_BLEND);
ParticleEngine.DrawParticles;

```

```

    if ParticleBlended then
glDisable(GL_BLEND);
//glFlush;
//glFinish;
// Swap backbuffer to front
OpenGLControl1.SwapBuffers;
end;
end;

procedure
TExampleForm.OpenGLControl1Resize(Se
nder: TObject);
begin
    if (AreaInitialized)
and OpenGLControl1.MakeCurrent then
        glViewport (0, 0,
OpenGLControl1.Width,
OpenGLControl1.Height);
end;

{ TglTexture }

destructor TglTexture.Destroy;
begin
    if Data<>nil then FreeMem(Data);
    inherited Destroy;
end;

end.

```

2.Файл описания

2.1 Задание

Создать программный продукт по 3-х мерному отображению средствами OpenGL- перемещение заготовки трапеция.

2.2 Общий алгоритм работы

За основу брался пример с заготовкой в виде куба. В ходе выполнения работы изменил заготовку куба на трапецию. Затем добавил перемещение трапеции по вертикальной линии. Все изменения представлены в текстовой части программы.

2.3 Основные функции процедуры

Рисование примитивов

Процедура рисования заключается в командные скобки

`glBegin(mode)`

... // команды, указывающие вершины фигуры
glEnd;

Главное назначение командных скобок – это задание режима, определяющего как соединять точки (вершины). Вершины задаются своими координатами (число координат зависит от пространства изображения) с помощью команд glVertex {2,3,4} {s, i, f, d} (arg).

Режим (mode), задающий правило соединения точек, определяет примитив. К примитивам относятся точки, линии, связанные линии, замкнутые линии, треугольники, связанные треугольники, четырёхугольники, связанные четырёхугольники и многоугольники.

Пример: задание стороны трапеции

```
glNormal3f( 0.0, 0.0, 1.0);
    glVertex3f( 0.5, 0.5, 0.5);
    glVertex3f( 0, 0.65, 0.5);
    glVertex3f(-0.5, 0.5, 0.5);
    glVertex3f(-0.65, 0, 0.5);
glEnd;
```

mode	Описание
GL_POINTS	Каждый вызов glVertex задает отдельную точку. Рисуется N точек
GL_LINES	Каждая пара вершин задает отрезок. Рисуется N/2 линий
GL_LINE_STRIP	Рисуется ломанная. Элементы n и n+1 определяют отрезок n. рисуется N – 1 отрезков
GL_LINE_LOOP	Рисуется ломанная, причем ее последняя точка соединяется с первой. элементы n и n+1 определяют отрезок n. последняя линия определяется элементом N и 1. рисуется N отрезков
GL_TRIANGLES	Каждые три вызова glVertex задают треугольник. элементы 3n – 2, 3n – 1, 3n определяют треугольник n. Рисуется N/3 треугольников.
GL_TRIANGLE_STRIP	Рисуются треугольники с общей стороной. Для нечетного n, элементы n, n+1, n+2 определяют треугольник n. Для четного n элементы n+1, n+2 определяют треугольник n. Рисуется N – 1 треугольников.
GL_TRIANGLE_FAN	Рисует группу соединенных треугольников. Один треугольник определяется для каждого элемента после двух предыдущих. Два последних элемента соединяются с первым. рисуется N – 2 треугольников.
GL_QUADS	Каждые четыре вызова glVertex задают четырехугольник. рисуется N/4 четырехугольника.
GL_QUAD_STRIP	Четырехугольники с общей стороной. соединяя четные элементы с четными, а нечетные с нечетными.

Сдвиг

Преобразование сдвига смещает точки в новые позиции в соответствии с заданным

вектором смещения. Перенос системы координат осуществляется командой

```
glTranslate (arg1, arg2, arg3)
```

arg1, arg2, arg3 – величины переноса по каждой из осей.

Для поворота вокруг произвольной фиксированной точки сначала нужно выполнить преобразование сдвига, совмещающую заданную фиксированную точку с началом координат, потом выполнить преобразование поворота вокруг начала координат, а затем обратное преобразование сдвига. Порядок манипуляции с системой координат: вначале перенос, затем поворот, по окончании рисования – в обратном порядке: поворот, затем перенос.

Наложение текстуры

Создание текстуры в памяти

После того как образ подготовлен, можно создавать текстуру в памяти. Для этого в OpenGL предусмотрены две команды: одна для одномерного и вторая для двумерного вариантов образа (обе работают только в режиме RGBA).

```
glTexImage1D (void glTexImage2D (  
    GLenum target, GLenum target,  
    GLint level, GLint level,  
    GLint components, GLint components,  
    GLsizei width, GLsizei width,  
    GLint border, GLsizei height,  
    GLenum format, GLint border,  
    GLenum type, GLenum format,  
    const GLvoid* pixels) GLenum type,  
    const GLvoid* pixels)
```

При создании текстуры можно определить несколько образов с различным разрешением. Если текстура имеет размер $2n \times 2m$, то можно определить $\max \{n, m\} + 1$ уменьшенных массивов. Первый имеет размер $2n \times 2m$, второй – $2n-1 \times 2m-1$, и т.д., пока последний не будет иметь размер 1×1 . Команды `glTexImage*D` предоставляют возможность определить $p = \max \{n, m\}$ таких массивов, в каждом из которых хранится уменьшенный образ исходного изображения. Наличие таких массивов позволит OpenGL использовать меньший образ для меньшего объекта, а больший для большего. Другими словами, чем меньше объект, тем меньше его деталей удастся рассмотреть.

Параметры текстуры

Один элемент на экране может покрывать несколько элементов массива образа, и,

чтобы избежать проблем, связанных с лестничным эффектом, необходимо учитывать все затрагивающие этот массив элементы.

Для этого определяются четыре точки в массиве образа, которые отображаются в четыре угла элемента на экране. Эти точки соединяются, и образуется четырехугольник. Значения попадающих в него элементов взвешиваются с учетом доли каждого элемента, содержащейся в многоугольнике, и затем суммируются.

Для учета особенностей текстуры необходимо настроить параметры текстуры, что можно сделать с помощью команды

`glTexParameter [i, f, v] (target, pname, param)`

`target` – определяет, с какой текстурой предполагается работать, – одномерной или двумерной

`pname` – определяет символическое имя параметра текстуры:

`param` определяет значение для параметра `pname`

2.4 Скриншоты

- 1.OPENGL_Краснов.CHM
- 2.Основы_OGL_Тарасов.chm
- 3.Учебник_Open_GL.chm