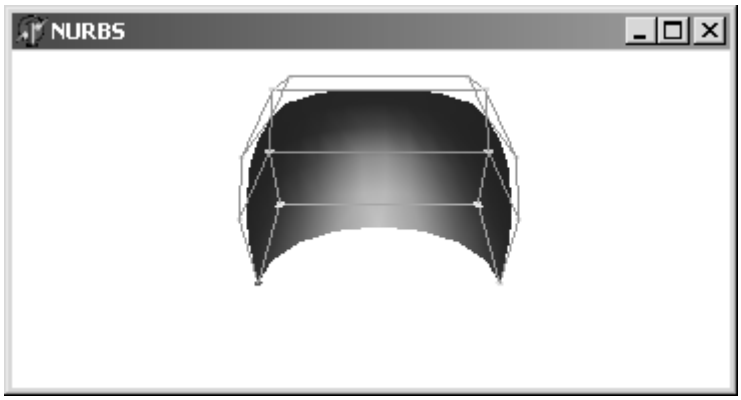
فصل بیستم

NURBS



مقدمه :

توابع NURBS (Non-Uniform Rational B-Spline) موجود در کتابخانه Glu32.dll نقاط کنترلی B-Spline را به نقاط کنترلی بزییر معادل آن تبدیل می کند و سپس توسط روتین های محاسبه گر OpenGL آنها را ترسیم می نماید .

الگوریتم استفاده از امکانات NURBS :

۱- اگر مایل هستید که از نورپردازی به همراه سطوح NURBS استفاده کنید تابع glEnable را به همراه آرگومان GL_AUTO_NORMAL بکار برید تا بردارهای نرمال سطح بصورت خودکار محاسبه شوند .

۲- سپس تابع gluNewNurbsRenderer اشاره گری را به شیء NURBS ایجاد می کند .
۳- در صورت تمایل از دستور gluNurbsProperty برای انتخاب مقادیر رندر کردن مانند حداکثر اندازه خطوط یا چند ضلعی هایی که برای رندر کردن شیء NURBS بکار می روند ، استفاده کنید .
۴- سطح دلخواه خود را با فراخوانی تابع gluBeginSurface آغاز کنید .
۵- برای تولید و رندر کردن سطوح از تابع gluNurbsSurface حداقل یکبار استفاده کنید . تکرار آنها به دفعات مکرر ممکن است برای تولید بردارهای نرمال و یا مختصات بافت ها صورت گیرد .
۶- برای تکمیل ترسیم ، تابع gluEndSurface بکار می رود .
۷- هنگام اتمام کار از تابع gluDeleteNurbsRenderer برای آزاد سازی حافظه بکار رفته استفاده نمایید .

اگر مایل هستید که به معدنی از مثالهای حرفه ای OpenGL دسترسی پیدا کنید به آدرس های زیر مراجعه نمایید :

www.pobox.com/~nate/glut.html
http://reality.sgi.com/opengl/glut3/glut3.html

و یا در Google.com عبارت زیر را جستجو کنید :

GLUT source code

مروری بر توابع :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
GLUnurbsObj* gluNewNurbsRenderer(void);	Type TGLUnurbsObj = record end; PGLUnurbsObj = ^TGLUnurbsObj; Function gluNewNurbsRenderer: PGLUnurbsObj; stdcall; external 'GLU32.DLL';

توضیح :

این تابع یک شیء NURBS را خلق کرده و اشاره گری را به آن بر می گرداند . اگر خروجی آن صفر باشد به این معنا است که حافظه کافی برای ایجاد شیء وجود ندارد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluNurbsProperty(GLUnurbsObj *nobj, GLenum property, GLfloat value);	Procedure gluNurbsProperty(nobj: PGLUnurbsObj; aproperty: GLenum; value: GLfloat); stdcall; external 'GLU32.DLL';

توضیح :

این تابع خواص NURBS را تنظیم می کند .

پارامتر nobj شیء NURBS می باشد که توسط تابع gluNewNurbsRenderer ایجاد گشته است .

property : خاصیتی که باید تنظیم شود . مقادیر زیر برای آن مجاز می باشند .

GLU_SAMPLING_TOLERANCE, GLU_DISPLAY_MODE, GLU_CULLING, GLU_AUTO_LOAD_MATRIX, GLU_PARAMETRIC_TOLERANCE, GLU_SAMPLING_METHOD, GLU_U_STEP, and GLU_V_STEP.

value : مقداری که خاصیت مورد نظر را تنظیم می نماید و یکی از ۳ مقدار زیر برای آن مجاز می باشد .

GLU_PATH_LENGTH, GLU_PARAMETRIC_ERROR, or GLU_DOMAIN_DISTANCE.

توضیحاتی در مورد ثوابت فوق :

GLU_SAMPLING_TOLERANCE : هنگامیکه روش نمونه برداری به GLU_PATH_LENGTH تنظیم می شود ، حداکثر طول را بر حسب نقطه (Pixel) تعیین می کند . مقدار پیش فرض ۵۰ می باشد .

GLU_DISPLAY_MODE : در این حالت پارامتر Value معین می کند که سطح Nurbs چگونه رندر شود . در این حالت آرگومان Value مقادیر زیر را می پذیرد :

Constant	Result
GLU_FILL	The surface is rendered as a set of polygons. This is the default value.
GLU_OUTLINE_POLYGON	The NURBS library draws only the outlines of the polygons created by tessellation.
GLU_OUTLINE_PATCH	Only the outlines of patches and trim curves defined by the user are drawn

GLU_CULLING : در این حالت پارامتر Value مقداری Boolean خواهد بود . هنگامیکه مقدار آن به GL_TRUE تنظیم می شود، نقاطی از سطوح که خارج از دریچه دید قرار دارند ترسیم نخواهند شد . مقدار پیش فرض آن GL_FALSE می باشد .

GLU_PARAMETRIC_TOLERANCE : هنگامیکه روش نمونه برداری به GLU_PARAMETRIC_ERROR تنظیم می شود ، حداکثر فاصله را برحسب نقطه تعیین می کند . مقدار پیش فرض ۰/۵ است .

GLU_SAMPLING_METHOD : طریقه tessellation سطح را معین می کند . سه مقدار زیر برای آن مجاز هستند .

Value	Meaning
GLU_PATH_LENGTH	The default value. Specifies that surfaces rendered with the maximum length, in pixels, of the edges of the tessellation polygons are no greater than the value specified by GLU_SAMPLING_TOLERANCE.
GLU_PARAMETRIC_ERROR	Specifies that in rendering the surface, the value of GLU_PARAMETRIC_TOLERANCE specifies the maximum distance, in pixels, between the tessellation polygons and the surfaces they approximate.
GLU_DOMAIN_DISTANCE	Specifies, in parametric coordinates, how many sample points per unit length to take in the *u* and *v* dimensions.

GLU_U_STEP و GLU_V_STEP : تعداد نقاط نمونه برداری را به ازای واحد طول در امتداد بعد u و یا v در مختصات پارامتریک معین می کند . این ثوابت هنگامی بکار می روند که GLU_SAMPLING_METHOD به GLU_DOMAIN_DISTANCE تنظیم شده باشد . مقدار پیش فرض برای هر کدام ۱۰۰ می باشد .

GLU_AUTO_LOAD_MATRIX : در این پارامتر value مقداری Boolean خواهد بود و هنگامیکه به GL_TRUE تنظیم می شود ، کتابخانه NURBS ماتریس های Projection ، ModelView و ViewPort را برای انجام محاسباتش بصورت خودکار بارگذاری می کند .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluBeginSurface(GLUnurbsObj *nobj); void gluEndSurface(GLUnurbsObj *nobj);	Procedure gluBeginSurface (nobj: PGLUnurbsObj); stdcall; external 'GLU32.DLL'; Procedure gluEndSurface(nobj: PGLUnurbsObj); stdcall; external 'GLU32.DLL';

توضیح :

توابع فوق آغاز و پایان تعریف یک سطح NURBS را مشخص می کنند .

پارامتر nobj شیء NURBS می باشد که توسط تابع gluNewNurbsRenderer ایجاد گشته است .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluNurbsSurface(GLUnurbsObj *nobj, GLint sknot_count, GLfloat *sknot, GLint tknot_count, GLfloat *tknot, GLint s_stride, GLint t_stride, GLfloat *ctlarray, GLint sorder, GLint torder, GLenum type);	Procedure gluNurbsSurface(nobj: PGLUnurbsObj; sknot_count: GLint; sknot: PGLfloat; tknot_count: GLint; tknot: PGLfloat; s_stride: GLint; t_stride: GLint; ctlarray: PGLfloat; sorder: GLint; torder: GLint; atype: GLenum); stdcall; external 'GLU32.DLL';

توضیح :

شکل رویه و سطح NURBS را تعریف می کند .

آرگومانها :

پارامتر nobj شیء NURBS می باشد که توسط تابع gluNewNurbsRenderer ایجاد گشته است .

sknot_count : تعداد گره ها در جهت پارامتری u .

sknot : آرایه ای از sknot_count در جهت پارامتری u .

tknot_count : تعداد گره ها در جهت پارامتری v .

tknot : آرایه ای از tknot_count در جهت پارامتری v .

s_stride : فاصله بین نقاط کنترلی در جهت پارامتری u در ctlarray .

t_stride : فاصله بین نقاط کنترلی در جهت پارامتری v در ctlarray .

ctlarray : آرایه حاوی نقاط کنترلی برای سطح NURBS .

sorder : درجه سطح NURBS در جهت پارامتری u .

torder : درجه سطح NURBS در جهت پارامتری v .

type : نوع سطح را معین می کند و تنها ثوابت محاسبه گرهای ۲ بعدی مانند GL_MAP2_VERTEX_3 را می پذیرد .

با توجه به پارامترهای فوق تعداد نقاط کنترلی باید مساوی حاصلضرب زیر باشد :

(sknot_count-sorder)*(tknot_count-torder)

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluDeleteNurbsRenderer(GLUnurbsObj *nobj);	Procedure gluDeleteNurbsRenderer(nobj: PGLUnurbsObj); stdcall; external 'GLU32.DLL';

توضیح :

تابع فوق سبب تخریب شیء NURBS می شود و حافظه مورد مصرف آنرا آزاد می کند .

پارامتر nobj شیء NURBS می باشد که توسط تابع gluNewNurbsRenderer ایجاد گشته است .

برنامه فصل :

```
unit ch20;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs ,OpenGL , SPF ;

type
  TForm1 = class(TForm)
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

  f_Hdc : LongInt;

implementation

{$R *.dfm}

var
  // User vaiables
  ShowLines : Boolean;
  Wireframe : Boolean;
  Selection : Boolean;         // selection buffer status
  SelectedControl : Integer;
//  Xcoord, Ycoord : Integer;    // mouse movement
```

```
  glMouseDown : Boolean;

  // Nurb variables
  Nurb : GLUnurbsObj;
  numPoints : Integer = 4;
  Knots : Array[0..7] of glFloat =
     (0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0);
  ControlPoints : Array[0..3, 0..3, 0..2] of glFloat = (
  ((-7.5, -8.0, 0.0), (-9.8, -2.7, 0.0 ),
  (-11.0, 4.0, 0.0 ), (-9.5,  9.5, 0.0)),
  ((-5.0, -8.3, 7.5), (-5.3, -7.2, 10.0),
   (-6.0, -1.8, 10.0), (-7.0,  7.8, 5.0)),
  (( 5.0, -8.3, 7.5), ( 5.3, -7.2, 10.0),
   ( 6.0, -1.8, 10.0), ( 7.0,  7.8, 5.0)),
  (( 7.5, -8.0, 0.0), ( 9.8, -2.7, 0.0 ),
   ( 11.0, 4.0, 0.0 ), ( 9.5,  9.5, 0.0)));

  // Lights
  LightPos : Array [0..3] of GLfloat = (0.0, 20.0, -24.0, 1);
  Specular : Array[0..3] of glFloat = (0.7, 0.7, 0.7, 1.0);
  Shine    : glFloat = 100.0;

procedure glBindTexture(target: GLenum; texture: GLuint);
        stdcall; external opengl32;

{------------------------------------------------------------------}
{  Function to convert int to string.                              }
{------------------------------------------------------------------}
function IntToStr(Num : Integer) : String;
begin
  Str(Num, result);
end;


procedure DrawControlPoints;
var I, J : Integer;
begin
  // Draw the lines between the points
  if ShowLines and NOT(Selection) then
  begin
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glColor3f(0, 1, 0);
    glBegin(GL_QUADS);
    For I :=0 to 2 do
      For J :=0 to 2 do
      begin
        glVertex3fv(@ControlPoints[I, J]);
        glVertex3fv(@ControlPoints[I,  J+1]);
        glVertex3fv(@ControlPoints[I+1, J+1]);
        glVertex3fv(@ControlPoints[I+1, J]);
      end;
    glEnd();
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
 end;

  // Draw all the points in the array
  For I :=0 to 3 do
  begin
```

```
    For J :=0 to 3 do
    begin
      glLoadName(I*4 + J);
      glBegin(GL_QUADS);
      //** glBegin(QUADS) has to sit here for selection buffer !!!!

        if SelectedControl = I*4 + J then
          glColor3f(1, 0, 0)
        else
          glColor3f(1, 1, 0);

        glVertex3f(ControlPoints[I, J, 0]-0.2,
           ControlPoints[I, J, 1]-0.2, ControlPoints[I, J, 2]);
        glVertex3f(ControlPoints[I, J, 0]+0.2,
           ControlPoints[I, J, 1]-0.2, ControlPoints[I, J, 2]);
        glVertex3f(ControlPoints[I, J, 0]+0.2,
           ControlPoints[I, J, 1]+0.2, ControlPoints[I, J, 2]);
        glVertex3f(ControlPoints[I, J, 0]-0.2,
           ControlPoints[I, J, 1]+0.2, ControlPoints[I, J, 2]);
      glEnd()
    end;
  end;
end;



{------------------------------------------------------------------}
{  Function to draw the actual scene                               }
{------------------------------------------------------------------}
procedure RenderScene();
begin
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
    // Clear The Screen And The Depth Buffer
  glLoadIdentity();
   // Reset The View

  glTranslatef(0.0,0.0,-35);

  glRotatef(-45, 1.0, 0.0, 0.0);
  glColor3f(0, 0.1, 0.6);

  if WireFrame then glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

  // draw the nurb surface
  if Selection = FALSE then
  begin
    gluBeginSurface(Nurb);
      gluNurbsSurface(Nurb,
          8, @Knots,        // No. of knots and knot array u direction
          8, @Knots,        // No. of knots and knot array v direction
          4 * 3,           // Distance between control points in u dir.
          3,              // Distance between control points in v dir.
          @ControlPoints,    // Control points
          4, 4,            // u and v order of surface
          GL_MAP2_VERTEX_3); // Type of surface
    gluEndSurface(Nurb);
  end;
```

```
  // Show the control points
  DrawControlPoints;

  SwapBuffers(f_Hdc);
end;


{------------------------------------------------------------------}
{  Initialise OpenGL                                               }
{------------------------------------------------------------------}
procedure glInit();
begin
  glClearColor(0.0, 0.0, 0.0, 0.0);  // Black Background
  glShadeModel(GL_SMOOTH);           // Enables Smooth Color Shading
  glClearDepth(1.0);                 // Depth Buffer Setup
  glDepthFunc(GL_LESS);                        // The Type Of Depth Test To Do
  glEnable(GL_DEPTH_TEST);           // Enable Depth Buffer

  glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    //Realy Nice perspective calculations

  ShowLines := TRUE;
  Selection := FALSE;

  // setup the lights
  glEnable(GL_LIGHTING);
  glLightfv(GL_LIGHT0, GL_POSITION, @LightPos);
  glEnable(GL_LIGHT0);

  // setup the material properties
  glEnable(GL_COLOR_MATERIAL);
  // glColorMaterial function causes a material
  //color to track the current color
  glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
  glMaterialfv(GL_FRONT, GL_SPECULAR, @Specular);
  glMaterialfv(GL_FRONT, GL_SHININESS, @shine);

  // setup nurb drawing properties
  Nurb := gluNewNurbsRenderer();
  gluNurbsProperty(Nurb, GLU_SAMPLING_TOLERANCE, 40.0);
    // Quality - Specifies the maximum length, in pixels, to use
  gluNurbsProperty(Nurb, GLU_DISPLAY_MODE, GLU_FILL);
     // Fillstyle
end;


{------------------------------------------------------------------}
{  Handle window resize                                            }
{------------------------------------------------------------------}
procedure glResizeWnd(Width, Height : Integer);
begin
  if (Height = 0) then             // prevent divide by zero exception
    Height := 1;
  glViewport(0, 0, Width, Height);  // Set the viewport for the OpenGL window
  glMatrixMode(GL_PROJECTION);      // Change Matrix Mode to Projection
  glLoadIdentity();                 // Reset View
  gluPerspective(45.0, Width/Height, 1.0, 100.0);
  // Do the perspective calculations. Last value = max clipping depth
```

```
  glMatrixMode(GL_MODELVIEW);        // Return to the modelview matrix
  glLoadIdentity();                  // Reset View
end;


{-------------------------------------------------------------------}
{  Processes all the keystrokes                                     }
{-------------------------------------------------------------------}
procedure ProcessKeys(Keys: Word);
begin
  if keys=Ord('1') then SelectedControl :=1;
  if keys=Ord('2') then SelectedControl :=2;
  if keys=Ord('3') then SelectedControl :=3;
  if keys=Ord('4') then SelectedControl :=4;
  if keys=Ord('5') then SelectedControl :=5;
  if keys=Ord('6') then SelectedControl :=6;
  if keys=Ord('7') then SelectedControl :=7;
  if keys=Ord('8') then SelectedControl :=8;
  if keys=Ord('9') then SelectedControl :=9;
  if keys=Ord('0') then SelectedControl :=0;

  // show control lines
  if keys=Ord('L') then
  begin
    ShowLines :=Not(ShowLines);
//    keys[Ord('L')] :=FALSE;
  end;

  // show wireframe
  if keys=Ord('W') then
  begin
    Wireframe :=Not(Wireframe);
//    keys[Ord('W')] :=FALSE;
  end;

  if (keys=VK_UP)    then ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 1] := ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 1] + 0.1;
  if (keys=VK_DOWN)  then ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 1] := ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 1] - 0.1;
  if (keys=VK_RIGHT) then ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 0] := ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 0] + 0.1;
  if (keys=VK_LEFT)  then ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 0] := ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 0] - 0.1;
  if (keys=VK_PRIOR) then ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 2] := ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 2] + 0.1;
  if (keys=VK_NEXT)  then ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 2] := ControlPoints[SelectedControl DIV 4,
    SelectedControl MOD 4, 2] - 0.1;

end;
```

```
{------------------------------------------------------------------}
{  Processes all mouse Clicks                                      }
{------------------------------------------------------------------}
procedure MouseClick(X, Y : Integer);
var selectBuff : Array[0..23] of glUint;
    viewport : Array[0..3] of glUint;
    hits  : glUint;
begin
  // Select buffer parameters
  glGetIntegerv(GL_VIEWPORT, @viewport);
  // Viewport = [0, 0, width, height]
  glSelectBuffer(512, @selectBuff);

  // Enter to selection mode
  glRenderMode(GL_SELECT);
  Selection :=TRUE;

  // Clear Select Buffer;
  glInitNames();
  glPushName(0);

  glMatrixMode(GL_PROJECTION);
  glPushMatrix();

    glLoadIdentity();

    // setup a viewing volume.  (x, y, width, height, viewport)
    // NOTE : y has -27 to account to caption bar
    gluPickMatrix(x, viewport[3]-y-27, 1, 1, @viewport);
       // Set-up pick matrix
    gluPerspective(45.0, viewport[2]/viewport[3], 1.0, 100.0);
     // Do the perspective calculations. Last value = max clipping depth
    glMatrixMode(GL_MODELVIEW);

    // Render all scene and fill selection buffer
    RenderScene;

    glMatrixMode(GL_PROJECTION);
  glPopMatrix();
  glMatrixMode(GL_MODELVIEW);

  // Get hits and go back to normal rendering
  hits := glRenderMode(GL_RENDER);

  // Get the element in the selection buffer if there was a hit.
  // we are just going to grab the first item in the list

  // selectBuff[0] = no names stored in first hit layer
  // selectBuff[1..2] = min..max depth of hit
  // selectBuff[3] = name of item in first second hit   <---
  // selectBuff[4] = no names in seconds hit
  // selectBuff[5..6] = depth of second hit
  // selectBuff[7] = name of second hit item   <----
  SelectedControl :=-1;
  if (hits > 0) then
    SelectedControl := selectBuff[3];

  Selection :=FALSE;
```

```
end;


procedure glMouseMove(X, Y : Integer);
//var viewport : Array[0..3] of glUint;
//    ModelViewMatrix  : Array[0..15] of glDouble;
//    ProjectionMatrix : Array[0..15] of glDouble;
//    SceneX, SceneY, SceneZ : glDouble;
begin
{  glGetIntegerv(GL_VIEWPORT, @viewport);
  glGetDoublev(GL_MODELVIEW_MATRIX,  @ModelViewMatrix);
  glGetDoublev(GL_PROJECTION_MATRIX, @ProjectionMatrix);

  // The gluUnProject function maps window
  coordinates to object coordinates.
  gluUnProject(X, viewport[3]-y-27, 0.97,
            @ModelViewMatrix, @ProjectionMatrix, @viewport,
            SceneX, SceneY, SceneZ);
}
  ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 0] := (X-400)/30;
  ControlPoints[SelectedControl DIV 4,
   SelectedControl MOD 4, 1] := (272-Y)/20;
  //SceneY;
end;


procedure TForm1.FormResize(Sender: TObject);
begin
   wglMakeCurrent(f_Hdc,hrc); //activate the RC
   glResizeWnd(Width, Height);
   InvalidateRect(Handle, nil, False);// DrawGLScene; Dr
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
 CleanUp(f_Hdc);// Clean up and terminate.
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 f_Hdc := GetDC(handle);
 SetDCPixelFormat(f_Hdc,16,16);// Create a rendering context.
 glInit();
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
 wglMakeCurrent(f_Hdc,hrc); //activate the RC
 RenderScene;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
 MouseClick(x,y);
 glMouseDown :=TRUE;
 InvalidateRect(Handle, nil, False);// DrawGLScene;
```

```
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin

 if (SelectedControl >= 0) AND (glMouseDown) then
   glMouseMove(x,y);
 InvalidateRect(Handle, nil, False);// DrawGLScene;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
 ProcessKeys(key);
 InvalidateRect(Handle, nil, False);// DrawGLScene;
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
       glMouseDown :=FALSE;
end;

end.
```