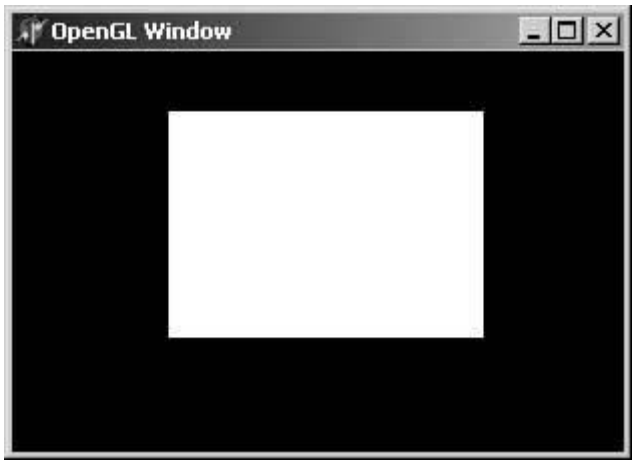
فصل دوم

بر پایی پنجره مخصوص OpenGL



مفاهیم اولیه برنامه نویسی OpenGl

فرمت نقطه ای : احتمالا می دانید که کارتهای VGA ، حالت ۲۵۶ رنگ را پشتیبانی می کنند که به آن عمق رنگ ۸ بیتی نیز گفته می شود . زیرا هر نقطه روی صفحه یک مقدار ۸ بیتی (با قابلیت ۲۵۶ رنگ مختلف) را دارا است . اگر به رنگ های واقع گرایانه تری نیاز باشد به کارتهای ویدیویی حالت ۱۵ بیتی (۳۲۷۶۸ رنگ) ، ۱۶ بیتی (۶۵۵۳۶ رنگ) ، ۲۴ بیتی (۱۶ میلیون رنگ) یا بیشتر نیاز خواهد بود .اگر کارت گرافیکی شما یک یا دو مگابایت حافظه روی آن داشته باشد از عمق رنگ بیشتری نیز می توانید استفاده کنید . برای مثال اگر شما مایل هستید که ۶۵۵۳۶ رنگ

را در حالت صفحه نمایش 800×600 داشته باشید ، حداقل به یک مگابایت حافظه ویدیویی احتیاج دارید :

2 bytes of color/pixel * 800*600 = 960000 bytes of memory

فرمت نقطه ای توضیح فرمت واقعی پنجره ای معین می باشد .

OpenGl و سایر API های گرافیکی مفاهیمی مانند **بافرهای رویی و پشتی** را نیز ارائه کرده اند. بافر (حافظه میانگیر) رویی آن چیزی است که هم اکنون در صفحه نمایش دیده می شود و بافر پشتی عمدتا در هنگام پویانمایی (انیمیشن) بکار برده شده و قسمتی از شیء متحرک در حال نمایش ، لحظه ای قبل در بافر پشتی (پنهان) ترسیم گردیده و ناگهان در بافر رویی نمایش داده می شود .

علاوه بر اینها بعضی از کارتهای گرافیکی ، نمایش **استریو** را که بافرهایی در سمت راست و چپ صحنه دارا می باشند ، نیز پشتیبانی می کنند و بدیهی است که به حافظه بیشتری نیاز دارند. خوشبختانه امروزه اینگونه کارتها به وفور و ارزان ، در اختیار عموم قرار دارد . همانطور که ذکر شد ، انتخاب فرمت نقطه ای وابسته است به توانایی های کارت گرافیکی رایانه شما .

مفهوم جدید دیگر **زمینه ارائه یا رندر کردن** (Rendering Context (RC)) می باشد . RC در OpenGL معادل DC در GDI ویندوز می باشد (Device Context) . RC مدخلی است که از آن فراخوانی های OpenGL ارائه می شوند. DC در GDI ویندوز مخزن متغیرهای حالت مانند رنگ قلم جاری می باشد و معادل آن در OpenGL ، رکوردهایی هستند که وضعیت تنظیمات جاری و مسیر فراخوانی توابع را ذخیره می کنند . ایجاد DC و RC قسمتی از *تشریفاتی* هستند که قبل از فراخوانی توابع OpenGL باید صورت گیرند .

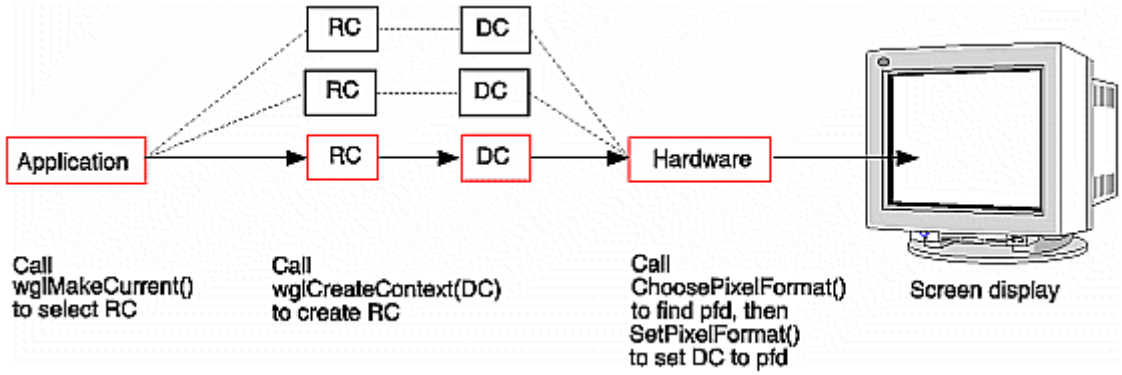
در اینجا لازم به نظر می رسد که توضیحات بیشتری درمورد انواع بافرها در OpenGL داده شود. بافر OpenGL تشکیل شده است از جمع تمام بافرهای مورد استفاده ی آن . این بافرها شامل بافر **رنگی** ، بافر **عمق** ، بافر **استنسیل** (Stencil) و بافر **انباشتگی** (Accumulation) می باشند .

بافرهای رنگی حاوی اطلاعات نقطه در مورد اینکه آیا از اندیس رنگی استفاده می شود و یا مقادیر RGBA (A یا آلفا مقدار شفافیت شیء را تنظیم می کند) ، هستند.

بافر عمق (بافر Z) حاوی مقادیر عمق برای هر نقطه است . نقاطی با مقادیر عمق بیشتر ، دورتر به نظر می رسند و نقاطی با مقادیر کمتر عمق بر روی نقاطی عمیق تر قرار خواهند گرفت (اگر هر دو در یک مکان واقع شده باشند).

بافر استنسیل ترسیمات برروی نواحی معینی از صفحه را محدود می کند .

بافر انباشتگی برای انباشتن تعداد زیادی تصویر در یک تصویر ترکیبی بکار مـی رود . بـرای استفاده از هرکدام از این بافرها باید ، فرمت نقطه ای مناسب ایجاد گردد .



آغاز به کار با OpenGL :

اولین قدم در ایجاد هر برنامه OpenGL **ساخت پنجره** ای است که فراخوانی های OpenGL بتوانند در آن اجرا شوند . در OpenGL مفـاهیمی ماننـد بافردوگانـه و اسـتریو وجـود دارد کـه تحـت GDI ویندوز تعریف نشده اند .

بنابراین ساخت یک چنین پنجره ای که بتواند فرمتی قابل قبولی را برای OpenGL فـراهم آورد ، الزامی می باشد و شامل مراحل زیر است :

۱- ایجاد DC

۲- برپایی فرمت نقطه ای

۳- ایجاد RC

۴- اجرای برنامه

۵- حذف RC و DC و خروج از برنامه

عدم موفقیت در هر کدام از مراحل برپایی پنجره دلخواه OpenGL ، سبب خواهد شد که شـما بـا صفحه ای **خالی** مواجه شوید .

مثالی از DC همان Canvas.Handle خودمان می باشد !

مرحله بعد استفاده از تابع SetPixelFormat برای برپایی فرمت نقطه ای مناسب می باشد . توابع مربوطه در فصلی دیگر برای عدم دلسردی شما از برنامه نویسی OpenGL ، بطور مفصلی ارائه می گردند!

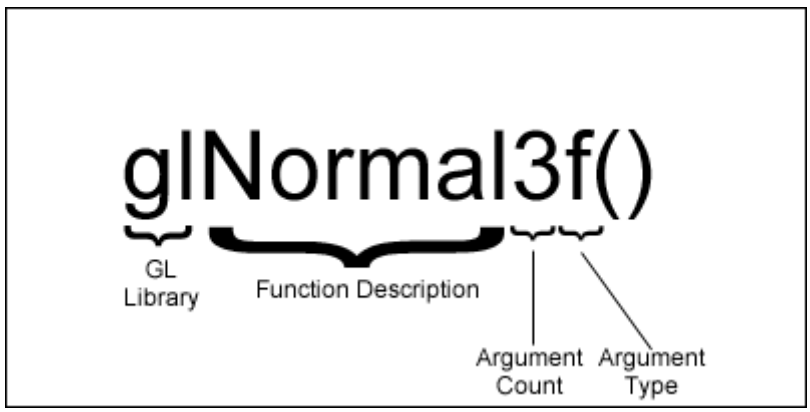
پس از انتخاب فرمت نقطه ای مناسب ، مرحله بعد ایجاد RC می باشد . اینکار با چند تابع که با wgl آغاز می شوند ، انجام می پذیرد (توضیحات مفصل آنها در فصلی دیگر ارائه خواهد شد) . تابع wglCreateContext یک DC مجاز که حاوی فرمت نقطه ای انتخاب شده است را در یافت کرده و یک RC بر می گرداند .

سپس شما نیاز دارید که این RC ایجاد شده را با تابع wglMakeCurrent تبدیل به RC جاری نمایید و خلاص ! در اینجا شما مجاز خواهید بود که از توابع OpenGL استفاده نمایید . همانگونه که ذکر شد موارد فوق در **تمام** برنامه های OpenGL تکرار می شوند ؛ پس می توان برای صرفه جویی در زمان ، آنهارا در یک واحد (Unit) معین قرار داد و استفاده نمود.

آخرین مرحله تکراری در هر برنامه OpenGL **بازگرداندن** منابع اخذ شده به سیستم می باشد (یک تعهد اخلاقی !) .

آشنایی با دستور زبان OpenGL

دستورات OpenGL با پیشوند gl شروع و با یک حرف بزرگ ادامه می یابند مانند glBegin() . بصورت مشابهی ثوابت این کتابخانه برنامه نویسی با GL_ شروع می شوند و تمام حروف مربوطه آن بزرگ می باشند مانند GL_COLOR_BUFFER_BIT .



بعضی از دستورات OpenGL با یک عدد و یک ، دو یا سه حرف در انتها ، برای یادآوری تعداد و نوع آرگومانهای ورودی ، خاتمه می یابند . برای مثال در دستور glVertex3fv() ، '3' به معنای وجود سه آرگومان ورودی ، 'f' به معنای اعشاری بودن آنها و 'v' به معنای این است که آرگومانها به فرم برداری هستند .

بطور خلاصه :

تعداد آرگومانها : ۲ ، ۳ یا ۴ .

نوع های داده ای :

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	Glbyte
s	16-bit integer	Short	Glshort
i	32-bit integer	int or long	GLint, Glsizei
f	32-bit floating-point	Float	GLfloat, Glclampf
d	64-bit floating-point	Double	GLdouble, Glclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, Glboolean
us	16-bit unsigned integer	unsigned short	Glushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, Glbitfield

فرمت ها :

'v' : بیانگر فرمت برداری است .

غیاب 'v' به معنای فرمت اسکالر می باشد .

توضیحی در مورد نوع های عددی OpenGL:

برای اینکه OpenGL مستقل از پلتفرم باشد بجای انواع داده های عددی معمولی از نوع های خاصی استفاده کرده است . برای مثال glFloat تحت ویندوز و بر روی پروسسورهای اینتل و سازگار با آن ، مشابه Single در دلفی و مطابق استاندارد IEEE مقداری ۳۲ بیتی و اعشاری می باشد .

و اما برنامه این فصل ...

در ادامه برنامه ای که تمام موارد فوق را به اجرا خواهد گذاشت ارائه می گردد .

```
unit ch02;

interface

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

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    { Private declarations }

  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  hrc: HGLRC;  // Permanent Rendering Context

  f_Hdc : LongInt;

  implementation

{$R *.DFM}

procedure InitGL;          // All Setup For OpenGL Goes Here
begin
// select clearing color
   glClearColor( 0, 0, 0, 0);
// initialize viewing values
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity;
   glOrtho( 0, 1, 0, 1, -1, 1);
end;

procedure DrawGLScene();
 // Here's Where We Do All The Drawing!!!
 //Right after glLoadIdentity and before DrawGLScene:=true
begin
  // Enable depth testing and clear the color and depth buffers.
  glEnable(GL_DEPTH_TEST);
  // Clear The Screen And The Depth Buffer
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

  glMatrixMode(GL_PROJECTION);  // Select The Projection Matrix
```

```
  glLoadIdentity();    // Reset The View
  glOrtho( 0, 1, 0, 1, -1, 1);
  // Set the drawing color to blue.
  glColor3f(1.0, 1.0, 1.0);
  glBegin (GL_POLYGON);
     glVertex3f (0.25, 0.25, 0);
     glVertex3f (0.75, 0.25, 0);
     glVertex3f (0.75, 0.75, 0);
     glVertex3f (0.25, 0.75, 0);
   glEnd;
  // Flush the drawing pipeline.
  glFlush;
end;

procedure CleanUp(Handle: HDC); //Properly Kill The Window
begin
  if hrc<> 0 then       //Is There A Rendering Context?
    begin
      //Are We Able To Release Dc and Rc contexts?
      if (not wglMakeCurrent(handle,0)) then
        MessageBox(0,'Release of DC and RC failed.'
                ,' Shutdown Error',MB_OK or MB_ICONERROR);
     //Are We Able To Delete The Rc?
      if (not wglDeleteContext(hRc)) then
       begin
        MessageBox(0,'Release of Rendering Context failed.',
                 ' Shutdown Error',MB_OK or MB_ICONERROR);
        hRc:=0;   //Set Rc To Null
       end;
    end;
end;

procedure SetDCPixelFormat(Handle: HDC;ColorBits,DepthBufferBits:integer);
var
  pfd: TPixelFormatDescriptor;
  nPixelFormat: Integer;

begin
  FillChar(pfd, SizeOf(pfd), 0);

  with pfd do begin
    nSize     := sizeof(pfd);            // Size of this structure
    nVersion  := 1;                      // Version number
    dwFlags   := PFD_SUPPORT_OPENGL Or PFD_DRAW_TO_WINDOW
              Or  PFD_TYPE_RGBA;       // Flags
    iPixelType:= PFD_TYPE_RGBA;          // RGBA pixel values
    cColorBits:= ColorBits;              // 24-bit color
    cDepthBits:= DepthBufferBits;        // 32-bit depth buffer
    iLayerType:= PFD_MAIN_PLANE;         // Layer type
  end;

  nPixelFormat := ChoosePixelFormat(Handle, @pfd);
  //Did We Find A Matching Pixelformat?
  if (nPixelFormat=0) then
    begin
    CleanUp(handle);                     //Reset The Display
      MessageBox(0,'Cant''t Find A Suitable PixelFormat.'
                ,'Error',MB_OK or MB_ICONEXCLAMATION);
```

```
    Halt(1); { Halt right here! }
   end;

  //Are We Able To Set The Pixelformat?
  if (not SetPixelFormat(Handle, nPixelFormat, @pfd)) then
   begin
    CleanUp(handle);                    //Reset The Display
    MessageBox(0,'Cant''t set PixelFormat.'
          ,'Error',MB_OK or MB_ICONEXCLAMATION);
    Halt(1); { Halt right here! }
   end;

  hrc := wglCreateContext(Handle);
  if (hRc=0) then
   begin
   CleanUp(handle);                    //Reset The Display
    MessageBox(0,'Cant''t create a GL rendering context.'
                ,'Error',MB_OK or MB_ICONEXCLAMATION);
    Halt(1); { Halt right here! }
   end;

  //Are We Able To Activate The Rendering Context?
  if (not wglMakeCurrent(Handle, hrc)) then
   begin
   CleanUp(handle);                    //Reset The Display
   MessageBox(0,'Cant''t activate the GL rendering context.'
                ,'Error',MB_OK or MB_ICONEXCLAMATION);
    Halt(1); { Halt right here! }
   end;

end;

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

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

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

procedure TForm1.FormResize(Sender: TObject);
 begin
  // Redefine the viewing volume and viewport
  //when the window size changes.
   wglMakeCurrent(f_Hdc, hrc);
```

```
  // Reset The Current Viewport And Perspective
  // Now, set up the viewing area-select the full client area
  glViewport( 0, 0, Width , Height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();  // Reset The Projection Matrix
  gluOrtho2D(0, Width, 0, Height);
  DrawGLScene();
 end;
end.
```

مرجع توابعی که در این فصل و خصوصا در برنامه فوق بکار رفته اند به شرح زیر است :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void glClearColor(
GLclampf red, GLclampf green,
GLclampf blue, GLclampf alpha);
	type FLOAT = Single;
	type GLclampf = FLOAT;
	Procedure glClearColor(
	red: GLclampf; green: GLclampf;
	blue: GLclampf; alpha: GLclampf);
	stdcall; external 'OPENGL32.DLL';

توضیح :

glClearColor رنگ پاک کننده جاری صفحه را به مقدار RGBA داده شده تنظیم می کند . مقادیر قرمز (R) ، سبز (G) ، آبی (B) و آلفا (A) (شفافیت) در بازه صفر و یک قرار دارند . رنگ پیش فرض معادل (0,0,0,0) می باشد (مشکی) . صفر ، تاریکترین و یک ، روشن ترین است . باترکیب شدتهای مختلف سه رنگ قرمز ، سبز و آبی ، رنگ های بیشماری را می توان تولید کرد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void glClear(GLbitfield mask);	type UNSIGNED_INT = Cardinal;
	type GLbitfield = UNSIGNED_INT;
	Procedure glClear(mask: GLbitfield);
	stdcall; external 'OPENGL32.DLL';

توضیح :

بافر همواره باید قبل از ترسیم صحنه پاک شود ؛ زیرا مقادیر رنگ هر نقطه در بافر ذخیره می شوند . تابع glClear بافر Mask را پاک می کند . Mask یکی از مقادیر زیر و یا ترکیبی از آنها می تواند باشد :

بافر رنگی : GL_COLOR_BUFFER_BIT

بافر عمق : GL_DEPTH_BUFFER_BIT

بافر انباشتگی : GL_ACCUM_BUFFER_BIT

بافر استنسیل : GL_STENCIL_BUFFER_BIT

با فراخوانی تابع فوق تصویر در حال نمایش حفظ شده و تنها بافر یا بافرهایی که بعنوان آرگومان به تابع معرفی می شوند ، پاک می گردند . بطور معمول ابتدا تابع glClearColor فراخوانی می شود و سپس این تابع ، تا glClearColor مؤثر واقع شود .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void glColor3f(GLfloat red,
GLfloat green, GLfloat blue);
	type FLOAT = Single;
	type GLfloat = FLOAT;
	Procedure glColor3f(red: GLfloat;
	green: GLfloat; blue: GLfloat);
	stdcall; external 'OPENGL32.DLL';

توضیح :

glColor3f برای تعیین رنگ اشیای روی صفحه بکار می رود و سه آرگومان اول آن همانند تابع glClearColor می باشد . تمام اشیاء پس از بکار بردن دستور glColor3f از این رنگ استفاده می کنند تا زمانیکه با فراخوانی دیگری از این تابع رنگ تغییر یابد . مثال :

```
glColor3f(0.0, 0.0, 0.0);      black
glColor3f(1.0, 0.0, 0.0);      red
glColor3f(0.0, 1.0, 0.0);      green
glColor3f(1.0, 1.0, 0.0);      yellow
glColor3f(0.0, 0.0, 1.0);      blue
glColor3f(1.0, 0.0, 1.0);      magenta
glColor3f(0.0, 1.0, 1.0);      cyan
glColor3f(1.0, 1.0, 1.0);      white
```

البته همانطور که از نام این تابع معلوم می باشد ، حرف 'f' به معنای اعشاری و اسکالر بودن آرگومانها هستند . نگارشی دیگر از این تابع نیز وجود دارد glColor3fv . در این حالت آرگومان ورودی اشاره گری به یک بردار (یا آرایه) می باشد. مثال :

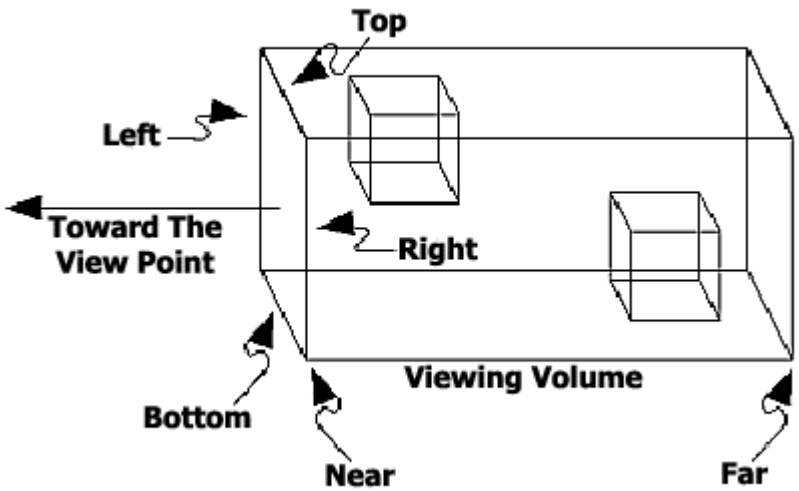
در زبان C	در زبان دلفی
GLfloat color_array[] = {1.0, 0.0, 0.0}; glColor3fv(color_array);	

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void glOrtho(
GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top,
GLdouble zNear, GLdouble zFar);
	type DOUBLE_ = Double;
	type GLdouble = DOUBLE_;
	Procedure glOrtho(
	left: GLdouble; right: GLdouble;
	bottom: GLdouble; top: GLdouble;
	zNear: GLdouble; zFar: GLdouble);
	stdcall; external 'OPENGL32.DLL';

توضیح :

دستوری که دید اورتوگرافیک حجم را بوجود می آورد . در این حالت حجم درحال مشاهده به شکل یک جعبه دیده می شود (شکل زیر).



شکلی از دید اورتوگرافیک .

آرگومانهای left و right مختصات صفحات عمودی برش چپ و راست را معین می سازند. top و bottom مختصات صفحات افقی برش بالا و پایین را معلوم می کنند و آرگومانهای near و far فاصله با صفحات برش نزدیکتر و دورتر هستند .

واژه جدید : Clipping Volume

مکعب مستطیلی است که دستورات ترسیمی شما در آن ارائه و رندر می شوند .

برای مثال دستور glOrtho(0,0,w,h,0,-1,1); دیدی معادل w*h بوجود می آورد بطوریکه نقطه (0, 0) در گوشه بالا سمت چپ آن قرار گرفته است .

دستور gluOrtho2D(left,right,bottom,top); با دستور glOrtho(left,right,bottom,top,-1,1); معادل است و برای ایجاد سیستم دستگاه مختصات دو بعدی بکار می رود .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

Void glBegin(GLenum mode);
void glEnd(void);
	type UNSIGNED_INT = Cardinal;
	type GLenum = UNSIGNED_INT;
	Procedure glBegin(mode: GLenum);
	stdcall; external 'OPENGL32.DLL';
	Procedure glEnd;
	stdcall; external 'OPENGL32.DLL';

توضیح :

برای شروع و پایان دستورات ترسیمی اشکال اولیه که با رئوسشان مشخص شده اند ، بکار می روند (البته بجز لیست های نمایشی) . آرگومان glBegin نوع ترسیمی که باید شروع شود را معین می سازد . برای مثال ;glBegin(GL_TRIANGLES) به معنای این است که می خواهیم رسم یک مثلث را شروع کنیم و ;()glEnd به OpenGL می گوید که ترسیم مثلث انجام شده است . آرگومانهای مجاز تابع glBegin به شرح زیر هستند :

GL_POINTS	با هر راس بصورت یک نقطه برخورد می شود . (حداقل یک راس نیاز است .)
GL_LINES	با هر جفت راس بصورت یک خط مستقل برخورد می شود . (حداقل دو راس نیاز است.)
GL_LINE_STRIP	یک گروه به هم متصل از خطوط را از راس اول تابه آخر ترسیم می کند.
GL_LINE_LOOP	یک گروه به هم متصل از خطوط را از راس اول تابه آخر ترسیم می کند و سپس به اولین باز می گردد .
GL_TRIANGLES	با هر سه راس به صورت یک مثلث مستقل برخورد می کند . (حداقل سه راس نیاز است.)
GL_TRIANGLE_STRIP	یک گروه متصل به هم از مثلث ها را ترسیم می کند . یک مثلث به ازای هر راس پس از دو راس اول تعریف می شود .
GL_TRIANGLE_FAN	یک گروه متصل به هم از مثلث ها را ترسیم می کند . یک مثلث به ازای هر راس پس از دو راس اول تعریف می شود .
GL_QUADS	با هر گروه چهارتایی از رئوس بصورت یک چهار ضلعی مستقل رفتار می کند . (حداقل چهار راس نیاز است.)
GL_QUAD_STRIP	یک گروه متصل به هم از چهار ضلعی ها را ترسیم می کند . یک چهار ضلعی به ازای هر راس پس از جفت اول تعریف می شود .

GL_POLYGON	یک چند ضلعی محدب ترسیم می کند . رئوس ۱ تا n این چند ضلعی را تعریف می کنند . (حداقل سه راس نیاز است .)

لازم به تذکر است که هیچگونه محدودیتی درتعداد رئوس رئوس تعریف شده درمیان glBegin و glEnd وجود ندارد . اشکالی که کاملا تعریف نمی شوند ، ترسیم نخواهند گشت .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glVertex3f(GLfloat x,GLfloat y,GLfloat z);	Procedure glVertex3f(x: GLfloat; y: GLfloat; z: GLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

یک راس از شکلی را که باید بین glBegin و glEnd ترسیم شود ، ایجاد می کند . در OpenGL تمام اشیای هندسی با مجموعه ای از رئوس بیان می شوند . باید خاطر نشان کرد که ترتیب ارائه رئوس بسیار مهم است . x ، y و z مختصات راس می باشند . باز هم تاکید می شود که بکارگیری glvertex3f خارج از جفت glBegin/glEnd رفتاری تعریف نشده را بهمراه خواهد داشت .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glFlush(void);	Procedure glFlush; stdcall; external 'OPENGL32.DLL';

توضیح :

برای اطمینان حاصل کردن از ترسیم تمام مواردی که باید رسم شوند ، در پایان دستورات ترسیمی از این تابع استفاده می شود .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glMatrixMode(GLenum mode);	Procedure glMatrixMode(mode: GLenum); stdcall; external 'OPENGL32.DLL';

توضیح :

ماتریس جاری را تعیین می کند . برای تغییر مکان شیء و همچنین مکان دوربین از ماتریس ها استفاده می شود . در هر لحظه یکی از ماتریس های زیر قابل تغییر هستند :

GL_MODELVIEW : modelview matrix
GL_PROJECTION : projection matrix
GL_TEXTURE : texture matrix

ماتریس قابل تغییر پیش فرض و جاری modelview می باشد. برای تغییر این امر از تابع فوق با سه آرگومان ذکر شده استفاده می شود . پس از فراخوانی این تابع دستورات بعدی روی ماتریس تنظیم شده جاری اثر خواهند گذاشت .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glLoadIdentity(void);	Procedure glLoadIdentity; stdcall; external 'OPENGL32.DLL';

توضیح :

ماتریس قابل تغییر جاری را به ماتریس واحد تبدیل می کند (Reset) .

ماتریس واحد :	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void glEnable(GLenum cap);
void glDisable(GLenum cap);
GLboolean glIsEnabled(GLenum cap);
	Procedure glEnable(cap: GLenum);
	stdcall; external 'OPENGL32.DLL';
	Procedure glDisable(cap: GLenum);
	stdcall; external 'OPENGL32.DLL';
	tyep GLboolean = Boolean; {
	unsigned char}
	Function glIsEnabled(cap: GLenum):
	GLboolean;
	stdcall; external 'OPENGL32.DLL';

توضیح :

برای فعال نمودن قابلیتی ، از glEnable وبرای غیرفعال نمودن آن ازglDisable استفاده می شود . بیش از ۴۰ نوع مقدار ثابت تعریف شده وجود دارد که بعنوان آرگومان می توان در این توابع مورد استفاده قرار داد . برای بررسی اینکه آیا حالتی هم اکنون فعال است یا خیر می توان از تابع glIsEnabled استفاده کرد . خروجی آن GL_TRUE و یا GL_FALSE می باشد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void glViewport(GLint x, GLint y,
GLsizei width,GLsizei height);
	type INT = Integer;
	type GLsizei = INT;
	Procedure glViewport(x: GLint;
	y: GLint; width: GLsizei; height: GLsizei);
	stdcall; external 'OPENGL32.DLL';

توضیح :

برای تنظیم دوباره درگاه دید جاری بکار می رود (انتخاب ناحیه دید) . بصورت پیش فرض OpenGL درگاه دید (ViewPort) را به کل مستطیل پنجره ای که برنامه با آن آغاز شده است ، تنظیم می کند . حال اگر اندازه پنجره تغییر کند ، برای تغییر و تطابق اندازه اشیای رسم شده با اندازه پنجره جاری می توان از تابع فوق استفاده کرد . x و Y گوشه بالا سمت چپ درگاه دید را مشخص می کنند و Width و Height ، اندازه مستطیل درگاه دید را . این تابع معمولا در روال رخداد OnResize پنجره جاری جای می گیرد .

واژه جدید : ViewPort

درگاه دید ، قسمتی از پنجره را مشخص می سازد که تمام ترسیمات در آن ناحیه اتفاق می افتد و قسمت هایی که خارج از این ناحیه قرار می گیرند ، خود بخود حذف می شوند .

اگر بعضی از مطالب فوق را متوجه نمی شوید زیاد نگران نباشید ؛ زیرا تمام توابع فوق در طی فصول آتی با ذکر مثالهای بیشتری ، توضیح داده خواهند شد .