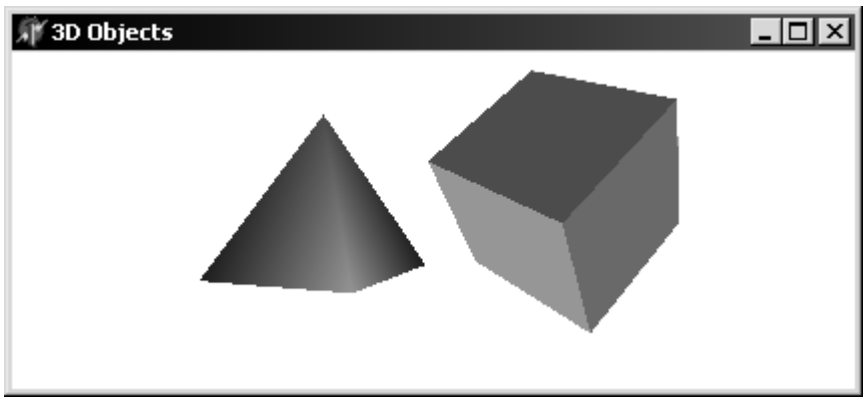
فصل ششم

اشیاء سه بعدی



مقدمه :

در این فصل قصد داریم بجای ترسیم اشیاء دو بعدی در دنیای سه بعدی ، با اضافه کردن بعد به آنها ، اشیاء سه بعدی واقعی خلق کنیم . اگر در فصل قبل از اشیاء دو بعدی ، اشیاء سه بعدی خلق کرده اید ، شاید بپرسید که چرا اشیاء تولید شده حول محور خود نمی چرخند ؟ برای چرخاندن شیءایی حول یک محور باید مرکزش بر (0,0,0) منطبق باشد و این کار باید توسط دستورات انتقال و تنظیم مجدد صورت گیرد .

ترسیم اشیاء سه بعدی :

نکته ای را که در ترسیم اشیاء سه بعدی باید بخاطر داشت این است که اگر برای مثال می خواهید یک هرم را رسم کنید ، پس از رسم یک وجه آن ، حتما glBegin را با glEnd خاتمه دهید

و نقطه ی دیگری را این بین ، اضافه نکنید ؛ زیرا در اینصورت OpenGL تصور خواهد کرد که نقطه ی چهارم آغاز مثلثی دیگر در راستای این وجه می باشد .

اگر هرم فقط حول محور y می چرخد ، نیازی به رسم قاعده آن نیست ، زیرا ما آنرا نمی بینیم .

نکته ی دیگر این است که رئوس مثلث ها و چهار ضلعی ها در جهت مخالف حرکت عقربه های ساعت رسم می شوند و ترتیب ارائه آنها نیز باید اینچنین باشد .

مروری بر مطالب :

در قسمت جاری مروری خواهیم داشت بر آنچه که گذشت . در OpenGL تنها چیزی که برای ایجاد یک دید سه بعدی لازم است ، برپایی مکعب مستطیلی است که عملیات رندر کردن در آن انجام می شود (Clipping Volume). این کار توسط تابع glFrustum و یا با مشابه قدرتمندتر آن gluPerspective انجام می شود . برای تغییر اندازه کل ناحیه ترسیمی از دستور glViewPort کمک گرفته می شود .

سپس برای ایجاد تصویری متحرک به عملیات ماتریسی برای دوران و یا انتقال اشیاء نیاز می باشد . خوشبختانه OpenGL توابعی مانند glRotate و glTranslate را ارائه داده است که اینکار را به سادگی انجام می دهند . فقط مطلبی را که باید بخاطر داشت این است که این توابع روی کل صحنه اثر می گذارند و برای تنظیم مجدد صحنه و بازگشت به (0,0,0) باید از تابع glLoadIdentity استفاده کرد .

برای کوچک و یا بزرگ کردن کل صحنه نیز می توان از تابع glScale کمک گرفت . برای مثال اگر فاکتور مقیاس ۲ باشد کل صحنه دو مرتبه بزرگتر می شود و یا می توان هر محور را به صورت جداگانه تحت تاثیر این دستور قرار داد تا جلوه های له شدگی ، کشیده شدن و یا انعکاس پدید آیند .

مرحله بعد آزمایش عمق می باشد . برای بدست آوردن صحنه ای واقع گرایانه معمولا از مفهوم z-Buffer استفاده می شود . مسؤولیت آن بررسی این موضوع است : آیا نقطه ای باید در عمق ترسیم شود و یا خیر ؟ برای مثال اگر حیوانی در جلوی خانه ای قرار گرفته است ، کدام نقاط باید ترسیم شوند و کدامیک نباید . برای فعال سازی این خاصیت از تابع glEnable و آرگومان خاص آن که در فصول قبلی توضیح داده شد ، استفاده می شود . سپس z-Buffer در انتهای هر فریم باید با دستور glClear پاک شود . پس بطور خلاصه از عمق بافر و آزمایش مربوطه برای مخفی کردن سطوحی که نباید ترسیم شوند استفاده می گردد .

پویانمایی (انیمیشن) تنها هنگامی با بهترین کارآیی قابل اعمال است که از تکنیک بافر دوگانه استفاده شود . در این حال ، در یک بافر عملیات ترسیم صورت می گیرد و در بافر دیگر با تعویض مکان ایندو ، اشیاء ترسیم شده نمایش داده می شوند . برای تعویض بافرها از دستور SwapBuffers به همراه تمهیدات مناسب در برپایی فرمت نقطه ای که شرح آن رفت ، کمک گرفته می شود .

تعدادی از توابع جدید مطرح شده در این قسمت با توضیحات بیشتر در ذیل ارائه خواهند شد :

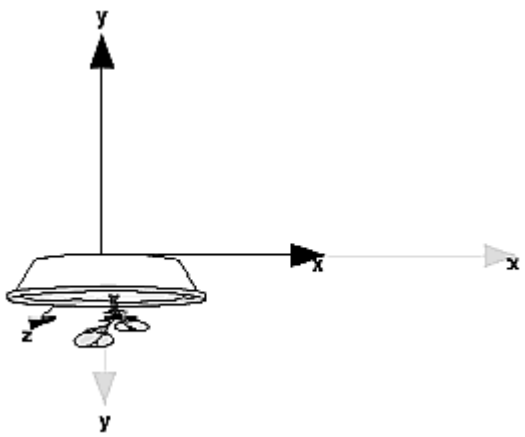
تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glScalef(GLfloat x, GLfloat y, GLfloat z);	Procedure glScalef(x: GLfloat; y: GLfloat; z: GLfloat); stdcall; external 'OPENGL32.DLL';

توضیح :

glScalef ماتریس جاری را در ماتریس مقیاس ضرب می کند . اینکار روی تمام نقاط اشیاء مطابق با آرگومانهای آن صورت می گیرد . ماتریس نهایی به صورت زیر است :

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

در شکل زیر تاثیر دستور glScalef(2.0,-0.5,1.0); را روی شکل ملاحظه می فرمائید :



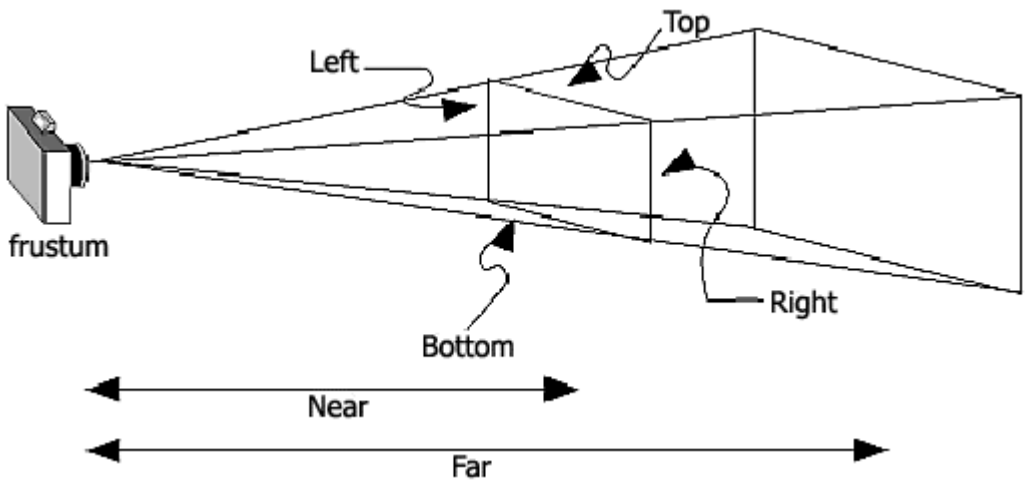
استفاده از آرگومانهایی با مقدار بیش از یک ، سبب کشیدگی شی ء و مقادیر کمتر از یک سبب انقباض آن می شوند.استفاده از آرگومان 1.0- سبب انعکاس شیء حول محور مربوطه می گردد .

هر چند می توان از آرگومانهایی با مقادیر صفر هم استفاده کرد اما اگر عملیات دیگری مانند نور پردازی نیز باید صورت گیرد ، انجام این کار سبب خواهد شد که ماتریس جاری ، دیگر معکوس نداشته باشد و این مساله ساز خواهد شد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble znear, GLdouble zfar);	Procedure glFrustum(left: GLdouble; right: GLdouble; bottom: GLdouble; top: GLdouble; zNear: GLdouble; zFar: GLdouble); stdcall; external 'OPENGL32.DLL';

توضیح :

glFrustum ماتریسی را برای ایجاد دید پرسپکتیو تولید کرده و درماتریس جاری ضرب می کند . حجم دید ، توسط پارامترهای (left,bottom,-near) و (right,top,-near) تعریف می شود که نقاط یاد شده ، نقاط سمت چپ ، پایین و سمت راست ، بالای صفحه برش می باشند . near و far همانگونه که در شکل زیر مشخص شده اند ، فواصل نقطه دید از صفحات برش نزدیک و دور هستند و همواره باید مثبت باشند .



این دید لزوما متقارن نبوده و نیز در امتداد محور z نیز می تواند نباشد. تابع gluPerspective نیز مشابه این تابع بوده و در فصول قبلی در مورد آن بحث گردید . ماتریس تولید شده توسط تابع glFrustum بصورت زیر است :

$$\begin{pmatrix} \frac{2\,near}{right-left} & 0 & A & 0 \\ 0 & \frac{2\,near}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$A = \frac{right+left}{right-left}$$

$$B = \frac{top+bottom}{top-bottom}$$

$$C = -\frac{far+near}{far-near}$$

$$D = -\frac{2\,far\,near}{far-near}$$

تذکر :

برای اجرای برنامه زیر لازم است که یک کنترل Timer را روی فرم برنامه خـود قـرار دهیـد و در زیر روال رخداد مربوط به آن کد ارائه شده را بنویسید .

برنامه فصل ... :

```
unit Ch06;

interface

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

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    procedure FormResize(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  rtri: GLfloat;      // Angle For The Triangle
  rquad: GLfloat;     // Angle For The Quad
  f_Hdc : LongInt;

 implementation

{$R *.DFM}

procedure InitGL;          // All Setup For OpenGL Goes Here
```

```
begin
  glShadeModel(GL_SMOOTH);          // Enables Smooth Color Shading
  glClearColor(1.0, 1.0, 1.0, 0.5);    // not Black Background!
  glClearDepth(1.0);                  // Depth Buffer Setup
  glEnable(GL_DEPTH_TEST);           // Enables Depth Testing
  glDepthFunc(GL_LESS);                        // The Type Of Depth Test To Do
  //Realy Nice perspective calculations
  glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST);
end;

procedure DrawGLScene;    // Here's Where We Do All The Drawing

begin
  // Clear The Screen And The Depth Buffer
  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();                    // Reset The View
  glTranslatef(-1.5,0.0,-6.0);
  glRotatef(rtri,0.0,1.0,0.0);       // Rotate The Triangle On The Y axis
  glBegin(GL_POLYGON);
    glColor3f(1.0,0.0,0.0);          // Red
    glVertex3f( 0.0, 1.0, 0.0);       // Top Of Triangle (Front)
    glColor3f(0.0,1.0,0.0);          // Green
    glVertex3f(-1.0,-1.0, 1.0);       // Left Of Triangle (Front)
    glColor3f(0.0,0.0,1.0);          // Blue
    glVertex3f( 1.0,-1.0, 1.0);       // Right Of Triangle (Front)

    glColor3f(1.0,0.0,0.0);          // Red
    glVertex3f( 0.0, 1.0, 0.0);       // Top Of Triangle (Right)
    glColor3f(0.0,0.0,1.0);          // Blue
    glVertex3f( 1.0,-1.0, 1.0);       // Left Of Triangle (Right)
    glColor3f(0.0,1.0,0.0);          // Green
    glVertex3f( 1.0,-1.0, -1.0);      // Right Of Triangle (Right)

    glColor3f(1.0,0.0,0.0);          // Red
    glVertex3f( 0.0, 1.0, 0.0);       // Top Of Triangle (Back)
    glColor3f(0.0,1.0,0.0);          // Green
    glVertex3f( 1.0,-1.0, -1.0);      // Left Of Triangle (Back)
    glColor3f(0.0,0.0,1.0);          // Blue
    glVertex3f(-1.0,-1.0, -1.0);      // Right Of Triangle (Back)

    glColor3f(1.0,0.0,0.0);          // Red
    glVertex3f( 0.0, 1.0, 0.0);       // Top Of Triangle (Left)
    glColor3f(0.0,0.0,1.0);          // Blue
    glVertex3f(-1.0,-1.0,-1.0);       // Left Of Triangle (Left)
    glColor3f(0.0,1.0,0.0);          // Green
    glVertex3f(-1.0,-1.0, 1.0);       // Right Of Triangle (Left)
  GlEnd();

  glLoadIdentity();
  glTranslatef(1.5,0.0,-6.0);
  glRotatef(rquad,1.0,1.0,1.0);      // Rotate The Quad On The X axis
//  glScalef(2.0,0.4,0.5);
  glBegin(GL_QUADS);                 // Start Drawing The Cube
    glColor3f(0.0,1.0,0.0);          // Set The Color To Blue
    glVertex3f( 1.0, 1.0,-1.0);       // Top Right Of The Quad (Top)
    glVertex3f(-1.0, 1.0,-1.0);       // Top Left Of The Quad (Top)
    glVertex3f(-1.0, 1.0, 1.0);       // Bottom Left Of The Quad (Top)
    glVertex3f( 1.0, 1.0, 1.0);       // Bottom Right Of The Quad (Top)
```

```
    glColor3f(1.0,0.5,0.0);          // Set The Color To Orange
    glVertex3f( 1.0,-1.0, 1.0);      // Top Right Of The Quad (Bottom)
    glVertex3f(-1.0,-1.0, 1.0);      // Top Left Of The Quad (Bottom)
    glVertex3f(-1.0,-1.0,-1.0);      // Bottom Left Of The Quad (Bottom)
    glVertex3f( 1.0,-1.0,-1.0);      // Bottom Right Of The Quad (Bottom)

    glColor3f(1.0,0.0,0.0);          // Set The Color To Red
    glVertex3f( 1.0, 1.0, 1.0);      // Top Right Of The Quad (Front)
    glVertex3f(-1.0, 1.0, 1.0);      // Top Left Of The Quad (Front)
    glVertex3f(-1.0,-1.0, 1.0);      // Bottom Left Of The Quad (Front)
    glVertex3f( 1.0,-1.0, 1.0);      // Bottom Right Of The Quad (Front)

    glColor3f(1.0,1.0,0.0);          // Set The Color To Yellow
    glVertex3f( 1.0,-1.0,-1.0);      // Top Right Of The Quad (Back)
    glVertex3f(-1.0,-1.0,-1.0);      // Top Left Of The Quad (Back)
    glVertex3f(-1.0, 1.0,-1.0);      // Bottom Left Of The Quad (Back)
    glVertex3f( 1.0, 1.0,-1.0);      // Bottom Right Of The Quad (Back)

    glColor3f(0.0,0.0,1.0);          // Set The Color To Blue
    glVertex3f(-1.0, 1.0, 1.0);      // Top Right Of The Quad (Left)
    glVertex3f(-1.0, 1.0,-1.0);      // Top Left Of The Quad (Left)
    glVertex3f(-1.0,-1.0,-1.0);      // Bottom Left Of The Quad (Left)
    glVertex3f(-1.0,-1.0, 1.0);      // Bottom Right Of The Quad (Left)

    glColor3f(1.0,0.0,1.0);          // Set The Color To Violet
    glVertex3f( 1.0, 1.0,-1.0);      // Top Right Of The Quad (Right)
    glVertex3f( 1.0, 1.0, 1.0);      // Top Left Of The Quad (Right)
    glVertex3f( 1.0,-1.0, 1.0);      // Bottom Left Of The Quad (Right)
    glVertex3f( 1.0,-1.0,-1.0);      // Bottom Right Of The Quad (Right)
  glEnd();                         // Done Drawing The Quad

  SwapBuffers(f_Hdc);
end;

procedure TForm1.FormResize(Sender: TObject);
var
  fWidth, fHeight: GLfloat;
begin
  wglMakeCurrent(f_Hdc,hrc); //activate the RC
// Prevent A Divide By Zero If The Window Is Too Small
// By Making The Height One
  if (Height=0) then
     Height:=1;
  // Reset The Current Viewport And Perspective Transformation
  glViewport(0, 0, Width, Height);
  glMatrixMode(GL_PROJECTION);       // Select The Projection Matrix
  glLoadIdentity();                  // Reset The Projection Matrix
  fWidth := width;
  fHeight := height;
  // Calculate The Aspect Ratio Of The Window
  gluPerspective(45.0,fWidth/fHeight,0.1,100.0);
  glMatrixMode(GL_MODELVIEW);        // Select The Modelview Matrix
  glLoadIdentity ;                   //Reset The Modelview Matrix
  InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
end;

procedure TForm1.Timer1Timer(Sender: TObject);
```

```
begin
  rtri := rtri + 5.2;     // Increase The Rotation Variable For The Triangle
  rquad := rquad - 5.15; // Decrease The Rotation Variable For The Quad

  Application.ProcessMessages;
  InvalidateRect(Handle, nil, False);//   DrawGLScene();
end;

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

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

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

end.
```