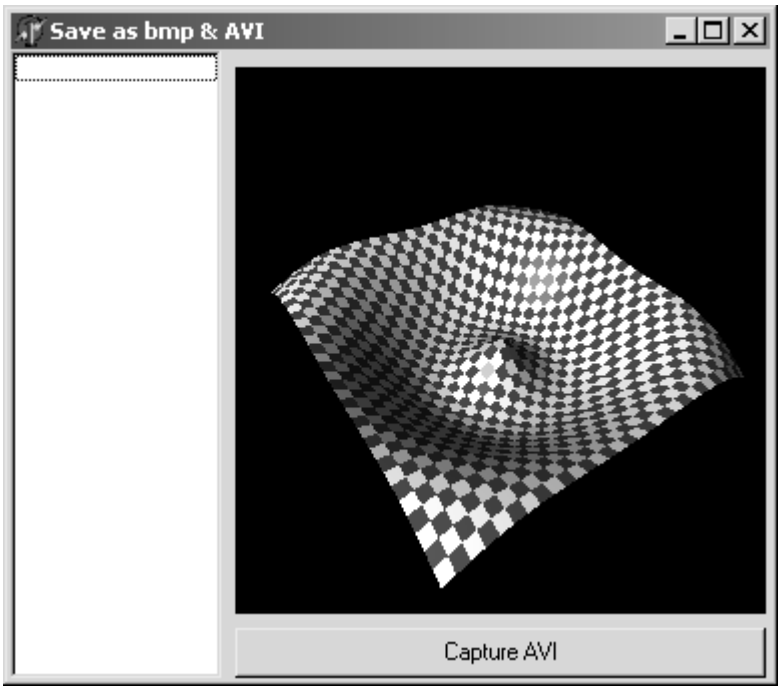
فصل بیست و چهارم

ذخیره کردن تصاویر در فایل و تبدیل آنها به AVI



مقدمه :

در این فصل نحوه ذخیره کردن تصاویر رندر شده با استفاده از OpenGL بیان می شود . برای انجام این کار روش های متعددی وجود دارد مانند : استفاده از کلیدهای Alt+PrintScreen و BitBlt نمودن hDC مکانی که تصویر در آنجا رندر شده به یک ImageBox و یا ابزاری مشابه و سپس ذخیره نمودن آن و غیره .

دو روش مطمئن برای ذخیره سازی تصاویر OpenGL :

الف) خواندن نقاط از بافر و تبدیل آنها به بیت مپ :

تابع glReadPixels محتویات بافر را به یک آرایه تعریف شده منتقل می کند . برای اینکار شما باید نحوه قرارگیری چهاربایتی را برای بیت مپ ویندوز تعریف نمایید و سپس مبدا و اندازه نقاطی که باید خوانده شوند را نیز معین کنید . پس از تعویض بایت های R و B بیت مپ ویندوز خلق می شود .

ب) رندر کردن به یک DC در حافظه و سپس ذخیره سازی آن بصورت بیت مپ .

تبدیل تصاویر ذخیره شده به یک فایل ویدئویی استاندارد ویندوز (AVI) :

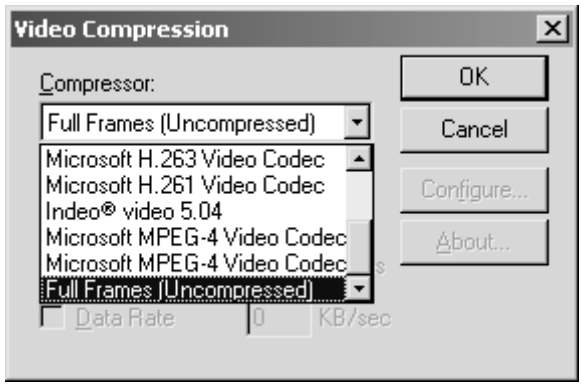
مطلب دیگری که در این فصل مورد بررسی قرار می گیرد ، ساخت فایل های متحرک AVI از تصاویر Bmp می باشد . آیا تابحال با استفاده از 3Dmax صحنه ای را رندر و به صورت AVI ذخیره کرده اید ؟ در صورت مثبت بودن پاسخ ، احتمالا به تلخی آنرا بیان نموده اید ! با استفاده از کد ارائه شده در این برنامه با سرعت باور نکردنی می توانید فایل های AVI را از تصاویر Bmp برنامه تولید کنید .

نحوه ی تبدیل فایل های Bmp به AVI :

یکی از کتابخانه های ویندوز ، فایل AVIFIL32.dll می باشد . اطلاعات کامل در مورد توابع و ثوابت این کتابخانه که برای تولید فایل های AVI بکار گرفته می شود در فایل vfw.h که جزئی از VC++ می باشد ، توسط مایکروسافت ارائه شده است . در این فصل قسمتی از این HeaderFile که به زبان C نوشته شده ، به زبان دلفی ترجمه گردیده است .

توسط فراخوانی تابع AVIFileInit به ویندوز می گوئیم که کتابخانه AVIFILE را برای بکار گیری آن آغاز نماید و بدیهی است که هنگام اتمام کار با این کتابخانه ، تابع AVIFileExit باید فراخوانی شود . برای باز کردن یک فایل AVI و بدست آوردن handle آن که در توابع دیگر این کتابخانه بکار می رود ، تابع AVIFileOpen باید بکار گرفته شود . پس از هر فراخوانی باید از انجام موفقیت آمیز آن اطمینان حاصل شود . در اینجا فقط کافی است که خروجی تابع با ثابت AVIERR_OK

مقایسه شود . در هنگام بروز خطا تابع AVIFileRelease سبب رهایش handle و بسته شدن فایل می شود . می توان صفحه دیالوگ انتخاب گزینه های فشرده سازی را برای تولید AVI توسط تابع AVISaveOptions نمایش داد (شکل زیر) . سپس توسط تابع AVIMakeCompressedStream از گزینه انتخاب شده توسط کاربر استفاده می شود .



در ادامه لازم است اطلاعات اولین بیت مپ را برای تنظیم فرمت اخذ نمائیم . رکورد حاصل به تابع AVIStreamSetFormat فرستاده می شود و سپس اطلاعات تک تک فایل ها توسط AVIStreamWrite ، درون فایل AVI نوشته می شود .

مروری بر توابع :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
const GLubyte *glGetString(GLenum name);	Function glGetString(name: GLenum): PGLubyte; stdcall; external 'OPENGL32.DLL';

توضیح :

رشته ای را بر می گرداند که حاوی اطلاعات مربوط به OpenGL نصب شده بر روی سیستم می باشد .

پارامتر name می تواند یکی از چهار ثابت زیر باشد :

GL_VENDOR : نام شرکتی را که سازنده کتابخانه OpenGL نصب شده بر روی سیستم شما است را بر می گرداند .

GL_RENDERER : نام رندر کننده را بر می گرداند . این نام عموما به تنظیمات سخت افزاری وابسته می باشد .

GL_VERSION : شماره نگارش کتابخانه OpenGL را بر می گرداند .

GL_EXTENSIONS : لیست امکانات سخت افزاری فراهم برای OpenGL را (وابسته به کارت گرافیکی شما) ، بر می گرداند .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
STDAPI_(VOID) AVIFileInit(VOID);	procedure AVIFileInit;

توضیح :

کتابخانه AviFile ویندوز را آغاز می نماید . این کار باید قبل از فراخوانی هرگونه تابعی از این کتابخانه ، صورت گیرد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
STDAPI_(VOID) AVIFileExit(VOID);	procedure AVIFileExit;

توضیح :

سبب خروج از کتابخانه AviFile می شود .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
STDAPI AVIFileOpen(PAVIFILE * ppfile, LPCTSTR szFile, UINT mode, CLSID pclsidHandler);	Function AVIFileOpen(var ppfile: PAVIFILE; szFile: LPCSTR; uMode: UINT; lpHandler: PCLSID): HResult;

توضیح :

یک فایل Avi را باز نموده و آدرس فایل واسط را که برای دسترسی به آن بکار می رود بر می گرداند .

ppfile : اشاره گری است به یک بافر که اشاره گر فایل جدید واسط را دریافت می کند .

szFile : رشته ای است مختوم به nil حاوی نام فایلی که باز می شود .

mode : حالت دسترسی به فایل هنگام گشودن آن می باشد . حالت پیش فرض OF_READ است و ثوابت زیر برای آن مجاز هستند :

OF_SHARE_DENY_NONE	OF_CREATE
OF_SHARE_DENY_READ	OF_SHARE_DENY_WRITE
OF_READ	OF_SHARE_EXCLUSIVE
OF_WRITE	OF_READWRITE

pclsidHandler : اشاره گری به یک کلاس تعیین کننده هویت یا handle سفارشی یا ویژه که شما مایل به استفاده از آن هستید . اگر مقدار آن nil باشد ، سیستم handler را از Registry انتخاب خواهد نمود .

تابع به فرمت زبان دلفی	تابع به فرمت زبان C
Function AVIFileCreateStream(pfile: PAVIFile; var ppavi: PAVIStream; var psi: TAVIStreamInfoA): HResult;	STDAPI AVIFileCreateStream(PAVIFILE pfile, PAVISTREAM * ppavi, AVISTREAMINFO * psi);

توضیح :

یک جریان (Stream) را در فایل موجود و واسطی را به آن خلق می نماید .

pfile : handle فایل Avi باز شده می باشد .

ppavi : اشاره گری به واسط جریان جدید .

psi : اشاره گری به ساختار (رکورد) حاوی اطلاعات جریان جدید که شامل نوع جریان و نرخ نمونه برداری آن است ، می باشد .

این تابع درصورت موفقیت صفر بر می گرداند .

تابع به فرمت زبان دلفی	تابع به فرمت زبان C
Function AVIStreamSetFormat(pavi: PAVIStream; lPos: LONG; lpFormat: PVOID; cbFormat: LONG): HResult;	STDAPI AVIStreamSetFormat(PAVISTREAM pavi, LONG lPos, LPVOID lpFormat, LONG cbFormat);

توضیح :

فرمت جریان را در موقعیتی تعیین شده تنظیم می نماید .

pavi : handle یک جریان باز می باشد .

lPos : موقعیتی در جریان که باید فرمت را دریافت نماید .

lpFormat : اشاره گری به ساختار (رکورد) حاوی اطلاعات فرمت جدید .

cbFormat : اندازه بلوک حافظه (به بایت) ارجاع شده به lpFormat .

handler (راه انداز) مخصوص نوشتن فایل های Avi تغییرات فرمت را نمی پذیرد . قبل از تنظیم فرمت اولیه جریان ، تنها تغییرات در پالت جریان ویدئویی مجاز می باشد . تغییرات پالت باید بعد از هر فریم که در فایل Avi نوشته می شود ، رخ دهد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
STDAPI AVIStreamWrite(PAVISTREAM pavi, LONG lStart, LONG lSamples, LPVOID lpBuffer, LONG cbBuffer, DWORD dwFlags, LONG * plSampWritten, LONG * plBytesWritten);	Function AVIStreamWrite(pavi: PAVIStream; lStart, lSamples: LONG; lpBuffer: PVOID; cbBuffer: LONG; dwFlags: DWORD; var plSampWritten: LONG; var plBytesWritten: LONG): HResult;

توضیح :

داده ها را در جریان می نویسد .

pavi : handle یک جریان باز می باشد .

lStart : اولین نمونه برای نوشتن .

lSamples : تعداد نمونه ها برای نوشته شدن .

lpBuffer : اشاره گری به بافر داده های مخصوص نوشتن .

cbBuffer : اندازه بافر ارجاعی به lpBuffer .

dwFlags : پرچم وابسته به داده . تنها پرچم AVIIF_KEYFRAME تعریف شده است . handle پیش فرض فایل Avi تنها نوشتن در انتهای جریان را پشتیبانی می کند .

plSampWritten : اشاره گری است به بافری که تعداد نمونه هایی که باید نوشته شوند را دریافت می کند . این آرگومان را به nil نیز می توان تنظیم کرد .

plBytesWritten : اشاره گری است به بافری که تعداد بایت هایی که باید نوشته شود را دریافت می کند . این آرگومان را به nil نیز می توان تنظیم کرد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
STDAPI_(LONG) AVIStreamRelease(PAVISTREAM pavi);	function AVIStreamRelease(pavi: PAVIStream): ULONG;

توضیح :

handle یک جریان واسط Avi را می بندد .

pavi : handle یک جریان باز می باشد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
STDAPI_(ULONG) AVIFileRelease(PAVIFILE pfile);	function AVIFileRelease(pfile: PAVIFile): ULONG;

توضیح :

handle مربوط به فایل Avi باز شده را می بندد .

pfile : handle یک جریان باز می باشد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
BOOL AVISaveOptions(HWND hwnd, UINT uiFlags, int nStreams, PAVISTREAM * ppavi, LPAVICOMPRESSOPTIONS * plpOptions);	function AVISaveOptions(hwnd : HWND; uiFlags : UINT; nStreams : int; var ppavi : PAVISTREAM; var plpOptions : PAVICOMPRESSOPTIONS): BOOL;

توضیح :

این تابع یک صفحه دیالوگ استاندارد را برای انتخاب گزینه های استاندارد فشرده سازی با استفاده از hWnd پنجره موجود ، نمایش می دهد . پس از انتخاب گزینه مناسب توسط کاربر ، خروجی توسط رکورد **AVICOMPRESSOPTIONS** با استفاده از plpOptions به برنامه بازگشت داده می شود .

hwnd : handle پنجره پدر می باشد .

uiFlags : پرچم هایی برای نمایش گزینه های فشرده سازی در صفحه استاندارد . ثوابت زیر برای آن تعریف شده اند :

ICMF_CHOOSE_KEYFRAME

ICMF_CHOOSE_DATARATE
ICMF_CHOOSE_PREVIEW

بهترین راه برای دریافت معانی ثوابت فوق اجرای آنها می باشد !

nStreams : اشاره گری به اشاره گرهای واسط جریان . nStreams تعیین کننده تعداد اشاره گر ها در آرایه است .

ppavi : اشاره گری به آرایه ای از اشاره گرهای واسط جریان .

plpOptions : اشاره گری به آرایه ای از اشاره گرها به ساختار **AVICOMPRESSOPTIONS** . این ساختار اطلاعات انتخاب شده توسط کاربر را در خود ذخیره می کند .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
LONG AVISaveOptionsFree(int nStreams, LPAVICOMPRESSOPTIONS *plpOptions);	function AVISaveOptionsFree(nStreams: int; var plpOptions: PAVICOMPRESSOPTIONS): HResult;

توضیح :

منابع تخصیص یافته بوسیله تابع AVISaveOptions را آزاد می نماید .

nStreams : کنت ساختار **AVICOMPRESSOPTIONS** ارجاع شده توسط plpOptions است .

plpOptions : اشاره گری به آرایه ای از اشاره گرها به ساختار **AVICOMPRESSOPTIONS** .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
STDAPI AVIMakeCompressedStream(PAVISTREAM * ppsCompressed, PAVISTREAM psSource, AVICOMPRESSOPTIONS * lpOptions, CLSID * pclsidHandler);	function AVIMakeCompressedStream(var ppsCompressed : PAVISTREAM; ppsSource : PAVISTREAM; lpOptions : PAVICOMPRESSOPTIONS; pclsidHandler : PCLSID): HResult;

توضیح :

این تابع یک جریان فشرده شده را از یک جریان فشرده نشده خلق می کند و آدرس اشاره گر به جریان را بر می گرداند .

ppsCompressed : اشاره گری است به بافری که اشاره گر جریان فشرده شده را دریافت می کند .

psSource : اشاره گری است به جریانی که باید فشرده شود .

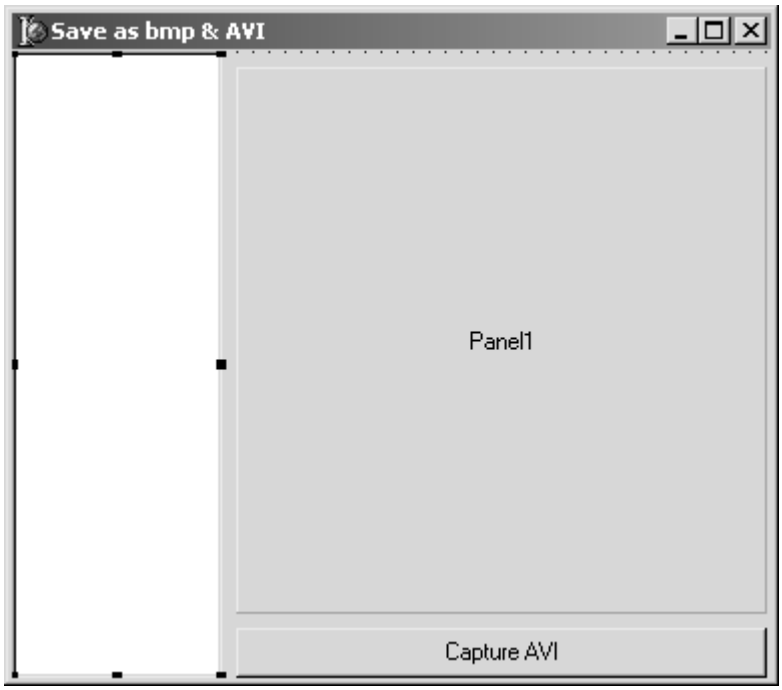
lpOptions : اشاره گری است به ساختاری که نوع فشرده سازی را بیان می کند .

pclsidHandler : اشاره گری به کلاس تعیین هویت بکار گرفته شده برای خلق جریان .

برای مشاهده سایر اطلاعات مربوط به این توابع و نحوه استفاده از آنها بـه اصـل برنامـه ایـن فصل مراجعه فرمایید .

برنامه فصل :

از کد برنامه این فصل بعنوان قالبی برای رسم انواع و اقسام رویه های سه بعدی نیز می تـوان استفاده کرد .



برای اجرای برنامه این فصل ابتدا یک Panel ، یک دکمـه و یـک ListBox مطـابق شـکل فـوق روی صفحه قرار دهید و نام ListBox را به BitmapListBox تغییر دهید .

```
unit ch24;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms,Dialogs , StdCtrls , ExtCtrls ,OpenGL ,SPF;
type
  TForm1 = class(TForm)
    BitmapListBox: TListBox;
    Panel1: TPanel;
    Button1: TButton;
```

```
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
     procedure Idle(Sender: TObject; var Done: Boolean);
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

{This allows us to subclass IUnknown without having to
 include the now defunct ole2.pas.}

type
{ Result code }
  HResult = Longint;
{ Globally unique ID }
  PGUID = ^TGUID;
  TGUID = record
    D1: Longint;
    D2: Word;
    D3: Word;
    D4: array[0..7] of Byte;
  end;

{ Interface ID }
  PIID = PGUID;
  TIID = TGUID;

{ Class ID }
  PCLSID = PGUID;
  TCLSID = TGUID;

{ IUnknown interface }
  IUnknown = class
  public
    function QueryInterface(const iid: TIID; var obj): HResult;
             virtual; stdcall; abstract;
    function AddRef: Longint; virtual; stdcall; abstract;
    function Release: Longint; virtual; stdcall; abstract;
  end;


{ don't know who wrote this -
  the AVI section for avifil32.dll - Thanks !!!}
 type

{ TAVIFileInfoW record }

  LONG = Longint;
  PVOID = Pointer;
```

```
  TAVIFileInfoW = record
    dwMaxBytesPerSec,   // max. transfer rate
    dwFlags,            // the ever-present flags
    dwCaps,
    dwStreams,
    dwSuggestedBufferSize,

    dwWidth,
    dwHeight,

    dwScale,
    dwRate,       // dwRate / dwScale == samples/second
    dwLength,

    dwEditCount: DWORD;
    // descriptive string for file type?
    szFileType: array[0..63] of WideChar;
  end;
  PAVIFileInfoW = ^TAVIFileInfoW;

{ TAVIStreamInfoA record }

  TAVIStreamInfoA = record
    fccType,
    fccHandler,
    dwFlags,       // Contains AVITF_* flags
    dwCaps: DWORD;
    wPriority,
    wLanguage: WORD;
    dwScale,
    dwRate, // dwRate / dwScale == samples/second
    dwStart,
    dwLength, // In units above...
    dwInitialFrames,
    dwSuggestedBufferSize,
    dwQuality,
    dwSampleSize: DWORD;
    rcFrame: TRect;
    dwEditCount,
    dwFormatChangeCount,
    szName:  array[0..63] of AnsiChar;
  end;
  TAVIStreamInfo = TAVIStreamInfoA;

{ TAVIStreamInfoW record }

  TAVIStreamInfoW = record
    fccType,
    fccHandler,
    dwFlags,       // Contains AVITF_* flags
    dwCaps: DWORD;
    wPriority,
    wLanguage: WORD;
    dwScale,
    dwRate, // dwRate / dwScale == samples/second
    dwStart,
    dwLength, // In units above...
    dwInitialFrames,
```

```
    dwSuggestedBufferSize,
    dwQuality,
    dwSampleSize: DWORD;
    rcFrame: TRect;
    dwEditCount,
    dwFormatChangeCount,
    szName:  array[0..63] of WideChar;
  end;

{ IAVIStream interface }

  IAVIStream = class(IUnknown)
    function Create(lParam1, lParam2: LPARAM): HResult;
          virtual; stdcall; abstract;
    function Info(var psi: TAVIStreamInfoW; lSize: LONG): HResult;
          virtual; stdcall; abstract;
    function FindSample(lPos, lFlags: LONG): LONG;
          virtual; stdcall; abstract;
    function ReadFormat(lPos: LONG; lpFormat: PVOID;
          var lpcbFormat: LONG): HResult;
          virtual; stdcall; abstract;
    function SetFormat(lPos: LONG; lpFormat: PVOID;
          lpcbFormat: LONG): HResult; virtual; stdcall; abstract;
    function Read(lStart, lSamples: LONG; lpBuffer: PVOID;
          cbBuffer: LONG; var plBytes: LONG;
          var plSamples: LONG): HResult; virtual; stdcall; abstract;
    function Write(lStart, lSamples: LONG; lpBuffer: PVOID;
          cbBuffer: LONG; dwFlags: DWORD;
          var plSampWritten: LONG; var plBytesWritten: LONG): HResult;
          virtual; stdcall; abstract;
    function Delete(lStart, lSamples: LONG): HResult;
          virtual; stdcall; abstract;
    function ReadData(fcc: DWORD; lp: PVOID; var lpcb: LONG): HResult;
          virtual; stdcall; abstract;
    function WriteData(fcc: DWORD; lp: PVOID; cb:  LONG): HResult;
          virtual; stdcall; abstract;
    function SetInfo(var lpInfo: TAVIStreamInfoW;
          cbInfo: LONG): HResult; virtual; stdcall; abstract;
  end;
  PAVIStream = ^IAVIStream;

{ IAVIFile interface }

  IAVIFile = class(IUnknown)
    function Info(var pfi: TAVIFileInfoW; lSize: LONG): HResult;
          virtual; stdcall; abstract;
    function GetStream(var ppStream: PAVIStream;
          fccType: DWORD; lParam: LONG): HResult;
          virtual; stdcall; abstract;
    function CreateStream(var ppStream: PAVIStream;
          var pfi: TAVIFileInfoW): HResult;
          virtual; stdcall; abstract;
    function WriteData(ckid: DWORD; lpData: PVOID;
          cbData: LONG): HResult; virtual; stdcall; abstract;
    function ReadData(ckid: DWORD; lpData: PVOID;
          var lpcbData: LONG): HResult; virtual; stdcall; abstract;
    function EndRecord: HResult; virtual; stdcall; abstract;
    function DeleteStream(fccType: DWORD; lParam: LONG): HResult;
```

```
          virtual; stdcall; abstract;
  end;
  PAVIFile = ^IAVIFile;

procedure AVIFileInit; stdcall;
procedure AVIFileExit; stdcall;
function AVIFileOpen(var ppfile: PAVIFile; szFile: LPCSTR;
        uMode: UINT; lpHandler: PCLSID): HResult; stdcall;
function AVIFileCreateStream(pfile: PAVIFile; var ppavi: PAVISTREAM;
        var psi: TAVIStreamInfoA): HResult; stdcall;
function AVIStreamSetFormat(pavi: PAVIStream; lPos: LONG;
        lpFormat: PVOID; cbFormat: LONG): HResult; stdcall;
function AVIStreamWrite(pavi: PAVIStream; lStart, lSamples: LONG;
        lpBuffer: PVOID; cbBuffer: LONG; dwFlags: DWORD;
        var plSampWritten: LONG;
        var plBytesWritten: LONG): HResult; stdcall;
function AVIStreamRelease(pavi: PAVISTREAM): ULONG; stdcall;
function AVIFileRelease(pfile: PAVIFile): ULONG; stdcall;

const
  AVIERR_OK       = 0;

  AVIIF_LIST      = $01;
  AVIIF_TWOCC     = $02;
  AVIIF_KEYFRAME  = $10;

  streamtypeVIDEO = $73646976; // DWORD( 'v', 'i', 'd', 's' )

{ AVI interface IDs }

  IID_IAVIFile: TGUID = (
    D1:$00020020;D2:$0;D3:$0;D4:($C0,$0,$0,$0,$0,$0,$0,$46));
  IID_IAVIStream: TGUID = (
    D1:$00020021;D2:$0;D3:$0;D4:($C0,$0,$0,$0,$0,$0,$0,$46));
  IID_IAVIStreaming: TGUID = (
    D1:$00020022;D2:$0;D3:$0;D4:($C0,$0,$0,$0,$0,$0,$0,$46));
  IID_IGetFrame: TGUID = (
    D1:$00020023;D2:$0;D3:$0;D4:($C0,$0,$0,$0,$0,$0,$0,$46));
  IID_IAVIEditStream: TGUID = (
    D1:$00020024;D2:$0;D3:$0;D4:($C0,$0,$0,$0,$0,$0,$0,$46));

{ AVI class IDs }

  CLSID_AVISimpleUnMarshal: TGUID = (
    D1:$00020009;D2:$0;D3:$0;D4:($C0,$0,$0,$0,$0,$0,$0,$46));
  CLSID_AVIFile: TGUID = (
    D1:$00020000;D2:$0;D3:$0;D4:($C0,$0,$0,$0,$0,$0,$0,$46));


implementation

{$R *.dfm}


// MS-AVI file support library = avifil32.dll
procedure AVIFileInit; stdcall;
        external 'avifil32.dll' name 'AVIFileInit';
procedure AVIFileExit; stdcall;
```

```
        external 'avifil32.dll' name 'AVIFileExit';
function AVIFileOpen(var ppfile: PAVIFILE; szFile: LPCSTR;
        uMode: UINT; lpHandler: PCLSID): HResult;
        external 'avifil32.dll' name 'AVIFileOpenA';
function AVIFileCreateStream(pfile: PAVIFile; var ppavi: PAVIStream;
        var psi: TAVIStreamInfoA): HResult;
        external 'avifil32.dll' name 'AVIFileCreateStreamA';
function AVIStreamSetFormat(pavi: PAVIStream; lPos: LONG;
        lpFormat: PVOID; cbFormat: LONG): HResult;
        external 'avifil32.dll' name 'AVIStreamSetFormat';
function AVIStreamWrite(pavi: PAVIStream; lStart, lSamples: LONG;
        lpBuffer: PVOID; cbBuffer: LONG; dwFlags: DWORD;
        var plSampWritten: LONG; var plBytesWritten: LONG): HResult;
        external 'avifil32.dll' name 'AVIStreamWrite';
function AVIStreamRelease(pavi: PAVIStream): ULONG;
        external 'avifil32.dll' name 'AVIStreamRelease';
function AVIFileRelease(pfile: PAVIFile): ULONG;
        external 'avifil32.dll' name 'AVIFileRelease';

const
 M_PI=3.14159265358979323846;
 SPACE= 35.0;
 EDGELEN= 0.08;
 POINTSPERSIDE= 32;
 STARTCOORD= (POINTSPERSIDE-1)*EDGELEN/2;
 ANIMLEN = 100;

type
  Point3d = record
    x, y, z : GLfloat;
  end;

var
 p : array[0..POINTSPERSIDE-1] of
    array[0..POINTSPERSIDE-1] of Point3d;
 material1 : array[0..3] of GLfloat = (1, 0.0, 0.0, 1.0);
 material2 : array[0..3] of GLfloat = (1, 1, 1, 1.0);
 lightPosition : array[0..3] of GLfloat = (-3.0, 0.0, 1.5, 1.0);
 matSpecular : array[0..3] of GLfloat = (0.8, 0.8, 0.8, 1.0) ;
 matShininess : array[0..0] of GLfloat = (80.0);

 spinX : GLfloat= 0.0;
 spinY :GLfloat = -50.0;
 spinZ :GLfloat= 0.0;
 m_x :GLfloat= 0.1 ;
 m_y :GLfloat= 0.1;
 m_z :GLfloat= 0.7;
 dispList : integer = 1;
 material : boolean= false;
 m_anim : boolean= true;
 m_spin : boolean= true;

 f_Hdc : longInt;

procedure  SaveAsBmp(hWnd : THandle; hDC : HDC;
                fileName: string; H,W:Integer);
 VAR
   DtCanvas : TCanvas;
```

```
    Bitmap:  TBitmap;
    NumColors : integer;
    LogPal : PLogPalette;
    Src,Dst : TRect;
begin
  Screen.Cursor := crHourGlass;

  DtCanvas:=TCanvas.Create;
  Bitmap := TBitmap.Create;

  TRY
    DtCanvas.Handle := GetDC(hWnd);
    Bitmap.Width  := w-17;
    Bitmap.Height := h-30;
    NumColors := GetDeviceCaps(hDC ,SizePalette);
    GetMem(LogPal,SizeOf(TLogPalette)+
          (NumColors-1)*SizeOf(TPaletteEntry));
    LogPal.palVersion :=$386;
    LogPal.palNumEntries := NumColors;
    GetSystemPaletteEntries(hDC ,
              8, NumColors, LogPal.palPalEntry);
    Bitmap.Palette := CreatePalette(LogPal^);
    FreeMem(LogPal);
    Src:=Rect(0,0,w,h);
    Dst:=Src;
    Bitmap.Canvas.CopyRect(Dst,DtCanvas,Src);

    Bitmap.SaveToFile(fileName);

  FINALLY
    Screen.Cursor := crDefault;
    bitmap.Free;
    DtCanvas.Free;
  END
end;

procedure InitializeBitmapInfoHeader(Bitmap: HBITMAP;
          var BI: TBitmapInfoHeader; Colors: Integer);
var
  BM: Windows.TBitmap;
begin
  GetObject(Bitmap, SizeOf(BM), @BM);
  with BI do
  begin
    biSize := SizeOf(BI);
    biWidth := BM.bmWidth;
    biHeight := BM.bmHeight;
    if Colors <> 0 then
      case Colors of
        2: biBitCount := 1;
        16: biBitCount := 4;
        256: biBitCount := 8;
      end
    else biBitCount := BM.bmBitsPixel * BM.bmPlanes;
    biPlanes := 1;
    biXPelsPerMeter := 0;
    biYPelsPerMeter := 0;
    if biBitCount>8 then biClrUsed := 0 else biClrUsed := Colors;
```

```
    biClrImportant := 0;
    biCompression := BI_RGB;
    if biBitCount in [16, 32] then biBitCount := 24;
    biSizeImage :=
        (((biWidth * biBitCount) + 31) div 32) * 4 * biHeight;
  end;
end;

procedure InternalGetDIBSizes(Bitmap: HBITMAP;
         var InfoHeaderSize: Integer;
         var ImageSize: DWORD; Colors: Integer);
var
  BI: TBitmapInfoHeader;
begin
  InitializeBitmapInfoHeader(Bitmap, BI, Colors);
  with BI do
  begin
    case biBitCount of
      24: InfoHeaderSize := SizeOf(TBitmapInfoHeader);
    else
      InfoHeaderSize :=
        SizeOf(TBitmapInfoHeader) + SizeOf(TRGBQuad) *
       (1 shl biBitCount);
    end;
  end;
  ImageSize := BI.biSizeImage;
end;

function InternalGetDIB(Bitmap: HBITMAP; Palette: HPALETTE;
  var BitmapInfo; var Bits; Colors: Integer): Boolean;
var
  OldPal: HPALETTE;
  Focus: HWND;
  DC: HDC;
begin
  InitializeBitmapInfoHeader(Bitmap,
         TBitmapInfoHeader(BitmapInfo), Colors);
  OldPal := 0;
  Focus := GetFocus;
  DC := GetDC(Focus);
  try
    if Palette <> 0 then
    begin
      OldPal := SelectPalette(DC, Palette, False);
      RealizePalette(DC);
    end;
    Result := GetDIBits(DC, Bitmap, 0,
            TBitmapInfoHeader(BitmapInfo).biHeight, @Bits,
            TBitmapInfo(BitmapInfo), DIB_RGB_COLORS) <> 0;
  finally
    if OldPal <> 0 then SelectPalette(DC, OldPal, False);
    ReleaseDC(Focus, DC);
  end;
end;

procedure SaveAs_AVI(filename :string; BitmapListBox :TListBox;
                  frame_per_sec : integer);
var
```

```
  i: Integer;
  pfile: PAVIFile;
  asi: TAVIStreamInfo;
  ps: PAVIStream;
  nul: Longint;

  BitmapInfo: PBitmapInfoHeader;
  BitmapInfoSize: Integer;
  BitmapBits: Pointer;
  BitmapSize: DWORD;
begin

  with BitmapListBox do
    begin
      AVIFileInit;

      if AVIFileOpen(pfile, PChar(FileName),
            OF_WRITE or OF_CREATE, nil)=AVIERR_OK then
      begin
        FillChar(asi,sizeof(asi),0);

        asi.fccType := streamtypeVIDEO;  //  Now prepare the stream
        asi.fccHandler := 0;
        asi.dwScale := 1;
        asi.dwRate := frame_per_sec;

        with Items.Objects[0] as TBitmap do
        begin
          InternalGetDIBSizes(Handle,BitmapInfoSize,BitmapSize,256);
          BitmapInfo := AllocMem(BitmapInfoSize);
          BitmapBits := AllocMem(BitmapSize);
          InternalGetDIB(Handle,0,BitmapInfo^,BitmapBits^,256);
        end;

        asi.dwSuggestedBufferSize := BitmapInfo^.biSizeImage;
        asi.rcFrame.Right := BitmapInfo^.biWidth;
        asi.rcFrame.Bottom := BitmapInfo^.biHeight;

        if AVIFileCreateStream(pfile,ps,asi)=AVIERR_OK then
          with (Items.Objects[0] as TBitmap) do
          begin
            InternalGetDIB(Handle,0,BitmapInfo^,BitmapBits^,256);
            if AVIStreamSetFormat(ps,0,BitmapInfo,
                        BitmapInfoSize)=AVIERR_OK then
            begin
              for i:=0 to Items.Count-1 do
                with (Items.Objects[i] as TBitmap) do
                begin
                  InternalGetDIB(Handle,0,BitmapInfo^,BitmapBits^,256);
                  if AVIStreamWrite(ps,i,1,BitmapBits,BitmapSize,
                        AVIIF_KEYFRAME,nul,nul)<>AVIERR_OK then
                  begin
                    raise Exception.Create('Could not add frame');
                    break;
                  end;
                end;
            end;
          end;
```

```
        FreeMem(BitmapInfo);
        FreeMem(BitmapBits);
      end;

    AVIStreamRelease(ps);
    AVIFileRelease(pfile);

    AVIFileExit;
  end;
end;

procedure glInfo();
var
  vendor,renderer,version,extension : pchar;
begin
        vendor := pchar (glGetString(GL_VENDOR));
        renderer := pchar ( glGetString(GL_RENDERER));
        version := pchar (glGetString(GL_VERSION));
        extension := pchar (glGetString(GL_EXTENSIONS));
      MessageBox(0,pchar('vendor: '+vendor+#13#10+
              'renderer: '+renderer+#13#10+
              'version: '+version+#13#10+
              'extension: '+extension),'glInfo',
              MB_OK+MB_ICONINFORMATION);
end;

procedure normalize(var v : array of single);
var
 d:single;
begin
   d := sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);
  if (d <> 0.0) then // avoid division by zero */
  begin
     v[0] :=v[0]/ d;
     v[1] :=v[1] / d;
     v[2] :=v[2] / d;
  end;
end;

procedure normCrossProd( v1,  v2 : array of single ;
                  var  m_out : array of single);
begin
  m_out[0] := v1[1]*v2[2] - v1[2]*v2[1];
  m_out[1] := v1[2]*v2[0] - v1[0]*v2[2];
  m_out[2] := v1[0]*v2[1] - v1[1]*v2[0];
  normalize(m_out);
end;

procedure changeMaterial();
begin
  if (material) then
     glMaterialfv(GL_FRONT_AND_BACK,
              GL_AMBIENT_AND_DIFFUSE, @material1)
  else
     glMaterialfv(GL_FRONT_AND_BACK,
              GL_AMBIENT_AND_DIFFUSE, @material2);
  material := not(material);
end;
```

```
procedure resetMaterial();
begin
   material := false;
   changeMaterial();
   material := false;
end;

procedure display();
begin
   glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
   glPushMatrix();
     glRotatef(spiny, 1.0, 0.0, 0.0);
     glRotatef(spinx, 0.0, 1.0, 0.0);
     glRotatef(spinZ, 0.0, 0.0, 1.0);
     glCallList(dispList);
   glPopMatrix();
   glFinish();

   SwapBuffers(f_Hdc); // high end machines may need this */
end;

procedure my_idle();
begin
   if (m_spin) then
   begin
      spinX :=spinX + m_x;
      if (spinX >= 360.0) then spinX :=spinX - 360.0;
      spinY :=spinY + m_y;
      if (spinY >= 360.0) then spinY :=spinY - 360.0;
      spinZ :=spinZ + m_z;
      if (spinZ >= 360.0) then spinZ :=spinZ - 360.0;
   end;

   if (m_anim) then
   begin
      dispList:=dispList-1;
      if (dispList = 0)  then dispList := ANIMLEN;
   end;

end;

procedure initDisplayList( anim : integer );
var
   i, j, k :integer;
   x, y :GLfloat;
   v1 , v2 , normal : array[0..2] of GLfloat;
begin

   y:=STARTCOORD;
   x:=-STARTCOORD;

   for j := 0 to POINTSPERSIDE-1 do
   begin
     for k:=0 to POINTSPERSIDE-1 do
      begin
       p[k,j].x := x;
       p[k,j].y := y;
```

```
      p[k,j].z := 0.30*sin(0.1*sqrt((x*SPACE*x*SPACE +
            y*SPACE*y*SPACE)) + (anim*2.0*M_PI/ANIMLEN));
        x:=x+EDGELEN;
      end;
      y :=y-EDGELEN;
      x:=-STARTCOORD;
   end;

 glNewList(anim+1, GL_COMPILE);
   resetMaterial();
   for j:=0 to POINTSPERSIDE-1-1 do
   begin
      for i :=0 to POINTSPERSIDE-1-1 do
      begin
         changeMaterial();
         glBegin(GL_QUADS);

         // do the normal vector */
         v1[0] := p[j,i].x - p[j+1,i].x;
         v1[1] := p[j,i].y - p[j+1,i].y;
         v1[2] := p[j,i].z - p[j+1,i].z;

         v2[0] := p[j+1,i+1].x - p[j+1,i].x;
         v2[1] := p[j+1,i+1].y - p[j+1,i].y;
         v2[2] := p[j+1,i+1].z - p[j+1,i].z;

         normCrossProd(v1, v2, normal);
         glNormal3fv(@normal);

         // vertexes */
         glVertex3f(   p[j,i].x,    p[j,i].y,    p[j,i].z);
         glVertex3f(  p[j+1,i].x,  p[j+1,i].y,  p[j+1,i].z);
         glVertex3f(p[j+1,i+1].x, p[j+1,i+1].y, p[j+1,i+1].z);
         glVertex3f(  p[j,i+1].x,  p[j,i+1].y,  p[j,i+1].z);
         glEnd();
      end;
   end;
 glEndList();

end;

procedure init();
var
  i : integer;
begin

   glClearColor(0.0, 0.0, 0.0, 0.0);
   glShadeModel(GL_SMOOTH);
   glEnable(GL_DEPTH_TEST);

   glMaterialfv(GL_FRONT, GL_SPECULAR, @matSpecular);
   glMaterialfv(GL_FRONT, GL_SHININESS, @matShininess);
   glLightfv(GL_LIGHT0, GL_POSITION, @lightPosition);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);

   for i :=0  to ANIMLEN-1 do
      initDisplayList(i);
```

```
    dispList := 1;
end;

procedure reshape( w,  h : integer);
begin
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0,  w/h, 1.0, 20.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -3.5);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 f_Hdc := GetDC(Panel1.Handle);
 SetDCPixelFormat(f_Hdc,16,16);// Create a rendering context.
 init();
 glInfo;
 Application.OnIdle := Idle;
end;

procedure TForm1.FormResize(Sender: TObject);
begin
   wglMakeCurrent(f_Hdc,hrc); //activate the RC
   reshape(Panel1.Width, Panel1.Height);
   InvalidateRect(Handle, nil, False);// Draw the scene.
end;

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

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

procedure TForm1.Idle(Sender: TObject; var Done: Boolean);
begin
 my_idle();
 InvalidateRect(Handle, nil, False);//  Draw the scene.
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
   case key of
       '1':
          m_x:=m_x-1;
       '2':
          m_x := 0.0;
       '3':
          m_x:=m_x+1;
       'q':
```

```
            m_y:=m_y-1;
        'w':
            m_y := 0.0;
        'e':
            m_y:=m_y+1;
        'a':
            m_z:=m_z-1;
        's':
            m_z := 0.0;
        'd':
            m_z:=m_z+1;
        'x':
            m_anim := not(m_anim);
        'z':
            m_spin := not(m_spin);
        'r':
            SaveAsBmp(Panel1.Handle ,f_hdc ,'dd.bmp',
                    Panel1.height,Panel1.width);

    end;
    InvalidateRect(Handle, nil, False);//  Draw the scene.
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   i: integer;
   MyBitmap: TBitmap;
begin
  Screen.Cursor := crHourGlass;
  Button1.Enabled := false;
     for i:=0 to 19 do
        begin
           SaveAsBmp(Panel1.Handle , f_hdc ,
                   'dd'+inttostr(i)+'.bmp',
                    Panel1.height,Panel1.width);
           MyBitmap := TBitmap.Create;
           MyBitmap.LoadFromFile('dd'+inttostr(i)+'.bmp');
           BitmapListBox.AddItem('dd'+inttostr(i)+'.bmp',MyBitmap);
           sleep(500);
           my_idle();
           InvalidateRect(Handle, nil, False);//  Draw the scene.
           application.ProcessMessages;
        end;
    SaveAs_AVI('dd.avi',BitmapListBox,10);
    ShowMessage('AVI file created.');
  Screen.Cursor := crDefault;
  Button1.Enabled := true;
end;

end.
```