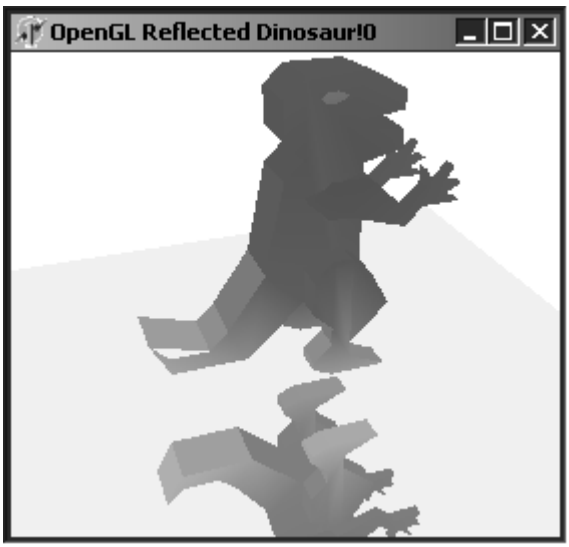
فصل بیست و یکم

انعکاس و مفروش سازی



مقدمه :

اشیاء در دنیای واقعی دارای انعکاس نیز می باشند . OpenGL تمامی امکانات مورد نیاز برای رندر کردن انعکاس های سریع و با کیفیت بالا را بر روی سطوح صفحه ای دارا است . شبیه سازی انعکاس نیازمند انتقالات سه بعدی ، Blending و Stenciling می باشد .

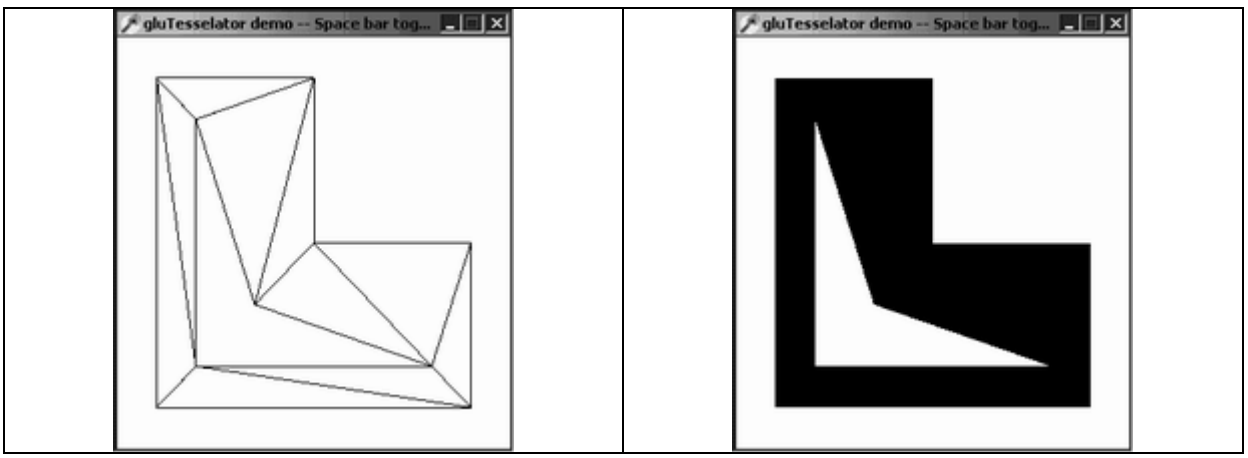
الگوریتم شبیه سازی انعکاس :

اولین مطلبی را که باید در نظر داشت این است که انعکاس بر روی یک سطح صفحه ای چیزی بیش از تصویرگری (Projection) صحیح شیء بر روی صفحه منعکس کننده نمی باشد . تابع glScalef(1,-1,1) همانگونه که پیشتر نیز ذکر گردیده بود ، سبب انعکاس شیء خواهد شد .

عموما شیء از روی سطوحی انعکاس می یابد که کاملا منعکس کننده نیستند مانند سطحی مرمری . برای رفع این مشکل می توان از توابع Blending استفاده نمود . توسط این روش رنگ صفحه منعکس کننده ، ترکیبی دلخواه از رنگ خودش و رنگ انعکاسی خواهد شد .
اگر شیء ترسیم شده را از زاویه دیدی کوچک ملاحظه کنیم ، دو شیء ترسیم شده را خواهیم دید . زیرا انعکاس هنگامی صورت خواهد گرفت که صفحه ای برای آن وجود داشته باشد . برای حل این مشکل می توان از Stenciling کمک گرفت . بدین طریق بدون نیاز به ، به روز در آوردن رنگ و عمق بافر ، می توان صفحه انعکاس را در بافر Stencil پیش ترسیم نمود . سپس هنگامی که انعکاس باید ترسیم گردد ، فقط کافی است تا به انعکاس اجازه داده شود ، نقاطی را که بعنوان مقادیر استنسیل صفحه انعکاس مشخص گشته اند ، به روز درآورد . در این حالت صفحه انعکاس ، چند ضلعی مسطح پیچیده ای نیز می تواند باشد .

مفروش سازی (Tesselation) چند ضلعی ها :

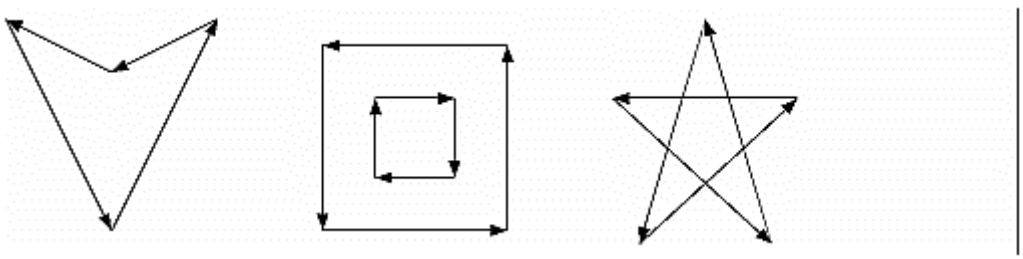
OpenGL تنها می تواند چند ضلعی های ساده ی محدب را نمایش دهد . در یک چند ضلعی ساده اضلاع تنها در رئوس تلاقی می کنند و رئوس المثنی نیز ندارند . برای نمایش چند ضلعی های غیر محدب ، مانند چند ضلعی هایی حاوی چند حفره ، باید در ابتدا آنها را به چند ضلعی های محدب تقسیم کرد (مفروش سازی آن ها با مثلث ها) . به این تقسیمات ، مفروش سازی گویند . کتابخانه GLU مجموعه ای از توابع را برای انجام اینکار فراهم آورده است .



الگوریتم استفاده از امکانات مفروش سازی چند ضلعی ها :

۱- شیء مفروش سازی را توسط دستور gluNewTess خلق کنید .
۲- gluTessCallback برای تعریف توابع CallBack که از آنها برای تولید مثلث ها بوسیله مفروش ساز استفاده می کنید ، بکار می رود .
۳- با استفاده از توابع gluTessBeginPolygon ، gluTessVertex ، gluNextContour و gluTessEndPolygon چند ضلعی های غیر محدبی را که باید مفروش شوند ، معین نمایید.

هنگامیکه تعریف چند ضلعی کامل شد ، مفروش ساز ، توابع Callback شما را در صورت نیاز فراخوانی می کند . شکل های زیر چند ضلعی هایی را نشان می دهند که برای ترسیم شدن نیاز به مفروش سازی دارند .



مروری بر توابع :

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
GLUtesselator *gluNewTess(void);	Function gluNewTess: PGLUtesselator; stdcall; external 'GLU32.DLL';

توضیح :

یک شیء مفروش سازی را خلق می کند و اشاره گری را به آن بر می گرداند . اگر خروجی آن صفر باشد به این معنی است که حافظه کافی برای انجام اینکار وجود ندارد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluTessCallback(GLUtesselator *tess, GLenum which, void (* fn)());	Procedure gluTessCallback(tess: PGLUtesselator; which: GLenum; callback: Pointer); stdcall; external 'GLU32.DLL';

توضیح :

یک Callback را برای شیء مفروش سازی تعریف می کند . شیء مفروش سازی این Callback ها را بکار می گیرد تا چگونگی تقسیم چند ضلعی را به مثلث ها ، معین کند .

آرگومان tess شیء مفروش سازی است که توسط تابع gluNewTess خلق شده است .

which : Callback ایی که تعریف می شود . مقادیر زیر برای آن مجاز می باشد :

GLU_TESS_BEGIN, GLU_TESS_BEGIN_DATA, GLU_TESS_EDGE_FLAG, GLU_TESS_EDGE_FLAG_DATA, GLU_TESS_VERTEX, GLU_TESS_VERTEX_DATA, GLU_TESS_END, GLU_TESS_END_DATA, GLU_TESS_COMBINE, GLU_TESS_COMBINE_DATA, GLU_TESS_ERROR, and GLU_TESS_ERROR_DATA.

به ازای هر Callback دو نگارش وجود دارد ، یکی با داده های چند ضلعی و دیگری بدون آن . اگر هر دو نگارش تعریف شوند ، Callback ایی انتخاب خواهد شد که داده های چند ضلعی را در بر دارد .

Fn : تابعی که فراخوانی خواهد شد .

توضیحات :

GLU_TESS_BEGIN : این Callback همانند glBegin برای تعیین شروع چند ضلعی بکار می رود . اگر خاصیت GLU_TESS_BOUNDARY_ONLY را به GL_FALSE تنظیم کرده باشید ، آرگومان به یکی از موارد زیر می تواند تنظیم شود :

GL_TRIANGLE_FAN, GL_TRIANGLE_STRIP, or GL_TRIANGLES

و اگر GLU_TESS_BOUNDARY_ONLY را به GL_TRUE تنظیم کنید ، آرگومان به GL_LINE_LOOP تنظیم می شود .

GLU_TESS_BEGIN_DATA : همانند GLU_TESS_BEGIN می باشد با این تفاوت که آرگومان دیگری را به صورت اشاره گر نیز دریافت می کند .

GLU_TESS_EDGE_FLAG : شبیه به تابع glEdgeFlage می باشد . تابع یک پرچم Boolean را که معین کننده این است که کدام ضلع روی مرز چند ضلعی قرار گیرد ، می پذیرد .

اگر استفاده از این Callback لازم بود ، باید قبل از ایجاد اولین Callback مربوط به راس ، تعریف شود . توجه داشته باشید که triangle fans و triangle strips پرچم های اضلاع فوق را پشتیبانی نمی کنند .

GLU_TESS_VERTEX : بین Callback های آغازین و انتهایی بکار گرفته می شود . همانند glVertex بوده و ئوس مثلث های تولید شده توسط فرایند مفروش سازی را تعریف می کند . تابع یک اشاره گر را بعنوان تنها آرگومان خودش می پذیرد .

GLU_TESS_END : این Callback همانند glEnd عمل کرده و انتهای چند ضلعی را مشخص می کند و آرگومانی را نیز نمی پذیرد .

GLU_TESS_COMBINE : از آن برای خلق راس جدید هنگامیکه مفروش ساز یک تقاطع را تشخیص می دهد استفاده می گردد . تابع آن چهار آرگومان را می پذیرد :

آرایه ای با سه عنصر برای تعیین موقعیت راس جدید .

آرایه ای از چهار اشاره گر به راس های موجود .

آرایه ای با چهار عنصر برای ضرایب ترکیب خطی .

اشاره گری به اشاره گر که در حقیقت اشاره گری خواهد بود به خروجی حاصل .

هنگامیکه مفروش ساز تقاطعی را تشخیص می دهد ، باید از این Callback استفاده نمود ، درغیر اینصورت خطا رخ خواهد داد .

GLU_TESS_ERROR : این Callback هنگامی فراخوانی می شود که خطایی رخ دهد و تنها آرگومان آن که بیانگر نوع خطا است یکی از ثوابت زیر می باشد :

GLU_TESS_MISSING_BEGIN_POLYGON, GLU_TESS_MISSING_END_POLYGON,
GLU_TESS_MISSING_BEGIN_CONTOUR, GLU_TESS_MISSING_END_CONTOUR,
GLU_TESS_COORD_TOO_LARGE, or GLU_TESS_NEED_COMBINE_CALLBACK

برای دریافت این خطا ها باید از gluErrorString استفاده نمود .

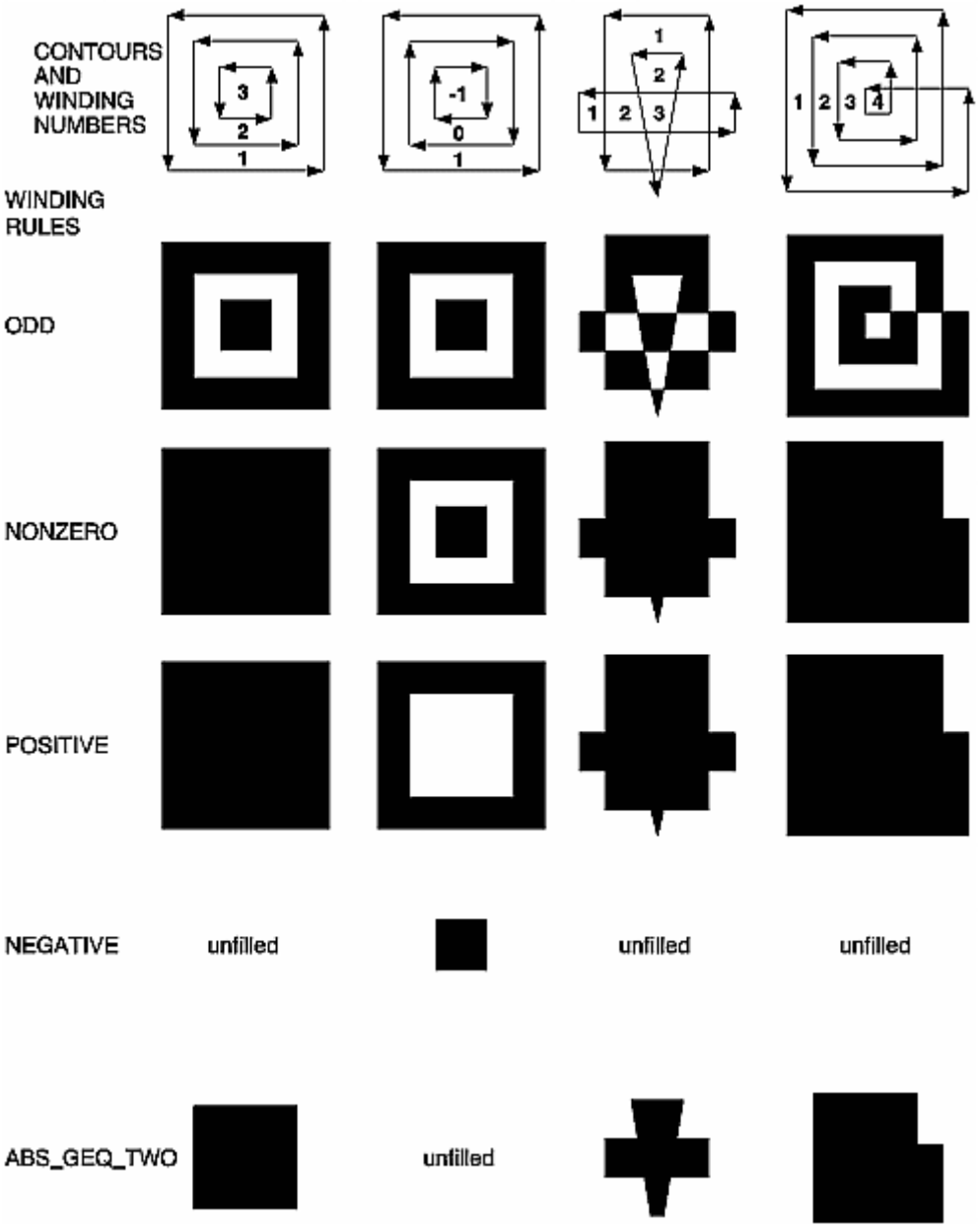
نکته :

برای صرفه جویی در انجام محاسبات ، چند ضلعی مفروش شده را در یک لیست نمایشی ذخیره کنید .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluTessProperty(GLUtesselator *tess, GLenum which, GLdouble value);	Procedure gluTessProperty(tess: PGLUtesselator; which: GLenum; value: GLdouble); stdcall; external 'GLU32.DLL';

توضیح :

خواص شیء مفروش ساز را تنظیم می کند و بر روی نحوه ترجمه و رندر شدن آن ، تاثیر گذار است .



winding rules

آرگومان tess شیء مفروش سازی است که توسط تابع gluNewTess خلق شده است .

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

GLU_TESS_WINDING_RULE, GLU_TESS_BOUNDARY_ONLY, and GLU_TESS_TOLERANCE

value : مقدار خاصیت معین شده است .

توضیحاتی در مورد ثوابت فوق :

GLU_TESS_WINDING_RULE : معین می کند که کدام قسمت های چند ضلعی بر روی قسمت درونی قرار دارند . پارامتر Value در این حالت یکی از مقادیر زیر می تواند باشد :

GLU_TESS_WINDING_ODD, GLU_TESS_WINDING_NONZERO, GLU_TESS_WINDING_POSITIVE, GLU_TESS_WINDING_NEGATIVE, or GLU_TESS_WINDING_ABS_GEQ_TWO

برای اینکه درک کنید WINDING_RULE چگونه کار می کند ، در ابتدا فرض کنید که چند ضلعی توسط کانتورهایی به چند ناحیه تقسیم شده است . WINDING_RULE معین می کند که کدامیک از این نواحی در درون چند ضلعی قرار گیرند .

GLU_TESS_BOUNDARY_ONLY : در این حالت Value مقداری Boolean خواهد بود . هنگامیکه مقدار آن به GL_TRUE تنظیم می شود ، بجای مفروش سازی ، مجموعه ای از کانتورهای بسته چند ضلعی را به قسمت های داخلی و خارجی تقسیم می کنند . کانتورهای بیرونی در جهت خلاف حرکت عقربه های ساعت ، با توجه به بردار نرمال ، قرار می گیرند و کانتورهای داخلی برعکس . Callback های GLU_TESS_BEGIN و GLU_TESS_DATA نوع GL_LINE_LOOP را برای هر کانتور بکار خواهند برد .

GLU_TESS_TOLERANCE : تلرانسی را برای ممزوج کردن به جهت کاهش خروجی معین می کند . برای مثال دو راسی که خیلی به هم نزدیک باشند ، یک راس در نظر گرفته می شوند .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void gluTessBeginPolygon(GLUtesselator *tess, void *polygon_data); void gluTessEndPolygon(GLUtesselator * tess);	Procedure gluTessBeginPolygon(tess: PGLUtesselator; polygon_data: Pointer); stdcall; external 'GLU32.DLL'; Procedure gluTessEndPolygon(tess: PGLUtesselator); stdcall; external 'GLU32.DLL';

توضیح :

این توابع تعریف یک چند ضلعی را محدود می کنند .

آرگومان tess شیء مفروش سازی است که توسط تابع gluNewTess خلق شده است .

polygon_data : اشاره گری است به داده های چند ضلعی .

اگر Callback های مناسبی توسط gluTessCallback تعریف شده باشند ، این اشاره گر به توابع Callback فرستاده می شود .

مثال (شکلی چهارگوش با حفره ای مثلثی در میان آن) :

```
gluTessBeginPolygon(tobj, NULL);
  gluTessBeginContour(tobj);
    gluTessVertex(tobj, v1, v1);
    gluTessVertex(tobj, v2, v2);
    gluTessVertex(tobj, v3, v3);
    gluTessVertex(tobj, v4, v4);
  gluTessEndContour(tobj);
  gluTessBeginContour(tobj);
    gluTessVertex(tobj, v5, v5);
    gluTessVertex(tobj, v6, v6);
    gluTessVertex(tobj, v7, v7);
  gluTessEndContour(tobj);
gluTessEndPolygon(tobj);
```

البته مثال فوق به زبان C نوشته شده است که تبدیل آن به دلفی کار مشکلی به نظر نمی رسد !

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void gluTessBeginContour(
  GLUtesselator *tess
);
 	
void gluTessEndContour(
  GLUtesselator *tess
);
	Procedure gluTessBeginContour(
	  tess: PGLUtesselator);
	  stdcall; external 'GLU32.DLL';
	Procedure gluTessEndContour(
	  tess: PGLUtesselator);
	  stdcall; external 'GLU32.DLL';

توضیح :

این توابع کانتور یک چند ضلی را معین می سازند . درون هر جفت از آنها می تواند صفر یا چند فراخوانی gluTessVertex صورت گیرد . آخرین راس هر کانتور به صورت اتوماتیک به اولین متصل می شود . شما می توانید این توابع را تنها بین gluTessBeginPolygon و gluTessEndPolygon بکار ببرید .

آرگومان tess شیء مفروش سازی است که توسط تابع gluNewTess خلق شده است .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی

void gluTessVertex(
  GLUtesselator *tess,
  GLdouble coords[3],
  void *data
  );
	Procedure gluTessVertex(tess: PGLUtesselator;
	var coords: TCoordArray; data: Pointer);
	  stdcall; external 'GLU32.DLL';

توضیح :

راسی را بر روی چند ضلعی معین می کند .

آرگومان tess شیء مفروش سازی است که توسط تابع gluNewTess خلق شده است .

coords : مکان راس

data : به موقعیت راس اشاره می کند .

شما می توانید از این تابع تنها در بین gluTessBeginContour و gluTessEndContour استفاده کنید .

تابع به فرمت زبان C
void gluNextContour(GLUtesselator *tess , GLenum type);

توضیح :

این تابع شروع کانتور دیگری را معین می نماید .

آرگومان tess شیء مفروش سازی است که توسط تابع gluNewTess خلق شده است .

type : نوع کانتوری است که در حال تعریف است . مقادیر زیر برای آن مجاز هستند :

GLU_EXTERIOR : کانتور بیرونی مرز بیرونی کانتور را تعریف می کند .

GLU_INTERIOR : کانتور داخلی مرز داخلی کانتور را تعریف می کند .

GLU_UNKNOWN : کانتور نامعین بوسیله کتابخانه مورد بررسی قرار گرفته و نوع داخلی یا خارجی بودن آن معین می شود .

GLU_CCW, GLU_CW : این نوع کانتورها خارجی در نظر گرفته می شوند . تمام سایر کانتورها خارجی درنظر گرفته خواهند شد اگر در جهتی یکسان (درجهت حرکت عقربه های ساعت و یا خلاف آن) با کانتور اول قرار گرفته باشند و در غیر اینصورت داخلی درنظر گرفته می شوند .

باید خاطر نشان کرد که اگر از این انوع ها استفاده شود ، تمام کانتورهای بعدی نیز باید از این انواع استفاده کنند در غیر اینصورت تبدیل به GLU_UNKNOWN خواهند شد.

مثال (به زبان C) :

```
gluBeginPolygon(tess);
   gluTessVertex(tess, v1, v1);
   gluTessVertex(tess, v2, v2);
   gluTessVertex(tess, v3, v3);
   gluTessVertex(tess, v4, v4);
gluNextContour(tess, GLU_INTERIOR);
```

```
    gluTessVertex(tess, v5, v5);
    gluTessVertex(tess, v6, v6);
    gluTessVertex(tess, v7, v7);
gluEndPolygon(tess);
```

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

توضیح :

در صحنه ای که اشیاء آنرا مشاهده می کنیم ، بدیهی است قسمتی از آنها که روبه ما نیستند ، قابل مشاهده نخواهند بود و اگر این قسمت ها را در هنگام ترسیم ، حذف نمائیم ، تاثیر بسزایی در سرعت رندر شدن تصاویر خواهند گذاشت .

برای انجام این کار ابتدا باید ترسیم و یا عدم ترسیم چند ضلعی هایی را که در عقب شیء واقع شده اند را با استفاده از توابع glEnable و glDisable با آرگومان **GL_CULL_FACE** معین نماییم .

تابع glFrontFace توسط آرگومان mode معین می کند که ترسیم رئوس چند ضلعی های روبه ما در جهت یا خلاف جهت عقربه های ساعت صورت گیرد . دو ثابت GL_CW و GL_CCW برای آن قابل قبول هستند . مقدار پیش فرض آن GL_CCW می باشد .

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

توضیح :

این تابع توسط آرگومان mode برای تعیین اینکه آیا جلو یا عقب یک شیء ترسیم گردد یا خیر بکار می رود . ثوابت GL_FRONT, GL_BACK, GL_FRONT_AND_BACK برای آن مجاز هستند .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glColorMask(GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha);	Procedure glColorMask(red: GLboolean; green: GLboolean; blue: GLboolean; alpha: GLboolean); stdcall; external 'OPENGL32.DLL';

توضیح :

این تابع نوشته شدن اجزاء رنگ بافر چارچوب را میسر می سازد و یا برعکس.

آرگومان های red, green, blue, alpha معین می کنند که آیا اجزاء قرمز ، سبز ، آبی و یا آلفا، می توانند به درون بافر چارچوب نوشته شوند یا خیر . مقدار پیش فرض برای تمام آنها GL_TRUE است .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glStencilOp(GLenum fail, GLenum zfail, GLenum zpass);	Procedure glStencilOp(fail: GLenum; zfail: GLenum; zpass: GLenum); stdcall; external 'OPENGL32.DLL';

توضیح :

تنظیمات آزمون استنسیل را تنظیم می کند . استنسیل همانند بافر Z ، ترسیم یا عدم ترسیم را بر مبنای یک نقطه ، میسر می سازد . بوسیله این امکان می توان ترسیمات را بر روی صفحات استنسیل انجام داد و سپس تصاویر را رندر نمود و آنها را در قسمتی از صفحه ، نمایش داد . از این توانایی عموما برای خلق جلوه های ویژه استفاده می شود .

آزمون استنسیل نقاط را بر مبنای مقایسه بین مقدار آنها در بافر استنسیل و مقدار مرجع ، انتخاب و حذف می کند . این آزمون توسط glEnable(GL_STENCIL); فعال می شود و توسط تابع glStencilFunc کنترل می گردد . تابع glStencilOp سه آرگومان را می پذیرد و توسط آنها معین می کند که چه اتفاقی باید بر روی مقدار ذخیره شده استنسیل رخ دهد .

آرگومان fail : عملیاتی که هنگام پایان کار و زوال این آزمون باید صورت گیرد را معین کرده و ثوابت زیر را می پذیرد :

GL_KEEP : مقدار جاری را حفظ می کند .

GL_ZERO : مقدار بافر استنسیل را به صفر تنظیم می کند .

GL_REPLACE : مقدار بافر استنسیل را به مقدار مرجع که توسط تابع glStencilFunc معین می گردد ، تنظیم می کند .

GL_INCR : مقدار جاری بافر استنسیل را افزایش می دهد .

GL_DECR : مقدار جاری بافر استنسیل را کاهش می دهد .

GL_INVERT : با استفاده از عملیات بیتی ، مقدار جاری بافر استنسیل را معکوس می کند .

آرگومان zfail : عملیات استنسیل ، هنگامیکه آزمون استنسیل انجام شده و آزمون عمق زوال و خاتمه یافته است را معین می کند . آن نیز همانند آرگومان fail مقادیر ثابت فوق را می پذیرد .

آرگومان zpass : عملیات استنسیل است هنگامیکه هر دو آزمون استنسیل و عمق انجام شده اند و یا هنگامیکه آزمون استنسیل پایان پذیرفته و عمق بافر و آزمون عمق غیرفعال هستند . آن نیز همانند آرگومان fail مقادیر ثابت فوق را می پذیرد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glStencilFunc(GLenum func, GLint ref, GLuint mask);	Procedure glStencilFunc(func: GLenum; ref: GLint; mask: GLuint); stdcall; external 'OPENGL32.DLL';

توضیح :

مقدار مرجع و تابع آزمون استنسیل را تنظیم می کند .

آرگومان func : تابع آزمون می باشد و طریقه مقایسه را معین می سازد . هشت ثابت زیر برای آن مجاز هستند :

GL_NEVER : همواره سبب خاتمه عملیات خواهد شد .

GL_LESS : انجام خواهد شد اگر (*ref* & *mask*) < (*stencil* & *mask*)

GL_LEQUAL : انجام خواهد شد اگر (*ref* & *mask*) ≤ (*stencil* & *mask*)

GL_GREATER : انجام خواهد شد اگر (*ref* & *mask*) > (*stencil* & *mask*)

GL_GEQUAL : انجام خواهد شد اگر (*ref* & *mask*) ≥ (*stencil* & *mask*)

GL_EQUAL : انجام خواهد شد اگر (*ref* & *mask*) = (*stencil* & *mask*)

GL_NOTEQUAL : انجام خواهد شد اگر (*ref* & *mask*) ≠ (*stencil* & *mask*)

GL_ALWAYS : همواره انجام خواهد شد .

ref : مقدار مرجع برای آزمون استنسیل . مقدار آن در بازه $[0,2^n-1]$ قرار دارد ، بطوریکه n تعداد صفحات بیتی در بافر استنسیل می باشد .

Mask : الگویی است که در هنگام اجرای آزمون ، با مقدار مرجع و مقدار ذخیره شده استنسیل ، AND خواهد شد .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glStencilMask(GLuint mask);	Procedure glStencilMask (mask: GLuint); stdcall; external 'OPENGL32.DLL';

توضیح :

طریقه نگارش بیت های منفرد را در صفحات استنسیل کنترل می کند .

پارامتر بیتی mask برای فعال و غیر فعال کردن نگارش بیت های منفرد در صفحات استنسیل بکار می رود .

چند مثال در زمینه کاربرد آزمون استنسیل :

اندود کردن :

فرض کنید که در حال ترسیم یک شیء محدب هستید که از چندین چندضلعی ساخته خواهد شد و یک صفحه که شیء را قطع خواهد کرد . در این حالت شاید مایل باشید که بجای قابل مشاهده بودن درون آن ، سطح با رنگی خاص ، اندود گردد . برای انجام این کار ، ابتدا بافر استنسیل را با صفر پاک کنید و با فعال سازی عملیات استنسیل ، مقایسه استنسیل را طوری تنظیم کنید که همواره سگمنت ها را بپذیرد . مقدار صفحات استنسیل را در هر لحظه ای که یک سگمنت پذیرفته می شود ، معکوس نمایید . بعد از اینکه تمام اجزاء شیء ترسیم شدند ، ناحیه ای از صفحه که نیاز به اندود کاری ندارد ، مقدار صفحات استنسیل آن مساوی صفر خواهد بود و نواحیی که نیاز به اندود سازی دارند ، غیر صفر می باشند . تابع استنسیل را دوباره طوری تنظیم نمایید که فقط در نواحی غیر صفر ترسیم گردد و سپس چند ضلعی حاصل را با رنگ اندود سازی ، بطوریکه کل صفحه را قطع کند ، ترسیم کنید .

ساخت چند ضلعی های شفاف که قسمتی از آنها روی هم قرار می گیرد :

برای انجام این کار از صفحات استنسیل کمک بگیرید تا از اینکه هر جزء با حداقل یک جزء پوشیده می گردد ، اطمینان حاصل کنید . ابتدا بافر استنسیل را با صفر پاک کنید و سپس ترسیم را فقط هنگامیکه صفحه استنسیل صفر است ، انجام دهید و مقدار صفحه استنسیل را هنگامیکه در حال ترسیم هستید ، افزایش دهید .

تابع به فرمت زبان C	تابع به فرمت زبان دلفی
void glBlendFunc(GLenum sfactor, GLenum dfactor);	Procedure glBlendFunc(sfactor: GLenum; dfactor: GLenum); stdcall; external 'OPENGL32.DLL';

توضیح :

این تابع عملیاتی حسابی را بر روی نقاط انجام می دهد . در حالت RGB ، می توان توسط این تابع ، اجزاء RGBA نقاط در حال ترسیم را (منبع) با RGBA نقاط موجود در بافر چارچوب (مقصد) ترکیب کرد و جلوه های ویژه جالبی را پدید آورد . بصورت پیش فرض این عملیات غیر فعال است و آنرا می توان توسط دستور ;(glEnable(GL_BLEND فعال نمود .

پارامتر sfactor : معین می کند که چگونه فاکتورهای قرمز ، سبز ، آبی و آلفای منبع محاسبه شوند . نه ثابت زیر برای این منظور قابل قبول هستند :

GL_ZERO, GL_ONE, GL_DST_COLOR, GL_ONE_MINUS_DST_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA, and GL_SRC_ALPHA_SATURATE

پارامتر dfactor : معین می کند که چگونه فاکتورهای قرمز ، سبز ، آبی و آلفای مقصد محاسبه شوند . هشت ثابت زیر برای این منظور قابل قبول هستند :

GL_ZERO, GL_ONE, GL_SRC_COLOR, GL_ONE_MINUS_SRC_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, and GL_ONE_MINUS_DST_ALPHA

درجدول و معادلات زیر ، اجزاء رنگ منبع و مقصد به ترتیب به صورت (R_S,G_S,B_S,A_S) و (R_d,G_d,B_d,A_d) معین شده اند . آنها مقادیر صحیحی هستند در بازه صفر و (k_R,k_G,k_R,k_A) ، بطوریکه :

$$k_R = 2^{mR} - 1$$

$$k_G = 2^{mG} - 1$$

$$k_B = 2^{mB} - 1$$

$$k_A = 2^{mA} - 1$$

و (m_R, m_G, m_B, m_A) تعداد صفحات بیتی قرمز ، سبز ، آبی و آلفا است . فاکتورهای مقیاس منبع و مقصد به ترتیب به صورت (s_R, s_G, s_B, s_A) و (d_R, d_G, d_B, d_A) نمایش داده شده اند . فاکتورهای مقیاس موجود در جدول نیز به صورت (f_R, f_G, f_B, f_A) بیان شده اند که برای منبع یا مقصد بکار گرفته می شوند . تمام فاکتورهای مقیاس در بازه صفر و یک قرار دارند .

Parameter	(f (R) ,f (G) ,f (B) ,f (A))
GL_ZERO	(0,0,0,0)
GL_ONE	(1,1,1,1)
GL_SRC_COLOR	$(R_S/k_R,G_S/k_G,B_S/k_B,A_S/k_A)$
GL_ONE_MINUS_SRC_COLOR	(1,1,1,1) - $(R_S/k_R,G_S/k_G,B_S/k_B,A_S/k_A)$
GL_DST_COLOR	$(R_d/k_R,G_d/k_G,B_d/k_B,A_d/k_A)$
GL_ONE_MINUS_DST_COLOR	(1,1,1,1)
GL_SRC_ALPHA	$(R_d/k_R,G_d/k_G,B_d/k_B,A_d/k_A)$ - $(A_S/k_A,A_S/k_A,A_S/k_A,A_S/k_A)$
GL_ONE_MINUS_SRC_ALPHA	(1,1,1,1) - $(A_S/k_A,A_S/k_A,A_S/k_A,A_S/k_A)$
GL_DST_ALPHA	$(A_D/k_A,A_D/k_A,A_D/k_A,A_D/k_A)$
GL_ONE_MINUS_DST_ALPHA	(1,1,1,1) - $(A_D/k_A,A_D/k_A,A_D/k_A,A_D/k_A)$
GL_SRC_ALPHA_SATURATE	(i,i,i,1)

در جدول :

$$i = \min (A_S,k_A - A_D) / /k_A$$

برای تعیین مقادیر نهایی RGBA ترکیب شده ، سیستم از معادلات زیر کمک می گیرد :

$$R (d) = \min(k_R,R_s s_R+R_d d_R)$$

$$G (d) = \min(k_G,G_s s_G+G_d d_G)$$

$$B (d) = \min(k_B,B_s s_B+B_d d_B)$$

$$A (d) = \min(k_A,A_s s_A+A_d d_A)$$

چند مثال :

شفافیت به بهترین شکل با استفاده از دستور زیر پیاده سازی می شود :

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```

برای رندر کردن نقاط و خطوط *antialiased* می توان از دستور زیر کمک گرفت :

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```

برنامه ی اول فصل :

```
unit ch21_01;

interface

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

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

procedure gluNextContour(tess: GLUtesselator; atype: GLenum);
 stdcall;external glu32;

 const
  // Contours types -- obsolete!
  GLU_CW          = 100120;
  GLU_CCW         = 100121;
  GLU_INTERIOR    = 100122;
  GLU_EXTERIOR    = 100123;
  GLU_UNKNOWN     = 100124;

{$R *.dfm}

var
  currentWinding : GLdouble = GLU_TESS_WINDING_ODD;
  currentShape : integer = 0;
  tobj : GLUtesselator; // the tessellation object
  list : GLuint;
  f_Hdc : longInt;

 rects : array[0..11] of array[0..2] of GLdouble =
 ((50.0, 50.0, 0.0), (300.0, 50.0, 0.0),
  (300.0, 300.0, 0.0), (50.0, 300.0, 0.0),
  (100.0, 100.0, 0.0), (250.0, 100.0, 0.0),
  (250.0, 250.0, 0.0), (100.0, 250.0, 0.0),
  (150.0, 150.0, 0.0), (200.0, 150.0, 0.0),
```

```
  (200.0, 200.0, 0.0), (150.0, 200.0, 0.0));

 spiral : array[0..15] of array[0..2] of GLdouble =
((400.0, 250.0, 0.0), (400.0, 50.0, 0.0),
 (50.0, 50.0, 0.0), (50.0, 400.0, 0.0),
 (350.0, 400.0, 0.0), (350.0, 100.0, 0.0),
 (100.0, 100.0, 0.0), (100.0, 350.0, 0.0),
 (300.0, 350.0, 0.0), (300.0, 150.0, 0.0),
 (150.0, 150.0, 0.0), (150.0, 300.0, 0.0),
 (250.0, 300.0, 0.0), (250.0, 200.0, 0.0),
 (200.0, 200.0, 0.0), (200.0, 250.0, 0.0));

 quad1 : array[0..3] of array[0..2] of GLdouble =
((50.0, 150.0, 0.0), (350.0, 150.0, 0.0),
 (350.0, 200.0, 0.0), (50.0, 200.0, 0.0));

quad2 : array[0..3] of array[0..2] of GLdouble =
((100.0, 100.0, 0.0), (300.0, 100.0, 0.0),
 (300.0, 350.0, 0.0), (100.0, 350.0, 0.0));

tri : array[0..2] of array[0..2] of GLdouble =
((200.0, 50.0, 0.0), (250.0, 300.0, 0.0),
 (150.0, 300.0, 0.0));

type
   my_arr = array of GLdouble;

{
 * This program demonstrates the winding rule polygon
 * tessellation property. Four tessellated objects are drawn,
 * each with very different contours. When a key is pressed,
 * the objects are drawn with a different winding rule.
}

 //Note: without the stdcall directive, Windows will not be able to access
 // the callback function.
 //This ensures parameters are passed in the correct order.
procedure errorCallback( errorCode :GLenum); stdcall;
var
  estring : Pchar;
begin

   Case errorCode of

      GLU_TESS_MISSING_BEGIN_POLYGON :
            MessageBox(0,'GLU_TESS_MISSING_BEGIN_POLYGON',
                  'error..',MB_OK);

      GLU_TESS_MISSING_END_POLYGON :
            MessageBox(0,'GLU_TESS_MISSING_END_POLYGON',
                  'error..',MB_OK);

      GLU_TESS_MISSING_BEGIN_CONTOUR :
            MessageBox(0,'GLU_TESS_MISSING_BEGIN_CONTOUR',
                  'error..',MB_OK);

      GLU_TESS_MISSING_END_CONTOUR :
            MessageBox(0,'GLU_TESS_MISSING_END_CONTOUR',
```

```
                    'error..',MB_OK);

      GLU_TESS_COORD_TOO_LARGE :
            MessageBox(0,'GLU_TESS_COORD_TOO_LARGE',
                     'error..',MB_OK);

      GLU_TESS_NEED_COMBINE_CALLBACK :
            MessageBox(0,'GLU_TESS_NEED_COMBINE_CALLBACK',
                    'error..',MB_OK);
    End ;

  estring := gluErrorString(errorCode);
  MessageBox(0,estring,'error..',MB_OK);

end;

procedure beginCallback(which : GLenum );stdcall;
begin
  glBegin(which);
end;

procedure endCallback();stdcall;
begin
  glEnd();
end;

procedure tessVertex(data: Pointer); stdcall;
begin

  { This callback is invoked for each vertex in the triangulation. The data
    parameter is a pointer we passed to gluTessVertex(). (see below)
    In this case, it points to a double-precision 3D vector. }
  glVertex3dv(data);
end;

{* combineCallback is used to create a new vertex when edges
* intersect. coordinate location is trivial to calculate,
* but weight[4] may be used to average color, normal, or texture
* coordinate data.}

{ void CALLBACK combineCallback(GLdouble coords[3],
   GLdouble *data[4],
   GLfloat weight[4], GLdouble **dataOut )
{
 GLdouble *vertex;
 vertex = (GLdouble *) malloc(3 * sizeof(GLdouble));
 vertex[0] = coords[0];
 vertex[1] = coords[1];
 vertex[2] = coords[2];
 *dataOut = vertex;
}


// this procedure has a little BUG!!
procedure combineCallback(
            coords : my_arr;
            var vdata   : pointer;
            weight  : pointer;
```

```
            var dataOut : my_arr); stdcall;
var
  vertex : my_arr;
begin
  setLength(vertex,2);

  vertex[0] := coords[0];
  vertex[1] := coords[1];
  vertex[2] := coords[2];

  dataOut := vertex;
end;

{ Make four display lists,
 each with a different tessellated object.}
procedure makeNewLists ();
var
   i : integer;
begin

  gluTessProperty(tobj, GLU_TESS_WINDING_RULE,
                currentWinding );

  glNewList(list, GL_COMPILE);
    gluTessBeginPolygon(tobj);
      gluTessBeginContour(tobj);
        for i := 0 to 3 do
          gluTessVertex(tobj, @rects[i], @rects[i]);
      gluTessEndContour(tobj);

      gluTessBeginContour(tobj);
        for i := 4 to 7 do
          gluTessVertex(tobj, @rects[i], @rects[i]);
      gluTessEndContour(tobj);

      gluTessBeginContour(tobj);
        for i := 8 to  11 do
          gluTessVertex(tobj, @rects[i], @rects[i]);
      gluTessEndContour(tobj);
    gluTessEndPolygon(tobj);
  glEndList();

  glNewList(list+1, GL_COMPILE);
      gluTessBeginPolygon(tobj);
        gluTessBeginContour(tobj);
          for i := 0 to 3 do
            gluTessVertex(tobj, @rects[i], @rects[i]);
        gluTessEndContour(tobj);

        gluTessBeginContour(tobj);
          for i := 7 downto 4 do
            gluTessVertex(tobj, @rects[i], @rects[i]);
        gluTessEndContour(tobj);

        gluTessBeginContour(tobj);
          for i := 11 downto 8 do
              gluTessVertex(tobj, @rects[i], @rects[i]);
        gluTessEndContour(tobj);
```

```
    gluTessEndPolygon(tobj);
  glEndList();

  glNewList(list+2, GL_COMPILE);
    gluTessBeginPolygon(tobj);
      gluTessBeginContour(tobj);
        for i := 0 to 15 do
          gluTessVertex(tobj, @spiral[i], @spiral[i]);
      gluTessEndContour(tobj);
    gluTessEndPolygon(tobj);
  glEndList();

  glNewList(list+3, GL_COMPILE);
    gluTessBeginPolygon(tobj);
      gluTessBeginContour(tobj);
        for i := 0 to 3 do
          gluTessVertex(tobj, @quad1[i], @quad1[i]);
      gluTessEndContour(tobj);

      gluTessBeginContour(tobj);
         for i := 0 to 3 do
           gluTessVertex(tobj, @quad2[i], @quad2[i]);
      gluTessEndContour(tobj);

      gluTessBeginContour(tobj);
        for i := 0 to 2 do
          gluTessVertex(tobj, @tri[i], @tri[i]);
      gluTessEndContour(tobj);
    gluTessEndPolygon(tobj);
  glEndList();

end;

procedure display ();
begin

  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 1.0);

  glPushMatrix();
    glCallList(list);
    glTranslatef(0.0, 500.0, 0.0);
    glCallList(list+1);
    glTranslatef(500.0, -500.0, 0.0);
    glCallList(list+2);
    glTranslatef(0.0, 500.0, 0.0);
    glCallList(list+3);
  glPopMatrix();

  SwapBuffers(f_Hdc);
end;

procedure initGL();
begin

  glClearColor(0.0, 0.0, 0.0, 0.0);
  glShadeModel(GL_FLAT);
```

```
  tobj := gluNewTess();


  gluTessCallback(tobj, GLU_TESS_BEGIN, @beginCallback);
  gluTessCallback(tobj, GLU_TESS_END, @endCallback);
  gluTessCallback(tobj, GLU_TESS_VERTEX, @tessVertex);
  gluTessCallback(tobj, GLU_TESS_COMBINE, @combineCallback);
//  gluTessCallback(tobj, GLU_TESS_ERROR,@errorCallback);

  list := glGenLists(4);

  makeNewLists();

end;

procedure reshape( w, h : integer);
begin
  glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   if (w <= h) then
     gluOrtho2D(0.0, 1000.0, 0.0, 1000.0 * h/w)
   else
     gluOrtho2D(0.0, 1000.0 * w/h, 0.0, 1000.0);

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
end;

procedure keyboard();
begin

 if (currentWinding = GLU_TESS_WINDING_ODD) then
  begin
       currentWinding := GLU_TESS_WINDING_NONZERO;
       form1.Caption :=  'GLU_TESS_WINDING_NONZERO';
   end;

 if (currentWinding = GLU_TESS_WINDING_NONZERO) then
  begin
      currentWinding := GLU_TESS_WINDING_POSITIVE;
      form1.Caption :=  'GLU_TESS_WINDING_POSITIVE';
  end;

 if (currentWinding = GLU_TESS_WINDING_POSITIVE) then
  begin
     currentWinding := GLU_TESS_WINDING_NEGATIVE;
     form1.Caption :=  'GLU_TESS_WINDING_NEGATIVE';
  end;


 if (currentWinding = GLU_TESS_WINDING_NEGATIVE) then
  begin
     currentWinding := GLU_TESS_WINDING_ABS_GEQ_TWO;
     form1.Caption :=  'GLU_TESS_WINDING_ABS_GEQ_TWO';
  end;
```

```
 if (currentWinding = GLU_TESS_WINDING_ABS_GEQ_TWO) then
  begin
     currentWinding := GLU_TESS_WINDING_ODD;
     form1.Caption :=  'GLU_TESS_WINDING_ODD';
  end;

  makeNewLists();

end;

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

procedure TForm1.FormResize(Sender: TObject);
begin
   wglMakeCurrent(f_Hdc,hrc); //activate the RC
     reshape( Width, 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.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  gluDeleteTess(tobj);
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  keyboard;
  InvalidateRect(Handle, nil, False);// Draw the scene.
end;

end.
```

برنامه ی دوم فصل :

```
unit ch21;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs ,OpenGL,SPF, StdCtrls ;

type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormResize(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormKeyPress(Sender: TObject; var Key: Char);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
  private
    { Private declarations }
     procedure Idle(Sender: TObject; var Done: Boolean);
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
type  displayLists=(
  RESERVED, BODY_SIDE, BODY_EDGE, BODY_WHOLE, ARM_SIDE,
  ARM_EDGE, ARM_WHOLE,  LEG_SIDE, LEG_EDGE, LEG_WHOLE,
  EYE_SIDE, EYE_EDGE, EYE_WHOLE );

 T3dArray = array [0..2] of GLfloat;
 TVertexf = T3DArray;  // An array of 3 float values.

var
 angle : GLfloat = 20;   //* in degrees */
 angle2 : GLfloat = 30;   //* in degrees */
 doubleBuffer :GLboolean = GL_TRUE;
 iconic :GLboolean= GL_FALSE;
 keepAspect :GLboolean = GL_FALSE;
 startx, starty : LongInt;
 moving : GLboolean = GL_FALSE;
 W :LongInt= 300;
 H :LongInt= 300 ;
 //* Initially, allow the artifacts. */
 useStencil : GLboolean = GL_FALSE;
 bodyWidth :GLdouble= 3.0;
 jump : single = 0.0;
//* *INDENT-OFF* */
  body : array of  TVertexf;
```

```
  arm : array of  TVertexf;
  leg : array of  TVertexf;
  eye : array of  TVertexf;
  lightZeroPosition : array[0..3] of GLfloat = (10.0, 14.0, 10.0, 1.0);
  lightZeroColor : array[0..3] of GLfloat= (0.8, 1.0, 0.8, 1.0);
  lightOnePosition : array[0..3] of GLfloat = (-1.0, 1.0, 1.0, 0.0);
  lightOneColor : array[0..3] of GLfloat = (0.6, 0.3, 0.2, 1.0);
  skinColor : array[0..3] of GLfloat = (0.1, 1.0, 0.1, 1.0);
  eyeColor : array[0..3] of GLfloat = (1.0, 0.2, 0.2, 1.0);
  lightZeroSwitch :GLboolean = GL_TRUE;
  lightOneSwitch :GLboolean= GL_TRUE;
  ElapsedTime, DemoStart, LastTime : DWord;
  wire: Boolean = FALSE;

  f_Hdc : longInt;

function Vertexf(x, y ,z: single): TVertexf;
begin
  // Utility function to build a vector:
  Vertexf[0] := x;
  Vertexf[1] := y;
  Vertexf[2] := z;
end;

procedure tessBegin(atype: GLenum); stdcall;
begin
  { This callback is invoked when the tesselator
    creates a new group of triangles. The atype
    parameter can be either GL_TRIANGLES, GL_TRIANGLE_FAN
    or GL_TRIANGLE_STRIP. }
  glBegin(atype);
end;

procedure tessEnd; stdcall;
begin
  // This callback is invoked when a group of triangles needs to be ended.
  glEnd;
end;

procedure tessVertex(data: Pointer); stdcall;
begin
  { This callback is invoked for each vertex
    in the triangulation. The data
    parameter is a pointer we passed to
    gluTessVertex(). (see below)
    In this case, it points to a float-precision 2D vector. }
  glVertex3fv(data);
end;

procedure InitVariables();
begin
 setlength(body,22);
 body[0]:=Vertexf(0,3,0);      body[1]:=Vertexf(1,1,0);
 body[2]:=Vertexf(5,1,0);      body[3]:=Vertexf(8,4,0);
 body[4]:=Vertexf(10,4,0);     body[5]:=Vertexf(11,5,0);
 body[6]:=Vertexf(11,11.5,0);  body[7]:=Vertexf(13,12,0);
 body[8]:=Vertexf(13,13,0);    body[9]:=Vertexf(10,13.5,0);
 body[10]:=Vertexf(13,14,0);   body[11]:=Vertexf(13,15,0);
```

```
 body[12]:=Vertexf(11,16,0);   body[13]:=Vertexf(8,16,0);
 body[14]:=Vertexf(7,15,0);    body[15]:=Vertexf(7,13,0);
 body[16]:=Vertexf(8,12,0);    body[17]:=Vertexf(7,11,0);
 body[18]:=Vertexf(6,6,0);     body[19]:=Vertexf(4,3,0);
 body[20]:=Vertexf(3,2,0);     body[21]:=Vertexf(1,2,0);

 setlength(arm,16);
 arm[0]:=Vertexf(8,10,0);     arm[1]:=Vertexf(9,9,0);
 arm[2]:=Vertexf(10,9,0);     arm[3]:=Vertexf(13,8,0);
 arm[4]:=Vertexf(14,9,0);     arm[5]:=Vertexf(16,9,0);
 arm[6]:=Vertexf(15,9.5,0);   arm[7]:=Vertexf(16,10,0);
 arm[8]:=Vertexf(15,10,0);    arm[9]:=Vertexf(15.5,11,0);
 arm[10]:=Vertexf(14.5,10,0); arm[11]:=Vertexf(14,11,0);
 arm[12]:=Vertexf(14,10,0);   arm[13]:=Vertexf(13,9,0);
 arm[14]:=Vertexf(11,11,0);   arm[15]:=Vertexf(9,11,0);

 setlength(leg,14);
 leg[0]:=Vertexf(8,6,0);     leg[1]:=Vertexf(8,4,0);
 leg[2]:=Vertexf(9,3,0);     leg[3]:=Vertexf(9,2,0);
 leg[4]:=Vertexf(8,1,0);     leg[5]:=Vertexf(8,0.5,0);
 leg[6]:=Vertexf(9,0,0);     leg[7]:=Vertexf(12,0,0);
 leg[8]:=Vertexf(10,1,0);    leg[9]:=Vertexf(10,2,0);
 leg[10]:=Vertexf(12,4,0);   leg[11]:=Vertexf(11,6,0);
 leg[12]:=Vertexf(10,7,0);   leg[13]:=Vertexf(9,7,0);

 setlength(eye,6);
 eye[0]:=Vertexf(8.75,15,0);     eye[1]:=Vertexf(9,14.7,0);
 eye[2]:=Vertexf(9.6,14.7,0);    eye[3]:=Vertexf(10.1,15,0);
 eye[4]:=Vertexf(9.6,15.25,0);   eye[5]:=Vertexf(9,15.25,0);
end;

procedure extrudeSolidFromPolygon(
    data : array of Tvertexf  ;  dataSize :GLuint;
   thickness :GLdouble; side, edge,  whole :GLuint);
var
  tobj : GLUtesselator;
  vertex :array[0..2] of  GLdouble;
  i,count :LongInt;

begin
  tobj :=nil;
  count := dataSize ;

 if (tobj = nil) then
   begin
    tobj := gluNewTess();  { create and initialize a GLU
                        polygon  tesselation object }

    gluTessCallback(tobj, GLU_TESS_BEGIN, @tessBegin);
    gluTessCallback(tobj, GLU_TESS_END, @tessEnd);
    gluTessCallback(tobj, GLU_TESS_VERTEX, @tessVertex);
   end;

 glNewList(side, GL_COMPILE);
   glShadeModel(GL_SMOOTH);  { smooth minimizes seeing tessellation }
    gluTessBeginPolygon(tobj);
     gluTessBeginContour (tObj);
       for i := 0 to count do
```

```
        begin
         vertex[0] := data[i,0];
         vertex[1] := data[i,1];
         vertex[2] := 0;
         form1.Caption  := 'OpenGL Reflected Dinosaur!'
                        + floattostr(vertex[2]);
         gluTessVertex(tobj, @vertex, @data[i]);
        end;
      gluTessEndContour (tObj);
     gluTessEndPolygon(tobj);
  glEndList();

  glNewList(edge, GL_COMPILE);
// flat shade keeps angular hands from being "smoothed"
  glShadeModel(GL_FLAT);
  glBegin(GL_QUAD_STRIP);
  for i := 0 to count do
  begin
    // mod function handles closing the edge
    glVertex3f(data[i mod count,0], data[i mod count,1], 0.0);
    glVertex3f(data[i mod count,0], data[i mod count,1], thickness);
  end;

  glEnd();
  glEndList();

  glNewList(whole, GL_COMPILE);
  glFrontFace(GL_CW);
  glCallList(edge);
  glNormal3f(0.0, 0.0, -1.0);  //* constant normal for side */
  glCallList(side);
  glPushMatrix();
    glTranslatef(0.0, 0.0, thickness);
    glFrontFace(GL_CCW);
    glNormal3f(0.0, 0.0, 1.0);  // opposite normal for other side */
    glCallList(side);
  glPopMatrix();
  glEndList();
end;

procedure makeDinosaur();
begin
  extrudeSolidFromPolygon(body, 21, bodyWidth,
    ord(BODY_SIDE), ord(BODY_EDGE), ord(BODY_WHOLE));
  extrudeSolidFromPolygon(arm, 15, bodyWidth / 4,
    ord(ARM_SIDE), ord(ARM_EDGE),ord(ARM_WHOLE));
  extrudeSolidFromPolygon(leg, 13, bodyWidth / 2,
    ord(LEG_SIDE), ord(LEG_EDGE), ord(LEG_WHOLE));
  extrudeSolidFromPolygon(eye, 5, bodyWidth + 0.2,
    ord(EYE_SIDE), ord(EYE_EDGE), ord(EYE_WHOLE));
end;

procedure drawDinosaur();
begin
  glPushMatrix();
    glTranslatef(0.0, jump, 0.0);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, @skinColor);
    glCallList(ord(BODY_WHOLE));
```

```
    glPushMatrix();
     glTranslatef(0.0, 0.0, bodyWidth);
     glCallList(ord(ARM_WHOLE));
     glCallList(ord(LEG_WHOLE));
     glTranslatef(0.0, 0.0, -bodyWidth -(bodyWidth / 4));
     glCallList(ord(ARM_WHOLE));
     glTranslatef(0.0, 0.0, -bodyWidth / 4);
     glCallList(ord(LEG_WHOLE));
     glTranslatef(0.0, 0.0, (bodyWidth / 2) - 0.1);
     glMaterialfv(GL_FRONT, GL_DIFFUSE, @eyeColor);
     glCallList(ord(EYE_WHOLE));
    glPopMatrix();
  glPopMatrix();
end;

procedure drawFloor();
begin
  glDisable(GL_LIGHTING);
  glBegin(GL_QUADS);
    glVertex3f(-18.0, 0.0, 27.0);
    glVertex3f(27.0, 0.0, 27.0);
    glVertex3f(27.0, 0.0, -18.0);
    glVertex3f(-18.0, 0.0, -18.0);
  glEnd();
  glEnable(GL_LIGHTING);
end;

procedure redraw();
begin
  if (useStencil)then
     // Clear; default stencil clears to zero. */
    glClear(GL_COLOR_BUFFER_BIT or
          GL_DEPTH_BUFFER_BIT or GL_STENCIL_BUFFER_BIT)
   else
     // Not using stencil; just clear color and depth. */
     glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

  glPushMatrix();
    // Perform scene rotations based on user mouse input. */
    glRotatef(angle2, 1.0, 0.0, 0.0);
    glRotatef(angle, 0.0, 1.0, 0.0);
    // Translate the dinosaur to be at (0,0,0). */
    glTranslatef(-8, -8, -bodyWidth / 2);

    glLightfv(GL_LIGHT0, GL_POSITION, @lightZeroPosition);
    glLightfv(GL_LIGHT1, GL_POSITION, @lightOnePosition);

    if (useStencil) then
    begin
      { We can eliminate the visual "artifact" of seeing the "flipped"
        dinosaur underneath the floor by using stencil.  The idea is
        draw the floor without color or depth update but so that
        a stencil value of one is where the floor will be.  Later when
        rendering the dinosaur reflection, we will only update pixels
        with a stencil value of 1 to make sure the reflection only
        lives on the floor, not below the floor. }
      // Don't update color or depth. */
      glDisable(GL_DEPTH_TEST);
```

```
  glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);

  // Draw 1 into the stencil buffer. */
  glEnable(GL_STENCIL_TEST);
  glStencilOp(GL_REPLACE, GL_REPLACE, GL_REPLACE);
  glStencilFunc(GL_ALWAYS, 1, $ffffffff);

  // Now render floor; floor pixels just get their stencil set to 1. */
  drawFloor();

  // Re-enable update of color and depth. */
  glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
  glEnable(GL_DEPTH_TEST);

  // Now, only render where stencil is set to 1. */
  glStencilFunc(GL_EQUAL, 1, $ffffffff);  // draw if ==1 */
  glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
end;
glPushMatrix();

  { The critical reflection step: Reflect dinosaur through the floor
    (the Y=0 plane) to make a relection. }
  glScalef(1.0, -1.0, 1.0);

  // Position lights now in reflected space. */
  glLightfv(GL_LIGHT0, GL_POSITION, @lightZeroPosition);
  glLightfv(GL_LIGHT1, GL_POSITION, @lightOnePosition);
  { XXX Ugh, unfortunately the back face culling reverses when we reflect
    the dinosaur.  Easy solution is just disable back face culling for
    rendering the reflection.  Also, the normals for lighting get screwed
    up by the scale; enabled normalize to ensure normals are still
    properly normalized despite the scaling.  We could have fixed the
    dinosaur rendering code, but this is more expedient. }
  glEnable(GL_NORMALIZE);
  glCullFace(GL_FRONT);

  // Draw the reflected dinosaur. */
  drawDinosaur();

  // Disable noramlize again and re-enable back face culling. */
  glDisable(GL_NORMALIZE);
  glCullFace(GL_BACK);
glPopMatrix();

// Restore light positions on returned from reflected space. */
glLightfv(GL_LIGHT0, GL_POSITION, @lightZeroPosition);
glLightfv(GL_LIGHT1, GL_POSITION, @lightOnePosition);

if (useStencil) then
  { Don't want to be using stenciling for drawing the actual dinosaur
    (not its reflection) and the floor. }
  glDisable(GL_STENCIL_TEST);

{ Back face culling will get used to only draw either the top or the
   bottom floor.  This let's us get a floor with two distinct
   appearances.  The top floor surface is reflective and kind of red.
   The bottom floor surface is not reflective and blue. }
// Draw "top" of floor.  Use blending to blend in reflection. */
```

```
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glColor4f(0.7, 0.0, 0.0, 0.3);
    drawFloor();
    glDisable(GL_BLEND);

    // Draw "bottom" of floor in blue. */
    glFrontFace(GL_CW);  // Switch face orientation. */
    glColor4f(0.1, 0.1, 0.7, 1.0);
    drawFloor();
    glFrontFace(GL_CCW);

    // Draw "actual" dinosaur, not its reflection. */
    drawDinosaur();

  glPopMatrix();
  SwapBuffers(f_Hdc);
end;

procedure controlLights(value : LongInt);
begin
   if value=1 then
   begin
    lightZeroSwitch := not(lightZeroSwitch);
    if (lightZeroSwitch) then
      glEnable(GL_LIGHT0)
     else
      glDisable(GL_LIGHT0);
   end;
   if value=2 then
   begin
    lightOneSwitch := not(lightOneSwitch);
    if (lightOneSwitch) then
      glEnable(GL_LIGHT1)
     else
      glDisable(GL_LIGHT1);
   end;
   if value=3 then
     useStencil := not(useStencil);
end;

procedure initGL();
begin
  InitVariables;
  redraw;
  makeDinosaur();
  glEnable(GL_CULL_FACE);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_LIGHTING);
  glMatrixMode(GL_PROJECTION);
  glEnable(GL_NORMALIZE);
  gluPerspective( { field of view in degree } 40.0,
   { aspect ratio } 1.0,
    { Z near } 1.0, { Z far } 80.0);
  glMatrixMode(GL_MODELVIEW);
  gluLookAt(0.0, 0.0, 40.0,  { eye is at (0,0,30) }
    0.0, 0.0, 0.0,      { center is at (0,0,0) }
    0.0, 1.0, 0.0);      { up is in postivie Y direction }
```

```
  glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, 1);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, @lightZeroColor);
  glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.1);
  glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.05);
  glLightfv(GL_LIGHT1, GL_DIFFUSE, @lightOneColor);
  glEnable(GL_LIGHT0);
  glLightfv(GL_LIGHT0, GL_POSITION, @lightZeroPosition);
  glLightfv(GL_LIGHT1, GL_POSITION, @lightOnePosition);
  glViewport (0, 0, form1.Width, form1.Height);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 f_Hdc := GetDC( form1.handle );
 SetDCPixelFormat(f_Hdc,16,16);// Create a rendering context.
 initGL();
 DemoStart :=GetTickCount;
 Application.OnIdle := Idle;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button=mbLeft then
  begin
    moving := GL_TRUE;
    startx := x;
    starty := y;
   InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
  end;
end;

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

procedure TForm1.FormResize(Sender: TObject);
begin
   wglMakeCurrent(f_Hdc,hrc); //activate the RC
   glViewport( 0, 0, Width, Height);
   InvalidateRect(Handle, nil, False);// DrawGLScene;  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
  reDraw();                  // Draw the scene
end;

procedure TForm1.Idle(Sender: TObject; var Done: Boolean);
begin
```

```
if moving=GL_FALSE then
 begin
  LastTime :=ElapsedTime;
  ElapsedTime :=GetTickCount() - DemoStart;  // Calculate Elapsed Time
  // Average it out for smoother movement
  ElapsedTime :=(LastTime + ElapsedTime) DIV 2;
  jump :=  6.0 * abs(sin(ElapsedTime/8));
  InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
 end;
end;

procedure TForm1.FormKeyPress(Sender: TObject; var Key: Char);
begin
  if Key = '4' then wire := not wire;
  if wire then glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
  else glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
  controlLights(strtoint(key));
  InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  if (moving) then
  begin
    angle := angle + (x - startx);
    angle2 := angle2 + (y - starty);
    startx := x;
    starty := y;
   InvalidateRect(Handle, nil, False);// DrawGLScene;  Draw the scene.
  end;
end;

end.
```