وزارت علوم ، تحقیقات و فن آوری

دانشگاه جامع علمی –کاربردی

مرکز آموزش عالی – کاربردی جهاد دانشگاهی

شعبه همدان

عنوان پروژه :

معرفیOenGLو APIگرافیکی درویندوز

استاد راهنما :

جناب آقای مهندس محمدی

نام دانشجو :

سیده رقیه موسوی شکیب

شماره دانشجویی :

8469191055

رشته تحصیلی:

کامپیوتر

عنوان پروژه :

معرفی openGL و API

گرافیکی در ویندوز

تقدیر و تشکر

سپاس خدای بزرگی را که با قلم توانای خویش مشتی خاک را توانی دیگر بخشید تا بتواند در این صحنهٔ زندگی بیندیشد وبا افکار خویش تصاویر زندگی را نقش بندد و استعدادها وتوانائیش را به خدمت گیرد. پس سپاس خدایی را سزاست که با دادن قدرت اندیشه لحظات را برایمان حلاوتی دیگر بخشید. پس ای معبودم تورا بخاطر دادن نعمات می‌ستایم وجوارحم را برای خدمت در راه تو به‌کار می‌گیرم وبعد از آن از استاد راهنما جناب آقای محمدی که درمراحل مختلف تحقیق با راهنماییهای خویش راهگشای من بوده‌اند وهمچنین از همه کسانی که در انجام این تحقیق به نوعی به ما کمک کرده‌اند سپاسگزاری نموده واز ایزد منان خواهان موفقیتهای روزافزون آنها هستم

]

چکیده:

OpenGLچیست :

OpenGL دقیقا به عنوان یک "رابط نرم افزاری برای سخت افزار گرافیکی" تعریف شده است. OpenGL در ماهیت خود یک کتابخانه مدل سازی و گرافیک سه بعدی میباشد که بسیار سریع و قابل انتقال است. با استفاده از OpenGL شما میتوانید تصاویر سه بعدی زیبا و جذابی طراحی کنید. بزرگترین فایده استفاده از OpenGL اینست که فوق العاده از یک ردیاب نور (ray tracer) سریعتر است. OpenGL از الگوریتمهایی استفاده میکند که توسط شرکت Silicon Graphics توسعه یافته و بهینه شده است.SGI یک رهبر تائید شده در دنیای گرافیک کامپیوتری و انیمیشن میباشد.

OpenGL یک زبان برنامه نویسی مانند c یا c++ نیست. OpenGL بیشتر شبیه کتابخانه زمان اجرای C می باشدکه یک سری توابع از پیش بسته بندی شده را تدارک دیده. در عمل چیزی به نام برنامه OpenGL وجود ندارد. وقتی ما میگوییم این یک برنامه OpenGL است یعنی در ساختار این برنامه از OpenGL به عنوان API گرافیکی اش استفاده کرده است همانطور که ما از توابع APIویندوز استفاده میکنیم تا بتوانیم به فایلها و امکانات شبکه ای و غیره ویندوز دسترسی پیدا کنیم. همین طور هم ما از توابع OpenGL استفاده میکنیم تا بتوانیم گرافیک سه بعدی بلادرنگ طراحی کنیم.

IRIS GL در ابتدا یک کتابخانه دوبعدی بود که پیشرفت کرد و به OpenGL تبدیل شد. در حقیقت OpenGL نتیجه تلاشی بود که شرکت SGI برای اصلاح و بهبود IRIS GL کرد.

OpenGL استاندارد به سازندگان شخصی سخت افزار گرافیکی این اجازه را میدهد که قابلیت های افزودنی خودشان را با عنوان Extension تهیه کنند که ممکن است بعضی از محدودیت های توابع OpenGL را کم کند یا راحت تر کند و یا اینکه قابلیت های جدیدی را به آن بیفزاید. Extension ها از توابع و ثابت های جدیدی ساخته شده اند که قابلیت های جدیدی را به OpenGL استاندارد می

افزایند.هر سازنده سخت افزار گرافیکی یک اختصار الفبایی مخصوص به خود برای نامگذاری **Extension** های خودش دارد. برای مثال شرکت **NVIDIA** از حروف اختصاری **NV** برای نامگذاری **Extension** هایی که درست میکنند استفاده میکنند.**OpenGL 2.0** توسط شرکت **D Labs3** ایجاد شد که نگران راکد ماندن و نداشتن یک مدیریت قوی برای**OpenGL** بود. . این شرکت قابلیت های جدیدی را به **OpenGL** اضافه کرد که پر اهمییت ترین آنها زبان سایه زنی **GLSL** بود.این قابلیت برنامه نویسان را قادر میساخت که خطوط لوله تکه و راس تابع ثابت **OpenGL** را با سایه زن های نوشته شده در زبانی شبیه به **C** تعویض کنند

منبع : کتاب OpenGL SuperBible 3rd Edition

فهرست:

مقدمه : آشنایی با راه و رسم و اهداف

ذخیره کردن و اعاده (پس دادن یا برگرداندن) حالت ها

مشخصات API و انواع داده در OpenGL :

قواعد نامگذاری توابع در openGL

استقلال از وابستگی به هر گونه پلتفرم خاص

استفاده از GLUT :

openGL یک API است نه یک زبان برنامه نویسی

کتابخانه ها و فایلهای سر آیند

OpenGL چگونه کار میکند

پیاده سازی عمومی

پیاده سازی سخت افزاری

The Pipeline

کاربردهاس مشترک گرافیک سه بعدی

Real-time 3D

گرافیک سه بعدی غیر همزمان .

مختصری درباره افکت های سه بعدی

پرسپکتیو :

رنگ و سایه

نور و سایه ها

نگاشت بافت

آمیختگی و شفافیت

ضد دندانه ای

مقدمه ای بر گرافیک کامپیوتری

API وارد میشود

ایجاد یونیت تشریفاتی برای برپایی فرمت نقطه ای

ترسیم یک مثلث ساده با استفاده ازopenGL

آشنایی با DirectX و OpenGL

APIگرافیکی چیست؟

openGL 2.0

نتیجه‌گیری

منابع ومراجع

مقدمه : آشنایی با راه و رسم و اهداف

OpenGL یک ماشین حالت است :

یک ماشین حالت یک مدل انتزاعی از مجموعه ای از متغیرهای حالت است که میتوانند ارزش های گوناگونی داشته باشند. روشن بودن **"on"** و یا خاموش بودن **"off"** و الی آخر. این مقدور نیست که ما هر موقع بخواهیم چیزی را در **OpenGL** ترسیم کنیم تمام متغیر های حالت را تعیین و مقداردهی نماییم. در عوض **OpenGL** از یک مدل حالت و یا ماشین حالت استفاده میکند تا رو تمام متغیر های حالت **OpenGL** را پیگیری نماید. هنگامی که یک متغیر حالت مقدار دهی شد همینطور برقرار میماند تا زمانی که تابع دیگری آن را تغییر دهد. در عمل بسیاری از حالت ها روشن و یا خاموش هستند. برای مثال نورپردازی یا در حالت روشن قرار دارد و یا خاموش است. وقتی نورپردازی در حالت خاموش قرار دارد ترسیمات هندسی ما بدون هیچگونه محاسبات نورپردازی اعمال شده بر روی رنگ اشیای هندسی انجام میگیرد. اما کلیه ترسیمات هندسی پس از روشن کردن حالت نورپردازی به همراه اعمال محاسبات نورپردازی بر روی صفحه ترسیم میشود.

برای روشن کردن این متغیر های حالت شما باید از تابع نمونه زیر کمک بگیرید :

```
void glEnable(GLenum capability);    // this way
```

و برای خاموش کردن از تابع زیر استفاده میکنید :

```
void glDisable(GLenum capability);    // this way
```

اینها شکل کلی این توابع بود. اما در مورد نورپردازی برای مثال شما میتوانید حالت نورپردازی را با استفاده از این حالت روشن (فعال) کنید :

```
glEnable(GL_LIGHTING);     // this way
```

و با کمک این تابع آن را خاموش (غیر فعال) میکنید :

```
glDisable(GL_LIGHTING);    // this way
```

برای اینکه امتحان کنید که یک متغیر حالت فعال است یا نه **OpenGL** یک مکانیسم راحت را تدارک دیده است :

```
capability); // this way Glboolean glIsEnabled(GLenum
```

البته همانطور که قبلا هم گفتم در نهایت تمام متغیر های حالت یا روشن هستند و یا خامو شند.بسیاری از این توابع میتوانند این متغیر های حالت را ارزش دهی کنند تا وقتی که آنها عوض شوند.شما میتوانید در هر لحظه ای که بخواهید ارزش حالت ها را چک کنید. یک مجموعه از توابع تحقیق کننده **"query functions"** به شما اجازه میدهند تا در هر لحظه که بخواهید ارزش هرکدام از متغیر های بولی یا صحیح یا ممیز شناور و یا ممیز شناور با دقت مضاعف را بفهمید. این چهار تابع بدین صورت نمونه سازی
شده اند :

```
this way  // ;(params* GLboolean ,pname void glGetBooleanv(GLenum
this way  // ;(params* GLdouble ,pname void glGetDoublev(GLenum
this way  // ;(params* GLfloat ,pname void glGetFloatv(GLenum
this way  // ;(params* GLint ,pname void glGetIntegerv(GLenum
```

هر تابع یک ارزش واحد و یکتا و یا آرایه ای کامل از ارزشها را برمیگرداند. نگهداری نتایج در آدرسهایی که
شما تدارک دیده اید. شما باید ممنون سادگی و قدرت ماشین حالت **OpenGL** باشید.

ذخیره کردن و اعاده (پس دادن یا برگرداندن) حالت ها :

OpenGL همچنین یک مکانیزم راحت و مناسب برای ذخیره کردن محدوده کاملی از ارزشهای حالت و بازگرداندن مجدد آنها دارد. "پشته" یا **"stack"** یک ساختمان داده ای مناسب است که اجازه میدهد داده ها به پشته وارد شوند (ذخیره کردن) و بعد از پشته بیرون بپرند تا بازیابی شوند. آیتم هایی که قبلا درپشته هل داده شده بودند (ذخیره شده بودند) با دستوری متضاد بازیافت میشوند. ما این را یک ساختمان داده ای بنام "بترتیب عکس ورود" و یا **"Last in First Out"** مینامیم. دقیقا مثل این میماند که ما بگوییم "هی لطفا این را ذخیره کن" و یا **"push it on the stack"** و مدت زمانی بعد بگوییم "چیزی که قبلا ذخیره کرده ام را بده" یا **"pop it off the stack"** .

در مقالات آتی شما خواهید دید که موضوع پشته **"stack"** نقش مهمی را در بکارگیری ماتریسها بازی میکند. یک ارزش حالت تنها در **OpenGL** یا یک محدوده کامل ار ارزشهای حالت مربوطه میتوانند در یک
پشته صفت و با کمک دستور زیر قرار بگیرند.

this way // ;(mask void glPushAttrib(GLbitfield

متقابلا ارزشها با کمک دستور زیر قابل بازیافت هستند :

this way // ;(mask void glPopAttrib(GLbitfield

منبع : کتاب OpenGL SuperBible 3rd Edition

: مشخصات API :

OpenGL توسط عده ای از مردم باهوش که تجربه زیادی در طراحی **API** های گرافیکی داشتند ایجاد شد. آنها قواعدی برای نامگذاری توابع و شیوه اعلان متغیر تعیین کردند. این **API** برای سازندگان سخت افزار گرافیکی به راحتی قابل توسعه است و به همین دلیل **OpenGL** سعی میکند از هرگونه سیاست جداگانه ای احتراز کند.

انواع داده در **openGL** :

برای اینکه انتقال کدهای نوشته شده در **OpenGL** به راحتی از یک پلتفرم به پلتفرم دیگر قابل انتقال باشند **OpenGL** انواع داده مخصوص به خودش را تعریف کرده است. هر محیط برنامه نویسی یا کامپایلری بهر حال قواعد خاصی برای اندازه و نوع حافظه متغیر های مختلف سی **"C"** دارد. اما با استفاده از انواع داده های **OpenGL** یعنی **OpenGL Data Types** شما میتوانید کد خودتان را در برابر این نوع تغییرات عایق کاری کنید.
جدول **1-2** انواع داده **OpenGL** را لیست کرده است و همچنین انواع داده متناظر آنها در **C** تحت محیط ویندوز **32** بیتی را نشان داده است.

OpenGL Data Type	Internal Representation	Defined as C Type	C Literal Suffix
GLbyte	8-bit integer	signed char	b
GLshort	16-bit integer	short	s
GLint, GLsizei	32-bit integer	long	l
GLfloat, GLclampf	32-bit floating point	float	f
GLdouble, GLclampd	64-bit floating point	double	d
GLubyte, GLboolean	8-bit unsigned integer	unsigned char	ub
GLushort	16-bit unsigned integer	unsigned short	us
GLuint, GLenum, GLbitfield	32-bit unsigned integer	unsigned long	ui

تمام انواع داده با یک **GL** شروع میشوند برای ایمکه مشخص باشد از انواع داده **OpenGL** میباشد و خیلی از آنها از انواع متناظرشان در زبان **C** منتج شده اند. مانند **... , byte , short , int , float** .بعضی در ابتدایشان یک حرف **u** دارند که بیانگر اینست که از نوع بدون علامت **"unsigned"** میباشد. مانند **ubyte** که نماینده **unsigned byte** است.

برای بعضی از مصارف روشن ترین و مشخص ترین نام تعیین شده است مانند نوع **size** که مشخص کننده مقدار طول یا عمق میباشد. بطور مثال **GLsizei** یکی از توابع **OpenGL** است که نشان دهنده این است که این تابع یک متغیر از نوع صحیح را به عنوان آرگومان میگیرد.

نقش **Clamp** یک تذکر جزیی است که نشان دهد مقدار باید در محدوده **0.0 – 1.0** باشد. متغیر های **GLboolean** بدین منظور استفاده میشوند تا مقادیر صحیح یا غلط را نشان دهند.

اشاره گرها و آرایه هاهیچگونه معادل خاصی در **OpenGL** ندارند. یک آرایه از ده عنصر **GLshort** بطور ساده به این صورت تعریف میشود :

```
GLshort shorts[10];   // this way
```

و آرایه ای از ده اشاره گر به متغیر های **GLdouble** بدین صورت تعریف میشود :

```
GLdouble *doubles[10];   //this way
```

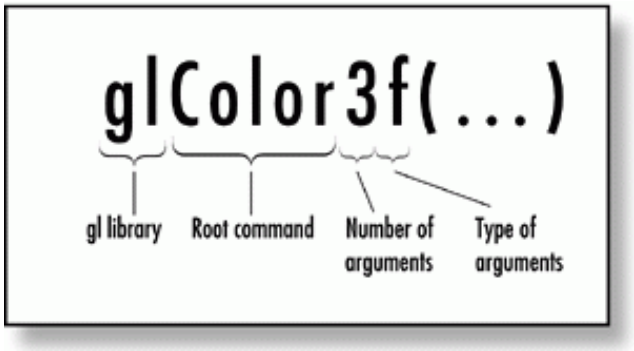
بعضی از انواع شی اشاره گر دیگر برای **NURBS** و چهاگوش ها استفاده میشوند که به توضیحات بیشتری احتیاج دارند و در مقالات بعدی پوشش داده خواهند شد.

قواعد نامگذاری توابع در **openGL** :

بسیاری از توابع **OpenGL** از یک قرارداد تبعیت میکنند تا به شما بفهمانند این تابع از کدام کتابخانه میباشد و در بعضی مواقع اینکه این توابع چه تعداد و چه نوعی از داده ها را به عنوان آرگومان میپذیرد. تمام توابع ریشه ای دارند که دستور متناظر **OpenGL** آنها را نشان میدهد. برای مثال تابع **glColor3f** ریشه اش **color** است و پیشوند **gl** نشان دهنده کتابخانه **gl** میباشد. پسوند **3f** نشان دهنده این است که این تابع سه عدد آرگومان از نوع ممیز شناور **(floating – point)** میگیرد. تمام توابع **OpenGL** قالب زیر را دارند :

<Library prefix><Root command><Optional argument count><Optional argument type>

تصویر **2-4** قسمتهای سازنده یک تابع را در **openGL** نشان میدهد. این تابع نمونه با پسوند **3f** سه عدد آرگومان ممیز شناور میگیرد. نوع دیگر این تابع **glColor3i** سه عدد صحیح را به عنوان آرگومان میپذیرد. ونوع **glColor3d** سه عدد از نوع **double** را به عنوان آرگومان میگیرد و همینطور الی آخر.



این شیوه اضافه کردن تعداد و نوع آرگومانها به انتهای توابع **OpenGL** به خاطر آوردن لیست آرگومانهای تابع را آسانتر میکند. بعضی از نسخه های تابع **glColor** چهار آرگومان میگیرد که آرگومان چهارم مربوط به تعیین مولفه آلفا (شفافیت) می باشد.

بسیاری از کامپایلر های C/C++ که برای محیط windows ساخته شده اند اینگونه تصور میکنند که هرلیترال ممیز شناور از نوع double است مگر آنکه بطور صریح توسط مکانیسم پسوند به کامپایلر اعلام شود. هنگامی که از لیترال ها (همان حروف اختصاری) برای اعلان آرگومان ممیز شناور استفاده میکنید اگر شما مشخص نکنید که این آرگومانها از نوع float هستند و نه از نوع double کامپایلر در زمان کامپایل یک اخطار میفرستد چون کشف کرده که شما دارید یک متغیر از نوع double را به تابع ارسال میکنید در صورتی که تابع به گونه ای تعریف شده که فقط انواع float را بپذیرد. در نتیجه ممکن است دقت کار از میان برود. موقعی که این اخطار ها به حد زیادی برسد میتواند حتی عمل شناسایی خطاهای نحوی درطول برنامه را مشکل سازد. البته شما میتوانید این اخطار ها را با خاموش کردن قسمت ارسال اخطار کامپایلر خود از بین ببرید اما من شدیدا توصیه میکنم که سعی کنید کدتان را اصلاح کنید تا تمیز و قابل انتقال بماند. پس تک تک اخطارها را با درست کردن شان از بین ببرید.

به علاوه ممکن است شما وسوسه بشوید که از توابعی استفاده کنید که آرگومانهایی از نوع ممیز شناور با دقت مضاعف را میپذیرند (double-precision floating-point) به جای float یا double .
البته OpenGL بطور ذاتی از نوع float استفاده میکند و در نهایت انواع دیگر را به نوع ممیز شناور با دقت
معمولی تبدیل میکند زیرا هر متغیر از نوع double دو برابر متغیری از نوع float حافظه مصرف میکند و درمقادیر بالا اینکار OpenGL باعث افزایش سرعت و کارایی برنامه میشود.

منبع : کتاب OpenGL SuperBible 3rd Edition

استقلال از وابستگی به هر گونه پلتفرم خاص :

OpenGL یک API پیشرفته و قدرتمند برای طراحی گرافیک سه بعدی میباشد با بیش از 300 تابع و دستور که همه چیز را از نشاندن رنگ ماده بر روی شئ و خصوصیات انعکاس تا چرخش و دگرگونی های پیچیده مختصاتی را پوشش میدهد. ممکن است که شما تعجب کرده باشید که چگونه OpenGL

حتی یک تابع برای امور پنجره و یا مدیریت صفحه ندارد. بعلاوه هیچ تابعی برای دریافت ورودی صفحه کلید یا فعل و انفعالات ماوس ندارد. بهر حال یکی از اهداف اولیه تیم طراح **OpenGL** استقلال آن از پلتفرم بود. شما به طرق مختلفی تحت سیستم عامل های مختلف برای برنامه خود پنجره میسازید.
اگر **OpenGL** دستوری برای باز کردن یک پنجره داشت آیا شما از آن استفاده میکردید یا شما دوست داشتید تا از توابع توکار **API** سیستم عامل خود کمک بگیرید.
موضوع دیگر در مورد پلتفرم روش رسیدگی به کیبورد و رخدادهای ورودی ماوس تحت سیستم عامل های مختلف است. اگر تمام محیط ها شیوه رسیدگی مشابهی برای این رخدادها داشتند ما تنها یک محیط داشتیم تا در باره اش فکر کنیم و احتیاجی به یک **API** باز نداشتیم. بهر حال موضوع ما این نیست و ممکن است این در طول عمرمان رخ ندهد. پس ارزش مستقل از پلتفرم بودن **OpenGL** در همین است که تابعی برای سیستم عامل و **GUI** ندارد.

استفاده از**GLUT** :

در آغاز تنها کتابخانه **AUX** یعنی کتابخانه کمکی **OpenGL** که مخفف **OpenGL Auxiliary Library** میباشد. این کتابخانه به منظور تسهیل یادگیری و نوشتن برنامه های **OpenGL** طراحی شده بود. بدون اینکه برنامه نویس با جزئیات محیطی که در آن برنامه مینویسد گیج بشود حالا چه سیستم عامل یونیکس چه لینوکس یا مکینتاش و یا ویندوز. شما هنگام استفاده از **AUX** کد نهایی را نمی نوشتید. آن به عنوان یک اسکلت مقدماتی برای تست ایده های شما بود. نبودن خصوصیات مقدماتی **GUI** در این کتابخانه استفاده از آن را برای ساخت برنامه های حرفه ای محدود میکند.
چندین سال پیش بیشتر برنامه های نمونه **OpenGL** روی وب با استفاده از کتابخانه **AUX** نوشته شده بودند. پیاده سازی تحت ویندوز این کتابخانه پر از ایراد بود و باعث میشد برنامه بارها **Crash** کند. نبودن هیچ خصوصیت **GUI** یکی از دلایلی بود که آن را در عصری که همه برنامه ها بر اساس **GUI** طراحی میشدند بی مصرف ساخت.
در نهایت کتابخانه **AUX** با کتابخانه **GLUT** تعویض شد. **GLUT** کتابخانه ای است مستقل از پلتفرم برای طراحی برنامه های نمونه و نمایشی. **GLUT** مخفف **OpenGL Utility Toolkit** میباشد.(لطفا با کتابخانه **GLU** اشتباه نگیرید). مارک کیلگارد زمانی که در شرکت **SGI** کار میکرد **GLUT** را به منظور

از دور خارج کردن **AUX** با یک کتابخانه بهتر و لایقتر نوشت. کتابخانه **GLUT** رایگان میباشد. شما میتوانید آخرین ورژن آن را از این آدرس دانلود کنید.

http://www.xmission.com/~nate/glut.html

http://www.sun.com/software/graphics/opengl/glut/download.xml

در موقع دانلود مواظب باشید که نگارش مخصوص ویندوز را دانلود نمایید. ما در بیشتر آموزشهایی که من از کتاب **OpenGL SuperBible 3rd Edition** ترجمه کرده ام از **GLUT** استفاده میکنیم.
این کار برای ما دو هدف را بر آورده میکند اول اینکه افراد بیشتری میتوانند از آموزشهای ما استفاده کنند و نه فقط برنامه نویسان ویندوز. با یک تلاش مختصر کاربران سیستم های لینوکس و مکینتاش میتوانند **GLUT** را در محیط برنامه نویسی شان آماده و تنظیم کنند و با ما آموزشهای این وبلاگ را دنبال کنند. اما دلیل دومی که ما از کتابخانه **GLUT** استفاده میکنیم اینست که استفاده از این کتابخانه هرگونه نیاز به دانستن برنامه نویسی **GUI** تحت پلتفرم خاص را از بین میبرد و ما میتوانیم همه حواسمان را به قابلیتها و یادگیری **OpenGL** معطوف کنیم.
بعد ها و سر فرصت ما میتوانیم روش ایجاد برنامه با توابع **API** ویندوز و استفاده از **openGL** را می آموزیم. فعلا ما نیاز داریم که تمام وقت خود را روی یادگیری قابلیتهای **OpenGL** صرف کنیم. البته ما نمیتوانیم همیشه از **GLUT** استفاده کنیم چون آن به درد برنامه های بزرگ و واقعی نمیخورد. اما نیاز ما را در طی آموزشهایمان به خوبی برطرف میکند. همانطور که قبلا نیز گفتم بعدها روش ایجاد و طراحی و مدیریت پنجره برنامه را با استفاده از توابع **API** ویندوز فرا خواهیم گرفت. و این که چطور برنامه های **OpenGL** خود را با کمک توابع **API** ویندوز بنویسیم.
منبع : کتاب OpenGL SuperBible 3rd Edition

openGLیکAPI است نه یک زبان برنامه نویسی :

OpenGL یک **API** است نه یک زبان برنامه نویسی. **OpenGL** یک رابط برنامه نویسی است. هر زمان که ما میگوییم که این برنامه بر اساس **OpenGL** است یا **OpenGL Based** یا این که این یک برنامه

OpenGL است منظورمان اینست که این برنامه در یک زبان برنامه نویسی مانند **C** و یا **++C** و یا **Java** نوشته شده که تعداد یک و یا بیشتر از توابع کتابخانه **OpenGL** را فراخوانی میکند.
ما نمیگوییم که برنامه تنها از **OpenGL** برای طراحی استفاده میکند ممکن است که ما از بهترین خصوصیات دو بسته مختلف گرافیکی استفاده کنیم. ممکن است ما از **OpenGL** برای انجام تعدادی از وظایف ویژه و از **GDI** برای کارهای دیگری استفاده کنیم. تنها استثنا در این مورد **GLSL** یعنی همان زبان برنامه نویسی سایه ها در **OpenGL** میباشد که بعدها در آموزشهایمان تدریس خواهد شد.
به عنوان یک **API** کتابخانه **OpenGL** از شیوه صدا زدن تابع در **C** یا **++C** پیروی میکند. آموزشهای ما بر مبنای زبان **C** میباشد. اما **++C** نیز میتواند به سادگی به توابع **API** مانند **C** دسترسی پیدا کند. در حقیقت اگر شما با **++C** آشنا هستید هیچ گونه مشکلی با آموزشهای ما نخواهید داشت. **OpenGL** در کلیه زبانهای برنامه نویسی از قبیل **C** و **++C** و **VB** و **#C** و **Delphi** و **Java** و ... قابل استفاده است.

کتابخانه ها و فایلهای سر آیند :

هر چند که **OpenGL** یک کتابخانه برنامه نویسی استاندارد است. این کتابخانه پیاده سازیهای مختلفی دارد. مایکروسافت **OpenGL** را به عنوان یک رندر کننده نرم افزاری ساپورت میکند. این به این معناست که هنگامی که یک برنامه نوشته شده تا از **OpenGL** استفاده کند در طی برنامه توابع بسیاری از **OpenGL** را فراخوانی میکند. پیاده سازی مایکروسافت رندر کردن توابع سه بعدی را انجام میدهد و شما نتیجه را بر روی صفحه در پنجره برنامه میبینید.
در آخر همین مقاله توضیح میدهم که چرا پیاده سازی مایکروسافت تنها بدرد لای جرز دیوار میخورد و ما باید برای استفاده از قابلیت های **OpenGL 2** در ویندوز چکار کنیم. البته این نظر شخصی من یعنی ساسان میباشد.پیاده سازی مایکروسافت در فایل **OpenGL32.dll** قرار دارد که یک کتابخانه پویا میباشد و در پوشه **c:\windows** قرار دارد. در بیشتر پلتفرم ها کتابخانه **OpenGL** همرا با یک کتابخانه سودمند کمکی ارائه میشود که به نام **GLU** معروف است.که در ویندوز با نام **Glu32.dll** ارائه شده و در آدرس **c:\windows** قرار دارد. این کتابخانه سودمند مجموعه ای از توابع است که وظایف مشترکی را انجام میدهند مانند محاسبات ماتریکس های ویژه و همچنین پشتیبانی از کلیه از انواع مختلی از منحنی ها و سطوح پیچیده را مهیا میکنند.مرحله تنظیم کامپایلر برای لینک کردن کتابخانه صحیح از محیطی به

محیط دیگر متفاوت است اما ما این مرحله را در آموزش بعدیمان توضیح خواهیم داد.نمونه های اولیه برای کلیه توابع **OpenGL** و انواع داده و ماکروها در فایل **gl.h** قرار دارد. محیط برنامه نویسی مایکروسافت این فایل را دارد. همچنین نمونه های اولیه توابع کتابخانه کمکی سودمند **OpenGL** در فایل **glu.h** قرار دارد. این فایلها معمولا در مسیر **include/** محیط برنامه نویسی شما قرار دارد. برای مثال قطعه کد زیر طریقه استفاده از **OpenGL** را در یک برنامه نمونه ویندوز نمایش میدهد.

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
```

و اما حالا میرسیم به توضیحاتی که قرار بود من در مورد پیاده سازی مایکروسافت انحصارطلب پدر سوخته از **OpenGL** بدهم.

در محیط برنامه نویسی ویژوال استدیو اگر خوب به فایلهای **gl.h** و **glu.h** دقت کنید متوجه میشوید که این فایلها نهایتا تا **OpenGL 1.1** را ساپورت میکند و همینطور فایل **opengl32.dll** که آن نیز یک پیاده سازی بر مبنای **OpenGL 1.1** میباشد. اما این به نظر شما یعنی چه. این یعنی مایکروسافت انحصار طلب چون طاقت دیدن حریف را ندارد با این کار سعی کرده پشتیبانی خود را از **OpenGL** محدود کند تا توسعه دهندگان به استفاده از دایرکت ایکس تشویق شوند. اما کور خوانده است. این دلیل نمیشود که چون مایکروسافت در محیط برنامه نویسی اش از **OpenGL 2** پشتیبانی نمیکند ما نتوانیم از **OpenGL 2** استفاده کنیم. حالا چاره چیست. شرکت هایی مانند **nVIDIA** و **ATI** هر کدام **OpenGL 2** را درقالب یک فایل اکستنشن ایجاد کرده اند که شما با اضافه کرد و الحاق آن به ابتدای برنامه خود میتوانید از قابلیتها و توابع **OpenGL 2** استفاده کنید البته مشروط بر این که کارت گرافیک شما از **OpenGL** به صورت سخت افزاری پشتیبانی نماید که دیگر امروزه تقریبا همه کارتها این ویژگی را دارند.مثلا من در کامپیوترم یک کارت **Geforce 5200** دارم که تا **OpenGL 1.5** را ساپورت میکند. خوب من حالا با اضافه کردن فایل **glext.h** به ابتدای برنامه ام میتوانم از قابلیتهای **OpenGL 1.5** استفاده کنم.البته فلسفه وجود اکستنشن ها چیز دیگری است . به طور مثال یک سازنده سخت افزار یک ویژگی خاص را در کارت خود قرار میدهد خوب حالا این شرکت برای این قابلیت مخصوص یک اکستنشن مینویسد تا برنامه نویسان بتوانند از این قابلیت در برنامه هاشان استفاده کنند. اما ما از این روش استفاده میکنیم تا به قابلیتها و توابع **OpenGL 2** دسترسی پیدا کنیم.

منبع : کتاب OpenGL SuperBible 3rd Edition و سایتهای gamedev.net و glew.sourceforge.net

OpenGL چگونه کار میکند :

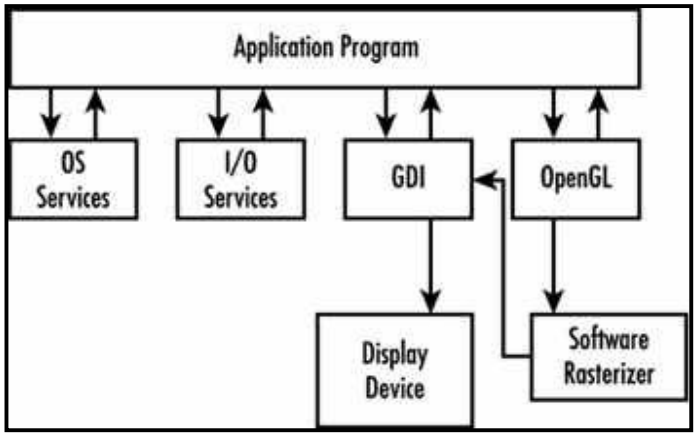
OpenGL بیشتر از آنکه یک API گرافیکی توصیفی باشد حالت رویه ای دارد. بجای توصیف صحنه و اینکه صحنه چگونه باید ظاهر شود برنامه نویس مراحل لازم را برای دست یافتن به نمایش معین یا یک افکت را تعیین میکند. این مراحل باعث فراخوانی دستورات زیادی از OpenGL میشود. این فرامین برای رسم اشکال ابتدایی گرافیکی مانند خط و نقطه و چندضلعی در صحنه سه بعدی استفاده میشوند.بعلاوه OpenGL نورپردازی و نگاشت بافت و آمیختگی و شفاف نمایی و انیمیشن و بسیاری دیگر از افکت های ویژه سه بعدی و قابلیت های زیاد دیگری را پشتیبانی میکند.
OpenGL شامل هیچ تابعی برای مدیریت پنجره و یا محیط بصری نمیباشد. برنامه نویسان این محیط ها را برای برطرف کردن نیازهای سطح بالایشان ایجاد میکنند و سپس با دقت آنها را با دستورات سطح پایین OpenGL برنامه نویسی میکنند.

پیاده سازی عمومی :

همانطور که قبلا عنوان شد یک پیاده سازی عمومی یک پیاده سازی نرم افزاری میباشد. پیاده سازیهای سخت افزاری برای دستگاههای سخت افزاری ویژه طراحی شده است مانند یک کارت گرافیکی یا یک مولد تصویر. یک پیاده سازی عمومی از لحاظ فنی میتواند بر روی هرکجا اجرا بشود مادامیکه سیستم بتواند تصاویر گرافیکی ساخته شده را نمایش دهد.

تصویر 1-2 مکان نمونه ای را نشان میدهد که OpenGL و یک پیاده سازی عمومی اشغال کرده اند هنگامی که یک برنامه در حال اجراست. برنامه نمونه توابع زیادی را فراخوانی کرده است. بعضی از توابعی که کاربر تولید کرده و بعضی ها که توسط سیستم عامل مهیا شده اند یا متعلق به کتابخانه زمان اجرای زبان برنامه نویسی هستند. زمانی که برنامه های ویندوز میخواهند که چیزی را بر روی صفحه

خروجی رسم کنند معمولا یکی از توابع **API** ویندوز را که (رابط دستگاه گرافیکی) نامیده میشود صدا میزنند. **GDI** شامل متد هایی است که به شما اجازه نوشتن متن و ترسیم اشکال دو بعدی ساده و غیره را میدهد.



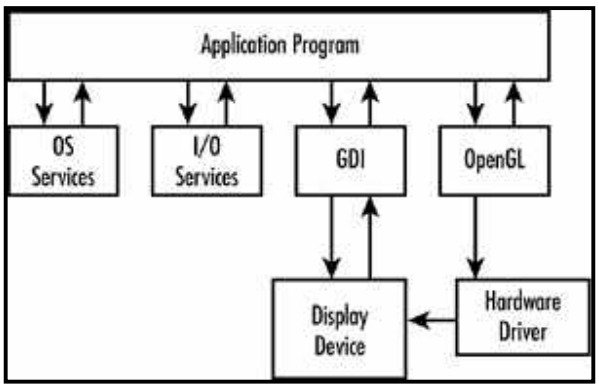
معمولا سازندگان کارت های گرافیکی یک درایور سخت افزاری با رابط های **GDI** تهیه میکنند که خروجی را بر روی مانیتور رسم کند. یک پیاده سازی نرم افزاری از **OpenGL** گرافیک های تقاضا شده توسط یک برنامه را میگیرد و از آن گرافیک سه بعدی یک تصویر دو بعدی رنگی ایجاد میکند. سپس این تصویر را به **GDI** میفرستد تا بر روی مانیتور نمایش دهد. در بقیه سیستم های عامل نیز وضع به همین منوال است اما شما **GDI** را با سرویس نمایش محلی سیستم عامل خود تعویض میکنید.

OpenGL یک جفت پیاده سازی نرم افزاری مشترک دارد. یکی پیاده سازی نرم افزاری مایکروسافت است که با هر ورژن از ویندوز مانند **NT 3.5** و بالاتر و **Win95** و **2000** و **XP** ارایه میشود.

SGI یک پیاده سازی نرم افزاری از **OpenGL** را برای ویندوز طراحی کرد که پیاده سازی مایکروسافت را از دور خارج میکند. این پیاده سازی دیگر به طور رسمی پشتیبانی نمیشود اما هنوز به مقدار زیادی توسط توسعه دهندگان استفاده میشود. که در انجمن های اوپن سورس از مقبولیت و پشتیبانی خوبی برخوردار است. **Mesa 3D** یک **OpenGL** مجوز دار نیست. بنابر این بیش از این که یک پیاده سازی رسمی باشد مانند یک همکار برای **OpenGL** است.

پیاده سازی سخت افزاری :

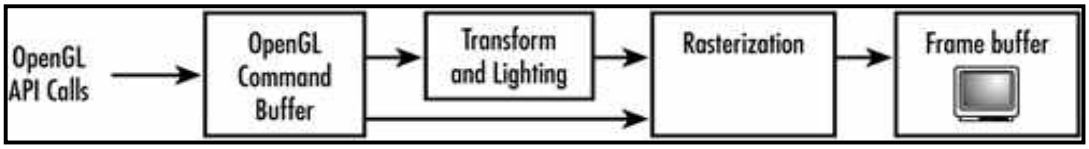
یک پیاده سازی سخت افزاری از **OpenGL** شکل یک درایور کارت گرافیکی را دارد. شکل **2-2** ارتباطش با برنامه مانند شکل **۲-1** است. توجه کنید که فراخوانی های توابع **OpenGL** به درایور سخت افزار پاس داده میشوند. این درایور خروجی خود را به **GDI** ویندوز برای نمایش پاس نمیدهد. رابط خودش مستقیما با سخت افزار نمایش گرافیکی رابطه دارد.



یک پیاده سازی سخت افزاری به عنوان یک پیاده سازی تسریع شده شناخته میشود. چون گرافیک سه بعدی با کمک سخت افزار بسیار بهتر و سریعتر از یک پیاده سازی صرفا نرم افزاری عمل میکند. چیزی که در تصویر **2-2** نشان داده نشده اینست که بخشی از قابلیتهای **OpenGL** هنوز به صورت نرم افزاری به صورت بخشی از درایور ایجاد میشود و بقیه قابلیتها و خصوصیات میتواند مستقیما به سخت افزار پاس داده شوند.

The Pipeline :

کلمه **pipeline** جهت شرح دادن پروسه ای که میتواند دو مرحله جداگانه یا بیشتر را در بر بگیرد استفاده میشود. تصویر **2-3** یک **pipeline** خلاصه شده **OpenGL** را نشان میدهد.به عنوان برنامه ای که توابع **API** مربوط به **OpenGL** را فراخوانی میکند دستورات در محلی بنام بافر دستور یا **Command Buffer** ذخیره میشود. این بافر بالاخره با اطلاعات راس و تکسچر و غیره پر میشود. وقتی این بافر تا آخرین حد پر شود توسط برنامه یا توسط طراحی درایور دستورات و اطلاعات به مرحله بعدی در



پروسه **Pipeline** پاس داده میشوند.

اطلاعات مربوط به رئوس معمولا تغییر شکل یافته هستند. در آموزشهای بعدی شما خواهید فهمید که این یعنی چه (برره ای بود). اما برای حالا همین قدر بدانید که "تغییر شکل و نورپردازی" یک مرحله شدیدا ریاضی گونه هستندکه نقاط برای تشریح مختصات هندسی اشیا استفاده میکنند.محاسبات نور پردازی به خوبی بر روی اطلاعات رئوس انجام میشوند تا نشان دهند هر راس با چه شدت رنگی و نوری باید نمایش داده شود.

هنگامی که این مرحله به پایان رسید اطلاعات به بخش بعدی **Pipeline** یعنی **Rasterization** خورانده میشود. **Rasterizer** در عمل یک تصویر رنگی از اطلاعات هندسی و رنگها و اطلاعات تکسچر میسازد. این تصویر سپس به بافر فریم **Frame Buffer** منتقل میشود. بافر فریم قسمتی از حافظه دستگاه نمایش گرافیکی (کارت گرافیک) میباشد. این بدین معنی است که تصویر در صفحه نمایش داده شده است. در یک سطح بالا این نمودار صحیح میباشد اما در یک سطح پایین تر قسمتهای زیاد دیگری نیز در این پروسه وجود دارد. همچنین استثنائاتی هم وجود دارد. همانطور که در نمودار هم پیداست بعضی از اطلاعات از مرحله **T&L** یا همان **Transform & Lighting** عبور نمیکنند.

در گذشته شتاب دهنده های سخت افزاری **OpenGL** چیزی جز **fast Rasterizer** نبودند. آنها تنها بخش **Rasterization** را شتاب میبخشیدند و پردازشگر سیستم میزبان مرحله **T&L** را به صورت نرم افزاری و به عنوان بخشی از **pipeline** انجام میداد. شتاب دهنده های با کیفیت تر (گرانتر) خودشان قسمت **T&L** را انجام میدادند. به این ترتیب بیشتر مراحل **Pipeline** در سخت افزار گرافیکی انجام میشد و گرافیک بیشتری بدست می آمد.

امروزه هر کارت ارزان قیمتی مرحله **T&L** را به صورت سخت افزاری انجام میدهند. خوبی این چیز در اینست که مدلهای با کیفیت بالاتر و تصاویر گرافیکی پیچیده تری در رندر گرافیکی بلادرنگ امکان پذیر میشود.

منبع : کتاب OpenGL SuperBible 3rd Edition

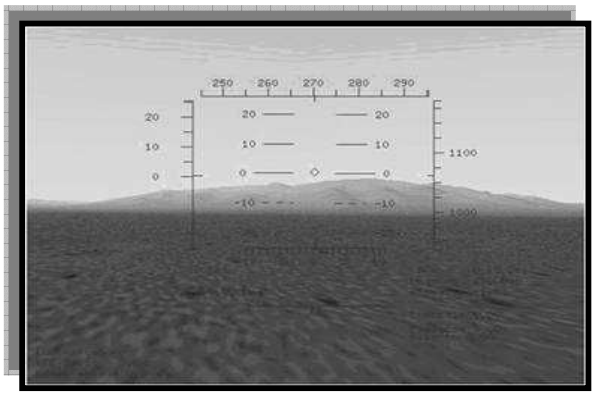
کاربردهاس مشترک گرافیک سه بعدی :

گرافیک سه بعدی در برنامه های کامپیوتری جدید کاربرد بسیاری دارد. استفاده که برنامه ها از گرافیک سه بعدی میکنند از بازیهای اینتراکتیو سه بعدی تا شبیه سازی و پزشکی و مصارف شغلی متفاوت است.

محصولات پر کیفیت سه بعدی راه خودشان را به سمت فیلمها و صنعت و آموزش به خوبی پیدا کرده اند.

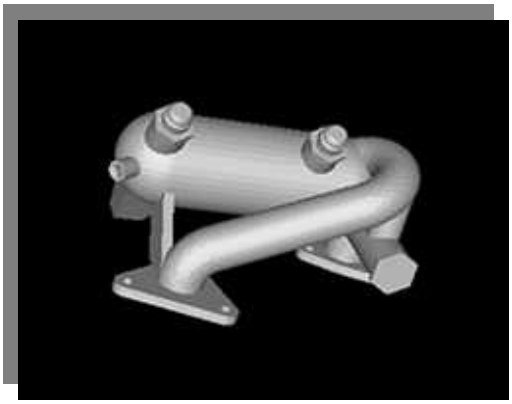
:: Real-time 3D

همانگونه که قبلا تعریف شد گرافیک های سه بعدی بلادرنگ متحرک هستند و با کاربر فعل و انفعال دارند. یکی از اولین استفاده ها از گرافیک بلادرنگ سه بعدی شبیه سازی پرواز در امور نظامی بود.هر چند امروزه شبیه سازهای پرواز به سرگرمی مشهوری برای مشتاقان خانگی تبدیل شده اند. تصویر **15-1** یک اسکرین شات از یک شبیه ساز پرواز معروف را نشان میدهد که از **OpenGL** برای رندر سه بعدی استفاده کرده است.



تصویر ۱-۱۵

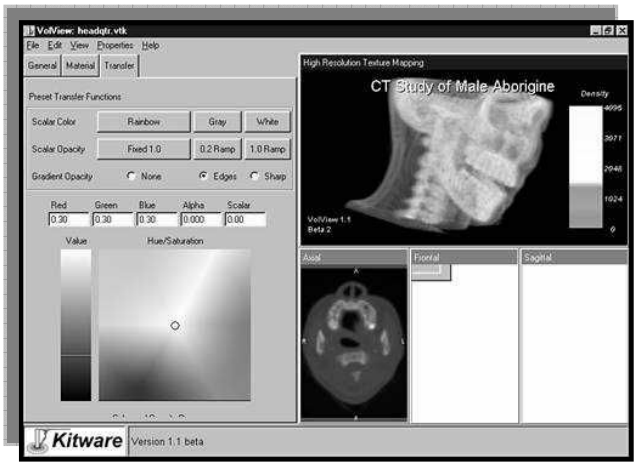
برنامه ها برای گرافیک سه بعدی بر روی کامپیوترها تقریبا بیشمار هستند. شاید عمومی ترین استفاده از گرافیک کامپیوتری سه بعدی بازیهای رایانه ای باشند. امروزه به سختی میتوان کامپیوتری را یافت که نیاز به یک کارت گرافیک سه بعدی نداشته باشد. سه بعدی همیشه برای تجسمات علمی و برنامه های مهندسی معروف بوده است. رابط های گرافیکی نرم افزاری هم از سخت افزار سه بعدی استفاده فراوان میبرند. برای مثال ورژن جدید سیستم عامل **Macintosh os x** برای رندر کردن تمام پنجره ها و کنترل ها و جلوه های تصویری از **OpenGL** استفاده میکند. تصاویر **16-1** ال **20-1** تعدادی از برنامه های بیشماری را نشان میدهد که برای رندر تصاویرشان و تولید تصاویر سه بعدی اینتراکتیو از **OpenGL** استفاده میکنند.



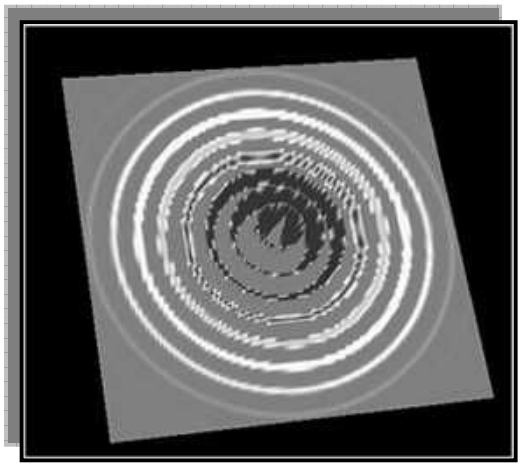
تصویر ۱-۱۶

تصویر ۱-۱۷

تصویر ۱-۱۸



تصویر ۱-۱۹



تصویر ۱-۲۰

گرافیک سه بعدی غیر همزمان Non-Real-Time :

برای برنامه هایی که از گرافیک سه بعدی بلادرنگ استفاده میکنند قانونی و.جود دارد. با دادن فرصت بیشتری برای پردازش تصاویر شما میتوانید گرافیک های سه بعدی با کیفیت بالاتری ایجاد نمایید. به طور مثال بعضی از نرم افزارهای مدل سازی از گرافیک سه بعدی بلادرنگ برای تقابل با هنرمند برای خلق محتوای مورد نظرش استفاده میکنند. سپس تصاویر به برنامه دیگری فرستاده میشوند (**ray tracer**) که تصاویر را رندر میکنند. رندر کردن یک فریم تنها برای انیمیشنی مانند داستان اسباب بازی به ساعتها زمان بر روی یک کامپیوتر سریع نیاز دارد. این پروسه رندر و ذخیره سازی صدها فریم یک انیمیشن را

میسازد که بطور رشته متوالی قابل پخش مجدد میباشد. اگرچه پخش تصاویر انیمیشن ممکن است یک عمل بلادرنگ به نظر برسد اما اینطور نیست. چون آن اینتراکتیو نیست در نتیجه آن بلادرنگ نیست بلکه بیشتر یک سری تصاویر از پیش رندر شده میباشد.

تصویر ۱-۲۱ عکسی را نشان میدهد که طرحش حروف کریستالی است و با **OpenGL** ایجاد شده است. حروف شفاف هستند و تکنیک های ضد دندانهای و افکت های آیینه ای و سایه بخوبی بر روی آن اعمال شده اند.

تصویر ۱-۲۱

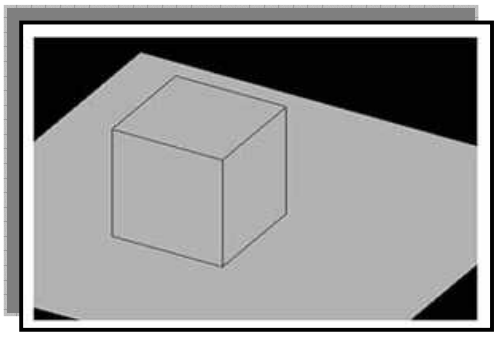
منبع : کتاب OpenGL SuperBible 3rd Edition

مختصری درباره افکت های سه بعدی :

حالا شما تقریبا میدانید که جلوه سه بعدی توسط یک کیف پر از پرسپکتیو و شگردهای هنری بر روی صفحه تخت کامپیوتر خلق میشود. اجازه دهید بعضی از این افکت ها را بررسی کنیم بعدها ما میتوانیم در طول آموزشهایمان به آنها مراجعه کنیم و شما میدانید که ما درباره چه چیزی صحبت میکنیم.

پرسپکتیو Perspective :

پرسپکتو به زاویه بین خطوط اشاره میکند که جلوه بعد سوم را ایجاد میکنند. شکل ۱-۴ یک مکعب سه بعدی را نشان میدهد که با خطوط طراحی شده اند. این جلوه قدرتمندی است. اما آن هنوز میتواند باعث خطای ادراک شود که قبلا نام در موردش صحبت کردیم.(مدتی به تصویر خیره نگاه کنید می بینید که آن شروع به پریدن به بیرون و داخل میکند).

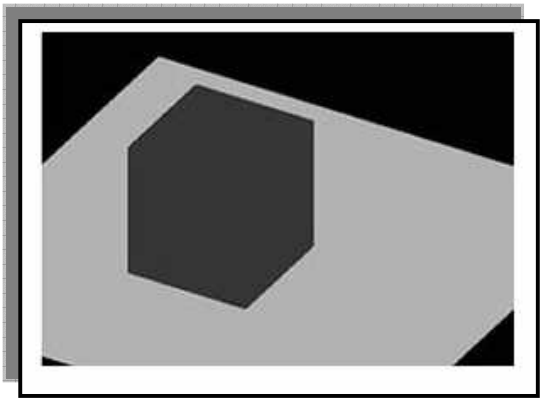


تصویر ۱-۵

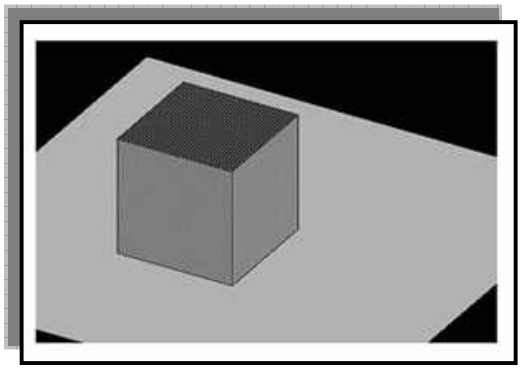
در شکل ۱-۵ مغز اطلاعات بیشتری از گرایش واقعی مکعب دارد علت آن برداشته شدن خطوط پنهان تصویر است. شما انتظار دارید که جلوی یک شی پشت آن را از نظر بپوشاند. برای سطوح سه بعدی ما این روش را حذف کردن خطوط پنهان مینامیم.

رنگ و سایه (Color and Shading) :

اگر ما به تصویر مکعب 1-5 به اندازه لازم خیره شویم میتوانیم خودمان را متقاعد کنیم که به یک تصویر تو رفته نگاه میکنیم و نه به سطوح بیرونی یک مکعب. برای افزایش ادراکمان ما باید به آنسوی خطوط طراحی حرکت کنیم و رنگ را اضافه کنیم تا یک شی سه بعدی خلق شود. تصویر 1-6 نشان میدهد که وقتی ما رنگ قرمز را به مکعب اضافه کنیم چه اتفاقی میافتد. آن دیگر شبیه یک مکعب نیست. با اضافه کردن یک رنگ متفاوت به هر سطح همانطور که در تصویر 1-7 نمایش داده شده ما دوباره درک سه بعدی مان از شئ را به دست می آوریم.



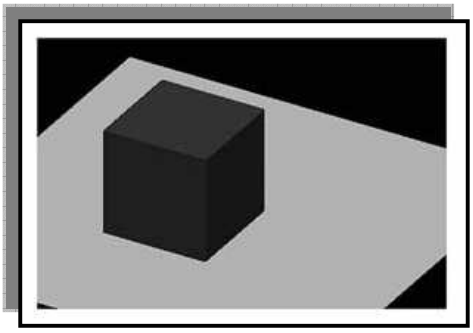
تصویر ۱-۶



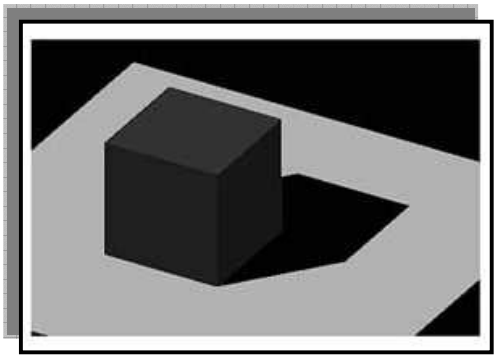
تصویر ۱-۷

نور و سایه ها (Light and Shadows) :

پوشاندن هر سطح از اطراف مکعب با یک رنگ متفاوت به چشم های ما کمک میکند تا سطوح مختلف مکعب را ببینند. با سایه زنی هر سطح به طور مقتضی میتوانیم به مکعب سیمای سه بعدب ببخشیم..این همچنین نشان میدهد که در بالای یکی از گوشه های مکعب چراغی روشن است. همانطور که در تصویر 1-8 نشان داده شده است. عکس 1-9 با اضافه کردن یک سایه در پشت مکعب یک گام جلوتر رفته است. در اینجا حیله ما بسیار متقاعد کننده است.



تصویر ۱-۸



تصویر ۱-۹

نگاشت بافت (Texture Mapping) :

دست یافتن به یک درجه بالا در رئالیسم (واقع گرایی) با چیزی غیر از هزاران میلیون ذره نور و چند ضلعی های سایه پردازی شده کاری سخت و دور از عقل میباشد. متاسفانه هر چه شما چند ضلعی های بیشتری را به سخت افزار گرافیکی بفرستید مدت زمان بیشتری را برای رندر شدن میطلبد. یک تکنیک زیرکانه اجازه میدهد شما از اشکال ساده تر هندسی استفاده کنید اما به درجه بالایی از رئالیسم دست یابید در این تکنیک عکسی پر از جزییات گرفته میشود و سپس آن تصویر را بر روی یک سطح از چند ضلعی اعمال میکند.بجای اشیائی که فقط از سطوح رنگی ساخته شده باشند شما میتوانید طرحی از چوب و یا لباس و یا آجر و غیره به چند ضلعی ها اعمال کنید. این تکنیک اعمال تصاویر بر روی چندضلعی ها برای تامین کیفیت بیشتر نگاشت بافت (Mapping Texture) نامیده میشود.

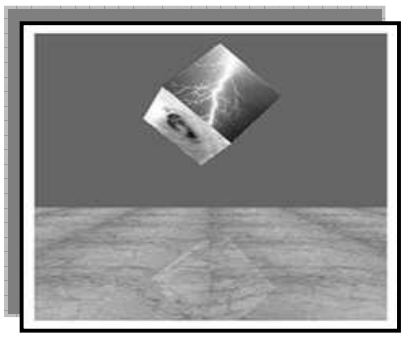
تصویر ۱-۱۰

تصویر ۱-۱۱

همه ما میدانیم که مه چیست.مه یک تاثیر جوی است که حالت گرفتگی یا ابهام را به شئ در یک صحنه می افزاید. که میزان آن بستگی به موقعیت و فاصله بیننده و غلظت مه دارد. اشیا خیلی دور ممکن است که بطور کامل محو شوند.شکل ۱-۱۱تاثیر مه را در صحنه نمایش میدهد. توجه کنید که مه چطور بر باور پذیری زمین تاثیر دارد.

آمیختگی و شفافیت (Blending and Transparency) :

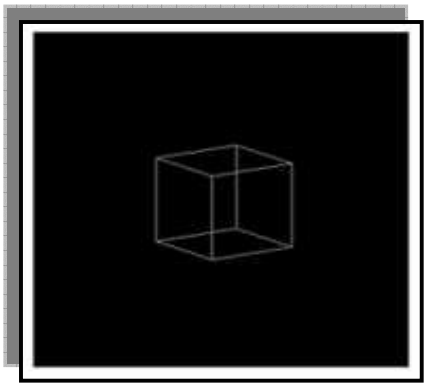
آمیختگی (Blending) به ترکیب رنگها یا اشیا در صحنه گفته میشود. مثل اینست که شما دو عدد فیلم عکاسی را بر روی هم قرار دهید. شما میتوانید از تکنیک آمیختگی برای اهداف مختلفی استفاده کنید. با تغییر میزان آمیختگی هر شی با صحنه شما میتوانید اشیا را شفاف کنید مثل اینکه شما هم اشیا را میبینید هم فضای پشت آن را. شما همچنین میتوانید از آمیختگی برای رسیدن به جلوه بازتاب و یا انعکاس استفاده کنید. هماطور که در تصویر 1-12 نشان داده شده است. شما یک مکعب پوشیده از بافت را میبینید که دوبار رندر شده است. اول مکعب به صورت واژگون و در زیر طبقه مرمرین رندر شده است. این کف مرمرین با صحنه آمیخته شده است که اجازه میدهد مکعب از میانش نمایان شود. سرانجام مکعب به صورت وارونه دوباره در بالای طبقه و به حالت شناور رسم شده است. نتیجه پیدایش یک سطح آیینه ای در سطح صیقلی مرمر میباشد.



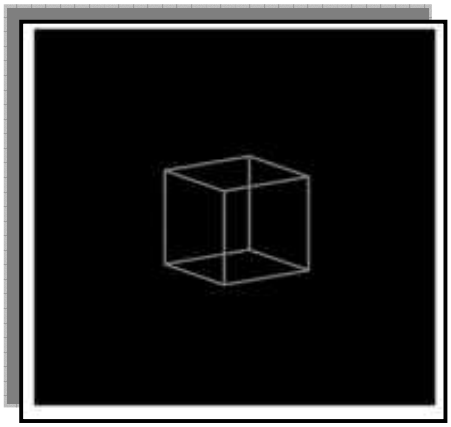
تصویر ۱-۱۲

ضد دندانه ای (Antialiasing) :

دندانه دار بودن (کنگره دار بودن) یک افکت مرئی بر روی صفحه است که علت آن اینست که تصویر از پیکسلهای گسسته تشکیل شده است. در تصویر **1-13** شما میتوانید ببینید که خطوطی که مکعب را ساخته اند لبه های دندانه دار دارند. با یک آمیختگی خوب و با دقت خطوط با پشت زمینه شما میتوانید لبه های دندانه دار را محو کنید و به خطوط سیمایی نرم و لطیف ببخشید. همانطور که در تصویر **1-14** نشان داده شده است.این تکنیک آمیختگی آنتی الیزینگ نامیده میشود.. شما همچنین میتوانید این تکنیک را به لبه چند ضلعی ها اعمال کنید تا صحنه ای واقعی تر داشته باشید.



تصویر ۱-۱۳



منبع : کتاب OpenGL SuperBible 3rd Edition

مقدمه ای بر گرافیک کامپیوتری :

اولین کامپیوترها شامل سطرهای بسیاری از چراغها و کلیدها بودند. متخصصین فنی و مهندسین برای ساعتها و روزها و هفته ها کار میکردند تا این ماشینها را برنامه ریزی کنند و نتایج محاسباتشان را بخوانند. الگوی روشن سازی چراغها اطلاعات مفیدی را نصیب کاربران کامپیوتر کرد. ممکن است شما بگویید اولین چیز در گرافیک کامپیوتری پانلی از لامپهای چشمک زن بود که این عقیده درست است.اما زمان تغییر کرد. از آن دستگاههایی که ماشین تفکر نامیده بودندش دستگاههای قابل برنامه نویسی جدیدی پدید آمد که میتوانستند بر روی لوله کاغذ با مکانیزمی شبیه ماشین های تحریر از راه دور چاپ کنند. داده ها باید بصورت موثر بر روی نوارهای مغناطیسی یا دیسک یا سطرهای کاغذ بشکل پانچ شده نگهداری میشدند. به علت این که هر کاراکتری از حروف الفبا اندازه و شکل ثابتی داشت برنامه نویسان خلاق بسیار خوشحال شدند چون میتوانستند تصاویری خلق کنند که تنها از علامت ستاره تشکیل میشد.

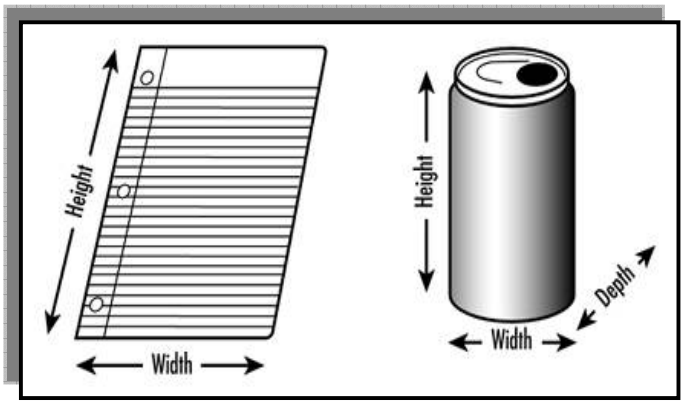
CRT وارد میشود :

کاغذ به عنوان یک خروجی متوسط برای کامپیوترها مفید است و استفاده از آن تا امروز هم تداوم یافته. پرینتر های لیزری و جوهر افشان امروزه هنر چاپ حروف اسکی را بخاطر کیفیت بالا و نزدیک به واقیت شان از بین برده است.بهر حال مصرف کاغذ میتواند ولخرجی در منابع طبیعی باشد مخصوصا به این دلیل کهدر بیشتر اوقات ما نیاز به یک کپی چاپی از محاسبات اطلاعاتی مان نداریم.لامپ اشعه کاتدی یک افزودنی لازم برای کامپیوترها بود. مانیتو های **CRT** اولیه در آغاز تنها خروجی هایی برای تصاویری بودند که تنها حروف اسکی را نمایش میداد. اما **CRT** ها بطور کلی توانایی بالای در رسم نقاط و خطوط داشتند درست مانند کارکترهای حروف. به زودی نشانه ها و تصاویر دیگری شروع به اضافه شدن به مجموعه کاراکترهای خروجی شدند. برنامه نویسان از کامپیوترها و مانیتورهایشان برای خلق تصاویری که به متون و جدول ها ضمیمه میشدند کردند. اولین الگوریتم ها برای خلق خطوط و منحنی ها توسعه یافت و منتشر شد. گرافیک کامپیوتری به چیزی بیشتر از سرگرمی تبدیل شد. اولین

گرافیک های کامپیوتری که بر روی این خروجی ها نمایش داده میشد دو بعدی (2D) بودند. این خطوط و دایره ها و چندضلعی ها برای طراحی گرافیکی برای مقاصد گوناگون استفاده میشدند. نمودارها و رسم ها میتوانستند اطلاعات علمی و آماری را نمایش دهند.

به صورتی که جداول و اشکال نمیتوانستند. بسیاری از برنامه نویسان ماجراجو توانستند بازیهای ساده ای نظیر Lunar Lander ویا Pong را طراحی کنند که با خطوط ساده ای طراحی شده بودند که چندین بار در ثانیه ریفرش میشدند.اصطلاح Real-time اولین بار به تصاویری که متحرک بودند اطلاق شد. یک استفاده گسترده از کلمه در علم کامپیوتر که به این معناست که کامپیوترها میتوانند اطلاعات را با همان سرعتی که وارد میشوند و حتی سریعتر پردازش نمایند. برای مثال تلفن زدن یک عمل real-time است که انسانها در آن شرکت دارند. شما صحبت میکنید و شنونده فورا صدای شما را میشنود و به آن واکنش نشان میدهد. و به شما این اجازه را میدهد که حرفهای او را شنیده و واکنش نشان دهید و همینطور الی آخر. در حقیقت یک تاخیری به علت مسائل الکترونیکی وجود دارد اما این تاخیر معمولا برای کسانی که مشغول مکالمه هستند جزیی میباشد. در مقام مقایسه نوشتن نامه یک عمل real-time نیست.بکارگیری اصطلاح real-time برای گرافیک کامپیوتری بدین معناست که کامپیوتر در حال ارائه کردن یک انیمیشن یا رشته ای از تصاویر است که بی درنگ به بعضی از ورودی ها واکنش نشان میدهد. مانند حرکت دادن جوی استیک و یا ضربه زدن به صفحه کلید و الی آخر. گرافیک کامپیوتری بلادرنگReal-time میتواند خروجی های قابل خواندن عددی یا بازیهای اینتراکتیو و شبیه سازیهای مجازی را نمایش دهد.

پیش به سوی سه بعدی : اصطلاح سه بعدی یا 3D به این معناست که یک شئ در حال نمایش سه بعد قابل اندازه گیری دارد طول و عرض و عمق. یک مثال برای یک شئ دو بعدی یک قطعه کاغذ است بر روی میزتان که عمق (ضخامت) چندانی ندارد. و مثالی برای یک شئ سه بعدی یک قوطی کنسرو است که پیرامون آن طول و عرضش را نشان میدهد و بلندی آن نمایشگر عمق است. بسته به پرسپکتیو شما میتوانید تعیین کنید که کدام طرف قوطی طول و یا عرض باشد اما در نهایت قوطی سه بعد دارد.شکل 1-1 نشان میدهد که ما چگونه میتوانیم ابعاد قوطی و کاغذ را مقایسه نماییم.

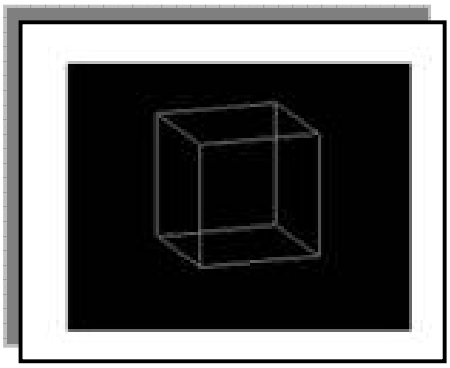


تصویر ۱-۱

برای قرنها هنرمندان میدانستند که چگونه تصاویری طراحی کنند که عمق واقعی داشته باشد. یک نقاشی ذاتا یک شئ دو بعدی است. چون آن چیزی غیر از یک تکه کرباس که رویش نقاشی کشیده باشند نیست. همینطور گرافیک سه بعدی کامپیوتری در حقیقت تصاویری دو بعدی هستند روی یک صفحه تخت که بعد سوم و یا خطای دید در عمق را مهیا میسازند.

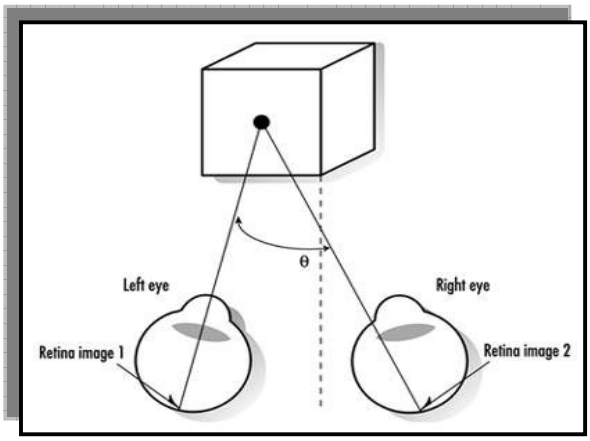
2D + Perspective = 3D

اولین گرافیک کامپیوتری بدون شک بصورت شکل ۱-۲ ظاهرشدند. جایی که شما میتوانید یک مکعب سه بعدی ساده را که با **12** خط ساخته شده ببینید. چیزی که نمای مکعب را سه بعدی ساخته پرسپکتیو است. (زاویه بین خطوط که خطای دید در عمق را به ما تلقین میکنند).



تصویر ۱-۲

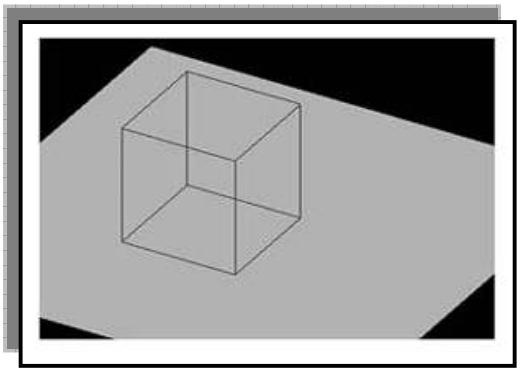
برای تماشای واقعی در فضای سه بعدی شما در حقیقت نیاز دارید که شئ را با هر دو چشم ببینید و هر چشم یک تصویر جداگانه از شئ را ثبت میکند. به تصویر 3-1 دقت کنید هر چشم یک تصویر دو بعدی را دریافت میکند که مانند یک تصویر موقت بر روی شبکیه هر چشم نگاشته میشود. این دو تصویر اندکی با هم تفاوت دارند چون از دو زاویه جداگانه دریافت شده اند.سپس مغز این دو تصویر را با هم ترکیب میکند تا یک تصویر مرکب سه بعدی در سر شما پدید آورد.



تصویر ۱-۳

در شکل 3-1 زاویه بین دو تصویر با دور شدن شئ کوچکتر میشود. شما میتوانید این اثر سه بعدی را با افزایش زاویه بین دو تصویر تقویت کنید.صفحه نمایش کامپیوتر یک تصویر تخت بر روی سطحی تخت است و نه دو تصویر متفاوت از منظر متفاوت که بر روی هر چشم بیفتد.همانطور که معلوم شد بیشتر چیزهایی که در گرافیک سه بعدی کامپیوتری مطرح میشوند در حقیقت شباهت زیادی است به سه بعدی واقعی. این شباهت محصول راههایی است که هنرمندان در طی سالها طراحی هایشان را با عمق زیاد تهیه میکردند. با استفاده از شگردهایی که طبیعت برای افراد یک چشمی تدارک دیده است.ممکن است بخاطر بیاورید که بعضی مواقع در زندگی اگر جلوی یکی از چشمانتان را میگرفتیددنیا فورا به شکل تخت (صاف) در نمی آمد. چه اتفاقی می افتاد وقتی که شما جلوی یکی از چشمانتان را میپوشاندید. ممکن است شما فکر کنید که هنوز در حال تماشای سه بعدی هستید.اما این آزمایش را انجام دهید. یک لیوان یا شئ دیگر دور از دستتان در سمت چپ قرار دهید. حالا چشم راست را با دست راستتان

بپوشانید و سعی کنید لیوان را لمس کنید. متوجه میشوید که شما مدت زمان بیشتری نیاز دارید تا بتوانید لیوان را لمس کنید. حالا دست خود را از جلوی چشمتان بردارید و لیوان را لمس کنید شما به راحتی تشخیص میدهید که چقدر باید دستتان را دراز کنید تا بتوانید لیوان را لمس کنید. حال شما درک میکنید که افراد یک چشمی چرا با درک مسافت مشکل دارند.پرسپکتیو تنها برای خلق سیمای سه بعدی کافی است به مکعب تصویر **1-2** نگاه کنید هر مکعب بدون رنگ و سایه ای هنوز یک سیمای سه بعدی دارد. به مدت طولانی به تصویر نگاه کنید میبینید که جلو و عقب تصویر جایشان را با هم عوض میکنند. مغز شما به علت نبود هیچ رنگی در طرح گیج شده است.



تصویر ۱-۴

مصنوعات سه بعدی :

دلیل اینکه وقتی شما جلوی یکی از چشمانتان را میپوشانید دنیا به یکباره تخت نمیشود این است که هنوز افکت های سه بعدی زیادی حاظر هستند وقتی که دو بعدی تماشا میشوند. این افکت ها (تاثیرات) به اندازه ای هستند که باعث شوند مغز شما به راحتی مسافت و عمق را تشخیص دهد. مسلم ترین چیز اینست که اشیا نزدیک تر بزرگتر از اشیا دور نمایان میشوند.این افکت پرسپکتیو **Foreshortening** نامیده میشود. این افکت و تغییرات رنگ و تکسچرها و نور ها و اختلاف در شدت رنگ همه با هم به ادراک ما یک تصویر سه بعدی را اضافه میکنند.

منبع : کتاب OpenGL SuperBible 3rd Edition

مرحله اول : در این مرحله ما قصد داریم با ارایه یکسری از آموزشهای مقدماتی در باب **OpenGL** که عموما از کتابهای **OpenGL Programming Guide fifth Edition** و **OpenGL SuperBible 3rd Editon** استخراج شده اصول اولیه و برنامه نویسی مقدماتی با استفاده از **OpenGL** را به شما دوستان بیاموزیم. همچنین در طول این مرحله مقالات و مطالبی نیز از گرافیک کامپیوتری مخصوصا گرافیک سه بعدی تقدیم میشود که بیشتر حالت مقدماتی داشته و خواندن آنها برای ادامه کار ما لازم میباشد.

مرحله دوم : تا اینجا ما تقریبا آشنایی مختصری با **OpenGL** وگرافیک سه بعدی داریم حالا ما آماده ایم تا با مفاهیم پیشرفته تری روبرو شویم. حال ما شروع به بررسی و آموزش سلسله آموزشهایی میکنیم که من از سایت های مختلف جمع آوری نموده ام. از جمله آموزشهای **NeHe** که در آدرس **http://NeHe.gamedev.net** قرار دارد و بسیاری دیگر که به بررسی تکنیکهای پیشرفته **OpenGL** میپردازد. همچنین در طی این دوره یکسری مقالات ومطالب در رابطه با تکنیکهای پیشرفته گرافیک کامپیوتری از قبیل **bump maping** و **Normal maping** و **Motion Blur** و **Anti Alaising** و روشهای مختلف **Rendering** و بسیاری دیگر تقدیم دوستان میشود و نحوه پیاده سازی این تکنیکها با مثالهایی کامل در **OpenGL** بررسی و آموزش داده میشود.

مرحله سوم : در این مرحله ما با زبان سایه زنی **OpenGL** یعنی **GLSL** آشنا میشویم و آن را بطور کامل می آموزیم و کلیه تکنیکهایی که با استفاده از **GLSL** قابل پیاده سازی هستند بررسی و تدریس میکنیم. در زمینه گرافیک کامپیوتری سعی میکنیم با ارایه یکسری مقالات پیشرفته و تکمیلی تقریبا با کلیه تکنیکهای گرافیک سه بعدی و کامپیوتری آشنا شده باشیم.

مرحله چهارم : اگر عمری باقی بود و توانستیم تا اینجا برسیم بطور کامل با موتور رندرگر **OGRE** آشنا میشویم و سعی میکنیم آن را بصورت یاد بگیریم و در نهایت سورس آن را مورد مطالعه قرار دهیم و از آن به بعد تمام سعی و تلاشمان را بر روی طراحی یک موتور سه بعدی متمرکز میکنیم.

خوب این روش و خط مشی کلی این وبلاگ بود که خواستم آن را در قالب یک برنامه ریزی منظم و ساخت یافته از همین الان به اطلاع خوانندگان برسانم. روند کار ما به صورتی خواهد بود که کلیه افراد

حتی بدون آشنایی مقدماتی با گرافیک کامپیوتری بتوانند با ما این آموزشها را شروع کرده و قدم به قدم و همراه با ما مسیر را از ابتدا تا انتها طی کنند.

اول : بسیاری از آموزشهای ما در مورد **OpenGL** مستلزم آشنایی با روش طراحی برنامه با استفاده از توابع **API** ویندوز میباشد. کسانی که با آن آشنایی ندارند میتوانند با عضویت در سایت **http://persian-designers.com** به مقالات و مطالب خوبی در ارتباط با برنامه نویسی مقدماتی ویندوز در سطحی که مورد نیاز ماست دسترسی داشته باشند و اقدام به دانلود و مطالعه این مقالات نمایند.

دوم : در بیشتر آموزشها ما از زبان **C** استفاده میکنیم و گهگاه و به ندرت از زبان **C++** .پس آشنایی داشتن با این دو زبان جهت مطالعه آموزشهای ما لازم میباشد.

ترسیم یک مثلث ساده با استفاده از**openGl** .

ایجاد یونیت تشریفاتی برای برپایی فرمت نقطه ای

قبل از هر چیزی برای اینکه تکرار مکررات در تمام مثالها صورت نگیرد و همانطور که در مقاله ی اول نیز ذکر گردید در برنامه نویسی **OpenGL** تحت ویندوز اولین کاری که باید صورت گیرد برپایی فرمت نقطه ای می باشد پس بهتر است یک یونیت تحت همین نام ایجاد نماییم تا به دفعات از آن در سورس های دیگر نیز استفاده گردد.

سورس کامل این یونیت تشریفاتی به صورت زیر بوده و این قاعده در تمام برنامه های ما تکرار و رعایت خواهد گردید و تحت عنوان **SPF** به برنامه ها الحاق می گردد:

کد

```
  unit SPF; // setup pixel format

interface
uses     { uses clause }
  Windows ;
var
  hrc: HGLRC;  // Permanent Rendering Context
```

```
procedure CleanUp(Handle: HDC); //Properly Kill The Window
procedure SetDCPixelFormat(Handle: HDC;ColorBits,DepthBufferBits:integer);

implementation

procedure CleanUp(Handle: HDC); //Properly Kill The Window
begin
  if hrc&lt;> 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;

end.
```

ترسیم یک مثلث ساده با استفاده از **opengl** .

ایجاد یونیت تشریفاتی برای برپایی فرمت نقطه ای

قبل از هر چیزی برای اینکه تکرار مکررات در تمام مثالها صورت نگیرد و همانطور که در مقاله ی اول

نیز ذکر گردید در برنامه نویسی **OpenGL** تحت ویندوز اولین کاری که باید صورت گیرد برپایی فرمت نقطه ای می باشد پس بهتر است یک یونیت تحت همین نام ایجاد نماییم تا به دفعات از آن در سورس های دیگر نیز استفاده گردد.

سورس کامل این یونیت تشریفاتی به صورت زیر بوده و این قاعده در تمام برنامه های ما تکرار و رعایت خواهد گردید و تحت عنوان **SPF** به برنامه ها الحاق می گردد:

کد:

```
unit SPF; // setup pixel format

interface
uses     { uses clause }
   Windows ;
var
  hrc: HGLRC;  // Permanent Rendering Context

procedure CleanUp(Handle: HDC); //Properly Kill The Window
procedure SetDCPixelFormat(Handle: HDC;ColorBits,DepthBufferBits:integer);

implementation

procedure CleanUp(Handle: HDC); //Properly Kill The Window
begin
  if hrc&lt;> 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;

end.
```

ترسیم یک مثلث ساده با استفاده از**openGL** .

با توجه به اینکه احتمالا مطالب عنوان شده در قسمت اول کمی پیچیده بودند در طی چند مثال که به تدریج ارائه خواهد شد این مطالب بیشتر موشکافی خواهند گردید.در این مثال قصد داریم یک مثلث ساده را با استفاده از **OpenGL** ترسیم کنیم. در اینجا با نحوه ی استفاده از توابع – **glViewport** **glMatrixMode – glLoadIdentity – gluPerspective- glClear - gluLookAt – glBegin - glEnd**بیشتر آشنا خواهیم شد . یک پروژه جدید در دلفی ایجاد نموده و یونیت **OpenGL** استاندارد دلفی و یونیت **SPF** را که قبلا در طی مقاله ای راجع به آن برای برپایی فرمت نقطه ای صحبت گردید را به برنامه الحاق نمایید یعنی:

کد:

```
unit ex01;

interface

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

قاعده ی دیگری که در تمام برنامه های **OpenGL** مشترک است ایجاد یک متغیر برای دریافت **HDC** فرم می باشد و پاس کردن آن به توابع موجود در یونیت **SPF** برای برپایی فرمت نقطه ای با استفاده از آن.

یعنی در ادامه خواهیم داشت:

کد:

```
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

var
 f_Hdc : LongInt;
```

حالا روی فرم دوبار کلیک نمایید تا کار برپا سازی فرمت نقطه ای را در اینجا انجام دهیم . یعنی:

کد:

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

از برگه ی خواص رخدادهای مربوط به فرم را انتخاب نمایید و در قسمت **OnDestory** مربوط به فرم دوبار کلیک کنید تا روال رخداد آن آماده شود . از آن برای پاکسازی برنامه استفاده خواهیم کرد (یک تعهد اخلاقی !) . یعنی:

کد:

```
procedure TForm1.FormDestroy(Sender: TObject);
begin
```

```
  CleanUp(f_Hdc);// Clean up and terminate.
end;
```

استاندارد دیگری که در تمام برنامه های **OpenGL** رعایت می شود بدین صورت است که یک تابع **InitGL**تعریف می گردد تا مقدمات کاربرنامه نویسی مانند تعریف رنگها و نحوه ی سایه زدن و امثال اینها که عموما در طی برنامه تغییر نمی کنند در آن گنجانده شود . این مورد در روال رخداد **FormCreate**همانطور که دقت کردید کاربرد دارد.

کد:

```
procedure InitGL;          // All Setup For OpenGL Goes Here
begin
// select clearing color
   glClearColor( 0.0, 0.0, 0.0, 0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
end;
```

قاعده ی دیگری که بازهم بصورت استاندارد بکار گرفته می شود برای خوانایی بیشتر کدهای **OpenGL** ایجاد تابع **DrawGLScene** برای نوشتن تمام دستورالعمل های ترسیمات برنامه است. در اینجا ما می خواهیم یک مثلث را ترسیم نماییم . بدین صورت:

کد:

```
procedure DrawGLScene ();
// Here's Where We Do All The Drawing!!!
begin
// سبب پاک شدن صفحه و عمق بافر می شود
          glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
//  ماتریس های  مورد استفاده و در نتیجه صفحه را ریست می کند
          glLoadIdentity();
// بدین وسیله محل قرار گیری دوربین را برای نمایش مثلث مشخص میکنیم
          gluLookAt(0, 0, 6,     0, 0, 0,     0, 1, 0);

          // The position has an X Y and Z.  Right now, we are standing at (0, 0, 6)
          // The view also has an X Y and Z.  We are looking at the center of the
axis (0, 0, 0)
          // The up vector is 3D too, so it has an X Y and Z.  We say that up is (0, 1,
```

```
0)
        // Unless you are making a game like Descent(TM), the up vector can stay
the same.

//در اینجا ما به اپن جی ال می گوییم کی میخواهیم ترسیم یک مثلث را شروع کنیم
        glBegin (GL_TRIANGLES);
// در ادامه سه راس مثلث را برایترسیم مشخص می نماییم
            glVertex3f(0, 1, 0);
             glVertex3f(-1, 0, 0);
             glVertex3f(1, 0, 0);
// و در اینجا خاتمه ترسیم را اعلام مینماییم
        glEnd();
        // You can have as many points inside the BEGIN and END, but it must be
in three's.
        // Try GL_LINES or GL_QUADS.  Lines are done in 2's and Quads done in
4's.

        SwapBuffers(f_Hdc);
//استفاده از تابع فوق الزامی است. در غیر اینصورت چیزی روی صفحه ترسیم نخواهد شد
end;
```

برای اینکه مطمئن شویم پنجره ی برنامه ما در هر حالتی دوباره ترسیم خواهد شد رخداد **OnPaint** را باید به برنامه اضافه کرد:

کد:

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

با توجه به اینکه فرم برنامه می تواند به هر اندازه ای از طرف کاربر تغییر کنید به رخداد **OnResize** هم نیاز داریم:

کد:

```
procedure TForm1.FormResize(Sender: TObject);
begin
        if (height=0) then
// در اینجا از تقسیم شدن ارتفاع صفحه بر صفر که در توابع بعدی استفاده می شوند جلوگیری میشود
          height:=1;

// در این حالت دریچه ی دید را کل صفحه انتخاب می نماییم
        glViewport(0,0,width,height);
        // The glViewport takes (x, y, width, height)
        // This basically means, what our our drawing boundries

        glMatrixMode(GL_PROJECTION);  // Select The Projection Matrix
        glLoadIdentity(); // Reset The Projection Matrix


        // The parameters are:
        // (view angle, aspect ration of the width to the height,
        //  The closest distance to the camera before it clips,
       // FOV      // Ratio  //  The farthest distance before it stops drawing)
        gluPerspective(45.0,width/height, 1 ,150.0);

        // * Note * - The farthest distance should be at least 1 if you don't want
some
        // funny artifacts when dealing with lighting and distance polygons.  This
is a special
        // thing that not many people know about.  If it's less than 1 it creates
little flashes
        // on far away polygons when lighting is enabled.

        glMatrixMode(GL_MODELVIEW);            // Select The Modelview Matrix
        glLoadIdentity();  // Reset The Modelview Matrix
```

```
end;
```

در این برنامه با روال معمول ایجاد یک برنامه (**OpenGL** در هر سطحی) و با رخدادهای کلیدی آن آشنا شدید. تمام اینگونه رخدادها در برنامه های **OpenGL** تکرار می شوند و مرور آنها لازم به نظر می رسید.

باید اعتراف کنم که سرعت دلفی واقعا به قول خارجی ها **Amazing** است !! (مخصوصا در مقایسه با این وی ژوال استودیوی جدید مایکروسافت که آدم را روی یک پنتیوم فور خفه می کند**!**

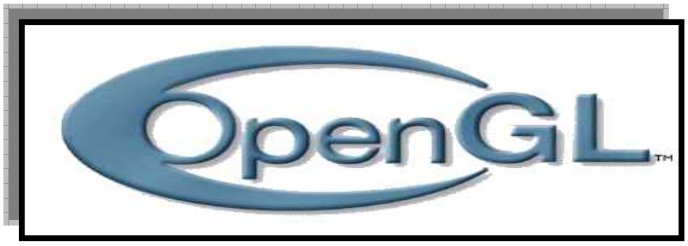
فرض کنید که می خواستیم دو مثلث را ترسیم کنیم . کد حاصل به صورت زیر در می آمد:

کد:

```
//          glBegin (GL_TRIANGLES);
//                    glVertex3f(0, 1, 0);        // Top point
//                    glVertex3f(-1, 0, 0);       // Bottom right point
//                    glVertex3f(1, 0, 0);    // Bottom left point
//
//                    glVertex3f(0, 1, 1);        // Top point
//                    glVertex3f(-1, 0, 1);       // Bottom right point
//                    glVertex3f(1, 0, 1);    // Bottom left point
//          glEnd();

// دید سه بعدی در اپن جی ال به صورت زیر است //
//
//                                  Y
//                                  |
//                                  |
//                                  |_________ X
//                                   \
//                                     \
//                                           Z
// بنابراین مثلث ما به صورت زیر ترسیم می شود//
//                                  Y
//                                  |
//                            /|\
//                           /_|_\_______ X
//                                   \
//                                     \
//                                           Z
// در مورد ماتریس ها در طی مقالات آتی بیشتر توضیح داده خواهد شد //
// در هر حال برای برنامه نویسی اپن جی ال نیازی به دانش بردارها و ماتریسها در حد ابتدایی است //
```

A Day Without Learning Is A Lost Day



آشنایی با DirectX و OpenGL

اشاره :

سوار AGPیک بازی کامپیوتری را روی کامپیوترتان اجرا می‌کنید. فعلاً کارت گرافیک شما روی اسلات می‌شود، پردازشگر سلرون دارید و ... پس از چند ماه یا چند سال کامپیوتر جدیدی می‌خرید. اکنون است و یک پردازشگر 64 بیتی دارید. همان بازی را روی PCI Expressاسلات کارت گرافیکی شما این کامپیوتر هم نصب و اجرامی‌کنید! شاید به نظر طبیعی میآید که همه چیز باید همین‌طور باشد. اما چگونه یک بازی روی کامپیوترهایی با تراشه‌ها و سخت‌افزارهای مختلف و گاه فناوری متفاوت اجرا های گرافیکی یا همان رابط‌های برنامه‌نویسی، بخش بزرگی از این مشکل را حل می‌کنند و APIمی‌شود؟ امکانات گسترده دیگری را نیز در اختیار برنامه‌نویسان و توسعه‌دهندگان بازی و برنامه‌های چندرسانه‌ای گرافیکی و صوتی هستند که برای آسان‌تر ساختن API، دو مجموعه DirectX وOpenGLقرارمی‌دهند. توسعه بازی‌ها و نرم‌افزارهای چندرسانه‌ای

طراحی شده‌اند.

APIگرافیکی چیست؟

API درواقع بین برنامه و سخت‌افزاری که برنامه روی آن اجرا می‌شود، نقش یک هماهنگ‌کننده را دارد و

مانند پلی میان سخت‌افزار و نرم‌افزار ارتباط ایجادمی‌کند. یعنی برنامه‌نویس کدهایی می‌نویسد که داده‌های گرافیکی خود را به وسیله دستورهای استانداردی به درایور **API** می‌فرستد نه مستقیماً به خود سخت‌افزار. سپس درایوری که شرکت سازنده سخت‌افزار تولیدکرده است، این کداستاندارد تولیدشده را به فرمت بومی و ویژه‌ای که برای آن مدل خاص سخت‌افزار قابل شناسایی است، ترجمه می‌کند.

Microsoft DirectX

شرکت مایکروسافت در سال **1995** **DirectX** را ساخته و توسعه داده‌است. این نرم‌افزار شامل مجموعه یکپارچه‌ای از ابزارهای برنامه‌نویسی است که به توسعه‌دهندگان امکان می‌دهد انواع مختلف نرم‌افزارهای مالتی‌مدیا را روی پلتفرم ویندوز تولید کنند. **DirectX** به برنامه‌ای که بر پایه آن طراحی شده امکان می‌دهد به آسانی قابلیت‌های سخت‌افزار کامپیوتر را شناسایی کند و پارامترهای برنامه را با آن هماهنگ سازد.

DirectX شامل **API**هایی است که دسترسی به بخش‌های ویژه‌ای از سخت‌افزار مانند تراشه‌های شتاب‌دهنده گرافیک سه‌بعدی و کارت صوتی را میسرمی‌کند. این **API**ها کنترل توابع سطح پایین، یعنی نزدیک به سخت‌افزار، شامل شتاب‌دهنده گرافیکی دو بعدی، پشتیبانی از دستگاه‌های ورودی مانند دسته بازی، صفحه‌کلید و ماوس، و کنترل میکس و خروجی صدا را انجام می‌دهند.

DirectX 7.0 در سال **1999** با شش کامپوننت عرضه شد که عبارت بودند از: **Direct3D**، **DirectDraw**، **DirectSound**، **DirectPlay**،**DirectInput** و **DirectMusic**. در اواخر سال **2000** میلادی، **DirectX 8.0** عرضه شد که در آن کامپوننت‌های **DirectSound** و **DirectMusic** با هم ادغام شدند و با نام کامپوننت **Audio Direct** معرفی شدند.

Direct3D و **DirectDraw** نیز با هم ادغام شدند و یک کامپوننت با نام **DirectX Graphics** را ساختند. **DirectShow** نیز به صورت یک **API** جداگانه پیاده‌سازی شد و به یکی از کامپوننت‌های **DirectX** تبدیل گردید.

DirectX 9.0 در ژانویه سال **2003** عرضه شد. ویژگی‌های خاص این نسخه عبارتند از:

قابلیت‌های صوتی جدید در **DirectSound**

سخت‌افزار رندرکننده ویدیویی با شتاب بیشتر

بهبود قابلیت برنامه‌ریزی گرافیکی

APIهای همه کامپوننت‌های **DirectX** برپایه **COM** یا **Component Object Model** هستند. در ادامه به بررسی هفت کامپوننت **DirectX 9.0** می‌پردازیم که عبارتند از: **Direct3D**،**DirectDraw** ، **DirectShow**، **DirectSound**، **DirectMusic**،**DirectInput** و **DirectPlay**.

1- DirectDraw

DirectDraw، کامپوننتی ویژه طراحی دوبعدی است که به برنامه‌نویس اجازه می‌دهد مستقیماً به حافظه کارت گرافیک دسترسی یابد، صحنه‌ها و فریم‌ها را با هم ترکیب نماید یا **bitmap**ها را در آنجا ذخیره کند. همچنین، برای برنامه‌ها امکان دسترسی به سخت‌افزارهای ویژه نمایش را مستقل از نوع سخت‌افزار فراهم می‌کند. هر برنامه کاربردی **DirectDraw** الگوی یکسانی دارد که عبارت است از:

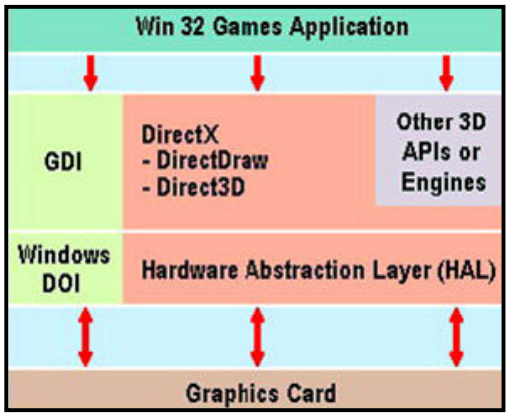
- ایجاد یک شی
- شروع حلقه
- انتقال به مانیتور
- پایان حلقه
- پاک کردن آن شی

منظور از واژه <یک شی> می‌تواند هر تصویر دوبعدی‌ای باشد و منظور از حلقه، حلقه‌ای است که در برنامه‌نویسی هنگام تکرار منظم دسته‌ای از داده‌ها یا دستورها به کار می‌بریم. تصویر ایجادشده پس از مدتی پاک می‌شود و جای خود را به تصویر دیگری می‌دهد.

Direct3D-2

این کامپوننت، دسترسی به توابع رندرکننده گرافیک سه‌بعدی تعبیه شده در بیشتر کارت‌های گرافیک را فراهم می‌کند. **Direct3D** یک **API** سطح پایین سه‌بعدی است که به نرم‌افزار امکان می‌دهد مستقل از سخت‌افزار، با سخت‌افزار شتاب‌دهنده ارتباط برقرار کند. لایه‌ای که برای توسعه‌دهندگان بازی و گرافیک کامپیوتری امکان طراحی و ساخت بازی‌ها را مستقل از سخت‌افزار کامپیوترها فراهم می‌کند، لایه‌ای به نام **HAL** (**Hardware Abstraction Layer**) است.

HAL با قابلیت‌هایی که به صورت گسترده در سخت‌افزارهای گرافیک سه‌بعدی پیاده‌سازی شده‌اند ارتباط ایجاد می‌کند و به سازندگان امکان‌می‌دهد درایورهایی را تولید کنند که لایه **HAL** را به سخت‌افزار پیوند دهد. این کار باعث می‌شود برنامه‌های کاربردی **D3 Direct** بدون این‌که برای نوع خاصی از قطعه سخت‌افزاری نوشته شده باشد، از ویژگی‌های بخش‌های خاص آن قطعه سخت‌افزاری بهره‌ببرد. در شکل یک چگونگی ارتباط لایه **HAL** با سخت‌افزار و نرم‌افزارهای مرتبط نشان داده شده است.



شکل **1** - چگونگی ارتباط لایه **HAL** با کارت گرافیک و نرم‌افزارهای مرتبط

همان‌گونه که در شکل یک، نشان داده شده، نرم‌افزار بازی بالاترین سطح است و پس از آن کامپوننت‌های ترسیم دوبعدی و سه بعدی، یعنی **DirectDraw** و **Direct3D** قرار دارند. لایه **HAL** یک رابط میان کامپوننت‌های **DirectX** و کارت گرافیک است.

در سیستم رندر **Direct3D**، ساختار اشیای سه‌بعدی پیش از آن‌که شتاب‌دهنده سه‌بعدی، یک صحنه سه‌بعدی را رندر نماید و آن را به مانیتور منتقل کند، به وسیله **PU** پردازش می‌شود. نسخه ششم کامپوننت **Direct3D** از قابلیت‌های کارت‌های گرافیک جدیدتر پشتیبانی می‌نماید و در هر گذر، چندین بافت را با هم رندر می‌کند.

کاهش زمان رندر به استفاده از نقشه بافت‌ها نیاز دارد. این نسخه تکنیک‌هایی برای افزودن جلوه‌ای واقعی‌تر به صحنه‌های سه بعدی را نیز دربردارد.

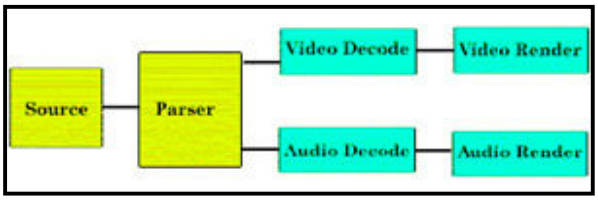
مانند **anistropic filtering** که عنصر عمق را به **trilinear filtering** و نقشه برجسته‌سازی می‌افزاید که موجب ایجاد شباهت بیشتر بافت‌ها و نیز منابع نور تابیده شده بر سطوح مسطح با نمونه‌های واقعی آن‌ها می‌شود. نسخه هفتم **DirectX** نسبت به نسخه‌های پیش از خود بیست درصد سریع‌تر و شامل چند ویژگی دیگر بود. مهم‌ترین آن‌ها پشتیبانی از تغییرات شتاب سخت‌افزاری و نوردهی (**L&T**) به وسیله اغلب کارت‌های گرافیک سه‌بعدی آن‌زمان به ویژه کارت‌هایی است که برپایه تراشه‌های **nVidia Geforce 256** و **S3 Savage 2000** ساخته شده‌اند. از زمانی که **L&T** عرضه شد، وقت‌گیرترین وظیفه **CPU** هنگام اجرای بازی‌های پیشرفته به شتاب‌دهنده سه‌بعدی داده شد و بخش بزرگی از ظرفیت پردازنده اصلی به کارهای دیگر مانند هوش‌مصنوعی بازی اختصاص داده شد و توسعه‌دهندگان بازی توانستند رندر را با جزئیات بیشتر انجام دهند و جلوه‌های ویژه پیچیده‌تری را در بازی‌ها به‌کار ببرند.

3- DirectShow

این کامپوننت از بسیاری از فرمت‌های صوتی و ویدیویی شامل **AVI**، **MPEG**، **ASF**، **WMA/WMV**، **DV** و **MP3** و **DirectX** پشتیبانی می‌کند و روی ویندوزهای **98**، **2000**، اکس‌پی و نرم‌افزار اینترنت اکسپلورر عرضه شده است.**DirectShow** پروسه کارهای مالتی‌مدیا مانند نمایش فایل ویدیویی را به مجموعه‌ای از مراحل که با نام**filter** شناخته می‌شوند تقسیم می‌کند.

فیلترها تعدادی **pin** ورودی و خروجی دارند که آن‌ها را به هم متصل می‌کند. طراحی کلی سازوکار اتصال به این صورت است که فیلترها می‌توانند به روش‌های مختلف به هم متصل شوند که هر نوع از این اتصال‌ها به معنی انجام دادن یک کار است و توسعه‌دهندگان نرم‌افزار می‌توانند افکت‌های خود یا فیلترهای دیگری را به بخشی از این گراف برای انجام کار ویژه‌ای بیفزایند. گراف فیلتر **DirectShow** به صورت گسترده در ضبط صدا و فیلم، و ویرایش آن‌ها به کار می‌رود.

در شکل دو، یک گراف نمایش برای فایل فیلمی از نوع **MPEG** نشان داده شده است. برنامه‌های کاربردی **DirectShow**، برای پردازش داده‌های مالتی‌مدیا، از این گراف استفاده می‌کنند.



شکل 2 – یک گراف فیلتر که کار نمایش یک فایل **MPEG**را نشان می‌دهد .

داده‌های چند رسانه‌ای در این گراف (در حالی که کارها به وسیله برنامه کاربردی کنترل می‌شوند) از فایل منبع به سمت مقصد که می‌تواند یک قطعه سخت‌افزاری باشد حرکت می‌کنند.

ولی در برخی مواقع، برنامه کاربردی علاوه بر کنترل گراف، دریافت‌کننده یا فرستنده داده نیز هست. هر گره این گراف، همانگونه که گفته شد، یک فیلتر است و کار ویژه خود را انجام می‌دهد. فیلتر **source**، داده‌ها را از یک فایل یا **URL** می‌خواند. فیلتر **Parser**، بخش‌هایی از داده‌های صوتی و ویدیویی را به رمزگشای مناسب می‌فرستد. رمزگشاها، داده‌های صوتی و ویدیویی را رمزگشایی می‌نمایند یا از حالت فشردگی خارج می‌کنند. فیلتر رندرکننده، داده‌های دریافت شده صوتی و ویدیویی از رمزگشا را پخش می‌کند یا آن‌ها را نمایش می‌دهد.

4- DirectSound

این کامپوننت همزمان با ساخت ویندوز **95**، زمانی که درایورهای صوتی از نوع **VXD** بودند به **DirectX** افزوده شد. در این کامپوننت **API**های ویژه‌ای ایجاد شد که نویسندگان درایورهای صوتی می‌بایست آن‌ها را به محصولات خود، که فرمت **VXD** داشت، می‌افزودند تا به درستی با **DirectSound** کار کند.

برنامه‌های چندرسانه‌ای با این کامپوننت به سخت‌افزارهای صوتی مانند کارت صوتی دسترسی پیدامی‌کنند. از مهم‌ترین ویژگی‌های این **API**، ترکیب صدا و کنترل سطح آن است.

DirectSound همچنین اجازه می‌دهد چندین برنامه کاربردی، بدون پیش آوردن وقفه، همزمان به کارت صوتی دسترسی داشته باشند. ایجاد افکت‌های صوتی از دیگر توانایی‌های **DirectSound** است. پس از سال‌ها توسعه، اکنون **DirectSound** یک **API** پخته و کامل است و بسیاری قابلیت‌های دیگر را نیز فراهم می‌کند؛ مانند قابلیت پخش صداهای چند کاناله با وضوح و دقت بالا.

5- DirectMusic

تاکنون بازی‌هایی را تجربه کرده‌اید که در تمام مدت یک مرحله، موسیقی یکنواخت و ثابتی دارند؟ بازی‌ای را در نظر بگیرید که برنامه‌نویسان آن می‌خواهند یک آهنگ، در تمام مدت، در یک مرحله از آن به صدا دربیاید. با استفاده از برنامه **DirectMusic Producer**، آن‌ها می‌توانند در آن مرحله برای آهنگ، یک درجه در نظر بگیرند.

این درجه می‌تواند بسته به نوع عملکرد شخصیت بازی، تغییر کند. اگر شخصیت بازی در حال راه رفتن است، آهنگ آرام و هنگامی که با دشمن خود مبارزه می‌کند، آهنگ تندتر می‌شود و یا نوع آهنگ تغییر می‌کند و هنگامی که مبارزه تمام می‌شود، آهنگ دوباره آرام می‌شود. این تغییرها بدون ایجاد وقفه، به صورت پویا و بدون دخالت کاربر انجام می‌شود. چون براساس **DirecMusic**، آهنگ به صورت شناور و بدون وقفه با نواختن واریاسیون‌های مختلف با قابلیت واکنش به رویدادهای بازی تولید می‌شود.

DirectMusic، با داده‌های موسیقی براساس پیام‌های حاوی اطلاعات کار می‌کند. یک آهنگ می‌تواند در داخل سخت‌افزار و با نرم‌افزارهای آهنگ‌ساز مانند **Microsoft Synthesizer** ساخته شود.

DirectMusic از استانداردهای**MIDI** و **DLS** پشتیبانی می‌کند.

6- DirectInput

این کامپوننت، سازوکار مشترکی را برای دسترسی به بسیاری از کنترل‌کننده‌های بازی مانند دسته بازی، گیم‌پد، صفحه کلید و ماوس فراهم می‌آورد. مهم‌ترین تغییری که هنگام عرضه **DirectX8** در **DirectInput** ایجاد شد، آمدن**action map** بود. **action map** از توابعی مانند راندن یک وسیله یا شلیک یک گلوله (که به‌وسیله دستگاه‌های ورودی ایجاد می‌شود) استفاده می‌کند. زمانی که یک سخت‌افزار ورودی مانند دسته بازی را می‌خرید، معمولاً **action map**نیز برای بسیاری از انواع رایج بازی‌ها مانند شبیه‌ساز پرواز، تیراندازی اول شخص و بازی‌های مسابقه‌ای در آن پیاده‌سازی شده است.

7- DirectPlay

این کامپوننت امکان بازی چند نفر را در بازی‌های چندنفره فراهم می‌آورد، دسترسی به سرویس‌های ارتباطی را آسان می‌سازد و راهی را برای بازی‌ها فراهم می‌کند تا مستقل از پروتکل یا نوع سرویس آنلاین

با یکدیگر در ارتباط باشند. همچنین از پروتکل‌های ارتباطی مطمئن پشتیبانی‌می‌کند تا مانع از گم شدن داده‌های مهم بازی روی شبکه شود. در واقع **DirectPlay** به صورت لایه‌ای است که روی پروتکل‌های معمول شبکه مانند **TCP/IP،IPX** و ... قرار دارد.در واقع یک **session** یا جلسه در **DirectPlay** یک کانال ارتباطی بین چندین کامپیوتر است. یک برنامه کاربردی پیش از آن‌که بتواند با سیستم‌های دیگر ارتباط برقرار کند، باید در یک **Session** یا جلسه باشد. هر جلسه تنها یک میزبان دارد و آن برنامه کاربردی‌ای است که آن جلسه را ایجاد کرده‌است. تنها میزبان می‌تواند ویژگی‌های یک **Session** را تغییر دهد.

DirectX 9.0

این کامپوننت، آخرین نسخه **DirectX** تا پیش از عرضه رسمی ویندوز ویستا است. مهم‌ترین چیزی که همراه **DirectX 9.0** عرضه شد، **HLSL (High-Level Shader Language)** است. زبان **HLSL** جایگزین زبان اسمبلی برای نوشتن **pixel shader**ها و **vertex shader**ها در **DirectX** است. پیش از ارائه **DirectX 9.0** توسعه‌دهندگان بازی باید**shader**ها را با استفاده از یک زبان اسمبلی سطح پایین توسعه می‌دادند. **HLSL** با فراهم‌آوردن یک محیط برنامه‌نویسی توسعه‌دهنده ساده، توسعه همه بخش‌های نرم‌افزار مانند انیمیشن و برنامه‌نویسی افکت‌ها را آسان می‌کند.

HLSL با همه پردازشگرهای گرافیکی (**GPU**) سازگار با **DirectX** کار می‌کند و به توسعه‌دهندگان امکان می‌دهد افکت‌های بصری را روی گستره وسیع‌تری از پلتفرم‌ها ایجاد کنند؛ بدون این‌که نیاز داشته باشند به جزئیات سخت‌افزار گرافیکی توجه کنند. **DirectX 9.0** روی ویندوز **95** نصب نمی‌شود. چون بازی‌هایی که به **DirectX 9.0** نیاز دارند، به کامپیوترهای جدیدتر و قوی‌تری هم نیاز دارند که ویندوز **98** یا نسخه‌های جدیدتر روی آن‌ها نصب می‌شود. تاکنون نسخه‌های **a،b** و **c** از **DirectX 9.0** ارائه شده است. هر نسخه جدیدتر از **DirectX** دارای امنیت، کارایی و سیستم رفع خطای بهتری است.

DirectX 10

دوستداران بازی باید خوشحال باشند از این‌که بدانند شرکت مایکروسافت **DirectX** را نیز تولید کرده است و همراه پیش توزیع **Direct3D 10** عرضه خواهد شد. همچنین نرم‌افزار **Microsoft Windows Explorer Game** نیز عرضه شده که به برنامه‌نویسان و توسعه‌دهندگان امکان می‌دهد امکانات بروزکردن

خودکار (**auto-updating**) را به بازی‌هایشان بیفزایند. مایکروسافت می‌خواهد **DirectX 9.0** و **DirectX 10** را روی ویندوز ویستا عرضه کند. به گفته **Rodolph Balaz** از برنامه‌نویسان توسعه‌دهنده **Direct3D** و **OpenGL** در مایکروسافت، **DirectX 10** تنها با سیستم‌عامل‌های جدید کار خواهد کرد و در حال حاضر مایکروسافت، برنامه‌ای برای پشتیبانی ویندوز اکس‌پی از آن ندارد.تا زمان نوشته شدن این مقاله هنوز نسخه رسمی ویندوز ویستا عرضه نشده است. ولی به نظر می‌آید این ویندوز، هم از **DirectX 10** و هم از **DirectX 9.0** پشتیبانی خواهد کرد.

SGL OpenGL

شرکت سیلیکون گرافیکس(**OpenGL،SGI**) را با هدف ساخت یک **API** برای توسعه برنامه‌های گرافیکی دوبعدی و سه بعدی عرضه‌کرده‌است. پیش از ساخته شدن **API**های گرافیکی مانند **OpenGL** و **DirectX**، بسیاری از تولیدکنندگان سخت‌افزار، کتابخانه‌های گرافیکی مختلف و متفاوتی داشتند. به همین دلیل پشتیبانی از نسخه‌های مختلف نرم‌افزارهایشان روی پلتفرم‌های سخت‌افزاری مختلف هزینه‌بر و انتقال یک برنامه کاربردی از یک پلتفرم سخت‌افزاری به پلتفرم سخت‌افزاری دیگر بسیار وقت‌گیر و سخت بود. بنابراین **SGI** نمونه برنامه‌ای را تولید کرد که تولیدکنندگان سخت‌افزار باید از آن برای توسعه درایورهای **OpenGL** در سخت‌افزارهایشان استفاده کنند. این برنامه به صورت اپن‌سورس ارائه شده‌است. ولی سازندگان این سخت‌افزارها می‌توانند قابلیت‌های گوناگونی را برپایه **OpenGL** در سخت‌افزارهایشان ایجاد کنند. تصمیم‌گیری درباره ایجاد تغییرات در **OpenGL** را کنسرسیوم **ARB** اتخاذ می‌کند.

این کنسرسیوم شامل اعضای مهمی همچون اپل، اینتل، آی‌بی‌ام، سان، **ATI**، دل، **nVIDIA**، سیلیکون‌گرافیکس و**Dlabs3** است و از سوی شرکت‌های معتبر دیگری مانند متراکس، **S3**،**Xi** و **Quantum 3D** حمایت می‌شود. توسعه‌دهندگان نرم‌افزار برای استفاده از **OpenGL** در نرم‌افزارهایشان نیازی به اخذ مجوز ندارند. ولی تولیدکنندگان سخت‌افزار برای پیاده‌سازی سخت‌افزاری **OpenGL** نیازمند اخذ مجوز از **SGI** هستند.

OpenGLچیست؟

در اوایل پیدایش **OpenGL**، از این **API** در کارهای صنعتی، طراحی وسایل داخلی، مکانیکی و نیز در آنالیزهای علمی و آماری استفاده می‌شد.در سال **1996**، نویسندگان و توسعه‌دهندگان بازی‌های کامپیوتری

از نسخه ویندوزی **OpenGL** برای ساخت بازی‌های کامپیوتری استفاده کردند. **OpenGL** برای پشتیبانی از گستره وسیعی از تکنیک‌های رندرکردن گرافیکی پیشرفته طراحی شده است که می‌توان پاره‌ای از آن‌ها را به این‌صورت نام برد:

نورپردازی: قابلیت تحلیل میزان رنگ هنگام تابش مدل‌های متفاوت نور به یک سطح از یک یا چند منبع نور مختلف.

سایه‌سازی نرم: قابلیت تحلیل افکت‌های سایه هنگام تابش نور به یک زاویه و ایجاد اختلاف نور خفیف در مقابل آن سطح (مانند نور کمی که هنگام تابش آفتاب به یک صخره یخی در اطراف آن ایجاد می‌شود).

حرکت محو ومدل‌سازی: توانایی تغییر مکان و اندازه پرسپکتیو یک شی در فضای سه بعدی. مجموعه امکانات **OpenGL** شبیه **Direct3D** است. ولی **API** سطح پایین‌تر آن (نزدیک‌تر به سطح سخت‌افزار) باعث می‌شود کنترل خوبی روی عناصر اصلی ایجاد صحنه‌های سه بعدی مانند اطلاعات سه‌ضلعی‌ها که سلول‌های تشکیل‌دهنده یک مدل سه بعدی هستند داشته باشد. دو سطح پشتیبانی از شتاب‌دهندگی سخت‌افزاری برای **OpenGL** وجود دارد: **installing client ICDs (driver)** که به نوردهی ایجاد تغییر و رسترکردن (تبدیل یک فریم سه بعدی چند ضلعی ذخیره شده در**frame buffer** به یک تصویر کامل با بافت‌ها و نشانه‌های عمق و نور) شتاب می‌دهد و **mini MCs (client server)** که از رسترکردن پشتیبانی می‌کند. **OpenGL 1.4** و **OpenGL 1.5** به‌ترتیب در تابستان **2002** و **2003** معرفی شدند که هر یک امکانات و کاربردهای بیشتری از نسخه‌های پیش از خود داشتند. بزرگ‌ترین آن‌ها **OpenGL Shading Language** بود؛ زبانی ویژه برنامه‌نویسی **vertex-shader** و **pixel-shader** که در صورت نیاز به **OpenGL** الصاق می‌شد. **OpenGLShading Language** زبانی شد که به سرعت در سطح گسترده‌ای مورد پشتیبانی یونیکس، ویندوز، لینوکس و دیگر سیستم‌عامل‌ها برای توسعه‌دهنده گرافیک‌های تعاملی و برنامه‌های کاربردی ترسیمی قرار گرفت.

openGL 2.0

OpenGL 2.0 آخرین نسخه عرضه شده تا اوایل سال **2006** میلادی است. **OpenGL Shader**

Language همراه با این نسخه عرضه شده و بر پایه استاندارد ANSYC طراحی شده است. برخی قابلیت‌های تازه این نسخه عبارتند از:

- سایه‌زنی قابل برنامه‌ریزی به‌وسیله OpenGL Shader Language و APIهای آن. قدرت ایجاد Shader و برنامه‌نویسی اشیا، بخش دیگری از تغییرات ایجاد شده در این نسخه است.
- رندر چندگانه که به shaderهای قابل برنامه‌نویسی امکان می‌دهد در بافرهای خروجی چندگانه در یک گذر مقادیر مختلفی بنویسند.

- بافت‌های دو طرفه، با قابلیت تعریف کاربرد آن بافت برای سطح جلو و پشت یک مدل اولیه که کیفیت حجم سایه و کارایی الگویم‌های رندر هندسی اشیای سخت را ارتقا می‌دهد.
- Spriteهای نقطه که مختصات بافت یک نقطه را با مختصات بافت قرار داده شده در مقابل آن نقطه جابه‌جا می‌کنند و رسم نقاط را در بافت‌های طراحی شده در کامپیوترهای معمولی نیز ممکن می‌سازند.
- بافت‌های Non-power-of-two که برای همه انواع بافت کاربرد دارد که در نتیجه از بافت‌های چهارگوش پشتیبانی می‌نماید و درعمل حافظه کمتری اشغال می‌کند.

openAL

OpenAL، یک API دیگر است که برای ایجاد و مدیریت صداهای سه بعدی در بازی‌های کامپیوتری و دیگر انواع نرم‌افزارها به صورت یک پروژه مشترک میان شرکت Loki Software و Creative ساخته شده است.

کتابخانه این API مجموعه‌ای از صداهای قابل حرکت در فضای سه‌بعدی را مدل‌سازی می‌کند. عناصر اصلی OpenAL شامل یک شنونده، یک منبع و یک بافر است. ممکن است تعداد زیادی بافر وجودداشته باشد که شامل داده‌های صوتی هستند. هر بافر می‌تواند به یک یا چند منبع ضمیمه شود. همیشه یک عنصر شنونده (برای محتوای صوتی) وجود دارد که موقعیت مکانی منبع صوتی که صدای آن شنیده می‌شود را نشان می‌دهد. OpenAL در موتورهای گرافیکی Epic Games Unreal نیز برای ساخت افکت‌های صوتی به کار می‌رود.

OpenGL Performer

OpenGL Performer، رابط برنامه‌نویسی قدرتمند و کاملی است که توسعه‌دهندگان برای شبیه‌سازی بصری از آن استفاده می‌کنند. ابزارهای موجود در آن، توسعه برنامه‌های شبیه‌سازی بصری، طراحی بر اساس شبیه‌سازی، واقعیت مجازی، نرم‌افزارهای علمی، سرگرمی‌های تعاملی، برنامه‌های ویدیویی و طراحی با کامپیوتر را آسان می‌کند. این رابط برنامه‌نویسی به برنامه‌نویسان امکان می‌دهد از قابلیت‌های سیستم به صورت بهینه استفاده کنند. آخرین نسخه این نرم‌افزار **OpenGL Performer 3.2** است.

OpenGL Volumizer

OpenGL Volumizer، یک **API** گرافیکی است که در بخش‌های انرژی، تولید، داروسازی و تجارت کاربرد دارد. این **API** برای انجام کارهای تعاملی با کیفیت بالا و بصری نمودن و شبیه‌سازی یک محیط با استفاده از مجموعه بزرگی از داده‌های حجمی (داده‌هایی که مختصات یک شی در فضای سه بعدی را نشان می‌دهند) طراحی شده است. برای نمونه در نرم‌افزارهای پزشکی برای شبیه‌سازی وضعیت بخش خاصی از بدن، از این نرم‌افزار استفاده می‌شود. **OpenGL Volumizer** آخرین نسخه این **API** تا اوایل سال **2006** میلادی است که بر پایه کتابخانه گرافیکی استاندارد**OpenGL** ساخته شده و شامل رابط کلاس **C++** و قابل‌استفاده در سیستم‌عامل‌های ویندوز و لینوکس **32**بیتی و **64**بیتی است.

OpenGL Multipipe SDK

OpenGL Multipipe SDK یک لایه **API** است که مدیریت برنامه‌های گرافیکی را در زیر سیستم‌ها و ساختارهای گرافیکی چندگانه آسان می‌کند. برنامه‌های کاربردی نوشته شده برپایه این **API** به نرمی و روانی، هم روی سیستم‌های رومیزی تک پردازنده‌ای و هم روی سیستم‌های چند پردازنده‌ای با سیستم‌های گرافیکی قدرتمند اجرا می‌شوند.

نتیجه‌گیری:

همان‌گونه که بیان شد ارتباط بین برنامه‌ها و سخت‌افزاری که آن‌را اجرا می‌کند برعهده **API** است. سازندگان بزرگ نرم‌افزار و سخت‌افزار **API** خاصی را برای برنامه‌های مالتی‌مدیا آماده کرده‌اند که مطرح ترین آن‌ها **DirectX** و **OpenGL** هستند.

منابع ومراجع:

کتاب: **OpenGL SuperBible 3rd Edition**

openGL Distilled.

وبلاگ ها

G:\opengl ba java\3meonline_com

OpenGL & Computer graphics
و وبلاگ آموزش گرافیک کامپیوتری و OpenGL و سایر موضوعات وابسته

و سایتهای **gamedev.net** و **glew.sourceforge.net** و **elf-stone.com/glee.php**

http://persian-designers.com

http://NeHe.gamedev.net

http://wiki.linuxquestions.org/wiki/OpenGL