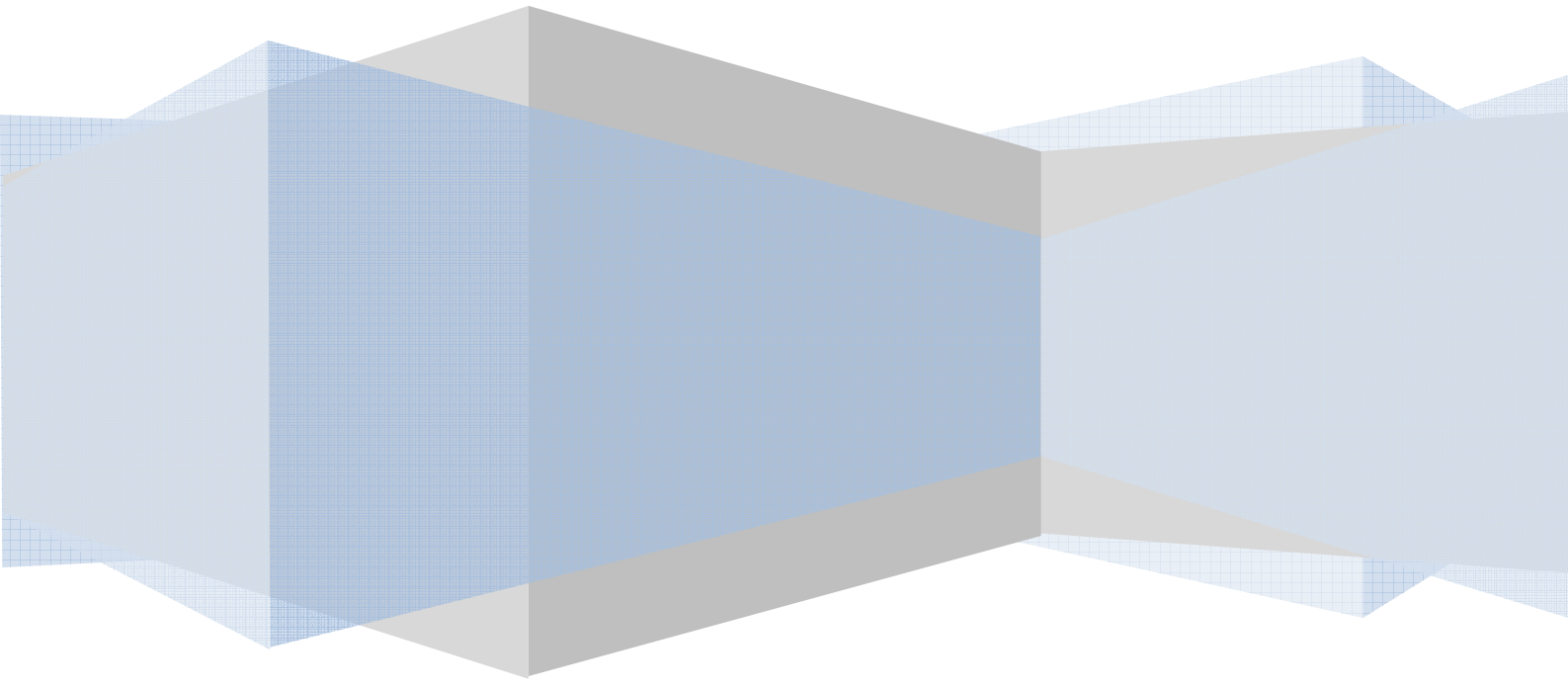
PS-Ware

OpenGL programming in Visual Basic 6

PART 1

By Pooya Shahinfar



فهرست

مقدمه

امروزه گرافیک کامپیوتری در تمام زندگی ما نفوذ کرده است. از تبلیغات مختلف گرفته تا کتابها,مجلات, روزنامه‌ها، متخصصین مختلف از گرافیک کامپیوتری برای برقراری ارتباط با مخاطب خود استفاده می‌کنند. با این روش هم مخاطب بیشتری جذب کرده و هم مفاهیم پیچیده را به زبان ساده‌تر انتقال می‌دهند.

اما در این میان می‌توان افرادی یافت که تمایل به تولید نرم‌افزارهای گرافیکی دارند. این گونه افراد باید با اصول گرافیک کامپیوتری و شیوه‌های ترسیم اشکال گرافیکی در کامپیوتر آشنا باشند، همانند یک نقاش چیره دست یک متخصص گرافیک کامپیوتری ابتدا باید ابزارهای گرافیکی را بشناسد این ابزارها بسته به نوع هدف متخصص می‌توانند, سخت افزارهای گرافیکی و یا روشهای ریاضی و برنامه‌نویسی برای تولید یک تصویر گرافیکی باشد. در طول این کتاب الکترونیکی ما شما را ابتدا با مفاهیم گرافیک کامپیوتری و سپس با برنامه‌نویسی OpenGL که یک رابط گرافیکی است تحت ویژوال بیسیک ۶ آشنا می‌کنیم. در طول قسمت اول این کتاب شما تنها با برنامه‌نویسی مقدماتی تحت OpenGL آشنا خواهید شد. اما فراگیری خوب مطالب این قسمت مقدمه‌ای بر تمام اصول برنامه‌نویسی گرافیکی در OpenGL است بنابراین پیشنهاد می‌شود با دقت تمام , تمامی مطالب این قسمت را دنبال کنید و احیانا در صورتی که به مشکل و یا ایرادی در طول مطالعه برخورد کردید خواهشمندیم آن را سریعا با تیم برنامه‌نویسی PS-Ware در میان بگذارید.

همینجا لازم است از تمامی زحمات و تلاشهای استاد خوبم جناب آقای دکتر مهری تکمه که مرا در این راه تشویق نمود و راهنماییهاشان همواره سرمشق من بود نهایت قدردانی و تشکر را بکنم.

پویا شاهین‌فر
زمستان ۱۳۸۵ – تبریز

فصل ۱ - نگرشی بر سیستم های گرافیکی

همانگونه که پیش از این هم اشاره کردیم شما قبل از اینکه شروع به رسم اشکال با OpenGL بکنید در ابتدا باید با تمامی ابزارها و گرافیکها آشنا شوید. در طول این فصل ما شما را با مفاهیم اولیه گرافیک کامپیوتری آشنا می‌کنیم . سپس در فصول بعدی شروع به نوشتن برنامه‌های گرافیکی با OpenGL در محیط برنامه‌نویسی ویژوال بیسیک ۶ می‌کنیم.

۱.۱ – اصطلاحات و ابزارها

در ابتدا باید گرافیک کامپیوتری را برای خود معنی کنیم : گرافیک کامپیوتری به معنی مجسم کردن, دستکاری اشیاء هندسی با کامپیوتر است.

۱.۱.۱-- گرافیک رستر

نمایش رستر شامل ماتریسی از نقاط به نام پیکسل[۱] (عنصر تصویری[۲]) است. یک پیکسل کوچیکترین واحد قابل آدرس‌دهی در صفحه نمایش است و پایه تمام اعمال گرافیکی در کامپیوتر محسوب می‌شود. صفحه نمایش هر کامپیوتر از تعدادی نقاط ریز به نام پیکسل تشکیل شده به عنوان مثال یک مانیتور می‌تواند ۱۲۸۰ * ۱۰۲۴ پیکسل داشته باشد. در صورتی که این پیکسلها را به صورت یک ماتریس در بیاوریم و سعی کنیم هر شکل را به ذرات کوچکی تقسیم کنیم و هر ذره را معادل یک پیکسل در نظر بگیریم به این نمایش نمایش رستری گفته می‌شود.

مقادیر رنگ تمام پیکسلها در یک مکان به نام بافر چهارچوب ذخیره می‌شود.هر ردیف از پیکسلها یک خط رویش یا یک خط رستری نامیده می‌شود.

۲.۱.۱-- تصاویر

اصلی‌ترین وظیفه یک برنامه گرافیکی تولید تصویر است. در صفحه نمایش رستری کل صفحه به مجموعه‌ای از پیکسلها تقسیم می‌شود. به همین دلیل بدون توجه به این مطلب که تصاویر چگونه در حافظه ذخیره شده‌اند، یک تصویر را با یک آرایه دو بعدی از پیکسلها با کد گذاری رنگی ذخیره می‌کنیم و نمایش می‌دهیم. هرچه یک تصویر را به نقاط ریزتری تقسیم کنیم، وضوح تصویر بیشتر می‌شود. بنابراین برای اینکه وضوح تصاویر قابل مقایسه باشند اصطلاحی به نام **ریزنمایی** مطرح می‌شود. ریز نمایی یک تصویر عبارت است از:

تعداد پیکسل در هر ستون * تعداد پیکسل در هر سطح

مهمترین مساله در نمایش نقاط رنگ آنها است. از روشهای متفاوتی برای کدگذاری رنگ در کامپیوتر استفاده می‌شود که آنها را **حالت رنگ** می‌نامیم. رنگ هر پیکسل به صورت یک عدد دودویی (سطح خاکستری) یا مجموعه‌ای از اعداد دودویی(RGB) کد بندی می‌شود. در تصاویری که بصورت RGB ذخیره می‌شوند، رنگ هر پیکسل دارای سه مولفه قرمز،سبز و آبی است. که تمامی رنگها با استفاده از این سه مولفه نمایش داده می‌شوند. هر یک از این سه مولفه می‌توانند مقادیری را مابین صفر تا ۲۵۶ بگیرند. در صورتی که مقدار صفر به هر مولفه داده شود مقدار رنگ آن مولفه به حداقل و در صورتی که مقدار ۲۵۵ به هر مولفه داده شود مقدار رنگ آن مولفه به حداکثر می‌رسد به عنوان مثال در صورتی که میزان رنگ یک پیکسل را به ترتیب ۲۵۵,۲۵۵,۰ بدهیم رنگ آن پیکسل در صفحه نمایش زرد دیده خواهد شد. در بعضی از کد گذاریها میزان کدر بودن ماده را نیز مشخص می‌کنند به

---

[۱] Pixel
[۲] Picture element

این حالت آلفا[۲] گفته می‌شود. در حالت آلفا در صورتی که یک ماده کاملا کدر باشد و نور را از خود عبور ندهد مقدار ۲۵۵ را خواهد داشت و در صورتی که نور را از خود عبور دهد مقادر کمتر از ۲۵۵ را خواهد داشت، به عنوان مثال ما برای یک شیشه سبز می‌توانیم مقادیر 0,255,0,150 را بدهیم. این حالت رنگ RGBA نامیده می‌شود.

۲.۱ سخت افزار گرافیکی

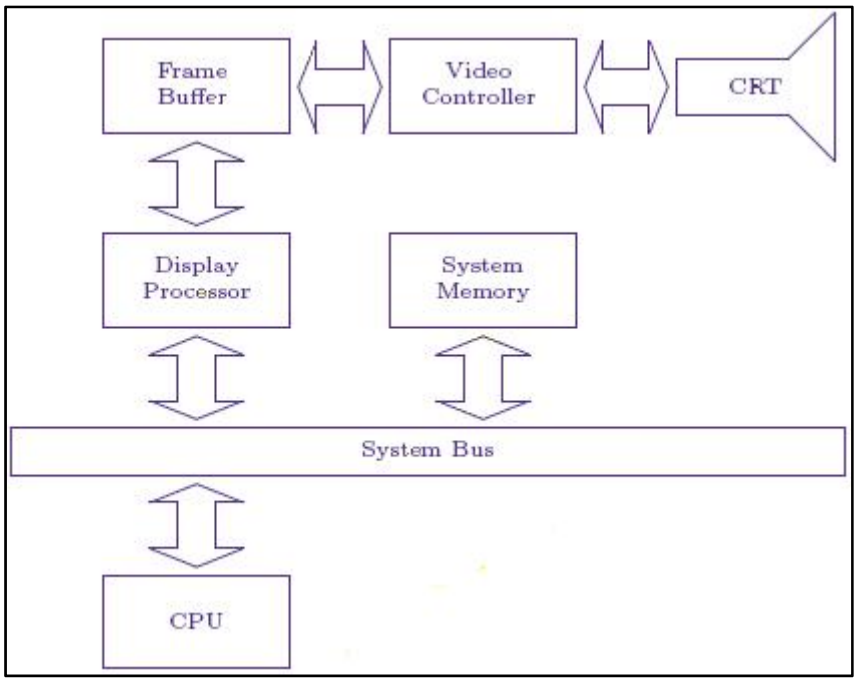
سخت افزار گرافیکی شامل ابزارهای ورودی , خروجی و واحدهای پردازش گرافیک است. برای مثال در یک کامپیوتر، ابزارهای ورودی شامل صفحه کلید و موشواره و ابزار خروجی مونیتور است.

۱.۲.۱— واحدهای داخلی

سیستم‌های پیشرفته گرافیک دارای یک پردازنده گرافیک (VGA) هستند. که علائمی را برای کنترل صفحه نمایش ایجاد می‌کند(شکل ۱.۱) این سخت افزارها می‌توانند به صورت توکار و یا بصورت زیر سیستم تصویری مجزا باشند که در این صورت قابل تعویض خواهند بود. از دید یک برنامه‌نویس کامپیوتری تفاوتی مابین توکار بودن و یا مجزا بدون آنها وجود ندارد. به همین دلیل ما نیز در این کتاب الکترونیکی بدون توجه به نوع آن با نام کارت تصویر به این زیر سیستم اشاره خواهیم کرد.

قبل از اینکه یک کارت تصویر بخواهد یک کاراکتر و یا یک نقطه را بر روی صفحه یک نمایشگر نشان دهد، باید موارد زیر را بداند:

- اینکه چه نویسه و یا نقطه گرافیکی را باید نمایش دهد
- رنگ نویسه و یا نقطه
- محل نویسه و یا نقطه در صفحه نمایش



شکل ۱ سیستم رستری با یک پردازنده نمایش

کارتهای تصویر کامپیوترهای شخصی دارای حافظه‌ای هستند که اطلاعات مربوط به هریک از نقاط یا محلهای صفحه نمایش را در خود دارد. این حافظه، RAM تصویر نام دارد و با RAM کامپیوتر مرتبط است, که امکان مراجعه مستقیم به آن از سوی ریز پردازنده را فراهم می‌کند.

---

[۳] Alpha

هریک از انواع کارتهای تصویر می‌توانند. بیش از یک حالت نمایش داشته باشند: نمایش متن وگرافیک این دو نوع نمایش بسیار متفاوت هستند. در نمایش متنی همانند آنچه که در MS-Dos می‌بینید تمامی کاراکترهای صفحه نمایش حروف هستند اما در نمایش گرافیکی کاراکتر ها را پیکسلها تشکیل داده. برای نمایشگر این دو حالت هیچ تفاوتی باهم ندارند, نمایشگر در حقیقت اطلاعات ارسال شده از توسط کارت تصویر را نشان می‌دهد اما برای یک برنامه‌نویس و یک کارت تصویر این دو حالت کاملا مجزا هستند و نیازمند برنامه‌نویسی کاملا متفاوتی هستند.

در حالت گرافیکی رنگ هر نقطه در یک یا چند بیت نگهداری شده و سپس محتوای RAM بصورت کمابیش مستقیم به صفحه نمایش فرستاده می‌شود. در حالت متنی از روش دیگری استفاده می‌شود. دراین حالت, برای هریک از محلهای صفحه نمایش، کد اسکی یک کاراکتر در RAM نگهداری می‌شود. هنگامیکه که کنترل کننده تصویر اطلاعات را بر روی صفحه نمایش به نمایش در می‌آورد، الگوی مربوط به آن کاراکتر از روی تراشه Ram کارت تصویر می‌خواند و الگوی مربوطه را به نقاط تبدیل می‌کند سپس آنها را به سمت نمایشگر تصویر می‌فرستد.

۱.۲.۳ -- بافر چهارچوب

برای اینکه انسان شکل را درک کند باید مدت زمان نمایش آن در حد کافی و طولانی باشد. بنابراین باید آن را در جایی ذخیره کنیم. رسانه‌هایی مانند چاپگر تصویر را در جایی مانند کاغذ ذخیره می‌کنند. اما برای یک نمایشگر تصویر در حافظه ذخیره شده و این تصویر برای بارها و بارها، این تصویر با سرعت ثابت رسم می‌شود بطوریکه ما آن را به صورت یک تصویر ثابت می‌بینیم.
برای نمایش بر روی یک نمایشگر، آرایه دو بعدی از پیکسلها بر روی حافظه برزگی به نام ذخیره می‌شود. بافر چهارچوب حاوی یک کلمه از حافظه برای هر پیکسل است. که ارزش رنگ هر پیکسل را درون خود ذخیره می‌کند. این کلمه حافظه می‌تواند به قدری کوچک باشد که در سیستمهای تکرنگ یک بیت از حافظه را اشغال کند و یا بقدری بزرگ باشد که هر پیکسل ۲۴ بیت برای سه رنگ اصلی قرمز، سبز و آبی نگهداری کند. در این حالت آن را رنگ واقعی می‌نامیم. بافرهای چهارچوب خاصی نیز وجود دارند که تعداد بیتهای بیشتری را برای رنگها اختصاص می‌دهند و یا یک کانال آلفا برای آمیختن رنگهای شکلهایی از چندین منبع دارند. تعداد بیتهایی که کانال آلفا اشغال می‌کند معادل تعداد بیتهایی است که هرکدام از رنگهای RGB استفاده می‌کنند که همان ۸ بیت است بنابراین در این سیستمها هر کلمه از حافظه بافر چهارچوب، ۳۲ بیت است.
یک بافر چهارچوب می‌تواند شامل حجم قابل توجهی از حافظه باشد. به عنوان مثال یک کارت تصویر استفاده شده برای طراحی به کمک کامپیوتر (CAD) می‌تواند تصویری از یک شکل با ابعاد ۱۰۲۴ *۱۲۸۰ با ۲۴ بیت برای هر پیکسل نمایش دهد. یعنی ۴ مگابایت از حافظه فقط برای نمایش به کار می‌رود. همچنین از آنجا که نوسازی صفحه نمایش نیازمند خواندن تمام حافظه از نو است، این حافظه باید بسیار سریع باشد تا بتواند پاسخوگی نوسازی مرتب صفحه نمایش باشد.
حافظه کامپیوتر می‌تواند به عنوان مثال حافظه با دستیابی تصادفی از نوع DRAM (که چرخه زمانی آن مابین ۶۰ الی ۱۲۰ نانو ثانیه است) باشد. بنابراین اکثر بافرهای چهارچوب از تراشه‌های حافظه خاصی به نام حافظه با دست یابی تصادفی ویدیوئی (VRAM) استفاده می‌کنند. VRAM می‌تواند دنباله‌ای از محل‌های حافظه را سریعتر بخواند، در حالی که در همان حال به کامپیوتر اجازه می‌دهد (با سرعت کمتر) با دستیابی تصادفی به حافظه دسترسی پیدا کند. کاهش بهای تراشه VRAM عامل اصلی برای تبدیل رابطهای کاربری از متنی مانند MS-DOS و Unix به گرافیکی مانند Windows و X Window در لینوکس است. برخی کارتهای تصویر کم بها از DRAM استفاده کرده و چندین پیکسل را به صورت موازی با هم می‌خوانند. اما این کارتها معمولا کندتر از کارتهایی هستند که از VRAM استفاده می‌کنند.

۱.۲.۴ -- بافر دوگانه

برای رسم شکل جدید در بافر چهار چوب، سیستم گرافیکی باید ابتدا شکل قبلی را با پاک کردن بافر چهار چوب حذف کند. اگر چنین کاری در یک پویانمایی به صورت مرتب انجام بگییرد باعث روشن و خاموش شدن (پرپر زدن) تصویر می‌شود. روش عمومی برای حل این مشکل استفاده از بافر دوگانه است. یعنی از دو بافر با مقدار حافظه دو برابر استفاده می‌شود. در این حالت، یک بافر برای نوسازی صفحه نمایش استفاده می‌شود و دیگری برای رسم اشکال استفاده می‌شود.هنگامی که برنامه گرافیکی رسم شکل را پایان داد به کارت تصویر اعلام می‌کند تا دو بافر را با یکدیگر جابجا کند. در نتیجه شکلهای رسم شده در بافر به نمایش در می‌آید و بعد از آن برنامه بافر گرافیکی را پاک کرده و دوباره شروع به رسم می‌کند و این عمل تا پایان برنامه ادامه دارد.

روش دیگری که بجای استفاده از بافر دوگانه بکار می‌رود استفاده از بافر چهارچوب است. اما در این حالت هرگز بطور مستقیم روی آن شکلی رسم نمی‌شود. بجای اینکار شکل جدید ابتدا بر روی یک حافظه دیگر (مانند DRAM موجود بر روی کارتهای گرافیکی و یا حتی حافظه اصلی) رسم می‌شود. هنگامی که سیستم گرافیکی رسم شکل جدید را به پایان برد، شکل جدید به سرعت به بافر چهارچوب (احتمالا توسط یک سخت‌افزار خاص) کپی می‌شود. این نوع از بافر دوگانه معمولا در سیستمهای پنجره‌ای(همانند فرمهای ویندوز) استفاده می‌شود، چون تنها نیازمند کپی کردن محتویات پنجره به حافظه بافر چهار چوب است.

۱.۲.۵ -- فهرستهای نمایشی

علاوه بر ذخیره کردن تصویر خروجی در بافر چهارچوب، برخی از سیستمهای گرافیکی ورودیهای نگارگر (یعنی اشکال پایه‌ای گرافیکی) را نیز ذخیره می‌کنند. حافظه‌ای که برای نگهداری اشکال پایه‌ای استفاده می‌شود فهرست نمایشی یا پایگاه‌داده گرافیکی نامیده می‌شود.
سیستمهای گرافیکی که بدون ذخیره کردن اشکال پایه‌ای تصویر را رسم می‌کنند، سیستمهای حالت‌فوری[٤] نامیده می‌شوند. مظور از حالت‌فوری در اینجا اینست که وقتی یک دستور گرافیکی اجرا می‌شود، خروجی آن بلافاصله ظاهر می‌شود. فایده اصلی اینگونه از سیستمها اینست که (مانند زبان اسمبلی) بسیار به سخت‌افزار نزدیک هستند بنابراین با کمی تلاش بیشتر نوشتن برنامه‌های گرافیکی سریعتر در آنها امکان پذیر است.
سیستمهای گرافیکی که اشکال پایه‌ای را در یک فهرست نمایشی و یا پایگاه داده نگهداری می‌کنند، سیستمهای حالت‌نگهدارنده[٥] نامیده می‌شوند. تفاوت یک فهرست نمایشی ساده با یک پایگاه داده گرافیکی در زمانی است که تنها یک بخش فهرست نمایشی تغییر کند. برای مثال، در یک صحنه گرافیکی صحنه حاوی دهها شکل پایه‌ای، یک پایگاه داده گرافیکی تنها اجازه تغییر یک را می‌دهد. درحالی که فهرست‌های نمایشی ساده نیازمند ارسال مجدد تمام فهرست هستند هم شامل آنهایی که تغییر کرده و هم شامل آنهایی که تغییر نکرده. این سیستمها بخصوص برای کامپیوترهای که دارای سیستم پنجره‌ای[٦] هستند بسیار مناسب هستند. چون در این حالت احتمال دارد بخشی از محتویات یک پنجره دیگر پوشیده شود و بخواهیم این پنجره را به جلو آورده آورده و بطور کامل نمایش دهیم، در این حالت شکل روی پنجره نیازمند رسم مجدد است، که اینکار در سیستمهای حالت نگهدارنده سریعتر انجام می‌شود.

مفاهیم سیستمهای گرافیکی بیشتر از آن چه است که در اینجا توضیح داده شده.اما برای یک نوآموز برنامه‌نویسی گرافیکی کافی است. شما بعدا در فصول بعدی با برنامه نویسی گرافیکی در OpenGL و گاها با مفاهیم سیستمهای گرافیکی در مابین دروس آشنا خواهید شد.

---

[٤] Immediate-mode systems
[٥] Retain-mode systems
[٦] توجه کنید که سیستم عامل ویندوز تنها سیستم عامل پنجره‌ای نیست بلکه سیستم عاملهای پنجره‌ای دیگری مانند X Window در لینوکس نیز موجود هستند.

فصل ۲

شروع برنامه‌نویسی گرافیکی به کمک OpenGL

OpenGL[۷] یک رابط نرم‌افزاری برای سخت افزار گرافیک است. این رابط نرم افزاری تقریبا شامل ۱۲۰ فرمان مجزا از هم است که به برنامه‌نویس اجازه می‌دهد اشکال مورد نیاز خود را به صورت دوبعدی و یا سه‌بعدی به راحتی تهیه کند.
این رابط نرم‌افزاری به صورت یک رابط ساده و مستقل از سخت‌افزار طراحی شده تا بر رو ی بسیاری از سخت‌افزارهای متفاوت پیاده سازی شود. برای رسیدن به این قابلیت هیچ فرمانی در OpenGL برای ایجاد پنجره و کار بر روی آنها و یا کار بر روی ورودی کاربر وجود ندارد اما همراه این رابط کتابخانه‌هایی مانند glut[۸] عرضه می‌شوند که قسمت عظیمی از این قابلیت‌ها را در خود گنجانده‌اند.
در OpenGL فرمانهای سطح بالا که به شما اجازه می‌دهند اشکال پیچیده مانند قوری, ماشین و غیره را طراحی کنید، وجود ندارند. بلکه شما باید این اشکال پیچیده را به کمک اشکال ساده هندسی مانند چند ضلعی ها و خطوط رسم کنید. اما اشکال ساده هندسی سه‌بعدی مانند کره و یا مکعب مستطیل در کتابخانه glut موجود هستند وشما می‌توانید از آنها استفاده کنید.
کتابخانه glut که ما در این کتاب دستورات آن را در کنار آموزش دستورات OpenGL به مرور به شما خواهیم آموخت برای کار با پنجره‌ها و ایجاد آنها در ابتدا در X Window توسط مارک ج.کیگارد[۹] ارائه شد و پس از وی نیت رابینز[۱۰] این کتابخانه را به سیستم عامل مایکروسافت ویندوز منتقل کرد.

چرا OpenGL ؟

شاید با تعریفی که تا کنون از OpenGL کرده باشیم این سوال در ذهن شما پیش آمده باشد که با وجود DirectX چرا باید از OpenGL استفاده کنیم؟! . بله این کاملا درست است که DirectX نسبت به OpenGL ساده‌تر است و اکثر مواردی را که برای ساخت یک بازی و یا برنامه‌گرافیکی لازم است را یکجا گرد آورده است اما نباید فراموش کنیم که DirectX نسبت به OpenGL گستره استفاده کمتری را در سیستم عاملهای متفاوت دارد، OpenGL تقریبا بر روی تمام سیستم عاملهای امروزی قابلیت نصب و اجرا را دارد و برنامه‌های نوشته شده با آن تنها با کامپایل مجدد قابلیت اجرا بر روی سیستم عاملهای مختلف را دارند. بنابراین شما احتایجی به برنامه نویسی مجدد نخواهید داشت. OpenGL نسبت به DirectX سرعت بیشتری دارد و می‌تواند گرافیک را با جزئیات بیشتری نمایش دهد. تنها برای کار با OpenGL شما باید تا حدودی زمان بیشتری صرف کنید. در ساخت بازیهای گرافیکی که احتیاج به گرافیک بالا و در عین حال سرعت بیشتری نیاز دارند(مانند Doom3) می‌توان گفت OpenGL گزینه مناسبتری نسبت به DirectX است. البته باید این نکته را هم اضافه کنم که به کمک موتورهای ساخت بازی که بر پایه OpenGL هستند بازی سازی در OpenGL نیز به مراتب ساده‌تر شده.
با توجه با این نکته که سیستم عامل لینوکس به سرعت در حال گسترش و محبوبیت پیدا کردن میان کاربران متفاوت است در آینده‌ای نچندان دور ما شاهد استفاده از لینوکس بجای ویندوز در بسیاری از موارد خواهیم بود و از آنجا که DirectX تحت سیستم عامل لینوکس پشتیبانی نمی‌شود می‌توان گفت OpenGL بهترین گزینه برای ساخت نرم‌افزارهای گرافیکی است. زیرا شما می‌توانید به راحتی

---

[۷] Open Graphics Library
[۸] GL Utility Toolkit
[۹] Mark J. Kilgard
[۱۰] Nate Robins

نرم‌افزار خود را هم در محیط Windows و هم در محیط لینوکس ویا سیستم عاملهای دیگر که قابلیت پشتیبانی از OpenGL را دارا هستند گسترش دهید

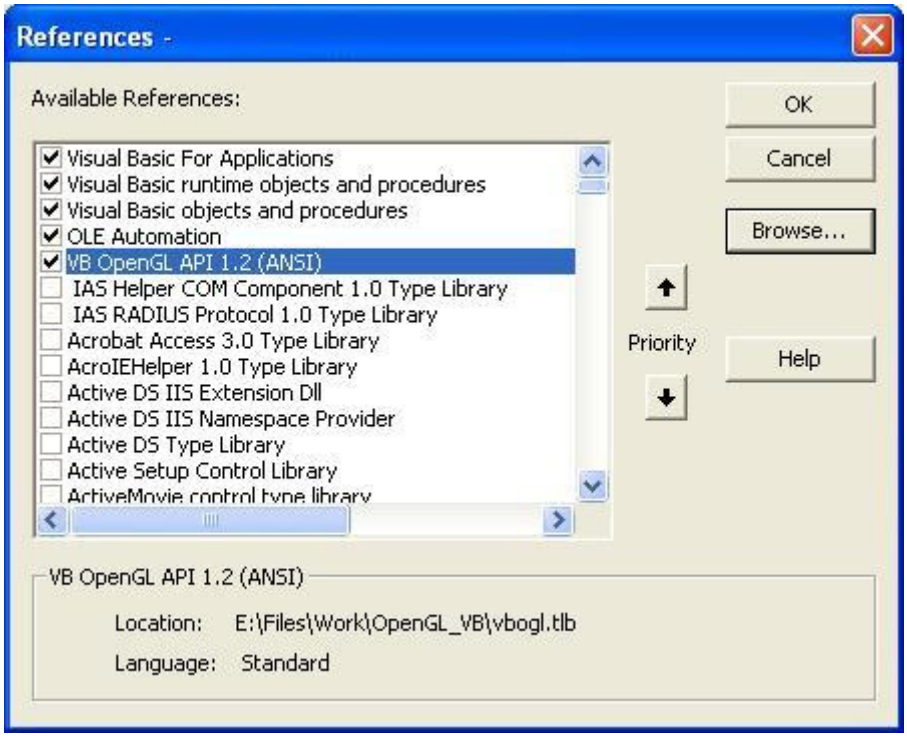
۲.۱ – برنامه‌نویسی OpenGL در ویژوال بیسیک

محیط برنامه‌نویسی ویژوال بیسیک بهترین محیط برای فراگیری مفاهیم برنامه‌نویسی گرافیکی در OpenGL است.شما به کمک این محیط ساده می‌توانید خیلی سریع شروع به یادگیری OpenGL کنید اما در صورتی که می‌خواهید به برنامه‌نویسی حرفه‌ای گرافیکی بپردازید به شما پیشنهاد می‌کنم از یک از زبانهای خانواده سی مانند Borland C++ 5 و یا Visual C++ 6 استفاده کنید تا از سرعت واقعی و تمامی امکانات این رابط گرافیکی لذت ببرید.

۲.۱.۱ – پیاده سازی OpenGL در ویژوال بیسیک

قبل از هر چیز ما باید OpenGL را بر روی ویژوال بیسیک پیاده سازی کنیم. خوشبختانه برای اینکار یک کتابخانه با نام VBOpenGL وجود دارد. این کتابخانه شامل اکثر توابع OpenGL و کتابخانه‌های همراه آن مانند glut است که کار شما را برای کار کردن با OpenGL بسیار ساده کرده است.

برای افزودن این کتابخانه به پروژه ویژوال بیسیک خود از منوی Project بر روی زیر منوی References... کلیک کنید (شکل ۱-۲) و سپس روی دکمه Borrows کلیک کنید. و فایل vbogl.tlb (این فایل را می‌توانند از روی وب سایت www.ps-ware.net و یا www.nehe.gamedev.net تهیه کنید) را انتخاب کنید و بعد از آن بر روی دکمه OK کلیک کنید تا این کتابخانه بر روی پروژه شما قرار بگیرد.



شکل ۲ -۱ : افزودن کتابخانه VBOGL به پروژه ویژوال بیسیک

به تنهایی با افزودن کتابخانه OpenGL به پروژه ویژوال بیسیک شما قادر به استفاده از آن نخواهید بود. بلکه شما باید یکسری از تنظیمات اولیه را انجام بدهید و سپس پنجره‌ای را که می‌خواهید در آن

اشکال را در آن رسم کنید را فراخوانی کنید. در قسمت بعد شیوه ایجاد یک پنجره را در OpenGL شرح خواهیم داد.

۲.۱.۲ – ایجاد پنجره در OpenGL

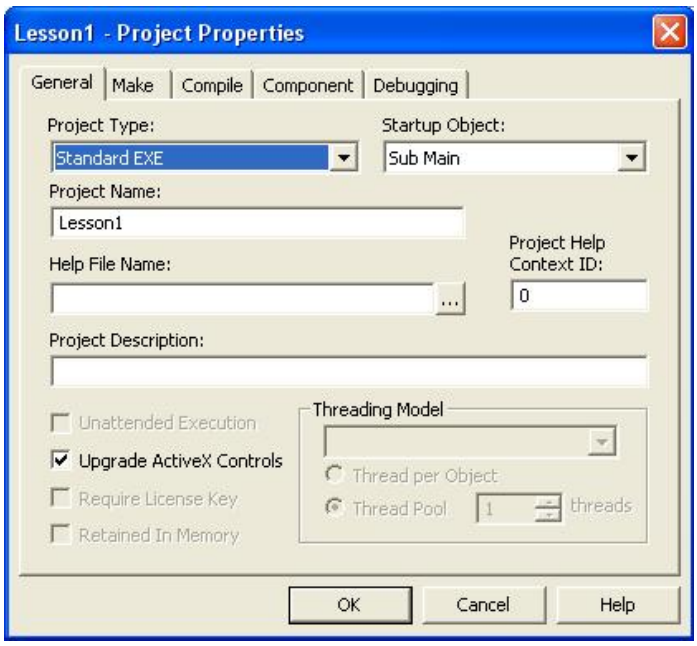
ایجاد پنجره در OpenGL کار نسبتا ساده‌ای است اما باید پیش از ایجاد پنجره در OpenGL یکسری تنظیمات اولیه را انجام دهیم تا سیستم گرافیکی بداند که ما تحت چه شرایطی می‌خواهیم پنجره را نمایش بدهیم. از جمله این تنظیمات می‌توان به موارد زیر اشاره کرد:

- اندازه پنجره (ارتفاع و طول آن)
- تعیین عمق رنگ
- تعیین حالت رنگ (مانند RGB ، RGBA و غیره...)
- تعیین اینکه آیا می‌خواهیم از بافر دوگانه استفاده کنیم و یا نه

و غیره...

برای ایجاد پنجره، شروع به کد نویسی می‌کنیم. و من سعی می‌کنم تا آنجا که مقدور است دستورات استفاده شده درون کد را همراه با کدنویسی توضیح بدهم. برای شروع کد نویسی مراحل زیر را دنبال کنید

۱. بر روی منوی Project کلیک کنید و گزینه Add Module را انتخاب کنید و سپس از دیالوگ باز شده بر روی دکمه Open کلیک کنید تا ویژوال بیسیک برای شما یک ماژول جدید ایجاد کند.
۲. نام ماژول را از پنجره Properties به OGLUtils (این نام اختیاری است.)تغییر دهید.
۳. بر روی منوی Project کلیک کنید و منوی Project Properties را انتخاب کنید تا دیالوگ خواص پروژه به نمایش در بی‌آید.(شکل ۳)



شکل ۳

۴. از لیست بازشوی (Combo Box) Startup Object: گزینه Sub Main را انتخاب کنید تا در شروع برنامه بجای نمایش فرم اصلی برنامه (در اکثر موارد نام آن Form1 است) تابع Main را که درون ماژول OGLUtils قرار دارد، را فراخوانی کند.
۵. بر روی ماژول کلیک کنید و یک تابع به نام Main بسازید.

حال درون تابع Main کدهای زیر را وارد کنید. (به کمک کد زیر ما یک پنجره ایجاد می‌کنیم.)

```
100 Dim Done As Boolean
200 Dim frm As Form
300 Done = False
400 Set frm = New Form1     ' create our form
500 If Not CreateGLWindow (frm, 640, 480, 16) Then Done = True
```

در ابتدا یک متغییراز نوع بولی تعریف می‌کنیم هنگامی که مقدار این متغییر true باشد از برنامه خارج می‌شویم، بنابراین در ابتدای کار مقدار false را به آن می‌دهیم. سپس یک شی به نام frm از نوع Form (خط ۲۰۰) را تعریف می‌کنیم در خط شماره ۴۰۰ ما شی frm را به Form1 منسوخ می‌کنیم. بعد از اینکه این مراحل را طی کردیم نوبت ایجاد پنجره می‌رسد ما برای ایجاد پنجره از یک تابع به نام CreateGLWindow استفاده می‌کنیم که این تابع را بعدا در ماژول OGLUtils تعریف می‌کنیم. تابع CreateGLWindow دارای چهار پارامتر است که این پارامترها به ترتیب برابرند با یک شی فرم، عرض ، ارتفاع و تعداد بیتهای رنگ که در اینجا شانزده است. این تابع در صورتی که بطور موفقیت آمیز کار خود را به پایان رسانده باشد مقدار True و در غیر این صورت مقدار False را برمی‌گرداند. در خط شماره ۵۰۰ اگر تابع مقدار False را برگرداند مقدار متغییر done را True می‌کنیم که از برنامه خارج شویم.

سپس برای ادامه کار باید تابع CreateGLWindow را تعریف کنیم. کد زیر تابع CreateGLWindow را نشان می‌دهد. به منظور جلوگیری از گمراهی شما خواننده عزیز در آینده به توضیح دستورات درون این تابع می‌پردازیم ولی این تابع برای تمامی مثالهای قسمت اول این کتاب الکترونیکی که هم اکنون در پیش روی شماست کافیست و شما می‌توانید از آن به راحتی در تمام مثالهای خود، تنها با کپی کردن آن استفاده کنید.

```
Public Function CreateGLWindow(frm As Form, Width As Integer, Height As Integer, Bits As Integer) As Boolean
    Dim PixelFormat As GLuint
    Dim pfd As PIXELFORMATDESCRIPTOR

    pfd.cColorBits = Bits               ' color depth
    pfd.cDepthBits = 16
    pfd.dwFlags = PFD_DRAW_TO_WINDOW Or PFD_SUPPORT_OPENGL Or PFD_DOUBLEBUFFER
    pfd.iLayerType = PFD_MAIN_PLANE
    pfd.iPixelType = PFD_TYPE_RGBA
    pfd.nSize = Len(pfd)
    pfd.nVersion = 1

    PixelFormat = ChoosePixelFormat(frm.hDC, pfd)
    If PixelFormat = 0 Then
      KillGLWindow
      MsgBox "Can't Find A Suitable PixelFormat.", vbExclamation, "ERROR"
      CreateGLWindow = False
    End If

    If SetPixelFormat(frm.hDC, PixelFormat, pfd) = 0 Then
       KillGLWindow
       MsgBox "Can't Set The PixelFormat.", vbExclamation, "ERROR"
       CreateGLWindow = False
    End If
```

```
    hrc = wglCreateContext(frm.hDC)
    If (hrc = 0) Then
        KillGLWindow
        MsgBox "Can't Create A GL Rendering Context.", vbExclamation, "ERROR"
        CreateGLWindow = False
    End If

   If wglMakeCurrent(frm.hDC, hrc) = 0 Then
        KillGLWindow
        MsgBox "Can't Activate The GL Rendering Context.", vbExclamation, "ERROR"
        CreateGLWindow = False
    End If
    frm.Show
    If Not InitGL () Then
        KillGLWindow
        MsgBox "Initialization Failed.", vbExclamation, "ERROR"
        CreateGLWindow = False
    End If
    CreateGLWindow = True
End Function
```

درون تابع CreateGLWindow دو تابع دیگر به نامهای KillGLWindow و InitGL فراخوانی شده این دوتابع جزء توابع OpenGL نبوده و خود ما باید آنها را در ماژول OGLUtils تعریف کنیم. کار تابع KillWindow آزاد کردن فضای استفاده شده توسط برنامه در سیستم است(در صورتی که از این تابع استفاده نکنیم امکان مواجه شدن با کمبود فضا بر روی حافظه اصلی سیستم و یا حافظه کارت گرافیکی افزایش می‌یابد.) کد مربوط به این تابع را می‌توانید در زیر مشاهده کنید. اما از توضیح دادن کد صرف نظر می‌کنم چون این کد برای تمامی پنجرهایی که ما در ویژوال بیسیک از آن استفاده می‌کنیم قابل استفاده است و در طول دوره آموزشی آن را تغییر نمی‌دهیم شما همانند کد تابع CreateGLWindow تنها کافیست که آن را درون ماژول OGLUtils کپی کنید. در صورتی که خواستار کسب اطلاع در مورد کد زیر هستید می‌توانید به انجمنهای تیم برنامه‌نویسی www.ps-ware.net مراجعه کنید و در قسمت بازیهای رایانه‌ای سوال خود را مطرح کنید.

```
Public Sub KillGLWindow ()
      If hrc Then
       If wglMakeCurrent(0, 0) = 0 Then
          MsgBox "Release of DC and RC Failed.", vbInformation, "SHUTDOWN ERROR"
       End If

       If wglDeleteContext(hrc) = 0 Then                    MsgBox "Release Rendering Context
Failed.", vbInformation, "SHUTDOWN ERROR"
       End If
       hrc = 0
    End If
End Sub
```

تابع InitGL خود نیز جزء توابعی است که ما باید آن را تعریف کنیم عملکرد این تابع پاک کردن صفحه نمایش و آماده سازی آن برای رسم اشکال است. ما همیشه قبل از شروع رسم بر روی صفحه نمایش برنامه این تابع را فراخوانی می‌کنیم تا تمام تنظیمات اولیه که به آن نیاز خواهیم داشت را بدینوسیله تعریف کنیم. در طی این فصل و فصول بعدی ما به صورت مکرر به دستکاری کد این تابع و توضیح قسمتهای مختلف آن می‌پردازیم اما در مثال اول این کتاب این تابع یک دستور بیشر ندارد و آن دستور glClearColor است. به کمک این دستور ما رنگ پیش زمینه صفحه نمایش را تعیین می‌کنیم. (رنگی که در آینده به کمک آن تابع glClear صفحه نمایش را پاک می‌کند.)

دستور glClearColor دارای چهار پارامتر است که مشخص می‌کند صفحه نمایش با چه رنگی باید پاک شود این پارامترها به ترتیب عبارتند از : قرمز،سبز, آبی و آلفا . هرکدام از این پارامترها شدت رنگ خود را مشخص می‌کنند. در دستور زیر به تابع glClear اعلام می‌کند که صفحه نمایش (همان صفحه Form1) را با رنگ زرد که ترکیبی از رنگ‌های قرمز و سبز است، پاک کند.

```
glClearColor  1, 1, 0, 0
```

حال کد تابع InitGL را درون ماژول OGLUtils وارد می‌کنیم. در کد زیر صفحه نمایش به کمک دستور glClearColor با رنگ سیاه پاک می‌شود. وسپس مقدار true برگردانده می‌شود تا تابع CreateGLWindow که تابع را فراخوانی می‌کند، متوجه شود که کار تابع با موفقیت به پایان رسیده است.

```
Public Function InitGL () As Boolean
   glClearColor 0#, 0#, 0#, 0
     InitGL = True
End Function
```

سپس باز به تابع Main باز می‌گردیم و در ادامه دستورات، کدهای زیر را وارد می‌کنیم. کار این تکه کد فراخوانی توابع رسم و پایان دادن به برنامه در صورت نیاز است

```
Do While Not Done
  If (DrawGLScene = False) Then
        Unload frm
      Else
        SwapBuffers (frm.hDC)
         DoEvents
      End If
      Done = frm.Visible = False
Loop
```

در ابتدا یک حلقه بینهایت Do تعریف می‌کنیم.این حلقه هنگامی به پایان می‌رسد که مقدار متغییر done برابر true باشد. کار این حلقه فراخوانی مکرر تابع DrawGLScene است که اشکالی را که می‌خواهیم آنها را رسم کنیم درون این تابع قرار می‌دهیم. در صورتی که این تابع مقدار true را برگرداند تابع کار خود را با موفقیت به پایان رسانده و برنامه باید بافر را تعویض کند و اشکال رسم شده را بر روی صفحه نمایش را نشان دهد. دستور SwapBuffers بافر رویی پنجره‌ای را که به آن داده شده با بافر پشتی (بافر رسم) تعویض می‌کند.
حال کار با تابع Main را به پایان رسانده‌ایم تابع Main برنامه شما باید چیزی مشابه این تابع باشد.

```
Sub Main()
   Dim Done As Boolean
   Dim frm As Form
   Done = False
   Set frm = New Form1
   If Not CreateGLWindow (frm, 640, 480, 16) Then Done = True
   Do While Not Done
      If (DrawGLScene = False) Then
        Unload frm
      Else
        SwapBuffers (frm.hDC)
         DoEvents
```

```
        End If
        Done = frm.Visible = False
    Loop

Set frm = Nothing
End
End Sub
```

در پایان تابع Main شی frm را برابر تهی قرار می‌هیم. و از برنامه با دستور end خارج می‌شویم.

همانطور که مشاهده کردید در تابع Main برنامه یک تابع دیگر به نام DrawGLScene معرفی کردیم از آنجا که این تابع جزء توابع OpenGL نیست خود ما باید آن را تعریف کنیم در زیر کد مربوط به این تابع را می‌بینید.

```
Public Function DrawGLScene () As Boolean
    glClear clrColorBufferBit
    DrawGLScene = True
End Sub
```

تمامی اشکالی را که ما می‌خواهیم آنها را رسم کنیم درون این تابع تعریف خواهیم کرد و با هر بار فراخوانی این تابع تمام صفحه نمایش از نوع رسم می‌شود بنابراین لازم است که اشکال رسم شده پیشین توسط دستور glClear پاک شوند. این دستور که جزء توابع OpenGL است بافر رسم (بافری که ما اید اشکال را بر روی آن رسم کنیم) را با رنگی که توسط تابع glClearColor[11] مشخص شده پاک می‌کند. تابع DrawGLScene در پایان مقدار True را بر می‌گرداند تا به برنامه بگوید که هیچ خطایی رخ نداده و می‌تواند به کار خود ادامه دهد.

حال طراحی برنامه تمام شده و اگر شما آن را اجرا کنید یک صفحه سیاه خواهید دید اما لازم است در برنامه چند تغییر دیگر نیز بدهیم تا پنجره به بهترین شکل ممکن آماده رسم شود برای همین یک تابع معرفی می‌کنیم که به کمک این تابع اگر پنجره تغییر اندازه داده شد. تمام مختصات دید کاربر به اندازه پنجره در بیاید و تمام شکلها همراه پنجره کوچک و بزرگ بشوند. برای این ما در ماژول OGLUtils یک تابع به نام ReSizeGLScene معرفی می‌کنیم. که کار این تابع تغییر اندازه محدوده رسم و اندازه اشکال به اندازه کل پنجره است. این تابع حتما باید از نوع عمومی (Public) باشد تا بتواند توسط رویداد Resize فرم فراخوانی بشود. تابع ReSizeGLScene دارای دو پارامتر است که به ترتیب این پارامترها از چپ به راست برابر عرض و ارتفاع پنجره است

```
Public Sub ReSizeGLScene (ByVal Width as GLsizei, ByVal Height As GLsizei)
    If Height = 0 Then Height = 1
    glViewport 0, 0, Width, Height
End Sub
```

تابع ReSizeGLScene به کمک دستور glViewport که از دستورات OpenGL است محدوده نمایش کاربر را به پارامترهای width و height تغییر می‌دهند. نکته جالبی که شاید در این تابع برخورد کنید نوع متغییر GLsizei است این نوع متغییر برابر همان نوع integer در ویژوال بیسیک است که توسط کتابخانه OpenGL در پروژه شما تعریف شده است. در مورد تابع GlViewport در فصول بعدی به تفضیل صحبت خواهد شد.

[11] این تابع درون تابع InitGL تعریف شده است

حال بر روی Form1 دابل کلیک کنید و کدهای زیر را درون آن قرار دهید. کدهای زیر در هنگام تغییر اندازه و بسته شدن فرم توابع ReSizeGLScene و KillGLWindow را فراخوانی می‌کنند. تا برنامه واکنش مناسب نسبت به آنها را انجام دهد.

```
Option Explicit

Private Sub Form_Resize()
    ReSizeGLScene ScaleWidth, ScaleHeight
End Sub

Private Sub Form_Unload(Cancel As Integer)
    KillGLWindow
End Sub
```

دوستان هم اکنون به این بخش از این فصل خاتمه می‌دهیم. در بخش دیگر با مختصات‌یابی دو بعدی و رسم خطوط و نقاط در OpenGL آشنا خواهید شد. به شما پیشنهاد می‌کنم که قبل از مطالعه بخش بعد، ابتدا مثال ex1 (که به همراه کتاب موجود است) را بررسی کنید وسعی کنید که با تغییر دادن پارامترهای مختلف دستور glClearColor ، نتیجه تغییرات را بر روی صفحه نمایش مشاهده کنید. توجه کنید که به هر یک از پارامترها باید عدد اعشاری مابین یک تا صفر بدهید.

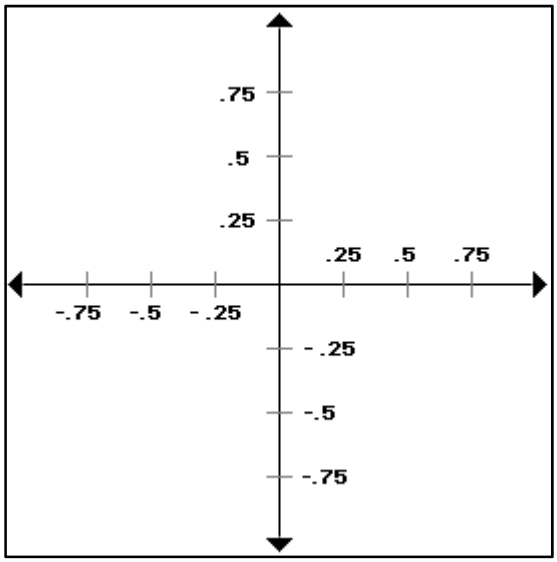
۲.۲ – رسم اشکال

تا کنون قسمت‌های بسیار مشکلی را پیش رو گذاشتیم که گاها گمراه کننده بودند. ولی در طی این قسمت و فصل بعد به رسم اشکال دو بعدی خواهیم پرداخت که قسمت بسیار جذاب و همراه با تنوع زیاد هستند، و یادگیری آنها بسیار آسانتر از قسمتهای پیشین است. اما قبل از شروع به رسم اشکال باید شیوه یافتن مختصات را در صفحه نمایش (در اینجا همان Form1 است) را فرا بگیریم. و با شیوه تعیین رنگ قلمو نقاشی آشنا بشویم

۲.۲.۱ – یافتن مختصات در صفحه

همیشه قبل از رسم اشکال هندسی بر روی کاغذ شطرنجی سعی می‌کنیم. ابتدا مختصات آنها را بیابیم سپس اقدام به رسم آنها بکنیم. به عنوان مثال ما برای رسم یک مثلث بر روی صفحه ابتدا مختصات سه راس آن را مشخص می‌کنیم سپس از هریک از رئوس به راس دیگر یک خط مستقیم می‌کشیم. در پنجره‌های گرافیکی OpenGL نیز باید به همین شکل عمل کنیم یعنی ابتدا مختصات هر راس را به OpenGL بدهیم سپس از آن بخواهیم که یک مثلث را برای ما رسم کند.

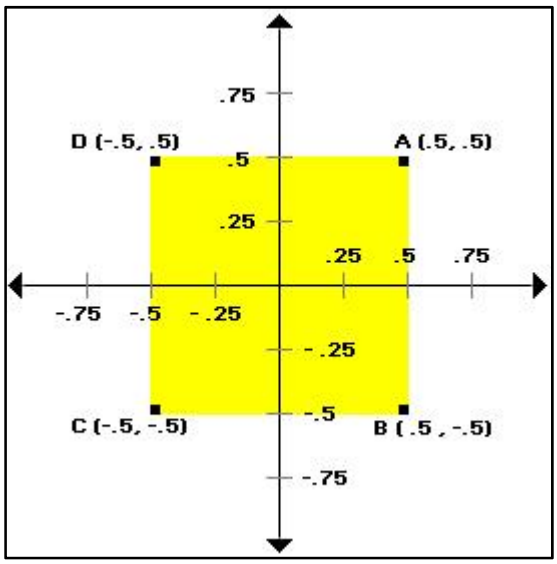
همانگونه که قبلا اشاره کردیم OpenGL سعی کرده کاملا بدون وابستگی به هیچ سیستمی به کار خود بپردازد به همین علت در OpenGL برای مشخص کردن آدرس یک نقطه در صفحه نمایش نمی‌توانیم از آدرس پیکسل مربوط به آن استفاده کنیم. در OpenGL صفحه نمایش بطور پیش‌فرض از منفی یک تا یک چه در جهت X و چه در جهت Y مندرج شده (شکل ۳) که منهای یک بر روی محور xها نشان دهنده ضلع سمت چپ صفحه و 1+ نشان دهنده ضلع سمت راست صفحه است و همین موضوع برای محور Yها نیز صدق می‌کند. به شکلی که 1+ نشان دهنده ضلع بالایی صفحه و منفی یک نشان دهنده ضلع پایینی است. به عنوان مثال مختصات (0,0) به مرکز صفحه اشاره می‌کند. در شکل شماره سه شیوه درجه بندی صفحه توسط OpenGL مشخص شده است. با نشان دادن مثالهای متفاوت سعی می‌کنیم کاری کنیم که شما درک بهتری نسبت به شیوه مختصات یابی در OpenGL پیدا کنید.



شکل ۴ - مختصات صفحه در OpenGL

در این مثال یک مربع را در شکل شماره چهار می‌بینید که رئوس آن دارای مختصاتهای زیر هستند:

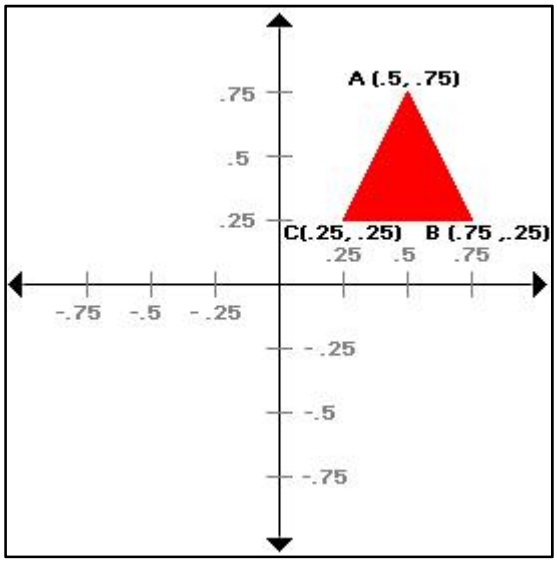
A (0.5, 0.5)
B (0.5, -0.5)
C (-0.5, -0.5)
D (-0.5, 0.5)



شکل ۵ - نمونه مربع رسم شده در مرکز صفحه نمایش

در مثال زیر (شکل شماره ۵) شما یک مثلث قرمز رنگ را می‌بینید که رئوس آن دارای مختصات‌های زیر هستند:

A (0.5, 0.75)
B (0.75, 0.25)
C (0.25, 0.25)



شکل ۶

هر چند این شیوه رسم تا حدودی سخت بنظر می‌رسد اما دارای مزیتهای بسیار زیادی است که از جمله این مزیتها می‌توان به این مطب اشاره کرد که اگر صفحه نمایش (همان Form) در اینجا تغییر اندازه پیدا کند، شما دیگر هیچ احتیاجی به یافتن مختصات جدید رئوس نخواهید داشت.

در مورد رسم باید به این نکته توجه کنید که شما باید رئوس را در جهت خلاف عقربه‌های ساعت رسم کنید و هر راس به ترتیب باید در مجاورت راس قبل و بعد خو باشد تا از تا خوردگی شکل جلوگیری شود. این مساله در نور پردازی بسیار پر اهمیت است.

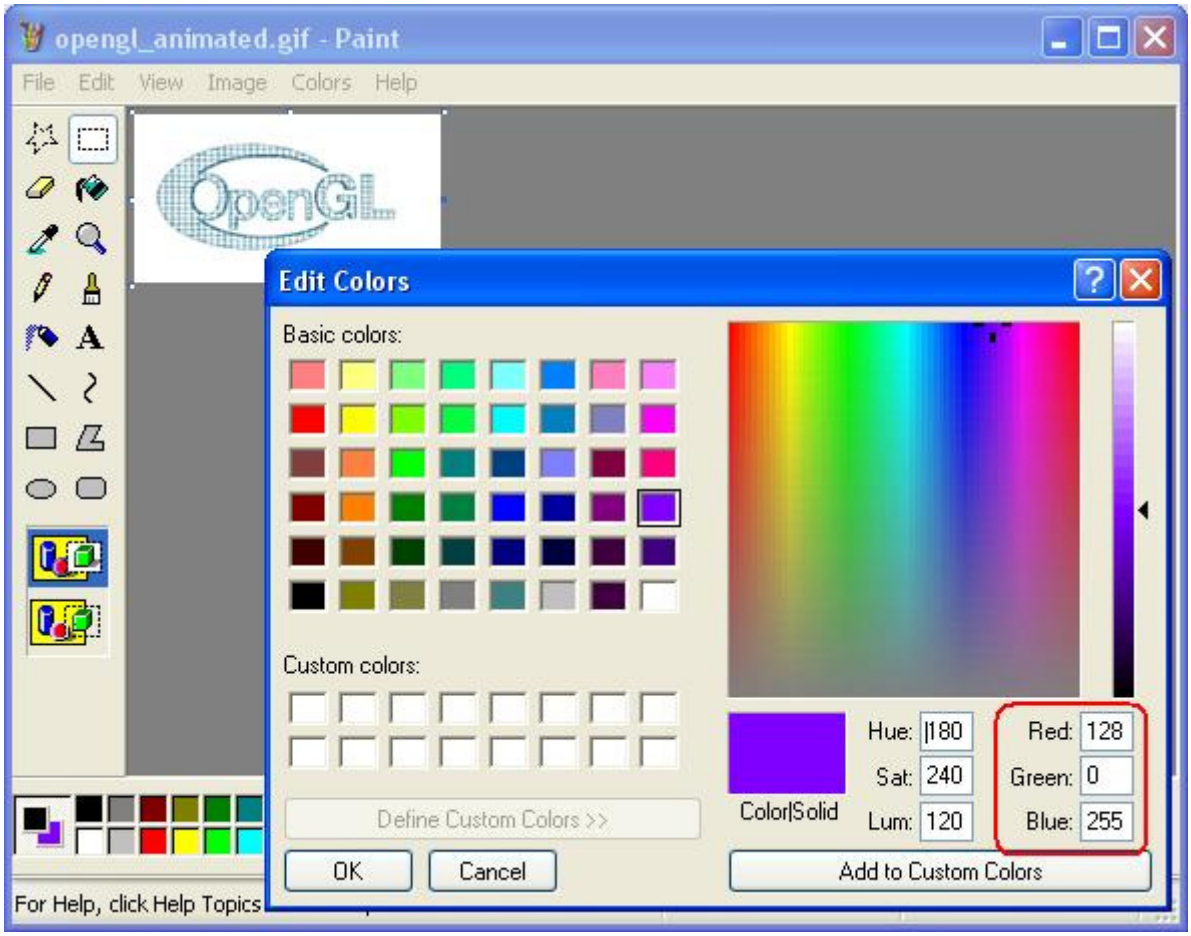
۲.۲.۲ – تعیین رنگ اشکال

قبل از اینکه یک شکل را بخواهید بر روی صفحه کاغذ با قلمو رسم کنید باید حتما قلموی خود را درون جوهر رنگ مورد نظر فرو ببرید وسپس شروع به نقاشی کنید. این مساله در مورد OpenGL هم صادق است یعنی قبل از اینکه بخواهید یک شکل را بر روی صفحه نمایش رسم کنید باید تعیین کنید که می‌خواهید از کدام رنگ استفاده کنید.

برای این کار OpenGL دارای چند دستور به نامهای glColor3f، glColor3d و غیره است. تمام این دستورها کارهای مشابه را انجام می‌دهند و تنها تفاوت آنها در نوع پارامترهای آنها است برای مثال پارامترهای glColor3d از نوع Integer است و می‌تواند هر عددی مابین ۰ تا ۲۵۵ باشد اما نوع پارامترهای glColor3f از نوع اعشاری است و می‌تواند شامل تمام اعداد اعشاری ما بین ۰ تا ۱ باشد.

ما در طول این کتاب الکترونیکی از دستور glColor4f استفاده خواهیم کرد این دستور همانند دستور glClearColor چهار پارامتر را می‌پذیرد که به ترتیب این پارامترها برابراند با رنگ قرمز، رنگ سبز، رنگ آبی و آلفا. مقدار هر کدام از این پارامترها بیانگر شدت رنگ است. به عنوان مثال اگر به رنگ آبی مقدار یک را بدهیم و به مابقی بجز آلفا مقدار صفر را بدهیم به این معنی است که رنگ آبی را انتخاب کردایم و یا اگر به تمامی رنگها مقدار یک را بدهیم یعنی رنگ سفید را انتخاب کرده‌ایم. همواره دانستن این موضوع که از چه مقادیری و ترکیب رنگی استفاده کنیم تا رنگ مورد نظر خود را پیدا کنیم، یک مساله گمراه کننده بوده اما یک راه حل بسیار ساده برای اینکار وجود دارد و آن استفاده از MS Paint است .

برای اینکه بتوانیم رنگ مورد نظر خود را پیدا کنیم از MS Paint استفاده می‌کنیم بنابراین واد محیط paint شوید و بر روی جعبه رنگها دوبار کیک چپ بکنید تا دیالوگ انتخاب رنگ برای شما ظاهر شود (شکل شماره ۷)



شکل ۷ - نمایی از برنامه MS Paint و دیالوگ انتخاب رنگ

سپس رنگ مورد نظر خود را انتخاب کنید و اطلاعات آن (میزان Red, green و Blue را که در گوشه سمت راست دیالوگ انتخاب رنگ قرار دارند را بردارید و آن را بر عدد ۲۵۵ تقسیم کنید و مقدار هر یک از آنها را در پارامتر مربوط به خود تا دو رقم اعشار وارد کنید. برای مثال دستور glColor4f برای رنگ انتخاب شده در این شکل برابر خواهد بود با:

glColor4f 0.5, 0, 1, 1

پارامتر آلفا میزان کدر بودن جسم را نشان می‌دهد. در صورتی که مقدار این پارامتر یک باشد جسم کاملا کدر خواهد بود و هیچ نوری را از خود عبور نخواهد داد و در صورتی که این پارامتر برابر با صفر باشد جسم مانند یک شیشه کاملا بدون رنگ خواهد بود که تمامی نور را از خود عبور می‌دهد. من همیشه شخصا مقدار نیم و چهاردهم را برای شیشه‌های رنگی ترجیح می‌دهم. این نکته لازم به ذکر است که شما تا زمانی که به OpenGL اختیار محاسبه قابلیت Alpha را نداده باشید، اجسام همیشه کدر دیده خواهند شد و فرقی نمی‌کند که شما چه مقدار به پارامتر آلفا بدهید. در مورد رنگ آلفا و به کار بردن آن بطور اختصاصی در یک فصل جداگانه بحث خواهیم کرد.

۲.۲.۳-- رسم نقطه و خط

تا کنون تمامی موارد را که برای رسم اشکال ساده دو بعدی به آنها احتیاج داشتیم را فرا گرفتیم و حال نوبت به رسم ساده‌ترین اشکال دو بعدی که نقطه و خط هستند رسیده.

با یک مثال کار خود را ادامه می‌دهیم. در این مثال ما سه نقطه متفاوت با رنگهای قرمز،سبز و آبی در سه محل متفاوت از صفحه نمایش رسم می‌کنیم. برای اینکار تابع DrawGLScene را بصورت زیر تغییر دهید:

```
Public Function DrawGLScene () As Boolean
glClear clrColorBufferBit    ' pak kardan baffer safe rasm baraye shoro be rasn jadid

glBegin bmPoints
      glColor4f 1, 0, 0, 1 'range germez
      glVertex2f 0.3, 0.2  'mokhtasate rase shomare 1
      glColor4f 0, 1, 0, 1 ' range sabz
      glVertex2f -0.3, 0.2  'mokhtasate nogte shomare 2
      glColor4f 0, 0, 1, 1 'range abi
      glVertex2f -0.3, -0.2 'mokhtasate ogte shomare 3
glEnd

   DrawGLScene = True
End Function
```

در اینجا ما با استفاده دستور glBegin به OpenGL اعلام می‌کنیم که آماده رسم یک شکل شود و تمام رئوسی را که با استفاده از دستور glVertex2f به آن داده شده را به شکل منتسب کند و در پایان با دستور glEnd به OpenGL اعلام می‌کنیم که تمامی رئوس داده شده‌اند و شکل باید رسم شود. دستور glBegin دارای یک پارامتر است که مقدار bmPoints به آن داده شده است این مقدار به OpenGL اعلام می‌کند که باید به ازای هر راس یک نقطه بر روی صفحه نمایش بکشد. همانگونه که می‌بینید قبل از دادن رئوس رنگ هر راس را به OpenGL اعلام کرده‌ایم در حقیقت با هر بار فراخوانی تابع glColor4f رنگ قلمو عوض می‌شود و شما می‌توانید فقط یکبار در ابتدا رنگ قلمو را مشخص کنید و دیگر شما احتیاجی به اینکار ندارید. در این مثال من تنها خواستم که استفاده از رنگ قلمو را برای هر راس نشان دهیم.

کار تابع glVertex2f تعیین هر راس یک شکل می‌باشد این تابع دارای دو پارامتر است که این پارامترها به ترتیب برابراند با X و Y که همان مکان x و y را بر روی صفحه نمایش نشان می‌دهند.

برای رسم هر خط در صفحه نمایش ما باید دو راس خط را تعیین کنیم تا OpenGL از یک راس به راس دیگر خط مستقیمی را رسم کند. در مثال زیر OpenGL از منفی یک تا یک بر روی محور xها یک خط را با رنگ زرد رسم می‌کند

```
glColor4f 1, 1, 0, 1  'rang zard
glBegin bmLines
      glVertex2f -1, 0
      glVertex2f 1, 0
glEnd
```

در اینجا تابع glBegin مقدار bmLines را گرفته این مقدار به تابع اعلام می‌کند که به ازای هر دو راس یک خط رسم کند.

برنامه و مثالهایی را که در این قسمت نوشیتم بر روی مثال Ex2 که همراه کتاب الکترونیکی قرار دارد. به شما پیشنهاد می‌کنم که قبل از شروع فصل بعد به دستکاری و تغییر دادن انواع مقادیر پارامترهای دستور glVertex2f بپردازید تا بر آن و نقطه‌یابی بر روی صفحه تسلط پیدا کنید. در فصل بعدی بطور مفصل در مورد تابع glBegin و شیوه رسم اشکال دو بعدی در OpenGL بحث خواهیم کرد بطوری که شما در پایان فصل بعد قادر خواهید بود اکثر اشکال دو بعدی را رسم کنید.

فصل ۳

رسم اشکال دو بعدی

در طی این فصل با دستورات ساده گرافیکی در OpenGL برای رسم اشکال هندسی آشنا می‌شویم. در طول این فصل به بررسی دستور glBegin به همراه انواع مقادیری که می‌تواند بگیرد می‌پردازیم و هریک از آنها را به شکل مفصل توضیح می‌دهیم.

۳.۱ -- تابع glBegin

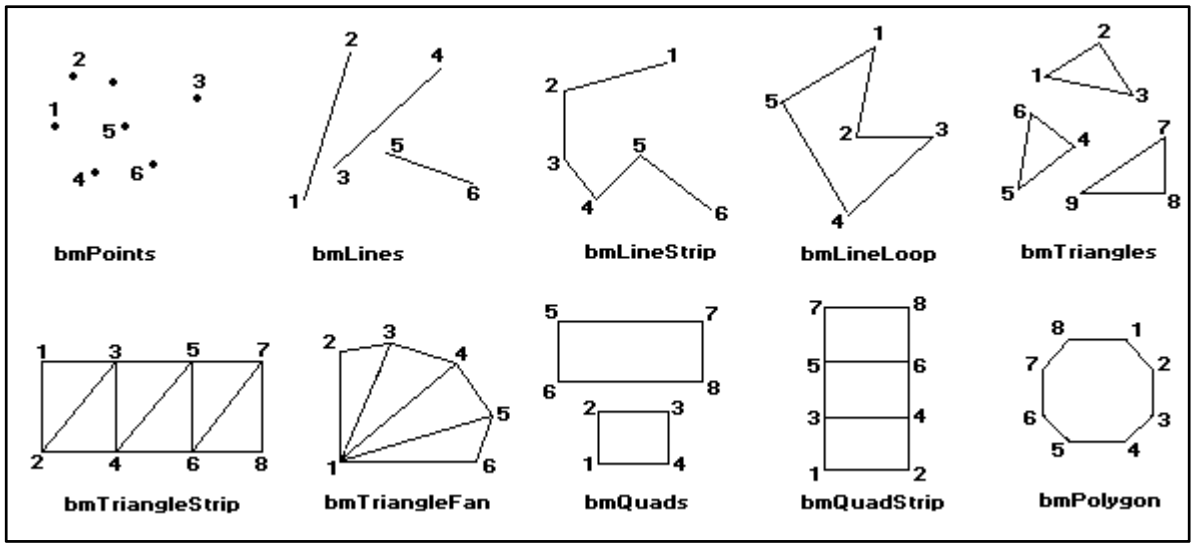
این تابع به OpenGL اعلام می‌کند که شما آماده رسم یک شکل ساده هندسی که به عنوان پارامتر، نوع آن شکل را شما به تابع ارسال کرده‌اید، هستید. و بعد از فراخوانی این تابع با مقدار مورد نظر خود با استفاده از دستور glVertex2f رئوس شکل را مشخص می‌کنید و سپس با فراخوانی تابع glEnd به OpenGL اعلام می‌کنید که وارد کردن رئوس شکل به پایان رسیده و باید OpenGL شکل شما را رسم کند. در این مود در فصل قبل توضیحاتی داده شده که می‌توانید به آنها مراجعه کنید.

دستور glBegin دارای یک پارامتر(از نوع glBeginModeConstants) به نام mode است که این پارامتر می‌تواند مقادیر زیر را به خود بگیرد. در جدول زیر تمامی پارامترهایی که این دستور می‌تواند بپذیرد را به همراه توضیحات آنها می‌توانید ببینید. همچنین در شکل شماره ۸، نام هر یک از این ثابت‌ها را به همراه نمونه شکل آنها نشان داده شده است.

جدول شماره یک

مقدار	توضیحات
bmPoints	نقاط جدا از هم
bmLines	پاره خط هایی با دو راس
bmLineStrip	مجموعه‌ای از خطوط به هم پیوسته
bmLineLoop	یک مجموعه از خطوط به هم پیوسته با انتهای بسته به شکلی که پاره خط آخری به صورت اتوماتیک ایجاد شود و راس آخر را به راس اول متصل می‌کند.
bmTriangles	مثلث‌های منفرد با رئوس سه‌گانه
bmTriangleStrip	یک مجموعه از مثلث‌های به هم پیوسته
bmTriangleFan	مثلث‌های به هم پیوسته که دارای یک راس مرکزی مشترک هستند.
bmQuads	چهار ضلعی های منفرد (چند ضلعی‌هایی که دارای یک راس مرکزی هستند.
bmQuadStrip	یک مجموعه از چهار ضلعی‌های به هم پیوسته
bmPolygon	یک چند ضلعی محدب با تعداد رئوس اختیاری.

تمامی این موارد که در جدول بالا را ذکر شده‌اند را در طول این فصل به شکل کامل همراه با حالتهای مختلف آنها و ذکر مثال بررسی خواهیم کرد. شکل زیر تفاوت هریک از این ثوابت را بر روی دستور glBegin نشان می‌دهد.



شکل ۸- مقادیری را که دستور glBegin می‌تواند آن را بپذیرد.

۳.۱.۱ -- رسم نقاط

نقاط ساده‌ترین اشکال هندسی در داخل OpenGL هستند.که رسم آنها بسیار آسان و سریع است. اگر شما بتوانید یک نقطه را بر روی صفحه نمایش رسم کنید، خواهید توانست با رسم یک مجموعه از نقاط بر روی صفحه نمایش تمامی اشکال گرافیکی را رسم کنید. مثال زیر چگونگی رسم یک نقطه را در OpenGL نشان می‌دهد.

```
glBegin bmPoints
        glVertex2f 0.0, 0.0
glEnd
```

در خط اول این مثال شما به OpenGL به وسیله ارسال ثابت bmPoints به تابع glBegin می‌گویید که در نظر دارید، تعدادی نقاط را بر روی صفحه نمایش رسم کنید. در خط بعد شما به OpenGL مختصات نقطه‌ای را که باید بر روی صفحه نمایش رسم شود را می‌دهید. سپس در پایان با فراخوانی تابع glEnd به OpenGL اعلام می‌کنید. که رسم شما به پایان رسیده و نقاطی را که مختصات آنها مابین بلوک glBegin/glEnd قرار دارد را باید بر روی صفحه نمایش نشان دهد.

در صورتی که شما به خواهید یک نقطه دیگر در مختصات (۱.۰ ، ۰.۰) را بر روی صفحه نمایش چاپ کنید. ممکن شما از تکه کد زیر استفاده کنید.

```
GlBegin bmPoints
        glVertex2f 0.0, 0.0
glEnd
glBegin bmPoints
        glVertex2f 0.0, 1.0
glEnd
```

هر چند که تکه کد بالا بدرستی قابل اجرا است اما این کد راندمان برنامه و خوانایی برنامه را بسیار کاهش می‌دهد. اگر شما به ثابت bmPoints توجه کنید متوجه خواهید شد که نام این ثابت یک کلمه جمع است و بجای نقطه به نقاط اشاره می‌کند. این مساله بیانگر این مطلب است که شما با دادن هر مختصات دلخواه ما بین بلوک glBegin/glEnd یک نقطه بر روی صفحه نمایش چاپ خواهد شد. بنابراین شما می‌توانید تکه کد بالا را به صورت زیر بنویسید.

```
glBegin bmPoints
        glvertex2f 0.0, 0.0
        glVertex2f 0.0, 1.0
glEnd
```

همانگونه که می‌بینید این تکه کد بسیار کوتاهتر، سریعتر و بهتر از آن تکه کد قبلی است.

۳.۱.۲ -- تغییر اندازه نقاط

شما در OpenGL این قابلیت را دارید که اندازه نقاط را تغییر بدهید و اندازه آنها را که بصورت پیش فرض یک پیکسل است به اندازه‌های بزرگتر همانند ۶۴ پیکسل تغییر دهید. برای تغییر اندازه نقاط در OpenGL تابعی با نام glPointSize وجود دارد. این تابع یک عدد اعشاری را می‌گیرد. و نقاط داده شده را به اندازه آن عدد تغییر می‌دهد.

```
glPointSize 2.2
glBegin bmPoints
        glVertex2f 1.0, 0.0
        glVertex2f 0.0, 0.0
glEnd
glPointSize 1
glBegin bmPoints
        glvertex2f 0.2, 0.3
glEnd
```

اگر به تکه کد بالا توجه کنید متوجه خواهید شد که از فراخوانی تابع glPointSize در مابین بلوک glBegin/glEnd جلوگیری کردیم. زیر اگر شما این تابع را درون این بلوک فراخوانی کنید موجب بروز خطایی از سوی OpenGL (خطاها در OpenGL موجب توقف برنامه نمی‌شوند. و شما برای اطلاع یابی از روی دادن آنها در OpenGL باید از دستور glGetError استفاده کنید که در آینده آن را شرح خواهیم داد) می‌شود. در نتیجه شما شکلی را مشاهده خواهید کرد که انتظار آن را ندارید.

۳.۱.۳ -- افزایش دقت گرافیکی نقاط

هرچند شما می‌توانید یک شکل را به بینهایت نقاط کوچک تقسیم کنید اما صفحه نمایش کامپیوتر دارای تعداد محدودی از نقاط (پیکسلها) است که بتواند یک شکل را، آنگونه که شما به نقاط ریز تفکیک کرده‌اید نشان بدهد. تعداد نقاط محدود در صفحه نمایش موجب می‌شود که لبه‌های اشکال بشکل دندانه‌دار دیده شوند. افزایش دقت گرافیکی به معنی هموار کردن این لبه‌ها به منظور اینکه شکل واقعی‌تر به نظر برسند. اگر شما می‌خواهید از این تکنیک برای دندانه زدایی اشکال استفاده کنید، باید ثابت glcPointSmooth را به تابع glEnable ارسال کنید.با اجرای خط زیر تکنیک دندانه‌زدایی برای نقاط فعال می‌شود.

```
glEnable glcPointSmooth
```

تابع glEnable: این تابع یکی از پرکاربردترین توابع OpenGL است. شما به کمک این تابع می‌توانید یکسری از قابلیت‌های OpenGL را همانند دندانه‌زدایی که بصورت پیش‌فرض فعال نیست را فعال کنید. در ادامه بحث خود درباره OpenGL اکثر ثوابتی را که این تابع می‌تواند بپذیرد را شرح خواهیم داد.

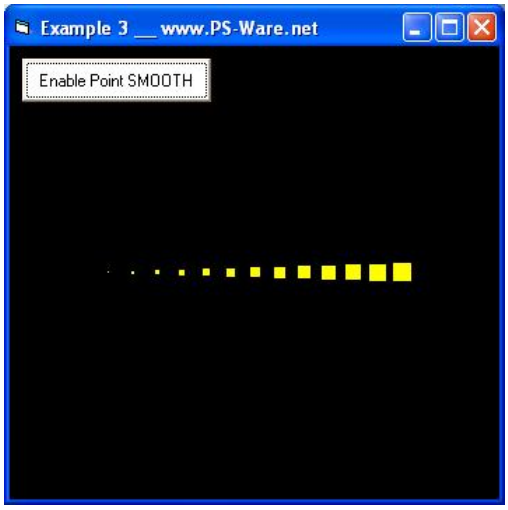
تابع glDisable: تابعی دیگر است که کار آن بالعکس تابع glEnable است. یعنی قابلیتی را که شما به عنوان پارامتر به آن ارسال می‌کنید را غیر فعال می‌کند. بنابراین برای غیر فعال کردن حالت دندانه‌زدایی از دستور زیر استفاده می‌کنیم

glDisable glcPointSmooth

امکان دارد در طول برنامه بخواهیم بررسی کنیم که آیا یک قابلیت خاص فعال است یا نه و با توجه به وضعیت آن تصمیم بگیریم. در OpenGL تابعی به نام glIsEnabled وجود دارد. وظیفه این تابع بررسی فعال بودن و یا غیر فعال بودن یک قابلیت است. به عنوان مثال در صورتی که قابلیت دندانه‌زدایی برای نقاط فعال باشد مقدار GL_TRUE توسط تابع برگشت داده خواهد شد و در غیر این صورت تابع مقدار GL_FALSE را برمیگرداند. در تکه کد زیر ابتدا بررسی می‌شود که خاصیت دندانه‌زدایی نقاط فعال است و یا خیر سپس با توجه به آن وضعیت دندانه‌زدایی را فعال و یا غیر فعال می‌کند.

```
If glIsEnabled(glcPointSmooth) = GL_TRUE Then
glDisable glcPointSmooth
Else
glEnable glcPointSmooth
End If
```

در شکل زیر نمایی از برنامه شماره ۳ (در داخل فایل ZIP همراه کتاب به نام Ex3 ذخیره شده است) را می‌بینید. در این برنامه، ۱۰ نقطه با اندازه‌های مختلف رسم شده که شما می‌توانید با کلیک کردن بر روی دکمه حالت دندانه‌زدایی را برای نقاط فعال و یا غیر فعال کنید.



شکل ۹ – نمای از برنامه شماره ۳

۳.۲ -- رسم خطوط

برای رسم خطوط در OpenGL از دستور glBegin با ثابت bmLines استفاده می‌کنیم. در این حالت بجای یکبار فراخوانی تابع glVertex2f در بلوک glBegin/glEnd، برای رسم هر خط دوبار آن را فراخوانی می‌کنیم.در مثال زیر یک خط با رنگ سبز از سمت چپ صفحه نمایش به سمت راست آن کشیده می‌شود.

```
glBegin bmLines
        glVertex2f -1, 0
        glVertex2f 1, 0
glEnd
```

در بلوک glBegin/glEnd بجای هر دو فراخوانی دستور glVertex2f، یک خط رسم می‌شود.

۳.۲.۱ -- تغییر دادن عرض خطوط

همانطور که می‌دانید تمامی خطوط دارای ضخامت هستند و شما می‌توانید خطوط مختلف را با ضخامت‌های متفاوت رسم کنید. برای این عمل در OpenGL تابعی با نام glLineWidth وجود دارد که عمل تنظیم عرض خط را (یا همان ضخامت آن) را انجام می‌دهد. این تابع یک مقدار اعشاری را می‌گیرد و ضخامت خط را با توجه به آن عدد تغییر می‌دهد. همچنین در OpenGL به شما این قابلیت داده شده که با استفاده از ثابت glgLineWidth و تابع glGetFloatv مقدار کنونی ضخامت خط را بدست آورید. برای درک بهتر مطلب به شما پیشنهاد می‌کنم که به مثال زیر توجه کنید

```
GLfloat oldWidth
glGetFloatv glgLineWidth , oldWidth
glLineWidth oldWidth + 1
```

۳.۲.۲ -- دندانه‌زدایی خطوط

داندانه‌زدایی برای خطوط همانند دندانه‌زدایی برای نقاط است.تنها تفاوت این دو در ارسال پارامتر glcLineSmooth به جای ثابت glcPointSmooth به تایع glEnable است. همچنین همانند دندانه‌زدایی نقاط شما با استفاده از تابع glDisable می‌توانید دندانه‌زدایی را برای خطوط غیر فعال کنید.

۳.۲.۳ -- تعیین الگو برای خطوط

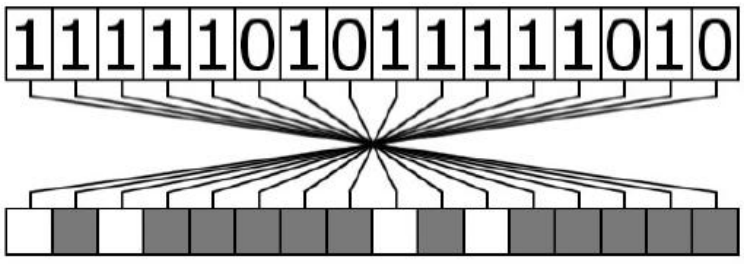
در OpenGL به شما این قابلیت داده شده که یک الگوی عرضی را خطوط تعیین کنید. شما به وسیله این الگو می‌توانید تعیین کنید که کدام قسمتها در هنگام رسم خطوط حذف شوند و یا کدام قسمتها رسم شوند. با استفاده از الگو برای رسم خطوط شما می‌توانید خطوطی را همانند خطوط نقطه چین رسم کنید.

برای تعیین الگو برای خطوط در OpenGL ابتدا باید قابلیت الگو دهی خطوط را فعال کنیم سپس اقدام به رسم خطوط بکنیم. برای فعال کرد این قابلیت در OpenGL، تابع glEnable را با ثابت glcLineStipple فراخوانی می‌کنیم. و برای غیر فعال کردن این قابلیت از تابع glDisable با همین ثابت استفاده می‌کنیم. بعد از فعال کردن این قابلیت از تابع glLineStipple برای تعیین الگوی خط استفاده می‌کنیم.

```
Sub glLineStipple (factor As GLint, pattern As GLushort)
```

پارامتر factor تعیین می‌کند که هر بیت از الگو چند بار باید تکرار شود. این پارامتر می‌تواند مقادیر مابین یک تا ۲۵۶ را بگیرد و استفاده از آن با مقادیر بزرگتر از یک موجب بزرگتر شدن طولی الگو می‌شود. من همیشه مقدار دو را برای این پارامتر ترجیح می‌دهم.

پارامتر pattern دارای یک مقدار ۱۶ بیتی است که شکل الگوی خط را تعیین می‌کند. برای ساختن الگوی مورد نظر خود را در ۱۶ خانه کنار هم رسم کنید سپس در عوض هر خانه‌ی سیاه مقدار یک و هر خانه سفید مقدار صفر را بگذارید.بعد از آن کد به دست آمده را که به صورت دودویی است را متقارن کنید و آن را به یک مقدار در مبای شانزده تبدیل کنید.



شکل ۱۰ - نمونه‌ای از یک الگوی عرضی خط

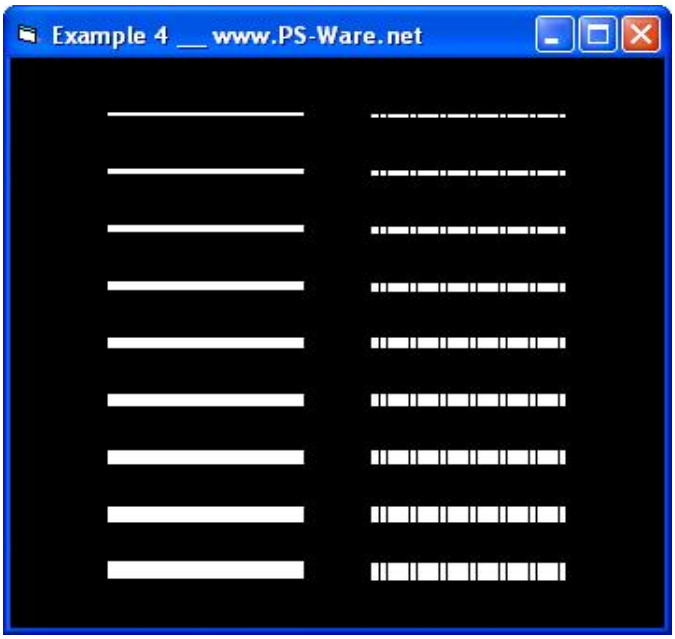
در زیر شما می‌توانید مثال مربوط به تعیین الگوی شکل شماره ۱۰ را به یک خط ببینید.

```
Dim LinePattern AS GLushort
GLushort = &HFAFA
glEnable glcLineStipple
glLineStipple 2, LinePattern

glBegin bmLines
        glVertex2f -0.5, 0
        glVertex2f +0.5, 0
glEnd

glDisable glcLineStipple
```

دستورات دیگری و یا ثابت‌های دیگری نیز برای رسم خطوط موجوداند که از حوصله این فصل از کتاب خارج هستند. در فصول بعدی در مورد این توابع بیشتر بحث خواهیم کرد. در شکل زیر شما نمایی از مثال شماره چهار که همراه این کتاب الکترونیکی است (فایل فشرده‌ای که نام آن Ex4 است) را می‌بینید.



شکل ۱۱ - نمایی از مثال شماره ۴

۳.۲.۴ -- رسم خطوط پیوسته

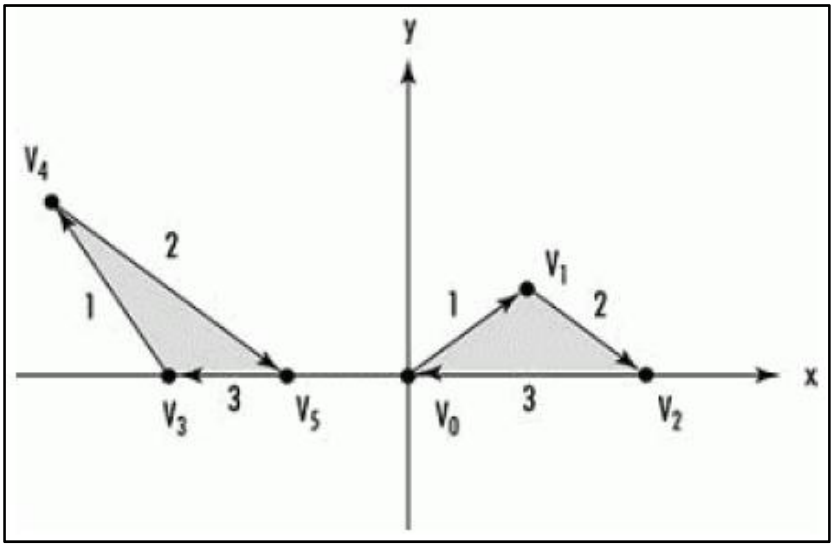
تا کنون شما کار با خطوط ساده را فرا گرفته‌اید اما در OpenGL دو نوع دیگر خط هم وجود دارد که عبارتند از bmLineStrip و bmLineLoop. این دو ثابت در تابع glBegin استفاده می‌شوند و تنها تفاوتی را که در استفاده از آنها با bmLines مشاهده می‌کنید این است که تمام خطوط به صورت پیوسته به یکدیگر متصل هستند و بجای هر دو راس یک خط جداگانه رسم نمی‌شود، بلکه راس جدید به راس قبلی آن متصل می‌شود. در صورتی که از ثابت bmLineLoop استفاده می‌کنید، راس آخر به راس اول متصل می‌شود. این حالت همانند این است که شما یک چند ضلعی توخالی رسم کرده‌اید. کار با این ثابتها را به خود شما واگذار می‌کنم. در صورتی که هنگام کار با این ثابتها به مشکل برخورد کردید می‌توانید مشکل خود را در انجمن ساخت بازیهای رایانه‌ای در وب سایت www.ps-ware.net مطرح کنید.

۳.۳ -- رسم مثلث

مثلث‌ها ساده‌ترین چندضلعی‌ها هستند. برای رسم آنها در OpenGL تنها کافیست ثابت bmTriangles را به تابع glBegin ارسال کنیم و در بلوک glBegin/glEnd سه راس را قرار دهیم. در مثال زیر شیوه رسم یک مثلث توپر در OpenGL نشان داده شده است.

```
glBegin bmTriangles
        glVertex2f -2.5, -2.5
        glVertex2f 2.5, -2.5
        glVertex2f 0.0, 2.5
glEnd
```

شیوه رسم مثلث‌های توخالی در ادامه همین فصل توضیح داده‌خواهد شد. به ازای هر سه راسی که شما در بلوک glBegin/glEnd قرار می‌دهید یک مثلث رسم خواهد شد.اما در صورتی که تعداد راس‌های شما مضربی از سه نباشد OpenGL تولید خطا خواهد کرد. نکته‌ی مهمی که باید همیشه در رسم مثلث‌ها و تمام چند ضلعی‌ها رعایت کنید اینست که همیشه ترتیب مختصات رئوس باید به صورت پاد ساعتگرد باشد. در صورتی که شما همانند شکل زیر مثلث‌ها در جهت ساعتگرد رسم کنید پشت آنها را خواهید دید، این مساله در طول همین فصل به صورت کامل توضیح داده خواهد شد.



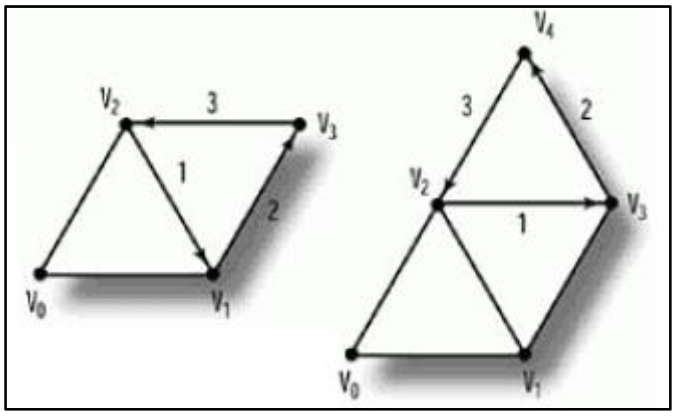
شکل ۱۲- ترتیب راسها در مثلثها و چندضلعی‌ها

این ترتیب را بدلیل این مساله رعایت می‌کنیم که پشت و روی اشکال هندسی را از همدیگر متمایز کنیم. در حقیقت هنگامی که شما یک شکل هندسی را در حالت سه بعدی رسم می‌کنید. این شکل دارای دو جهت است، همانند یک برگ کاغذ که دارای پشت و رو است. اگر شما مختصات رئوس شکل را در جهت پاد عقربه‌های ساعت بدهید آن را به رو و در صورتی که آن را در جهت عقربه‌های ساعت رسم کنید آن را به پشت رسم کرده‌اید. شاید این مساله در حال حاضر چندان اهمیت نداشته باشد اما هنگام تخصیص بافر و یا نور پردازی دارای اهمیت ویژه‌ای است.

۳.۳.۱ -- مثلث‌های پیوسته

شما با کشیدن مثلث‌های متصل به هم به راحتی می‌توانید اشکال پیچیده هندسی مانند یک کره و یا یک مخروط را رسم کنید. برای کشیدن اینگونه از مثلث‌ها از ثابت bmTriangleStrip در تابع glBegin استفاده می‌کنیم. در اینگونه از اشکال ترتیب راس‌ها دارای اهمیت ویژه‌است. بطور که هر راس جدید

به دو راس قبل از خود متصل می‌شود. رعایت نکردن ترتیب رئوس شکل ممکن است منجر به به نمایش در آمدن شکلی دور از انتظار شما شود. در شکل شماره ۱۳ شما می‌توانید ترتیب رسم رئوس را برای مثلث‌های بهم پیوسته ببینید.

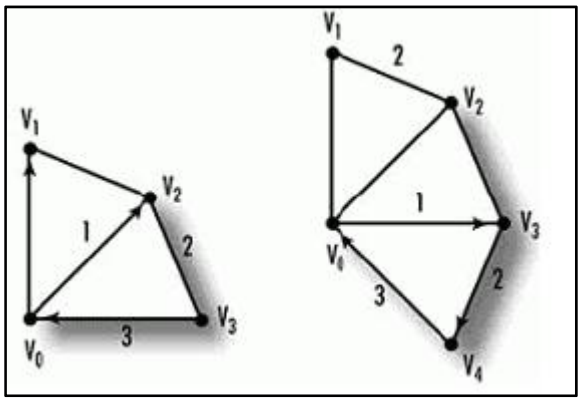


شکل ۱۳- ترتیب رعایت رئوس در مثلث‌های پیوسته

استفاده از این روش بجای استفاده از مثلث‌های جداگانه دارای مزیتهای زیر است:

- افزایش سرعت برنامه
- افزایش کارایی برنامه
- اشغال کمتر حافظه
- سهولت در برنامه‌نویسی

در OpenGL ثابتی دیگری برای رسم مثلث‌های بهم پیوسته وجود دارد و آن bmTriangleFan است. به کمک این ثابت شما می‌توانید مثلث‌هایی رسم کنید که همگی‌آنها دارای یک راس مشترک هستند. شکل شماره ۱۴ شیوه رسم این گونه اشکال را در OpenGL نمایش می‌دهد.



شکل ۱۴- ترتیب رئوس مثلث‌هایی که دارای راس مشترک هستند.

همانگونه که در شکل بالا می‌بینید، راس اول همیشه به عنوان راس مشترک انتخاب شده و رئوس بعدی به راس اول متصل شده است.

برای کار با چند ضلعی‌ها و مثلث‌ها توابع زیادی درOpenGL موجود هستند که به شما قابلیت بسیار زیادی را می‌دهند به گونه‌ای که شما قادر خواهید بود هر تغییر دلخواهی را برروی آنها اعمال کنید. بسیاری لز این توابع در ادامه این فصل توضیح داده می‌شوند اما قبل از اینکه به تشریح این توابع بپردازیم ابتدا سعی می‌کنیم انواع چند ضلعی‌ها را معرفی کنیم.

۳.۴ -- چهارضلعی‌ها

مربع‌ها و مستطیل‌ها دو نمونه از معروفترین چهار ضلعی‌ها هستند. برای رسم چهار ضلعی‌ها در OpenGL از ثابت bmQuads در تابع glBegin استفاده می‌کنیم. هنگامی که ما از این ثابت استفاده می‌کنیم OpenGL در ازای هر چهار راس (مختصات داده شده) یک چهار ضلعی رسم می‌کند پس اگر ما دورن بلوک glBegin/glEnd هشت راس را تعریف کنیم OpenGL برای ما دو چهار ضلعی را رسم خواهد کرد. در هنگام رسم چهار ضلعی‌ها با رئوس به ترتیب همانند مثلث‌ها در جهت پاد عقربه‌های ساعت بکشیم. در مثال زیر شما نمونه کد رسم یک چهار ضلعی را در OpenGL می‌توانید ببینید.

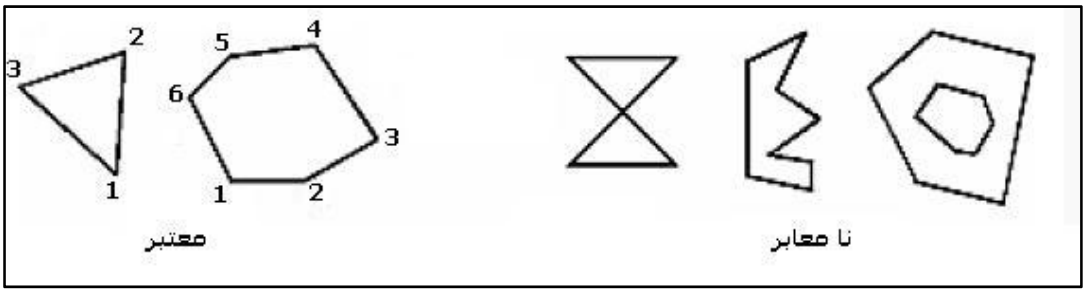
```
glBegin bmQuads
        glVertex2f -0.5, -0.5
        glVertex2f -0.5, 0.5
        glVertex2f 0.5, 0.5
        glVertex2f 0.5, -0.5
glEnd
```

۳.۴.۱ -- رسم چهار ضلعی‌ها پیوسته

از جمله از شکل‌هایی که در OpenGL کاربرد بسیار دارند چهار ضلعی‌های متصل به هم هستند. برای رسم اینگونه از چهار ضلعی‌ها از ثابت bmQuadStrip استفاده می‌کنیم. هنگامی که شما از این ثابت استفاده می‌کنید OpenGL به ازای هر دو راسی که شما به آن می‌دهید یک مربع ایجاد می‌کند که دو راس دیگر آن دو راس آخر چهار ضلعی قبل از آن است. شیوه کار با اینگونه از چهار ضلعی‌ها تقریبا همانند شیوه کار با چهار ضلعی‌های ساده است بنابراین دیگر بیشتر از این به توضیح اینگونه از چهار ضلعی‌ها نمی‌پردازیم.

۳.۵ -- چند ضلعی‌ها

هر شکل هندسی را که بیشتر از دو راس داشته باشد چند ضلعی می‌نامیم.تمام مثلث‌ها، چهار ضلعی‌ها و یا حتی پنج ضلعی‌ها همگی جزء چند ضلعی‌ها محسوب می‌شوند. برای کشیدن چند ضلعی‌ها در OpenGL به تابع glBegin ثابت bmPolygon را ارسال می‌کنیم. در شکل شماره ۱۵ شما می‌توانید نمونه‌هایی از چند ضلعی‌ها را ببینید. اگر به این شکل دقت کنید متوجه می‌شوید که رئوس در جهت پاد ساعتگرد رسم شده‌اند.

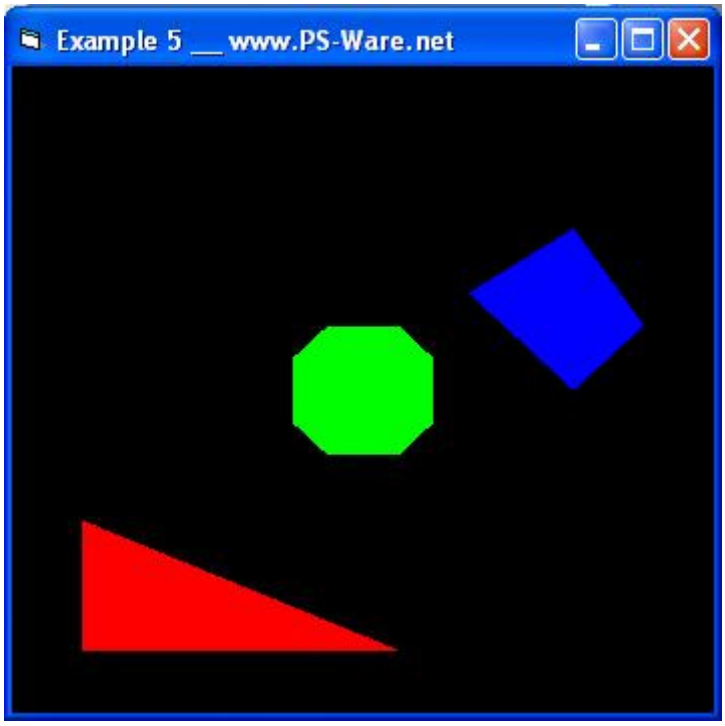


شکل ۱۵

همانگونه که در شکل بالا می‌توانید ببینید در تابع glBegin با استفاده از ثابت bmPolygon تنها اجازه رسم چند ضلعی‌های محدب به شما داده می‌شود. در آینده در مورد رسم چند ضلعی‌های غیر محدب نیز بحث خواهیم کرد. حال کار خود را با یک مثال ادامه می‌دهیم. در این مثال ما یک هشت ضلعی سبز رنگ را در مرکز صفحه نمایش رسم خوایم کرد.

```
glColor4f 0, 1, 0, 1
glBegin bmPolygon
        glVertex2f -0.1, -0.2
        glVertex2f 0.1, -0.2
        glVertex2f 0.2, -0.1
        glVertex2f 0.2, 0.1
        glVertex2f 0.1, 0.2
        glVertex2f -0.1, 0.2
        glVertex2f -0.2, 0.1
        glVertex2f -0.2, -0.1
glEnd
```

همانگونه که می‌بینید کار با چند ضلعی‌ها به سادگی کار با مثلث و با چهار ضلعی‌ها است تنها کاری که شما برای کار با چند ضلعی‌ها باید بکنید این است که مختصات رئوس آنها را پیدا کنید. در شکل زیر شما نمایی از سورس کد شمار ۵ (فایل ex5.zip) را می‌بینید.



شکل ۱۶- نمایی از برنامه نمونه شماره ۶

۳.۶ -- ویژگیهای مشترک چند ضلعی‌ها

تمامی چندضلعی‌ها، سه ضلعی‌ها و یا حتی چهار ضلعی‌ها همگی دارای یکسری از ویژگی‌های مشترک هستند. برای اینکه بتوانیم از تمام قدرت OpenGL و انعطاف‌پذیری آن استفاده بکنیم.باید با این ویژگی‌ها آشنایی داشته باشیم.

هنگام رسم چند ضلعی‌ها در OpenGL بصورت پیش‌فرض تمام آنها به صورت توپر رسم خواهند شد. در OpenGL این قابلیت به شما داده شده است که این رفتار پیش‌فرض را به دلخواه خود تغییر بدهید. به عنوان مثال کاری کنید که تمام چند ضلعی‌ها به صورت تو خالی دیده شوند. برای تغییر این حالت پیش‌فرض از تابع glPolygonMode استفاده می‌کنیم.

Sub glPolygonMode (face as glFaceConstants, mode as glPolygonModeConstants)

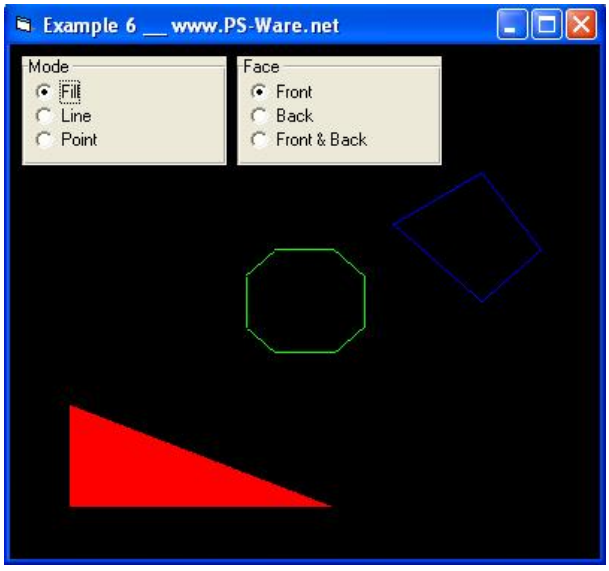
همانگونه که پیش از این نیز اشاره کردیم OpenGL هر شکل را (به هر میزان که بخواهد نازک باشد) به دو جهت جلو و پشت تفکیک می‌کند؛ در نتیجه هنگامی که شما تابع glPolygonMode را تعیین می‌کنید. باید تعیین کنید که پشت و یا روی شکل را می‌خواهید تغییر دهید. برای اینکه به OpenGL بگویید تغییرات را در پشت شکل اعمال کند از ثابت faceFront برای جلوی اجسام , faceBack برای پشت اجسام و faceFrontAndBack برای هر دو روی آنها در پارامتر face استفاده می‌کنیم.

پارامتر mode می‌تواند سه ثابت مختلف را قبول کند که در جدول زیر نام ثابت‌ها همرا با توضیح آنها آمده است.

جدول شماره ۲

ثابت	توضیح
pgmFILL	این ثابت بیانگر همان حالت پیش‌فرض است که استفاده از آن موجب می‌شود که چند ضلعی‌ها به صورت توپر رسم شوند.
pgmLine	همانند فراخوانی تابع glBegin با ثابت bmLineLoop است. هنگام استفاده از این ثابت چند ضلعی‌ها به صورت تو خالی رسم می‌شوند.
pgmPoint	هنگام استفاده از این ثابت به ازای هر راس یک نقطه رسم می‌شود. استفاده از این ثابت همانند استفاده از ثابت bmPoints در تابع glBegin است.

در سورس کد نمونه شماره ۶ (فایل ex6.zip) که همراه این کتاب الکترونیکی است. کار با انواع حالت‌های این تابع نشان داده شده است. در شکل شماره ۱۷ شما می‌توانید نمایی از این برنامه را ببینید. در این برنامه یک مثلث وجود دارد که به پشت رسم شده و دو شکل بعد به رو رسم شده است.

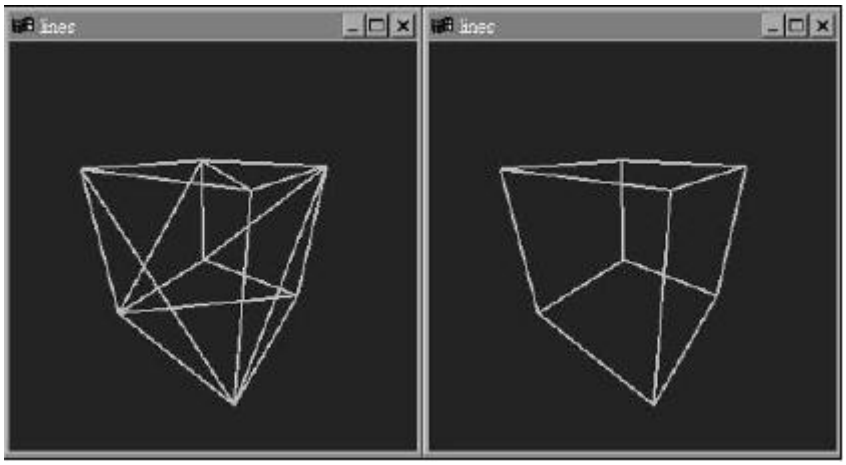


شکل ۱۷- نمایی از سورس کد شماره ۶

۳.۶.۲ -- حذف اضلاع اضافی

در هنگام رسم چندضلعی‌ها در OpenGL ممکن است بخواهید بعضی از اضلاع آنها رسم نشود. به عنوان مثال هنگامی‌که می‌خواهید به کمک دو مثلث یک مربع را رسم کنید ممکن است نخواهید که ضلع مشترک مثلث‌ها دیده شود. برای اینکار در OpenGL تابعی به نام glEdgeFlag وجود دارد. شما به کمک این تابع می‌توانید اضلاع اضافی را در OpenGL حذف کنید.

تابع glEdgeFlag دارای یک پارامتر است در صورتی که به این پارامتر مقدار GL_FALSE را بدهیم ضلع بعد از راس رسم نخواهد شد و در صورتی که مقدار GL_TRUE را به آن بدهیم ضلع‌های بعد از تابع رسم خواهند شد. در سورس کد نمونه شماره ۷ (فایل ex7.zip) که همراه این کتاب الکترونیکی است نمونه کار با این تابع آمده است. در شکل زیر شما یکی از کاربرد های این تابع را می‌توانید ببینید.



شکل ۱۸- یکنمونه کار سه بعدی ازتابع glEdgeFlag

۳.۶.۳ -- دندانه زدایی چند ضلعی‌ها

همانند نقاط و خطوط حال دندانه‌دایی در چند ضلعی‌ها هم وجود دارد. شما با کمک ثابت glcPolygonSmooth می‌توانید قابلیت دندانه‌زدیی را برای چند ضلعی‌ها فعال یا غیر فعال کنید. و با کمک ثابت glePolygonSmooth در تابع glIsEnabled می‌توانید این حالت را چک بکنید.

```
If glIsEnabled (glePolygonSmooth) = GL_FALSE then
        glEnable glcPolygonSmooth
```

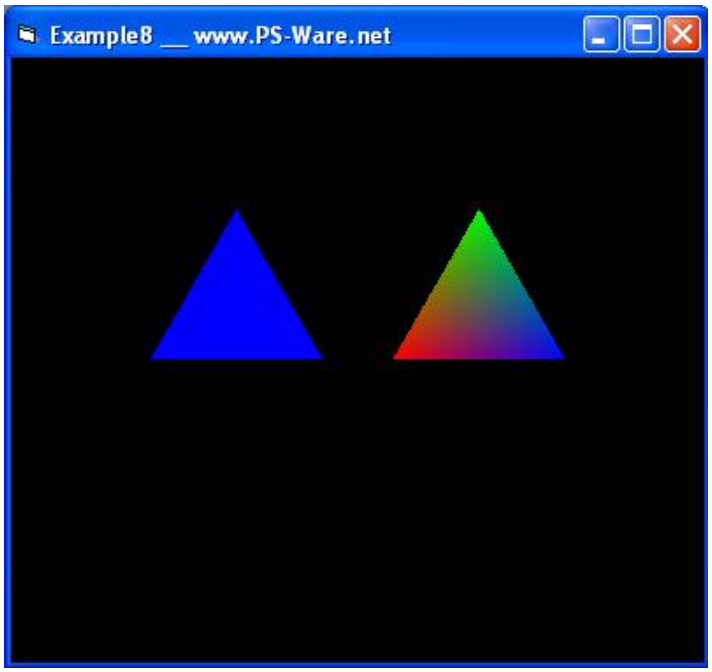
تشریح خواص مشترک دیگر چند ضلعی‌ها را به فصل پنج این کتاب الکترونیکی که مربوط به رسم اشکال سه بعدی است واگذار می‌کنیم و در همین قسمت به این مبحث خاتمه می‌دهیم.

۳.۷ -- ترکیب رنگ

در OpenGL این قابلیت به شما داده شده که برای هر راس یک رنگ خاص را تعریف کنید. در حالت پیش‌فرض اگر شما رنگهای متفاوتی را به رئوس بدهید خواهید دید که کل شکل با رنگ آخرین راس ترسیم خواهد شد. اما اگر بخواهیم به هر راس رنگ جداگانه بدهیم از تابع glShadeModel استفاده می‌کنیم. این تابع یک پارامتر دارد که این پارامتر دو ثابت smSmooth و smFlat را به عنوان ورودی قبول می‌کند. در مثال زیر دو مثلث در دو حالت مختلف ترکیب رنگ رسم می‌شوند که نتیجه این رسم این دو مثلث را در OpenGL می‌توانید در شکل شماره ۱۹ ببینید.

```
glShadeModel smSmooth
glBegin bmTriangles
    glColor4f 1, 0, 0, 1
    glVertex2f 0.1, 0
    glColor4f 0, 0, 1, 1
    glVertex2f 0.6, 0
    glColor4f 0, 1, 0, 1
    glVertex2f 0.35, 0.5
glEnd

GL.glShadeModel smFlat
glBegin bmTriangles
    glColor4f 1, 0, 0, 1
    glVertex2f -0.6, 0
    glColor4f 0, 1, 0, 1
    glVertex2f -0.1, 0
    glColor4f 0, 0, 1, 1
    glVertex2f -0.35, 0.5
glEnd
```



شکل ۱۹

در صورتی که شما از ثابت smSmooth استفاده کنید رنگ رئوس شکل با هم ترکیب می‌شوند و در غیر این صورت رنگ آخرین راس به کلیه رئوس دیگر تاثیر می‌گذارد.

در همینجا به قسمت اول از سری آموزش برنامه نویسی OpenGL در ویژوال در ویژوال بیسیک خاتمه می‌دهیم. امیدوارم مطالب این قسمت از این سری آموزشی برای شما مفید بوده باشد.

منابع

۱. وب سایت www.Nehe.gamedev.net
۲. وبلاگ www.OpenGL.blogfa.com
۳. کتاب الکترونیکی Beginning OpenGL Game Programming
۴. کتاب الکترونیکی اصول گرافیک کامپیوتری - نوشته دکتر جواد مهری
۵. کتاب الکترونیکی OpenGL Red Book
۶. وب سایت http://www.devmaster.net

برنامه‌نویسی OpenGL در ویژوال بیسیک ۶

دسته‌بندی: برنامه‌نویسي و گرافیک کامپیوتری
پوشش می‌دهد: شیوه آموزش و مبانی اولیه برنامه‌نویسی OpenGL در ویژوال بیسیک ۶
سطح کاربر: متوسط به بالا

تهیه شده: توسط پویا شاهین‌فر
انتشارات: انتشارات مجازی PS-Ware (PS-Press)
تعداد صفحات: ۳۸ صفحه
تاریخ انتشار: ۱۲ بهمن ۱۳۸۵
