به نام خدا

گرافیک در زبانهای مختلف، ابزارهای مختلف دارد. یکی از این ابزارها که بین چند زبان و platform مشترک است SDL یا Simple Directmedia Layer است. چون SDL دستورات پایه مانند خط و دایره و مستطیل و ... را ندارد، کتابخانه‌ی sdl gfx primitives به وجود آمده تا بتوان از این دستورات استفاده کرد.

برای هر ترسیم ابتدا باید یک صفحه حاضر کرد. با استفاده از دستورات زیر می‌توان این صفحه را آماده کرد:

```
   SDL_Init( SDL_INIT_VIDEO );
   SDL_Surface* screen = SDL_SetVideoMode( WINDOW_WIDTH, WINDOW_HEIGHT, 0,
SDL_HWSURFACE | SDL_DOUBLEBUF | SDL_FULLSCREEN );
   SDL_WM_SetCaption( WINDOW_TITLE, 0 );
```

پس از اجرای این دستورات، می‌توان از screen به‌عنوان شئ مورد ترسیم استفاده کرد. همه‌ی دستورات معمولاً به‌عنوان اولین پارامتر ورودی شئ screen را می‌پذیرند.

دستور SDL_Init کتابخانه‌ی SDL را راه‌اندازی می‌کند و باید قبل از هر تابع دیگری صدا زده شود. ورودی آن نیز یک پرچم است که نشان می‌دهد کدام زیر سیستم‌ها از SDL را می‌خواهیم فعال کنیم. در جدول زیر لیست آن‌ها آمده است:

SDL_INIT_TIMER	timer subsystem
SDL_INIT_AUDIO	audio subsystem
SDL_INIT_VIDEO	video subsystem
SDL_INIT_JOYSTICK	joystick subsystem
SDL_INIT_HAPTIC	haptic (force feedback) subsystem
SDL_INIT_GAMECONTROLLER	controller subsystem
SDL_INIT_EVENTS	events subsystem
SDL_INIT_EVERYTHING	all of the above subsystems
SDL_INIT_NOPARACHUTE	compatibility; this flag is ignored

هر کدام از این زیر سیستم‌ها را بخواهیم باید با هم or کنیم، مثلاً اگر audio, video را می‌خواهیم فعال کنیم، باید تابع را به‌صورت `SDL_Init( SDL_INIT_VIDEO | SDL_INIT_AUDIO );` صدا بزنیم.

تابع `SDL_SetVideoMode` نیز پنجره ترسیم را آماده می‌کند و به‌صورت کلی زیر تعریف شده است:

```
SDL_Surface *SDL_SetVideoMode(int width, int height, int bpp, Uint32 flags);
```

تکلیف width و height که مشخص است و طول و ارتفاع پنجره را مشخص می‌کنند. bpp نیز بیت بر پیکسل است که برای استفاده از مقدار پیش‌فرض آن را 0 می‌گذاریم. اما falgs پرچم‌هایی را مشخص می‌کند که مثل دستور قبل حالات مختلف آن را می‌توانیم با or کردن به‌وجود بیاوریم. لیست پارامترهای قابل قبول به شرح زیر است:

SDL_SWSURFACE	Create the video surface in system memory
SDL_HWSURFACE	Create the video surface in video memory
SDL_ASYNCBLIT	Enables the use of asynchronous updates of the display surface. This will usually slow down blitting on single CPU machines, but may provide a speed increase on SMP systems.
SDL_ANYFORMAT	Normally, if a video surface of the requested bits-per-pixel (*bpp*) is not available, SDL will emulate one with a shadow surface. Passing SDL_ANYFORMAT prevents this and causes SDL to use the video surface, regardless of its pixel depth.
SDL_HWPALETTE	Give SDL exclusive palette access. Without this flag you may not always get the the colors you request with SDL_SetColors or SDL_SetPalette.
SDL_DOUBLEBUF	Enable hardware double buffering; only valid with SDL_HWSURFACE. Calling SDL_Flip will flip the buffers and update the screen. All drawing will take place on the surface that is not displayed at the moment. If double buffering could not be enabled then SDL_Flip will just perform a SDL_UpdateRect on the entire screen.
SDL_FULLSCREEN	SDL will attempt to use a fullscreen mode. If a hardware resolution change is not possible (for whatever reason), the next higher resolution will be used and the display window centered on a black background.
SDL_OPENGL	Create an OpenGL rendering context. You should have previously set OpenGL video attributes with SDL_GL_SetAttribute.
SDL_OPENGLBLIT	Create an OpenGL rendering context, like above, but allow normal blitting operations. The screen (2D) surface may have an alpha channel, and SDL_UpdateRects must be used for updating changes to the screen surface. NOTE: This option is kept for compatibility only, and is *not* recommended for new code.
SDL_RESIZABLE	Create a resizable window. When the window is resized by the user a SDL_VIDEORESIZE event is generated and SDL_SetVideoMode can be called again with the new size.
SDL_NOFRAME	If possible, SDL_NOFRAME causes SDL to create a window with no title bar or frame decoration. Fullscreen modes automatically have this flag set.

دستور `SDL_WM_SetCaption` نیز عنوان و آیکون صفحه‌ی گرافیکی را مشخص می‌کند. تابع SDL_Flip نیز به نوعی صفحه را refresh می‌کند تا اگر ترسیمات جدیدی انجام شده، کاربر آن‌ها را مشاهده کند.

SDL_Event یک نوع تعریف شده در کتابخانه‌های SDL است که رخدادهای مختلف را می‌تواند نگه دارد. از فشرده شدن یک کلید تا تغییر وضعیت موس و پنجره و ... برای این کار می‌توان از تابع SDL_PollEvent(&event) استفاده کرد. کلیه‌ی توضیحات مربوط به رخدادها در https://wiki.libsdl.org/SDL_Event آمده است. مثلاً می‌توان فهمید چه کلیدی فشرده یا رها شده است (با استفاده از event.key.keysym.sym یا event.key.keysym.scancode). البته می‌توان برخی از این پارامترها را جداگانه نیز بررسی نمود، مثلاً اخذ مختصات فعلی موس با تابع SDL_GetMouseState امکان‌پذیر است.

دستورات گرافیکی پایه، خیلی عجیب نیستند. این دستورات عمدتاً دو دسته هستند:

۱. دستوراتی که به color ختم می‌شوند
۲. دستوراتی که به RGBA ختم می‌شوند

دستوراتی که به RGBA ختم می‌شوند، در انتهای همه‌ی پارامترهای خود، ۴ پارامتر r, g, b, a یعنی قرمز و سبز و آبی و آلفای رنگ (میزان تیره و روشن بودن رنگ) نیز از کاربر می‌گیرد تا رنگ ترسیم مورد نظر را تنظیم کند. دستورات ختم شده به color نیز دقیقاً همین چهار پارامتر را می‌گیرند، ولی به شکل یک عدد (معمولاً مبنای ۱۶ به شکل 0xRRGGBBAA). مثلاً اگر کاربر عدد 0xFF0000FF را وارد کند، یعنی قرمز خالص. در ضمن پارامتر آلفا را معمولاً بر روی ۲۵۵ یا FF هگز تنظیم می‌کنیم که هر رنگ حداکثر اثر واقعی خود را داشته باشد. اگر آلفا صفر شود، رنگ‌ها سیاه می‌شوند.

حال مثلاً اگر بخواهیم یک خط قرمز ترسیم کنیم هر یک از دو دستور زیر می‌تواند کار مورد نظر را انجام دهد:

```
lineColor(screen, 100, 100, 400, 200, 0xFF0000FF );
lineRGBA(screen, 100, 100, 400, 200, 255, 0, 0, 255 );
```

با توجه به این توضیحات، کافیست توابع مختلف را در این زمینه ببینید.

pixel

```
int pixelColor( surface, x, y, color );
int pixelRGBA(  surface, x, y, r, g, b, a );
```

Draws a pixel at point `x`/`y`. You can pass the color by `0xRRGGBBAA` or by passing 4 values. One for red, green, blue and alpha.

```
 pixelColor(surface, 2, 2, 0xFF0000FF);  // red pixcel
pixelRGBA( surface, 4, 4, 0x00, 0xFF, 0x00, 0xFF); // green pixel
```

hline

```
int hlineColor( surface, x1, x2, y, color );
int hlineRGBA(  surface, x1, x2, y, r, g, b, a );
```

Draws a line horizontally from `x1`/`y` to `x2`/`y`.

vline

```
int vlineColor( surface, x, y1, y2, color );
int vlineRGBA(  surface, x, y1, y2, r, g, b, a );
```

Draws a line vertically from `x`/`y1` to `x`/`y2`.

rectangle

```
int rectangleColor( surface, x1, y1, x2, y2, color );
int rectangleRGBA(  surface, x1, y1, x2, y2, r, g, b, a );
```

Draws a rectangle. Upper left edge will be at `x1`/`y1` and lower right at `x2`/`y`. The colored border has a width of 1 pixel.

box

```
int boxColor( surface, x1, y1, x2, y2, color );
int boxRGBA(  surface, x1, y1, x2, y2, r, g, b, a );
```

Draws a filled rectangle.

line

```
int lineColor( surface, x1, y1, x2, y2, color );
int lineRGBA(  surface, x1, y1, x2, y2, r, g, b, a );
```

Draws a free line from `x1`/`y1` to `x2`/`y`.

aaline

```
int aalineColor( surface, x1, y1, x2, y2, color );
int aalineRGBA(  surface, x1, y1, x2, y2, r, g, b, a );
```

Draws a free line from `x1`/`y1` to `x2`/`y`. This line is anti aliased.

circle

```
int circleColor( surface, x, y, r, color );
int circleRGBA(  surface, x, y, rad, r, g, b, a );
```

arc

```
int arcColor( surface, x, y, r, start, end, color );
int arcRGBA(  surface, x, y, rad, start, end, r, g, b, a );
```

Note: You need lib SDL_gfx 2.0.17 or greater for this function.

aacircle

```
int aacircleColor( surface, x, y, r, color );
int aacircleRGBA(  surface, x, y, rad, r, g, b, a );
```

Note: You need lib SDL_gfx 2.0.17 or greater for this function.

filled_circle

```
int filledCircleColor( surface, x, y, r, color );
```

```
int filledCircleRGBA(  surface, x, y, rad, r, g, b, a );
```

ellipse

```
int ellipseColor( surface, x, y, rx, ry, color );
int ellipseRGBA(  surface, x, y, rx, ry, r, g, b, a );
```

aaellipse

```
int aaellipseColor( surface, xc, yc, rx, ry, color );
int aaellipseRGBA(  surface, x, y, rx, ry, r, g, b, a );
```

filled ellipse

```
int filledEllipseColor( surface, x, y, rx, ry, color );
int filledEllipseRGBA(  surface, x, y, rx, ry, r, g, b, a );
```

pie

```
int pieColor( surface, x, y, rad, start, end, color );
int pieRGBA(  surface, x, y, rad, start, end, r, g, b, a );
```

This draws an opened pie. `start` and `end` are degree values. `0` is at right, `90` at bottom, `180` at left and `270` degrees at top.

filled pie

```
int filledPieColor( surface, x, y, rad, start, end, color );
int filledPieRGBA(  surface, x, y, rad, start, end, r, g, b, a );
```

trigon

```
int trigonColor( surface, x1, y1, x2, y2, x3, y3, color );
int trigonRGBA(  surface, x1, y1, x2, y2, x3, y3, r, g, b, a );
```

aatrigon

```
int aatrigonColor( surface, x1, y1, x2, y2, x3, y3, color );
int aatrigonRGBA(  surface, x1, y1, x2, y2, x3, y3, r, g, b, a );
```

filled trigon

```
int filledTrigonColor( surface, x1, y1, x2, y2, x3, y3, color );
int filledTrigonRGBA(  surface, x1, y1, x2, y2, x3, y3, r, g, b, a );
```

polygon

```
int polygonColor( surface, vx, vy, n, color );
int polygonRGBA(  surface, vx, vy, n, r, g, b, a );
```

Example:

```
 polygonColor(display, [262, 266, 264, 266, 262], [243, 243, 245, 247, 247],
5, 0xFF0000FF);
```

aapolygon

```
int aapolygonColor( surface, vx, vy, n, color );
int aapolygonRGBA(  surface, vx, vy, n, r, g, b, a );
```

filled polygon

```
int filledPolygonColor( surface, vx, vy, n, color );
int filledPolygonRGBA(  surface, vx, vy, n, r, g, b, a );
```

textured polygon

```
int texturedPolygon( surface, vx, vy, n, texture, texture_dx, texture_dy );
```

bezier

```
int bezierColor( surface, vx, vy, n, s, color );
int bezierRGBA(  surface, vx, vy, n, s, r, g, b, a );
```

`n` is the number of elements in `vx` and `vy`, and `s` is the number of steps. So the bigger `s` is, the smother it becomes.

Example:

```
 bezierColor(display, [390, 392, 394, 396], [243, 255, 235, 247], 4, 20,
0xFF00FFFF);
```

character

```
int characterColor( surface, x, y, c, color );
int characterRGBA(  surface, x, y, c, r, g, b, a );
```

`c` is the character that will be drawn at `x`,`y`.

string

```
int stringColor( surface, x, y, c, color );
int stringRGBA(  surface, x, y, c, r, g, b, a );
```

set_font

```
void set_font(fontdata, cw, ch );
```