Első programunk: 01_HelloSDL

A project elérése:
http://cg.elte.hu/~bsc_cg/gyak/01/01_HelloSDL.zip

Az SDL

A Simple DirectMedia Layer (SDL) könyvtár egy crossplatform multimédiás könyvtár, ami alacsony szintű, hatékony hozzáférést ad

- audio,
- bemeneti (egér, billentyűzet, joystick),
- grafikus (OpenGL-en keresztül GPU-hoz pl.)

eszközökhöz. A gyakorlatokon az *SDL 2.0*-át fogjuk használni.

A használatához szükséges **header**:

```
#include <SDL.h>
```

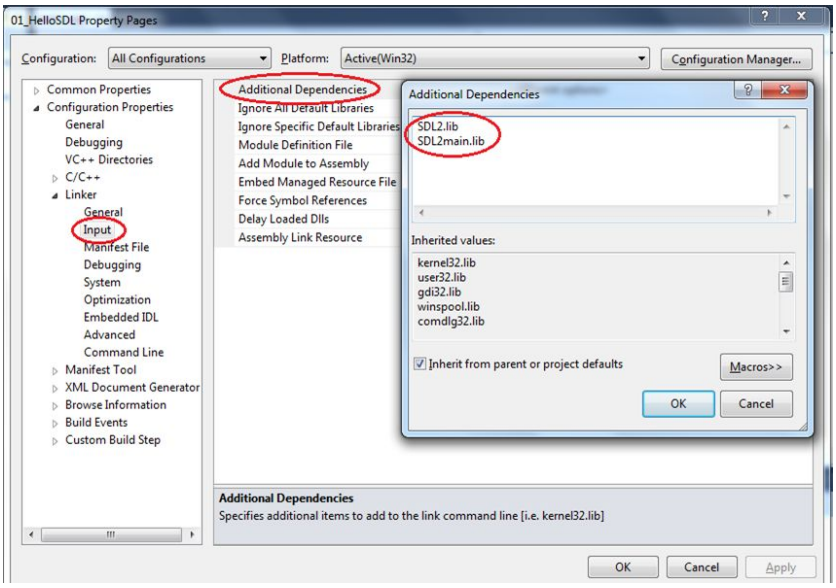
A programhoz linkelendő **lib**-ek:

- SDL2.lib
- SDL2main.lib

A fordításhoz szükséges lib-eket meg kell adni a Visual Studio-nak. Erre két lehetőség van:

- *Project properties-ben:*

  Nyissuk meg a *Solution Explorer*ben project nevére jobb gombbal kattintva lenyíló menüből a *Properties* opciót, majd az *All Configurations* kiválasztása után a bal oldalon a *Configuration Properties/ Linker/ Input* meüben az *Additional Depedencies*-ek közé vegyük fel a fent említett két *lib*-et:



- *Kódból:*

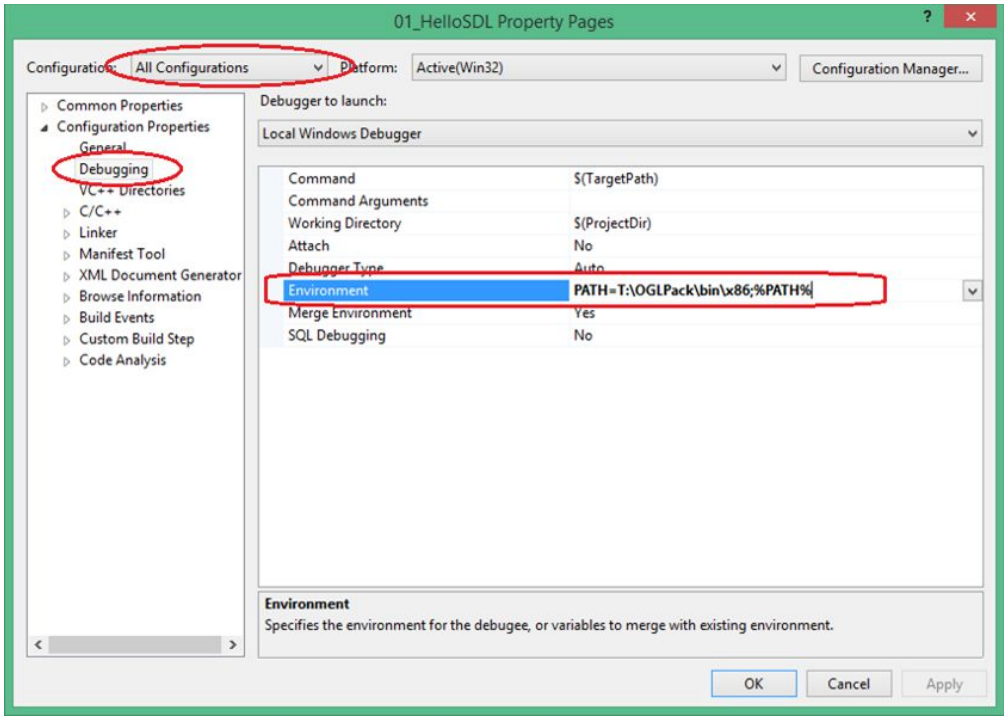
  A következőket írjuk be az #include-ok után:

```
#pragma comment(lib, "SDL2.lib")
#pragma comment(lib, "SDL2main.lib")
```

A programhoz megadandó **DLL** fájlok az OGLPack\bin mappán belül találhatóak:

- \x86: 32 bites DLL-ek - mi ezeket fogjuk használni
- \x64: 64 bites DLL-ek

A programunknak szüksége lesz arra, hogy lássa az SDL-es függvények implementációit. Ezek vannak a különböző DLL fájlokban. A szükséges DLL-eket olyan helyre kell másolni, ahol a rendszer látja (PATH környezeti változó által lefedett helyre, munkakönyvtárba).
Vagy amit már csináltunk (ld. a Visual Studio leírásánál az *Environment* környezeti változó beállításánál):



A program lépésről lépésre

A program felépítése

A project egyetlen forráskódot tartalmazó fájlja a main.cpp. Ennek tartalma:

- include-ok (SDL és iostream)
- `void exitProgram()` nevű függvény
- `int main( int argc, char* args[] )`, amit részletesen tárgyalunk a következőkben.

A main függvény kinézete

```
int main( int argc, char* args[] )
{
	atexit( exitProgram );
	[...]
	return 0;
}
```

Az `atexit()` parancs a paraméterben kapott függvénypointer által mutatott függvényt hívja meg az alkalmazás terminálásakor. Ez a függvény most az `exitProgram`, amit csak arra használunk, hogy a konzolablak eltűnése előtt megvárjunk egy billentyűleütést.

A main függvény működése a következő:

1. Inicializálja az SDL-t
2. Létrehoz egy ablakot
3. Létrehoz egy *Renderer*-t, azaz rajzolót, amit ablakhoz rendel
4. Rajzol
5. Megszünteti a Renderer-t
6. Megszünteti az ablakot és leállítja az SDL-t

Nézzük át a lépéseket alaposabban!

1. Az SDL inicializálása

```
if ( SDL_Init( SDL_INIT_VIDEO ) == -1 )
{
  std::cout << "[SDL indítása]Hiba az SDL inicializálása közben: "
    << SDL_GetError() << std::endl;
  return 1;
}
```

`int SDL_Init(Uint32 flags)`: Inicializálja a paraméterben kapott SDL alrendszereket. Ha többet akarunk, akkor bináris VAGY művelettel kapcsoljuk össze őket. Ezek az alrendszerek a következők lehetnek:

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	don't catch fatal signals

`SDL_GetError()`: Ezzel a függvénnyel az összes SDL-es utasítás által dobott hibának kideríthető az oka, ha a hiba keletkezése után meghívjuk, és kiiratjuk.

2. Ablak létrehozása

```
SDL_Window *win = 0;
win = SDL_CreateWindow( "Hello SDL!", 100, 100, 640, 480, SDL_WINDOW_SHOWN);

if (win == 0)
{
  std::cout << "[Ablak létrehozás]Hiba az SDL inicializálása közben: "
    << SDL_GetError() << std::endl;
  return 1;
}
```

`SDL_Window` a következőképpen van deklarálva:

```
typedef struct SDL_Window SDL_Window;
```

Egy SDL-es ablak-azonosító. Ez a deklaráció azért szerepel így, hogy az ablak attribútumait ne a pointeren keresztül próbáld lekérni vagy módosítani. Erre külön függvények vannak (például `SDL_SetWindowTitle` és társai).

`SDL_CreateWindow`: Létrehoz egy w széles, h magas ablakot a title fejléccel, az (x,y) képernyőkoordinátákon, flags (`SDL_WindowFlags`) megjelenítési attribútumokkal, és siker esetén visszaadja az azonosítóját (különben `nullptr`).

```
SDL_Window* SDL_CreateWindow(const char* title,
                             int         x,
                             int         y,
                             int         w,
                             int         h,
                             Uint32      flags)
```

`Uint32 flags - SDL_WindowFlags`: Megjelenítési tulajdonságok, szintén binárisan VAGY-olandóak:

SDL_WINDOW_FULLSCREEN	fullscreen window
SDL_WINDOW_FULLSCREEN_DESKTOP	fullscreen window at the current desktop resolution
SDL_WINDOW_OPENGL	window usable with OpenGL context
SDL_WINDOW_SHOWN	window is visible
SDL_WINDOW_HIDDEN	window is not visible
SDL_WINDOW_BORDERLESS	no window decoration
SDL_WINDOW_RESIZABLE	window can be resized
SDL_WINDOW_MINIMIZED	window is minimized
SDL_WINDOW_MAXIMIZED	window is maximized
SDL_WINDOW_INPUT_GRABBED	window has grabbed input focus
SDL_WINDOW_INPUT_FOCUS	window has input focus
SDL_WINDOW_MOUSE_FOCUS	window has mouse focus
SDL_WINDOW_FOREIGN	window not created by SDL
SDL_WINDOW_ALLOW_HIGHDPI	window should be created in high-DPI mode if supported (>= SDL 2.0.1)

3. Renderer létrehozása

```
SDL_Renderer *ren = 0;
ren = SDL_CreateRenderer(
            win,
            -1,
            SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);

if (ren == 0)
{
  std::cout << "[Renderer létrehozása]Hiba az SDL inicializálása közben: "
    << SDL_GetError() << std::endl;
  return 1;
}
```

SDL_CreateRenderer: *2D-s renderer context* létrehozására használandó, azaz lényegében lehetővé teszi, hogy tudjunk rajzolni az ablakunkra. (Képzelheted úgy, mintha az ablak lenne a műterem, a renderelő pedig a festőállvány a vászonnal. Kevésbé kreatívak pedig gondolhatnak úgy rá, mint egy festő-mázoló szakemberre, az ablakra pedig úgy, mint az általa lefestendő falfelületre) Ha sikertelen a renderelő létrehozása, akkor NULL pointerrel tér vissza.

```
SDL_Renderer* SDL_CreateRenderer(
        SDL_Window* window,
        int index,
        Uint32 flags)
```

SDL_Window* window: Az ablak azonosítója, amire rajzolni szeretnénk a létrehozandó renderer segítségével.

int index: A renderer indexe. Vagy -1, ha a kért képességeket (hardveresen gyorsított rajzolás stb., lásd flags) támogató első renderer megfelel a céljainknak.

Uint32 flags - SDL_RendererFlags: Mi hardveresen gyorsítottat szeretnénk. A többi lehetőség:

SDL_RENDERER_SOFTWARE	the renderer is a software fallback
SDL_RENDERER_ACCELERATED	the renderer uses hardware acceleration
SDL_RENDERER_PRESENTVSYNC	present is synchronized with the refresh rate
SDL_RENDERER_TARGETTEXTURE	the renderer supports rendering to texture

4. Rajzolás

```
SDL_SetRenderDrawColor(ren, 0, 0, 0, 255);
SDL_RenderClear(ren);
SDL_SetRenderDrawColor(ren, 0, 255, 0, 255);
SDL_RenderDrawLine(ren, 10, 10, 10, 60);
SDL_RenderPresent(ren);
SDL_Delay(2000);
```

SDL_SetRenderDrawColor: Beállítja a *renderer* színét az *(r,g,b)* színre, *az* átlátszósággal. Minden egyes színkomponens és az átlátszóság erőssége 0-255 közötti egész szám. (A festőtermes analógia mentén ez határozza meg a az ecset színét.)

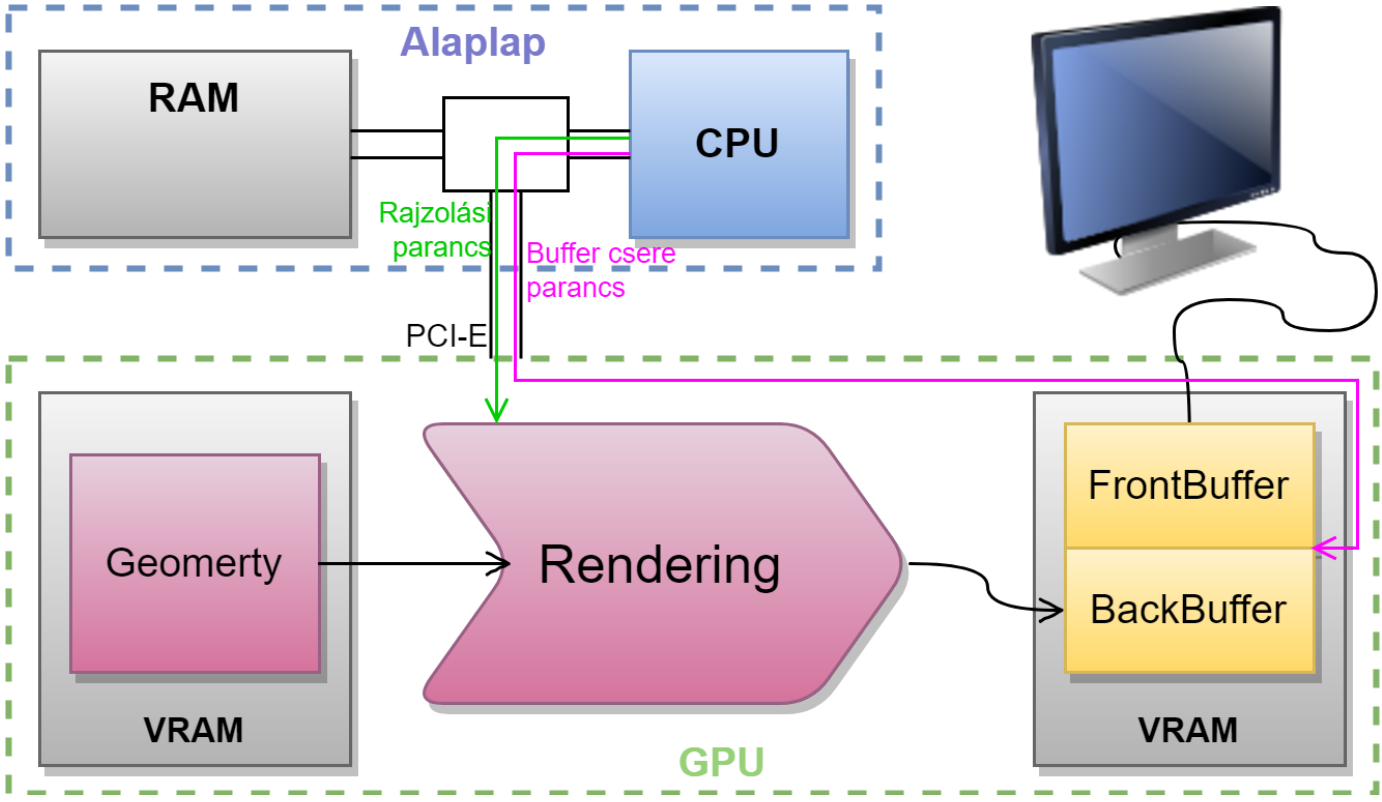
```
int SDL_SetRenderDrawColor( SDL_Renderer* renderer,
                            Uint8 r, Uint8 g, Uint8 b, Uint8 a );
```

SDL_RenderClear: Az aktuális rajzolási színnel „törli" az ablak rajzterületét. (Veszünk egy tiszta, megfelelő színű vásznat.) Sikertelen törlés esetén negatív számmal tér vissza.

```
int SDL_RenderClear( SDL_Renderer* renderer );
```

SDL_RenderDrawLine: Ez is, mint a többi SDL rajzoló utasítás, negatív számmal tér vissza, ha sikertelen. Egyébként a *renderer*-rel (az ő ablakára) rajzol egy egyenes szakaszt, ami az *(x1,y1)* és *(x2,y2)* pontokat köti össze. Egyelőre ezt csak egy képernyőn kívüli memóriaterületre rajzoljuk (piszkozatot készítünk a festményünkről az új vásznon).

```
int SDL_RenderDrawLine( SDL_Renderer* renderer,
                        int x1, int y1, int x2, int y2 );
```



SDL_RenderPresent: Alapesetben minden rajzolást egy képernyőn kívüli memóriaterületre végzünk. Ezzel az utasítással jelezzük, hogy végeztünk az aktuális képkocka, vagyis *frame* kirajzolásával és most már ezt kell megjeleníteni. (Fogjuk magunkat, és a piszkozatvásznat feltesszük az állványra ezzel kinyilvánítva, hogy azt késznek tekintjük.)

```
void SDL_RenderPresent( SDL_Renderer* renderer );
```

SDL_Delay: A paraméterben megadott mennyiségű milliszekundumot várakozik a program futása, hogy megcsodálhassuk a művünket.

```
void SDL_Delay(Uint32 ms);
```

5. Renderer megszüntetése

```
SDL_DestroyRenderer( ren );
```

A fenti parancs értelemszerűen megszünteti a renderelőt. (Addig ugrálunk a vásznon és az állványon, amíg az szilánkosra nem törik. Környezetbarátoknak ez higgadt sétának felel meg a szelektív gyűjtőhöz.)

6. Ablak megszüntetése, SDL leállítása

```
SDL_DestroyWindow( win );
SDL_Quit();
```

Az első utasítás megszünteti a paraméterként feltüntetett ablakot, (azaz eladjuk a műtermet,) majd a második leállítja az SDL-t (új foglalkozás után nézünk).