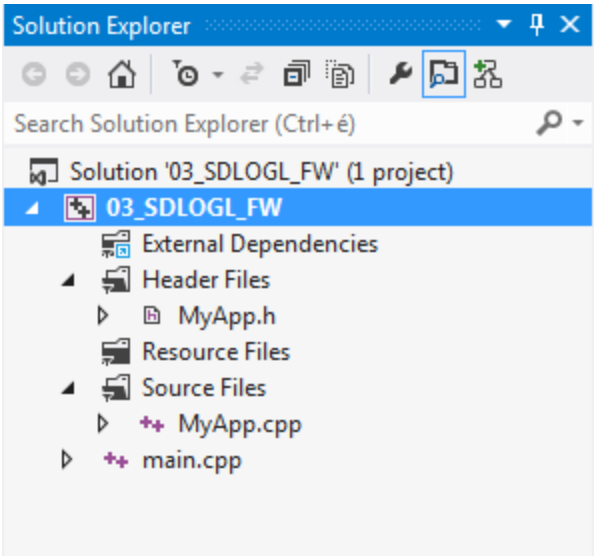
2. Gyakorlat - 03_SDLOGL_FW

Forrás: http://cg.elte.hu/~bsc_cg/gyak/02/03_SDLOGL_FW.ZIP

Ezen a ponton megismerkedünk a keretrendszerrel, amit a továbbiakban használni fogunk. Ahogy a programunk egyre összetettebbé válik, érdemes külön fordítási egységekbe tagolni. A program immár három forrásfájlból áll:

- main.cpp
- MyApp.cpp
- MyApp.h

Ezeket a Solution Explorer-ben (View menü, Solution Explorer) is láthatjuk:



A **main.cpp** továbbra is azt a logikát tartalmazza, ami inicializálja az OpenGL-t és megteremti az alkalmazáshoz szükséges környezetet. . (Létrehozza az OpenGL *context*-et, alapesetben a lehető legmagasabb verziószámmal. Beállítja a színek kezeléshez és a bufferelésehez szükséges adatokat, valamint a megjelenítő ablak adatait. Létrehozza a context-et, inicializálja a Glew-t, lekérdezi az inicializált OpenGL verzióját. Az alkalmazás lefutása után megfelelően felszabadítja az erőforrásokat.)

Az alkalmazás tényleges logikáját az átláthatóság kedvéért kiszervezzük egy külön **`CMyApp`** osztályba, melyet a **`MyApp.h`** és **`MyApp.cpp`** fájlokban deklarálunk/definiálunk. Az alkalmazásunkat képviselő egyetlen `CMyApp app` példányt a main.cpp-ben hozzuk létre. Az eseménykezelő ciklus ennek az osztálynak a megfelelő metódusait fogja meghívni különböző események esetén.

```
// INIT
bool quit = false;
SDL_Event ev;
CMyApp app;
```

```
if (!app.Init()) {
	SDL_DestroyWindow(win);
	std::cout << "[app.Init] Az alkalmazás inicializálása közben
		hiba történt!" << std::endl;
	return 1;
}
// RUN
while (!quit) {
	while ( SDL_PollEvent(&ev) ) {
		switch (ev.type) {
		case SDL_QUIT:
			quit = true;
			break;
		case SDL_KEYDOWN:
			if ( ev.key.keysym.sym == SDLK_ESCAPE )
				quit = true;
			app.KeyboardDown(ev.key);
			break;
		case SDL_KEYUP:
			app.KeyboardUp(ev.key);
			break;
		case SDL_MOUSEBUTTONDOWN:
			app.MouseDown(ev.button);
			break;
		case SDL_MOUSEBUTTONUP:
			app.MouseUp(ev.button);
			break;
		case SDL_MOUSEWHEEL:
			app.MouseWheel(ev.wheel);
			break;
		case SDL_MOUSEMOTION:
			app.MouseMove(ev.motion);
			break;
		case SDL_WINDOWEVENT:
			if (ev.window.event == SDL_WINDOWEVENT_SIZE_CHANGED)
				app.Resize(ev.window.data1, ev.window.data2);
			break;
		}
	}

	app.Update();
	app.Render();
	SDL_GL_SwapWindow(win);
```

```
}

// DESTROY
app.Clean();
SDL_GL_DeleteContext(context);
SDL_DestroyWindow( win );
```

Most pedig vegyük sorra, hogy mik ezek a függvények, mikor hívja meg őket a main függvény, és mire valók! Ehhez a **MyApp.h**-ra kell vetnünk egy pillantást. (A függvényeket később a **MyApp.cpp**-ben fogjuk megírni.)

Elsőként include-ok szerepelnek a kódban:

```
#pragma once
#include <GL/glew.h>
#include <SDL.h>
#include <SDL_opengl.h>
```

Ezután vezetjük be a **CMyApp** osztályunkat:

```
class CMyApp{
public:
	CMyApp(void);
	~CMyApp(void);

	bool Init();
	void Clean();

	void Update();
	void Render();

	void KeyboardDown(SDL_KeyboardEvent&);
	void KeyboardUp(SDL_KeyboardEvent&);
	void MouseMove(SDL_MouseMotionEvent&);
	void MouseDown(SDL_MouseButtonEvent&);
	void MouseUp(SDL_MouseButtonEvent&);
	void MouseWheel(SDL_MouseWheelEvent&);
	void Resize(int, int);
};
```

A CMyApp osztály függvényei három kategóriába sorolhatóak: élettartam-vezérlő, megjelenítési, ill. eseménykezelő függvények. Ezek jól összeegyeztethetők a main.cpp-ben struktúrájával:

A main függvény áll egy inicializálási (INIT), egy ciklikus futási (RUN) és egy megsemmisítési (DESTROY) részből. A ciklus futása végtelennek tekinthető, egészen amíg ki nem lépünk. Minden függvény, ami a ciklusban hívódik (megjelenítés és eseménykezelők), potenciálisan **nagyon sokszor** le fog futni, míg a cikluson kívül hívottak (élettartam-vezérlők) csak pontosan **egyszer**, ezért nagyon fontos ezek elkülönítése!

C++ objektum élettartama

Konstruktor, Destruktor

Jelenleg a CMyApp konstruktora és destruktora az objektum létrehozásán / megsemmisítésén kívül semmit sem csinál, és ez most jól is van így. A CMyApp objektum adattagjait majd manuálisan (az Init és Clean metódusokban) inicializáljuk és töröljük; később látni is fogjuk, hogy miért.

Alkalmazás élettartama

(INIT)

Init()

```
bool CMyApp::Init()
{
	glClearColor(0.125f, 0.25f, 0.5f, 1.0f);
	glEnable(GL_CULL_FACE);
	glEnable(GL_DEPTH_TEST);

	return true;
}
```

Ez a függvény csak **egyszer** fut le, miután már inicializáltuk az SDL-t és a Glew-t, létrehoztuk az ablakot és az OpenGL context-et, és minden környezet készen áll a rajzolási feladatok megkezdéséhez. Most az Init() függvényben az alábbiakat végezzük el:

`glClearColor`: Beállítjuk a törlési háttérszínt, jelen esetben kékes árnyalatot. R - red, G - green, B - blue, A - alpha (azaz átlátszóság), mind 0 és 1 közötti értékekkel. (Az f a számok végén float típusú konstansra utal.)

glEnable: Párja a **glDisable**. Ezekkel lehet OpenGL-beli “kapcsolókat” (flag-eket) állítani. Jelen esetben bekapcsoljuk a hátrafelé néző lapok eldobását (`GL_CULL_FACE`) és a mélységi tesztet (`GL_DEPTH_TEST`). (Ezek értelméről később lesz szó.)
A későbbi példakódokban az *Init()* függvény fog szolgálni nagy méretű vektorok, mátrixok, képek létrehozására, hogy azokat csak egyszer készítsük el, gyorsítva ezzel a futás sebességét.
A visszatérési értékben jelezzük, hogy sikerrel zárult-e az objektum egyszeri inicializálása. (Ez ebben a konkrét példa projektben mindig igaz.)

(DESTROY)

Clean()

Az *Init()* párja. Mivel még az *Init()*-ben nem hoztunk létre nagy objektumokat, így itt egyelőre semmi dolgunk nincs. Később azonban ez a metódus felel majd az Init()-ben létrejött objektumok/erőforrások felszabadításáért. A *Clean()* függvényt még a context és az ablak megsemmisítése előtt meghívjuk, hogy a speciális objektumok kiürítése akadálymentesen megtörténhessen a jövőben.

(RUN)

Ismételten érdemes felhívni a figyelmet arra, hogy az alkalmazás megjelenítését és eseménykezelését végző függvények nagyon sokszor le fognak futni (konkrétan minden egyes iterációban), így amit csak lehet, inkább rajtuk kívül, még az *Init()*-ben inicializáljunk. Természetesen az időben változó dolgokat nem elég egyszer megadni, azokat a következő függvények valamelyikében változtassuk.

Megjelenítés

Update()

Az *Update()* függvényben fogjuk a bonyolultabb számítási feladatokat elvégezni, mint pl. megváltoztatni egy tárgy aktuális pozícióját, helyzetét stb. Mivel objektumunk ebben a példában még nincs, így ez a függvény jelenleg üres, de ez persze a későbbi példákban már nem így lesz. A kísérletező kedvűek esetleg már most kipróbálhatják, hogyan lehet az Update függvényben pl. időtől függően megváltoztatni a háttér színét.

Render()

```
void CMyApp::Render()
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}
```

Ez a függvény a mi alkalmazás-struktúránkban mindig az *Update()* után hívódik meg egyszer. (Sok más alkalmazásban a kettő hívása jobban elkülönül, ilyenkor az *Update()* sokkal

gyakoribb, míg a *Render()* csak adott sokszor fut le másodpercenként - ennek a sebességét nevezzük *fps*-nek, azaz *frames per second*-nek.)
A *Render()* felelős a képkocka elkészítéséért, az objektumok kirajzolásáért. Minden futása egy képkocka elkészítését jelenti, aminek a tényleges képernyőn való megjelenítése a függvény lefutása után meghívott `SDL_GL_SwapWindow(win)` parancs hatására történik. Ebben a konkrét példában egy képkocka kirajzolása az alábbi módon történik:
`glClear()`: Előkészíti az új képkockát azáltal, hogy törli a frampuffert (GL_COLOR_BUFFER_BIT) és a mélységi Z puffert (GL_DEPTH_BUFFER_BIT). A törlendő puffereket bitenkénti vaggyal (|) is össze tudjuk kapcsolni, így egy utasítással lefut mindkettő törlése.

Eseménykezelő függvények

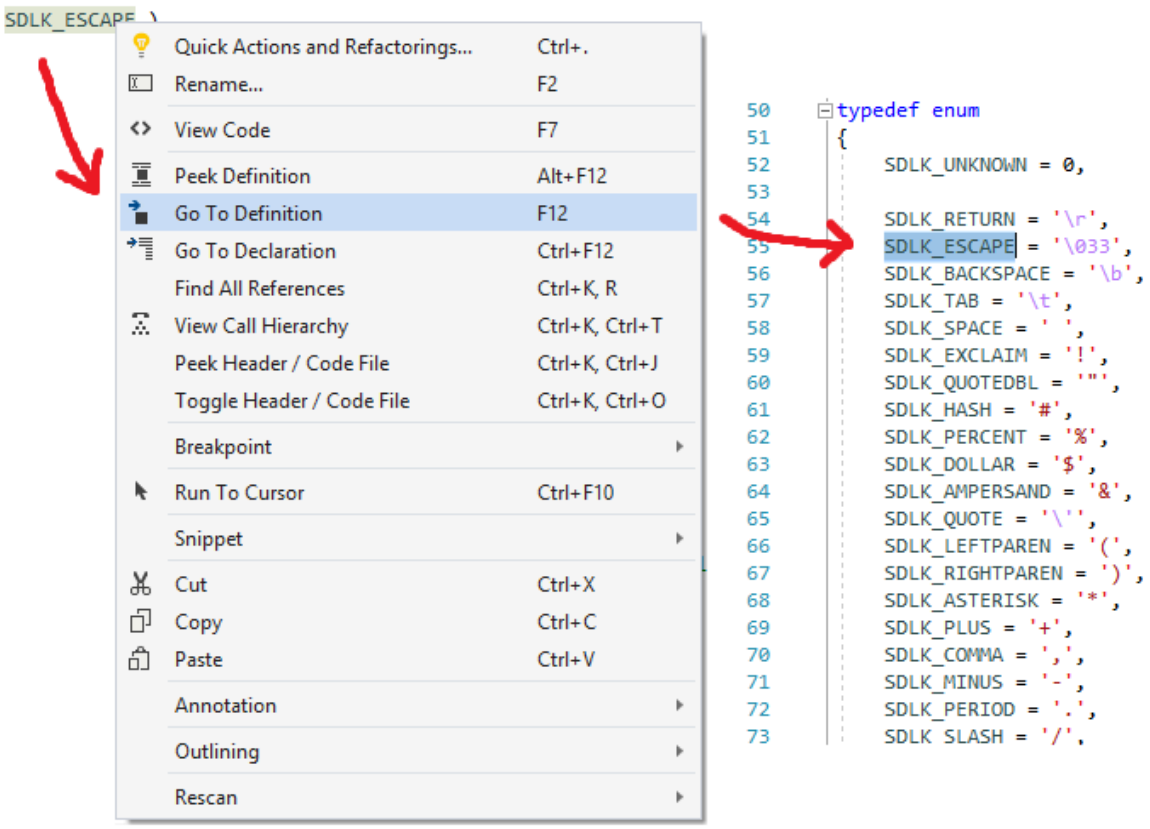
Ezek a függvények a különböző, számunkra is releváns eseményeket kezelik le, amiket az esemény-sorból (event queue) nyerünk ki minden ciklusban. Mivel egy eseményeket kezelő alkalmazásban általában számítunk rá, hogy események történjenek, így ezek is nagyon sokszor lefuthatnak. Ezek segítségével lehet például a felhasználó által kívánt módon mozgatni tárgyakat, mozogni a kamerával, változtatni beállításokat stb. A későbbiekben ezekre részletesebben is ki fogunk térni, amikor majd az interakcióval foglalkozunk behatóbban.

Resize(int, int)

```
void CMyApp::Resize(int _w, int _h)
{
	glViewport(0, 0, _w, _h );
}
```

A főablak méretének megváltozása esetén hívódik. A **glViewport** függvény beállítja, hogy az ablak melyik részére rajzolunk – mi ezt a teljes elérhető területre állítjuk. Az első két paraméter (x és y) mondja meg a rajzterület bal felső sarkának pozícióját (alapesetben 0,0), a második két paraméter (width = _w, height = _h) pedig a szélességet és magasságot. Ezeket az ablak átméretezésekor paraméterként megkapjuk, így hozzá igazíthatjuk a rajzterületet.

KeyboardDown(SDL_KeyboardEvent& key), KeyboardUp(SDL_KeyboardEvent& key)
Adott billentyű lenyomására illetve felengedésére reagáló függvények. Paraméterként kapják a billentyű kódját (`key.keysym.sym`). Ez lehet egyszerű karakter (`'a'`, `'w'`, `'s'`, `'d'`, …), de speciális billentyű is (`SDLK_ESCAPE`, `SDLK_SPACE`, `SDLK_BACKSPACE`, ...).
Ha tudni szeretnéd, hogy hogyan kell egy betű SDL-kódját leírni, a Visual Studioban egy jobb egér nyomással pl. a 'Go To Definition' menüponttal megnézheted a header file-okat, ahol könnyen megtalálhatóak ezek az adatok.


SDLK_ESCAPE
Quick Actions and Refactorings... Ctrl+.
Rename... F2
View Code F7
Peek Definition Alt+F12
Go To Definition F12
Go To Declaration Ctrl+F12
Find All References Ctrl+K, R
View Call Hierarchy Ctrl+K, Ctrl+T
Peek Header / Code File Ctrl+K, Ctrl+J
Toggle Header / Code File Ctrl+K, Ctrl+O
Breakpoint
Run To Cursor Ctrl+F10
Snippet
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Annotation
Outlining
Rescan
50 typedef enum
51 {
52 SDLK_UNKNOWN = 0,
53
54 SDLK_RETURN = '\r',
55 SDLK_ESCAPE = '\033',
56 SDLK_BACKSPACE = '\b',
57 SDLK_TAB = '\t',
58 SDLK_SPACE = ' ',
59 SDLK_EXCLAIM = '!',
60 SDLK_QUOTEDBL = '"',
61 SDLK_HASH = '#',
62 SDLK_PERCENT = '%',
63 SDLK_DOLLAR = '$',
64 SDLK_AMPERSAND = '&',
65 SDLK_QUOTE = '\'',
66 SDLK_LEFTPAREN = '(',
67 SDLK_RIGHTPAREN = ')',
68 SDLK_ASTERISK = '*',
69 SDLK_PLUS = '+',
70 SDLK_COMMA = ',',
71 SDLK_MINUS = '-',
72 SDLK_PERIOD = '.',
73 SDLK_SLASH = '/',

`MouseDown(SDL_MouseButtonEvent&), MouseUp(SDL_MouseButtonEvent&)`
Hasonlóan a billentyűzet eseményeihez, itt az egérgombok megnyomása, illetve felengedése hatására hívódik meg a függvény. Azt, hogy melyik gombról van szó, itt is paraméterként kapjuk meg.

bal egérgomb	`SDL_BUTTON_LEFT`
középső egérgomb	`SDL_BUTTON_MIDDLE`
jobb egérgomb	`SDL_BUTTON_RIGHT`

`MouseWheel(SDL_MouseWheelEvent&)`
Az egérgörgő tekerének hatására hívódik meg; a paraméterben az esemény típusával.

`MouseMove(SDL_MouseMotionEvent&)`
Az egérmutató főablak feletti mozgatásakor hívódik, a pozíció lekérdezésével a kurzor mozgásához köthetünk eseményeket.