

0. DGE SDK 2.0

1. Overview

DGE SDK 목적: CAANOO 게임 기기는 Linux를 기반으로 동영상, 플래시, 게임 등을 구현 할 수 있는 소형 mobile 기기로 DGE SDK는 이 기기를 배경으로 한 게임 제작에서 편리성을 도모하기 위해서 만들어졌습니다.

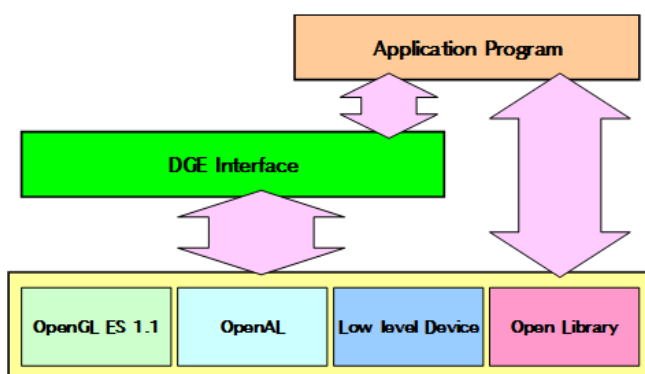
DGE는 먼저 크로스 컴파일 지원을 위해 에뮬레이터용 마이크로 소프트의 윈도우용 라이브러리와 게임을 기기에서 실행 하기 위한 Linux용 라이브러리가 있습니다. 윈도우용 에뮬레이터에 대한 라이브러리는 아날로그 조이스틱과 같은 특별한 장치에 대해서는 일부만 지원됩니다.

SDK는 크게 Base, Auxiliary, Utility 세 부분으로 구성되어 있습니다.

Base는 게임 프로그램을 구동시키기 위한 기본 구조로 그 자체만으로도 게임을 만들 수 있도록 구성되어 있습니다. Base는 프로그램의 생성, 비디오 디바이스와 그래픽, 수학 함수, 저 수준 인풋 등과 크로스 컴파일에서 지원되지 않는 c-run time 함수들이 포함되어 되어 있으며 그래픽에는 2D, 3D 게임 제작에 필요한 Sprite와 비디오 장치 interface가 있습니다. 게임 프로그램 작성에서 제일 먼저 해야 할 일은 Base에 포함되어 있는 함수들을 호출해야 올바르게 실행이 됩니다.

DGX(DGE Auxiliary)는 Base안의 저 수준 함수들을 클래스로 구현한 상부 구조와 화면에 문자열을 출력하는 폰트 인터페이스, 그리고 다른 기기와 통신이 가능하도록 한 네트워크 함수들로 구성되어 있습니다. 폰트와 같은 Auxiliary 일부 객체는 화면 출력이 필요하기 때문에 DGE Base를 필요로 합니다. 또한 DGX에는 카메라, 메쉬 등의 3D 자료 등도 포함되어 있어서 DGE, DGX 두 부분만으로도 CAANOO용 게임을 만들 수 있습니다.

DGT(DGE Utility)는 3D 자료와 게임 Resource의 압축 등 Base와 Auxiliary에 포함되지 않은 각종 응용 프로그램을 모아놓은 곳입니다.



< DGE Layer>

GPH SDK는 OpenGL ES 1.1, OpenAL 등과 같이 Mobile 등의 기기에서 표준으로 안착되었거나 신뢰성 높은 공개 소프트웨어를 기반으로 게임 제작에 필요한 모듈들을 모아서 구성해 놓은 라이브러리입니다. 만약 여러분이 프로그램을 작성할 때 DGE에 포함되어 있지 않은 내용이라도 GPH SDK에서 원하는 내용

을 찾아서 구현할 수가 있습니다. 참고로 GPH SDK는 embedded LINUX 운영 체제에 관련된 라이브러리와 함수들이 제공되고 있어서 프로세서 관리, 메모리 관리, 각종 입/출력 장치를 제어할 수 있고, OpenGL ES, SDL 등의 공개용 라이브러리 등이 포함되어 있어서 독자적인 라이브러리를 만들 수 있습니다.

2. 함수의 반환 값과 주요 데이터 타입

1. 자원을 할당하거나 값을 읽는 DGE SDK의 모든 함수는 처리에 대한 결과를 반환 합니다. 반환 값은 성공의 경우 0보다 크거나 같은 값을 가지며 실패할 경우 음수 값을 반환합니다. 성공과 실패에 대한 반환의 값은 "DGE_Type.h" 파일에 다음과 같이 정의되어 있습니다.

```
#define DGE_DEFAULT                ((UINT) -1)
#define DGE_OK                     0x00000000
#define DGE_EFAIL                  -1
#define DGE_EINVALIDATE_CALL       0xF0000001
#define DGE_ECANNOTALLOCATED       0xF0000002
#define DGE_ECANNOTFIND_RESOURCE   0xF0000003
#define DGE_ENOTIMPLEMENTED        0xF0000004
#define DGE_ECANNOTACQUIRE_DEVICE 0xF0000005
#define DGE_INVALID_DEVICE         -1
```

DGE_OK 반환 값은 해당 함수의 호출은 완벽하며 추가적인 어떤 작업도 필요 없음을 의미하고, DGE_EFAIL 반환은 실패에 대한 Default 값으로 실패의 원인이 분명하지 않거나 실패의 내용을 해석할 특별한 일이 없을 때 주로 사용합니다. 데이터 타입은 32비트 시스템에 맞추어져 있으며 특별히 자주 사용되는 자료를 구조체를 다음과 같이 선언해 놓았습니다.

DGE_POINT[n], DGE_INT[n]: 32비트 정수형 n차원 구조체

DGE_SHORT[n]: 16비트 정수형 n차원 구조체

DGE_FLOAT[n]: 단정도 실수형 n차원 구조체

DGE_COLORB: 32비트 색상 구조체

DGE_COLORF: 32비트 색상을 float 형으로 처리하기 위한 구조체

DGE_MATRIX: float 형 4x4행렬

게임 프로그램에서는 수학 연산을 편리하게 사용하기 위해 단순히 구조체의 멤버 변수뿐만 아니라 멤버 함수, 그리고 연산자 재정의(Operator Overriding) 함수도 같이 정의하는 경우가 많습니다. "DGE_Math.h" 파일에는 벡터(Vector), 행렬(Matrix), 사원수(Quaternion), 색상(Color), 등의 구조체와 수학 연산에 대한 멤버 함수 등이 선언되어 있습니다.

MS Visual C++ 개발 환경과 ARM Compiler의 크로스 컴파일에서 구조체를 사용할 경우 여러분은 메모리 Packing 또는 Padding에 대해서 아주 많은 주의를 기울려야 합니다. 만약 여러분이 선언한 구조체 자료형의 멤버가 모두 32비트이면 문제가 없지만 16비트 또는 8비트 변수들이 포함되어 있는 경우 Visual C++과 ARM Compiler는 다른 모습을 보이고 있습니다. 마이크로소프트의 Visual C++ Compiler는 #pragma pack(...)과 같은 Directive로 메모리 크기를 packing 할 수 있지만 "#pragma" 는 마이크로소프트 플랫폼에서만 유효합니다. WIZ, CAANOO등의 기기에 적용하려면 "__attribute__" 키워드를 사용해서 메모리 크기를 packing 합니다.

2. DGE Base

- 라이브러리: CAANOO - libdge20.so, WIZ - libdge20w.so, Windows - dge20_.lib
- System과 Graphic으로 구성

2.1 Basic 구성 요소

2.1.1 System

- DGE Type: 기본적인 자료형 정의 → DGE_Type.h
- DGE System: SDK 버전, 빌드 날짜, 하드웨어를 사용하기 위해서 프로그램을 생성하고 초기화를 담당. → DGE_System.h
- DGE Input: Analog Joy-stick, Touch Screen, Button의 사용과 DGE 1.0의 8-방향 디지털 방식의 joy-stick에 대한 가상 매핑이 포함 → DGE_Input20.h
- DGE Time: 날짜, 시간, 타이머 → DGE_Time.h
- DGE Util: 플랫폼의 호환이 되지 않는 C-Rum Time 관련 함수를 재구성 → DGE_Util.h
- DGE Gsensor: 중력 센서 → DGE_Gsensor.h
- DGE Haptic: 진동 모터 → DGE_Haptic.h
- DGE Math: 벡터, 행렬, 색상 등의 구조체와 처리 함수를 구현 → DGE_Math.h

2.1.2 Graphic

- IDGE_Device: OpenGL의 그래픽 파이프라인을 객체화. → DGE_Base.h
- IDGE_Sprite: 2D 그래픽 처리 객체 → DGE_Base.h
- IDGE_Image: 이미지 파일을 해석하고 픽셀을 메모리에 적재 → DGE_Base.h
- IDGE_Texture: 이미지 파일, 메모리 버퍼의 픽셀을 가지고 텍스처 생성 → DGE_Base.h
- IDGE_VB: Vertex Buffer Object 객체 → DGE_Base.h
- IDGE_IB: Index Buffer Object 객체 → DGE_Base.h

2.2 함수

2.2.1 System

DGE_Version: SDK 버전 정보와 빌드 날짜 반환

입력 인수:

- DWORD* dVer: Version 정보 Major(Hi-word), Minor(Low-word)
- DWORD* dDate: Built-date: Hex를 문자열로 출력

DGE_Init: 프로그램을 실행하기 위해서 시스템을 초기화하며 게임 프로그램 시작에서 main() 함수 다음 최초로 호출

DGE_Close: 프로그램을 실행하기 위해서 얻은 자원을 반납 프로그램 종료 직전 호출

- 이 함수를 호출해야 메뉴로 돌아감

DGE_CreateWindow: 게임 화면 생성. 성공하면 윈도우 핸들 반환. 실패 시 NULL 반환

입력 인수:

- pDGE_MsgProc: 윈도우 메시지 프로시저 함수. CAANOO에서는 NULL 전달
- char* ClassName: 프로그램 클래스 이름
- INT ScreenW: 화면 너비: 0 값을 입력 하면 CAANOO:320, windows: 640 화면 너비를 만듦
- INT ScreenH: 화면 높이: 0 값이 입력 되면 CAANOO:240, windows: 480 화면 높이를 생성

DGE_DestroyWindow: 윈도우를 해제하고 자원을 반납

입력 인수: DGE_HANDLE: 해제할 윈도우 핸들

DGE_SetMainRunFunc: 윈도우 프로그램용 메시지 프로시저 설정 함수. CAANOO(x), 윈도우(0)

입력 인수: 메시지 처리 CALLBACK 함수 포인터

DGE_Run: 메인 처리 함수. 윈도우 애플레이터의 경우 메시지를 얻어와서 DGE_SetMain-

RunFunc 함수로 설정한 메시지 프로시저 함수로 운영체제로부터 받은 메시지 Dispatch

반환 값: 프로그램 종료 상황이 되면 음수 값 반환.

DGE_WindowSize: 설정된 윈도우 크기를 가져옴

입력 값: INT* pWidth: 윈도우 너비 쓸 주소 INT* pHeight: 윈도우 높이 쓸 주소

<프로그램 생성>

```

#ifdef _WIN32
    #pragma comment(lib, "dge20_.lib")
    #include <windows.h>
#endif

#include <stdio.h>
#include <limits.h>
#include <DGE_Type.h>
#include <DGE_System.h>
#include <DGE_Time.h>
#include <DGE_Math.h>

// Window Message Procedure
// 윈도우 애플레이터는 메시지 프로시저 함수가 반드시 필요합니다.
LRESULT DGE_WINAPI WndProc(DGE_HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
#ifdef _WIN32
    switch (uMsg)
    {
        ...

    }

    return DefWindowProc(hWnd, uMsg, wParam, lParam);
#else
    // DGE는 아무 처리도 합니다. 단지 윈도우와 형태를 맞추기만 합니다.
    return DGE_OK;
#endif
}

INT DGE_CALLBACK Run(void*)
{
#ifdef _WIN32
    static MSG msg = {0};

    if(msg.message==WM_QUIT)
        return -1;

    if( PeekMessage( &msg, NULL, 0U, 0U, PM_REMOVE ) )

```

```

        {
            ::TranslateMessage( &msg );
            ::DispatchMessage( &msg );
        }
    #else

    #endif

    // Main Proc
    ...
    return 0;
}

// Main Program
int main(int argc, char* argv[])
{
    DWORD    dVersion=0;
    DWORD    dDate=0;
    // DGE 초기화
    if(DGE_FAILED(DGE_Init()))
        return -1;

    // DGE 버전 확인
    DGE_Version(&dVersion, &dDate);
    printf("Version: %x Date: %x\n", dVersion, dDate);

    // Window 생성
    DGE_HANDLE hWnd = DGE_CreateWindow(WndProc, "DGE_Tutorial", 320, 240);
    if(NULL == hWnd)
        goto End;

    DGE_SetMainRunFunc(Run);          // Main Loop 함수 설정

    while(1)                          // 프로그램 실행
    {
        if(DGE_FAILED(DGE_Run()))
            break;
    }
}

```

```

        DGE_DestroyWindow(hWnd);          // Window 해제
End:
        DGE_Close();                      // DGE 해제(Menu로 돌아감)
        return 0;
}

```

2.2.2 Input20

Analog Joy-stick, Touch Screen, Button의 사용과 DGE 1.0의 8-방향 디지털 방식의 joy-stick에 대한 가상 매핑이 포함. 버튼, 8방향 디지털에 대한 Read 값은 1 또는 0. EDGE_INPUT은 인풋처리를 응용 프로그램에서 사용하기 위해 정의.

DGE_Input[Open|Close]: Input 장치 사용을 위한 초기화/ 해제

입력 인수: HWND: 윈도우 Emulating에서만 필요.

반환 값: 성공: DGE_OK, 실패: (-) 값

```

        hWnd = CreateWindow(...);
...
        hr = DGE_InputOpen(hWnd);
        if(DGE_FAILED(hr))
            return DGE_ECANNOTACQUIRE_DEVICE;
...
        DGE_InputClose();
...
        DGE_DestroyWindow(hWnd);

```

DGE_InputReadAjs: Analog Joystick 장치의 x, y 값 읽기

입력 인수: DGE_INPUT_AJS 구조체 변수 주소

반환 값: 함수 처리 성공 DGE_OK, 처리 실패 (-) 값이며 대부분 실패의 원인은 장치에서 값을 읽어오는 중.

DGE_INPUT_AJS 구조체: Analog Joystick 값을 읽어오기 위한 자료 구조

- int st: 8방향 디지털 조이스틱 방향 [0, 7] 값 리턴. 없으면 -1

- int x, y: Joy-stick의 x, y 값

```

        FLOAT m_vAjCur[3];                // Analog Joystick Current Position

```

```

...
DGE_INPUT_AJS ajTemp;
hr = DGE_InputReadAjs(&ajTemp);
if(DGE_SUCCEEDED(hr))
{
    m_vAjCur[0] = (FLOAT)ajTemp.x;
    m_vAjCur[1] = (FLOAT)ajTemp.y;
}

```

DGE_InputReadTsb: Touch Screen 장치의 x, y 값 읽기

입력 인수: DGE_INPUT_TS 구조체 변수 주소

반환 값: 함수 처리는 이벤트 숫자. 이벤트 숫자가 0이면 내부에서 처리한 것이 없음.

장치 오류가 있을 때 실패 (-) 값 반환.

DGE_INPUT_TS 구조체: Touch Screen 값을 읽어오기 위한 자료 구조

- int st: 장치의 x, y 값을 읽어오는 상태이며

- 처리 중: DGEINPUT_READ_WAIT

- 읽기 완료: DGEINPUT_READ_COMPLETE

- int x, y: Joy-stick의 x, y 값.

```

FLOAT m_vTsbCur[3];                // Tsb Current Position
...
DGE_INPUT_TS tsTemp;
int nEvent=0;                       // Tsb의 Event Number
nEvent = DGE_InputReadTsb(&tsTemp);
if(DGE_FAILED(nEvent))
    return Error;

if(0<nEvent)
{
    m_vTsbCur[0] = (FLOAT)tsTemp.x;
    m_vTsbCur[1] = (FLOAT)tsTemp.y;
    m_TsbCur[0] = tsTemp.st;        // 누름 상태
}

```

DGE_InputReadKey: 버튼 읽기

입력 인수: unsigned char 32 byte 이상의 배열

반환 값: 함수 처리 성공 DGE_OK, 처리 실패 (-) 값

```
BYTE m_KeyCur[DGEINPUT_MAX_KEY];           // Current Keyboard  
hr = DGE_InputReadKey(m_KeyCur);
```

2.2.3 Time

시간 관련 eabi와 윈도우의 Cross Compile 지원.

DGE_tzset: 시간 관련 환경을 설정하는 함수. tzset() 함수에 해당.

반환 값: 성공 - DGE_OK. 실패 - 음수 값

DGE_GetTickCount: 시스템 부팅 후 시간 반환. 윈도우 GetTickCount 함수와 대응

반환 값: DWORD형 millisecond 반환. 1초 = 1000ms

DGE_Sleep: 프로그램을 주어진 값의 시간만큼 대기 상태로 설정

입력 인수: DWORD형 millisecond

DGE_DateandTime: 날짜와 현재 시간을 동시에 반환.

입력 인수: char형 배열 128byte가 필요

<프레임 카운트(Frame Count)>

```
FLOAT GetAppFrame()  
{  
#if defined(_WIN32)  
    static INT          MaxCount = 2000;  
#else  
    static INT          MaxCount = 10;  
#endif  
  
    static DOUBLE  fFps = 0.;  
    static INT     iCnt = 0;  
    static DWORD   dBgn = ::DGE_GetTickCount();  
    DWORD   dCur = ::DGE_GetTickCount(); // GetTickCount()  
    ++iCnt;  
    if(iCnt>=MaxCount)  
    {  
        fFps = DOUBLE(dCur-dBgn);  
        fFps = iCnt*1000./fFps;
```

```

        iCnt = 0;
        dBgn = dCur;
    }
    return (FLOAT)fFps;
}

```

2.2.4 Utility

Cross Compile과 자주 사용되는 함수들, 매크로 등을 정의.

DgeMem_Allock: 메모리 동적 할당 함수. 내부에서 malloc 함수로 버퍼를 할당하고 memset 함수로 초기화

입력 인수:

- int BuffSize: 버퍼 크기
- int Count: 버퍼의 수
- int InitSetVal: 초기화 값

DgeMem_Free: DgeMem_Allock 함수로 동적 할당한 버퍼를 해제. free 함수를 호출하고 NULL로 설정
free(*v) and *v = NULL;

입력 인수: void** v: 버퍼의 주소

DgeStr_tolower: 영문자를 대문자로 변환

DgeStr_toupper: 영문자를 소문자로 변환

DgeStr_Stricmp: 두 문자열을 대소문자 구분 없이 비교. _stricmp 함수와 대응

DgeStr_Strnicmp: 두 문자열을 길이만큼 대소문자 구분 없이 비교. _strnicmp 함수와 대응

DgeStr_SplitPath: 파일의 Full Name을 드라이버, 폴더, 파일 이름, 확장자 등을 분리 _splitpath
함수와 대응

DgeFile_FindFirst: 파일 또는 폴더 검색을 위한 함수. _findfirst 함수에 대응

DgeFile_FindNext: 파일 또는 폴더 검색을 위한 함수. _findnext 함수에 대응

DgeFile_FindClose: 파일 또는 폴더 검색을 위한 함수. _findClose 함수에 대응

DgeFile_GatherAllFiles: 입력 인수의 하위 폴더에 포함된 모든 파일 또는 폴더

입력 인수:

- TDGE_FINF** pOut: 파일과 폴더의 이름을 저장하기 위한 데이터. D-Linked List 구조
- const char* sStartRoot: 검색할 시작 폴더 이름

DgeFile_GatherFree: 파일 검색에 사용한 Linked List 해제

입력 인수: - TDGE_FINF**: Linked List 시작 주소

DgeFile_mkdir: 폴더 만들기

입력 인수:

- const char* sFolder: 폴더 이름
- unsigned int mod: 사용과 접근 권한 Mode

2.2.4 G-sensor

중력 처리 센서를 제어하는 함수들을 포함한 헤더파일. 중력 센서는 화면의 가로 방향을 x, 화면의 높이 또는 측면방향을 y, 그리고 화면의 수직 방향을 z로 해서 설정된 중력 값의 크기로 변환된 값을 읽고 [-, +] 범위의 실수 값을 반환.

센서가 중력의 방향으로만 값을 측정하므로 만약 수평상태에 놓여 있으면 x, y는 0이고, z는 (+) 값. 기기를 수평인 상태에서 화면을 정면을 바라보게 하면 y는 (+), x, z는 0. 비슷하게 측면 방향으로 기기를 세워 놓으면 x는 (+) 또는 (-)이 되고 y, z는 0.

DGE_Gsensor[Open[Close]: G-sensor 사용 초기화/해제

반환 값: 하드웨어 제어를 얻는 경우 DGE_OK 반환. 실패 시 음수 값 반환

DGE_GsensorSetEnv: G-sensor를 사용하기 위한 환경 설정

입력 인수: DGE_GSENSOR_ENV 구조체 변수

반환 값: 함수 처리 성공 DGE_OK, 처리 실패 (-) 값

DGE_GSENSOR_ENV: G-sensor의 환경을 설정하기 위한 자료구조

- int Range: 중력 범위. 2G, 4G, 8G 중 하나를 선택
- int Resolution: 해상도 8-bit, 12-bit 중 하나를 선택
- int TapHz: 두들김 등의 간단한 충격 측정. 50, 100, 200, 400 Hz 중 하나를 선택
- int TiltHz: 화면 Tilt 측정. 1, 6, 12, or 50 Hz 중 하나를 선택
- int TiltTimer: Tilt 측정 시간 [0, 255] 범위의 정수 선택

DGE_GsensorRead: G-sensor 장치의 x, y, z 축에 대한 절대 값 읽기

입력 인수: DGE_GSENSOR_VAL 구조체 변수

반환 값: 읽기 성공 DGE_OK, 실패 (-) 값

DGE_GSENSOR_VAL: G-sensor 값을 읽기 위한 자료 구조

- int x, y, z 3축에 대한 중력 값. 값이 없으면 0xFFFF

int TapMod: 두들김 측정 +/-
int TapDir: 두들김 방향.

<G-sensor>

```
#include <DGE_Type.h>
#include <DGE_Gsensor.h>

...

DGE_GSENSOR_ENV g_GsEnv;                // G 센서 환경
DGE_GSENSOR_VAL g_GsRead;               // G 센서 반응

...

if(DGE_FAILED(DGE_GsensorOpen()))        // G 센서 사용 초기화
    return -1;

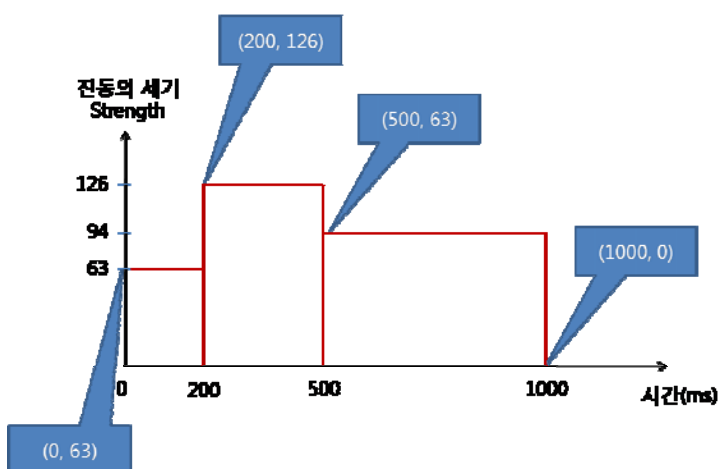
// Run Main Loop
{
    hr = DGE_GsensorRead(&g_GsRead); // Read G-sensor
    if(-1 != hr)
        printf("%3d %3d %3d\n", g_GsRead.x, g_GsRead.y, g_GsRead.z);
}

...

DGE_GsensorClose();                      // G 센서 사용 종료 처리
```

2.2.5 Haptic

CAANOO는 2개(왼쪽, 오른쪽)의 진동 모터가 장착되어 있고 진동의 크기, 지속시간 등을 제어 가능.



<진동의 세기와 지속시간 설정 방법>

연속적으로 진동이 변화하는 프로그램을 작성할 때 진동의 세기와 지속시간은 하나의 쌍으로 설정하고 이것을 진동 모터에 전달.

DGE_Haptic[Open|Close]: 진동 모터 사용 초기화/해제

반환 값: 하드웨어 제어를 얻는 경우 DGE_OK 반환. 실패 시 음수 값 반환

DGE_HapticStatus: 진동 모터의 상태

반환 값: busy = VIB_STATUS_BUSY, stop = VIB_STATUS_IDLE

DGE_HapticEnable: 진동 모터 사용 여부 설정

입력 인수: 정수형. Enable=TRUE, Disable = FALSE

반환 값: 성공 - DGE_OK. 실패 - 음수 값

DGE_HapticPlay: 진동 모터 동작 시작

입력 인수:

- int time_line: 지속 시간

- int strength: 진동 세기

DGE_HapticStop: 진동 모터 동작 중지

DGE_HapticPlayPattern: 진동 모터의 실행 패턴을 설정

입력 인수: pattern_data_t 구조체 배열

pattern_data_t: 진동 패턴을 구성하는 자료 구조

- short time_line: 진동의 지속 시간

- short vib_strength: 진동의 세기

<진동 모터>

```
#include <DGE_Type.h>
```

```
#include <DGE_Haptic.h>
```

```
const int basisActivateTime = 1000;          // 진동 기본 작동 시간( 단위 1/1000 초)
```

```
vibration_act_t vibrationRecord[DGE_VIB_MAX_ACTION];    // 진동 동작 기록 저장 공간
```

```
int recordTime = 0;                                // 진동 동작 기록 시간 (단위 1 / 1000 초, 절대 시간)
```

```
int recordPosition = 0;                            // 진동 동작 기록 위치
```

```
enum VIBRATION_POWER                                // 진동 세기
```

```

{
    VIBRATION_POWER_ZERO = 0,
    VIBRATION_POWER_WEAK = 40,
    VIBRATION_POWER_MIDDLE = 80,
    VIBRATION_POWER_STRONG = DGE_VIB_STRENGTH_MAX,
};

// 이전 진동 동작 정지
void VibrationStop()
{
    // 현재 진동 장치 상태 확인
    if(DGE_VIB_STATUS_BUSY == DGE_HapticStatus())
        DGE_HapticStop();    // 진동 정지
}

// 진동 동작 기록 초기화
void InitializeVibrationRecord()
{
    for(int count = 0; count != DGE_VIB_MAX_ACTION; ++count)
    {
        vibrationRecord[count].time_line = 0;    // 진동 동작 기록 초기화
        vibrationRecord[count].vib_strength = VIBRATION_POWER_ZERO;
    }

    recordTime = 0;    // 진동 동작 기록 시간 초기화
    recordPosition = 0;    // 진동 동작 기록 위치 초기화
}

// 진동 동작 기록( 패턴 생성 )
void RecordVibrationPlay(int playTime, int strength)
{
    // 진동 동작 기록 위치 범위 확인
    // ( 최대 개수, 종료 처리 사용 - 1, 배열 인덱스 사용 - 1)
    if((0 > recordPosition) || ( DGE_VIB_MAX_ACTION - 2 < recordPosition))
        return;    // 허용 범위 이외(기록 불가능)

    // 허용 범위 이내(기록 가능)

```

```

// 진동 동작 기록
vibrationRecord[recordPosition].time_line = (short)recordTime;
vibrationRecord[recordPosition].vib_strength = (short)strength;

recordTime += playTime;          // 다음 진동 동작 기록 시간 설정
++recordPosition;                // 다음 진동 동작 기록 위치 설정
}

// 진동 동작 기록 재생
void PlayVibrationRecord()
{
    // 진동 동작 기록 위치 범위 확인
    if(0 >= recordPosition)
        return;                  // 허용 범위 이외(재생 불가능, 내용 없음)

    VibrationStop();              // 이전 진동 동작 정지

    // 진동 동작 기록 종료 처리
    vibrationRecord[recordPosition].time_line = (short)recordTime;
    vibrationRecord[recordPosition].vib_strength = (short)VIBRATION_POWER_ZERO;

    // 패턴 생성
    pattern_data_t vibrationPattern;
    // 진동 동작 기록 개수 설정(배열 인덱스 사용 + 1)
    vibrationPattern.act_number = recordPosition + 1;

    // 진동 동작 기록 저장 공간 크기만큼 반복
    for(int count = 0; count != DGE_VIB_MAX_ACTION; ++count)
    {
        // 진동 동작 기록 정보 복사
        vibrationPattern.vib_act_array[count].time_line=
            vibrationRecord[count].time_line;
        vibrationPattern.vib_act_array[count].vib_strength=
            vibrationRecord[count].vib_strength;
    }

    // 진동 동작 기록 재생

```

```

        DGE_HapticPlayPattern(&vibrationPattern);
    }

    int main(int argc, char* argv[])
    {
        ...

        DGE_HapticOpen();           // 진동 장치 초기화
        DGE_HapticEnable(1);        // 진동 장치 활성화
        InitializeVibrationRecord(); // 진동 동작 기록 초기화
        ...

        // Main Loop
        {
            if(isRecord)            // 진동 동작 기록 여부 확인
            {
                VibrationStop(); // 이전 진동 동작 정지

                // 진동 동작 재생
                DGE_HapticPlay(basisActivateTime, vibrationStrength);

                // 진동 동작 기록
                RecordVibrationPlay(basisActivateTime, vibrationStrength);

                // 정보 변경
                isChangeInformation = true;
            }
        }

        DGE_HapticClose();         // 진동 장치 종료
        ...
    }

```

2.3 Graphics

2.3.1 IDGE_Device

OpenGL의 그래픽 파이프라인과 프로그램의 주요 인스턴스를 저장하고 사용하기 위해 만들어진 인터페이스. DGE_CreateDevice 함수로 객체 생성. DGE_DestroyDevice 함수로 인스턴스 해제.

<디바이스 생성>

```
#include <DGE_System.h>
#include <DGE_Math.h>
#include <DGE_Base.h>

DGE_HANDLE      m_hWnd;                // Window 핸들
IDGE_Device*    m_pDev = NULL;        // Device

...

INT DGE_CALLBACK Run(void*)
{
    ...

    m_pDev->BeginScene();                // Prepare Rendering...

    // Clear Backbuffer
    m_pDev->Clear(DGE_CLEAR_COLOR_BUFFER | DGE_CLEAR_DEPTH_BUFFER);

    // Rendering...

    m_pDev->EndScene();                  // Post Rendering...
    return 0;
}

int main(int argc, char* argv[])
{
    DGE_Init();                          // DGE 초기화
    DGE_SetMainRunFunc(Run);              // Main Procedure 함수 설정

    // Window 생성
    m_hWnd = DGE_CreateWindow(WndProc, "DGE_Tutorial", 320, 240);
    if(NULL == m_hWnd)
    {
        printf("DGE_CreateWindow Failed\n");
        goto End;
    }

    // Device 생성
    if(DGE_FAILED(DGE_CreateDevice(NULL, &m_pDev, m_hWnd, NULL)))
```

```

    {

        printf("DGE_CreateDevice Failed\n");
        goto End;
    }

    while(1)                                // Main Loop
    {

        if(DGE_FAILED(DGE_Run()))
            break;

        m_pDev->Present(0,0,0,0); // Flipping
    }

    End:

    DGE_DestroyDevice(&m_pDev);           // Device 해제
    DGE_DestroyWindow(m_hWnd);           // Window 해제
    DGE_Close();                          // DGE 해제

    return 0;
}

```

2.3.1.1 렌더링 머신 설정 함수

SetTransform: 정점의 변환(Transform) 행렬 설정 함수

정점의 위치는 그래픽 파이프라인에서 장면을 구성하기 위해 먼저 월드 변환(World Transform)을 거치고 다음으로 카메라 공간의 위치로 뷰 변환(View Transform)을 거치며 장치에 독립적인 정규 변환을 거칩니다. 이 때 그래픽 파이프라인은 4x4 행렬을 사용해서 크기(Scaling), 회전(Rotation), 이동(Translation)을 한 번에 진행합니다. OpenGL은 월드 변환과 뷰 변환을 묶어서 한 번에 처리하는 Model-View 변환을 합니다. DGE는 Direct3D와 OpenGL ES 2.0 이상에서 유연성을 높일 수 있도록 월드, 뷰, 투영 변환을 분리해 놓고 있습니다.

각각의 변환에 대한 함수는 SetTransform 함수를 호출해서 사용하며 이 함수의 첫 번째 인수에 변환의 종류를 넣고 다음 인수에 float 형 16개의 배열을 전달합니다. DGE는 변환에 대해서 행렬의 연산을 정점의 위치를 왼쪽에 놓고 좌에서 우로 곱셈이 진행됩니다. 따라서 보통 수학에서 사용되는 오른쪽에서 왼쪽으로 진행되는 것에 반대 방향이 됩니다.

행렬의 연산에 관련해서 DGEMATRIX 구조체는 Direct3D과, OpenGL 등에서 커다란 수정 없이 호환성이 높도록 구성되어 있어서 이 구조체를 사용하면 좀 더 편리합니다.

SetTransformIdentity: 정점의 변환(Transform) 행렬을 단위 행렬로 설정. 가장 많이 사용되는 곳은

월드 변환으로 만약 파이프라인의 월드 변환 행렬을 SetTransform 함수로 수정했다면 장면을 구성한 후에 SetTransformIdentity 함수로 월드 변환 행렬을 초기화.

BeginScene/EndScene: 장면의 렌더링 상태의 시작과 끝을 알리는 함수.

모든 2D, 3D 객체는 이 함수 사이에서 Draw를 호출해야 정상적으로 작동.

<월드 행렬 설정>

```
PDGE_DEVICE      m_pDev          ;          // DGE_DEVICE
...
DGXMATRIX        mtRotX;
DGXMATRIX        mtRotZ;
DGXMATRIX        mtWld;
mtRotZ.SetupRotationZ( DGXToRadian(c*3.5f));
mtRotX.SetupRotationX( DGXToRadian(c*3.5f));
mtWld = mtRotZ * mtRotX;
mtWld._41 = 10;
mtWld._42 = -10;
m_pDev->SetTransform(DGE_TS_WORLD, &mtWld);
```

SetRenderState: 렌더링 가상 머신의 상태 값을 설정하는 함수

렌더링 머신 상태는 EDGE_RENDERINGSTATE에 정의되어 있고, 각각의 상태 값은 EDGE_RENDERINGSTATUS에 정의. OpenGL에서 상태 머신의 On/Off 기능은 glEnable/glDisable 함수로 설정. 이 것은 SetRenderState 함수에서 충분히 처리할 수 있지만 SetStatusEnable/SetStatusDisable 함수로 처리해도 됨.

<상태 머신 값 설정>

```
m_pDev->SetRenderState(DGE_RS_SHADE_MODEL, DGE_ST_SMOOTH); // Shade Model
m_pDev->SetRenderState(DGE_RS_DEPTH_TEST, DGE_ST_ENABLE); // Depth Test Enable
m_pDev->SetRenderState(DGE_RS_CULL_FACE, DGE_ST_ENABLE); // Culling Enable
```

SetTexture...(): 텍스처의 샘플링(Sampling)과 멀티 텍스처 처리를 설정하는 함수

SetTextureAddressS: S 방향 샘플링의 주소 Mode 설정. DGE_TS_REPEAT, DGE_TS_CLAMP

SetTextureAddressT: T 방향 샘플링의 주소 Mode 설정. DGE_TS_REPEAT, DGE_TS_CLAMP

SetTextureFilterMin: 텍스처의 Minification에 대한 필터링 설정.

- NEAREST, LINEAR, [NEAREST|LINEAR]_MIPMAP_[NEAREST|LINEAR]

SetTextureFilterMag: 텍스처의 Magnitude에 대한 필터링 설정, NEAREST, LINEAR

SetTextureStateOp: 멀티 텍스처 처리에서 단계별 색상 혼합 방법 설정.

- MODULATE, DECAL, ADD, ADD_SIGNED, SUBTRACT
SetTextureEnv: OpenGL의 glTexEnvf 함수

<멀티 텍스처 처리 상태 설정>

```
m_pDev->SetTexture(0, m_pTex0);           // 0 번 텍스처 설정
m_pDev->SetTextureAddressS (0, DGE_TS_REPEAT);
m_pDev->SetTextureAddressT (0, DGE_TS_REPEAT);
m_pDev->SetTextureFilterMin(0, DGE_TS_LINEAR);
m_pDev->SetTextureFilterMag(0, DGE_TS_LINEAR);

m_pDev->SetTexture(1, m_pTex1);           // 1 번 텍스처 설정
m_pDev->SetTextureAddressS (1, DGE_TS_REPEAT);
m_pDev->SetTextureAddressT (1, DGE_TS_REPEAT);
m_pDev->SetTextureFilterMin(1, DGE_TS_LINEAR);
m_pDev->SetTextureFilterMag(1, DGE_TS_LINEAR);
```

2.3.1.2 렌더링 데이터 설정 함수

SetFVF: 입력 스트림인 정점 스트림의 버텍스 포맷 설정

SetTexture: 멀티 텍스처 단계의 텍스처 설정. 총 2단계까지 지원

SetColor: 렌더링 파이프라인의 처리과정에서 Diffuse 색상을 설정. OpenGL glColor[f|v]와 대응

SetStreamSource: VBO(Vertex Buffer Object)를 파이프라인에 연결

SetIndices: IBO(Index Buffer Object)를 파이프라인에 연결

2.3.1.3 출력 함수

DrawPrimitive[f|i]: 인덱스가 없는 VBO를 출력 (f: float 형 데이터, i: Fixed Point Data)

DrawIndexedPrimitive[f|i]: IBO를 VBO에 결합해서 출력

DrawPrimitiveUP[f|i]: Stack, Heap 공간의 사용자 정의 데이터(User Pointer)를 출력.

DrawIndexedPrimitiveUP[f|i]: Stack, Heap 공간의 사용자가 지정한 인덱스와 정점 데이터를 출력

Clear: 프레임 버퍼의 색상 버퍼, 깊이 버퍼 등을 설정한 값으로 초기화. glClear에 대응

Present: 프레임 버퍼의 내용을 화면에 출력. eglSwapBuffer 와 glFlush 함수에 대응

2.3.1.4 응용 프로그램 변수 설정

SetMainApp: 응용 프로그램의 메인 데이터 또는 클래스 인스턴스를 저장. 사용자가 처리

SetSprite: IDGE_Sprite 인스턴스가 만들어질 때 이 인스턴스를 저장.

SetInput: IDGX_Input 인스턴스가 생성될 때 이 인스턴스를 저장.

SetNetwork: 네트워크의 메인 인스턴스를 저장. 사용자가 처리.

SetUserData[1...4]: 사용자가 지정하는 주요 데이터의 포인터 저장

IDGE_Device에 저장되어 있는 응용 프로그램 변수들은 IDGE_Device 인스턴스만 있으면 Get...() 함수로 즉시 사용

2.3.2 IDGE_Sprite

2D 그래픽 처리. DGE_CreateSprite 함수로 인스턴스 생성. DGE_DestroySprite 함수로 인스턴스 해제. 모든 2D 그래픽은 Begin/End 함수 사이에서 출력해야 함.

Begin: 2D 그래픽 출력 초기 상태 설정. 2D 그래픽 출력은 내부에서 월드 변환, 뷰 변환, 투영 변환 행렬을 초기화하고 이를 저장

End: Begin에서 저장된 값들을 복원

Draw: 크기, 회전 변환이 없는 2D 그래픽 출력. DrawEx보다 출력 속도 높음

DrawEx: 크기, 회전, 이동 변환이 있는 2D 그래픽 출력. 크기 변환 값이 음수이면 수평, 수직 축에 대칭으로 출력.

2.3.3 IDGE_Image

파일 또는 버퍼의 픽셀 데이터 객체. 비 압축 BMP 와 TGA, PNG 지원. JPG는 윈도우 시스템에서만 사용 가능. 윈도우 Emulator를 실행할 경우 GPT 파일 생성. GPT 파일은 32비트 픽셀을 알파 채널과 색상 키 (Color Key)를 가지고 16비트 색상으로 변환. 색상 키가 없으면 R5G6B5. 색상 키가 있으면 R5G5G5A1. 알파 채널이 있으면 R4G4B4A4. 색상 변환은 DGE_Convert32to16Color 함수에서 처리. DGE_CreateImage 함수로 객체 생성 DGE_DestroyImage 함수로 인스턴스 해제.

GetRscType: 픽셀의 리소스(Resource) 타입 반환. BMP, TGA, PNG, GPT

GetFormat: OpenGL 형식의 픽셀 포맷 반환. GL_RGB, RGBA, LUMINANCE, LUMINANCE_ALPHA

GetType: OpenGL 형식의 픽셀 타입 반환. UNSIGNED_SHORT565,
UNSIGNED_SHORT4444, UNSIGNED_SHORT5551

GetImgW: 이미지 가로 폭 반환. 윈도우 Emulator는 2의 승수로 구성

GetImgH: 이미지 세로 폭 반환. 윈도우 Emulator는 2의 승수로 구성.

GetImgD: 픽셀의 깊이 바이트 반환. 1 = 8비트, 2 = 16비트, 3 = 24비트, 4 = 32비트 색상

GetPixel: 픽셀 버퍼

SaveToFile: 픽셀을 TGA, GPT 파일로 저장

2.3.4 IDGE_Texture

텍스처 객체. 픽셀 데이터 필요. DGE_CreateTexture 함수로 파일 또는 픽셀 버퍼에서 텍스처 생성. 이미지 파일에서 텍스처 객체를 생성할 경우 내부에서 IDGE_Image 객체를 사용. 텍스처는 중복 생성을 막기 위한 Owner가 존재. DGE_DestroyTexture 함수로 인스턴스 해제할 때 Owner가 없는 경우만 해제. Owner는 SetOwner 함수로 사용자가 설정. NULL을 전달하면 Owner 해제

윈도우 시스템에서 DGE_CreateTexture 함수로 텍스처를 생성할 경우 색상 키와 이미지의 Channel에 따라 기기에 맞는 적절한 포맷의 16비트 색상의 gptf 형식 이미지 파일 생성.

Channel 수	사용자가 지정한 Color Key	gptf 파일의 픽셀 포맷
3 (24bit)	0x00000000 (지정 안 함)	GL_UNSIGNED_SHORT_5_6_5
	0x00000000 값 이외	GL_UNSIGNED_SHORT_5_5_5_1
4 (32bit)	Color Key 상관 없음	GL_UNSIGNED_SHORT_4_4_4_4

기기는 먼저 gptf 파일을 찾고 이 파일이 없으면 png, tga, bmp 등을 찾아서 이미지를 읽어온 후, gptf 파일 포맷으로 16비트 텍스처를 생성. bmp 파일의 팔레트는 32비트로 픽셀을 변경하기 때문에 24비트 또는 32비트 이미지 사용을 권장.

이외의 포맷은 OpenGL ES 포맷에 맞게 사용자가 구성해야 함.

주의: 이미지 크기는 2의 승수로 맞추는 것이 성능에 가장 좋음. 이미지의 너비와 높이는 4의 배수로 맞추어야 출력됨.

GetName: IDGE_Texture는 관리를 위해 이름이 존재. 보통은 파일 이름을 그대로 사용

GetId: OpenGL ID 반환

GetFormat: 픽셀의 포맷 반환

GetType: 픽셀의 타입 반환

GetImgW: 이미지 너비 반환

GetImgH: 이미지 높이 반환

GetImgD: 이미지 깊이 반환

GetImgC: 설정된 색상 키 반환

SetOwner: 텍스처 Owner 설정. NULL 전달하면 Owner 해제

GetOwner: Owner 반환.

BindTexture: OpenGL의 BindTexture 함수 호환을 위한 함수. 멀티 텍스처, 필터링, 어드레싱 설정

<2D 렌더링>

```

#include <DGE_Base.h>

PDGE_SPRITE          m_pSprite= NULL; // 2D Sprite
vector<PDGE_TEXTURE>  m_vTex;          // Texture List
...
// Sprite 생성
if(DGE_FAILED(DGE_CreateSprite(NULL, &m_pSprite, m_pDev)))
    return -1;

// Texture 생성
for(i=0;...)
{
    ...
    PDGE_TEXTURE      pTex = NULL;
    if(DGE_FAILED(DGE_CreateTexture(NULL, &pTex, (DGE_HANDLE)sFile)))
        return -1;
    m_vTex.push_back(pTex);
}

m_pDev->BeginScene();
m_pDev->Clear(DGE_CLEAR_COLOR_BUFFER | DGE_CLEAR_DEPTH_BUFFER);

m_pSprite->Begin(); // Sprite Begin

m_pSprite->Draw(m_vTex[0], NULL, NULL);

DGE_RECT  rc;
DGXVECTOR2 s;  DGXVECTOR2 r;  DGXVECTOR2 t;  DGXCOLOR d;
rc = DGE_RECT( 0, 0, 128, 128);      t = DGXVECTOR2( 10, 10);
    m_pSprite->Draw(m_vTex[1], &rc, &t);

for(INT i=5; ...)
{
    FLOAT X = FLOAT(rand())%320)-60;
    FLOAT Y = FLOAT(rand())%240)-60;

    t = DGXVECTOR2(X, Y);

```

```

        m_pSprite->Draw(m_vTex[i], NULL, &t, NULL);
    }

    float fRadian = DGXToRadian(c);
    rc = DGE_RECT(0,0, m_vTex[45]->GetImgW(), m_vTex[45]->GetImgH());
    s = DGXVECTOR2( -.5f, .5f);
    r = DGXVECTOR2(100, 100);
    t = DGXVECTOR2(100, 100);
    d = DGXCOLOR(0,1,1,0.5f);
    m_pSprite->DrawEx(m_vTex[45], &rc, &s, &r, fRadian, &t, &d);

    m_pSprite->End(); // Sprite End

    m_pDev->EndScene(); // Post Rendering...

    ...
    // Texture 해제
    for(i=0;i<m_vTex.size(); ++i)
    {
        PDGE_TEXTURE pTex = m_vTex[i];
        DGE_DestroyTexture(&pTex);
    }

    m_vTex.clear(); // Texture list clear

    DGE_DestroySprite(&m_pSprite); // Sprite 해제

```

2.3.5 IDGE_VB

Vertex Buffer Object를 위한 인터페이스. DGE_CreateVB 함수로 객체 생성. DGE_DestroyVB 함수로 객체 해제. 버텍스 버퍼는 공용해서 사용하는 경우가 많아 내부에 Owner가 존재. Owner가 없어야 DGE_DestroyVB 함수로 객체 해제가 안됨.

Fill: 버텍스 버퍼의 데이터 갱신.

Bind: 파이프라인에 VBO 바인딩. glBindBuffer 함수를 내부에서 호출

SetOwner: Owner 설정. NULL 전달할 때 Owner 해제. Default Owner = NULL

2.3.6 IDGE_IB

Index Buffer Object를 위한 인터페이스. DGE_CreateIB 함수로 객체 생성. DGE_DestroyIB 함수로 객체 해제. 인덱스 버퍼는 공용해서 사용하는 경우가 많아 내부에 Owner가 존재. Owner가 없어야 DGE_DestroyIB 함수로 객체 해제가 됨.

Fill: 인덱스 버퍼의 데이터 갱신.

Bind: 파이프라인에 IBO 바인딩. glBindBuffer 함수를 내부에서 호출

SetOwner: Owner 설정. NULL 전달할 때 Owner 해제. Default Owner = NULL

<VBO, IBO 렌더링>

```
PDGE_VB      m_pVB;
PDGE_IB      m_pIB[6];
...
// 시스템 메모리에 정점 데이터 생성
GFIXED vertices[] =
{
    -10, -10, -10,   +10, -10, -10,  +10, +10, -10,  -10, +10, -10,
    -10, -10, +10,   +10, -10, +10,  +10, +10, +10,  -10, +10, +10
};

DWORD  dFVF = DGE_FVF_XYZ;
INT     nSize = sizeof(vertices);
// DGE_VBO 생성과 데이터 채우기
if(DGE_FAILED(DGE_CreateVB(NULL, &m_pVB, m_pDev, &dFVF, vertices, &nSize)))
    return -1;

dFVF = DGE_FMT_INDEX16;
nSize = sizeof(WORD) * 4;
// DGE_IBO 생성
for(i=0; i<6; ++i)
{
    if(DGE_FAILED(DGE_CreateIB(NULL, &m_pIB[i], m_pDev, &dFVF)))
        return -1;
}

// IBO 데이터 채우기
```

```

WORD front [] = {2, 1, 3, 0};          // front face
m_pIB[0]->Fill(front , nSize);

...

// 렌더링
m_pDev->BeginScene();

m_pDev->Clear(DGE_CLEAR_COLOR_BUFFER | DGE_CLEAR_DEPTH_BUFFER);

...

static int c =0;
++c;

DGXMATRIX      mtRotX;
DGXMATRIX      mtRotZ;
DGXMATRIX      mtWld;

mtRotZ.SetupRotationZ( DGXToRadian(c*3.5f));
mtRotX.SetupRotationX( DGXToRadian(c*3.5f));
mtWld = mtRotZ * mtRotX;
mtWld._41 = 10;
mtWld._42 = -10;

m_pDev->SetTransform(DGE_TS_WORLD, &mtWld);
m_pDev->SetStreamSource(m_pVB);
m_pDev->SetFVF(DGE_FVF_XYZ);

m_pDev->SetColor(1, 0, 0, 1);
m_pDev->SetIndices(m_pIB[0]);
m_pDev->DrawIndexedPrimitive(DGE_PT_TRIANGLE_STRIP, 4);

m_pDev->SetTransformIdentity(DGE_TS_WORLD);
m_pDev->EndScene();

// VBO, IBO 해제
DGE_DestroyVB(&m_pVB);          // VBO 해제

for(i=0; i<6; ++i)
    DGE_DestroyIB(&m_pIB[i]);    // IBO 해제

```

3. DGX

- 라이브러리: CAAN00 - libdgx20.so, WIZ - libdgx20w.so, Windows - dgx20_.lib
- Input, Media, Sound, Network, 3D Util로 구성. 3D 사운드는 OpenAL 참조.

3.1 IDGX_Input

저 수준 DGE Input을 클래스화. 버튼, 아날로그 조이스틱, 터치 스크린 처리.

Update: 아날로그 조이스틱, 버튼, 터치 스크린 순으로 이벤트 처리

GetKeyMap: 키 이벤트를 저장한 배열의 주소를 반환.

GetTsbMap: 터치 스크린 이벤트를 저장한 배열의 주소를 반환

KeyDown: Down 상태는 이전 상태 == 0, 현재 상태 == 1 일 경우 Down 상태 반환.

KeyUp: Up 상태는 이전 상태 == 1, 현재 상태 == 0 일 경우 Up 상태 반환

KeyPress: 현재 상태 == 이전 상태 == 1일 경우 Press 반환

KeyState: 버튼 상태 반환. Down = 1, Up = 2, Press = 3

TsbDown: 터치 스크린 Down 상태 반환. 상태의 조건은 버튼과 동일

TsbUp: 터치 스크린 Up 상태 반환

TsbPress: 터치 스크린 Press 상태 반환

TsbState: 터치 스크린 상태 반환

GetTsbPos: 터치 스크린 위치 반환. FLOAT x 3 필요

GetTsbDelta: 터치 스크린의 상대 좌표 값 반환. FLOAT x 3 필요.

IsTsbMove: 터치 스크린 동작이 있는지 반환

SetCurKey: 버튼의 이벤트 값을 설정

SetCurTsb: 터치 스크린의 현재 상태를 설정.

SetTsbPos: 터치 스크린의 현재 위치 설정.

IsInRect: 주어진 직사각형 영역 안에 터치 스크린 위치가 존재하는지 판별

Reset: DGX_Input 데이터의 모든 값을 0으로 초기화

SetActive: 활성화/비활성화 설정. 내부의 값들은 변경하지 않고 Down, Up, Press의 반환 값만 0으로 리턴. Default: Active

GetActive: 활성화/비활성화 값을 반환

<IDGX_Input>

```
#include <DGE_Input20.h>
```

```
#include <DGX_Input.h>
```

```

PDGE_DEVICE      m_pDev  = NULL;          // Device
PDGX_INPUT       m_pInput= NULL;          // Input

// Device 생성
if(DGE_FAILED(DGE_CreateDevice(NULL, &m_pDev, m_hwnd, NULL)))
    return -1;

// Input 생성
if(DGE_FAILED(DGX_CreateInput(NULL, &m_pInput, m_pDev, m_hwnd)))
    return -1;;

...

// Main Loop
m_pInput->Update();          // Input Update

// Input 터치 스크린 데이터(위치, 상태) 가져오기
DGXVECTOR3       vcTsbPos = m_pInput->GetTsbPos();
DGE_STATE        nTsbState= m_pInput->TsbState();

if(nTsbState)
    printf("%3.f %3.f %d\n", vcTsbPos.x, vcTsbPos.y, nTsbState);

// 버튼
const BYTE*      pKey = m_pInput->GetKeyMap();
for(int i=0; i<DGEINPUT_KEY_TOTAL; ++i)
{
    if(pKey[i])
        printf("You Pressed %d: state: %d\n", i, pKey[i]);
}

...

// Input 해제
DGX_DestroyInput(&m_pInput);

// Device 해제
DGE_DestroyDevice(&m_pDev);

```

3.2 Media

녹음, 볼륨 조절, USB 등 기기의 주변 장치들을 제어.

3.2.1 Mixer

왼쪽, 오른쪽 소리 크기 조절

DGX_MixerOpen: Mixer 장치 사용 초기화.

DGX_MixerClose: Mixer 장치 사용 해제

DGX_MixerGetVolumeRange: Mixer 장치의 볼륨 최소값, 최대 값 얻기(0~127).

DGX_MixerSetVolume: Mixer 장치의 볼륨(전체 볼륨) 설정

DGX_MixerGetVolume: Mixer 장치의 볼륨 크기 얻기

3.2.2 Microphone

마이크 녹음

DGX_RecordingSound: 녹음. 소리를 녹음하기 위해 1024 KB가 필요

3.2.3 기타

DGX_Battery: 배터리 잔여 량. High=1, Middle=2, Low=3, Empty=4

3.3 Network

TCP/IP 통신 처리. I/O 모델은 Select 방식 사용

DGX_NetDeviceOpen: 네트워크 장치 사용 권한 얻기

DGX_NetDeviceClose: 네트워크 장치 사용 반납.

DGX_SocketAddr: IP, Port를 입력 받아 소켓 주소 구조체 설정

DGX_SocketClose: 생성된 소켓 해제

DGX_SocketTcpCreate: TCP용 소켓 생성

DGX_SocketUdpCreate: UDP용 소켓 생성

DGX_SocketConnect: 클라이언트가 서버에 접속

DGX_SocketBind: 소켓과 소켓 주소 구조체 변수를 바인딩

DGX_SocketListen: 서버의 소켓 Listen

DGX_SocketAccept: 클라이언트의 접속을 기다리고 소켓, 소켓 주소 구조체를 반환

DGX_SocketSend: Data 전송

DGX_SocketRecv: Data 수신

DGX_SocketSelect: 네트워크 입/출력을 Multiplexing으로 처리(Select 방식)

DGX_SocketNonBlocking: 소켓을 Non-Blocking으로 만들.

DGX_SocketNaggleOff: TRUE를 전달하면 Nagle Off

DGX_LocalIpAddress: 기기의 IP 주소 반환.

3.4 IDGX_Sound

OpenAL 기반의 wave 파일을 재생하기 위한 객체. 반드시 사운드 디바이스 생성 후 DGX_CreateSound 함수로 IDGX_Sound 인스턴스 생성. DGX_DestroySound 함수로 인스턴스 해제.

주의: DGE OpenAL 사운드 버퍼는 16 비트 stereo 22kHz로 설정되어 있고 PCM Wave 파일만 지원. 따라서 사운드 파일을 PCM 16비트 22kHz 모노/스테레오로 만드는 것이 적합

DGX_SoundDeviceOpen: 사운드 디바이스 생성

DGX_SoundDeviceClose: 사운드 디바이스 해제

Query: 사운드 객체의 명령 또는 상태 값을 제어하기 위한 함수.

GetType: 사운드 리소스 타입. wave or midi 파일

Play: 사운드 재생

Stop: 재생 중지

Reset: 사운드 재생 위치를 맨 앞으로 이동

Pause: 일시 중지

SetVolume: 볼륨 설정. 원음 크기 = 1.0F

SetRepeat: 반복 횟수. 1번 또는 무한 반복 = DGX_SND_INFINITE

[Get|Set]Status: 중지, 재생, 일시 중지, 초기 위치 등의 상태를 반환

<사운드 재생>

```
#include <DGE_System.h>
#include <DGE_Time.h>
#include <DGE_Math.h>
#include <DGE_Base.h>
#include <DGX_Sound.h>

#define MEDIA_DIR    "../../../media/"

...

#define MAX_SOUND    4
```

```

PDGX_SOUND      m_pSnd[MAX_SOUND];      // Sound Instance

// Sound Device 생성
if(DGE_FAILED(DGX_SoundDeviceOpen()))
    return -1;

char* soundfile[] =
{
    MEDIA_DIR"ding.wav",
    MEDIA_DIR"2.wav",
    MEDIA_DIR"3.wav",
    MEDIA_DIR"groove1.wav",
};

// Sound 객체 생성
for(i=0; i<MAX_SOUND; ++i)
{
    hr = DGX_CreateSound(NULL, &m_pSnd[i], soundfile[i]);
    if(DGE_FAILED(hr))
        return -1;

    m_pSnd[i]->SetVolume(0.8F);      // 볼륨 설정
    m_pSnd[i]->SetRepeat(1);        // 반복 횟수
}

...

// Play Sound
if(m_pInput->KeyDown(DGEKEY_FA))
{
    for(int i=0; i<MAX_SOUND; ++i)
        m_pSnd[i]->Play();
}

...

// Sound 객체 해제
for(i=0; i<MAX_SOUND; ++i)
    DGX_DestroySound(&m_pSnd[i]);

```

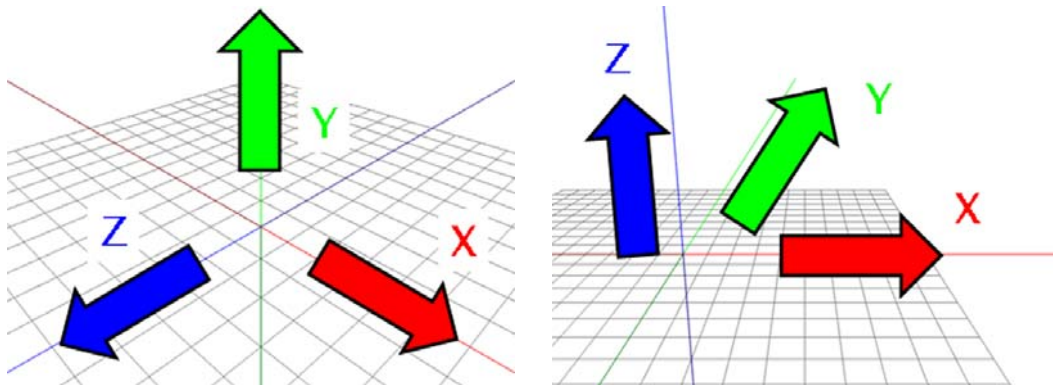
```
// Sound Device 해제
DGX_SoundDeviceClose();
```

3.5 3D Utility

3D 프로그램에 관련된 카메라, 모델(Mesh), 간단한 도형, Grid 등에 대한 Utility

3.5.1 IDGX_Camera

1인칭 카메라 인터페이스. OpenGL은 x, z축 평면에 y축이 하늘을 바라보는 오른손 좌표계를 사용하나 IDGX_Camera는 3D Max와 동일한 x, y 축 평면에 z축이 하늘 방향인 오른손 좌표계 사용



<OpenGL 좌표 축과 DGE Camera 좌표 축>

DGXMATRIX의 SetupViewRH 함수는 IDGX_Camera 기준으로 뷰 행렬(Viewing Matrix)을 만들도록 구성. 참고로 DGXMATRIX의 SetupViewLH 함수는 DirectX의 왼손 좌표 기준으로 한 뷰 행렬.

카메라 클래스의 목적은 프로그램에서 필요한 뷰 변환의 뷰 행렬과 정규 변환의 투영 행렬을 만들고 이것을 파이프라인에 설정하는 것. DGX_CreateCamera 함수로 객체 생성. 반드시 IDGE_Device가 필요. DGX_DestroyCamera 함수로 객체 해제.

Microsoft의 Direct3D에 사용된 데이터를 이용할 경우 y와 z 축의 값을 교환이 필요.

Update: 카메라 데이터 갱신. 카메라의 축, Frustum 등을 계산.

FrameMove: Update 함수를 호출하고 3D 파이프라인에 뷰 행렬과 투영 행렬을 설정

SetTransformViw: 3D 파이프라인의 뷰 행렬 설정.

OpenGL 환경에서 X-Y 평면에 Z 축이 위로 향하는 공간으로 뷰 행렬을 만들 경우 행렬의 2, 3 열의 교환 14, 24, 34의 반전 필요

<파이프라인에 적용할 뷰 행렬 설정>

```

DGXMATRIX mtVw = m_mtVw; // Camera View 행렬
mtVw._12 = m_mtVw._13; mtVw._13 = -m_mtVw._12; mtVw._14 = -m_mtVw._14;
mtVw._22 = m_mtVw._23; mtVw._23 = -m_mtVw._22; mtVw._24 = -m_mtVw._24;
mtVw._32 = m_mtVw._33; mtVw._33 = -m_mtVw._32; mtVw._34 = -m_mtVw._34;
mtVw._42 = m_mtVw._43; mtVw._43 = -m_mtVw._42; mtVw._44 = +m_mtVw._44;

```

SetTransformPrj: 파이프 라인의 투영 행렬 설정. m_pDev->SetTransform(DGE_TS_PROJ, &m_mtPrj);

SetupParamView: 카메라의 뷰 행렬 설정

<뷰 행렬>

```

DGXVECTOR3 vcY = normalize(LookAt - Eye); // 카메라의 y 축
DGXVECTOR3 vcX = normalize(cross(vcY, vcUp)); // 카메라의 x 축
DGXVECTOR3 vcZ = cross(vcX, vcY); // 카메라의 z 축

DGXMATRIX mtVw;
Identity(mtVw); // 항등 행렬로 설정
mtVw._11 = vcX.x; mtVw._12 = vcY.x; mtVw._13 = vcZ.x;
mtVw._21 = vcX.y; mtVw._22 = vcY.y; mtVw._23 = vcZ.y;
mtVw._31 = vcX.z; mtVw._32 = vcY.z; mtVw._33 = vcZ.z;
mtVw._41 = - *pEye * vcX; mtVw._42 = - *pEye * vcY; mtVw._43 = - *pEye * vcZ;

```

SetupParamProj: 카메라의 투영 행렬 설정

<투영 행렬>

```

DGXMATRIX mtPrj;
fFov *= 0.5f; // Field of View

FLOAT cot = (FLOAT)cos(fFov)/(FLOAT)sin(fFov);
FLOAT h = cot;
FLOAT w = cot/fAsp;
Identity(mtPrj); // 항등 행렬로 설정

mtPrj._11 = w;
mtPrj._22 = h;
mtPrj._33 = ( fNear + fFar ) / ( fNear - fFar );
mtPrj._34 = -1.f;

```

```
mtPrj._43 = (2 * fNear * fFar) / (fNear - fFar);
mtPrj._44 = 0.f;
```

SetupMatrixViw: 외부의 행렬을 카메라의 뷰 행렬에 복사.

SetupMatrixPrj: 외부의 행렬을 카메라의 투영 행렬에 복사.

GetAxis[X|Y|Z]: 카메라의 [X|Y|Z] 축 반환

MoveSideward: 1인칭 카메라에 대한 좌/우 이동

MoveForward: 1인칭 카메라에 대한 전/후진 이동

RotateYaw: Z축을 중심으로 한 회전

RotatePitch: X축을 중심으로 한 회전

<IDGX_Camera>

```
#include <DGE_Type.h>
#include <DGE_System.h>
#include <DGE_Math.h>
#include <DGE_Base.h>
#include <DGX_Input.h>
#include <DGX_Camera.h>

PDGX_CAMERA    m_pCam;                // Camera Instance

// Camera 생성과 View, Projection Parameter 설정
if(DGE_FAILED(DGX_CreateCamera(NULL, &m_pCam, m_pDev)))
    return DGE_EFAIL;

DGXVECTOR3      vcEye (30, -100, 30);
DGXVECTOR3      vcLook(30,  0,  0);
DGXVECTOR3      vcUp  ( 0,  0,  1);
m_pCam->SetupParamView( &vcEye, &vcLook, &vcUp);

FLOAT    fFov = DGXToRadian(45.0F);
FLOAT    fScnW= FLOAT(m_nScnW);
FLOAT    fScnH= FLOAT(m_nScnH);
FLOAT    fNear= 1.0F;
FLOAT    fFar  = 5000.0F;
```

```

m_pCam->SetupParamProj(fFov, fScnW, fScnH, fNear, fFar);

...

// Input Update
m_pInput->Update();
const BYTE* pKey = m_pInput->GetKeyMap();

// Camera Move, Rotation Speed 설정
FLOAT speed = (FLOAT)m_fElapsed;
speed *=100.f;

// Translation
if(pKey[DGEKEY_UP]) m_pCam->MoveForward(+speed);
if(pKey[DGEKEY_DOWN]) m_pCam->MoveForward(-speed);
if(pKey[DGEKEY_LEFT]) m_pCam->MoveSideward(-speed);
if(pKey[DGEKEY_RIGHT]) m_pCam->MoveSideward(+speed);

// Rotation
float fAngle_degree = 2.0f;

if(pKey[DGEKEY_FA]) m_pCam->RotateYaw(fAngle_degree, speed);
if(pKey[DGEKEY_FB]) m_pCam->RotateYaw(fAngle_degree, -speed);
if(pKey[DGEKEY_FY]) m_pCam->RotatePitch(fAngle_degree, speed);
if(pKey[DGEKEY_FX]) m_pCam->RotatePitch(fAngle_degree, -speed);

m_pCam->Update();

...

// 렌더링 상태 설정
m_pDev->BeginScene();
m_pDev->Clear(DGE_CLEAR_COLOR_BUFFER | DGE_CLEAR_DEPTH_BUFFER);

// 카메라의 뷰 행렬, 투영 행렬을 렌더링 파이프라인에 반영
m_pCam->SetTransformPrj();
m_pCam->SetTransformViw();

// Rendering
...
m_pDev->EndScene();

```

```
...
DGX_DestroyCamera(&m_pCam);           // 카메라 객체 해제
```

3.5.2 IDGX_Entity

3D 테스트 프로그램에서 간단한 도형, xyz 축 지시선, 크기를 비교할 수 있는 grid 출력.
 DGX_CreateEntity 함수로 객체 생성, DGX_DestroyEntity 함수로 객체 해제.

FrameMove: Update Polygon Data.

Render: 3D 데이터 렌더링.

Query: 데이터 설정 또는 값을 가져오기 위한 함수

<XYZ축과 Grid 생성>

```
#include <DGX_Entity.h>

PDGX_ENTITY    m_pXYZ ;           // X, Y, Z Axis Line
PDGX_ENTITY    m_pGrid ;          // Grid Instance
...
if(DGE_FAILED(DGX_CreateEntity("Axis", &m_pXYZ, m_pDev)))
    return DGE_EFAIL;

if(DGE_FAILED(DGX_CreateEntity("Grid", &m_pGrid, m_pDev)))
    return DGE_EFAIL;

...
// 렌더링
m_pDev->BeginScene();

...
m_pXYZ->Render();
m_pGrid->Render();

m_pDev->EndScene();

...
// 해제
DGX_DestroyEntity(&m_pXYZ);
DGX_DestroyEntity(&m_pGrid);
```

3.5.3 IDGX_Mdl

3D Animation Model 인터페이스. 3D Max의 ASE, GPH의 GPM 지원. 객체 생성 - DGX_CreateMdl 함수, 객체 해제 - DGX_DestroyMdl

윈도우 시스템에서 DGX_CreateMdl 함수는 ASE 파일을 해석한 후 게임에 필요한 asb Binary 파일을 생성.

FrameMove: 모델 내부의 Animation에 관련된 데이터 갱신

Render: 갱신된 데이터 렌더링

SetAttrib: 모델의 데이터 설정. 명령어는 EDGX_MDL에서 정의.

GetAttrib: 모델 내부의 데이터 가져오기. 명령어는 EDGX_MDL에서 정의.

<IDGX_Mdl>

```
#include <DGX_Mdl.h>

PDGX_MDL m_pMdlOrg1;           // Original 1
PDGX_MDL m_pMdlIns1;           // 복제 Instance 1
PDGX_MDL m_pMdlOrg2;           // Original 2
PDGX_MDL m_pMdlIns2;           // 복제 Instance 2
...

char    sMdl1[DGE_MAX_PATH]= MEDIA_DIR"model/mong.ase";
char    sMdl2[DGE_MAX_PATH]= MEDIA_DIR"model/mong.asb";

// 원본 데이터 생성
if(DGE_FAILED( DGX_CreateMdl("Ase Txt", &m_pMdlOrg1, m_pDev, (void*)sMdl1)))
    return DGE_EFAIL;

if(DGE_FAILED( DGX_CreateMdl("Ase Bin", &m_pMdlOrg2, m_pDev, (void*)sMdl2)))
    return DGE_EFAIL;

// 텍스처 로드
DWORD ColorKey = 0x0;
m_pMdlOrg1->SetAttrib(DGXM_EXEC_TEXTURE_LOAD, &ColorKey);

// 원본 데이터에서 모델 복제
if(DGE_FAILED( DGX_CreateMdl("Ase Bin", &m_pMdlIns1, NULL, NULL, m_pMdlOrg2)))
    return DGE_EFAIL;
```

```

if(DGE_FAILED( DGX_CreateMdl("Ase Bin", &m_pMdlIns2, NULL, NULL, m_pMdlOrg2)))
    return DGE_EFAIL;

...

// 애니메이션에 필요한 시간과 행렬 설정
DOUBLE fElapsedTime = this->GetElapsedTime();
DOUBLE fTime;

DGXMATRIX      mtRotX;
DGXMATRIX      mtRotZ;
DGXMATRIX      mtWld;
DGXVECTOR3     vcOrg1(-45, 0, 0);

mtWld.Identity();
mtRotZ.SetupRotationZ(DGXToRadian(0));
mtWld.SetupTranslation( &vcOrg1, FALSE);
mtWld = mtRotX * mtRotZ * mtWld;
fTime = fElapsedTime*0.8;

// Model 1 Update
m_pMdlOrg1->SetAttrib(DGXM_WORLD_MATRIX, &mtWld);
m_pMdlOrg1->SetAttrib(DGXM_ADVANCE_TIME, &fTime);
m_pMdlOrg1->FrameMove();

DGXVECTOR3     vcIns1(-15, 0, 0);
mtWld.Identity();
mtRotZ.SetupRotationZ(DGXToRadian(90));
mtWld.SetupTranslation(&vcIns1, FALSE);
mtWld = mtRotZ * mtWld;
fTime = fElapsedTime*.4;

// Model 2 Update
m_pMdlIns1->SetAttrib(DGXM_WORLD_MATRIX, &mtWld);
m_pMdlIns1->SetAttrib(DGXM_ADVANCE_TIME, &fTime);
// 애니메이션 구간 설정
{
    struct _TT{ INT B; INT E;} t={10, 25};
    m_pMdlIns1->SetAttrib(DGXM_LOOP_SECTION, &t);
}

```

```

}

m_pMdlIns1->FrameMove();

DGXVECTOR3      vcOrg2(15, 0, 0);
mtWld.Identity();
mtRotZ.SetupRotationZ(DGXToRadian(-90));
mtWld.SetupTranslation(&vcOrg2, FALSE);
mtWld = mtRotZ * mtWld;
fTime = fElapsedTime*.6f;

// Model 3 Update
m_pMdlOrg2->SetAttrib(DGXM_WORLD_MATRIX, &mtWld);
m_pMdlOrg2->SetAttrib(DGXM_ADVANCE_TIME, &fTime);
// 애니메이션 구간 설정
{
    struct _TT{ INT B; INT E;} t={25, 50};
    m_pMdlOrg2->SetAttrib(DGXM_LOOP_SECTION, &t);
}
m_pMdlOrg2->FrameMove();

DGXVECTOR3      vcIns2(45, 0, 0);
mtWld.Identity();
mtRotZ.SetupRotationZ(DGXToRadian(-90));
mtWld.SetupTranslation(&vcIns2, FALSE);
mtWld = mtRotZ * mtWld;
fTime = fElapsedTime*1.f;
// Model 4 Update
m_pMdlIns2->SetAttrib(DGXM_WORLD_MATRIX, &mtWld);
m_pMdlIns2->SetAttrib(DGXM_ADVANCE_TIME, &fTime);
{
    struct _TT{ INT B; INT E;} t={140, 140};
    m_pMdlIns2->SetAttrib(DGXM_LOOP_SECTION, &t);
}
m_pMdlIns2->FrameMove();
...

m_pDev->BeginScene();

```

```

...
// 모델 렌더링
m_pMdlOrg1->Render();
m_pMdlIns1->Render();
m_pMdlOrg2->Render();
m_pMdlIns2->Render();
m_pDev->EndScene();

...
// 해제
DGX_DestroyMdl(&m_pMdlOrg1);
DGX_DestroyMdl(&m_pMdlIns1);

DGX_DestroyMdl(&m_pMdlOrg2);
DGX_DestroyMdl(&m_pMdlIns2);

```

4. DGT

- 라이브러리: CAANOO - libdgt20.so, WIZ - libdgt20w.so Windows - dgt20_.lib
- 2D 모아 찍기, 압축, DRM 등 게임 Resource 관리 등으로 구성

4.1 IDGT_SpriteMx

2D 모아 찍기. 가능하면 동일한 텍스처에 Polygon Layer를 만들어 출력.

Begin: 2D 렌더링 환경 설정

End: 3D 렌더링 환경으로 복원

Add: 텍스처와 주어진 이미지 영역을 3D Polygon으로 변환해서 추가. 만약 마지막 Layer의 텍스처와 같으면 마지막 Layer에 3D Polygon만 추가. 같지 않으면 새로운 Layer를 만들고 Polygon 추가.

단순한 렌더링이면 AddEx 보다 처리가 빠름.

AddEx: Add의 확장. 회전, 크기 변환 등 복잡한 2D 렌더링에 대한 Polygon을 만들고 Layer에 추가.

AddtoExist: 존재하는 Texture의 Layer에 Polygon 추가. 텍스처가 이전의 Layer들에 존재하지 않으면 새로운 Layer를 만들고 Polygon 추가. 단순한 렌더링이면 AddtoExistEx보다 빠름.

AddtoExistEx: 회전, 크기 변환 등에 대한 AddtoExist 확장.

DrawLayer(nFirstL, nLastL): 내부의 Layer 등을 부분 출력

DrawLayer(PDGE_TEXTURE): Argument에 주어진 텍스처와 일치하는 모든 Layer 출력.

< IDGT_SpriteMx>

```
#include <DGT_SpriteMx.h>

#define MEDIA_DIR        "../../../media/"

DGE_HANDLE      m_hWnd;           // Window 핸들
PDGE_DEVICE      m_pDev  = NULL;   // Device
PDGE_SPRITEMX    m_pSpriteMx= NULL; // 2D SpriteMx
PDGE_TEXTURE      m_pTex;          // Texture List

INT DGE_CALLBACK Run(DGE_HANDLE)
{
    DGE_RECT      rc;
    DGXVECTOR2 s;   DGXVECTOR2 r;   DGXVECTOR2 t;   DGXCOLOR d;
    for(...)
    {
        FLOAT X = FLOAT(rand()%320)-80;
        FLOAT Y = FLOAT(rand()%240)-80;

        t = DGXVECTOR2(X, Y);
        m_pSpriteMx->Add(m_pTex, NULL, &t, NULL);
    }

    m_pDev->BeginScene();
    m_pDev->Clear(DGE_CLEAR_COLOR_BUFFER | DGE_CLEAR_DEPTH_BUFFER);

    m_pSpriteMx->Begin();           // 2D Rendering 환경 설정
    m_pSpriteMx->DrawLayer();       // Draw Layer
    m_pSpriteMx->End();             // 환경 복귀
    m_pSpriteMx->Clear();           // Layer Clear

    m_pDev->EndScene();
    return 0;
}

INT main(INT argc, char* argv[])
{
```

```

PDGE_TEXTURE    pTex = NULL;

char            sFile[160]= MEDIA_DIR"tex/bg.bmp";

// SpriteMx 생성
if(DGE_FAILED(DGT_CreateSpriteMx(NULL, &m_pSpriteMx, m_pDev)))
    return -1

// Texture 생성
if(DGE_FAILED(DGE_CreateTexture(NULL, &pTex, (DGE_HANDLE)sFile)))
    return -1

m_pTex = pTex;

while(1)        // Main Loop
{
    if(DGE_FAILED(DGE_Run()))
        break;

    m_pDev->Present(0,0,0,0);        // Flipping
}

DGE_DestroyTexture(&pTex);          // Texture 해제
DGT_DestroySpriteMx(&m_pSpriteMx);  // SpriteMx 해제
DGE_DestroyDevice(&m_pDev);         // Device 해제
DGE_DestroyWindow(m_hWnd);          // Window 해제
DGE_Close();                        // DGE 해제
return 0;
}

```

4.2 DGT Lzo

공개 압축 Lzo(<http://www.oberhumer.com/>) 처리

DgtLzo_Init: Lzo 초기화. 실패하면 음수 반환

DgtLzo_Close: Lzo 자원 반납

DgtLzo_CompressFile: Source 파일을 Dest 파일로 압축

DgtLzo_DeCompressFile: 압축된 파일을 특정 폴더에 풀기

DgtLzo_DeCompressFromFile: 압축된 파일로부터 압축이 풀린 데이터를 버퍼에 복사.

버퍼는 free 함수로 해제

DgtLzo_CompressBuf: 버퍼를 압축. 압축된 버퍼의 해제는 free 함수 사용

int DgtLzo_DeCompressBuf: 압축된 버퍼에서 버퍼로 압축 해제.

압축이 해제된 버퍼는 free 함수로 자원 반납.

<Lzo 압축>

```
#include <DGE_Type.h>
#include <DGT_Lzo.h>
#define MEDIA_DIR        "../../../media/"

int main(int argc, char* argv[])
{
    if(DGE_FAILED(DgtLzo_Init()))                // Lzo 초기화
        return 0;

    char    sSrc[260]=MEDIA_DIR"page_03.lua";
    char    sDst[260]=MEDIA_DIR"t.gpz";
    if(DGE_FAILED(DgtLzo_CompressFile(sDst, sSrc))) // 압축
        return -1;

    // 압축 파일을 특정 폴더에 풀기
    if(DGE_FAILED(DgtLzo_DeCompressFile("../", "t.gpz")))
        return -1;

    unsigned char* sbuf;
    unsigned long  lbuf;

    // 압축 파일에서 데이터 직접 가져오기
    if(DGE_FAILED(DgtLzo_DeCompressFromFile(&sbuf, &lbuf, "t.gpz")))
        return -1;

    // Debugging...
    FILE* fp = fopen("tt.txt", "wt");
    fwrite(sbuf, sizeof(unsigned char), lbuf, fp);
    free(sbuf);
    fclose(fp);
}
```

```

    char sLine[512];
    fp = fopen("tt.txt", "rt");

    while(!feof(fp))
    {
        fgets(sLine, 512, fp);
        printf("%s", sLine);
    }

    fclose(fp);
    DgtLzo_Close();
    return 0;
}

```

4.3 DGT DRM

DRM 문서 참조

-끝-