

0. DGE SDK 2.0

1. Overview

DGE SDK Objectives CAANOO gaming device is a compact mobile device based on Linux and is capable of video, flash and games. DGE SDK has been created to seek usability and convenience in developing game titles which can be implemented by this device.

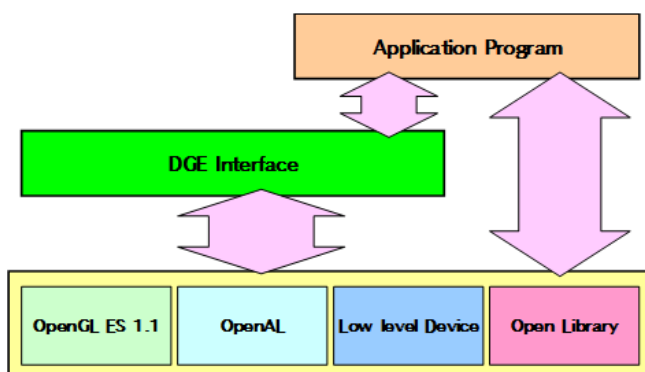
Firstly, DGE offers the Microsoft Windows Library for Windows Emulator to support cross-compiling and the Linux Library to run games on the device. The Microsoft Windows Library for Windows Emulator is partially available for specific devices such as analog joysticks.

SDK consists of three units - Base, Auxiliary, and Utility.

Base is the default structure for operating a game program and is designed to create a game by itself. Base contains functions for program creation, video device, graphics, mathematical functions, low-level input, and C-runtime functions which are not supported by a cross compiler. In addition, the graphics include the video device interface and the sprite for producing games in 2D or 3D. The first thing in preparation for game programming is to bring up the functions contained in Base for correct execution.

DGX (DGE Auxiliary) comprises the upper structure that has implemented the low-level functions within Base as the class, font interfaces that display character strings on the screen, and the network functions that allow communication with other devices. Some objects of Auxiliary, such as fonts, require display output. Therefore, they need DGE Base. Moreover, DGX contains 3D data such as camera or mesh that you can use to develop a game for CAANOO with just two units of DGE and DGX.

DGT (DGE Utility) is where Base, such as 3D data or compressed game resources, and a variety of applications programs not included in Auxiliary are aggregated.



< DGE Layer>

GPH SDK is a library where modules needed for game production are aggregated and organized on the base of open software established as a standard or considered highly reliable with mobile devices such as the OpenGL ES 1.1, OpenAL, etc. If what you prepare for a program is not included in the DGE, you can track what you need from GPH SDK for implementation. Note that GPH SDK offers libraries and functions

that regard the embedded LINUX operating system to control processor, memory and various input/output devices. In addition, open libraries such as OpenGL ES or SDL are included for you to build an independent library of your own.

2. Return Value of Functions and Major Data Types

Every function of DGE SDK that either allocates resources or reads values, returns the result of what is processed. The return value is "0" or higher than "0" for successful cases. However, it is a negative value for unsuccessful cases. The return values for success or failure are defined in the "DGE_Type.h" file as follows:

```
#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 return value implies that the calling of the corresponding function is perfect and no further work is needed. DGE_EFAIL return value refers to a default value for the failure. It is largely used for cases where the cause of such failure is unclear or there is no need to interpret the failure in detail. The data type is set to a 32-bit system and data structures that are frequently used have been declared as follows:

DGE_POINT[n], DGE_INT[n]: 32-bit integer n-dimensional structure
DGE_SHORT[n]: 16-bit integer n-dimensional structure
DGE_FLOAT[n]: Single-precision and double n-dimensional structure
DGE_COLORB: 32-bit color structure
DGE_COLORF: Structure for processing 32-bit colors into floating-point format
DGE_MATRIX: Float-type 4 x 4 matrix

It is common for game programming to define not only the member variables of a structure, but also the member function and operating-overriding function for the purpose of using arithmetic operations more conveniently. For example, the "DGE_Math.h" file is declared with structures such as vector, matrix, quaternion, color, and the member functions on arithmetic operations.

If you work with a structure under the environment of MS Visual C++ development, and the cross compiler of the ARM compiler, take care when packing or padding. It is not necessary for every member of the data structure to be declared as 32-bit format. However, if it includes a 16-bit or 8-bit variable, Visual C++ and ARM compiler should take different forms. While the Visual C++ Compiler of Microsoft can pack memory sizes with the same directive as the `#pragma pack (...)`, "`#pragma`" only works on the Microsoft platform. To apply it to WIZ or CAANOO, use the "`__attribute__`" keyword for memory packing.

2. DGE Base

- Library: CAANOO - libdge20.so, WIZ - libdge20w.so, Windows - dge20_.lib
- Consists of system and graphics

2.1 Basic Components

2.1.1 System

- DGE Type: Defines the basic data types → DGE_Type.h
- DGE System: Responsible for initialization and for creating a program to use the SDK version, date of build, and hardware. → DGE_System.h
- DGE Input: Includes the use of an analog joystick, touchscreen, buttons, and virtual mapping of a joystick in an 8-directional digital mode of DGE 1.0 → DGE_Input20.h
- DGE Time: Date, hours, timer → DGE_Time.h
- DGE Util: Re-organize functions related to C-runtime which are not compatible with a platform → DGE_Util.h
- DGE Gsensor: Gravity sensor → DGE_Gsensor.h
- DGE Haptic: Vibration motor → DGE_Haptic.h
- DGE Math: Implement processing functions and structures such as vector, matrix or color. → DGE_Math.h

2.1.2 Graphic

- IDGE_Device: Objectifies the graphics pipeline of the OpenGL. → DGE_Base.h
- IDGE_Sprite: Objects for processing 2D graphics → DGE_Base.h
- IDGE_Image: Analyzes image files and accumulates pixels in the memory → DGE_Base.h
- IDGE_Texture: Creates texture with pixels of image files and memory buffer → DGE_Base.h
- IDGE_VB: Vertex Buffer Object → DGE_Base.h
- IDGE_IB: Index Buffer Object → DGE_Base.h

2.2 Function

2.2.1 System

DGE_Version: Returns the SDK version information and the date of build.

Input Argument:

- DWORD* dVer: Version information Major (Hi-word), Minor (Low-word)
- DWORD* dDate: Built-date: Prints hex in character strings.

DGE_Init: Initializes the system to execute a program and is the first to be called up after the main() function at the beginning of a game program.

DGE_Close: The resources obtained to execute a program should be returned and called up immediately before the program is closed.

- To return to the menu, this function should be called up.

DGE_CreateWindow: Creates game screen. If successful, the Microsoft Windows Handle is returned. In the event of failure, NULL is returned.

Input Argument:

- pDGE_MsgProc: Message procedure function for Microsoft Windows programs. NULL will be delivered under CAANOO.
- char* ClassName: Refers to the name of the program class.
- INT ScreenW: Screen width: Enter "0" value to create a screen width of 320 for CAANOO and 640 for Windows.
- INT ScreenH: Screen height: Enter "0" value to create a screen height of 240 for CAANOO and 480 for Windows.

DGE_DestroyWindow: Releases Microsoft Windows and returns resources.

Input Argument: DGE_HANDLE: Microsoft Windows handle to be released

DGE_SetMainRunFunc: Message procedure function for Microsoft Windows programs CAANOO (x), Windows (O)

Input Argument: Callback function pointer for processing messages

DGE_Run: Main processing function As for Windows Emulator, dispatches a message is obtained from the message procedure function that has been set by the DGE_SetMain-RunFunc function.

Return value: Returns a negative value when a program enters closing mode.

DGE_WindowSize: Brings up the set window size.

Input Value: INT* pWidth: An address to indicate the window width INT* pHeight: An address to indicate the window height

<Creation of Program>

```
#if defined(_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
// The message procedure function is indispensable for Windows Emulator.
LRESULT DGE_WINAPI WndProc(DGE_HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    #if defined(_WIN32)
        switch (uMsg)
        {
            ...
        }
        return DefWindowProc(hWnd, uMsg, wParam, lParam);
    #else
        // DGE processes nothing. DGE only customizes a form with Microsoft Windows.
        return DGE_OK;
    #endif
}

INT DGE_CALLBACK Run(void*)
{
    #if defined(_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;

    if(DGE_FAILED(DGE_Init()))        // Initializes DGE.
        goto End;

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

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

    DGE_SetMainRunFunc(Run);        // Sets Main Loop function.

    while(1)                        // Executes program.

```

```

    {
        if(DGE_FAILED(DGE_Run()))
            break;
    }

    DGE_DestroyWindow(hWnd);          // Releases Windows.
End:
    DGE_Close();                      // Releases DGE (return to Menu).
    return 0;
}

```

2.2.2 Input20

The use of an analog joystick, touchscreen, and button, and the virtual mapping of a joystick in an 8-directional digital mode of DGE 1.0 are included. The "read" value for a button or an 8-directional digital mode comes as "1" or "0". The EDGE_INPUT of a header file is defined to use an input process for an application program.

DGE_Input[Open|Close]: Initialization/release for using the input device

Input Argument: HWND: Only Microsoft Windows Emulator requires this.

Return value: Success: DGE_OK, Failure: (-) value

```

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

```

DGE_InputReadAjs: Read the x and y value of the analog joystick device.

Input Argument: Address of the structure variable of DGE_INPUT_AJS

Return value: If function process is successful, it returns DGE_OK. If failure occurs, it returns a negative value. Most of the causes of failure occurs while the values are being read from the device.

DGE_INPUT_AJS structure: Data structure to read the value of the analog joystick.

- int st: Returns [0, 7] for the 8-directional joystick in digital mode. Returns "-1" if none exists.
- int x, y: x and y value of the joystick

```

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: Reads the x and y value of the touchscreen.

Input Argument: Address of the structure variable of DGE_INPUT_TS

Return value: The number of events reflects function processing. If the event number is "0", it means nothing has been processed internally.

Returns the failure value (negative value) if there is an error in the device.

DGE_INPUT_TS structure: Data structure to read the touch screen value.

- int st: Currently in the process of reading the x and y value.
 - Reading under way: DGEINPUT_READ_WAIT
 - Reading complete: DGEINPUT_READ_COMPLETE
- int x, y: x and y value of joystick

```

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

if(0<nEvent)
{
    m_vTsCur[0] = (Float)tsTemp.x;
    m_vTsCur[1] = (Float)tsTemp.y;
}

```

```

        m_TsbCur[0] = tsTemp.st;           // Press and hold
    }

```

DGE_InputReadKey: Reads button.

Input Argument: Array of unsigned characters of 32 bytes or more.

Return value: DGE_OK for successful function handling, (-) value for failed handling.

```

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

```

2.2.3 Time

This file supports eabi associated with time and also enables cross compiling of Windows.

DGE_tzset: A function for setting the environment related to time. Applicable for the txset() function.

Return value: Success - DGE_OK. Failure - Negative value

DGE_GetTickCount: Returns time after booting the system. Corresponds to the Windows GetTickCount function.

Return value: Returns DWORD-type millisecond. 1 sec = 1000 ms

DGE_Sleep: Sets the program to standby mode as long as the time of the given value allows.

Input Argument: DWORD-type millisecond

DGE_DateandTime: Returns both date and current time at once.

Input Argument: Character-type array of 128-byte is required.

<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

This file defines cross compiling and frequently-used functions or macros.

DgeMem_Allock: A function that allocates dynamic memory. Allocates the buffer internally through the malloc function and initializes it through the memset function.

Input Argument:

- int BuffSize: Buffer size
- int Count: Number of buffers
- int InitSetVal: Initialization value

DgeMem_Free: Releases the buffer allocated as dynamic through the DgeMem_Allock function. Calls up the free function and sets it as NULL.

free(*v) and *v = NULL;

Input Argument: void** v: Address of buffer

DgeStr_tolower: Converts an English character to upper case.

DgeStr_toupper: Converts an English character to lower case.

DgeStr_Stricmp: Compares two character strings without being case-sensitive. Corresponds to the _stricmp function.

DgeStr_Strnicmp: Compares two character strings as much as length allows without being case-sensitive. Corresponds to the _strnicmp function.

DgeStr_SplitPath: Divides the full filename into driver, folder, filename and extension. Corresponds to the _splitpath function.

DgeFile_FindFirst: A function for searching a file or folder. Corresponds to the _findfirst function.

DgeFile_FindNext: A function for searching a file or folder. Corresponds to the _findnext function.

DgeFile_FindClose: A function for searching a file or folder. Corresponds to the _findClose function.

DgeFile_GatherAllFiles: Every file or folder that comes under the sub-folder of the Input Argument.

Input Argument:

- TDGE_FINF** pOut: Data to save the name of a file and a folder. The structure of D-Linked List.
- const char* sStartRoot: Name of the first folder to search

DgeFile_GatherFree: Releases the linked list used for searching files.

Input Argument: - TDGE_FINF**: Start address for the linked list

DgeFile_mkdir: Create a folder.

Input Argument:

- const char* sFolder: Folder name
- unsigned int mod: Usage and access control mode

2.2.4 G-sensor

This is a header file that contains functions which control the gravity processing sensor. The gravity sensor reads the value that is converted into the gravity value by setting the width of the screen as "x", the height or length of the screen as "y", and perpendicular to the screen face as "z". It returns the real number value within the range of [-, +].

As the sensor measures values only in the direction that gravity works, x and y will be "0" and z will be positive (+) if the screen is laid in a horizontal state. If you put the screen facing the front with the device width horizontal, y will be positive (+), the x and z will be "0". Similarly, if the screen is facing the front with the device length horizontal, x will be positive (+) or negative(-), and y and z will be "0".

DGE_Gsensor[Open|Close]: Initializes/releases the use of G-sensor.

Return value: Returns DGE_OK if hardware control is achieved. Returns a negative value if not achieved.

DGE_GsensorSetEnv: Environment setting to use the G-sensor

Input Argument: Structure variable of the DGE_GSENSOR_ENV

Return value: DGE_OK for successful function handling, (-) value for failed handling.

DGE_GSENSOR_ENV: Data structure to set the environment of the G-sensor.

- int Range: Range of gravity Select one of 2 G, 4 G, or 8 G.
- int Resolution: Select the resolution from either 8-bit or 12-bit.
- int TapHz: Measure a simple form of impact such as tapping. Select one of 50, 100, 200, or 400

Hz.

- int TiltHz: Measure screen tilting. Select one of 1, 6, 12, or 50 Hz.

- int TiltTimer: Select a constant number for measuring the time for tilting within the range of 0 to 255.

DGE_GsensorRead: Read the absolute value of the x, y, and z axis of the G-sensor device.

Input Argument: Structure variable of the DGE_GSENSOR_VAL

Return value: DGE_OK for successful reading, negative (-) value for failed reading.

DGE_GSENSOR_VAL: Data structure to read the value of G-sensor.

Gravity value to 3 axes of int x, y, and z. 0xFFFF if the value is unavailable.

int TapMod: Measures tapping in +/-.

int TapDir: Direction of tapping.

<G-sensor>

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

...

DGE_GSENSOR_ENV g_GsEnv;           // Environment of G-sensor
DGE_GSENSOR_VAL g_GsRead;          // Response of G-sensor

...

if(DGE_FAILED(DGE_GsensorOpen()))   // Initializes the usage of G-sensor.
    return -1;

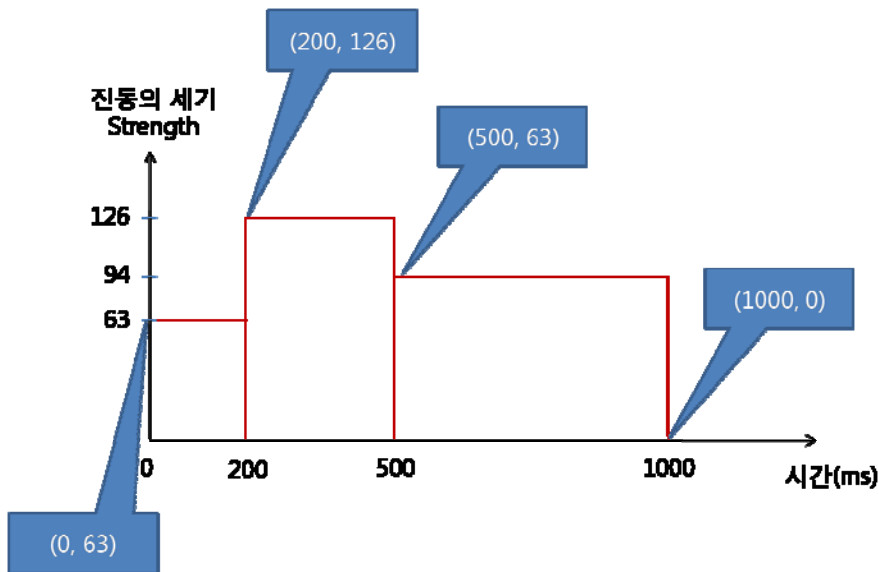
// Run Main Loop
{
    hr = DGE_GsensorRead(&g_GsRead); // Read G-sensor
    if(-1 != hr)
        printf("%3d %3d %3d\n", g_GsRead.x, g_GsRead.y, g_GsRead.z);
}

...

DGE_GsensorClose();                 // Process of G-sensor termination handling.
```

2.2.5 Haptic

CAANOO is installed with two vibration motors (left and right) and the strength and duration of vibrations can be controlled.



<Setting the Strength and Duration of Vibration>

When the user creates a program where vibration varies continuously, this file sets the strength and duration of vibration as a single pair and delivers this to the vibration motor.

DGE_Haptic[Open|Close]: Initializes/releases the use of vibration motor.

Return value: Returns DGE_OK if hardware control is achieved. Returns a negative value if not achieved.

DGE_HapticStatus: State of the vibration motor.

Return value: busy = VIB_STATUS_BUSY, stop = VIB_STATUS_IDLE

DGE_HapticEnable: Sets whether the vibration motor is used or not.

Input Argument: Constant number Enable=TRUE, Disable = FALSE

Return value: Success - DGE_OK. Failure - Negative value

DGE_HapticPlay: Initiates the motion of vibration motor.

Input Argument:

- int time_line: Duration
- int strength: Vibration Strength

DGE_HapticStop: Stops the motion of vibration motor.

DGE_HapticPlayPattern: Sets the execution pattern of the vibration motor.

Input Argument: Array of the pattern_data_t structure

pattern_data_t: Data structure that consists of the vibration patterns

- short time_line: Duration of vibration
- short vib_strength: Strength of vibration

<Vibration Motor>

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

// Default operating time of vibration (Unit: 1/1000 sec)
const int basisActivateTime = 1000;

// Space for recording/storing vibration motions
vibration_act_t vibrationRecord[DGE_VIB_MAX_ACTION];

// Time of recording vibration motion (Unit: 1/1000 sec, absolute time)
int recordTime = 0;
// Position of recording vibration motions
int recordPosition = 0;

enum VIBRATION_POWER // Vibration strength
{
    VIBRATION_POWER_ZERO = 0,
    VIBRATION_POWER_WEAK = 40,
    VIBRATION_POWER_MIDDLE = 80,
    VIBRATION_POWER_STRONG = DGE_VIB_STRENGTH_MAX,
};

void VibrationStop() // Stops the previous vibration motion.
{
    // Checks the current state of vibration device.
    if(DGE_VIB_STATUS_BUSY == DGE_HapticStatus())
        DGE_HapticStop(); // Stops Vibration
}

void InitializeVibrationRecord() // Initializes the records of vibration motion.
{
    for(int count = 0; count != DGE_VIB_MAX_ACTION; ++count)
```

```

{
    // Initializes the records of vibration motion.
    vibrationRecord[count].time_line = 0;
    vibrationRecord[count].vib_strength = VIBRATION_POWER_ZERO;
}

// Initializes the recording time of vibration motion.
recordTime = 0;
// Initializes the recording position of vibration motion.
recordPosition = 0;
}

// Records vibration motion (creates patterns).
void RecordVibrationPlay(int playTime, int strength)
{
    // Checks the range of recording position of vibration motion.
    // (Maximum number, use termination handling - 1, use array index - 1)
    if((0 > recordPosition) || ( DGE_VIB_MAX_ACTION - 2 < recordPosition))
        return; // Beyond the allowed range (Unable to record)

    // Within the allowed range (Able to record)
    // Records vibration motion.
    vibrationRecord[recordPosition].time_line = (short)recordTime;
    vibrationRecord[recordPosition].vib_strength = (short)strength;

    // Sets the recording time of the following vibration motion.
    recordTime += playTime;
    // Sets the recording position of the following vibration motion.
    ++recordPosition;
}

// Plays the records of vibration motion.
void PlayVibrationRecord()
{
    // Checks the range of recording position of vibration motion.
    if(0 >= recordPosition)
        return; // Beyond the allowed range (Unable to play, no content)
}

```

```

VibrationStop();          // Stops the previous vibration motion.

// Processes the records on vibration motion as terminated.
vibrationRecord[recordPosition].time_line = (short)recordTime;
vibrationRecord[recordPosition].vib_strength = (short)VIBRATION_POWER_ZERO;

// Creates patterns.
pattern_data_t vibrationPattern;
// Sets the number of records on vibration motion (Use array index +1).
vibrationPattern.act_number = recordPosition + 1;

// Repeats as much as the space available for storing records on vibration motion.
for(int count = 0; count != DGE_VIB_MAX_ACTION; ++count)
{
    // Copies the information of records on vibration motion.
    vibrationPattern.vib_act_array[count].time_line=
        vibrationRecord[count].time_line;
    vibrationPattern.vib_act_array[count].vib_strength=
        vibrationRecord[count].vib_strength;
}

// Plays the record of vibration motion.
DGE_HapticPlayPattern(&vibrationPattern);
}

int main(int argc, char* argv[])
{
    ...
    DGE_HapticOpen();          // Initializes vibration device.
    DGE_HapticEnable(1);       // Activates vibration device
    InitializeVibrationRecord(); // Initializes the records of vibration motion.
    ...
    // Main Loop
    {
        if(isRecord)           // Checks if vibration motion has been recorded.
        {
            VibrationStop(); // Stops the previous vibration motion.

```

```

        // Plays vibration motion.
        DGE_HapticPlay(basisActivateTime, vibrationStrength);

        // Records vibration motion.
        RecordVibrationPlay(basisActivateTime, vibrationStrength);

        // Edits information.
        isChangeInformation = true;
    }

}

DGE_HapticClose();    // Closes vibration device.
...
}

```

2.3 Graphics

2.3.1 IDGE_Device

This is an interface made for saving and using the graphic pipeline of the OpenGL and the major instances of the program. It creates objects through the DGE_CreateDevice function. It releases instances through the DGE_DestroyDevice function.

<Creation of Device>

```

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

DGE_HANDLE      m_hwnd;                // Windows Handle
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();                      // Initializes DGE.
    DGE_SetMainRunFunc(Run); // Sets main procedure function.

    // Creates window.
    m_hwnd = DGE_CreateWindow(WndProc, "DGE_Tutorial", 320, 240);
    if(NULL == m_hwnd)
    {
        printf("DGE_CreateWindow Failed\n");
        goto End;
    }

    // Creates device.
    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);     // Releases device.
    DGE_DestroyWindow(m_hwnd);      // Releases windows.

```

```

        DGE_Close();                // Releases DGE.
        return 0;
    }

```

2.3.1.1 Function for Setting the Rendering Machine

SetTransform: A function for setting the Transformation Matrix of the vertex

To organize the scene from the graphic pipeline, the vertex position first goes through the world transformation, followed by the view transformation at the space for the camera, and then regular transformation for each device. At this point, the graphic pipeline adopts the 4 x 4 matrix to perform scaling, rotation and translation at the same time. Meanwhile, the OpenGL goes through the model-view transformation that lumps the world transformation and the view transformation to process them at the same time. DGE divides the world, view and projection transformation so as to improve the flexibility of Direct3D and OpenGL ES 2.0 or higher.

A function for the respective transformations is used by calling up the SetTransform function. Insert the type of transformation into the first factor of this function. Deliver 16 floating-point arrays to the following factor. At DGE, the operation of the matrix sets the vertex position to the left and the multiplication proceeds from left to right for transformation. Therefore, it takes the opposite direction from common mathematics where multiplication proceeds from right to left.

As for the operation of the matrix, the DGEMATRIX structure is designed to be highly compatible with Direct3D and OpenGL without the need for significant revision. Thus, it is more convenient to use this structure.

SetTransformIdentity: Sets the transformation array of the vertex as the unit array.

The world transformation is the most commonly used. If the array of the world transformation of the pipeline is revised by the set transform function, the array of the world transformation should be initialized through the set transform identity function after drawing a scene.

BeginScene/EndScene: A function that indicates the start and end of rendering status of the given scene.

Draw function should be called up by each 2D or 3D object between these functions for normal operation .

<Set World Matrix>

```

    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: A function that sets the state value of the virtual rendering machine.

The state of the rendering machine is defined at the EDGE_RENDERINGSTATE. The respective state values are defined at the EDGE_RENDERINGSTATUS. The on/off feature of the state machine under the OpenGL should be set through the glEnable/glDisable function. This can be processed through the SetRenderState function. Still, you can use the SetStatusEnable/SetStatusDisable function for the same purpose.

<Setting State Machine Value>

```

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...(): A function for setting the sampling of texture and multi-texture processing.

SetTextureAddressS: Sets the address mode of the S-direction sampling. DGE_TS_REPEAT, DGE_TS_CLAMP

SetTextureAddressT: Sets the address mode of the T-direction sampling. DGE_TS_REPEAT, DGE_TS_CLAMP

SetTextureFilterMin: Sets filtering regarding the minification of texture.

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

SetTextureFilterMag: Sets filtering regarding the magnitude of the texture: Nearest, Linear.

SetTextureStateOp: Sets the mixing method of colors by stages in multi-texture processing.

- MODULATE, DECAL, ADD, ADD_SIGNED, SUBSTRACT

SetTextureEnv: glTexEnvf function of the OpenGL

<Setting the State of Multi-texture Processing>

```

m_pDev->SetTexture(0, m_pTex0); // Sets texture no.0.
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);           // Sets texture no.1.
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 Functions for Setting Rendering Data

SetFVF: Sets the vertex format of the vertex stream which is an input stream.

SetTexture: Sets the texture at the multi-texture phase. Supports up to two phases altogether.

SetColor: Sets the color of diffuse in the process of handling the rendering pipeline. Corresponds to the OpenGL glColor[f|v] function.

SetStreamSource: Connects the VBO (Vertex Buffer Object) with the pipeline.

SetIndices: Connects the IBO (Index Buffer Object) with the pipeline.

2.3.1.3 Output Functions

DrawPrimitive[f|i]: Outputs VBO with no index. (f: Floating-point data, i: Fixed-point Data)

DrawIndexedPrimitive[f|i]: Combines IBO and VBO to output.

DrawPrimitiveUP[f|i]: Outputs user-defined data (user pointer) in the space of stack and heap.

DrawIndexedPrimitiveUP[f|i]: Outputs the index and the vertex data designated by a user in a space of stack and heap: Initializes the color buffer, depth buffer, etc., of the frame buffer to the set value. Corresponds to glClear.

Present: Outputs the content of the frame buffer to the screen. Corresponds to the eglSwapBuffers and glFlush functions.

2.3.1.4 Setting a Variable of the Application Programs

SetMainApp: Stores the main data or the class instance of an application program. The user handles the process.

SetSprite: Stores this instance when it is made.

SetInput: Stores this instance when it is being created.

SetNetwork: Stores the main instance of the network. The user handles the process.

SetUserData[1...4]: Stores the pointer of the main data designated by a user.

Variables of the application programs stored in the IDGE_Device are instantly available through the Get...() function, only when you have the IDGE_Device instance.

2.3.2 IDGE_Sprite

Handling of 2D graphics. Creates an instance through the DGE_CreateSprite function. Releases an instance with the DGE_DestroySprite function. Every 2D graphic should be printed between the Begin/End functions.

Begin: Sets the initial state of the 2D graphic output. 2D graphic output internally initializes and stores the matrixes of the world, view and projection transformation.

End: Restores the values stored by the begin function.

Draw: Prints 2D graphics with no scaling or rotational transformation. More accelerated output speed than the DrawEx.

DrawEx: Prints 2D graphics with scaling, rotation, translation transformation. Prints the value of the scaling transformation as symmetrical to the horizontal and vertical axis if the value is negative.

2.3.3 IDGE_Image

A pixel data object of a file or buffer. Supports the uncompressed BMP, TGA and PNG format. JPG format is available only in the Windows OS. Creates the GPT file if Windows Emulator is executed. GPT file converts a 32-bit pixel into a 16-bit color through an alpha channel and a color key. If there is no color key, the color format is R5G6B5. If there is a color key, the format is R5G5G5A1. If an alpha channel exists, the format is R4G4B4A4. Color conversion is processed through the DGE_Convert32to16Color function. Creates an object through the DGE_CreateImage function. Releases an instance through the DGE_DestroyImage function.

GetRscType: Returns the resource type of a pixel. BMP, TGA, PNG, GPT

GetFormat: Returns the pixel format of the OpenGL type. GL_RGB, RGBA, LUMINANCE, LUMINANCE_ALPHA

GetType: Returns the pixel type of the OpenGL type. UNSIGNED_SHORT565, UNSIGNED_SHORT4444, UNSIGNED_SHORT5551

GetImgW: Returns the horizontal width of an image. Windows Emulator consists of a multiplier of two.

GetImgH: Returns the vertical width of an image. Windows Emulator consists of a multiplier of two.

GetImgD: Returns the depth byte of a pixel. Colors in 1 = 8-bit, 2 = 16-bit, 3 = 24-bit, 4 = 32-bit.

GetPixel: Pixel buffer

SaveToFile: Stores pixels in the TGA and GPT file format.

2.3.4 IDGE_Texture

Texture object Requires pixel data. Creates texture in a file or a pixel buffer through the DGE_CreateTexture function. Uses the IDGE_Image object internally if the texture object is created from an image file. Texture has the owner to prevent duplicate creations. Releasing an instance through the DGE_DestroyTexture function is possible only when the owner does not exist. The user sets the owner through the SetOwner function. Delivery of NULL releases the owner.

In the event that texture is created via the DGE_CreateTexture function in the Windows system, create a 16-bit image file with a "gptf" extension which adopts the format appropriate to given device in accordance with the color key and the image channel.

Number of Channels	User-designated Color Key	"gptf" File Pixel Format
3 (24-bit)	0x00000000 (not designated)	GL_UNSIGNED_SHORT_5_6_5
	Other than 0x00000000 value	GL_UNSIGNED_SHORT_5_5_5_1
4 (32-bit)	Not applied to Color Key.	GL_UNSIGNED_SHORT_4_4_4_4

The device first starts to look up the "gptf" file format. If there is none, it looks up the "png", "tga" or "bmp" file formats to read image files. Then, it creates a 16-bit texture in "gptf" file format. As the palette of bmp files converts pixels into 32-bit, it is recommended that 24-bit or 32-bit images are used.

As for file formats other than the above, the user should organize them in a way that corresponds to the OpenGL ES format.

Precautions: It is best to build the size of an image in multiples of 2. The width and height of an image should be built in multiples of 4.

GetName: A name exists to manage the IDGE_Texture. Generally, the filename is used as it is.

GetId: Returns OpenGL ID.

GetFormat: Returns the pixel format.

GetType: Returns the pixel type.

GetImgW: Returns the image width.

GetImgH: Returns the image height.

GetImgD: Returns the image depth.

GetImgC: Returns the set color key.

SetOwner: Sets the owner of the texture. Delivery of NULL releases the owner.

GetOwner: Returns owner.

BindTexture: A function to be compatible with the BindTexture function of the OpenGL. Sets multi-texture,

filtering and addressing.

<2D Rendering>

```
#include <DGE_Base.h>

PDGE_SPRITE      m_pSprite= NULL;    // 2D Sprite
vector<PDGE_TEXTURE> m_vTex;          // Texture List

...
// Creates Sprite.
if(DGE_FAILED(DGE_CreateSprite(NULL, &m_pSprite, m_pDev)))
    return -1;

// Creates texture.
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...

    ...
    // Releases texture.
    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);                 // Releases Sprite.

```

2.3.5 IDGE_VB

This is the interface for the Vertex Buffer Object. It creates an object through the DGE_CreateVB function. It releases an object through the DGE_DestroyVB function. Owner exists internally as it is common that the vertex buffer is used jointly. An object is not released through the DGE_DestroyVB when there is no owner.

Fill: Renews the data of the vertex buffer.

Bind: Binds the VBO to the pipeline. Calls the glBindBuffer function from inside.

SetOwner: Sets owner. Delivery of NULL releases the owner. Default Owner = NULL

2.3.6 IDGE_IB

This is the interface for the Index Buffer Object. It creates an object through the DGE_CreateIB. It releases an object through the DGE_DestroyIB function. Owner exists internally as it is common that the index buffer is used jointly. An object is released through the DGE_DestroyIB when there is no owner.

Fill: Renews the data of the index buffer.

Bind: Binds the IBO to the pipeline. Calls the glBindBuffer function from inside.

SetOwner: Sets owner. Delivery of NULL releases the owner. Default Owner = NULL

<VBO, IBO Rendering>

```
PDGE_VB      m_pVB;
PDGE_IB      m_pIB[6];
...
// Creates vertex data in the system memory.
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);
// Creates DGE_VBO and fill up data.
if(DGE_FAILED(DGE_CreateVB(NULL, &m_pVB, m_pDev, &dFVF, vertices, &nSize)))
    return -1;

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

```

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

...

// Rendering
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->DrawIndexedPrimitivef(DGE_PT_TRIANGLE_STRIP, 4);

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

// Releases VBO, IBO.
DGE_DestroyVB(&m_pVB);           // Releases VBO.

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

```

3. DGX

- Library: CAANOO - libdgx20.so, WIZ - libdgx20w.so, Windows - dgx20_.lib
- Comprises Input, Media, Sound, Network, and 3D Utility. Refer to the OpenAL for 3D sounds.

3.1 IDGX_Input

Classify the low-class DGE input. Handles a button, analog joystick and touch screen.

Update: Handles an event in the order of analog joystick, button and touchscreen.

GetKeyMap: Returns an address of an array that has stored a key event.

GetTsbMap: Returns an address of an array that has stored an event of the touchscreen.

KeyDown: Returns the Down state if the previous state == 0 and current state == 1.

KeyUp: Returns the Up state if the previous state == 1 and current state == 0 for the Up state.

KeyPress: Returns the Press state if both the current state and previous state == 1.

KeyState: Returns the button state. Down = 1, Up = 2, Press = 3

TsbDown: Returns the Down state of the touchscreen. The condition for the state is the same as that for a button.

TsbUp: Returns the Up state of the touchscreen.

TsbPress: Returns the Press state of the touchscreen.

TsbState: Returns the state of the touchscreen.

GetTsbPos: Returns the position of the touchscreen. FLOAT x 3 is required.

GetTsbDelta: Returns the value of the relative coordinates of the touchscreen. FLOAT x 3 is required.

IsTsbMove: Returns the motion of the touchscreen.

SetCurKey: Sets the event value of a button.

SetCurTsb: Sets the current state of the touchscreen.

SetTsbPos: Sets the current position of the touchscreen.

IsInRect: Estimates whether the touch screen is positioned within the range of a given rectangle.

Reset: Initializes every value of the DGX_Input as "0".

SetActive: Sets activation or deactivation. Returns the return value to "0" for Down, Up and Press without changing the internal values. Default: Active

GetActive: Returns the value of activation or deactivation.

<IDGX_Input>

#include <DGE_Input20.h>

```

#include <DGX_Input.h>

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

// Creates device.
if(DGE_FAILED(DGE_CreateDevice(NULL, &m_pDev, m_hwnd, NULL)))
    return -1;

// Creates input.
if(DGE_FAILED(DGE_CreateDevice(NULL, &m_pDev, m_hwnd, NULL)))
    return -1;

...

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

// Brings the data of the input touchscreen (position and state).
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);

// Button
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]);
}

...

// Releases input.
DGX_DestroyInput(&m_pInput);

// Releases device.
DGE_DestroyDevice(&m_pDev);

```

3.2 Media

Controls peripheral devices such as recording, volume control, USB, etc.

3.2.1 Mixer

Adjusts the volume of the left and right side.

DGX_MixerOpen: Initializes the usage of the mixer device.

DGX_MixerClose: Releases the usage of the mixer device.

DGX_MixerGetVolumeRange: Obtains the minimum and maximum value of the mixer device (0 to 127).

DGX_MixerSetVolume: Sets the volume (overall volume) of the mixer device.

DGX_MixerGetVolume: Obtains the volume size of the mixer device.

3.2.2 Microphone

Microphone recording

DGX_RecordingSound: Records sounds. To record sounds, storage of 1024 KB is required.

3.2.3 Miscellaneous

DGX_Battery: Battery indicator. High=1, Middle=2, Low=3, Empty=4

3.3 Network

Processes TCP/IP communications. The I/O model uses the Select method.

DGX_NetDeviceOpen: Obtains authority to use the network device.

DGX_NetDeviceClose: Returns the network device after use.

DGX_SocketAddr: Receives the IP and port input and sets the socket address structure.

DGX_SocketClose: Releases the created socket.

DGX_SocketTcpCreate: Creates the socket for TCP.

DGX_SocketUdpCreate: Creates the socket for UDP.

DGX_SocketConnect: Client accesses the server.

DGX_SocketBind: Binds the socket and the structure variables of the socket address.

DGX_SocketListen: Listens to the server Socket

DGX_SocketAccept: Stands by for client access and returns the socket and the structure of the socket address.

DGX_SocketSend: Transmits data

DGX_SocketRecv: Receives data

DGX_SocketSelect: Handles the input/output of the network as multiplexing. (Select method.)

DGX_SocketNonBlocking: Creates a socket as non-blocking.

DGX_SocketNaggleOff: Nagle off when TRUE is delivered.

DGX_LocalIpAddress: Returns the IP address of the device.

3.4 IDGX_Sound

This is an object for playing a wave file based on OpenAL. Sound device must be created before creating the IDGX_Sound instance through the DGX_CreateSound function. The instance is released through the DGX_DestroySound function.

Precautions: The sound buffer of DGE OpenAL is set in 16-bit stereo 22 kHz, and supports PCM Wave file formats exclusively. Therefore, it is best to build a sound file in PCM 16-bit 22 kHz mono/stereo format.

DGX_SoundDeviceOpen: Creates sound device.

DGX_SoundDeviceClose: Releases sound device.

Query: A function for controlling the command or the state value of a sound object.

GetType: A sound resource type. Wave or midi file.

Play: Plays sound file.

Stop: Stops playing file.

Reset: Rewinds the sound play position to the beginning.

Pause: Temporarily interrupts sound play.

SetVolume: Sets volumes. Default volume = 1.0F

SetRepeat: Number of repetitions. Repeat once or unlimited repetition = DGX_SND_INFINITE

[Get|Set]Status: Returns states such as stop, play, pause, reset, etc.

<Sound Play>

```
#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

// Creates sound device.
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",
};

// Creates sound object.
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);      // Sets volume.
    m_pSnd[i]->SetRepeat(1);      // Number of repetitions
}

...

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

...

// Releases sound object.
for(i=0; i<MAX_SOUND; ++i)
    DGX_DestroySound(&m_pSnd[i]);

```

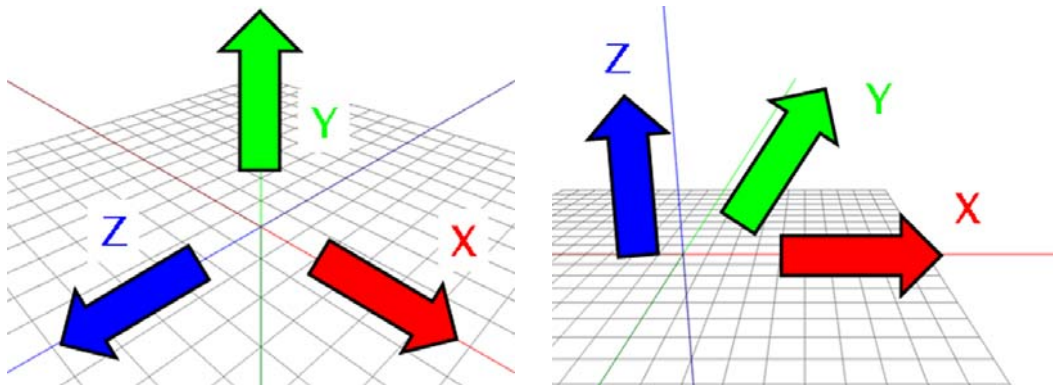
```
// Releases sound device.
DGX_SoundDeviceClose();
```

3.5 3D Utility

Utility that regards a camera, model (mesh), simple figure, grids and so on for 3D programs

3.5.1 IDGX_Camera

This is the interface for first-person camera. While the OpenGL uses right-hand coordinates where the y axis is directed upward on a plane of the x and z axis, the IDGX_Camera uses right-hand coordinates where the z axis is directed upward on a plane of the x and y axis.



<Coordinate Axis of OpenGL and DGE Camera>

SetupViewRH function of the DGXMATRIX is designed to create the Viewing Matrix from IDGX_Camera. For your information, SetupViewLH function of the DGXMATRIX creates the Viewing Matrix from the left-hand coordinates of DirectX.

The camera class aims to create the view matrix of the view transformation needed for a program and the projection matrix of the regular transformation, and then set these to the pipeline.

Creates an object through the DGX_CreateCamera. In doing so, the IDGE_Device is a prerequisite. An object is released through the DGX_DestroyCamera function.

The value of the y and z axis should be exchanged if the data used for the Microsoft Direct3D are adopted.

Update: Renews camera data. Calculates the axis of camera, frustum, etc.

FrameMove: Calls the Update function and sets the view matrix and projection matrix in the 3D pipeline.

SetTransformViv: Sets the View Matrix in the 3D pipeline.

If the view matrix is to be created in a space where the z axis on the x-y plane is directed upwards

in the OpenGL environment, row 2 and 3 of the matrix should be exchanged and row 14, 24, and 34 should be reversed.

<Setting View Matrix to be Applied to Pipeline>

```
DGXMATRIX mtViw = m_mtViw;           // Camera View Matrix
mtViw._12 = m_mtViw._13; mtViw._13 = -m_mtViw._12; mtViw._14 = -m_mtViw._14;
mtViw._22 = m_mtViw._23; mtViw._23 = -m_mtViw._22; mtViw._24 = -m_mtViw._24;
mtViw._32 = m_mtViw._33; mtViw._33 = -m_mtViw._32; mtViw._34 = -m_mtViw._34;
mtViw._42 = m_mtViw._43; mtViw._43 = -m_mtViw._42; mtViw._44 = +m_mtViw._44;
```

SetTransformPrj: Sets the projection matrix in the pipeline.

```
m_pDev->SetTransform(DGE_TS_PROJ, &m_mtPrj);
```

SetupParamView: Sets the view matrix of the camera.

<View Matrix>

```
DGXVECTOR3 vcY = normalize(LookAt - Eye);           // y axis of camera
DGXVECTOR3 vcX = normalize(cross(vcY, vcUp));        // x axis of camera
DGXVECTOR3 vcZ = cross(vcX, vcY);                   // z axis of camera

DGXMATRIX mtViw;
Identity(mtViw); // Sets as identity matrix.
mtViw._11 = vcX.x;      mtViw._12 = vcY.x;      mtViw._13 = vcZ.x;
mtViw._21 = vcX.y;      mtViw._22 = vcY.y;      mtViw._23 = vcZ.y;
mtViw._31 = vcX.z;      mtViw._32 = vcY.z;      mtViw._33 = vcZ.z;
mtViw._41 = - *pEye * vcX; mtViw._42 = - *pEye * vcY; mtViw._43 = - *pEye * vcZ;
```

SetupParamProj: Sets the projection matrix of camera.

<Projection Matrix>

```
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); // Sets as identity matrix.
```

```

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;

```

SetupMatrixViv: Copies the external matrix to the view matrix of the camera.

SetupMatrixPrj: Copies the external matrix to the projection matrix of the camera.

GetAxis[X|Y|Z]: Returns the [X|Y|Z] axis of the camera.

MoveSideward: Moves a first-person camera back and forth.

MoveForward: Moves back and forth of a first-person camera.

RotateYaw: Rotates centered on the z axis.

RotatePitch: Rotates centered on the x axis.

<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

// Creates camera and sets view, projection Arguments.
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();

    // Sets camera move and rotation speed.
    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();
    ...
    // Sets rendering state.
    m_pDev->BeginScene();
    m_pDev->Clear(DGE_CLEAR_COLOR_BUFFER | DGE_CLEAR_DEPTH_BUFFER);

    // Reflects the view matrix and, projection matrix of the camera to the rendering
    pipeline.

```

```

m_pCam->SetTransformPrj();
m_pCam->SetTransformViw();

// Rendering
...
m_pDev->EndScene();
...
DGX_DestroyCamera(&m_pCam);           // Releases camera object.

```

3.5.2 IDGX_Entity

Prints simple figures, index lines of the x, y, and z axis, and grids for scaling in a 3D test program.

Creates an object through the DGX_CreateEntity function. Releases an object through the DGX_DestroyEntity function.

FrameMove: Update Polygon Data.

Render: 3D data rendering.

Query: A function for setting data or bringing up values.

<XYZ Axis and Grid Creation>

```

#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;
...
// Rendering
m_pDev->BeginScene();
...
m_pXYZ->Render();
m_pGrid->Render();

m_pDev->EndScene();

```

```

...
// Release
DGX_DestroyEntity(&m_pXYZ);
DGX_DestroyEntity(&m_pGrid);

```

3.5.3 IDGX_Mdl

Interface for 3D Animation Model. Supports ASE of 3D Max and GPM of GPH. Creates objects by the DGX_CreateMdl function. Releases objects by DGX_DestroyMdl

FrameMove: Renews data related to the animation inside the model.

Render: Renders renewed data.

SetAttrib: Sets the data of a model. The command is defined at EDGX_MDL.

GetAttrib: Brings up data from inside the model. The command is defined at EDGX_MDL.

```

<IDGX_Mdl>

#include <DGX_Mdl.h>

PDGX_MDL m_pMdlOrg1;           // Original 1
PDGX_MDL m_pMdlIns1;           // Copy Instance 1
PDGX_MDL m_pMdlOrg2;           // Original 2
PDGX_MDL m_pMdlIns2;           // Copy Instance 2
...
char    sMdl1[DGE_MAX_PATH]= MEDIA_DIR"model/mong.ase";
char    sMdl2[DGE_MAX_PATH]= MEDIA_DIR"model/mong.asb";

// Creates source data.
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;

// Loads texture.
DWORD ColorKey = 0x0;
m_pMdlOrg1->SetAttrib(DGXM_EXEC_TEXTURE_LOAD, &ColorKey);

```

```

// Copies a model from the source data.
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;

...

// Sets the time and matrix needed for animation.
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);
// Sets animation range.

```

```

{
    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);
// Sets animation range.
{
    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();

...

// Model rendering
m_pMdlOrg1->Render();
m_pMdlIns1->Render();
m_pMdlOrg2->Render();
m_pMdlIns2->Render();
m_pDev->EndScene();

...

// Release
DGX_DestroyMdl(&m_pMdlOrg1);
DGX_DestroyMdl(&m_pMdlIns1);


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

```

4. DGT

- Library: CAANOO - libdgt20.so, WIZ - libdgt20w.so Windows - dgt20_.lib
- Consists of game resource management such as 2D gathering shots, compression, DRM, etc.

4.1 IDGT_SpriteMx

2D gathering shots If possible, create a polygon layer on the same textures for output.

Begin: Sets the environment for 2D rendering.

End: Restores the environment for 3D rendering.

Add: Converts the texture and the range of an image into 3D polygon for addition. If it is the same as the texture of the last layer, the last layer is added with the 3D polygon only. Unless it is the same, a new layer is created to add polygon . If it is for simple rendering work, the processing speed is quicker than the AddEx.

AddEx: Expansion of "Add". Creates polygon for complex 2D rendering such as rotation or scaling conversion to add it to a layer.

AddtoExist: Adds polygon to the layer of existing texture. Unless texture exists in the previous layers, a new

layer will be created to add polygon. If it is for simple rendering work, the processing speed is quicker than the AddtoExistEx.

AddtoExistEx: Expansion of AddtoExist for rotation or scaling conversion and so on.

DrawLayer: Prints internal layers partially.

DrawLayer: Prints every layer that matches the texture given to Argument.

```
< IDGT_SpriteMx>

#include <DGT_SpriteMx.h>

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


DGE_HANDLE      m_hwnd;                // Windows Handle
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();                // Sets the environment for 2D rendering.
    m_pSpriteMx->DrawLayer();             // Draw Layer
    m_pSpriteMx->End();                   // Returns to the environment.
    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";

        // CreatesSpriteMx.
        if(DGE_FAILED(DGT_CreateSpriteMx(NULL, &m_pSpriteMx, m_pDev)))
            return -1

        // Creates texture.
        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);            // Releases texture.
        DGT_DestroySpriteMx(&m_pSpriteMx);    // Releases SpriteMx.
        DGE_DestroyDevice(&m_pDev);          // Releases device.
        DGE_DestroyWindow(m_hWnd);           // Releases Windows.
        DGE_Close();                          // Releases DGE.
        return 0;
    }
}

```

4.2 DGT Lzo

Handles the open compression Lzo (<http://www.oberhumer.com/>).

DgtLzo_Init: Initializes Lzo. Returns a negative value if the attempt fails.

DgtLzo_Close: Returns the Lzo resources.

DgtLzo_CompressFile: Compresses source files in the form of Dest files.

DgtLzo_DeCompressFile: Extracts the compressed files into a specific folder.

DgtLzo_DeCompressFromFile: Copies the data extracted from a compressed file to a buffer.

Buffer can be released through the free function.

DgtLzo_CompressBuf: Compresses buffer. Compressed buffer can be released through the free function.

DgtLzo_DeCompressBuf: Extracts buffers out of the compressed buffer.

Extracted buffer returns resources through the free function.

<Lzo Compression>

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

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

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

    // Extracts the compressed files into a specific folder.
    if(DGE_FAILED(DgtLzo_DeCompressFile("../", "t.gpz")))
        return -1;

    unsigned char* sbuf;
    unsigned long lbuf;

    // Brings data directly from the compressed file.
    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

Refer to the DRM document.

-End-