I. Mathematical Prerequisites
3. Transformations

Game Graphic Programming
Kyung Hee University
Software Convergence
Prof. Daeho Lee

Transformation

- Examples of geometric transformations are translation, rotation, and scaling.
- Transformation function
  - Input and output: vector or position

$$\tau(\mathbf{v}) = \tau(x, y, z) = (x', y', z')$$

  - Linear transformation
    - Transformation between two vector spaces preserves the operations of vector addition (additivity) and scalar multiplication (homogeneity).

$$\tau(\mathbf{u} + \mathbf{v}) = \tau(\mathbf{u}) + \tau(\mathbf{v})$$
$$\tau(k\mathbf{u}) = k\tau(\mathbf{u})$$

$$\tau(\mathbf{u}) = \mathbf{uA} = \begin{pmatrix} x & y & z \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

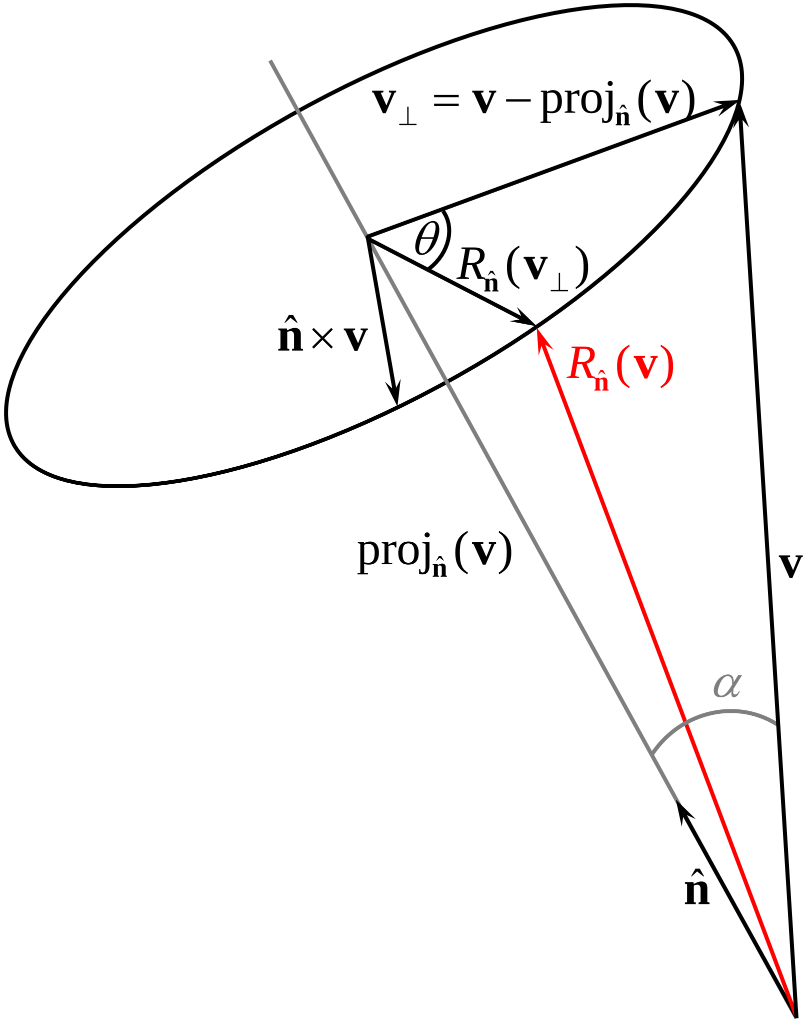
pre-multiplication

Scaling

$$S(\mathbf{v}) = S(x, y, z) = (s_x x, s_y y, s_z z)$$

$$S(\mathbf{u}) = \mathbf{uS} = \begin{pmatrix} x & y & z \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{pmatrix}$$

Rotation (1)



$$R_{\hat{\mathbf{n}}}(\mathbf{v}) = \mathrm{proj}_{\hat{\mathbf{n}}}(\mathbf{v}) + R_{\hat{\mathbf{n}}}(\mathbf{v}_\perp)$$

Both reference vectors have the same length and lie on the circle of rotation.

$$R_{\hat{\mathbf{n}}}(\mathbf{v}_\perp) = \cos\theta\,\mathbf{v}_\perp + \sin\theta(\hat{\mathbf{n}} \times \mathbf{v})$$

$$\begin{aligned} R_{\hat{\mathbf{n}}}(\mathbf{v}) &= (\hat{\mathbf{n}} \cdot \mathbf{v})\hat{\mathbf{n}} + \cos\theta\,\mathbf{v}_\perp + \sin\theta(\hat{\mathbf{n}} \times \mathbf{v}) \\ &= \cos\theta\,\mathbf{v} + \left(1 - \cos\theta\right)(\hat{\mathbf{n}} \cdot \mathbf{v})\hat{\mathbf{n}} + \sin\theta(\hat{\mathbf{n}} \times \mathbf{v}) \end{aligned}$$

$$R_{\hat{\mathbf{n}}} = \begin{pmatrix} c + (1-c)x^2 & (1-c)xy + sz & (1-c)xz - sy \\ (1-c)xy - sz & c + (1-c)y^2 & (1-c)yz + sx \\ (1-c)xz + sy & (1-c)yz - sx & c + (1-c)z^2 \end{pmatrix}$$

$$c = \cos\theta, s = \sin\theta$$

Kyung Hee University
nize@khu.ac.kr

Rotation (2)

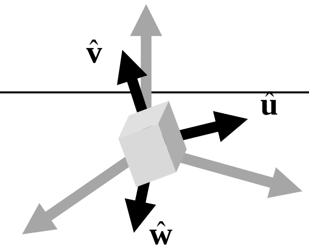
- Rotation matrices
  - Each row vector is a unit length, and the row vectors are mutually orthogonal.
  - The row vectors are orthonormal (i.e., mutually orthogonal and unit length).
  - A matrix whose rows are orthonormal is said to be an orthogonal matrix.
  - The inverse of an orthogonal matrix is equal to its transpose.

$$R_{\hat{\mathbf{n}}}^{-1} = R_{\hat{\mathbf{n}}}^{T}$$

$$\hat{\mathbf{n}} = (1, 0, 0) \qquad R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}$$

DOF (degrees of freedom): the number of independent parameters that define its configuration or state.

Euler angles: 3DOF, gimbal lock, discontinuity ($[0,\pi)$, $[-\pi, \pi)$, …)
Axis-angle (rotation vector, $\mathbf{v}=\theta\hat{\mathbf{v}}$): 3DOF, discontinuity ($[0,\pi)$, $[-\pi, \pi)$, …)
Unit quaternions ($q = w + ix + iy + kz$, $w^2+x^2+y^2+z^2 = 1$)
Rotation matrices: orientation of new rotated frame



$$R = \begin{pmatrix} \hat{\mathbf{u}}_x & \hat{\mathbf{u}}_y & \hat{\mathbf{u}}_z \\ \hat{\mathbf{v}}_x & \hat{\mathbf{v}}_y & \hat{\mathbf{v}}_z \\ \hat{\mathbf{w}}_x & \hat{\mathbf{w}}_y & \hat{\mathbf{w}}_z \end{pmatrix}$$

Homogeneous Coordinates

- Homogeneous coordinates provide a convenient notational mechanism to handle points and vectors uniformly.
- With homogeneous coordinates, we augment to 4-tuples and what we place in the fourth w-coordinate depends on whether we are describing a point or vector.
- Using the homogeneous notation, transformations of vectors and positions are handled in a unified way.
  - $(x, y, z, 0)$ for vectors.
  - $(x, y, z, 1)$ for points.

Affine Transformation

- A linear transformation cannot describe the translation.

$$\alpha(\mathbf{u}) = \tau(\mathbf{u}) + \mathbf{b}$$

- Linear transformation + translation ➔ affine transformation

$$\alpha(\mathbf{u}) = \tau(\mathbf{u}) + \mathbf{b} = \begin{pmatrix} x & y & z \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} + \begin{pmatrix} b_x & b_y & b_z \end{pmatrix}$$

$$= \begin{pmatrix} x & y & z & 1 \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ b_x & b_y & b_z & 1 \end{pmatrix}$$

Kyung Hee University
nize@khu.ac.kr

Composition of Transformations

- Matrix multiplication is associative.

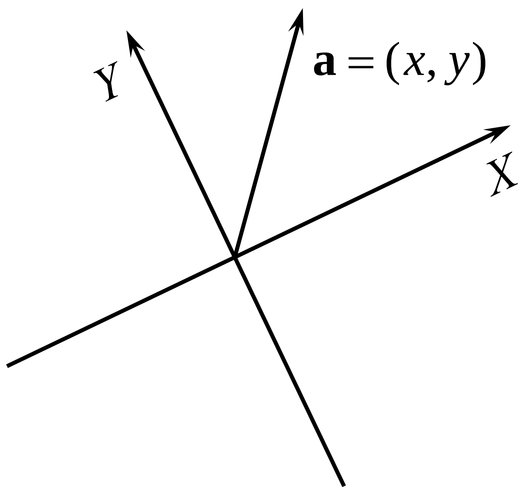
$$\left(\left(\mathbf{vS}\right)\mathbf{R}\right)\mathbf{T} = \mathbf{v}\left(\mathbf{SRT}\right)$$

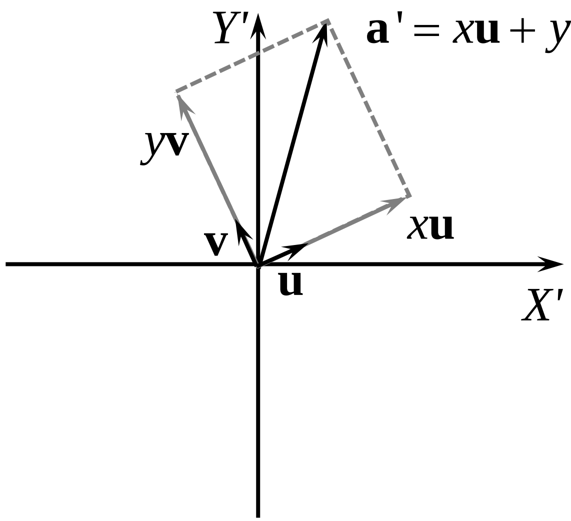
- Matrix multiplication is not commutative.

$$\left(\mathbf{vR}\right)\mathbf{T} \neq \left(\mathbf{vT}\right)\mathbf{R}$$

Change of Coordinate Transformations

- A vector **a** in XY-plane.
- The vector **a** in $X'Y'$-plane (**a'**).





- Points

$$\mathbf{p} = x\mathbf{u} + y\mathbf{v} + z\mathbf{w} + \mathbf{q}$$

DirectX Math Transformation Functions (1)

```
// Constructs a scaling matrix:
XMMATRIX XM_CALLCONV XMMatrixScaling(
    float ScaleX, float ScaleY, float ScaleZ); // Scaling factors

// Constructs a scaling matrix from components in vector:
XMMATRIX XM_CALLCONV XMMatrixScalingFromVector(FXMVECTOR Scale);
    // Scaling factors (sx, sy, sz)

// Constructs a x-axis rotation matrix Rx: clockwise angle θ
XMMATRIX XM_CALLCONV XMMatrixRotationX(float Angle);

// Constructs a y-axis rotation matrix Ry:
XMMATRIX XM_CALLCONV XMMatrixRotationY(float Angle);

// Constructs a z-axis rotation matrix Rz:
XMMATRIX XM_CALLCONV XMMatrixRotationZ(float Angle);

// Constructs an arbitrary axis rotation matrix Rn:
XMMATRIX XM_CALLCONV XMMatrixRotationAxis(
    FXMVECTOR Axis,       // Axis n to rotate about
    float Angle);         // Clockwise angle θ to rotate
```

DirectX Math Transformation Functions (2)

```
// Constructs a translation matrix:
XMMATRIX XM_CALLCONV XMMatrixTranslation(
   float OffsetX, float OffsetY, float OffsetZ);
   // Translation factors

// Constructs a translation matrix from components in a vector:
XMMATRIX XM_CALLCONV XMMatrixTranslationFromVector(FXMVECTOR Offset);
   // Translation factors

// Computes the vector-matrix product vM where vw  = 1
// for transforming points:
XMVECTOR XM_CALLCONV XMVector3TransformCoord(
   FXMVECTOR V, CXMMATRIX M);          // Input V and M

// Computes the vector-matrix product vM where vw  = 0
// for transforming vectors:
XMVECTOR XM_CALLCONV XMVector3TransformNormal(
   FXMVECTOR V, CXMMATRIX M);          // Input V and M
```

DirectX Math Transformation Functions (3)

```
#include <windows.h>
#include <DirectXMath.h>
#include <iostream>
#include <iomanip>
std::ostream& XM_CALLCONV operator <<(std::ostream& os, DirectX::FXMVECTOR v) {
    DirectX::XMFLOAT4 dest;
    DirectX::XMStoreFloat4(&dest, v);
    os << "(" << dest.x << ", " << dest.y << ", " << dest.z << ", "
       << dest.w << ")";
    return os;
}
std::ostream& XM_CALLCONV operator <<(std::ostream& os, DirectX::FXMMATRIX m) {
    for (int i = 0; i < 4; ++i) {
        os << DirectX::XMVectorGetX(m.r[i]) << "\t";
        os << DirectX::XMVectorGetY(m.r[i]) << "\t";
        os << DirectX::XMVectorGetZ(m.r[i]) << "\t";
        os << DirectX::XMVectorGetW(m.r[i]) << std::endl;
    }
    return os;
}
```

DirectX Math Transformation Functions (4)

```
int main() {
    if (!DirectX::XMVerifyCPUSupport()) {
        std::cout << "directx math not supported" << std::endl;
        return 0;
    }
    std::cout << std::fixed << std::setprecision(10);

    float theta = DirectX::XM_PIDIV4;
    DirectX::XMVECTOR Q = DirectX::XMVectorSet(0.0f, 0.0f,
       DirectX::XMScalarSin(theta/2.f)*1.0f, DirectX::XMScalarCos(theta/2.f));
    // ((ux, uy, yz)*sin(theta/2), cos(theta/2))

    DirectX::XMMATRIX A = DirectX::XMMatrixAffineTransformation(
        DirectX::XMVECTOR({ 1.0f, 1.0f, 1.0f, 0.0f }),
        DirectX::XMVECTOR({ 0.0f, 0.0f, 0.0f, 0.0f }),
        Q,
        DirectX::XMVECTOR({ 0.0f, 0.0f, 0.0f, 0.0f }));

    std::cout << "A = " << std::endl << A << std::endl;
```

DirectX Math Transformation Functions (5)

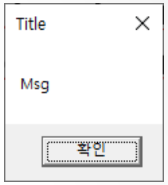
```
    DirectX::XMVECTOR X, Y;
    X = DirectX::XMVectorSet(-1.f, 1.f, 0.0f, 0.0f);
    Y = DirectX::XMVector3TransformNormal(X, A);

    std::cout << "X = " << std::endl << X << std::endl;
    std::cout << "Y = " << std::endl << Y << std::endl;

    DirectX::XMMATRIX A2 = DirectX::XMMatrixInverse(nullptr, A);
    X = DirectX::XMVector3TransformNormal(Y, A2);

    std::cout << "X = " << std::endl << X << std::endl;

    return 0;
}
```

DirectX Math Transformation Functions (6)

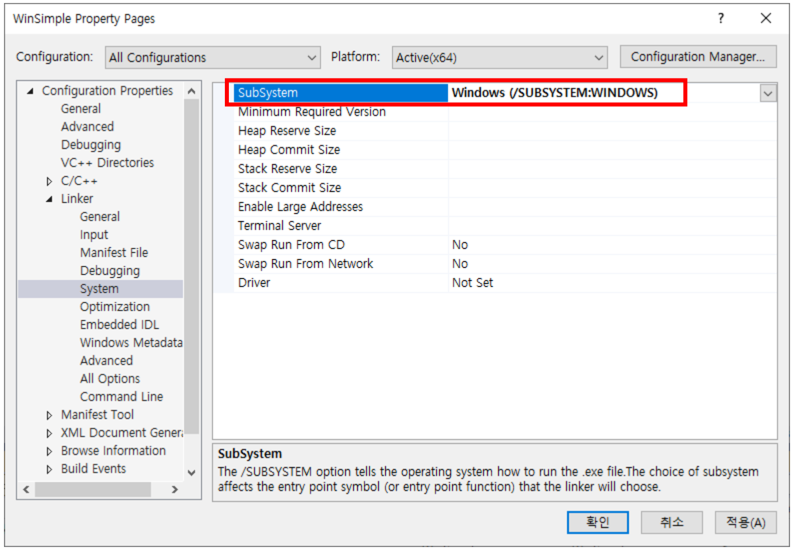
```
A =
0.7071068287    0.7071067691    0.0000000000    0.0000000000
-0.7071067691   0.7071068287    0.0000000000    0.0000000000
0.0000000000    0.0000000000    1.0000000000    0.0000000000
0.0000000000    0.0000000000    0.0000000000    1.0000000000

X =
(-1.0000000000, 1.0000000000, 0.0000000000, 0.0000000000)
Y =
(-1.4142136574, 0.0000000596, 0.0000000000, 0.0000000000)
X =
(-1.0000001192, 1.0000000000, 0.0000000000, 0.0000000000)
```

Sample Program

- New project (empty)
- [Linker] – [System] – [SubSystem]: Windows




```
#include <windows.h>
int WINAPI WinMain(HINSTANCE hInstance,
  HINSTANCE prevInstance,PSTR cmdLine, int showCmd) {
   return MessageBox(nullptr, L"Msg", L"Title", MB_OK);
}
```




```
#ifdef UNICODE
#define MessageBox  MessageBoxW
#else
#define MessageBox  MessageBoxA
#endif // !UNICODE
```

Windows Data Types

- Windows data types
  - https://learn.microsoft.com/en-us/windows/win32/winprog/windows-data-types
  - **`BOOL`**: A Boolean variable (should be **`TRUE`** or **`FALSE`**). **`typedef int BOOL;`**
  - **`BOOLEAN`**: A Boolean variable. **`typedef BYTE BOOLEAN;`**
  - **`BYTE`**: A byte (8 bits). **`typedef unsigned char BYTE;`**
  - **`DWORD32`**: A 32-bit unsigned integer. **`typedef unsigned int DWORD32;`**
  - **`DWORD64`**: A 64-bit unsigned integer. **`typedef unsigned __int64 DWORD64;`**
  - **`FLOAT`**: A floating-point variable. **`typedef float FLOAT;`**
  - **`UINT`**: An unsigned INT. The range is 0 through 4294967295 decimal. **`typedef unsigned int UINT;`**
  - **`INT_PTR`**: A signed integer type for pointer precision.
    - ```
      #if defined(_WIN64)
       typedef __int64 INT_PTR;
      #else
       typedef int INT_PTR;
      #endif
      ```
  - …
      ```

API References

- API references
  - https://learn.microsoft.com/en-us/windows/win32/api/d3d12/
  - Interfaces
  - Functions
    - **`[in]`**, **`[in, out]`**, **`[in, optional]`** (default **`NULL`**)
  - Callback functions
  - Structures
  - Enumerations

WinMain

- **WinMain** is the conventional name used for the application entry point.
- ```
  int __stdcall WinMain(
    [in]           HINSTANCE hInstance,
    [in, optional] HINSTANCE hPrevInstance,
    [in]           LPSTR     lpCmdLine,
    [in]           int       nShowCmd
  );
  ```
  - **__stdcall** calling convention is used to call Win32 API functions. The callee cleans the stack.
  - **__cdecl** is the default calling convention for C and C++ programs. The stack is cleaned up by the caller.
  - Handle is a variable that identifies an object; an indirect reference to an operating system resource.
  ```

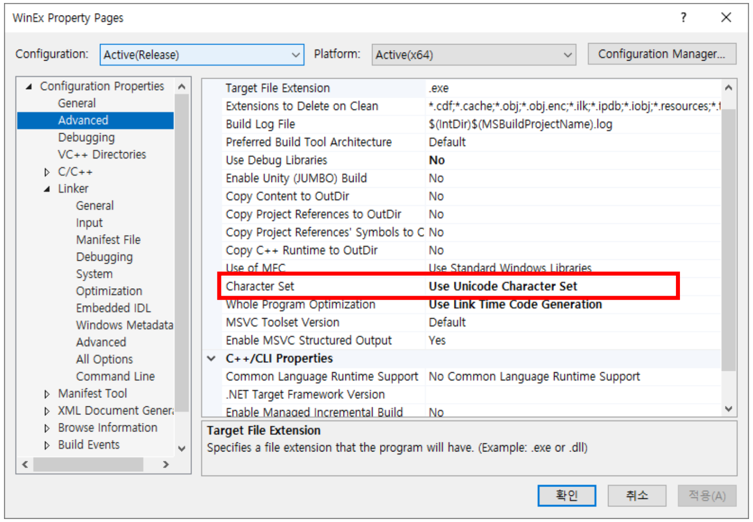
MessageBox

- **`MessageBox`** displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.

- ```
  int MessageBox(
     [in, optional] HWND    hWnd,
     [in, optional] LPCTSTR lpText,
     [in, optional] LPCTSTR lpCaption,
     [in] UINT    uType
  );
  ```

- **`L`**: wide literal encoding (for Unicode)
  - **`L'A', L"Ex"`**

```
#ifdef UNICODE
#define MessageBox  MessageBoxW
#else
#define MessageBox  MessageBoxA
#endif // !UNICODE
```


```

Windows Application (1)

- Every Windows desktop application must have a window-procedure function.
  - A function that processes all messages sent or posted to all windows of the class.
  -
```
LRESULT CALLBACK WndProc(
    [in] HWND   hWnd,
    [in] UINT   message,
    [in] WPARAM wParam,
    [in] LPARAM lParam
);
```
  - This function is called a **`WndProc`**, but you can give it whatever name you like in your code.
  - It is called handling an event.

A callback function is a function that is passed (as an argument) to another (function) and is executed after the completion of some operations.

Windows Application (2)

- Register window-class
  - ```
    typedef struct tagWNDCLASSA {
      UINT      style;
      WNDPROC   lpfnWndProc;
      int       cbClsExtra;
      int       cbWndExtra;
      HINSTANCE hInstance;
      HICON     hIcon;
      HCURSOR   hCursor;
      HBRUSH    hbrBackground;
      LPCSTR    lpszMenuName;
      LPCSTR    lpszClassName;
    } WNDCLASSA;
    ```
  - ```
    ATOM RegisterClassA(
      [in] const WNDCLASSA *lpWndClass
    );
    ```

Windows Application (3)

- Create window
  - ```
    HWND CreateWindowA(
      [in, optional] LPCSTR    lpClassName,
      [in, optional] LPCSTR    lpWindowName,
      [in]           DWORD     dwStyle,
      [in]           int       x,
      [in]           int       y,
      [in]           int       nWidth,
      [in]           int       nHeight,
      [in, optional] HWND      hWndParent,
      [in, optional] HMENU     hMenu,
      [in, optional] HINSTANCE hInstance,
      [in, optional] LPVOID    lpParam
    );
    ```
- Show window
  - ```
    BOOL ShowWindow(
      [in] HWND hWnd,
      [in] int  nCmdShow
    );
    ```


Kyung Hee University
nize@khu.ac.kr

Windows Application (4)

- Message loop
  - ```
    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
    // Translates virtual-key messages into character messages.
      TranslateMessage(&msg);
    // Dispatches a message to a window procedure
      DispatchMessage(&msg);
    }
    ```


Kyung Hee University
nize@khu.ac.kr
    ```

Simple Window Application (1)

```
#include <windows.h>
LRESULT CALLBACK WndProc(HWND hwnd, UINT message,
   WPARAM wParam, LPARAM lParam)
{
    switch (message) {
    case WM_CLOSE:
        PostQuitMessage(0);
        break;
    }

    return (DefWindowProc(hwnd, message, wParam, lParam));
}
```


Kyung Hee University
nize@khu.ac.kr

Simple Window Application (2)

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE prevInstance,
    PSTR cmdLine, int showCmd) {
    HWND hWnd;
    WNDCLASS wc;
    MSG msg;

    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(0, IDI_APPLICATION);
    wc.hCursor = LoadCursor(0, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)GetStockObject(NULL_BRUSH);
    wc.lpszMenuName = 0;
    wc.lpszClassName = L"ClassName";
```

Simple Window Application (3)

```
    RegisterClass(&wc);

    hWnd = CreateWindow(L"ClassName", L"WindowName",
      WS_OVERLAPPEDWINDOW, 0, 0, 640, 480,
      0, 0, hInstance, nullptr);
    ShowWindow(hWnd, showCmd);

    while (1) {
        if (PeekMessage(&msg, 0, 0, 0, PM_REMOVE)) {
            if (msg.message == WM_QUIT) break;
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int)msg.wParam;
}
```