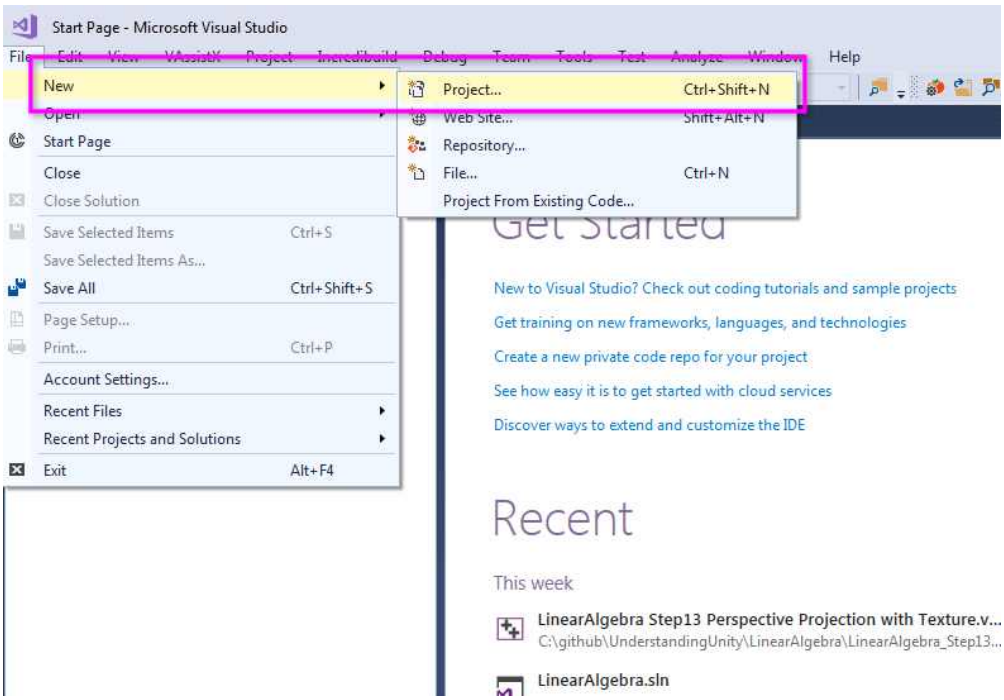
GDI

3D 그래픽 프로그래밍의 원리를 이해하기 위해 우리는 윈도우즈의 GDI(Graphics Device Interface) 혹은 GDI+를 이용할 것입니다. GDI는 다양한 2D 그래픽 함수들을 제공합니다. 우리는 몇 단계까지는 이 함수들 중 일부를 이용할 것이므로 윈도우즈 응용 프로그램의 프레임웍(framework)을 어느 정도는 이해해야 합니다.

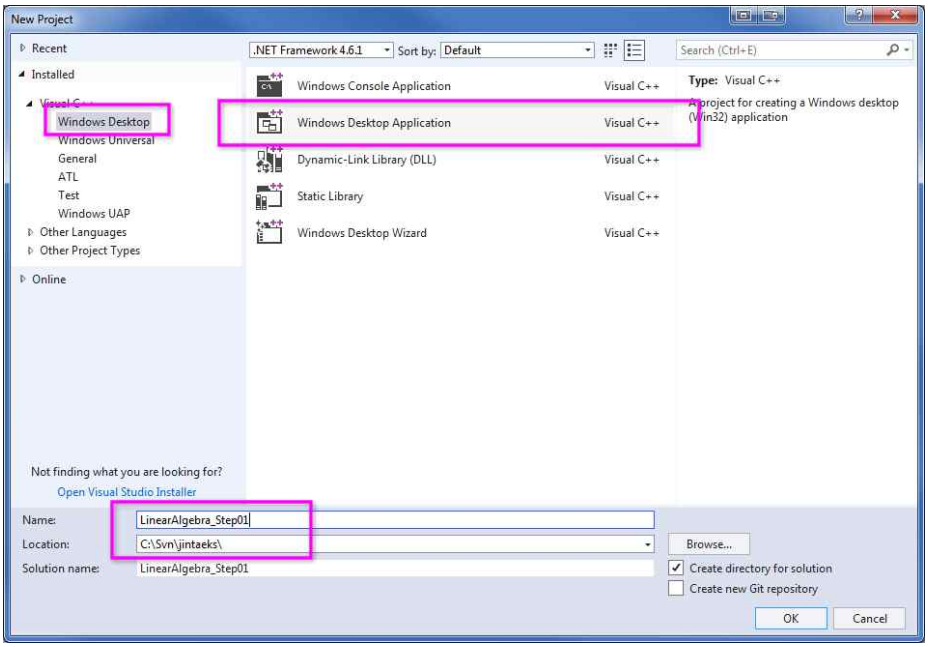
1. Win32코드의 생성

기본적인 바탕코드를 생성하기 위해, Visual Studio 2017을 이용하여 Windows Desktop Application 프로젝트를 작성합니다. 위저드(Wizard)가 자동으로 생성한 코드는 윈도우즈 프레임웍에 맞게 동작하는 코드를 자동으로 생성합니다.



[그림] Windows Desktop Application 프로젝트의 생성: [New-->Project...]를 선택합니다.

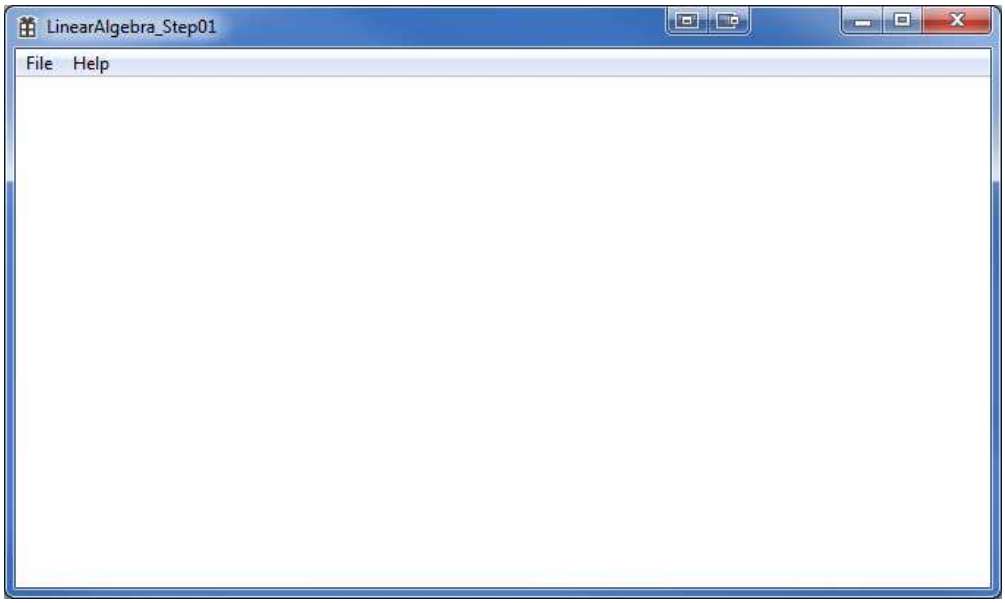
Visual Studio 2017에서 [New-->Project...]를 선택합니다. 그러면 New Project 대화창이 실행됩니다.



[그림] Windows Desktop Application의 생성: Visual C++탭의 Windows Desktop을 선택하고, 프로젝트 종류에서 Windows Desktop Application을 선택합니다. Name:에 프로젝트 이름을 입력하고, Locations:에 프로젝트를 생성할 폴더를 지정합니다.

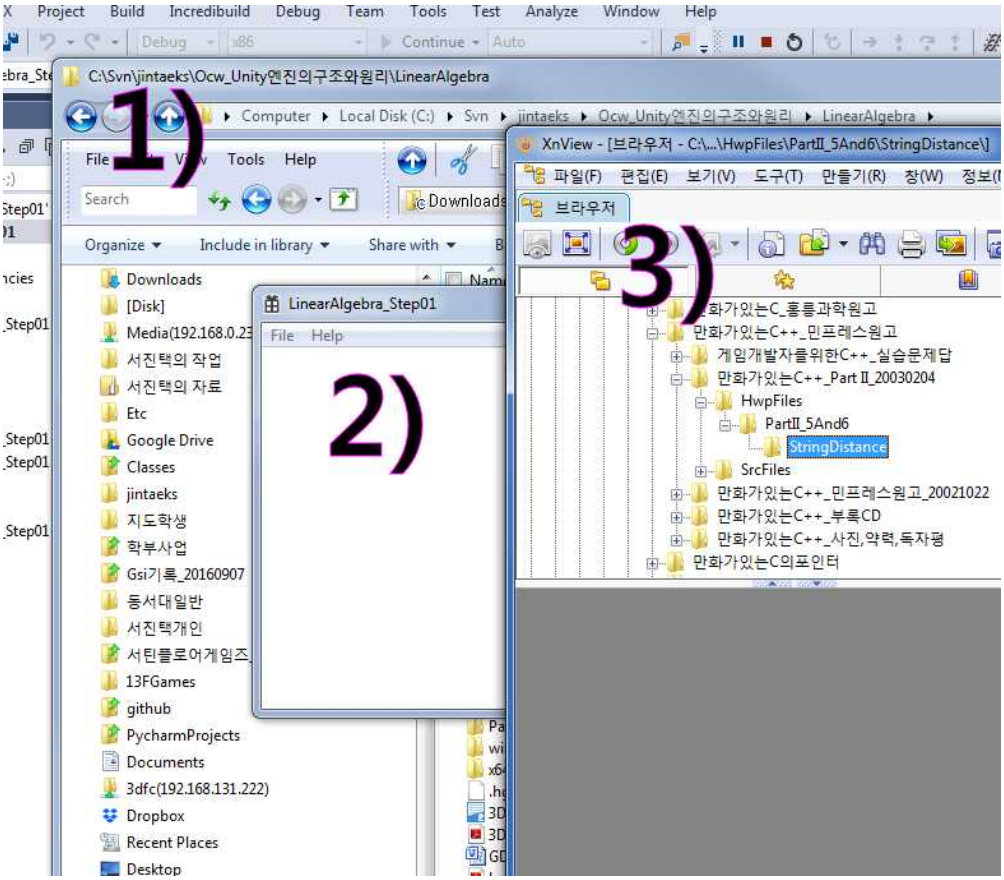
대화창에서 Visual C++탭의 Windows Desktop을 선택하고, 프로젝트 종류에서 Windows Desktop Application을 선택합니다. Name: 텍스트 박스에 프로젝트 이름을 입력하고, Locations: 텍스트 박스에 프로젝트를 생성할 폴더를 지정합니다. Ok버튼을 누르면 위저드가 코드를 자동으로 생성합니다.

F7을 눌러서 코드를 빌드(Build)합니다. 그리고 [Debug--> Start Debugging]을 선택하면 응용 프로그램을 실행할 수 있습니다.



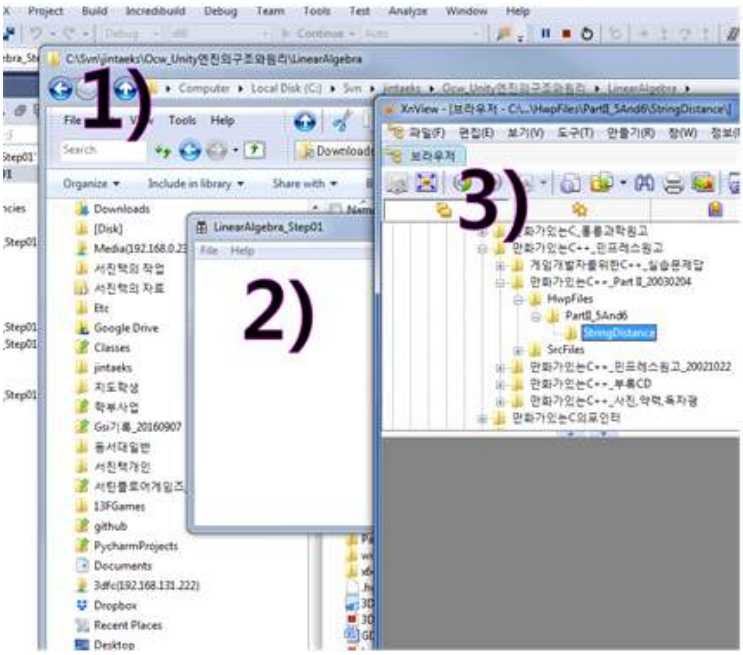
[그림] 응용 프로그램의 실행: [Debug --> Start Debugging]을 선택하면 응용 프로그램을 실행할 수 있습니다.

어떤 윈도우의 특정한 영역에 무언가를 그리기 위해서는 어떠한 정보가 필요할까요? 아래 그림과 같은 세 개의 윈도우를 고려해 봅시다. 우리는 이 중 두 번째 윈도우의 클라이언트 영역에 선을 그리려고 합니다.



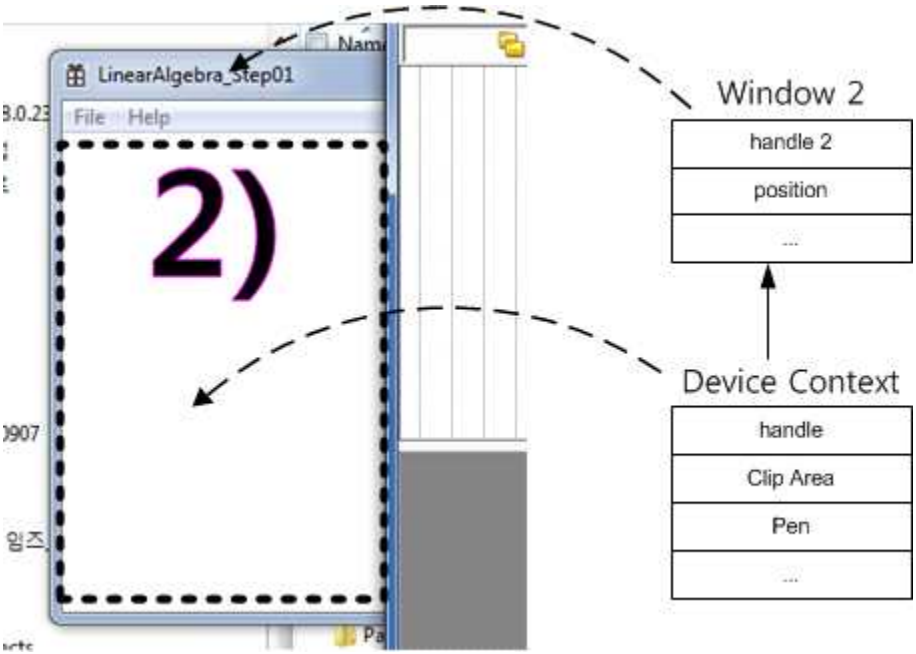
[그림] 세개의 윈도우: 운영체제는 각 윈도우를 관리하는 구조체를 유지합니다. 특정 윈도우의 정보를 얻어내기 위해서는 먼저 이 구조체를 접근해야 합니다.

운영체제는 각 윈도우를 관리하는 구조체(Structure)를 유지합니다. 특정 윈도우의 정보를 얻어내기 위해서는 먼저 이 구조체를 접근해야 하는데, 각 구조체를 구분하는 유일한 정수값을 **핸들(Handle)**이라고 합니다. 그러므로 윈도우에 그리기 위해서는 먼저 **윈도우 핸들(Windows handle)**을 얻어야 합니다.



[그림] 윈도우 핸들(window handle): 운영체제가 유지하는 각 종류의 구조체를 구분하는 유일한 ID를 핸들이라고 합니다.

위 그림에서 3개의 윈도우에 대해 윈도우즈 운영체제가 3개의 윈도우 정보 구조체를 유지한다고 합시다. 운영체제는 각 구조체를 구분하는 유일한 ID값을 유지하는데 이 값을 윈도우 핸들이라고 합니다. 위 그림에서 2) 번 영역의 윈도우에 무언가를 그리기 위해서는 먼저 2) 윈도우의 윈도우 핸들을 얻어야 합니다.



[그림] 디바이스 컨텍스트(DC, Device Context) 핸들: 윈도우 구조체 정보를 전달하면, 그리기 정보를 유지하는 디바이스 컨텍스트 구조체의 핸들 HDC를 얻을 수 있습니다. 선택된 펜(pen), 브러시(brush), 클리핑 영역(clipping area), 색 깊이(color depth)등 다양한 그리기 정보를 DC에 설정할 수 있습니다.

선을 그린다는 말은 많은 자세한 부분이 생략되어 있습니다. 선의 굵기, 선의 색, 좌표계, 클리핑, 바탕화면의 현재 색 깊이 등 고려해야할 부분이 많이 있습니다. 이러한 그리기 정보에 대한 구조체를 **디바이스 컨텍스트(DC, device context)**라고 하며, 디바이스 컨텍스트는 특정 윈도우에 항상 종속되어 있으므로 DC에 대한 핸들(HDC)을 얻기 위해서는 윈도우 핸들이 필요합니다.

그러므로 DC를 얻어내는 함수 GetDC()등의 첫 번째 파라미터는 윈도우 핸들이며, 대부분의 그리기 함수의 첫 번째 파라미터는 HDC입니다.

2. 그리기 코드의 추가

자동으로 생성된 코드에 그리는 함수를 추가하도록 하겠습니다. 그리기 함수의 이름을 OnPaint()라고 하겠습니다. 이 함수는 HDC를 파라미터로 받습니다. WndProc()함수의 정의 바로 위에 이 함수를 추가합니다.

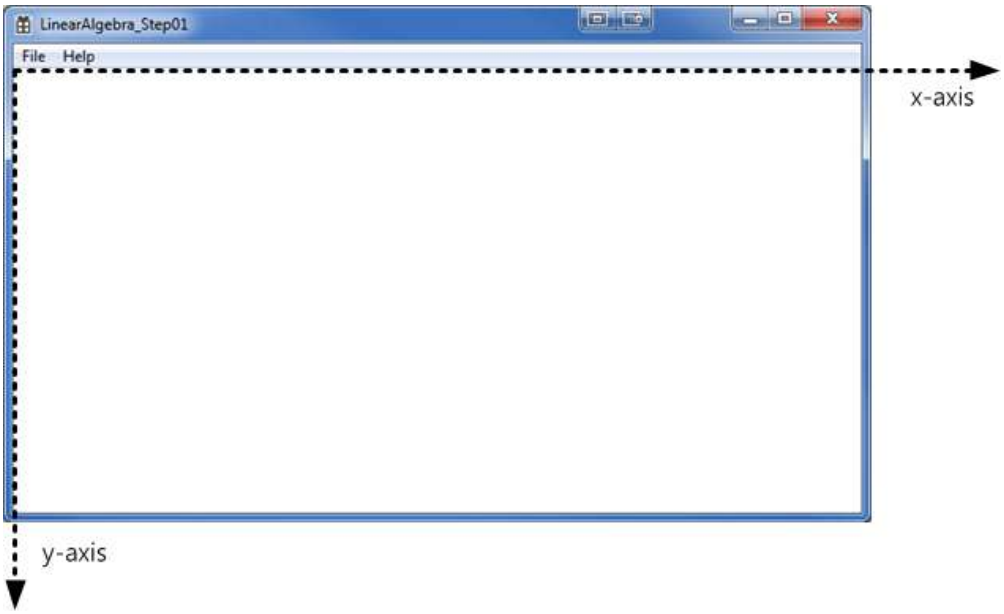
```
void OnPaint(HDC hdc)
{
}
```

이제 WndProc()의 WM_PAINT 메시지 처리 부분을 아래와 같이 수정합니다. EndPaint()전에 OnPaint()를 호출하도록 했습니다. WM_PAINT메시지에서 HDC를 얻는 함수는 BeginPaint()입니다. 이 함수는 WM_PAINT메시지를 메시지 큐에서 제거하고, 페인트 구조체를 Out 파라미터로 얻으며, HDC를 리턴합니다.

```
    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            // TODO: Add any drawing code that uses hdc here...
            // _20180519_jintaeks
            OnPaint( hdc );
            EndPaint(hWnd, &ps);
        }
        break;
```

이제 윈도우의 화면이 무효화(Invalidate)되어서 무언가를 그려야 할 상황이 되면, OnPaint()가 호출됩니다. OnPaint()에 그리기 코드를 추가하도록 하겠습니다.

클라이언트 영역에 그리기 함수는 스크린 좌표계(Screen Coordinate System)를 사용합니다. 스크린 좌표계에서 디폴트 원점은 클라이언트 영역의 왼쪽 상단입니다.

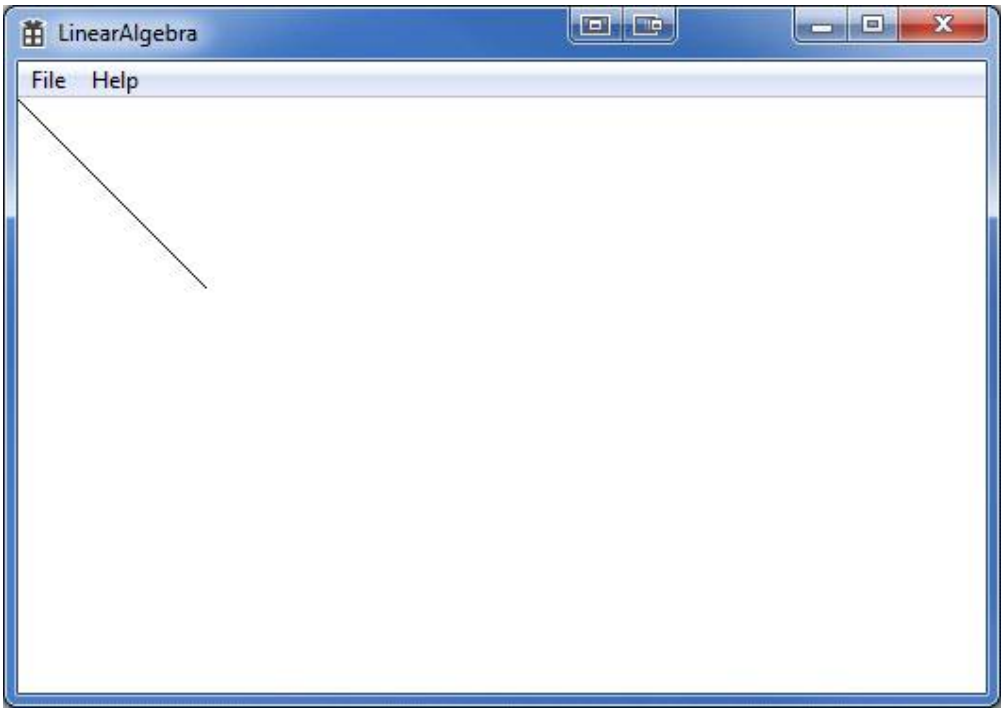


[그림] 스크린 좌표계: 스크린 좌표계의 원점은 좌측 상단이며, x축은 오른쪽 방향으로 증가하는 값을 가지고, y축은 아래 방향으로 증가하는 값을 가집니다.

스크린 좌표계는 기본적으로, x축은 오른쪽 방향으로 증가하는 값을 가지고, y축은 아래 방향으로 증가하는 값을 가집니다. DC 구조체는 그리기를 위한 현재 점(dot)위치를 가지는데 이것을 CP(Current Point)라고 합니다. 이 DC를 (0,0)위치로 옮기고, 이 곳에서 (100,100)위치로 선을 그리는 코드를 아래와 같이 OnPaint()내부에 추가합니다.

```
void OnPaint( HDC hdc )
{
    MoveToEx( hdc, 0, 0, NULL );
    LineTo( hdc, 100, 100 );
}
```

이제 프로그램을 실행해 보면, (0,0)에서 (100,100)에 선이 그려지는 것을 확인할 수 있습니다.



[그림] 선 그리기: (0,0)을 시작점으로 하고 (100,100을 끝점으로 하는 선을 그립니다.

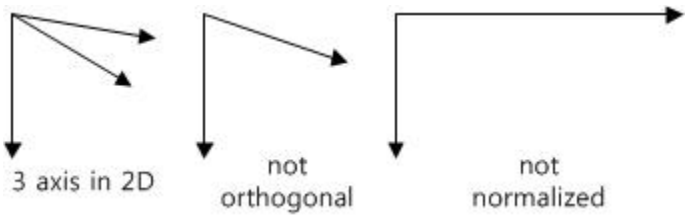
아래의 코드는 HDC의 2D그래픽에서의 현재 위치(CP, current point)를 (0,0)으로 이동합니다.

```
MoveToEx( hdc, 0, 0, NULL );
```

MoveToEx()의 마지막 파라미터는 무시합시다. CP란 보이지 않는 그래픽 커서라고 생각하면 됩니다. 이제 CP위치에서 (100,100)위치로 선을 그리기 위해서 LineTo()를 호출합니다.

```
LineTo( hdc, 100, 100 );
```

좌표계에 대한 의문들이 있습니다.



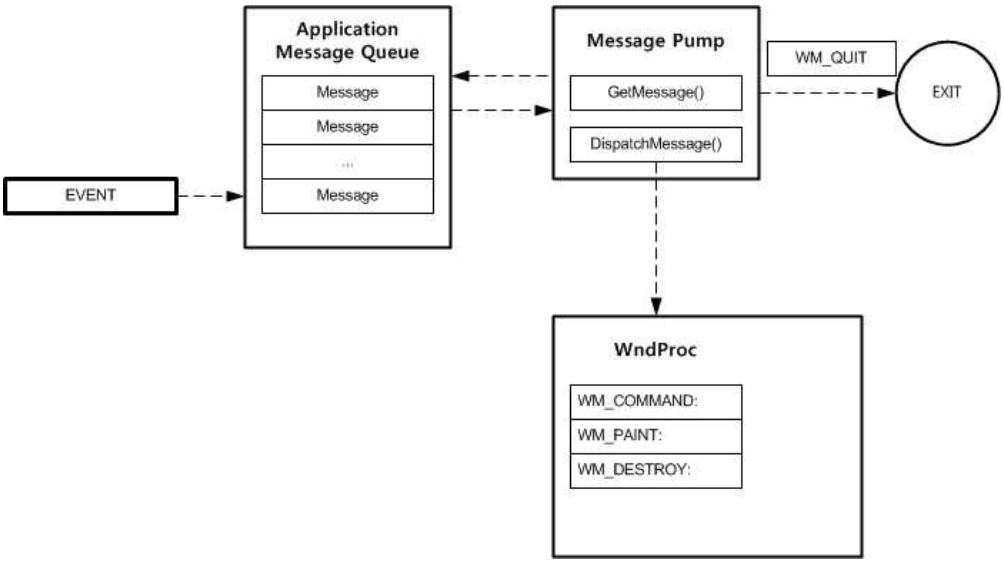
[그림] 좌표축의 선택: 2차원에 3개의 좌표축을 선택, 서로 직교가 아닌 좌표축의 선택, 서로 길이가 다른 좌표축의 선택

윈도우 영역에 그리기 위해서는 왜 좌표축을 2개 사용해야 할까요? 3개 사용해도 되지 않나요? 그리고 좌표축은 왜 항상 서로 직각일까요? 직각이 아니어도 표현이 가능하지 않나요? 그리고 좌표축은 왜 서로 같은 단위 길이(Unit Length)를 사용할까요? 단위 길이를 사용하지 않으면 어떤 문제가 있을까요?

이러한 질문에 대한 대답은 이어지는 내용에서 살펴볼 것입니다.

3. 윈도우즈 프로그래밍

필자는 1990년에 Windows 3.1에서 처음으로 윈도우즈 프로그래밍을 배웠습니다. MS-DOS에서만 프로그래밍 하다가, Windows 응용 프로그램을 작성하려고 했을 때 여러가지 장애물이 있었지만, 가장 이해하기 힘들었던 부분은 윈도우즈 메시지 펌프와 GetDC()와 BeginPaint()가 존재하는 이유였습니다. 그래서 이곳에서는 이것들의 개념에 대해서 설명하도록 하겠습니다.



[그림] 윈도우즈 메시지 펌프: GetMessage()는 응용 프로그램 메시지 큐에서 메시지를 가져 옵니다. 가져온 값이 WM_QUIT메시지이면 0을 리턴합니다.

윈도우즈 운영체제는 특정한 이벤트가 발생하면 이벤트에 해당하는 메시지를 각 응용 프로그램의 메시지 큐(Message Queue)에 넣습니다. 그러므로 응용 프로그램에는 자신의 메시지 큐에서 메시지를 꺼내오는 루프가 있어야 하는데, 이것을 **메시지 루프(Message Loop)**라고 합니다.

메시지 루프의 내부에서는 메시지 큐를 접근해서 메시지를 꺼내오는 함수를 호출해야 합니다. 메시지를 꺼내기 위해서는 GetMessage()를 사용할 수 있습니다. GetMessage()와 더불어서 PeekMessage()도 많이 사용하는데 두 함수의 차이점은 다음과 같습니다. GetMessage()는 응용 프로그램 메시지 큐에 메시지가 없으면 계속 대기합니다. PeekMessage()는 메시지가 없는 경우에도 즉시 리턴합니다. 그러므로, 게임처럼 계속해서 Update()작업을 해야 하는 경우, 메시지 루프 내부에서는 PeekMessage()를 사용합니다.

그리고 GetMessage()는 가져온 메시지가 WM_QUIT이면 0을 리턴합니다. 그러므로 GetMessage()의 리턴값이 0이면 루트를 탈출하도록 메시지 루프를 작성해야 합니다. GetMessage()가 가져온 메시지에 대해서 이 메시지를 처리하기 위해서는 **윈도우 프로시져(Windows Procedure)**라고 하는 특별한 함수를 호출해야 합니다. 이 함수는 내가 호출하는 것이 아니라, 윈도우가 호출하는 것이므로 CALLBACK으로 작성되어야 하고, 특정 메시지에 대해서 이 함수를 호출해 달라고 윈도우 운영체제에게 요청해야 합니다. 이 역할을 하는 것이 DispatchMessage()입니다. 그러므로 가장 간단한 메시지 루트는 다음과 같은 구

조를 가집니다.

```
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        DispatchMessage(&msg);
    }
```

자동으로 생성된 코드에서 메시지 루프를 위와 같이 수정해 보세요. 그래도 정상적으로 동작합니다.

윈도우즈의 어딘가를 갱신해야 한다면 윈도우즈 운영체제는 WM_PAINT메시지를 생성합니다. 이 메시지는 아주 특별하게 처리되는데, 다른 메시지와는 다르게 GetMessage()를 호출 했을 때 메시지 큐에서 제거되지 않습니다. 그렇게 되는 이유는 여러 번 갱신되어야 할 필요가 있을 때, 한번만 갱신하기 위함입니다.

화면을 10번 갱신해야 한다고 합시다. 그런데 아직 이전 WM_PAINT에 대해서 아직 갱신루틴을 호출하지 않았는데, 또 WM_PAINT가 발생했습니다. 그러면 같은 갱신을 2번해야 하는 문제가 발생합니다. 그래서 WM_PAINT메시지의 경우, 응용 프로그램 메시지 큐에 WM_PAINT 메시지가 있으면 추가적인 메시지를 메시지 큐에 넣지 않습니다.

그러므로 WM_PAINT를 처리할 때 이 메시지를 제거해 주어야 하는데, BeginPaint()가 그 일을 합니다. 만약 WM_PAINT메시지를 처리할 때, BeginPaint()를 호출해 주지 않으면, WM_PAINT처리 루틴이 무한히 반복될 것입니다.

WM_PAINT메시지와 상관없이 화면에 뭔가를 그리기 위해서는 HDC를 얻어야 합니다. 이 때 사용하는 것이 GetDC()입니다. 만약 WM_PAINT에서 GetDC()를 사용하면 어떻게 될까요? 상관없습니다. 하지만, WM_PAINT 메시지를 응용 프로그램 메시지 큐에서 제거하기 위해서 반드시 BeginPaint()를 호출해 주어야 합니다.

자동으로 생성된 코드의 다른 부분은 설명하지 않습니다. 윈도우 클래스를 등록하고, 윈도우를 생성하고 또 메뉴를 정의하고 메뉴 메시지를 처리하는 등의 기능은 다른 자료를 참고해서 이해하기 바랍니다.

실습문제

1. BeginPaint()와 GetDC()의 차이점에 대해서 설명하세요(힌트: GetDC()는 응용 프로그램 메시지 큐를 접근하지 않습니다).

2. SetROP2(hDC, R2_NOT);으로 설정된 DC의 연속적인 선 그리기는 이미 존재하는 선을 지우거나, 배경에 상관없이 선을 그리고/지우기를 반복할 수 있습니다. 동작을 상세하게 설명하세요.

@