2D Game Programming 04

afewhee@gmail.com

- FPS = Frame / 시간 간격
- timeGetTime() 함수 이용

```
UpdateFrame()
{
    static int          iCnt =0;
    static DWORD        dB = timeGetTime();
    DWORD               dE = timeGetTime();

    ++iCnt;

    if(iCnt>30)         // 30Frame을 기준
    {
      m_fFps = FLOAT(dE-dB);
      m_fFps = iCnt*1000/m_fFps;

      iCnt = 0;
      dB = dE;
    }
}
```


- DX9.0b Sprite의 Draw()함수
  - Draw(pTexture, Scale, RotCenter, Angle, Translation, color)
  - DX9.0c이후 행렬을 이용해서 표현
  - 2D는 행렬까지 손댈 필요는....

- 9.0b와 같은 결과를 위한 최종 행렬 계산
  - mtW = Scale * $RotCenter^{-1}$ * RotZ * RotCenter * Translate;
  - 특별한 경우가 아니라면 이 방법은 비추

- 9.0C 에서의 최종 행렬 계산 방법
  - 최종 행렬 = T^{-1} * (S * R) * T

3. 화면 전환 (Full Mode ←→ Window Mode)

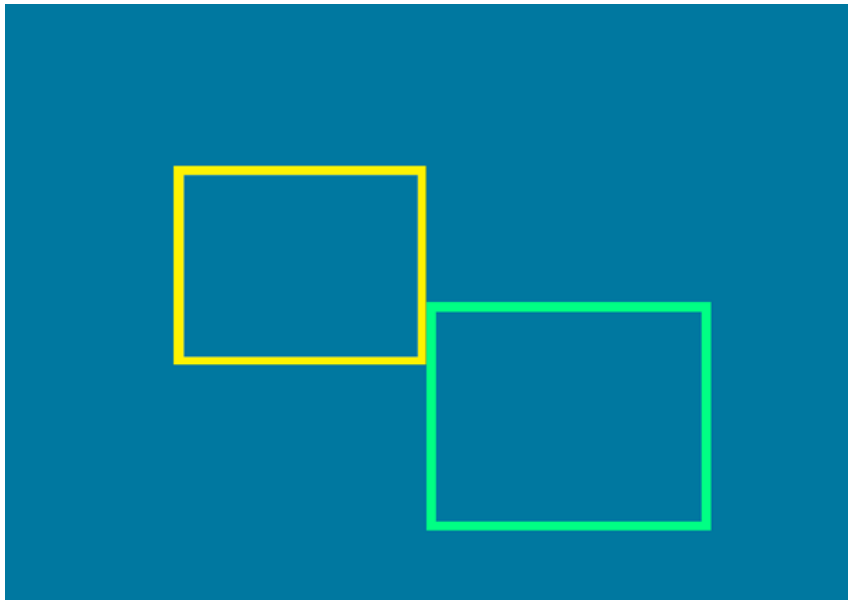
- 후면버퍼 전면버퍼 교체(flipping)

```
m_d3dpp.Windowed      = m_bWindow;
m_d3dpp.FullScreen_RefreshRateInHz = m_bWindow? 0 : 60;

m_pd3dSprite->OnLostDevice();
m_pd3dDevice->Reset(&m_d3dpp);
m_pd3dSprite->OnResetDevice();
```

- 전체 화면 상태에서 다른 윈도우 활성화 될 때(ALT+TAB, CTRL+ESC 등) 처리

```
검사: hr= m_pd3dDevice->Present( NULL, NULL, NULL, NULL);

디바이스를 로스트 상태로
if( D3DERR_DEVICELOST == hr )
        m_bDeviceLost = TRUE;


// 디바이스를 계속해서 활용 할 수 있는지 테스트.
if( FAILED( hr = m_pd3dDevice->TestCooperativeLevel() ) )
{
        // 디바이스를 리셋
}
```

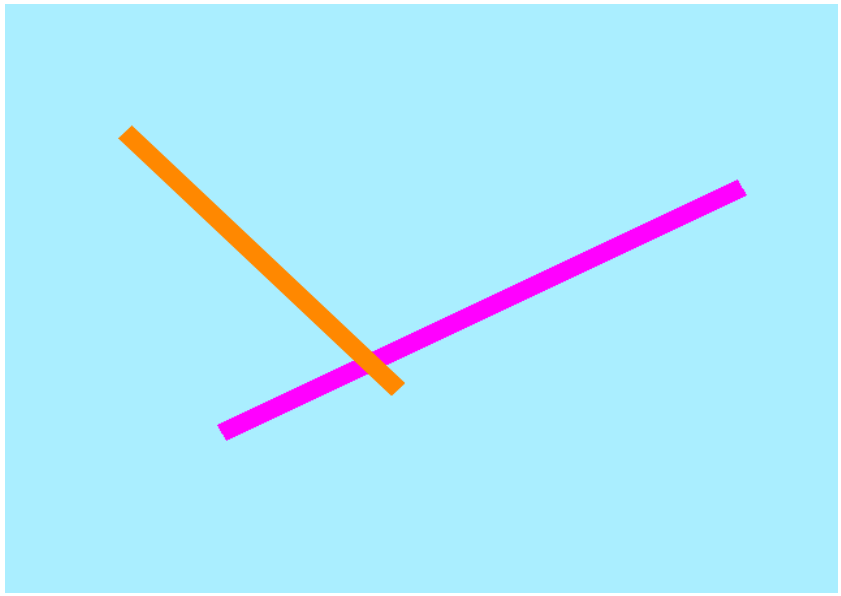
직사각형 충돌

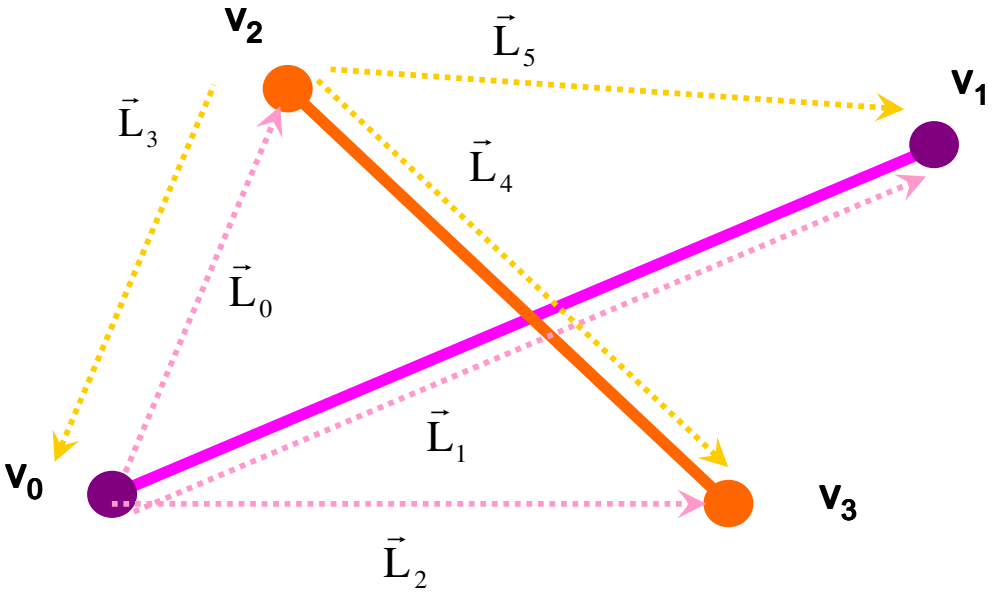


```
RECT rcCol1;
RECT rcCol2;

INT bColl=0;

if( rcCol1.left <= rcCol2.right  &&
      rcCol1.right>= rcCol2.left   &&

      rcCol1.top    <= rcCol2.bottom  &&
      rcCol1.bottom >= rcCol2.top  )
{
      bColl =1;
}
```

- 한 평면 위의 두 직선이 교차할 조건

$$(\vec{L}_0 \times \vec{L}_1) \cdot (\vec{L}_1 \times \vec{L}_2) > 0 \quad \&\ \&$$
$$(\vec{L}_3 \times \vec{L}_4) \cdot (\vec{L}_4 \times \vec{L}_5)$$






```
VEC2 L0 = v2 - v0;
VEC2 L1 = v1 - v0;
VEC2 L2 = v3 - v0;

VEC2 L3 = v0 - v2;
VEC2 L4 = v3 - v2;
VEC2 L5 = v1 - v2;

FLOAT D1 =
  (L0.x * L1.y - L0.y * L1.x) * (L1.x * L2.y - L1.y * L2.x);

FLOAT D2 =
  (L0.x * L4.y - L0.y * L4.x) * (L4.x * L5.y - L4.y * L5.x);

// collision
if(D1>0 && D2 >0)
    return 0;
```

추적 대상: Tar, 추적: Obj
속도 벡터, 속력 필요

1. 방향 벡터 구하기: 추적 대상 위치 – 추적 객체 위치
   Normalize(Direction = Tar pos – Obj pos)

2. 새로운 속도 방향 벡터 구하기
   Velocity = Velocity + Direction * fWeight;
   Normalize(Velocity)

3. 새로운 위치 = 현재 위치 + 새로운 속도 벡터 방향 벡터 * 스피드
   Obj pos = Obj pos + Velocity * fSpeed;