3D Game Programming 04

afewhee@gmail.com

- Vertex Buffer
- Index Buffer
- Primitive
- UP
- 실습

1. Vertex Buffer

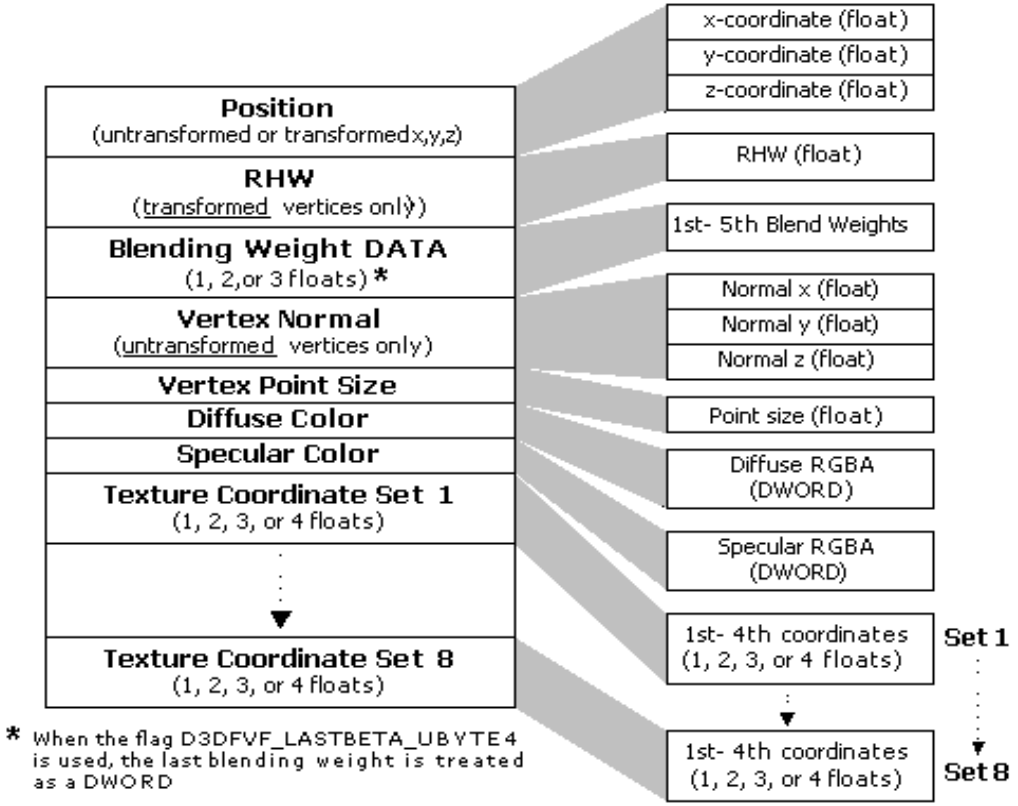
- 정점 (Vertex)
  - Primitive를 구성하는 기본 단위
  - 같은 Vertex를 가지고도 다른 Primitive를 만들어 낼 수 있음
  - 위치에 대한 기하 정보 뿐만 아니라 Lighting과 연산을 위한 법선 벡터(Normal Vector), 난반사(Diffuse), 정반사(Specular) 색상, 텍스쳐 매핑 좌표 등을 포함하기도 함
  - 정점의 구조는 유저가 작성함으로 디바이스에게 정점구조의 형식을 알려야 함 ➔ SetFVF()
  - 생명체와 비교하면 일종의 분자 구조

- 정점 구조의 예

```
Ex)
struct VtxNDUV2
{
    float3      position;
    float3      normal;
    DWORD       diffuse;
    float2      tex0;
    float2      tex1;

    enum{ FVF=(D3DFVF_XYZ|₩
          D3DFVF_NORMAL|₩
          D3DFVF_DIFFUSE|₩
          D3DFVF_TEX2), };
};
```

  - 정점 구조의 필드 순서는 지켜야 하고
  - 구조의 형식과 FVF는 일치해야 함



- 프리미티브 (Primtive) ➔ Primitive 참조

- 정점 버퍼(Vertex Buffer)
  - 사용자가 요청한 Memory Pool에 Device가 렌더링을 위해 Vertex의 연속으로 구성된 메모리 공간.
  - 시스템 메모리 또는 비디오 메모리에 생성
  - 생성: pDevice->CreateVertexBuffer()
  - 해제: Vertex Buffer또한 COM을 상속 받으므로 Release()로 해제
  - 갱신: 정점 버퍼는 Lock()함수를 통해 메모리 포인터를 얻어오고 이 포인터를 가지고 데이터 갱신
  - 정보: pVertexBuffer->GetDesc()

- 렌더링
  - 디바이스에 정점 버퍼를 연결: pDevice->SetStreamSouce()
    - 디바이스는 여러 정점 버퍼를 결합해서 렌더링을 할 수 있음
  - 정점의 형식 지정: pDevice->SetFVF()
    - FVF: Flexible Vertex Format. ➔ Direct3D가 지원하는 정점 형식을 조합해서 구성
  - 렌더링: pDevice->DrawPrimitive(), 인덱스를 사용하는 경우 pDevice->DrawIndexedPrimitive()
- User Point Memory:
  - 정점 버퍼를 사용하지 않고 시스템 메모리에 확보된 정점들을 이용해서 렌더링 하는 방법 ➔단일 스트림만 가능

- 인덱스의 필요성
  - Primitive의 종류에 따라 필요한 정점의 수가 차이가 심함
  - 장면 연출에 필요한 Geometry 수가 적을 수록 렌더링 속도가 우수
  - 장면에 필요한 Vertex 숫자를 조금이라도 줄이기 위해 Index를 사용

- 인덱스 종류
  - FMT_INDEX16: WORD(16비트), 한 번에 디바이스에 적용할 수 있는 인덱스의 최대 개수 2^{16}
  - FMT_INDEX32: DWORD(32비트), 한 번에 디바이스에 적용 가능한 인덱스의 최대 개수 2^{32}

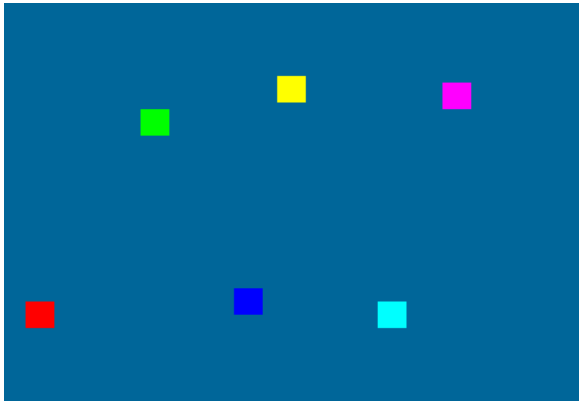
  - 16비트만 가지고도 충분히 렌더링 ➔ 이보다 큰 경우 Geometry를 분할하는 것이 유리

- 렌더링
  - 파이프라인에 연결: pDevice->SetIndices();
  - 렌더링: pDevice->DrawIndexedPrimitive();

- 렌더링 Primitive Type
  - 인덱스 버퍼, 또는 시스템의 인덱스 메모리를 사용하는 경우 Primitive Type 는 TRIANGLE_LIST

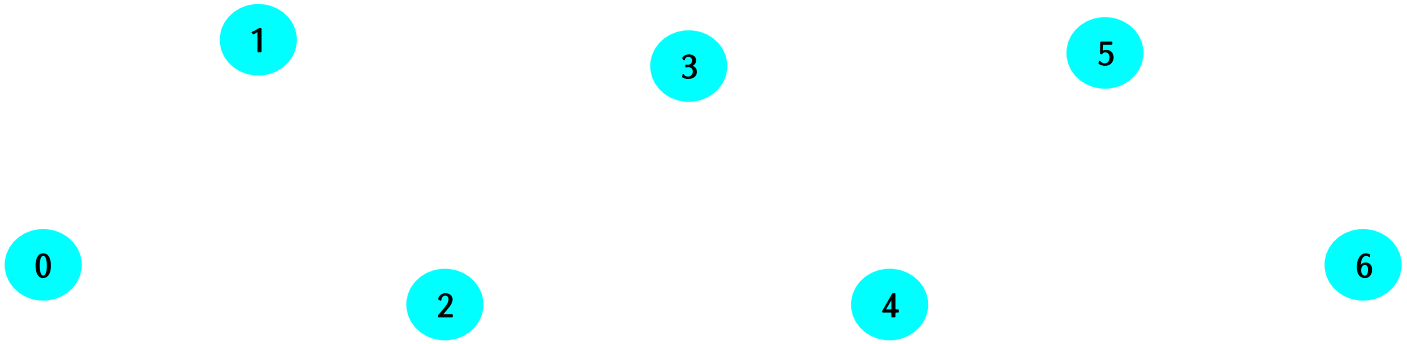
- 프리미티브 (Primitive)
  - 그래픽 출력의 기본 단위 ➔ 생명체의 세포
  - 정점을 조합해서 렌더링을 결정
- 종류
  - 점(Point), 선(Line), 삼각형(Triangle), 사각형(Quad: OpenGL)
- 타입
  - List, Strip, Fan
- 적용 방법
  - 점: Point List
  - 선: Line List, Line Strip
  - 삼각형: Triangle List, Triangle Strip, Triangle Fan
  - 사각형: Quad List, Quad Strip ← OpenGL에서만 지원

- 포인트 리스트 (Point Lists)
  - 점에 대한 단 하나의 프리미티브
  - 개별적인 점을 렌더링
  - 필요한 정점의 수 = 프리미티브 수
  - pDevice->DrawPrimitive(D3DRS_POINTLIST,...)
  - 포인트 사이즈 설정
    pDevice->SetRenderState(D3DRS_POINTSIZE,...)

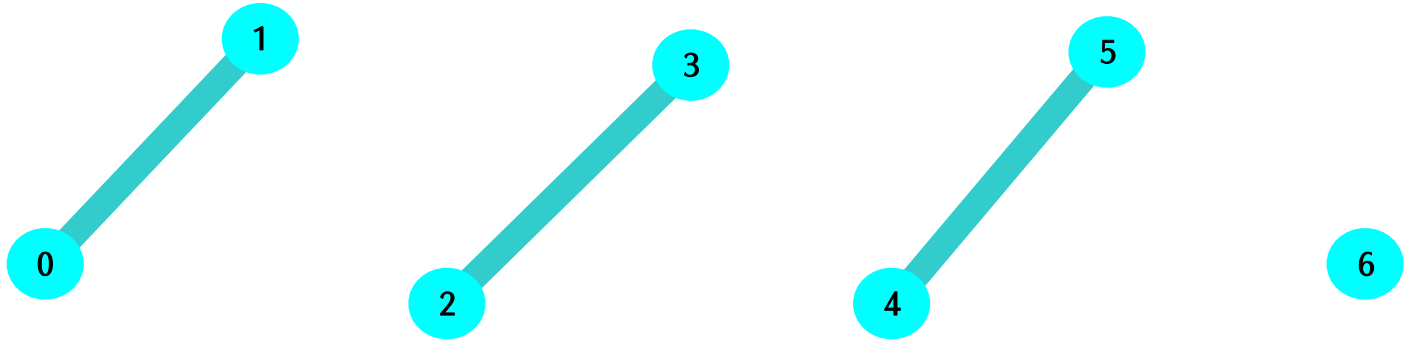


Ex)

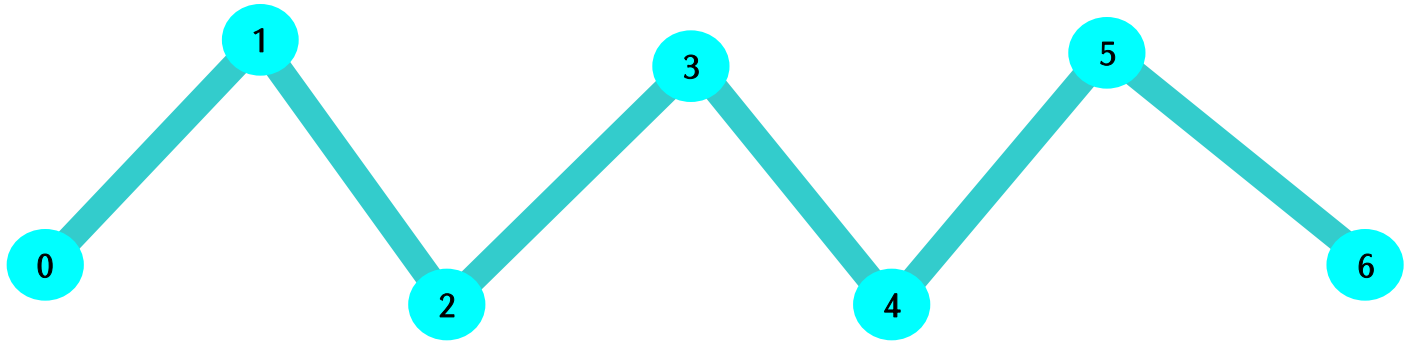
```
// 포인트 사이즈 설정
float fPointSize = 40.F;
pDevice ->SetRenderState( D3DRS_POINTSIZE, *((DWORD*)&fPointSize));
pDevice->DrawPrimitive(D3DPT_POINTLIST, 정점 시작 인덱스, 프리미티브의 개수);
```



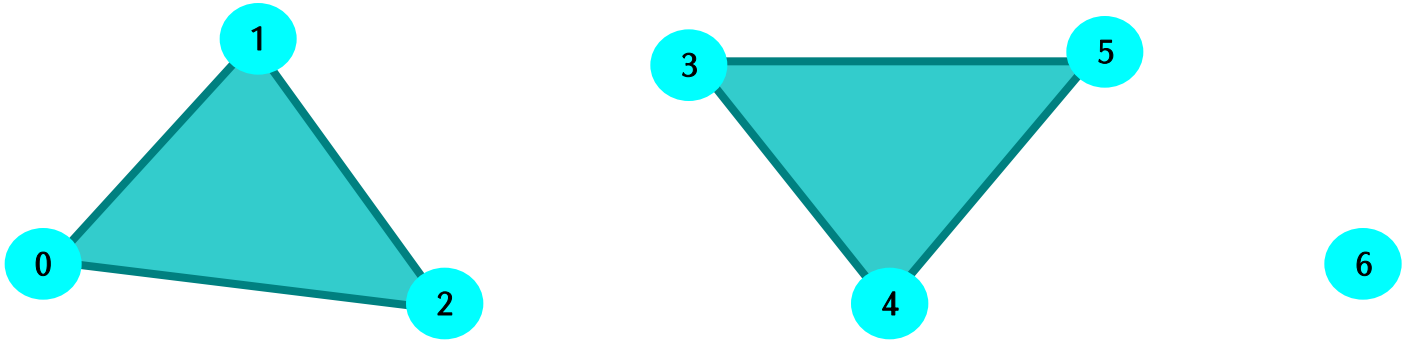
- 라인 리스트(Line Lists)
  - 두 개의 정점을 하나의 쌍으로 설정해서 라인을 그리는 방식
  - 필요한 정점의 수 = 프리미티브 수 * 2
  - pDevice->DrawPrimitive(D3DPT_LINELIST, 정점 시작 인덱스, 프리미티브의 개수);



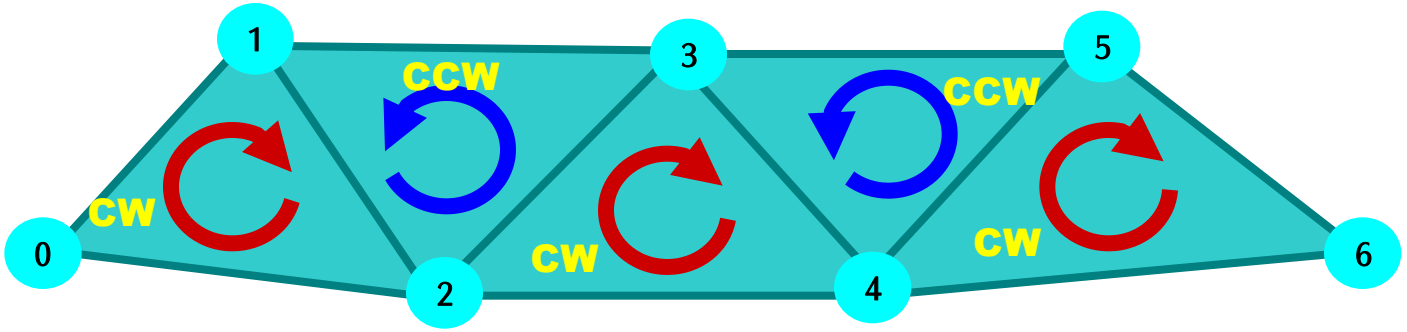
- 라인 스트립(Line Strip)
  - 직선을 정점의 연속으로 렌더링
  - 필요한 정점의 수 = 프리미티브 수 + 1
  - pDevice->DrawPrimitive(D3DPT_LINESTRIP, 정점 시작 인덱스, 프리미티브의 개수);



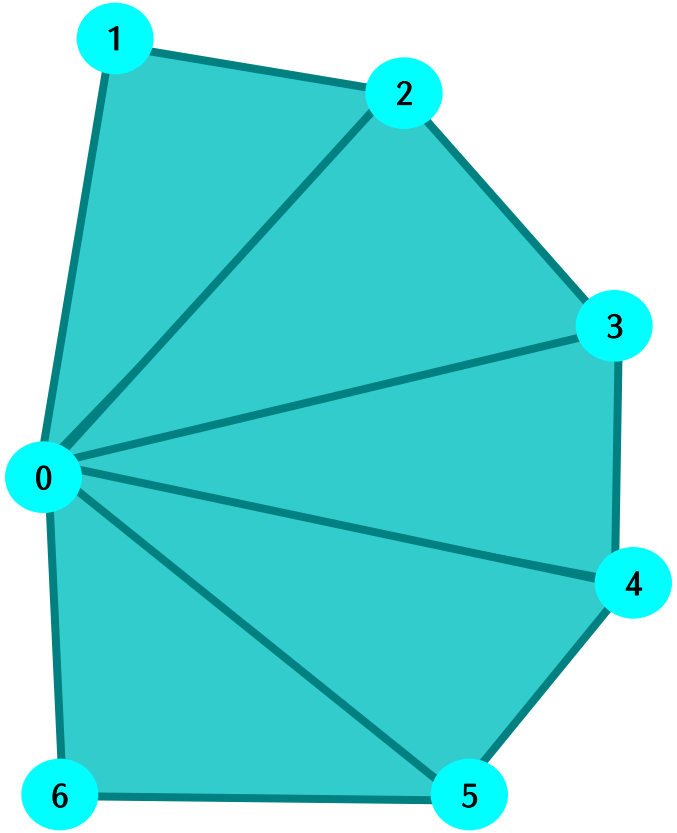
- 삼각형 리스트(Triangle Lists)
  - 3 개의 정점을 하나의 쌍으로 설정해서 삼각형을 그리는 방식
  - 필요한 정점의 수 = 프리미티브 수 * 3
  - pDevice->DrawPrimitive(D3DPT_TRIANGLELIST, 정점 시작 인덱스, 프리미티브의 개수);

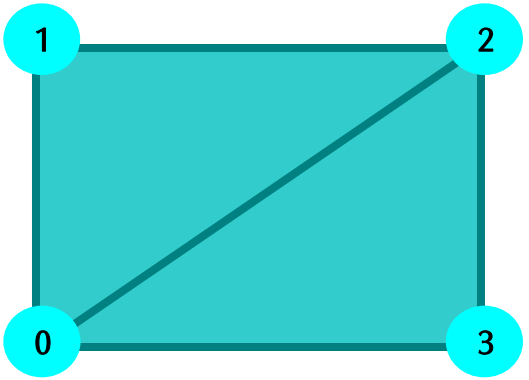


- 삼각형 스트립(Triangle Strip)
  - 렌더링 순서 0->1->2, 1->2->3, 2->3->4 , 3->4->5, 4->5->6 순으로 CW와 CCW를 번갈아 교대하면서 렌더링
  - 필요한 정점의 수 = 프리미티브 수 + 2
  - pDevice->DrawPrimitive(D3DPT_TRIANGLESTRIP, 정점 시작 인덱스, 프리미티브의 개수);



- 삼각형 팬 (Triangle Fan)
  - 삼각형으로 구성된 부채꼴 모양의 다각형을 차례대로 그리는 방식
  - 필요한 정점의 수 = 프리미티브 수 + 2
  - 렌더링 순서 0->1->2->3->4->5->6...
  - pDevice->DrawPrimitive(D3DPT_TRIANGLEFAN, 정점 시작 인덱스, 프리미티브의 개수);





- Index 사용
  - Geometry Object를 Triangle Strip, Fan의 방식으로 그릴 수 있다면 List보다 1/3 정도 메모리만 필요
  - 대부분의 Geometry Object 패턴은 대부분 List로 그려야만 하는데 Index List를 사용하면 메모리 절약 가능
  - 정점을 0번부터 차례로 설정, 삼각형을 구성하는 인덱스 리스트 작성, 인덱스에 맞게 프리미티브를 렌더링

  Ex) Triangle Strip을 인덱스로 바꾸어 보기
  v0->v1->v2 ➔ index(0,1,2)
  v1->v2->v3 ➔ index(1,2,3)
  v2->v3->v4 ➔ index(2,3,4)
  v3->v4->v5 ➔ index(3,4 5)

- 사용 메모리 크기:
  - Strip : ~프리미티브 수 * 정점 사이즈
  - Index List: 프리미티브 수 * 정점 사이즈 + Index List Size(~0) (➔ 정점 사이즈에 비해 작아음)
  - 인덱스 리스트가 WORD(16bit:2Byte) 이면 하나의 삼각형을 구성하는데 6Byte가 필요
  - 정점은 위치를 포함하므로 3개의 정점은 최소 3* sizeof(float3) = 3* 4 * 3 = 36 Byte. 게임에서는 보통 텍스처 좌표와 법선 벡터가 포함되므로 3 * (sizeof(position) + sizeof(normal) + sizeof(texture coord)) = 3 * 4 * (3 + 3 + 2) = 96Byte
  - 인덱스와 정점의 크기는 최대 6/36 = 1/6 보다 작고 일반적인 경우에는 6/96~0.07 가 되므로 인덱스를 사용하면 거의 Strip 방식에서 사용하는 메모리 크기와 비슷
  - 적은 메모리를 사용하면 속도의 이득이 있으므로 Index List를 사용하면 Strip, Fan과 비슷한 렌더링 속도

- 렌더링
  - 파이프라인에 연결: pDevice->SetIndices();
  - 렌더링: pDevice->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, ...);

  - ※ 인덱스를 사용할 때 프리미티브 타입이 Triangle List임을 주의

4. User Memory Pointers

- User Memory Pointers
  - 디바이스를 통해서 만든 정점 버퍼를 이용하지 않고, 사용자가 전역, 지역, heap등에 할당한 연속 메모리 공간을 정점 버퍼로 사용하는 방식
  - 입력 스트림을 분리할 수 없는 단일 스트림
  - 텍스처 메모리는 디바이스를 통해 생성하고, 정점은 Heap에 생성하면 저 사양 그래픽 카드를 가진 컴퓨터에서도 렌더링 가능

- 사용 방법
  - SetStreamSource() 함수가 필요 없음
  - 정점 스트림만 있는 경우: pDevice->DrawPrimitiveUP(...)
  - 인덱스 메모리가 있는 경우: pDevice->DrawIndexedPrimitiveUP(...)

Ex) 정점 Stream만 있는 경우
```
pDevice->SetFVF( ...);
pDevice->DrawPrimitiveUP( 프리미티브 타입, 프리미티브 수, 정점 Stream 시작 주소, 한 정점의 크기);
```

Ex) 인덱스와 같이 있는 경우
```
pDevice->SetFVF( FVF_VTXUV1 );

pDevice->DrawIndexedPrimitiveUP(D3DPT_TRIANGLELIST
                                , 최소 정점 인덱스 ➔ 보통 0
                                , 인덱스의 수 ➔ 삼각형의 경우 삼각형 * 3
                                , 프리미티브 수 ➔ 삼각형의 개수
                                , 인덱스 데이터 포인터
                                , 인덱스 데이터 포맷 ➔ D3DFMT_INDEX16 or D3DFMT_INDEX32
                                , 정점 Stream의 시작 주소
                                , 한 정점의 크기
                                );
```

- X, Y, Z 지시선을 그리시오.
- 프리미티브 타입 중에서 Point Lists를 이용해서 화면 보호 프로그램을 작성해 보시오.
- 직육면체를 Triangle List로 작성해 보시오.
- 위의 직육면체를 Index Buffer를 사용해 보시오.
- 삼각형에 대한 인덱스용 구조체를 작성하고, 이를 토대로 앞서 만든 직육면체를 UP를 이용해서 렌더링 해보시오.
- 직육면체를 여러 개 이용해서 로봇을 만들어 보시오.