25_ 머티리얼 제작

<제목 차례>

인천대학교 컴퓨터공학부 박종승

1. 개요

이 장에서 머티리얼의 제작에 대해 학습한다.

이전 장에서는 머티리얼의 기초에 대해서 학습하였다. 또한 메인 머티리얼 노드에 **베이스 컬러** 입력핀에 대해서 학습하였다. 이 장에서는 **베이스 컬러** 입력핀 이외의 다른 입력핀의 활용에 대해서 학습한다.

메인 머티리얼 노드의 입력핀에는 **베이스 컬러**(Base Color), 메탈릭(Metallic) 스페큘러(Specular), 러프니스(Roughness), 이미시브 컬러(Emissive Color), 노멀(Normal) 등을 포함하여 많은 입력핀들이 있다.

<참고> 입력핀에 대한 자세한 내용은 다음의 문서를 참조하자.
https://docs.unrealengine.com/material-inputs-in-unreal-engine/

2. 노말 입력 사용하기

이 절에서 노말 입력핀에 대해서 학습한다.

메인 머티리얼 노드에는 **Normal** 입력핀이 있다. 이 입력핀에는 노말맵을 입력할 수 있도록 한다. 노말맵은 깊이감을 느낄 수 있게 하는 매우 효과적인 방법이다.

지금부터 노말맵에 대해서 알아보자.
물체의 표면이 평면인 경우를 생각해보자. 평면 위의 모든 점에서 노말 벡터가 동일할 것이다. 노말 벡터가 동일하므로 빛의 강도도 동일하게 결정되고 깊이감이 느껴지지 않을 것이다.
만약 표면이 평면이더라도 표면 위의 각 지점에서 노말 벡터를 다르게 지정해준다면 어떻게 보일지에 대해서 생각해보자. 평면상의 일부 위치에서 노말 벡터가 변하도록 수정해준다면 마치 그 위치에서 표면의 각도가 튀어나오거나 안으로 들어간 것처럼 보이게 된다. 왜냐하면 그 위치에서는 실제 평면과 다른 노말 벡터를 사용하여 빛의 디퓨즈 성분이나 스페큘러 성분이 계산되기 때문이다.
이러한 원리를 활용하기 위해서 각 픽셀이 노말 벡터를 표현하는 텍스처를 만들어 활용한다. 이러한 텍스처를 노말맵이라고 한다. 노말맵은 실제로 기하 구조를 울퉁불퉁하게 바꾸는 것이 아니라 빛이 반사되어 카메라로 관찰되는 컬러를 계산할 때에 필요한 노말 성분을 수정해서 깊이가 있는 것처럼 계산되도록 하는 기법이다. 즉 울퉁불퉁하게 깊이가 있는 것처럼 착시가 일어나도록 하는 기법이다.
노말맵은 표면에 파인 홈이나 갈라진 금이나 스크래치 같은 것을 사실적으로 표현할 수 있는 효과적인 방법이다.

노말맵의 표현에 대해서 알아보자.
노말맵에서의 각 픽셀은 노말 벡터를 표현한다. 노말 벡터의 각 X,Y,Z 성분의 범위는 [-1,1]이다. 예를 들어 XY 평면인 경우에는 노말 벡터가 (0,0,1)이다. 표면이 평평하다면 노말맵의 모든 노말 벡터가 (0,0,1)일 것이다. 표면이 울퉁불퉁하다면 노말 벡터가 울퉁불퉁한 경사만큼 (0,0,1)에서 달라질 것이다. 한편 아무리 달라져도 텍스처는 항상 전면을 향해 있으므로 Z성분은 음수가 될 수는 없다. 즉 X,Y성분은 [-1,1]의 범위의 값을 가지지만 Z성분은 [0,1] 범위의 값을 가져야 한다.
노말맵은 디퓨즈 컬러를 표현하는 텍스처와 동일한 크기의 텍스처로 표현한다. 노말맵에서 픽셀의 각 성분은 한 바이트로 저장된다. 각 픽셀은 RGB값이 아닌 XYZ값을 나타낸다. 노말 벡터의 성분의 [-1,1]의 값은 [0,255]의 범위로 매핑된다. 노말 벡터의 0에 해당하는 픽셀 바이트 값이 128이다. 따라서 평면인 경우의 노말 벡터 (0,0,1)을 실제로 (128,128,255)의 픽셀값(#8080FF)에 해당한다. 텍스처가 전체적으로 연보라색으로 보인다면 노말맵을 표현하는 텍스처일 것이다.

이 절의 예제에서는 5개의 머티리얼을 만들어볼 것이다.
첫 번째 머티리얼 **M_Normal_Colorwithout**은 흰색 컬러를 **베이스 컬러** 입력핀에만 연결하고 노말을 사용하지 않는다. 이 머티리얼은 이전 예제에서 이미 구현하였지만 두 번째 머티리얼과의 비교를 위해서 추가하였다.
두 번째 머티리얼 **M_Normal_Colorwith**는 흰색 컬러와 더불어 노말을 함께 사용한다.
세 번째 머티리얼 **M_Normal_Texwithout**은 텍스처를 **베이스 컬러** 입력핀에만 연결하고 노말을 사용하지 않는다. 이 머티리얼은 이전 예제에서 이미 구현하였지만 네 번째 머티리얼과의 비교를 위해서 추가하였다.

네 번째 머티리얼은 **M_Normal_Texwith**는 텍스처와 더불어 노말도 함께 사용한다.
다섯 번째 머티리얼은 **M_Normal_Texwithweight**는 노말 입력핀에 연결하는 네트워크에서 연산을 추가하여 노말이 더욱 잘 드러나도록 한다.

이제부터 예제를 통해서 학습해보자

---

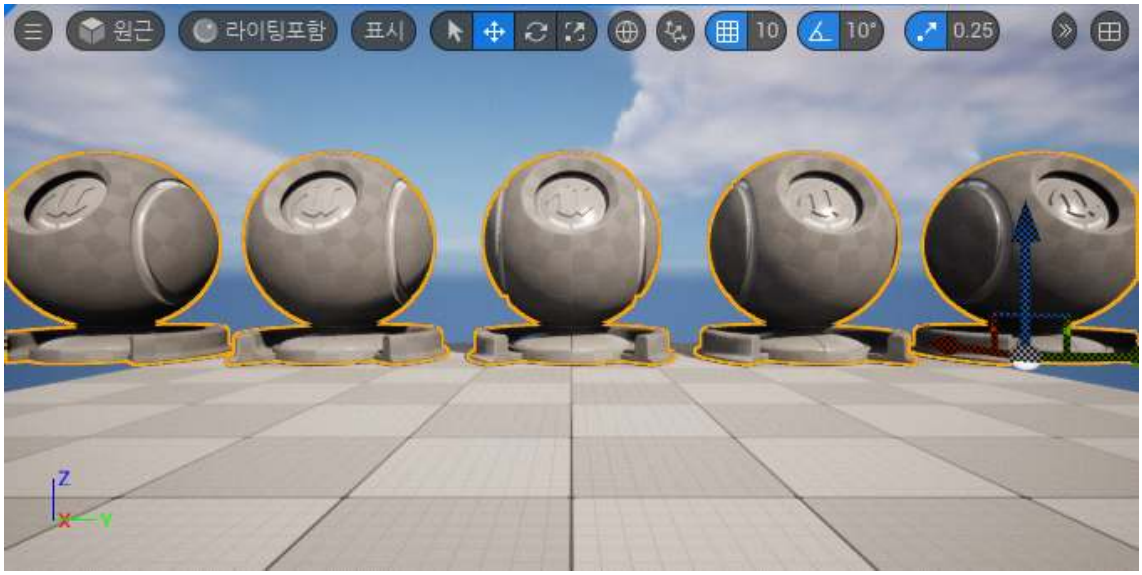
1. 새 프로젝트 **Pmtlnodenormal**를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 Pmtlnodenormal로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.
창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자. 그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 MyMap으로 저장하자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 MyMap으로 수정하자.
그리고, 툴바의 **액터 배치** 아이콘을 클릭하고 **기본** 아래의 **플레이어 스타트** 액터를 드래그하여 레벨에 배치하자. 위치를 (0,0,92)로 지정하자.
그리고, **아웃라이너**에서 **Floor**를 선택하고, **디테일** 탭에서 **스케일**을 (8,8,8)에서 (1,1,1)로 수정하자.

2. 이번 프로젝트에서 필요한 외부 애셋을 준비하자.
필요한 애셋은 **시작용 콘텐츠**에 포함되어 있다. **시작용 콘텐츠**가 포함되어 있는 다른 프로젝트를 찾아보거나 또는 **시작용 콘텐츠**가 포함되도록 새 임시 프로젝트를 만들자. 임시 프로젝트에서 **Content** 폴더 아래에서 다음의 파일들을 찾아보자. 다음의 파일들을 우리의 프로젝트로 복사하여 가져오자. 폴더 구조가 동일하게 되도록 하자. 가져온 후에 임시 프로젝트는 종료하고 삭제하면 된다.

```
StarterContent/Textures/T_Brick_Clay_New_D.uasset
StarterContent/Textures/T_Brick_Clay_New_N.uasset
```

<참고> 만약 프로젝트 생성 시에 **시작용 콘텐츠**가 포함되지 않게 생성되었다고 하더라도 이후에 추가할 수 있다. 콘텐츠 브라우저에서 **+추가** 버튼을 클릭하고 드롭다운 메뉴에서 **피처 또는 콘텐츠 팩 추가**를 선택하면 **프로젝트에 콘텐츠 추가** 창이 뜬다. 창의 상단에 있는 **콘텐츠** 탭을 클릭하고 **시작용 콘텐츠**를 선택하여 프로젝트에 추가할 수 있다.
한편, **시작용 콘텐츠**는 633MB 정도의 고용량의 콘텐츠 팩이다. 따라서 우리는 간편한 프로젝트 생성을 위해서 용량을 최소한으로 줄이도록 우리의 예제에서 필요한 파일만 선별하여 준비하였다.

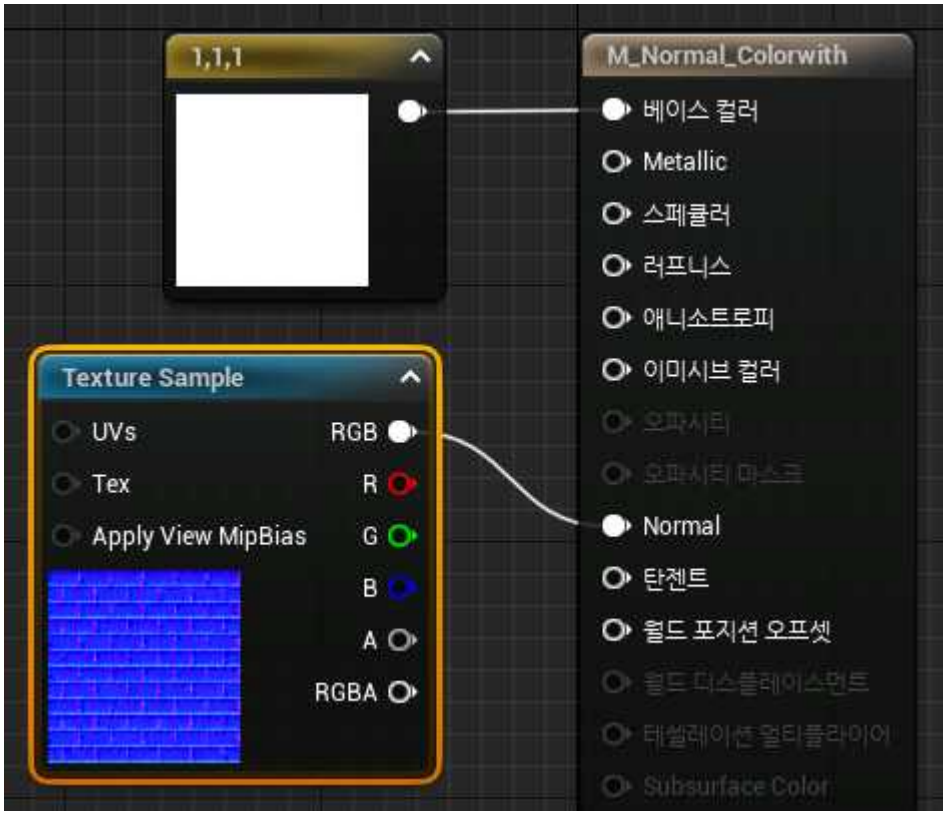
3. 이전 예제에서와 동일한 방법으로 테스트를 위한 메시 액터를 레벨에 배치하자.
콘텐츠 브라우저에서 왼쪽 탭에서 **엔진** 폴더를 선택하고 **SM_MatPreviewMesh_01**를 검색하여 배치하자. 배치된 액터의 이름을 **MeshLeftmost**로 수정하자. 위치는 (300,-600,0)으로 수정하고 회전은 (0,0,90)으로 수정하자.
배치된 액터를 Y축 방향으로 **Alt+드래그**로 4개의 복사본을 만들어 모두 5개의 액터가 되도록 하자. 배치된 복사본의 이름을 각각 **MeshMiddleleft**, **MeshMiddle**, **MeshMiddleright**, **MeshRightmost**로 수정하자. 위치는 (300,-300,0), (300,0,0), (300,300,0), (300,600,0)이 되도록 하자.
그다음, **PlayerStart** 액터의 위치를 (-450,0,92)로 수정하자.



4. 첫 번째 머티리얼을 생성하자. **콘텐츠 브라우저**에서 **콘텐츠** 폴더에서 **+추가**를 클릭하고 새 머티리얼을 추가하여 생성하고 머티리얼 이름을 **M_Normal_Colorwithout**로 수정하자.
그다음, 생성된 머티리얼 애셋을 더블클릭하여 머티리얼 에디터를 열자.
그다음, **3+좌클릭**하여 **Constant3Vector** 노드를 배치하자. 그리고, 디폴트 속성값을 1,1,1로 지정하자.
그다음, 노드를 메인 머티리얼 노드의 **베이스 컬러** 입력핀에 연결하자. 저장하자.

콘텐츠 브라우저에서 **M_Normal_Colorwithout**를 드래그하여 레벨의 **MeshlLeftmost**에 드롭하자. 메시가 흰색으로 보일 것이다.

5. 두 번째 머티리얼을 생성하자. 이번에는 첫 번째 머티리얼을 **Ctrl+C**를 누르고 **Ctrl+V**를 눌러 복사하여 생성하자. 머티리얼 이름을 **M_Normal_Colorwith**로 수정하자.
그다음, 생성된 머티리얼 애셋을 더블클릭하여 머티리얼 에디터를 열자.
그다음, **T+좌클릭**하여 **TextureSample** 노드를 배치하자. 그리고, **디테일** 탭에서 **텍스처(Texture)** 속성값에 **T_Brick_Clay_New_N** 텍스처를 검색하여 지정하자.
그다음, 노드의 RGB 출력핀을 메인 머티리얼 노드의 **Normal** 입력핀에 연결하자. 저장하자.

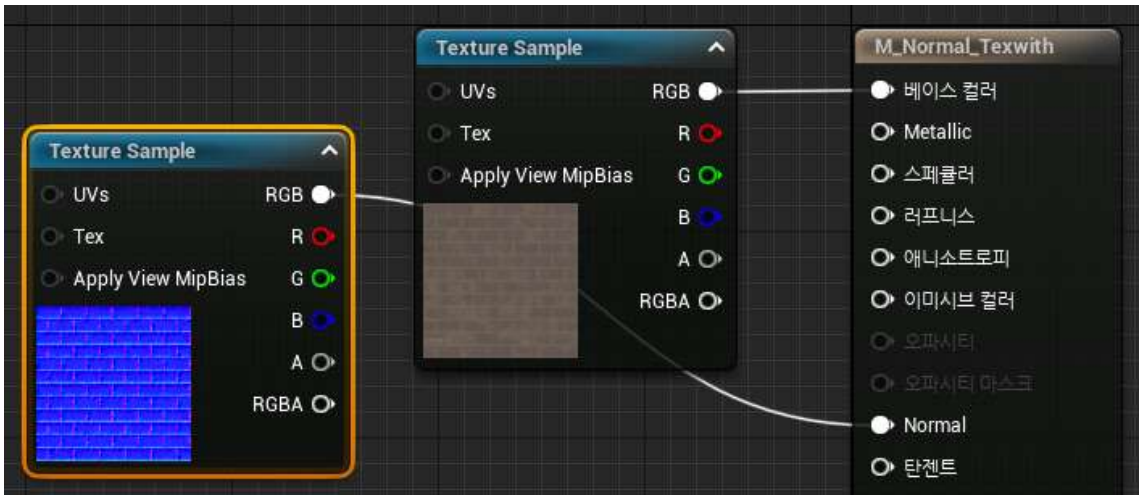


콘텐츠 브라우저에서 **M_Normal_Colorwith**를 드래그하여 레벨의 **MeshlMiddleleft**에 드롭하자. 메시가 흰색 벽돌벽 모양으로 보일 것이다.

6. 세 번째 머티리얼을 생성하자. **콘텐츠 브라우저**에서 **+추가**를 클릭하고 새 머티리얼을 추가하여 생성하고 머티리얼 이름을 **M_Normal_Texwithout**로 수정하자.
그다음, 생성된 머티리얼 애셋을 더블클릭하여 머티리얼 에디터를 열자.
그다음, **T+좌클릭**하여 **TextureSample** 노드를 배치하자. 그리고, **디테일** 탭에서 **Texture** 속성값에 **T_Brick_Clay_New_D** 텍스터를 검색하여 지정하자.
그다음, 노드의 RGB 출력핀을 메인 머티리얼 노드의 **베이스 컬러** 입력핀에 연결하자. 저장하자.

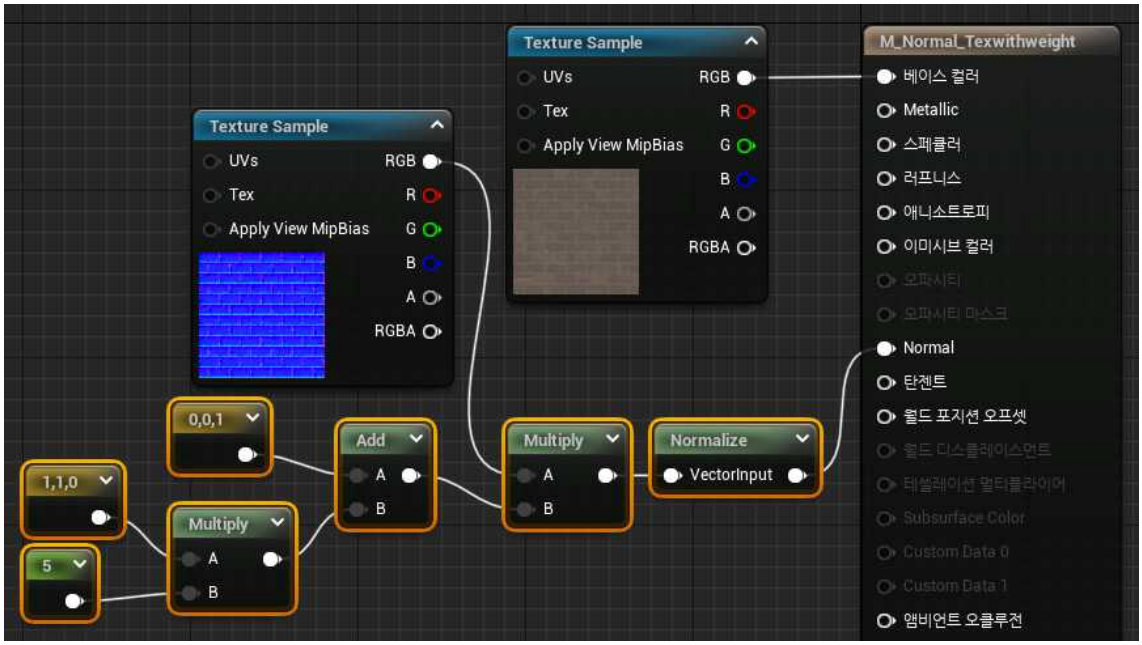
콘텐츠 브라우저에서 **M_Normal_Texwithout**를 드래그하여 레벨의 **MeshlMiddle**에 드롭하자. 메시가 벽돌벽 모양으로 보일 것이다. 그러나 입체감이 없이 도배지처럼 보일 것이다.

7. 네 번째 머티리얼을 생성하자. 이번에는 세 번째 머티리얼 **M_Normal_Texwithout**를 **Ctrl+C**를 누르고 **Ctrl+V**를 눌러 복사하여 생성하자. 머티리얼 이름을 **M_Normal_Texwith**로 수정하자.
그다음, 생성된 머티리얼 애셋을 더블클릭하여 머티리얼 에디터를 열자.
그다음, **T+좌클릭**하여 **TextureSample** 노드를 배치하자. 그리고, **디테일** 탭에서 **Texture** 속성값에 **T_Brick_Clay_New_N** 텍스터를 검색하여 지정하자.
그다음, 노드의 RGB 출력핀을 메인 머티리얼 노드의 **Normal** 입력핀에 연결하자. 저장하자.



콘텐츠 브라우저에서 **M_Normal_Texwith**를 드래그하여 레벨의 **MeshlMiddleright**에 드롭하자. 메시가 벽돌벽 모양으로 보일 것이다. 그리고 입체감도 있을 것이다.

8. 다섯 번째 머티리얼을 생성하자. 이번에는 네 번째 머티리얼 **M_Normal_Texwith**를 **Ctrl+C**를 누르고 **Ctrl+V**를 눌러 복사하여 생성하자. 머티리얼 이름을 **M_Normal_Texwithweight**로 수정하자.
그다음, 생성된 머티리얼 애셋을 더블클릭하여 머티리얼 에디터를 열자.
이제 간단한 노드 네트워크를 작성해보자.
먼저, **Constant3Vector** 노드를 배치하고 값을 (1,1,0)으로 하자.
그다음, **Constant** 노드를 배치하고 값을 5로 하자.
그다음 두 노드를 곱하는 **Multiply** 노드를 배치하자.
그다음, 두 번째 **Constant3Vector** 노드를 배치하고 값을 (0,0,1)로 하자.
그다음, 두 번째 **Constant3Vector** 노드와 **Multiply** 노드를 더하는 **Add** 노드를 배치하자.
지금까지의 노드 네트워크에 대해서 생각해보자. **Add** 노드의 결과는 (1,1,0)*5+(0,0,1)을 연산해서 (5,5,1)을 출력한다. 이 값은 벡터의 각 요소의 가중치에 해당한다. 3차원 벡터는 RGB 또는 XYZ를 표현하는데 우리는 지금 노말을 다루고 있으므로 벡터는 XYZ를 표현한다. 즉 XY 성분은 5만큼의 가중치를 더 부여하는 것에 해당한다.
그다음, 배치되어 있던 노말 텍스처의 **TextureSample** 노드와 **Add** 노드를 곱하는 **Multiply** 노드를 배치하자.
그다음, 두 번째 **Multiply** 노드의 출력을 **Normalize**하는 노드를 배치하자. 노말 벡터는 방향 벡터이므로 크기가 1이어야 한다. 따라서 크기를 1로 정규화하여 예외 상황이 발생하지 않도록 처리해주어야 한다.
그다음, **Normalize** 노드의 출력을 메인 머티리얼 노드의 **Normal** 입력핀에 연결하자. 이제 모든 노드 네트워크가 완성되었다.



노드 네트워크를 정리하자. 특히 상수 노드는 오른쪽 위의 꺾쇠 표시를 위로 올려서 프리뷰 없이 작게 보이도록 만들자. 여기에서의 상수 노드의 값은 컬러값으로 사용되는 것이 아니므로 프리뷰가 의미가 없다. 따라서 오히려 보이지 않도록 하는 것이 좋다.

위의 노드 네트워크를 살펴보자. 우리는 노말 성분 (X,Y,Z)에 대해서 X,Y에는 가중치 5를 주고 Z에는 가중치 1을 주어서 노말 성분을 변형시킨 후에 다시 정규화하였다. 즉, 노말 벡터가 X,Y 방향으로 더 틀어지도록 수정하였다. 이렇게 하면 해당 위치에 대해서 면이 더 들어가거나 튀어나오는 것처럼 보이게 된다.

이제, 콘텐츠 브라우저에서 **M_Normal_Texwithweight**를 드래그하여 레벨의 **MeshlRightmost**에 드롭하자. 메시가 벽돌벽 모양으로 보일 것이다. 그리고 이전보다 깊이감이 더욱 증폭되어 보일 것이다.

9. 다섯 번째 머티리얼을 이번에는 다른 방식으로 다시 생성해보자.

먼저, 이전 단계에서 만들어둔 노말 입력핀으로의 노드 네트워크 중에서 **TextureSample** 노드와 **Normalize** 노드를 남겨두고 6개의 노드를 모두 삭제하자.

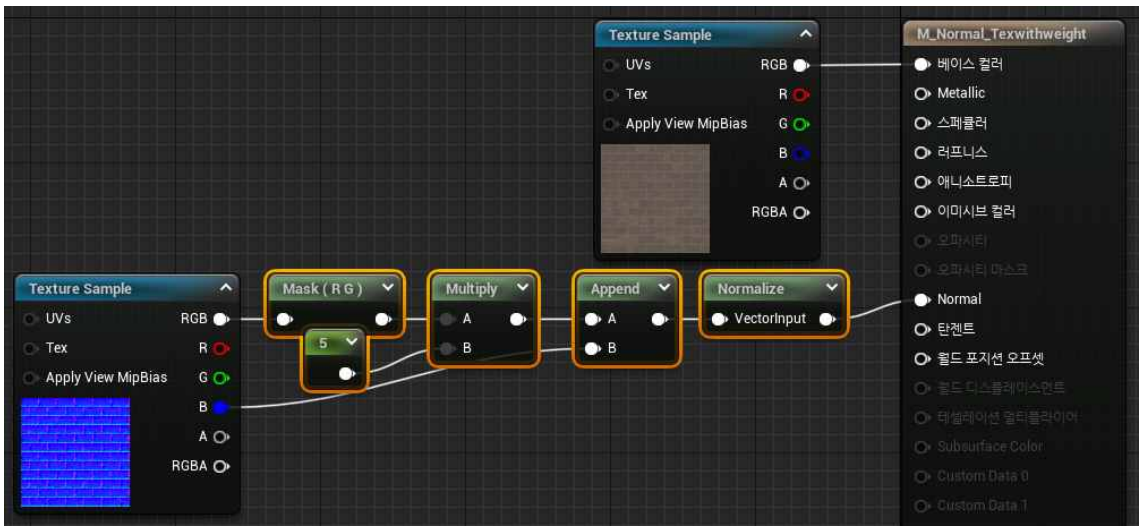
그다음, **TextureSample** 노드의 RGB 출력핀을 드래그해서 **ComponentMask** 노드를 배치하자. **Component Mask** 노드는 벡터 입력에서 일부 요소를 선택하여 출력하는 기능의 노드이다. 입력은 RGBA 중 하나 이상의 채널을 포함하면 된다. 출력은 RGBA 중에서 원하는 채널을 다중 선택할 수 있다. 디폴트로 RG가 선택되어 있다. 우리는 채널 선택을 디폴트로 그대로 두어서 RGB 입력을 받아서 RG를 추출하여 출력하도록 하자.

그다음, **1+좌클릭**하여 **Constant** 노드를 배치하고 값을 5로 지정하자.

그다음, **M+좌클릭**하여 **Multiply** 노드를 배치하고, 이전의 **Mask** 노드와 **Constant** 노드를 연결하자.

그다음, **AppendVector** 노드를 배치하고 **Multiply** 노드의 출력과 **TextureSample** 노드의 B 출력핀을 연결하자. **AppendVector** 노드는 여러 채널의 벡터를 두 입력으로 받고 이들 채널을 모두 합쳐서 벡터를 만들어 내보낸다. 예를 들어 두 개의 **Constant** 값을 입력으로 받으면 **Constant2Vector**를 출력으로 내보낸다. 이제, 우리는 중요한 두 노드를 배웠다. **AppendVector** 노드는 벡터를 합치는 기능을 제공하고 **ComponentMask** 노드는 벡터의 일부를 떼 내는 기능을 제공한다. 이들은 벡터를 자유롭게 다루는데 유용하다.

그다음, **AppendVector** 노드의 출력을 **Normalize** 노드로 연결하고 그 결과를 다시 **메인 머티리얼 노드**의 **Normal** 입력핀에 연결하자.



이전과 동일한 기능의 노드 네트워크가 완성하였다. 이런 방식으로 다양한 종류의 머티리얼 그래프를 작성하면 된다.

10. 전체적으로 다음과 같은 모습이 될 것이다.

이 절에서는 노말 입력핀에 대해서 학습하였다.

<참고> 노말맵을 생성해주는 공개 프로그램인 xNormal을 사용하면 편리하다. 다음의 사이트에서 다운받을 수 있다.

https://xnormal.net/

xNormal을 사용하여 노말맵을 생성하는 절차를 설명한 다음의 문서도 참조하자.

https://docs.unrealengine.com/4.27/RenderingAndGraphics/Textures/NormalMaps/Creation/

또한 노말맵을 설명하는 다음 문서도 도움이 된다.

https://blog.naver.com/ghdcjdfo/140209402987

3. 이미시브 컬러 입력과 다이내믹 머티리얼 인스턴스 사용하기

이 절에서 이미시브 컬러 입력에 대해서 학습하고 또한 다애내믹 머티리얼 인스턴스에 대해서 학습한다.

먼저, 이미시브 컬러 입력에 대해서 알아보자.
이미시브 컬러(Emissive Color) 입력핀은 머티리얼이 스스로 빛나게 한다. 광원과 무관하게 스스로 빛을 발산하는 효과를 낸다. 네온사인 등을 구현하기 위해서 사용한다.
입력값은 0 이상의 실수값을 사용하면 된다. 1이 넘는 큰 값에 대해서도 유효하게 동작한다.

<참고> 이미시브 컬러에 대해서는 다음의 문서를 참조하자.
https://docs.unrealengine.com/using-the-emissive-material-input-in-unreal-engine/

다음으로, 다이내믹 머티리얼 인스턴스에 대해서 알아보자.
머티리얼 인스턴스는 머터리얼 에디터에서 제작하는 머티리얼 인스턴스가 일반적이다. 이런 유형의 머티리얼 인스턴스를 상수(constant)형 머티리얼 인스턴스라고 한다. 이전 예제에서 만들어본 머티리얼 인스턴스가 이 상수형에 해당한다. 상수형 머티리얼 인스턴스는 실행 전에 한번만 계산되고 실행 중에는 변경이 불가능하다. 따라서 실행 중에는 컴파일하지 않으므로 성능상에 이점이 있다. 상수형 머티리얼 인스턴스에서도 머티리얼 파라메터를 이용하면 인자값을 컴파일하지 않고도 플레이 전이나 플레이 중에 즉시 바꾸어볼 수 있다. 그러나 에디터에서 수정해볼 수는 있지만 프로그램 내부에서 수정하는 것은 매우 어렵다. 또한 게임 실행 중에 파라미터를 변경하면 해당 머티리얼 인스턴스가 적용된 모든 액터의 외양이 바뀌게 된다.
이런 문제를 해결하기 위해서 머티리얼 인스턴스에는 다이내믹(dynamic)형도 지원하고 있다. **다이내믹 머티리얼 인스턴스**란 C++ 코드나 블루프린트 그래프 내에서 동적으로 생성한 머티리얼 인스턴스를 의미한다. 코드에서 **CreateDynamicMaterialInstance** 함수를 호출하여 만들면 된다. **다이내믹 머티리얼 인스턴스**에서는 실행 중에 인자값을 편리하게 수정할 수 있다.

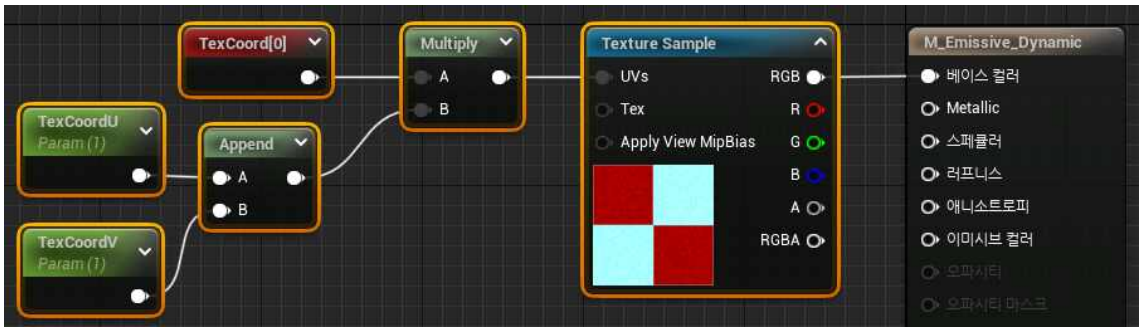
이제부터 예제를 통해서 학습해보자

---

1. 새 프로젝트 `Pmtldynamic`를 생성하자. 먼저, 언리얼 엔진을 실행하고 **언리얼 프로젝트 브라우저**에서 왼쪽의 **게임** 탭을 클릭하자. 오른쪽의 템플릿 목록에서 **기본** 템플릿을 선택하자. **프로젝트 이름**은 `Pmtldynamic`로 입력하고 **생성** 버튼을 클릭하자. 프로젝트가 생성되고 언리얼 에디터 창이 뜰 것이다.
창이 뜨면, 메뉴바에서 **파일** » **새 레벨**을 선택하고 **Basic**을 선택하여 기본 템플릿 레벨을 생성하자.
그리고, 레벨 에디터 툴바의 저장 버튼을 클릭하여 현재의 레벨을 `MyMap`으로 저장하자.
그리고, **프로젝트 세팅** 창에서 **맵&모드** 탭에서의 **에디터 시작 맵** 속성값을 `MyMap`으로 수정하자.
그리고, 툴바의 **액터 배치** 아이콘을 클릭하고 **기본** 아래의 **플레이어 스타트** 액터를 드래그하여 레벨에 배치하자. 위치를 (0,0,92)로 지정하자.
그리고, **아웃라이너**에서 **Floor**를 선택하고, **디테일** 탭에서 **스케일**을 (8,8,8)에서 (1,1,1)로 수정하자.

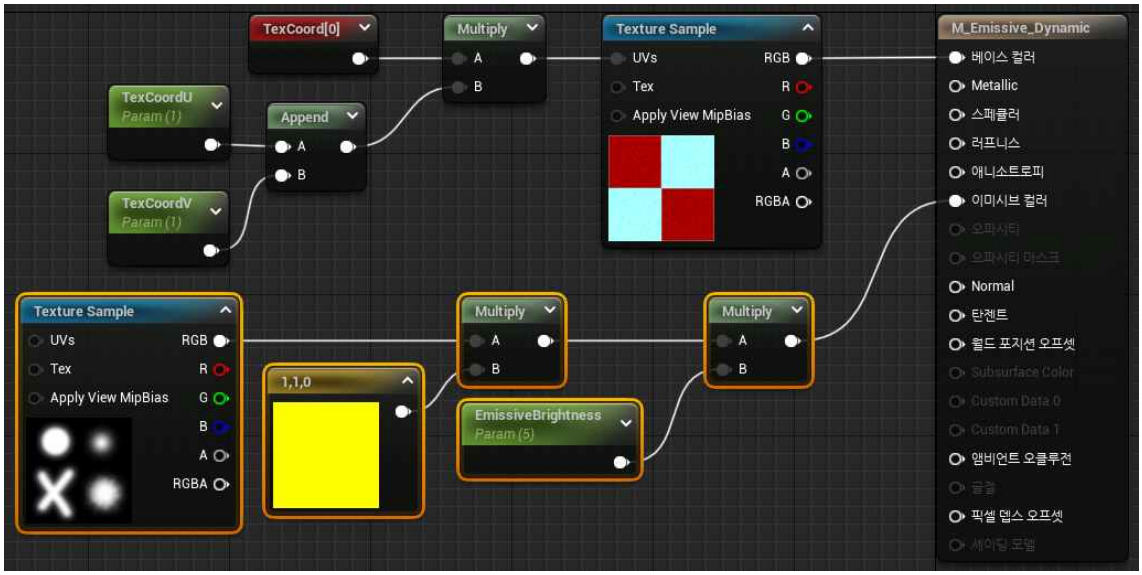
2. 머티리얼을 생성하자. **콘텐츠 브라우저**에서 **+추가**를 클릭하고 **머티리얼**을 선택하여 생성하자. 머티리얼 이름을 **M_Emissive_Dynamic**으로 수정하자.

그다음, 생성된 머티리얼 애셋을 더블클릭하여 머티리얼 에디터를 열자.
그다음, **T+좌클릭**하여 **TextureSample** 노드를 배치하자. 그리고, **디테일** 탭에서 **텍스처(Texture)** 속성값에 **T_Default_Material_Grid_M** 텍스처를 검색하여 지정하자.
그다음, 노드의 RGB 출력핀을 메인 머티리얼 노드의 **베이스 컬러** 입력핀에 연결하자.
그다음, **S+좌클릭**하여 **ScalarParameter** 노드를 두 개를 배치하자. 이름을 각각 **TexCoordU**와 **TexCoordV**로 하자. 디폴트 값을 모두 1로 지정하자.
그다음, **AppendVector** 노드를 배치하고 두 **ScalarParameter** 노드를 연결하자.
그다음, **U+좌클릭**하여 **TextureCoordinate** 노드를 배치하자. 이 노드는 2채널 벡터값 유형으로 UV 텍스처 좌표를 출력한다. 이 텍스처 좌표값을 사용하면 UV 좌표를 수정하여 다른 용도로 사용할 수 있다.
그다음, **M+좌클릭**하여 **Multiply** 노드를 배치하자. 그리고, **TextureCoordinate** 노드와 **AppendVector** 노드를 입력핀에 연결하자.
그다음, **Multiply** 노드의 출력을 **TextureSample** 노드의 UVs 입력핀에 연결하자.
이제 **베이스 컬러** 입력핀의 노드 네트워크를 완성하였다.



<참고> **TextureCoordinate** 노드의 기능을 알아보자. 노드의 출력인 UV 좌표는 [0,1]의 범위이다. 이 좌표를 스케일하여 텍스처가 반복되도록 할 수 있다. 이 기능을 타일링 기능이라고 한다. 예를 들어서 U의 표준 좌표 범위가 [0,1]인데 이것을 [0,4]로 하면 수평 방향으로 텍스처가 4번 반복되도록 매핑된다. 이러한 기능은 U 좌표와 V 좌표 각각에 대해서 독립적으로 지정할 수 있다. 타일링 기능은 노드의 디테일 탭에서 속성값을 수정해서 사용하는 방법도 있다.

3. 이미시브 컬러 입력핀에 대한 노드 네크워크를 작성하자.
먼저, **T+좌클릭**하여 **TextureSample** 노드를 배치하자. 그리고, **디테일** 탭에서 **Texture** 속성값에 **TestCard** 텍스처를 검색하여 지정하자.
그다음, **3+좌클릭**하여 **Constant3Vector** 노드를 배치하고 값을 노란색인 1,1,0으로 지정하자.
그다음, **M+좌클릭**하여 **Multiply** 노드를 배치하자. 그리고, **TextureSample** 노드의 RGB 출력핀과 **Constant3Vector** 노드를 **Multiply** 노드에 연결하자.
그다음, **S+좌클릭**하여 **ScalarParameter** 노드를 배치하자. 이름을 **EmissiveBrightness**로 하자. 디폴트 값을 5로 지정하자.
그다음, **M+좌클릭**하여 두 번째 **Multiply** 노드를 배치하자. 그리고, 첫 번째 **Multiply** 노드의 출력과 **EmissiveBrightness** 노드의 출력을 두 번째 **Multiply** 노드에 연결하자.
그다음, 두 번째 **Multiply** 노드의 출력을 메인 머티리얼 노드의 **이미시브 컬러** 입력핀에 연결하자.
이제 **이미시브 컬러** 입력핀의 노드 네트워크를 완성하였다. 저장하자.



4. 레벨 에디터로 돌아가자.

테스트를 위해서 레벨에 배치할 블루프린트 클래스를 생성하자.

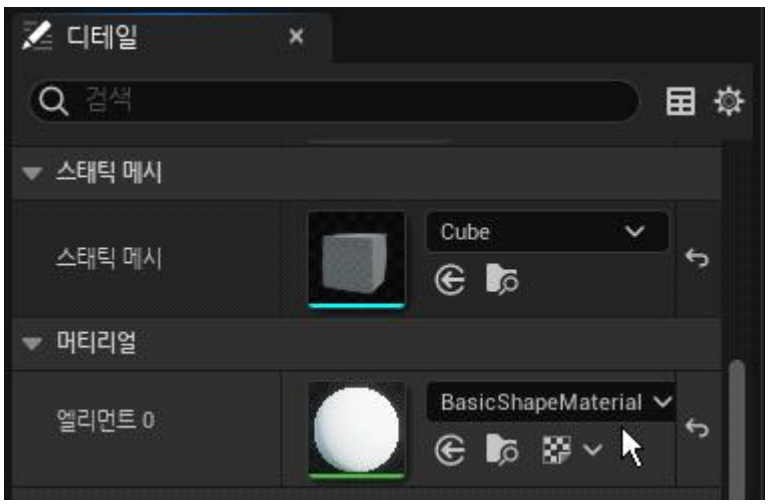
콘텐츠 브라우저에서 **+추가**를 클릭하고 **블루프린트 클래스**를 선택하여 생성하자. **부모 클래스**는 **액터(Actor)**로 하자. 이름은 **MyCube**로 하자.

그다음, **MyCube**를 더블클릭하여 블루프린트 에디터를 열자.

그다음, **컴포넌트 탭**에서 **+추가**를 클릭하고 목록에서 **큐브**를 선택하자. 큐브 모양의 스태틱 메시 컴포넌트가 추가될 것이다. 이름은 **Cube**로 그대로 두자.

이제, **Cube** 컴포넌트를 선택한 상태에서 오른쪽의 **디테일** 탭의 **머티리얼** 영역의 **엘리먼트 0** 속성값을 살펴보자. 디폴트인 **BasicShapeMaterial**로 되어있을 것이다. 그대로 두자.

우리는 이것을 우리가 만든 머티리얼인 **M_Emissive_Dynamic**로 바꾸지 않고 그대로 둘 것이다. 왜냐하면 이번에는 상수형 머티리얼 인스턴스를 지정하는 것이 아니라 다이내믹 머티리얼로 지정할 것이기 때문이다.

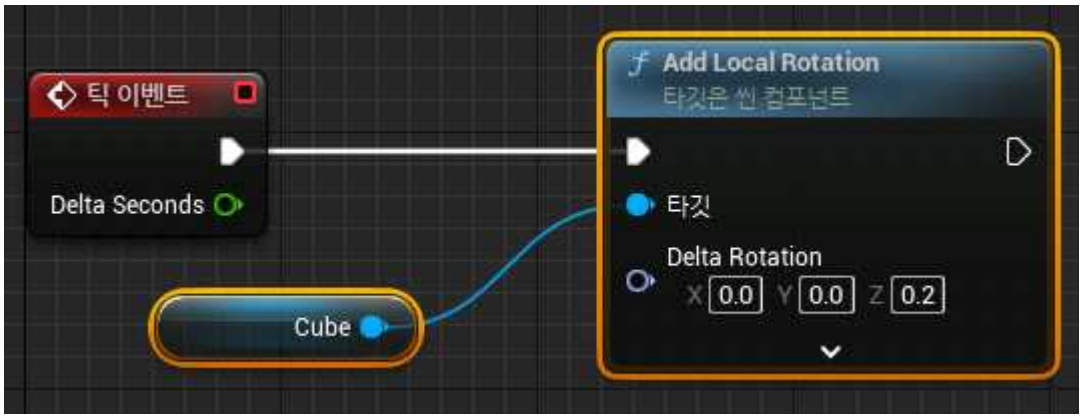


5. 이제, **MyCube** 블루프린트 에디터에서 이벤트 그래프를 작성하자.

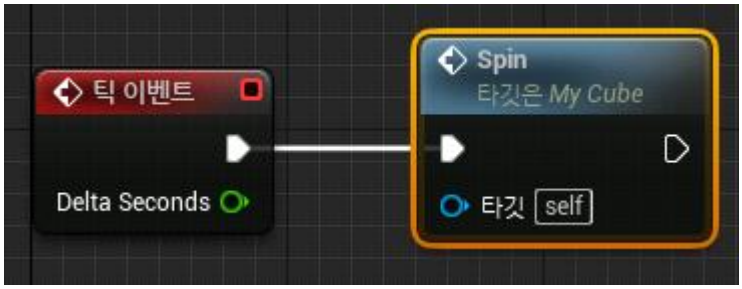
이벤트 그래프 탭을 클릭하고, **BeginPlay 이벤트** 그래프를 작성하자.

먼저, **BeginPlay 이벤트** 노드를 드래그하고 액션선택 창에서 **CreateDynamicMaterialInstance (Cube)** 노드를 검색하여 배치하자. **Cube** 레퍼런스 노드와 **CreateDynamicMaterialInstance** 노드가 함께 배치되며 연결될 것이다.
그다음, **CreateDynamicMaterialInstance** 노드의 **Source Material** 입력핀을 클릭하고 **M_Emissive_Dynamic**을 검색하여 지정하자.
그다음, **CreateDynamicMaterialInstance** 노드의 **Return Value**를 당기고 액션선택 창에서 **변수로 승격**을 선택하자. 변수의 이름을 **CubeMID**라고 수정하자.
CreateDynamicMaterialInstance 노드는 다이내믹 머티리얼 인스턴스를 생성한 후에 그 레퍼런스를 리턴해준다. 이 레퍼런스를 나중에 사용할 수 있도록 **CubeMID** 변수에 저장해두었다.

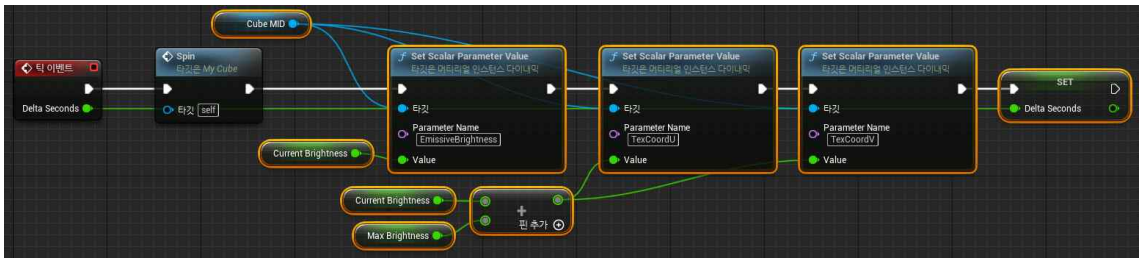
6. 계속해서 **MyCube** 블루프린트 에디터에서 이벤트 그래프를 작성하자.
이번에는 **틱(Tick) 이벤트** 그래프를 작성하자.
먼저, **틱(Tick) 이벤트** 노드를 드래그하고 액션선택 창에서 **AddLocalRotation (Cube)** 노드를 검색하여 배치하자. **Cube** 레퍼런스 노드와 **AddLocalRotation** 노드가 함께 배치될 것이다. **AddLocalRotation** 노드의 **DeltaRotation** 입력값에 (0, 0, 0.2)를 입력하자.



7. **Cube** 레퍼런스 노드와 **AddLocalRotation** 노드를 선택한 상태에서 우클릭하고 **함수로 접기**를 선택하자. 이름을 **Spin**으로 하자.

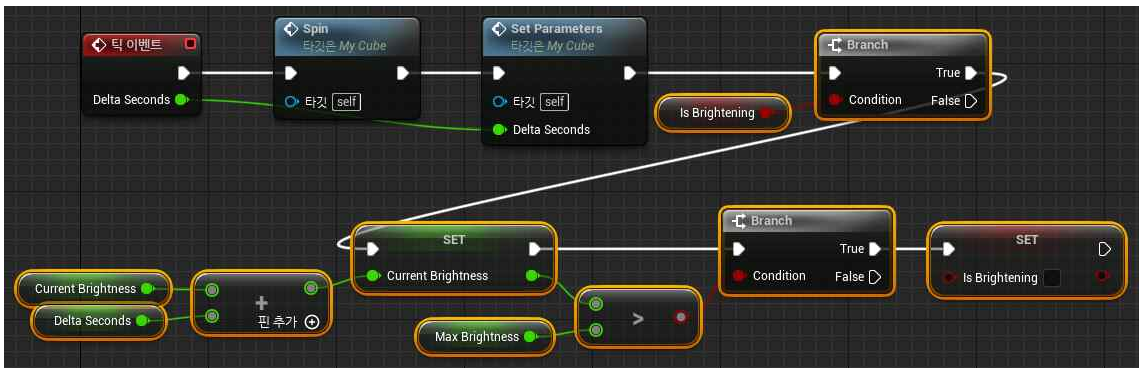


8. **Spin** 함수 호출 노드의 오른쪽에서 작업을 계속하자.
먼저, **CubeMID** 변수의 **Get** 노드를 배치하자. 그리고, **Get** 노드를 당기고 **SetScalarParameterValue** 노드를 배치하자. 다시 한번, **Get** 노드를 당기고 두 번째 **SetScalarParameterValue** 노드를 배치하자. 다시 한번, **Get** 노드를 당기고 세 번째 **SetScalarParameterValue** 노드를 배치하자. 그리고, **Spin** 함수 호출 노드와 세 **SetScalarParameterValue** 노드의 실행핀을 모두 연결하자.
SetScalarParameterValue 노드의 **ParameterName** 입력핀에는 각각 순서대로 **EmissiveBrightness**, **TexCoordU**, **TexCoordV**를 문자열로 입력하자.
그다음, **내 블루프린트** 탭에서 변수를 추가하자. 변수 **CurrentBrightness**를 **플로트** 유형으로 추가하고, 변수 **MaxBrightness**를 **플로트** 유형으로 추가하자. 컴파일하자. **MaxBrightness** 변수의 디테일 탭에서 **인스턴스 편집 가능**에 체크하자. 그리고 기본값을 1로 지정하자. 이제 우리는 레벨 에디터에서 **MaxBrightness** 변수의 기본값을 수정할 수 있게 하였다.
그다음, 변수 **CurrentBrightness**의 **Get** 노드를 배치하고 이를 첫 번째 **SetScalarParameterValue** 노드의 **Value** 입력핀에 연결하자.
그다음, 변수 **CurrentBrightness**와 **MaxBrightness**의 **Get** 노드를 하나씩 배치하고 두 노드를 더하고 그 결과를 두 번째와 세 번째 **SetScalarParameterValue** 노드 모두의 **Value** 입력핀에 연결하자.
그다음, **Tick 이벤트** 노드의 **Delta Seconds** 출력핀을 당기고 **변수로 승격**을 선택하자. 변수 이름은 **Delta Seconds** 그대로 두자. 변수 **Delta Seconds**가 생성되고 **Set** 노드가 배치될 것이다.
그다음, 세 번째 **SetScalarParameterValue** 노드의 실행핀을 이 **Set** 노드의 실행핀에 연결하자.
이제 다음과 같은 노드가 완성되었다.

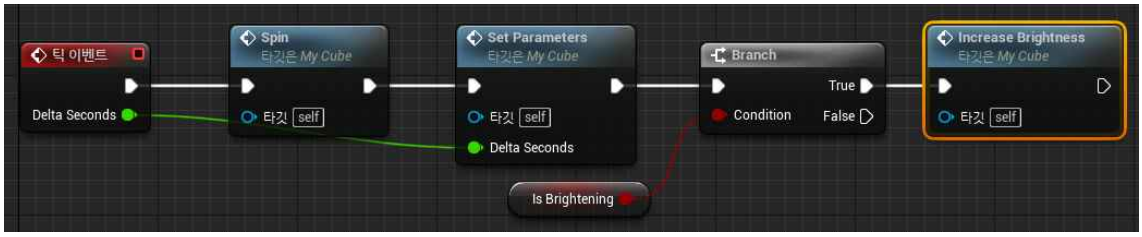


9. **Spin** 함수 호출 노드 뒤의 모든 노드를 선택한 상태에서 우클릭하고 **함수로 접기**를 선택하자. 이름을 **SetParameters**으로 하자.

10. 계속해서 이벤트 그래프를 작성하자.
먼저, 변수 **IsBrightening**을 **부울(Boolean)** 유형으로 추가하자. 컴파일하자. 변수의 **디테일** 탭에서 기본 값 영역에서 **IsBrightening** 체크박스에 체크하여 **true**로 시작되도록 하자.
그다음, **SetParameters** 함수 호출 노드의 출력 실행핀을 드래그하고 **Branch** 노드를 배치하자. 그리고 변수 **IsBrightening**의 **Get** 노드를 배치하고 이를 **Branch** 노드의 **Condition** 입력핀으로 연결하자.
그다음, **CurrentBrightness**의 **Get** 노드를 배치하고 **Delta Seconds**의 **Get** 노드를 배치하고, 이들을 더해서 그 결과를, **CurrentBrightness**의 **Set** 노드를 배치하고 연결하자. 그리고, **Branch** 노드와 **Set** 노드의 실행핀을 연결하자.
그다음, **Set** 노드의 실행핀을 당기고 다시 **Branch** 노드를 배치하자.
그리고 **MaxBrightness**의 **Get** 노드를 배치하고, **Set** 노드의 출력이 **MaxBrightness**의 **Get** 노드보다 큰 경우를 체크하는 노드를 배치하고 그 결과를 **Branch** 노드의 **Condition** 입력에 연결하자.
그다음, **Branch** 노드의 **True** 출력 실행핀을 당기고 **IsBrightening**의 **Set** 노드를 배치하자. **Set** 노드의 입력핀의 체크박스는 체크되지 않도록 하여 **false**가 지정되도록 하자. 지금까지 다음과 같이 완성되었다.

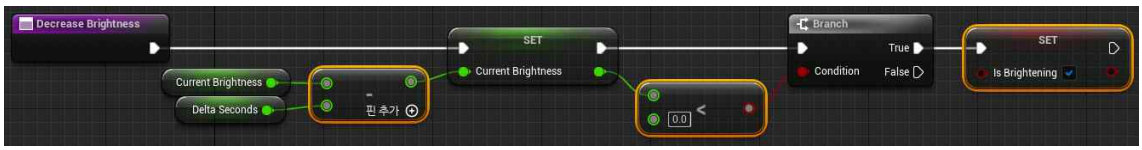


11. **Branch** 노드 뒤의 모든 노드를 선택한 상태에서 우클릭하고 **함수로 접기**를 선택하자. 이름을 **IncreaseBrightness**로 하자.

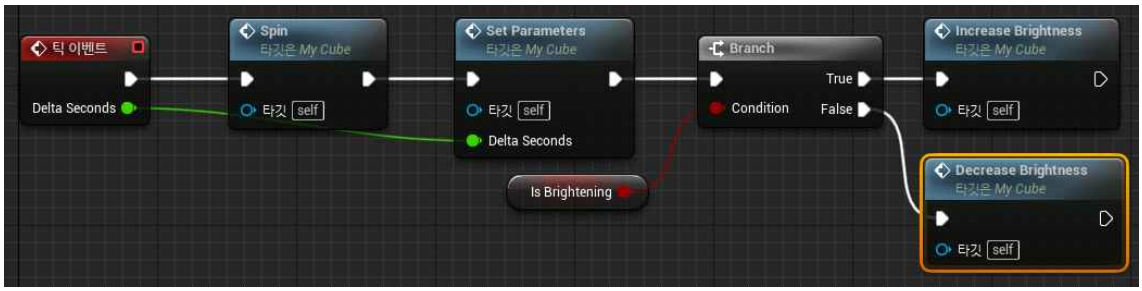


12. 다음으로, 내 블루프린트 탭에서 **IncreaseBrightness** 함수를 선택하고 **Ctrl+C**를 누르고 **Ctrl+V**를

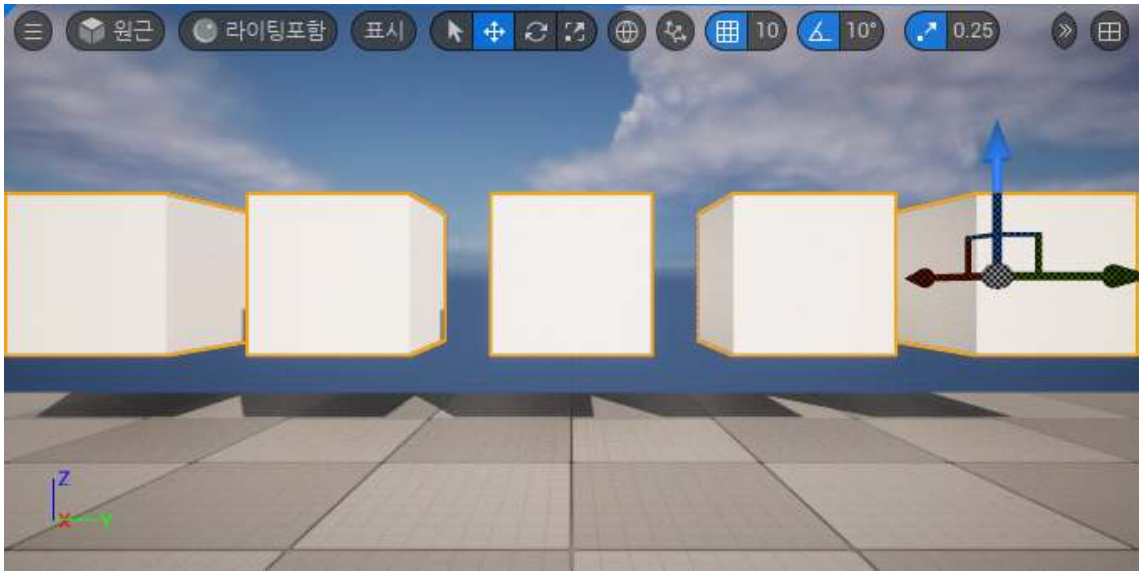
눌러 함수를 복사하자. 복사된 함수의 이름을 **DecreaseBrightness**로 수정하자.
그리고, **DecreaseBrightness**를 더블클릭하여 함수 그래프로 가자.
그다음, 함수 그래프에서 **Set** 노드의 입력핀에 연결된 + 노드를 - 노드로 교체하자.
그다음, > 노드를 < 노드로 교체하자. 그리고 두 번째 입력핀에 연결된 **MaxBrightness**의 **Get** 노드는 삭제하자. 그리고 두 번째 입력핀에는 상수값 0을 지정하자.
그다음, 마지막 **Set** 노드의 입력핀의 체크박스에 체크하여 **true**가 지정되도록 하자. 지금까지 다음과 같이 완성되었다. 컴파일하고 저장하자.



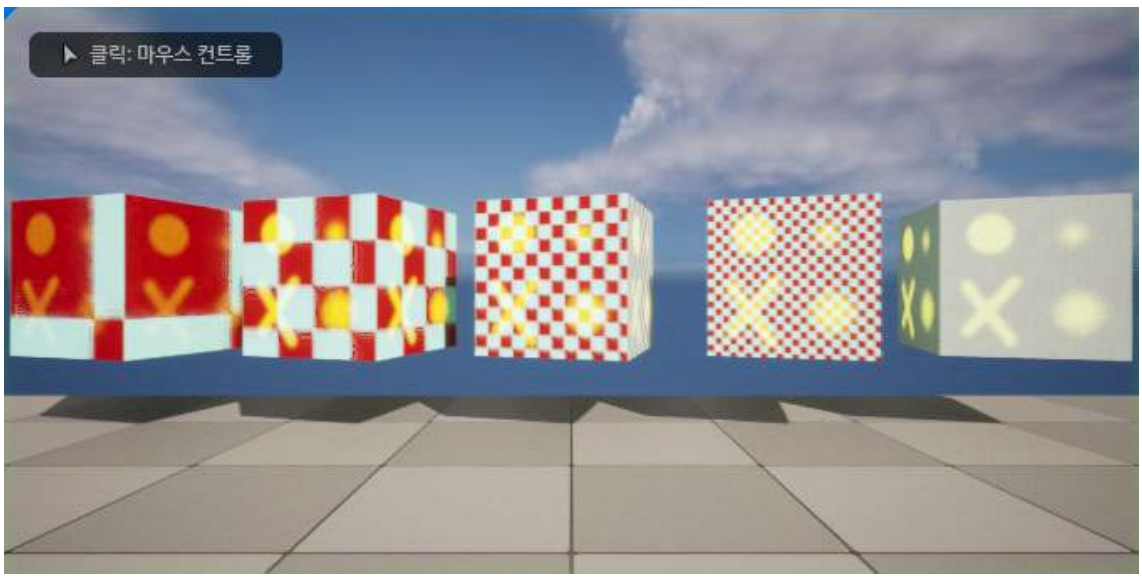
13. 이제, 이벤트 그래프 탭을 클릭하고 **틱(Tick) 이벤트** 그래프로 가자.
그리고, **Branch** 노드의 **False** 실행핀을 당기고 **DecreaseBrightness** 함수 호출 노드를 배치하자.
이제 **틱(Tick) 이벤트** 그래프가 모두 완성되었다.



14. 이제 레벨 에디터로 가자.
테스트를 위해 **MyCube** 액터를 레벨에 배치하자. **콘텐츠 브라우저**에서 **MyCube**를 드래그하여 5개를 배치하자. 각 액터의 이름을 **CubeLeftmost**, **CubeMiddleleft**, **CubeMiddle**, **CubeMiddleright**, **CubeRightmost**로 수정하자. 각 액터의 위치는 (400,-300,100), (400,-150,100), (400,0,100), (400,150,100), (400,300,100)으로 하자.
그리고, 디테일 탭에서 **전체** 탭을 선택하고 **디폴트** 영역의 **MaxBrightness** 속성값을 각각 0.5, 1, 5, 10, 50으로 지정하자.



15. 플레이해보자.



이 절에서는 이미시브 컬러 입력에 대해서 학습하고 또한 다애내믹 머티리얼 인스턴스에 대해서 학습하였다.

□