# DirectX11 배경 이미지 미표시 현상 분석 및 수정 방안

## 1. 문제 재확인

이전 분석에서 배경 이미지가 카메라 움직임에 따라 잘리는 현상을 해결하기 위해 `graphicsclass.cpp`의 `Render` 함수에서 배경 렌더링 시 `viewMatrix` 대신 항등 행렬(Identity Matrix)을 사용하도록 제안했습니다. 하지만 이 수정 이후 배경 이미지가 아예 화면에 나타나지 않는 새로운 문제가 발생했습니다.

## 2. 원인 분석

이전 수정(`viewMatrix`를 항등 행렬로 변경)은 배경 이미지가 카메라의 시점 변화에 영향을 받지 않도록 하는 올바른 접근 방식이었습니다. 배경 이미지는 3D 씬의 오브젝트와 달리 화면에 고정되어야 하므로, 뷰 행렬의 영향을 받지 않아야 합니다.

새롭게 발생한 배경 이미지 미표시 현상의 가장 유력한 원인은 **Z-값(깊이 값) 문제**입니다.

`bitmapclass.cpp`의 `UpdateBuffers` 함수를 살펴보면, 비트맵의 버텍스 Z-값이 `0.0f`로 고정되어 있습니다.

```
// bitmapclass.cpp - UpdateBuffers 함수
// ...
    vertices[0].position = XMFLOAT3(left, top, 0.0f);  // Top left.
// ...
```

반면, `d3dclass.cpp`의 `Initialize` 함수에서 프로젝션 행렬(`orthoMatrix`)을 생성할 때 `screenNear` 값은 `0.1f`로 설정되어 있습니다.

```
// d3dclass.cpp - Initialize 함수
// ...
const float SCREEN_NEAR = 0.1f;
const float SCREEN_DEPTH = 1000.0f;
// ...
m_orthoMatrix = XMMatrixOrthographicLH((float)screenWidth,
(float)screenHeight, screenNear, screenDepth);
// ...
```

DirectX의 렌더링 파이프라인에서 `screenNear` 값은 카메라에 가장 가까운 절단면(Near Clipping Plane)을 정의합니다. 이 절단면보다 Z-값이 작은 오브젝트는 렌더링되지 않고 클리핑됩니다. 따라서 `BitmapClass`에서 설정된 Z-값 `0.0f`는 `screenNear` 값 `0.1f`보다 작기 때문에, 배경 이미지가 렌더링 파이프라인 초기에 클리핑되어 화면에 나타나지 않게 되는 것입니다.

이전에는 `viewMatrix`가 항등 행렬이 아니었기 때문에, `viewMatrix`와 `worldMatrix`의 곱셈 과정에서 버텍스의 Z-값이 미묘하게 변경되어 우연히 `screenNear` 값보다 커지면서 렌더링되었을 가능성이 있습니다. 하지만 `viewMatrix`를 항등 행렬로 변경하면서 이러한 '우연한' Z-값 조정이 사라져 클리핑 문제가 명확하게 드러난 것으로 보입니다.

## 3. 수정 방안

배경 이미지가 올바르게 렌더링되도록 하려면, `BitmapClass`에서 버텍스의 Z-값을 `screenNear` 값보다 크고 `screenDepth` 값보다 작은 유효한 범위 내의 값으로 설정해야 합니다. 일반적으로 2D UI나 배경은 3D 씬의 다른 오브젝트와 겹치지 않도록 Z-버퍼를 끄고 렌더링되지만, 프로젝션 행렬에 의해 변환될 때 유효한 Z-값을 가지는 것이 중요합니다.

**`bitmapclass.cpp` 파일 수정:**

`BitmapClass::UpdateBuffers` 함수 내에서 버텍스의 Z-값을 `0.0f`에서 `0.5f`와 같이 유효한 값으로 변경합니다. `0.5f`는 `screenNear` (0.1f)와 `screenDepth` (1000.0f) 사이의 중간 값으로, 렌더링에 적합합니다.

**기존 코드:**

```
 // bitmapclass.cpp - UpdateBuffers 함수 내
// ...
    vertices[0].position = XMFLOAT3(left, top, 0.0f);  // Top left.
    vertices[0].texture = XMFLOAT2(0.0f, 0.0f);

    vertices[1].position = XMFLOAT3(right, bottom, 0.0f);  // Bottom right.
    vertices[1].texture = XMFLOAT2(1.0f, 1.0f);

    vertices[2].position = XMFLOAT3(left, bottom, 0.0f);  // Bottom left.
    vertices[2].texture = XMFLOAT2(0.0f, 1.0f);

    // Second triangle.
    vertices[3].position = XMFLOAT3(left, top, 0.0f);  // Top left.
    vertices[3].texture = XMFLOAT2(0.0f, 0.0f);

    vertices[4].position = XMFLOAT3(right, top, 0.0f);  // Top right.
    vertices[4].texture = XMFLOAT2(1.0f, 0.0f);

    vertices[5].position = XMFLOAT3(right, bottom, 0.0f);  // Bottom right.
```

```
    vertices[5].texture = XMFLOAT2(1.0f, 1.0f);
// ...
```

**수정 제안:**

```
 // bitmapclass.cpp - UpdateBuffers 함수 내
// ...
    vertices[0].position = XMFLOAT3(left, top, 0.5f);  // Top left.  <-- Z-값 변경
    vertices[0].texture = XMFLOAT2(0.0f, 0.0f);

    vertices[1].position = XMFLOAT3(right, bottom, 0.5f);  // Bottom right. <-- Z-값 변경
    vertices[1].texture = XMFLOAT2(1.0f, 1.0f);

    vertices[2].position = XMFLOAT3(left, bottom, 0.5f);  // Bottom left. <-- Z-값 변경
    vertices[2].texture = XMFLOAT2(0.0f, 1.0f);

    // Second triangle.
    vertices[3].position = XMFLOAT3(left, top, 0.5f);  // Top left. <-- Z-값 변경
    vertices[3].texture = XMFLOAT2(0.0f, 0.0f);

    vertices[4].position = XMFLOAT3(right, top, 0.5f);  // Top right. <-- Z-값 변경
    vertices[4].texture = XMFLOAT2(1.0f, 0.0f);

    vertices[5].position = XMFLOAT3(right, bottom, 0.5f);  // Bottom right. <-- Z-값 변경
    vertices[5].texture = XMFLOAT2(1.0f, 1.0f);
// ...
```

이 수정은 `graphicsclass.cpp`에서 `viewMatrix`를 항등 행렬로 설정한 것과 함께 적용되어야 합니다. 이렇게 하면 배경 이미지는 카메라 움직임에 독립적으로, 그리고 올바른 깊이 값으로 렌더링되어 화면에 다시 나타날 것입니다.

## 4. 추가 아이디어 (재언급)

배경 전용 셰이더 및 렌더링 파이프라인 분리는 여전히 장기적인 관점에서 고려해볼 만한 개선 사항입니다. 현재의 Z-값 문제를 해결한 후, 다음과 같은 이점을 위해 분리를 고려할 수 있습니다.

- **코드 명확성 및 유지보수성:** 2D UI/배경 렌더링과 3D 씬 렌더링 로직을 분리하여 코드를 더 깔끔하게 관리할 수 있습니다.
- **성능 최적화:** 2D 렌더링에 불필요한 3D 관련 계산(예: 뷰 행렬 곱셈)을 제거하여 셰이더를 더 효율적으로 만들 수 있습니다.
- **유연성:** 2D 요소에 특화된 셰이더 기능(예: UI 애니메이션, 특수 효과)을 쉽게 추가할 수 있습니다.

이러한 분리를 통해 렌더링 파이프라인을 더욱 견고하고 확장 가능하게 만들 수 있습니다.