3D게임프로그래밍
-CHAPTER11-

SOULSEEK

목차

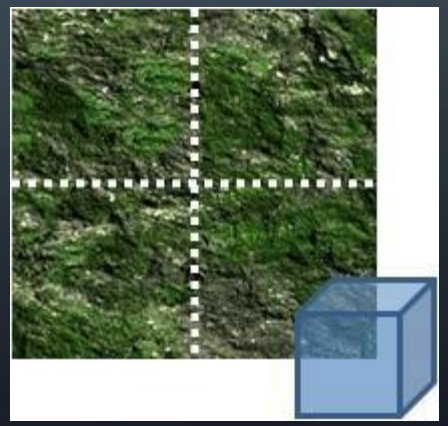
QUARDTREE CULLING

1. QUARDTREE CULLING

쿼드트리는 절두체 컬링과 결합되었을 때 그 위력을 발휘한다.
각 사분면의 중앙으로부터 가장 끝점까지의 거리를 반지름으로 하는 경계구를 사용해서 절두체 판정을 하는 것이다

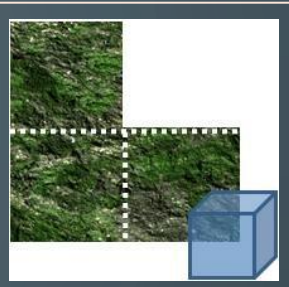
컬링의 과정

1. 컬링을 하기 전, 상자가 절두체구 지형을 1사분면부터 4사분면으로 나눈다.

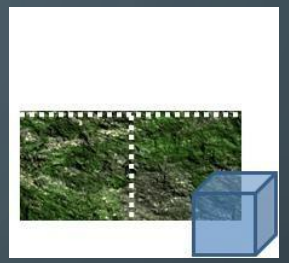


1. QUARDTREE CULLING

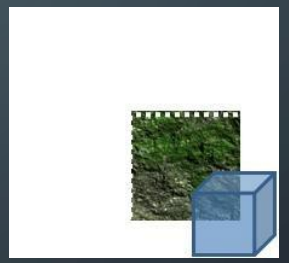
1. **1**사분면에 절두체가 없으니까 **2**사분면을 검사한다



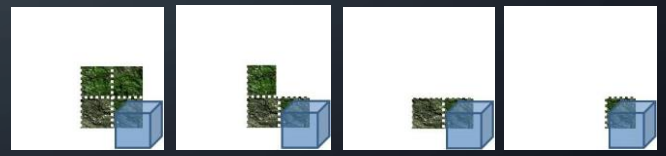
2. **2**사분면에도 절두체가 없으니까 제외시키고 **3**사분면을 검사한다



3. **3**사분면에 절두체가 없으니 제외시킨다**.**
4. **4**사분면에 절두체가 있기 때문에 **4**사분면을 검사한다**.**



똑같은 방법으로 1사분면부터 4사분면까지 검사한다.
4사분면에 절두체가 있으므로, 탐색을 중지한다.
쿼드트리의 특성을 잘 이용하면 커다란 덩어리의 영역을
한꺼번에 제외시킬 수 있다, 이러한 이유로 지형처리를 할 때
쿼드트리를 사용하는 것이다.



1. QUARDTREE CULLING

```
class ZQuadTree
{
        /// 쿼드트리에 보관되는 4개의 코너값에 대한 상수값
        enum CornerType { CORNER_TL, CORNER_TR, CORNER_BL, CORNER_BR };

        /// 쿼드트리와 프러스텀간의 관계
        enum QuadLocation
        {
                FRUSTUM_OUT = 0, /// 프러스텀에서 완전벗어남
                FRUSTUM_PARTIALLY_IN = 1, /// 프러스텀에 부분포함
                FRUSTUM_COMPLETELY_IN = 2, /// 프러스텀에 완전포함
                FRUSTUM_UNKNOWN = -1
        };

private:
        ZQuadTree* m_pChild[4]; /// QuadTree의 4개의 자식노드
        int m_nCenter; /// QuadTree에 보관할 첫번째 값
        int m_nCorner[4]; /// QuadTree에 보관할 두번째 값
        BOOL m_bCulled; /// 프러스텀에서 컬링된 노드인가?
        float m_fRadius; /// 노드를 감싸는 경계구(bounding sphere)의 반지름

private:
        /// 자식 노드를 추가한다.
        ZQuadTree* _AddChild( int nCornerTL, int nCornerTR, int nCornerBL, int nCornerBR );
        /// 4개의 코너값을 셋팅한다.
        BOOL _SetCorners( int nCornerTL, int nCornerTR, int nCornerBL, int nCornerBR );
        /// Quadtree를 4개의 하위 트리로 부분분할(subdivide)한다.
        BOOL _SubDivide(); // Quadtree를 subdivide한다.
        /// 현재 노드가 출력이 가능한 노드인가?
        BOOL _IsVisible() { return ( m_nCorner[CORNER_TR] - m_nCorner[CORNER_TL] <= 1 ); }
        /// 출력할 폴리곤의 인덱스를 생성한다.
        int _GenTriIndex( int nTris, LPVOID pIndex );
        /// 메모리에서 쿼드트리를 삭제한다.
        void _Destroy();
        /// 현재노드가 프러스텀에 포함되는가?
        int _IsInFrustum( TERRAINVERTEX* pHeightMap, ZFrustum*
        pFrustum );
        /// _IsInFrustum()함수의 결과에 따라 프러스텀 컬링 수행
        void _FrustumCull( TERRAINVERTEX* pHeightMap,
        ZFrustum* pFrustum );

public:
        /// 최초 루트노드 생성자
        ZQuadTree( int cx, int cy );
        /// 하위 자식노드 생성자
        ZQuadTree( ZQuadTree* pParent );
        /// 소멸자
        ~ZQuadTree();
        /// QuadTree를 구축한다.
        BOOL Build( TERRAINVERTEX* pHeightMap );
        /// 삼각형의 인덱스를 만들고, 출력할 삼각형의 개수를 반환한다.
        int GenerateIndex( LPVOID pIndex, TERRAINVERTEX*
        pHeightMap, ZFrustum* pFrustum );
};
```

1. QUARDTREE CULLING

- 멤버변수로 BOOL m_bCulled가 추가되었다. 이 값이 참이면 그 노드는 컬링된 것이므로 아래쪽 자식노드를 검색할 필요 없이 즉시 리턴하면 된다.
- **GenerateIndex() 함수도 절두체 컬링을 하기 때문에 절두체 객체의 포인터를 전달인자로 받고 있음을 기억**하자.

```
int ZQuadTree::GenerateIndex( LPVOID pIndex, TERRAINVERTEX* pHeightMap,
                              ZFrustum* pFrustum )
{
    _FrustumCull(pHeightMap, pFrustum);
    return _GenTriIndex( 0, pIndex ); // 삼각형의 인덱스를 만들고, 출력할 삼각형의 개수를 반환한다.
}
```

- **GenerateIndex()**함수는 먼저 **_FrustumCull()** 함수를 호출하여 절두체 컬링을 한 뒤, 필요없는 노드들의 **m_bCulled** 값을 **TRUE**로 만든다. 그리고 **_GenTriIndex()** 함수에서 **m_bCulled**값이 **FALSE**인 노드들만을 대상으로 삼각형의 인덱스를 만든다.
- 이 때, **_GenTriIndex()**함수를 잘 살펴보면 **_IsVisible()**이라는 함수가 있는데, 현재 이 함수의 역할은 리프노드인지 검사하는 것 뿐이지만, **LOD(Level Of Detail)**처리를 할 때 유용하다

1. QUARDTREE CULLING

모든 절두체 컬링은 _**FrustumCull()**함수와 _**IsInFrustum()**함수에서 처리하고 있다.

```
int ZQuadTree::_IsInFrustum( TERRAINVERTEX* pHeightMap, ZFrustum* pFrustum )
{
	BOOL b[4];
	BOOL bInSphere;
	// 경계구 안에 있는가?
	//if(m_fRadius == 0.0f) g_pLog->Log("index : [%d], Radius : [%f]", m_nCorner, m_fRadius);
	bInSphere = pFrustum->IsInSphere((D3DXVECTOR3*)(pHeightMap+m_nCenter), m_fRadius);

	if(!bInSphere)
		return FRUSTUM_OUT; // 경계구 안에 없으면 점 단위의 절두체 테스트 생략

	// 쿼드트리 4군데 경계 절두체 테스트
	b[0] = pFrustum->IsIn((D3DXVECTOR3*)(pHeightMap+m_nCorner[0]));
	b[1] = pFrustum->IsIn((D3DXVECTOR3*)(pHeightMap+m_nCorner[1]));
	b[2] = pFrustum->IsIn((D3DXVECTOR3*)(pHeightMap+m_nCorner[2]));
	b[3] = pFrustum->IsIn((D3DXVECTOR3*)(pHeightMap+m_nCorner[3]));

	// 4개 모두 절두체 안에 있음
	if((b[0] + b[1] + b[2] + b[3]) == 4)
		return FRUSTUM_COMPLETELY_IN;

	// 일부분이 절두체에 있는 경우
	return FRUSTUM_PARTIALLY_IN;
}
```

1. QUARDTREE CULLING

```
void ZQuadTree::_FrustumCull(TERRAINVERTEX *pHeightMap, ZFrustum *pFrustum)
{
	int ret;

	ret = _IsInFrustum(pHeightMap, pFrustum);

	switch(ret)
	{
		case FRUSTUM_COMPLETELY_IN : // 절두체에 완전 포함, 하위노드 검색필요없음
			m_bCulled = FALSE;
		return;
		case FRUSTUM_PARTIALLY_IN : // 절두체에 일부 포함됨, 하위노드 검색 필요
			m_bCulled = FALSE;
		break;
		case FRUSTUM_OUT :
			m_bCulled = TRUE;
		return;
	}

	if(m_pChild[0]) m_pChild[0]->_FrustumCull(pHeightMap, pFrustum);
	if(m_pChild[1]) m_pChild[1]->_FrustumCull(pHeightMap, pFrustum);
	if(m_pChild[2]) m_pChild[2]->_FrustumCull(pHeightMap, pFrustum);
	if(m_pChild[3]) m_pChild[3]->_FrustumCull(pHeightMap, pFrustum);
}
```

먼저 **_IsInFrustum()**함수는 현재 판정하려는 쿼드트리의 영역이 완전히 절두체에서 벗어나있는가**(FRUSTUM_OUT)**, 완전히 절두체에 포함되어 있는가**(FRUSTUM_COMPLETELY_IN)**, 부분적으로 절두체에 포함되어 있는가**(FRUSTUM_PARTIALLY_IN)**라는 **3**가지 경우를 판정한다. **_FrustumCull()**함수는 **_IsInFrustum()** 함수의 **3**가지 결과값에 따라서 여기서 리턴할지, 아니면, 더 하위노드로 내려갈지를 판단하게 된다.

1. QUARDTREE CULLING

- 함수의 모양에 바뀜에 따라서 **ZTerrain**의 **Draw()** 함수와 **HeightMap**의 **Render()**함수의 전달인자가 바뀌었다

```
void Render()
{
	/// 후면버퍼와 Z버퍼 초기화
	g_pd3dDevice->Clear( 0, NULL, D3DCLEAR_TARGET|D3DCLEAR_ZBUFFER,
				D3DCOLOR_XRGB(200,200,200), 1.0f, 0 );
	g_pd3dDevice->SetRenderState( D3DRS_FILLMODE, g_bWireframe ? D3DFILL_WIREFRAME : D3DFILL_SOLID );


	/// 애니메이션 행렬설정
	Animate();
	/// 렌더링 시작
	if( SUCCEEDED( g_pd3dDevice->BeginScene() ) )
	{
		g_pTerrain->Draw(g_pFrustum);

		if( !g_bHideFrustum )
			g_pFrustum->Draw( g_pd3dDevice );


		/// 렌더링 종료
		g_pd3dDevice->EndScene();
	}

	/// 후면버퍼를 보이는 화면으로!
	g_pd3dDevice->Present( NULL, NULL, NULL, NULL );
}
```

```
HRESULT ZTerrain::Draw(ZFrustum *pFrustum)
{
	LPDWORD pI;

	if( FAILED( m_pIB->Lock( 0, (m_cxDIB-1)*(m_czDIB-1)*2 *
				sizeof(TRIINDEX), (void**)&pI, 0 ) ) )
		return E_FAIL;

	m_nTriangles = m_pQuadTree->GenerateIndex( pI,
					m_pvHeightMap, pFrustum );
	m_pIB->Unlock();

	// g_pLog->Log( "Triangles=%d", m_nTriangles );
	_Render();

	return S_OK;
}
```