COLLISION DATA COMPRESSION

Homepage: https://sites.google.com/site/doc4code/

Email: goldpotion@outlook.com

2013/03/28

이 문서에서는 그래픽 물리 엔진에서 사용하는 충돌 데이터 압축방법에 대해서 생각해본다.

참고문헌 [1]의 방법을 좀 더 개선한 방법이다.

삼각형으로 이루어진 메시 데이터를 AABB 방식으로 보관하는 Binary Tree 이다.

DEFINITION

```
struct CollisionNode
{
	enum FLAG {
		MIN_X = 1,
		MIN_Y = 2,
		MIN_Z = 4,
		MAX_X = 8,
		MAX_Y = 0x10,
		MAX_Z = 0x20,
	};
	BYTE m_flag;
	BYTE m_split;			// ratio of left descendants to descendants
	BYTE m_min[3], m_max[3];
};

struct CollisionIndex {
	enum ATTRIBUTE {
		HIDDEN = 1,
		PICK = 2,
	};
```

```
        WORD attribute;
        WORD vertex[3];
};


CollisionNode NodeArray[FACE_COUNT - 1];
CollisionIndex IndexArray[FACE_COUNT];
typedef float Float3[3];
Float3     VertexArray[VERTEX_COUNT];
#define PAGE_SIZE (65536/3)
Float3 *PageTable[(FACE_COUNT + PAGE_SIZE-1)/PAGE_SIZE];
        // pointer of VertexArray element
Float3 Min, Max; // AABB
```

CollisionNode 는 AABB tree 의 중간 노드(nonterminal node)의 구조이다. CollisionIndex 는 AABB tree 의 단말 노드(terminal node)이다.

삼각형의 개수가 FACE_COUNT 라고 하면, 중간 노드의 개수는 FACE_COUNT – 1 이 된다.

CollisionIndex 에서는 VertexArray element 의 인덱스를 가진다. 그런데, VertexArray element 의 포인터를 직접 가지면 vertex 의 개수(VERTEX_COUNT)가 65537 이상이면 그 위치를 WORD 에 저장할 수 없게 된다. 그래서, PAGE_SIZE 만큼의 CollisionIndex 마다 vertex 를 별도로 모아서 저장하고, 각 Page 의 시작 위치를 PageTable 에 관리한다.

각 CollisionNode 와 CollisionIndex 는 Depth First Order 순으로 저장된다.

ALGORITHM

```
void Traverse()
{
        TraverseRecursive(0, FACE_COUNT, 0, Min, Max);
}

void TraverseRecursive(int nNodeIndex, int nFace, int nStart,
        const Float3 &vMin, const Float3 &vMax)
{
        if (!AABB_Test(vMin, vMax))
                return;
        if (nFace != 1) {
                CollisionNode *pNode = &NodeArray[nNodeIndex];

                Float3 vLMin, vLMax;
                Float3 vRMin, vRMax;
                for (int i = 0; i < 3; ++i) {
```

```
                        float e;
                        e = (vMax[i] - vMin[i]) * (1 / 256.0f) ;

                        vLMin[i] = vRMin[i] = vMin[i];
                        if (pNode->m_flag & (MIN_X << i))
                                vLMin[i] += pNode->m_min[i] * e;
                        else
                                vRMin[i] += pNode->m_min[i] * e;

                        vLMax[i] = vRMax[i] = vMax[i];
                        if (pNode->m_flag & (MAX_X << i))
                                vLMax[i] -= pNode->m_max[i] * e;
                        else
                                vRMax[i] -= pNode->m_max[i] * e;
                }

                int nLeftFace = (nFace * pNode->m_split) / 256 + 1;
                TraverseRecursive(nNodeIndex + 1, nLeftFace, nStart, vLMin, vLMax);
                TraverseRecursive(nNodeIndex + nLeftFace, nFace - nLeftFace,
                        nStart + nLeftFace, vRMin, vRMax);
        }
        else {
                // leaf node
                CollisionIndex *pIndex = &IndexArray[nStart];
                Float3 *pTable = PageTable[nStart / PAGE_SIZE];
                for (int i = 0; i < 3; ++i) {
                        Float3 &vertex = pTable[pIndex->vertex[i]];
                        printf("X:%f Y:%f Z:%f\n", vertex[0], vertex[1], vertex[2]);
                }
        }
}
```

메모리를 절약하기 위해, 이진 트리의 왼쪽 자식과 오른쪽 자식의 AABB 를 m_flag, m_min, m_max 를 사용하여 실시간으로 계산한다. 이 방법은 Miguel Gomez. “Compressed Axis-Aligned Bounding Box Trees”. In *Game Programming Gems 2*, Charles River Media, 2001, pp. 388–393. 에 나온다.

또, 이진 트리의 왼쪽 자식과 오른쪽 자식의 포인터도 절약하기 위해, 왼쪽 자손의 수를 CollisionNode 의 m_split 를 이용해 실시간으로 계산 한다.

어떤 중간 노드의 주소가 &NodeArray[**nNodeIndex**]라고 하자.

그리고, 이 노드가 **nFace** 만큼의 CollisionIndex 를 자손으로 가진다고 하고, IndexArray[**nStart**]부터 IndexArray[nStart+nFace-1]까지의 CollisionIndex 를 자손으로 가진다고 하자.

그러면, (자기 자신은 빼고) nFace-2 만큼의 CollisionNode 를 자손으로 가진다. 그 자손 노드는 NodeArray[nNodeIndex+1] 부터 NodeArray[nNodeIndex+nFace-2] 까지이다.

왼쪽 CollisionIndex 자손의 수는 (nFace * pNode->m_split) / 256 + 1 로 계산한다.

그러면, 왼쪽, 오른쪽 자식들의 CollisionNode 와 CollisionIndex 의 시작 주소를 계산할 수 있다.

트리를 생성하는 과정은 다음과 같다.

1. 우선 전체 삼각형의 AABB 를 구한다.
2. 그 다음, AABB 의 x, y, z 축 중 가장 긴 축을 선택한다.
3. 삼각형들을 무게중심이 이 축의 순서대로 정렬되도록 sorting 한다.
4. 삼각형들을 정렬된 순서에 의해 왼쪽 자손과 오른쪽 자손으로 나눈다. 이 때, 왼쪽 자손의 AABB 와 오른쪽 자손의 AABB 의 축의 길이가 비슷해지도록, m_split 을 선택하는 것이 좋다.
5. 4 번에서 구한 왼쪽, 오른쪽 자손에 대해서, 1-4 과정을 recursive 하게 반복한다.

위의 3 번 과정에서 완전히 sorting 할 필요는 없다. 미리 중간 값에 의해 왼쪽 자손과 오른쪽 자손으로 나누고, 왼쪽 자손 삼각형에 대하여 std::priority_queue(heapsort)으로 부분 sorting 한다. 그 다음, (왼쪽 자손 수가 m_split 에 의해 표시가 될 수 있도록) 왼쪽 자손 몇 개를 빼서 오른쪽 자손으로 이동시킨다.

참고문서

[1] Miguel Gomez, Compressed Alis-Aligned Bouding Box Tress
Game Programming Gems 2. 4.4.