Journal of Korea Game Society 2013 Aug; 13(4): 25-34
http://dx.doi.org/10.7583/JKGS.2013.13.4.25

게임 프로그래밍

3D 게임 공간 분할 트리에서 트리 빌드 휴리스틱

김영식
한국산업기술대학교 게임공학과
kys@kpu.ac.kr

Tree Build Heuristics for Spatial Partitioning Trees of 3D Games

Youngsik Kim
Dept. of Game & Multimedia Engineering, Korea Polytechnic University

요 약

3D 게임에서 충돌 탐지를 효과적으로 하기 위해 구성하는 공간 분할 트리는 분할 평면을 결정하는데 트리 밸런스와 분할 평면과 겹치는 폴리곤의 개수 등을 고려해야 한다. 본 논문에서는 3D 게임 공간 분할 트리에서 트리 빌드 조건에 대한 가중치를 제어하는 휴리스틱 알고리즘을 제안하였다. 가중치의 변화에 따라서 트리 빌드 시간, 분할 평면과 겹치는 폴리곤을 쪼갤 때 시각적 불일치를 유발할 수 있는 T-junction 의 제거 시간, 트리 밸런스에 따른 렌더링 속도 (frame per second) 등을 3D 게임 시뮬레이션을 통하여 분석하였다.

ABSTRACT

Spatial partitioning trees are needed for processing collision detections efficiently. In order to select split planes for spatial partitioning trees, the tree balance and the number of polygons overlapped with the split plane should be considered. In this paper, the heuristic algorithm controlling weight values of tree build criteria is proposed for spatial partitioning trees of 3D games. As the weight values are changed, tree build time, T-junction elimination time which can cause visual artifacts in splitting polygons overlapped with the split plane, rendering speed (frame per second: FPS) according to tree balance are analysed under 3D game simulations.

Keywords : 3D Game (3D 게임), Spatial Partitioning Tree (공간 분할 트리), KD-tree (KD-트리), Split Plane (분할 평면), Heuristic (휴리스틱), T-junction Elimination (T-junction 제거)

Received: Jul. 19, 2013 Accepted: Aug. 12, 2013
Corresponding Author: Youngsik Kim(Korea Polytechnic Univ.)
E-mail: kys@kpu.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

© The Korea Game Society. All rights reserved. This is an open-access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.otg/licenses/by-nc/3.0), which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

1. 서 론

3D 게임에서 AABB(axis aligned bounding box) 와 같은 경계 볼륨(bounding volume) 기반으로 충돌 처리를 한다면 장면(scene) 안의 모든 폴리곤(polygon)과 AABB 사이에 충돌 체크를 해야 하므로 시간이 많이 필요하다. 3D 콘텐츠의 크기가 작을 경우에는 렌더링 속도에 큰 문제가 없지만 거대하고 복잡한 콘텐츠를 가질 경우에는 렌더링 속도가 저하될 수 있다. 따라서 3D 게임에서 충돌 처리 및 카메라 워크에 대한 절두체 컬링(frustum culling)을 효율적으로 하기 위해서 공간 분할 트리(Spatial Partitioning Tree)를 구성 한다[1,2,3,4].

공간 분할 트리는 Quad-트리, Octree, KD-트리, BSP-트리 등이 대표적이다. Quad-트리는 각 노드마다 자식 노드가 4개이고 각 노드는 2개의 2차원 평면 축에 정렬된 유니폼 형태로 분할된다. 따라서 공간은 X축과 Z축에 대해서만 분할된다. Octree는 각 노드마다 자식노드가 8개이다. Quad-트리와 비슷하지만 각 노드는 3개의 3차원 평면 축에 정렬된 유니폼 형태로 분할된다. 따라서 3차원 X축, Y축, Z축에 대해서 유니폼 분할된다. Quad-트리에 비해서 일반적으로 더 많은 노드를 가지게 되어서 트리 탐색(tree traversal)이 길어지게 된다. 하지만 Y축에 대해서 분할하게 되므로 Y축 공간 분할을 할 수 있다.

KD-트리는 k-dimension 공간에서 폴리곤들을 조직하는 가장 잘 알려진 공간 분할 자료 구조 중에 하나이다. 각 노드는 2개의 자식을 가진다. X축, Y축, Z축 차례로 돌아가면서 축에 정렬된 평면을 기준으로 유니폼하지 않은 형태로 분할한다. Octree는 KD-트리의 특별한 형태라고 할 수 있다. KD-트리는 3D 게임 공간 분할 트리뿐 만 아니라 광선 추적(ray tracing) 기법에서 가속 구조(acceleration structure)로써 광선과 폴리곤의 교차(intersection) 검사를 위한 폴리곤 컬링(culling), 포톤 매핑(photon mapping), 정점 클라우드 모델링(point cloud modeling), 파티클 기반 유체 시뮬레이션(particle based fluid simulation)등의 여러 그래픽 응용 분야에 사용되어 왔다[5]. 또한 실감 컴퓨터 그래픽스 처리를 위하여 사용되는 광선 추적 기법은 광선과 장면의 모든 폴리곤과의 교차를 계산하기 위해서 많은 시간이 소요된다[6,7]. 따라서 일반적으로는 광선 추적을 가속화하기 위해 공간을 특정한 가속 구조(acceleration structure)로 구성하여 광선을 추적하는데, KD-트리 구조가 가장 빠르고 가장 널리 쓰인다고 알려져 있다[5].

BSP-트리는 분할 평면(split plane)이 축에 정렬되지 않지만 나머지 Quad-트리, Octree, KD-트리 등은 분할 평면이 축에 정렬(axis aligned)되어서 트리 빌드가 비교적 빠르다. back-to-front painter's 알고리즘을 적용하여 알파 블렌딩 렌더링을 해야 할 때 BSP-트리를 사용할 수 있다[1].

본 논문에서는 3D 게임 공간 분할 트리로 널리 사용되는 KD-트리를 빌드 할 때 분할 평면을 결정하는 제약조건으로 트리 밸런스와 분할 평면과 겹치는 폴리곤의 개수를 고려한다. 이를 위하여 트리 빌드 조건에 대한 가중치를 제어하는 휴리스틱 알고리즘을 제안하고 가중치 휴리스틱 변화에 따라서 트리 빌드 시간, 분할 평면과 겹치는 폴리곤을 쪼갤 때 시각적 불일치를 유발할 수 있는 T-junction 의 제거 시간, 트리 밸런스에 따른 렌더링 속도(frame per second) 등을 3D 게임 시뮬레이션을 통하여 분석하였다.

2. 관련연구

3D 게임 공간 분할 트리 혹은 광선 추적 기법의 가속 구조를 위한 연구가 많이 수행되었다. 특히 충돌 처리 혹은 광선 추적 교차 검사에서 폴리

곤 컬링을 위한 효율적인 KD-트리 빌드 최적화에 많은 연구들이 집중되었다. KD-트리 최적화의 핵심은 분할 평면을 결정하는 것이다. 분할 평면을 결정하기 위해서는 노드 분할 비용을 계산하는 것이 중요하다. 가장 간단하고 그러나 가장 많이 사용되는 방법은 spatial median splitting 이다. 이 방법은 장면에서 가장 긴 축의 spatial median을 분할 평면으로 선택하는 것이다. 그러나 이 방법은 KD-트리 빌드 최적화를 만족할 수 없다.

광선추적 교차 검사를 효율적으로 하기 위해서 KD-트리 구조를 구축할 때 각 노드의 면적을 광선이 노드와 교차하는 확률로 사용하여 분할하는 SAH(Surface Area Heuristic)[8,9] 방법이 소개되었다. SAH KD-트리가 광선 추적 기법의 정적 장면(static scene)에서 가속 구조(acceleration structure)를 위해서는 가장 좋은 방법으로 알려졌다[10]. 그러나 포톤 매핑(photon mapping)에서는 VVH(Voxel Volume Heuristic) KD-트리와 같은 다른 휴리스틱 방법이 더 효율적이라고 알려졌다[11]. 비용을 계산하는 방법으로 제안된 SAH (Surface Area heuristic)[8] 과 VVH(Voxel Volume Heuristic)[11] 은 KD-트리의 렌더링 속도를 유지해야 한다. 그러나 모든 트리 노드들의 비용을 정확히 계산하는 것은 실시간 트리 빌드를 하기 위해서는 불가능할 정도로 비용이 많이 든다. 비록 [9]에서 SAH KD-트리의 O(nlogn) 빌드 알고리즘을 고안하였으나 여전히 거대한 동적 장면에서는 비용이 많이 든다. 또한 SAH 방법은 좋은 근사치를 제공하나, 삼각형들이 이루는 장면에 대한 정보를 거의 사용하지 않는다[13]. 주위의 삼각형들의 위치 정보를 고려하여 실제 광선이 분포하는 밀도를 예측하여 KD-트리를 구축하는 SVH (Surface Visibility Heuristic)[13] 방법이 제안되었다.

비용이 적게 드는 휴리스틱 알고리즘으로 median splitting[12]과 empty space maximizing [9] 방법이 소개되었다. 또한 CPU가 아닌 GPU용 kd-트리 탐색 방법이 소개되기도 하였다[14]. 이러한 방법들도 트리 밸런싱과 분할 평면과 겹치는 폴리곤 개수는 고려하지 않는다.

본 논문에서는 이전 연구들과 달리 KD-트리 빌드 비용이 많이 들지 않고 트리 밸런싱과 분할 평면과 겹치는 폴리곤 쪼갬에 의한 오버헤드 등을 모두 고려할 수 있는 분할 평면 결정 비용 계산 휴리스틱 알고리즘을 제안한다.

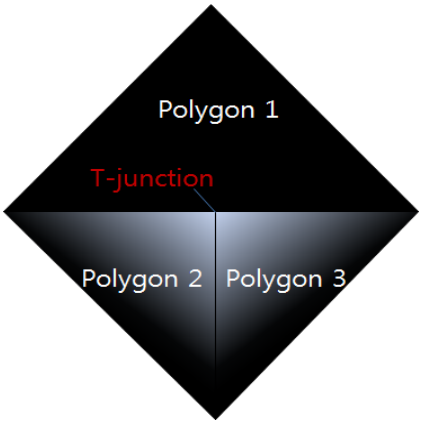
3. 트리 빌드 휴리스틱

이번 절에서는 3D 게임 공간 분할 트리에서 분할 평면과 겹치는 폴리곤을 쪼갬으로써 시각적 불일치를 유발하는 T-junction 제거와 트리 밸런스 제약 조건에 대한 가중치를 제어하는 트리 빌드 휴리스틱 알고리즘을 제안하고 설명한다.

3.1 T-junction 제거

T-junction 이란 [Fig. 1]과 같이 T 모양의 정점으로써 3D 콘텐츠 모델링 과정에서 자연스럽게 폴리곤들이 공유하는 에지에 존재하기도 하고, 3D 게임의 충돌 체크를 위한 공간 분할 트리를 구성할 때 분할 평면과 겹치는 폴리곤을 좌우 자식 트리 노드로 쪼개서 저장할 때도 발생한다[1].

[Fig. 1]은 3D 게임에서 발생하는 라이팅의 시각적 불일치를 보여준다. 정점 라이팅(vertex lighting)과 고라우드(gouraud) 쉐이딩을 사용하는 렌더링 시스템에서 Polygon 2와 Polygon 3의 정점 라이팅 소스가 Polygon 1의 에지에 존재하면 Polygon 1은 T-junction 부근에 라이팅 소스가 없으므로 시각적 불일치가 발생한다. 정점 라이팅 시스템을 사용하지 않더라도 T-junction을 공유하는 에지에서 렌더링할 때 라운딩 에러에 의해서 시각적 불일치가 발생할 수 있다.



[Fig. 1] Visual lighting disparity according to T-junction [1,3]

이러한 시각적 불일치를 유발하는 T-junction을 제거하는 의사 알고리즘[1,3] 이 [Fig. 2]에 소개되어 있다. T-junction을 제거하는 알고리즘은 우선 장면의 모든 폴리곤에 대해서 폴리곤 에지 위에 에지를 공유하는 다른 폴리곤의 정점이 존재하는지 찾는다. 폴리곤 에지 위에 다른 폴리곤의 정점이 존재하는 지 판단하기 위해서 정점과 에지를 포함한 직선 사이의 거리 D를 구해서 그 값이 0 이면 직선위에 존재하는 것이다. 공유하는 정점이 존재하면 이 정점이 T-junction 이며 제거하는 방법은 그 위치에 다른 폴리곤의 정점과 똑같은 정점을 복사하고 추가된 정점을 이용해서 폴리곤을 쪼갠다(tessellation).

[Fig. 2] T-junction 제거 의사 알고리즘은 폴리곤의 개수가 n개이면 n^2 회수만큼 테스트를 해야 하므로 계산시간이 많이 걸린다. 따라서 시각적 불일치를 유발하는 T-junction은 공간 분할 트리를 빌드할 때 분할 평면에 의해서 쪼개지는 폴리곤의 개수와 연관이 있으므로 트리 빌드 제약조건으로 고려해야 한다.

Pseudo Algorithm T-junction elimination

```
foreach (PolygonA in Scene)
 foreach (EdgeA in PolygonA)
   foreach (PolygonB in Scene)
     foreach (VertexB in PolygonB)
       D=Calc. Distance from VertexB to EdgeA
       If ( D !=0 [with epsilon] )
       else If (VertexB is on EdgeA)
            // VertexB is T-junction in EdgeA
            Copy VertexB and Insert it in EdgeA;
     end foreach
   end foreach
 end foreach
end foreach
```

[Fig. 2] A pseudo algorithm to eliminate T-junction [1,3]

3.2 분할 평면 결정 휴리스틱

3D 게임 공간 분할 트리에서 일관적인 탐색 성능을 내기 위해서는 트리 밸런스 (tree balance)가 중요하다. 만약 매우 큰 콘텐츠를 공간 분할 트리로 빌드할 때 밸런스를 유지하지 않고 한쪽으로 치우쳐서 빌드하면 탐색 시간은 일정하지 않고, 트리 깊이가 깊어져서 탐색할 때 필요한 스택의 크기가 커야 한다.

[Fig. 3]은 KD-트리에서 분할 평면을 결정하기 위해서 트리의 밸런스와 분할 평면과 겹치는 폴리곤의 개수를 모두 고려하는 휴리스틱 의사 알고리즘이다. 본 논문에서 제안하는 분할 평면 결정 휴리스틱 알고리즘은 Game Institute 3D 게임[1]의 BSP-트리에 적용된 휴리스틱 알고리즘을 수정한 방법이다.

트리 밸런스를 유지하기 위한 제약조건으로 분할 평면에 의해서 앞과 뒤로 나뉘는 폴리곤의 개수의 차를 최소화해야 한다. 그리고 시각적 불일치를 유발하는 T-junction을 야기할 수 있는 분할 평면에 의한 폴리곤 쪼개짐 개수를 최소화해야 한

다. [1]의 (eq. 1)에서 BSP-트리에 적용한 휴리스틱 알고리즘은 트리 밸런스와 분할 평면과 겹치는 폴리곤 개수를 모두 고려하지만 분할 평면과 겹치는 폴리곤 개수에 대한 가중치만 조절할 수 있다. 본 논문에서 수정하여 KD-트리에 적용한 알고리즘은 (eq. 2)와 같이 트리 밸런스와 폴리곤 쪼개짐 개수에 대한 가중치를 모두 조절할 수 있는 알고리즘이다.

[Fig. 3]에서 우선 제약조건에 대한 가중치 휴리스틱 변수 Heuristic을 정의하고 BestScore 변수는 최대값으로 초기화 한다. 그리고 축에 평행한 모든 평면이 분할 평면이 될 수 있다. Score, Splits, BackFaces, FrontFaces 변수들을 0으로 초기화 한다. 축에 평행한 평면 PlaneB에 대해서 장면에 존재하는 각각의 폴리곤 PolygonB를 다음과 같이 분류한다. 폴리곤 PolygonA가 평면 PlaneB의 앞에 존재하면 FrontFaces를 1 증가하고, 폴리곤 PolygonA가 평면 PlaneB의 뒤에 존재하면 BackFaces를 1 증가하고, 폴리곤 PolygonA가 평면 PlaneB와 겹치면 Splits을 1 증가한다.

장면의 모든 폴리곤들을 분류하면서 FrontFaces, BackFaces, Splits을 계산한 후 다음과 같이 분할 평면을 결정하기 위한 휴리스틱 수식을 계산한다.

$$Score = abs(FrontFaces - BackFaces) + Splits * Heuristic; \quad \text{(eq. 1)[1]}$$

$$Score = abs(FrontFaces - BackFaces) * (20-Heuristic) + Splits * Heuristic; \quad \text{(eq. 2)}$$

분할 평면에 의해서 앞과 뒤로 나뉘는 폴리곤의 개수의 차를 최소화하기 위해서 abs(FrontsFaces-BackFaces)를 계산한다. 그리고 분할 평면에 의한 폴리곤 쪼개짐 개수 Splits를 최소화해야 한다. 최적의 분할 평면을 선택하기위해서 Heuristic 가중치 변수를 고려한다. Heuristic 변수는 [0:20] 사이에서 변하는 정수 값이다. Heuristic 변수가 0 이면 트리 밸런스만 고려하고 폴리곤 쪼개짐 개수는 고려하지 않는다. Heuristic 변수가 1에서부터 증가할수록 폴리곤 쪼개짐 개수의 가중치가 증가한다. Heuristic 변수가 20이면 트리 밸런스는 고려하지 않고 폴리곤 쪼개짐 개수만 고려하게 된다. 이렇게 계산한 Score 변수가 BestScore 보다 작으면 BestScore 변수는 Score 변수 값으로 변경하고, PlaneB를 분할 평면으로 결정한다.

Heuristic 변수의 범위 [0:20]는 트리 밸런스와 폴리곤 쪼개짐 개수의 제약조건에 대한 가중치 변화의 정밀도를 나타내며 실험을 통하여 적절한 범위의 정밀도를 선택한 것이다.

Pseudo Algorithm Split Plane Selection Heuristic

```
Parameter = "Heuristic";
BestScore = Infinitely High;
foreach (PlaneB in parallel with Axis)
 Score = Splits = BackFaces = FrontFaces = 0;
 foreach (PolygonA in Scene)
  Classify (PolygonA, PlaneB):
  if (Polygon is in Front of PlaneB)
    FrontFaces ++;
  if (Polygon is in Behind of PlaneB)
    BackFaces ++;
  if (Polygon is overlapped with PlaneB)
    Splits ++;
 end foreach
 Score = abs(FrontFaces - BackFaces)
         * (20-Heuristic) + Splits * Heuristic ;
 if (Score < BestScore)
   BestScore = Score;
   Split Plane = PlaneB;
end foreach
```

[Fig. 3] A heuristic pseudo algorithm to select the split plane

4. 실험 및 평가

본 논문에서는 Game Institute 3D 게임[1]에서 정적 콘텐츠의 충돌 체크 및 절두체 컬링 (frustum culling)을 위해서 사용하는 공간 분할 트리들 중에

서 KD-트리를 실험에 사용하였다. Game Institute 3D 게임의 KD-트리 빌드에서 3.2절에서 제안한 분할 평면 결정 휴리스틱 알고리즘을 적용하도록 프로그램을 수정하여 실험하였다. 실험은 MacBookPro MC226KH/A 모델 (Intel(R) Core(TM)2 Duo CPU T9600 @ 2.80GHz 2.80GHz 4GB RAM)에서 측정하였다.

[Fig. 4]는 Game Institute 3D 게임의 첫 장면이다. Game Institute 3D 게임의 공간 분할 트리는 KD-트리와 BSP-트리 2가지를 사용한다. 건물 전체의 대부분은 KD-트리를 사용하고 일부 투명 건물 중앙을 표현하기 위해서 BSP-트리를 사용하였다. BSP-트리를 구성하면 투명 건물을 표현하기 위해서 알파블렌딩을 위한 painter's 알고리즘을 적용하기 쉽다. 즉 카메라 시점에서 back-to-front 정렬을 시키면서 렌더링하기 쉽다. Game Institute 3D게임에서 KD-트리로 구성하는 정적 건물 콘텐츠의 크기는 mesh 개수 = 679, texture 개수 = 63, material 개수 = 17, vertex 개수 = 46146, polygon 개수 = 15382 이다. BSP-트리로 구성하는 건물 중앙의 반투명 부분의 콘텐츠 크기는 polygon 개수 = 164 로 적다. 이와 같이 KD-트리와 BSP-트리 중에서 비교적 콘텐츠의 크기가 큰 KD-트리를 실험에 사용하였다.

[Fig. 5] 는 3.2절의 [Fig .3] 분할 평면 결정 휴리스틱 의사 알고리즘에 의해서 가중치 휴릭스틱 변수 Heuristic를 [0:20] 범위에서 변화를 주었을 때 폴리곤 쪼개짐 개수를 나타낸다. 3.2절에서 분할 평면을 결정하기 위한 수식 (eq. 1)에서 예상할 수 있듯이 Heuristic 변수를 증가하면 폴리곤 쪼개짐 개수의 가중치를 더 크게 하여 폴리곤 쪼개짐 개수가 줄어든다. [Fig. 5]에서 보듯이 Heuristic 변수의 값이 0에서 9까지 변화할 때는 폴리곤 쪼개짐 개수가 줄어들지 않고 10이상으로 증가할 때 줄어들기 시작한다. 그 이유는 실제 Heuristic 변수 범위가 [0:9] 일 때는 Splits 값이 최대값으로 변화가 없다가 Heuristic 변수를 10 이상 증가하면 폴리곤 쪼개짐에 대한 가중치가 트리 밸런스 가중치 abs(FrontFaces-BackFaces) 보다 크게 되어 Splits 값이 감소하는 것이다.

[Fig. 4] Game Institute 3D Game [1]

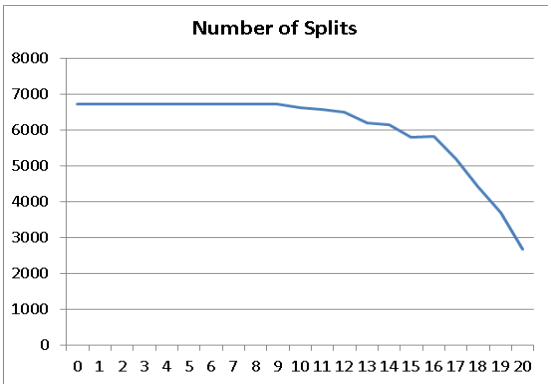
[Fig. 6]은 가중치 휴릭스틱 변수 Heuristic를 [0:20] 범위에서 변화를 주었을 때 T-junction 제거 시간을 나타낸다. T-junction 제거는 3.1절의 [Fig. 2] 알고리즘을 따른다. Heuristic 변수를 증가하면 폴리곤 쪼개짐 개수가 줄어들고 폴리곤 쪼개짐 개수가 줄어들면 T-junction의 개수도 같이 줄어들게 되어 T-junction 제거 시간이 줄어들게 된다. 흥미로운 사실은 [Fig. 5]의 폴리곤 쪼개짐 개수와 [Fig. 6]의 T-junction 제거 시간은 Heuristic 변수가 [0:19] 범위에서 변화할 때는 대체로 유사한 상관관계를 나타낸다. 그러나 Heuristic 변수가 20일 때는 유사한 상관관계를 따르지 않고 T-junction 제거 시간이 커진다. 그 이유는 Heuristic 변수가 20이면 트리 밸런스를 전혀 고려하지 않는다. 또한 트리의 깊이가 깊어져서 T-junction 제거를 위한 트리 탐색 시간이 더 큰 영향을 미치는 것으로 생각된다.

[Fig. 7]은 가중치 휴릭스틱 변수 Heuristic를 [0:20] 범위에서 변화를 주었을 때 렌더링 속도 (FPS: Frame Per Second)를 나타낸다. 그런데 렌더링 속도는 KD-트리 콘텐츠만 반영하는 것이 아

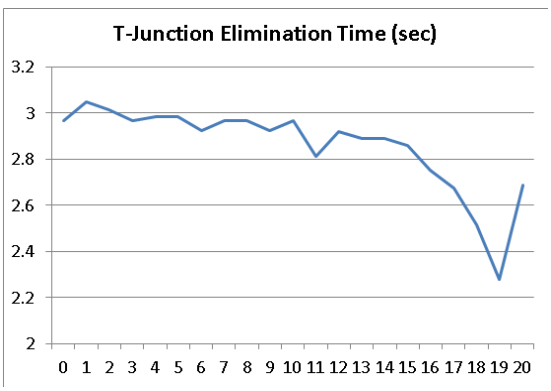
니라 건물 중앙의 투명 건물 BSP-트리 콘텐츠와 솔저 플레이어 렌더링도 같이 반영하는 것이라서 KD-트리 품질의 일부를 반영하는 것이다. 그러나 콘텐츠 중에서 KD-트리 콘텐츠가 차지하는 비중이 가장 크다. [Fig. 7]의 렌더링 속도는 Heuristic 변수가 [0:20] 사이에 변할 때 양 끝으로 가면서 저하되는 것을 보인다. 특히 Heuristic 변수가 20에 가까워지면 트리 밸런스를 유지하기 어려워 져서 트리 탐색 시간의 증가에 따라서 렌더링 속도가 저하된다.

[Fig. 8]은 가중치 휴릭스틱 변수 Heuristic를 [0:20] 범위에서 변화를 주었을 때 트리 빌드 시간과 T-junction 제거 시간을 보여준다. Heuristic 변수를 증가하면 [Fig. 6]에서 알 수 있듯이 T-junction 제거 시간이 줄어든다. 그러나 반대로 트리 빌드 시간은 트리 밸런스가 나빠지고 트리 깊이가 깊어지므로 서서히 증가하게 된다. 결과적으로 Heuristic 변수[0:18] 범위에서는 트리 빌드 시간과 T-junction 제거 시간을 합한 값은 대체로 크게 변화가 없다. Heuristic 변수가 [19:20] 에서는 트리 빌드 시간이 T-junction 제거 시간이 줄어든 것에 비하여 더 크게 증가하여 결과적으로 합한 값은 더 크게 된다. 따라서 Game Institute 3D 게임에서 트리 빌드 시간과 T-junction 제거 시간의 합, 렌더링 속도 등을 성능 지표로 고려할 때 분할 평면을 결정하는 Heuristic 변수 값을 [4:17] 범위에서 결정할 수 있다.

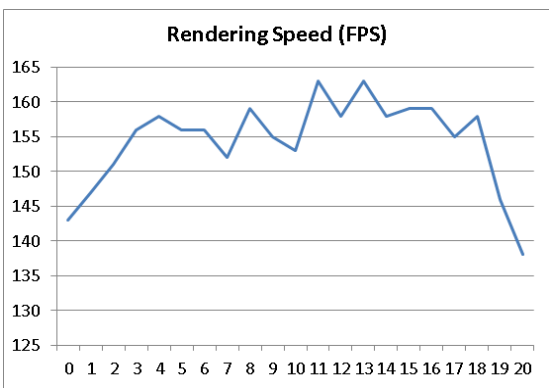
그런데 만약 트리 탐색 알고리즘이 더 효율적이고 폴리곤 쪼갬과 T-junction 제거 시간이 더 큰 비중을 차지하게 되면 Heuristic 변수의 값을 더 크게 설정해야 한다. 그리고 정점 라이팅 시스템을 사용하지 않거나 렌더링 라운딩 에러가 발생할 가능성이 적어서 T-junciton 제거의 비중이 크지 않으면 Heuristic 변수 값을 더 작게 설정해야 한다.



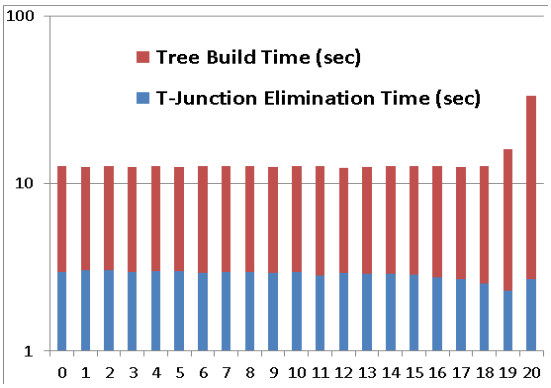
[Fig. 5] The number of polygon splits according to changing weight heuristics [0:20]



[Fig. 6] T-junction elimination time (sec) according to changing weight heuristics [0:20]



[Fig. 7] Rendering speed (FPS: Frame Per Second) according to changing weight heuristics [0:20]



[Fig. 8] Tree build time and T-junction elimination time(sec) according to changing weight heuristics[0:20]

5. 결 론

본 논문에서 3D 게임 공간 분할 트리에서 트리 밸런스와 분할 평면과 겹치는 폴리곤의 쪼개짐 개수 등을 고려하는 분할 평면 결정 휴리스틱 알고리즘을 제안하고 분석하였다. 가중치 휴리스틱 변수를 증가하면 트리 밸런스가 유지되기 어려워서 트리 빌드 시간이 커지고, 반대로 분할 평면과 겹치는 폴리곤을 쪼갤 때 시각적 불일치를 유발할 수 있는 T-junction 의 제거 시간은 감소한다. 따라서 트리 밸런스와 T-junction 제거 시간의 비중에 따라서 가중치 휴리스틱을 변경할 수 있는 알고리즘을 제시하였다.

REFERENCES

[1] Game Institute, Graphics Programming with Direct X: Module II (Book 2) chapter 14, chapter 16, E-INSTITUTE PUBLISHING INC, www.gameinstitute.com, 2004.

[2] Eric Lengyel, Mathematics for 3D Game Programming and Computer Graphics, Second Edition, Charles River Media, Inc., 2004.

[3] Youngsik Kim, "T-junction Elimination for Spatial Partitioning Trees of 3D Games," Digital Contents Society Conference 2009, Vol. 10, No. 1, 2009.

[4] Christer Ericson. Real-Time Collision Detection. The Morgan Kaufman series in interactive 3D technology. Morgan Kaufmann, 2005.

[5] Kun Zhou, Qiming Hou, Rui WangBaining Guo, "Real-time KD-tree construction on graphics hardware," ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH Asia 2008 Volume 27 Issue 5, December 2008.

[6] Jae-Ho Nah, Woo-Chan Park, Tack-Don Han, "A Shadow Culling Algorithm for Interactive Ray Tracing," Journal of Korea Game Society, Vol. 9, No. 6, pp.179~190, 2009

[7] Sangduk Kim, Jin-Woo Kim, Woo-Chan Park, Tack-Don Han, "Bounding Box based Shadow Ray Culling Method for Real-Time Ray Tracer," Journal of Korea Game Society, Vol. 13, No. 3, pp.85~94, 2013.

[8] Goldsmith, and J. Salmon, "Automatic creation of object hierarchies for ray tracing," IEEE Computer Graphics and Applications, Volume 7, Issue 5, pp14-20. May 1987.

[9] I. Wald, and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in O(Nlog N)," In Proceedings of IEEE Symposium on Interactive Ray Tracing, pp61-69, 2006.

[10] G. Stoll, "Part II: Achieving real time - optimization techniques," In SIGGRAPH 2005 Course on Interactive Ray Tracing, 2005.

[11] I. Wald, J. G¨unther, and P. Slusallek, "Balancing considered harmful - faster photon mapping using the voxel volume heuristic," In Proceedings of Eurographics'04, pp595-603, 2004.

[12] V. Havran, "Heuristic Ray Shooting Algorithms," PhD thesis, Czech Technical University in Prague, 2001.

[13] Hyuk Kim, "KD-Tree construction using surface visibility heuristic," Dept. of computer engineering, Sogang Univ., Master thesis, 2010.

[14] Byungjoon Chang and Insung Ihm, "Performance Analysis and Enhancing

Techniques of Kd-Tree Traversal Methods on GPU," Journal of KIISE:Computing Practices and Letters, Vol. 16, No. 2, pp171 ~185, 2010.

김 영 식 (Youngsik Kim)

1993년 연세대학교 컴퓨터과학과 학사
1995년 연세대학교 컴퓨터과학과 석사
1999년 연세대학교 컴퓨터과학과 박사
1995년-1997년 연세대학교 시간강사
1999년-2005년 삼성전자 System LSI 책임연구원
2005년-현재 한국산업기술대학교 부교수

관심분야 : 게임기구조, 컴퓨터구조, 3차원 그래픽가속기, 임베디드 시스템 등