가상현실에서 이동 객체를 위한 인덱스 설계

문형석*, 엄기현
동국대학교 대학원 컴퓨터공학과
{dabechi*, khum}@dgu.ac.kr

An Index Structure for Moving Object in Virtual Reality

Hyoungsuk Mun*, Kyhyun Um
Dept of Computer Engineering, Graduate School of Dongguk University

요 약

가상현실 시스템은 프레임마다 객체를 식별하고 이벤트에 대한 빠른 응답시간을 요구한다. 대부분 개상현실 시스템들을 빠른 응답시간과 객체 식별을 빠르게 하기 위하여 인덱스를 사용한다. 이러한 인덱스들은 그리드 형태로 공간을 분할하여 트리형태의 구조를 가지기 때문에 영역질의에는 취약한 구조이며 이동 객체의 이동횟수에 따라 부가적인 연산비용이 생긴다. 따라서 3차원 영역질의에 우수한 성능을 보이는 R-트리를 가상현실 시스템에 적용하였을 때에 발생되는 문제점을 정의하였다. 또한 발생한 문제점을 해결하기 위해 가시성을 고려한 영역 질의, 초기 삽입 정책을 제안하였다.

1. 서론

최근 가상현실에 관한 연구가 활발히 진행되고 있다. 가상현실은 현실적인 환경 안에서 학습과 경험을 가상으로 체험할 수 있는 공간을 제공하는 인간과 컴퓨터 간에 인터페이스(Interface)이다[1]. 이러한 가상현실은 상호작용(Interactive) 성질을 가진다. 상호작용 성질은 사용자 또는 가상현실에 존재하는 객체가 어떤 이벤트나 명령을 지시하였을 때 그에 따른 적절한 응답을 해야 한다. 따라서 가상현실은 빠른 응답시간을 요구한다[2]. 가상현실 시스템은 매 프레임(Frame) 사용자 화면에 보이는 객체를 식별하여 올바른 위치에 정확하게 표현해야한다. 따라서 객체의 식별을 빠르게 하여 응답시간을 최소화 시키기 위해서 인덱스를 사용한다.

가상현실을 위한 인덱스들은 가상현실의 구현 환경에 따라서 다른 형태를 가지지만 그리드 형태로 공간을 분할하여 트리형태의 구조를 가진다. 가상현실의 공간이 실내 환경인 경우 BSP(Binary Space Partitioning) 트리와 PVS(Potential Visibility Set)를 사용한다[3]. 가상현실의 공간을 재귀적으로 두개씩 공간을 나누어 생성한 BSP 트리의 리프노드에서 보일 수 있는 영역의 목록을 저장한 것이 PVS이다[4]. 실외 환경에서는 대부분 쿼드(Quad) 트리와 옥(Oct) 트리를 사용한다. 옥 트리는 가상현실의 공간을 8방향으로 나누어 각각의 공간에 최소의 객체가 놓여질 때까지 계속해서 공간을 분할을 하여 나누어진 공간을 하나의 노드로 생성한다. 실외환경은 공간의 특성상 시야의 폭이 넓고 깊어 많은 사용자가 가상현실에 참여하는 경우 검색대상이 많아져 응답시간이 증가된다.

3차원 가상현실 시스템에서는 플레이어 캐릭터를 중심으로 일정 영역을 설정하여 이를 가시영역으로 두어, 가시영역에 속하는 기하객체를 검색한다. 이때 특정 캐릭터의 가시영역에 속하는 기하객체 목록을 영역질의(Range Query)를 통해서 얻는다. 질의영역과 가시영역의 범위가 같고 이동 객체가 끈임 없이 이동한다면 프레임 당 1회 질의하며, 질의영역이 가시영역보다 크거나 작으면 영역의 차이와 이동 객체의 이동속도에 따라 질의 횟수가 증감한다.

기존의 그리드 형태로 가상현실의 공간을 분할하여 설계된 인덱스들은 위치질의(Point Query)에서는 우수하나 영역질의에는 취약한 구조이고, 이동 객체의 이동횟수에 따라 노드의 합병 및 분할이 생겨 부가적인 연산비용이 생긴다[5][6]. 때문에 영역질의에

본 연구는 한국과학재단 특정기초연구(과제 번호: R01-2002-000-00298-0) 내용의 일부임.

적합한 구조를 가지고 이동 객체의 이동횟수에 따라 생기는 부가적인 연산 비용을 줄일 수 있는 인덱스가 필요하다. 본 논문에서는 3차원 영역질의에 우수한 성능을 보이는 R-트리를 R-트리가 가진 최소 영역 증가 정책과 중첩 최소화 정책, 노드 분할 정책을 변경하여 3차원 가상현실 시스템에 적용 가능한 구조로 설계하기 위한 방법을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 설명하고, 3장에서는 R-트리 적용환경과 적용할 때 생기는 문제점을 정의한다. 4장에서는 이동 객체를 위한 인덱스 설계 방법에 대해 설명하고 5장에서 성능 분석을 한다. 마지막으로 6장에서는 결론을 맺는다.

2. 관련 연구

이동 객체를 위한 인덱스로 TB 트리[7], TPR 트리[8], 쿼드 트리의 문제점을 보완한 쿼드 트리 해싱 방법 등이 있다[9]. TB 트리는 이동 객체의 궤적을 질의하기 위한 인덱스로 이동 객체의 궤적을 보존하는데 중점을 두고 설계된 것이다. TPR 트리는 현재 위치뿐만 아니라 과거와 미래의 위치를 다루기 위해 R-트리에 시간 요소를 추가하여 최소 경계 상자(Minimum Bounding Box)를 설정한다. 3차원 가상현실 시스템에서 카메라 가시영역 안에 속하는 기하객체 목록을 얻기 위해 영역질의를 주로 수행한다. 하지만 이 인덱스들은 영역질의 보다 위치질의와 궤적질의와 같이 특수한 형태의 질의를 위해 설계한 것으로 가상현실의 이동 객체를 위한 인덱스 구조로 하기에는 많은 제약사항이 따른다. 따라서 3차원 영역질의에 우수한 성능을 보이는 R-트리 기반의 인덱스들을 설명한다[10].

2.1. R-트리 기반 인덱스

R^+-트리와 R^*-트리는 최소 중첩 정책을 변형하는데 중점을 두어 설계되었다. R^+-트리는 겹침으로 인한 탐색 성능이 저하되는 R-트리의 단점을 중첩을 피함으로써 해결하였다. 중첩이 발생할 경우 중첩된 최소 경계 상자를 분할하여 새로운 노드를 생성하였다. 따라서 R^+-트리는 겹침으로 인한 탐색 성능 저하를 개선하였지만 중첩으로 인하여 분할한 노드를 저장하기 위해 공간 사용이 증가된다는 단점이 있다. R^*-트리는 중첩 최소화 정책에서 수평적 분할 기법을 사용하여 겹치는 영역을 최소화하였다. 노드에 오버플로우(Overflow)가 발생할 경우 이를 분할하지 않고 일부를 형제 노드에 재삽입하는 방법을 취하였기 때문에 노드의 클러스터링 효과를 얻을 수 있다. 하지만 수평적 분할 축을 선택하기 어려운 상황에서는 중첩된 영역과 최소 경계 상자의 면적을 최소화 하기위해 상대적으로 처리하는데 소요시간이 증가한다. Hilbert R-트리는 노드에 객체를 저장하는 방법의 변형하는데 중점을 두었다. 최소 경계 상자에 속하는 객체를 Hilbert 곡선을 통해서 구해진 Hilbert 값 순서로 정렬하는 것이다. 따라서 탐색 속도가 빠르지만 삽입, 삭제할 때에 Hilbert 값을 얻어야하는 추가적인 비용이 발생한다. 따라서 R-트리 기반의 인덱스 중에 R-트리가 3차원 가상현실의 이동 객체를 위한 인덱스 구조로 상대적으로 적합하다.

2.2. 장면 그래프(Scene Graph)

장면 그래프는 그래픽적인 데이터베이스로 3차원 가상현실 시스템에서 사용자의 화면에 기하객체들과 특수 효과 표현하기 위한 데이터들을 트리 구조 저장한 것이다[11]. 장면 그래프는 카메라 가시영역에 놓여진 객체(Rendering Object)들과 객체들의 속성, 충돌처리를 위한 경계 상자(Bounding volumes)등을 포함하는 통합적인 데이터 구조이다. 일반화된 장면 그래프에서 노드는 계층적인 방법으로 렌더링 객체를 구성한다[12]. 렌더링 객체는 해당 객체의 모양(Shapes)과 속성(Attributes)을 가진다. 속성은 세부적으로 단일속성(Mono Attributes)과 복합속성(Poly Attributes), 기술속성(Techniques Attributes), 핸들러(Handlers)를 가진다. 단일속성은 렌더링 객체의 색이나 질감(Material)과 같은 속성으로 스택에 저장한다. 복합속성은 조명과 같이 하나의 유형에 여러 개의 속성을 가질 수 있는 속성이다. 기술속성은 단일·복합속성을 표현하기 위해 사용하는 렌더링 알고리즘을 서술한 속성이다. 핸들러는 복잡한 기하객체를 간단한 기하객체로 만들거나 좌표변환과 같이 부가적인 성능향상을 얻을 수 있는 속성들을 가진다.

사용자의 화면에 표현할 한 장면을 만들기 위해서는 장면 그래프를 한번 탐색으로서 필요한 데이터를 모두 얻을 수 있다. 하지만 이동 객체가 이동함에 따라 장면 그래프를 갱신해야한다. 장면 그래프를 갱신할 때 카메라 가시영역 안에 존재하는 기하객체를 영역질의를 통해서 검색하는데 이때에 인덱스를 통해서 얻는다면 시스템 전체의 성능 향상 시킬 수 있다.

3. R-트리 적용환경 및 문제점

3.1. R-트리 적용환경

클라이언트-서버 환경에서 클라이언트는 사용자가 조작하는 이동 객체의 위치 정보를 주기적으로 서버에 전송한다. 또한 클라이언트는 다른 사용자가 조작하는 이동 객체의 현재 위치를 얻기 위해 서버에 영역질의를 하여 그 결과를 얻는다. 서버는 가상환

경의 모든 이동 객체들의 위치 정보를 저장하고 있어 클라이언트가 요청한 질의에 응답한다.

3.2. 문제 정의

R-트리를 3차원 가상현실 시스템에서 이동 객체를 위한 인덱스 구조로 적용하였을 때 발생되는 문제는 다음과 같다.

첫째, 공간상의 거리가 가장 근접해 있어 동일 노드에 속하였지만 서로간의 가시영역에 속하지 않아 보이지 않는 객체를 탐색 결과로 생성할 수 있다. 이는 질의 영역내의 결과 레코드를 결정할 때 가시성을 고려하지 않았기 때문이다.

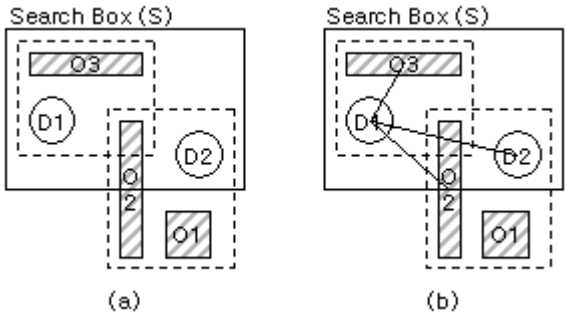
둘째, 장면 그래프는 고정 객체와 이동 객체를 포함하여 생성한다. 하지만 R-트리 구조로 고정 객체와 이동 객체를 혼합하였을 경우에 이동 객체가 끈임 없이 움직이기 때문에 삽입 시점에서 아무런 문제가 되지 않지만 모든 객체를 삽입하고 난후 중첩 최소화 정책이 위배되는 경우가 빈번히 발생한다.

4. 이동 객체를 위한 인덱스 설계

R-트리를 가상현실 시스템에 적용하였을 때 발생되는 문제점을 해결하기 위해 가시성을 고려한 영역 질의, 초기 삽입 정책을 제안한다.

4.1. 가시성을 고려한 영역 질의

질의 영역(S)과 중첩되어 검색된 결과 레코드들 중에서 이동 객체의 가시영역에 해당되지 않아 불필요한 레코드들이 결과 레코드에 포함될 수 있다. 따라서 불필요한 레코드를 제거하는 가시성 판단 작업이 필요하다. 사용자는 이동 객체의 이동을 제어하고 이동에 따른 주변 사물을 보기 때문에 이동 객체를 중심으로 질의 영역 내에서 가시성을 판단한다.



[그림 1] 질의 영역내의 가시성 판단
(a) 질의 영역, (b) 가시성을 고려한 비교

[그림 1]의 (a)처럼 질의 영역이 주지면 질의 영역내에 포함되는 객체들의 중심점을 구하여 (b)와 같이 질의한 객체의 중심점과 연결하여 가시성 판단을 한다. 다음은 가시성을 고려한 영역 질의 정책이다.

S1 질의 영역(S)과 중첩되는 레코드들을 얻는다.

S2 질의 객체의 중심점과 얻어진 레코드들의 중심점을 구한다.

S3 질의 객체의 중심점과 얻어진 레코드들의 중심점간의 직선을 연결한다.

S4 연결된 직선들이 다른 객체의 영역을 통과하는지 검사한다. 만약 다른 객체의 영역을 통과한다면 연결선을 제거한다.

S5 질의 객체와 연결된 객체들을 결과 레코드로 결정한다.

4.2. 초기 삽입 정책

트리에 어떠한 객체도 삽입되어있지 않는 상태에서 객체들을 삽입하기 위한 정책이다. 초기에 객체를 삽입할 때 고정 객체와 이동 객체를 구분하여 삽입한다. 고정 객체와 이동 객체의 구분은 가상현실 공간에 객체를 놓을 때에 이미 구분 지을 수 있는 속성을 부여받는다. 고정 객체의 경우 가상현실에서 장애물로 인식되어 이동이나 가시거리를 제한한다. 또한 고정 객체는 한번 삽입되면 수정, 삭제 연산이 거의 일어나지 않는다. 따라서 고정 객체를 기준으로 이동 객체를 삽입할 경우에 문제 정의에서 나타난 두 번째 문제를 해결할 수 있다. 다음은 초기에 고정 객체와 이동 객체 삽입하는 정책이다.

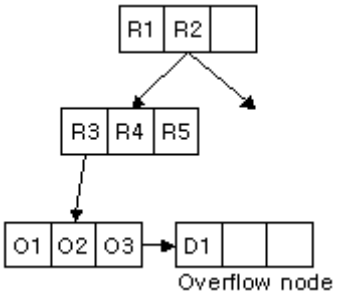
[단계1] 고정 객체를 트리에 삽입한다. 고정 객체의 삽입하기 위한 정책과 노드 선택, 노드 분할, 트리 조정은 R-트리와 동일한 정책을 사용한다.

[단계2] 이동 객체(D)를 트리에 삽입한다.

ID1 새로운 레코드(D)를 삽입하기 위해 단말 노드(L)를 선택한다.

ID2 선택된 노드(L)가 삽입할 공간이 충분하면 레코드(D)를 삽입한다. 그렇지 않으면 단말 노드(L)를 R-트리처럼 분할하지 않고 넘침 노드(Overflow node)를 생성하여 레코드(D)를 삽입한다.

ID3 새로 삽입된 레코드(D)를 포함하는 최소 경계 상자를 구한다.



[그림 2] 넘침 노드

초기 삽입 정책 [단계2]의 ID2에서 단말 노드(L)를 분할하지 않고 넘침 노드를 생성한 이유는 이동 객체의 이동성 때문이다. 즉, 이동 객체가 끈임 없이 움직이기 때문에 한시적인 위치를 위한 노드 분할

및 재구성 비용보다 넘침 노드를 생성하는 것이 더 효율적이다.

5. 성능 분석

4장에서 제안한 인덱스의 성능을 분석하기 위해서 주 연산을 수행 하였을 때 소요된 비용을 분석한다.

먼저 질의에 소요되는 비용은 접근한 노드 수와 엔트리를 비교하는 비용, 비교하는 엔트리의 개수로 정해진다.

질의 비용(C_s) = #ANode + C_e * #CEntry

#ANode = 접근한 노드 수

#CEntry = 노드의 활용도(u * M)

u = 노드의 활용도

M = 노드가 포함하는 엔트리의 최대 개수

C_e = 한 개의 엔트리를 비교하는 비용

질의 비용을 구하기 위해 접근한 노드 수와 비교한 엔트리 수를 R-트리와 제안한 인덱스 둘 다 동일하지만 한 개의 엔트리를 비교하는 비용인 C_e는 제안한 인덱스에서 약간 높게 측정된다. 즉, 가시성 판단을 하기위해 소요되는 비용이 있기 때문이다.

R-트리에서 삽입은 대부분 삽입할 단말 노드를 찾고 트리를 재조정하는데 소요된다. 따라서 삽입에 소요되는 비용(C_i)은 트리의 깊이(I), 비교 엔트리 수와 분할에 드는 비용(Sl)으로 정해진다.

$$C_i = \sum_{n=1}^{I-2}(u * M) + u * Sl$$

제안한 인덱스에서 오버플로우가 발생한 경우 노드를 분할하지 않으므로 노드의 활용도(u)가 높을수록 R-트리보다 삽입 비용이 적게 들지만, 넘침 노드로 인하여 노드의 크기 증가하게 된다.

6. 결론 및 향후 연구

기존 가상현실 시스템을 위한 인덱스들은 영역 질의에는 취약한 구조이며 이동 객체의 이동횟수에 따라 부가적인 연산비용이 생겼다. 따라서 본 연구에서는 영역질의에 우수한 성능을 보이는 R-트리를 가상현실 시스템을 위한 인덱스 구조로 제시하였다. 또한 R-트리를 가상현실 시스템에 적용하였을 때에 발생되는 영역 질의에 대한 부정확한 결과 값과 고정 객체와 이동 객체를 혼합하였을 경우 발생되는 문제를 해결하기 위해 가시성을 고려한 영역 질의, 초기 삽입 정책을 제안하였다. 본 논문에서 제안한 인덱스 구조를 바탕으로 향후 시뮬레이션 도구를 구현하고 실험하는 과제가 남아있다.

참고문헌

[1] John N. Latta, David J. Oberg, "A Conceptual Virtual Reality Model", IEEE Computer Graphics & Applications, January 1994.

[2] 문형석, 엄기현, 조형제, "3D 렌더링 엔진의 프레임 기반 성능 측정 및 분석", 한국게임학회, 2004년 동계 학술발표대회 논문집, pp403-408 , 2004.1.

[3] Gordon, D., Chen, S., "Front-to-back display of BSP trees", Computer Graphics and Applications, IEEE, Volume: 11, Issue: 5, Page(s): 79-85, Sept. 1991.

[4] David Luebke, Chris Georges, "Portals and mirrors: simple, fast evaluation of potentially visible sets", Proceedings of the 1995 symposium on Interactive 3D graphics, Pages: 105 - ff., April 1995.

[5] H.Samet. "The Design and Analysis of Spatial Data Structures", Addison-Wesley, Reading, MA, 1990.

[6] P. Agarwal and J. Erickson, "Geometric Range Searching and Its Relatives", Advances in Discrete and Computational Geometry, volume 23 of Contemporary Mathematics, pp.1-56. American Mathematical Society Press, Providence, RI, 1999.

[7] D. Pfoser, C. S. Jensen, Y. Theodoridis, "Novel Approaches in Query Processing for Moving Objects". Proc. Of the 26th VLDB Conf., Cairo, Egypt, September 2000.

[8] S. Saltenis, C. S. Jenen, S. T. Leutenegger, and M. A. Lopez, "Indexing the Positions of Continuously Moving Objects", Proc. ACM SIGMOD on Management of data, 2000.

[9] Zhexuan Song, Nick Roussopoulos, "Hashing Moving Objects", Mobile Data Management, 161-172, 2001.

[10] Y.Theodoridis, M.Vazirgiannis, T.Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications", Proc. Of the 3rd IEEE Conf. On Multimedia Computing and Systems, ICMCS'96, Hiroshima, Japan, June 1996.

[11] Lars M. Bishop, Dave Eberly, Mark Finch, Michael Shantz, "Designing a PC Game Engine", IEEE Computer Graphics and Applications, Jan, 1998

[12] J. Döllner, K. Hinrichs, "A Generalized Scene Graph API". Vision, Modeling, Visualization 2000 (VMV 2000), Saarbrücken. Akademische Verlagsgesellschaft Aka, 247-254, 2000.