공간 뷰 재작성을 위한 점진적 변경 방법
(Incremental Update Methods for Adapting of Spatial Views)

반 재 훈 † 문 상 호 †† 홍 봉 희 †††
(Chae-Hoon Ban) (Sang-Ho Moon) (Bong-Hee Hong)

요 약 실체화된 공간 뷰가 사용자의 관점에 따라 재정의되면 기존의 실체화된 뷰 객체들을 변경하는 것이 필요하며, 이것을 공간 뷰의 재작성이라 한다. 공간 뷰의 재작성을 위한 방법으로는 재정의된 뷰-정의 질의를 수행하는 재계산 방법보다는 영향 받는 뷰 객체들만을 변경하는 점진적 변경이 더 효율적이다. 이것은 공간 뷰가 기존 뷰와는 달리 공간연산자를 포함하는 공간 질의에 의해 정의되므로 실체화 시간이 많이 걸리기 때문이다.

본 논문에서는 먼저 공간 뷰의 재정의 유형들을 분류하고, 각 유형들에 따라 재작성을 위한 점진적 변경 방법들을 제시한다. 이 방법들에서는 기존의 실체화된 뷰 객체들과 뷰 객체와 대응되는 소스 객체들간의 뷰 유도관련성을 추가 정보로 이용한다. 그리고 점진적 재작성의 유형은 기존의 실체화된 뷰 객체를 변경하는 것과 새로운 뷰 객체를 삽입하는 것으로 분류한다. 본 논문에서 제시한 점진적 재작성을 실현하기 위하여 고딕상에 공간 뷰 재작성기를 설계 및 구현한다. 또한 실제 데이타를 대상으로 재계산 방법과의 성능 평가를 통하여 점진적 재작성의 우수성을 보인다.

Abstract The adaptation of spatial view is to update materialized view objects when spatial view is redefined. There are two kinds of adaptation: incremental updates and recomputation. The incremental update changes related view objects and it is more efficient than the recomputation which evaluates redefined view-defining query because spatial view is defined by spatial query including high-cost spatial operator.

This paper proposes the several incremental update methods according to the types of changing the definition of a spatial view. There are two kinds of incremental view adaptation: the method of using only the existing view objects and the view derivation relationship between view objects and their sources. This incremental update is achieved by updating the current materialized view objects or by inserting new materialized view objects. Spatial view adapter is implemented and tested on top of the object-oriented geographic information system. This paper evaluates performance between the recomputation and the incremental update method through real data.

1. 서 론

공간 데이타베이스에서 가상 클래스로 정의되는 공간 뷰(spatial view)는 다양한 사용자의 관점에 따른 지리 객체의 서로 다른 공간 표현을 지원한다. 이러한 공간 뷰는 기존의 관계 뷰나 객체지향 뷰와는 달리 대용량의 공간 객체들에 대하여 CPU 계산 시간이 많이 소요되는 공간 연산에 의해 정의된다. 따라서 공간 뷰에 대한 질의를 빠르게 수행하기 위하여 실체화하는 것이 필요하다[1, 2].

실체화된 공간 뷰는 적용되는 응용 또는 시간에 따라 사용자의 관점이 달라질 수 있다. 이와 같이 사용자 관점의 변경에 따라 공간 뷰의 정의를 변경하는 것을 뷰의 **재정의(redefinition)**라 한다. 공간 뷰의 재정의는 기존의 데이타베이스에 저장된 뷰-정의 질의(view-defining query)를 변경하는 것이다. 공간 뷰가 실체화된 경우에는 이러한 공간 뷰의 재정의에 따라 기존의

† 학생회원 : 부산대학교 컴퓨터공학과
jhpan@hyowon.cc.pusan.ac.kr
†† 정 회 원 : 위덕대학교 컴퓨터공학과 교수
shmoon@mail.uiduk.ac.kr
††† 종신회원 : 부산대학교 컴퓨터공학과 교수
bhhong@hyowon.cc.pusan.ac.kr
논문접수 : 1999년 4월 21일
심사완료 : 1999년 12월 13일

데이타베이스에 저장된 실체화된 뷰 객체들을 변경하는 것이 필요하다. 이것을 뷰의 **재작성(adaptation)**이라 하며, 공간 뷰의 재정의에 따라 기존의 실체화된 뷰 객체의 일관성을 유지시켜 준다.

뷰의 재작성에는 처음부터 다시 연산을 수행하는 **재계산(recomputation)** 방법과 기존의 실체화된 뷰 객체들을 이용하여 변경하는 **점진적 변경(incremental update)** 방법이 있다[3, 4]. 재계산 방법은 기존의 실체화된 뷰 객체들을 모두 삭제하고 재정의된 뷰-정의 질의를 처음부터 다시 수행하여 새로 객체들을 생성하는 방법이다. 이 방법은 뷰가 재정의될 때마다 대용량의 공간 객체들에 대하여 복잡한 공간 연산을 수행하기 때문에 시간이 많이 걸린다. 반면에 점진적 변경 방법은 기존의 실체화된 객체들을 최대한 이용하여 최소한의 부분 변경만으로 재작성을 수행하는 방법이다. 따라서 이 방법은 재계산 방법보다 변경 속도가 빨라서 효율적이지만 변경 방법이 복잡해지는 단점이 있다.

일반적으로 데이타베이스에서 뷰는 사용자의 데이타 관리를 위한 질의의 다른 형태이다. 즉, 질의를 뷰로 정의하면 복잡한 질의를 간단한 질의(shorthand query)로 표현이 가능하고, 또한 뷰의 결과를 나중에 다시 이용하면 효율적이다. 특히 공간 뷰에서는 사용자의 관점이 변경된 경우에 뷰의 변경된 결과를 빠르게 보여주는 것이 필요하다. 즉, 사용자 관점의 변경으로 인해 재정의된 경우에, 실체화된 공간 뷰를 빠르게 재작성 해야 한다. 예를 들어 그림 1(a)와 같이 사용자의 관점에 따라 공간 뷰 유해업소가 정의되고 실체화되었다고 가정하자. 이 공간 뷰는 학교로부터 100미터 이내에 있는 모든 카페들을 표현한 것이다. 여기서 사용자 관점의 변경으로 인해 이 공간 뷰가 그림 1(b)와 같이 학교로부터 50미터 이내에 있는 모든 카페들로 재정의된다. 이 경우에 실체화된 뷰 객체들을 모두 삭제하고 처음부터 다시 뷰-정의 질의를 수행하여 실체화하는 재계산 방법보다 기존에 실체화된 공간 뷰 객체 중에서 학교로부터 50미터 이상 떨어진 것들을 삭제하는 점진적 변경이 더 효율적이다.

본 논문에서는 공간 뷰의 재정의에 따른 뷰의 재작성 방법을 다룬다. 앞에서 언급한 것과 같이 재작성은 뷰의 재정의에 따라 일관성 유지를 위하여 뷰 객체들을 변경하는 것으로, 여기서 뷰 객체들을 변경하는 방법은 크게 재계산과 점진적 변경이 있다. 본 논문에서는 공간 데이타 베이스상에서 공간 뷰가 재정의된 경우에 점진적 변경을 이용한 재작성 방법을 제시한다. 그리고 효율적인 공간 뷰의 재작성을 지원하기 위하여 뷰의 재정의 유형을 분류한다.

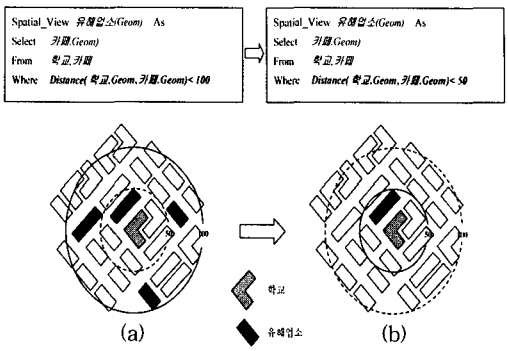


그림 1 공간 뷰 재정의

본 논문에서 재작성을 위하여 제시한 점진적 변경은 기존의 실체화된 뷰 객체를 이용한 방법과 추가 정보를 이용한 방법이다. 추가 정보를 이용한 방법에서는 공간 뷰 객체들과 대응되는 소스 객체들간의 뷰 유도관련성(VDR: View Derivation Relationship)을 이용한다. 여기서 공간 뷰 재작성에 따른 점진적 변경 유형은 기존의 실체화된 뷰 객체를 변경하는 것과 새로운 뷰 객체를 삽입하는 두 가지 경우가 있다.

본 논문에서 제시된 공간 뷰 재작성을 위한 점진적 변경 방법을 상용 객체지향 GIS 시스템인 고딕(GOTHIC)상에서 설계 및 구현한다. 그리고 이 방법의 성능 평가를 위하여 재계산과 기존의 관계 뷰를 위한 재작성 방법과의 실험 평가를 시뮬레이션이 아닌 실제 구현된 시스템에서 수행한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 관련 연구를 기술하고 3장에서는 공간 뷰 재작성의 정의와 점진적 변경 유형을 분석한다. 그리고 4장에서는 공간 뷰의 재정의 유형에 따른 점진적 변경 알고리즘을 제시한다. 5장에서는 고딕상에서 구현한 공간 뷰 재작성기를 소개하며, 실험을 통하여 점진적 변경 방법을 이용한 재작성의 성능 평가를 수행한다. 마지막으로 6장에서는 결론 및 향후 연구를 기술한다.

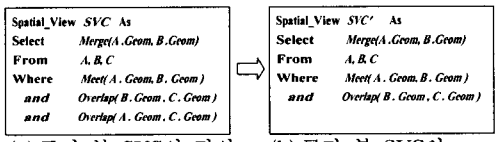
2. 관련연구

기존 연구들에서는 관계 뷰를 위한 재작성 방법은 일부 제시가 되었지만, 객체지향 뷰나 공간 뷰의 재정의에 따른 재작성 방법에 대한 연구는 거의 없다. 관계 데이타베이스에서 뷰 재정의에 따라 실체화된 뷰 테이블을 변경하는 재작성은 [3, 4]에서 제시하였다. 기존의 재계

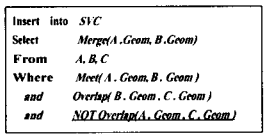
산 방법은 뷰의 재정의에 따른 뷰 재작성을 위하여 실체화된 뷰 테이블을 모두 삭제하고 재정의된 뷰-정의 질의를 수행하여 새로운 뷰 테이블을 생성한다. 그러나 [3, 4]에서는 기존의 실체화된 뷰 테이블을 최대한 이용하여 재작성을 수행하는 점진적 변경 방법을 제시하였다.

[3, 4]에서는 뷰-정의 질의의 재정의 유형을 Select-From-Where절로 분류하고 각 경우에 대한 재작성 방법을 제시하였다. Select절에 애트리뷰트 추가는 추가 정보를 이용하여 소스 튜플(source tuple)에 접근한 후에, 추가할 애트리뷰트 값을 검색하여 뷰 튜플에 삽입시킨다. 여기서는 뷰 튜플을 유도하는 소스 튜플의 기본키를 뷰의 외래 키로 저장한 것을 추가 정보로 이용한다. 그리고 애트리뷰트 삭제는 뷰 테이블의 애트리뷰트를 삭제하여 재작성을 수행한다. From절에 조인을 위한 테이블 추가는 기존 뷰 테이블에 대하여 조인을 수행하며, 테이블 삭제는 재계산을 수행한다. Where절에 프레디킷 추가는 기존의 뷰 테이블에 대하여 추가된 프레디킷에 관한 질의를 수행한 후에, 프레디킷을 만족하지 않는 튜플만을 삭제하여 재작성을 수행한다. 그리고 프레디킷 삭제는 대수적 관계를 이용하여 추가되는 부분만을 계산하여 재작성을 수행한다.

[3, 4]에서 제시한 재작성 방법은 관계 뷰를 위한 것이므로 공간 뷰에 적용시키면 다음과 같은 문제점이 발생한다. 첫째, 이 연구에서 이용한 추가 정보는 뷰 튜플을 유도하는 소스 튜플의 기본 키를 뷰의 외래 키로 저장한 것이므로, 이것은 객체지향 공간 뷰의 재작성을 위한 추가정보로 바로 이용할 수 없다. 둘째, 기존의 관계 뷰와는 달리 공간 뷰는 정의 내에 뷰의 기하데이타를 유도하기 위한 기하-사상 함수(GMF: Geometry-Mapping Function)를 사용한다. 이것은 사용자의 관점에 따른 지리 객체의 다양한 표현을 지원하기 위한 것으로 공간 뷰의 중요한 특징이지만, 이 연구에서는 이러한 기하-사상 함수에 따른 재작성 방법은 제시되지 않았다. 따라서 공간 뷰를 위한 재작성에서는 기하-사상 함수의 수정에 따른 점진적 변경 방법의 제시가 필요하다. 셋째, 관계 뷰-정의 질의에 사용되는 프레디킷은 비교적 저비용의 비교 연산자이지만, 공간 뷰-정의 질의에 사용되는 프레디킷은 고비용의 공간 연산자이다. 그러므로 프레디킷 삭제의 경우에 이 연구에서 제시한 재작성 방법을 공간 뷰에 적용하면 추가되는 고비용의 공간 연산자로 인해서 재작성 시간이 길어지는 문제점이 발생할 수 있다. 예를 들어, 그림 2(a)의 공간 뷰 ***SVC***를 (b)와 같이 ***SVC'***로 재정의하는 경우는 프레디킷의 삭제이다. 이 경우에 [3, 4]의 방법은 그림 2(c)와 같이 기존의 실체화된 뷰 객체는 그대로 두고 새로 추가할 객체만을 계산하여 생성한다. 이 경우에는 재정의된 그림 2(b)의 뷰 질의를 수행하는 재계산 방법보다 공간 프레디킷 을 추가한 그림 2(c)의 변경 시간이 더 길어질 수 있다.



(a) 공간 뷰 SVC의 정의 (b) 공간 뷰 SVC의 재정의(SVC')



(c) 공간 뷰 SVC'의 재작성

그림 2 공간 뷰의 프레디킷 삭제에 따른 점진적 변경 예

기존의 연구에서 관계 뷰의 재정의에 대한 뷰 재작성 방법이 제시되었으나, 앞에서 언급한 것과 같이 기존의 방법을 공간 뷰에 그대로 적용시키기가 어렵다. 이것은 뷰 정의에 있어서 관계 뷰와 공간 뷰가 가지는 특성이 다르기 때문이다. 따라서 공간 뷰의 재작성에서 공간 뷰에 대한 재정의 유형들을 분류하고, 이에 따라 적용할 수 있는 새로운 재작성 방법의 제시가 필요하다.

관련 연구 [1, 2]는 본 논문의 선행 연구로서 공간 뷰 개념 및 모델, 공간 뷰의 시멘틱, 공간 뷰 실체화, 그리고 실체화된 공간 뷰의 점진적 변경 방법 등이 제시되었다. 본 논문은 이 연구들의 내용을 기반으로 하여 공간 뷰 재정의에 따른 점진적 재작성 방법을 중점적으로 다루고 있다. 여기서 본 연구와 선행 연구들 간의 비교 대상은 점진적 변경 방법이다. [1, 2]에서는 빠른 공간 질의 수행을 위하여 실체화된 공간 뷰와 대응하는 소스 객체의 변경이 발생한 경우에 뷰의 일관성을 유지하기 위한 점진적 변경 방법을 제시하였다. 이 점진적 변경에서는 소스 객체의 변경에 따라 직접적으로 영향받는 뷰 객체를 빠르게 검색하는 것이 중요한 문제이다. 이를 위하여 공간 뷰 객체와 소스 객체들 간의 뷰 유도 관련성을 제시하였고, 이 유도관련성의 카디널리티 제약조건을 이용한 뷰의 변경 시멘틱을 정의하였다. 반면에 본 논문에서의 점진적 변경은 실체화된 공간 뷰의 뷰-정의 질의가 변경되는 경우에 일관성 유지를 위한 것이

다. 이 방법에서는 공간 뷰의 재정의 유형에 따라 점진적 변경이 달라진다. 즉, 뷰-정의 질의의 Select-From-Where절 변경에 따라 점진적 변경 방법이 결정된다. 결론적으로 [1, 2]과 본 논문에서 각각 제시된 점진적 변경은 추가 정보를 이용하여 기존의 실체화된 뷰 객체들 중에서 영향 받는 객체들을 변경하는 점은 유사하지만, 점진적 변경의 주된 초점과 변경 시멘틱 등이 완전히 다르다. 따라서 본 논문에서는 [1, 2]에서 제시된 점진적 변경 알고리즘은 적용할 수가 없고, 공간 뷰 재정의 유형 들에 따른 새로운 점진적 변경 알고리즘을 제시해야 한다.

3. 공간 뷰 재작성

이 장에서는 본 논문에서 사용되는 용어를 정의하고 재작성과 재정의 유형에 대하여 알아본다. 그리고 점진적 변경 방법의 유형과 점진적 변경시 사용되는 추가 정보인 VDR에 대하여 기술한다.

3.1 용어 정의 및 뷰 모델

공간 뷰 재작성 방법을 제시하기에 앞서 관련된 용어들을 다음과 같이 정의한다. 이 용어들은 대부분 관련 연구들마다 조금씩 차이가 있으나 대부분이 공통적으로 사용되고 있다[1, 2, 3, 5, 6, 7].

- 뷰-정의 질의(view-defining query) : 뷰에 대한 정의문으로 뷰 객체를 유도하는데 사용된다. 뷰-정의 질의를 뷰 사상(view mapping) 함수라고도 한다.
- 뷰 클래스(view class) : 뷰-정의 질의에 의해 유도되는 가상 클래스이다. 뷰 클래스는 한 개 이상의 기본 클래스 또는 다른 뷰 클래스로부터 유도된다.
- 뷰 객체(view object) : 뷰-정의 질의에 의해 유도되는 뷰 클래스의 인스턴스로 실제 데이타베이스에는 저장되지 않는다.
- 소스 클래스(source class) : 뷰가 정의될 때 사용되는 클래스로, 뷰-정의 질의의 FROM 절에 지정된다. 기본 클래스뿐만 아니라 뷰 클래스도 소스 클래스가 될 수 있으며, 하나의 뷰 클래스에 대하여 하나 이상의 소스 클래스가 존재한다.
- 소스 객체(source object) : 소스 클래스의 인스턴스로 뷰 객체를 유도하기 위해 사용된 객체이다. 기본 객체뿐만 아니라 뷰 객체가 소스 객체가 될 수 있다.
- 기하-사상 함수(GMF: Geometry-Mapping Function) : 공간 뷰의 기하데이타를 유도하는데 사용되는 함수로 뷰-정의 질의의 SELECT절에서 정의된다.

본 논문에서의 뷰 모델은 [1, 2]에서 제시한 객체지향 공간 뷰 모델을 기반으로 한다. 이 공간 뷰 모델은 객체지향 데이타베이스의 뷰 개념을 확장한 것으로, 공간 뷰를 소스 클래스로부터 공간 질의에 의해 유도되는 가상 클래스로 정의한다. 그리고 공간 뷰 클래스에는 다양한 사용자 관점에 따른 공간 표현을 지원하기 위하여 기하데이타를 가지는 'geom'이라는 애트리뷰트를 정의한다. 이 애트리뷰트의 값은 공간 뷰가 실체화될 때, 뷰-정의 질의에 정의된 기하-사상 함수로부터 유도된다. 따라서 공간 뷰는 사용자 관점에 따라 공간관련성을 가지는 하나 이상의 소스 클래스들로부터 기하-사상 함수에 의해 다양한 공간 표현을 지원한다. 객체지향 공간 뷰 모델링에 대한 세부적인 내용은 [1, 2]에 언급되어 있다.

3.2 재작성

재정의된 뷰를 위한 재작성은 크게 재계산 방법과 점진적 변경 방법이 있다. 그림 3과 같이 뷰 VC가 소스 클래스 SC로부터 유도되고 실체화되었다고 가정하자. 그 후에 VC로부터 VC'가 재정의되었다면 기존의 실체화된 뷰 객체는 VC'를 만족하지 않을 수 있으므로 일관성 유지를 위하여 재작성을 수행해야 한다. 재계산은 기존의 실체화된 VC의 모든 객체들을 삭제하고 SC로부터 재정의된 뷰 VC'를 새로 실체화하여 재작성하는 방법이다. 반면에 점진적 변경은 기존의 실체화된 VC의 객체들을 최대한 이용하여 최소한의 부분 변경(partial update)으로 재정의된 VC'를 재작성하는 방법이다.

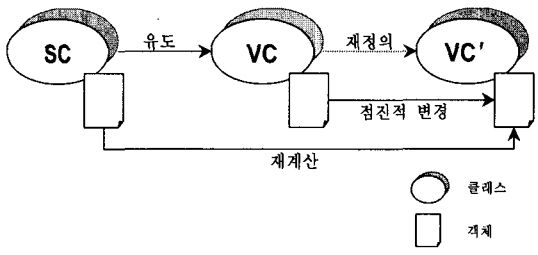


그림 3 뷰의 재작성

점진적 변경을 이용한 재작성은 기존의 실체화된 뷰 객체들 중에서 관련된 객체들만을 변경하므로 재계산에 비해 복잡하며, 효율적인 변경을 위하여 부수적인 정보가 필요하다. 부수적인 정보로는 먼저 재정의 전과 재정

의 후의 뷰-정의 질의가 필요하다. 그리고 기존의 실체화된 뷰 객체들과 점진적 변경을 위한 추가 정보가 필요하다.

공간 뷰 재정의와 재작성은 기존의 재정의와 재작성을 공간 뷰에 적용시킨 것이다. 공간 뷰 재정의가 기존 관계 뷰의 재정의와 다른 점은 첫째, 사용자의 관점에 따른 지리 객체의 서로 다른 공간 표현을 위한 기하-사상 함수에 대한 재정의가 발생하며 둘째, 공간 뷰 재정의시 Where절에 사용되어지는 프레디킷으로 고비용의 공간 연산자를 사용한다. 마지막으로 From절 내에 새로운 클래스를 추가하면 Where절 내에 새로 추가된 클래스와 관련된 공간 프레디킷 추가를 수반한다.

3.3 재정의 유형

본 논문에서는 먼저 공간 뷰의 재정의 유형들을 다음과 같이 정의한다. 이 유형들은 공간 뷰가 재정의되어지는 최소 단위이며 점진적 변경에 있어서 기본이 된다. 공간 뷰의 재정의 유형들은 다음과 같다.

1. Select절의 애트리뷰트 추가 또는 삭제
2. Select절의 기하-사상 함수 수정
3. From절의 클래스 추가 또는 삭제
4. Where절의 프레디킷 추가, 삭제 또는 수정

3.4 점진적 변경의 유형 및 방법

뷰의 재정의에 따른 점진적 변경의 유형은 기존의 실체화된 뷰 객체를 변경하는 경우와 새로운 뷰 객체를 삽입하는 경우이다. 기존의 실체화된 뷰 객체를 변경하는 경우는 새로운 객체의 삽입(생성) 없이 기존 객체를 삭제하거나 객체의 애트리뷰트를 삽입, 삭제, 수정하는 경우로, 여기서 객체의 애트리뷰트는 비기하(non-geometry) 애트리뷰트이거나 기하(geometry) 애트리뷰트이다. 반면에 새로운 객체를 삽입하는 경우는 기존 객체의 변경만으로는 재정의된 뷰-정의 질의를 만족할 수 없으므로 새로 추가될 객체만을 계산하여 삽입하는 경우이다.

점진적 변경은 뷰 객체를 이용하는 방법과 추가 정보를 이용하는 방법으로 분류된다. 뷰 객체를 이용하는 방법은 별도의 추가 정보 없이 뷰 객체만을 이용하여 재작성하는 것이고, 반면에 추가 정보를 이용하는 방법은 뷰 객체뿐 아니라 추가 정보를 이용하여 재작성한다. 본 논문에서는 점진적 변경을 위한 추가 정보로서 공간 뷰 객체들과 대응되는 소스 객체들간의 유도관련성 정보를 이용한다.

공간 뷰의 재정의 유형에 따른 점진적 변경 유형과 방법은 표 1과 같다. 뷰의 재정의 유형을 크게 Select절, From절, 그리고 Where절 변경으로 분류하며 관련된 세부적인 내용은 4장에서 기술한다. 앞에서 언급한 것처럼 변경 유형은 기존의 뷰 객체 변경과 새로운 뷰 객체 삽입으로, 그리고 변경 방법은 VDR 사용과 뷰 객체 사용으로 구분하여 재정의 유형들에 따라 적용한다.

표 1 재정의 유형에 따른 점진적 변경의 유형과 방법

공간 뷰의 재정의 유형 \ 점진적 변경 유형과 방법		변경 유형	변경 방법
SELECT절	애트리뷰트 추가	기존의 뷰 객체 변경	VDR 사용
	애트리뷰트 삭제	기존의 뷰 객체 변경	뷰 객체 사용
	GMF 함수명 수정	기존의 뷰 객체 변경	뷰 객체 사용
			VDR 사용
	GMF 매개 변수 수정	기존의 뷰 객체 변경	뷰 객체 사용
			VDR 사용
	GMF 시그니처 수정	기존의 뷰 객체 변경	뷰 객체 사용
			VDR 사용
FROM절	클래스 추가	기존의 뷰 객체 변경	뷰 객체 사용
			VDR 사용
WHERE절	프레디킷 추가	기존의 뷰 객체 변경	뷰 객체 사용
			VDR 사용
	프레디킷 삭제	새로운 객체 삽입	VDR 사용
	프레디킷 수정	기존의 뷰 객체 변경	뷰 객체 사용
			VDR 사용
		새로운 객체 삽입	VDR 사용

3.5 점진적 변경을 위한 추가정보

본 논문에서는 공간 뷰 재작성을 위한 점진적 변경을 효율적으로 수행하기 위한 추가 정보로서 [1, 2]에서 제시한 뷰 유도관련성(VDR)을 이용한다. VDR은 [1, 2]에서 소스 객체의 변경에 따른 실체화된 뷰를 점진적으로 변경하기 위하여 제시되었다. VDR은 뷰 객체와 소스 객체들의 식별자들로 구성되며, 뷰 객체와 대응되는 소스 객체들 간에 유도 관계를 나타낸다. 그리고 VDR은 뷰 객체와 대응되는 소스 객체들 간의 카디널리티 비율에 따라 일대일, 일대다, 다대일, 다대다 관련성을 가진다. 예를 들어, VDR 인스턴스가 '$<SVO_I, H_I, T_I>$'이면 뷰 객체 SVO_I이 각 소스 클래스의 객체 H_I, T_I로부터 유도된 것을 나타낸다. 이때, 뷰 객체 SVO_I는 각 소스 클래스의 객체인 H_I와 T_I 간에 일대일 관계를 가진다. 뷰 유도관련성의 세부적인 내용은 [1, 2]에 자세하게 언급되어 있다.

점진적 변경은 재계산 방법과는 달리 실체화된 공간 뷰에서 영향 받는 뷰 객체들 만을 변경하므로 추가 정

보의 이용이 필수적이다. [1, 2]에서 제시한 소스 객체의 변경에 따른 실체화된 뷰의 점진적 변경과 마찬가지로 공간 뷰 재정의에 따른 실체화된 뷰 객체의 변경에서도 뷰 객체와 소스 객체들 간의 유도관련성이 필요하다. 따라서 본 논문에서는 공간 뷰의 재정의시 VDR을 추가 정보로 이용하여 점진적 변경을 수행한다. 이 점진적 변경에서 뷰 객체를 유도하는 소스 객체에 접근이 필요한 경우에 이것을 추가 정보로 이용한다. 예를 들어, Select절의 애트리뷰트 추가 시에 적용되는 점진적 변경 방법은 추가되는 애트리뷰트 값을 소스 객체로부터 검색하여 기존의 실체화된 뷰 객체의 애트리뷰트 값으로 삽입한다. 따라서 뷰 객체에 애트리뷰트 값을 복사하기 위해서는 대응되는 소스 객체에 접근이 필요하며, 이러한 경우에 VDR을 추가 정보로 이용한다.

점진적 변경에서 뷰 객체들의 변경에 따라 VDR의 일관성의 유지가 필요한 경우가 발생한다. 점진적 변경을 위하여 새로운 뷰 객체의 삽입 또는 기존 뷰 객체의 삭제가 발생하는 경우에 관련된 VDR 인스턴스의 삽입과 삭제가 필요하다. 또한 From절에 새로운 소스 클래스를 삽입하는 경우에는 뷰 객체를 유도하는 소스 객체의 수가 늘어나므로, 뷰 객체와 관련된 VDR 인스턴스에 새로운 소스 객체들의 식별자들을 추가해야 한다. 따라서 공간 뷰 재정의에 따른 점진적 변경에서 기존의 실체화된 뷰 객체들의 변경과 함께 관련된 VDR 인스턴스들의 일관성을 유지하는 것이 필요하다.

4. 점진적 변경 알고리즘

이 장에서는 공간 뷰의 재정의 유형에 따른 재작성 방법인 점진적 변경 알고리즘을 제시한다. 공간 뷰의 점진적 변경 방법은 SQL문의 데이타 정의어(DDL)와 데이타 조작어(DML) 또는 추가 질의를 이용하여 나타낸다. 여기서 SVC는 기존에 정의된 공간 뷰를, SVC'는 재정의된 공간 뷰를 나타낸다. 그리고 SVO_k는 기존 공간 뷰의 한 객체를, SCO_i는 공간 뷰 객체를 유도하는 소스 객체, GMF는 기존의 기하-사상 함수를 GMF_{new}는 수정된 기하-사상 함수를 나타낸다.

4.1 공간 뷰-정의 질의

공간 뷰 정의를 위한 뷰-정의 질의는 [1, 2]에서 제시된 것을 이용하며, 형식은 그림 4과 같다. 공간 뷰-정의 질의가 객체지향 뷰 또는 관계 뷰-정의 질의와 다른 점은 Select절에 사용자의 관점에 따른 지리 객체의 공간 표현을 위한 기하-사상 함수를 가진다는 것이다. 이러한 기하-사상 함수를 사용함으로써 다양한 사용자의 관점에 따른 지리 객체의 서로 다른 공간 표현이 가능하다. 본 논문에서 정의된 공간 뷰의 애트리뷰트들은 Select절에서 정의된 애트리뷰트와 동일하거나 또는 다른 이름을 가진다.

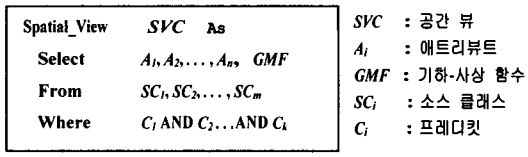


그림 4 공간 뷰-정의 질의

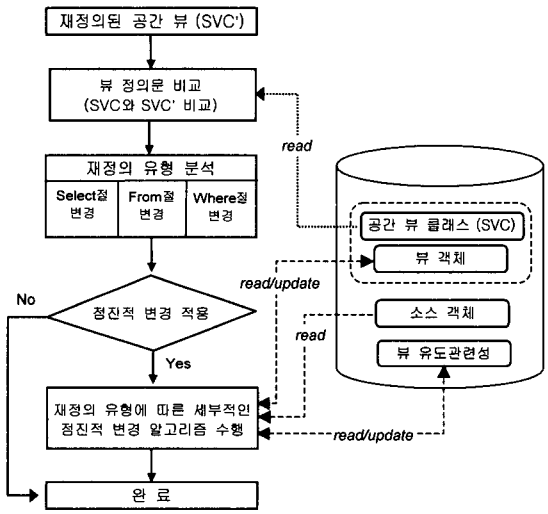


그림 5 점진적 변경을 이용한 공간 뷰 재작성의 수행 과정

4.2 기본 메커니즘

공간 뷰 재정의에 따른 재작성을 위한 점진적 변경 방법의 전체적인 수행 과정은 그림 5와 같다. 먼저 공간 뷰가 재정의된 경우에 공간 DB에 저장된 기존의 공간 뷰 클래스와 재정의된 뷰 정의문을 비교한다. 이 비교를 통하여 입력된 공간 뷰에 대한 재정의 유형을 분석하며, 여기서는 크게 Select절, From절, Where절의 변경으로 분류한다. 그리고 이 재정의 유형들에 대하여 점진적 변경 방법이 필요한지를 판단한다. 만약 필요하다면 전단계에서 분석된 재정의 유형들에 따라 적합한 점진적 변경 알고리즘을 적용하여 실체화된 뷰의 재작성을 수행한다. 재정의 유형들에 따른 재작성 방법의 특징과 세부적인 점진적 변경 방법 및 알고리즘은 4.3절부터 4.5절에 자세하게 설명한다. 이 점진적 변경 알고리즘 수행 단계에서 중요한 것은 실체화된 뷰의 재작성을 위한 점진적 변경을 위한 추가 정보로서 뷰 유도관련성을 이용

한다는 것이다. 뷰 유도관련성을 이용하여 뷰 객체와 관련된 소스 객체들을 검색하여 점진적 변경을 수행한다. 이 때, 필요하다면 변경된 뷰 객체들과 관련 있는 VDR 인스턴스들도 변경해야 한다.

4.3 Select절의 변경

4.3.1 애트리뷰트의 추가 및 삭제

공간 뷰-정의 질의에서 Select절 내의 애트리뷰트 추가 및 삭제는 관계 뷰를 위한 재작성 방법과 거의 유사하다. 새로운 애트리뷰트 추가의 경우에는 그림 6과 같이 먼저 ADD ATTRIBUTE를 사용하여 추가할 애트리뷰트 A_{n+1}를 기존 공간 뷰 클래스 SVC에 추가한다. 그리고 기존의 실체화된 뷰 객체에 애트리뷰트 A_{n+1} 의 값을 추가하기 위해서 뷰 객체와 소스 객체와의 유도관련성 정보인 VDR을 이용하여 소스 객체 SCO_i 에서 애트리뷰트 A_j의 값을 검색한 후에 뷰 객체에 SET을 사용하여 삽입한다.

```
ALTER  CLASS SVC
ADD ATTRIBUTE A_{n+1}
UPDATE  SVO_k  IN  SVC
SET     SVO_k.A_{n+1} = SCO_i.A_j
```

그림 6 애트리뷰트의 추가시의 점진적 변경

애트리뷰트 삭제의 경우에는 그림 7과 같이 기존의 공간 뷰 SVC에서 삭제할 애트리뷰트 A_j를 DROP을 사용하여 삭제한다. 이 경우에는 뷰 클래스에 애트리뷰트를 삭제하면 관련된 뷰 객체들의 애트리뷰트 값은 자동적으로 삭제가 된다.

```
ALTER CLASS  SVC
DROP ATTRIBUTE  A_j          (1 ≤ j ≤ m)
```

그림 7 애트리뷰트의 삭제시의 점진적 변경

4.3.2 기하-사상 함수의 수정

기하-사상 함수는 공간 뷰-정의 질의 Select절 내에 포함되어 공간 뷰의 기하데이타를 유도하는 함수이다. 본 논문에서는 기하-사상 함수의 수정 유형을 함수명 수정, 매개 변수 수정, 시그니처(signature) 수정의 3가지로 나눈다.

- **함수명 수정**

함수명 수정은 뷰-정의 질의에서 기하-사상 함수를 수정하는 것이다. 예를 들어, 기존 공간 뷰-정의 질의에서 기하-사상 함수인 merge를 intersect로 수정하는 경우이다. 이런 경우에 적용하는 점진적 변경의 방법에는 두 가지가 있다.

먼저 기존의 실체화된 공간 뷰 객체만을 이용하는 방법이다. 예를 들어, 기하-사상 함수가 buffer에서 centroid로 수정된 경우에는 단순히 기존 뷰 객체의 기하데이타만을 수정하여 재작성한다. 즉, 기존에 버퍼링된 모든 뷰 객체들의 기하데이타를 바로 centroid하여 변경시킨다. 따라서 기존 뷰 객체의 기하데이타를 이용할 수 있는 경우에는 그림 8과 같이 뷰 객체의 기하인 SVO_k . $Geom$에 수정된 기하-사상 함수 GMF_{new}를 수행시켜 기존 뷰 객체의 기하데이타를 수정한다.

```
UPDATE  SVO_k  IN  SVC
SET     SVO_k.Geom = GMF_new( SVO_k.Geom <, buffer_parameter> )
```

그림 8 함수명 수정시 공간 뷰 객체를 이용한 점진적 변경

두번째는 추가 정보를 이용하여 재작성하는 방법이다. 예를 들어 기하-사상 함수를 merge에서 intersect로 수정한 경우에는 뷰 객체는 이용할 수 있으나 위의 방법과는 달리 기존의 뷰 객체의 기하데이타를 이용하여 변경할 수 없다. 따라서 그림 9와 같이 VDR을 이용하여 관련된 소스 객체 SCO_i, SCO_j로부터 기하데이타를 검색한 후에, 수정된 기하-사상 함수 GMF_{new}를 적용하여 재정의된 뷰 객체의 기하데이타로 삽입한다.

```
UPDATE  SVO_k  IN  SVC
SET     SVO_k.Geom = GMF_new( SCO_i.Geom <, SCO_j.Geom>)
```

그림 9 함수명 수정시 추가 정보를 이용한 점진적 변경

- **매개 변수 수정**

매개 변수 수정은 기하-사상 함수는 변경되지 않고 단지 함수내의 매개 변수가 수정된 경우이다. 이 경우는 수정된 매개 변수의 유형에 따라 숫자 매개 변수 수정과 비숫자 매개 변수 수정으로 분류된다.

숫자 매개 변수 수정은 기하-사상 함수가 buffer인 경우에만 적용되며, 이 경우에는 buffer에 매개 변수인 지름을 수정한 경우이다. 이러한 경우에는 그림 10과 같이 기존의 뷰 객체의 기하를 수정된 매개 변수 값 $num_parameter_{new}$와 수정되기 전의 매개 변수 값 $num_parameter_{old}$의 차이 값만큼 버퍼링하여 뷰 객체의 기하데이타로 삽입한다. 예를 들어 기하-사상 함수를

buffer(SV.Geom, 50)에서 buffer(SV.Geom, 100)으로 수정한 경우에는 ***SVC***의 모든 객체 SVO_k 에 대하여 buffer(SVO_k.Geom, (100-50))를 적용하여 뷰 객체의 기하데이타를 계산한다.

```
UPDATE  SVO_i  IN  SVC
SET     SVO_i.Geom = GMF( SVO_i.Geom , (num_parameter_new - num_parameter_old))
```

그림 10 숫자 매개 변수 수정시의 점진적 변경

비숫자 매개 변수 수정의 경우는 함수에 적용되는 공간 객체의 대상이 바뀌는 경우이다. 이 경우는 기존의 공간 뷰 객체는 이용할 수 있으나, 뷰 객체의 기하데이타는 바로 이용할 수 없다. 따라서 추가 정보를 이용하여 관련된 소스 객체의 기하데이타에 대하여 연산을 수행한다. 예를 들어 merge(X.Geom, Y.Geom)를 merge(X.Geom, Z.Geom)으로 변경한 경우에는 기존의 공간 뷰 객체의 기하데이타를 이용할 수 없다. 따라서 그림 11과 같이 VDR을 이용하여 뷰를 유도한 소스 클래스 X의 객체 SCO_i와 Z의 객체인 SCO_j로부터 기하데이타를 검색한 후에 기하-사상 함수 ***GMF***를 수행하여 뷰의 기하데이타로 삽입한다.

```
UPDATE  SVO_i  IN  SVC
SET     SVO_i.Geom = GMF( SCO_i.Geom <, SCO_j.Geom> <, numeric_parameter> )
```

그림 11 비 숫자 매개 변수 수정시의 점진적 변경

- **시그니처 수정**

시그니처 수정은 기하-사상 함수의 이름과 매개 변수 모두가 수정된 경우로 매개 변수 수정의 경우와 유사하게 재작성한다. 즉, 기존 공간 뷰 객체는 이용할 수 있으나 기하데이타를 바로 이용하지 못하기 때문에, VDR을 이용하여 점진적 변경을 수행한다. 예를 들어 merge(X.Geom, Y.Geom)를 intersect(X.Geom, Z.Geom)로 변경한 경우에는 그림 12와 같이 기존의 뷰 ***SVC***의 객체 SVO_k와 관련된 소스 객체들인 SCO_i, SCO_j의 기하데이타를 새로운 기하-사상 함수 GMF_{new}에 적용시켜 뷰 객체 SVO_k의 기하데이타를 구한다.

```
UPDATE  SVO_i  IN  SVC
SET     SVO_i.Geom = GMF_new ( SCO_i.Geom <, SCO_j.Geom> <, num_parameter> )
```

그림 12 시그니처 수정시의 점진적 변경

위에서 제시한 함수명 수정, 매개 변수 수정, 시그니처 수정의 경우에 변경의 유형은 기존의 실체화된 뷰 객체를 변경하는 경우이다. 그리고 점진적 변경은 기존의 실체화된 뷰 객체를 이용하는 방법과 VDR을 이용하는 방법을 모두 이용한다.

그림 13은 기하-사상 함수의 수정에 따른 점진적 변경 알고리즘이다. 여기서 Update_Geometry()는 뷰 객체의 기하를 수정하는 함수이고 Search_SourceObj_in_VDR()은 VDR에서 뷰 객체를 유도한 소스 객체를 찾는 함수이다. 세부적으로 뷰 객체의 기하를 직접 이용할 수 있으면 Update_Geometry() 함수를 호출하여 뷰 객체의 기하에 새로운 기하-사상 함수를 적용시켜 변경을 수행한다. 반면에 뷰 객체의 기하를 이용할 수 없으면 Search_SourceObj_in_VDR() 함수를 호출하여 VDR에서 뷰 객체를 유도한 소스 객체들을 검색한 후에 이 객체들에 대상으로 Update_Geometry() 함수를 호출하여 뷰 객체의 기하를 수정한다.

```
SVC   :  Spatial View Class
SVO_k :  Spatial View Object in SVC
SO_i  :  Source Object which derives  SVO_k
GMF   :  Geometry Mapping Function for adaptation
```

```
Procedure  Select_GMF(SVC,GMF,BufferParameter)
begin
  for  each object  SVO_k  in  SVC
    if  use Spatial View's geometry  OR  GMF is Buffer
      Update_Geometry(SVO_k,GMF,SVO_k,BufferParameter)
    else      // in case of using VDR
      GeomSourceObjSet[SO_i] :=  Search_SourceObj_in_VDR(SVO_k)
      Update_Geometry(SVO_k,GMF,GeomSourceObjSet[SO_i],BufferParameter)
  end // for
end
```

그림 13 기하-사상 함수의 수정에 따른 점진적 변경 알고리즘

4.4 From절의 변경

4.4.1 클래스의 추가

From절에서 클래스 추가는 Where절 내에 추가된 클래스와 기존 클래스들 간의 공간 프레디킷 추가를 수반한다. 따라서 재계산 방법을 이용하여 재작성하는 경우에는 추가된 프레디킷으로 인하여 기존의 뷰를 유도하는 공간 연산을 수행한 후에 한번의 공간 조인을 더 수행하므로 시간이 많이 걸린다. 반면에 점진적 변경 방법은 그림 14와 같이 단지 기존의 실체화된 뷰와 추가된 클래스와의 공간 조인만을 수행하므로 재작성 시간이 빠르다.

클래스 추가에 따른 점진적 변경 유형은 기존의 실체화된 뷰 객체를 변경하는 경우이다. 기존의 공간 뷰와

새로 추가된 클래스와 공간 조인을 수행한 후에 기존의 뷰 객체들 중에서 공간 조인의 결과가 아닌 뷰 객체들, 즉 공간 조인 프레디킷을 만족하지 않는 뷰 객체들을 삭제하여 재작성한다.

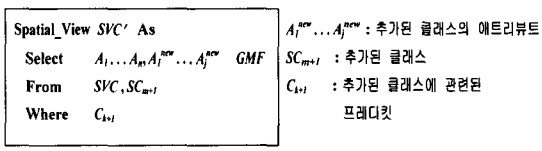


그림 14 클래스 추가시의 점진적 변경

이 경우에 적용되는 점진적 변경은 기존의 실체화된 뷰 객체를 이용하는 방법과 VDR을 이용하는 방법이다. 기존의 실체화된 뷰 객체를 이용하는 방법은 뷰 객체의 기하데이타를 이용할 수 있는 경우로서, 추가된 프레디킷의 매개 변수로 포함되는 기하데이타가 뷰 객체의 기하데이타와 동일한 경우이다. 이 경우는 앞에서 언급한 것과 같이 단순히 기존 공간 뷰와 추가된 클래스간의 공간 조인을 수행한다. 반면에 VDR을 이용하는 방법은 기존 공간 뷰 객체의 기하데이타를 이용할 수 없는 경우이다. 즉 추가된 프레디킷의 매개 변수가 뷰 객체의 기하데이타와 다른 경우이다. 이 경우에는 추가된 프레디킷의 매개 변수로 사용된 소스 객체의 기하를 VDR을 이용하여 가져온 후에 추가된 클래스와의 공간 조인으로 결과를 얻는다.

```
SVC      : Spatial View Class name before redefining
SC_m+1   : Source Class name to be added
C_k+1    : Condition to be added

Procedure From_Add(SVC, SC_m+1, C_k+1)
begin
   if Check_Use_View(SVC, C_k+1)
      SourceObjSet := Spatial_Join(SVC, SC_m+1, C_k+1)
   else      // in case of using VDR
      SourceObjSet := Spatial_Join_with_VDR(SVC, SC_m+1, C_k+1)

   DeleteViewObjSet := Find_View_Obj(SVC.SourceObjSet)
   Delete_View_Obj(DeleteViewObjSet)
   Delete_Obj_in_VDR(DeleteViewObjSet)
   Update_Obj_in_VDR(SVC.SourceObjSet)
end
```

그림 15 클래스 추가에 따른 점진적 변경 알고리즘

From절 내에 새로운 클래스가 추가된 경우에 재작성을 위한 점진적 변경 알고리즘은 그림 15와 같다. 여기서 Check_Use_View()은 추가되는 프레디킷의 매개 변수로 뷰의 기하데이타를 이용할 수 있는지를 판단하는 함수이고, Spatial_Join()은 공간 조인을 수행하는 함수이며, Spatial_Join_with_VDR()은 VDR을 이용하여 공간 조인을 수행하는 함수이다. Find_Vew_Obj()는 삭제되어질 뷰 객체들을 찾는 함수이며, Delete_View_Obj()는 뷰 객체들을 삭제하는 함수이다. 그리고 Delete_Obj_in_VDR()은 삭제된 뷰 객체들에 대한 VDR 정보를 삭제하는 함수이며, Update_Obj_in_VDR()은 VDR의 일관성 유지를 위하여 VDR 정보를 수정하는 함수이다.

세부적으로 먼저 Check_Use_View() 함수를 호출하여 프레디킷의 매개 변수로 뷰의 기하데이타를 이용할 수 있는지를 판단한다. 뷰의 기하 데이타를 이용할 수 있는 경우에는 Spatial_Join() 함수를 호출하여 기존의 뷰와 추가된 클래스간의 공간 조인을 수행한다. 반면에 뷰의 기하데이타를 이용할 수 없는 경우에는 Spatial_Join_with_VDR() 함수를 호출하여 VDR을 이용해 관련된 소스 객체들로부터 기하데이타를 검색한 후에 공간 조인을 수행한다. 그리고 Find_View_Obj() 함수를 호출하여 기존의 뷰 객체들 중에서 공간 조인의 결과가 아닌 뷰 객체들을 찾는다. 이러한 삭제될 뷰 객체들을 Delete_View_Obj() 함수를 호출하여 삭제하고, 그와 관련된 VDR 정보를 Delete_Obj_in_VDR()을 호출하여 삭제한다. 마지막으로 재정의된 뷰-정의 질의를 만족하는 최종 뷰 객체들은 추가된 클래스에 속하는 새로운 소스 객체로부터 유도되어지므로 Update_Obj_in_VDR()을 호출하여 VDR에 새로운 소스 객체에 대한 정보를 추가한다.

4.4.2 클래스의 삭제

From절에서 클래스의 삭제는 Where절에서 이 클래스와 관련된 공간 프레디킷의 삭제를 수반한다. 이 경우에는 기존의 공간 뷰 객체로부터 재정의된 공간 뷰 객체를 구하기가 어렵다. 따라서 이 경우에는 재정의된 공간 뷰의 정의 질의를 다시 수행해야 한다.

4.5 Where절의 변경

4.5.1 프레디킷의 추가

공간 뷰의 재정의 유형이 Where절에 프레디킷을 추가하는 경우에는 기존의 공간 뷰가 새로 정의된 공간 뷰를 포함한다. 이 경우에는 그림 16과 같이 기존의 실체화된 뷰 객체 중에서 필요 없는 객체, 즉 추가한 프레디킷을 만족하지 않는 객체를 삭제하면 된다. 따라서 이 경우에 변경 유형은 기존의 실체화된 뷰 객체를 변경(삭제)하는 경우에 속한다.

이 경우에 적용되는 점진적 변경은 뷰 객체만을 이용하는 방법과 VDR을 이용하는 방법에 모두 속한다. 추

가된 프레디킷 내에 매개 변수로 사용된 애트리뷰트를 뷰가 가지는 경우에는 그림 16과 같이 단순히 기존의 뷰 객체들 중에서 추가된 프레디킷을 만족하지 않는 객체들을 삭제한다. 그러나 뷰가 추가된 프레디킷 내에 매개 변수를 가지지 않는 경우에는 추가 정보인 VDR을 이용하여 관련된 소스 객체로부터 해당하는 애트리뷰트를 검색한 후에, 위와 동일하게 추가된 프레디킷을 만족하지 않는 객체들을 삭제한다.

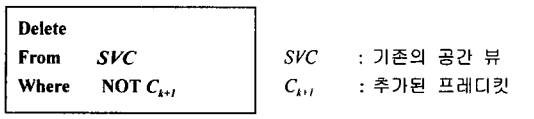


그림 16 프레디킷 추가시의 점진적 변경

Where절에 새로운 프레디킷이 추가된 경우에 공간 뷰의 재작성을 위한 점진적 변경 알고리즘은 그림 17와 같다. 여기서 Check_Use_View()는 추가한 프레디킷에 사용되는 매개 변수를 뷰가 가지고 있는지를 검사하는 함수이고, Is_Delete()는 뷰 객체가 추가한 프레디킷을 만족하는지를 검사하는 함수이다. Delete_View_Obj()는 뷰 객체를 삭제하는 함수이며, Delete_Obj_in_VDR()은 뷰 객체의 VDR 정보를 삭제하는 함수이다. 그리고 Search_SourceObj_in_VDR()은 VDR에서 뷰 객체를 유도한 소스 객체들을 찾는 함수이다.

```
SVC  : Spatial View Class name before redefining
SVO_k : Spatial View Object in SVC
SO_i  : Source Object which derives SVO_k
C_add : Added Condition

Procedure Where_Add(SVC, C_add)
begin
 for each object SVO_k in SVC
    if Check_Use_View(SVC, C_add)
      if Is_Delete(SVO_k, C_add)
         Delete_View_Obj(SVO_k)
         Delete_Obj_in_VDR(SVO_k)
    else // in case view doesn't have attribute in added predicate
       SourceObjSet{SO_i} := Search_SourceObj_in_VDR(SVO_k)
       for each object SO_i in SourceObjSet{SO_i}
          if Is_Delete(SO_i, C_add)
             Delete_View_Obj(SVO_k)
             Delete_Obj_in_VDR(SVO_k)
      end // for
 end // for
end
```

그림 17 프레디킷 추가에 따른 점진적 변경 알고리즘

세부적으로 먼저 Check_Use_View() 함수를 호출하여 프레디킷의 매개 변수로 뷰 객체를 이용할 수 있는지를 판단한다. 뷰를 이용할 수 있는 경우에는 Is_Delete() 함수를 호출하여 뷰가 추가되는 프레디킷을 만족하는지를 판단한다. 만약 뷰가 추가되는 프레디킷을 만족하지 않으면 Delete_View_Obj() 함수를 호출하여 뷰 객체를 삭제하고 Delete_Obj_in_VDR() 함수를 호출하여 삭제한 뷰 객체의 VDR 정보를 삭제한다. 반면에 뷰를 이용할 수 없는 경우에는 Search_SourceObj_in_VDR() 함수를 호출하여 VDR에서 뷰를 유도한 소스 객체들을 검색한다. 그리고 Is_Delete() 함수를 호출하여 소스 객체가 추가한 프레디킷을 만족하는지를 판단하고, 만족하지 않으면 Delete_View_Obj() 함수와 Delete_Obj_in_VDR() 함수를 차례로 호출하여 뷰 객체와 VDR정보를 삭제한다.

4.5.2 프레디킷의 삭제

Where절에서 프레디킷을 삭제하는 경우에는 기존 뷰가 재정의된 공간 뷰에 포함되기 때문에, 기존 뷰 객체들은 그대로 두고 재정의된 뷰를 만족하는 새로운 뷰 객체를 추가하면 된다. 이 경우에는 새로 추가되는 뷰 객체를 얼마나 빨리 찾느냐가 중요한 문제이다. 이를 위하여 본 논문에서는 재정의된 공간 뷰의 정의 질의를 수행할 때 VDR을 이용한다. VDR에 있는 소스 객체들에 대해서는 이미 뷰 객체가 생성되어 있으므로 정의 질의 수행 시 VDR상의 소스 객체들을 제외한 나머지 부분에 대하여 정의 질의를 수행한다.

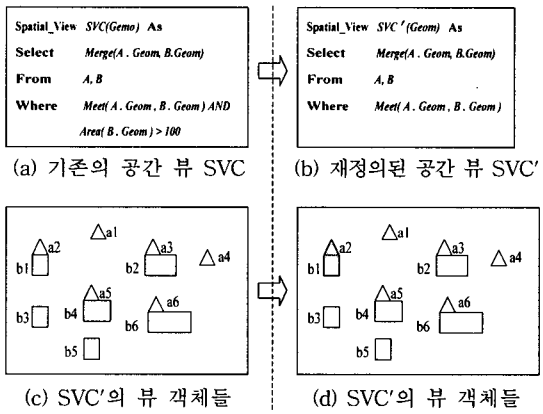


그림 18 프레디킷 삭제의 예

예를 들어, 그림 18(a)의 공간 뷰 ***SVC***에서 프레디킷을 삭제하여 (b)와 같이 공간 뷰 ***SVC'***로 재정의한다고

가정한다. 공간 뷰 ***SVC*** 는 A와 B가 접하고 B의 넓이가 100 이상인 공간 뷰로서 *(a3, b2), (a5, b4), (a6, b6)*로 구성된다. 공간 뷰 ***SVC***가 ***SVC'***로 재정의된 경우에 기존의 실체화된 공간 뷰 객체들은 재정의된 뷰-정의 질의에 만족한다. 따라서 기존의 실체화된 공간 뷰 객체들은 그대로 두고 새로 추가할 부분만을 계산하여 삽입시킨다. 기존의 방법의 경우에는 A의 모든 객체와 B의 모든 객체에 대하여 Meet(A.Geom, B.Geom)을 연산한다. 이 경우에는 일반적으로 6x6개의 객체 쌍에 대하여 연산을 수행한다. 그러나 VDR에 이미 기존의 *(a3, b2), (a5, b4), (a6, b6)* 쌍에 대한 정보가 들어 있으므로 이 객체 쌍들을 연산에서 제외시키고, 나머지 객체 쌍들에 대해서만 연산을 수행한다. 연산 수행후의 결과는 *(a2, b1)*이며 이 쌍에 대하여 새로운 공간 뷰 객체를 생성하여 삽입한다. 이처럼 프레디킷 삭제에 따른 새로운 점진적 변경 방법은 기존의 실체화된 공간 뷰 객체들은 제외시키므로, 질의 수행 범위를 줄여 빠르게 재작성을 수행할 수 있다.

프레디킷 삭제에 따른 점진적 변경 방법은 VDR 정보를 이용하여 탐색 범위를 줄인다. 그리고 기존의 실체화된 뷰 객체는 재정의된 뷰-정의 질의인 프레디킷을 삭제한 뷰를 만족하므로 삭제하지 않는다. 따라서 동일한 객체를 삭제하고 새로이 생성하는 재계산보다 빠르게 수행될 수 있다. 이 경우의 점진적 변경 유형은 새로운 뷰 객체를 삽입하는 경우이다. 그리고 점진적 변경은 VDR을 이용하는 방법이다.

```
SVC      : Spatial View Class name before redefining
SVC'query : Spatial View Class Define Query after redefining

Procedure Where_Delete(SVC, SVC'query)
begin
    SourceObjSet :=  Search_SourceObj_in_VDR(SVC)
    AddSourceObjSet := Spatial_Query_with_VDR(SVC'query, SourceObjSet)
    NewViewObjSet := Create_View_Obj(AddSourceObjSet)
    Insert_Obj_in_VDR(NewViewObjSet, AddSourceObjSet)
end
```

그림 19 프레디킷 삭제에 따른 점진적 변경 알고리즘

프레디킷의 삭제에 따른 뷰의 재작성을 위한 점진적 변경 알고리즘은 그림 19과 같다. 여기서 Search_SourceObj_in_VDR()은 VDR에서 뷰 객체를 유도한 소스 객체들을 찾는 함수이고 Spatial_Query_with_VDR()은 VDR을 이용한 질의 처리 함수이다. Create_View_Obj()은 재정의된 뷰-정의 질의에 따라 새로운 뷰 객체를 만드는 함수이며, Insert_Obj_in_VDR()은 새로 만든 뷰 객체에 대한 VDR 정보를 삽입하는 함수이다.

세부적으로 먼저 Search_SourceObj_in_VDR() 함수를 호출하여 기존의 실체화된 뷰 객체를 유도하는 소스 객체의 쌍들을 모두 검색한다. 그리고 Spatial_Query_with_VDR() 함수를 호출하여 검색된 소스 객체 쌍들을 제외하여 질의를 수행한다. 마지막으로 Create_View_Obj() 함수를 호출하여 질의 결과에 대하여 뷰 객체를 생성하고 Insert_Obj_in_VDR() 함수를 호출하여 새로 생성한 뷰 객체에 대한 VDR 정보를 삽입한다.

4.5.3 프레디킷의 수정

본 논문에서는 Where절에서의 프레디킷의 수정을 세가지 경우로 분류한다. 첫째, 수정되기 전의 프레디킷과 수정된 후의 프레디킷이 동일한 기하 연산자이면서 아래의 조건을 만족하는 경우이다.

geom_op {<,<=} ***C*** → ***geom_op*** {<,<=} ***C'*** (***C*** => ***C'***)
또는,
geom_op {>,=>} ***C*** → ***geom_op*** {>,=>} ***C'*** (***C*** <= ***C'***)
(***geome_op*** 는 기하연산자. ***C***, ***C'*** 는 상수)

예를 들어, 면 객체의 주변 길이를 구하는 기하연산자를 이용한 프레디킷인 perimeter(X.Geom)>100을 perimeter(X.Geom)>150으로 변경하는 경우이다. 이 경우에는 기존의 공간 뷰가 재정의된 공간 뷰를 포함하기 때문에 앞에서 언급한 프레디킷의 추가시의 점진적 변경 방법을 수행한다.

둘째, 변경 전의 공간 프레디킷과 변경 후의 공간 프레디킷이 서로 역관계(reverse)이면서 인수가 바뀐 경우이다. 이 경우에는 재정의된 뷰가 기존 뷰와 같은 의미이기 때문에 별도의 재작성이 필요 없다. 예를 들어, 공간 프레디킷 encloses(X.Geom,Y.Geom)를 within(Y.Geom, X.Geom)으로 수정하는 경우에 두 공간 연산자는 역관계이고 인수가 서로 바뀌었으므로 같은 의미의 공간 뷰를 나타낸다.

셋째는 앞에서 분류된 경우들을 제외한 기타 경우로서, 이 경우에는 프레디킷의 추가 후 삭제를 연속하여 수행한다. 특별히 추가 후 삭제 과정을 처리하는 이유는 프레디킷의 추가를 먼저 수행하는 경우에 탐색 범위를 줄여 효율적으로 삭제를 수행할 수 있기 때문이다.

5. 구현 및 성능평가

5.1 공간 뷰 재작성기의 구현

본 논문에서 제시한 공간 뷰 재정의에 따른 점진적 변경을 실현하기 위하여 공간 뷰 재작성기와 공간 뷰 서비스 제공기를 상용 지리정보시스템인 고딕[8]을 기반으로 구현하였다. 공간 뷰 재작성기는 대부분이 고딕에서 제공하는 LULL 언어를 사용하며, 일부는 C 언어로 구현하였다. 그림 20과 같이 공간 뷰 인터페이스를 통해 입력된 새로운 뷰-정의 질의는 먼저 분석기를 통해 사용자가 입력한 공간 뷰의 재정의에 대하여 재작성의 형태를 분석하고, 각 시멘틱에 따라 Select 재작성기, From 재작성기, Where 재작성기 모듈을 호출하게 된다. 이때 각 재작성기 모듈은 공간 뷰 서비스 제공자가 제공하는 다양한 서비스를 이용하여 재작성을 수행한다.

다음은 공간 뷰 재작성기와 공간 뷰 서비스 제공자의 세부 모듈에 관한 설명이다.

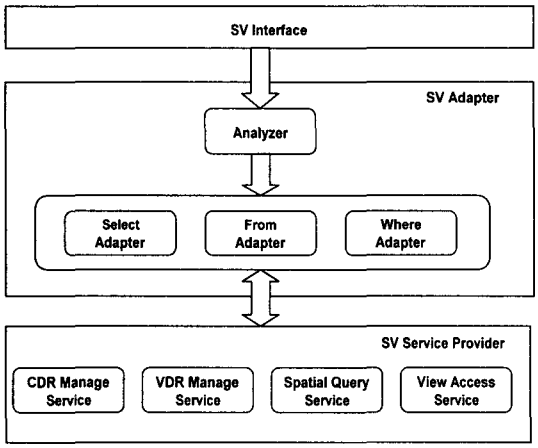


그림 20 공간 뷰 재작성기 및 공간 뷰 서비스 제공자의 세부 모듈

- 공간 뷰 재작성기
  - 분석기(Analyzer) : 사용자가 변경한 뷰-정의 질의를 분석하여 적합한 모듈을 호출
  - Select 재작성기(Select Adapter) : 뷰-정의 질의 Select절의 변경시 재작성을 담당
  - From 재작성기(From Adapter) : 뷰-정의 질의 From절의 변경시 재작성을 담당
  - Where 재작성기(Where Adapter) : 뷰-정의 질의 Where절의 변경시 재작성을 담당
- 공간 뷰 서비스 제공자
  - CDR 관리 서비스(CDR Manage Service) : CDR을 조작, 관리하는 서비스를 제공. CDR은 뷰 클래스와 소스 클래스간의 유도관련성 정보이다.
  - VDR 관리 서비스(VDR Manage Service) : VDR을 조작, 관리하는 서비스를 제공
  - 공간 질의 서비스(Spatial Query Service) : 공간 질의를 처리하는 서비스를 제공
  - 뷰 접근 서비스(View Access Service) : 뷰에 대한 생성, 삭제, 수정, 조작 등에 관한 서비스를 제공

5.2 실험 평가 대상

본 논문에서의 실험 평가 대상은 기본적으로 재계산 방법과 점진적 변경 방법이다. 먼저 공간 뷰 재정의 유형에 따라 표 2와 같이 두 가지 방법을 비교 평가한다. 애트리뷰트 삭제의 경우에는 고딕이 한번 정의된 클래스에 대하여 애트리뷰트 삭제를 지원하지 않으므로 애트리뷰트 삭제는 실험에서 제외한다.

표 2 공간 뷰 재정의 유형에 따른 실험 평가

재작성 방법 / 재정의 유형	점진적 변경	재계산
애트리뷰트 추가	O	O
애트리뷰트 삭제	-	-
기하-사상 함수 수정	O	O
클래스 추가	O	O
프레디킷 추가	O	O
프레디킷 삭제	O	O

점진적 변경 방법의 경우에는 기존의 실체화된 뷰 객체 수에 많은 영향을 받는다. 표 3의 재정의 유형에 따라 실체화된 객체 수를 변화 시키면서 성능 평가를 실시한다. 이 경우에는 두 가지 재정의 유형인 애트리뷰트 추가와 프레디킷 삭제에 대하여 실체화된 뷰 객체 수를 변화 시켜가며 성능 평가를 수행한다. 그 이유는 애트리뷰트 추가와 기하-사상 함수의 변경이 재작성을 수행하여도 실체화된 객체 수에는 변화가 없는 동일한 특성을 가지며, 프레디킷 삭제와 클래스 추가 및 프레디킷 추가는 재작성을 수행하면 실체화된 객체 수에 변화가 있는 동일한 특성을 가지기 때문이다. 특히 프레디킷의 삭제의 경우에는 VDR을 이용한 점진적 변경 방법과 재계산, 그리고 [3]에서 제시한 방법을 비교 평가한다.

표 3 객체 수 변화에 따른 실험 평가

재작성 방법 / 재정의유형	점진적 변경	관계 뷰의 점진적 변경	재계산
애트리뷰트 추가	O	-	O
프레디킷 삭제	O	O	O

5.3 실험 데이타 및 환경

본 논문에서 제시한 점진적 변경 방법의 성능 평가를

위하여 이용한 실험 데이타는 그림 21과 같다. 정확한 성능 평가를 위하여 현재 지자체에서 사용하고 있는 실제 데이타를 이용한다. 클래스의 수는 총 71개로서 건물, 구역, 등고선 등 다양한 공간 객체들로 구성되어 있으며 총 객체 수는 131,534개이다. 그리고 실험 환경으로 CPU 속도는 400MHz이고 메모리는 512M인 디지탈(DEC)사의 ALPHA 스테이션을 이용한다.

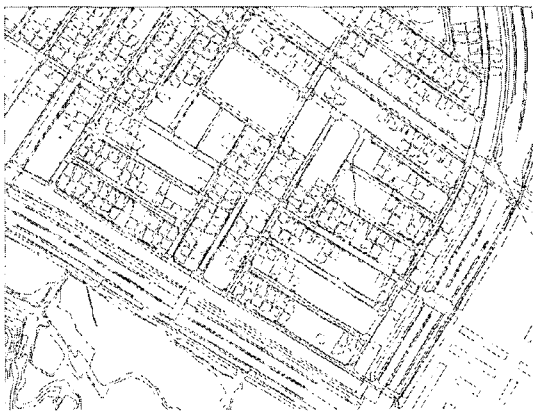


그림 21 성능 평가 데이타

표 4 실험 대상으로 정의된 공간 뷰

	재정의 전의 공간 뷰	재정의 후의 공간 뷰
1	Spatial_View 아파트(분류코드,Geom) As Select 건물,분류코드,건물.Geom From 건물 Where 건물.분류코드 = "아파트"	Spatial_View 아파트(분류코드,소유주,Geom) As Select 건물,분류코드,건물.소유주, 건물.Geom From 건물 Where 건물.분류코드 = "아파트"
2	Spatial_View 아파트(분류코드,Geom) As Select 건물,분류코드,건물.Geom From 건물 Where 건물.분류코드 = "아파트"	Spatial_View 아파트(분류코드,Geom) As Select 건물,분류코드, Buffer(건물.Geom, 10) From 건물 Where 건물.분류코드 = "아파트"
3	Spatial_View 아파트(분류코드,Geom) As Select 건물,분류코드,건물.Geom From 건물 Where 건물.분류코드 = "아파트"	Spatial_View 아파트(분류코드,Geom) As Select 건물,분류코드,건물.Geom From 건물,법정동경계 Where 건물.분류코드 = "아파트" and Within(건물,법정동경계) and 법정동경계.부코드 = "반지동"
4	Spatial_View 동내가로수(구간번호,Geom) As Select 가로수,구간번호,가로수.Geom From 가로수,법정동경계 Where Within(가로수,법정동경계) and 법정동경계.부코드 = "반지동"	Spatial_View 동내가로수(구간번호,Geom) As Select 가로수,구간번호,가로수.Geom From 가로수,법정동경계 Where Within(가로수,법정동경계) and 법정동경계.부코드 = "반지동" and 가로수.구간번호 = 3
5	Spatial_View 동내가로등전선관(Geom) As Select 가로등전선관.Geom From 가로등전선관,법정동경계 Where Within(가로등전선관,법정동경계) and Length(가로등전선관) < 50	Spatial_View 동내가로등전선관(Geom) As Select 가로등전선관.Geom From 가로등전선관,법정동경계 Where Within(가로등전선관,법정동경계)

위의 데이타를 이용하여 성능 평가를 실시한 공간 뷰는 표 4와 같다. [공간 뷰1]은 재정의 유형 중에서 애트리뷰트 추가에 해당하는 공간 뷰이며, [공간 뷰2]는 기하-사상 함수 수정에 해당한다. [공간 뷰3]은 클래스 추가인 경우이며 [공간 뷰4]와 [공간 뷰5]는 각각 프레디킷 추가와 삭제인 경우이다.

5.4 공간 뷰의 재정의 유형에 따른 성능 평가

본 논문에서 제시한 점진적 변경 방법과 재계산 방법간의 성능 평가를 위하여 5.3절에서 언급한 공간 뷰의 재정의 유형에 따른 성능 평가를 실시한다. 표 5와 같이 [공간 뷰1]은 재정의 유형이 애트리뷰트 추가로서, 재정의 전의 뷰 객체 수와 재정의 후의 뷰 객체 수에는 변화가 없다. 마찬가지로 [공간 뷰2]는 기하-사상 함수 수정의 경우로 재정의 전의 뷰 객체 수와 재정의 후의 뷰 객체 수의 변화는 없다. [공간 뷰3]은 클래스 추가로서 재정의 전의 뷰 객체 수는 477개이나 클래스 추가로 인하여 재정의 후의 뷰 객체 수는 줄어든다. [공간 뷰4]는 프레디킷 추가로서 추가되는 프레디킷으로 인해 공간 뷰 객체가 재정의 후에 줄어든다. 마지막으로 [공간 뷰5]는 프레디킷 삭제로서 삭제되는 프레디킷을 만족하는 공간 뷰 객체가 생기므로 재정의 후의 뷰 객체 수는 증가하게 된다.

표 5 성능 평가에 사용되는 공간 뷰의 분석

(단위 : 개)

	재정의 유형	소스 객체 수	재정의 전의 뷰 객체 수	재정의 후의 뷰 객체 수
공간 뷰 1	애트리뷰트 추가	13565	477	477
공간 뷰 2	기하-사상 함수 수정	13565	477	477
공간 뷰 3	클래스 추가	13565 * 1	477	239
공간 뷰 4	프레디킷 추가	3867 * 19	478	45
공간 뷰 5	프레디킷 삭제	542 * 19	145	419

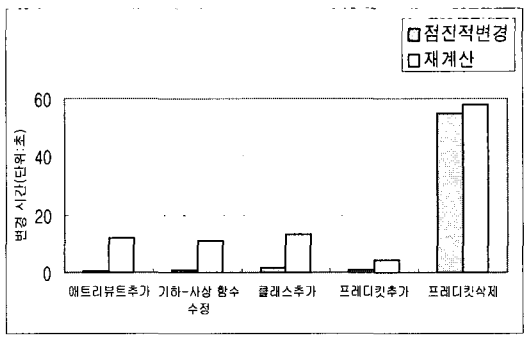


그림 22 점진적 변경 및 재계산의 수행 시간 비교

그림 22은 공간 뷰의 재정의 유형에 따른 점진적 변경 방법과 재계산 방법에 대한 성능 평가의 결과이다. 애트리뷰트 추가의 경우에 점진적 변경 방법은 VDR을 이용하여 기존의 뷰 객체만을 변경하므로 기존의 객체

를 삭제하고 다시 새로 생성하는 재계산 방법보다 약 30배정도 빠르다. 마찬가지로 기하-사상 함수 수정, 클래스 추가, 프레디킷 추가는 점진적 변경 방법이 재계산 방법보다 빠르다. 반면에 프레디킷 삭제의 경우에 점진적 변경 방법은 VDR을 탐색하여 VDR에 있는 객체들을 질의 수행에서 제외시키는데, VDR의 구조가 복잡하고 탐색을 위한 어떠한 색인도 구성되어 있지 않으므로 재계산 방법과 비교하여 개선이 크게 이루어지지 않았다. 그러나 객체수가 다를 경우에 성능 개선 효과가 다르다는 것을 그림 24에서 보여줄 것이다.

5.5 객체 수 변화에 따른 성능 평가

점진적 변경 방법은 재정의 전에 기존의 실체화되어 있는 뷰 객체 수에 많은 영향을 받는다. 특히 애트리뷰트 추가의 경우에 점진적 변경 방법은 기존의 실체화된 뷰 객체들에 새로운 애트리뷰트의 값들을 추가한다. 반면에 재계산 방법은 모든 객체를 삭제하고 다시 생성하므로 기존의 실체화된 뷰 객체들이 많을수록 동일한 뷰에 대하여 점진적 변경 방법이 재계산 방법보다 더 효율적이다.

그림 23은 애트리뷰트 추가의 경우인 [공간 뷰1]을 변형시켜 기존의 실체화된 뷰 객체들의 수를 다르게 하고 각각에 대하여 성능을 평가하였다. 애트리뷰트 추가의 경우이므로 변경 전과 변경 후의 뷰 객체 수의 변화는 없다. 이 결과에서는 기존의 실체화된 뷰 객체 수가 많을수록 점진적 변경 방법이 재계산 방법보다 빠르다는 것을 보여준다.

(단위 : 개)

	재정의 유형	소스 객체 수	재정의 전의 뷰 객체 수	재정의 후의 뷰 객체 수
공간 뷰 1-1	애트리뷰트 추가	13565	124	124
공간 뷰 1-2	애트리뷰트 추가	13565	477	477
공간 뷰 1-3	애트리뷰트 추가	13565	964	964

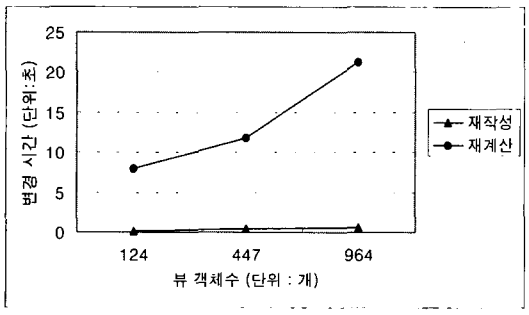


그림 23 애트리뷰트 추가시의 수행 시간 비교

그림 24는 프레디킷 삭제의 경우에 점진적 변경, [1], 재계산 방법을 기존의 실체화된 뷰 객체 수에 따라 비교 평가한 것이다. 먼저 [공간 뷰5]를 변형시켜 실체화된 객체 수를 변경하였다. 프레디킷 삭제의 경우이므로 소스 객체 수나 재정의 후에 뷰 객체 수는 동일하며 단지 재정의 전의 뷰 객체 수만 다르다.

재계산 방법의 경우에는 기존의 실체화된 뷰 객체수가 많을수록 조금씩 느려진다. 그 이유는 재계산 방법이 이미 실체화된 뷰 객체를 삭제한 후에 다시 생성하므로 실체화된 뷰 객체 수가 많을수록 삭제하는 시간이 더 걸리기 때문이다. 그러나 삭제에 걸리는 시간은 전체 변경 시간에 비하여 매우 적기 때문에 거의 변화가 없다. 본 논문에서 제시한 점진적 변경이나 [3]의 방법의 경우에는 기존의 실체화된 뷰 객체들에 추가하는 뷰 객체만을 계산하여 추가하므로 기존 뷰 객체가 많을수록 변경 시간이 빠르다.

(단위 : 개)

	재정의 유형	소스 객체 수	재정의 전의 뷰 객체 수	재정의 후의 뷰 객체 수
공간 뷰 5-1	프레디킷 삭제	542 * 19	69	419
공간 뷰 5-2	프레디킷 삭제	542 * 19	145	419
공간 뷰 5-3	프레디킷 삭제	542 * 19	326	419

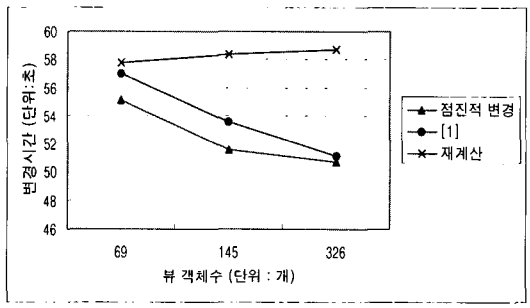


그림 24 프레디킷 삭제시 수행 시간 비교

6. 결론 및 향후 연구

다양한 사용자의 관점에 의해 정의되는 공간 뷰는 일반적으로 질의 수행의 향상을 위해 실체화한다. 실체화된 공간 뷰에 대하여 사용자의 관점이 변경되는 경우에는 공간 뷰의 재정의가 필요하며, 일관성 유지를 위하여 기존의 실체화된 뷰 객체들을 변경하는 재작성이 필요하다. 재작성 방법은 이미 실체화된 객체를 모두 삭제하고 처음부터 다시 생성하는 재계산과 기존의 실체화된 객체를 이용하여 새로 생성할 객체만을 생성하거나 기존의 객체만을 변경하는 점진적 변경이 있다.

본 논문에서는 먼저 공간 뷰의 재정의 유형을 분류하고 재정의시 일관성 유지를 위한 점진적 변경 방법을 제시하였다. 공간 뷰-정의 질의의 변경을 Select-From-Where로 나누고 각각에 적용될 수 있는 점진적 변경 방법을 제시하고, 이것을 상용 지리정보시스템인 고딕상에 설계 및 구현하였다.

마지막으로 점진적 변경 방법의 성능을 검증하기 위하여 실제 데이타를 대상으로 실험 평가하였다. 먼저 공간 뷰의 재정의 유형에 따라 점진적 변경 방법과 재계산 방법과의 성능을 비교하였다. 또한 본 논문에서 공간 뷰 재작성을 위해 제시된 점진적 변경과 기존의 관계 뷰를 위한 점진적 변경을 비교하였으며 기존의 실체화된 뷰 객체 수의 변화에 대해서도 비교하였다. 성능 평가를 통하여 본 논문의 점진적 변경 방법이 재계산 방법 및 기존의 변경 방법보다 빠르다는 것을 입증하였으며, 재정의 전의 실체화된 뷰 객체가 많을수록 더 효율적이라는 것을 알 수 있었다.

향후 연구로는 공간 뷰의 재작성을 위해 제시된 점진적 변경 알고리즘들을 더 효율적으로 수행하기 위한 뷰 유도관련성의 데이타 구조에 대한 계속적인 연구와 본 논문에서 분류한 최소 단위의 재정의 유형들이 혼합하여 발생하는 경우에 처리 방법에 대한 연구가 필요하다. 그리고 뷰에 대한 변경이 소스 클래스에 전파될 수 있는 변경 가능한 공간 뷰(updatable spatial view)에 관한 연구가 필요하며, 지금까지 설계 및 구현한 공간 뷰 시스템을 확장하여 클라이언트-서버 구조 및 분산 컴퓨팅 환경에서의 공간 뷰의 일관성 제어에 관한 연구를 수행할 계획이다.

참 고 문 헌

[1] Sang-Ho Moon, Bong-Hee Hong, Incremental Update Algorithms for Materialized Spatial Views by Using View Derivation Relationships, DEXA 97, pages 539-550, 1997.

[2] 문상호, 김동우, 반재훈, 홍봉희, 객체지향 공간 뷰의 설계 및 구현, 한국정보과학회 논문지, vol. 26, no. 2, 1999

[3] Ashish Gupta, Inderpal Singh Mumick, Kenneth A. Ross., Adapting materialized views after redefinitions., SIGMOD, pages 211-222, 1995

[4] Ashish Gupta, Inderpal Singh Mumick, Kenneth A. Ross., Adapting materialized views after redefinitions., Columbia University Technical Report number CUCS-010-95, 1995

[5] Ashish Gupta and Inderpal Singh Mumick, *Maintenance of Materialized Views: Problems*, Techniques, and Applications, Proc. of Int'l Conf. on Data Engineering, pages 3-18, 1995.

[6] C. Souza dos Santos, Design and Implementation of Object-Oriented Views, Proc. of Int'l Conf. and Workshop on Database and Expert Systems Applications (DEXA), pages 91-102, 1995.

[7] Harumi. A. Kuno and Elke A. Rundensteiner, Materialized Object-Oriented Views in MultiView, Int'l Workshop on Research Issues on Data Engineering : Distributed Object Management, 1995

[8] Laser Scan Ltd, GOTHIC CONCEPTS, Training Course, 1995.

[9] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, Kyuseok Shim, Optimizing Queries with Materialized Views, IEEE, pages 190-200, 1995

[10] Ashish Gupta, Inderpal Singh Mumick, and V. S. Subrahmanian, Maintaining Views Incrementally, Proc. of ACM-SIGMOD Int'l Conf. on Management of Data, pages 157-166, May, 1993.

[11] C. Souza dos Santos and Emmanuel Waller, The O_2 Views User's Manual, version 1.0, December, 1993.

[12] Michel Scholl and Agnes Voisard, Object-Oriented Database Systems for Geographic Applications: an Experiment with O_2, Proc. of Geographic Management Systems Workshop, pp 103-137, 1991.

[13] C. J. Date, "Views, An Introduction to Database System, Addison-Wesley Publishing Company, pages 466-496, 1995.

[14] Giovanna Guerrini, Elisa Bertino, Barbara Catania, and Jesus Garcia-Molina, A Formal Model of Views for Object-Oriented Database Systems, Technical Report, DISI-96-2, 1996.

[15] G. Wiederhold, Views, Objects, and Databases, IEEE Computer, pages 37-44, 1986.

[16] Harumi A. Kuno and Elke A. Rundensteiner, The MultiView OODB View System: Design and Implementation, Theory and Practice of Object Systems, vol. 2, no. 3, pages 203-225, 1996.

[17] Serge Abiteboul, Anthony Bonner, Objects and Views, Proc. of ACM-SIGMOD Int'l Conf. on Management of Data, pages 238-247, May, 1991.

[18] Laser-Scan, Writing and Developing Applications using GOTHIC ADE, Issue 2.0, 1995.

[19] Nick Roussopoulos, Ahungmin M. Chen, and Stephen Kelley, The ADMS Project: Views R Us, Int'l Conf. on Data Engineering, pages 19-28, 1995.

[20] 강성주, 박지숙, 이석호, 객체 지향 DBMS에서 질의 재수행 과정이 없는 뷰 메카니즘의 설계, '95 한국정보과학회 가을 학술발표논문집, vol. 22, no. 2, pages 85-88, 1995.

[21] 문상호, 김동현, 홍봉희, 실체화된 공간 뷰의 점진적 변경, 한국정보과학회 논문지, vol. 25, no. 1, pages 37-50, 1998

반 재 훈

1997년 2월 부산대학교 컴퓨터공학과 공학사. 1998년 2월 부산대학교 컴퓨터공학과 공학석사. 1999년 3월 ~ 현재 부산대학교 컴퓨터공학과 박사과정. 관심분야는 지리정보시스템(GIS), 공간 뷰, GIS 표준화

문 상 호

1991년 2월 부산대학교 컴퓨터공학과 공학사. 1991년 ~ 1993년 한국기계연구원 전산시스템실 연구원. 1994년 2월 부산대학교 컴퓨터공학과 공학석사. 1998년 2월 부산대학교 컴퓨터공학과 공학박사. 1998년 3월 ~ 현재 위덕대학교 컴퓨터공학과 전임강사. 관심분야는 지리정보시스템(GIS), 공간뷰, 객체지향데이타베이스, GIS 표준화

홍 봉 희

1982년 서울대학교 전자계산기공학과 졸업(공학사). 1984년 서울대학교 대학원 전자계산기공학과 졸업(공학석사). 1988년 서울대학교 대학원 전자계산기공학과 졸업(공학박사). 현재 부산대학교 공과대학 컴퓨터공학과 정교수. 관심분야는 병렬공간 DB, 분산공간 DB, 개방형 GIS