멀티패러다임 3D 모델링 DSEL의 설계

장학상° 김재우 권기항
동아대학교 컴퓨터공학과
{blitzergO}@mail.donga.ac.kr kizoo@bluette.com khkwon@daunet.donga.ac.kr

A Design of A Multiparadigm 3D Modeling DSEL

Haksang Jang° Jae-Woo Kim Kee-hang Kwon
Dept. of Computer Engineering, Dong-A University

요 약

본 논문은 멀티패러다임 프로그래밍을 이용한 효과적인 문제 해결 방식을 보이고 멀티패러다임 사고를 적용한 소프트웨어 설계의 실용적 타당성을 예시하기 위하여 3D 모델링 분야를 위한 멀티패러다임 DSEL을 설계한다. 먼저 3D 모델링 응용에 필요한 프로그래밍 요소를 문제의 특성을 고려하여 패러다임 영역별로 분할하고, 이에 따라 분할된 요소를 대응하는 단일 패러다임을 이용해 설계한다. 마지막으로 단일한 패러다임을 이용해 설계된 각 기능을 통합하고 보완하여 멀티패러다임의 협력 효과를 극대화한다.

1. 서 론

프로그래밍 패러다임은 컴퓨터에서 계산 수행의 의미와 실행될 작업들의 구성 방법에 관한 개념적인 수단을 의미한다[3]. 패러다임은 일관된 관점을 통해 문제 해결에 집중하기 위한 명확한 분석 능력과 효과적인 사고 방법을 제공한다.[5] 그러나 복잡한 시스템은 본질적으로 다양한 패러다임적 특성을 지니기 때문에 단일한 패러다임만을 사용하는 것은 적절하지 못하다.[7]

복잡한 시스템의 한 가지 예인 3D 모델링 분야는 다양한 문제 분야를 내포하고 있을뿐만 아니라 응용 분야에 따라 다양한 개념을 지니므로 멀티패러다임적 사고방식을 적용하기에 알맞다.

3D 모델링을 위한 멀티패러다임 프로그래밍 언어를 설계하기 위해, 강력한 추상화 수단을 제공하는 모 언어(host language)를 이용하여 효율적으로 전문 분야를 표현하는 DSEL(Domain Specific Embedded Language) 접근방식이 효과적이다.

본 논문은 멀티패러다임 프로그래밍을 이용한 효과적인 문제 해결 방식을 보이고 멀티패러다임 사고를 적용한 소프트웨어 설계의 실용적 타당성을 예시하기 위하여 3D 모델링 분야를 위한 멀티패러다임 DSEL을 설계한다.

2. 멀티패러다임 3차원 모델링 DSEL 설계

다양한 개념적 도구를 이용하여 복잡한 문제에 대해 다양한 패러다임을 적용하여 효과적으로 해결하는 멀티패러다임의 실용적 타당성을 제시하기 위한 방법으로, 복잡한 시스템의 하나의 예인 3D 모델링 분야를 선택한다. 또한 멀티패러다임 3D 모델링 언어를 구현하기 위해, 모 언어를 이용하여 전문 분야에 대해 고급 추상화 수단을 제공하는 DSEL 접근방법을 사용한다. 이를 위해 먼저 3D 모델링 응용에 필요한 프로그래밍 요소를 문제의 특성을 고려하여 패러다임 영역별로 분할하고, 이에 따라 분할된 요소를 단일 패러다임별로 언어 기능을 설계하며, 마지막으로 패러다임간 협력효과를 극대화하기 위해 각각의 기능을 통합하고 보완한다.

2.1 멀티패러다임 프로그래밍

프로그래밍 패러다임은 컴퓨터에서 계산 수행의 의미와 실행될 작업들의 구성 방법에 관한 개념적인 수단을 의미한다.[3] 대표적인 프로그래밍 패러다임으로는 객체지향, 함수형, 논리형 등이 있다.

패러다임은 일관된 관점을 통해 문제 해결에 집중하기 위한 명확한 분석 능력과 효과적인 사고 방법을 제공한다.[5] 그러나 복잡한 시스템은 본질적으로 다양한 패러다임적 특성을 지니기 때문에 단일한 패러다임만을 사용하여 구현하는 것은 적절치 못하다.[7]

이러한 단점을 해결하기 위해서는 단일한 사고를 강요하기보다 문제 해결을 위한 다양한 개념적 도구를 이용하여 부분적인 문제에 대해 적절한 패러다임을 적용하고 이들을 결합하여 전체 문제를 효과적으로 해결하는 것이 바람직 하다.

이렇듯 복잡한 문제에 대해 여러 패러다임을 사용해 효과적으로 해결 하는 방법을 멀티패러다임 프로그래밍이라고 하며. 그러나 멀티패러다임 프로그래밍을 하기 위한 각 패러다임을 위한 기능을 무원칙하게 조합하는 것이 아니라 언어의 유연성(flexibleness)과 직교성(othogonality)을 만족하도록 유기적으로 결합된 언어를 멀티패러다임 언어라고 한다.

2.2 3D 모델링

3D 컴퓨터 그래픽(3D computer graphic)의 영역은 크게 세 부분으로 구분된다. 3D 컴퓨터 그래픽에서 원근조절(perspective)나 레스터 변환(rasterization)등의 하드웨어와의 파이프라인을 담당하는 영역을 3D 그래픽스(3D graphics)라고 하고, 움직임(motion)이나 충돌(collision), 변형(morphing) 등의 객체의 연속적인 움직임과 반응을 표현하는 영역을 3D 애니메이션(3D animation)이라 한다. 3D 그래픽스의 자원을 사용해 3D 애니메이션을 위한 구조를 표현하거나 관리를 하는 역할을 하는 영역을 3D 모델링(3D modeling)한다.[1]

3D 모델링 라이브러리는 일반적으로 3D 그래픽스를 포함하며 모델링 객체를 효율적으로 관리하게 하는 장면그래프(scene graph)나 모델링 객체를 간단한 연산의 조합으로 표현할 수 있는 CSG(constructive solid geometry)와 같은 솔리드 모델링(solid modeling) 등의 개념을 사용하며, 3D 애니메이션과 사건 처리(event handling)도 일부 지원하기도 한다.

2.3 DSL과 DSEL

DSL(Domain Specific Language)은 특정 분야에서 사용하기 위한 프로그래밍 언어로, 범용적인 목적은 아니지만 전문 분야에 대한 강력한 추상화 수단을 제공한다. DSL의 대표적인 예로는 HTML이나 VHDL, Emacs의 eLisp 등이 있다. DSL 프로그래밍은 범용 언어를 이용한 프로그래밍보다 더 간결한 코드를 작성하고, 개발 비용이 적게 들고 검증이나 최적화가 더 쉽다.[4]

그러나 DSL은 수행 효율이 떨어지고, 언어를 설계하고 구현하는데 비용이 많이 소모되므로, 이미 개발된 언어에서 공통된 요소를 내려받아 모 언어의 장점을 포용하면서 효율적으로 전문 분야를 표현할 수 있는 DSEL의 접근방법이 안정적이고 효과적이다.

2.4 설계 원칙과 방법

멀티패러다임 DSEL의 설계를 위해 다음과 같은 방법을 사용한다.

- 패러다임 영역 분해(Paradigm Domain Decomposition) - 3D 모델링 응용에 필요한 프로그래밍 요소를 문제의 특성을 고려하여 적절한 패러다임에 적용되도록 분해한다.
- 패러다임 영역 별 언어 기능 설계(Paradigm-Specific Language Elements Design) - 분해된 문제 요소를 각 단일 패러다임에 대응시키고 언어 기능을 설계하여 효과적인 문제 해결 방법을 보인다.
- 패러다임 합성을 위한 언어 기능 설계(Language Elements Design for Paradigm Combination) - 각 패러다임별 기능을 통합하고 보완함으로써 패러다임간 협력효과를 높이고 멀티 패러다임 사고를 적용한 소프트웨어 설계의 실용적 타당성을 예시한다.

3. 패러다임 영역 분해

3D 모델링 언어에서 기본 요소는 기하형태(geometry)와 연산(operation), 외관(appearance)으로 이루어진다. 기하형태에는 크게 솔리드(solid) 형태와 정점(vertex)간의 연결정보를 이용한 형태가 있고, 연산은 형태의 변환(transform)을 위한 연산으로 이동(translation)과 회전(rotation), 크기변환(scaling)이 있으며, 솔리드 모델링[6]의 불린(boolean) 연산으로 합집합(union), 교집합(intersection), 차집합(difference)을 제공한다. 외관은 형태의 질감(material) 등을 표현하기 위한 것으로 기본적으로 색깔(color)만을 제공한다. 또한 3D 모델링 언어는 사건처리(event handling)이나 3D 애니메이션을 지원하기도 하나 본 논문에서는 설계의 간결함을 위해 생략한다.

- 객체지향 패러다임과 장면 그래프 관리

  기본 요소를 조합하기 위해 3D 모델링에서는 장면 그래프를 제공한다. 장면 그래프는 3D 모델링 라이브러리에서 대부분 사용하는 자료 구조로 객체지향 패러다임에 기반을 두고 있다.

  장면 그래프는 Bounded 타입과 Bounded의 서브타입(subtype)인 Transformable과 Constructive을 통해 서브타입 다형성을 구현한다.

  기본 요소(형태, 연산, 속성) 데이터는 노드(node)가 될 수 있으며 Transformable하고 Constructive한 특징을 지닌다. 이러한 노드는 그룹(group)의 자식(child)이 될 수 있다. 이러한 그룹은 더 상위의 그룹으로 묶여질 수 있으며, 이렇게 계층적인 구조를 이룬다.

  이렇게 구성된 장면 그래프는 그룹에 소속된 노드 전체의 경계(bound)를 관리하게 되어 렌더링시에 필요 없는 노드를 추려내고(scene graph culling) 충돌검사(collision detection)시에 계산이 간편해지는 장점을 지니게 된다.

  장면 그래프 관리는 서브타입 다형성을 지니는 객체지향 패러다임의 고유의 특징으로 인해 장면 그래프에 여러 형태의 노드를 사용하는 것이 가능하여 재사용성과 확장성을 지닌다.

- 함수형 패러다임의 연산 처리와 솔리드 모델링

  대부분의 장면 그래프에서 변환(transform) 노드는 클래스 형태를 지니고 있으나, 변환의 특성은 오히려 함수적인 특성을 지닌 연산에 가깝다.

  이러한 변환 연산(translate, rotate, scale)을 함수 형태로 처리함으로써 함수합성(function composition)을 통해 더욱 간단하게 처리될 수 있게 되었다.

CSG에 대한 불린 연산(union, interaction, difference)은 장면 그래프를 활용해 CSG-트리(tree)와 유사하게 설계함으로써 솔리드 모델링의 특성을 잘 표현할 수 있다. 불린 연산 또한 함수 합성을 통해 복잡한 연산을 단순화 시켜 표현할 수 있다.

연산 처리는 함수가 기본요소(first-class)인 함수형 패러다임 고유의 특징[7]과 커리(curry)나 폴딩(folding) 등의 함수형 언어가 가지는 장점를 지님으로써 유연성과 포괄성을 유지한다.

4. 패러다임 영역 별 언어 기능 설계

4.1 장면 그래프 관리

장면 그래프는 Haskell[2]의 기능을 사용하여 객체지향 프로그래밍의 타입 요소를 프로그래밍한다.

```
type BBox = (Vec3f, Vec3f)
class Bounded a where
    bbox :: a -> BBox
class Bounded a => Transformable a where
    scale, translate :: vec3f -> a -> a
    rotate :: rotation -> a -> a
class Bounded a => Constructive a where
    union, intersection, difference :: a -> a -> a
```

정의 1. 장면 그래프의 타입 요소

Bounded는 객체가 지닌 경계 상자(bounding box)의 값을 돌려주는 bbox 연산을 정의한다. bbox 연산은 장면 그래프의 기본적인 요소를 이루고 다른 클래스를 구성하는 기본이 된다.

Transformable는 Bounded의 서브타입으로 객체의 변환에 필요한 크기변환(scale), 회전변환(rotate), 위치변환(translate) 연산을 정의한다. 이러한 변환의 결과로 각 객체의 정보가 변환되고 경계 상자는 다시 계산된다.

Constructive는 CSG를 위한 union, intersection, difference의 세가지 불린 연산을 포함하는 Bounded의 서브타입이다. 불린 연산에 의해 조합된 객체의 경계 상자는 재계산 된다.

```
data Node = Geo Geometry | Appr Appearance | Op Operation
type Geometry = (Group, BBox)
type Appearance = Material
instance Bounded Node where
    bbox n = ···
instance Bounded Node => Transformable Node where
    scale s n = ···
    rotate r n = ···
    translate t n = ···
instance Bounded Node => Constructive Node where
    union n n = ···
    intersection n n = ···
    difference n n = ···
appearance c :: Material -> Node
```

정의 2. Node 데이터의 객체지향적 요소

Node는 Transformable하고 Constructive한 특징을 지닌 언어에서 가장 기본을 이루는 데이터이다. Geometry는 기하형태를 나타내는 형으로, 그룹과 경계박스로 이루어진다. Appearance는 재질을 가지는 형으로 설계의 단순함을 위해 단지 색상만을 유지한다. 타입에 대한 인스턴스는 Node 타입으로 적용이 된다. appearance는 속성에 대한 생성자(constructor)로 Appearance 노드를 생성한다. Appearance 노드는 형제노드에 대해서만 적용되는 특징을 지닌다.

```
data Group = Group [Node] | Leaf Shape
group :: [Node] -> Node
```

정의 3. Group 데이터와 생성자

Group은 실질적인 기하적인 정보를 들고 있는 Shape를 표현하거나 그러한 Shape로 구성된 노드를 그룹화 하기 위한 형이다. Group은 부모 Group의 자식노드로 묶어질 수 있으므로 Group과 Node는 상호간에 재귀적으로 선언된다. group은 그룹에 대한 생성자로 여러 노드를 묶는다.

```
data Shape = Cube Vec3f FaceSet | Sphere Float FaceSet
           | FS FaceSet | LS LineSet | PS PointSet
                        ...
cube :: Vec3f -> Node
sphere :: Float -> Node
faceSet :: Coord -> CoordIndex -> Node
lineSet :: Coord -> CoordIndex -> Node
                        ...
```

정의 4. Shape 데이터와 생성자

Shape는 실질적인 기하정보를 담고 있는 형이다. Cube와 Sphere는 솔리드 정보를 지니고 있으며, FaceSet, LineSet, PointSet은 복수의 정점과 연결 정보를 표현하는 형이다. cube, sphere 등은 솔리드 모델의 생성자이며 faceSet, lineSet 등은 면이나 선, 점등으로 이루어진 기본적인 기하정보의 조합을 생성하기 위한 생성자이다.

```
group [ sphere 2, appearance(0,255,0)
        group [ cube (2,2,4), appearance (255,255,0) ] ]
```

예제 1. 간단한 장면 그래프 예제

예제 1과 같이 파란색 구와 노란색 상자를 보여주는 코드를 간단히 작성할 수 있다.

4.2 함수형 프로그래밍 요소

Operation 형에 사용되는 변환 연산과 불린 연산에 대한 함수는 아래와 같다.

```
type Operation = Node -> Node
operation :: Node -> Node
```

정의 5. 연산(Operation) 데이터와 생성자

Operation은 노드를 인자로 받아서 노드를 리턴 하는 함수의 형이 된다. Operation은 함수형 언어의 특징인 curry를 사용하여 Transformable한 Node나 Constructive한 Node로부터 유연하게 확장될 수 있다. Operation는 VRML의 경우처럼 자식노드에 대해서만 처리하는 방법이 아닌, 형제노드와 그 자식노드에 대해서 처리하는 그룹화(grouping)을 수행한다.

```
geonode :: Node
geonode = translate (0,1.4,0) (cube (2,2,2))
```

예제 2. 완전하게 연산을 적용한 기하형태 노드

예제 2는 정육면체를 이동하여 새로운 장소에 위치를 시키는 간단한 예제이다. 이렇게 생성된 기하형태는 변환이 일어난 상태로 그룹화되게 된다.

```
opnode :: Node -> Node
opnode = operation (difference (sphere 3))
```

예제 3. 완전하게 연산을 적용하지 않은 기하형태 노드

예제 3은 구체를 기본으로 하여 불린 연산을 적용할 수 있는 연산 노드를 생성했다. 이렇게 생성된 노드는 어떤 연산도 일어나지 않은 상태로 그룹화되며 렌더링 시에 다른 노드를 이용해 찻집합을 구하게 된다.

```
proto x y z = group [ sphere x, appearance z
                      group [ box y, appearance z ] ]
```

예제 4. 장면 그래프의 추상화

예제 2은 Haskell이 기본적으로 제공하는 함수를 통해 장면 그래프의 추상화를 달성했다. 추상화의 요소는 언어 설계의 중요한 부분이다. Haskell과 같은 함수형 언어에서는 함수가 고차(higher-order) 특징과 기본요소(first-class)의 특성을 지니고 있으므로 이보다 복잡한 표현식도 간단히 확장할 수 있다.

5. 패러다임 합성을 위한 언어 기능 설계

멀티패러다임이란 각각의 패러다임 요소를 단순히 결합하는(combine) 것이 아니라 적절하게 조화를 이루는(blend) 형태이어야 한다. 장면 그래프의 확장성 있는 구조를 유지하는 객체 지향 패러다임의 요소와 유연성을 추가한 함수형 패러다임 요소는 서로 다른 하나로 존재하는 것이 아니라, 자연스럽게 합성되어야 한다.

```
group [ operation ((translate (0,5,0)).(rotate (1,0,0,0.8))),
        (cube (3,1,4)) 'union' (cube (1,2,3)),
        operation (intersection (sphere 5)),
        group [ operation (translate (0.1,5,0)), sphere 4 ] ]
```

예제 5. 패러다임 합성을 사용한 장면 그래프 예제

예제 5에서는 두 개의 큐브를 받아들여 union한 상태의 노드와 sphere 5에 대해 intersection할 연산, 그리고 sphere 4를 translate할 연산이 그룹화 되었다.

결과적으로 설계된 DSEL은 객체지향 패러다임과 함수형 패러다임의 자연스러운 합성으로 한층 뛰어난 표현력과 확장력을 지니게 되었다.

6. 결 론

본 논문에서는 3D 모델링 분야를 위한 멀티패러다임 DSEL의 설계를 통해서 멀티패러다임 프로그래밍을 이용한 효과적인 문제 해결 방식을 보이고 멀티패러다임 사고를 적용한 소프트웨어 설계의 실용적 타당성을 예시하였다. 설계된 멀티패러다임 DSEL는 객체지향적 구조가 가지는 재사용성과 확장성을 그대로 유지하면서 함수형 패러다임이 가지는 유연성과 포괄성이 유기적으로 잘 결합되었다. 멀티패러다임 DSEL 설계시 사용된 소프트웨어 설계 기법은 단일 패러다임만으로 응용프로그램을 작성하는 프로그래머는 물론 3D 모델링 응용 프로그래밍을 위한 라이브러리를 설계하는 구현자에게 문제 해결을 위한 다양한 사고과 효과적인 방법를 제공한다.

그러나 Haskell은 3D 모델링 분야의 다양한 문제를 효과적으로 풀어낼 수 있는 모든 프로그래밍 패러다임을 제공한다고 할 수 없기 때문에 본 논문에서 제안한 DSEL은 모 언어의 표현력이 가지는 결함을 공유할 수 밖에 없다. 그러므로 사건 처리나 기하학적 자료처리를 논리형 프로그래밍 패러다임으로 설계한다면 보다 다양한 패러다임을 제공하는 언어를 모 언어로 채택하거나 Haskell 언어에 부족한 패러다임을 표현하는 프로그래밍 기능을 추가하는 것이 보다 바람직한 방법이다.

참고문헌

[1] J. Rossignac, Theoretical Foundations and Practical Algorithms for 3D Modeling, Graphics, and Animation, http://www.gvu.gatech.edu/~jarek/courses/6491/, 2003

[2] P. Hudak, S. Peyton Jones, and P. Wadler (editors, Report on the Programming Language Haskell, A Non-strict Purely Functional Language (Version 1.2), ACM SIGPLAN Notices, 27(5), 1992

[3] R.W. Floyd, The Paradigms of Programming, CACM, 22(8), pp.455-460, 1979

[4] S. Kamin, Research on domain-specific embedded languages and program generators, Electronic Notes in Theoretical Computer Science, Elsevier Press, 1998

[5] T. Budd, Multiparadigm Programming in Leda, Addison-Wesley, 1994

[6] T. Shinji, Y. Kunitomo and Y. Ohta, Solid Model in Geometric Modeling System : HICAD, ACM IEEE Design Automation Conference Proceedings, pp.360-366, 1983

[7] P. Hudak, Conception, Evolution, and Application of Functional Programming Languages, ACM Computing Surveys, 21(3), pp.359-411, 1989

[8] P. Hudak, The Promise of Domain Specific Languages, http://www.cs.yale.edu/homes/hudak-paul/hudak-dir/dsl/, 1997

[9] Zave, Pamela, A compositional Approach to Multiparadigm Programming, IEEE Software, 1989