한국CDE학회 논문집
Korean Journal of Computational Design and Engineering

Vol. 21, No. 4, pp. 453-461. December 2016
DOI http://dx.doi.org/10.7315/CDE.2016.453

〈학술논문〉

pISSN 2508-4003
eISSN 2508-402X

건축설계과정에서 Grasshopper 프로그래밍의 효율적 접근에 관한 연구

김민석†
부경대학교 건축학과

A Study on Efficient Approaches for Grasshopper Programming in Architectural Design Process

Minseok Kim†
Dept. of Architecture, Pukyong Nat'l Univ.

Received 7 September 2016; received in revised form 21 October 2016; accepted 23 October 2016

ABSTRACT

The trend of using Grasshopper with Rhino3D actively in architectural design process is recently spreading around the world. Well-known architects and designers such as Zaha Hadid, Patrik Schmacher is famous for using Grasshopper as their main design tool. As a tool for so-called 'Parametric Design', Grasshopper is receiving much attention all over the world. Grasshopper as a visual programming language has an advantage that designers and non-professionals of computer can easily learn it and use it to their works. However, those designers tend to make inefficient approaches with Grasshopper compared to computer programming professionals. Meanwhile, the difference between other programming languages and Grasshopper leads to the need of different approaches from other programming languages. This study aims to propose desired approaches of Grasshopper programming or scripting to be able to break through the inefficient approaches that designer is likely to make, by examining the characteristics of Grasshopper and exploring the appropriate programming approaches for Grasshopper.

Key Words: Architectural Design Computation, Dataflow Programming, Grasshopper, Parametric Design, Visual Programming Language

1. 서 론

근래 Rhino와 함께 Grasshopper가 건축설계 과정에서 널리 활용되고 있고, 그러한 흐름이 점차 확산되는 추세에 있다. Zaha Hadid, Patrik Schmacher 등과 같이 저명한 건축가, 디자이너들이 Grasshopper를 건축설계에 적극적으로 활용하는 것으로 잘 알려져 있다. 이른바 파라매트릭 디자인 방법론의 주요 도구로서 전세계적으로 각광받고 있다. Grasshopper는 시각적 프로그래밍 언어라는 특성 때문에 컴퓨터 비전공자들과 일반 디자이너들도 어렵지 않게 배우고 활용이 가능하다는 장점을 지닌다. 그런데, 이들 일반 디자이너들은 정식으로 컴퓨터 프로그래밍을 배우지 않았기 때문에 프로그래밍 언어의 일종인 Grasshopper를

†Corresponding Author, lado7595@gmail.com

활용하는 데에서 프로그래밍 전문가에 비해 비효율적인 접근을 보이는 경우가 많다. 한편, Grasshopper가 지니는 일반 프로그래밍 언어와의 차이로 인하여 컴퓨팅 분야에서의 일반적인 프로그래밍 방식과는 다소 다른 접근이 요구되기도 한다. 이에, 본 연구에서는 Grasshopper의 특성을 파악하고 그에 적절한 프로그래밍 또는 스크립팅 접근법을 모색함으로써, Grasshopper 정의 작성에 있어서 일반 건축 디자이너들이 빠지기 쉬운 비효율적 접근을 극복할 수 있는 대안을 제시하고자 한다.

이를 위해 본 연구에서는 다음과 같은 절차로 연구를 수행하고자 한다. 첫째, Grasshopper의 특징 및 타 프로그래밍 언어와의 차이점, 그리고 바람직한 Grasshopper 정의의 조건을 고찰한다. 둘째, 특정 건축모델링 사례의 구현을 위한 다양한 Grasshopper 정의들을 제안한다. 셋째, 제안된 Grasshopper 정의들을 비교분석하여 효율적인 Grasshopper 정의를 모색한다. 여기서 Grasshopper 정의(definition)란 Grasshopper 프로그래밍 작업을 통해 만들어진 일종의 소스코드 또는 스크립트를 지칭한다.

2. Grasshopper 프로그래밍의 특징

2.1 Grasshopper의 특징

Grasshopper는 NURBS 기반 3D 모델링 소프트웨어인 Rhino 3D에서 구동되는 Plug-in의 일종으로, Rhino 3D에서 추가 확장 기능을 개발하기 위한 개발 도구이자 개발 환경이다. 추가 확장 기능이란 MS 오피스 제품군 상에서 구동되는 매크로(macro)와 같이 모(母) 소프트웨어에서 기본적으로는 제공되지 않고 사용자가 필요에 따라 직접 개발하여 사용하는 기능을 말한다.

Rhino3D는 Grasshopper 이외에도 RhinoScript와 같은 추가 확장 기능 개발환경을 제공한다. 이 둘의 차이점으로는 RhinoScript가 VB의 문자기반(text-based) 프로그래밍 언어를 활용하는 환경이라면, Grasshopper는 시각적 프로그래밍 언어(Visual/Graphic Programming Language)를 활용하는 환경이다. 여기서 시각적 프로그래밍 언어란, C/C++나 Java, VB와 같은 문자기반 프로그래밍 언어와는 달리 상자나 연결선, 아이콘, 화살표 등과 같은 시각적 기호들을 일정 규칙에 따라 배열하고 연결함으로써 프로그래밍 작업을 수행하는 방식의 언어를 말한다.[5]

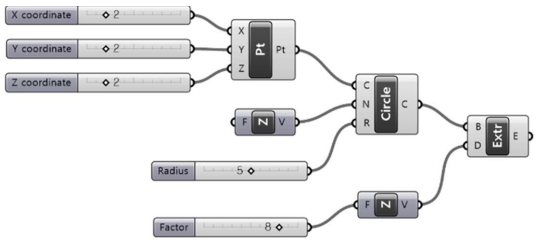


Fig. 1 Grasshopper definition for making a cylinder

Grasshopper 정의는 컴퍼넌트(component)라 불리는 상자들과 그들을 연결하는 선인 와이어(wire)들로 구성된다(Fig. 1). 컴퍼넌트는 특정의 작업(operation)을 수행하는 일종의 하위 함수이고, 와이어는 컴퍼넌트들 간의 자료의 흐름을 나타낸다. 자료는 와이어를 통해 왼쪽에서 오른쪽으로 흐르고, 컴퍼넌트는 왼쪽단자(들)에 연결된 와이어(들)로부터 자료를 입력받아 특정 작업을 수행한 후 그 결과 자료를 오른쪽단자(들)에 연결된 와이어(들)로 출력한다. ‘Pt’, ‘Circle’, ‘Extr’의 컴퍼넌트들이 각각 점을 생성하고, 이 점을 중심으로 원을 생성하며, 이 원을 위로 돌출시켜(extrude) 원기둥을 생성하는 작업을 한다.

문자기반 프로그래밍언어가 순차적(sequential) 또는 제어흐름(control flow) 방식으로 연산 순서가 정해지는 반면, Grasshopper는 자료의 흐름에 따라 연산 순서가 정해지는 특징을 가진다. 이러한 프로그래밍 패러다임을 자료흐름(dataflow) 패러다임이라 하는데, Grasshopper를 비롯하여 MVPL(Microsoft Visual Programming Language), LabVIEW와 같은 대부분의 시각적 프로그래밍 언어들이 자료흐름 패러다임에 기반하고 있다.[1] 또한 이들 자료흐름 프로그래밍 언어들은 알고리즘이 단순한 경우 소스코드가 매우 간명하게 표현되

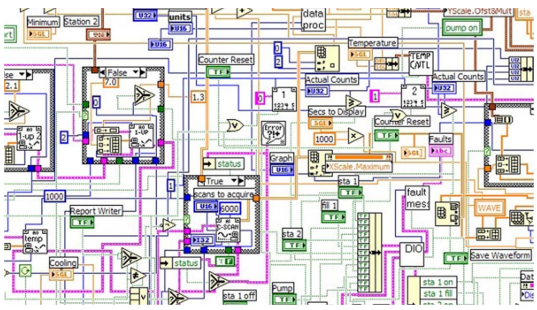


Fig. 2 Complicated Block Diagram of LabVIEW[6]

는 반면, 알고리즘이 복잡해짐에 따라 소스코드가 심하게 혼잡해지는 단점이 있다.[6]

Grasshopper는 문자기반 프로그래밍 언어들은 물론 LabVIEW와 같은 시각적 프로그래밍 언어들과도 큰 차이점을 지닌다. Grasshopper에는 C/C++의 'If', 'Switch'와 같은 분기문에 해당하는 작업구문이 제공되지 않는다. Grasshopper에서 분기문을 구현하기 위해서는 C#, VB과 같은 문자기반 스크립팅 방식을 차용해야 한다. 물론 이 경우 온전한 시각적 프로그래밍이라고 보기 어렵게 된다. 또한 Grasshopper에는 C/C++의 'For', 'While'과 같은 반복문의 작업구문 역시 기본적으로 제공되지 않는다. Grasshopper에서는 대신에 리스트, 트리와 같은 Grasshopper의 자료구조를 적절히 활용함으로써 반복문과 유사한 작업을 수행하는 것은 가능하다. 컴퍼넌트의 자료 전달 방식에서도 Grasshopper에서는 값호출(Call by value) 방식만이 사용되고 참조호출(Call by reference) 방식은 사용되지 않는다. 따라서 컴퍼넌트 생성을 남발할 경우 컴퓨터의 메모리 자원을 낭비하게 될 수 있다.

2.2 바람직한 Grasshopper 정의

이처럼 Grasshopper가 여타 프로그래밍 언어와 큰 차이점을 지니고 있으므로, Grasshopper만의 특징을 잘 반영하는 적절한 Grasshopper 정의의 요건을 정리할 필요가 있다. 건축설계 및 여타 디자인 전문가들과 같이 컴퓨터 프로그래밍 분야의 비전문가들의 Grasshopper 활용이 늘어나고 있는데, 컴퍼넌트의 과도한 생성과 얽히고 설킨 와이어 연결로 타인은 물론 작성자 본인까지도 해석이 불가능한 수준에 이르고 있다. 이 시점에서 바람직하고 효율적인 Grasshopper 정의의 요건 및 작성법에 대한 고찰이 절실하다.

Grasshopper 역시 프로그래밍 언어의 일종이므로 일반적으로 말하는 좋은 프로그램 코드의 요건부터 살펴보는 것이 유의미할 것이다. 좋은 프로

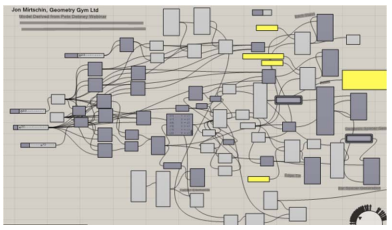


Fig. 3 Example of complicated grasshopper definition[7]

Table 1 Requisites for a good program code[2]

Item	Description
Functionality	Operating well to serve its purpose
Readability	Easy to read with much comments
Manageability	Good for debugging, testing
Reusability	Extendibility, Modularity
Simplicity	Simple and clear
Performance	Speed, Reliability, Stability

그램 코드의 요건으로 기능 적합성, 가독성, 관리용이성, 재사용성(확장성), 단순성(간결성), 성능(속도 및 안정성) 등을 들 수 있다.

Grasshopper 만의 특성에 따른 요건들도 있다. 알고리즘이 복잡할수록 Grasshopper 정의의 혼잡도가 급격히 증가하므로, 와이어들끼리 겹치는 것을 최소화하고 그룹핑 기능을 활용하여 각 부분을 모듈화함으로써 전체적으로 명료성을 높이는 방식이 요구된다. 또한 'Call by reference' 방식을 사용하지 않으므로 컴퍼넌트의 남발을 자제하고, 특히 규모가 큰 리스트나 트리 구조의 자료를 다루는 컴퍼넌트들에 대해서는 더욱 주의를 요한다. 반복구문을 구현하는 경우에도 여타 프로그래밍 언어에서의 접근방식이 불가능하므로 리스트나 트리 등의 자료구조를 효과적으로 구성하고 활용하는 것이 요구된다.

Grasshopper가 건축설계 분야에서 주목 받으면서 건축 이론가들은 Grasshopper의 알고리즘적 특성에 주목하였다. Grasshopper 정의로서 구현된 디자인 프로세스가 '함수'를 표상하고, '함수'에 입력되는 변수(parameter)를 달리하면 전혀 새로운 디자인 결과물이 출력된다. '파라메트릭 디자인(Parametric Design)' 또는 '알고리즈믹 디자인(Algorithmic Design)'이 이러한 디자인 개념을 일컫는 용어이다. 파라메트릭 디자인에서는 선언된 변수를 적절히 활용하고 변수의 의미를 사용자가 알 수 있게 하는 것이 중요하다. 따라서 파라메트릭 디자인에 충실한 것 역시 바람직한 Grasshopper 정의의 요건 중 하나가 된다.

3. Grasshopper 프로그래밍 접근법 분석

이제 Grasshopper 프로그래밍, 즉 Grasshopper 정의 작성의 실제 사례들을 비교 분석함으로써 보다 효율적이고 바람직한 Grasshopper 정의 작성

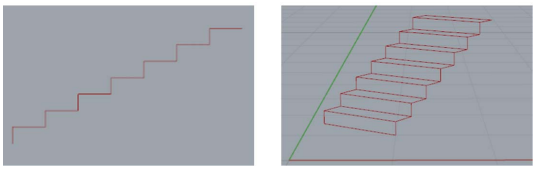


Fig. 4 Two steps for stair modelling

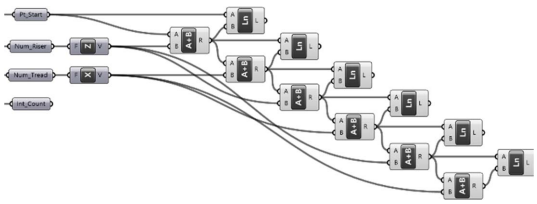


Fig. 6 Grasshopper definition for approach alt. 1

방안을 모색하도록 한다. 이를 위하여 건축설계시 자주 등장하는 계단 모델링을 사례로 삼도록 한다. 또한 명료한 비교분석을 위하여 계단 모델링 과정을 (1) XZ 평면 상의 2차원 계단 단면 모델링, (2) 임의의 평면 상의 3차원 계단 모델링의 2단계로 구분하여 진행하도록 한다.

먼저 파라메트릭 디자인 개념에 충실할 수 있도록 계단 모델링의 필요 변수들을 설정하도록 한다. 디딤판 폭, 챌판 높이, 계단 단수, 계단 폭, 계단의 시작점과 같은 변수들을 Grasshopper의 고정 파라메터 컴퍼넌트로 생성한다. 디딤판 폭, 챌판 높이, 계단 폭은 실수형의 고정 파라메터 컴퍼넌트로, 계단 단수는 정수형의 고정 파라마터 컴퍼넌트로, 그리고 계단의 시작점은 점(point) 형의 고정 파라메터 컴퍼넌트로 생성한다. 또한 각각의 고정 파라메터 컴퍼넌트들에는 해당 변수의 역할을 나타낼 수 있는 이름을 부여하여 가독성 및 재사용성을 제고시킨다.

3.1 접근법별 구현

(1) 대안 1

대안 1은 계단 단면의 개별 선분들을 아래에서부터 하나씩 순차적으로 작도하는 방식이다. 이 접근법은 계단 단면을 손으로 직접 그리는 과정을 알고리즘화한 것으로, 일반적으로 생각할 수 있는 가장 단순하고 직관적인 방식이다.

대안 1 접근법에 대한 Grasshopper 정의는 그림 6과 같다. 점①-점②에 대한 벡터를 Z축 방향 단위벡터에 'Num_Riser'를 곱하여 생성하고, 점②-점③에 대한 벡터를 X축 방향 단위벡터에 'Num_Tread'를 곱하여 생성하였다. 'Pt_Start'에 점①-점②에 대한 벡터를 더하여 점②를 구한 후 점①과 점②를 잇는 선분을 그려주었고, 점②에 점②-점③에 대한 벡터를 더하여 점③을 구한 후 점②와 점③을 잇는 선분을 그려주었다. 이것이 계단 한 단의 모델링 과정이고, 이를 7번 반복하면 7단의 계단 단면을 그릴 수 있다.

Fig. 6의 Grasshopper 정의는 3개 단만을 그리는 것이고, 7단을 그리는 Grasshopper 정의는 물론 이보다 훨씬 복잡하게 된다. 또한 그림에서 확인할 수 있는 것처럼 'Int_Count'의 계단 단수 변수는 활용되지 않고 있다. 이렇게 볼 때 대안 1 접근법은 Grasshopper 정의가 복잡해지고 추후 확장 및 재사용 면에서 취약하며, 파라메트릭 디자인에 충실하지 못한 방식임을 알 수 있다. 이는 반복작업을 위한 알고리즘을 고려하지 않고 단순히 1차원적으로 알고리즘을 구현한 데에 기인한다. 이와 같은 근본적 취약성에 따라 대안 1에 대한 3차원 계단 모델링 작업 역시 극도로 복잡해지고 효율성이 떨어지게 된다. 이런 이유로 본 연구에서는 대안 1에 대한 3차원 계단 모델링 작업은 생략하도록 한다.

(2) 대안 2

대안 2는 Rhino 모델링 작업과정을 모방함으로써 대안 1에서처럼 직관성에 충실한 접근법이다. 그러나 대안 2는 대안 1과는 달리 리스트 자료구조를 적용하여 반복작업을 효율적으로 구현하는 방식을 취한다. 즉, 여기에서는 계단 한 단을 먼저 모델링하고, 그것을 쌓아가는 식으로 필요한 단 수만큼 반복하여 복사(copy)한다. 계단 한 단을 이동 복사하는 작업이 여기에서의 반복 작업이다. 이를

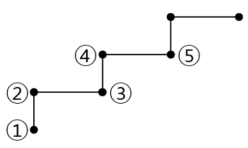


Fig. 5 Modeling process of approach alt. 1

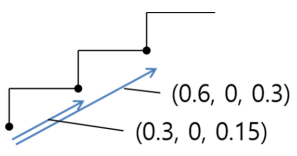


Fig. 7 Modeling process of approach alt. 2

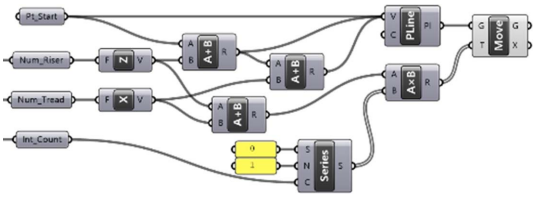


Fig. 8 Grasshopper definition for approach alt. 2

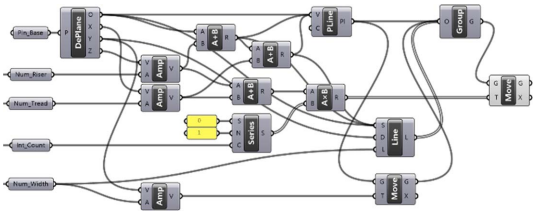


Fig. 9 Grasshopper definition for approach alt. 2 (2)

위해 이동복사에 필요한 벡터 리스트를 생성하고, 이를 활용하여 이동복사 반복작업을 구현한다.

계단 한 단을 모델링하는 작업은 대안 1에서의 초기 작업과 거의 유사하다. 다만 여기에서는 이후의 이동복사 작업의 편의를 위하여 디딤판 선분과 챌판 선분을 'PolyLine' 컴퍼넌트를 활용하여 하나의 폴리라인으로 통합하도록 한다. 다음으로 이동복사를 위한 벡터 리스트를 작성한다. 첫번째 계단 한 단을 이동복사하여 두번째 계단을 생성하기 위해서는 x성분이 디딤판 폭, z성분이 챌판 높이인 벡터가 필요하고, 이 벡터는 X축 방향 단위벡터에 디딤판 폭을 곱한 벡터와 Z축 방향 단위벡터에 챌판 높이를 곱한 벡터를 더함으로써 구할 수 있다. 디딤판 폭이 0.3, 챌판 높이가 0.15일 경우 (0.3, 0. 0.15)가 된다. 이 벡터에 정수 리스트를 곱하면 이동복사를 위한 벡터 리스트가 생성된다. 최종적으로 계단 한 단의 폴리라인과 벡터 리스트를 'Move' 컴퍼넌트에 입력함으로써 작업이 완료된다.

이 접근법은 대안 1에 비해 컴퍼넌트 수가 현저히 적고, 그 구조 또한 단순 명료하고 이해하기 쉽다. 리스트 자료구조를 적절히 활용하여 반복작업이 효율적으로 구현되었고, 'Int_Count' 변수가 제대로 활용되어 파라메트릭 디자인에도 충실해졌다.

이를 발전시켜, 임의의 평면 상에 3차원 계단을 모델링하는 Grasshopper 정의를 작성하도록 한다. 이를 위한 접근방식은, 먼저 임의의 평면 상에 3차원 계단 한 단을 모델링하고, 이를 필요한 단수만큼 이동복사한다. 이동복사 방식은 위의 작업에서와 정확히 동일하므로, 2차원 계단 한 단의 단면 모델링을 3차원 계단 한 단의 모델링으로 바꿔주는 것이 관건이다.

위의 Grasshopper 정의에 변수명 'Pln_Base'의 고정 파라메터 컴퍼넌트를 추가한다. 이는 임의의 평면을 입력받기 위한 변수이다. 이 변수의 평면 자료로부터 원점 및 X, Y, Z축 방향벡터를 추출하기 위해 'Deconstruct Plane' 컴퍼넌트를 활용한다. 이로써 고정된 XZ 평면이 아닌 사용자가 임의로 입력한 평면 상에 계단 모델링이 가능해진다.

다음으로 이미 모델링된 2차원 계단 한 단을 Y축 방향으로 계단폭만큼 이동복사하고, 위에서 모델링한 계단 한 단의 각 꼭지점에서 계단폭 길이만큼 Y축 방향의 선분을 생성함으로써 3차원 계단 한 단을 모델링한다. 이동복사 작업은 'Move' 컴퍼넌트, 선분 생성은 'Line SDL' 컴퍼넌트를 활용한다. 마지막으로 위에서 모델링한 계단 단면과 그것을 이동복사한 것, 그리고 그 둘을 잇는 선분들을 'Group' 컴퍼넌트로 묶은 후 이를 'Move' 컴퍼넌트에 위에서의 벡터 리스트와 함께 입력해주면 작업이 완료된다.

이처럼 대안 2를 3차원 계단으로 확장시킨 결과, Grasshopper 정의가 다소 복잡해지고 명료성이 저하되었다. 대안 2가 대안 1에 비해 간명하고 효율적인 것은 사실이나, 대안 1에서의 과도한 직관성의 잔재가 Grasshopper 정의의 확장성 또는 재사용성을 약화시킨 것으로 보인다.

(3) 대안 3

대안 3은 대안 1, 2에서의 직관적 모델링과는 달리 주도면밀한 좌표계산을 통하여 일괄적으로 모

Table 2 Vertices coordinates of stair section for alt. 2

X	Y	Z	X	Y	Z
0.0	0.0	0.0	1.2	0.0	0.6
0.0	0.0	0.15	1.2	0.0	0.75
0.3	0.0	0.15	1.5	0.0	0.75
0.3	0.0	0.3	1.5	0.0	0.9
0.6	0.0	0.3	1.8	0.0	0.9
0.6	0.0	0.45	1.8	0.0	1.05
0.9	0.0	0.45	2.1	0.0	1.05
0.9	0.0	0.6			

델링하는 방식이다. 대안 3에서는 계단 단면을 이루는 모든 꼭지점들의 좌표를 계산하여 생성한 후 꼭지점들을 잇는 단면선을 일괄 생성한다. 따라서 이 접근법에서는 각각의 꼭지점 좌표에 대한 세세한 계산 과정이 핵심이다.

Table 2는 디딤판 폭 0.3, 챌판 높이 0.15, 계단 단수 7단, 계단 시작점은 원점인 경우 계단 단면을 이루는 꼭지점들의 좌표를 정리한 것이다. 계단 단면이 XZ 평면 상에 작성되므로 Y좌표는 모두 0이다. 이들 좌표수치들을 각각 X좌표 리스트와 Z좌표 리스트로 만들어 'Construct Point' 컴퍼넌트에 입력해주면 각각의 꼭지점들이 생성되고, 이들 꼭지점들을 'PolyLine' 컴퍼넌트에 넣어주면 계단 단면 모델링이 완료된다.

이제 관건은 X좌표 리스트와 Z좌표 리스트의 형성이다. Table 2를 보면 X좌표와 Z좌표 모두 같은 수가 한번씩 반복하여 등장한다는 규칙이 있다. 단, X좌표의 경우 마지막 '2.1'이, Z좌표의 경우 처음의 '0.0'이 한번만 등장한다. 먼저 'Series' 컴퍼넌트를 활용하여 '0.0, 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1'의 X좌표 리스트와 '0.0, 0.15, 0.3, 0.45, 0.6, 0.75, 0.9, 1.05'의 Z좌표 리스트를 생성한다. 그런 다음 각 리스트의 항목들을 'Stack' 컴퍼넌트를 활용하여 한번씩 반복해준다. 마지막으로 X좌표 리스트에서는 끝 항목인 '2.1'을, Z좌표 리스트에서는 첫 항목인 '0.0'을 삭제해준다. 'Cull Index'와 'Shift' 컴퍼넌트가 그 일을 해준다. 이렇게 생성된 X좌표와 Z좌표의 리스트를 위에서 언급한 'Construct Point' 컴퍼넌트에 입력해주면 Grasshopper 정의가 완성된다.

이 접근법은 대안1에 비해 적은 컴퍼넌트 수를 보이고, 대안 2에 비해서도 비슷한 수준을 보인다. 구조 면에서도 대안 2에 비해 크게 복잡하다고 볼 수는 없으나, X좌표 리스트와 Z좌표 리스트를 가공하는 과정이 간명하지 않고 다소 난해해 보인다.

이 결과를 발전시켜, 임의의 평면 상에 3차원 계단을 모델링하는 Grasshopper 정의를 작성하도록

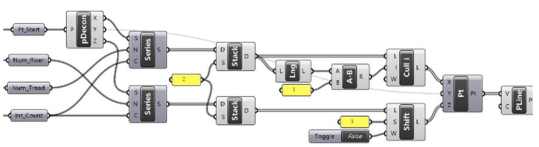


Fig. 10 Grasshopper definition for approach alt. 3

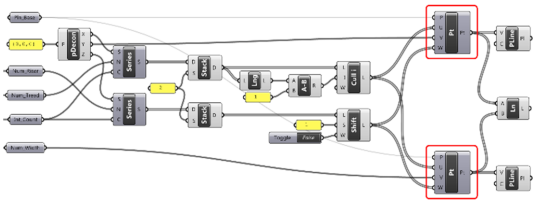


Fig. 11 Grasshopper definition for approach alt. 3 (2)

한다. 접근방식은, 이미 모델링된 계단 단면으로부터 계단 폭만큼 떨어진 위치에 동일하게 한번 더 계단 단면을 모델링하고 각각의 꼭지점들을 잇는 선분을 그리는 방식이다.

이전 단계의 Grasshopper 정의에서 'Construct Point' 컴퍼넌트가 'Point Oriented' 컴퍼넌트로 대체되었다. 이는 계단 모델링의 기준평면을 XZ평면에서 임의의 평면으로, 즉 절대좌표에서 상대좌표로 전환시키기 위한 것이다. 'Point Oriented' 컴퍼넌트를 아래쪽에 하나 더 생성하고 여기에 계단 폭 수치를 입력함으로써 또 하나의 계단 단면을 모델링하였고, 두 개의 'Point Oriented' 컴퍼넌트에서 출력된 꼭지점들을 각각 잇는 선분을 'Line' 컴퍼넌트로 그려주었다.

대안 2에서와는 다르게 3차원 계단 모델링의 Grasshopper 정의가 명료하게 잘 정리되었고, 그 결과도 가독성 면에서 뛰어나다. 따라서 대안 3은 확장 재사용 면에서 대안 2에 비해 상당히 효과적이라 할 수 있다. 그러나 이 접근법은 필요한 모든 좌표를 직접 계산하여 그 결과를 모델링에 반영하는 방식이므로 추후 확장성에서 취약할 수밖에 없다. 예를 들어 곡선계단의 모델링에 적용할 경우 각 꼭지점의 좌표를 일일이 계산하는 일이 Grasshopper 정의 작성자에게 상당한 부담으로 작용하게 되고, 작성 상의 실수로 이어질 가능성이 커진다.

(4) 대안 4

대안 4는 대안 1, 2의 직관성과 단순성에 대안 3의 주도면밀성을 가미하는 방식이다. 이렇게 함으로써 대안 3에 비해 덜 난해하고 직관적이면서 직관성에 치우친 대안 1, 2의 취약점을 보완할 수 있다.

대안 4의 기본방향은 계단 한 단을 그리는 방식으로 여러 단을 그리는 것이다. Grasshopper의 모든 컴퍼넌트는 리스트 구조의 자료를 입력받았

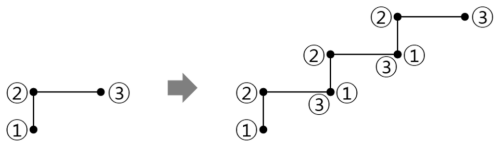


Fig. 12 Modeling process of approach alt. 4

을 때 해당 리스트의 모든 항목(item)에 대해 자신의 작업(operation)을 반복한다. Fig. 12의 왼쪽과 같이 ①, ②, ③의 3개의 꼭지점을 입력받아 계단 한 단을 작성하는 컴퍼넌트가 있다면, 거기에 ①의 점들, ②의 점들, ③의 점들을 리스트로 만들어 입력함으로써 여러 단을 반복적으로 그릴 수 있을 것이다.

계단 한 단을 그리는 방식은 대안 1, 2에서와 동일하다. 관건은 ①의 점들, ②의 점들, ③의 점들에 대한 리스트를 만드는 것이다. ①의 점들 중 가장 아래에 있는 점은 계단의 시작점이고, 이 점에 챌판 높이 크기의 Z축 방향 벡터와 디딤판 폭 크기의 X축 방향 벡터를 한 번씩 더하면 두 번째 ①의 점이 만들어진다. 같은 방식으로 두 벡터를 한 번씩 더해주면 세 번째 ①의 점이 만들어지고, 이를 계단 단 수만큼 반복하면 필요한 ①의 점들을 모두 생성할 수 있다. 이렇게 만들어진 ①의 점들을 이용하면 손쉽게 ②의 점들과 ③의 점들을 만들 수 있다. 즉, ①의 점들에 챌판 높이 크기의 Z축 방향 벡터를 더하면 ②의 점들이 생성되고, ②의 점들에 디딤판 폭 크기의 X축 방향 벡터를 더하면 ③의 점들이 생성된다.

이를 위해 챌판 높이 크기의 Z축 방향 벡터에 '0, 1, 2, …, 6'을 각각 곱한 벡터들을 리스트로 구성하고, 디딤판 폭 크기의 X축 방향 벡터에 '0, 1, 2, …, 6'을 각각 곱한 벡터들을 리스트로 구성한다. 전자를 챌판 벡터 리스트, 후자를 디딤판 벡터 리스트라 할 때, 계단 시작점에 챌판 벡터 리스트와 디딤판 벡터 리스트를 더하면 ①의 점들이 리스트로서 생성된다. Grasshopper에서 리스트 자료구조가 처리되는 방식을 적절히 활용한 것이다. 마

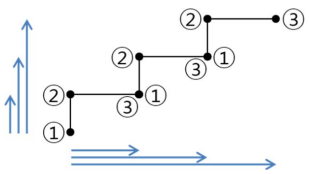


Fig. 13 Making vertices group ①, ②, and ③

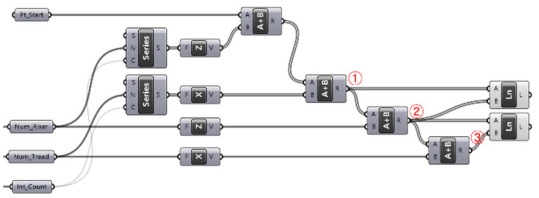


Fig. 14 Grasshopper definition for approach alt. 4

찬가지로 ①의 점들 리스트에 챌판 벡터 리스트를 더하여 ②의 점들 리스트를 생성할 수 있고, ②의 점들 리스트에 디딤판 벡터 리스트를 더하여 ③의 점들 리스트를 생성할 수 있다.

이 접근법은 대안 2, 3과 비교하여 유사한 정도의 컴퍼넌트 수를 보이고, 대안 3에서의 다소 까다로운 리스트 처리 과정도 없으며, 무엇보다 직관성과 명료성, 가독성 면에서 뛰어나다. Fig. 14의 오른쪽 부분은 대안 1, 2에서의 계단 한 단 모델링 부분과 거의 유사하고, 챌판 벡터 리스트와 디딤판 벡터 리스트를 생성하는 왼쪽 부분에서도 대안 3에서 보였던 다소 까다로운 리스트 처리 부분이 없이 매우 간명하다.

이 결과를 발전시켜, 임의의 평면 상에 3차원 계단을 모델링하는 Grasshopper 정의를 작성하도록 한다. 접근방식은 대안 3에서와 동일하다. ①의 점들, ②의 점들, ③의 점들 리스트에 계단폭 크기의 Y축 방향 벡터를 더하여 ①'의 점들, ②'의 점들, ③'의 점들 리스트들을 생성하고, 이들 리스트들로부터 또 하나의 계단 단면을 그린다. 마지막으로 ①-①', ②-②', ③-③'을 각각 잇는 선분들을 그려준다.

물론 대안 3에서보다 컴퍼넌트 수가 많아지긴 했으나, 3차원 계단으로의 확장 작업에서도 2차원 계단 작업에서처럼 명쾌하게 작업이 가능하였다. 이는 가독성과 명료성이 좋은 코드의 특징으로, 추후 재사용과 기능확장, 유지보수가 용이하다는 장

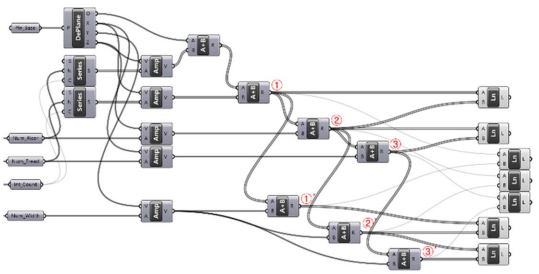


Fig. 15 Grasshopper definition for approach alt. 4 (2)

점을 가진다.

3.2 최적 대안의 선정

지금까지 계단 모델링에 대한 Grasshopper 정의를 4개 대안에 걸쳐 살펴보았다. 대안 1은 직관적 접근으로 인하여 정의가 매우 복잡하고 파라메트릭 디자인에 충실하지 못한 단점이 있었다. 대안 2는 대안 1의 직관성에 효율적 반복작업을 도입하여 대안 1에 비해 명료한 구조를 지니나, 확장성과 재사용성에서는 여전히 취약함을 보였다. 대안 3은 직관적 접근이 아닌 주도면밀한 접근을 보임으로써 역시 명료한 구조를 지니나, 까다로운 리스트처리와 추후 확장성에서의 취약성을 보였다. 대안 4는 직관성과 주도면밀성을 결합하고 반복작업의 효율성을 추구하여 명료성, 가독성은 물론 추가작업이 쉬워 재사용성, 기능확장, 유지보수성 등이 뛰어났다.

대안 4의 접근법은 Grasshopper가 반복작업에 대처하는 고유한 방식을 명시적으로 드러낸다. Grasshopper는 반복구문이 없는 대신 리스트와 트리의 자료구조를 활용하여 반복작업을 처리하는데, 이러한 방식은 수열의 일반항 표현식에 비유할 수 있다. 일반항 $an = ar^{n-1}$에 n 대신 1,2,3,…의 수치들을 입력하면 초항 a, 공비 r의 등비수열이 만들어지는 것처럼, Grasshopper에서도 등비수열의 Grasshopper 정의에 1,2,3,…의 수치 리스트를 변수로 입력하면 반복작업에 의해 등비수열의 리스트가 결과로 도출된다. 대안 4의 접근법은 이러한 특징을 십분 활용하여, 계단 한 단을 모델링하는 Grasshopper 정의를 작성한 후 거기에 변수로 3개 꼭지점의 리스트를 입력하도록 하였다. 또한 3개 꼭지점 리스트를 생성하는 부분에서도 챌판 벡터와 디딤판 벡터에 0,1,2,…의 수치 리스트를 곱하여 챌판 벡터 리스트와 디딤판 벡터 리스트를 수열 생성과정과 같이 만들어냈는데, 이는 정의 전체의 논리와 세부 논리가 정합한다는 면에서 부가적인 의의를 가진다. 이러한 특징들이 Grasshopper 정의의 명료성과 가독성을 높이는 요소가 된다.

4. 결 론

본 연구에서는 시각적 프로그래밍 언어이자 환경인 Grasshopper를 활용하는 효율적이고 바람직한 방향의 모색을 목적으로 하여, 바람직한 Grasshopper 정의의 조건을 고찰하고, 계단 모델링 사례를 대상으로 Grasshopper 정의 작성의 다양한 접근법을 살펴보았으며, 이 접근법들을 비교분석하여 효율적이고 바람직한 Grasshopper 정의 작성의 접근법을 살펴보았다. 그 결과, 대안 4와 같이 직관성을 잃지 않으면서도 리스트 자료구조를 적극 활용함으로써 Grasshopper만의 독특한 반복작업 구현방식에 충실하게 접근하는 방식이 Grasshopper 정의의 가독성(명료성), 단순성, 재사용성(확장성), 유지보수성 면에서 여타 방식에 비해 뛰어남을 알 수 있었다.

건축설계 실무자나 디자이너와 같은 컴퓨팅 비전문가들이 Grasshopper 작업을 할 때, 반복작업에 익숙치 않거나 또는 그를 염두에 두지 않고 직관성에 매몰되어 대안 1, 2와 같은 비효율적인 접근법을 보이는 경향이 있다. 이를 극복하기 위해서는 Grasshopper의 특성을 보다 정확히 인지하고 그에 맞는 알고리즘적 사고를 펼쳐나가는 것이 필요하다. 물론 한가지 접근법만이 옳고 나머지가 틀렸다는 식의 결론 역시 위험하다. 다양한 접근법을 고려하여 적용해보고, 가독성, 단순성, 재사용성 등의 여러 관점에서 장단점을 파악함으로써 요구 기능과 상황에 적절한 Grasshopper 정의를 작성해나가야 할 것이다.

근래 Grasshopper를 많이 활용하는 건축설계 실무자와 여타 디자이너들은 컴퓨터 활용 능력이 뛰어난 반면 프로그래밍과 같은 보다 전문적인 컴퓨팅 지식은 낮은 편이다. Grasshopper 작업 역시 프로그래밍의 일종이므로 보다 전문적인 컴퓨팅 지식과 기술을 갖추는 것이 좋으나, 이는 현실성이 다소 떨어진다. 오히려 이들 비전문가들이 보다 효과적으로 Grasshopper 작업을 할 수 있도록 환경과 지원을 제공하는 것이 필요하고, 본 연구가 그에 대한 토대가 될 수 있기를 기대한다.

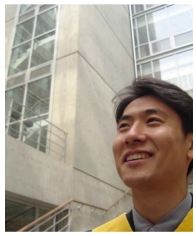
본 연구에서는 건축 모델링의 기본적 사례인 계단 모델링에 대하여 효율적인 Grasshopper 정의 작성을 살펴보았다. 그러나 건축설계실무에서는 설계대안생성과 같은 보다 고도의, 다양한 모델링에의 적용이 요구된다. 본 연구의 결과를 바탕으로 추후에는 보다 실질적인 건축설계모델링에의 Grasshopper 활용 효율화 방안을 모색하여야 할 것이다.

감사의 글

본 논문은 한국CDE학회 2016 부문연합 학술대회에 발표된 논문을 수정, 보완한 것임.

References

1. Johnston, W.M., Hanna, J.R.P. and Millar, R.J., 2004, Advances in Dataflow Programming Languages, *ACM Computing Surveys*, 36(1), pp.1-34.
2. Sommerville, I., 2004, *Software Engineering*, 7th Ed., Pearson Education Ltd.
3. Tedeschi, A., 2014, *AAD_Algorithms-Aided Design*, Le Penseur.
4. Woodbury, R., 2010, *Elements of Parametric Design*, Routledge.
5. Wikipedia, 'Visual Programming Language', https://en.wikipedia.org/wiki/Visual_programming_language
6. Johnson, P., 2012, "10 Programming Languages Driving Developers Crazy", ITWorld.
7. Jon Mirtschin's blog, 'Geometry Gym', http://geometrygym.blogspot.kr/2011/06/grasshopper-gsa-form-finding-examples.html
8. Kim, Y.M., 2012, Fundamentals of Programming with LabVIEW, INFINITY BOOKS.



김 민 석

2000년 2월 서울대학교 건축학과 공학사
2006년 2월 서울대학교 건축학과 공학석사
2010년 8월 서울대학교 건축학과 공학박사
1999년 9월~2000년 1월 대흥멀티미디어통신(주) SI개발팀 연구원
2000년 2월~2003년 9월 ㈜코스펙정보 기술연구소 전임연구원
2006년 9월~2007년 8월 LH공사 토지주택연구원 위촉연구원
2008년 6월~2008년 8월 서울대학교 건설환경종합연구소 선임연구원
2008년 9월~2010년 2월 서울대학교 건축학과 조교
2010년 3월~2011년 3월 서울대학교 건설환경종합연구소 선임연구원
2011년 4월~2012년 8월 삼우종합건축사사무소 기술연구소 부실장
2012년 9월~현재 부경대학교 건축학과 조교수
2016년 7월~현재 한국CDE학회 정회원
관심분야: 건축·도시 계획·설계, 건축·도시 공간분석, CAD, BIM, Digital Design