http://dx.doi.org/10.5392/JKCA.2013.13.03.119

뉴미디어 예술 작품에 적용된 알고리즘의 미학적 함의 : 라이브 코딩을 중심으로

Aesthetic Implications of the Algorithm Applied to New Media Art Works : A Focus on Live Coding

오준호
서강대학교 영상대학원

Junho Oh(junhooh@sogang.ac.kr)

요약

본 논문은 알고리즘이 물질성과 표현성을 획득할 수 있음을 라이브 코딩을 통해 연구한다. 라이브 코딩은 실시간으로 코드를 작성하면서 소리를 생성하고, 코드를 스크린에 투사하는 즉흥 음악 장르이다. 기존의 라이브 코딩 연구는 공연을 효과적으로 뒷받침할 수 있는 개발 환경에 초점을 맞추어 왔다. 그러나 본 연구는 라이브 코딩에서 주로 활용되는 ChucK, Impromptu, 라이브 코드의 시각화의 언어적 특성 분석과 "aa-cell"과 "slub"의 실제 공연 사례 분석을 통해 알고리즘 구현에 내재된 미학적 태도를 연구한다. 라이브 코딩의 미학적 태도는 대수적 태도와 기하학적 태도로 나눌 수 있다. 대수적 태도는 시간상에 순차적인 개념의 전개에 초점을 맞추고, 기하학적 태도는 개념의 구조를 공간상에 시각적 구조로 물질화하는데 중심을 둔다. 이러한 태도의 차이는 개념시와 구체시를 통해 표명된 개념과 물질 사이의 긴장 관계가 라이브 코딩에서 유사하게 반복된다는 것을 의미한다. 라이브 코딩에서 언어에 대한 입장이 개념과 물질 중에서 무엇을 강조하는가에 따라 알고리즘의 표현성이 규정된다.

■ 중심어 : | 라이브코딩 | 프로그래밍언어 | 소프트웨어연구 | 알고리즘연구 | 뉴미디어예술 |

Abstract

This paper researches the algorithm, whose materiality and expressiveness can be obtained through live coding. Live coding is an improvised genre of music that generates sounds while writing code in real time and projecting it onto a screen. Previous studies of live coding have focused on the development environment to support live coding performance effectively. However, this study examines the aesthetic attitude immanent in the realization of the algorithm through analyzing mostly used languages such as ChucK, Impromtu, and the visualization of live code and cases of "aa-cell" and "slub" performance. The aesthetic attitudes of live coding performance can be divided into algebraic and geometric attitudes. Algebraic attitudes underline the temporal development of concepts; geometric attitudes highlight the materialization of the spatial structure of concepts through image schemas. Such a difference echoes the tension between conception and materiality, which appears in both conceptual and concrete poetry. The linguistic question of whether conception or materiality is more greatly emphasized defines the expressiveness of the algorithm.

■ keyword : | Live Coding | Programming Language | Software Studies | Algorithm Studies | New Media Art |

* 이 논문은 2010년도 정부재원(교육과학기술부 인문사회연구역량강화사업비)으로 한국연구재단의 지원을 받아 연구되었음(NRF-2010-332-G00013)

접수번호 : #130117-004
접수일자 : 2013년 01월 17일
심사완료일 : 2013년 03월 06일
교신저자 : 오준호, e-mail : junhooh@sogang.ac.kr

I. 서 론

1. 연구배경

최근에 뉴미디어 연구에서 “소프트웨어 연구(Software Studies)”라는 분야가 새롭게 등장하여 관련 연구 성과가 빠르게 축적되고 있다. 그동안 소프트웨어 관련 연구는 소프트웨어 개발이나 새로운 소프트웨어의 등장과 사회현상의 관계를 설명하는데 초점을 맞추어왔다.

그러나 소프트웨어 연구는 현대 사회에서 소프트웨어가 광범위하게 사용되고 우리의 일상생활에 스며들어 있기 때문에 사회 현상을 관찰하는 것만으로는 소프트웨어를 제대로 이해할 수 없다고 본다. 소프트웨어 연구는 코드에 직접 접근해서 개발자가 소프트웨어에 부여한 철학적인 틀을 파악하고자 한다. 이를 바탕으로 해서 소프트웨어가 일상생활의 양식뿐만 아니라 인간의 사고와 행동에 가져오는 변화들을 면밀하게 설명하고자 한다. 소프트웨어 연구는 기술의 내부적 구조를 이해하고 소프트웨어를 일상생활의 맥락에서 파악하고자 하는 시도를 결합하고자하기 때문에 융합적인 연구 분야라고 할 수 있다.

매튜 풀러(Matthew Fuller)는 소프트웨어 연구가 다루는 분야를 다음과 같이 제시한다. “논리적 기능이 매우 근본적이기 때문에 대부분의 사용자에게 인식되지 않는 알고리즘, 논리 영역에서 새어 나와 일상생활의 영역으로 스며들어가는 사고와 행위의 방식들, 컴퓨팅에 구성된 가치와 미학의 판단들, 프로그래밍 자체의 하위문화와 프로그래밍의 암시적 혹은 명시적인 정치학, 또는 현실을 만들고, 명명하고, 재생산하고, 조절하며 연관시키데 작용하는 견고하게 형식화된 구성단위들이 소프트웨어 연구의 대상”이다[1].

또 다른 측면에서 소프트웨어 연구는 뉴미디어가 비물질적이라는 가정에 도전한다. 일반적으로 디지털 기술을 통해 구현된 매체는 추상화와 형식화된 정보를 통해 매개하기 때문에 인간의 지각에 감각되는 물질성을 갖지 않는다고 간주된다. 그러나 소프트웨어 연구는 소프트웨어가 시간, 공간, 기억, 발화 등에 영향을 미쳐서 인간 경험 차원에서 구체적으로 가져오는 변화들을 설명하고, 디지털 기술을 통한 매개의 물질성을 규명하고자 한다. 데이비드 베리(David M. Berry)는 매체 특정성(medium specificity)의 관점에서 매체의 변화가 어떻게 인식의 변화를 가져오는지를 생각해볼 수 있는 방법으로 디지털 매체의 구성 요소들을 검토할 것을 제안한다[2]. 매체 특정성은 클레멘트 그린버그가 추상표현주의 작품들을 이론화하면서 제안한 개념으로 매체의 물질성을 바탕으로 고유의 예술 형식을 탐색해야 한다는 요청으로 이해되어 왔다. 따라서 소프트웨어 연구는 비물질적이라고 간주되는 뉴미디어의 물질성을 규명함으로써, 뉴미디어 예술을 매체 특정성의 관점에서 고찰하고 모더니즘 예술과의 연속성과 비연속성을 설명할 수 있는 근거를 제공해준다.

본 연구는 소프트웨어 연구 방법을 라이브 코딩에 적용한다. 라이브 코딩에서 주로 활용되는 ChucK, Impromptu, 라이브 코드의 시각화의 언어적 특성 분석과 "aa-cell"과 "slub"의 실제 공연 사례 분석을 통해 알고리즘 구현에 내재된 미학적 태도를 연구하고자 한다.

2. 연구대상 및 방법

본 논문은 연구의 대상을 라이브 코딩(live coding)으로 한다. 프로그래밍을 통해 구현된 뉴미디어 예술 작품들 대부분에서 알고리즘은 작품의 기술적 구조 안에 은폐되어 작품의 외적 특성을 통해 사용된 알고리즘을 유추할 수밖에 없는 제한이 발생한다. 그러나 라이브 코딩은 알고리즘을 전면화하여 관객에게 지각 대상으로 제공하기 때문에 연구의 접근이 수월하다.

라이브 코딩은 실시간 프로그래밍을 통해서 즉흥적으로 소리와 이미지를 생성하는 퍼포먼스를 지칭하는데, 연주자나 개발자에 따라 ‘연속식 프로그래밍(on-the-fly programming)’ 혹은 ‘적시 프로그래밍(just in time programming)’이라고도 불린다. 라이브 코딩은 공연에서 중요한 시각적 요소인 연주라는 행위가 결여되어있다는 비판에 직면하여 이를 극복하기 위한 시도로서 등장했다[3]. 라이브 코딩에서 연주자는 작성 중인 코드를 프로젝터를 통해 스크린에 투사해서, 관객이 실시간으로 생성되는 소리와 이미지를 연주자의 코드 작성 행위와 동기화할 수 있도록 한다.

기존의 즉흥 음악에서는 연주자의 행위와 오브제 간의 물리적인 상호 작용으로 발생되는 소리의 관계가 관객에게 직관적으로 전달된다. 그러나 라이브 코딩에서는 추상화된 코드와 알고리즘을 통해 코드를 작성하는 연주자의 행위와 소리가 매개되기 때문에 프로그래밍 언어를 이해하지 못 하는 관객은 감상이 어렵다는 문제가 발생한다. 따라서 라이브 코딩 연주자들은 알고리즘과 코드를 감상의 대상으로 만들기 위해서 노력한다. 이를 다시 말하면 일반적으로 추상적 논리와 동일하게 간주되는 알고리즘을 인간이 지각하고 감상할 수 있는 물질적 대상으로 전환하는 것이라고 볼 수 있다. 소프트웨어 연구가 디지털 매체의 구성 요소들을 매체 특정성의 관점에서 고찰하여 개별 요소들의 물질성을 연구하는 것과 동일한 맥락에서 라이브 코딩이라는 장르를 통해 코드로 작성된 알고리즘의 물질성과 미학적 함의를 설명하려는 시도가 가능하다.

이를 위해 본 논문은 다음과 같은 순서로 연구를 전개한다. 우선 알고리즘의 물질성을 탐색한 기존 연구들을 간단하게 정리하고, 라이브 코딩을 통한 연구가 기존의 “알고리즘 연구(Algorithm Studies)”와 갖는 차별성을 제시한다. 그리고 라이브 코딩을 위한 프로그래밍 언어 설계의 특성을 살펴보고 이를 바탕으로 현재 라이브 코딩에 활발하게 사용되면서 서로 다른 언어적인 특성을 보여주는 ChucK, Impromptu, 언어의 의미를 기하학적 배치로 시각화하는 언어(이후 라이브 코드의 시각화)를 비교 정리한다. 그리고 “더블 에이-셀(aa-cell)”과 “슬럽(slub)”이라는 라이브 코딩 연주 팀의 공연을 분석하여 알고리즘 구현에 드러나는 미학적 태도를 구체시와 개념시와 비교하여 연구한다.

II. 알고리즘 연구

1. 알고리즘 연구의 선행연구 분석

디지털 기술 구성요소를 대상으로 물질성을 규명하는 소프트웨어 연구에서 알고리즘 연구는 상대적으로 적은 편이다. 알고리즘이 프로그래밍 언어와 프로그램을 실행하는 기계에 독립적이라면 알고리즘은 물질적 토대를 갖지 않는 순수 추상과 동일하게 되기 때문이다.

따라서 알고리즘을 형식 논리가 아닌 측면에서 설명할 필요가 생긴다. 앤드류 고피(Andrew Goffey)는 로버트 코왈스키(Robert Kowalski)가 “알고리즘=논리+통제”라고 간명하게 표현한데서 알고리즘의 물질성을 설명할 수 있는 가능성을 찾는다. 튜링 기계와 같은 보편 연산 기계(universal computing machine)에서 알고리즘은 문제를 해결하는 효과적인 절차로 이해되어 기계에 입력되어야 하는 명령들의 집합과 동일하고 형식 논리 추론과 같다. 그러나 튜링 기계는 실재하는 기계가 아니라 무한한 테이프를 가정하는 가상적인 기계이다. 실제 컴퓨터에서는 실용적인 관점에서 특정 목적을 수행하기 위해 프로그램이 작성되며, 알고리즘을 구성한다는 것은 특정 과제를 달성하기 위해 일련의 단계들을 정교하게 통제하는 것을 의미한다. 따라서 알고리즘이 논리와 통제의 결합이라는 관점은 알고리즘이 순수한 논리를 넘어서 행위와 연결될 수 있다는 것을 의미한다. 알고리즘은 어떤 기계가 특정 과제를 수행하기 위한 명령 구조를 구체화하기 때문에 행위를 한다고 말할 수 있고 수행하는 행위를 통해서 구체적인 실재성을 획득한다[4].

아드리안 맥켄지(Adrian Mackenzie)는 무선 통신 시스템에서 디지털 신호 처리에 필수적인 비터비(Viterbi) 알고리즘에 대한 구체적인 연구를 통해 비터비 알고리즘이 조합하고 생성하는 공간의 특성을 연구하였다. 비터비 알고리즘은 Wi-Fi, WiMAX, 3G, 4G 등과 같은 무선 네트워크에 필수적인 알고리즘으로서 신호의 전송과 수신에 수반되는 잡음(noise)의 개입에도 불구하고 통신이 가능하도록 한다. 비터비 알고리즘은 잡음의 개입 문제를 신호의 가장 개연성 있는 감춰진 상태(hidden states)를 찾아서 해결할 수 있다는 개념에 기반한 알고리즘이다. 비터비 알고리즘은 길쌈 부호(convolutional code)를 통해 인코딩된 데이터 스트림(data stream)을 디코딩하는데, 길쌈 부호는 현재 시점에서 전송되는 상태를 이전 상태와 포개어서 인코딩하기 때문에 감춰진 상태를 포함한다[5].

따라서 길쌈 부호와 비터비 알고리즘에서 시공간은 뉴턴적인 의미에서 균일하고 연속적인 것이 아니라 이

질적인 시간과 공간이 뒤섞여 데이터 스트림으로 코드화되기 때문에 양자 세계와 같은 공간이라고 할 수 있다[6]. 맥켄지의 연구는 두 가지 측면에서 알고리즘 연구의 전형을 보여준다. 첫 번째는 기술적으로 난해한 비터비 알고리즘을 충실하게 연구했다는 점이고 두 번째는 비터비 알고리즘이 구성하는 공간의 특성을 철학적 개념을 통해서 설명한다는 점이다.

2. 라이브 코딩에서 알고리즘 연구의 차별성

고피가 알고리즘의 물질성을 특정 기계가 수행하는 행위를 통해 설명한 바와 같이, 라이브 코딩에서 알고리즘의 물질성은 악기라는 도구적 수행에서 비롯된다. 라이브 코딩으로 발생되는 물리적인 소리는 기존의 음악에서처럼 음표와 악기를 통한 연주에 의한 것이 아니라 코드로 구현된 알고리즘에 의해 실현되기 때문이다.

라이브 코딩에서 알고리즘의 물질성 연구는 두 가지 측면에서 흥미롭다. 첫째, 실시간으로 소리를 생성한다는 목적에 의해 물리적 시스템의 한계와 제약이 드러난다. 둘째, 코드를 스크린에 투사하여 드러내기 때문에 프로그래밍 언어의 구문론과 의미론이 중요한 문제로 부각된다.

연주자가 코드를 실시간으로 작성하고 변경함에 따라 소리가 생성되기 위해서는 신경망 알고리즘이나 진화 알고리즘처럼 복잡하고 연산에 많은 시간이 소요되는 알고리즘은 적합하지 않다. 시스템이 수행해야 하는 행위와 물리적인 한계가 알고리즘을 제한하는 것이다. 오히려 라이브 코딩에서 중요한 것은 실시간으로 코드를 작성 및 결과를 출력할 수 있는 프로그래밍 언어 설계와 알고리즘의 구조와 의미가 관객에게 쉽게 전달되기 위한 직관적 표기법의 개발이다. 다시 말하면, 라이브 코딩에서는 소리의 생성이라는 고유의 목적 때문에 알고리즘이 프로그래밍 언어나 코드를 실행하는 기계에 독립적이지 않다는 측면이 분명하게 드러난다. 그리고 표기 방식과 상관성을 갖기 때문에 알고리즘의 물리적인 실재화 과정을 프로그래밍 언어와 코드라는 측면에서 접근할 수 있게 된다.

동시에, 라이브 코딩에서 알고리즘이 구체화되는 프로그래밍 언어는 전문적-소수의 프로그래밍 언어(esoteric programming language)로서 발화나 코드를 문어(written language)로 옮기는 것이 인간-기계 표현의 풍부함을 표현하는데 실패한다는 것을 강조하기 때문에[7], 알고리즘의 논리적 측면 이외에 표현성을 살펴볼 수 있게 한다. 맥켄지의 연구가 비터비 알고리즘과 그로 인해 구성되는 무선 공간의 특성을 설명하였다면, 라이브 코딩에서는 특정 알고리즘이 구체화되는 코드를 통해서 알고리즘의 표현성을 연구할 수 있게 되기 때문에 기존 연구와 차별성을 갖는다.

III. 라이브 코딩을 위한 프로그래밍 언어와 개발 환경

1. 프로그래밍 언어의 설계

기존의 음악 연주에서는 악보를 쓰는 작곡가, 악기를 제작하는 장인과 연주자가 서로 분리되어 있었다. 그러나 라이브 코딩에서는 많은 연주자들이 프로그래밍 언어의 설계와 코드 작성을 동시에 하기 때문에, 기존의 역할 구분이 더 이상 유효하지 않다. 그럼에도 기존의 역할 구분에 기대어 비유를 하자면, 프로그래밍 언어는 연주자가 실연할 수 있는 소리의 범위를 한정하는 악기와 같고 연주자가 구현할 수 있는 알고리즘은 그 연주자가 구사할 수 있는 기교라고 할 수 있다[8]. 연주자가 실연할 수 있는 기교가 악기라는 물적인 토대에 의해 제한되듯이, 라이브 코딩에서 연주자가 실행할 수 있는 알고리즘과 그 표현성은 프로그래밍 언어 설계에 영향을 받기 때문에 프로그래밍 언어와 관련된 기본적인 원리를 간단하게 살펴볼 필요가 있다.

"프로그래밍 언어는 기계-판독 가능하고 인간-판독 가능한 형태로서 계산을 서술하기 위한 표기 체계이다." 따라서 언어 설계에서 가장 중요한 것은 기계가 번역하는데 필요한 엄밀성과 단순성을 유지하면서 인간이 쉽게 이해할 수 있도록 판독성과 표현력을 갖추도록 하는 것이다. 인간-판독성을 위해서는 인간이 기계에 대한 세부적인 사항을 모르더라도 컴퓨터 동작을 가능하게 할 수 있는 추상화를 제공해야 한다. 추상화는 데이터 추상화와 제어 추상화로 구분되며, 전자는 일반적

인 데이터 값의 컴퓨터 내부 표현을 추상화하고 후자는 프로그램의 실행 경로를 수정하는 것의 성질을 추상화한다[9].

프로그래밍 언어는 연산의 패러다임에 따라 명령형(imperative), 함수형(functional), 논리(logic), 객체-지향형(object-oriented), 병행형(concurrent) 등으로 나눌 수 있다. 명령형은 명령의 순차적 수행으로 특징되고 함수형은 함수의 계산 혹은 함수의 호출을 기본 메커니즘으로 한다. 논리 프로그래밍은 무엇이 참인가를 서술하는 진술들의 집합으로 이루어져 계산의 성질을 진술하는 것으로 프로그래밍을 가능하게 한다. 객체-지향형은 여러 메모리의 위치와 그 메모리의 위치 값을 변경할 수 있는 모든 연산들을 모은 객체라는 개념을 기반으로 한다[10]. 그리고 병행형은 연산의 프로세스들이 상호작용하면서 병렬적으로 실행될 수 있도록 하는 방식을 말한다. 개별 언어는 연산의 패러다임 중에서 어느 하나에 속할 수도 있고 여러 종류의 패러다임을 종합해서 설계될 수도 있는데 이 경우에는 다중-패러다임 언어(multi-paradigm language)라고 한다.

프로그래밍 언어는 해석에 적합하거나 컴파일에 적합하게 설계될 수 있다. 프로그래밍 언어를 유용하게하기 위해서는 그 언어로 작성된 프로그램을 바로 수행하거나 수행하기에 적절한 형태로 변환하는 프로그램이 있어야 하는데 이를 번역기라고 한다. 프로그램을 바로 수행하는 번역기를 해석기라고 부르고 입력된 프로그램을 수행에 적합한 형태로 변환하는 번역기를 컴파일러라고 한다. 그리고 수행 이전에 확정될 수 있는 프로그래밍 언어의 성질들을 정적 성질이라고 하고 수행하는 동안에 결정될 수 있는 성질들을 동적 성질들이라고 한다. 보다 동적인 성질이 많은 언어는 해석에 적합하고 엄격한 정적 구조를 갖는 언어는 컴파일에 보다 적합하다. 역사적으로 명령형 언어는 정적인 성질을 갖고 있기 때문에 컴파일 되어 왔고, 함수형이나 논리 프로그래밍 언어는 보다 동적이기 때문에 해석되어 왔다[11]. 라이브 코딩에서는 코드의 변경에 따라 실시간으로 소리와 이미지가 출력되어야하기 때문에 기본적으로 해석에 적합한 언어를 설계해야 한다.

프로그래밍 언어도 자연 언어와 같이 구문과 의미로 구분될 수 있다. 프로그래밍 언어에서 구문은 그 언어에서 올바르게 구조화된 것으로 간주되는 규칙들의 집합이라고 할 수 있고, 의미론은 기호들의 조합에 주어진 의미를 다룬다[12]. 그러나 컴파일러 구성이 순전히 구문적이고 프로그래밍 언어는 기호들의 구문적인 조작 이외에 어떤 것도 의미하지 않기 때문에 세계에 대한 언어적이고 상식적인 이해를 표현할 수가 없다. 컴퓨터 언어에서 사용되는 영어 단어인 “if”, “then”, “else”, “for”, “while” 등에서처럼 프로그래밍 언어의 의미론은 그 기호가 담당하는 산술적인 작동에 내포되어 있고 기호들은 어떠한 의미론적 진술도 표현할 수 없다. 따라서 인간은 기호들이 연상시키는 것을 통해서 메타포적으로 의미를 읽어야 한다[13]. 이것이 라이브 코딩에서 관객들이 스크린에 투사되는 코드와 알고리즘을 감상하기 어려운 이유이며 라이브 코딩 연주자들이 언어를 설계하면서 도전하는 중요한 부분이다.

2. 라이브 코딩을 위한 언어의 종류와 특성

앞서 살펴본 바대로, 추상화의 수준, 연산의 패러다임, 언어의 번역, 구문과 의미에 따라 프로그래밍 언어와 그 특성을 구분해볼 수 있다. 여기서는 라이브 코딩에서 현재 주로 사용되는 언어로 ChucK, Impromptu 그리고 라이브 코드의 시각화를 살펴보고 그 특성들을 비교해보고자 한다.

라이브 코딩의 특성상 프로그래밍 언어 설계에서 두 가지 요건이 만족되어야 한다. 첫째, 실시간 제어가 중요하기 때문에 동시성(concurrency)과 타이밍(timing)을 용이하게 다룰 수 있어야 한다. 둘째, 일반인도 의미가 이해될 수 있는 표기법이 필요하다. 여기서는 이 두 요건이 어떻게 충족되는지에 초점을 두고 개별 언어와 그 개발 환경을 분석한다.

2.1 ChucK

ChucK는 게 왕(Ge Wang)과 페리 쿡(Perry R. Cook)이 2003년에 개발한 연속식 프로그래밍 언어로 객체-지향형, 병행형, 명령형 패러다임을 조합한 다중-패러다임 언어이다. 이 언어는 연주자가 의도한 시점에 소리를 생성 및 변경하면서, 생성된 소리들이 동기화될

수 있는 프레임워크를 제공하는 것에 주안점을 둔다. 이 프레임워크의 핵심적인 역할을 슈레듈러(shreduler)와 외부 인터페이스가 담당한다. 슈레듈러는 쓰레드(thread)와 유사한 슈레드(shred)들을 자동적으로 동기화한다. 그리고 외부 인터페이스에 개별 슈레드를 제어할 수 있는 명령어를 제공한다. ChucK의 코드는 연속식 컴파일러, 가상 명령 해석기(virtual instruction interpreter), 네이티브(native) 오디오 엔진, 슈레듈러와 입출력 매니저(I/O manager)로 구성되는 ChucK 가상 머신[그림 1]에 의해 컴파일 되고 실행된다[14].

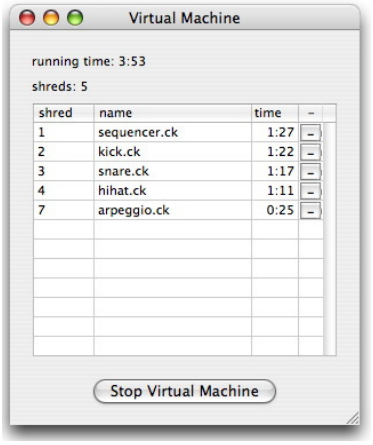

그림 1. ChucK 가상 머신

ChucK의 개발 환경 miniAudicle은 구문을 색상으로 강조해주는 텍스트 에디터, ChucK 가상 머신과 가상 머신의 상태 정보를 보여주는 콘솔 모니터로 구성되어 있다. 연주자는 여러 개의 텍스트 에디터에서 코드를 작성하여 슈레드들을 생성할 수 있고, 생성된 슈레드에서 발생되는 소리들을 중첩시키면서 즉흥 연주를 진행할 수 있다. [그림 1]에서와 같이, 가상 머신에서는 개별 슈레드들의 이름과 생성 후 경과 시간을 확인하고 실행중인 슈레드를 쉽게 선택적으로 삭제할 수 있다. 따라서 관객이 코드와 슈레드 그리고 소리의 동시적 상호관계를 파악하기 쉽다. 그러나 ChucK는 동시성과 타이밍을 위해 설계되어 언어적 추상화 수준이 높고, 부가적인 표기법에 대한 고려가 적기 때문에 관객이 코드를 직관적으로 이해하기 어렵다.

2.2 Impromptu

Impromptu는 앤드류 소렌센(Andrew Sorensen)이 2005년도에 개발한 WTP(With Time Programming) 언어로 Lisp 언어군의 하나인 Scheme을 기반으로 만들어졌다. 이 언어는 구문적으로는 Scheme과 같은 함수형 연산 패러다임을 갖고 있고, 의미론적으로는 C 언어에 가까운 다중-패러다임 언어이다[15].

Impromptu도 ChucK처럼 동시성과 타이밍을 위한 구문과 의미를 설계한다. ChucK는 슈레듈러를 통해 동기화를 제공하기 때문에 슈레드가 생성된 시간의 관리로 타이밍이 간결하게 처리된다. 그러나 Impromptu는 서버에서 동기화를 처리하는 분산 프로그래밍을 기반으로 하기 때문에 시작 시간뿐만 아니라 실행 시간도 고려가 되어야 한다. 시작 시간은 하나의 함수가 반드시 실행되어야 하는 가장 빠른 데드라인을 의미하고, 실행 시간은 주어진 함수가 실행되기 위해 가능한 최대 시간을 말한다. Impromptu는 '가장-빠른-데드라인-우선(earliest-deadline-first)'에 기초한 스케쥴링 엔진을 가지고 있다. 이 엔진이 지정된 시간 내에 실행을 강제하고, 지정한 시간에 실행이 되지 않은 이벤트는 종료 혹은 계속을 선택할 수 있는 호출이 발생되도록 설계되었다. 그리고 Impromptu는 Scheme 해석기, Scheme-to-x86 컴파일러, 동시적인 가비지 콜렉터(garbage collector), 디버거, 오디오 DSP 아키텍처와 그래픽 서브시스템으로 구성되어 있다. 언어 구성에서 OSX의 그래픽 시스템에 접근할 수 있는 그래픽 아키텍처를 포함한다는 것이 ChucK와 가장 차별화되는 특징이다[16].

Impromptu는 ChucK와 같이 언어적 추상의 수준이 높은 언어이면서 분산 프로그래밍 언어이기 때문에 ChucK 가상 머신에서 실행 상태가 보이는 슈레드와 같은 코드와 소리의 매개적인 지표가 없다. 그러나 [그림 2]의 하단부에 나타나는 바와 같이 그래픽 디스플레이를 제공하기 때문에 브이제잉(VJing)과 같은 소리와 이미지의 동기화가 가능하다. 그리고 [그림 2]의 오른쪽에 보이는 바와 같이 오디오 유닛(AudioUnit) 인터페이스를 제공하기 때문에 관련 변수들을 시각적으로 조작하면서 실시간적인 소리의 변경이 가능하다.

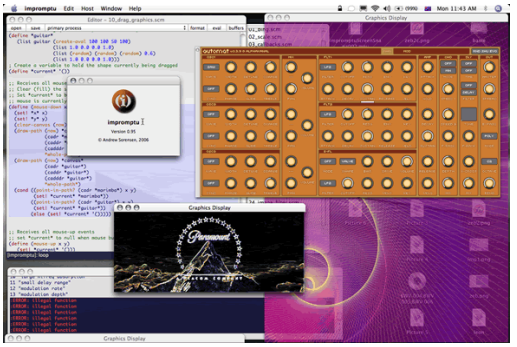

그림 2. Impromptu의 통합 개발 환경

2.3 라이브 코드의 시각화

상호작용적인 개발 환경에서는 구문을 강조하기 위한 색과 코드를 위해 디자인된 폰트 그리고 코드의 트리 구조를 탐색할 수 있는 시각적 도구들을 제공한다. 이러한 부가적인 표기법은 코드를 일차원적인 일련의 기호들로 해석하는 컴퓨터에는 의미 없는 정보이지만, 인간이 코드를 보다 쉽게 이해하는데 도움을 준다. 라이브 코드의 시각화에 해당하는 언어와 개발 환경은 바로 이 부가적인 정보를 프로그래밍 언어의 주요 구문과 의미로 전환시키고자 하는 시도의 연장선상에 있다. 색을 주요 구문으로 사용하는 ColorForth[17]나 이차원 그리드에 색을 배열하여 프로그래밍 하는 Piet[18]가 그러한 예에 해당한다.

라이브 코딩에서 기하학적 배치를 프로그래밍의 주요 구문으로 설정하는 대표적인 개발 인터페이스로 데이브 그리피스(Dave Griffiths)가 개발한 Scheme Bricks[그림 3]을 들 수 있다. Scheme Bricks는 코드를 데이터와 동등하게 간주하는 Scheme 언어의 장점과 턴테이블에서 레코드판을 손으로 돌리면서 소리를 만들어내는 과정을 프로그래밍 과정으로 차용한 Scratch[19]의 촉각적이고 시각적인 인터페이스에 영감을 받아 만들어졌다. [그림 3]에서 볼 수 있듯이 괄호(parenthesis)를 대신해 색상을 사용함으로써 Lisp 언어군에서 쉽게 발생하는 구문상의 에러를 피해갈 수 있도록 했다. [그림 3]의 오른쪽에 보이는 바와 같이, 일렬로 배열된 벽돌 모양의 코드들을 드래그 앤 드롭(drag and drop)하여 [그림 3]의 왼쪽과 같이 쌓아 올려 프로그래밍이 가능하도록 한다. 그리고 재생되는 소리에 해당되는 벽돌들이 깜빡거리게 함으로써 연주자와 관객이 코드와 소리의 관계를 직관적으로 이해할 수 있게 한다[20].

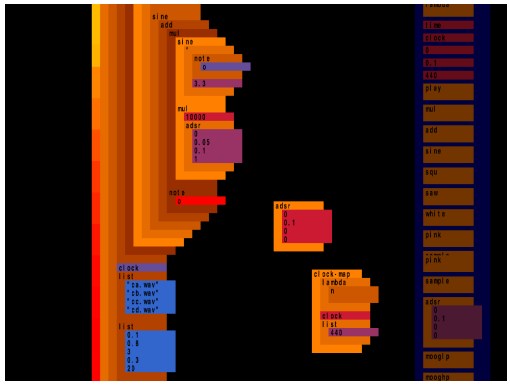

그림 3. Scheme Bricks

Scheme Bricks는 Scheme 언어의 장점을 유지하면서 색상 벽돌의 기하학적 배치로 프로그래밍이 가능하기 때문에 관객이 코드와 발생되는 소리의 관계를 공간지각적으로 이해할 수 있도록 한다. 이러한 의미에서 Scheme Bricks은 그림언어 혹은 표의문자와 같은 특성을 가진다고 할 수 있다.

IV. 라이브 코딩 연주에서 두 가지 미학적 태도

1. 대수적 태도: aa-cell

aa-cell은 Impromptu를 개발한 앤드류 소렌센과 그의 동료 앤드류 브라운(Andrew R. Brown)으로 구성된 라이브 코딩 연주 팀[그림 4]이다. Impromptu는 Object-C 언어로 작성되는 코드가 스크린에 투사되기 때문에, C나 C++과 같은 범용 언어를 이해하는 관객은 코드로 구현되는 알고리즘과 그로 인해 생성되는 소리를 연관시켜 감상하기에 용이하다. 코드를 작성해서 소리를 생성하는 미학적 태도는 순차적인 연산에 기초하는 대수적인 사고를 대변하고, 쇤베르크의 음렬주의나 야니스 크세나크시스(Iannis Xenakis), 레자렌 힐러

(Lejaren Hiller) 등의 알고리즘적 작곡을 미학적으로 계승한다.

그림 4. aa-cell의 실제 공연 장면

```
;; A Simple diatonic progression - 1st Order Markov
(define chords
   (lambda (degree)
      (play-chord 60 80 3
                  (pc:diatonic 0 '^ degree)
                  *second*)
      (callback (+ (now) *second*) 'chords
                (random (cdr (assoc degree
                                    '((i v7 iv ii)
                                      (ii v7)
                                      (iv ii v7 i)
                                      (v7 i))))))))
```

그림 5. 공연에 사용된 1차 마르코프 체인의 코드

aa-cell은 음악적으로 풍부한 결과를 산출할 수 있는 알고리즘에 관심을 두는데 특히 확률, 선형 함수, 주기 함수, 집합 이론과 재귀에 초점을 맞춘다. 확률적인 관심은 정교하게 제한된 방식의 임의성을 활용하는 방식으로 실천되며, 선형 분포나 가우시안 분포가 다양성을 효과적으로 산출하기 때문에 이를 주로 사용한다. [그림 5]는 1차 마르코프 체인을 구현한 코드의 예이다. 온음계 음악 이론에서 비롯된 1차 마르코프 체인은 음의 조화로운 진행을 가능하게 한다. 선형 함수는 음색, 음높이, 지속 시간 등을 선형적으로 배열하는 방식으로 활용되고, 주기 함수는 소리의 진폭을 매핑하여 규칙적인 패턴을 산출하는 방식으로 활용된다. 집합 이론은 "음높이 클래스 집합(Pitch Class Sets)"을 통해서 음악적 모티브를 역전, 확장 및 축소, 역행과 조 옮김을 함으로써 패턴을 조작하는 방식으로 실행된다. Impromptu는 비동기적인 속성 때문에 동시성을 위해 시간적 재귀를 제공하는데, 이를 이용해서 독립적인 음악적 주제를 동시적으로 전개할 수 있다[21].

알고리즘을 순차적인 코드 작성을 통해 구현하는 대수적 태도는 즉흥 음악으로서 역설을 내포하는 측면이 있다. 실시간으로 알고리즘을 구현하는 것은 흥미로운 패턴을 순간적으로 만들어내는 즉흥적인 기교에 해당한다. 그러나 aa-cell의 경우처럼 공연 전에 특정 알고리즘에 대한 미학적 선호도를 전제하는 것은 우연성을 강조하는 즉흥 음악의 미학과 충돌한다고 볼 수 있기 때문이다. 그럼에도 불구하고 특정 알고리즘과 산출되는 소리는 일대일로 결정되는 것이 아니라 다양한 변주가 가능한 비결정적이고 생성적인 관계이기 때문에 "비의도의 의도성(intentionality of nonintention)"이라는 즉흥 음악의 미학을 실천한다고 봐야 한다.

2. 기하학적 태도: Slub

Slub은 애드리언 워드(Adrian Ward), 알렉스 맥린(Alex McLean)과 데이브 그리피스로 구성된 라이브 코딩 연주 팀이다. Slub은 특정 언어에 국한하지 않고 라이브 코드의 시각화에 해당하는 언어나 개발 환경을 지속적으로 개발하면서 관객이 코드를 직관적으로 이해할 수 있게 하는데 초점을 맞춘다. [그림 6]은 Scheme Bricks([그림 6]의 오른쪽 부분)와 함께 Al-Jazari[그림 7]를 사용하여 진행된 공연의 한 장면이다. Al-Jazari는 <베타블로커(BetaBlocker)>라는 게임을 차용한 언어로 게임패드를 이용하여 작성된다. 그리고 간단한 그래픽 언어로 로봇끼리 상호작용할 수 있게 하고, 로봇 위에 생각 구름 이미지를 띄워 실행되고 있는 코드를 표시한다. Slub은 Scheme Bricks와 Al-Jazari 이외에도 Texture[22]나 Fluxus[23]와 같은 시각적 언어를 개발하여 공연에 사용한다. 시각적 요소들의 공간적 배치를 통해 프로그래밍을 가능하게 하는 방식은 시각적 조작을 통해 직관적 이해를 추구하는 기하학적 태도를 대변한다. 미학적으로는 존 케이지나 크리스티안 울프(Christian Wolf)의 '그래픽적 표기법'을 계승한다.

그림 6. Slub의 실제 공연 장면

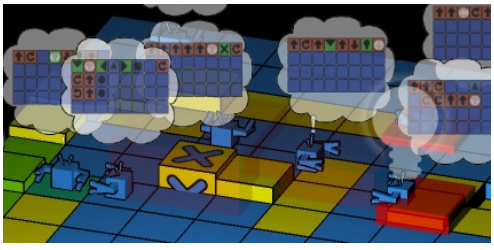

그림 7. 공연에 사용된 AI-Jazari

시각적 요소를 프로그래밍 언어의 주요 구문으로 만들고자 하는 기하학적 태도는 어떤 장면을 언어 부호와 함께 시각 부호로 함께 표상하면 기억이 향상된다는 이중 부호화(dual coding) 이론을 통해 뒷받침된다. 이 이론에서는 시각-공간적 인지와 언어적 인지가 서로를 지원하면서 병행한다고 본다. 따라서 언어 문법 내에서 개별적으로 표상된 기호를 유사언어적인 구조를 갖는 시각-공간적인 배열로 보완해주면 코드에 대한 이해가 쉬워질 수 있다. 그리고 "표기법의 인지적 차원들(the cognitive dimensions of notation)"의 관점에서 프로그래밍 언어는 "점도(viscosity)"와 "매핑의 근접도(closeness of mapping)"라는 두 차원을 갖는다. 점도는 프로그램이 얼마나 쉽게 수정될 수 있는가의 정도를 의미하고, 매핑의 근접도는 프로그래밍 언어의 표기법이 그것이 산출하는 결과와 얼마나 연관되어 있는가를 의미한다. 언어의 추상화 수준이 높을수록 점도와 매핑의 근접도는 낮아져서 프로그램의 수정이 유연해지지만 프로그램이 산출하는 결과와 코드의 인지적 상관성은 떨어진다[24].

따라서 Slub이 라이브 코딩에서 취하는 기하학적 태도는 인지적 상관성을 위해 추상화의 수준을 프로그램이 수행하는 목적에 부합하도록 최소화하면서 코드와 알고리즘에 물질성을 부여하고자 한다고 볼 수 있다. 그리고 이를 통해 연주에서 가능한 행위가 확장된다. 대수적 태도에서 가능한 행위는 쓰기에 국한되지만, 기하학적 태도에서는 시각적이고 촉각적인 인터페이스를 통해서 지원될 수 있는 행위가 언어와 개발 환경의 설계에 따라 다양해질 수 있기 때문이다. 기존의 즉흥 음악에서 다양한 물리적 대상을 조작하고 충돌시키는 행위를 통해 우연적인 소리들을 생성하듯이, 기하학적 태도에서는 코드와 알고리즘을 물리적으로 조작함으로써 즉흥적인 소리를 만들어낼 수 있다. 비록 물리적으로 조작할 수 있는 정도와 그로 인해 발생할 수 있는 우연성의 정도가 언어와 개발 환경에 의해 제한되지만, 구체적인 물질적 대상의 조작과 산출되는 우연성의 정도도 개별 대상의 물질성에 의해 제한된다. 따라서 기하학적 태도에서 물질과 행위의 관계는 기존의 즉흥 음악과 매개 차원에서 구조적으로 유사하다.

V. 라이브 코딩에서 알고리즘의 미학적 함의

라이브 코딩이 알고리즘적 작곡과 구분되는 것은 과정의 강조와 작품의 형식적 원리인 알고리즘이 시각적으로 드러나는 데 있다. 다시 말하면, 라이브 코딩에서는 알고리즘의 실행 결과뿐만 아니라 지속적으로 수정되는 과정과 시각적으로 표현되는 방식 또한 동등하게 중요하다. 알고리즘적 작곡에서 알고리즘이 실행의 원리로서 작품 내에 은폐되어 있다면, 라이브 코딩에서 알고리즘은 실행과 표현의 차원을 동시에 갖는다.

제오프 콕스(Geoff Cox)와 알렉스 맥린 그리고 애드리안 워드는 라이브 코딩의 미학을 시와 비교한다. 시의 감상이 쓰인 언어를 낭독함으로써 이루어지듯이, 라이브 코딩의 감상은 표기된 코드의 실행에 의해서 이루어지기 때문에 유사성을 갖는다. 시를 음성적이고 시각적인 측면에서 어떻게 구조화할 것인가가 그 시의 기본 개념에 해당한다면, 라이브 코딩에서 알고리즘이 그 위

치를 차지한다. 그리고 시를 읽고 들을 때, 강세, 억양, 템포, 스타일에 따라 다양한 변주가 가능하기 때문에 시의 낭독은 라이브 코딩과 같이 생성적이다[25].

그러나 시에서 개념과 개념의 표기 그리고 실행이 단계적으로 분리되어 있다면 라이브 코딩에서는 그 단계들이 하나로 통합되는 차이가 있다[26]. 라이브 코딩이 시와 유사하다면, 라이브 코딩의 대수적 태도와 기하학적 태도에 대응하는 시적 태도를 개념시(conceptual poetry)와 구체시(concrete poetry)에서 각각 발견할 수 있다.

개념시는 케네스 골드스미스(Kenneth Goldsmith)가 1999년 버팔로에서 인터넷이라는 새로운 미디어에 적절한 쓰기의 방식을 고민하면서 제안한 "개념적인 글쓰기(conceptual writing)"가 시에 적용된 결과이다. 개념시는 시인의 주관적 감정 표현과 이를 읽는 독자에게 감정적인 촉발을 일으키는 낭만주의 시의 프레임에서 벗어나서 생각과 개념을 시에서 가장 중요한 측면으로 본다.

예를 들어 케네스 골드스미스의 <하루(Day)>라는 작품은 2000년 9월 1일자 <뉴욕 타임스(The New York Times)>를 재작업해서 900 페이지 분량의 책으로 재인쇄한 작품이다. 여기서 이 책을 다 읽어서 의미를 이해하고자 하는 시도는 실패한다. 이러한 작업에서 보다 중요한 것은 "독자로서의 자질(readership)"이 아니라 "생각하는 자로서의 자질(thinkership)"을 불러일으키는 것이다[27]. 개념시에서 언어는 시인이 의미, 운율, 박자 등을 독창적으로 구성하는 미학적 대상이 아니라, "물질로서의 언어이고, 과정으로서의 언어이며, 기계 안으로 치워지고 여러 페이지에 걸쳐 펼쳐지며, 단지 폐기되고 다시 재사용될 뿐인 언어"[28]이다.

라이브 코딩에서 대수적 태도도 언어에 대해 개념시와 동일한 입장을 취한다. 일반 관객 입장에서 연주자가 실시간으로 작성하는 코드는 읽어도 의미가 전달될 수 없는 기호들의 나열이다. 관객이 완성된 텍스트를 독해하듯이 라이브 코딩에서 작성되는 코드를 읽는다면 감상은 실패할 수밖에 없다. 관객은 지속적으로 변화하는 코드와 연주자의 행위 그리고 생성되는 소리의 관계를 파악하기 위해서 의식적으로 개입해야하기 때문에 개념시에서와 마찬가지로 "생각하는 자로서의 자질"이 요구된다. 그리고 그 자질을 발휘해 파악하는 것은 시간상에서 소리의 패턴을 직조하는 알고리즘이다.

구체시는 1956년 브라질 상파울루에서 노이간드레스(Noigandres)라는 그룹이 개최한 전시에서부터 비롯되었다. 구체시는 시에서 전통적으로 중요한 요소로 간주되어 왔던 리듬이나 의미뿐만 아니라 기호들의 시각적 배치를 강조한다. 다시 말해서, 구체시는 페이지로 구성된 책과 활자로 인쇄되는 언어의 물질성을 배열, 철자, 타이포그래피 등을 통해 조작함으로써 의미를 구성한다.

마이크 보켄트(Mike Borkent)는 인지과학에서 체화된 인지(embodied cognition) 접근을 이용하여 구체시를 설명한다. 체화된 인지 접근에서 언어는 기표와 기의의 임의적 조합이 아니라, 인간 몸의 경험에서부터 비롯되는 "개념적 메타포들(conceptual metaphors)"과 "체현된 형태들(embodied gestalts)"이 언어의 근간을 이룬다고 본다. "체현된 형태들"은 인간과 인간을 둘러싼 사물들 관계의 의미를 제공한다. "이미지 도식들(image schemas)"은 "체현된 형태들"로 구성되고 지각과 개념을 연결한다. 그러므로 이미지 도식들은 의미의 본질적인 측면이며 문법적인 시스템과 구문의 수준에서 사물들을 이해하는데도 영향을 끼친다. 따라서 조한나 드러커(Johanna Drucker)의 표현대로 개념시는 "전통적으로 비언어적인 시각 형식의 능력을 구문적 혹은 의미론적 관계들로 밀어붙이는" 것이라는 정의가 가능하다[29].

라이브 코딩에서 기하학적 태도가 시도하는 것은 상하좌우의 위치 관계나 계층 구조 등의 공간적 관계를 시각화함으로써 언어의 구문과 의미를 구성하는 것이다. 따라서 기하학적 태도에서는 알고리즘이 이미지 도식으로 표현되며 사고의 흐름을 공간상의 구조로 표상할 수 있고 관객의 직관을 불러일으킨다.

VI. 결 론

본 논문은 일반적으로 형식 논리라는 측면에서 연구되는 알고리즘이 어떻게 지각적 대상으로서 물질성을

획득하는가와 미학적인 감상의 대상이 되는가의 문제를 라이브 코딩을 통해서 다루었다. 라이브 코딩에서는 실시간으로 작성 및 변경되는 코드에 따라 소리가 생성되고 코드를 스크린에 투사해서 공개하기 때문에, 동시성과 타이밍 그리고 이해하기 쉬운 표기법을 고려한 프로그래밍 언어의 설계가 중요하다.

그러나 이 두 요건을 동시에 만족하는 언어의 설계는 모순에 직면한다. 동시성과 타이밍의 유연한 제어를 위해서는 언어의 추상화 수준이 높아져야 하고, 추상화 수준이 높으면 코드와 프로그램이 산출하는 결과의 인지적 상관성이 떨어지기 때문이다. 현재 라이브 코딩에서 주로 사용되는 ChucK, Impromptu는 추상화 수준이 높은 언어이고, 라이브 코드의 시각화는 직관적인 표기법에 초점을 맞춘 언어이다.

Impromptu를 사용하는 "aa-cell"의 공연과 라이브 코드의 시각화를 사용하는 "slub"의 사례 분석에서 나타나듯이, 라이브 코딩의 미학적 태도는 연주자가 언어 설계에서 취하는 입장에 따라 대수적 태도와 기하학적 태도로 나눌 수 있다. 대수적 태도는 시간상에 순차적인 개념의 전개에 초점을 맞추고, 기하학적 태도는 개념의 구조를 공간상에 시각적 구조로 물질화하는데 중심을 둔다.

대수적 태도에서 언어 자체는 일반 관객에게 의미를 전달하는 매개체가 아니라 의미가 독해될 수 없는 기호들의 나열이다. 그러나 관객은 실시간으로 표기되는 기호들과 소리 그리고 연주자의 행위에 의식적으로 개입하여, 연주자의 의도와 작품의 전개 과정을 파악한다. 이러한 맥락에서 대수적 태도는 개념시의 미학과 연결된다. 기하학적 태도는 언어의 시각적 형식을 강조한다. 공간상의 배열과 계층 구조 등을 통해서 언어의 구문과 의미를 구성하고, 이미지 구조로 표현된 알고리즘을 관객이 직관적으로 이해할 수 있도록 한다. 이러한 맥락에서 기하학적 태도는 구체시의 미학과 연결된다. 그리고 미학적 태도의 차이는 라이브 코딩 공연에서 가능한 연주자의 행위를 제한한다. 대수적 태도에서 연주자는 쓰기라는 행위만 가능한 반면에 기하학적 태도에서 연주자는 코드를 물질적 대상처럼 조작할 수 있다.

알고리즘이 프로그래밍 언어와 기계에 독립적이라고 간주한다면, 앞서 살펴본 라이브 코딩의 미학적 태도 차이를 간과하게 된다. 라이브 코딩 연주에서 대수적 태도와 기하학적 태도로 대변되는 언어에 대한 입장 차이는 개념과 물질 중에 무엇을 강조하는가에 달려있고 이 차이가 라이브 코딩에서 알고리즘의 표현성을 규정한다.

참 고 문 헌

[1] M. Fuller, "Introduction, the Stuff of Software," in Matthew Fuller (ed.), *Software Studies: A Lexicon*, The MIT Press, p.1, 2008.

[2] D. M. Berry, "Introduction: Understanding the Digital Humanities," in David M. Berry (ed.), *Understanding Digital Humanities*, Palgrave Macmillan, p.4, 2012.

[3] A. McLean, "Visualisation of Live Code," Proceedings of Electronic Visualisation and the Arts Conference, pp.26-30, 2010.

[4] Andrew Goffey, "Algorithm," in Matthew Fuller (ed.), *Software Studies: A Lexicon*, The MIT Press, pp.15-19, 2008.

[5] A. Mackenzie, "Intensive movement in wireless digital signal processing: From calculation to envelopment," Environment and Planning A, Vol.41, pp.1294-1308, 2009.

[6] R. Kitchin and M. Dodge, *Code/Space: Software and Everyday Life*, The MIT Press, p.248, 2011.

[7] G. Cox and A. McLean, *Speaking Code: Coding as Aesthetic and Political Expression*, The MIT Press, p.6, 2013.

[8] IOhannes m Zmölnig and Gerhard Eckel, "Live Coding: An Overview," Proceedings of the International Computer Music Conference, pp.295-298, 2007.

[9] K. C. Louden, 김도형, 이수현, 창병모 공역, *프로그래밍 언어 제2판*, 사이텍미디어, pp.1-16, 2005.

[10] K. C. Louden, 위의 책, pp.16-24, 2005.
[11] K. C. Louden, 위의 책, pp.28-36, 2005.
[12]http://en.wikipedia.org/wiki/Syntax_(programming_languages)
[13] F. Cramer, "Language," in Matthew Fuller (ed.), *Software Studies: A Lexicon*, The MIT Press, p.169, 2008.
[14] Ge Wang and P. R. Cook, "On-the-fly Programming: Using Code as an Expressive Musical Instrument," Proceedings of the 2004 International Conference on New Interfaces for Musical Expression, pp.138-143, 2004.
[15]http://en.wikipedia.org/wiki/Impromptu_(programming_environment)
[16] A. Sorensen and H. Gardner, "Programming with time: Cyber-physical programming with Impromptu," Proceedings of the ACM international conference on Object oriented programming system languages, pp.822-834, 2010.
[17] http://www.colorforth.com/cf.htm
[18] http://www.dangermouse.net/esoteric/piet.html
[19] http://scratch.mit.edu/
[20] A. McLean, "Visualization of Live Code," http://yaxu.org/writing/visualisation-of-live-code.pdf.
[21] Andrew Sorensen and Andrew R. Brown, "aa-cell in practice: An approach to musical live coding," Proceedings of the International Computer Music Conference, pp.292-299, 2007.
[22] http://yaxu.org/category/texture/
[23] http://www.pawfal.org/fluxus/
[24] A. McLean and G. Wiggins, "Texture: Visual Notation for Live Coding of Pattern," Proceedings of the International Computer Music Conference, pp.621-628, 2011.
[25] G. Cox, A. McLean, and A. Ward, "The Aesthetics of Generative Code," http://ada.lynxlab.com/staff/steve/public/docu/lidia/docu/cox_aesthetics_generative_code.pdf
[26] F. Cramer, "Concepts, Notations, Software, Art,"http://www.netzliteratur.net/cramer/concepts_notations_software_art.html
[27] K. Goldsmith, "Conceptual Writing: A Worldview,"http://www.poetryfoundation.org/harriet/2012/04/conceptual-writing-a-worldview/
[28]http://epc.buffalo.edu/authors/goldsmith/Goldsmith_ConceptualWriting.pdf
[29] M. Borkent, "The Materiality of Cognition: Concrete Poetry and the Embodied Mind," WRECK, Vol.3, No.1, pp.6-12, 2010.

저 자 소 개

오 준 호(Jun-Ho Oh) 정회원

- 2001년 2월 : 서울대학교 재료공학부(공학사)
- 2006년 5월 : The School of the Art Institute of Chicago, Film/Video/New Media(MFA)
- 2012년 8월 : 카이스트 문화기술대학원(Ph.D)
- 2013년 3월 ~ 현재 : 서강대학교 영상대학원

<관심분야> : 디지털문화콘텐츠, 뉴미디어 예술, 소프트웨어 연구, 비판적 코드 연구, 디지털 인문학