한글 글립의 조형적 분석에 기반한 중간 폰트 생성

(Intermediate Font Generation based on Shape Analysis of Hangul Glyph)

구 상 옥 † 정 순 기 ††
(Sang Ok Koo) (Soon Ki Jung)

요 약 본 논문에서는 외곽선 폰트의 한글 글립을 분석하고 서로 다른 두 폰트에 대한 중간 폰트를 생성하는 방법을 제안한다. 한글 글립은 글자, 자소, 획과 같이 계층적으로 표현되고 분석된다. 글립 분석 결과를 바탕으로 같은 글자를 나타내는 서로 다른 두 글립에 대해서 모핑을 수행함으로써 여러 개의 중간 글립들을 얻는다. 자연스러운 글립 외곽선 모핑을 위해 스트링의 가중 평균(weighted-mean)에 의한 커브 모핑 방법을 사용하며, 위상이 다른 글립 간 변환을 위한 네 가지 연산을 제공한다. 제안된 한글 글립 모핑 방법은 기존의 폰트 또는 손글씨로 부터 새로운 폰트를 생성하는 데 사용될 수 있다.

키워드 : 한글 글립 분석, 외곽선 폰트, 외곽선 모핑, 폰트 생성

Abstract This paper presents a method for analyzing Hangul glyphs with their outline fonts and obtaining intermediate fonts with two different fonts. The glyphs are represented and analyzed hierarchically such as characters, components(letters) and strokes. With the analysis results, we obtain several intermediate glyphs by morphing two different glyphs of same character. For a natural glyph contour morphing, we employ the curve morphing algorithm by weighted mean of strings. In addition, we provide four operations for transformation of glyphs with different topology. As a result, it is illustrated that the proposed Hangul glyphs morphing scheme is useful for new font generation from any exist fonts or handwritings.

Key words : Hangul glyph analysis, outline font, contour morphing, font generation

1. 서 론

컴퓨터의 보급으로 문서 작성 시 워드 프로세서 소프트웨어의 활용이 일반화됨에 따라, 각 국가별, 언어별로 다양한 디지털 폰트(digital font)가 개발 및 활용되고 있다. 또한 방송 자막과 광고, 개인용 PC를 비롯하여 웹과 모바일 환경으로 점차 폰트의 쓰임새가 다변화되고 있다. 최근에는 디지털 폰트의 획일성과 표현의 한계 때문에, 좀 더 자연스럽고 다양한 감정이나 개성을 표현할 수 있는 손글씨의 감성을 살린 다양한 디지털 폰트가 제작되고 있다[1-3]. 손글씨의 디지털 폰트화 작업은 디지털의 느낌을 최대한 감추고 아날로그적인 감성과 필력을 느낄 수 있는 서체를 일반 PC 사용자들이 쉽게 사용할 수 있게끔 하는데 목적이 있다. 이는 디지털 폰트를 통해 편리성과 대중성뿐만 아니라, 손글씨의 심미성과 독창성 및 개성을 동시에 추구하고자 하는 사용자의 요구를 반영하는 자연스러운 추세로 여겨진다. 손글씨가 개인의 정체성 또는 개성을 나타내는 매개체로 사용된다면, 특정 언어권 또는 특정 집단의 사람들이 사용하는 디지털 폰트는 해당 집단의 정체성을 표방하기 위한 목적으로 개발되기도 한다[4]. 예를 들어, 세계의 일부 도시들은 정체성 확립과 도시 브랜드 강화를 위해

· 이 논문 또는 저서는 2007년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2007-521-D00425)

† 학생회원 : 경북대학교 컴퓨터공학과
sokoo@vr.knu.ac.kr
†† 종신회원 : 경북대학교 컴퓨터공학과 교수
skjung@knu.ac.kr

논문접수 : 2008년 11월 28일
심사완료 : 2009년 4월 6일

정보과학회논문지: 시스템 및 이론 제36권 제4호(2009.8)

고유의 서체를 지정하여 공공문서, 포스터, 간행물, 공공사인 등에 사용해왔다. 이처럼 디지털 폰트도 이미지와 느낌을 전달하는 매체로 활용되어진다.

그러나 디지털 폰트를 제작하는 과정은 숙련된 타이포그래퍼가 한글 완성형 폰트 한 벌인 2,350자를 만드는데 몇 개월의 긴 시간이 걸리는 힘든 작업이다. 비록 폰토그라퍼(FontoGrapher)[5], 폰트랩(FontLab Studio), 폰트크리에이터(FontCreator) 또는 폰트매니아(Font-Mania) 등의 디지털 폰트 개발 소프트웨어의 사용이 예전 주조활자 시대에 비해 서체 개발에 드는 시간과 노력을 매우 많이 절약해 준다고 하더라도 완성도 높은 서체를 개발하는 작업은 매우 길고 힘든 작업이다. 게다가 한글은 초성, 중성, 종성으로 한 글자가 만들어지므로, 글자의 균형에 따라, 획의 길이에 따라서 글자의 느낌이 상당히 달라진다. 그러나 수동으로 글자의 느낌을 결정하는 여러 가지 인자들을 조금씩 바꾸어 보면서 모든 글자들에 대해서 일일이 테스트하는 작업은 많은 시간과 노력을 요구한다. 이러한 서체 개발의 어려움에도 불구하고, 다양한 글꼴환경에 대한 사용자의 요구가 높아지면서, 자신만의 개성이나 목적에 맞게 서체를 직접 개발, 제작해 쓰는 경우가 늘어나고 있는 추세이다. 앞으로 더욱 다양하고 새로운 폰트에 대한 요구는 계속 증가할 것으로 예상된다.

폰트는 글립의 크기(size), 비뚤어짐(slant), 특정 획의 모양(stroke shape) 등의 속성에 따라 느낌이 많이 달라지며, 기존 폰트에 약간의 변형만 가하여도 새로운 느낌이 나는 경우가 있다. 실제로 서체 개발자들이 새로운 폰트를 개발할 때에도 기존 폰트의 모양에서 약간씩의 변형을 꾀하거나 기존의 두 개의 폰트의 특성을 융합하여 새로운 폰트를 만드는 경우도 있다[6].

본 논문에서는 외곽선 폰트의 한글 글립을 한글의 계층적인 구조를 이용하여 분석하는 방법을 제안한다. 분석된 정보를 바탕으로 매칭(matching)되는 두 글립 간의 모핑(morphing)을 통해 자동으로 중간 폰트를 생성할 수 있음을 보인다. 문자 모핑을 적용하는 선행 연구로는 먼저 랜덤 폰트(random font) 생성에 관한 연구가 있다[7]. 이는 사용자로부터 같은 글자에 대한 여러 샘플 손글씨를 입력 받아, 샘플 글자들의 곡률을 기준으로 최소신장트리(minimum spanning tree)를 만들고 임의의 보간(interpolation) 및 외삽(extrapolation)을 통해 조금씩 다른 폰트를 생성한다. 다음으로 타블렛이나 PDA와 같은 펜 기반 인터페이스에서 가독성(legibility)을 높이기 위해, 손으로 입력한 문자 및 기호 정보를 래스터(raster) 폰트로 변환할 때, 점차적으로 부드럽게 변환되도록 문자를 모핑(morphing)하는 Smart Text 시스템이 있다[8]. 두 연구 모두 영어 문자에 대해 적용한 것이며, 한글에 대한 연구는 없었다.

본 논문에서는 글자의 형상 정보를 획득하기 위해 운영체제 또는 각종 소프트웨어에서 제공되는 디지털 폰트 중 외곽선 폰트(outline font)를 이용한다. 외곽선 폰트는 글자의 모양을 이루는 한 개 이상의 닫힌곡선(closed curves, contour)을 베지어 곡선(Bézier curve)으로 표현한다. 현재 가장 널리 사용되는 대표적인 외곽선 폰트 형식으로는 애플사와 마이크로소프트사가 매킨토시와 윈도우즈 등 운영체제에 상관없이 사용할 수 있도록 공동으로 명세한 트루타입(TrueType) 폰트[9,10]와 이를 확장하여 포스트스크립트(postscript) 파일 형식[11]을 포함하도록 마이크로소프트사와 어도비사가 공동 개발한 오픈타입(OpenType) 폰트가 있다[12]. 트루타입 폰트는 2차의 B-스프라인 곡선으로 글자의 외곽선을 표현하며, 포스트스크립트 폰트는 3차 B-스프라인 곡선을 사용한다. 오픈타입 폰트는 두 가지 표현을 모두 포함한다. 우리는 기존에 만들어진 외곽선 폰트를 이용함으로써 다양한 크기와 모양의 글자 정보를 손쉽게 얻을 수 있다.

모든 외곽선 폰트는 글립(glyph)이라고 불리는 글자 모양에 대한 정보, 즉 한 폰트 내의 각 글자(character)의 이미지(image)를 가지고 있고, 각 글립마다 고유한 인덱스를 부여한다. 외곽선 폰트에서 한 글자안의 각각의 독립된 폐곡선(contour)은 조절점(control point)의 열로 표현된다. 각 조절점들은 노트(knot)로 불리는 곡선 위의 점(on-curve) 또는 곡선 밖의 점(off-curve)으로 분류된다. 외곽선들은 B-스프라인들로서, 연속적인 2차 또는 3차의 베지어 곡선과 같다. 트루타입 폰트처럼 2차 B-스프라인의 경우 각 베지어 곡선은 세 개의 조절점으로 정의된다. 만약 베지어 곡선의 세 조절점이 z_0, z_1, z_2라면, t가 0에서 1까지 변화함에 따라 생성되는 곡선 위의 모든 점 $Z(t)$는 식 (1)과 같이 나타낼 수 있다. 여기서 점 z_0와 z_2는 곡선 위의 점이고, z_1은 곡선 밖의 조절점이다.

$$Z(t)=(1-t)^2 z_0+2t(1-t)z_1+t^2 z_2, \quad (0\le t\le 1). \quad (1)$$

포스트스크립트 폰트처럼 3차 베지어 곡선의 경우, 4개의 2차원 조절점이 z_0, z_1, z_2, z_3일 때, 곡선은 식 (2)의 포인트 $Z(t)$의 집합으로 정의된다.

$$Z(t)=(1-t)^3 z_0+3t(1-t)^2 z_1+3t^2(1-t)z_2+t^3 z_3, \quad (0\le t\le 1). \quad (2)$$

글자의 외곽선은 위와 같이 정의된 베지어 곡선의 연속으로 표현된다.

본 논문의 2절에서는 이러한 외곽선 폰트로부터 한글의 조형적 특징을 이용하여 한글 글립의 구성성분, 즉 획과 자소를 추출하는 알고리즘을 제안한다. 3절에서는 대응되는 외곽선끼리의 모핑을 통해 두 글립 사이의 중

간 글립을 생성해내는 방법을 설명한다. 4절에서는 글립 분석 결과와 중간 글립 생성에 대한 다양한 결과를 보여주고, 마지막으로 5절에서 결론을 맺고 향후 연구에 대해 논한다.

2. 한글 글립 분석

한글은 초성 자음과 중성 모음, 그리고 선택적인 종성 자음으로 하나의 음절을 나타내는 글자를 생성하는 조합형 문자 체계를 가지고 있다. 이는 한글만의 고유한 글자 생성 체계로서, 글자 인식이나 분석 및 생성에 있어 알파벳 기반 언어와는 다른 규칙 및 방법의 적용이 필요하다. 이 절에서는 한글의 조형적 특징을 바탕으로 외곽선 폰트로부터 얻은 한글 글립 데이터의 표현 및 처리에 대해 설명하고, 서로 다른 두 폰트의 한글 글립으로부터 획과 자소를 추출하는 방법에 대해 설명한다.

2.1 한글의 조형적 특징

한글 음절은 19개의 초성 자음, 21개의 중성 모음, 그리고 공백을 포함한 27개의 종성 자음으로 이루어진다. 우리가 지금 사용하고 있는 한글의 문자 목록은 [초성+중성]의 결합, [초성+중성+종성]의 결합만을 하나의 글자로 인정한다면, 모두 11,172개의 한글 문자가 가능하다. 문자목록에 고유한 숫자를 할당한 문자코드 중, 현재 국제표준으로 되어 있는 유니코드(UNICODE)의 코드표에서는 한글 '가'~'힣'의 문자에 0xAC00~0xD7A3를 할당해서 사용하고 있다. 유니코드는 한글 한 글자에 하나의 값을 할당하는 완성형 코드가 아니라, 초성, 중성, 종성의 자소 목록에 코드를 할당하고 각각의 초성-중성-종성의 값을 조합해서 문자를 구성하는 조합형 코드이다[13]. 따라서 한 글자의 유니코드 값은 0xAC00(유니코드 한글 시작점) + [초성번호]*(21*28) + [중성번호]*28 + [종성번호] 로 손쉽게 구할 수 있다. 반대로 이 공식을 이용하여 한글 유니코드로부터 어떤 글자가 어떤 초성-중성-종성 요소들의 집합인지 쉽게 알아낼 수 있다.

한글 글립의 전체적인 모양은 초성, 중성, 종성 세 구성요소가 하나의 사각형 안에 배치된 형태이다. 세 구성요소의 배치 형태에 따라 크게 세로모임꼴과 가로모임꼴 글자로 나뉠 수 있다. 보다 세분화하면 그림 1과 같이 6가지 형태의 글자꼴로 나뉜다. (a),(b),(c)는 가로모임꼴 글자들이고, (d),(e),(f)는 세로모임꼴 글자들이다. 한글 글꼴의 배치는 중성의 모양 따라 다시 12가지 또는 18가지로 세분화될 수 있다. 한글 글립 형태에 관한 연구로 네모체 음절 한글 디자인에 관한 연구에서는 음절에서 사용하는 모음, 즉 중성의 모양에 따라 한글 글립의 형태가 결정됨을 보고, 모음의 모양과 글립 내에서의 위치에 따라 'ㅏㅑㅓㅕ', 'ㅐㅒㅔㅖ', 'ㅗㅛㅜㅠ', 'ㅘㅙㅝㅞ', '새게', 'ㅡ', 'ㅣ'의 7가지 계열로 구분하였다[14].

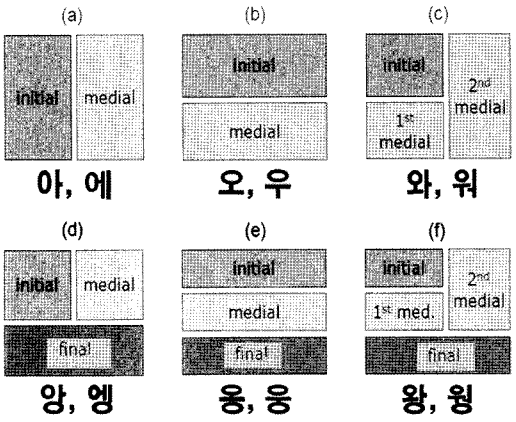


그림 1 한글 글자 조합의 6가지 형태

한글 음절의 받침(종성)이 없는 경우와, 단자음 받침인 경우, 복자음 받침인 경우 이렇게 3개의 유형으로 구분하고, 각 유형마다 위 7가지 계열로 나누어 총 21개의 계열로 구분한 뒤, 각 계열마다 초성 컴포넌트가 단자음이냐 복자음이냐에 따라 다시 구분하여, 총 42가지의 유형으로 분류하였다. 42가지 유형에 해당되는 글자로부터 추출한 자소는 모두 903개이다. 이 연구는 출판용 활자 제작의 네모꼴 한글 글꼴(자소의 줄에 돌기가 없는 산세리프(san-serif) 글꼴. 돋움체, 고딕체 등) 디자인에 있어, 903개의 핵심자소를 먼저 제작하고, 이를 조합한다면 글꼴 제작 시간을 보다 단축할 수 있을 것으로 예상하였다.

한글 자소는 하나 이상의 획으로 이루어진다. 본 논문에서는 한글 자소를 이루는 기본 획을 표 1과 같이 정의한다. 각 한글 자소가 기본 획으로 구성되는 원리는 글립 매칭, 모핑, 조합 시 중요한 정보로 활용된다.

일반적으로 한글 기본 획은 가로획, 세로획, 왼쪽삐침획, 오른쪽삐침획, 닫힌곡선획 이렇게 5가지 형태로 정의한다. 본 논문에서는 여기에 오른쪽터닝획과 왼쪽터닝획을 추가하였는데, 이 획들은 가로획과 세로획으로 분리가 가능하지만 일반적으로 한글 쓰기에서 끊김이 없이 한 번에 쓰므로 개념적으로 한 획이며, 'ㄷㄹㅁㅌㅋ' 등의 부분 획이기도 하다. 또한 'ㅊㅎ'의 맨 위쪽 첫 번째 획과 같이 길이가 매우 짧아서 가로획인지 세로획인지 구분이 모호한 획들을 분류하기 위한 점획(또는 아래아)을 추가하였다.

2.2 계층적 글자 모델과 데이터의 표현

한글 글자의 모양은 획(stroke), 자소(component or letter), 글자(character), 단어(word), 문장(sentence), 문단(paragraph)으로 계층적인 글자 모델을 가진다. 글자를 기준으로 하여 획과 자소는 글자를 이루는 단위들

표 1 한글 기본 획 및 쓰는 방향

획 이름	Label	Basic Stroke	Start Point	End Point
가로획	HORIZONTAL	—	Left	Right
세로획	VERTICAL	|	Top	Bottom
왼쪽삐침	LEFTFALLING	/	Right top	Left bottom
오른쪽삐침	RIGHTFALLING	\	Left top	Right bottom
닫힌곡선	CLOSEDCURVE	o	Center top	Center top (CCW)
오른쪽터닝	CWTURNING	ㄱ	Left top	Right bottom
왼쪽터닝	CCWTURNING	ㄴ	Left top	Right bottom
점획(아래아)	DOT(ARAE-A)	•	Left, Top	Right, Bottom

이고, 단어, 문장, 문단은 글자들이 일정한 규칙대로 모인 것이다. 글자는 자소의 조합이며, 자소는 일정한 결합 규칙을 가지는 획의 집합이다. 획은 글자를 이루는 기본 단위이다.

획의 모양은 다음과 같은 4가지 방식으로 표현될 수 있다.

(1) 점의 집합 - 획을 이루는 모든 점의 집합
(2) 외곽선 - 획의 모양, 굵기의 변화(볼륨) 표현
(3) 획의 경로(골격) 정보 - 획의 골격을 이루는 점의 집합을 시작점부터 끝점까지 추적(trace)한 경로
(4) 방향 벡터(또는 그 집합) - 각 획을 1개(또는 N개)의 방향벡터로 단순화한 것

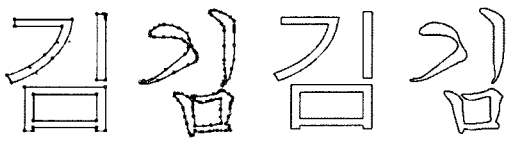
한글 글립 분석 과정에서 위 네 가지 표현 방식은 다음과 같이 사용된다.

(1) 점의 집합 - thinning 및 획 볼륨 복원 시
(2) 외곽선 - 표준폰트(고딕체류) 획 분할, 글자 매칭 및 모핑 시
(3) 획의 경로(골격) 정보 - 붓글씨체류 획 분할 및 글자 매칭 시
(4) 방향벡터의 집합 - 획의 방향 정의 및 획 분류 및 연결획(ligature) 생성 시

외곽선 폰트로부터 얻을 수 있는 데이터는 오직 글자 전체의 모양을 이루는 외곽형태에 대한 정보뿐이다. 이 정보로 부터 자소 정보 및 획 정보 등을 추출해 내어 글자의 의미적인 정보와 기하학적인 정보를 일치 시켜야 한다.

트루타입 폰트 및 오픈타입 폰트로부터 얻은 글자 글립에 대한 정보는 베지어 곡선의 조절점 리스트이다. 우리는 이를 래스터라이징하여 글자의 외곽선 이미지(boundary image)를 얻는다. 외곽선 이미지에서 글자의 내부를 채움(filling)으로써 외곽선 내부가 1(foreground)의 값을 가지고, 외부가 0(background)의 값을 가지는 이진의(binary) 글자 이미지를 얻을 수 있다. 이에 형태학적 분석(morphological analysis) 방법 중 하나인 thinning을 수행함으로써 골격 이미지를(skeleton image) 얻을 수 있다. 그림 2는 글자 모양을 표현하는 4가지 이미지들을 보여준다.

그림 2(a)의 조절점 리스트 및 (b) 외곽선 이미지에서 외곽선 정보를 체인 코드(chain-code) 형식으로 불러 올 때, 외곽선의 시작점을 정해 주어야 한다. 본 논문에서는 각 외곽선의 시작점을 각 외곽선의 최소경계사각형(Minimum Bounding Rectangle or Bounding Box)의 좌상단(left-top)점에서 가장 가까운 점으로 정하였다. 일반적으로 외부 외곽선은 반시계방향(Counter ClockWise (CCW))으로 진행되며, 내부 외곽선은 시계방향(Clock-Wise(CW))으로 진행된다. 가끔 오픈폰트인 맑은 고딕체와 같이, 외부 외곽선이 시계방향인 경우가 가끔 있는데, 일반적인 표현 방법대로 외부 외곽선은 CCW, 내부 외곽선은 CW방향으로 진행되도록 수정한다.



(a) 조절점 리스트 (b) 외곽선 이미지

(c) 글자 이미지 (d) 골격 이미지

그림 2 글자 글립 데이터의 여러 가지 표현들

2.3 획 추출(Stroke Extraction)

한글 획의 추출은 한글의 각 자소가 어떤 기본 획으로 구성되는지에 대한 정보를 미리 알고 있는 것에서 출발한다. 즉, 입력 글자가 어떤 자소로 이루어지는가에 대한

정보를 유니코드 분리를 통해 알아낼 수 있고, 각 자소가 어떤 획들로 구성되는지에 대한 정보를 이용하여, 입력 글자를 이루는 획의 종류와 글자 글립 내에서의 예상 위치를 알아낼 수 있다. 획 추출 작업은 이러한 사전 정보를 바탕으로 글자 이미지에서 글자를 이루는 각 획의 크기, 위치, 방향 및 외곽선 정보를 알아내고 각 획 마다 해당되는 기본 획 레이블을 할당하는 일이다.

그러나 글꼴에 따라 각 자소를 이루는 획의 개수나 종류 또는 외곽선의 개수나 모양이 달라질 수 있다. 예를 들어, 예를 들어 자음 'ㅈ'(지읒)의 경우, '맑은 고딕체' 등에서는 가로획+왼쪽삐침획+오른쪽삐침획으로 구성되지만, '바탕체'나 '휴먼옛체'의 경우 'ㅈ'과 같이 오른쪽 터닝획+오른쪽삐침획으로 구성된다. 또 자음 'ㅎ'(히읗)의 경우, '바탕체'와 같은 글꼴에서는 'ㅎ'과 같이 점획, 가로획, 닫힌곡선획이 각각 독립적인 외곽선을 가지지만, '맑은 고딕체'에서는 'ㅎ'과 같이 점획과 가로획이 합쳐져서 하나의 외곽선을 이룬다. 다른 예로는 'ㅊ(ㅊ)', 'ㅌ(ㅌ)', 'ㅎ(ㅎ)' 등이 있다. 또한 고딕체류의 경우 획의 굵기 변화가 적고, 획의 모양이 비교적 일정하게 나타나지만, 붓글씨체류나 손글씨체류의 경우 획의 굵기 변화가 불규칙적이고, 획의 종류를 결정하기에 모호한 경우가 많다. 따라서 획 추출 문제는 글꼴의 종류에 따라 다른 방식이 적용되는 것이 바람직하다.

획을 추출하는 방법은 크게 외곽선 대응벡터 기반, 외곽선 특징점 기반, 골격 정보 기반, 그리고 부분 모양 매칭 기반 방법으로 나눌 수 있다. 외곽선 대응벡터 기반 방법은 외곽선에서 획을 이루는 라인벡터 쌍을 찾고 두 선이 이루는 사각형을 각 획으로 정의하는 방법이다[15]. 외곽선 특징점 기반 방법은 외곽선의 각 점에서의 곡률(curvature) 또는 tangent angle의 local maxima인 점들을 찾고, 그 중 두 획의 접점에 해당되는 점의 쌍을 찾아, 외곽선을 분리하는 방법이다. 골격 정보 기반 방법은 골격 이미지에서 분기점을 찾아 분리한 뒤, 같은 방향성을 가지는 조각끼리 병합하는 방법이다. 마지막으로 부분 모양 매칭(partial shape matching) 기반 방법은 각 기본 획에 대한 모양 데이터베이스를 구축하고, 전체 글자 모양에 대한 부분 매칭을 수행함으로써 획을 추출하는 방법이다.

2.3.1 외곽선 대응벡터 기반 방법

고딕체류나 획의 끝 부분에 약간의 세리프(serif)가 추가된 명조체(바탕체)류와 같이, 대체로 획의 굵기 변화가 적고 외곽선이 단순한 글씨체의 경우에는, 외곽선에서 서로 대응되는 라인벡터 쌍을 찾기가 수월하다. 일정한 길이 이상의 대응되는 라인벡터 쌍은 하나의 획으로 정의될 수 있다.

이러한 개념에 기반 한 획 추출 과정은 다음과 같다: (1) 컨투어 그룹핑[1], (2) 라인벡터화, (3) 대응벡터 찾기(획 만들기), (4) 조절점 복원, (5) 기본 획 레이블링. 입력 데이터는 어떤 글자를 구성하는 모든 컨투어의 집합이다. 각 컨투어들은 독립적으로 해당 외곽선을 이루는 베지어 곡선의 조절점 리스트만을 가지고 있을 뿐, 다른 컨투어들과의 상대적인 위치 또는 포함 관계에 대한 어떤 정보도 가지고 있지 않다. 따라서 먼저 서로 포함관계에 있는 컨투어들을 하나의 단위로 그룹핑한다. 즉, 'ㅇ'이나 'ㅂ'과 같이 외부 컨투어와 내부 컨투어 쌍으로 이루어지는 컨투어들을 하나의 그룹으로 묶어준다. 그런 다음 각 컨투어 그룹의 최소경계사각형(minimum bounding rectangle) 위치를 가지고 좌측 위의 그룹부터 우측 아래의 그룹 순으로 정렬한다. 컨투어 그룹핑 후에는 각 컨투어 그룹 별로 베지어 곡선의 조절점 중 선 밖의 점을 제거하고 선 위의 점만을 남겨 두어, 라인벡터를 구한다. 두 라인벡터 사이의 거리, 각도, 방향 등을 토대로 하나의 획을 이루는 대응벡터 쌍을 찾는다. 각 대응벡터 쌍들은 하나의 획으로 간주되며, 각 획마다 단계(2)에서 제거된 라인 밖 조절점을 복원하여 원래 획의 모양을 회복한다. 추출된 획들은 획의 방향성, 길이 및 획 간의 연결 관계를 검사하여 8가지 기본 획으로 레이블링한다. 그림 3은 그 과정을 맑은 고딕체에 적용한 것을 중간 결과 이미지로 보여준다. 위의 획 추출 방법은 고딕체류의 한자폰트로부터 획을 추출하는 기존 방법을 한글에 적용한 것이다[15].

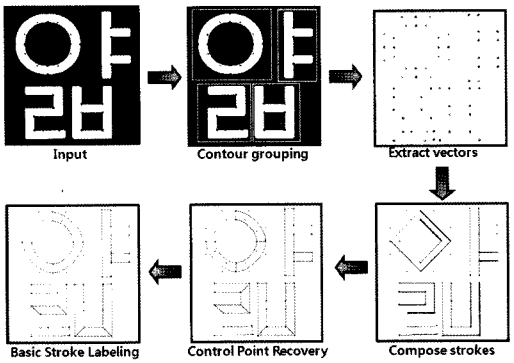


그림 3 고딕체류의 획 추출 과정

2.3.2 외곽선 특징점 기반 방법

한양해서체, 궁서체와 같이 붓글씨의 느낌을 살린 글씨체나, 일반적인 손글씨 이미지의 경우에는 획의 굵기와 모양이 불규칙하기 때문에 2.3.1절의 획 추출 방법을 적

1) '컨투어(contour)'는 '외곽선'이라는 한글 용어로 대체될 수 있으나, 이 부분에서는 'B-스프라인 형식으로 표현된 데이터'를 특별히 지칭하므로 '컨투어'라는 외국어로 표기한다.

용할 수 없다. 이 경우 외곽선의 모든 점들 중에서 두 획의 접점에 해당되는 두 점을 찾아 획을 분리할 수 있다.

두 획이 만나는 지점은 일반적으로 급격한 곡률의 변화를 가지는 두 대응점이 나타나므로, 외곽선의 모든 점들 중에서 곡률이 급격하게 변화하는 특징점을 찾고, 특징점 집합에서 획이 분리되는 대응점 쌍을 구하여 그 대응점 쌍을 기준으로 외곽선을 분리한다.

보다 자세한 외곽선 특징점 기반 획 추출 과정은 다음과 같다: (1) 컨투어 그룹핑, (2) 라인벡터화, (3) 획 분리 대응점 찾기, (5) 컨투어 분리, (5) 기본 획 레이블링. 먼저 컨투어 그룹핑 후, 선 위의 조절점만으로 라인벡터 집합을 구성한다. 컨투어 진행 방향으로 연속되는 라인벡터 쌍들 간의 각도를 계산한다. 두 벡터가 이루는 각도의 절대치가 임계값 이하인 점 중에서, 두 라인벡터 사이의 진행이 컨투어 진행방향의 반대가 되는 점들을 구한다. 그림 4와 같이, 일반적으로 컨투어 진행방향은 반시계방향인데 반해서, 국지적으로 두 벡터가 시계방향으로 연결되는 부분이 있는데, 그 때의 두 라인벡터 사이의 점은 컨투어의 오목한(concave) 부분, 즉 다른 획으로 분리되는 점일 가능성이 높다. 이러한 특징점의 집합으로부터 두 점 사이의 맨하탄 거리(Manhattan distance or City-block distance)가 임계값 이하인 점들의 쌍, 즉 획 분리 대응점들을 구한다. 각 대응점 쌍을 기준으로 컨투어를 분리하고, 각 분리된 컨투어 조각의 끊어진 부분을 연결해 주면, 한 개의 컨투어가 두 개 이상의 독립적인 획 컨투어로 분할된다. 각 획들은 획의 방향성, 길이 및 획 간의 연결 관계를 검사하여 8가지 기본 획으로 레이블링한다.

그림 5는 위의 획 추출 방법을 한양해서체에 적용한 예를 보여준다. 각 컨투어 그룹 별로 정확히 기본 획들이 추출되었음을 볼 수 있다. 그림 5에서 자음 'ㄱ'이나 'ㄹ'의 오른쪽터닝획의 경우, 꺾이는 부분의 안쪽 점은 각도가 임계값 이하이므로 특징점으로 추출되지만, 바깥쪽은 상대적으로 완만한 곡선이므로 특징점이 잘 찾아지지 않거나, 찾아지더라도 안쪽 점과의 거리가 임계값을 초과하여 대응점으로 검출되지 않으므로 자연스럽게 오른쪽터닝획이 찾아진다. 또 한양해서체의 경우, 우리가 입력으로 쓰는 외곽선 폰트 데이터에서 '밟'의 'ㅂ', 'ㅏ', 'ㄺ' 이 각각 독립적인 컨투어로 표현되어 있다. 즉, 그림 5의 '밟', '펼', '됐', '축'에 표시된 원 안의 컨투어가 겹쳐지는 부분은 위에서 제안된 방법으로 획이 분할된 결과가 아니라, 입력 데이터에서 이미 다른 컨투어인 것이다. 한글 폰트를 제작할 때에는 먼저 기본이 되는 자음 및 모음 컴포넌트와 또는 글자 조합 모듈을 만들고 이를 가지고 그 크기나 위치를 수정하여 조합하면서 모든 한글 글자에 대한 외곽선을 만든다. 이 때, '맑은 고딕체'와 같이 컴포넌트의 외곽선이 겹쳐지는 부분을 다시 하나의 컨투어로 병합하여 데이터의 크기를 줄이고 보다 자연스러운 모양이 되도록 수정하기도 하고, '한양해서체'와 같은 조합형 폰트에서는 그대로 두기도 한다. 외곽선 기반 획 추출 방법은 이와 같이 이미지 상에서는 획이 겹쳐지더라도 폰트 정보 상에서는 독립적인 컨투어로 표현된 경우, 복잡한 형태의 획을 분리해야하는 수고를 덜 수 있다는 장점이 있다.

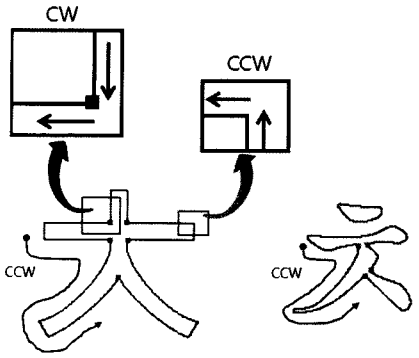


그림 4 라인벡터 간 연결방향에 따른 획 분리점 추출

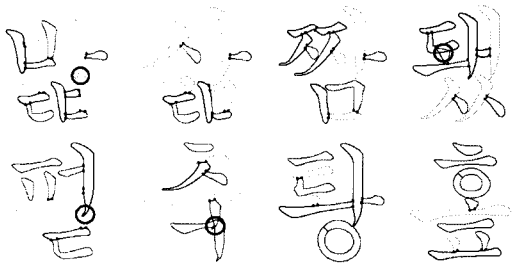


그림 5 한양해서체의 획 분리 예

외곽선 대응벡터 기반 방법과 외곽선 특징점 기반 방법은 모두 각도 및 거리에 대한 임계값들을 사용하고 있다. 거리 임계값들은 폰트 종류 및 글자의 크기가 달라짐에 따라 조금씩 변경시켜 주어야 좋은 결과를 얻을 수 있다. 각도 임계값들은 폰트의 크기에는 상관이 없지만 폰트의 종류가 달라지면 변경시켜 주는 것이 좋다. 비록 사용자가 설정해 주어야 하는 임계값의 수는 두세 개 정도로 적고 사람이 직관적으로 판단할 수 있는 수준이지만, 임계값 설정에 따라 결과의 품질이 달라지는 것은 두 방법의 주된 단점이라고 볼 수 있다.

2.3.3 골격 정보 기반 방법

글자의 골격 정보를 이용하여 획을 추출하기 위해서는 먼저 글자 이미지를 thinning하여 골격 이미지(skeleton image)를 얻는다. 골격 이미지에서 끝점(end point)과 분기점(branch point)을 찾아서 모든 부분 획

을 찾는다. 부분 획의 방향성을 계산하고, 같은 방향성을 가지는 부분 획끼리 병합한다. 병합된 획들은 획의 방향성, 길이 및 획 간의 연결 관계를 검사하여 8가지 기본 획으로 레이블링한다. 이때, 끝점이 없거나, 획의 시작점과 끝점간의 거리가 매우 짧은 경우 닫힌곡선획으로 처리하고, 가로획과 세로획이 임계 범위내의 각도로 연결되는 터닝획의 조건을 만족하는 경우에 하나의 획으로 병합한다. 레이블링 된 획들은 획의 진행방향으로 스캔 컨버전(scan conversion)하면서, 원래 획의 볼륨(volume)을 회복해가는 방법이다. 그림 6은 그 과정을 중간 결과 이미지로 보여준다.

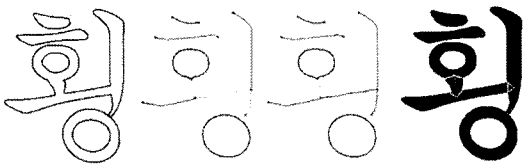


그림 6 글자의 골격을 이용한 획 분할 방법: (a) 글자이미지, (b) 골격이미지의 끝점과 분기점으로 분리된 획, (c) 획 방향성 검사를 통해 병합된 획, (d) 획의 볼륨 복원

이 방법은 사용자 입력으로 획의 개수 및 획 벡터에 대한 정보가 있을 때에는 정확한 결과를 보인다[16]. 하지만, 자동으로 처리할 경우, 모양의 복잡도 및 굵기 변화의 불규칙성에 따라 골격 이미지의 품질이 달라지고, 이로 인해 너무 많은 획으로 분리되는 오류가 발생할 수 있다. 또한 획의 병합 단계에서 휴리스틱한 룰을 적용하게 되므로, 그 또한 오류 발생의 원인이 될 수 있다.

2.3.4 부분 모양 매칭 기반 방법

부분 모양 매칭을 이용하여 획을 추출할 수도 있다. 이 방법은 각 기본 획에 대한 모양 데이터베이스를 구축하고, 전체 글자 모양에 대한 부분 매칭을 수행함으로써 획을 추출하는 방법이다. 획 모양 매칭은 적절한 척도(measure)로써 글자의 외곽선 모양과 기본 획 모양과의 차이(difference or distance)를 구하고, 가장 유사한 기본 획으로 레이블링 하는 것이다. 이 방법은 두 모양 간의 차이를 계산하기 위한 척도를 어떻게 정의하느냐가 매우 중요한데, 일반적으로 점의 위치, 곡률 및 컨투어 시작점으로부터의 거리 등을 기준으로 두 모양 간의 대응점 리스트를 구하고, 두 모양 간 변형 비용(transformation cost)을 계산함으로써 두 모양 간의 차이를 구한다. 모양 매칭을 위한 기존 방법들로는 Distance Transform, Fréchet Distance, Shape Context 등이 있다. 부분 매칭은 쿼리 모양(query shape)과 가장 일치하는 컨투어의 부분을 찾아내는 것이다. 즉 컨투어에서 모든 시작점과 모든 길이(length)에 대해 거리 계산을 하고 그 중 최소값을 찾는 문제이므로, 주로 동적프로그래밍(dynamic programming)으로 이를 해결한다. 그림 7은 맑은 고딕체의 기본 획 모양 데이터베이스의 일부분과 부분 매칭을 통한 획 레이블링의 결과를 보여준다.

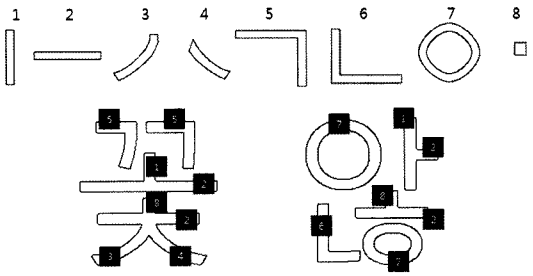


그림 7 기본획 데이터베이스와 부분 매칭을 통한 획 레이블링

이 방법은 어떤 매칭 알고리즘을 사용하는가와 기본 획 데이터베이스를 얼마나 충실히 구축하느냐에 따라 결과의 정확도가 달라진다. 위의 다른 방법들에 비해서 수행시간이 보다 많이 걸리고, 획 데이터베이스를 구축해야 하는 등의 수고가 따라야 하는 단점이 있다. 하지만, 보다 다양한 폰트에도 적용가능하고, 획의 모양이 글자마다 매우 불규칙하게 변하는 손글씨 이미지에서의 획 추출에도 적응적으로 적용 가능한 방법이므로 향후 연구에서 보다 광범위한 실험과 명확한 비교 연구가 요구된다.

2.4 자소 추출(Component Extraction)

자소 분리 문제는 문자인식 분야에서 한글의 높은 인식률을 위해 많이 연구되었던 문제 중의 하나이다[17]. 대부분의 문자들이 각 자소의 모양과 위치가 다르고, 자소 성분의 연결 및 분리 형태가 다양하게 나타나므로, 다양한 폰트 및 손글씨에서 자소를 정확히 분리하는 문제는 어려운 문제로 남아 있다.

인식을 위한 자소 분리는 해당 글자의 코드 정보를 모르는 상태에서, 글자 이미지가 어떤 자소로 이루어 졌는가에 대한 패턴 인식을 바탕으로 글자 코드를 알아내기 위한 것이다. 즉, 많은 폰트 또는 손글씨 이미지에서 각 자소의 특징을 추출하여 학습시킨 뒤, 입력 이미지에 대해서 특정 클리핑(clipping) 영역의 특징을 추출하여, 학습된 데이터 중에서 가장 유사한 특징을 가지는 자소로 분류하는 것이다. 이 때, 각 자소의 특징(점, 선분)에 대한 간단한 정보 또는 어떤 부분이 자소부분이고 어떤 부분이 잡음(noise) 부분인지에 대한 정보만 있을 뿐, 자소의 정확한 모양을 추출해낼 수 없다. 반면, 본 논문에서의 자소 추출은 글자 코드를 알고 있는 상태에서

전체 글자 모양 속에서 각 자소가 어느 위치에 어떤 모양인지를 정확히 추출해 내는 것이 목표이다. 따라서 자소 인식에 사용된 문제해결 방법을 이에 적용할 수 없으며, 보다 강력한 특징정보, 즉 각 자소가 어떤 획으로 구성되는지에 대한 지식과 3.2 절에서 추출된 글자 글립의 정확한 획 구성 정보가 필요하다.

본 논문에서는 글자 코드 정보, 한글 조합 유형 정보, 획 추출 정보를 이용하여 정확한 자소의 모양을 추출해 내고자 한다. 먼저 한글 유니코드로부터 그 글자의 초성, 중성, 종성 정보를 알아내고, 그 정보로 부터 각 초성, 중성, 종성 컴포넌트가 어떤 배치를 가지는지에 대한 한글 조합 유형 정보를 알아낸다. 그림 8과 같이 컨투어 그룹의 개수가 한글 컴포넌트 집합의 개수와 일치하고, 그 배치가 한글 컴포넌트 배치와 일치하는 경우에는 컨투어 그룹의 최소경계사각형들 간의 배치 정보를 이용하여 자소 추출이 가능하다. 즉 우리는 중성 모음의 종류를 알기 때문에 그림 1의 한글 조합 유형 중 어떤 형태에 속하는지를 알 수 있고, 각 컨투어 그룹의 최소경계사각형의 중심점 간의 각도 및 상대적인 위치관계를 검사하여, 각 컨투어가 어떤 컴포넌트에 속하는지를 결정할 수 있다. 다음은 이러한 알고리즘을 보여준다. 아래 알고리즘에서 중성 타입(medial_type)은 폰트에 따라 보다 세분화할 수 있다.

그러나 실제로 그림 9와 같이 컨투어 그룹과 컴포넌트 집합이 일치하지 않는 경우가 많으므로, 이런 경우를 해결하기 위해서는 각 컨투어가 어떤 획들로 구성되는지에 대한 정보, 즉 획 분리 정보와 각 자소가 어떤 획들로 구성되는지에 대한 정보가 필요하다. 즉 컨투어 집합을 기본 획 집합으로 해체하고, 해당 한글 자소를 이루는 획들로 재구성해야만, 정확한 자소 추출이 가능하다.

획 정보를 이용한 자소 추출 과정은 다음과 같다. 먼

```
input: contour groups of a character,
       character code (initial, medial, final component code)

get_contour_groups_BoundingBox();
contour_groups_sorting( left_top_to_right_bottom );
if ( final_code == null ) {
  switch (medial_type ) {
  case 1: // ㅏ, ㅑ, ㅐ, ㅒ, ㅣ, ㅓ, ㅕ, ㅔ, ㅖ
    component_extraction_by_Type1_BB(); break;
  case 2: // ㅗ, ㅛ, ㅜ, ㅠ, ㅡ
    component_extraction_by_Type2_BB(); break;
  case 3: // ㅘ, ㅙ, ㅝ, ㅞ, ㅢ
    component_extraction_by_Type3_BB(); break;
  }
} else {
  switch (medial_type ) {
  case 1: component_extraction_by_Type4_BB(); break;
  case 2: component_extraction_by_Type5_BB(); break;
  case 3: component_extraction_by_Type6_BB(); break;
  }
}

output: contours of all of components of the character
```

알고리즘 1 컨투어 집합 기반 자소 추출 알고리즘

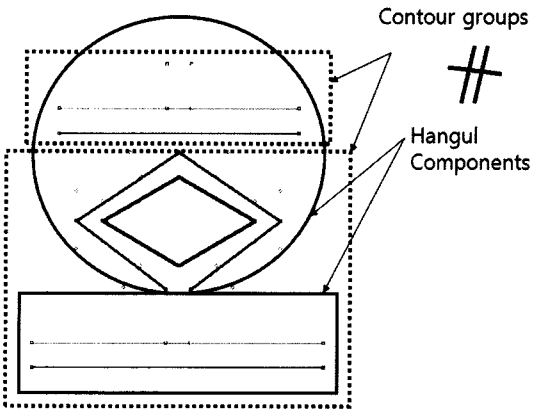


그림 9 컨투어 그룹과 컴포넌트 집합이 다른 경우

그림 8 맑은 고딕체와 한양해서체 자소 분할 예

저 중성 컴포넌트의 종류에 따라 한글 조합 유형(*Type*)이 결정된다. 그 유형에 따른 초성, 중성, 종성 컴포넌트의 초기 최소경계사각형 정보에 따라, 각 컴포넌트 최소경계사각형에 포함되는 획을 해당 컴포넌트로 추가한다. 각 자소가 어떤 획들로 구성되는지에 대한 정보를 바탕으로, 해당 컴포넌트에 추가된 획 중 만약 필요 없는 획이 있다면 제외하고, 부족한 획이 있다면 표시한다. 어떤 컴포넌트에도 포함되지 않는 획이 있으면, 획이 부족한 컴포넌트에 추가한다. 마지막으로 구해진 각 컴포넌트의 최소경계사각형 크기 및 위치와, 중심점끼리 이루는 각도를 검사하여, 글자 코드와 구해진 초성, 중성, 종성 글립이 정확히 정합되는지 확인한다. 다음 알고리즘은 이러한 자소 추출 과정을 간략히 보여준다. 그림 10은 맑은 고딕체 한글 글자들의 획 추출 결과 및 자소 추출 결과를 보여준다.

```
Type: Hangul composition type
N: the number of strokes
S = {s1, s2, ... , sN} : the set of strokes

stroke_sorting( left_top_to_right_bottom );
for (i = 1; i<=N; i++) {
 if ( si.BB <= initial_component_BB_of_Type )
   add_to_initial_component( si );
 else if ( si.BB <= medial_component_BB_of_Type )
   add_to_medial_component( si );
 else if ( si.BB <= final_component_BB_of_Type )
   add_to_final_component( si );
 else
   add_to_unclassified_strokes( si );
}
while ( num_of_unclassified_stroke > 0 ) {
  add_to_insufficient_components( su );
  if ( is_excess_strokes_of_components )
    add_to_unclassified_strokes( sex );
}

output: the stroke sets consisting of all of components
```

알고리즘 2 획 분할 정보 기반 자소 추출 알고리즘

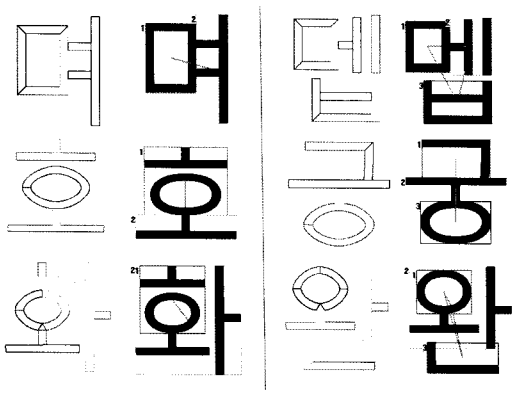


그림 10 획 분할 후 자소 분할 예(맑은 고딕체)

3. 외곽선 모핑을 통한 중간 글립 생성

모핑(morphing)은 컴퓨터 그래픽스 및 산업 디자인, 또는 로봇공학 등의 응용분야에서 널리 사용되는 기술이며, 2차원 이미지 공간, 3차원 복셀(voxel) 공간, 곡선 및 다면체(polyhedra) 등을 위한 다양한 모핑 알고리즘들이 개발되었다. 외곽선 모핑은 하나의 외곽선에서 다른 외곽선으로의 점진적인 변형(transformation)을 계산하는 것이다. 외곽선 모핑 방법은 두 외곽선 간의 대응점을 구하고, 소스(source) 외곽선의 대응점에서 타겟(target) 외곽선의 각 대응점으로의 중간 경로를 찾는 방법[18-20]과 편미분(partial differential equation) 조건에서의 함수 보간을 곡선 보간 문제로 바꾸어 푸는 방법[21] 등이 있다. 또한 두 문자열(string) 간의 편집거리(edit distance or Levenshtein distance)를 곡선 모핑에 응용한 가중 평균(weighted mean) 기반 모핑 방법이 있다[22,23]. 이 방법은 원(circle) 모양과 같은 위상을 가진 닫힌곡선(closed curve) 간의 모핑에 대해 비교적 자연스러운 중간 결과를 생성해낸다. 본 논문에서는 가중 평균에 의한 곡선 모핑 방법을 추출된 각 자소 외곽선 모핑에 적용하여 중간 글립들을 생성하고자 한다.

그런데 본 논문은 서로 다른 한글 폰트의 자소 모양 간의 모핑을 목적으로 하므로, 그림 11과 같이 위상(topology)이 다른 모양(shape) 간의 모핑을 해야 하는 경우가 있다. 외곽선 폰트는 글자의 모양을 외곽선의 집합으로 나타내므로, 다른 폰트 간에 글자 글립의 전체적인 모양은 비슷하지만, 외곽선의 개수 및 위상이 서로 다른 경우가 있다. 두 글립 간의 자연스러운 모핑을 위해서는 두 글립의 외곽선의 개수와 위상을 일치시켜야 하므로 이를 해결할 방법이 필요하다.

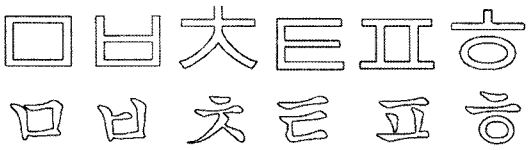


그림 11 위상이 다른 자소의 예

가중 평균에 의한 곡선 모핑 방법은 3.1절에서 자세히 다룬다. 3.2절에서는 위상이 다른 글자 모양 간의 모핑에 대해 설명한다.

3.1 가중 평균에 의한 글립 외곽선 모핑 방법

주어진 소스 외곽선 상의 점 p_1과 타겟 외곽선 상의 점 p_2가 있을 때, p_1에서 p_2로의 모핑은 다음과 같은 선형 보간(linear interpolation)을 이용하는 것이 일반적이다. 이 때, 중간 점 $p(t)$는 p_1과 p_2의 가중 평균이라

하며, $p(t=0.5)$는 p_1와 p_2 사이의 정확한 중간 값이다.

$$p(t)=(1-t)p_1+tp_2,\quad t\in[0,1]. \tag{3}$$

임의의 공간 U에서 거리 함수 $d(x,y)$, $x,y\in U$가 주어질 때, 소스 오브젝트 p_1이 연속적인 오브젝트 $p(t)$를 거쳐 타겟 오브젝트 p_2로 변형 시, 다음 속성이 지켜진다.

$$\begin{aligned} d(p(t),p_2) &= t\cdot d(p_1,p_2),\\ d(p(t),p_2) &= (1-t)\cdot d(p_1,p_2). \end{aligned} \tag{4}$$

2차원 외곽선은 xy 평면에서 연속적인 점들의 집합 $C=\{(x_1,y_1),\ldots,(x_n,y_n)\}$ 이다. 곡선을 스트링으로 매핑하기 위해서, 일정 간격으로 샘플링한 후, 연속적인 점들 사이의 방향 벡터를 구한다. 즉, 곡선 C는 Δ 거리 간격으로 샘플링 된 곡선 $C^s=\{(x_i^s,y_i^s),\ldots,(x_m^s,y_m^s)\}$ $(E\{d((x_i^s,y_i^s),(x_{i+1}^s,y_{i+1}^s))\}=\Delta,\ i=1,\ldots,m-1)$로 만들고, 벡터 집합 $V=z_1,z_2,\ldots,z_{m-1}$ (z_i는 (x_i^s,y_i^s)에서 (x_{i+1}^s,y_{i+1}^s)로의 방향 벡터)를 생성한다. 각 방향벡터는 스트링의 각 문자로 간주되어 처리된다.

두 외곽선 사이의 거리함수는 두 스트링의 차이점을 측정할 때 널리 사용되는 Levenshtein 거리[23]를 사용한다. 즉 세 가지 편집 연산(edit operation)을 이용하여 소스에서 타겟 스트링으로 가는 비용이 최소가 되는 편집 시퀀스(edit sequence)를 찾는다. 세 가지 편집 연산은 삭제(deletion) $(a\rightarrow\epsilon)$, 삽입(insertion) $(\epsilon\rightarrow a)$, 그리고 치환(substitution) $(a\rightarrow b)$이다. 각 연산의 비용을 $c(a\rightarrow\epsilon)$, $c(\epsilon\rightarrow a)$, $c(a\rightarrow b)$로 표기할 때, $c(a\rightarrow\epsilon)=c(\epsilon\rightarrow a)=|a|=\Delta$이고, $c(a\rightarrow b)=|a-b|$이다.

소스 외곽선 X에서 타겟 외곽선 Y로의 임의의 편집 연산 시퀀스(sequence)를 $S=e_1,\ldots,e_k$라고 하자. S의 비용은 $c(S)=\Sigma_{i=1}^{k}c(e_i)$로 정의되며, 두 외곽선 사이의 편집 비용 $d(X,Y)$는 X에서 Y로의 모든 편집 연산 시퀀스 중에서 가장 비용이 적게 드는 시퀀스의 비용이 된다. $d(X,Y)$의 계산은 다음과 같은 동적 프로그래밍(dynamic programming) 알고리즘으로 계산된다. 아래 식에서 최종적으로 $D(n,m)=d(X,Y)$가 된다.

D: $(n+1)\times(m+1)$ *edit mat.*

$$\begin{aligned} &D(0,0)=0,\\ &D(0,j)=D(0,j-1)+c(\epsilon\rightarrow Y_j), && \text{for } j=1,\ldots,m\ ,\\ &D(i,0)=D(i-1,0)+c(\epsilon\rightarrow X_i), && \text{for } i=1,\ldots,n\ , \end{aligned}$$

$$D(i,j)=\min\begin{Bmatrix} D(i-1,j-1)+c(X_i\rightarrow Y_j)\\ D(i-1,j)+c(X_i\rightarrow\epsilon)\\ D(i,j-1)+c(\epsilon\rightarrow Y_j) \end{Bmatrix}. \tag{5}$$

우리는 $\gamma=\sum_{e\in S'}c(e)$의 비용을 가지는 S의 부분 시퀀스 S'에 대해서, p_1에 S'을 적용함으로써 외곽선 I를 얻을 수 있다. 이 때, $d(I,X)=\gamma=t\cdot d(X,Y)$이고, $d(I,Y)=d(X,Y)-\gamma=(1-t)\cdot d(X,Y)$이 된다. 우리는 모핑을 위해 $t=\frac{\gamma}{d(X,Y)}$ 의 값이 $\frac{1}{k+1},\frac{2}{k+2},\ldots,\frac{k}{k+1}$ (k는 전체 중간 외곽선의 개수)와 같은 값일 때의 중간 외곽선들을 연속적으로 구한다.

3.2 위상이 다른 글립 간 모핑

위상이 다른 자소 간의 모핑까지 해결하기 위한 손쉬운 방법으로, 그림 12와 같이 획 분할 및 매칭 후, 대응되는 획 별로 각각 모핑을 하고, 마지막으로 연결된 획들을 하나의 외곽선으로 합치는 방법을 생각해 볼 수 있다. 그러나 이 방법은 획 분할 및 획 정합 결과가 정확하지 않으면, 다소 이상한 중간 결과가 생성될 수 있다. 또한 실험 결과, 되도록 모핑 할 외곽선의 개수를 줄이는 것이 보다 나은 모핑 결과를 생성함을 알 수 있었다.

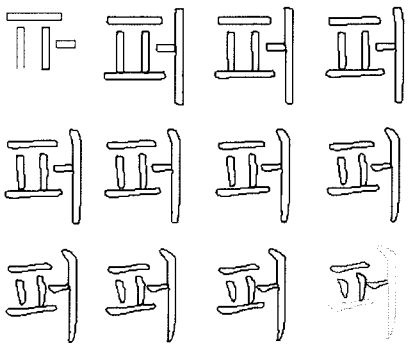


그림 12 각 획 별로 모핑한 결과 예

본 논문에서는 에러를 줄이고 보다 나은 결과를 생성하기 위해서 모든 글립에 대해서 무조건 획을 분할하는 것이 아니라, 그림 11과 같이 위상이 다른 글립들에 대해서 위상을 변화시켜 동일한 위상으로 만들기 위한 네 가지 연산을 제안한다. 네 가지 연산은 Merge, Split, Add, Delete 이며, 그림 13은 이러한 연산들을 도식화한 것이다. 다음과 같은 경우들에 각각 다음 연산들이 사용될 수 있다. 그림 14는 ㅁ, ㅂ, ㅎ 글립을 Split 연산을 사용하여 맑은 고딕체에서 한양해서체로 모핑한 결과이다. 맑은 고딕체의 경우 ㅁ, ㅂ의 외곽선 개수가 두 개로 외부 및 내부 외곽선으로 이루어져 있으나, 한양해서체는 한 개의 외곽선으로 이루어진다. 즉 맑은 고딕체는 토러스(torus)와 같은 위상을 가지고 한양해서체는 원(circle)과 같은 위상을 가진다. 토러스를 원으로 위상을 변화 시킬 때, 외부와 내부 컨투어가 이루는 토러스의 내부 공간 중 한 지점을 Split 하거나 내부 컨투

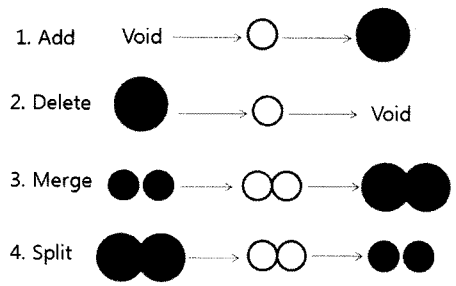


그림 13 위상 변화를 위한 네 가지 연산들

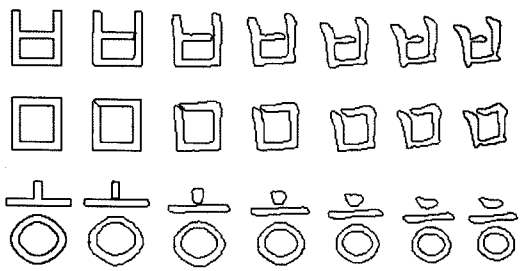


그림 14 위상이 다른 글립들의 모핑 예(Split 연산 사용)

어를 소멸시키는 Delete 연산을 사용할 수 있다. 그러나 우리는 한글 자소를 모핑할 때 중간 모양 또한 글자 모양이 되도록 유지시켜야 하므로, Delete 연산은 사용하지 않았다. 마찬가지로 Add 연산도 모핑 시 글자 모양에 불필요한 왜곡을 초래할 수 있으므로 사용하지 않고, Merge 및 Split 연산만을 사용하였다. Add, Delete 연산은 'ㅇ'에서 'ㅎ' 또는 그 반대 변환과 같이 서로 다른 자소 글립 간 모핑 시 사용될 수 있다. 그림 14의 ㅎ의 경우에는 N개의 컨투어에서 $N-1$개의 컨투어로 변화되는 경우이며, Split 연산을 사용하였다. Split 연산 시에는 획 분할 한 결과를 이용해서, 각 자소마다 적절한 분할점을 찾아 분할하고자 하는 외곽선의 개수(타겟 외곽선의 개수) 만큼 분할한다.

(1) Circle to Torus : Merge or Add
(2) Torus to Circle : Split or Delete (예) ㅁ, ㅂ, ㅍ
(3) N contours -> N-1 contours : Merge or Delete
(4) N contours -> N+1 contours: Split or Add (예) ㅊ, ㅌ, ㅍ, ㅎ

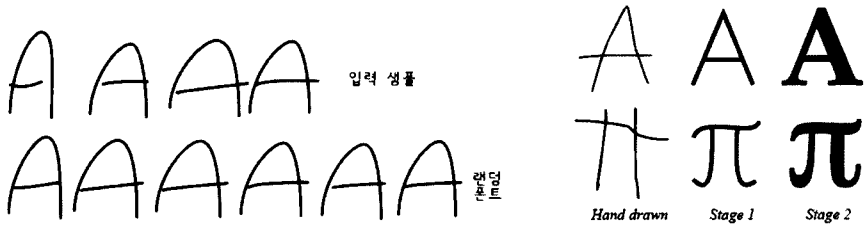
마지막으로, 모핑 결과 생성된 중간 글립들의 외곽선들을 트루타입폰트의 외곽선 표현 방식처럼 베지어 곡선으로 표현한다.

4. 결과 및 토의

자체 선정한 한글 글립 100개에 대해서 맑은 고딕체와 한양해서체 폰트에 대한 획 분할을 수행하였다. 맑은 고딕체는 외곽선 대응벡터 기반 방법으로 한양해서체는 외곽선 특징점 기반 방법을 사용하였으며, 초기 획 분할 정확도는 맑은 고딕체의 경우 98%, 한양해서체의 경우 87%였다. 맑은 고딕체의 'ㅅ', 'ㅈ', 'ㅊ'의 경우, 획 굵기 임계값 및 대응벡터의 각도 임계값이 달라짐에 따라 왼쪽 삐침획과 오른쪽 삐침획을 제대로 찾아내지 못하는 경우가 있었다. 이 경우, 외곽선 최소경계사각형 위치로 자소 분할 후, 자소에 따라 획 분할 시 사용되는 임계값들을 재설정해 주면, 정확한 획을 추출할 수 있다. 한양해서체와 같이 붓글씨체 또는 손글씨체의 경우, 모든 획을 정확히 추출하는 것이 쉽지 않다. 이 경우에도 연속되는 두 벡터 사이의 각도 및 획 굵기에 대한 임계값에 따라 정확도에 차이가 난다. 주된 오류는 곡률(curvature)이 너무 완만하여 획이 분할되어야 할 지점에서 분할되지 않는 경우였는데, 이런 오류는 획 분할 다음 과정인 자소 추출의 정확도에 크게 영향을 주지 않는 것으로 나타났다.

자소 추출 결과는 폰트 글립의 모양이 매우 정규화되어있기 때문에, 두 폰트 모두 100%의 정확도를 보였다. 그러나 획의 생략 및 변형이 자유로운 손글씨의 경우에는, 추출된 획 중에 어느 자소에도 포함되지 않는 획이 존재한다던가, 필요한 획이 추출되지 않은 경우 등이 있을 수 있다. 이 경우, 획의 위치와 다른 에 의해 어느 자소에 속할지를 추정해야 하므로, 릴렉세이션 레이블링(Relaxation Labeling) 또는 다른 확률적 접근이 필요할 수 있다.

폰트 모핑에 관한 연구로는 랜덤 폰트 생성에 관한 연구와 Smart Text 시스템이 있는데, 두 연구 모두 영어 알파벳에 적용한 것이다[7,8]. 이 방법들은 외곽선 정보가 아니라, 글자 또는 획의 골격 정보를 이용하여 소스 글자와 타겟 간의 대응점을 구한다. 랜덤 폰트 생성 연구의 경우, 그림 15(a)와 같이 여러 벌의 입력 샘플들 사이의 임의의 보간 및 외삽을 통해 조금씩 다른 폰트를 생성하는데, 입력 샘플들에 대한 획 정보가 미리 주어져야 한다. 그러나 한글은 글자의 개수가 영어 알파벳에 비해 매우 많고 자소 배치에 따라 자소 및 획의 모양 변화가 심하므로, 랜덤 폰트 연구에서와 같이 수동으로 획 정보를 주는 일은 매우 어렵다. 따라서 이 방법을 한글에 적용한다 하더라도, 한글의 조형적 특성을 고려한 획 및 자소의 분리가 필요하다. Smart Text 시스템은 온라인 글자 입력 시스템에서 손으로 입력한 문자를 시스템 폰트로 변환하기 위해, 그림 15(b)와 같이 먼저 글자의 골격을 추출한 뒤 소스와 타겟 골격 사이의 대응점을 구하여 모핑을 하고, 다음 단계로 획의 볼륨을 복원하는 모핑을 수행한다. 이 방법은 골격 이미지의 품



(a) (b)

그림 15 랜덤폰트 생성 시스템(a)과 Smart Text 시스템(b)의 입력 및 결과 예 [7,8]

질에 따라 모핑 품질이 달라지는데, 획의 굵기 변화가 심한 글씨체의 경우 골격 추출 결과가 사람이 인지하는 것과 달라 종종 좋지 않은 결과를 보일 가능성이 있다. 두 방법 모두 골격 기반의 모핑이기 때문에 생성된 중간 글립들이 소스와 타겟 폰트 사이의 세밀한 획의 굵기 변화를 반영하지 못한다.

그림 16, 17과 18은 맑은고딕체에서 한양해서체로의 한글 단어 글립의 모핑 결과를 보여준다. 비록 모양의 차이가 매우 큰 오브젝트 간의 모핑에 비해서 극적인 효과가 있는 것은 아니지만, 생성된 중간 글립들은 서로 다른 자소 간 또는 서로 다른 글자 간에 글씨체로서의 일관성을 가지므로, 하나의 글씨체로서 기능하기에 충분하다고 여겨진다. 특히 외곽선 모핑의 장점으로서 소스 글립에서 타겟 글립으로의 획의 굵기 변화가 자연스럽게 표현됨을 볼 수 있다.

그림 19는 '아리랑'이란 단어를 다양한 서체로부터 한양해서체로 모핑한 결과를 보여준다. 3절에서 설명한 바와 같이 먼저 획 및 자소 분할 후, 자소 간 외곽선 모핑을 수행하였다. 편지체, 소녀체, 아혜체의 경우 손글씨 모양의 서체이며, 산타할머니, 휴먼옛체, 수평선체의 경우 비교적 기하학적인 모양의 서체이다. 두 종류의 서체 모두 붓글씨 모양의 서체인 한양해서체로 자연스럽게 변화함을 볼 수 있다.

한글의 경우 자소의 배치에 따라 글자의 느낌이 달라

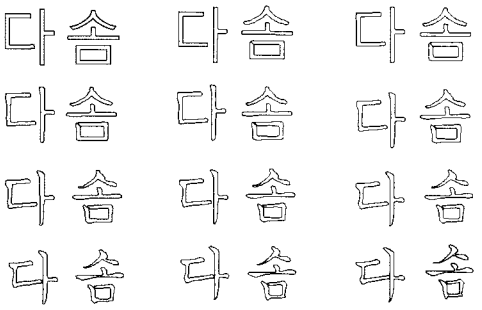


그림 16 한글 단어의 중간글립 생성 결과(10단계)

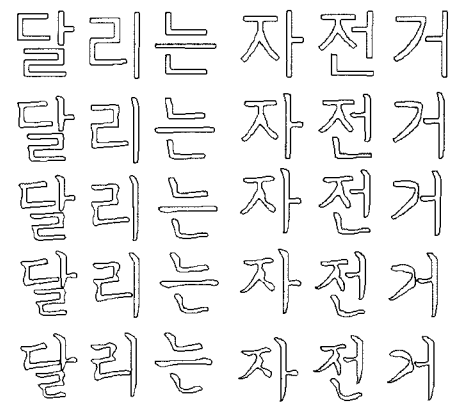


그림 17 한글 단어의 중간 글립 예

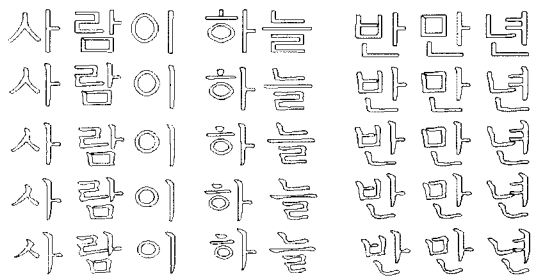


그림 18 한글 단어의 중간 글립
(위상이 다른 자소 간 모핑 예)

지므로 각 글씨체마다 자소의 배치가 약간씩 차이가 난다. 그래서 모핑 시에도 자소 배치의 변화를 고려해야 한다. 그런데 본 논문에서 제안하는 모핑 방법은 별도의 이동 연산이 없지만, 중간 글립의 자소들은 소스와 타겟 글립의 중간 지점에 자연스럽게 배치되는 것을 볼 수 있다. 경우에 따라서는 보다 다양한 글자 모양의 생성을 위해 자소 배치에 임의의 변화를 줄 수도 있다.

손글씨 모양의 서체들은 자소 추출 시 외곽선 분리가 종종 필요하다. 예를 들어, 소녀체 '랑'자의 경우, 원래 소스 글립의 초성 'ㄹ'과 중성 'ㅏ'가 하나의 외곽선으로

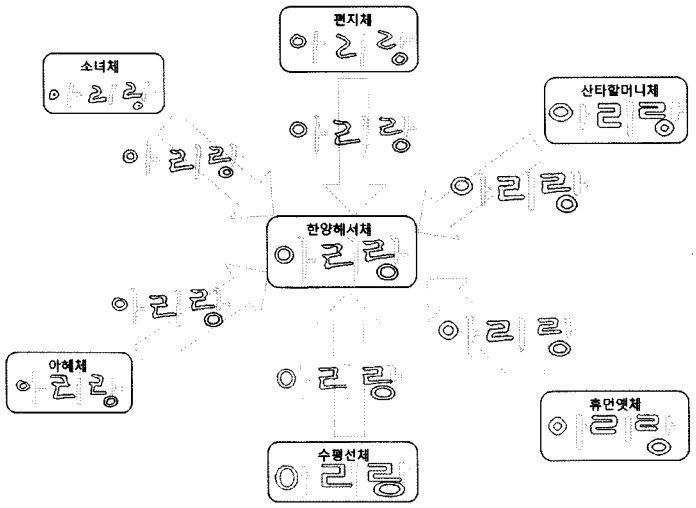
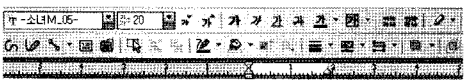


그림 19 다양한 서체로부터 한양해서체로의 모핑 결과

이루어져 있는데, 이를 분리하여 각각 타겟 자소 글립과의 모핑을 적용하였다.

그림 19의 중간 글립들을 비교해보면 다양한 한글 폰트 간의 중간 폰트들은 소스와 타겟 두 폰트의 특징을 반영하면서도 다른 폰트와 구별되는 느낌을 가지는 것을 볼 수 있다. 그림 20의 (a)는 이러한 다양한 기존 폰트에서 한양해서체로의 중간 폰트들을 트루타입폰트 형식으로 만들어 한글워드프로세서에서 사용한 예를 보여준다. 그림 20의 (b)는 그 중 소녀체와 휴먼옛체의 모핑 과정상의 모든 중간 폰트를 생성한 결과를 보여준다. 이는 제안된 방법을 통해 사용자의 선택에 따라 기존에 보유하고 있는 폰트들을 활용하여 새로운 폰트를 생성할 수 있음을 보여준다.



아리랑
아리랑
아리랑
아리랑
아리랑
아리랑

	소녀체	휴먼옛체
0	아리랑	아리랑
.1	아리랑	아리랑
.2	아리랑	아리랑
.3	아리랑	아리랑
.4	아리랑	아리랑
.5	아리랑	아리랑
.6	아리랑	아리랑
.7	아리랑	아리랑
.8	아리랑	아리랑
.9	아리랑	아리랑
1	아리랑	아리랑

(a) (b)

그림 20 워드프로세서에서의 중간 폰트 사용 예

한편, 그림 21은 모핑 시 각 컴포넌트별로 독립적인 모핑 단계를 적용함으로서 보다 다양한 한글 글립을 생성할 수 있음을 보여준다. 소스 폰트인 독도체를 0, 타겟 폰트인 휴먼둥근헤드라인체를 1이라고 했을 때, 일반적으로 중간 글립이란 그림 21의 맨 윗줄의 중앙에 있는 글립처럼 초성, 중성, 종성 컴포넌트의 모핑 정도가 모두 같다. 그런데 이때 각 컴포넌트별로 다르게 모핑 단계를 적용하면 그림 21의 맨 아랫줄처럼 다양한 글립을 얻을 수 있다. 예를 들어, 맨 아랫줄의 가장 왼쪽 글립은 초성은 0.7, 중성은 0.5, 종성은 0.2 단계의 모핑 결과를 조합한 것이다.

5. 결론 및 향후 과제

본 논문에서는 한글 트루타입폰트 글립의 조형적 특징을 기하학적으로 분석하고 코드 정보와 연결함으로써, 한글 글립을 단순히 외곽선의 집합에서 스타일(style)에 대한 의미적인(semantic) 정보를 추출하였다. 즉 각 글자는 독립적인 자소로 구성되고, 각 자소들은 또다시 기본 획들로 구성되는 계층적 구조를 완성하고, 각 자소 및 획의 모양(shape)과 배치(layout)에 대한 정보를 추출하였다. 그리고 스타일이 다른 두 개의 폰트의 획과 자소를 추출한 뒤, 각 자소별로 모핑을 함으로써, 두 폰트 사이의 중간 폰트를 생성할 수 있음을 보였다. 생성된 중간 글립은 자소 간 및 글자 간에 일관성을 가지므로, 각 중간 글립들로 단어 또는 문장을 만들어 보았을 때, 비교적 자연스러운 결과를 얻을 수 있었고 가독성이 떨어지지도 않았다. 본 논문은 기존의 폰트들을 활용하여 다양한 중간 폰트를 자동으로 생성하거나 새로운 폰트 생성 시 기존 폰트들과의 유사도를 테스트 할 때에

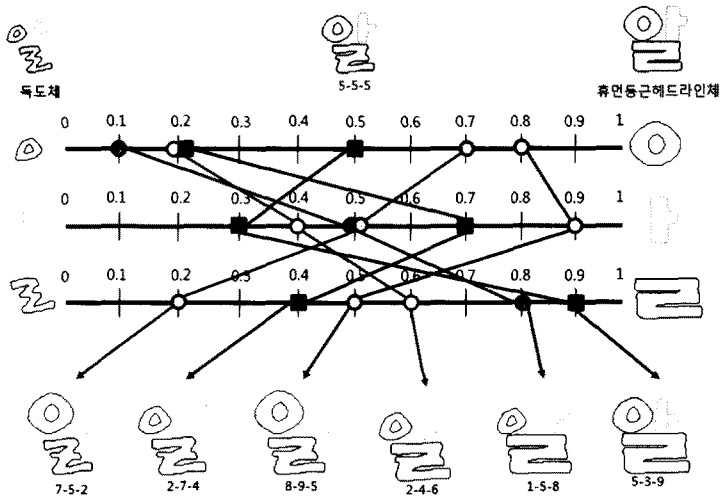


그림 21 컴포넌트별 독립적인 모핑 단계 적용

도 활용될 수 있다. 또한 한글 완성형 폰트에서는 '똠방각하'의 '똠'자와 같은 글자들을 표현하지 못하는데, 본 논문의 결과를 응용하면 완성형 폰트를 분석하여 자소를 분할 한 뒤, 조합형 폰트처럼 사용할 수 있도록 완성형 2,350자를 제외한 나머지 글립의 모양들을 채워 넣을 수 있다. 즉 본 논문에서 제안한 글립 분석 및 모핑 방법은 다양한 한글 폰트 개발을 위한 도구 제작에 사용될 수 있다. 또 생성된 중간 폰트는 스플라인 피팅 및 트루타입(TrueType) 형식화 후 운영체제에서 일반 시스템폰트와 같이 사용될 수 있다.

본 논문은 사람이 직접 쓴 손글씨 이미지와 폰트 간의 모핑을 통해, 기존 폰트 스타일의 개인 손글씨 폰트를 생성하는 데 응용될 수 있다. 그림 22는 맑은 고딕체 폰트에서 손글씨로의 모핑 결과 및 폰트 생성을 위한 베지어 곡선 근사화 과정을 보여준다. 그림 23은 서로 다른 글자('타', '개')의 자소 'ㅌ'과 'ㅐ'를 결합하여 사용자가 입력하지 않은 '태'라는 글자를 생성한 예를 보여준다. 개인의 손글씨 이미지를 이용하여 폰트를 생성하는 작업은 매우 매력적인 작업이지만, 사용자의 입력이 매우 많이 필요하다는 단점을 지닌다. 이 논문에서의 획

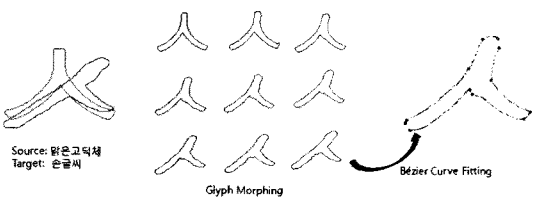


그림 22 기존 폰트에서 손글씨로의 'ㅅ' 글립 모핑과 베지어 곡선화

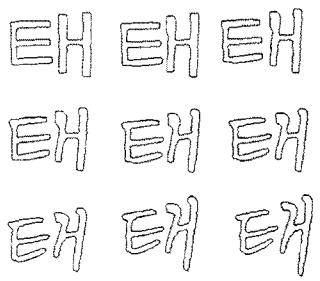


그림 23 손글씨에서의 자소 추출 및 모핑

분할 및 자소 추출과 자소 모핑 과정을 손글씨에 적용 가능하도록 발전시키고, 자소 간 배치 및 글자의 전체적인 기울기와 같은 스타일 데이터를 활용하면, 적은 양의 손글씨 입력으로도 폰트로 사용하기에 필수적인 글자들을 생성해 낼 수 있으리라 예상된다.

한글의 각 자소는 네모, 원, 세모꼴과 같은 기본 도형의 형태를 가지므로, 자연 또는 생활 속의 다양한 사물을 담은 사진 속에는 한글 자소와 닮은꼴의 오브젝트가 많다. 그래서 다양한 사진 이미지 속에서 한글 자소와 닮은꼴의 오브젝트를 찾고, 이들을 조합하여 글자 모양을 만든다면, 이미지를 활용한 다양한 한글 글자들을 생성해 낼 수 있다. 이러한 시도도 영어 알파벳에 비해 서체 디자인에서의 다양성이 많이 부족한 한글 타이포그래피의 발전에 기여할 수 있으리라 생각된다.

각 글자는 또한 고유의 이름 또는 음운을 가진다. 이는 모핑 또는 다른 형태의 글자 애니메이션 시, 소리에서 느껴지는 이미지(예를 들어, 강함과 약함, 부드러움과 거침 등)로 글자의 모양을 변형하거나, 글자의 모양이 커지거나 작아짐에 따라 소리의 강약이 조절되도록

동기화하는 등의 설정을 함으로써, 애니메이션의 효과를 증대시킬 수 있다. 이와 같이 글자 모양에 감성을 적용한 예는 최근 캘리그래피라고 불리는 서예를 응용한 디자인 분야 및 영화 타이틀 등에서 다양한 형태를 선보이고 있다.

참 고 문 헌

[1] 손글씨용 서체('봄날'), http://yoonfont.co.kr/, 윤디자인연구소.

[2] 손글씨용 서체('석금호체'), https://www.sandoll.co.kr/, 산돌커뮤니케이션.

[3] 장인 서체, http://www.smfont.com/, 직지소프트.

[4] 스타 폰트, http://www.hanyang.co.kr/, 한양정보시스템.

[5] Fontographer, http://www.fontlab.com/font-editor/fontographer/, FontLab Ltd.

[6] 폰트의 제작과정, http://www.onhangeul.com/, 온라인 한글 박물관.

[7] Luc Devroye and Michael McDougall, "Random fonts for the simulation of handwriting," *Electronic Publishing: Origination, Dissemination, and Design*, vol.8, no.4, pp.281-294, 1995.

[8] James Arvo and Kevin Novins, "Smart Text : A Synthesis of Recognition and Morphing," *AAAI Smart Graphics Symposium*, March 2000.

[9] Apple Computer, Inc. The TrueType Font Format Specification, Version 1.0, 1990.

[10] Microsoft Typography, http://www.microsoft.com/typography/.

[11] Adobe Systems, Inc. *Adobe Type 1 Font Format*. Addison-Wesley, 3rd edition, 1993.

[12] Microsoft Corporation. OpenType specification version 1.4. Available from http://www.microsoft. com/typography, 2002.

[13] Unicode, http://www.unicode.org

[14] 이기성, 네모체 음절 1만 1172개의 한글디자인에 관한 연구, *한국출판학술지 제34권 제1호*, 한국출판학회, pp.221-263, 2008.

[15] 장현규, 구상옥, 정순기, "트루타입폰트 기반 한자 자동 획 분할 및 자동 획순 부여", *한국컴퓨터그래픽스학회 논문지*, vol.11, no.3, pp.10-18, 2005.

[16] 구상옥, 장현규, 정순기, "모바일 한자 학습 애니메이션 생성", *한국정보과학회 논문지*, vol.33, no.12, pp. 894-906, 2006.

[17] 이진수, 권오준, 방승양, "개선된 자소 인식 방법을 통한 고인식률 인쇄체 한글 인식", *한국정보과학회 논문지*, vol.23, no.8, pp.841-851, 2005.

[18] E. Carmel and D. Cohen-Or, Warp-guided object-space morphing, *The Visual Computer*, vol.13, pp. 465-478, 1997.

[19] V. Ranjan and A. Fournier, "Matching and interpolation of shapes using unions of circles," In *Proceedings of EUROGRAPHICS*, pp.129-142, 1996.

[20] T.W. Sederberg and E. Greenwood, "A physically based approach to 2-D shape blending," In *Proceedings of SIGGRAPH*, pp.25-34, 1992.

[21] G. Gong and B. Parvin, "A new regularized approach for contour morphing," *Proceeding of CVPR*, pp.458-463, 2000.

[22] X. Jiang, H. Bunke, K. Abegglen, A. Kandel. "Curve Morphing by Weighted Mean of Strings," In *Proceedings of the Sixteenth Conference on Pattern Recognition (ICPR 2002)*, vol.4, pp.192-195. IEEE Press, 2002.

[23] H. Bunke, X. Jiang, K. Abegglen and A. Kandel, On the Weighted Mean of a Pair of Strings, Pattern Analysis & Applications, vol.5, no.1, pp. 23-30, 2002.

구 상 옥

2001년 경북대학교 컴퓨터공학과 졸업(공학사). 2003년 경북대학교 컴퓨터공학과 석사 졸업(공학석사). 2004년 3월~현재 경북대학교 컴퓨터공학과 박사과정 관심분야는 Computer Graphics, Computer Vision, Information Visualization, Augmented Reality

정 순 기

1990년 경북대학교 컴퓨터공학과 졸업(공학사). 1992년 한국과학기술원 전산학과(이학석사). 1997년 한국과학기술원 전산학과(공학박사). 1997년~1998년 University of Maryland, Research Associate. 2001년~2002년 IRIS, University of Southern California, Research Associate. 1998년~현재 경북대학교 컴퓨터공학과 부교수. 1999년~현재 (주)아이디스 기술고문. 관심분야는 Virtual Reality, Artificial Intelligence, Computer Vision, Image Processing, Computer Graphics