모바일 한자 학습 애니메이션 생성

(Animation Generation for Chinese Character Learning on Mobile Devices)

구 상 옥 [†] 장 현 규 [††] 정 순 기 [†††]
(Sang Ok Koo) (Hyun Gyu Jang) (Soon Ki Jung)

요 약 모바일 기기의 성능 및 화면, 무선 네트워크의 속도 등의 제약으로 모바일 컨텐츠 개발에는 많은 어려움이 있다. 단순히 유선 웹상에서 기존에 서비스 되던 컨텐츠의 가시적인 축소만으로는 양질의 컨텐츠 제작이 어렵다. 빠르게 변화하는 모바일 컨텐츠 시장에 적응하기 위해서는 컨텐츠 특성에 최적화된 데이타 표현 기법 및 저작 도구의 개발이 이루어져야 한다.

본 논문에서는 모바일 기기 상에서의 한자 학습을 위한 적은 용량의 모바일 컨텐츠 및 저작 도구를 개발하였다. 본 연구에서 개발한 모바일 컨텐츠는 단순히 한자 이미지와 설명 정보를 보여주는 것이 아니라, 한자 획순으로 붓으로 쓰는 것과 같은 애니메이션 효과를 줄 수 있다. 또한 저작 도구는 사용자가 그래픽이나 한자, 모바일 프로그래밍에 관한 전문가가 아니더라도 쉽고 빠르게 컨텐츠를 생성할 수 있는 개발 환경을 제공한다.

본 논문은 트루타입 폰트로부터 글자 모양을 획득하여, 간단한 사용자 입력으로 획 분할 및 획 순서 정보를 얻고, 자동으로 획의 방향을 추출, 각 획마다 붓으로 쓰는 효과의 애니메이션을 생성한다. 다음으로 모바일 기기에서의 효율적인 글자 애니메이션을 위해 애니메이션 데이타를 압축한다. 본 논문은 한자뿐 아니라, 한글 또는 다른 형태의 그래픽에도 이용될 수 있으며, 향후 획 분할 및 획 순서 결정을 자동화하는 방법을 연구하고자 한다.

키워드 : 모바일 컨텐츠, 트루타입 폰트, 한자 학습, 글자 애니메이션, 데이타 압축

Abstract There are many difficulties to develop a mobile contents due to many constraints on mobile environments. It is difficult to make a good mobile contents with only visual reduction of existing contents on wire Internet. Therefore, it is essential to devise the data representation and to develop the authoring tool to meet the needs of the mobile contents market.

We suggest the compact mobile contents to learn Chinese characters and developed its authoring tool. The animation which our system produces is realistic as if someone writes letters with pen or brush. Moreover, our authoring tool makes a user generate a Chinese character animation easily and rapidly although she or he has not many knowledge in computer graphics, mobile programming or Chinese characters.

The method to generate the stroke animation is following: We take basic character shape information represented with several contours from TTF (TrueType Font) and get the information for the stroke segmentation and stroke ordering from simple user input. And then, we decompose whole character shape into some strokes by using polygonal approximation technique. Next, the stroke animation for each stroke is automatically generated by the scan line algorithm ordered by the stroke direction. Finally, the ordered scan lines are compressed into some integers by reducing coordinate redundancy. As a result, the stroke animation of our system is even smaller than GIF animation. Our method can be extended to rendering and animation of Hangul or general 2D shape based on vector graphics. We have the plan to find the method to automate the stroke segmentation and ordering without user input.

† 학생회원 : 경북대학교 컴퓨터공학과
sokoo@vr.knu.ac.kr
†† 비 회 원 : 경북대학교 컴퓨터공학과
serion@vr.knu.ac.kr
††† 종신회원 : 경북대학교 컴퓨터공학과 교수
skjung@knu.ac.kr
논문접수 : 2005년 11월 8일
심사완료 : 2006년 10월 11일

Key words : Mobile Contents, TrueType Font, Chinese Character Learning, Character Animation, Data Compression

1. 서 론

급속도로 확대되고 있는 모바일 시장에서는 특화된 모바일 기술과 더불어 수요자를 흡입할 수 있는 양질의 컨텐츠를 필요로 하고 있다. 특히 교육용 컨텐츠의 경우 이미 다양한 형태의 서비스를 통해 많은 부가가치를 창출하고 있다. 유선 인터넷상에서의 교육용 컨텐츠 개발은 최근 비약적인 성장을 거듭해 왔다. 하지만 유선 인터넷상에서의 대용량의 컨텐츠를 모바일 기기에 그대로 적용하는 것은 모바일 단말기의 성능 상의 제약 및 무선 통신 상의 제약 때문에 불가능하다. 이 때문에 모바일 컨텐츠는 기존의 유선 인터넷상에서의 서비스와 별도로 새롭게 개발되고 있는 실정이다.

모바일 컨텐츠 개발은 일반 웹이나 PC용 컨텐츠에 비해 많은 제약이 존재하므로. 모바일 컨텐츠 개발 시에는 모바일 기기의 속도 및 성능, 화면(LCD) 크기, 그리고 컨텐츠의 용량과 같은 사항을 고려해야 한다. 첫째, 모바일 기기는 경량의 메모리 및 성능이 낮은 프로세서를 가지므로 많은 연산을 필요로 하거나, 메모리를 많이 소모하는 작업은 최대한 피해야 한다. 둘째, 모바일 기기의 작은 LCD화면 크기 및 해상도, 지원 색상에 적합한 컨텐츠를 제공해야 한다. 셋째, 다운로드 속도 및 전송량을 고려하여 컨텐츠의 전체 크기를 적절히 유지하여야 한다. 특히 컨텐츠의 용량을 줄이면서도, 양질의 서비스를 최대한 신속하게 제공하는 것이 중요하다. 그래서 컨텐츠의 데이타 용량을 줄이기 위한 방법이 많이 연구되고 있다. 마지막으로 모바일 컨텐츠는 빠르게 제작 및 갱신되는 것이 중요하기 때문에 컨텐츠 개발 시간이 최대한 짧아야 한다. 따라서 컨텐츠 개발 시간에 드는 시간과 노력을 줄이면서도 양질의 일관된 서비스를 가능케 하는 컨텐츠 저작 도구의 개발이 요구된다. 특히 반복 작업이나 까다로운 그래픽 수작업이 많은 경우에 적합한 저작 도구를 사용함으로써 개발 시간을 상당부분 단축할 수 있다.

최근 중국어 및 한자 학습에 대한 수요가 증가하면서, 한자 학습 컨텐츠의 종류와 양이 크게 늘어나고 있다[1,2]. 한자는 표의 문자이기 때문에, 각 글자의 모양과 소리와 뜻을 연계하여 암기하여야 한다. 암기를 위해서는 글자의 획순으로 여러 번 글자를 쓰면서 모양을 익히고, 동시에 소리와 뜻을 입으로 소리 내어 연습하는 방법이 가장 널리 이용되어 왔으며, 학습 효과도 큰 것으로 알려져 있다. 한자 암기를 위해서는 반복적인 학습이 중요하므로, 항상 휴대하고 다니는 모바일 기기를 이용한 학습이 효과적이라고 말할 수 있다. 기존의 유선 인터넷상의 한자 학습 컨텐츠는 이미지, GIF 애니메이션, FLASH 애니메이션 등 각종 대용량 멀티미디어 자료 형태로 서비스되고 있다. GIF 애니메이션은 그대로 모바일 환경에 적용하기에는 용량이 너무 크며, FLASH 애니메이션은 FLASH 플레이어를 설치할 수 있는 모바일 기기에서만 이용이 가능하다. 또한 GIF나 FLASH 애니메이션의 제작과정은 수동으로 행해져야 하므로 컨텐츠 제작자의 많은 시간과 노력을 필요로 한다. 그래서 기존에 모바일 기기에서 서비스되고 있는 한자 학습 컨텐츠는 대부분 텍스트 위주의 내용에 단조로운 글씨체의 한자 이미지가 제공되는 방식이다. 모바일 상의 한자 학습 컨텐츠의 경우 학습 효과를 높이기 위해서는 획순을 보여주는 애니메이션이 가미되면 좋겠지만, 유선 인터넷에서 주로 사용하는 GIF 애니메이션 형태로 제작하였을 경우에는 데이타양이 매우 커지게 된다. 데이타의 양이 커지면, 다운로드 속도가 느려지고, 전송 패킷 양에 따라 요금이 부과되는 무선 인터넷의 실정 때문에 실제적인 모바일 서비스가 불가능하다. 따라서 모바일 기기에서 기존의 애니메이션 보다 훨씬 경량의 데이타를 가지고 효과적으로 글자 이미지와 획 애니메이션을 표현할 수 있는 기술이 필요하다.

본 논문에서는 모바일 기기에서 효율적인 렌더링이 가능한 적은 용량의 한자 학습 컨텐츠를 위한 한자 애니메이션 생성 방법을 제안한다. 컨텐츠 개발 시간을 단축시키기 위해 기존의 폰트를 이용하여 글자의 모양 정보를 획득하였고, 직관적인 사용자 인터페이스를 가진 저작 도구를 개발하였다. 본 논문의 2장에서는 폰트의 종류 및 본 논문에서 사용한 트루타입 폰트에 대해 설명한다. 3장에서는 트루타입 폰트로부터 한자 애니메이션을 생성하기 위한 방법 및 과정을 자세히 기술하고, 아울러 애니메이션 데이타의 압축방법에 대해 설명한다. 4장에서는 한자 학습 컨텐츠 저작 도구 및 모바일 기기 상에서의 한자 렌더링에 대해 보여준다. 5장은 제안한 방법에 의해 개발된 한자 애니메이션과과 기존의 애니메이션 간의 데이타 용량을 비교하는 실험과 그 결과를 보여주고, 6장에서 결론을 맺는다.

2. 연구 배경

본 논문에서는 다양한 글자 모양 정보를 자동으로 획득하기 위해 디지털 폰트(digital font)를 이용한다. 이 장에서는 폰트의 종류에 대해서 알아보고, 본 논문에서

사용하는 트루타입 폰트(TrueType font)의 특징에 대해 보다 자세히 살펴본다.

2.1 폰트(Font)

글자의 형상 정보를 획득하기 위한 한 가지 방법은 운영체제 또는 각종 소프트웨어에서 제공되는 글자 형상 데이타의 집합인 디지털 폰트를 이용하는 것이다. 폰트는 파일 형식대로 개인이 직접 폰트를 제작 및 배포할 수도 있어서 폰트 자체가 하나의 중요한 디지털 컨텐츠이기도 하다[3-5].

현재 각 언어마다 수많은 디지털 폰트가 제작되어 사용되고 있는데, 우리나라에서는 대표적으로 신문이나 논문과 같은 문서에서 한글, 한자, 영문 알파벳을 모두 사용하므로 그림 1과 같이 세 언어를 모두 지원해주는 폰트가 주로 사용되고 있다.

굴림체	바탕체	한양해서체
書體	書體	漢陽楷書體
Gulim	Batang	Hanyang Haeseo

그림 1 한글, 한자, 알파벳을 지원하는 폰트의 예

폰트는 글자의 형상을 표현하는 형태에 따라 크게 비트맵 폰트(bitmap font 또는 raster font)와 윤곽선 폰트(outline font or scalable font), 그리고 구조적 폰트로 나뉜다.

- 비트맵 폰트는 글자를 점의 집합으로 표현한 것으로 글자의 내부에 해당하는 부분은 1로, 외부를 0으로 표현하여 한 글자에 해당하는 사각형을 점 행렬(dot matrix)로 저장하고 있다. 비트맵 폰트는 복잡한 연산을 거치지 않기 때문에 빠르게 표시되어 화면 표시용이나 프린터용으로 많이 쓰인다. 그러나 해상도가 낮은 출력장치에는 글자의 비트맵 크기가 작으므로 효과적이지만, 반대의 경우 비트맵의 크기가 커져야 하며 글자의 확대, 축소 또는 기울임을 하게 되면 윤곽선이 거칠어지고 형태가 일그러진다. 그래서 글자의 크기(도트 수) 별로 글자 데이타를 준비해야 하므로, 해상도가 높은 시스템이나 다양한 크기의 글자를 제작해야 할 때, 제작에 드는 시간 및 데이타의 양이 많아지게 된다.
- 윤곽선 폰트는 3차 스프라인(cubic spline), 베지어(bézier), 2차 B-스프라인(quadratic B-spline) 등의 함수 곡선을 가지고 글자의 외곽선(contour)을 나타낸다. 이를 화면에 표시하거나 프린트 할 경우는 도트 정보로 래스터라이징(rasterizing)하는 작업이 필요하므로 출력의 속도가 느려지게 된다. 하지만, 하나의 글자 정보만으로 임의의 확대, 축소 또는 기울임을 형태의 일그러짐 없이 처리할 수 있고, 또한 높은 품질의 글자를 만들어 낼 수 있다. 현재 사용되고 있는 윤곽선 폰트로는 3차의 베지어 곡선을 사용하여 글자의 외형을 나타내는 포스트스크립트 폰트(postscript font)와 2차의 B-스프라인 곡선을 사용하는 트루타입 폰트가 있다. 트루타입 폰트는 현재 윈도우, 매킨토시, 리눅스 등의 각종 운영체제에서 널리 사용되고 있다[5,6]. 우리는 기존에 만들어진 트루타입 폰트를 이용함으로써 다양한 크기와 모양의 글자 정보를 쉽게 얻을 수 있다.
- 구조적 폰트는 획의 굵기와 같이 글자의 모양에 영향을 주는 여러 가지 인자들을 정의함으로써 글자의 모양을 자동으로 계산하여 표현 하는 방식이다[7,8]. 가장 대표적인 구조적 폰트인 메타폰트(metafont) 시스템은 영문 폰트를 개발하기 위한 일종의 프로그래밍 언어와 그 언어의 인터프리터로 구성되어 있다. 글자의 모양을 붓이 지나간 자취로 정의하고, 붓의 모양과 그것이 지나간 궤적을 프로그램으로 표현하기도 한다.

지금까지 모바일 기기에서 제공되었던 폰트는 비트맵 폰트이다. 하지만, 한자 폰트는 글자 수가 많아 제작 시간이 많이 걸리고 저장 공간을 많이 필요로 하므로, 우리나라에서는 일부 PDA 및 사양이 매우 높은 휴대폰에서만 한자 폰트가 지원된다. 비록 한자에 대한 비트맵 폰트가 모바일 기기에서 제공된다 하더라도 12에서 18pt 정도의 매우 작은 크기의 비트맵으로 제공될 것이므로, 컨텐츠 제작을 위해 임의의 크기로 확대할 때 글자의 외곽선 모양이 거칠게 변하게 된다. 구조적 폰트는 인터프리터로 글자의 모양을 렌더링 할 때, 많은 계산을 요구하므로, 저사양의 모바일 기기에서 사용되기에 적합하지 않다. 본 논문에서는 학습 컨텐츠 제작자가 임의의 크기에서 글자의 모양을 다루고, 임의의 크기로 컨텐츠를 저장할 수 있도록 하기 위하여, 윤곽선 폰트의 일종인 트루타입 폰트를 이용한다.

2.2 트루타입 폰트(TrueType Font)

트루타입 폰트 파일은 "*.TTF" 또는 "*.TTC"라는 확장자를 가지며, 운영체제에서 지원해 주는 기본 폰트와 워드 프로세서와 같은 응용프로그램에서 제공해 주는 확장 폰트가 일반적으로 많이 사용된다. 그림 1 또한 트루타입 폰트를 윈도우즈용 워드 프로세서 프로그램인 '흔글'에서 11pt의 크기로 래스터라이징 한 결과이다.

모든 폰트는 글립(glyph)이라고 불리는 글자 모양에 대한 정보를 가지고 있다. 트루타입 폰트에서 한 글자안의 각각의 독립된 폐곡선(contour)은 점의 열로 표현된다. 각 점들은 크놋(knot)으로 불리는 곡선 위의 점(on-curve) 또는 곡선 밖의 점(off-curve)으로 분류된다. 즉 트루타입에서는 글자의 외곽선을 다음과 같이 정의한다[6,9,10].

- 곡선 위의 점 두 개가 연속일 때 두 점은 직선으로 연결된다.
- 하나의 곡선 밖의 점이 두 개의 곡선 위의 점 사이에 있을 때, 세 점은 2차의 베지어 곡선 조각을 위한 조절점(control point)으로 다루어진다.
- 두 개의 곡선 밖의 점이 인접할 때, 그것들을 연결하는 선분의 중간점이 암묵적으로 그것들 사이의 곡선 위의 점으로 다루어진다.

트루타입의 곡선들은 2차의 B-스프라인들로서, 각 스프라인은 연속적인 2차의 베지어 곡선과 같다. 각 베지어 곡선은 세 개의 조절점으로 정의된다. 만약 베지어 곡선의 세 조절점이 (A_x,A_y), (B_x,B_y) 그리고 (C_x,C_y) 라면, t가 0에서 1까지 변화함에 따라 생성되는 곡선 위의 모든 점 (p_x,p_y)는 식 (1)과 같이 나타낼 수 있다.

$$p_x = (1-t)^2 \cdot A_x + 2t(1-t) \cdot B_x + t^2 \cdot C_x \quad (1)$$
$$p_y = (1-t)^2 \cdot A_y + 2t(1-t) \cdot B_y + t^2 \cdot C_y$$

시각적으로 표현하면, 위 베지어 곡선은 두 곡선 위의 점 A, C 사이를 지나가며, 곡선 밖의 점 B가 자신의 방향으로 선분 AC를 당기는 모양이 된다. 예를 들어, 그림 2의 세 점 A, B, C는 식 (1)에서의 세 점에 각각 대응한다.

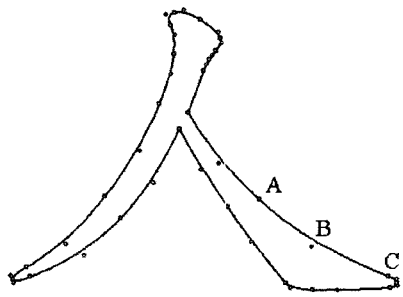


그림 2 트루타입 외곽선의 예

트루타입 폰트 파일은 일련의 연결된 테이블들로 구성된다[4-6]. 'cmap' (character to glyph mapping) 테이블은 글자 코드에 해당하는 글립 식별자(glyph id)를 찾는데 사용된다. 'cmap' 테이블은 다시 맵핑 서브테이블(mapping subtables)들로 구성되는데, 이를 통해 서로 다른 운영체제에서도 같은 폰트 파일을 사용할 수 있게 해준다. 'glyf' (glyph outline) 테이블은 글립을 정의하는 데이타, 즉 각각의 폐곡선들을 나타내는 조절점들을 포함한다. 'glyf'테이블은 그 자체가 하나의 글립인 단순 글립뿐 아니라, 다른 글립들로 구성되는 복합 글립도 지원한다. 이 외에 글립 이름을 저장하는 'name' 테이블, 글자와 글자가 만났을 때 두 글자간의 간격을 시각적으로 일정하게 유지하는 커닝(kerning)에 관한 정보를 저장하는 'kern' 테이블, 힌팅(hinting)에 관한 정보를 가지는 'cvt' (control value table) 등 총 24개의 테이블들이 있다. 본 논문에서는 'cmap' 테이블과 'glyf' 테이블을 이용하여 글립 데이타를 얻어온다.

3. 획 정보 추출과 획 애니메이션 생성

본 논문에서는 트루타입 폰트를 이용해 한자의 외곽선 정보를 획득한다. 외곽선 정보와 사용자 입력을 바탕으로 한자의 획을 분할하고, 각 획의 방향을 결정한 다음, 한자를 붓으로 쓰는 것과 같은 효과의 애니메이션을 생성한다. 이 장에서는 외곽선 정보의 획득부터 애니메이션 생성까지의 과정에 대해 자세히 설명한다.

3.1 글자 외곽선 정보 획득

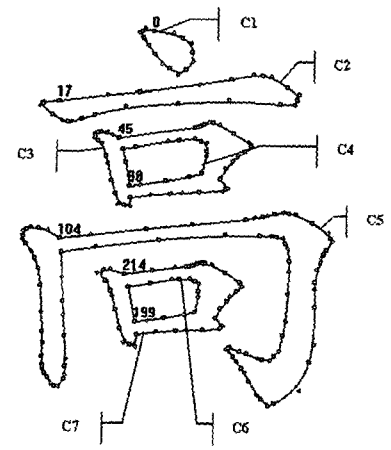
트루타입 폰트의 'glyf' 테이블로부터 글자의 외곽선 정보, 즉 베지어 곡선의 조절점 좌표들을 읽어 온다. 그림 3은 트루타입 폰트로부터 읽어온 글자 정보의 예로서 한양해서체를 폰트 패밀리(font family)로 가지는 '高(높을 고, 0x9ad8)'자의 글립 데이타를 보여준다. 본 논문에서는 윈도우즈가 제공해주는 폰트 중에서 가장 붓글씨와 유사한 서체의 한자 폰트를 지원하고 있는 한양해서체를 기본으로 사용한다. 한양해서체의 '高(고)'자는 총 7개의 폐곡선으로 구성되며, 조절점은 총 256개이다. 조절점의 (x,y)좌표 다음의 이진수는 그 조절점이 곡선 위의 점인지(1), 곡선 밖의 점인지(0)를 나타낸다.

그림 4의 (a)는 그림 3의 조절점들을 가지고 외곽선 렌더링 한 결과이다. 이 그림에서 Cn은 각 폐곡선을 나타내고, 각 폐곡선마다 명시된 숫자는 폐곡선이 시작하는 조절점의 인덱스를 나타낸다. 글자의 크기 변경이나 기울임은 조절점 좌표들을 변환(transform)함으로써 간단히 처리할 수 있다. 위치가 변환된 조절점들을 이용하여 베지어 곡선을 그림으로써, 크기나 각도 변화에 관계

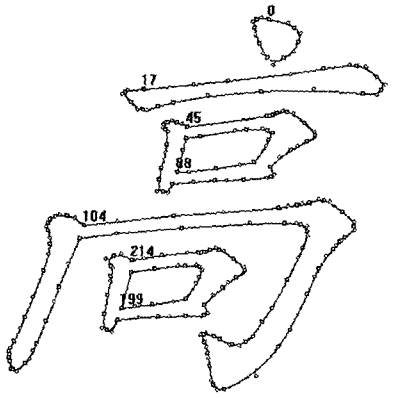
Font family: HYhaeseochar
Code: 0x9ad8
Num of contours: 7
Num of control points: 256

Contour 1 (point id 0~16)	Contour 2 (point id 17~44)	...	Contour 7 (point id 214~255)
(1905,3587)1	(2106,3562)0		(1869, 443)0
(1905,3587)1	(879,2825)1		(1606, 947)1
(1814,3613)0	(879,2816)1		(1606, 947)1
(1779,3587)1	(874,2816)1		(1462,1011)0
(1746,3566)0	(700,2816)0		(1391, 990)1
(1787,3507)1	(689,2797)1		(1320, 965)0
(1838,3426)0	(669,2775)0		(1412, 879)1
(1909,3337)1	(810,2689)1		(1433, 862)0
(1996,3227)0	(919,2624)0	...	(1461, 765)1
(2059,3180)1	(984,2624)1		(1478, 710)0
(2154,3104)0	(1030,2619)0		(1515, 566)1
(2230,3176)1	(1136,2654)1		(1569, 358)1
(2304,3244)0	(1279,2703)1		(1610, 206)0
(2304,3354)1	⋮		⋮
(2288,3452)0	(1404,2884)0		(1917, 998)1
(2197,3502)1	(887,2829)1		(1724, 968)0
(2106,3562)0			

그림 3 트루타입 폰트의 글립 정보 예 (高, 한양해서체)



(a)



(b)

그림 4 글립 렌더링 예 (高, 한양해서체): (a)는 글자의 각 폐곡선을 이루는 조절점과 외곽선을, (b)는 20도 각도로 기울인 글자의 모양을 보여준다.

없이 글자의 형태가 유지된다. 그림 4의 (b)는 (a)를 20도 각도로 기울였을 때의 렌더링 결과이다.

본 논문에서 제안하는 컨텐츠 저작도구는 기존의 트루타입 폰트를 활용함으로써 한자의 외곽선 정보를 획득하므로 픽셀 드로잉과 같이 일일이 글자의 모양을 편집할 필요가 없다. 따라서 컨텐츠 개발 시간이 단축되고, 추가적인 노력 없이 원하는 크기의 글자를 생성해 낼 수 있다.

3.2 획 추출 알고리즘

획(畫, stroke)은 글씨나 그림에서, 펜이나 붓으로 그은 줄이나 점 같은 것을 말하여, 그 줄이나 점을 세는 단위를 지칭하기도 한다. 특히 한글이나 한자는 여러 개의 획으로 구성되며, 글자의 획 및 획순(또는 필순)을 익히는 것이 글을 배우는 데 있어서 매우 중요한 부분 중의 하나이다. 일반적인 한자 사전에서 글자의 획 및 획순을 보여주기 위해 사용하는 방법은 그림 5와 같이 획순으로 글자를 쓰는 그림을 차례로 나열하는 것이다. 웹이나 멀티미디어를 이용한 한자 교육 컨텐츠 등에서는 이 그림들을 일정한 시간 간격으로 차례로 바꿔가며 보여주는 애니메이션을 만들어 보여주기도 한다. 그러나 두 경우 모두 획을 그림의 비트맵 크기가 커지고 획수가 많아지면 데이타의 용량이 매우 커지게 된다. 본 논문에서는 기존의 방법보다 훨씬 용량이 적은 한자 획 애니메이션을 개발하였다. 뿐만 아니라, 일정한 시간 간격이 지나고 나서 다음 획이 갑자기 나타나는 것이 아니라, 붓으로 글자를 그리는 것과 같은 자연스러운 애니메이션을 보여준다.

그림 5 한자 획순을 보여주는 예

트루타입 폰트의 글립 데이타에서 폐곡선의 인덱스는 폰트 제작자가 폰트를 만들 때 생성하는 순서대로 붙여지며, 조절점의 인덱스는 하나의 폐곡선 안에서는 연속적이다. 그러나 이러한 인덱스에만 의존하여 이용하여 한자의 획 정보를 알아내는 것은 불가능하다. 즉 폰트 데이타는 글자의 모양에 대한 정보만을 가지고 있을 뿐, 획과 같은 의미적인 정보를 제공해 주지는 않는다. 예를 들어, 그림 4의 (a)를 보면, '입 구(口)' 자의 경우, 실제 사람이 인지하는 획은 맨 왼쪽의 세로획('丨')과 기역자('ㄱ')모양의 획, 그리고 아래의 가로획('一')이지만, 글립 데이타는 실제 획의 모양이나 순서와는 전혀 관계없이 외부 폐곡선과 내부 폐곡선의 쌍으로 그 글자의 모양을 표현하고 있다. 따라서 한자 획순으로 붓으로 그리는 효과의 애니메이션을 생성하기 위해서는 먼저 글자 모양을 사람이 사용하는 일련의 획으로 분할하는 작업이 필요하다.

본 연구에서는 그림 6과 같은 사용자 입력으로부터 획 정보를 얻는다. 먼저 사용자는 로딩된 글자 모양위에 마우스나 전자펜으로 글자의 크기에 맞추어 글자를 쓴다. 그림 6에서 ①,②,③,④에 해당하는 검은 선들은 사용자가 시간적 순서에 따라 마우스나 전자펜을 떼지 않고 한 번에 그린 점들의 리스트이다.

각 점들의 리스트는 2차원 상의 점의 좌표를 원소(element)로 하는 벡터로 생각할 수 있다. 예를 들어, 그림 6의 각 사용자 입력들이 각각 k, l, m, n 개의 점들로 이루어졌다면, 각 사용자 입력 벡터는 다음과 같다.

사용자 입력 획①:

$$S_1 = ((i_0, j_0), (i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k))$$

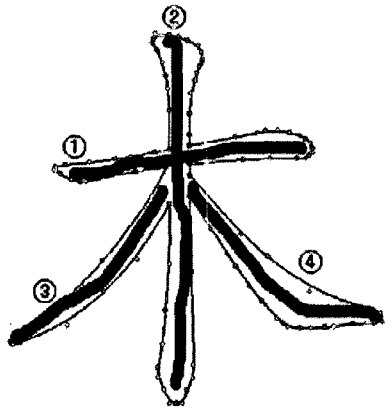


그림 6 사용자 입력 예

사용자 입력 획②:

$$S_2 = ((i_0,j_0),(i_1,j_1),(i_2,j_2),\ldots,(i_l,j_l))$$

사용자 입력 획③:

$$S_3 = ((i_0,j_0),(i_1,j_1),(i_2,j_2),\ldots,(i_m,j_m))$$

사용자 입력 획④:

$$S_4 = ((i_0,j_0),(i_1,j_1),(i_2,j_2),\ldots,(i_n,j_n))$$

위와 같은 사용자 입력 벡터는 두 가지 중요한 획 정보를 제공해 준다. 하나는 획을 이루는 점들이 분포한 좌표의 범위이며, 다른 하나는 벡터의 방향 즉 획의 시작과 끝 정보이다. 본 논문에서는 사용자 입력 벡터를 이용하여, 위와 같이 하나의 폐곡선으로 이루어진 글자 외곽선정보를 여러 개의 폐곡선으로 분할한다.

획 추출 과정은 사용자 입력 벡터에 대응하는 트루타입 폰트의 조절점 벡터를 구성하는 과정이다. 먼저 사용자 입력 벡터를 구성하는 각 점의 좌표와 가장 거리가 가까운 조절점을 찾는다. 사용자 입력 벡터의 점과 트루타입 폰트의 조절점 사이의 거리가 임계치 T_w를 넘지 않으면, 이를 사용자 입력 벡터에 대응하는 조절점 벡터의 원소로 등록한다. 이러한 과정을 모든 사용자 입력 벡터에 대해서 수행하고 나면, 각 사용자 입력 벡터에 대응하는 다음과 같은 조절점 벡터가 만들어진다.

획 ①에 대한 조절점 벡터:

$$P_1 = ((i_0,j_0),(i_1,j_1),(i_2,j_2),\ldots,(i_u,j_u))$$

획 ②에 대한 조절점 벡터:

$$P_2 = ((i_0,j_0),(i_1,j_1),(i_2,j_2),\ldots,(i_v,j_v))$$

획 ③에 대한 조절점 벡터:

$$P_3 = ((i_0,j_0),(i_1,j_1),(i_2,j_2),\ldots,(i_x,j_x))$$

획 ④에 대한 조절점 벡터:

$$P_4 = ((i_0,j_0),(i_1,j_1),(i_2,j_2),\ldots,(i_y,j_y))$$

하나의 획은 하나의 완전한 폐곡선 형태이어야 한다. 그러나 위의 조절점 벡터들은 단순히 원소로 등록된 순서를 따르고 있으므로, 그 순서대로 베지어 커브를 그렸을 때, 꼬임이 일어날 수 있다. 따라서 우리는 각 획에 대한 조절점 벡터 안의 원소들을 가지고 베지어 커브를 그렸을 때 하나의 완전한 폐곡선이 되도록 원소를 정렬(arrangement)해 주어야 한다. 각 획의 조절점 벡터들의 정렬을 위해서는 원래 트루타입 폰트에서 주어진 조절점의 진행 순서를 바탕으로 하여 거리 및 각도 조건을 이용한다. 트루타입 폰트에서 주어진 조절점의 진행 방향은 시계반대 방향이므로, 모든 연산은 시계반대 방향으로 진행하면서 수행된다.

하나의 획을 구성하는 조절점 벡터의 원소를 완전한 폐곡선이 되도록 정렬하는 알고리즘은 다음과 같다. 먼저 부분적으로 연속적인 조절점끼리 그룹핑 한 뒤, 시계반대 방향으로 정렬한다. 이때, 연속적인 조절점 그룹의 끝점의 On-curve 조절점과 그 점 바로 이전의 조절점으로 이루어지는 벡터를 끝점 벡터, 그룹의 시작점의 On-curve 조절점과 그 점 바로 이후의 조절점으로 이루어지는 벡터를 시작 벡터라고 하자. 본 논문에서는 계산의 복잡성을 줄이기 위하여 끝점 벡터 및 시작 벡터를 추출할 때에는 Off-curve 조절점은 고려하지 않았다. 조절점 그룹 중 시작점의 번호가 가장 작은 조절점 그룹을 시작으로 하여 시계반대 방향으로 진행해 가며, 조절점 그룹의 끝점에서 좌표상의 거리가 가장 가깝고 벡터의 각도가 가장 유사한 다른 그룹의 시작점을 찾아 연결한다.

그림 7의 '木(나무 목)' 자의 첫 번째 획을 예로 들면, 이 획은 연속적인 조절점 그룹 A와 그룹 B로 나뉜다.

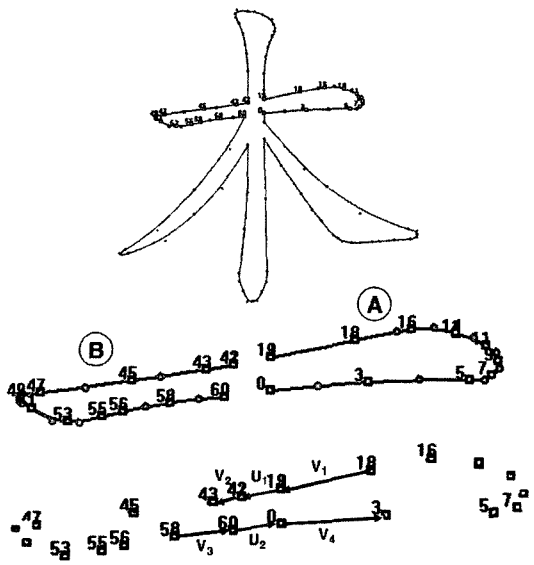


그림 7 조절점 정렬을 통한 획의 폐곡선 완성 (예) '木(나무 목)' 자의 첫 번째 획

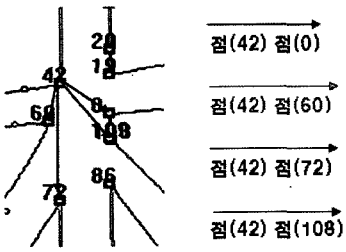


그림 8 조절점 정렬을 위한 우선순위 결정 예

그룹 A는 조절점이 번호 '0'부터 시작하며, 그룹 B는 조절점이 번호 '42'부터 시작하므로, 불연속 부분의 벡터를 찾는 작업은 그룹 A로부터 시작한다. 우리는 그룹 A의 끝점('19'번)을 시작으로 하고 그룹 B의 시작점('42'번)을 끝으로 하는 벡터 U_1을 쉽게 구할 수 있다. 그룹 B의 시작점은 그룹 A의 끝점과 가장 가까운 다른 그룹의 시작점이며, 벡터 U_1은 근접 벡터 V_1, V_2와 각도가 거의 같다. 유사한 방법으로 그룹 B의 끝점('60'번)을 시작으로 하고 그룹 A를 끝으로 하는 벡터 U_2를 구할 수 있다. 그림 8과 같이 하나의 끝점에서 갈 수 있는 다른 그룹의 시작점의 수가 여러 개일 경우에는 거리와 각도를 모두 고려하여 우선순위를 정한다. 그림 8에서는 점 42을 시작으로 하고 점 60을 끝으로 하는 벡터가 불연속 부분의 벡터로 선택되었다.

위에서 설명한 바와 같이 그룹핑되고 정렬된 일련의 조절점들로 베지어 곡선을 그려서 생성되는 폐곡선들이 글자를 이루는 획의 모양이 된다.

3.3 획의 방향 결정 및 부분 획 분할

획의 방향은 사용자의 획 입력 시 얻어진 획의 시작과 끝에 관한 정보와 획의 외곽선 위의 점들의 주축(principle axis)을 계산하여 결정한다. 외곽선 픽셀 좌표들의 공분산행렬(variance-covariance matrix)을 구한 다음 그것의 고유 벡터(eigen vector) 성분을 구한다. 본 논문에서는 SVD(singular value decomposition)를 수행하여 획의 방향 벡터를 구하였으며, 그 결과로 구해진 주축이 획의 방향이 된다. 그림 9는 획 '丿'의 경계 사각형(bounding box)에서, 장축(a)와 단축(b)를 구하였을 때 장축이 획의 주축이 되며, 사용자가 입력 벡터로부터 얻은 시작점과 끝점을 적용하여 최종적으로 획의 방향 벡터를 알아냄을 보여준다.

한자의 획 중에는 하나의 획이 비교적 한 가지 방향만을 가지는 획이 있는가 하면, 획이 그려지는 중에 방향이 바뀌어 하나의 획이 여러 방향을 가지기도 한다. 이러한 현상은 '한양해서체'와 같이 보다 붓글씨에 가까운 자연스러운 서체일수록 더 심하다. 만약 이러한 획을 단일 방향을 가지는 획과 마찬가지로 획을 이루는 모든

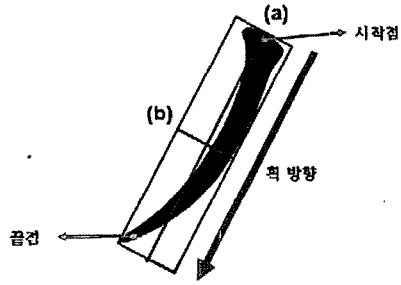


그림 9 획 방향 결정

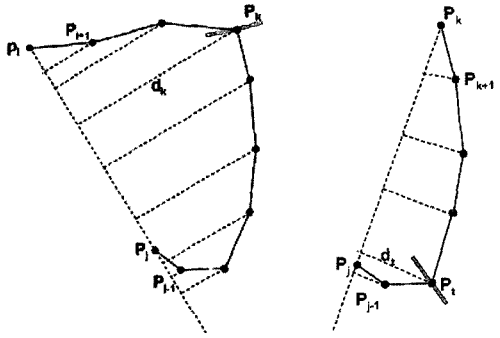
점들의 고유 벡터를 구하여 방향을 결정하면, 엉뚱한 방향으로 획이 그려질 것이다. 본 논문에서는 다각형 근사화 (polygonal approximation) 기법을 통해 다중 방향을 가지는 획을 각각의 단일 방향을 가지는 부분 획(sub-stroke)으로 분할함으로써 이 문제를 해결하였다.

이차원 상의 복잡한 모양의 선형 곡선을 보다 단순한 직선들의 집합으로 근사화하는 문제는 컴퓨터 그래픽스의 각종 분야, 벡터화(vectorization), 벡터 지도 생성 및 처리(vector map processing) 등에 있어서 매우 중요한 문제이다[11]. 본 논문에서는 다각형 근사 기법을 이용하여 여러 방향으로 구부러지는 획을 단일 방향을 가지는 부분 획으로 분할한다. 사용자가 입력해 준 점의 집합을 P라고 하면, 2차원 평면에서 N개의 점(vertex)을 가진 열린 다각 곡선 P는 순서가 있는 점들의 집합인 $P=p_1,p_2,\ldots,p_N=(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)$로 표현된다. 일반적인 다각형 근사화는 $M+1$ (단, $M<N$) 개의 점을 가지며, 기하학적으로 P에 가장 가까운 P의 부분 집합 $Q=q_1,q_2,\cdots,q_{M+1}$을 구하는 것이다. 이 때, P와 Q의 양 끝 점은 같다. 즉, $q_1=p_1, q_{M+1}=p_N$이다.

다각형 근사 기법은 주어진 문제에 따라서 주로 두 가지 형태의 최적화 기법과 연결될 수 있다. 하나는 M 즉 출력 선분의 개수가 변함없을 때, P와 Q 사이의 오차 $E(P)$가 최소가 되는 Q를 구한다. 다른 하나는 $E(P)$가 주어진 최대 임계값 ϵ을 넘지 않을 때, M이 최소가 되는 Q를 구하는 방법이다. 근사 곡선 Q는 어떤 오차 기준을 만족해야 하는데, 그것은 각 응용에 따라서 적절히 정해지게 된다. 가장 많이 사용되는 실질적인 오차 측정 방법(error measure)은 입력 곡선 P와 근사된 선분들의 집합 Q의 각 점들 사이의 거리에 기반한 방법이다. 식 (2)는 곡선상의 점 p_k로부터 선분 (p_i,p_j)에 이르는 거리 $d_k(i,j)$를 정의한다. 계수 $a_{i,j}=(y_j-y_i)/(x_j-x_i)$ 이고, $b_{i,j}=y_i-a_{i,j}x_i$ 이다.

$$d(k;i,j)=\frac{|y_k-a_{i,j}x_k-b_{i,j}|}{\sqrt{1+a_{i,j}^2}} \quad (2)$$

(a) 획 모양 및 사용자 입력해 준 점의 집합



(b) 1차 분할 (c) 2차 분할

그림 10 획 분할을 위한 다각형 근사화 과정의 예

본 논문에서는 사용자가 획을 따라 그려준 점의 집합을 입력으로 하며, 이것이 몇 개의 부분 획으로 분할 될 수 있을지를 알아내야 하므로, 최적화 기법 두 가지 중 두 번째를 적용한다. 보다 자세히 기술하면, p_i 에서 p_j 사이의 점들 중에 선분 (p_i,p_j)에 이르는 거리가 가장 최대가 되는 점이 p_k 라고 할 때, 거리 d_k 가 임계값 T 보다 크다면, 원래의 점집합을 두 개의 점집합 $p_i,\cdots,p_k$ 와 $p_k,\cdots,p_j$ 로 나눈다. 그리고 각각의 점집합에 대해 이 과정을 반복한다. 그림 10의 (b)와 (c)는 (a)의 사용자가 입력해 준 점집합을 분할하는 과정을 도식화 한 것이다. (b)는 p_i 에서 p_j 사이의 분할점 p_k를, (c)는 p_k 에서 p_j 사이의 분할점 p_t 를 찾는 과정을 보여준다. 즉 기역자('ㄱ') 모양의 획 (a)는 p_k 지점에서 한 번 분할되고, p_t 지점에서 다시 한 번 분할되므로, 총 3개의 부분 획, $p_i,\cdots,p_k$, $p_k,\cdots,p_t$, 그리고 $p_t,\cdots,p_j$ 으로 분할된다.

본 논문에서는 사용자의 입력이 대부분의 경우 올바르다고 가정하고, 부분 획 분할을 위한 입력을 사용자가 그려준 점의 집합으로 정의하였다. 이와는 다르게, 사용자의 입력으로부터는 단지, 획의 모양 및 시작점과 끝점의 정보만을 취하고, 부분 획 분할을 위한 입력으로 획의 중간 축(medial axis) 상의 점들을 사용할 수도 있다. 사용자의 입력이 획의 모양을 크게 벗어나지 않을 때, 두 경우 부분 획 분할 결과는 거의 같다.

그림 11은 본 논문에서 제안하는 획 분할 알고리즘 전체를 보여준다.

```
Q is a set of control points consist of a glyph.
S is a set of point vectors for user strokes.
T_w is the threshold to decide whether a control point is belonged to a stroke or not.

void StrokeSegmentation ( int stroke_num,
                          point_vector Q,
                          set_of_point_vector S[],
                          set_of_control_point P[],
                          const number Tw )
{
  set_of_control_point_vectors  P[1 ... stroke_num];

  //stroke grouping
  for (int n=0; n<stroke_num; n++)
  {
     for (int k=0; k<S[n].point_num; k++)
        for (int t=0; t<Q.point_num; t++)
        {
           if (Dist(S[n](k), Q(t)) < Tw)
              P[n].AddControlPoint(Q(t));
        }
  }

  //stroke segmentation
  for (int n=0; n<stroke_num; n++)
  {
     P[n].SetSubStrokes( PolygonalApproximation (P[n]) );
  }

  //control point arrangement
  for (int n=0; n<stroke_num; n++)
  {
     for (int k=0; k<P[n].SubStrokeNum ; k++)
     {
        P[n].SubStroke[k].ControlPointRearrangement();
     }
  }
}
```

그림 11 획 분할 알고리즘

그림 12는 본 연구에서 제안하는 저작도구를 이용하여 사용자 입력을 받아 '成(이루다 성)'자에 대한 획 추출 및 부분 획 분할의 예를 보여준다. 그림 12에서 '성(成)' 자의 3획과 4획은 각각 3개씩의 부분 획으로 다시 분할되었으며, 각 부분 획들은 단일 방향을 가지는 것을 볼 수 있다.

본 논문에서 제안하는 부분 획 분할 방법을 적용하면

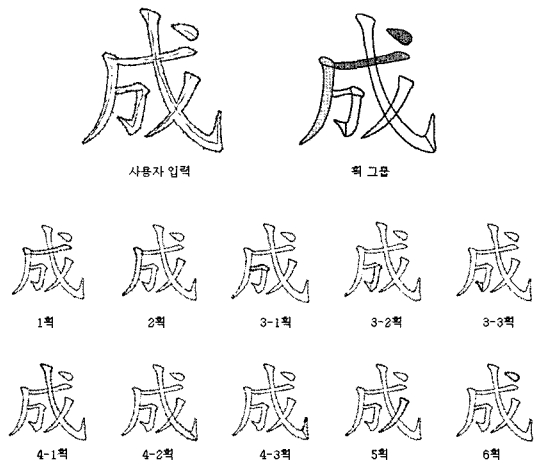


그림 12 한자 획 분할의 예 ('成(이루다 성)')

그림 13의 '口(입 구)'자와 같이 내폐곡선과 외폐곡선의 쌍으로 이루어진 글자 모양도 성공적으로 획을 추출할 수 있다. 내폐곡선은 시작하여 시계 방향으로 조절점 벡터가 진행되며, 외폐곡선은 일반적인 조절점의 진행 방향인 시계반대 방향으로 진행되므로, 조절점을 정렬하는 일이 불가능하거나 복잡할 것으로 보인다. 하지만, 먼저 사용자가 입력한 점 벡터를 가지고, 부분 획으로 분할하게 되면, 획 추출 작업은 단일한 방향을 가지는 각 부분 획들에 대해서 조절점을 정렬하는 일이므로 3.2절의 획 추출 알고리즘으로 조절점의 정렬이 가능하다. 그 결과는 그림 14 및 그림 15와 같다.

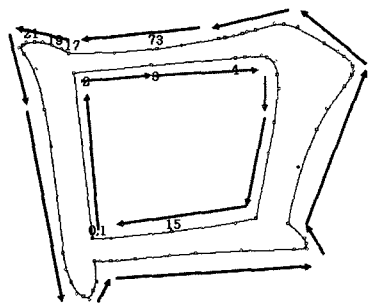


그림 13 내폐곡선과 외폐곡선의 조절점 벡터의 진행 방향

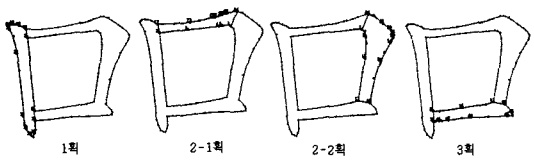
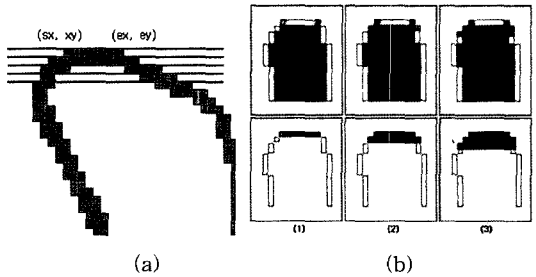


그림 14 한자 획 분할의 예 ('口(입 구)')

3.4 애니메이션 생성 및 압축

획 분할 및 획 방향이 결정된 후에는 한자 애니메이션 생성을 위해 획의 방향으로 순차적으로 픽셀을 채워가는 스캔 라인 알고리즘(scan line algorithm)이 필요하다. 본 논문에서는 그림 16의 (a)와 같이, 베지어 곡선에 의해 결정되는 획의 외곽선의 모든 픽셀 좌표를 획 방향을 따라서 정렬한다. 획의 시작부터 끝까지 획 방향에 수직인 선을 픽셀 단위로 그었을 때, 같은 선상(회색선)에 있는 픽셀들 가운데 최 외곽 픽셀들(파란색 픽셀)만을 선택한다. 선택된 최 외곽 픽셀들의 좌표쌍이 최종적으로 저장될 애니메이션 데이타이다. 이 점들의 쌍을 이은 직선을 연속적으로 그려주면 한자를 붓으로 그리는 애니메이션 효과를 얻을 수 있다. 그림 16의 (b)는 연속적으로 추출되어 저장되는 점들의 쌍(붉은 색)과 그것들 사이를 라인으로 채워 넣음으로써 점차적으로 획이 그려지는 애니메이션이 렌더링되는 과정을 보여준다.



(a) (b)

그림 16 스캔 라인 알고리즘(a)과 한자 애니메이션 렌더링(b)

스캔 라인 알고리즘의 입력으로 들어가는 획은 단일한 방향만을 가지도록 분할된 획들이다. 본 논문에서는 각 획의 방향을 결정하고, 획의 주축(principal axis)을 중심으로 스캔 라인 알고리즘을 적용한다. 즉, 획이 대각선 방향이라면, 대각선이 수직선이 되도록 좌표계를 회전한 뒤, 위의 스캔 라인 알고리즘을 적용하고, 다시 원래의 좌표계로 변환한다.

본 논문에서 한자 획 애니메이션 데이타는 최종적으로 모바일 기기 상으로 전송되는 데이타이기 때문에 데이타의 양을 최대한 줄여야 한다. 위에서 언급했듯이 최종적으로 저장될 애니메이션 데이타는 글자를 구성하는 최 외곽 픽셀의 좌표들인데, 이 좌표들은 획 방향을 따라 정렬된다. 또한 대부분의 픽셀들은 정렬된 리스트 상에서 자신의 이전 그리고 이후 픽셀들과 연결되어 있다. 즉, 어떤 픽셀은 이전 픽셀의 위, 아래, 옆 또는 대각선 방향에 위치하게 된다. 이러한 성질을 이용하여

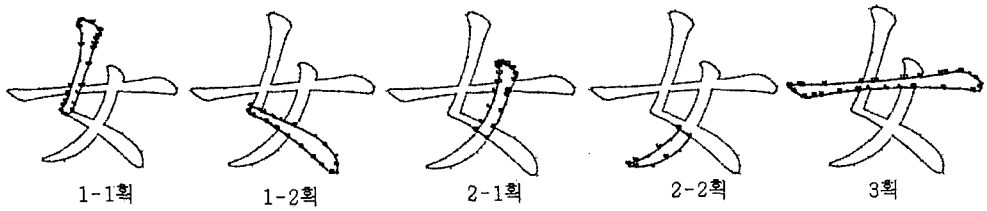


그림 15 한자 획 분할의 예 ('女(계집 여)')

맨 처음 픽셀의 좌표는 절대 좌표로 저장한 뒤, 이후에 연속되는 픽셀부터는 이전 픽셀과의 상대적인 좌표만을 저장하면 애니메이션 데이타를 효율적으로 압축할 수 있다. 본 연구에서는 픽셀의 상대적인 좌표 변화를 표현하기 위하여 단지 2비트를 할당한다. 표 1은 상대적인 좌표의 표현을 위한 규칙을 나타내고 있으며, 이는 x, y 좌표 각각에 적용된다. 그림 17은 표 1을 이용한 애니메이션 데이타 표현의 예를 보여준다. 그런데 만약 표 1의 비트 '11', 즉 이전 픽셀과 불연속인 픽셀이 나타날 경우, 다음 좌표는 다시 절대 좌표로 저장해야 한다.

표 1 압축을 위한 비트 표현

비트	내 용
00	변화 없음
01	이전 픽셀 좌표 +1
10	이전 픽셀 좌표 -1
11	이전 픽셀 좌표와 관계 없음(불연속)

그림 18은 '高(고, 0x9ad8)' 자의 압축된 애니메이션 데이타이다. 비트로 표현된 좌표값들은 여러 개를 한꺼번에 모아서 하나의 정수(integer)로 표현할 수 있으므로, 꽤 복잡한 모양의, 즉 많은 획을 가진 한자라 할지라도 데이타의 양이 1~2kbyte 내외이다.

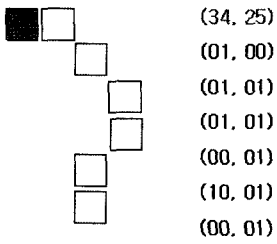


그림 17 데이타 표현 예

4. 저작 도구와 모바일 기기에서의 렌더링

본 논문에서는 한자 학습 컨텐츠의 빠른 개발 및 일관성 있는 컨텐츠의 관리를 위한 컨텐츠 저작 도구를 개발하였다. 저작 도구를 이용하면 사용자가 그래픽이나 한자, 모바일 프로그래밍에 관한 전문가가 아니라 하더라도 쉽고 빠르게 한자 학습용 컨텐츠를 개발할 수 있다. 컨텐츠 저작 도구의 인터페이스는 그림 19과 같다.

본 논문에서 제안하는 저작 도구를 이용해서 한자 학습 모바일 컨텐츠를 작성하는 방식은 다음과 같다. 첫째, 작성하고자 하는 한자의 음(한글)을 입력하고, 그 음에 해당하는 한자(漢字)를 선택한다. 그러면, 한자의 모양과 외곽선에 관한 베지어 곡선의 조절점이 화면에 표시된다. 이 때, 글자의 모양은 트루타입 폰트 정보를 읽어 오므로, 사용자는 자신이 원하는 글씨체를 자유롭게

```
10          //획수
16,42,14,33,18,29,66,14,32,17      //각 획의 점의 개수
2,4,2,9,3,3,12,2,7,3    //획당 SX를 표현한 Integer의 개수
2,4,2,5,3,3,8,2,4,3     //획당 SY를 표현한 Integer의 개수
2,4,2,10,3,3,14,2,9,3  //획당 DX를 표현한 Integer의 개수
2,4,2,5,4,3,12,2,9,5    //획당 DY를 표현한 Integer의 개수
// SX - 스캔 라인 시작점의 X좌표
33,8921642, 10,357913941,357913941,1398101, 22,2629664, 39,679481514,14,30,14,27,234,22,8714,
39,715827882,10, 11,8,559104, 13,218384,20,3413,27,218453,37,5592405,12288,40,17842436,69, 23,19400832,
39,671090730,3754,26,14,23,8714, 39,715827882,2
// SY - 스캔 라인 시작점의 Y좌표
9,706906240, 14,134219776,3389120, 1130504, 32,44739242, 30,14848,26,4458666,664193, 28,4194570,5,
64,715827882,44739242, 38,537002053,131072,357912576,3,57,340071505,16, 56,44739242,
55,44737152,144720896,2, 53,16418,1
// DX - 스캔 라인 끝점의 X좌표
35,707306016, 10,357913941,357913941,1398101, 25,42467456, 40,14,44,211978376,35,938,29,938,23,2698,
39,715827882,10, 15,0,44171264, 13,13,16,13653,24,3413,31,13653,39,3413,46,289739093,2176,8704,
26,42074240, 40,572555862,50,34,938,28,14,25,2602, 39,715827882,2
// DY - 스캔 라인 끝점의 Y좌표
7,8521866, 16,2129937,2,0, 32,44739242, 29,12,23,101196322,8454208, 30,205520896,33,0,
64,715827882,44739242, 38,537001992,33562624,339805248,13,42,13,45,223696213,60,13,63, 56,44739242,
54,12,52,3,50,3,48,101222952,525376, 55,50331648,57,48,55
```

그림 18 高자(10획)의 압축된 데이타

선택할 수 있다. 둘째, 사용자가 글자의 모양 위에 마우스나 전자펜으로 획을 입력한다. 사용자의 입력이 끝나면 각 획 별로 조절점 정보 및 획의 방향 정보, 최종적인 획 애니메이션 정보가 자동으로 생성된다. 저작 도구는 자동으로 분할된 획 정보에 사용자가 직접 컨트롤 포인트를 추가 또는 삭제 할 수 있는 수동 편집 모드를 지원한다. 그래서 자동 획 분할이 잘못된 결과를 출력한 경우, 사용자가 이를 수정할 수 있다. 셋째, 현재 글자에 대한 정보를 저장하면, 모바일 응용 프로그램에 바로 임베드 될 수 있는 형태의 코드가 출력된다. 컨텐츠 개발자는 이 코드를 모바일 응용 프로그램에 삽입하여 바로 한자 학습을 위한 모바일 컨텐츠를 제작 및 테스트 해 볼 수 있다.

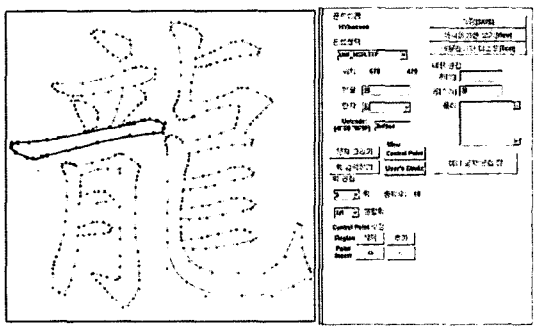
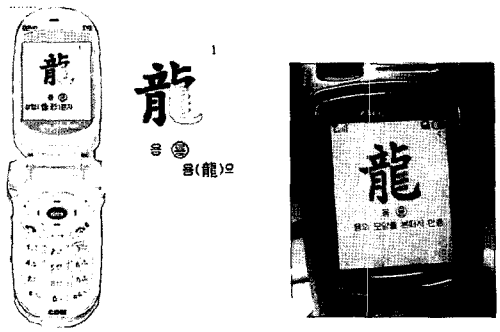


그림 19 한자 학습 컨텐츠 저작 도구

모바일 기기로 전송되는 정보는 단지 획 애니메이션에 필요한 글자의 외곽선 정보이다. 모바일 기기에서의 획 애니메이션을 위해서는 해당 인덱스의 글자 정보를 로딩하고, 압축된 글자정보를 디코딩하여 원래의 데이타를 복원한 다음, 그 결과를 화면에 그려주는 과정이 필요하다. 본 논문에서 제안하는 한자 학습 모바일 컨텐츠는 한자 획 애니메이션뿐 아니라, 한자의 음과 훈, 부수, 풀이에 관한 정보를 볼 수 있고, 검색 및 책갈피 기능을 제공한다. 단, 한자의 풀이에 관한 정보에서 보이는 작은 크기의 한자들은 단지 작은 크기의 정적인 비트맵으로 만들어 전송한다. 본 시스템에서는 모바일 기기에서 보여 지는 글자의 크기를 고려하여 애니메이션 데이타의 크기는 100×100, 한자 풀이에 사용된 글자는 18×18 픽셀 크기의 비트맵으로 만들었다. 그림 20은 PC상의 에뮬레이터 및 실제 모바일 기기에서의 컨텐츠 렌더링을 보여준다.



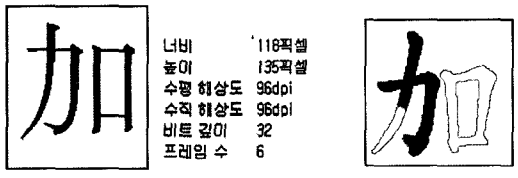
(a) 에뮬레이터 (b) 실제 모바일 기기

그림 20 모바일 환경에서의 렌더링 (예: GVM)

5. 실험 및 결과

본 논문에서 제안한 한자 획 애니메이션 방법이 기존의 GIF 방식의 한자 획순 애니메이션 보다 데이타 용량 및 컨텐츠 개발 시간 면에서 얼마나 효율적인가를 비교하기위해 다음과 같은 실험을 하였다. 먼저 테스트할 한자를 총 획수 별로 20개 내외를 선정하였다. 다음으로 본 논문에서 개발한 저작도구를 이용하여 해당 한자들에 대한 획 애니메이션을 제작 및 테스트 하였다. 본 논문에서는 제안한 시스템과의 용량 비교를 위해 포털 웹사이트에서 현재 서비스 중인 한자 사전 컨텐츠에서 제공하는 GIF 형식의 한자 획순 애니메이션을 사용하였다.

그림 21의 (a)에서 볼 수 있듯이 기존의 한자 애니메이션 컨텐츠는 실제 붓글씨와 다소 차이가 있는 폰트(바탕체)이다. 제안한 방법으로 제작한 한자 애니메이션의 폰트는 (b)에서 보이는 바와 같이 실제 붓글씨와 유사한 한양해서체이다. 같은 글자라 하더라도 한양해서체는 바탕체보다 글자의 모양이 복잡하고 곡선이 섬세하여 폰트의 모양을 결정하는 조절점의 개수가 훨씬 많다. 하지만 한자의 모양이 붓글씨에 가깝고 아름다우며, 한자의 획에 관한 정보가 보다 정확하게 표현되기 때문에 보다 학습의 효과가 높다. 하지만 획의 굵기 변화가 훨씬 심하기 때문에 본 시스템에서 제안한 방법으로 애니메이션을 압축할 때, 데이타의 양이 많아진다.



(a) (b)

그림 21 실험에 사용된 두 가지 애니메이션

기존의 GIF 애니메이션 (a)와 본 논문에서 제안한 애니메이션인 (b)의 가장 큰 차이점은, (a)는 획 순에 따라 한 획씩 갑자기 나타나는 것인데 반해, (b)는 실제

펜이나 붓으로 글씨를 쓸 때와 같이 획의 방향으로 획이 서서히 그려진다는 것이다. (a)에서 더할 가(加) 자는 총 6획의 한자이므로 6개의 프레임으로 이루어진 GIF 애니메이션이 된다. 만약 GIF 애니메이션으로 본 논문에서 제안한 애니메이션과 같이 붓으로 그려지는 효과를 보여준다면, 프레임의 수 및 데이타의 크기가 훨씬 커질 것이다.

본 논문에서 제안한 한자 획 애니메이션이 컨텐츠의 용량 면에서 얼마나 효율적인가를 검증하기 위해 총 획수 별로 비슷한 크기의 한자 애니메이션을 제작하여, 기존의 GIF 애니메이션과 데이타의 용량을 비교하였다. 표 2는 이러한 비교를 보여주며, 총 획수가 많아질수록, GIF 애니메이션의 데이타 용량이 크게 증가하는 것을 볼 수 있다. 반면, 제안하는 애니메이션은 GIF 애니메이션보다 용량이 훨씬 작을뿐더러, 획수가 증가하더라도 데이타의 양이 크게 늘어나지 않는다. 이것은 GIF 애니메이션은 획수가 증가하면 프레임의 수가 증가하는 데 반해, 제안한 시스템은 렌더링 순서대로 정렬된 연속적인 점의 쌍을 가지기 때문이다. 획수의 증가에 따라 글자의 모양이 점점 복잡해지므로 약간의 용량 증가는 있지만, 글자의 외곽선 상의 점의 쌍의 개수는 크게 늘어나지 않는다.

GIF 애니메이션의 크기가 118×135 이기 때문에, 비교를 위하여 표 2에서 실험을 위해 제작된 본 논문의 한자 애니메이션은 120×120 크기로 만들었다. 그러나 표 3에서 알 수 있듯이 제안한 방법으로 애니메이션을 제작할 경우, 렌더링 되는 애니메이션의 크기가 몇 배 커져도 컨텐츠 용량은 크게 늘어나지 않는다.

표 2 애니메이션 데이타 용량 비교

총 획수	글자 개수	GIF 획순 애니메이션 데이타 용량 (byte)	제안하는 애니메이션 데이타 용량 (byte)
1	2	1,304.00	579
2	18	1,968.39	743
3	20	2,611.45	893.25
4	25	3,211.52	968
5	25	4,120.36	1,167.50
6	25	4,774.72	1,361.50
7	23	5,820.22	1,418.00
8	23	6,566.44	1,584.00
9	23	7,605.61	1,591.00
10	24	8,938.71	1,872.75
15	24	12,810.13	2,506.80
20	22	19,035.59	3,139.25
25	13	32,272.08	3,330.20
30	1	24,263.00	4,001.00
32	1	48,545.00	4,021.00
평균		12,256.48	1,945.08

표 3 렌더링 크기에 따른 데이타 용량 변화(예: 입구(口))

렌더링 크기 (가로 × 세로 pixel)	데이타 용량 (byte)
100 × 100	592
120 × 120	705
200 × 200	1,119
300 × 300	1,477
400 × 400	1,932

무엇보다 본 논문에서 개발한 저작도구는 컨텐츠의 개발 속도를 크게 향상시킨다. 한 글자의 한자 애니메이션을 제작하는데 있어 글자 모양의 복잡한 정도에 따라 차이가 있지만, 0.5~2분 안에 애니메이션 결과를 확인할 수 있다.

한자 학습 컨텐츠 개발 시간 및 개발상의 용이성을 평가하기 위하여 픽셀 드로잉과 본 논문에서 제안한 저작 도구를 이용한 방법을 이용하여 총 획수가 골고루 분포된 100자의 한자를 한자와 모바일 프로그래밍에 대한 비전문가 1인이 컨텐츠로 작성하는 데 드는 시간을 비교하였다. 표 4는 두 방법의 컨텐츠 개발에 드는 시간을 비교한 것이다. 제안한 방법은 한 글자 당 약 2분 정도의 시간이 소요된 반면, 픽셀 드로잉에 의한 방법은 글자 당 약 10분 이상 소요되었다. 이는 저작 도구를 사용할 때, 사용자의 입력은 단지 실제 글자를 획순으로 그려주는 것 뿐, 나머지는 대부분 자동으로 처리되기 때문이다.

표 4 컨텐츠 개발에 드는 시간 비교 (단위: 분)

글자 수	저작 도구 이용	픽셀 드로잉
10	25	62
50	92	255
100	185	-

컨텐츠의 질적인 면에서도 많은 차이가 있다. 저작 도구를 이용하면 사용자가 원하는 한자의 모양 정보를 트루타입 폰트 파일만 있으면 가능하므로 획득할 수 있으므로 한자의 모양에 대해 일관성이 유지되지만, 픽셀 드로잉에 의한 방법은 개발자에 따라 차이가 날 수 있다.

본 논문에서 제안된 획 추출 및 획 분할 알고리즘은 현재까지는 한글이나 영문자에서 볼 수 있는 붓을 떼지 않고 한 번에 그리는 동그라미 형태의 획(예를 들어, 이응 'o' 또는 알파벳 'a, o')을 효과적으로 자연스럽게 처리하지 못한다. 왜냐하면, 획 분할을 위해 다각형 근사화 기법을 사용하는데, 한 번에 그리는 동그라미 형태의 획은 획의 방향이 한 획 내에서 매우 자주 변하므로, 많은 세부 획으로 나누어진다. 이를 스캔 라인 알고리즘을 이용하여 애니메이션 데이타를 생성하였을 때 부드럽게

한 획으로 그려지지 않고 각 세부 획 별로 약간씩 끊김 현상이 나타나 애니메이션의 결과가 만족스럽지 못하였다. 한자의 경우 세부 획 간의 끊김 현상이 실제 붓으로 글씨를 쓸 때 방향의 전환이나 힘의 강약 조절과 같은 현상과 비슷하여 자연스럽게 보인다. 그러나 한 번에 그리는 동그라미 획에서의 끊김 현상은 사용자들이 부자연스럽게 느낀다. 물론 저작 도구의 수동 편집 모드를 이용하여 적절하게 세부 획을 다시 나누어 주어, 보다 자연스러운 애니메이션이 생성되도록 할 수는 있다. 하지만 한글이나 영문 또는 다양한 형태의 기호나 그림을 처리하기 위해서는 동그라미 형태의 획을 효과적으로 처리하기 위한 방법이 필요하다. 본 논문에서는 렌더링 대상을 현재 한국에서 통용중인 한자를 대상으로 하므로 이러한 방법에 대해서는 언급하지 않는다.

6. 결론 및 향후 과제

본 논문에서는 모바일 한자 학습 컨텐츠를 위한 경량의 애니메이션 생성 방법을 제안하였다. 제안된 방법은 기존의GIF 애니메이션보다 자연스러운 한자 획순 애니메이션이 가능하며 그 용량 또한 훨씬 작아, 충분히 효율적인 모바일 컨텐츠로서의 가치가 있다. 본 논문에 서 제안하는 컨텐츠 저작 도구는 트루타입 폰트를 이용하여 한자 정보를 로딩하고 자동으로 획 애니메이션 데이타를 작성해 줌으로써, 한자나 모바일 프로그래밍에 대한 전문 지식이 없는 사용자도 매우 빠르고 쉽게 한자 애니메이션 컨텐츠를 개발할 수 있다.

본 연구의 한자 애니메이션모듈은 한자뿐 아니라, 한글 또는 다른 형태의 그래픽 등에서 도 이용될 수 있으며 디지털 서예(digital caligraphy)와 같은 예술적 분야에 응용될 수도 있다[12]. 이 경우 한글이나 영문자에는 한자에는 출현하지 않는 동그라미와 같은 형태의 획이 존재하므로, 본 논문에서 제안한 획 추출 알고리즘이 확장되어야 할 필요가 있다. 향후 다양한 문자에 대한 적용 및 획순 정보를 분류하고 학습시킴으로써 자동으로 획을 분할하고 획의 순서를 결정하는 방법에 대해 연구하고자 한다.

참 고 문 헌

[1] 마법천자문 http://www.magichanja.com/

[2] 아이한자 http://www.ihanja.com/

[3] Donald E. Knuth, Digital Typography, Center for the Study of Language and Information - Lecture Note, ISBN 1-57586-010-4, 1999.

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

[5] TrueType Typography http://www.truetype-typography.com/

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

[7] 오길록, 최기선, 박세영, 한글공학, 대영사, 1995.

[8] Donald E. Knuth, METAFONT: The Program, Addison Wesley, 1986.

[9] Edward Angel, Interactive Computer Graphics, A top-down approach with OpenGL (Third Edition), Addison-Wesley, ISBN 0-201-77343-0, 2003.

[10] Douglas E. Zongker, Geraldine Wade and David H. Salesin, "Example-based hinting of true type fonts," SIGGRAPH00: Proc. of the 27th annual conference on Computer graphics and interactive techniques, pp. 411-416, 2000.

[11] Kolesnikov A. and Fränti P., "Polygonal approximation of closed contours," in Lecture Notes in Computer Science, Vol.2749 : Proceeding of the Scandinavian Conference on Image Analysis (SCIA'2003), Göteborg, Sweden, pp. 778-785, June 2003.

[12] Songhua Xu, Fransis C. M. Lau, Kwok-Wai Cheung, Yunhe Pan, "Automatic Generation of Artistic Chinese Calligraphy," IEEE intelligent Systems 20, pp. 32-39, 2000.

구 상 옥

정보과학회논문지 : 시스템 및 이론
제 33 권 제 2 호 참고

장 현 규

2004년 2월 경북대학교 컴퓨터공학과 졸업(공학사). 2006년 2월 경북대학교 컴퓨터공학과 석사 졸업(공학석사). 2006년 1월~현재 (주)아이디스. 관심분야는 Computer Graphics, e-Learning

정 순 기

정보과학회논문지 : 시스템 및 이론
제 33 권 제 2 호 참고