The Asian Journal of TEX, Volume 3, No. 1, June 2009
Article revision 2009/6/25

한글 텍 : 과거, 현재, 그리고 미래

Hangul TEX: Past, Present, and Future

김강수 Kangsoo Kim*
한글텍사용자그룹 karnes@ktug.or.kr

Keywords Korean TEX, Hangul TEX, kɑTEX, KTUG Collection

Abstract This article looks back upon the past and the current status of Hangul TEX system, and tries to take a view on the future of the Hangul TEX. Specifically, we will look into a set of required features of Hangul TEX system by describing the tasks that kɑTEX has faced and tackled. Our focus will be laid on the issues regarding implementing proper Hangul typography as well as basic typesetting of Hangul characters.

1 서론

1.1 개관

텍은 도널드 크누쓰 교수가 1980년대에 이를 만든 이래 아직까지도 사용되고 있는 컴퓨터에 의한 문서 조판 언어/시스템이다.[1] 개발된 지 이미 30년이 지난 세월이 흘렀으니 낡았다면 매우 낡은 프로그램이지만 지금도 급격한 발전을 거듭하고 있는 대단히 새로운 프로그램/시스템이기도 하다. 컴퓨터를 사용함으로써 얻을 수 있는 획기적인 변화 중의 하나가 글쓰기와 문서의 생산–처리 방식임을 생각할 때 문서 자체를 구성하고 인쇄 출판물을 만들어내는 텍이라는 시스템의 중요성은 다시 말할 필요가 없을 것이다. 더욱이 텍은 하나의 프로그래밍 언어로서의 성격도 가지고 있어서 무한한 응용 범위를 가지기 때문에 정보의 가공과 전달에 있어 뺄 수 없는 한 기능을 담당하고 있다고 볼 수 있다.

이 글은 "한글"이라는 문자 체계를 중심으로 하는 문서를 식자하고 조판하여 궁극적으로 한국어 문헌을 생산하는 데 텍이라는 도구가 어떻게 사용될 수 있는가 하는 문제와, 텍에서 한글 문서 또는 한글 문헌을 어떻게 구현하고 처리할 것인가 하는 문제에 관심을 집중한다. 따라서 텍 자체와 그 매크로 패키지인 라텍 또는 컨텍스트 등에 대해서는 필요한 만큼만 언급하게 될 것이다.

텍에서 한글을 사용하는 역사는 짧게 잡아도 1990년대 초까지 거슬러 올라간다. 초창기의 선구자들은 텍이라는 시스템에 한글 문자를 식자하는 데 많은 관심을 기울였다. 그 까닭은

*이 글은 2009년 한국텍학회 학술발표회에서 발표한 내용을 바탕으로 다시 쓴 것이다. 발표 기회를 주신 한국텍학회 및 제안과 비평을 해주신 여러 선생님들, 특히 귀중한 제안을 해주신 이기황 박사께 감사드린다.

1. 텍에 관한 전반적인 사항은 [15]를 보라. 텍 자체는 크누쓰의 원본 TEX을 가리키는 것이지만 이 글에서는 TEX에서 파생된 pdfTEX, LuaTEX 등을 모두 가리키는 통칭으로 쓰겠다.

텍 자체는 영문자 체계에 적합하도록 개발되었기 때문에 한글과 같은 "2바이트 문자"[2]를 식자하기 위해서 어떤 식으로든 노력이 필요했기 때문이다. 오늘날 발전하고 있는 유니코드 기반의 텍-개량시스템들이 특정 언어 문자의 식자 자체에 기술적인 어려움을 느끼고 있지 않은 것에 비하면 이 때의 노력은 일면 허무한 감이 있기도 하나, 이러한 노력이 있었던 덕분에 텍 자체의 발전에 대응할 수 있었으므로 전혀 무익한 것이었다고 생각하지는 않는다.

이 글에서는 좀 더 관점을 넓혀보고자 한다. 단순히 한글이라는 특정 문자를 식자한다는 협소한 시점을 넘어서, 진정으로 한국어 문서와 문헌을 조판하는 데 텍이 어떻게 활용되어 왔으며 활용될 수 있는가를 검토해보고자 하는 것이다. 또한 이 과정에서 한글텍사용자그룹(KTUG)이 기여한 바를 살펴보고, 이를 통하여 텍과 한글의 관계를 역사적으로 조망하고 앞으로의 발전 가능성에 대해서도 가능한 대로 짚어보고자 한다. 주된 관심은 현재 한글텍사용자그룹과 한국텍학회가 공식적으로 배포하고 있는 한글 텍 ko.TEX이 직면했던 문제들과 이를 해결한 과정을 소개하고, 장차의 가능성에 대해서 짚어보고자 하는 것이다. 따라서 ko.TEX 이외의 한글 식자를 가능하게 하는 패키지에 대해서는 이 글에서 관심을 가지지 않는다. 다만 한글 텍의 역사를 회고하기 위하여 ko.TEX의 바탕이 되었던 1990년대와 2000년대 초반의 여러 시도들은 필요한 만큼 언급할 것이다.

이와 더불어, 궁극적으로 텍이 (온라인이든 오프라인이든) 출판 소프트웨어의 지위를 가지고자 한다면 그 사회의 문화적 축적으로서 출판 자체의 상황을 고려하지 않을 수 없다. 텍 사용자라는 공동체와 그 기여가 텍 자체의 발전뿐 아니라 출판 문화의 발전과 향상에 어떻게 기여해 왔으며 앞으로의 향방은 어떠한가를 살펴보는 것도 이 논고의 논점 중 하나이다.

1.2 용어

글의 진행을 좀 더 자연스럽게 하기 위해 여기서 몇 가지 용어를 정리해 두고자 한다.

텍 시스템 텍 자체는 단지 하나의 tex 컴파일러에 불과하지만 이를 컴퓨터에서 운영되도록 하기 위해서는 방대한 유틸리티, 폰트, 매크로 패키지 등을 필요로 한다. 이러한 텍 관련 프로그램들이 유연하게 동작할 수 있도록 하나의 독자적인 시스템을 꾸려야 하는데, 이를 쉽게 운영하도록 도와주는 프로그램 및 관련 파일 모음을 일컫는 뜻으로 텍 시스템이란 용어를 사용한다.

텍 엔진 텍의 핵심 기능은 사용자가 입력한 텍스트 형식의 입력 파일을 "조판"하여 출력물로 변환하는 것이다. 텍 언어의 기본 매크로인 원시명령(primitive)을 정의하고 조판 규칙을 적용하는 핵심 프로그램을 텍 엔진이라고 부른다. 크누스의 원본 TEX은 물론이고 현재 표준인 pdfTEX, 장래의 표준으로 기대를 받고 있는 LuaTEX 등이 모두 텍 엔진에 속한다.[3]

2. 이 표현은 엄격히 말해 올바른 것이 아니다. 그러나 당시 한글을 통칭하여 2바이트 문자라고 불렀던 것은 사실이고 그 영향은 아직도 엄연히 남아 있으므로 이 표현을 빌려 쓰기로 한다.

3. 오직 크누쓰의 원본 TEX만 실행 파일의 이름으로 `tex`을 쓸 수 있다는 데 주의하라. 확장 텍은 `etex`, pdfTEX은 `pdftex`이라고 한다.

매크로 패키지 사용자 인터페이스를 위하여 텍 엔진 위에 미리 정의된 일련의 매크로 커맨드들을 제공하는 패키지를 뜻한다. 라텍, 컨텍스트 등은 일종의 매크로 패키지이다. 라텍 포맷 위에서 또다시 매크로를 정의하여 스타일이나 클래스 형식으로 제공할 수 있는데, 이것을 2차 매크로 패키지라 한다. 이 글에서는 주로 라텍에서의 한글 문제에 관심이 집중되어 있으므로, 특별히 밝힐 필요가 없을 때, 매크로 패키지라 하면 라텍 위에 정의된 2차 매크로 패키지를 가리킨다.

조판 전통적인 인쇄 작업에서 '판'을 만드는 것이 조판(typesetting)인데, 이것은 좁은 의미로는 활자를 배열하여 행을 만들고 행을 배열하여 페이지를 만드는 것을 말한다. 여기에는 공백 처리 규칙, 하이프네이션 규칙, 그 밖의 타이포그래피에 관련된 일련의 규칙들이 적용된다. 단순히 하나의 활자를 심는 것을 식자라 하고 식자를 반복하고 행나누기, 페이지짜기 규칙을 적용하는 것을 조판이라고 하지만 넓은 의미에서 조판은 식자를 포함한다.

폰트와 글꼴 컴퓨터 조판에서 식자된 문자를 표시하기 위해 사용하는 글자의 기본 모양, 그리고 그 글자의 기본 모양을 모아 둔 프로그램 덩어리를 폰트라 지칭하는데, 이 글에서는 폰트와 글꼴을 구별없이 사용한다. 같은 글자라 하더라도 폰트마다 글자의 폭, 띄어쓰기 간격, 모양 등이 다르기 때문에 식자와 조판을 위해서는 먼저 폰트의 메트릭(metric) 정보가 필요하다.

2 한글과 텍

한글이란, 한국어를 표현하기 위해서 주로 사용되는 문자/기서 체계이다. 현재 이 문자는 한국과 북한에서 그 언어를 적는 표준 문자로 사용되고 있다. 자음과 모음을 결합하여 적는 음소문자의 특징을 가지고 있으면서도 음절 단위로 한 글자로 적는 음절 문자의 특징도 동시에 가지고 있는데, 유니코드 표준에서는 음절 단위로 코드를 할당하는 음절 영역과 음소 단위로 코드를 할당하는 소위 조합형 음소 영역을 모두 갖추고 있다.

음소를 조합하여 음절을 만들고 한 음절을 한 개의 문자단위로 인식한다는 이러한 특징으로 인해, 음절 문자로서의 한글은 현대어 글자만 일만 자[4] 이상의 서로 구별되는 문자로 구성된다. 애초부터 한글이라는 문자 체계를 부호화하는 데 있어서 음소 문자를 중시하는 흐름과 음절 문자를 중시하는 흐름이 대립해 있었는데, 이것이 소위 조합형-완성형 논쟁이다. 21세기에 접어든 현재 이 두 경향이 모두 유니코드 표준에 반영되었음에도 불구하고 마이크로소프트 윈도의 강력한 영향 아래 완성형 문자 체계가 사실상의 표준(*de facto* standard)으로 받아들여지게 되었다.

이 절에서는 텍에서 한글 또는 한국어를 사용하기 위해 필요한 조건들을 알아보고, 초창기의 텍 한글 구현에 대해서 일별해보려 한다.

4. 한글 음절 영역은 11,172자로 이루어져 있다.

2.1 텍에서의 한글 구현과 한글 텍

1990년대 당시 한국어 문헌을 식자 조판함에 있어서 텍을 사용해보자고 생각하는 순간, 두 가지 선택이 문제가 되었다. 그 하나는 텍이라는 프로그램과 시스템을 되도록 건드리지 않고 주어진 것으로 받아들이면서, 텍의 문법과 논리 안에서 한글이라는 글자를 표현하는 방안이었고, 다른 하나는 텍의 근간은 되도록 유지하되, 한글이라는 특수성을 충분히 고려해서, 원래의 텍에는 없는 원시 명령이나 기능을 필요한 대로 구현하여 파생-텍 시스템을 독자적으로 구축하는 것이었다. 좀더 기술적인 표현을 쓴다면 첫 번째 것은 텍이 가진 타이포그래피의 철학과 원리를 그대로 받아들이되 한글 문자를 매크로 레벨에서 구현하는 것이라면, 후자는 한글 식자의 원리를 서양어의 그것과 다르다고 보고 한글 텍 엔진을 만들어 보자는 것이라고 하겠다. 이 각각을 "매크로 수준" 방향과 "엔진 수준" 방향이라고 이름 붙이자.

현재 ko.TEX은 "매크로 수준" 방향으로부터 발전해 온 것이다. 이 방향을 선택할 수밖에 없었던 사정은 대략 다음과 같다.

1. 한글 텍 엔진을 구현하기 위해서는 미리 잘 정비된 한글 타이포그래피의 준칙이 존재해야 했다. 그러나 그런 것이 존재하지 않았거나 존재했더라도 준용되고 있지 않았기 때문에, 한글 문서 조판의 기본 원칙부터 새로 수립한다는 것은 너무 지나친 노력을 요구하는 면이 있었다. 반면, 일본의 경우 굉장히 잘 정비된 출판 관련 국가 표준이 이미 존재했고, 따라서 일본어 텍 엔진을 개발하는 방향으로 나아갈 수 있었다.

2. 텍에서 한글 구현을 담당했던 주체들이 대부분 교수나 학생이었고 이들은 텍 전문가였다기보다는 자신의 필요 때문에 한글 구현을 시작한 사람들이었다. 그들에게 한글 텍 엔진을 만들 정도의 충분한 시간과 비용이 주어질 수 없었던 것이 또 하나의 사정이었다고 본다.

3. 당시 컴퓨터 상의 한글 기서 방식이 영문 문서의 포맷에 한글만을 구현해도 별 불편을 느끼지 않을 정도로 한글 문서의 독자적인 점이 부각되지 않거나 주목받지 못했다. 나아가서 일본어와 같이 영문자와는 다른 문장부호를 쓴다거나 한자를 섞어쓴다거나 하는 것이 치명적으로 요구되지 않고 한글만으로도 대략 문서를 작성할 수 있는 것처럼 보였던 점이 이러한 방향을 선택하도록 내몰았다.

그런데 지금에 와서 돌이켜보면, "한글 텍 엔진"을 만들지 못했던 것이 오히려 다행이었던 점이 없지 않다. 전적으로 표준 텍 엔진에 의존했기 때문에 최근 들어 눈부신 텍 엔진의 발전을 가장 선진적으로 적용할 여지가 존재했으며 텍 엔진 자체가 다국어를 표현할 수 있도록 변화함으로써 이제는 한국어 문헌의 식자를 "매크로 레벨"에서 하는 것이 오히려 당연하게 여겨지는 시점에 와 있기 때문이다. XETEX이나 LuaTEX의 발전은 한글 사용 환경에는 절호의 기회이고 그동안 낭비로만 보였던 한글의 구현 자체를 엔진 수준에서 지원받으면서 한글 문서의 참 모습을 구현하는 데 더 많은 노력을 기울일 기회가 주어졌다. 어떤 의미에서는 전화위복이다.

2.2 한글 텍의 요구사항

이 글은 "한글의 식자"와 "한국어 문헌의 조판"을 구별해서 다루고 있음을 분명히 해두고자 한다. 한글의 식자는 문자로서 한글이 인쇄되도록(찍히도록) 하는 것이다. 그러나 한국어 문헌의 조판은 한글의 식자를 바탕으로, 한국어 문헌의 조판 관행을 모두 받아들이고 한국어 문서/문헌으로서의 품위를 갖추는 조판을 가리킨다. 예를 들어, 한글이 찍히기는 했지만 영문 문서에나 적당할 글자 간격, 단어 간격, 줄 간격, 여백 설정, 수식과 텍스트의 거리 등을 그대로 사용하여 문서를 작성하는 것은 어디까지나 영문 문서에 한글을 표시한 것에 불과할 따름이다. 한국어 문헌에는 나름의 미적 기준, 나름의 조판 관행이 있고, 이것을 텍을 통해서 충분히 표현할 수 있어야 한다. 이것은 현재 kɑTEX이 취하고 있는 관점이기도 하다.[5]

그렇다면 도대체 "한국어 문헌 조판의 필수적 요구사항"은 무엇인가? 이 점을 알아보기 전에, 초창기 텍에서의 한글 구현에 임했던 사람들이 기본적인 요소라고 생각했던 것이 무엇인지 검토해보는 일로부터 시작하려 한다.

2.3 초기 텍 한글화의 관심사

초창기 개발자들이 남긴 한글 텍 관련 문서에서 어떤 생각을 가지고 있었는지 살펴볼 수 있는 문헌은 다음 몇 가지가 있다.

류성준은 1994년 마이크로소프트웨어에 실린 글 [6]에서 다음과 같이 말하고 있다.

> 한글 TEX을 만들면서 첫번째 목표로 설정했던 것은 영문 TEX과의 호환성을 최대한 살리면서 한글을 같이 쓰도록 하자는 것이었다. 따라서 일본의 아스키 jTEX과 같은 방법을 택하지 않고 ··· (중략) ··· 한글 TEX의 가장 큰 장점은 기존의 각종 스타일 파일을 고치지 않고 그대로 사용할 수 있게 한 것과 한글이 기능이 필요한 것은 옵션으로 추가할 수 있게 한 것이다. ··· (중략) ··· 특히 한글 TEX은 동양권에서 개발된 TEX 버전 중 TEX을 수정하지 않았음에도 불구하고 유일하게 전처리기를 사용하지 않는 버전으로 일본어 TEX이나 중국어 TEX 버전에 비해 한두 단계 앞선 것임을 자부할 수 있다.
>
> 두번째로 설정한 목표는 300DPI급 정도가 아닌 고해상도 출력기에서 출력했을 때 깨끗하게 나와야 한다는 것이었다. 이것은 DVIPS를 적절히 세팅해 비트맵 PK가 아닌 포스트스크립트 폰트를 이용하게 함으로써 가능해졌다. (띄어쓰기와 맞춤법은 원문 그대로)

이 글에서 "한글 TEX"이라 한 것은 hLATEXp, hLATEXn 등의 전신으로 당시 개발되고 있던 한국과학기술원(KAIST) 수학과의 한글 텍 시스템을 말하는 것이다.[6] 이 생각은 나중에 『한글과 TEX』 [1]에서 다음과 같이 기술된다.

5. 이것이, kɑTEX이 다른 한글 패키지와 구별되는 점이라고 개발자들은 자부하고 있다.
6. 조진환, "한글 라텍이 걸어온 길과 kɑTEX" [13]을 보라. "본격적인 한글 텍 개발은 1990 년 당시 한국과학기술원 학부과정의 전신이었던 한국과학기술대학(KIT) 고기형 교수를 중심으로 시도되었다고 알려진다."

> 한글LATEX안에는 한글과 영어가 아주 자연스럽게 어울어져 있다. 지금까지 보아온 여러 가지 예들로부터 알 수 있듯이 한글 사용과 영어 사용에 있어 아무런 차이가 없다. ··· (중략) ··· 따라서, 사용자들은 별다른 부담없이 한글LATEX에서 한글을 사용할수 있다. (띄어쓰기와 맞춤법은 원문 그대로)

이 두 인용문을 보건대, 초창기 한글 텍의 개발은 거의 전적으로, "영문 LATEX" 의 기능, 장점, 포맷 위에 한글 글자를 표시하는 것이 최대 관심사였음을 알 수 있다. "한글과 영문이 어우러진다" 는 표현에 이르면 "우리말" 을 한글이라는 문자로 식자한다는 자각조차 없다. 그리고 그런 선택은 의도적인 것으로서, 자국언어를 표현하기 위하여 독자적인 엔진과 포맷을 만드는 길을 선택한 일본의 예에 비교하여 "우월하다" 고 판단하고 있음을 볼 수 있다.

이보다는 진일보하여, 한글 텍이 갖추어야 할 요건을 적어둔 것을 차재춘의 한글과 텍 홈페이지 [7]에서 발견할 수 있다.

> 한글 코드 어떤 글자들을 사용할 수 있어야 하는가?
> 처리 속도 영문 TEX에 뒤지지 않는 빠른 한글 TEX
> 호환성 영문 TEX과의 호환성 보장
> 한글 처리의 안정성 영문은 되는데 한글은 안 된다? 깨지지 않는 한글
> 한글다운 한글의 처리 한글 줄바꿈, 조사, ···
> 한글 글꼴 관리 다양한 한글 글꼴은 어떻게?
> 한글 TEX의 표준화 왜 여러 가지 한글 TEX의 사용 방법이 다른가?

필자가 보기에 이것은 아마도 한글 텍 시스템에 대한 최초의 구체적인 고민이 담긴 문서인 것으로 판단한다. 특히 조사와 한글 줄바꿈에 대하여 언급한 것은 한글 문서의 특징, 또는 한글 텍이 갖추어야 할 매우 중요한 한 측면을 지적한 것으로서, 비록 가볍게 지나가는 말로 언급되는 데 그치고 실제 당시 한글 텍 매크로에서 비중있게 다루어지거나 구현되지 못했지만 그 중요성은 가볍게 볼 수 없는 것이다. 단지 이 시기의 주요 고민은 "한글이 깨지지 않는 것" 과 "글꼴" 또는 "한글 코드" 와 같은 문제에 집중해 있었던 것임을 확인할 수 있다. 요컨대 한글이라는 글자 자체의 입력과 출력의 문제이다.

한편, 최우형은 다음과 같은 중요한 언질을 남기고 있다 [14]. 조금 길지만 인용한다.

> 한글 LATEX의 개발 초기의 가장 큰 문제점들중의 하나는 구현의 방법이었다. 구현의 방법에는 크게 두 가지가 있을 수 있는데, 그 하나는 TEX을 바닥부터 고쳐서 진정한 의미의 한글 TEX을 구현하는 방법이었고, 또 하나는 지금의 한글 LATEX처럼 응용계층에서 한글을 처리하도록 하고 이를 돕기 위한 전처리기를 만드는 방법이었다. (중략)
>
> TEX을 바닥에서부터 한글화한다는 것은 호환성이란 면에서 몇 가지 문제점을 일으킨다. 우선 한글을 위한 TEX primitives들이 정의되고 따라서 그 결과인 dvi

7. http://knot.kaist.ac.kr/htex/index.htm

> 파일 역시 한글에 관련된 추가적인 부분을 담고 있다. 따라서 진정한 한글 TEX을 만든다는 것은 한글 DVI previewer, 한글 device driver를 만들어야 한다는 난관에 부딪치게 된다. (중략)
>
> 또하나의 이유는 ... transparency라는 관점에서의 문제점이다. 만약 jTEX과 같이 device driver나 previewer를 한글화한다면 사용자는 새로운 도구들을 다시 컴파일하여 설치해야 하고 이는 대개의 사용자가 하기에는 쉽지 않은 일이다. (띄어쓰기와 맞춤법은 원문 그대로)

이와 같은 관점에서 그는 "한글 LATEX" 의 기능적 조건을 transparency, localization, device independency라고 요약하는데, 이 가운데 transparency라고 부른 것은 실은 사용자에게 새로운 프로그램의 설치와 같은 추가적인 부담을 주지 않고 최소한의 노력을 들여 현재 설치되어 있는 텍 시스템을 그대로 이용할 수 있어야 한다는 의미이고, localization이라 한 것은 book에 대응하는 hbook, article에 대응하는 harticle을 제공하여, 예컨대 `\today`를 '2009년 6월 25일' 로 표현되게 하는 정도의 한국어 문서 형식을 제공한다는 뜻으로 쓴 것이다.[8]

이러한 고민을 안고 한글 텍 또는 텍 한글화가 시작되었다. 최초의 고민, 즉 한글 텍 엔진을 만들 것인가 아니면 기존의 텍 위에 한글 라텍 매크로를 작성하는 데 그칠 것인가 하는 것은 위에 적시한 여러 인용문에서 볼 수 있듯이, "부득이하게" 매크로 레벨의 개발 형식을 따르게 되었다.

일본어나 일본문자의 식자와 비교해볼 때,[9] 한글 텍이 매크로 레벨의 개발로 만족할 수 있었던 이유가 몇 가지 눈에 띈다.

1. 영문자와 한글을 혼용하는 글쓰기 습관이 고정화되어 가고 있었다. 특히 학술문헌에서는 영문자를 쓰지 않는 한글 문서를 상상하기 어려웠다.
2. 문장부호를 한글식이 아니라 영문자 폰트의 것을 빌어쓰는 글쓰기 방식이 주류를 이루고 있었다. 따라서 문장부호의 식자에 관한 한 한글의 특수성을 내세울 이유가 없었다.
3. 띄어쓰기가 없거나 거의 없는 일본어와 달리 우리 글쓰기는 띄어쓰기가 있어서 영문 문서의 단어 사이 간격을 그대로 써도 별 위화감을 느끼지 못했다.

2.4 ko.TEX 이전까지 해결되거나 남겨진 문제들

이 소절에서는 전절에서 언급한 한글 텍 또는 텍 한글화의 문제설정들이 어떤 식으로 해결되거나 발전해갔는지를 간략히 회고해보려 한다. 일부 문제는 텍 시스템 자체가 발전하면서 하위 레벨에서 자연스레 해결되었고 다른 문제는 한글 텍 매크로 시스템의 발전에 따라 관심과 해결책의 폭이 넓어져 갔다.[10]

8. h*.sty를 제공하는 방식은 나중에 폐기되었다. LATEX2ε 이 보편화되면서 article과 같은 표준 포맷을 쓰고 한글을 부수 스타일로 제공하는 것이 현재까지 대세를 이루고 있다.
9. 현재 가장 널리 쓰이고 있는 일본어 텍 시스템인 pTEX은 독자적인 엔진을 개발한 대표적인 사례이다.
10. 여기서 주로 언급하려 하는 것은 은광희의 HLATEX이다.

2.4.1 입력의 문제

한글을 다루는 데 있어서 가장 치명적인 진입상의 문제는 한글 입력에 관련된 문제였다. 입력과 관련해서 적어도 다음 세 가지가 문제가 된다.

1. **에디터의 문제.** 편리하게 쓸 만한 에디터가 존재하는가. 이것은 플랫폼을 가리지 않는 문제이다.
2. **한글 코드 문제.** 한글 텍이 활발하게 개발되던 시기는 조합형–완성형 논쟁의 여진이 남아 있는 가운데 완성형이 우위를 보여가던 때였다. 한글 표현을 조합형으로 할 것인지 완성형으로 할 것인지 하는 문제는 당시로서는 매우 중요한 선택의 문제였다.
3. **전처리기의 문제.** TEX 3.14까지는 "2바이트 문자"를 직접 처리하는 것이 매우 어려웠다. 이것은 3.141 이후에 비로소 가능해졌다. 그 전까지는 입력된 한글 코드를 미리 정의된 매크로로 바꾸어서 처리하였기 때문에 입력 상태의 소스와 TEX에게 처리를 맡기는 소스가 서로 달랐고, 이를 위한 변환기가 필요하였다.

이 가운데 첫 번째 에디터 문제는 여기서 논외로 하기로 한다.[11]

한글 코드 문제는 사실상 완성형으로 자리를 잡게 되었다. 이 시기 소위 "한글이 깨진다"는 것은 거의 한글 코드 선택이나 변환의 실패에서 기인한 것으로 볼 수 있다. 마이크로소프트 윈도가 완성형을 채택했던 것도 영향을 끼쳤을 것이다. 그 결과 상당한 사용자를 가지고 있던 조합형 한글 텍, yahTEX은 급격히 사라져갔다. 이후 유니코드의 등장과 활용 시기까지 사실상 한글 텍은 완성형 일색이었다. 그 결과, "모든 한글의 표현(이 불가능)"이라는 뜻하지 않은 문제가 등장하게 되었다. 당시의 KSC 5601 (나중의 KS X 1001 또는 EUC-KR) 완성형 코드 체계는 대표 한글 2,350자만을 포함하여 "똠"이나 "띡"과 같은 글자를 쓸 수 없었던 것이다. 비록 가끔씩밖에 쓰이지 않는 글자라 하더라도 모든 한글을 다 찍을 수 없다는 점은 한글 텍 사용자를 오랫동안 괴롭혔다.

한TEX, hLATEXp, HLATEX이 한글 텍 환경의 주류로 환영받은 이유 중의 하나가 전처리기의 불편함이 없다는 것이었다. LG에서 1994년 공개한 WinLATEX이라는 시스템은 여전히 전처리기를 사용하고 있었던 반면, 이들 시스템은 전처리기 없이 동작하였다. 현재 이 전처리기의 유무는 더이상 문제삼을 필요가 없어졌다. 전처리기를 없앨 수 있었던 것은 텍 자체의 발전에 힘입은 바가 크다.[12]

2.4.2 출력의 문제

1. **한글 폰트 문제.** 한글 텍 시스템의 최대 고민은 폰트 문제였다. 여기에는 사용할 수 있는 자유로운 폰트가 있는가, 폰트를 사용하는 방법을 어떻게 설정할 것인가, 폰트의 자유로운 변경이 가능한가 하는 여러 문제들이 겹쳐 있고 실제로 한글 텍 매크로들은 거의 대부분의 노력을 폰트 문제를 해결하는 데 바쳤다고 해도 과언이 아니다.

11. DOS의 emTEX을 써본 사람은 Q에디터나, 괘선을 그을 수 있던 TGEdit를 기억할지도 모르겠다. 당시 필자는 파스칼로 직접 에디터를 만들어서 썼는데, 버그가 많아서 파일을 제법 유실시키곤 했다.
12. 전처리기를 채용한 한글 텍에 대해서, 조진환 [13]을 보라.

2. 폰트 문제와 불가분의 것이지만, DVI 드라이버의 문제가 있었다. 폰트는 윤곽선 폰트를 사용했는데 출력은 오직 비트맵으로밖에 얻을 수 없다든가, 아예 한글 폰트 자체를 처리하는 DVI 드라이버가 없다든가, DVI 화면표시기가 오작동한다든가 하는 문제들은 여전히 잘 해결되지 않고 있었다. 이와 더불어 고품위 출력을 위한 ps 및 pdf 파일로의 변환시 윤곽선 폰트 내장과 같은 고급 요구에 대응할 필요도 점점 커졌다.

초창기 한글 텍은 300dpi의 탁상용 프린터에서 결과물을 출력하기에 적당한 정도였다. 실제로 폰트의 라이센스 문제 때문에 mf 소스를 공개할 수 없었던 hLaTeXp는 그 대신 pk 비트맵 폰트를 300dpi, 600dpi용으로 제공했고, 후에 출판용 1200dpi 폰트를 제공하게 되었다. 그러나 현재 1200dpi 수준으로는 출판에 적합하지 않다고 보는 것이 일반적이다.

한TeX[13]은 윈도용 트루타입 폰트를 영문 CM 글꼴에 대응시키는 방법을 적용하였다. 그러나 한TeX에 포함된 글꼴의 라이센스가 불분명했고 한TeX 자체가 개발이나 사용자 지원이 중단되었기 때문에 이 탁월한 기법은 더이상 진전을 보지 못하였다. 그 대신 라이센스의 부담이 없는 hLaTeXp가 일반에 공개되었으나, 기본 글꼴은 미리 변환된 pk 비트맵 폰트뿐이었다. 이것으로는 탁상용 프린터에서의 인쇄 말고 달리 할 수 있는 것이 아무것도 없었다.[14]

폰트 문제에 있어 중요한 진일보는 HLaTeX으로부터 왔다. HLaTeX의 저자인 은광희는 오늘날 은 글꼴의 기원이 되는 UHC 글꼴을 직접 제작하여 이전에 사용되던 문화부 글꼴을 대체했다. 이 글꼴은 포스트스크립트 윤곽선 글꼴이며, 가상폰트 기법을 사용하여 자소를 조합하는 형태였다. 포스트스크립트+가상폰트+서브폰트 기법에 의하여 한글을 출력하던 HLaTeX은, dvips라는 DVI 드라이버에 힘입어 pdf나 ps를 만들어도 윤곽선 폰트의 느낌을 그대로 살릴 수 있어 환영받았다. 그러나 일부 프린터에서의 폰트 캐시 문제 등으로 인한 약간의 문제점이 발견되었고, 마침내 박원규에 의하여 은 글꼴이 발표됨으로써 한글 텍에 있어서 글꼴 문제를 근본적으로 해소하게 되기에 이른다.

2.4.3 "한글화"의 문제들

최우형이 'localization'이라 하고 차재춘이 "한글다운 한글"이라 불렀던 문제, 즉 한글로 표현된 한국어 문서의 특징을 드러내는 부분을 구현하는 문제는 (어떤 이유에선가) 기술적으로 큰 주목을 받지 못했지만 나름대로 상당한 발전을 보였다. 그러나 다른 한편 복잡해지는 문서 작성 환경과 사용자 기대 수준의 상승으로 인하여 그만큼 많은 문제들이 제기되기도 하였다. 특히 중요하다고 생각되는 몇 가지를 돌아본다.

1. **자동조사 문제.** 자동조사는 한국어에서 볼 수 있는 독특한 특성이다. 자동조사 자체는 hLaTeXp에서 처음 도입되어 HLaTeX에서도 이를 구현하였다. HLaTeX의 자동조사 기능은 (pdf를 제작하지 않는 상황에서) 매우 잘 동작하였고, 글쓰기의 생산성을 상당히

13. 한TeX 평가판이 월간 마이크로소프트웨어의 부록으로 제공된 적이 있다. [7]을 보라.
14. 600dpi 또는 1200dpi pk 폰트를 이용하여 제작된 책이 두어 권 있는 것으로 알고 있다. 안타깝게도 이 책들은 출력 품위에 관한 한 텍이라는 도구의 신뢰성을 낮추는 데 기여했다. 한동안 인쇄출판업에서는 텍을 기피하는 이유 중의 하나로 출력의 품질을 들곤 했었다.

높여주었다. 그러나 자동조사 처리 기능의 설계에 근본적인 문제점이 있었는데 이에 대해서는 필자의 다른 글 [2]를 참고하라.

2. **한국어 행자름 문제.** 한글 텍 매크로들은 대부분 행자름(line breaking)을 영문자의 행자름 방식을 준용하되 하이픈 기호를 없애고 모든 한글 글자 사이에서 행자름이 일어날 수 있도록 해두고 있었다. 여기에 HLaTeX은 약간의 금칙처리 루틴을 두고 있었지만 완전하지 못했다. 예를 들면 소위 '고아' 문제와 같은 것이 자동화되지 않았다. kotex이 등장하기 이전까지 한글 문서작성에서 피곤한 문제들 중 하나가 영문 단어에 한글 조사가 붙을 때 영문 단어의 하이프네이션도 억제되고 한글 앞에서 행이 잘리지도 않는다는 점이었다.

3. **자간과 행간 문제.** 한글 텍 매크로들은 자간에 큰 신경을 쓴 것으로 보이지 않는다. 한글 문서의 품위를 결정하는 결정적 요소 중 하나로 자간에 주목하고 처음으로 이를 제어하려 한 것은 필자의 hlatex-interword라는 패키지에서였다. 그 후 HLaTeX 1.0에서 자간을 설정하거나 제어하는 매크로를 도입하게 된다. 행간에 관해서는 일찍부터 한글 문서의 행간이 영문 문서의 기본값보다 큰 것이 좋겠다는 합의가 있었다. hLaTeXp는 이미 1.3배 행간을 제시하고 있었으며 HLaTeX도 1.0 버전에서부터 1.3배 행간을 한글 서식의 기본값으로 채택하였다.

4. **한국어 문서의 장절.** 영문에서 Chapter 1이라고 식자하는 것을 우리글에서는 제1장이라고 적어야 하는 경우가 있다. HLaTeX은 라텍 표준 패키지의 chapter 및 section 관련 명령을 전부 재정의하여 이러한 한국형 장절 수식어를 쓰거나 제어할 수 있었다. 다만 이 "전부 재정의"로 말미암아 뜻하지 않은 부작용이 있었는데, 그것은 장절 명령을 제어하는 패키지와 함께 쓰면 한국형 장절이 모두 무력화되거나 패키지 충돌을 일으키기도 했다는 것이다. 이밖에 표준 패키지의 장 제목 글자 크기가 우리 문서의 기준으로 보아서 지나치게 크다든가 하는 점도 지적되곤 했는데, 이런 것은 실제로 간단히 조절할 수 있는 문제였지만 매번 문서마다 그런 조절을 위한 코드를 일일이 넣어야 한다는 것과 더불어, 어느 정도의 크기가 적당하냐에 대한 지침이 없었다. 한국어 문서의 장절을 손쉽게 쓸 수 있는 일종의 표준을 제공해야 할 필요성이 있었다.

5. **다시 폰트 문제.** 폰트 사용에 있어서 영문자 폰트와 한글 식자 폰트를 함께 쓰는 조판 관행과 더불어, 각 한글 텍 매크로들은 폰트 선택이나 지정을 위한 방법이 통일되어 있지 않았다. HLaTeX의 이른바 HFSS[15]는 NFSS[16]와 별도의 매크로 체계를 설정한 것으로 지나치게 복잡하거나 다른 문서와의 호환성을 낮추는 결과를 가져왔고 영문자 이외에 한글/한자/상징문자를 모두 별도로 지정하여야 하는 번거로움이 있었다. 이 당시 폰트에 관련된 요구는, (1) UHC 글꼴 이외에 다른 일반 폰트를 사용할 수 없는가, (2) 영문자와 숫자에 한글 폰트에 내장된 것을 가져다 쓸 수는 없는가, (3) 트루타입 폰트를 pdf에 내장할 수는 없는가, (4) 장평을 자유롭게 조절할 수는 없는가 하는 여러

15. HFSS에 대해서는 HLaTeX guide [8]을 보라.

16. NFSS (New Font Selection Scheme)란, LaTeX2ε 의 폰트 선택 체계를 말한다. `http://faq.ktug.or.kr/faq/NFSS`를 참조하라.

가지가 있었는데, 적어도 2000년 이전까지 이 질문에 대해 전적으로 가능하다고 답변하기 어려웠다. 이 모든 요구에 대한 답은 ko.TEX과 더불어 해결을 보게 된다. 이에 대해서는 후술한다.

2.4.4 텍 시스템, 사용자 환경, 인터페이스, 호환성

일반 사용자의 입장에서는 한글 텍 시스템을 얼마나 편리하게 설치할 수 있느냐가 중요하다. 또한 손쉽게 컴파일하고 결과를 미리 볼 수 있는 작업환경의 제공도 매우 중요하다. 가장 많은 사용자를 가진 윈도 운영체제에서 텍 작업환경이 어떻게 제공되고 있었는지만을 일별하겠다.

1995년에 나온 한TEX은 이 점에서 획기적인 작업환경을 제공하였다. 윈도 운영체제에서 하나의 창이 열리고 그 안에서 편집, 미리보기, 출력, 수정, 컴파일을 할 수 있는 단축키 (또는 버튼)을 제공한다는 것은, 이미 외국에서 판매되던 상업용 텍 소프트웨어에서 아이디어의 많은 부분을 빌어온 것으로 짐작되지만 현재도 하나의 모델이 될 만한 프로그램이었다. 더불어 윈도용 트루타입 폰트를 사용한다는 새로운 아이디어도 주목할 만하였다. 물론 심각한 메모리 누수, 편집기의 기능적 빈약성, 설정값의 조절이 제한되는 점, 그림 처리의 치명적 에러 등 문제점도 많았고 곧 제작사가 프로그램을 포기하고 방기한 결과 비극적인 역사로 남을 수밖에 없었다는 점이 대단히 유감이다.

한글 텍에서 HLATEX이 주류로 자리잡게 되면서, 주로 MiKTEX이나 fpTEX과 같은 텍 시스템에 이를 간편하게 설치하는 해결책이 유행하게 되었다. 그러나 여전히 압축 파일을 풀고 설정 파일을 수정하거나 하는 이른바 "설치"에 상당한 시간을 할애하지 않으면 안 되었다.

명령행에서 컴파일하는 과정에서, 에러 메시지가 한글일 경우 터미널에서 알아보지 못하는 문제도 지적되었다.[17] 텍 엔진이라 부르기에는 적절하지 않지만 독자의 실행파일을 사용한 hTEX은 에러 메시지를 가로채 윈도 명령행에서 한글이 보이도록 하는 기능을 제공했다. 하지만 그다지 많이 활용되지 않았다. 적절한 에디터를 활용함으로써 이 불편을 우회할 수 있었기 때문일 것이라 짐작한다. 더구나 hTEX은 오늘날의 유니코드 환경에서는 쓸 수 없다.

많은 사용자를 갖게 된 HLATEX이었지만, 오히려 영문 문서와의 호환성에 문제가 생기는 일이 있었다. 대표적인 것이 hangul 패키지와 hyperref 패키지의 충돌이었으며, 이밖에도 endfloat, ulem 같은 패키지와 함께 쓰지 못하거나 충돌이 발생하는 경우가 보고되었다. 한글 서식이나 자동조사를 포기하고 hfont 패키지를 쓰거나, 1.0 이후 버전에서 람다로 컴파일하면 대부분 해결되는 문제였지만 결과적으로 한글 문서의 조판에 있어 완전하지 않다는 인상을 주는 경우가 없지 않았다. 이러한 호환성 문제가 HLATEX에서만 보고된 이유는 아마도 HLATEX만큼 본격적으로 문서 작성에 활용된 패키지가 이전에 없었기 때문이지 HLATEX이 유달리 부실하게 작성된 패키지였기 때문은 아니다. 그렇게 판단하는 이유는 패키지 충돌을 일으키는 코드들이 HLATEX에서만 존재한 것은 아니었기 때문이다.

17. UTF-8 터미널을 이용하지 못하는 윈도우 운영체제에서는 현재의 XETEX-ko나 LuaTEX-ko로 컴파일하는 문서의 대화창에서 한글을 제대로 보여주기 어렵다. 이에 대한 대안은 TEXworks라는 에디터를 사용하는 것이다. 이 에디터의 컴파일 상태 창은 한글 에러 메시지를 보여준다.

3 한글 텍의 필요조건과 ko.TEX의 현단계

그렇다면 과연 현 시점에서 한글 텍을 개발함에 있어 요구되는 조건이란 어떤 것일지, 그리고 ko.TEX은 현재 그러한 조건을 얼마나 충족하고 있는지 살펴보겠다. 여기서는 두 가지 큰 범주를 살펴보려 하는데 하나는 기술적인 것이고 다른 하나는 사회-공동체적인 것이다.

기술적인 측면은 한글을 구현하고 한글 문서의 조판에 요구되는 기능들을 제공할 수 있느냐는 점으로서, 어떤 "기능"이 요구되며 이에 얼마나 응답할 수 있느냐 하는 것이다. 한글 텍에 대해서 주로 이러한 점이 검토되고 토론되어 왔다.

다른 한편, 한글 텍의 개발에 있어 사회적 측면이란, 현재와 같이 자발적인 기여자에 의하여 개발이 주도되는 상황에서 텍 사용자 공동체가 어떤 역할을 하고 있느냐, 그리고 문서 생산 도구로서 텍이 어느 정도 실용적으로 활용될 수 있느냐 하는 문제에 국한하고자 한다.

3.1 기술적 필요조건

한글 텍의 궁극적인 목적은 "한국어 문헌의 아름다운 조판"이라고 할 수 있다. 이것은 다시 한글의 자유로운 표현과 한글 문서의 아름다운 표현이라는 두 가지 목적을 달성하는 것으로 집약된다.

3.1.1 한글의 자유로운 표현

한글 코드 앞서 언급한 바와 같이 "완성형" 한글 코드를 중심으로 발전한 이전의 한글 텍은 "모든 한글을 표현"하지 못하였다. 마이크로소프트 윈도 운영체제가 확장완성형을 채택함으로써 한글 표현의 제약을 제거한 데 반해 상당히 늦게까지 완성형이라는 한계에 갇혀 있었던 것은 텍 사용자의 불만을 가중시켰다.

2000~2002년 간, 한글텍사용자그룹에서는 이러한 제약조건을 극복하는 데 많은 노력을 기울였다. 이 때 대안으로 등장한 것은 윈도 운영체제가 사용하는 CP949 코드 체계를 텍의 식자에 이용하는 방안과, 유니코드를 채택하는 방법이었다. 그러나 CP949 코드를 전통적인 TEX으로 구현하려 하는 한, 두 개의 옥텟으로 이를 쪼개어 처리할 수밖에 없는데 이 때 TEX 제어코드와의 충돌을 회피하는 방법이 문제거리로 등장했다. (실제로 거의 사용하지 않는) 한글 몇 글자를 표현하기 위해서 몇몇 제어 문자의 카테고리 코드를 바꾸어야 하는 것은 효율성 면에서나 미학적인 면에서 바람직하지 않은 것으로 생각되어 이 방법은 일찍이 고려 대상에서 제외되었다.

모든 한글을 사용하는 또하나의 방법으로 제시된 것은 오메가라는 유니코드 확장 텍 엔진을 이용하는 것이었다. 오메가와 그 라텍 버전인 람다를 이용하는 경우 입력을 유니코드로 하기만 하면 적어도 한글 표현에 있어서의 제약은 사라지는 것이 확실하였다. 이 점에 주목하여 HLATEX 1.0 버전은 오메가와 람다를 이용한 한글 조판을 장래의 표준으로 잠정적으로 간주하고 이에 관련된 많은 기여를 하게 된다.[18] 그러나 오메가/람다를 이용하여 한글 문서를

18. "Ω는 이제 거의 완숙한 단계에 도달하고 있고 앞으로는 텍을 완전히 대체하고 본격적으로 사용되는 방향으로 전개될 것을 희망하고 있습니다." 은광희 [8, p. 3].

조판하는 것은 뜻하지 않은 제약이 있음이 드러났다. 구체적으로는 오메가/람다 엔진 자체의 안정성에 대한 신뢰 문제와 더불어, 사용자들의 요구 중 하나였던 pdf 문서의 제작에 불편한 점이 있다는 것, 한글 폰트를 사용하기 위한 전처리 과정이 조금 복잡하다는 것 등이 문제로 등장한 것이다. 다른 한편, 2003년 이후 오메가/람다는 개발이 중단되어 이를 이용하여 한글 조판의 문제를 해결하겠다는 기대는 무산되고 말았다. 오메가/람다를 이용하는 방법은 HLaTeX 1.0 이외에도 김도현의 DHHangul이라는 패키지가 제작되기도 하였는데, 특히 중세 문헌의 조판 등에 이 패키지가 일부 활용된 예가 있다.

그러나 오메가/람다를 이용한 실험의 만족스럽지 못한 결과에도 불구하고, 향후 한글 텍이 사용할 한글 코드로 유니코드를 사용하자는 데에 대체적인 합의를 이루게 된다. 마침 UTF-8 인코딩을 이용하여 다국어를 조판하는 latex-ucs 패키지에 주목한 김도현, 김강수는 Hangul-ucs라는 대안 패키지를 개발하게 되고, 이것이 k*o*.TeX의 한 근간을 이루게 되었다.

Hangul-ucs는 그 후 latex-ucs 종속성을 벗어나서 발전하게 되는데, LaTeX/pdfLaTeX 조판의 결과는 매우 만족스러웠으며, 라텍에 있어서 한글 표현 상의 제약을 완전히 제거하게 되는 첫 번째의 성과물이 되었고, 이것은 k*o*.TeX에 그대로 이어졌다. 이에 덧붙여 1933년 이전 표기법(중세국어)으로 만들어진 한글 문헌의 조판에도 괄목할 만한 진전을 이루었는데, 이것은 유니코드 표준을 따르는 한글 자소 조합 루틴을 한글 텍에 도입한 결과로서였다.[19] 말하자면 k*o*.TeX은 "한글의 완전한 표현"을 달성한 최초의 한글 라텍 패키지라고 할 수 있다.

한글 텍의 입력 코드가 유니코드로 방향을 잡은 이후 문제가 된 것은 특히 UTF-8 인코딩을 거의 지원하지 않는 윈도 운영체제 사용자들의 불편이었다. 지금은 많은 에디터들이 UTF-8을 지원하고 있지만 당시 가장 많은 사용자를 가지고 있던 WinEdt라는 에디터로는 유니코드를 입력할 수 없었다. 이 문제는 현재도 완전히 해결되어 있지 않다. 에디터가 유니코드를 지원하지 못하기 때문에 유니코드 한글 텍을 쓰지 못한다는 것은 본말이 전도된 감이 있으나, 사용자에게는 절실한 문제일 수 있는 것이다. 에디터 문제는 길고 긴 탐색을 거쳐서 KC2008에서는 아예 한글 코드 문제에 관한 한 매우 자유로운 선택을 가능하게 하는 Notepad++를 제공하게 됨으로써, 일반 사용자의 불편을 줄이고자 노력하였다.

그러나 생각해보면 입력코드가 무엇이든 내부적으로 유니코드 처리를 하기만 한다면, 즉 입력–처리–출력의 각 단계에서 적절하게 코드를 변환할 수 있다면 지금까지 논의한 모든 문제들이 사실상 해결될 것이다. 다행히 XeTeX은 이 문제를 텍 엔진이 직접 처리해주고 있으며 LuaTeX 역시 그 특유의 확장성으로 이 문제를 근본적으로 해결한다. 향후 한글 텍이 XeTeX이나 LuaTeX을 통하여 발전할 수밖에 없는 이유 가운데 하나라고 하겠다. 실제로 현재 XeTeX-k*o*는 UTF-8뿐 아니라 EUC-KR 코드로도 동작하며,[20] LuaTeX-k*o*는 CP949 코드를 처리하는 능력을 가지고 있다 [5]. 개발자의 입장에서 완전히 검증되었다고 자신할 수 없고 CP949 또는 EUC-KR 코드 사용 자체가 바람직하지 않아서 범용의 사용을 권장하고 있지는 않지만 적어도 입력에 있어서 한글 코드의 문제가 해결된 것은 틀림없다. 이 텍 엔진들은

19. k*o*.TeX의 dhucs-midkor 패키지가 여기에 해당한다.

20. pdfTeX 엔진을 쓰는 경우 k*o*.TeX의 `[euc]` 옵션으로 EUC-KR 코드를 처리할 수 있지만 이것은 utf k*o*.TeX과는 별도의 매크로 체계이다. 즉, k*o*.TeX의 모든 기능을 다 활용할 수 없다. 그러나 XeTeX-k*o*와 LuaTeX-k*o*의 EUC-KR 또는 CP949 기능은 이와 다르게 내부적으로 유니코드로 변환하여 처리하는 것이므로 동일한 매크로가 동작한다.

입력이 어떤 코드로 되든 간에 내부적으로는 유니코드로 동작한다.

이를 통하여, 향후로는 한글 코드가 민감한 문제가 아닌 것으로 느끼게 될 것으로 기대한다. 즉, 이 문제는 해결된 문제이다.

한글 글꼴 한글 글꼴 문제는 앞서도 말한 바와 같이 한글 텍에 있어서 최고, 최대의 문제였다. 여기에는, 한글 텍의 "기본 글꼴"을 무엇으로 할 것인가 하는 점과, 다양한 한글 글꼴을 텍이 처리할 수 있느냐 하는 두 가지 문제가 제기되어 있었다. 이것은 역으로 생각하면, 기존의 한글 텍이 사용하던 글꼴이, pdf 문서를 얻기에 충분히 만족스러운 품위를 보여주지 못하거나, 인쇄 출판에 이용하기에 독자를 설득할 만큼 충분한 아름다움을 보여주지 못하거나, 폰트 사용의 제약이 너무 커서 일반 폰트를 사용할 수 없다고 인식되어 왔다는 것을 의미한다. 그리고 이 문제점들은 현 단계에 이르러 완전히 해소되었다고 할 만하다.

기본 글꼴 문제를 생각함에 있어서 고려해야 하는 사항은 대략 다음과 같다.

1. **자유 글꼴 여부.** 한글 텍이 자유 소프트웨어에 크게 의존하는 한, 글꼴도 사용과 배포에 제한이 없는 자유 글꼴이 아니면 안 된다. 특정 기업으로부터 라이센스를 얻어 개발했던 hLaTeXp의 경우는 사용권의 제약 때문에 결국 글꼴 자체를 공개하지 못하였다.
2. **품위.** 아무리 자유 글꼴이라고 하나, 적어도 인쇄의 품질이 어느 정도 보장될 정도의 만족감을 주는 글꼴이 아니면 안 된다.
3. **범위.** 한글 글꼴은 하나의 자면(glyph)이 초성/중성/종성을 모두 조합한 완성형 글자 하나로 이루어진다. 그 결과, 하나의 폰트에 현대어의 모든 글자를 다 포함하게 되면 일만 자 이상의 자면을 가지는 폰트가 되어야 한다. 여기에 한자와 라틴 문자 및 특수 문자까지 고려하여야 하는 것이다. 그러나 이렇게 제작되는 폰트는 사실 드물고 대부분 완성형 2,350자만을 포함하는 폰트들이 다수 제작되어 유통되고 있다.
4. **포맷.** HLaTeX의 UHC 글꼴은 위의 요건을 대부분 충족하지만 가상 폰트를 이용한다는 점과 서브폰트 기법에 의하여 한글을 표현한다는 점 때문에 텍 이외에는 사용할 수 없었다. 시스템 폰트로도 사용할 수 있는 트루타입이나 오픈타입 형식의 글꼴이 필요하였다.

이러한 조건을 모두 충족하는 "은 글꼴"의 등장은 한글 텍 발전의 신기원을 이루었다고 할 만하다. 은 글꼴이 없었다면 현재와 같은 높은 수준의 한글 텍 시스템이 개발되는 것은 아주 어려웠을 것이다. 은 글꼴은 은광희의 UHC 글꼴을 박원규가 트루타입으로 재구성한 것으로서, GNU GPL 자유 소프트웨어 라이센스를 가진 폰트이다. ko.TeX은 당연히 이 글꼴을 표준 글꼴로 채택하고 있다.

현재 은 글꼴은 두어 가지 개발의 여지를 남겨두고 있다. 그 첫째는 은 바탕 글꼴에 오픈타입의 속성(feature)을 이용하여 중세 문자의 표현이 가능하도록 확장하는 것이다. 이 부분은 상당한 시도가 이루어지고 있다. 또 하나는 부족한 한자 부분을 보충하는 것이다. 한글 영역은 채워졌지만 한자는 여전히 KS X 1001의 4,888자에 그쳐 소위 확장 한자를 표현할 수 없다는 문제가 남아 있다.

이와 더불어, 텍 사용자들은 자신이 구매하여 가지고 있는 상업용 글꼴을 문서에 활용하길 원하고 있었다. 한글 폰트는 대부분 트루타입이었으므로 다양한 트루타입 폰트를 사용하여 문서를 제작하는 기술이 요구되었던 것이다. 이 분야에서는 특히 DVIPDFMx 드라이버의 개발과 더불어 많은 문제들이 해결되어 갔다. 그러나 pdfTEX 엔진을 사용하는 한 트루타입 폰트로부터 많은 수의 tfm 파일을 직접 추출하고 fd 파일을 만드는 복잡한 절차를 거쳐야 하는 문제가 최근까지 남아 있었다.[21] 트루타입과 오픈타입을 직접 다루는 XETEX과 LuaTEX이 마지막 남은 이 문제까지 엔진 레벨에서 해결함으로써, 이제 텍 사용에 있어 폰트 문제는 거의 완전히 해결을 보게 되었다.

앞으로 개선이 필요한 부분은 폰트 선택 체계의 통일이다. 현재 k*o*TEX의 euc 버전과 utf 버전은 폰트 지정 방식이 조금 다르다. 그리고 XETEX-k*o*와 LuaTEX-k*o*도 엔진의 차이 때문에 역시 조금씩 다른 폰트 선택 명령을 사용한다. 이것을 통일하거나 적어도 하위 호환성을 제공하면 좋을 것이다.

입력 문자의 처리 라텍은 유니코드 UTF-8 인코딩 문자를 처리하는 방법을 제공한다. 간단히 말하면 UTF-8 인코딩으로 들어오는 입력을 바이트 단위로 일단 나누고 이를 재정의하여 각 문자별 폰트에 대응시키는 방식이다. 한글 음절 문자 영역은 UTF-8로 인코딩하면 세 바이트가 되므로 hangul-ucs는 라텍의 이러한 처리방법에 준하여 한글을 세 옥텟열로 쪼개어 처리함으로써 유니코드 입력된 한글을 처리하는 데 성공했다.

유니코드 처리 기능이 없는 pdfTEX 엔진은 이와 같이 매크로 레벨에서 유니코드 입력을 처리할 수 밖에 없었다. 그러나 XETEX은 유니코드 문자 자체의 처리를 엔진 레벨에서 구현해준다. 오메가를 통해서 꿈꾸던 이상[22]이 보다 완전한 형태로 실현 가능하게 된 것이다.

예를 들어, 자동 조사 명령 \가를 HLATEX은 다음과 같이 정의했다.

```
\def\^^b0{\@ifnextchar^^ed\only@gt\ga@or@gwa}
   %% \only@gt 는 '고딕'이라는 명령을 정의하기 위한 것임
\def\ga@or@gwa#1{%
  \ifx#1^^a1{\@josa{가}{이}}\else%          가(B0A1)
  \ifx#1^^fa{\@josa{와}{과}}\fi\fi}%        과(B0FA)
```

이것은 EUC-KR의 '가'에 해당하는 코드 'B0A1'을 한 옥텟씩 쪼개서 정의해야 했던 흔적을 보여준다. '고딕'의 '고(B0ED)'와 조사 '과(B0FA)'가 모두 B0을 첫 옥텟으로 하기 때문에 이로부터 복잡한 분기가 불가피하였다. 반면, 이것을 XETEX에서는 다음과 같이 할 수 있다.

```
\def\가{\@josa{가}{이}} % 유니코드의 '가'는 [U+AC00]
```

XETEX이 유니코드 문자를 직접 다룰 수 있다는 것을 보여주는 한 예이다.

이제 한글 입력 문자를 하나의 폰트에 대응시켜서 식자하는 것 자체를 걱정해야 하는 시기는 지난 것으로 본다. 만약 한글 텍을 그저 "한글을 표시하는 것"에만 한정하여 생각한다면,

21. 이를 간편하게 배치 프로세스로 실행해주는 유틸리티가 k*o*TEX에 의해 제공되기는 했지만 여전히 번거로운 과정이 필요했던 것만은 변함이 없다.
22. 각주 18을 보라.

한글 텍 자체의 필요성이 없어진 것이나 마찬가지다. 실제로 ko.TEX과 같은 한글 매크로 없이도 XETEX만으로 한글을 식자하는 것은 얼마든지 가능하다. 그러나 한글 텍의 기능은 이를 넘어선 데 있다. 그것은 한글 자체의 표시가 가능해진 지금, "한글 문서" 다운 한글 문서를 제작하도록 하는 여러 기능을 정비하고 표준을 제시하는 것이다.

3.1.2 한글 문서다운 한글 문서

한글 문서의 아름다운 표현을 위해서 고려되어야 할 사항은 여러 가지가 있다. 이 대부분은 상위 수준 매크로로 구현될 수밖에 없는 성질을 가지고 있으나, 일부는 엔진이나 기본 매크로와도 관련되어 있다. 한편, 한국어 문서의 표준이라 할 만한 문서 작성 또는 출판 표준과 관행이 수립되어 있지 않은 경우도 많으므로 이러한 점에 대하여 일정한 방향이나 선택의 여지를 제공하는 것도 한글 텍의 기능 중 하나가 된다.

한글 문헌의 이름, 카운터, 장절표제 ko.TEX은 LATEX의 hangul 패키지에서 정의한 한글식 '이름'을 조금 수정하여 다음과 같이 정의하고 있다.

`\today`	년, 월, 일	`\contentsname`	차례
`\listfigurename`	그림 차례	`\listtablename`	표 차례
`\refname`	참고문헌	`\indexname`	찾아보기 (색인)
`\tablename`	표	`\figurename`	그림

장절표제는 대체로 '장'에만 접두어 '제'와 접미어 '장'을 붙이는 것을 기준으로 삼는데, 실제 편장절 체제의 일관된 표준이 없어서 곤란을 겪는 측면도 있다. 보통 편-장-절-소절의 체제나 장-관-항-목, 장-절-조-항-호 체제가 쓰이지만 편이나 장까지 한글식을 채택하는 방식의 글쓰기가 늘고 있다. 사용자의 문서 구성에 따라 쉽게 편장절 체제를 제어할 수 있도록 하는 매크로를 제공하는 것이 좋을 것이다. 표준 라텍 매크로와의 대응은 다음과 같다.

`\part`	편	–	제 1 편
`\chapter`	장	장(章)	제 1 장
`\section`	절	관(款)	제 1 절
`\subsection`	(소절)	항(項)	(1.1)

다른 한편 장절의 식자 방법도 논란거리인데, 우리 문헌에 있어서 특히 장 표제의 식자에 장식성을 요구하는 일이 많고 이 부분은 편집자나 디자이너의 의사가 크게 반영되는 것이므로 유의할 것은 좀 쉬운 디자인 도구를 제공하는 정도라고 하겠다.

카운터를 한글식으로 제어하는 것은 대부분 정의되어 있고, 특히 enumerate 환경에서 문단 번호를 붙이는 데 있어서 몇 가지 한글식 문단 번호를 설정하는 것을 용이하게 해주는 방법을 제공하게 되었다 [9]. 이 부분은 ko.TEX의 중요한 성과 중 하나이다.

참조와 인용, 색인 LATEX의 hbibtex은 한글 문헌 목록의 정렬을 도와주는 유틸리티였다. 그러나 문헌 목록의 형식이나 나열 방식 등을 모두 해결해주지는 못하였다.

k*o*.TEX에서는 문헌의 정렬 문제는 거의 해결된 듯하나, 문헌인용의 방식과 관련해서 아직도 해결해야 할 문제가 남아 있다. 그것은 한편으로는 문헌인용의 표준적 방식에 대한 합의가 없다는 점과, 한글식 문헌인용 방식을 요구하는 바가 크지 않았던 데서 기인하는 듯하다. 소규모의 논문이라면 문제가 없지만 특히 BIBTEX을 이용하여 문헌 데이터베이스를 관리해야 하는 경우, 구미 문헌과 한글 문헌을 하나의 문헌 목록에서 차등하여 다루거나 하는 문제가 충분히 해결되어 있지 않다.

현재는 영문 문헌목록에 약간의 수작업을 거쳐야 제대로 된 결과를 얻는 정도인데, natbib, latexbib 등의 여러 패키지를 검토하여 우리 문헌에 맞는 문헌 목록의 생산이 가능하도록 하려고 다양한 시도를 하고 있는 형편이다.

다만, 색인에 관해서는 어느 정도 해결책을 제시할 수 있다. komkindex 유틸리티를 이용하여 색인을 정렬하고 `.ist`를 통하여 색인 포맷을 결정하는 것은 손쉽게 가능하다.

조사의 자동화 자동조사에 대해서는 일찍부터 많은 관심이 기울여졌다. HLATEX의 자동조사 기능이 hyperref 패키지와 충돌을 일으키면서 한때 위기를 겪기도 하였지만 조사자동화 없는 한글 텍은 생각할 수 없다. 현재의 k*o*.TEX은 HLATEX을 거의 그대로 이어받은 euc 버전에서조차도 hfont만으로 자동조사를 처리해준다. HLATEX의 hfont 패키지는 자동조사 기능을 제외하고 있었다.

최근 개발된 LuaTEX-k*o*는 '자동조사는 완전하게 동작' 한다고 선언하고 있다. 루아 언어를 이용함으로써 이제 참조 번호뿐 아니라 일반적 문자열을 포함한 어떤 경우에도 자동조사가 제대로 작동하게 되었다. 자동조사 문제는 k*o*.TEX이 해결한 가장 중요한 문제이다.

페이지 스타일과 판면 페이지 스타일과 판면에 대해서는 oblivoir 패키지가 많은 기여를 하였다. 이 패키지는 hangul 페이지 스타일과 fapapersize라는 판면 조절 기능을 제공한다. 여백만을 설정함으로써 페이지의 크기나 범위를 쉽게 조절할 수 있고 한글 문헌에 적당한 면주 형식을 제공하고 있다. 이와 더불어, 한국어 문헌에 일반적인 각주 형식을 별도의 옵션으로 제공하기도 한다.

페이지 스타일이나 판면 자체는 편집자나 저자의 의도를 반영하여 쉽게 설정하거나 수정할 수 있으면 된다. 그러나 다른 한편 "표준적인" 양식의 옵션이 제공될 필요도 있다.

페이지 스타일, 판면 등은 k*o*.TEX의 핵심 기능에 해당하지는 않는다. 또한 종래의 한글 텍 매크로들도 이 부분에 대하여 어떤 기여를 하지는 않았다. 흔히 쓰이는 패키지를 이용해서 쉽게 조절할 수 있기 때문이다. 그렇지만 쉽게 변경할 수 있다는 것과, 기본값이 제공된다는 것은 전혀 다른 의미이다.

타이포그래피 한글 텍의 역사에서[23] 자간과 행간에 대한 관심은 매우 늦게 나타났다. 영문자의 경우 자간과 행간은 폰트의 속성에 해당하는 것으로 이를 인위적으로 조절할 필요가 크지 않았던 데 비해, 한글 문서의 경우 소위 '마이너스 자간' 을 적용하는 것이 현실이었다. 이

23. 한글 텍과 타이포그래피에 대한 전반적인 사항은, 이주호, "한글의 가독성과 k*o*.TEX의 타이포그래피", [12]를 참고하라. 이 글은 타이포그래피에 관하여 본 논문의 논지 몇 가지를 상세히 다루고 있다.

마이너스 자간은 현재 인쇄 출판의 거의 정칙이나 마찬가지로 받아들여지고 있다. 실제로 은 글꼴을 기준으로도 약간의 마이너스 자간을 적용하는 것이 문서의 전체적인 품위를 높이는 데 현저히 기여한다. HLaTeX에서 자간을 조절하는 매크로가 등장한 것이 2005년이었음을 생각하면 이 분야에서 한글 텍의 대응이 얼마나 늦었는가를 알게 된다.[24]

자간과 마찬가지로 단어간격도 중요한 문제였다. 영문자의 단어간격은 3분각 이하가 적당하지만 한글의 경우에는 반각은 너무 넓고 3분각은 너무 좁은 듯한 느낌을 주기 때문이다. 어간은 특히 행자름과도 깊은 관련을 지니고 있어 그 기준을 잘 정하는 것이 매우 중요하다.

자간과 어간은 oblivoir에 의하여 한글 문서에 적당하게 그 폭이 조절되었다.[25] 그러나 이것은 은 글꼴을 기준으로 한 것으로서 다른 폰트를 쓰면 그 값을 적절히 바꾸어주어야 한다. 따라서 자간과 어간을 쉽게 조절할 수 있는 사용자 인터페이스를 제공해야 하는데, ko.TeX에는 이것이 구현되어 있다.

행간은 일찍부터 한글 문서에 적당한 값으로 1.3이 제시되고 있었다. oblivoir는 1.333을 기본 행간으로 쓰고 있다. 워드 프로세서 한글에서는 이보다 더 큰 160% 를 기본값으로 제시하고 있는데, 이 값은 이론적으로 거의 배행간에 해당한다.

한편, oblivoir는 각주, 플로트, 인용문 안에서 넓은 행간이 아니라 좁은 행간을 적용하는 추가적인 기능을 제공한다. 특히 각주는 기본 행간을 그대로 쓰면 각주의 글자 크기가 작아지는 까닭에 행간이 지나치게 황량해보이는 결과를 초래한다. 각주와 관련해서, 문서의 기본 행간은 1.3으로 늘리면서 각주 사이의 간격은 바꾸지 않음으로써 각주 문단 자체의 간격은 넓고 각주 사이의 행은 좁아지는 불균형이 종래 한글 텍 문서에서 종종 등장했다. 현재도 문서 클래스를 article 등으로 하고 단순히 kotex 패키지만 얹은 경우에는 같은 방식으로 동작하므로, oblivoir를 쓰지 않는 한 이 문제를 사용자가 스스로 조절해야 할 필요가 있다.

타이포그래피는 궁극적으로 폰트와 연관되어 있는 것이다. 폰트 사용의 제약이 없어진 이상, 폰트 사용의 모든 책임이 편집자나 디자이너 또는 저자에게 넘어오게 되었다. pdfTeX을 기본 엔진으로 사용하면서 트루타입을 사용할 수 있게 되기는 했지만 여전히 자유로운 폰트 활용이라고 말한 만한 것은 못 되었다. 적어도 복잡한 tfm 추출 등의 절차를 거쳐야 했기 때문이고, 특히 장평의 자유로운 조절이 불가능했기 때문이다. 폰트 활용과 관련하여 최근 나타난 변화를 정리하면 다음과 같다.

1. 트루타입, 오픈타입을 포함하여 시스템 폰트를 다른 조작이나 tfm 없이 바로 사용할 수 있게 되었다.
2. 폰트를 이름으로 참조하거나 파일이름으로 참조하는 것이 가능해졌다. 종래의 NFSS 방식과는 폰트를 지정하는 방법이 달라졌다. (물론 NFSS 방식도 여전히 동작한다.)
3. 오픈타입의 속성들을 활용할 수 있게 되었다. 그 결과 폰트의 폭, 변형 등을 인자를 조작하여 지정할 수 있다. 종래 거의 불가능하거나 복잡한 과정을 거쳐야 하던 fakebold

24. 2003년에 필자는 hlatex-interword 패키지를 통하여 HLaTeX에서 자간을 조절할 수 있도록 하는 추가 패키지를 제공했다. 여기서 말하는 2005년은 HLaTeX 자체에 해당 매크로가 갖추어졌음을 말하는 것이다.
25. oblivoir의 제작 목적 중의 하나가 한글 문서에 적당한 타이포그래피의 기본값을 제공하고 타이포그래피 요소를 지정하거나 변경하는 방법을 제공하는 것이었다. [11] 참조.

나 장평 조절 등이 손쉽게 가능해졌다.

4. 특정 폰트에 없는 글자를 다른 폰트를 통해 식자할 수 있다. 없는 문자를 찾는 순서를 미리 정해두면 된다.
5. 한 문서에 언어별로 다른 폰트를 쓸 수 있다. 그래서 중국어/일본어가 뒤섞인 복합언어 문서도 손쉽게 작성할 수 있다.
6. 폰트 자체의 자간/어간을 반영해 식자 할 수 있으므로 폰트 디자인을 더 잘 활용한다.
7. 문서의 기본 폰트 패밀리 rm/sf/tt 이외에 임시로 특정 폰트를 일시적으로 이용하는 것이 쉽다. 장식적 문서에서는 아주 요긴한 기능이다.

행자름과 금칙처리 한국어 문장의 행자름에 대해서 몇 가지 논점이 있다.

1. 원칙적으로 모든 문자 사이에서 행을 나눌 수 있다.
2. 영문자나 숫자와 함께 쓰는 경우, 영문자는 영문 자체의 하이프네이션이 동작해야 하고, 숫자나 수식은 원칙적으로 행으로 나뉘지 못한다.
3. **고아 회피.** 한 글자 한 행은 피하는 것이 좋다.
4. **행말 금칙.** 여는 괄호, 여는 따옴표 앞에서는 행을 나눌 수 있으나 뒤에서는 나누지 못한다.
5. **행두 금칙.** 닫는 괄호, 닫는 따옴표, 마침표, 쉼표, 느낌표, 가운뎃점 등의 문장 부호는 그 앞에서 행을 나눌 수 없다.

이 중 기존의 한글 텍에서 문제가 되던 것은 주로 2와 3이다. 한국어 특성상 영문 단어를 쓴 뒤에 조사를 이어붙여야 하는데, HLaTeX은 한글 문자 뒤에서 행을 나누었기 때문에 조사가 앞말에 달라붙는 성질을 가지고 있었고 한글 단어 식자 시에는 영문자 하이픈이 억제되었기 때문에 결과적으로 영문 단어와 조사가 모두 판면 오른쪽으로 튀어나가 버리는 경우가 흔했다. 이것은 아주 치명적인 문제를 가져오는 경우가 많았다.

2의 문제는 ko.TeX에서 완전히 해결했다. 즉 영문 단어를 쓰고 그 뒤에 조사를 이어쓰는 경우에 영문 단어에 하이픈이 적용되고 조사나 괄호 앞에서 행이 잘리도록 한 것이다.

3은 이른바 '고아' 문제라 부르는 것으로, 논란의 대상이다. 최근 서적들의 조판에서 이 원칙은 엄격히 지켜지지 않는 것 같다. 고아 회피는 주로 판면의 색조가 너무 흐려지는 것을 방지하기 위한 것으로 전통적인 조판에서 흔히 적용되던 원칙이었다. 현재 고아 회피는 부분적으로 이를 적용하도록 하고 있다. 고아 회피를 어느 정도 적용할 것이냐 하는 데 있어서 단계를 두어 사용자가 선택할 수 있도록 개선하면 좋을 것이다.

ko.TeX은 한글 행자름 문제를 근본적으로 해결함으로써 한글 텍의 발전에 있어 중요한 진보를 이루었다. 또한 행자름은 단어 간격과도 연관되어 있으므로 표준적인 단어 간격을 제시함으로써 행자름에 의한 판면의 미적 측면의 개선에 기여하려 하고 있다.

문장부호, finemath, 간격 문장부호의 조판에 있어 우리 글 문서의 가장 현저한 특징은 영문 폰트의 문장부호를 그대로 가져다 쓴다는 점이다. 그 결과 한글 글꼴과의 약간의 위화감이 판면에 생겨나는데, 이를 지적하고 영문 폰트 문장부호의 수직 위치를 조절하는 문제를 제기한 것은 김강수의 글 [3]이었다. 이 제안이 받아들여져서 ko.TEX은 문장부호, 특히 마침표의 수직 위치를 한글 베이스라인에 맞게 수정하였다. 그리고 그 결과 이전과는 질적으로 다른 조판 결과를 얻을 수 있게 되었다.

finemath는 원래 수식을 한글과 섞어쓸 때 발생하던 간격 문제에서 왔다. 예를 들어 "$x+y$가" 라고 문장 중에 수식을 쓰는 경우에 y와 그 뒤에 이어지는 조사 '가' 사이가 보통 간격으로 주어지면 필연적으로 수식 자체의 가독성이 떨어지는 결과를 가져왔다. 수식이 매우 중요한 문서에서 이것을 개선하기 위하여 종래 수식과 조사 사이를 띄어쓰는 관행이 사용된 적이 많은데, 이 또한 조사와 수식의 간격이 너무 멀어서 문제가 되었다.

조진환과 김도현은 ko.TEX에 finemath라는 새로운 기능을 도입하였다 [4]. 즉, 문장 중의 수식, 괄호, 숫자, 영문자와 한글 문자 사이에 미리 정해진 미세한 간격을 주자는 것이다. 마이크로소프트 워드와 같은 워드프로세서에서도 영문자나 수식과 한글 사이에 스페이스를 넣는 기능이 있다. ko.TEX에서 사용하는 finemath 간격은 워드에서 처리하는 것보다 더 좁고 섬세하여 문서를 매우 세련되게 보이게 하는 데 기여하고 있다. 과학 기술 문헌의 조판에서 ko.TEX이 가지는 중요한 강점 중의 하나가 이 finemath가 될 것임을 의심치 않는다.

마지막으로, 아직 완전히 해결하지 못한 문제가 하나 있는데, 그것은 이른바 수직 간격 문제이다. 한글 문서의 기본 행간을 1.333으로 늘림으로써 디스플레이 수식과 한글 문단 사이의 간격이 의도하지 않게 넓어지는 결과를 초래하였다. oblivoir 패키지는 adjustmath 옵션을 통해서 시험적으로 이 간격을 조절하고자 하였으나 충분히 만족스러운지에 대한 확신이 없어 시험적으로만 테스트하고 있다. 수식이 간단한 경우와 달리 소위 큰 수식기호가 있는 경우에는 오히려 약간 넓은 쪽이 더 나아 보일 수도 있기 때문이다. 이 간격은 매크로 레벨에서 쉽게 조절할 수 있으므로, 다양한 의견이 개진되어 표준값을 제시할 수 있게 되기를 희망한다.

마이크로타이포그래피 마이크로타이포그래피(micro-typography)[26]는 pdfTEX의 제작자인 한 테 탄의 논문 [17]에서 다루어진 후 pdfTEX의 중요한 특징으로 자리잡게 된, 현대 텍 기술 발전의 표징 중의 하나이다. 이것은 판면의 오른쪽 끝을 가지런하게 정렬하기 위하여 문장부호를 여백 쪽으로 조금 밀어넣는 '마진 커닝'과 문단 중의 스페이스 폭을 되도록 균일하게 유지하기 위하여 폰트의 장평을 극도로 미세하게 조절하는 '폰트 확장'으로 이루어져 있다. 즉, 마이크로타이포그래피를 적용함으로써 문단의 배열과 스페이스 간격이 균일하게 처리됨으로써 판면의 미적 안정성이 극적으로 향상되게 하는 결과를 가져온다.

독자적인 문장부호를 사용하고 띄어쓰기가 없는 일본어나 중국어의 경우 마이크로타이포그래피의 요구가 크지 않을 수 있으나, 띄어쓰기 즉 스페이싱을 사용하는 우리 문장의 경우는 이 마이크로타이포그래피를 적용함으로써 얻을 수 있는 것이 대단히 많다. ko.TEX은

26. 마이크로타이포그래피 자체에 대해서는 [18]를 보라. 텍으로 이를 구현하는 문제에 대해서는 [17]이 독보적이다.

pdfTEX과 LuaTEX 엔진 하에서 마이크로타이포그래피가 자동적으로 동작하도록 되어 있다. 다만, XETEX은 아직 이 기능을 엔진에 탑재하고 있지 않은데 조만간 이 방향으로의 발전이 있을 것이라는 기대가 있다. ko.TEX 문서에서 마이크로타이포그래피 효과는 pdfLATEX이나 LuaLATEX으로 컴파일하면 그 결과를 볼 수 있다. 따라서 LATEX이나 XELATEX으로 컴파일한 것과 판면의 결과가 달라질 수밖에 없다.

pdf ko.TEX이 개발되던 시기는 pdfTEX이 주력 텍 엔진으로 자리잡던 시기와 일치한다. 최종 출력 포맷이 pdf로 거의 굳어져 가고 있던 시점이므로 pdf 친화적인 기능들에도 많은 노력을 기울이게 되었다. HLATEX의 hangul 패키지를 얹으면 아예 pdf에 책갈피 등을 구현할 수 없었던 것과 비교해보면 실로 중요한 변화가 있었다고 볼 수 있다.

1. 한글 문자의 검색과 추출이 가능한 pdf를 제작할 수 있게 되었다. 종래 한글 텍으로는 이런 것이 불가능했으므로 완전한 의미에서 온라인 문서로서의 pdf를 제작할 수 없었던 것과 비교해 볼 수 있다.
2. pdf 책갈피를 hyperref 패키지를 통하여 삽입하는 것이 가능해졌다.
3. DVIPDFMx 드라이버에 힘입어 pdf 스페셜 명령을 자유자재로 활용할 수 있게 되어, 필요하다면 고급 pdf 기능인 자바스크립트, 인터랙티브 pdf, 멀티미디어, 어노테이션 등을 활용할 수 있다.

이를 통하여 ko.TEX은 한글 pdf 제작 툴로서도 매우 훌륭한 결과물을 보여주게 되었다.

세로쓰기와 중세국어 세로쓰기 조판은 그 수요가 많지 않지만 꼭 필요할 때가 간혹 있다. 종래 한글 텍에서는 세로쓰기 조판을 고려한 적이 없었다. 한글텍사용자그룹에서 다양한 실험이 이루어지던 무렵, 폰트를 회전시켜 세로쓰기 조판을 구현하는 시도들이 있었는데, 현재 LuaTEX-ko는 마침내 세로쓰기 자체를 패키지의 기능으로 구현하게 되었다.

중세국어의 조판은 거의 해결된 상태이다. pdfTEX 엔진 하에서도 kotex-midkor를 통하여 유니코드 표준을 따르는 모든 글자(수백만 자)를 다 표현할 수 있으며 [10], LuaTEX-ko와은 글꼴을 통하여 이를 더 발전시켜나가고 있다. 다만 중세국어 글자의 미적 품위가 소위 한양 폰트 PUA를 이용하는 것에 비하여 조금 떨어지는 흠은 있으나, 이 문제는 폰트 수준에서 개선이 필요하다.

다국어 한글 문서에서 주로 고려되는 다국어는 영문이나 라틴 문자 계통이 아니라 일본어나 중국어를 함께 쓰는 경우의 조판이다. 그 이유는 아랍, 키릴, 헤브루 문자 등을 표현하거나 한글과 함께 쓰는 데, 적어도 유니코드를 사용하는 한 큰 장애가 없기 때문이다.

일본 문자(한자와 가나)를 식자하는 그 자체는 아무런 문제가 없다. XETEX이나 LuaTEX으로 이 문제는 (적절한 폰트만 있다면) 이미 해결되어 있다. 그러나 예컨대 일본어를 한글 문서에서 활용하는 데 있어서 단순히 글자를 표현하기만 하면 되는 것이 아니라 일본어 조판 규칙을 어느 정도 따르지 않으면 안 된다. 일본어의 경우 행 길이가 아주 중요한 문제로서 글자

수에 따라 행길이를 정해야 하는데 우리 문서는 띄어쓰기의 덕분으로 행길이에서 자유롭기 때문에 이를 조화시키는 것도 매우 중요하며, 일본어의 문장부호 간격을 적절히 처리하는 것도 아주 어려운 일이다.[27]

기술적으로 이 복잡한 문제에 대한 돌파구는 LuaTEX이라는 강력한 프로그래밍 도구를 갖춤으로써 극복할 수 있을 것으로 본다. 현재의 LuaTEX-k*o*에서는 중국어와 일본어 문장을 제법 높은 수준의 품위로 식자할 수 있는 환경을 개발해가고 있는 중이다.

3.2 사회적 필요조건

한글텍사용자그룹의 형성은 위에서 언급한 대로 한글 문서의 조판에 있어서 획기적이고 중요한 변화를 가능하게 한 공동체적 토대가 되었다. 그 이전까지 텍이 주로 수학, 물리학을 비롯한 과학 문헌의 저술에 사용되었으며 그나마도 한글의 사용은 그 비중이 매우 낮았던 것에 비교하면 한글텍사용자그룹을 통하여 이제야말로 한글 문서를 텍을 이용하여 처리할 수 있다는 자신감을 얻게 되었다고 할 수 있는 것이다.

이 소절에서 문제삼을 것은 다음과 같다.[28]

1. 텍 사용자의 공동체는 형성되어 있는가?
2. 텍 사용자들은 한글 텍 기술을 신뢰하고 이를 자신의 저작에 활용하고 있는가?
3. 장단기적이고 안정적인 사용자 지원의 바탕은 형성되어 있는가?
4. 인쇄 및 출판에 있어서 텍은 그 도구로서의 일정한 지위를 획득하였는가?
5. 사용자층의 재생산과 교육 지원은 이루어지고 있는가?

3.2.1 사용자

먼저 텍 사용층에 대한 것이다. 현재 텍 사용자층은 대략 몇 부류로 나누어볼 수 있다.

1. 한글 텍을 크게 필요로 하지 않지만 저작에 텍을 충분히 활용하고 있는 경우. 주로 수학자, 물리학자, 통계학자, 공학자들이 이에 해당하고 텍의 수식 표현 능력에 관한 한 대안을 찾지 못한 사람들이다. 논문을 작성해야 하는 대학원생 등도 여기에 속한다.
2. 전문적인 텍 사용자는 아니지만 이를 저작도구의 하나로 활용하고 텍 자체에서 재미를 얻고 문서작성, 웹사이트, (소규모) 데이터베이스 등에서 그 활용 목적을 발견하는 경우. 주로 학생층이나 아마추어 사용자가 여기에 해당할 것이다.
3. 직업적인 출판에 이용하는 경우. 주로 과학문헌의 저작으로서, 저자의 자가출판에 가까운 형식으로 이루어진다. 그 이유는 텍 출판에 관한 전문적인 사업체나 기관이 국내에 없기 때문이다.

27. 일본어 조판의 요구사항과 그 구현에 대해서는 2008년 한국텍학회 학술대회의 오쿠무라 하루히코 교수의 발표 [16]을 통해 자세히 알 수 있다.
28. 이 소절의 성격상 제기된 문제에 대하여 사회조사적 성격의 연구가 요청되는 측면이 있다. 이 글은 전반적인 의미에서 연구의 방향을 제안하려는 성격이 강하므로 경험적인 입증은 추후의 과제로 미룬다.

4. 텍과 데이터베이스를 연동하여 웹사이트를 운영하거나 출판을 통하여 수익을 얻는 사업에 이를 활용하는 경우.

어떤 경우든 그 기반이 폭넓은 편은 아니다. 그러나 위의 모든 부류의 사용자들이 현재는 한국텍학회가 지원하는 한글텍사용자그룹으로부터 필요한 대부분의 정보를 얻게 된 것으로 보인다. 공동체로서의 연대성은 대단히 취약하지만 텍 사용상의 문제를 대부분 해결할 수 있는 온라인 커뮤니티가 형성되는 데는 성공했다고 판단할 수 있다.

3과 관련하여, 저자의 자가출판에 의존하도록 되어 있는 현재의 실정은 개선되어야 한다. 북디자인에서 조판에 이르기까지 텍을 도구로 하는 출판의 전 과정에 적절한 서비스를 제공할 수 있어야 할 것이다. 텍으로 만들어진 서적이 내용은 차치하고 매력적인 책으로 다가오지 못하게 된다면 이것은 탁월한 조판 도구인 텍으로서는 스스로 그 용도를 제한하는 결과를 가져오는 것이다. 이를 위해서는 표준적인 저작 도구를 제공하고 그 효용성에 대한 정보를 확산시키는 것이 불가결하다.

3.2.2 사용자 지원

한국텍학회와 한글텍사용자그룹은 한글 사용이 가능한 텍시스템을 개발하여 배포해왔고 그 지원을 꾸준히 해왔다. KTUG Collection이라고 불리는 한글텍사용자그룹의 텍 시스템은 현재 사실상 유일한 표준 한글 사용 환경으로 정착하였다.

KTUG Collection은 처음에는 MiKTEX과 HLATEX을 기반으로 해서 시작되었으나, 곧 한글화에 적합한 독자적인 배포판의 체계를 갖추어 나갔다. TEX Live나 W32TEX을 기초로 다양한 유틸리티와 KCmenu로 대표되는 사용자 인터페이스 환경과 에디터를 제공함으로써, 단일 설치 즉시 사용이라는 편리한 환경을 제공하는 데 어느 정도 성공하였다. 이것은 MiKTEX의 편리한 패키지 관리 시스템과 TEX Live의 표준적이고 강력한 텍 시스템을 결합한다는 아이디어에서 출발하여 어느 정도 성과를 거두었다고 볼 수 있다. 현재 개발이 진행중인 TEX Live 2009는 이전에 KTUG Collection이 제공하던 대부분의 기능을 자체적으로 제공하게 되었으나, 더 편리하게 한글 사용 환경을 이용할 수 있도록 하는 지원은 계속될 것이다.

KTUG Collection의 기여 중 하나는 문서화에 관련된 것이다. 한글텍사용자그룹은 상당한 양의 사용자 설명서와 샘플 문서를 제작하고 배포해 왔다. 사용자 설명서로서는 lshort-kr, memucs-hangul, kotexguide 등이 있고, KTUG Collection과 함께 제공되던 kc2008-guide 등도 변화하는 텍 환경에 대한 정보를 일부분 제공하였다. 그러나 문서화 자체가 상당한 노력과 시간을 요구하는 과정인 데다 텍 시스템과 한글 텍의 발전 속도가 너무나 빠르다보니 이를 따라잡지 못하여 이전의 낡은 정보를 신속하게 갱신하지 못하는 면이 많다. 사용자 지원의 한 형태로서 문서화에 더 많은 관심을 가지게 되기를 희망한다.

3.2.3 한글 텍 기술에 대한 신뢰

많은 텍 사용자들이 한글 텍의 발전에 대해서 그 상세한 내용을 파악하지 못하는 것으로 보인다. 또한, 위에서 자세히 적은 한국어 문헌 조판에 필요한 요소들을 잘 이해하지 못하거나

필요로 하지 않는 경우도 많은 듯하다. 여전히 HLATEX을 자신의 주된 한글 사용방식으로 하고 있는 경우가 많은 것은 그래서일 것이다.

실제로 ko.TEX은 최신 기술을 지나치게 선도적으로 도입해 온 감이 없지 않다. 예를 들면 KC2008이 제공하던 ko.TEX 개발 버전은 TEX Live 2008에서 추가적인 실행 프로그램 패치가 없이는 컴파일이 불가능할 정도이다. 일반 사용자 입장에서는 이렇게 극도로 첨단화된 기술을 조기 도입하는 패키지의 개발 과정을 그대로 따라가기도 벅차거니와, 안정화될 때까지 기다리자는 심리가 작용할 수 있다. 그러나 이미 지적한 대로 한글 문서 작성에 있어서 ko.TEX의 변모는 이전에 상상하기 어려웠던 새로운 기능들이 추가되어 가는 경향이 있어서 안정 버전을 고정화하기 힘들었다.

한글 텍의 변모에 대한 교육 내지 홍보 프로그램이 개발되고 이를 더 잘 전달할 수 있는 체계가 요구된다고 하겠다.

3.2.4 인쇄 출판에의 적용 가능성

이주호는 텍을 이용한 출판 경험을 요약한 바가 있다. 이 보고에 의하면 생각보다 많은 책들이 텍을 이용하여 출판되었다. 텍 출판은 거의 대부분 수식이 많은 전문적이고 과학적인 분야의 책들로 이루어져 있다. 한국텍학회의 학술대회 때 텍으로 출판된 책들의 간단한 전시회가 열렸는데, 여기에서 일부는 매우 고품위의 조판 상태를 보여주어 텍으로 조판하는 것에 대한 일각의 회의적인 시선을 불식하는 데 기여한 바가 있다.

텍을 출판에 이용하는 데 있어서 종래 문제로 인식되던 것은 대략 다음과 같다.

1. 텍은 정형화된 디자인밖에 사용할 수 없다는 오해.
2. 텍은 폰트를 선택하거나 설정할 수 없다는 오해.
3. 텍 원고를 수정하거나 교정하는 것이 어렵다는 점.
4. 출판 현장에 텍 원고나 텍 문서를 다룰 전문가가 부족하다는 점.

이 가운데 전문가의 부족은 현재로서는 피할 수 없는 문제이다. 비근한 예로 텍이 제작한 pdf를 제대로 출력할 수 없는 경우 이런 문제에 대처할 전문가가 부족한 것이 현실이다. 그러나 첫째와 둘째 문제는 이미 해결된 것이고 수정이나 교정의 어려움은 저자의 관여가 출판 과정에서 지나치게 큰데 비해 저자 자신이 디자인의 전문가일 수 없다는 점이 함께 작용하고 있으므로 이에 대응할 적절한 서비스를 필요로 한다고 하겠다.

기능적 측면에서만 보면 한글 서적 출판 도구로서 텍의 성능은 다른 어떤 도구에 못지 않다. 다만 다루기 어렵고 잘 아는 사람이 적다는 것이 결정적인 걸림돌일 따름이다.

실제 출판에 있어서 텍의 활용성은 현재 극도로 낮은 편이다. 많은 학술지들이 텍을 저작 도구로 활용하고 있지 않거나 포기해가고 있으며 단행본의 출간에 있어서도 부득이한 경우 예외적으로 처리하고 있는 것이 현실일 것이다.

다만 고무적인 것은, 수학 교재와 같이 다량의 데이터베이스를 처리하는 데 있어서 프로그래밍 언어로서의 성격을 충분히 발휘할 수 있다는 점을 이용한 용례가 증가하고 있다는 점 정도일 것이다.

출판 현장에서의 요구를 즉시 수용하고 이를 반영할 수 있는 체계적인 지원 시스템이 필요하다. 또한 저자들이 자가출판의 부담을 안지 않고 저작활동을 할 수 있도록 하는 클래스, 스타일의 제공에서 조판과 최종 출력에 이르기까지의 전과정-지원이 필요할 것이다. 당장 이 분야는 수익을 창출할 정도의 규모를 가지고 있지 못하므로 비영리적인 접근이 요구된다고 보겠고, 이 역할을 한국텍학회가 수행해야 하지 않겠는가 조심스럽게 제안한다.

4 결어

이상에서 살펴본 바, 지난 10여년간 한글 텍은 극적인 발전과 변모를 겪어왔다. 타이포그래피에서 사용환경에 이르기까지 당시 제시되었던 요구를 거의 대부분 충족하게 된 것이다. 이제 한글 텍은 그 "필요조건"을 거의 만족하고 있다고 보아도 될 것이다. 여기에 텍 자체의 강점이 충분하므로 한글 텍의 발전 가능성은 무한하다고 하겠다. 이제 텍 자체가 가지고 있는 진입장벽을 조금 낮추고 그 현실적 활용가능성을 계발하는 것이 중요한 문제로 대두되었다.

한편 한글 텍 환경은 급격한 변화의 시기이기도 하다. 향후로는 LuaTEX에 기초한 *ko*.TEX이 한글 텍의 주류가 될 것이라고 추측할 수 있다. LuaTEX 자체가 현재 개발중인 엔진이므로 언젠가 이를 사용하여 만족스러운 결과를 얻게 되겠으나, 이 글을 쓰는 현재 시점에서 이를 대중적으로 사용하도록 권하기에 충분한 테스트를 거치지 못했다는 의견이 있다. 반면 XETEX은 손쉽게 한글 문서를 생산해내는 데 적용해도 좋을 정도로 발전해 있다고 판단한다.

텍 시스템의 발전에는 헌신적인 개발자도 중요하지만 테스터의 존재도 필수적이다. 더 많은 분들이 한글 텍 발전에 관심을 가지고 현재 진행되고 있는 개발 과정에 관심을 기울이고 피드백한다면 조만간 더 훌륭하고 완벽한 한글 조판 도구를 가지게 될 것이다. 이와 더불어 텍 사용층이 확산되고 실제 출판에서 그 진가를 발휘하게 되는 때가 오기를 기대해 마지 않는다.

참고 문헌

1. 고기형 · 한국과학기술원 산업수학연구실, 《한글과 TEX》, 청문각, 1995.
2. 김강수, 한글 라텍에서 조사 이형태 교체의 자동화에 관하여, The Asian Journal of TEX **1** (2007), no. 2, 153–166.
3. _____, 한글 문장부호의 조판 관행에 대하여, The Asian Journal of TEX **1** (2007), no. 1, 17–30.
4. 김도현, 유니코드 *ko*.TEX에서 finemath 기능의 구현, The Asian Journal of TEX **1** (2007), no. 2, 135–151.
5. _____, LuaTEX-*ko* 간단 매뉴얼, `http://ftp.ktug.or.kr/KTUG/texlive/texmf-dist/doc/lualatex/kotex-dev/luatexko/luatexko-doc.pdf`
6. 류성준, 수식편집기의 대명사, TEX: TEX과 한글이 만나다, 《마이크로소프트웨어》, 1994년 1월호, pp. 153–188.
7. _____, 한글 윈도우용 한TEX 1.5 평가판, 《마이크로소프트웨어》, 1995년 8월호, pp. 422–428.
8. 은광희, 한글 라텍 길잡이, 버전 1.0.1, 2005. `http://project.ktug.or.kr/hlatex/hlguide.pdf`
9. 은광희 · 김도현 · 김강수, 한국어 텍 *ko*.TEX v0.1.0 사용 설명서, 2007. `http://project.ktug.or.kr/ko.TeX/kotexguide.pdf`

10. 이기황, Hangul-ucs를 이용한 옛한글 조판, The Asian Journal of TEX **1** (2007), no. 1, 31–43.
11. _____, Oblivoir를 이용한 문서 작성, The Asian Journal of TEX **1** (2007), no. 2, 123–134.
12. 이주호, 한글의 가독성과 ko.TEX의 타이포그래피, The Asian Journal of TEX **2** (2008), no. 2, 69–113.
13. 조진환, 한글 라텍이 걸어온 길과 ko.TEX, The Asian Journal of TEX **1** (2007), no. 2, 113–121.
14. 최우형, 한글 LATEX Technical Guide, 1992.
15. Donald E. Knuth, *The TEX book*, Addison-Wesley, 1986.
16. Haruhiko Okumura, *pTEX and Japanese Typesetting*, The Asian Journal of TEX **2** (2008), no. 1, 43–51.
17. Hàn Thế Thành, *Micro-typographic extensions to the TEX typesetting system*, Ph.D. thesis, Masaryk University, 2000.
18. Herman Zapf, *About micro-typography and the hz-program*, Electronic Publishing **6** (1993), no. 3, 283–288.