Truetype 글꼴 외곽선의 임의 패턴 분할

서명배°, 지용준, 이성태, 김판구, 이윤배
조선대학교 대학원 전자계산학과 멀티미디어 시스템 연구실
e-mail : kisses@infoman.chosun.ac.kr

Pattern Division of Truetype Font Outline

Myoung-Bae Seo° ,Young-Jun Ji, Sung-Tae Lee, Pan-Koo Kim, Yun-Bae Lee
Multimedia System Lab, Dept. of Computer Science,
Graduate School, Chosun University

요약

Truetype 글꼴은 외곽선정보를 벡터 데이터 형태로 가지고 있기 때문에 사용자의 수정이 매우 용이한 글꼴이다. 본 논문에서는 이러한 truetype 글꼴의 특성을 이용하여 글꼴의 외곽선 정보를 알아낸 후 외곽선을 다양한 패턴으로 분할하는 방법을 제시하려 한다. 직선에서는 Brensenham 알고리즘을 이용한 패턴 분할을 행하였고, 곡선에서는 Casteljar 알고리즘을 바탕으로 선형보간법을 이용한 재귀호출 방법을 사용하여 패턴 분할을 행하였다. 이러한 임의 패턴 분할을 이용해서 다양한 길이를 가진 형태의 패턴을 생성함으로써, 글꼴의 외곽선에 다양한 효과를 줄 수 있다.

1. 서론

글꼴은 인쇄하거나 또는 화면에 보여질 수 있도록 특별한 스타일과 크기로 되어 있는 서식문자들을 말하는 것으로 글꼴에는 비트맵 글꼴, 트루타입 글꼴, 그리고 포스트스크립트 글꼴로 크게 나누어진다. 비트맵 글꼴은 주로 디스플레이용 글꼴로 쓰이는데 모니터상에서는 글자를 확대하면 외곽선이 깨끗하게 나오지 않고 프린터를 이용하여 출력할 경우에도 프린터가 그 서체를 가지고 있지 않는 경우 깨끗하게 출력되지 않는다. 그와는 달리 트루타입 글꼴은 비트맵 글꼴에 비해서 파일 사이즈가 커지는 대신에 글꼴의 외곽선 윤곽 정보를 가지고 있기 때문에 글자를 확대하더라도 Device 에 비종속적이기 때문에 깨끗하게 나온다. 실제로 Truetype 글꼴은 외곽선 정보를 벡터 데이터로 가지고 있는 매우 유연성이 좋은 글꼴이고, 프로그래머가 응용하기에 다양한 분야로의 접근을 시도할 수 있다는 장점을 가지고 있다. 일단 글꼴의 외곽선 데이터를 패턴분할하기 위해서 직선일때와 곡선일때를 나누어 처리한다. 글꼴의 외곽선 데이터의 값을 구한 후 직선일때는 Brensenham 알고리즘을 이용해서 임의의 길이의 패턴분할을 하였고, 곡선에서는 Casteljar 알고리즘을 바탕으로한 선형 보간법을 이용한 재귀호출 방법을 사용하여 임의의 패턴분할을 행하였다. 예를 들면 히읏(ㅎ)의 경우에서 위의 두 획은 직선에서의 패턴분할방법을 사용하였고, 아래의 이응(ㅇ)은 곡선에서의 패턴분할을 이용하였다.

2. 글꼴 외곽선의 임의의 패턴분할

Truetype 글꼴의 외곽선은 벡터데이터로 되어 있기에 본 논문에서는 벡터데이터의 임의 패턴분할에 중점을 둔다. 일단 글꼴의 외곽선 데이터를 베지어 곡선형태로 변환 후 사용하기로 한다. 베지어 데이터는 4개의 점으로 이루어지며, 2개의 양 끝점 데이터와, 그 두 점의 방향성과 길이 데이터를 가지고 있는 2개의 컨트롤 포인트로 이루어져있는데, 이 두 개의 컨트롤 포인트를 이용해서 쉽게 곡선을 수정할 수 있다는 장점을 가지고 있다. 때문에 본 논문에서는 베지어 커브에서의 임의의 패턴분할을 어떻게 행할 것인가에 대해서 중점적으로 다루어진다. 또한 임의의 패턴분할을 위해서 주어지는 패턴은 unsigned integers 두 개가 한 쌍으로 구성되어 지며, 각 숫자가 라인조각의 길이를 가리키고 있고, 두 번째 숫자는 다음 라인까지의 gap을 나타내도록 설계했다. 단 패턴은 효율성 측면에서 6개로 제한한다. 예를 들면 dash-dot-dot 패턴을 그리기 위해서는 아

래와 같은 패턴이 요구되어지고, 본 논문에서 제시되는 패턴의 형태는 다음의 형태를 따른다.

```
unsigned Pattern[] = {30, 10, 10, 10, 10, 10};
                     // Dash gap dot gap dot gap
```

2.1 Line 에서의 패턴 분할

라인에서의 패턴분할은 먼저 직선을 효율적으로 임의의 길이의 패턴으로 나눠야 한다. 또한, 두 점사이의 직선을 그릴 때 패턴에 맞추어서 그려야 하기 때문에 속도의 효율성면에서, 직선 그리는 알고리즘으로 가장 많이 쓰이고 있는 Brensenham 알고리즘을 사용해서 두 점사이를 보간해 가면서 일정한 패턴으로 분할을 행하였다.

2.2 Curve 패턴 분할

곡선에서의 패턴분할은 베지어 커브의 패턴분할을 통해서 이루어진다. 베지어 커브는 양끝 2점과 2개의 제어점으로 이루어지는데, 베지어 곡선을 임의의 길이의 패턴으로 나누기위해서는 먼저 베지어 곡선의 길이를 구해야 한다.

제어점 $Q_0,Q_1,\ldots, Q_n$을 가진 n차 Bezier 곡선 b(t)에서 곡선의 길이는 아래의 수식에 의해 구해진다.

$$\text{arc length} : L(b) = \int_1^0 b'(t)\,dt,$$

$$\text{polygon length} : Lp(b) = \sum_{i=1}^{n} |Qi - Qi-1|,$$

$$\text{chord length} : Lc(b) = |Qn - Q0|,$$

Casteljar's algorithm을 통해서 특정점에서의 곡선의 분할이 가능하다. 곡선의 분할은 아래의 수식으로 가능하다.

$$Q_i^0 = Qi, \qquad i=0,\ldots,n,$$

$$Q_i^0 = \frac{1}{2} Q_i^{k-1} + \frac{1}{2} Q_{i+1}^{k-1}, \qquad i=0,\ldots,n-k,\ k=1,\ldots,n$$

곡선에서는 Casteljau 알고리즘을 이용한 선형 보간법에 의한 재귀호출 방법으로 직선처럼 충분히 짧은 여러 개의 선으로 베지어 커브를 나눌 수 있다. 그것은 베지어 커브를 dash 나 dot로 구성되어진 여러 개의 짧은 곡선 조각으로 나누는 것도 역시 가능하게 한다. 3차 곡선 형태로 구성되어진 베지어 커브는 0과 1사이의 t 값으로 특징적인 표현이 가능하게 하며, 베지어 커브는 곡선 위 어느 점에서나 특징적인 t 값을 가지고 나눌 수 있다. 선형보간 후에 파라미터 t 값은 2등분되며, 이것은 베지어 커브 안에서 우리가 임의의 길이를 얻을 때 t 값을 사용할 수 있다는 것 또한 알 수 있다.

다음은 베지어 커브에서의 적절한 arc의 길이를 구하는 알고리즘이다.

```
float BezierLength(float b[4],float eps)
{
   Lp = poly_length(b);
   Lc = chord_length(b);
   n = degree(b);
   err = error();
   if(err<eps)
      return (2*Lc+(n-1)*Lp)/(n+1);
   else
   {
      b1, b2 = subdivision(b);
      eps1,eps2 = tolerance();
      return length(b1,eps1)+length(b2,eps2);
   }
}
```

3. 구현 및 실험

효율적인 패턴분할을 위해서 직선일 경우와 곡선일 경우를 각각 나누어서 클래스 설계를 하였다. 직선을 경우에는 CDashLine 클래스를 이용하였고, 곡선일 경우에는 LBezier 클래스를 이용하여 임의의 길이의 패턴 분할을 행하였다.

3.1 직선에서의 패턴분할을 위한 클래스 설계

SetPattern 과 GetPattern 함수를 이용해서 직선에서 구현할 수 있는 임의의 길이 패턴을 설정하고, 그 값을 얻어 올 수 있도록 하였고, m_Pattern 에 있는 임의의 패턴값을 이용해서 픽셀단위로 그려지는 두 포인트 위치 x, y를 brensenham 알고리즘을 이용해서 그릴 수 있도록 설계하였다.

```
Class CDashLine
{
 public:
   CDashLine();
   ~CDashLine();
   void SetPattern(unsigned* pattern, unsigned  count);
   static unsigned GetPattern(unsigned*
     pattern, bool round, unsigned pensize,
     unsigned style);
 protected:
   unsigned int   m_CurPat;
   unsigned int   m_Count;
   unsigned int   m_CurStretch;
   unsigned int*  m_Pattern;
   CPoint         m_CurPos;
   void Reset();
   void Bresenham(CDC* pDC,
```

```
        LONG x, LONG y);
}
```

4.2 곡선에서의 패턴분할을 위한 클래스 설계

아래는 t 값을 이용한 효율적인 베지어 곡선 위에서의 패턴 분할을 위한 클래스 설계 예이다.

```
class Lbezier
{
    public:
        LDPoint   p[4];
        void GetCPoint(CPoint* out);
        void TSplit(LBezier& Output, double t);
        void Split(LBezier& Left, LBezier& Right)
        const;
        double TAtLength(unsigned int& len)
        const;
        double TAtLength(double& len, double
        error = 0.5) const;
        double Length(double error = 0.5) const;
};
```

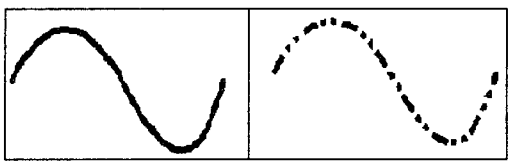

그림 1 패턴분할이 적용된 베지어커브

실제로 폰트의 아웃라인에 임의의 길이의 패턴분할을 적용한 예이다. 오른쪽그림은 왼쪽그림을 400% 확대한 그림 중 (ㅎ)의 윗부분이다.

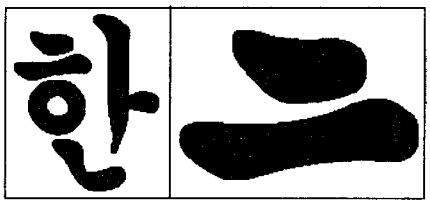

그림 2 Truetype font data

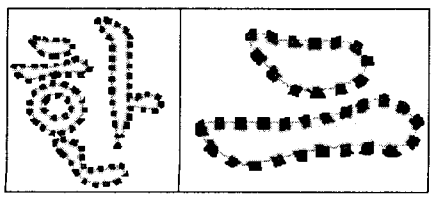

그림 3 Font data의 외곽선에 Dot pattern (· · · · ·)을 적용

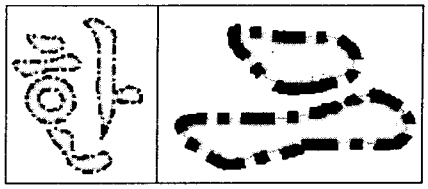

그림 4 Font data의 외곽선에 Dash dot pattern(— · · — · · —)을 적용

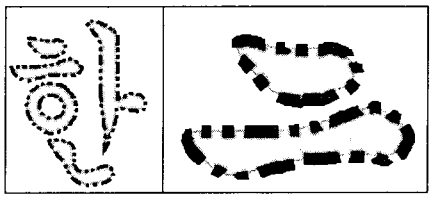

그림 5 Font data의 외곽선에 Dash dot dot pattern(— · · — · · —)을 적용

6. 결론

Truetype 글꼴에서 아웃라인 데이터를 이용한 임의의 패턴분할은 베지어커브의 길이분할을 이용해서 가능할 수 있었다. 실험결과 각 직선이 예각으로 만나는 경우 두 선분이 매끄럽게 연결되지 않는 현상이 발생하였고, 확대축소시 소숫점 연산과정에서 생기는 오차가 발생하여 원래의 벡터 이미지가 변형되는 현상이 발생하였다. 향후 연구과제로는 커브의 만나는 점에서의 매끄러운 처리방법과 화면 확대축소시에 발생하는 이미지 왜곡현상 개선이 필요하리라 본다.

참고문헌

[1]. www.kendeekis.com/tutorial7.html

[2]. "Graphics Gems V" (Editor Alan Paeth)

[3]. wdvl.com/Authoring/Graphics/Tools/PSP/bezier.html

[4]. Jens Gravesen: "Adaptive subdivision and the length of Bezier curves" mat-report no. 1992-10, Mathematical Institute, The Technical University of Denmark

[5]. T. Lindgren, J. Sanchez, and J. Hall. Curve tesselation criteria through sampling. In D.Kirk, editor, Graphics Gems III, pages 262-265, Academic Press, 1992

[6]. Guenter and R.Parent. Computing the arc lenght of parametric curve. IEEE Computer Graphics and Applications, 72-78, May 1990