Fontaine: Web 文字画像化と行内レイアウトシステムの開発

松田 聖大　　安村 通晃
{t05844sm, yasumura}@sfc.keio.ac.jp
慶應義塾大学 環境情報学部

概要

Web 上の文字の外観は閲覧環境に左右され、制作者の意図するフォントを自由に用いることはできない。書体を重要視する制作者たちは、局所的に文字画像を用いることで体裁を保ってきた。しかし、CMS のように更新頻度の高い動的なコンテンツを扱う手法が普及してきた今では、これまでの手作業による文字画像の作成は難しい状況にある。本研究は、以上の問題を解決するため、文字画像化と行内レイアウトを手段としたシステム Fontaine を提案する。これにより次の 4 項目の実現を目指す。(1) 閲覧環境にかかわらず文字が意図したフォントで表示されること、(2) 可変な表示領域で通常の文字と同等にレイアウトされること、(3) 既存のWebモデルに対して透過的であること、(4) 文字情報としてのユーザビリティを保つこと。本稿では、提案する文字画像利用モデルと手法、システムの設計と動作およびその実験･検証結果についての報告を行う。

Fontaine: An Automated Font Image Generation and Inline Typesetting System for Web

Shota Matsuda　　Michiaki Yasumura
Keio University, Faculty of Environment and Information Studies

Abstract

Because typography on the web depends on the environment of viewers, many web designers have been used text images for typographic consistency. With the applications of dynamic contents are spreading in recent days, it becomes difficult to make and embed text images into HTML document by hand. This study proposes an automated font image generation and inline typesetting system named Fontaine and its architecture, which aims for the realization of the following requirements: (1) cross-brower consistency and compatibility of typographic features, (2) metrical layout within mutable viewports, (3) transparency to the existing web model, (4) availability of literal information. In this paper, we will describe our proposed application of text images, the detailed implementation of Fontaine, and the result of some practical experiments in its performance.

1 はじめに

一般的な Web ページは HTML と CSS から構成される。この書類構造（HTML）と装飾（CSS）の分離によって、構造を変えるときにいちいち「何ポイントの何色」といった具体的なヴィジュアルに気を配らずに済むようになった。変更を加えるときは「その文章がどのような意味をもつか」を扱えばよいし、逆もまた然りである。また、近年普及し

てきたBlogを含める広義のCMS[1]は、テンプレートを巧みに操り書類内容と構造を分離し、更なる抽象化を達した。今やWebページの更新･追加は大した作業量を伴うものではなくなったし、日に数回更新されるものも少なくない。

現在、Web上の書体は閲覧環境に任されている。それは、文字をどのように表示するかについて閲覧者側に多くの自由度を与えているためだ。CSSではfont-familyという属性を用いてフォントの指定を行うが、これはフォントの優先順位を列挙するものであって、絶対的な指定ではない。これはフォント自体の通信が一般化していない状況下で多種の閲覧環境に対応するためには良策と言えるだろう。しかしながら一方で、細かなデザイン上の指定が行えなかったり、意図したように表示されないという問題も生じる。これはCSSの標準化を行ったW3Cも認める点である[1]。

この閲覧環境に依存した仕様は、「最も普及していて誰でも持っている書体」を使用しなければならないという制約をつくり出していると言えよう。

書体を重要視するWebデザイナたちは、局所的に文字画像を用いることでフォントを繋ぎ止めてきた。画像編集ソフトを用いて対応する文字を好ましいフォントとタイプセッティングにて描画し、適当な寸法で切り抜き、 適当なフォーマットで保存する。そしてHTMLにimgタグを挿入する。文字画像を用いることは閲覧環境への依存性をほとんど無視することができる。また、CSSには制定されていない柔軟なタイプセッティングも扱えるため、広く見受けられる方法である。

文字画像の利用にはもう一つの理由もある。それはアンチエイリアスを施した美しいレンダリングを保てることにある。20世紀のタイポグラフィの巨匠であるAdrian Frutigerは、判別性Legibility、可読性Readability、誘目性Inducibilityという、活字が備えるべき均整のとれた規範を導き出した[2]。これはスクリーン上のビットマップフォントにも受け継がれ、8･9･10･12･14･18･24ppemといった基準サイズが一般化し、グリッド吸着の技術も作られるようになった。ただし、充分にチューニングされたグリッド吸着点を持ったフォントは希であり、スクリーンフォントとプリンタフォントの溝は縮まっていない。QuartzやClearTypeなどのアンチエイリアシング技術が普及したことで、この理由での文字画像化の意味は薄れたが、依然アンチエイリアシングを使わないで表示するブラウザは少なくない。

しかしながら手作業での文字画像の作成は、前述した更新速度の速いWebでは作業量の面から破綻する。また、内容と構造、装飾の分離と抽象化の恩恵を逃すこととなる。さらに、そもそも文字でないのだから、通常の文字と混在したレイアウトをはじめ、SEO対策やコピー･ペーストなどのユーザビリティの面でも問題を抱えることになる。

本研究では、以上の問題を解決するため、文字画像化と行内レイアウトを用いて、Webの閲覧環境に関わらず意図した書体を使えるようにすることを目的とする。これにより、効率の良い文字画像の利用方法と柔軟なタイプセッティングを実現する。

本稿では2章にて関連研究と技術について述べ、3章にて本研究が達成すべき事項をまとめた上で提案手法を述べる。4章ではシステムの開発について述べ、5章にて現在行っている実験と検証結果について述べる。6章ではまとめと今後の課題について述べる。

2 関連研究･技術

HTML書類においてフォントの埋め込みを可能とするべくこれまでにいくつかの技術が開発されてきた。坂口ら[4][5]は閲覧環境に依存しない多言語表示を目的として、MHTML[2]を用いた文字画像の埋め込みを行った。MIMEマルチパートを利用した通信のカプセル化には多くのメリットがあるが、対応するブラウザが限定的であるために広くは用いられていないのが現状である。

---

[1] Content Management System

[2] MIME Encapsulation of Aggregate HTML (http://www.faqs.org/rfcs/rfc2557.html)

Microsoft の WEFT [3] は OpenType フォントの部分的な通信をすることで HTML へのフォントの埋め込みを可能にした。類似のものに CSS 2.0 から制定された @font-rule 規則がある。しかしながらこれら 2 つの技術もまた対応するブラウザは少ない。互換性が維持されなければ、それが保証された手作業での文字画像に置き換えることはできない。

Inman による sIRF [4] はフォントを埋め込んだ Flash 書類を配置する手法を提案している。Flash は広く浸透している仮想環境であり、互換性の点でも優れている。また、HTML の内容を反映し、CSS による記述にて装飾を変更するため、既存の Web モデルに対して透過性をもっている。問題は、その表示領域が固定的であり、CSS が規定するインライン成形文脈（Inline Formatting Context）とは親和性が低いことにある。つまり、可変な領域を埋めるように文字をレイアウト､あるいは文字との混在ができない。

しかし WEFT や sIFR の問題はむしろ、タイプセッティングを閉鎖的なプラットフォームに依拠することで、その自由度を大きく失っている点にある。仮にプラットフォームにフォントの合成が用意されていなければ、どう足掻いてもフォントを合成することはできまい。Web は標準化、マルチメディア化、多様化が進み、表現プラットフォームとしての役割も増してきている。ここで求められるタイポグラフィはプラットフォームによってパッケージ化された環境にあるべきではなく、文字通り「ソフト」であるべきであり、フォントに直接触れることができなければならない。

3 提案手法

以上に述べた Web の閲覧上の問題と関連研究･技術から、本研究が開発する Fontaine が達成すべき事項をまとめる。

(1) 閲覧環境に関わらず文字が意図する書体で表示される
(2) 可変な表示領域で通常の文字と同等にレイアウトされる
(3) 既存の Web モデルに対して透過性をもつ
(4) 文字情報としてのユーザビリティを保つ

その上で既存の技術に対して Fontaine がどのようなアドバンテージを持つかについては図 1 を参照されたい。Fontaine を利用する利点は以下にまとめる。

- あまり普及していないものを含む多様なフォントを指定できる
- 水平垂直スケーリングや回転などの高度なタイプセッティングが可能である
- CSS で装飾を指定するため、既存の Web モデルを崩さない
- 可変な表示領域に対して文字画像を用いることができる
- 通常の文字と同じようにコピー･ペースト、文字サイズ変更などを行える
- オープンソースであり、容易に拡張/変更可能である。

これらを踏まえ、本章では本研究の提案手法について説明する。3.1 節では文字画像を利用する全体的なモデルを説明する。次に 3.2 節にてメトリクスの取り扱いを説明し、行内レイアウト手法を示し、3.3 節では既存の Web モデルに対する透過性や文字情報としてのユーザビリティを保つ手法について述べる。

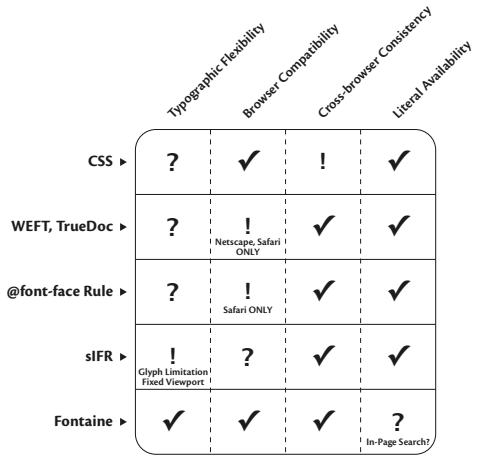

図 1 : 既存技術との比較

[3] The Web Embedding Fonts Tool (http://www.microsoft.com/typography/web/embedding/weft3/default.htm)

[4] Scalable Inman Flash Replacement (http://wiki.novemberborn.net/sifr/)

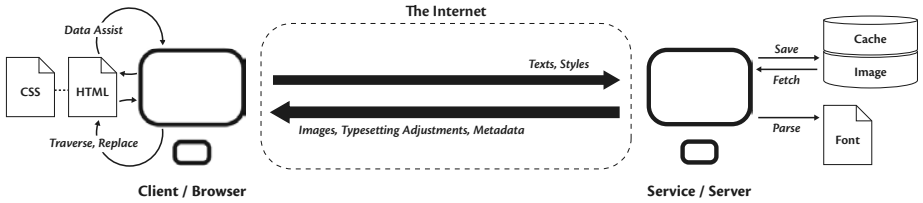

図2: 文字画像モデル

3.1 文字画像モデル

本研究の提案手法にはサーバクライアントモデルを用いる。まず、書類読み込み完了と同時にブラウザ側プログラムがUserAgentを介してHTMLとCSSを走査し、指定された要素内の文字内容と装飾を取得、サーバ側スクリプトへ非同期に送信する。サーバ側スクリプトは要求された情報に基づいて、事前に解析したフォントからメトリクスやアウトライン情報を取得、キャッシュを検索あるいは文字画像を動的に生成し、そのURIを位置調整値（Marginal Adjustments）とともに返信する。ブラウザ側プログラムは対応する文字列ノードを文字画像で置換し、位置調整値を適用する。これら文字画像利用モデルの全体図を図2に示した。

このモデルでは、サーバからHTMLを返すすべての書類に適用し、既存のWebモデルに対して透過的に処理を行う。また、HTMLソースを走査する検索エンジンや、スクリプトが動作しない携帯電話などでは通常の文字として認識されることになる。

3.2 行内レイアウト

Webに適した文字画像の生成とインライン成形文脈との親和性の高いタイプセッティングを目指すため、本提案では文字画像の行内レイアウトを定義する。これは、ラスタライズされた字形の境界の1pxのはみ出しもゆるさない矩形で切り抜かれた文字画像をCSSの行上に挿入、marginとvertical-alignにて位置を調整し、メトリクスを考慮して配置する手法である。これにより通常の文字との混在や、可変な領域を埋めるように文字をレイアウトを行い、CSSが用意しないタイプセッティングを実現する。

字体メトリクスの扱い　字体メトリクスとは、字体のレイアウトを行うために必要な、字体の始点を原点とした座標上の値の集合である。一般に字体メトリクスは字形の高さと幅、advance、lsb、tsbについての値から成り、すべての字体があらかじめ持っている。またそれにrsb、bsbを加えるものもある。ただし定義は曖昧であり、本研究で扱う横書きの字体メトリクスを図3に示す。

書字方向が異なる場合にはadvanceの正負が変わる。以後とくに明記しないがぎり、本手法は書字方向に関わらず適用する。

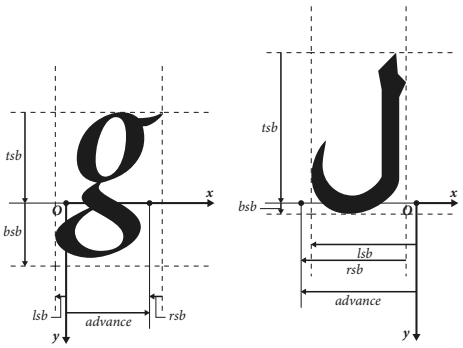

図3: 字体メトリクス
（矢印の方向は正負を表す）

字体の配置と行の境界　字体を行に配置するとき、行の始点を原点とした座標上に字体の始点をもとめる。一般にn個の順序づけられた字体が与えられたとき、i番目の字体の始点$\mathbf{a}_i = (x_i, y_i)$は次のようになる。

$$\mathbf{a}_0 = \mathbf{o},\ v_0 = 0$$
$$k(0,1) = k(n,n+1) = 0$$
$$\mathbf{a}_i = \begin{pmatrix} x_{i-1} + v_{i-1} + k(i-1,i) \\ 0 \end{pmatrix} \quad (i = 1, 2, ..., n+1)$$

このとき v は字体の advance、$k(a, b)$ は a 番目と b 番目の字体との間のカーニング値を返す関数である。

ここで行の論理境界と視覚境界をもとめる。行の論理境界（Logical Line Bounds: llB）は、行が配置される座標上複数の行を接続するための矩形であり、字形と関係なく決定される。行の視覚境界（Visual Line Bounds: vlB）は、行が配置される座標上で、すべての字形が収まるようなできるだけ小さな矩形である。行の ascent を t、descent を u として、

$$llB = \begin{pmatrix} 0 & x_n + v_n \\ -t & -u \end{pmatrix}$$
$$vlB = \begin{pmatrix} x_1 + lsb_1 & x_n + v_n - rsb_n \\ min(y_1 - tsb_1, \ldots, y_n - tsb_n) & max(y_1 + bsb_1, \ldots, y_n + bsb_n) \end{pmatrix}$$

である。特にラスタライズされた字形の境界は常に整数である。ここでピクセル座標上に行の始点 $\mathbf{a}$ を置き、行の視覚境界を丸める。この境界をピクセル境界（Pixel Line Bounds: plB）と言うことにする。後述するようなグリッド吸着がなされるとき、字形のアウトライン（スプライン曲線）は ±1~2px の範囲で変化する。そのため、1px のはみ出しもゆるさない矩形に丸めるには、

$$plB = \begin{pmatrix} floor(\mathbf{a} + vlB_{\cdot,1}) - 1 & ceil(\mathbf{a} + vlB_{\cdot,2}) + 1 \end{pmatrix}$$

とすればよい。また、行のピクセル境界はそのまま文字画像の境界となる。

位置調整値の算出　まず margin 属性によって画像文字のボックスを、通常文字がつくるであろうサイズに揃える。通常文字がつくるボックスの高さは、そのフォント･メトリクスの ascent: t と descent: u の差に一致する。ゆえに margin 属性値は、line height: h が与えられると、

$$top\,margin = \frac{h+t+u}{2} + vlB_{2,1}$$
$$right\,margin = llB_{1,2} + vlB_{1,2}$$
$$bottom\,margin = \frac{h-t-u}{2} + vlB_{2,2}$$
$$left\,margin = llB_{1,1} + vlB_{1,1}$$

となる。次に、vertical-align 属性で画像文字の baseline と通常文字のそれを揃える。

$$vertical\,align = -\frac{h-t-u}{2}$$

これら位置調整値を図 4 に図示した。

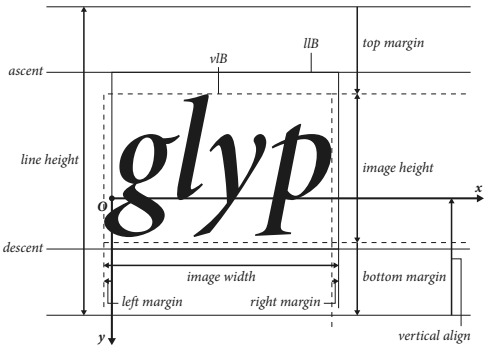

図4: 位置調整値
（矢印の方向は正負を表す）

字体の変形　ピクセル境界で切り抜かれ、位置調整値が適用された文字画像は、その論理的な関係が保たれている、つまりメトリクスの定義に反しない限り自由なタイプセッティングが可能である。例えば、

- トラッキング
- 水平・垂直スケーリング
- 斜体角度
- 回転

など Photoshop や Illustrator に備えているような字形の変形を考える。回転以外の各変形は字体の原点を変形の中心にすればよいのは明らかである。回転変形は次のように行う。回転後の字体の advance を v' として、

$$\mathbf{c}_i = \frac{1}{2}\begin{pmatrix} w_i - v_i cos\theta - (t+u) sin\theta \\ (t+u)(cos\theta - 1) - v_i sin\theta \end{pmatrix}$$

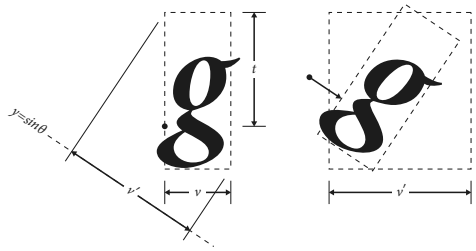

図5: 字体の回転

を中心に字形を回転する（図5）。

$$\begin{bmatrix} \mathbf{a}' \\ 1 \end{bmatrix} = \begin{bmatrix} R_\theta & \mathbf{c}_i \\ 0\ \ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix}$$

直角回転は書字方向の転換と同等であり、$\theta = -\frac{\pi}{2}$ のときに書字方向の逆転換、$\theta = \pi$ のときには ascent と descent が反転する。

3.3 透過性とユーザビリティ

既存の Web モデルに対して透過性を持つために、CSS 属性を独自に拡張する手法をとる。UserAgent は認識できない CSS 属性を無視するので互換性を保てる（中にはブロック内の以降の属性すべてを無視するため、ブロック最後に記述するのが望ましい）。拡張属性名はハイフン《-》から始め、共通の接頭詞が先導する。

文字情報を保ちながら文字画像を HTML に配置する方法はこれまでに多くの手法が提案されてきた。特に Leahy/Langridge Method と Shea Enhancement[5] では、原理的にコピー・ペーストと文章内検索が実現される。ブラウザ上の文字サイズ変更へ対応するには、ビーコンとなる要素を HTML 上に配置しておき、その寸法を一定間隔で検査しながら変化をイベントとして取得する。イベントを受け取ると再度文字画像を置き換える。

4 Fontaine: システムの開発

本章では、前章で提案した手法を用いて開発した Fontaine の概略を述べる。図6にシステム構成図を示した。

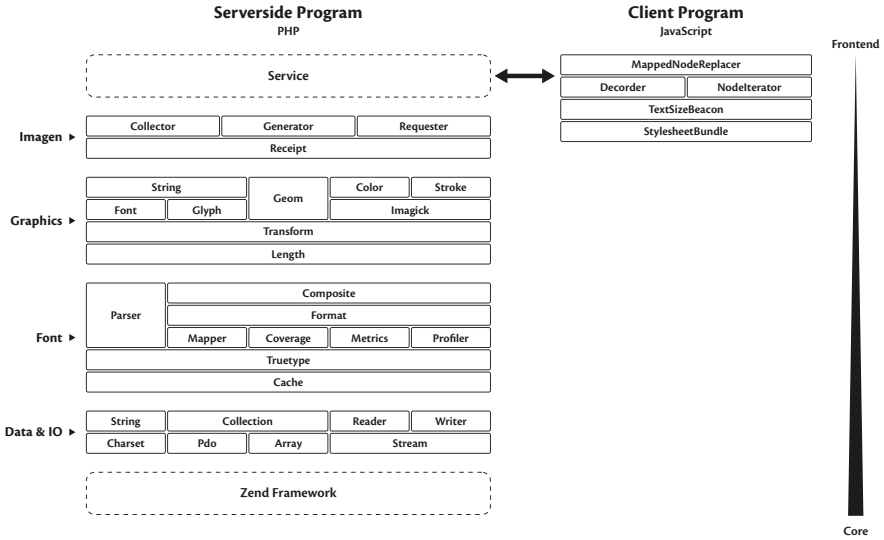

図6: システム構成図
（左端名称はパッケージ、枠はサブパッケージを表し、複数のクラスから構成される）

[5] Shea, M.: Revised Image Replacement. At *http://www.mezzoblue.com/tests/revised-image-replacement/*, 2008.

Data & IO　マルチバイト文字列操作、データベースと配列の抽象化やバイトストリーム入出力を定義する基底パッケージである。特にTrueTypeフォントの解析に必要な種々の整数や固定小数点読み込みを行う。

Font　フォントの解析や検索、実体化などを行う。文字コードと字体コードを変換するMapper、出力可能な文字集合を参照するCoverage、行や字体のメトリクスを格納するMetrics、フォントの固有情報をデータベース化･検索するProfileが定義される。これらの集約であるFormatがフォーマット抽象化レイヤーの役割を果たす。Formatの集約はCompositeであり合成フォントを示す。Parserはフォントを解析し、Cacheを構築する。現在は抽象化レイヤーの下にTrueTypeのみがあるが任意に拡張可能である。

Graphics　長さを表すLength（図7）とアフィン変換行列Transformを基盤とする解像度非依存描画パッケージである。色変換･構築を行うColorと線変換･構築のStrokeを除いてグラフィックエンジンは抽象化されている。フォントはこの時点から物理量と文字情報を持ち、基本的なタイプセッティングが定義されたStringで表される。

Imagen　リクエストの処理と文字画像のキャッシング、抽象化された画像生成を行い、Client Programとの橋渡しをする。Requestorは任意の入力に基づいて画像生成処理コンテナであるReceiptを構築しCollectorへ渡す。CollectorはReceiptのハッシュからキャッシュを検索し、対応する画像が未生成であればGeneratorへ送り、生成画像を格納した上で返信する。

Client Program　文字と画像の置換、CSS属性の拡張、文字サイズ検出、サーバとの連携を行う。StylesheetBundleが拡張CSS属性を検出･解析した上で、対応するHTML要素を定める。NodeIteratorは要素内を走査し、文字列と装飾をServer Programへ非同期通信する。Server Programから返信されたデータからDecorderがMappedNodeReplacerを構築し、文字列ノードと画像を置換した後、位置調整値を適用する。その後TextSizeBeaconが文字サイズ検出処理を実行し続け、変化があればMappedNodeReplacerへ直接通知し、再びServer Programへリクエストを送る。

実装にはPHP 5 / Apache 2を用いた。これはWebとの滑らかな繋がりと完全なオブジェクト指向であることからの選択だが、言語レヴェルの機能が軟弱で、Fontaineが多くの基底クラスを率いている理由である。最下層に位置するZend FrameworkはDBMSの抽象化が目的である。

5 実験と検証

5.1 フォントの実体化速度

フォントの解析にかかる時間は処理方法やフォントやフォーマットの種類によって異なる。測定では255個の字体

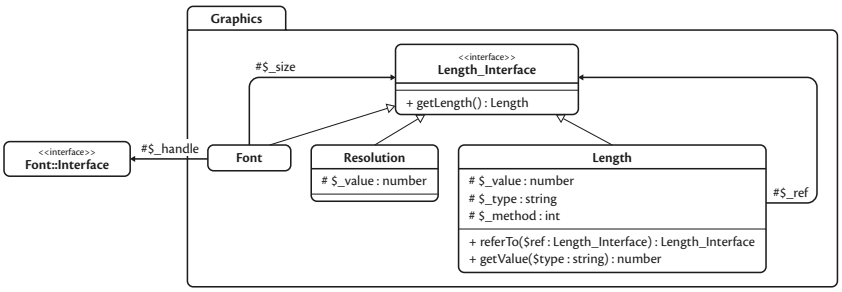

図7: 長さを表すLengthの構造図
（Length_Interfaceを介して全ての処理がLengthに帰着される）

を含む Helvetica で 82ms、約 1 万 5 千個を含む小塚ゴシックで 2,486ms かかった。また、キャッシュからのメモリ復元では 52ms と 297ms にまで短縮される。しかし、要するメモリの大部分は字体に関する情報であり、その復元にかかる時間はフォントが含む字体数に比例する（図 8）。これは日本語など多くの字体を含むフォントで問題となる。

ここですべての情報をメモリ上に展開するのではなく、データベースから必要に応じてフォント情報を取得できるようにする。取得処理のオーバーヘッドは増えるが、画像生成に用いられる字体数とフォント全体のそれを考慮すれば効率的である。この場合フォントの実体化にかかる速度は字体数にかかわらず一定にすることができた。

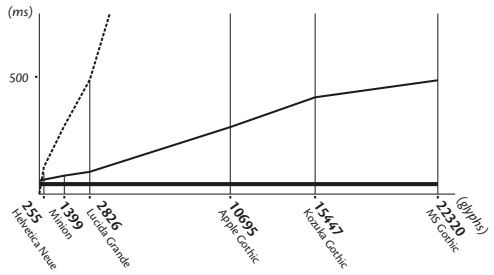

図 8: フォントの実体化速度
（波線が解析時間、細線がメモリからの復元、
太線がデータベース利用時を示す）

5.2 行の分割

HTML に配置される文字画像は、インライン成形文脈レイアウトが行われるものでなければならない。したがって、行を分割するための区切りが必要となる。改行処理を考えればラテンやギリシャ、キリルといった空白文字によって単語を分割する文字と、漢字やひらがなど単語境界が曖昧な文字で異なる。前者では単語の境界で段落が分割し、後者は一文字ごとに分割するのが好ましい。しかし、画像数が増えると通信時のアドレス解決や置き換え処理などのオーバーヘッドが増えるため、いくつかの分割方法を検証することにした。

表 1、表 2 に、異なる分割方法で段落を分割した結果を示した。日本語文章は 523 の全角文字からなるものを用い、英語文章は 3,918 のアルファベット及び記号を含むものを用いた。分割に際しては分割後に最大で文字数 ***n*** を含み、かつ空白文字で分割する方法をとった。例えば ***n=2*** で「Human Interface」を分割すると「Hu-ma-n-In-te-rf-ac-e」となる。重複割合とは、分割後に同じ部分文字列が含まれる割合であり、0 以上であればむろん置換数よりも画像数が少なくなる。

これらの結果から分割処理のパフォーマンスについて少なくとも次のことが言える。

- サーバの処理能力またはクライアントの通信速度を基準とすると、最大分割文字数が小さいとき効率が高い。
- クライアントの処理能力を基準とすると、最大分割文字数は大きい必要がある。

また、フォントサイズが大きい場合には個々の文字の境界で行の処理が行われる可能性が大きくなるため、最大分割文字数は充分に小さくなければならない。その逆は必ずしも必要ではない。以上から、行分割方法を 1 つに定めるよりも場合による最適な設定を行うことが適切と考える。

表 1: 日本語文章での行分割

N	*Time Taken* sec	*Size* kb	*Images n*	*Replace n*	*Duplication* %
1	2.02	87.55	190	523	63.7
2	3.45	157.33	225	262	14.1
4	3.25	153.15	129	131	1.5
8	2.83	135.39	66	66	0.0
16	2.73	119.20	33	33	0.0
32	2.58	105.09	17	17	0.0

表 2: 英語文章での行分割

N	*Time Taken* sec	*Size* kb	*Images n*	*Replace n*	*Duplication* %
1	2.73	17.83	46	3268	98.6
2	5.47	107.40	274	1795	84.7
4	11.91	282.61	509	1071	52.5
8	12.69	295.83	407	738	44.9
16	12.56	293.59	361	654	44.8
Word	13.06	292.98	358	651	45.0

5.3 部分グリッド吸着

TrueType や OpenType には、アウトラインデータとビットマップを仲介するグリッド吸着（Grid-fitting）が備わっている。これは、9pt や 14pt などの小さなサイズの字形を明瞭にレンダリングするための技術である。例えば《m》という字形があたえられたとき、フォントの種類に関わらず 3 本の足が分かれて表示されていなければ《m》なのか《n》なのか認識しづらくなるだろう。適切に作りこまれたグリッド吸着は字形の読みやすさを向上させる。

しかし前述したように、細かなディティールまで再現するようにグリッド吸着点が定められたフォントは少なく、そのフォントの面影さえも失われる。文字は判別できるがフォントは判別できないという状況が生まれる。そこで、精細さを増すためにグリッド吸着点を s^2 倍する部分グリッド吸着（Subgrid-fitting）を考案した。部分グリッド吸着は通常のダウンサンプリングのような負荷はかからないが、s^2 倍された座標の展開に必要なメモリ量は消費される。品質と速度のバランスを検証したところ、およそ s=4、16 倍部分グリッド吸着が望ましい結果を出した（図 9）。

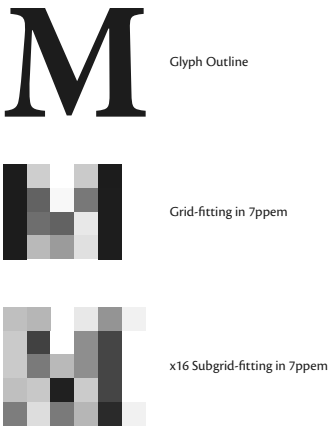

図 9: 部分グリッド吸着
（左右の足の太さやセリフなどが保たれることが分かる）

6 まとめと今後の課題

本研究では、Web 上の書体に一定の一貫性やタイプセッティングの自由度を与える手法を提案し、それを実装する Fontaine を開発した。しかしながら、すべてのブラウザがフォントの通信や高度なタイプセッティングをサポートするのが本来的に健全な姿である。その意味で Fontaine は過渡的なものだ。

Web 上のタイポグラフィにはまだまだ開拓の余地がある。Web は幅が可変で縦方向へ無限に続く特殊な紙面である。その可能性は定形紙へ印刷するそれと根本的に異なってくるだろう。現在はまだ印刷技術で培われたタイポグラフィをそこに移行している段階であり、今後はそれを解体·再構築するべきと考えている。

そこまで大きいことを言わずとも、多種のブラウザに対する互換性向上や、画像生成速度の向上、画像であることの利点を活かすための機能追加など、細かな課題は依然残っている。

謝辞

Fontaine は独立行政法人情報処理推進機構（IPA）より、2008 年度上期未踏 IT 人材発掘･育成事業（未踏ユース）の支援を受けて開発されたものである。PM としてご助言とご指導を頂いた東京大学大学院 竹内郁雄教授を初めとして、株式会社創夢の小林様、高橋様、また多くの未踏関係者の皆さまに厚く感謝したい。

参考文献

[1] Andersson, O., et al.: Scalable Vector Graphics (SVG) 1.1 Specification. At *http://www.w3.org/TR/SVG11/*, 2003.

[2] Frutiger, A: 活字の宇宙、朗文堂、2001

[3] Microsoft Corporation: OpenType specification. At *http://www.microsoft.com/typography/otspec/*, 2004

[4] Sakaguchi, T., et al.: A Browsing Tool for Multi-lingual Documents for Users without Multi-lingual Fonts. In *Proceedings of the first ACM international conference on Digital libraries*, 1996.

[5] Sakaguchi, T., et al.: A Multilingual Browser for WWW without Preloaded Fonts. In *Proceedings of International Symposium on Digital Libraries*, 1995.