SRA GROUP
GSLetterNeo Vol.96
2016年7月

テキストデータ群の重ね合わせによるヒストリと特徴のインタラクティブ表示(2)

松原 伸人、土屋 正人
matubara@sra.co.jp, m-tsuchi@sra.co.jp

◆ はじめに

大量のデータをインタラクティブに操作する、Web ブラウザ上で動作するアプリケーションを開発しています。

「カタマリを見つけて辿れるヒストリデータブラウザ」(GSLetterNeo Vol.91, 92, 93, 94)は、歴史データや、開発ログといった大量のイベントを時間軸上に並べて表示することで、イベントが連続的に起きている時間帯や切れ目を見つけて辿れるようにする Web アプリケーションのプロトタイプです。

Vol.95 では、データに GSLetterNeo を使い、

- **ドキュメントタイトル、著者等を表示するレイヤー**
- **特徴語群を表示するレイヤー**
- **希少語群を表示するレイヤー**

の 3 層を画面奥に向かって重ねて配置することで、ドキュメントの時間関係と特徴がひとまとまりに見えるようにしたものを紹介しました。3 つのレイヤーの仮想 3D 空間上での形とサイズをハイライト表示したものを再掲します(図 1～3)。

今回は実装面を紹介します。

図 1　タイトルなどの情報を表示するレイヤー

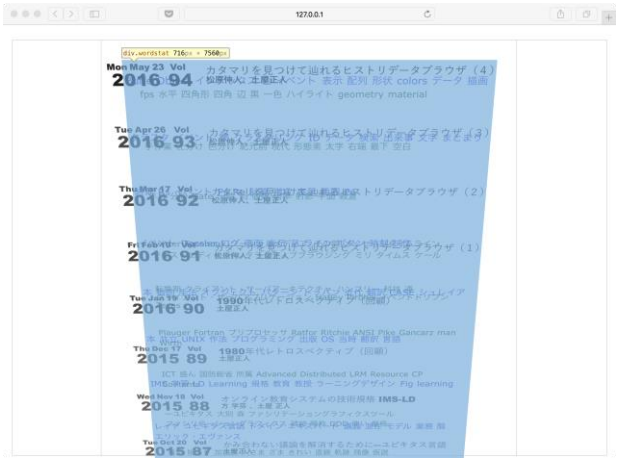


図 2　特徴語群を表示するレイヤー

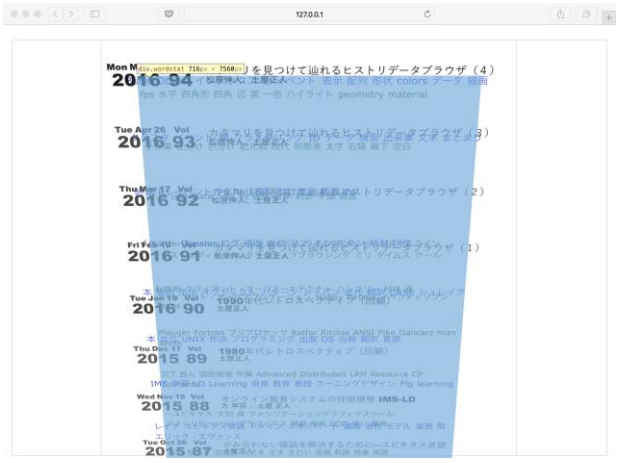


図 3　希少語群を表示するレイヤー

◆ 仮想空間への配置と表示

仮想空間への各レイヤーの配置と視界の移動には、GSLetterNeo Vol.94 で少し触れている JavaScript 用の 3 次元プログラミングライブラリ three.js を利用しています。

three.js の主な API は、WebGL 向けのプログラムを簡単に書くためのものですが、現在は example として THREE.CSS3DRenderer と THREE.CSS3DObject を提供しています。

この 2 つのクラスを使うと、three.js で WebGL 向けに書くのとほとんど同じ記述で、HTML の 3D 表示を記述できるようになっています。

three.js をご存知の方は、WebGL 用のクラス THREE.WebGLRenderer と THREE.Object3D を見たことがあると思いますが、CSS3D では THREE.CSS3DRenderer と THREE.CSS3DObject を代わりに使います。

各ドキュメントの出版年月日、号、タイトル、著者などのドキュメント情報は、GSLetterNeo の Web ページにある RSS フィードをダウンロードして用いています。

GSLetterNeo の RSS フィード
http://www.sra.co.jp/public/sra/gsletter/gsletter.xml

特徴語群と希少語群は、GSLetterNeo 各号からテキストデータを export し、IPA 辞書と Web ブラウザ上で動かせる形態素解析エンジン kuromoji.js で名詞を抽出し、同じ単語の数、単語を含むドキュメント数、単語の tf-idf、単語の希少度合いを計算し、ドキュメントごとに tf-idf の高い 10 語を特徴語群、希少度合いの高い 10 語を希少語群としています。

ドキュメント情報を表示するレイヤーの html エレメントを rssLayer、特徴語群を表示するレイヤーを tfidfLayer、希少語群を表示するレイヤーを minorityLayer とすると、仮想空間に表示する 3D オブジェクトを作成するコードは次のように書けます。

```
rssLayer3dObject = new THREE.CSS3DObject(rssLayer);
tfidfLayer3dObject = new THREE.CSS3DObject(tfidfLayer);
minorityLayer3dObject = new THREE.CSS3DObject(minorityLayer);
```

作成した 3 つのレイヤーを表示するシーンを作成し、シーンに 3 つのレイヤーを登録します。

```
scene = new THREE.Scene();
scene.add(rssLayer3dObject);
scene.add(tfidfLayer3dObject);
scene.add(minorityLayer3dObject);
```

視野角 fov と画面の縦横比率 aspect を指定してシーンを見るカメラを作成します。

```
camera = new THREE.PerspectiveCamera(fov, aspect, 1, 10000);
```

3 つ目のパラメータ 1 と 4 つ目のパラメータ 10000 は、目の位置からの距離で、視界の表示範囲を指定するものですが、CSS3D では今のところ視界の範囲を限定できないので未使用のパラメータです。

CSS 用のレンダラーを作成し、画面の縦横幅 width, height を設定して、camera と scene で仮想空間を画面に射影します。

```
renderer = new THREE.CSS3DRenderer();
renderer.setSize(width, height);
renderer.render(scene, camera);
```

このプロトタイプでは、3 つのレイヤーを仮想空間の原点(x, y, z) = (0, 0, 0)から 200px ずつ奥にずらし、レイヤーの上側が手前にくるように 30 度傾けて配置しています。

```
angle = 30;
pitch = Math.PI / 180 * angle;
rssLayer3dObject.position.set(0, 0, 0);
rssLayer3dObject.rotation.set(pitch, 0, 0);
tfidfLayer3dObject.position.set(0, 0, -200);
tfidfLayer3dObject.rotation.set(pitch, 0, 0);
minorityLayer3dObject.position.set(0, 0, -400);
minorityLayer3dObject.rotation.set(pitch, 0, 0);
```

カメラの位置と注視点の初期状態は、ドキュメント情報を表示する rssLayer の上端が画面の上部にくるようにしました。

near は仮想空間の原点から画面までの距離で、画面の幅と視野角で計算できます。

```
fovRad = fov / 360 * Math.PI,
layerHeight = rssLayer.clientHeight,
py = layerHeight * Math.cos(pitch) / 2 - width / 2,
near = width / 2 / Math.tan(fovRad / 2) / 2,
pz = near + (layerHeight * Math.sin(pitch) / 2);
camera.position.set(0, py, pz);
renderer.render(scene, camera);
```

画面上で上下にドラッグして、カメラをレイヤーの面に沿って動かすコードでは、mousemove イベント検出時に、マウスの移動量 mv を求め、カメラの縦方向の位置 y に

mv.y を加え、near とカメラの縦方向の位置 y から奥行き方向の位置 z を計算した値で、カメラの位置 position を更新します。

lookAt はカメラの注視点を指定する関数です。注視点にはカメラの位置と縦方向に平行になるような位置を指定しています。

```
var p = camera.position,
        px = p.x,
        py = p.y + mv.y,
        pz = near + (py + height / 2) * Math.tan(pitch);
p.set(px, py, pz);
camera.lookAt(new THREE.Vector3(0, p.y, 0));
renderer.render(scene, camera);
```

◆ 形態素解析[1]と特徴抽出

特徴語群と希少語群の算出するために、形態素解析エンジンの 1 つ kuromoji.js を使用して、各ドキュメントの文字列を品詞ごとに分解し名詞だけ抜き出した単語群を作りました。

kuromoji.js は、Web ブラウザ上で実行できる JavaScript で書かれた形態素解析用プログラムです。node.js からも利用できます。

```
JavaScript implementation of Japanese morphological analyzer [github]
https://github.com/takuyaa/kuromoji.js
```

使用方法はとても簡単で、テキストを単語に分解して品詞が名詞の単語だけ抜き出すコードは、次のように書けます。

```
<script src="kuromoji/kuromoji.js"></script>
<script type="text/javascript">
kuromoji.builder({ dicPath: "kuromoji" }).build(function (err, tokenizer) {
    var pos = '名詞',
        text = "テキストデータ群の重ね合わせによるヒストリーと特徴のインタラクティブ表示",
        result = tokenizer.tokenize(text).filter(function (token) {
            return token.pos === pos;
        });
    console.log(result);
});
</script>
```

結果はコンソールに以下のように出ます(品詞情報を全部表示すると長いので省略)。

```
Array (7)
0{word_id: 498860, word_type: "KNOWN", word_position: 1, surface_form: "テキスト", pos: "名詞",  }
1{word_id: 822200, word_type: "KNOWN", word_position: 5, surface_form: "データ", pos: "名詞",  }
2{word_id: 2602290, word_type: "KNOWN", word_position: 8, surface_form: "群", pos: "名詞",  }
3{word_id: 560250, word_type: "KNOWN", word_position: 18, surface_form: "ヒストリー", pos: "名詞",  }
4{word_id: 489480, word_type: "KNOWN", word_position: 24, surface_form: "特徴", pos: "名詞",  }
5{word_id: 761220, word_type: "KNOWN", word_position: 27, surface_form: "インタラクティブ", pos: "名詞",  }
6{word_id: 2535160, word_type: "KNOWN", word_position: 35, surface_form: "表示", pos: "名詞",  }
```

ドキュメント内での単語の使用数の計測では、解析データの surface_form が同じ語をカウントします。使用数が多い順にソートして出力するコードは次のように書けます。

```
kuromoji.builder({ dicPath: "kuromoji" }).build(function (err, tokenizer) {
    var pos = '名詞',
        text = "テキストデータ群の重ね合わせによるヒストリーと特徴のインタラクティブ表示¥n 大量のデータをインタラクティブに操作する、Web ブラウザ上で動作するアプリケーションを開発しています。¥n「カタマリを見つけて辿れるヒストリデータブラウザ」(GSLetterNeo Vol.91, 92, 93, 94)は、歴史データや、開発ログといった大量のイベントを時間軸上に並べて表示することで、イベントが連続的に起きている時間帯や切れ目を見つけて辿れるようにするWeb アプリケーションのプロトタイプです。¥n¥n このプロトタイピングでは、どのような実装をすれば大量のデータをインタラクティブに扱えるか？反対に、実現したいインタラクションを、実装してみた結果としてインタラクティブ性能がどうなるか？ということを試行錯誤して探っています。¥n¥n 大量のテキストデータの、個々のテキストの特徴と時間関係をインタラクティブに見ていくビューアのプロトタイプを紹介します。(図1)",
        result = tokenizer.tokenize(text).filter(function (token) {
            return token.pos === pos;
        });
    console.log(result);

    var countByToken = {};
    result.forEach(function (token) {
        var count = countByToken[token.surface_form];
        if (count == null) {
            count = 0;
        }
        count += 1;
        countByToken[token.surface_form] = count;
    });
    var words = Object.keys(countByToken).sort(function (word1, word2) {
        return countByToken[word2] - countByToken[word1];
    });
    console.log(words.map(function (word) {
        return "{" + word + ": " + countByToken[word] + "}";
    }).join(", "));
});
```

```
{, : 8}, {データ: 5}, {インタラクティブ: 5}, {大量: 4}, {テキスト: 3}, {時間: 3}, {特徴: 2}, {表示: 2}, {Web: 2}, {上: 2}, {アプリケーション: 2}, {開発: 2}, {イベント: 2}, {こと: 2}, {よう: 2}, {プロトタイプ: 2}, {実装: 2}, {1: 1}, {91: 1}, {92: 1}, {93: 1}, {94: 1}, {群: 1}, {ヒストリー: 1}, {操作: 1}, {ブラウザ: 1}, {動作: 1}, {カタマリ: 1}, {ヒストリデータブラウザ: 1}, {GSLetterNeo: 1}, {Vol: 1}, {歴史: 1}, {ログ: 1}, {軸: 1}, {連続: 1}, {的: 1}, {帯: 1}, {切れ目: 1}, {プロトタイピング: 1}, {反対: 1}, {実現: 1}, {インタラクション: 1}, {結果: 1}, {性能: 1}, {試行: 1}, {錯誤: 1}, {個々: 1}, {関係: 1}, {ビューア: 1}, {紹介: 1}, {図: 1}
```

---

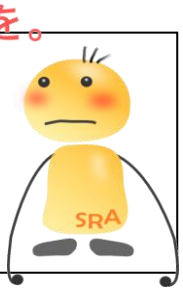
1 文法的な情報の注記の無い自然言語のテキストデータ（文）から、対象言語の文法や、**辞書**と呼ばれる単語の品詞等の情報にもとづき、形態素（*Morpheme*, おおまかにいえば、言語で意味を持つ最小単位）の列に分割し、それぞれの形態素の品詞等を判別する作業（Wikipedia より引用）。

このように、GSLetterNeoの全号について単語ごとの出現数を計測しておくと、各単語がどの号で使われているかを計測できます。

次号では、形態要素解析で使用する辞書について紹介します。

GSLetterNeo Vol.96
2016年7月20日発行
発行者●株式会社SRA 先端技術研究所
編集者●土屋正人

バックナンバを公開しています●http://www.sra.co.jp/gsletter
ご感想・お問い合わせはこちらへお願いします●gsneo@sra.co.jp



株式会社SRA
〒171-8513 東京都豊島区南池袋２－３２－８