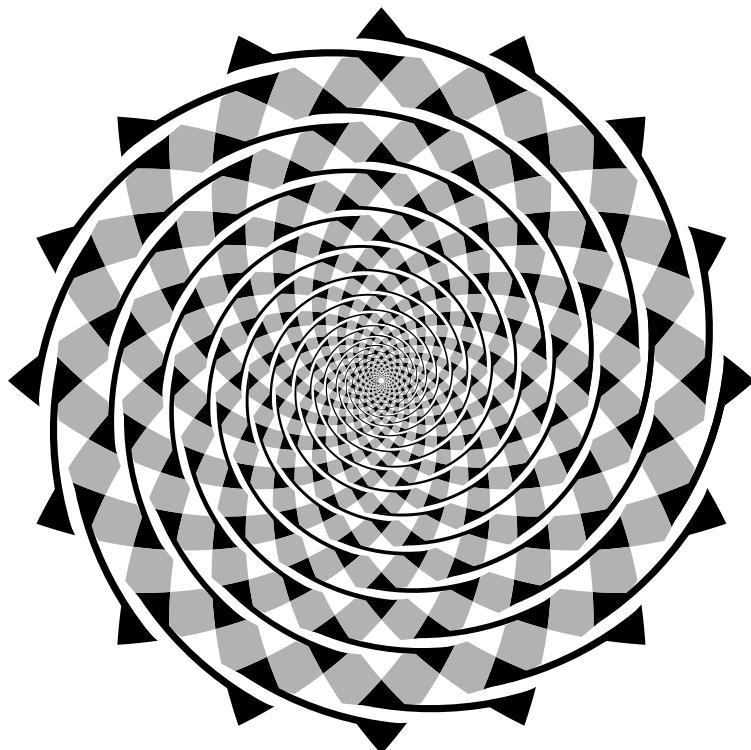


SVG ではじめる Web Graphics 2017年度版
(情報メディア専門ユニットI)
平成29年3月8日改訂



フレイザーの渦巻き錯視

この文書についてお問い合わせは下記のメールまでご連絡ください。

©2006 – 2017 平野照比古

e-mail:hilano@ic.kanagawa-it.ac.jp

URL:<http://www.hilano.org/hilano-lab>

目 次

第1章 SVGについて	1
1.1 SVGとはなにか	1
1.2 XMLとは	2
1.3 SVGの画像を表示する方法	4
1.3.1 SVGが取り扱えるソフトウェア	5
1.4 HTML5について	5
1.4.1 HTML5までの道程	5
1.4.2 HTML5における新機能	5
第2章 SVG入門	7
2.1 SVGファイルを作成する方法と作成上の注意	7
2.2 SVGの基礎	11
2.2.1 座標系について	11
2.2.2 SVG文書におけるコメントの記入方法	11
2.2.3 色について	11
2.3 直線	12
2.4 長方形	18
2.5 円と橙円	22
2.6 グラデーション	25
2.6.1 線形グラデーション	26
2.6.2 放射グラデーション	33
2.7 不透明度	35
第3章 SVGの図形	39
3.1 折れ線と多角形	39
3.2 道のり(Path)	43
3.2.1 直線の組み合わせ	44
3.2.2 橙円の一部となる曲線	46
3.2.3 Bézier曲線	50
3.3 transformについての補足	56
3.3.1 原点以外を中心とする回転	57
3.3.2 座標軸方向へのゆがみ	58
3.3.3 一般的な線形変換	60

3.4	SVG パターン	61
3.5	画像の取り込み	67
3.6	画像の一部を見せる	69
第 4 章	アニメーション	73
4.1	アニメーションにおける属性	73
4.2	位置を動かす (<animateTransform> 要素)	74
4.3	道のりに沿ったアニメーション (<animateMotion> 要素)	78
4.4	いろいろな属性に動きをつける	79
4.4.1	一般の属性に変化をつける (<animate> 要素)	79
4.4.2	属性値をすぐに変える—<set> 要素を利用したアニメーション	83
4.5	複数の値を指定する (keyTimes,values)	83
4.6	イベントを利用したアニメーション	84
4.6.1	イベントとは	84
4.6.2	マウスのイベントを利用したアニメーション	84
4.6.3	アニメーション終了のイベントを利用する	86
4.7	アニメーションがついた錯視図形	88
第 5 章	文字列の表示	91
5.1	文字列表示の基礎	91
5.2	部分的に文字の表示を変える方法	92
5.3	文字をグラデーションで塗る	94
5.4	道程に沿った文字列の配置	96
5.5	円周上に沿って文字列が移動するアニメーション	97
第 6 章	フィルタ	101
6.1	フィルターに関する一般的な注意	101
6.2	画像をぼかす (feGaussianBlur フィルタ)	101
6.3	影をつける (feOffset フィルタと feMerge フィルタ)	105
6.4	画像を合成する (feImage フィルタと feBlend フィルタ)	106
6.5	与えられた画像の色を変える	109
6.5.1	feColorMatrix フィルタ	109
6.5.2	feComponentTransfer フィルタ	116
6.6	画像を单一色で塗りつぶす (feFlood フィルタ)	116
6.7	複雑な画像の合成 (feComposite フィルタ)	118
6.8	光を当てる	120
6.8.1	光源の種類	121
6.8.2	feDiffuseLighting フィルタ	122
6.8.3	feSpecularLighting フィルタ	122
6.9	feTurbulence フィルタ	124

第 7 章 JavaScript を利用した SVG 図形の生成	127
7.1 インタラクティブな SVG を作成するための準備	127
7.1.1 JavaScript	127
7.1.2 DOM(Document Object Model)	128
7.1.3 DOM のメソッドとプロパティ	130
付録 A SVG の色名	付録 1
付録 B 参考文献について	付録 5
付録 C JavaScript 入門	付録 9
C.1 JavaScript とは	付録 9
C.2 データの型	付録 9
C.2.1 プリミティブデータ型	付録 9
C.2.2 配列	付録 11
C.3 演算子	付録 11
C.3.1 代入、四則演算	付録 11
C.3.2 比較演算子	付録 12
C.4 制御構造	付録 13
C.4.1 if 文	付録 13
C.4.2 switch 文	付録 13
C.4.3 for 文と while 文	付録 14
C.5 関数	付録 15
C.5.1 関数の定義と呼び出し	付録 15
C.5.2 仮引数への代入	付録 16
C.5.3 arguments について	付録 17
C.6 変数のスコープと簡単な例	付録 18
C.6.1 JavaScript における関数の特徴	付録 21
C.6.2 クロージャ	付録 23
C.7 配列のメソッド	付録 26
C.8 オブジェクト	付録 29
C.8.1 配列とオブジェクト	付録 29
C.8.2 コンストラクタ関数	付録 31
C.8.3 オブジェクトリテラルと JSON	付録 35
C.8.4 ECMAScript5 のオブジェクト属性	付録 36
C.8.5 エラー オブジェクトについて	付録 41
付録 D CSS について	付録 43

索引	付録 47
SVG の用語	付録 47
HTML の用語	付録 54
JavaScript の用語	付録 55
一般	付録 57

第1章 SVGについて

1.1 SVGとはなにか

Web 上での各種規格は World Wide Web Consortium(W3C) と呼ばれる組織が中心になって制定しています。その Web Design and Applications¹ (図 1.1 参照) には次のように書かれています。

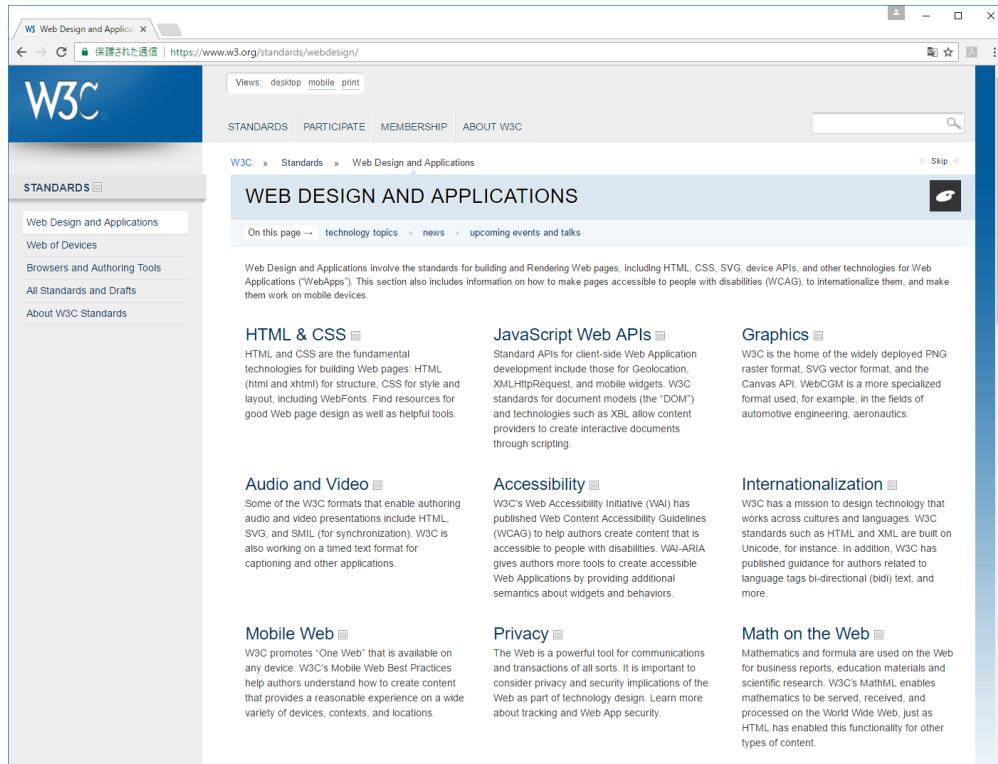


図 1.1: World Wide Web Consortium の Web Design and Application のページ (2017/2/27 参照)

Web Design and Applications involve the standards for building and Rendering Web pages, including HTML, CSS, SVG, device APIs, and other technologies for Web Applications (“ WebApps ”). This section also includes information on how to make pages accessible to people with disabilities (WCAG), to internationalize them, and make them work on mobile devices.

¹<http://www.w3.org/standards/webdesign/>

このページの Graphics をクリックすると W3C のグラフィックス関係の規格の解説のページへ移動します(図 1.2)。

このページの下に SVG の簡単な説明があります。

Scalable Vector Graphics (SVG) is like HTML for graphics. It is a markup language for describing all aspects of an image or Web application, from the geometry of shapes, to the styling of text and shapes, to animation, to multimedia presentations including video and audio. It is fully interactive, and includes a scriptable DOM as well as declarative animation (via the SMIL specification). It supports a wide range of visual features such as gradients, opacity, filters, clipping, and masking.

The use of SVG allows fully scalable, smooth, reusable graphics, from simple graphics to enhance HTML pages, to fully interactive chart and data visualization, to games, to standalone high-quality static images. SVG is natively supported by most modern browsers (with plugins to allow its use on all browsers), and is widely available on mobile devices and set-top boxes. All major vector graphics drawing tools import and export SVG, and they can also be generated through client-side or server-side scripting languages.

SVG の正式な規格書は図 1.2 のページの右側にある CURRENT STATUS の部分の SVG をクリックするとみることができます(図 1.3)。

画像を作成し、保存する形式を大きく分けるとビットマップ方式とベクター方式があります。

ビットマップ方式は画像を画素(ピクセル)という単位に分け、その色の情報で画像を表します。したがってはじめに決めた大きさで画像の解像度が決まります。Adobe 社の Photoshop や Windows に標準でついてくるペイントはビットマップ方式の画像を作成するツールです。

これに対し、ベクター方式では線の開始位置と終了位置(または長さと方向)、線の幅、色の情報などをそのまま持ります。したがって画像をいくら拡大しても画像が汚くなることはありません。ベクター方式のソフトウェアはドロー系のソフトウェアとも呼ばれます。代表的なものとしては Adobe 社の Illustrator があります。

SVG はその名称からもわかるようにベクター形式の画像を定義するフォーマットのひとつです。

予習問題 1.1 次の事柄について調べなさい。

1. 画像の保存形式を調べ、それがビットマップ方式かベクター方式か調べなさい。
2. ビットマップ方式とベクター方式の画像形式の利点と難点をそれぞれ述べなさい。

1.2 XML とは

XML は eXtensible Markup Language の略です。“Markup Language” の部分を説明します。ワープロなどで文書を作成するとき、ある部分は見出しなので字体をゴシックに変えるということをします。このように文書には内容の記述の部分とそれを表示するための情報を含んだ部分があることになります。このように本来の記述以外の内容を含んだ文書をハイパーテキストと呼びます。Microsoft Word のようなワープロソフトが作成する文書もハイパーテキストです。このよ

The screenshot shows the W3C Graphics page. The left sidebar has a 'Graphics' link under 'WEB DESIGN AND APPLICATIONS'. The main content area has a 'GRAPHICS' section with sub-sections: 'What are Graphics?', 'What are Graphics Used For?', 'What is PNG?', and 'What is SVG?'. A right sidebar lists 'CURRENT STATUS' (SVG, PNG, WebCGM, Graphics), 'USE IT' (Tutorials, Business Case, Software), and 'LOGOS' (SVG).

GRAPHICS

On this page → what are Graphics • what are Graphics used for • examples • learn more • current status of specifications and groups

The Web is about more than text and information, it is also a medium for expressing artistic creativity, data visualization, and optimizing the presentation of information for different audiences with different needs and expectations. The use of graphics on Web sites enhances the experience for users, and W3C has several different and complementary technologies that work together with HTML and scripting to provide the creators of Web pages and Web Applications with the tools they need to deliver the best possible representation of their content. Learn more below about:

This intro text is boilerplate for the beta release of w3.org. Our intent is to invite the community to develop this template and help provide useful content and links. For a more complete example, see the page for [HTML & CSS](#).

What are Graphics?

Web graphics are visual representations used on a Web site to enhance or enable the representation of an idea or feeling, in order to reach the Web site user. Graphics may entertain, educate, or emotionally impact the user, and are crucial to strength of branding, clarity of illustration, and ease of use for interfaces.

Examples of graphics include maps, photographs, designs and patterns, family trees, diagrams, architectural or engineering blueprints, bar charts and pie charts, typography, schematics, line art, flowcharts, and many other image forms.

Graphic designers have many tools and technologies at their disposal for everything from print to Web development, and W3C provides many of the underlying formats that can be used for the creation of content on the open Web platform.

What are Graphics Used For?

Graphics are used for everything from enhancing the appearance of Web pages to serving as the presentation and user interaction layer for full-fledged Web Applications.

Different use cases for graphics demand different solutions, thus there are several different technologies available. Photographs are best represented with PNG, while interactive line art, data visualization, and even user interfaces need the power of SVG and the Canvas API. CSS exists to enhance other formats like HTML or SVG. WebCGM meets the needs for technical illustration and documentation in many industries.

What is PNG?

Portable Network Graphics (PNG) is a static file format for the lossless, portable, well-compressed storage and exchange of raster images (bitmaps). It features rich color control, with indexed-color, grayscale, and truecolor support and alpha-channel transparency. PNG is designed for the Web, with streaming and progressive rendering capabilities. It is supported ubiquitously in Web browsers, graphical authoring tools, image toolkits, and other parts of the creative tool chain. PNG files have the file extension ".PNG" or ".png" and should be deployed using the Media Type "image/png". PNG images may be used with HTML, CSS, SVG, the Canvas API, and WebCGM.

What is SVG?

Scalable Vector Graphics (SVG) is like HTML for graphics. It is a markup language for describing all aspects of an image or Web application, from the geometry of shapes, to the styling of text and shapes, to animation, to multimedia presentations including video and audio. It is fully interactive, and includes a scriptable DOM as well as declarative animation (via the SMIL Specification). It supports a wide range of visual features such as gradients, opacity, filters, clipping, and masking.

The use of SVG allows fully scalable, smooth, reusable graphics, from simple graphics to enhance HTML pages, to fully interactive chart and data visualization, to games, to standalone high-quality static images. SVG is natively supported by most modern browsers (with plugins to allow its use on all browsers), and is widely available on mobile devices and

図 1.2: W3C の SVG に関するトップページ (2017/2/27 参照)

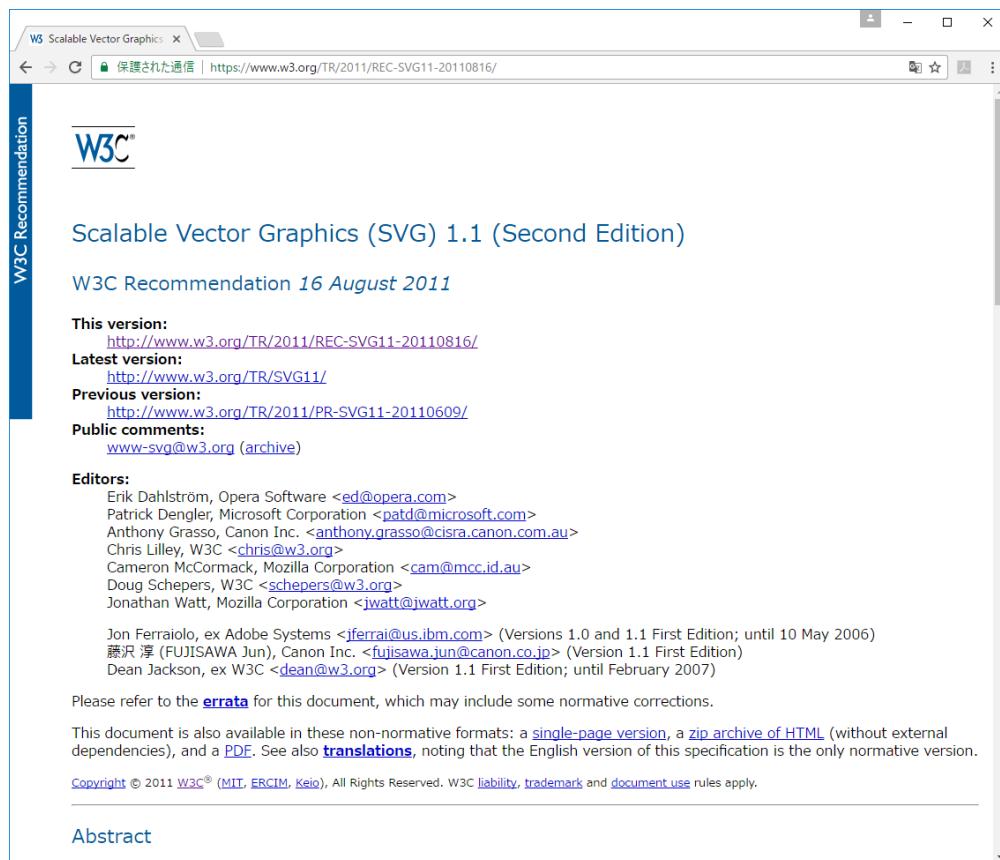


図 1.3: SVG の規格のページ (2017/2/27 参照)

うなハイパーテキストを表現するために文書の中にタグと呼ばれる記号を挿入したものが Markup Language です。

Web のページは普通 HTML(HyperText Markup Language) と呼ばれる形式で記述されています。HTML 文書は先頭が <HTML> 要素で最後が</HTML> で終わっています。ここで現れた <HTML> 要素がタグと呼ばれるものです。HTML 文書ではこのタグが仕様で決まっていてユーザが勝手にタグを定義することができません。一方、XML では extensible という用語が示すようにタグは自由に定義できます。したがって、XML を用いることでいろいろな情報を構造化して記述することが可能になっています。XML についてもっと知りたい場合には文献 [15] などを参照してください。

1.3 SVG の画像を表示する方法

SVG に基づいた画像を作成する方法については次章で解説をします。SVG は XML の構造を持ったベクター形式の画像を定義するフォーマットです。したがって、SVG に基づいたファイルだけでは単なる文字の羅列にしか見えず、これだけでは画像を表示することはできません。画像を表

示するためにはソフトウェアが必要になります。このテキストの題名である Web Graphics の観点からブラウザにより表示することを主に考えます。最近のブラウザは SVG の画像の表示をサポートしていますが、SVG の機能の一つであるアニメーション(第4章で説明します)のうち色のアニメーションについてはサポートしていません。色のアニメーションは CSS で実現できるので 2017 年度からはこの方法で説明をします。

1.3.1 SVG が取り扱えるソフトウェア

ブラウザ以外のいくつかのソフトウェアでも SVG ファイルを取り扱うことができます。

Adobe 社のドローソフト Illustrator は簡単な SVG のファイルを表示したり、結果を SVG ファイルとして保存することができます。

1.4 HTML5について

2015年3月現在、代表的なブラウザは最新の HTML5 機能をインプリメントしています。

1.4.1 HTML5までの道程

W3C は文法上あいまいであった HTML4 の規格を厳密な XML の形式に合う形にするために XHTML[28] を制定しました。これとは別にベンダー企業は 2004 年に WHATWG (Web Hypertext Application Technology Working Group) を立ち上げ、XHTML とは別の規格を検討し始めました。2007 年になって W3C は WHATWG と和解をし、WHATWG の提案を取り入れる形で HTML5 の仕様の検討が始まり、2014 年 10 月に規格が正式なもの (Recommendation) になりました。

1.4.2 HTML5における新機能

HTML5 では HTML 文書で定義される要素だけではなく、それに付随する概念も含めます。HTML5 における新機能としては次のようなものがあります。

- 文書の構成をはっきりさせる要素が導入されています。文書のヘッダー部やフッター部を直接記述できます。
- フォントの体裁を記述する要素が非推奨となっています。これらの事項は CSS で指定することが推奨されています。
- SVG 画像をインラインで含むことができます。
- WebStorage

ローカルのコンピュータにその Web ページに関する情報を保存して、あとで利用できる手段を与えます。機能としては Cookie と同じ機能になりますが、Cookie がローカルに保存され

たデータを一回サーバーに送り、サーバーがそのデータを見てWebページを作成するのに対し、WebStorageではそのデータがローカルの範囲で処理されている点が一番の違います。

Webページの中には2度目に訪れた時に以前の情報を表示してくれるところがあります。これはCookieに情報を保存しておき、再訪したとき、ブラウザが保存されたデータを送り、そのデータを用いてサーバーがページを構成するという手法で実現しています。WebStorageを用いると保存されたデータをサーバーに送ることなく同様のことが実現可能となるのでクライアントとサーバーの間での情報の交換する量が減少します。また、WebStorageでは保存できるデータの量がCookieと比べて大幅に増加しています。

- <canvas>要素

インラインでの画像表示の方法として導入されました。画像のフォーマットはピットマップ方式で、JavaScriptを用いて描きます。描かれた図形はオブジェクト化されないので、マウスのクリック位置にある図形はどれかなどの判断は自分でプログラムする必要があります。

また、アニメーションをするためには途中の図形の形や位置などを自分で計算する必要があり、複数の図形のアニメーションの同期も自分で管理する必要があります。

このテキストでははじめは単独でSVGによる画像を描きますが、途中からはHTML文書内でSVGの画像を表示し、それに対して構成要素を変えるプログラミングについて解説します。

第2章 SVG 入門

2.1 SVG ファイルを作成する方法と作成上の注意

SVG はテキストベースの画像表現フォーマットです。これを作成するためにはテキストエディタと呼ばれるソフトウェアを使います。Windows で標準についてくるメモ帳はテキストエディタの例です。

問題 2.1 エディタソフトウェアにはどのようなものがあるか調べなさい。特に *Unix* ではどのようなもののが有名であるか調べること。また、*XML* 文書を編集するためのテキストエディタにどのようなものがあるか調べなさい。

テキストエディタで SVG ファイルを作成するときの注意を次にあげておきます。

- SVG ファイルの標準の文字コードは UTF-8 です。Windows のメモ帳では文字コードを UTF-8 で保存するとファイルの先頭に BOM と呼ばれるコードが付き、XML 形式のファイルとしては正しくないものになってしまいます。エディタによっては保存する文字コードにこの BOM なしで保存を選択できるものがあります。
- SVG ファイルの内容は HTML ファイルのようにタグをつけた形で記述します。タグで定義されるものを要素と呼びます。
- HTML ファイルのタグでは次のようなことが可能です。
 - 要素名は大文字小文字の区別がなく、要素の開始を表す部分と終了を表す部分で大文字小文字が違っていても問題は起きません。
 - 改行を示す
 要素のように終了部分がない要素もあります。
- SVG ファイルは厳密な XML の構文に従うので要素の開始部分に対応する対応する終了部分を必ず記述する必要があります（簡略な形式もあります）。また、大文字小文字も完全に一致している必要があります。うまく動かないときにはこの点を確かめましょう。¹
- 要素などに記述する単語は英語です。複数形を示す s がついているのを忘れたりするだけで動かなくなるのでよくチェックしましょう。
- ファイルの拡張子は svg が標準です。

¹厳密な XML の構文に従う HTML の文書としては XHTML の形式があります。この形式では
 要素の代わりに
 と記述することで終了タグが不要になります。

このテキストではいろいろなSVGファイルのリストが出てきます。リストの入力に対しては次のことに注意してください。

- リストの各行の先頭には行番号が書いてありますが、これは説明のためにつけたものなので、エディタで入力するときは必要ありません。
- 要素の間の空白は原則的にはひとつ以上あれば十分です。見やすく読みやすくなるようにきれいに記述しましょう。

Google ChromeにおけるXML文書としてのエラーがあった場合の表示例を解説します。

SVGリスト2.1: XML文書としてエラーがあった場合

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3   xmlns:xlink="http://www.w3.org/1999/xlink"
4   height="100%" width="100%">
5   <title>XML文書としてエラーがある場合</title>
6   <rect x="50"y="50" width="100" height="100" fill="gray" />
7 </svg>
```

リスト2.1では6行目でx="0" y="0"と記すべきところをx="0"y="0"と空白を入れなかった場合のエラーの表示です。

Google Chromeではエラーがあった位置を行数と桁で指摘してくれます。

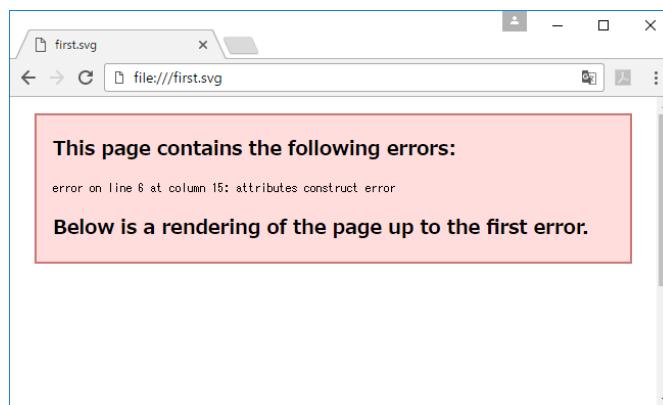


図2.1: Google Chromeにおけるエラーメッセージ

これでよくわからない場合には <http://w3c.github.io/developers/tools/> を利用する方法があります(図 2.2)。

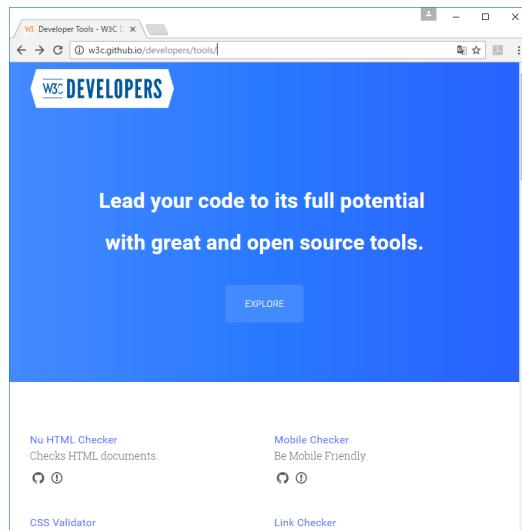


図 2.2: Validator のトップ画面

1. 図 2.2 で Nu Html Checker をクリックすると図 2.3 が表示されます。

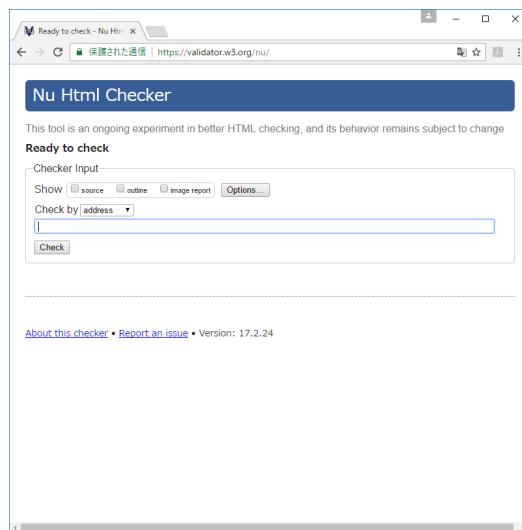


図 2.3: Nu Html Checker の画面

2. ここで「Check by」のプルダウンメニューで「file upload」を設定し、「ファイル選択」でチェックしたいファイルを指定します図2.4。

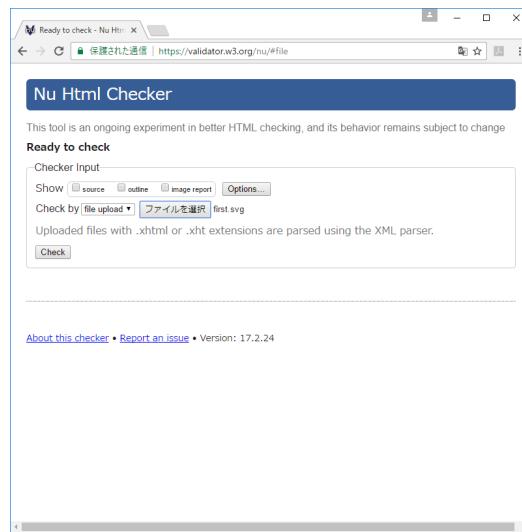


図2.4: ファイルのアップロード

3. ここで「check」のボタンを押すとデータが送られてデータの検証が行われます(図2.5)。

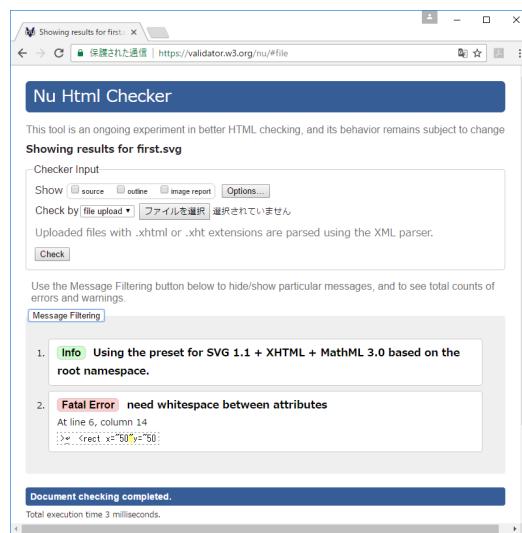


図2.5: 検証結果の表示画面

このページを下のほうに間違いの理由と場所がより具体的に指摘されています。

2.2 SVG の基礎

2.2.1 座標系について

ものの位置を示すためには座標系とその単位が必要です。SVG の場合は初期の段階では左上隅が原点 $(0, 0)$ になり、単位はピクセル(px)です。水平方向は右のほうへ行くにつれて、垂直方向は下に行くほどそれぞれ値が増大します(図 2.6 参照)。

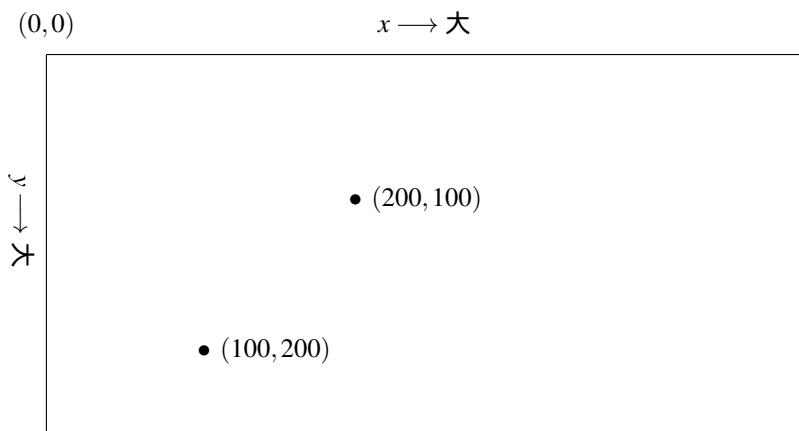


図 2.6: SVG の座標系

後で述べるように SVG ではいくつかの図形をまとめて平行移動したり、回転させることができます。また、水平方向や垂直方向に拡大することも可能です(座標系がすでに回転していればその方向に拡大が行われます)。

2.2.2 SVG 文書におけるコメントの記入方法

SVG 文書でコメントを入れる方法は HTML 文書と同様にコメントの部分を`<!--`と`-->`ではさみます。この本の中ではコメントをほとんど利用していません。

2.2.3 色について

SVG で色を指定する方法は次の 3 種類です。

1. 名称による指定

英語名で色を指定します。どのような名称がどのような色になるかは付録 A を見てください。

2. `rgb` 関数による色の指定

色の 3 原色、赤、緑、青のそれぞれに対し 0 ~ 255 の値を指定します。なお、`rgb` 関数は 0 ~ 255 の代わりに % で色の割合を指定することも可能です。たとえば `red` は `rgb(100%,0%,0%)` と表されます。

3. 16進数による色の指定

0～255までの数は16進数2桁で表せるのでrgbで定めた色を16進数6桁で表示ができます。また、各色の構成を16進数1桁で表し、16進数3桁で指定することもできます。

なお、特別な色として、塗らないことを指示するnoneがあります。

2.3 直線

図形のうちで一番簡単な直線を描いてみましょう。次の例を見てください。

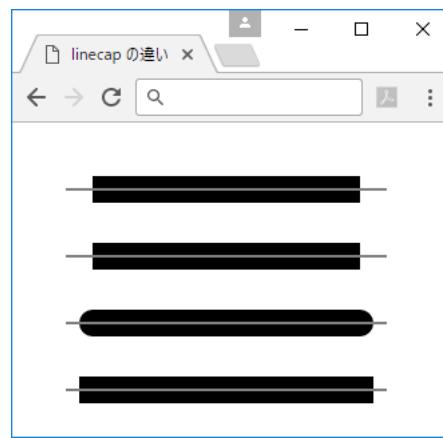


図 2.7: 直線と端の指定 stroke-linecap の違い

これを描いたSVGファイルの内容は次のとおりです。

SVGリスト 2.2: 直線の例

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3   xmlns:xlink="http://www.w3.org/1999/xlink"
4   height="100%" width="100%">
5 <title>linecap の違い</title>
6 <g transform="translate(60,50)">
7   <line x1="0" y1="0" x2="200" y2="0" stroke-width="20" stroke="black"/>
8   <line x1="0" y1="50" x2="200" y2="50" stroke-width="20" stroke="black"
9     stroke-linecap="butt"/>
10  <line x1="0" y1="100" x2="200" y2="100" stroke-width="20" stroke="black"
11    stroke-linecap="round"/>
12  <line x1="0" y1="150" x2="200" y2="150" stroke-width="20" stroke="black"
13    stroke-linecap="square"/>
14  <line x1="-20" y1="0" x2="220" y2="0" stroke-width="2" stroke="gray" />
15  <line x1="-20" y1="50" x2="220" y2="50" stroke-width="2" stroke="gray" />
16  <line x1="-20" y1="100" x2="220" y2="100" stroke-width="2" stroke="gray" />
17  <line x1="-20" y1="150" x2="220" y2="150" stroke-width="2" stroke="gray" />
```

```
18  </g>
19  </svg>
```

- 1行目はこのファイルが XML の規格に基づいて書かれていることを宣言しています。この行の先頭に空白があってはいけません。また<?xml の部分も空白があってはいけません。
- それに続く version や encoding の部分は属性、=の右側はその属性値とそれぞれよばれます。属性値は必ず"で囲む必要があります。
 - version は XML の規格のバージョンを表します。
 - encoding はこのファイルが使用している文字の表し方（エンコーディングとよばれます。）を表します。SVG で日本語を含む文字列を扱うためには UTF-8 と呼ばれるエンコーディングにする必要があります。XML 文書は通常、UTF-8 でエンコーディングするのが標準となっています。
- 2行目から4行目の部分は<svg> 要素を定義しています。このように一番最初に現れる要素をルート要素とよびます。SVG のファイルではルート要素は必ず<svg> 要素となります。ルート要素は XML 文書では1回しか現れてはいけません。
 - <svg> 要素の属性値として xmlns が指定されています。これは XML 名前空間とよばれていて、要素やそれに対する属性が定義されている場所を示しています。ここでは <svg> 要素が定義されている場所を指定しています²。
 - 次の属性 xmlns:xlink とは xlink が定義する名前空間の参照場所を指定しています。このファイルではこの名前空間を使用していないので省略してもかまいません。
 - width と height はこの SVG の画像を表示する大きさを指定しています。ここでは両者とも 100% なのでブラウザの画面全体という意味になります。
- 6行目はこれから描く図形をひとまとめにするために<g> 要素を用いています。transform はこのグループ化された図形を移動することを指定しています。この属性値は表 2.1 のようなものがあります。

表 2.1: transform の属性値

属性値の関数名	機能	例
translate	平行移動	translate(20,40)
rotate	回転（単位は度、向きは時計回り）	rotate(30)
scale	拡大・縮小	scale(0.5), scale(1,-1)

ここでは translate(60,50) となっているので原点の位置が左から 60、上から 50 の位置に移動します。

²XML 文書にはこのほかに<!DOCTYPE で始まる DTD(Document Type Definition)への参照が通常あります。なくても動作するのでこのテキストでは省略しました。

- 7行目から13行目で `stroke-linecap` で指定される直線の両端の形が異なるものを4つ定義しています。
 - <line>要素は直線を定義します。この要素は<g>要素の内側に<line>要素は<g>要素の子要素になっているといいます。また、<g>要素は<line>要素の親要素であるとも言います。
 - 属性としては表2.2を参考にしてください。

表2.2: <line>要素の属性

属性名	意味	属性値
x1	開始位置のx座標	数値
y1	開始位置のy座標	数値
x2	終了位置のx座標	数値
y2	終了位置のy座標	数値
stroke	直線を塗る色	色名またはrgb値で与える
stroke-width	直線の幅	数値
stroke-linecap	直線の両端の形を指定	butt (default) 何もつけ加えない round 半円形にする square 長方形にする

- はじめの直線は開始位置が(0,0)で終了点が(200,0)となっています。
 - 線の幅(stroke-width)が20でその色(stroke)を黒(black)に指定しています。
 - はじめの二つが同じ形をしているので linecap のデフォルト値が butt であることがわかります。
 - 最後の文字列/>はこの要素が子要素を含まないことを示すための簡略的な記述方法です。
- 14行目から17行目でははじめの直線4本より幅が狭いものを灰色(gray)で描いています。これは描かれる線分の位置がどこに来るかを確かめるためです。
この図から直線の幅は与えられた点の位置から両側に同じ幅で描かれることがわかります。
- この灰色の線がすべて見えていることから後から書かれた図形のほうが優先して表示されることがわかります。
- 18行目では<g>要素の内容が終了したことを示す</g>があります。
- 19行目では<svg>要素の内容が終了したことを示す</svg>があります。

フィックの錯視 直線を組み合わせてできる一番簡単な錯視図形がフィックの錯視 [32, 61ページ] です(図2.8)。

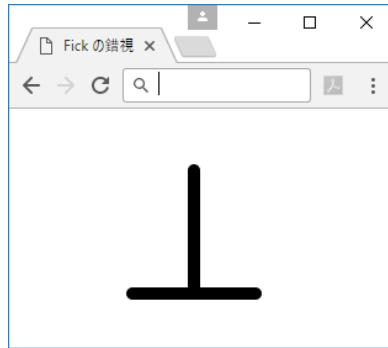


図 2.8: フィックの錯視

水平の線分と垂直な線分の長さが同じなのにその位置が中央にあると長く見えるというとして知られているものです。簡単な図形ながら錯視量が大きいことで有名な図形です。これを描く SVG 文書のリストがリスト 2.3 です。

SVG リスト 2.3: フィックの錯視

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>Fick の錯視</title>
6  <g transform="translate(100,150)">
7      <line x1="0" y1="0" x2="100" y2="0"
8          stroke-width="10" stroke="black" stroke-linecap="round" />
9      <line x1="50" y1="0" x2="50" y2="-100"
10         stroke-width="10" stroke="black" stroke-linecap="round" />
11  </g>
12  </svg>
```

- 7 行目から 8 行目で水平の直線を描いています。
- 9 行目から 10 行目で垂直の直線を描いています。

問題 2.2 フィックの錯視に関連して次のことを行いなさい。

1. 90° 回転した図形でも同じように見えることを確認しなさい。
2. 垂直な直線の位置を水平方向に移動すると見え方はどのように変わるか調べなさい。
3. 中央の線分の長さをどれだけ縮小したら同じ長さに見えるか調べなさい。

ミューラー・ライヤーの錯視 図 2.9 は中央の線分が同じ長さなのに両端の矢印の向きが異なることで大きさが違って見えるミューラー・ライヤーの錯視 [32, 57 ページ] として知られる図形です。

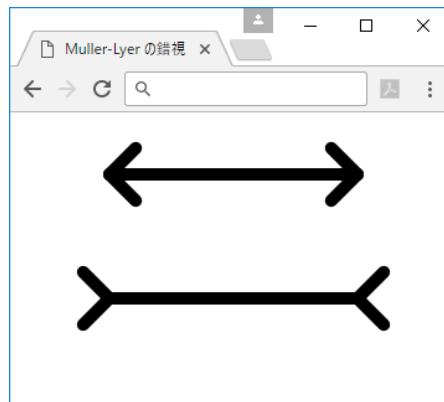


図 2.9: ミューラー・ライヤー錯視

この図形のリストでは両端の矢印の長さや角度を最小限の手間で変えられるようにしています。

SVG リスト 2.4: ミューラー・ライヤーの錯視

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3   xmlns:xlink="http://www.w3.org/1999/xlink"
4   height="100%" width="100%">
5 <title>Muller-Lyer の錯視</title>
6 <defs>
7   <line class="Line" id="Line" x1="0" y1="0" x2="30"
8     stroke-width="10" stroke-linecap="round" stroke="black" />
9   <g id="edge">
10    <g transform="rotate(45)" >
11      <use xlink:href="#Line" />
12    </g>
13    <g transform="rotate(-45)" >
14      <use xlink:href="#Line" />
15    </g>
16  </g>
17 </defs>
18 <g transform="translate(80,50)" >
19   <line x1="0" y1="0" x2="200" y2="0"
20     stroke-width="10" stroke-linecap="round" stroke="black" />
21   <use xlink:href="#edge"/>
22   <g transform="translate(200,0)" >
23     <g transform="scale(-1,1)" >
24       <use xlink:href="#edge"/>
25     </g>
26   </g>
27 </g>
28 <g transform="translate(80,150)" >
29   <g>
30     <line x1="0" y1="0" x2="200" y2="0"
31       stroke-width="10" stroke-linecap="round" stroke="black"/>
32     <g transform="scale(-1,1)">
```

```

33      <use xlink:href="#edge"/>
34    </g>
35    <g transform="translate(200,0)">
36      <use xlink:href="#edge"/>
37    </g>
38  </g>
39 </g>
40 </svg>

```

- 両端にある矢印をすべて同じ長さにするために離形を定義することにします。離形は<defs>要素の中に記述します(6行目)。
- 矢印を構成する直線を7行目から8行目で定義します。この要素は後で参照するためにidで名前を付けておきます。ここではLineという名称になっています。
- 次にこの直線を使って片側の矢印を構成します。二つの直線からなるので後でまとめて一つのものと見て参照するために二つの矢印をグループ化します。それに利用するのが9行目にある<g>要素です。後で参照するためにedgeという名称を与えています。
- この中に二つの直線を描きますが、それぞれを $\pm 45^\circ$ 傾けるためにさらに<g>要素を用います(10行目と13行目)。10行目の<g>要素にはtransformでrotate(45)という値を指定しているので原点(0,0)を中心として 45° 時計回りに傾くことになります。
- この<g>要素のなかに先ほど定義した直線を参照する記述をします。これが<use>要素です(11行目と14行目)。参照先を指定するためにxlink:hrefを用います。参照先はidで定義された名称の前に#を付けます。
- 18行目から27行目の間が上の矢印を記述している部分です。

```

18 <g transform="translate(80,50)" >
19   <line x1="0" y1="0" x2="200" y2="0"
20     stroke-width="10" stroke-linecap="round" stroke="black" />
21   <use xlink:href="#edge"/>
22   <g transform="translate(200,0)" >
23     <g transform="scale(-1,1)" >
24       <use xlink:href="#edge"/>
25     </g>
26   </g>
27 </g>

```

- 19行目から20行目で横の直線を定義しています。
- 21行目で左の矢印の部分を定義しています。
- 22行目から26行目で右の矢印を描いています。
- 右の矢印は全体を横の直線の左端に平行移動させます(transform="translate(200,0)")。
- 矢印の向きを反転させるために23行目でtransform="scale(-1,1)"をしています。x座標の値を-1倍し、y座標の値をそのまま(1倍)に指定しています。

- 28行目から39行目で下の部分の図形を定義しています。

```

28 <g transform="translate(80,150)" >
29   <g>
30     <line x1="0" y1="0" x2="200" y2="0"
31       stroke-width="10" stroke-linecap="round" stroke="black"/>
32     <g transform="scale(-1,1)">
33       <use xlink:href="#edge"/>
34     </g>
35     <g transform="translate(200,0)">
36       <use xlink:href="#edge"/>
37     </g>
38   </g>
39 </g>
```

- 矢印の向きが上と逆になっているので左の矢印は属性 `transform="scale(-1,1)"`を持つ`<g>`要素の中に入れて左右の向きの入れ替えを行っています。
- 右の矢印は属性 `transform="translate(200,0)"`を持つ`<g>`要素の中に入れて平行移動しています。

問題 2.3 ミューラー・ライヤーの錯視図形で次のことを調べなさい。

1. 両端の矢印の長さを変える
2. 両端の矢印の色を変えることとそのときの見え方の違い
3. 両端の矢印の角度を変えることとそのときの見え方の変化
4. `<defs>`要素を用いないでこの図形を定義して、矢印の長さを変えたときの手間の違い

2.4 長方形

長方形を表すのは`<rect>`要素です。長方形の属性を表 2.3 に掲げます。

ポッケンドルフの錯視 図 2.10 は中央部が隠されているために直線の対応が見誤るというポッケンドルフの錯視 [32, 58 ページ] として知られる図形です。長方形と直線を使って描いています。

SVG リスト 2.5: ポッケンドルフ錯視

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3    xmlns:xlink="http://www.w3.org/1999/xlink"
4    height="100%" width="100%">
5  <title>ポッケンドルフ錯視</title>
6  <g transform="translate(170,140)">
7    <line x1="100" y1="-100" x2="-100" y2="100" stroke-width="3" stroke="black"/>
8    <line x1="0" y1="-20" x2="-100" y2="80" stroke-width="3" stroke="black"/>
9    <line x1="0" y1="-40" x2="-100" y2="60" stroke-width="3" stroke="black" />
10   <rect x="-30" y="-120" width="60" height="220" stroke-width="3" stroke="black" fill="white" />
```

表 2.3: <rect> 要素の属性

属性名	意味	属性値
x	長方形の左上の位置の x 座標	数値
y	長方形の左上の位置の y 座標	数値
width	長方形の幅	負でない数値
height	長方形の高さ	負でない数値
stroke-width	長方形の縁取りの幅	負でない数値
stroke	長方形の縁取りの色	色
fill	長方形の内部の塗りつぶしの色	色
rx	長方形の角に丸みを付けはじめる水平方向の位置	負でない数値
ry	長方形の角に丸みを付けはじめる垂直方向の位置	負でない数値

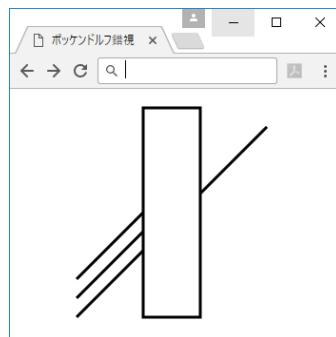


図 2.10: ポッケンドルフ錯視

```

11    </g>
12  </svg>

```

- 3 本の直線を下から描いています。
- 9 行目で一番下にある、右側に飛び出している直線を描いています。
- 7 行目、8 行目に一番下の直線より半分の長さのものを縦方向に 20 ずつ移動した形で描いています。
- 最後に、中央部を隠すように長方形を描いています(10 行目)。ここでは内部を白で塗りつぶしているので先に描かれた直線の部分でこれと重なる部分は見えなくなります。

色の対比 図 2.11 は正方形の内部が同じ色なのに周りの色で見え方が異なるという錯視图形です。

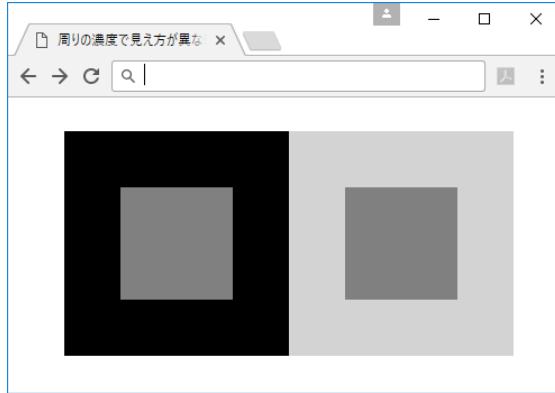


図 2.11: 周りの濃度で見え方が異なる

SVG リスト 2.6: 周りの濃度で見え方が異なる

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>周りの濃度で見え方が異なる</title>
6  <g transform="translate(50,30)" >
7      <rect x="0" y="0" width="200" height="200" fill="black" />
8      <rect x="50" y="50" width="100" height="100" fill="gray" />
9  </g>
10 <g transform="translate(250,30)" >
11     <rect x="25" y="25" width="150" height="150" fill="gray"
12         stroke-width="50" stroke="lightgray" />
13 </g>
14 </svg>

```

この図形は左の部分は二つの正方形を重ねて描いています。右の部分は縁取りの幅を大きくして同じような大きさの図形になる用の表示位置や長方形の大きさを調整しています。同じ図形を描くのでもいろいろな方法があることを確認してください。

- 左の部分は6行目から9行目の部分で描かれています。大きな正方形(7行目)の上に小さな正方形(8行目)を描いています。
- 右の部分は11行目から12行目のひとつの正方形で描かれています。線の幅(stroke-width)の半分だけ元来の図形の外側にはみ出しますので左上の位置をその分だけ移動させています。また、線の幅だけwidthとheightの値が左の正方形の値より小さくなっています。

問題 2.4 図 2.12 の 6 個の長方形は一番左が赤 (#FF0000) で一番右がオレンジ (#FFA500) で塗られています。間にある長方形の色はこの 2 つの色を補間しています [37, カラー図版 3]。

境界での色の差が強調されて見えますが、境界を指で隠すと両者の色の区別がつかなくなります。この図を作成しなさい。また、ほかの色の組み合わせでも調べなさい。

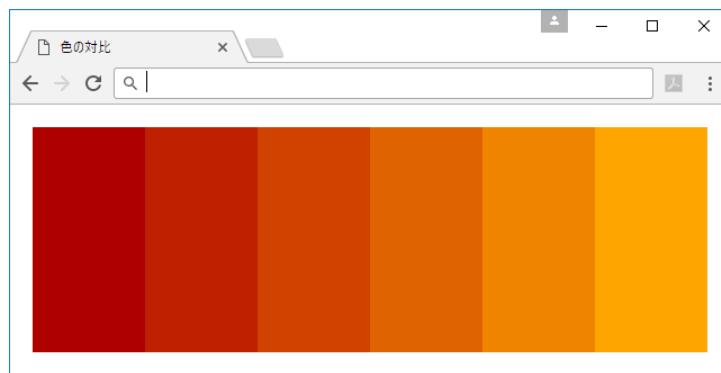


図 2.12: 色の対比

ヘルマン格子 図 2.13 は白い線の交差している位置に灰色のちらつきが表れるというヘルマン格子 [32, 180 ページ] と呼ばれるものです。

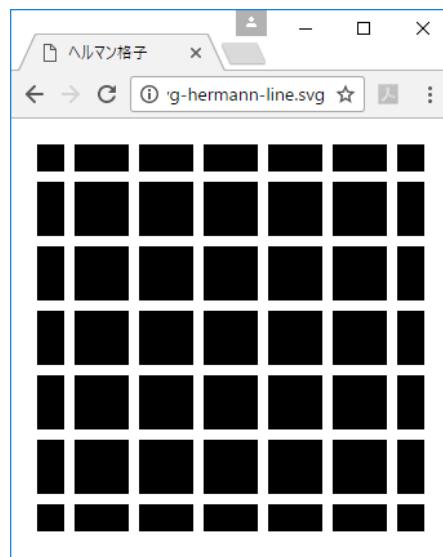


図 2.13: ヘルマン格子

SVG リスト 2.7: ヘルマン格子

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="330" width="330">
5  <title>ヘルマン格子</title>
6  <defs>
7      <line id="vertical" x1="0" y1="0" x2="0" y2="300" stroke-width="8" stroke="white"/>
8      <g id="horizontal" transform="rotate(-90)">
```

```

9      <use xlink:href="#vertical"/>
10     </g>
11   </defs>
12   <g transform="translate(20,20)">
13     <rect x="0" y="0" width="300" height="300" fill="black" />
14     <use x="25" y="0" xlink:href="#vertical" />
15     <use x="75" y="0" xlink:href="#vertical" />
16     <use x="125" y="0" xlink:href="#vertical" />
17     <use x="175" y="0" xlink:href="#vertical" />
18     <use x="225" y="0" xlink:href="#vertical" />
19     <use x="275" y="0" xlink:href="#vertical" />
20
21     <use x="0" y="25" xlink:href="#horizontal" />
22     <use x="0" y="75" xlink:href="#horizontal" />
23     <use x="0" y="125" xlink:href="#horizontal" />
24     <use x="0" y="175" xlink:href="#horizontal" />
25     <use x="0" y="225" xlink:href="#horizontal" />
26     <use x="0" y="275" xlink:href="#horizontal" />
27   </g>
28 </svg>
```

- 7行目で垂直線の開始と終了の位置、幅と色を定めています。
- 8行目から10行目で7行目の垂直線を回転させて水平方向の直線にしています。
- 13行目で背景を黒にするための正方形を定義しています。
- 14行目から19行目で垂直方向の直線を描いています。<use>要素のなかで属性xやyを指定することで参照している図形の表示位置を移動できます。
- 21行目から26行目で水平方向の直線を描いています。

問題2.5 リスト2.7について次の事柄を検討しなさい。

1. 成分の間隔や幅、または色をいろいろ変えてどれが一番良く錯視が見えるか
2. 垂直線をひとつのグループにして、それを参照することで水平線の記述を簡略化すること
3. 正方形を並べて同じような図形を描くこと

問題2.6 図2.14はカフェウォール錯視[32, 62ページ]と呼ばれる錯視図形です³。水平線はすべて平行なのですが黒い正方形がずれているために平行に見えません。この図形を描きなさい。

2.5 円と橢円

<circle>要素は円の属性を表します。また、<ellipse>要素は橢円を表します。表2.4にこの二つの要素の代表的な属性を掲げます。

橢円の属性は円の場合の半径を表すrの代わりにx軸、y軸の方向の軸の長さをそれぞれ表すrx、ry属性があります。この値を同じにすれば円となります。

³水平線が黒の場合にはミュンスターベルグ錯視と呼ばれます。

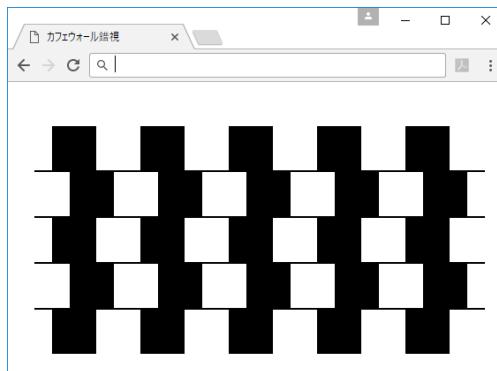


図 2.14: カフェウォール錯視

表 2.4: 円と楕円の属性

属性名	説明	値
cx	円、楕円の中心の x 座標	数値
cy	円、楕円の中心の y 座標	数値
r	円の半径	数値
rx	楕円の x 軸方向の長さ	数値
ry	楕円の y 軸方向の長さ	数値
stroke	縁取りを塗る色	色名または rgb 値で与える
stroke-width	縁取りの幅	数値
fill	内部を塗る色	色名または rgb 値で与える

周りの大きさで見え方が変わる 例 2.15 は同じ大きさの円の周りに大きさの違う円を並べたものです。中央の円は、周りの円が小さいと大きく、周りの円が大きいと小さく見えます。

SVG リスト 2.8: 周りの大きさで見え方が変わる

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>周りの大きさで見え方が変わる</title>
6  <defs>
7      <circle cx="0" cy="0" id="CCircle" r="40" fill="black" />
8      <circle cx="0" cy="0" id="LCircle" r="60" fill="black" />
9      <circle cx="0" cy="0" id="SCircle" r="20" fill="black" />
10 </defs>
11 <g transform="translate(150,200)">
12     <use xlink:href="#CCircle"/>
13     <g transform="translate(0,75)">
14         <use xlink:href="#SCircle"/>
15     </g>

```

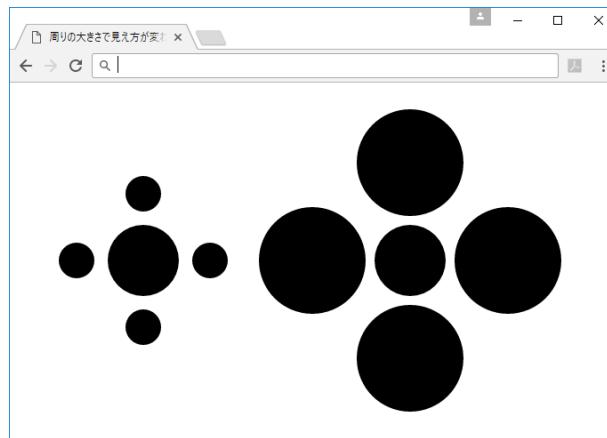


図 2.15: 周りの大きさで見え方が変わる

```
16      <g transform="rotate(90),translate(0,75)">
17          <use xlink:href="#SCircle"/>
18      </g>
19      <g transform="rotate(180),translate(0,75)">
20          <use xlink:href="#SCircle"/>
21      </g>
22      <g transform="rotate(270),translate(0,75)">
23          <use xlink:href="#SCircle"/>
24      </g>
25  </g>
26  <g transform="translate(450,200)">
27      <use xlink:href="#CCircle"/>
28      <g transform="translate(110,0)">
29          <use xlink:href="#LCircle"/>
30      </g>
31      <g transform="rotate(90),translate(110,0)">
32          <use xlink:href="#LCircle"/>
33      </g>
34      <g transform="rotate(180),translate(110,0)">
35          <use xlink:href="#LCircle"/>
36      </g>
37      <g transform="rotate(270),translate(110,0)">
38          <use xlink:href="#LCircle"/>
39      </g>
40  </g>
41 </svg>
```

- 7行目で中央にある円を定義し、それに CCircle という id を付けています。
- 8行目で右側の周りに置く円のひとつを定義し、それに LCircle という id を付けています。
- 9行目で左側に描く円を定義しています。この円には LCircle という id を付けています。

- 11 行目から 25 行目で左側の図形を定義しています。 $(0,0)$ の位置に中央の円を描き、その周りに SCircle で参照する小さな円を 4 つ付けています。小さな円を描く方法として 16 行目で transform の値を rotate(90), translate(0, 75) と二つ指定しています。これは次のように記述したものと同じです。

```
<g transform="rotate(90)">
  <g transform="translate(0,75)">
    <use xlink:href="#SCircle"/>
  </g>
</g>
```

- 右側の図形も 26 行目から 40 行目で同様に描いています。

問題 2.7 リスト 2.15 において次のことをしなさい。

- <defs> 要素内で定義された、周りにある円の属性値を変えて、全体の記述を簡単にしなさい。
- 中心の円や周りの円の色を変えたときに見え方が変わらかどうか調べなさい。
- 円の代わりに正方形で同様の図形を作成しなさい。

問題 2.8 図 2.16 はデルブーフの錯視 [32, 59 ページ] と呼ばれています。この図は [37, 131 ページ 図 12.7] からとりました。左右の単独にある円が中央で重なっています。小さいほうの円は左より大きく見え、大きいほうの円は右より小さく見えます。

この現象は円を正方形に変えてでも起こります。正方形による同様な図を描いて確認しなさい。

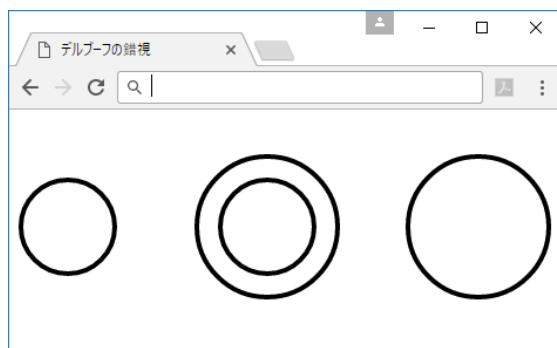


図 2.16: デルブーフの錯視

2.6 グラデーション

今まで長方形の内部を単一の色で塗りつぶしました。これ以外に色が順に変化するグラデーションという塗り方をがあります。SVG では 2 種類のグラデーションが利用できます。ここでは簡単なグラデーションの定義と使い方だけを紹介します。

グラデーションは<defs> 要素の中で定義し、後から参照するための属性 id をつけます。

2.6.1 線形グラデーション

線形グラデーションの基礎 開始位置と終了位置の色を指定することで途中の色が中間の色に変わっていくものです。次の例を見てください。

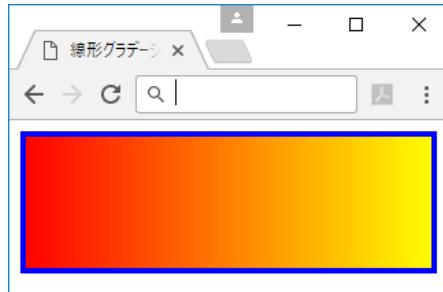


図 2.17: 線形グラデーション

SVG リスト 2.9: 線形グラデーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5      <title>線形グラデーション</title>
6      <defs>
7          <linearGradient id="Gradient1" gradientUnits="objectBoundingBox">
8              <stop stop-color="red" offset="0%" />
9              <stop stop-color="yellow" offset="100%" />
10             </linearGradient>
11         </defs>
12         <g transform="translate(10,10)">
13             <rect x="0" y="0" width="300" height="100" stroke-width="4" stroke="blue"
14                 fill="url(#Gradient1)" />
15         </g>
16     </svg>

```

- グラデーションのパターンは`<defs>`要素の中で定義します(11行目で`</defs>`終了)。
- 7行目が線形の定義の開始を示す`<linearGradient>`要素です。この要素には次のような属性が与えられています。
 - `id`後で参照するための名称を定義するものです。14行目で参照されています。`id`の属性値は他のものと重複してはいけません。
 - `gradientUnits`はグラデーションをどの座標系で指定するかを表します。ここではオブジェクトの座標系で決めることを示す`objectBoundingBox`を指定しています。このほかにグラデーションを適用するオブジェクトをとりまく座標系を基準にする`userSpaceOnUse`があります。この違いについては29ページ以降で説明します。

- 8行目と9行目にグラデーションの特定の位置に対する場所(offset)とそこでの色(stop-color)を<stop>要素で定義しています。ここでは開始位置(offset="0%")の色を赤に、終了位置(offset="100%")の色を黄色に指定しています。
したがって、このグラデーションは左の赤から右に行くにしたがい黄色へと変化することになります。なお、<stop>要素は途中の値も指定できるので2つよりも多くてもかまいません。
- <linearGradient>要素のなかで別の要素を宣言しているのでこの要素に対する終了を示す</linearGradient>が必要です(10行目)。
- 13行目から14行目で長方形のオブジェクトを宣言しています。fillの値はグラデーションの定義を参照するためにurl(#Gradient1)と表しています。#の後にidで定義したラベルを用いていることに注意してください。

問題 2.9 図 2.18 はマッハバンド錯視 [32, 99 ページ] と呼ばれています。この図では左 1/3 の位置から右 1/3 の間だけにグラデーションを付けているだけです。グラデーションの開始の位置では少し明るく色が強調されて見えています。

この図を SVG で作成しなさい。また、色やグラデーションの位置をいろいろ変えてどのように見えるか確認しなさい。

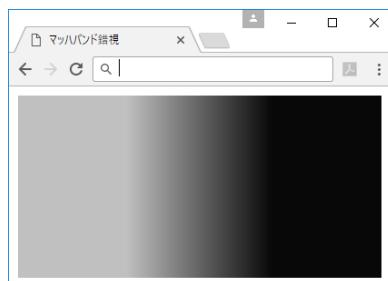


図 2.18: マッハバンド錯視

問題 2.10 図 2.19 はザバーニョの錯視と呼ばれています。グラデーションがついた長方形を 90° ずつ回転したものを並べただけですが中央部がより明るく見えます。この図を SVG で作成しなさい。また、色やグラデーションをいろいろ変えてどのように見えるか確認しなさい。

線形グラデーションの向きを変える

図 2.20 はグラデーションが左上から右下に変化しています。

リスト 2.10 はこの図を描くものです。

SVG リスト 2.10: 傾いた方向の線形グラデーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"

```

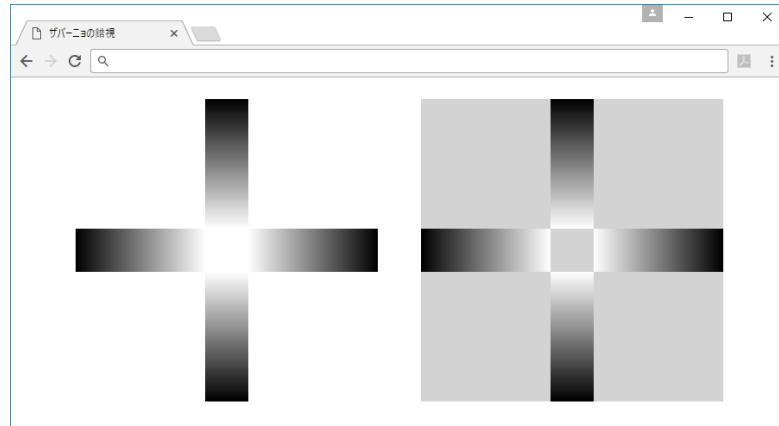


図 2.19: ザバーニョの錯視

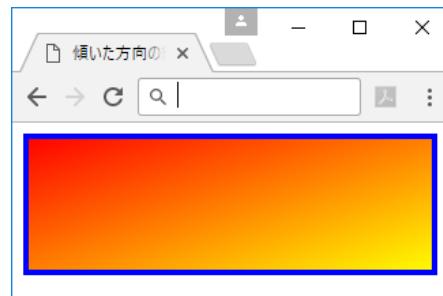


図 2.20: 傾いた方向の線形グラデーション

```

3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5      <title>傾いた方向の線形グラデーション</title>
6      <defs>
7          <linearGradient id="Gradient1" gradientUnits="objectBoundingBox"
8              x1="0%" y1="0%" x2="100%" y2="100%">
9              <stop stop-color="red" offset="0%" />
10             <stop stop-color="yellow" offset="100%" />
11         </linearGradient>
12     </defs>
13     <g transform="translate(10,10)">
14         <rect x="0" y="0" width="300" height="100"
15             stroke-width="4" stroke="blue" fill="url(#Gradient1)" />
16     </g>
17 </svg>
```

このリストとリスト 2.9 の違いは8行目の記述が追加してあるだけです。x1 と y1 でグラデーションの開始位置を指定できます。ここではともに 0% なので左上が指定されます。x2 と y2 でグラデーションの終了位置を指定できます。ここではともに 100% なので右下が指定されます。したがって、

グラデーションの方向は左上から右下に向かうことになります。

問題 2.11 リスト 2.10 について次のことを調べなさい。

1. $x1$ 、 $y1$ 、 $x2$ と $y2$ の値をいろいろ変えたときグラデーションの向きがどのように変わるか
2. 長方形の縦横比が異なったものにたいしてグラデーションがどのように変化するか
3. グラデーションの方向が右上から左下 45° の方向になるようにすること

問題 2.12 図 2.21 はコフカリング [32, 90 ページ] です。背景の明度の差がある部分を中央のグラディエーションの部分が隠しているので、同じ色の円の縁取りが明るさの違う色に見えます。縁取りと同じ色で塗っても明度に差があるように見えることを確認しなさい。

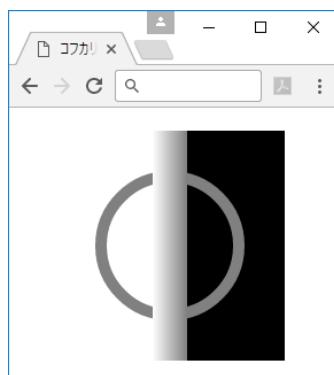


図 2.21: コフカリング

問題 2.13 図 2.22 はひし形を用いたクレイク・オブライエン効果と呼ばれています ([37, 58 ページ 図 6.4])。ひし形を塗っているグラデーションはどこも同じですが下の方が明るく見えます。

`gradientUnits` の値の違い `gradientUnits` の値として `objectBoundingBox` と `userSpaceOnUse` があることはすでに説明しました。図 2.23 はこの違いを説明するためのものです。色の変化を見やすくするために色の変化が完全に別な色になるようなグラディエーションを付けています。

SVG リスト 2.11: `gradientUnits` の値の違い

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>線形グラデーションの gradientUnits の違い</title>
6  <defs>
7      <linearGradient id="GradBB" gradientUnits="objectBoundingBox">
8          <stop stop-color="red"      offset="0%" />
9          <stop stop-color="red"      offset="20%" />
10         <stop stop-color="yellow" offset="20%" />
11         <stop stop-color="yellow" offset="80%" />
```

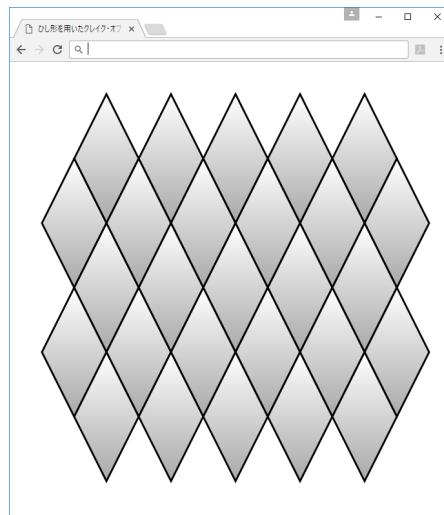


図 2.22: ひし形を用いたクレイク・オブライエン効果

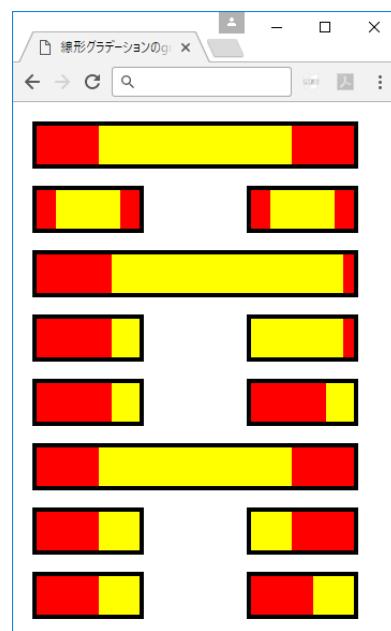


図 2.23: 線形グラデーションの gradientUnits の違い

```
12      <stop stop-color="red"      offset="80%" />
13      <stop stop-color="red"      offset="100%" />
14    </linearGradient>
15    <linearGradient id="GradUS1" gradientUnits="userSpaceOnUse"
16      x1="0%" y1="0%" x2="100%" y2="0%">
17      <stop stop-color="red"      offset="0%" />
```

```

18      <stop stop-color="red"      offset="20%" />
19      <stop stop-color="yellow"   offset="20%" />
20      <stop stop-color="yellow"   offset="80%" />
21      <stop stop-color="red"      offset="80%" />
22      <stop stop-color="red"      offset="100%" />
23  </linearGradient>
24  <linearGradient id="GradUS2" gradientUnits="userSpaceOnUse"
25      x1="0" y1="0" x2="300" y2="0" >
26      <stop stop-color="red"      offset="0%" />
27      <stop stop-color="red"      offset="20%" />
28      <stop stop-color="yellow"   offset="20%" />
29      <stop stop-color="yellow"   offset="80%" />
30      <stop stop-color="red"      offset="80%" />
31      <stop stop-color="red"      offset="100%" />
32  </linearGradient>
33  <rect id="RL" width="300" height="40" stroke-width="4" stroke="black" />
34  <rect id="RS" width="100" height="40" stroke-width="4" stroke="black" />
35 </defs>
36 <g transform="translate(20,20)">
37      <use xlink:href="#RL" x="0" y="0" fill="url(#GradBB)" />
38      <use xlink:href="#RS" x="0" y="60" fill="url(#GradBB)" />
39      <use xlink:href="#RS" x="200" y="60" fill="url(#GradBB)" />
40 </g>
41 <g transform="translate(20,140)">
42      <use xlink:href="#RL" x="0" y="0" fill="url(#GradUS1)" />
43      <rect x="0" y="60" width="100" height="40"
44          stroke-width="4" stroke="black" fill="url(#GradUS1)" />
45      <rect x="200" y="60" width="100" height="40"
46          stroke-width="4" stroke="black" fill="url(#GradUS1)" />
47      <use xlink:href="#RS" x="0" y="120" fill="url(#GradUS1)" />
48      <use xlink:href="#RS" x="200" y="120" fill="url(#GradUS1)" />
49 </g>
50 <g transform="translate(20,320)">
51      <use xlink:href="#RL" x="0" y="0" fill="url(#GradUS2)" />
52      <rect x="0" y="60" width="100" height="40"
53          stroke-width="4" stroke="black" fill="url(#GradUS2)" />
54      <rect x="200" y="60" width="100" height="40"
55          stroke-width="4" stroke="black" fill="url(#GradUS2)" />
56      <use xlink:href="#RS" x="0" y="120" fill="url(#GradUS2)" />
57      <use xlink:href="#RS" x="200" y="120" fill="url(#GradUS2)" />
58 </g>
59 </svg>

```

7 行目から14 行目では gradientUnits の値が objectBoundingBox となる線形グラデーションを定義しています。

```

7  <linearGradient id="GradBB" gradientUnits="objectBoundingBox">
8      <stop stop-color="red"      offset="0%" />
9      <stop stop-color="red"      offset="20%" />
10     <stop stop-color="yellow"   offset="20%" />
11     <stop stop-color="yellow"   offset="80%" />
12     <stop stop-color="red"      offset="80%" />
13     <stop stop-color="red"      offset="100%" />
14 </linearGradient>

```

- 色の変化位置を明確にするために同じ位置で二つの色を定義しています(9行目と10行目、11行目と12行目)。これにより左から20%までの位置は赤に、そこから80%までは黄色に、そして残りの部分は再び赤に塗られます。
- 15行目から23行目ではgradientUnitsの値がuserSpaceOnUseとなる線形グラデーションを定義しています。このグラデーションは16行目でx1、y1、x2とy2を割合(%)で定義しています。グラデーションの割合は前と同じです。
- 24行目から32行目でもgradientUnitsの値がuserSpaceOnUseとなる線形グラデーションを定義しています。このグラデーションでは25行目でx1、y1、x2とy2を数値で定義しています。このグラデーションの割合も前と同じです。
- 33行目と34行目ではグラデーションをつける二種類の長方形を定義しています。
- 初めのグラデーションを付けているグループは上から2つ並んでいるものです。

```

36 <g transform="translate(20,20)">
37   <use xlink:href="#RL" x="0" y="0" fill="url(#GradBB)"/>
38   <use xlink:href="#RS" x="0" y="60" fill="url(#GradBB)"/>
39   <use xlink:href="#RS" x="200" y="60" fill="url(#GradBB)"/>
40 </g>
```

- これらの長方形は33行目と34行目で定義されたものを<use>要素を用いて利用しています。
- これらのグラデーションはgradientUnitsの値がobjectBoundingBoxとなっているので色が塗られるオブジェクトの位置が基準になっています。したがって、長方形の幅や位置に関係なくグラデーション全体が指定された割合でつくことになります。
- 次の3行にわたって並んでいる長方形は15行目から23行目で指定したグラデーションで塗られています。

```

41 <g transform="translate(20,140)">
42   <use xlink:href="#RL" x="0" y="0" fill="url(#GradUS1)" />
43   <rect x="0" y="60" width="100" height="40"
44     stroke-width="4" stroke="black" fill="url(#GradUS1)" />
45   <rect x="200" y="60" width="100" height="40"
46     stroke-width="4" stroke="black" fill="url(#GradUS1)" />
47   <use xlink:href="#RS" x="0" y="120" fill="url(#GradUS1)"/>
48   <use xlink:href="#RS" x="200" y="120" fill="url(#GradUS1)"/>
49 </g>
```

- 2つ目のグループの小さい長方形の色の変化の位置がその上の長方形と同じです。この列の位置の基準が41行目にある<g>要素で規定されている(userSpaceOnUse)からです。
- 3つ目のグループの小さい長方形は前に定義した小さな長方形を<use>要素で引用しています。この場合にはこのなかで新しい基準が用いられるので両方とも左端が赤になっています。
- 最後のグループは24行目から32行目で定義したグラデーションで塗られています。

```

50 <g transform="translate(20,320)">
51   <use xlink:href="#RL" x="0" y="0" fill="url(#GradUS2)"/>
52   <rect x="0" y="60" width="100" height="40"
53     stroke-width="4" stroke="black" fill="url(#GradUS2)"/>
54   <rect x="200" y="60" width="100" height="40"
55     stroke-width="4" stroke="black" fill="url(#GradUS2)"/>
56   <use xlink:href="#RS" x="0" y="120" fill="url(#GradUS2)"/>
57   <use xlink:href="#RS" x="200" y="120" fill="url(#GradUS2)"/>
58 </g>

```

- この場合には上の二つの色の変化位置は同じです。また、一番最後の行はやはり小さな長方形の左端が赤くなっています。
- 2番目のグループと3番目のグループでは色の変化の場所が少し異なります。これは2番目のグループの右端の基準がブラウザ画面の右端になっているためです。それが証拠に、ブラウザの画面の横幅を変えると2番目のグループだけが色に変化が起こります。

問題 2.14 図 2.23 でブラウザの幅を変化させたとき、グラデーションがどのように変化するか調べ、その理由を考えなさい。

2.6.2 放射グラデーション

一点を中心として順次色の変化がつく放射グラデーションを表す<radialGradient>要素があります。放射グラデーションには次の属性があります。

表 2.5: 放射グラデーションの属性

属性名	意味	値
cx	放射グラデーションの終了位置の円の中心の x 座標	数値または割合
cy	放射グラデーションの終了位置の円の中心の y 座標	数値または割合
r	放射グラデーションの終了位置の円の半径	数値または割合
fx	放射グラデーションの中心の x 座標	数値または割合
放射グラデーションの中心の y 座標	数値または割合	

これらの属性の値は gradientUnits の値により解釈が異なります。userSpaceOnUse のときは数値がそのまま採用され、objectBoundingBox のときは使用されるオブジェクトの大きさに対する割合を意味します。次の二つの例を見比べてください。

リスト 2.12 は図 2.24 の左側の図のものです。

SVG リスト 2.12: 放射グラデーション (userSpaceOnUse の場合)

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3    xmlns:xlink="http://www.w3.org/1999/xlink"
4    height="100%" width="100%">

```

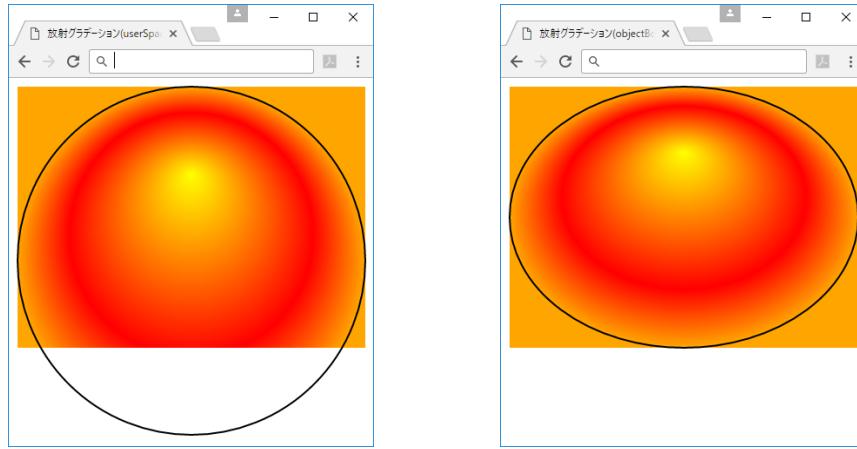


図 2.24: 放射グラデーション (userSpaceOnUse(左) と objectBoundingBox(右))

```

5   <title>放射グラデーション (userSpaceOnUse の場合)</title>
6   <defs>
7     <radialGradient id="radGradient2" gradientUnits="userSpaceOnUse"
8       cx="200" cy="200" r="200" fx="200" fy="100" >
9       <stop stop-color="yellow" offset="0%" />
10      <stop stop-color="red" offset="70%" />
11      <stop stop-color="orange" offset="100%" />
12    </radialGradient>
13  </defs>
14  <g transform="translate(10,10)">
15    <rect x="0" y="0" width="400" height="300" fill="url(#radGradient2)" />
16    <circle cx="200" cy="200" r="200"
17      stroke-width="2" stroke="black" fill="none"/>
18  </g>
19</svg>

```

- 7行目から12行目で放射グラデーションを定義しています。ここではグラデーションをする基準の座標系を userSpaceOnUse としています。
- グラデーションの終了位置を示す円の位置と大きさとグラディエーションの開始位置をすべて数値で指定しています(8行目)。
- 9行目から11行目でグラデーションの途中の色を線形グラデーションと同じ方法で指定しています。
- 15行目で定義している長方形の内部をここで定義したグラディエーションで塗っています。終了位置の円の外側は最後の色をそのまま使うのがデフォルトです。
- 放射グラデーションの端を示すために16行目から17行目で円の縁を描いています(fill の値が none になっていることに注意すること)。

リスト 2.12 は図 2.24 の右側の図のものです。

SVG リスト 2.13: 放射グラデーション (objectBoundingBox の場合)

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>放射グラデーション (objectBoundingBox の場合)</title>
6  <defs>
7      <radialGradient id="radGradient2" gradientUnits="objectBoundingBox"
8          cx="50%" cy="50%" r="50%" fx="50%" fy="25%" >
9          <stop stop-color="yellow" offset="0%"/>
10         <stop stop-color="red" offset="70%"/>
11         <stop stop-color="orange" offset="100%"/>
12     </radialGradient>
13 </defs>
14 <g transform="translate(10,10)">
15     <rect x="0" y="0" width="400" height="300" fill="url(#radGradient2)" />
16     <ellipse cx="200" cy="150" rx="200" ry="150"
17         stroke-width="2" stroke="black" fill="none"/>
18 </g>
19 </svg>
```

- 7 行目から 12 行目で放射グラデーションを定義しています。ここでグラデーションをする基準の座標系を objectBoundingBox としています。
- グラデーションの終了位置を示す円の位置と大きさとグラデーションの開始位置をすべて割合で指定しています (7 行目)。
- グラデーションの途中の色は線形グラデーションと同じ方法で指定しています (9 行目から 11 行目)。

問題 2.15 図 2.25 はクレイク・オブライエン効果とよばれるものです。内側にある境界部分の黒が内部の白を外部よりより白く見えさせています。放射グラディエーションを利用してこの図を描きなさい。また、色を変えて同じような図を作成した場合にはどのようになるか調べなさい。

2.7 不透明度

不透明度⁴を設定した図形はその設定した値の応じて色合いが薄くなり、その下にある図形が見えるようになります。不透明度が 1 では下の図形がまったく見えず、0 では設定された図形がまったく見えなくなります。不透明度が設定できる属性は表 2.6 を参照してください。

図 2.26 は円の内部の不透明度を 0.2 に設定して重ね合わせたものです。

⁴不透明度はアルファ値とかアルファチャンネル呼ばれることがあります。

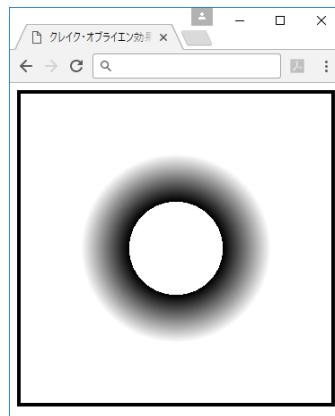


図 2.25: クレイク・オブライエン効果

表 2.6: 不透明度の種類

設定できる要素	設定のための属性名	説明
図形一般	opacity	対象の図形全体に不透明度が設定される。
図形一般	stroke-opacity	図形の属性 <code>stroke</code> に設定される。
図形一般	fill-opacity	図形の属性 <code>fill</code> に設定される。
<stop> 要素	stop-opacity	不透明度のグラデーションが設定できる。

SVG リスト 2.14: 円の内部に不透明度を設定した例

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3   xmlns:xlink="http://www.w3.org/1999/xlink"
4   height="100%" width="100%">
5 <title>円の内部に不透明度を設定した例</title>
6 <defs>
7   <circle id="Circle" cx="50" cy="0" r="100"/>
8   <g id="Fig0" >
9     <use xlink:href="#Circle"/>
10    <g transform="rotate(60)">
11      <use xlink:href="#Circle"/>
12    </g>
13    <g transform="rotate(120)">
14      <use xlink:href="#Circle"/>
15    </g>
16  </g>
17  <g id="Fig">
18    <use xlink:href="#Fig0"/>
19    <g transform="rotate(180)">
20      <use xlink:href="#Fig0"/>
21    </g>
22  </g>

```

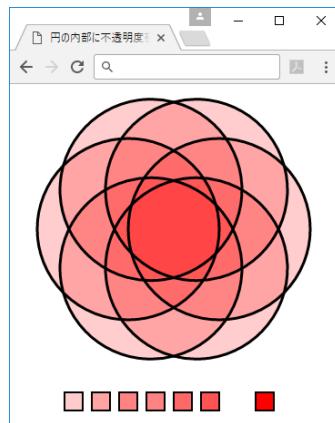


図 2.26: 円の内部に不透明度を設定した例

```

23   </defs>
24   <g transform="translate(180,160)">
25     <use xlink:href="#Fig" fill-opacity="0.2" fill="red" stroke="none"/>
26     <use xlink:href="#Fig" stroke-width="3" stroke="black" fill="none"/>
27     <rect x="-120" y="180" width="20" height="20"
28       fill="rgb(100%,80%,80%)" stroke-width="2" stroke="black"/>
29     <rect x="-90" y="180" width="20" height="20"
30       fill="rgb(100%,64%,64%)" stroke-width="2" stroke="black"/>
31     <rect x="-60" y="180" width="20" height="20"
32       fill="rgb(100%,51.2%,51.2%)" stroke-width="2" stroke="black"/>
33     <rect x="-30" y="180" width="20" height="20"
34       fill="rgb(100%,51.2%,51.2%)" stroke-width="2" stroke="black"/>
35     <rect x="0" y="180" width="20" height="20"
36       fill="rgb(100%,40.96%,40.96%)" stroke-width="2" stroke="black"/>
37     <rect x="30" y="180" width="20" height="20"
38       fill="rgb(100%,32.768%,32.768%)" stroke-width="2" stroke="black"/>
39     <rect x="90" y="180" width="20" height="20"
40       fill="rgb(100%,0%,0%)" stroke-width="2" stroke="black"/>
41   </g>
42 </svg>
```

- 7 行目でこの図形を描くための共通の円を定義しています。この円には `fill` や `stroke` など通常の属性がまったく指定されていないことに注意してください。
- この円を 60° ずつ回転して全体で 6 個並べた図形を作成するためにまず、半分だけ `<use>` 要素を用いて作成します (8 行目から 16 行目)。
- これを 180° 回転したものと組み合わせて図形の離形を作成します (17 行目から 22 行目)。
- 25 行目で `fill-opacity`、`fill`、`stroke` の値を設定しています。引用された図形ではこの値が採用されます。

- 26行目では `stroke-width`、`stroke`、`fill` の値を設定しています。
- `opacity` がある図形の色は

`opacity`が定義された図形の色 × この図形の色 + (1 - この図形の `opacity`) × この図形の下にある色

という計算式で求められます。背景が白なのでこの図形ではすべての位置で RGB の赤の成分は 100% です。青と緑の成分はひとつ重なるごとに 0.8 倍されます。

- 具体的に `rgb` で指定した色に塗った正方形をこの図形の下に順番に描いています (27行目から40行目)。なお、一番右は赤に塗っています。

問題 2.16 リスト 2.14 に関して次の問い合わせに答えなさい。

1. `fill` と `stroke` を別に設定している図形を二つ重ねている理由はなにか。
2. 図のある部分の一番下を青で塗ったらどのような図形になるか
3. 円を放射グラデーションで塗ったらどうなるか
4. 円を放射グラデーションに `stop-opacity`を入れたらどうなるか

問題 2.17 図 2.27 はピラミッドの稜線とよばれています ([37, カラー図版 13])。色の濃度が異なる正方形がいくつか並んでいますが、頂点に沿って直線が描いてあるように見えます。左の二つは色を RGB 値で直接指定し、右の二つは不透明度を利用して一番下に指定した色を透かして見せています。この図を作成しなさい。

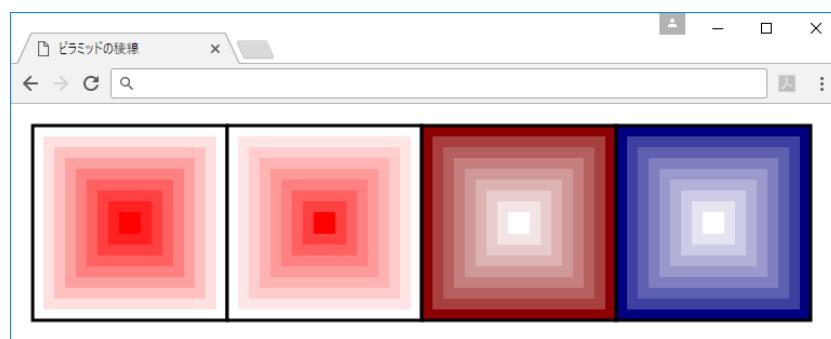


図 2.27: ピラミッドの稜線

第3章 SVG の図形

3.1 折れ線と多角形

直線をつなげて図形を描くものとして<polyline>要素と<polygon>要素があります。この二つの違いは図形が閉じない(<polyline>要素)か閉じるか(<polygon>要素)の違いです。両方のタグも点の位置はpointsで指定します。次の例は正5角形の頂点を結ぶ図形を描きます。

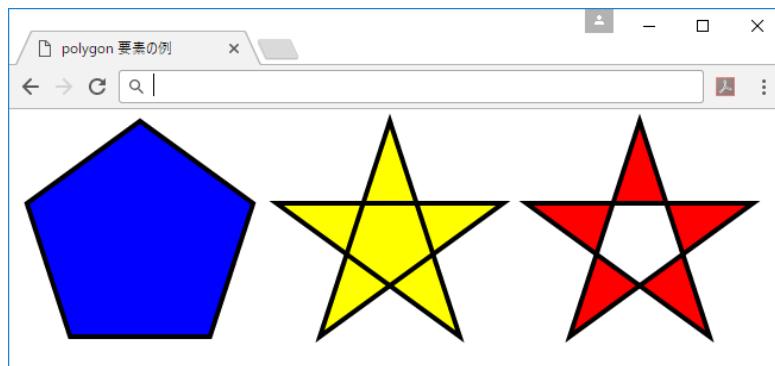


図 3.1: <polygon> 要素の例

SVG リスト 3.1: <polygon> 要素の例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>polygon 要素の例</title>
6  <g transform="translate(110,110)">
7      <g transform="scale(1,-1)">
8          <polygon
9              points="0,100 -95.1,30.9 -58.8,-80.9 58.8,-80.9 95.1,30.9"
10             stroke="black" stroke-width="4" fill="blue" />
11          <g transform="translate(210,0)">
12              <polygon
13                  points="0,100 -58.8,-80.9 95.1,30.9 -95.1,30.9 58.8,-80.9"
14                  stroke="black" stroke-width="4" fill="yellow" />
15          </g>
16          <g transform="translate(420,0)" fill="red" fill-rule="evenodd" >
17              <polygon
18                  points="0,100 -58.8,-80.9 95.1,30.9 -95.1,30.9 58.8,-80.9"
```

```

19      stroke="black" stroke-width="4"/>
20    </g>
21  </g>
22 </g>
23 </svg>
```

- 7行目では`transform`の`scale(1,-1)`を用いて座標系を数学で使われる通常の形にしています。
- 8行目から10行目で一番左にある正5角形を`<polygon>`要素を用いて描いています。頂点の座標は属性`points`を用いて与えます。頂点の`x`座標と`y`座標を空白または,で区切って与えます。ここでは`x`座標と`y`座標の間を,で、点の区切りを空白で区切って関係がわかりやすくなるように記述しました。
なお、点の座標は計算機などで別に計算しておく必要があります。¹ここでは円の半径が100なので一番画面の頂点にある点の位置は(0,100)となり、残りの点の位置はこれを72°ずつ回転して得られます。したがって、 $(100\cos(90+72^\circ), 100\sin(90+72^\circ))$ などの式を用いて計算しています。
- 塗られる範囲は点の位置を与えた順で決まります。12行目から14行目は正5角形の頂点の位置をひとつおきに与えた図形(星形)を描いています。通常はこれらの直線で囲まれた部分が塗られます。また、縁取りもすべて描かれます。
- 17行目から19行目の図形では新しい属性`fill-rule`があります(16行目)。
- 属性`fill-rule`の値は`evenodd`です。これにより図形の内部の点と無限点とを結んだ直線が縁取りの直線と奇数回交わる領域が`fill`で指定された色で塗られます。

この図では中央にある小さな5角形の部分が偶数回交わる領域なので塗られなくなります。

問題3.1 図3.1の個々の図形をグラデーションを用いて塗りつぶしなさい。

問題3.2 図3.2は図3.1の3つの図形をひとつと見て全体をひとつのグラディエーションで塗っています。下の長方形はグラデーションの比較のために描いています。この図形を描きなさい。

¹後の章ではSVGファイルの起動時に計算する方法を紹介します。

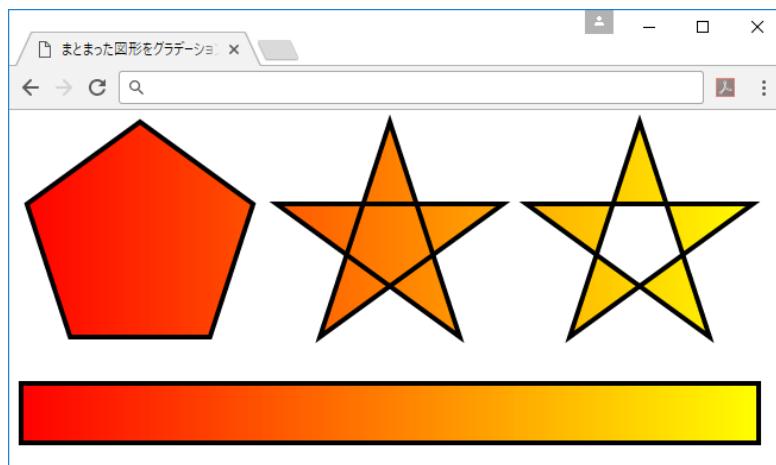


図 3.2: まとめた図形をグラデーションで図形を塗りつぶす

折れ線を結ぶとき頂点の形を制御することができます (図 3.3)。この図では線分の中央をわかりやすくするために細い線を追加しています。

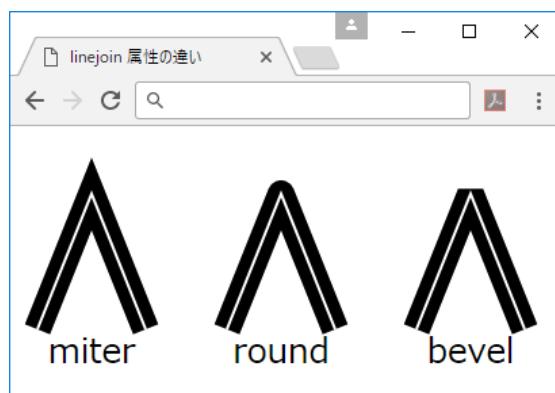


図 3.3: linejoin 属性の違い

- 線分をつなぎ合わせる頂点の形を制御する属性が `stroke-linejoin` です。デフォルトは `miter` です。
`miter` の欠点は二つの線分の交わる角度が小さいときは先端が鋭くなり、頂点の部分が大きくなる場合があります。
- その他の値として丸くする `round` と切り捨ててしまう `bevel` があります。
- なお、属性 `miterlimit` で交わる角度が一定角度より小さいときは `bevel` に、それより大きいときは `miter` になるように設定できます。

- なお、折れ線の指定では内部がないように思われるかもしれません、内部を塗る `fill` が指定できます。ここでは `none` にして塗らないようにしています。

この図のソースはリスト3.2です。図に文字を表示するために`<text>`要素を使用しています。文字列の表示については第5章で解説します。

SVG リスト3.2: `linejoin` 属性の違い

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5      <title> linejoin 属性の違い</title>
6      <defs>
7          <polyline id="hairline" points="-40,100 0,0 40,100"
8              fill="none" stroke-width="2" stroke="white"/>
9      </defs>
10     <g transform="translate(60,50)">
11         <polyline points="-40,100 0,0 40,100"
12             fill="none" stroke-width="20" stroke="black"/>
13         <use xlink:href="#hairline"/>
14         <text text-anchor="middle" x="0" y="125" font-size="25">miter</text>
15     </g>
16     <g transform="translate(200,50)">
17         <polyline points="-40,100 0,0 40,100" stroke-linejoin="round"
18             fill="none" stroke-width="20" stroke="black"/>
19         <use xlink:href="#hairline"/>
20         <text text-anchor="middle" x="0" y="125" font-size="25">round</text>
21     </g>
22     <g transform="translate(340,50)">
23         <polyline points="-40,100 0,0 40,100" stroke-linejoin="bevel"
24             fill="none" stroke-width="20" stroke="black"/>
25         <use xlink:href="#hairline" />
26         <text text-anchor="middle" x="0" y="125" font-size="25">bevel</text>
27     </g>
28 </svg>

```

- 7行目から8行目で折れ線の中央部に描く細い線を定義しています。この折れ線は13行目、19行目と25行目で引用されています。
- 11行目から12行目で左の直線を、17行目から18行目で中央の直線を、23行目から24行目で右の直線をそれぞれ描いています。
- それぞれの直線に `stroke-linejoin` が設定されていることを確認してください。

問題3.3 図3.4はシェバードの錯視 [32, 64ページ] と呼ばれています。二つの平行四辺形は同じ形をしていますが見え方が異なります。通常はテーブルの形に見せるように足を付けますが、ここでは省略しました。

この図を作成しなさい。

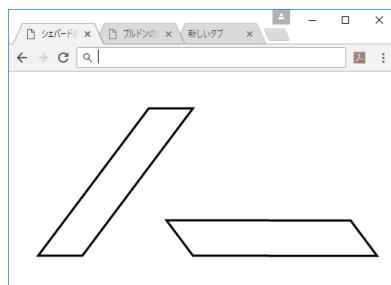


図 3.4: シェパーードの錯視

問題 3.4 図 3.5 はブルドンの錯視 [32, 73 ページ] と呼ばれています。二つの三角形の左の辺は一直線上に並んでいるのですが少し曲がって見えます。

この図を作成しなさい。また、傾きを変えたときにどのように見えるか確認しなさい。

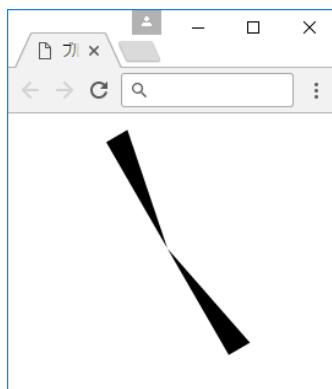


図 3.5: ブルドンの錯視

3.2 道のり (Path)

SVG は<path> 要素を用いて曲線や直線を組み合わせた図形を記述できます。<path> 要素 では図形は属性 `d` で指定します。表 3.1 は指定できるパラメータの一覧です。パラメータは大文字と小文字がありますが、大文字の場合は絶対座標で、小文字の場合は直前の位置からの相対座標で指定することを意味します。

次の節から順にこれらの指定の方法を見ていきます。

表3.1: dで指定できるパラメータ

パラメータ	指定する点の数	説明
M,m	1	指定した位置へ移動
L,l	1	指定した位置までパスを設定(デフォルト(省略可能))
A,a	1	橍円の弧の一部を指定した位置まで描く。このほかに6個のパラメータが必要
C,c	3	3次のBézier曲線を描く
S,s	2	直前のBézier曲線の後の制御点と最後の点に関して対称な位置にある制御点を持つ3次のBézier曲線を描く。
Q,q	2	2次のBézier曲線を描く。
T,t	1	直前のBézier曲線の後の制御点と最後の点に関して対称な位置にある制御点を持つ2次のBézier曲線を描く。
z		直前のMで開始した位置まで戻る。

3.2.1 直線の組み合わせ

図3.6はヴィントの錯視図形と呼ばれます。斜線により中央の平行線が中央部で細く見えるというものです。

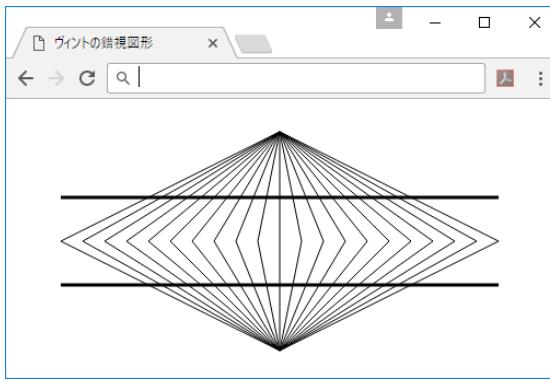


図3.6: ヴィントの錯視図形

SVGリスト3.3: ヴィントの錯視図形

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3   height="100%" width="100%">
4   <title>ヴィントの錯視図形</title>
5   <g id="canvas" transform="translate(250,130)">
6     <path stroke-width="1" stroke="black" fill="none">

```

```

7   d="M0,-100 -200,0 0,100 200 0 0,-100 -180,0 0,100 180,0 0,-100
8     -160,0 0,100 160,0 0,-100 -140,0 0,100 140,0 0,-100
9     -120,0 0,100 120,0 0,-100 -100,0 0,100 100,0 0,-100
10    -80,0 0,100 80,0 0,-100 -60,0 0,100 60,0 0,-100
11    -40,0 0,100 40,0 0,-100 -20,0 0,100 20,0 0,-100 0,100" />
12  <path stroke-width="3" stroke="black" fill="none"
13    d="M-200,-40 200,-40 M-200,40 200,40" />
14  </g>
15 </svg>

```

- 6 行目から 11 行目で錯視の原因となる上部の一点から放射状に広がって下部の一点へ集まる図形を `<path>` 要素を用いて作成しています (この図形は `<polyline>` 要素で描くことも可能です)。
 - `<path>` 要素の形状を示すための属性は `d` です。
 - 点の座標はすでに見てきた `<polyline>` 要素や `<polygon>` 要素と同様の方法で記述します。
 - 7 行目の `d` の先頭にある `M` で開始点への移動を指示しています。この位置は上部の線分が集中している場所の位置 $(0, -100)$ です。
 - 次は中央部の一番左の位置 $(-200, 0)$ です。`M` などの指定がありません。この場合には `L` を指定したものとみなされます。
 - 次は下部の線分が集中している位置 $(0, 100)$ です。
 - 次は中央部の右端の位置 $(0, 200)$ です。
 - 次は上部の線分が集中している位置に戻っています。
 - その後は x 座標の位置を少しずつ中央部に移動しながら残りの部分の位置を指定しています。
 - 直線だけしか描かない場合には属性 `fill` を `none` に指定しないと開始点と最後の点を結んでできる図形の内部が黒く塗られてしまいます (12 行目)。
 - 水平線も同様に `<path>` 要素で描いています。ここで注意することは属性 `d` の値に `M` が 2 回現れていることです (13 行目)。この前後で直線がつながなくなります。したがって、2 本の平行線はひとつの `<path>` 要素で定義することができます。
- このように `d` で指定された道のりは必ずしも連続でつながっている必要はありません。

問題 3.5 ヴィントの錯視図形で次のことを確かめなさい。

1. 放射状にでてくる線の位置や水平線の間隔を変えたときの図形はどのように見えるか確かめなさい。
2. 水平線の代わりに円や正方形を描いたときどのように見えるか確かめなさい ([36, 85 ページ参照])。

3. ヴィントの錯視图形を片目で見ると見え方が変わらかどか確かめなさい。

なお、このような图形では描く直線の数や間隔を変えたい場合にはそれぞれの点の座標を変えることになりますがその手間は大変です。このような場合にはプログラムで点の位置を出力するといろいろな場合の图形が簡単にかけます。プログラムで作成する方法については第7章で説明します。

<path>要素を用いて正方形を描くことができます。絶対座標で指定すると

```
d="M0,0 0,100 100,100 100,0z"
```

となります。zを使うので開始位置を再び指定する必要がありません。

一方、相対座標で指定すると

```
d="M0,0 10,100 100,0 0,-100z"
```

と初めの移動先の位置を指定するときに一度だけ¹で移動を定義しておくと残りは移動量だけですみます。

問題 3.6 正方形を描くとき、zを用いなくて

```
d="M0,0 0,100 100,100 100,0 0,0"
```

と指定した場合に图形の形に変化はあるか調べなさい(ヒント:stroke-widthを少し大きく指定すること)。

3.2.2 楕円の一部となる曲線

楕円の一部となる曲線は次のような値を与えて描きます。

- Mで開始点に移動します。すでに何らかのパスを指定した後ならば不要です。
- Aのあとに楕円のx軸の方向の長さとy軸方向の長さを指定します。
- その後に描くこの選択をするためのパラメーターを次の順序で指定します。
 1. 軸を傾ける角度
 2. 描く弧の指定。0の時は短いほうの弧(劣弧)1のときは長いほうの弧(優弧)を描きます。
 3. 描く向きの指定。0のときは反時計回り、1のときは時計回りです。²
- 最後に終了点の座標を指定します。

この弧の指定方法では描く弧の中心位置はシステムのほうで計算します。

図3.7は赤い円がはじめの点、青の点が終了点になっています。これらの円弧は<path>要素の属性でd="M50,0 A50,50 0 0 0 0,50"とすることで描くことができます。

²[2, p.135]の図をまねて図3.7を描きましたが、このパラメーターの指定が逆になっています。この図のほうが時計回り反時計回りの見た目とあっていますが、SVGではy座標の正の方向が下向きなので解釈では反対にも取れます。

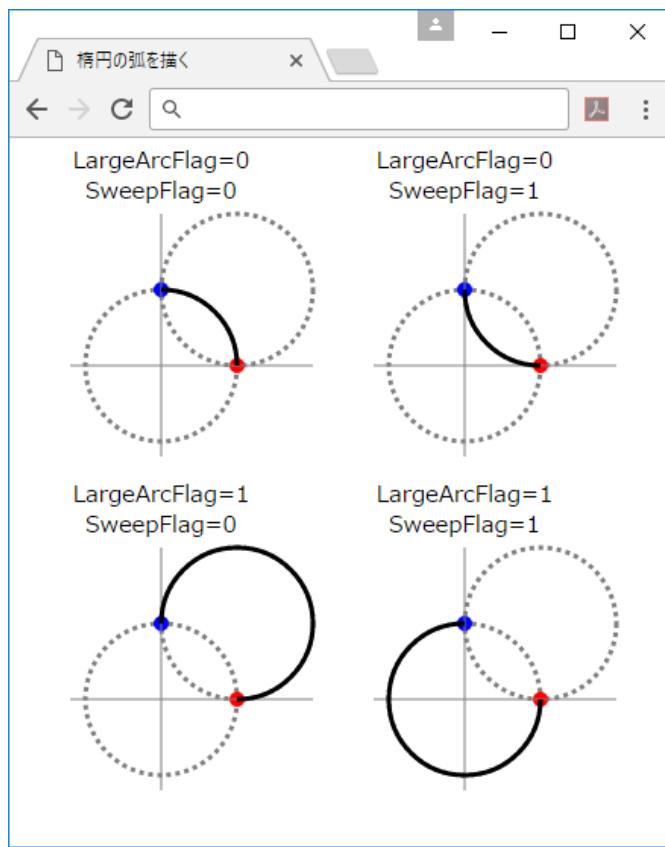


図 3.7: 楕円の弧を描く

SVG リスト 3.4: 楕円の弧を描く

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5      <title>椭円の弧を描く</title>
6      <defs>
7          <g id="Points" stroke="gray">
8              <circle cx="0" cy="0" r="50" fill="none"
9                  stroke-width="3" stroke-dasharray="3 3"/>
10             <circle cx="50" cy="-50" r="50" fill="none"
11                 stroke-width="3" stroke-dasharray="3 3"/>
12             <circle cx="50" cy="0" r="5" fill="red" stroke="none"/>
13             <circle cx="0" cy="-50" r="5" fill="blue" stroke="none"/>
14             <line x1="-60" y1="0" x2="100" y2="0" stroke-width="1"/>
15             <line x1="0" y1="-100" x2="0" y2="60" stroke-width="1"/>
16         </g>
17     </defs>
18     <g stroke-width="3" stroke="black" fill="none"
19         text-anchor="middle" font-size="15px">

```

```

20   <g transform="translate(100,150)">
21     <text y="-130" fill="black" stroke="none">LargeArcFlag=0</text>
22     <text y="-110" fill="black" stroke="none"> SweepFlag=0</text>
23     <use xlink:href="#Points"/>
24     <path d="M50,0 A50,50 0 0 0 0,-50"/>
25   </g>
26   <g transform="translate(300,150)">
27     <text y="-130" fill="black" stroke="none">LargeArcFlag=0</text>
28     <text y="-110" fill="black" stroke="none"> SweepFlag=1</text>
29     <use xlink:href="#Points"/>
30     <path d="M50,0 A50,50 0 0 1 0,-50" />
31   </g>
32   <g transform="translate(100,370)">
33     <text y="-130" fill="black" stroke="none">LargeArcFlag=1</text>
34     <text y="-110" fill="black" stroke="none"> SweepFlag=0</text>
35     <use xlink:href="#Points"/>
36     <path d="M50,0 A50,50 0 1 0 0,-50"/>
37   </g>
38   <g transform="translate(300,370)">
39     <text y="-130" fill="black" stroke="none">LargeArcFlag=1</text>
40     <text y="-110" fill="black" stroke="none"> SweepFlag=1</text>
41     <use xlink:href="#Points"/>
42     <path d="M50,0 A50,50 0 1 1 0,-50"/>
43   </g>
44   </g>
45 </svg>

```

(ここでは後で解説する文字列の表示も使用しています。また円弧を破線で描くために stroke-dasharray を用いています)。

図3.8はカニツツア錯視 [32, 88 ページ] とよばれるものです。三角形が描かれていないのに三角形があることが認知されるというものです。

SVGリスト3.5: カニツツアの主観的三角形

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3    xmlns:xlink="http://www.w3.org/1999/xlink"
4    height="400" width="400">
5    <title>カニツツアの主観的三角形</title>
6    <defs>
7      <g id="Base">
8        <g transform="translate(150,0),scale(1.1)">
9          <path d="M0,0 L-43.3,-25 A50,50 0 1 1 -43.3,25 z" fill="black"/>
10         </g>
11         <g transform="rotate(60),translate(130,0),scale(1.3)">
12           <polyline points="-43.3,-25 0,0 -43.3,25"
13             fill="none" stroke-width="2" stroke="black" />
14         </g>
15       </g>
16     </defs>
17     <g transform="translate(210,250)">
18       <g transform="rotate(30)">
19         <use xlink:href="#Base"/>

```

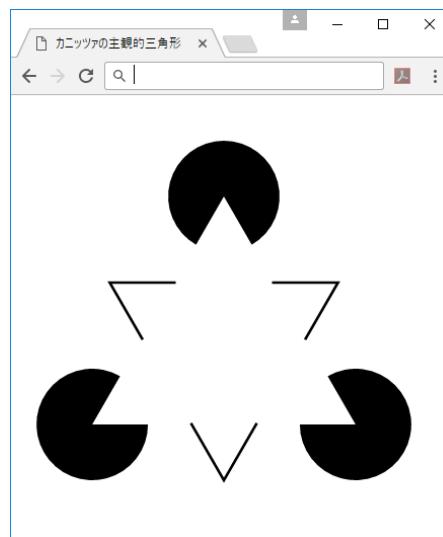


図 3.8: カニツツアの主観的三角形

```

20   </g>
21   <g transform="rotate(150)" >
22     <use xlink:href="#Base"/>
23   </g>
24   <g transform="rotate(270)">
25     <use xlink:href="#Base"/>
26   </g>
27 </g>
28 </svg>
```

- 円の一部が欠けている図形は円を描いた上に白で正三角形を描けば可能ですがここでは直接、図形を定義することにします。
- 円の一部が欠けている図形は9行目で定義しています。この図は原点を中心として x 軸の負の方向上下に 30° の方向に欠けるようにしました。
 - 出発点を原点 $(0,0)$ ($M0,0$) にとります。
 - $(50\cos(-120^\circ), 50\sin(-120^\circ))$ へ直線で移動 ($L-43.3, -25$) します。
 - $(50\cos(120^\circ), 50\sin(120^\circ))$ ($-43.3, 25$) まで時計回りに円弧 (優弧)($A50,50$ 0 の後の 1) を描きます。
 - パスを閉じます (z)。
- 30° 回転した方向に折れ線で正三角形の頂点近くの図を描きます (12行目から13行目)。
- この図形を平行移動してさらに回転して目的の位置へ配置しています (18行目から20行目、21行目から23行目と24行目から26行目)。

問題3.7 図3.9は主観的輪郭のネオン輝度現象[32, 232ページ]と呼ばれます。中央部の1/4の赤い円弧の間が薄く赤に塗られているように見えますが、実際は塗られていません。

この図を作成しなさい。

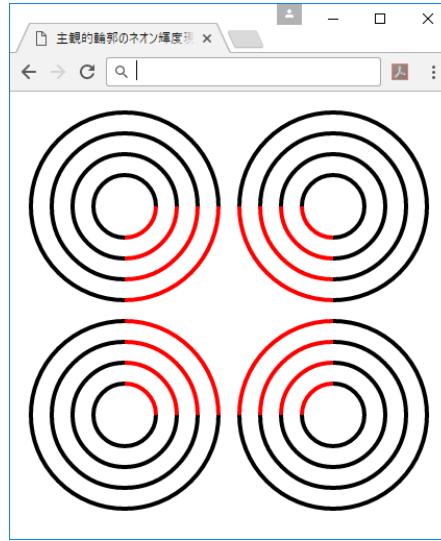


図3.9: 主観的輪郭のネオン輝度現象

3.2.3 Bézier曲線

Bézier曲線の歴史については[16, p.13–14]に解説があります。なお、[?, 28ページ]によるとBézierは自動車会社の技術者だったそうです。

3次のBézier曲線の定義

平面上に4点 $P_i(x_i, y_i)$, ($i = 0, 1, 2, 3$)をとります。この4点に対して $0 \leq t \leq 1$ の範囲で t を動かしたときにできる、次の式で定められる曲線を3次のBézier曲線といいます。³

$$x = (1-t)^3 x_0 + 3(1-t)^2 t x_1 + 3(1-t) t^2 x_2 + t^3 x_3 \quad (3.1)$$

$$y = (1-t)^3 y_0 + 3(1-t)^2 t y_1 + 3(1-t) t^2 y_2 + t^3 y_3 \quad (3.2)$$

この曲線は次の性質を持ちます。

1. $t = 0$ のときは点 P_0 , $t = 1$ のときは点 P_3 となります。
2. P_0 におけるこの曲線の接線は直線 P_0P_1 であり, P_0 におけるこの曲線の接線は直線 P_2P_3 です。

³ここではベクトルを用いて表していませんので x 座標と y 座標の値を別々に書いています。

3. $t = \frac{1}{2}$ における点は次のように作図して得られます。

(a) 点 P_0 と点 P_1 の中点を P_{01} , 点 P_1 と点 P_2 の中点を P_{12} , 点 P_2 と点 P_3 の中点を P_{23} とします。このとき、 P_{01}, P_{12}, P_{23} の x 座標はそれぞれ $\frac{1}{2}(x_0+x_1), \frac{1}{2}(x_1+x_2), \frac{1}{2}(x_2+x_3)$, となります。

(b) 点 P_{01} と点 P_{12} の中点を P_{012} , 点 P_{12} と点 P_{23} の中点を P_{123} とおきます。

$$\begin{aligned} P_{012} \text{の } x \text{ 座標} &= \frac{1}{2} \left(\frac{1}{2}(x_0+x_1) + \frac{1}{2}(x_1+x_2) \right) = \frac{1}{4}(x_0+2x_1+x_2) \\ P_{123} \text{の } x \text{ 座標} &= \frac{1}{2} \left(\frac{1}{2}(x_1+x_2) + \frac{1}{2}(x_2+x_3) \right) = \frac{1}{4}(x_1+2x_2+x_3) \end{aligned}$$

(c) 点 P_{012} と点 P_{123} の中点 P_{0123} が求めるものです。

$$P_{0123} \text{の } x \text{ 座標} = \frac{1}{2} \left(\frac{1}{4}(x_0+2x_1+x_2) + \frac{1}{4}(x_1+2x_2+x_3) \right) = \frac{1}{8}(x_0+3x_1+3x_2+x_3)$$

(d) このとき、4 点 $P_0, P_{01}, P_{012}, P_{0123}$ で定義される 3 次の Bézier 曲線と 4 点 $P_{0123}, P_{123}, P_{23}, P_3$ で定義される 3 次の Bézier 曲線はそれともとの Bézier 曲線の $0 \leq t \leq \frac{1}{2}$ の部分と $\frac{1}{2} \leq t \leq 1$ の部分に一致します。

4. 式 (3.1) と (3.2) を変数 t で微分すると

$$\begin{aligned} \frac{dx}{dt} &= -3(1-t)^2 x_0 + (-6(1-t)t + 3(1-t)^2)x_1 + (-3t^2 + 6(1-t)t)x_2 + 3t^2 x_3 \\ \frac{dy}{dt} &= -3(1-t)^2 y_0 + (-6(1-t)t + 3(1-t)^2)y_1 + (-3t^2 + 6(1-t)t)y_2 + 3t^2 y_3 \end{aligned}$$

となり、 $t = 0$ を代入すると x の式は $3(x_1 - x_0)$ 、 y の式は $3(y_1 - y_0)$ となります。微分の定義からベクトル $(x_1 - x_0, y_1 - y_0)$ は Bézier 曲線の点 P_0 における接線の方向であり、これは点 P_0 から P_1 へ向かう方向です。

5. 与えられた Bézier 曲線は 4 点 $P_i(x_i, y_i)$, ($i = 0, 1, 2, 3$) を頂点とする凸な四角形の内部 (周も含む) に含まれます。これは式 (3.1) の前の係数が $0 \leq t \leq 1$ に対して常に正であり、その和が

$$(1-t)^3 + 3(1-t)^2 t + 3(1-t)t^2 + t^3 = ((1-t)+t)^3 = 1$$

からそのような性質を持つことがわかります。

図 3.10 の左の図は $P_0(0,0), P_1(280,0), P_2(160,-320), P_3(0,160)$ としたときの上で解説した点の位置関係を示したもので、右の図は左の図での点 P_1 と P_2 の位置を取り替えたものです。このように P_1 と P_2 の位置によって Bézier 曲線は形を変えます。この 2 点を特に、この Bézier 曲線の制御点と呼びます。

図 3.10 の左の図を点の名称なしで SVG で書くと次のようにになります。⁴

⁴図 3.10 は PostScript というプログラミング言語で書きました。

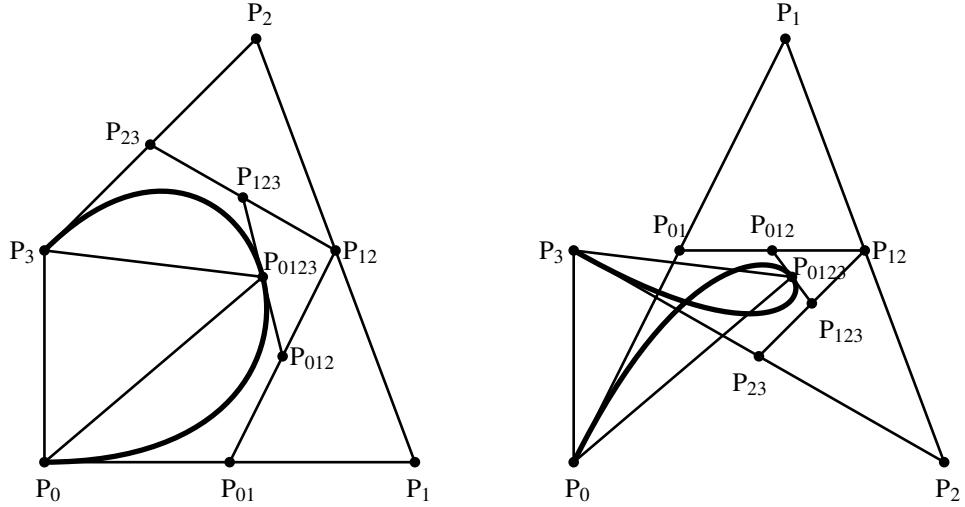


図 3.10: Bézier 曲線の解説

SVG リスト 3.6: Bézier 曲線の例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>Bezier 曲線の例</title>
6  <g transform="translate(150,350)">
7      <path stroke-width="4" stroke="black" fill="none"
8          d="M0,0 C280,0 160,-320 0,-160"/>
9      <circle cx="0" cy="0" r="6" fill="black"/> <!--P_0-->
10     <circle cx="280" cy="0" r="6" fill="black"/> <!--P_1-->
11     <circle cx="160" cy="-320" r="6" fill="black"/> <!--P_2-->
12     <circle cx="0" cy="-160" r="6" fill="black"/> <!--P_3-->
13     <polygon stroke-width="2" stroke="black" fill="none"
14         points="0,0 280,0 160,-320 0,-160"/>
15
16     <circle cx="140" cy="0" r="6" fill="black"/> <!--P_01-->
17     <circle cx="220" cy="-160" r="6" fill="black"/> <!--P_12-->
18     <circle cx="80" cy="-240" r="6" fill="black"/> <!--P_23-->
19     <polyline stroke-width="2" points="140,0 220,-160 80,-240"
20         stroke="black" fill="none"/>
21
22     <circle cx="180" cy="-80" r="6" fill="black"/> <!--P_012-->
23     <circle cx="150" cy="-200" r="6" fill="black"/> <!--P_123-->
24     <line x1="180" y1="-80" x2="150" y2="-200"
25         stroke-width="2" stroke="black" />
26     <circle cx="165" cy="-140" r="6" fill="black"/> <!--P_0123-->
27   </g>
28 </svg>
```

- Bézier 曲線は<path> 要素の d のなかで定義します(8行目)。4点は $P_0(0,0), P_1(280,0), P_2(160,-320), P_3(0,-160)$ となっています。
- P_0 の位置は M で定めその後に Bézier 曲線を定義を開始する C(cubic(3次) Bézier) を書き、残りの3点の位置を与えていきます。
- 9行目から12行目でこの4点の位置に小さな円を描いています。
- これらの点の中点の座標は次のようにになります(16行目から18行目)。

$$\begin{aligned}P_{01} &= \left(\frac{0+280}{2}, \frac{0+0}{2} \right) = (140, 0) \\P_{12} &= \left(\frac{280+160}{2}, \frac{0+(-320)}{2} \right) = (220, -160) \\P_{23} &= \left(\frac{160+0}{2}, \frac{(-320)+(-160)}{2} \right) = (80, -240)\end{aligned}$$

- さらに、これら3点の中点の座標は次のようにになります(22行目から23行目)。

$$\begin{aligned}P_{012} &= \left(\frac{140+220}{2}, \frac{0+(-160)}{2} \right) = (180, -80) \\P_{123} &= \left(\frac{220+80}{2}, \frac{(-160)+(-240)}{2} \right) = (150, -200)\end{aligned}$$

- さらにこれら2点の中点の座標は次のようにになります(26行目)。

$$P_{0123} = \left(\frac{180+150}{2}, \frac{(-80)+(-200)}{2} \right) = (165, -140)$$

このとき、次の関係が成立していることが確かめられます。

$$\begin{aligned}&\frac{1}{8}(P_0 + 3P_1 + 3P_2 + P_3) \\&= \left(\frac{1}{8}(0 + 3 \times 280 + 3 \times 160 + 0), \frac{1}{8}(0 + 3 \times 0 + 3 \times (-320) + (-160)) \right) \\&= (165, -140)\end{aligned}$$

したがって、この点は Bézier 曲線上にあることが確認できました。

なお、SVG の規約では Bézier 曲線の後に直線や曲線をつなぐことができます。Bézier 曲線を d に指定したときには最後の点 P_3 の位置から<path> 要素が引き継がれます。

二つの Bézier 曲線をつなぐとき、曲線を滑らかにつなぐためには最低限接線の方向を合わせる必要があります。接線の方向を一致させるためには計算が必要です。SVG では d の中で s を指定することでこの点を計算しなくてすむことが可能です。新しい Bézier 曲線の P_1 は前の Bézier 曲線の P_2 を前の Bézier 曲線の点 P_3 (=新しい Bézier 曲線の P_0) に関して対称な位置に移した点になります。

s の使用例は 55 ページの「円を Bézier 曲線で近似する例」の解説(リスト 3.7)を見てください。

問題 3.8 図 3.11 はカードのスーツの絵です。この図を描きなさい。

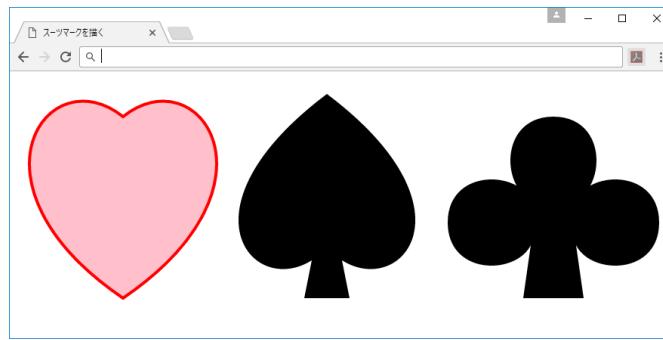


図 3.11: スースマークを描く

2次の Bézier 曲線

SVG では制御点がひとつである 2 次の Bézier 曲線を指定することができる `Q` も利用が可能です。接線を同一にするための `T` もあります。この曲線は次の式で与えられます ($0 \leq t \leq 1$ で考えることは 3 次の場合と同じです)。

$$\begin{aligned} x &= (1-t)^2 x_0 + 2(1-t)t x_1 + t^2 x_2 \\ y &= (1-t)^2 y_0 + 2(1-t)t y_1 + t^2 y_2 \end{aligned}$$

2 次の Bézier 曲線も 3 次の Bézier 曲線と同様の性質が成立します。

図 3.12 は 2 次と 3 次の Bézier 曲線の違いを表したものです。

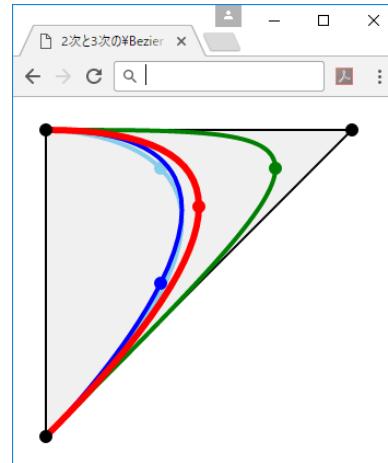


図 3.12: 2 次と 3 次の Bézier 曲線の違い

赤が $Q_0(0,0)$, $Q_1(280,0)$, $Q_2(0,280)$ とした 2 次の Bézier 曲線です。小さな赤い円は $t = \frac{1}{2}$ に対応する位置です。

残りの 3 つの曲線は制御点をそれぞれ次のように取った 3 次の Bézier 曲線です。小さな円はそれぞれの曲線の $t = \frac{1}{2}$ に対応する位置です。

色	P ₀	P ₁	P ₂	P ₃
緑	Q ₀	Q ₁	Q ₁	Q ₂
水色	Q ₀	Q ₀	Q ₁	Q ₂
青	Q ₀	Q ₁	Q ₂	Q ₂

2 次の Bézier 曲線は TrueType フォントの形状を記述する⁵ ために使われています。

円を Bézier 曲線で近似する

原点を中心とする半径 1 の円の第 1 象限の部分を Bézier 曲線で近似することを考えます。

円の対称性と接線の方向から $P_0(1, 0)$, $P_1(1, a)$, $P_2(a, 1)$, $P_3(0, 1)$ とおきます。 $t = \frac{1}{2}$ のとき、点 $\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ を通るように a を定めると式 (3.1) より

$$\frac{1}{\sqrt{2}} = \frac{1}{8} + 3 \times \frac{1}{8} + 3 \times \frac{1}{8}a$$

これより $a = \frac{4}{3}(\sqrt{2} - 1) \approx 0.55228\dots$ が得られます。

この基づいて円を書くと図 3.13 のようになります。なお、この図では<circle> 要素の図形の比較のために少し幅の広い円の上に Bézier 曲線を重ねて描いています。

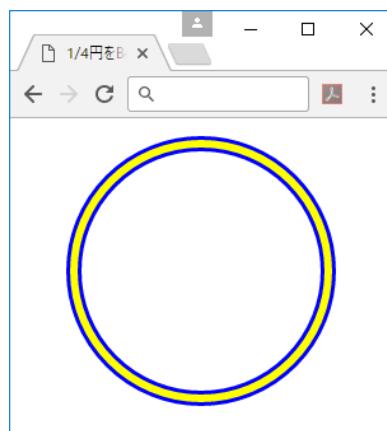


図 3.13: 1/4 円を Bézier 曲線で近似する

SVG リスト 3.7: 1/4 円を Bézier 曲線で近似する

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
```

⁵<http://www.microsoft.com/typography/otspec/TTCH01.htm>

```

3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5      <title>1/4 円を Bézier 曲線で近似する</title>
6      <g transform="translate(150,120)">
7          <circle cx="0" cy="0" r="100"
8              stroke-width="12" fill="none" stroke="blue"/>
9          <path d="M-100,0
10             C-100,-55.228 -55.228,-100 0,-100
11             S100,-55.228 100,0
12             S55.228,100 0,100
13             S-100,55.228 -100,0z"
14             stroke-width="6" fill="none" stroke="yellow"/>
15     </g>
16 </svg>
17

```

- SVG の<circle> 要素で書いたのと比較するために `stroke-width` を大きめにした円を描いています (7 行目から 8 行目)。
- 円を 4 つの 1/4 円をつなげて描きます。
 - まず M-100,0 で開始点へ移動します (9 行目)。
 - 次に C-100,-55.228 -55.228,-100 0,-100 で左上の部分の 1/4 円を描く Bézier 曲線の点の値を記述しています (9 行目)。
 - 値が上の解説と異なり負の値になっているのは *y* 座標の正の向きが下向きになっているからです。
 - 次に、右上の部分を書きます。はじめの制御点は前の Bézier 曲線と最後の点に関して対称な位置にいますので S を用いて記述するのが簡単です。ここでは 100,-55.228 100,0 となります。
 - 残りの部分も同様に S を用いて記述できます。
 - 最後に z をつけて道のりを閉じます (13 行目)。

図 3.7 をみるとよく近似されていることがわかります。[16, p.14] には 1/4 円を Bézier 曲線で近似すると 0.06%以下の精度で描くことができるとの記述があります。

問題 3.9 半円をひとつの 3 次の Bézier 曲線で近似しなさい。近似の度合いはどのようにになっているか確かめなさい。

3.3 transformについての補足

`transform` の値として平行移動(`translate`)、回転(`rotate`)と拡大縮小(`scale`)があることはすでに解説しました。ここでは今までに説明しなかったものに対して解説をします。

3.3.1 原点以外を中心とする回転

`rotate` は回転の角度だけではなく回転の中心位置を指定できます。`rotate(<回転角度>, <回転の中心の x 座標>, <回転の中心の y 座標>)` の形をとります。

図 3.14 はこの属性値を使った例です。y 座標軸上にある中抜きの丸が回転の中心を表します。また、薄い色の長方形が元の位置にあるものです。

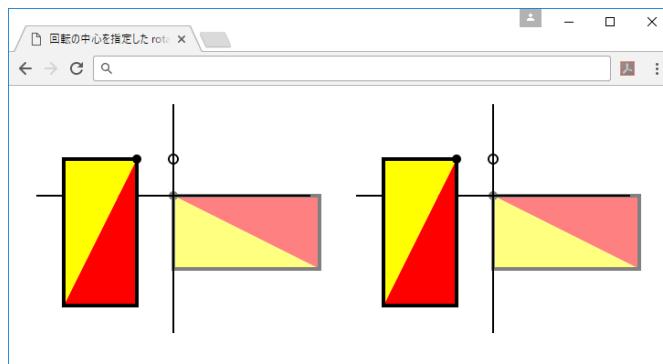


図 3.14: 回転の中心を指定した `rotate`

SVG リスト 3.8: 回転の中心を指定した `rotate`

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>回転の中心を指定した rotate</title>
6  <defs>
7      <g id="axis">
8          <path d="M-150,0 150,0 M 0,-100 0,150"
9              fill="none" stroke-width="2" stroke="black"/>
10         <circle cx="0" cy="-40" r="5" fill="none"
11             stroke-width="2" stroke="black"/>
12     </g>
13     <g id="fig">
14         <path d="M0,0 160,0 160,80" fill="red"/>
15         <path d="M0,0 0,80 160,80" fill="yellow"/>
16         <path d="M0,0 160,0 160,80 0,80z" fill="none"
17             stroke-width="4" stroke="black"/>
18         <circle cx="0" cy="0" r="5" fill="black"/>
19     </g>
20 </defs>
21 <g transform="translate(180,120)">
22     <use xlink:href="#axis"/>
23     <use xlink:href="#fig" opacity="0.5"/>
24     <g transform="translate(0,-40) rotate(90) translate(0,40)">
25         <use xlink:href="#fig"/>
26     </g>
27 </g>
28 <g transform="translate(530,120)">
```

```

29      <use xlink:href="#axis"/>
30      <use xlink:href="#fig" opacity="0.5"/>
31      <g transform="rotate(90,0,-40)">
32          <use xlink:href="#fig"/>
33      </g>
34  </g>
35 </svg>
```

- 8行目から9行目で x と y の座標軸をまとめて`<path>`要素を用いて定義しています。
- 10行目から11行目で回転の中心を表す円を定義しています。
- これらの二つの図形をまとめて`axis`と名付けています(8行目)。
- 13行目から19行目で回転する図形を定義しています。
 - 回転の向きを明確にするために三角形二つで長方形を作成しています(14行目と15行目)。
 - その上に塗りつぶしのない長方形を描いて(16行目から17行目)ひとつの図形に見せてています。
 - 左上の位置(元の図形では原点にある)に小さい円を付けています(18行目)。
- 21行目から27行目で原点を中心とする回転と平行移動を用いて中心を指定した回転を実現しています。これは次の3つの操作の組み合わせで実現できます。

回転の中心を原点に移動 ⇒ そこで回転 ⇒ 回転の中心位置を原点から元に戻す

- 28行目から34行目は回転の中心を指定した移動を指定しています。31行目に記述があります。
- 両者とも同じ位置に図形が移動していることを確認してください。

3.3.2 座標軸方向へのゆがみ

x 軸に垂直な直線を角度 α だけ傾ける変形をするのが`skewX(α)`です。同様に y 軸に垂直な直線を角度 α だけ傾ける変形をするのが`skewY(α)`です。

SVGの開始の座標系は y 軸が下向きになっているので変形する方向には注意してください。図3.15はどちらの図形も -45° 傾かせています。

SVGリスト3.9: 座標軸方向への歪み

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>座標軸方向への歪み</title>
6  <defs>
7      <g id="axis">
8          <path d="M-150,0 150,0 M 0,-100 0,150"
```

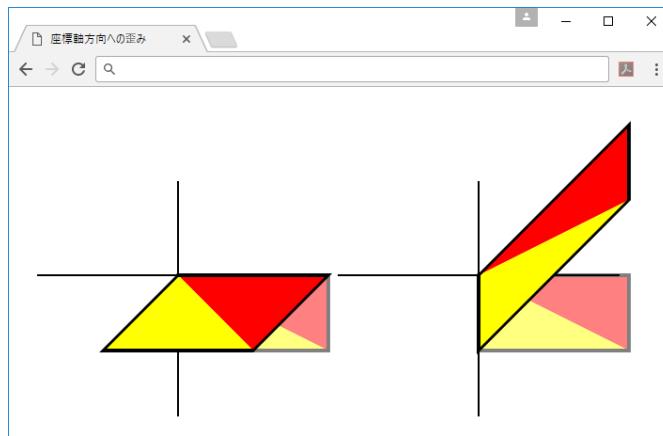


図 3.15: 座標軸方向への歪み

```

9      fill="none" stroke-width="2" stroke="black"/>
10     </g>
11     <g id="fig">
12       <path d="M0,0 160,0 160,80" fill="red" />
13       <path d="M0,0 0,80 160,80" fill="yellow" />
14       <path d="M0,0 160,0 160,80 0,80z" fill="none"
15         stroke-width="4" stroke="black"/>
16     </g>
17   </defs>
18   <g transform="translate(180,200)">
19     <use xlink:href="#axis"/>
20     <use xlink:href="#fig" opacity="0.5"/>
21     <g transform="skewX(-45)">
22       <use xlink:href="#fig"/>
23     </g>
24   </g>
25   <g transform="translate(500,200)">
26     <use xlink:href="#axis"/>
27     <use xlink:href="#fig" opacity="0.5"/>
28     <g transform="skewY(-45)">
29       <use xlink:href="#fig"/>
30     </g>
31   </g>
32 </svg>
```

- 8 行目から9 行目で座標軸を定義しています。
- 11 行目から16 行目で基準となる図形を定義しています。
- 18 行目から24 行目で左側の図形を定義しています。
- 21 行目で -45° だけ x 軸方向に傾かせています。角度が負なので左方向に傾きます。

- 25行目から31行目で右側の図形を定義しています。
- 28行目で -45° だけ x 軸方向に傾かせています。角度が負なので上方向に傾きます。

3.3.3 一般的な線形変換

今までに説明してきた `transform` をすべて含むものが `matrix(a,b,c,d,e,f)` です。この変換は現在の位置 $(x_{\text{now}}, y_{\text{now}})$ を次の式で計算して新しい位置 $(x_{\text{new}}, y_{\text{new}})$ にします⁶。ここでの値はブラウザの左上を原点とする位置と考えます。

$$\begin{cases} x_{\text{new}} &= ax_{\text{now}} + cy_{\text{now}} + e \\ y_{\text{new}} &= bx_{\text{now}} + dy_{\text{now}} + f \end{cases} \quad (3.3)$$

行列で表すと次のようにになります。

$$\begin{pmatrix} x_{\text{new}} \\ y_{\text{new}} \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \end{pmatrix} \begin{pmatrix} x_{\text{now}} \\ y_{\text{now}} \\ 1 \end{pmatrix} \quad (3.4)$$

または、

$$\begin{pmatrix} x_{\text{new}} \\ y_{\text{new}} \\ 1 \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{\text{now}} \\ y_{\text{now}} \\ 1 \end{pmatrix} \quad (3.5)$$

と表されます。この表現では変換行列が正方行列になり、点の位置を表す座標の形も両辺で同じ形になるので式の扱いが簡単になります⁷。なお、この行列と点の位置のあらわし方は日本の線形代数学の慣例に従っています。アメリカなどのCGの教科書では点の位置を横ベクトルで表すようです。その記法に従うと式(3.5)は次のようにになります。

$$(x_{\text{new}}, y_{\text{new}}, 1) = (x_{\text{now}}, y_{\text{now}}, 1) \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{pmatrix} \quad (3.6)$$

`translate(a,b)` の場合 図形を x 方向に a 、 y 方向に b 移動するので次の式で表されます。

$$\begin{cases} x_{\text{new}} &= x_{\text{now}} + a \\ y_{\text{new}} &= y_{\text{now}} + b \end{cases}$$

`rotate(α)` の場合

$$\begin{cases} x_{\text{new}} &= \cos \alpha x_{\text{now}} - \sin \alpha y_{\text{now}} \\ y_{\text{new}} &= \sin \alpha x_{\text{now}} + \cos \alpha y_{\text{now}} \end{cases}$$

⁶係数のアルファベットの順序がおかしいと思うかもしれません。がこの順序はSVGの仕様書内で使われているものと同じです。

⁷より正確には同次座標系の特別な場合です。この形式を用いるとCGの基本である射影変換を取り扱うことができます。

`scale(a,b)` の場合 図形を x 方向に a 倍、 y 方向に b 倍するので次の式で表されます。

$$\begin{cases} x_{\text{new}} &= ax_{\text{now}} \\ y_{\text{new}} &= by_{\text{now}} \end{cases}$$

`skewX(α)`、`skewY(α)` の場合 `skewX(α)` は図形を x 方向に角度 α だけ傾けるので

$$\begin{cases} x_{\text{new}} &= x_{\text{now}} + \tan \alpha y_{\text{now}} \\ y_{\text{new}} &= y_{\text{now}} \end{cases}$$

となります。同様に `skewY(α)` は次の式で表されます。

$$\begin{cases} x_{\text{new}} &= x_{\text{now}} \\ y_{\text{new}} &= \tan \alpha x_{\text{now}} + y_{\text{now}} \end{cases}$$

問題 3.10 直線 $y = x$ や $y = -x$ に関して図形を対称移動するためにはどのような変換を指定すればよいか。

3.4 SVG パターン

長方形などの内部を塗るための `fill` には繰り返しのパターンを指定することができます。

ヘルマン格子(図 2.13)をパターンを利用して描くと次のようになります。

SVG リスト 3.10: ヘルマン格子(パターンで描く)

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="330" width="330">
5      <title>ヘルマン格子(パターンで描く)</title>
6      <defs>
7          <pattern id="Hermann" width="50" height="50"
8              patternUnits="userSpaceOnUse">
9              <rect x="0" y="0" width="50" height="50"
10                 stroke-width="8" stroke="white" fill="black"/>
11          </pattern>
12      </defs>
13      <g transform="translate(20,20)">
14          <rect x="0" y="0" width="300" height="300" fill="url(#Hermann)" />
15      </g>
16  </svg>
```

- 内部が黒で、縁取りを白で塗る正方形を敷き詰める形でこの図形を描きます。
- 基本となる図形は`<pattern>`要素で定義します(7行目から11行目)。`<pattern>`要素はグラデーションのときと同じように`<defs>`要素内に記述します。
 - `<pattern>`要素では後で参照するための属性 `id` とパターンの大きさを `width` と `height` で指定します(7行目)。

- 8行目で<pattern>要素を塗る基準の座標系を属性 patternUnits を用いて指定しています。グラデーションのときと同様に objectBoundingBox と userSpaceOnUse が指定できます。
- <pattern>要素内の図形は大きさが50の正方形で縁取りの幅を8にした正方形です(9行目から9行目)。
- このパターンで内部を塗る長方形は14行目で定義されています。fillに<pattern>要素の属性 id の値を指定します。この場合には url(#Hermann) と url() を付けます。

問題 3.11 図 3.16 はザヴィニーの錯視 [37, 132 ページ] とよばれます。両方のパターンを囲む正方形の大きさは同じなのですがパターンの向きにより異なった方向のほうが長い長方形に見えます。パターンには<path>要素を用いると良いでしょう。

この図を描きなさい。

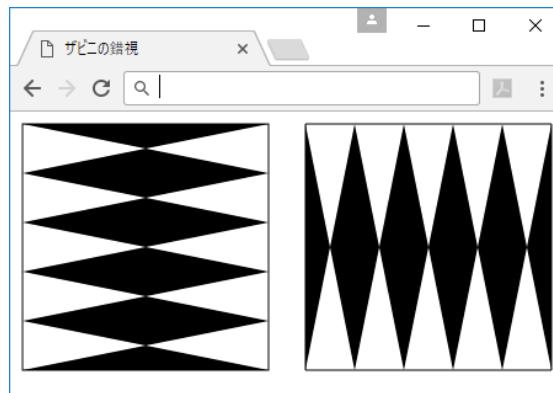


図 3.16: ザビニの錯視

パターン内の図形としてはいくつかの図形を組み合わせてもかまいません。図 3.17 は別のグラデーションで塗った複数の正方形を組み合わせてパターンを構成しています。

SVG リスト 3.11: 二つの線形グラデーションを市松模様に並べた例

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      width="100%" height="100%">
5  <title>二つの線形グラデーションを市松模様に並べた例</title>
6  <defs>
7      <linearGradient id="LinGrad1" x1="0%" y1="0%" x2="0" y2="100%">
8          gradientUnits="objectBoundingBox" >
9          <stop offset="0%" stop-color="#FF0000"/>
10         <stop offset="100%" stop-color="#600000"/>
11     </linearGradient>
12     <linearGradient id="LinGrad2" x1="0%" y1="0%" x2="100%" y2="0%">
13         gradientUnits="objectBoundingBox" >

```

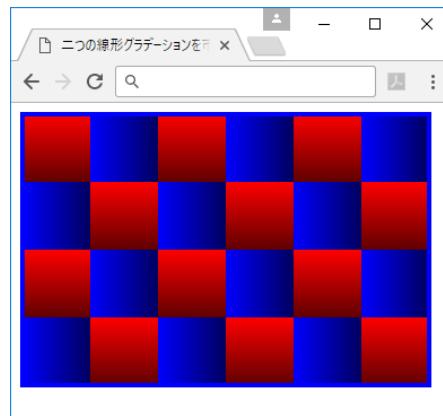


図 3.17: 二つの線形グラデーションを市松模様に並べた例

```

14      <stop offset="0%" stop-color="#0000FF"/>
15      <stop offset="100%" stop-color="#000060"/>
16  </linearGradient>
17  <rect id="Rect1" fill="url(#LinGrad1)" width="60" height="60"/>
18  <rect id="Rect2" fill="url(#LinGrad2)" width="60" height="60"/>
19  <pattern id="checkerPattern" width="120" height="120"
20      patternUnits="userSpaceOnUse">
21      <use xlink:href="#Rect1" x="0" y="0"/>
22      <use xlink:href="#Rect2" x="60" y="0"/>
23      <use xlink:href="#Rect2" x="0" y="60"/>
24      <use xlink:href="#Rect1" x="60" y="60"/>
25  </pattern>
26 </defs>
27 <g transform="translate(10,10)">
28   <rect x="0" y="0" width="360" height="240"
29     fill="url(#checkerPattern)" stroke-width="4" stroke="blue"/>
30   </g>
31 </svg>
```

- 7 行目から 11 行目と 12 行目から 16 行目で 2 つの線形グラデーションを定義しています。
 - 7 行目から 11 行目は上から下へ向かう線形グラデーションを定義しています。
 - 12 行目から 16 行目は左から右へ向かう線形グラデーションを定義しています。
- 辺の長さが 60 の正方形を二つこれらのグラデーションを使って塗ります。
- この二つの正方形を市松模様に並べたパターンを 19 行目から 25 行目で定義します。
 - パターンの大きさは正方形を縦横二つずつ並べたものなので大きさは $2 \times 60 = 120$ です。したがって、width と height の値はそれぞれ 120 です (19 行目)
 - グラデーションで塗られた 2 種類の正方形を <use> 要素を用いて対角線上に並べます (21 行目から 24 行目)。

- <use>要素のなかに配置する位置xやyを指定しています。
- 28行目から始まる長方形をこのパターンで塗りつぶします。ひとつのパターンの大きさが 120×120 なのでこのパターンが横に3回、縦に2回繰り返されます。

問題3.12 図3.18はないはずの点がよりはっきり見えるようになる輝くヘルマン格子 [32, 180ページ]です。この図では正方形の間の隙間は少し明るめの灰色で、交差部分には白で塗られた円を描いています。これもパターンを用いてこの図を作成しなさい。

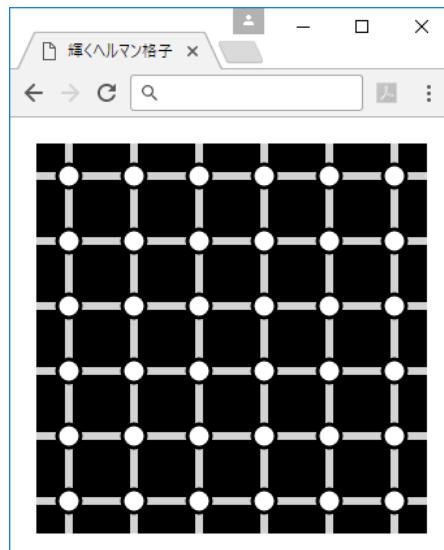


図3.18: 輝くヘルマン格子

問題3.13 カフェウォール錯視(図2.14)をパターンを用いて作図しなさい。

問題3.14 図3.19はモーガンのねじれひも [32, 76ページ]とよばれる錯視图形です。これを作成しなさい。

パターンを塗りつぶす图形は長方形でなくてもかまいません。

図3.20は大内元によって作成された錯視图形を内部のパターンを簡略化した图形です ([37, 74ページ図7.5])。⁸中央の円がページの画面から浮き上がってゆらゆらします。片目で見ると立体感が増すように見えます。

これは、円をパターンで塗りつぶしています。

SVGリスト3.12: 浮動する円

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
```

⁸[17, 75ページ]も参照のこと。この本にはほかにも面白い錯視图形が載っています。

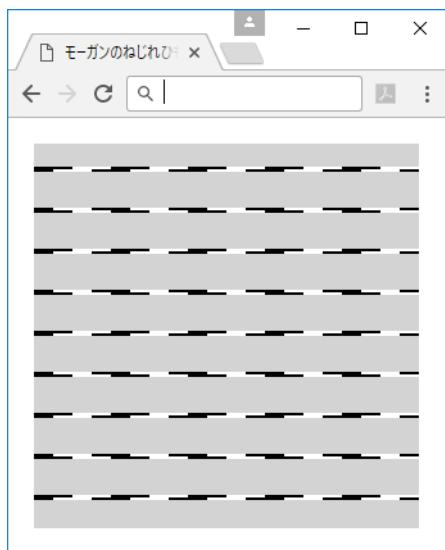


図 3.19: モーガンのねじれひも

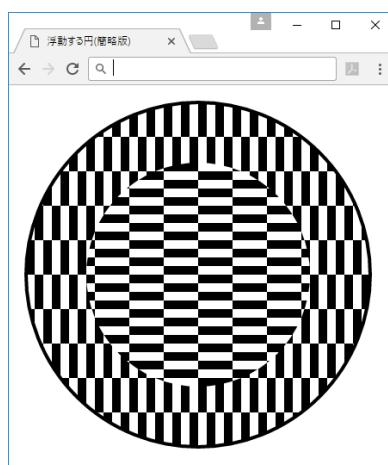


図 3.20: 浮動する円(簡略版)

```
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%" >
5  <title>浮動する円(簡略版)</title>
6  <defs>
7    <pattern id="P" width="20" height="80" patternUnits="userSpaceOnUse">
8      <rect x="0" y="0" height="100%" width="100%" fill="white"/>
9      <path d="M0,0 110,0 10,80 110,0 10,-40 1-20,0z" fill="black"/>
10     </pattern>
11   </defs>
12   <g transform="translate(20,20)">
```

```

13   <circle cx="200" cy="200" r="200" fill="url(#P)"
14     stroke="black" stroke-width="4"/>
15   <g transform="rotate(90,200,200)">
16     <circle cx="200" cy="200" r="130" fill="url(#P)"/>
17   </g>
18 </g>
19 </svg>
```

- 7行目から10行目でパターンを定義しています。
- パターンは領域全体を白で塗りつぶされた長方形(8行目)とその上に対角線上に二つ並んだ黒く塗りつぶされた長方形(9行目)からなります。
- 13行目から14行目で外側の円をこのパターンで塗りつぶしています。
- その上に小さな円を同じパターンで塗りつぶし(16行目)、さらに(200,200)を中心に90°回転(15行目)したものを描いています。

問題 3.15 図 3.1 の内部をパターンで塗りつぶしなさい。パターンの開始がどこから始まっているか調べること。

なお、`<pattern>`要素の属性にはパターンの配置を変形させる `patternTransform` があります。この値は図形を移動させる属性 `transform` の値と同じものが書けます。

問題 3.16 図 3.21 は図 3.17 のグラデーションのパターンを回転させた図形を内部に持つ長方形です。この図は`<pattern>`要素に属性 `patternTransform` を用いて作成できます。この図を作成しなさい。

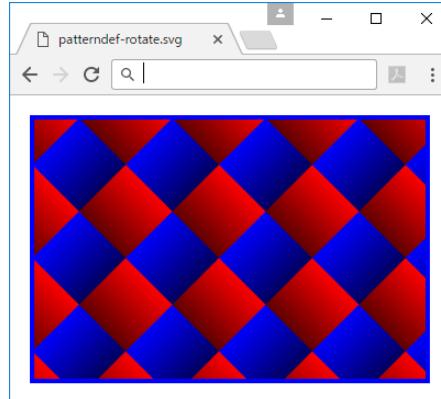


図 3.21: 図 3.17 のグラデーションのパターンを回転させる

3.5 画像の取り込み

<image> 要素を用いると外部の画像ファイルを読み込むことができます。取り込める画像の種類は JPG や PNG があります。

図 3.22 は JPG 形式の同一の画像ファイルを属性値を変えて取り込んだものです。

この SVG 文書は<image> 要素で指定した画像 (大きさは 1800 × 1800) を取り込む範囲と属性値との関係を示すためにその範囲と同じ大きさの長方形を描いています。

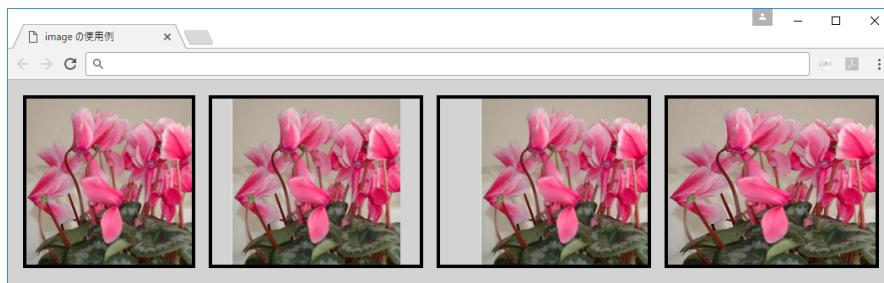


図 3.22: <image> 要素の使用例

SVG リスト 3.13: <image> 要素の使用例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5      <title>image の使用例</title>
6      <rect x="0" y="0" width="100%" height="100%" fill="lightgray"/>
7      <image xlink:href="cyclamen-2.jpg" width="200" height="200" x="20" y="20"/>
8      <rect x="20" y="20" width="200" height="200" fill="none"
9          stroke-width="4" stroke="black"/>
10
11     <image xlink:href="cyclamen-2.jpg" width="250" height="200" x="240" y="20"/>
12     <rect x="240" y="20" width="250" height="200" fill="none"
13         stroke-width="4" stroke="black"/>
14
15     <image xlink:href="cyclamen-2.jpg" width="250" height="200" x="510" y="20"
16         preserveAspectRatio="xMaxYMid"/>
17     <rect x="510" y="20" width="250" height="200" fill="none"
18         stroke-width="4" stroke="black"/>
19
20     <image xlink:href="cyclamen-2.jpg" width="250" height="200" x="780" y="20"
21         preserveAspectRatio="none"/>
22     <rect x="780" y="20" width="250" height="200" fill="none"
23         stroke-width="4" stroke="black"/>
24  </svg>
```

- 6 行目で表示画面全体を明るい灰色に塗りつぶしています。これは画像が表示されなかった場所がどのように扱われるかを示すためです。

- 7行目で cyclamen-2.JPGを取り込んでいます。
 - 取り込む範囲は縦横ともに200ピクセルの正方形の領域です。
 - 画像はその範囲に縮小されて表示されています。元の画像が正方形で表示する範囲も正方形なので9行目で示した正方形の枠内にちょうど収まります。
- 11行目は表示領域の大きさを横250、縦200に変えて同じ図形を表示しています。
 - この画像の画像の収縮比は縦と横が同じであることがわかります。
 - 表示位置も横について画面の中央にあることがわかります。
 - 画像が表示されない部分は下の画像が表示されています。
- 表示領域に対して画面の表示位置を移動させるためには `preserveAspectRatio` を用います。
 - 16行目ではその値を `xMaxYMid` としています。`xMax` の部分で水平方向の位置を一番右(`x`方向の最大値)に、`YMid`で縦方向は上下の中央にすることを意味します。
 - 指定できる値は水平方向が `xMax`、`xMid` と `xMin` の3種類、垂直方向が `yMax`、`YMid`、`yMin` の3種類あるので組み合わせて合計9種類あることになります(`Y`が大文字になっていることに注意してください)。
 - このほかに `none` という値も指定できます(21行目)。この場合は縦横比(aspect ratio)を指定しないという意味になり、表示領域いっぱいに画像が表示されます。ここでは横方向に拡大されています。

なお、SVG内部では画像はすべてカラー画像(+不透明度)に変換されて保存されます。

問題 3.17 `preserveAspectRatio` のとりうる値をすべて記しなさい。

問題 3.18 `preserveAspectRatio` の値をいくつか変えて画像を表示させなさい。

図3.23は画像を`<pattern>`要素で使用しています。

SVGリスト3.14: 画像を`<pattern>`要素で使用する

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3    xmlns:xlink="http://www.w3.org/1999/xlink"
4    height="100%" width="100%">
5  <title>画像を pattern で使用する</title>
6  <defs>
7    <pattern id="image" width="200" height="200" patternUnits="userSpaceOnUse">
8      <image xlink:href="cyclamen-2.jpg" width="200" height="200"/>
9    </pattern>
10   </defs>
11   <rect x="0" y="0" width="1000" height="600" fill="url(#image)"/>
12 </svg>

```

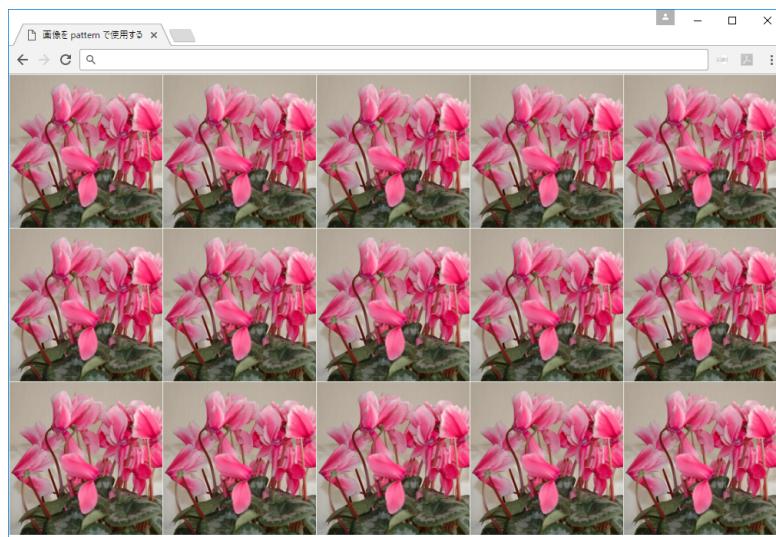


図 3.23: 画像を<pattern>要素で使用する

- 7行目から9行目でパターンを定義しています。パターンの大きさは縦横 200 の正方形です。
- 8行目でパターンに使用する画像を引用しています。この画像を取り込む大きさも縦横 200 の正方形です。
- 11行目で内部をこのパターンで塗るように指定した長方形を定義しています。

3.6 画像の一部を見せる

図形の内部だけに画像を表示させることを行うには<mask>要素を用います。

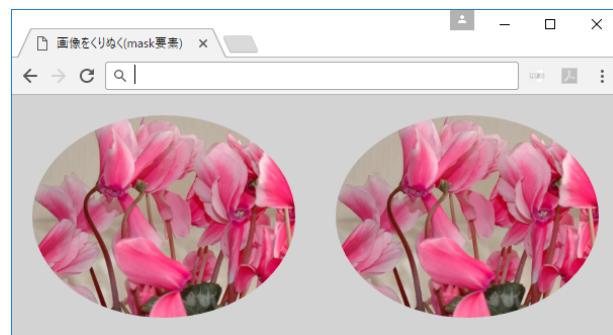


図 3.24: 画像をくりぬく(<mask>要素)

SVGリスト3.15: 画像をくりぬく(<mask>要素)

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>画像をくりぬく (mask要素)</title>
6  <defs>
7      <mask id="mask1" maskUnits="userSpaceOnUse"
8          x="0" y="0" width="300" height="300">
9          <ellipse cx="150" cy="120" rx="130" ry="100" fill="white"/>
10     </mask>
11     <image id="image" xlink:href="cyclamen-2.jpg" width="300" height="300"/>
12     <pattern id="pattern" width="300" height="300"
13         patternUnits="userSpaceOnUse" >
14         <use xlink:href="#image"/>
15     </pattern>
16 </defs>
17     <rect x="0" y="0" width="100%" height="100%" fill="lightgray"/>
18     <image x="0" y="0" xlink:href="cyclamen-2.jpg" width="300" height="300"
19         mask="url(#mask1)"/>
20     <ellipse cx="450" cy="120" rx="130" ry="100" fill="url(#pattern)"/>
21 </svg>

```

- ここでは<mask>要素を用いる方法と図形の属性fillで行う方法を比較します。
- <mask>要素は7行目から8行目で定義されています。
 - ここでは後で引用するためにidとしてmask1を与えています。
 - maskUnitsで対象となる座標系としてuserSpaceOnUseを指定しています⁹。
 - 左上の座標位置をxとyで指定し、大きさをwidthとheightで指定します。
 - <mask>要素は内部に含む図形の各点の明るさを不透明度(一般にはアルファ値)に変換した図形に変換し、引用された図形の上にかぶせます。色がwhiteのときが不透明度が0に、blackのときには不透明度が1に設定されます。
 - ここでは橜円の内部をwhiteに塗っているのでこの部分だけ表示されることになります。
- 18行目から19行目でこの<mask>要素を使って画像を表示しています。
- 11行目では<pattern>要素で利用する図形を定義しています。
- 12行目から15行目で20行目で定義された橜円の内部を塗るためにパターンを定義しています。<image>要素をfillでの参照要素に指定できないのでパターンを利用しています。
- 14行目ではパターンに利用する画像を引用しています。

問題3.19 図3.25は<mask>要素のなかにある橜円を放射グラデーションで塗りつぶしたものを利用しています。適当な画像を使用してこれと同じような図形を作成しなさい。

⁹objectBoundingBoxも定義できるのですが表示をうまくコントロールできませんでした。

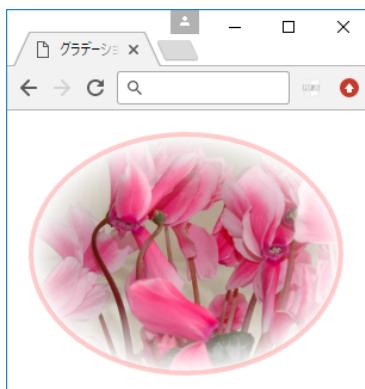


図 3.25: グラデーションを使用した<mask>要素

問題 3.20 図 3.1 を<mask>要素を使用して作成しなさい。

図 3.26 では画像を<mask>要素として利用しています。

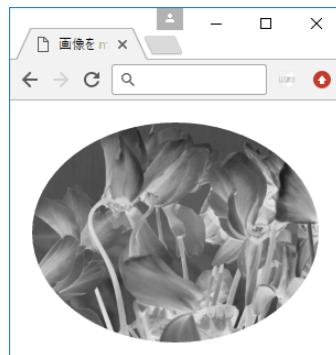


図 3.26: 画像を<mask>要素につかう

楕円を黒で塗りつぶしているので明るさが逆になるモノクロのネガ画像が得られます。

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink" height="100%" width="100%">
4      <title>画像を mask 要素につかう</title>
5      <defs>
6          <mask id="mask1" maskUnits="userSpaceOnUse"
7              x="0" y="0" width="300" height="300">
8              <image id="image" xlink:href="cyclamen-2.JPG" width="300" height="300"/>
9          </mask>
10     </defs>
```

```
11 <ellipse cx="150" cy="120" rx="130" ry="100" fill="black" mask="url(#mask1)"/>
12 </svg>
```

問題 3.21 図 3.27 は二つの画像を縦に細かく分けて交互に表示したものです。どちらかの画像がないと隠れている部分があっても何の画像かわかります。

適当な画像を二つ用意してこの図を作成しなさい。

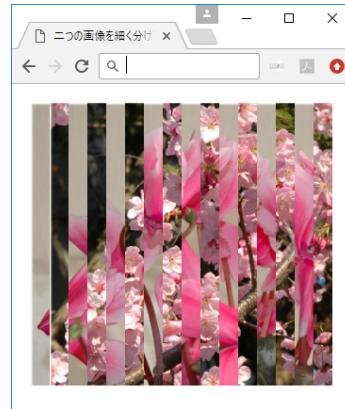


図 3.27: 二つの画像を細く分けて互い違いに並べて表示する

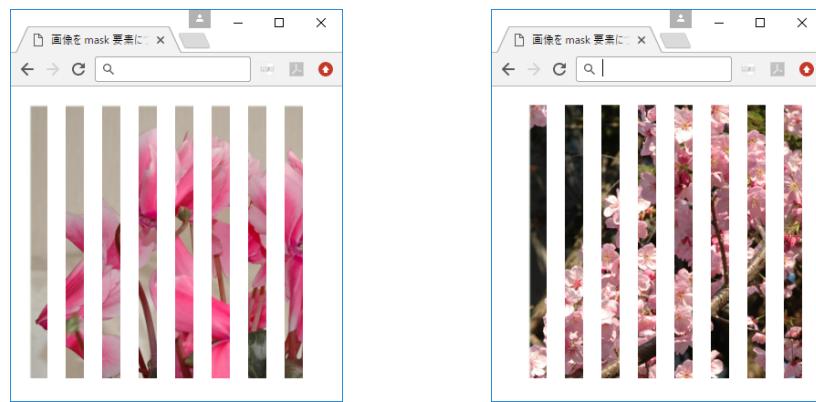


図 3.28: 画像を<mask>要素につかう (元画像)

第4章 アニメーション

4.1 アニメーションにおける属性

SVG では指定したオブジェクトの属性の値を時間の経過とともに変化させるアニメーションの機能があります。アニメーションではオブジェクトの属性値以外に開始の時間と終了の時間、開始時と終了時の値、終了時の状態、繰り返しの回数などを指定する必要があります(表 4.1 参照)。

表 4.1: アニメーションに共通の属性

属性名	意味	とりうる値
attributeName	属性名	属性名なら何でも可
attributeType	属性値の種類	XML または CSS
from	開始時の属性の値	
to	終了時の属性の値	
dur	変化の継続時間	2s(2 秒), 1m(1 分)
begin	開始時間	時間を与える。例 2s(2 秒), 1m(1 分)
fill	終了時の属性値の指定	freeze(終了値で固定) remove(はじめの値に戻る)
repeatCount	繰り返し回数	indefinite は無限回の繰り返し
values	属性値を複数指定	セミコロンで区切って値を設定
keyTimes	アニメーション全体の時間の割合で values で指定した値に順次変化	0 から 1 の間の値をセミコロンで区切る
calcMode	補間方法の指定	discrete 値が不連続に変わる。 linear 値を一次式で補間 (<animateMotion> 要素以外でデフォルト) paced アニメーション中一定の割合で変化する (<animatMotion> 要素 のデフォルト) spline 3 次の Bézier で値を補間

4.2 位置を動かす(<animateTransform>要素)

平行移動 グループ化されたオブジェクト<g>要素は属性 transform で位置の移動ができました。<animateTransform>要素を用いると transform に対してアニメーションがつきます。次の例は二つの長方形をグループ化し、それに平行移動のアニメーションをつけています。

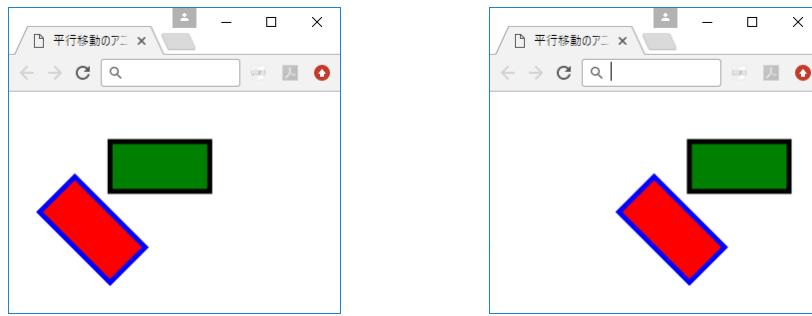


図 4.1: 平行移動のアニメーション(開始時-左-と終了時-右-)

SVG リスト 4.1: 図形の平行移動のアニメーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%" >
5  <title>平行移動のアニメーション</title>
6  <g transform="translate(100,50)" >
7      <rect x="0" y="0" width="100" height="50"
8          stroke-width="5" stroke="black" fill="green"/>
9      <g transform="rotate(45)">
10         <rect x="0" y="50" width="100" height="50"
11             stroke-width="5" stroke="blue" fill="red"/>
12     </g>
13     <animateTransform attributeName="transform" attributeType="XML"
14         type="translate" from="100,50" to="200,50" dur="10s" fill="freeze"/>
15   </g>
16 </svg>
```

- 6 行目から 15 行目で定義されているグループに平行移動のアニメーションをつけています。
- このグループには 7 行目から 8 行目にある長方形と、10 行目から 11 行目にある長方形を 45° 回転したもの(9 行目で定義)の二つの図形が含まれています。
- 13 行目から 14 行目にかけて<animateTransform>要素を用いて平行移動のアニメーションを定義しています。このアニメーションでは次の属性を与えています。
 - attributeName はアニメーションをさせる属性を定義します。ここでは transform の値が与えられています。

- transformには3種類のタイプがあるのでそれを指定するためにtypeを用います。この値は平行移動の場合translateとなります。
- 図形の平行移動ではtranslate(100,100)のように指定しますが、アニメーションの開始位置や終了位置を指定するfromやtoでは100,100のように括弧をつけません。
- ここでは(100,100)(fromでの値)から(200,100)(toの値)へ一定の速度で移動します。
- アニメーションの継続時間は属性durで指定します。ここでは10sなので10秒(s)間で上記の移動が行われます。時間の単位としてはこのほかにm(分)などもあります。
- アニメーションが終わったときにの状態はfillで与えます。はじめの状態に戻る(remove)と終了状態のままでいる(freeze)を指定できます。

このほかにアニメーションの属性としては繰り返しを指定するrepeatCountがあります。指定した回数だけアニメーションを繰り返すことができます。

なお、長方形ひとつだけを平行移動させるのであればxやyにアニメーションを付ける方法もあります。4.4.1を参照してください。

問題4.1 フックの錯視(図2.8)において垂直な線分が水平な線分の左端から右端へ移動するアニメーションをつけなさい。そのとき、見え方がどのように変化するかを調べなさい。

回転 次の例は例4.1における傾いた長方形に回転のアニメーションをつけたものです。

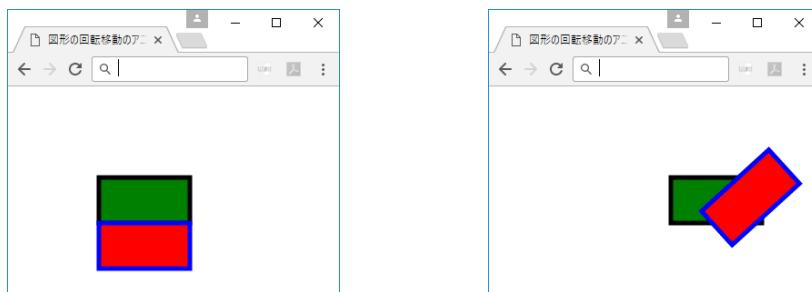


図4.2: 図形の回転のアニメーション – 開始時(左)と平行移動終了時(右)

SVGリスト4.2: 図形の回転のアニメーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%" >
5      <title>図形の回転移動のアニメーション</title>
6      <g transform="translate(100,100)" >
7          <rect x="0" y="0" width="100" height="50"
8              stroke-width="5" stroke="black" fill="green"/>
9          <rect x="0" y="50" width="100" height="50"
10             stroke-width="5" stroke="blue" fill="red">
11             <animateTransform attributeName="transform" attributeType="XML"
12               type="rotate" from="0" to="360" dur="10s" repeatCount="indefinite"/>

```

```

13   </rect>
14   <animateTransform attributeName="transform" attributeType="XML"
15     type="translate" from="100,100" to="200,100" dur="10s" fill="freeze"/>
16   </g>
17 </svg>

```

- 前のリスト 4.1 の 9 行目から 12 行目の部分がこの例で 9 行目から 13 行目に変わっています。
- まず、長方形<rect>要素にアニメーションをつけるのでこの要素の開始要素と終了要素が分かれています。10 行目の最後が>となり、13 行目に長方形の終了要素</rect>があります。
- 回転のアニメーションは type が rotate となり、開始が 0° で終了が 360° となっています。アニメーションを停止させないために repeatCount に indefinite を指定します(12 行目)。

この例では回転している長方形のアニメーションはそれを含む図形が始めの 10 秒間平行移動しているので、この間は回転と平行移動が同時に起きます。平行移動は 10 秒後に停止しますが、回転のアニメーションは動き続けます。

問題 4.2 ミューラー・ライヤーの錯視(図 2.9)の両端にある矢印に反対向きの回転のアニメーションをつけ、見え方の変化を観察しなさい。

問題 4.3 図 4.3 はジャッドの錯視 [32, 66 ページ] という図形です。線分の中央にある円が左に偏った場所にあるように見えます。この図を作成し、さらに両端の矢印の間の角度が開く回転のアニメーションをつけ、見え方の変化を観察しなさい。



図 4.3: ジャッドの錯視

拡大縮小 図形の拡大縮小する transform の属性値 scale にアニメーションを付ける例が図 4.4 です。さらに translate にアニメーションを付けることで水平線と垂直線に円は接したまま大きさを変えます。

SVG リスト 4.3: 拡大縮小と移動のアニメーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3    xmlns:xlink="http://www.w3.org/1999/xlink"
4    height="100%" width="100%" >

```

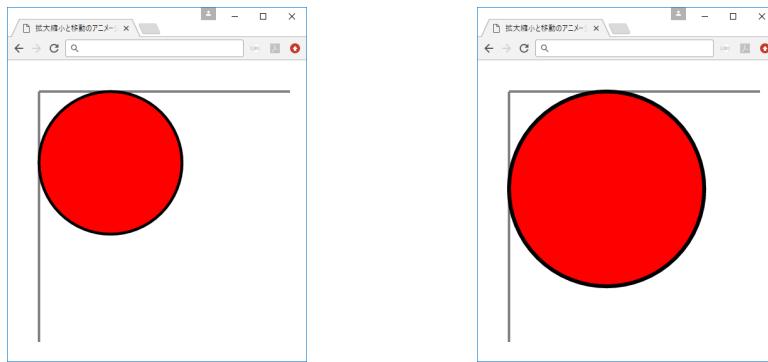


図 4.4: 拡大縮小と移動のアニメーション

```

5 <title>拡大縮小と移動のアニメーション</title>
6 <g transform="translate(50,50)">
7   <line x1="0" y1="0" x2="400" y2="0" stroke-width="4" stroke="gray"/>
8   <line x1="0" y1="0" x2="0" y2="400" stroke-width="4" stroke="gray"/>
9   <g>
10    <g>
11      <circle cx="0" cy="0" r="100"
12        stroke-width="4" stroke="black" fill="red"/>
13      <animateTransform attributeName="transform" attributeType="XML"
14        type="scale" from="1" to="2" dur="20s" fill="freeze"/>
15    </g>
16    <animateTransform attributeName="transform" attributeType="XML"
17        type="translate" from="100,100" to="200,200" dur="20s" fill="freeze"/>
18  </g>
19 </g>
20 </svg>
```

- 7 行目と 8 行目ではアニメーションをする円の上端と左端の位置が変わらないことを確認するための直線を引いています。
- 円の大きさを `scale` で変化させるために `<circle>` 要素を囲む `<g>` 要素を用意します (10 行目)。
- この要素に `scale` の値が 1 から 2 へ変化するアニメーションを付けます (13 行目から 14 行目)。
- 11 行目の中心が (0,0) なので `scale` によりこのままでは上端と左端の直線から円ははみ出してしまう。これを避けるため 10 行目の `<g>` 要素の外側をさらに `<g>` 要素で囲み (9 行目)、この要素に `translate` のアニメーションを付けています (16 行目から 17 行目)

問題 4.4 `scale` を用いて長方形が横に伸びるアニメーションを作成しなさい。

4.3 道のりに沿ったアニメーション(<animateMotion>要素)

指定した道のりに沿って動くアニメーションでは<animateMotion>要素を用います。道程は属性pathで指定します。

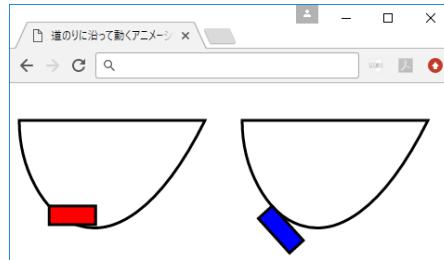


図 4.5: 道のりに沿って動くアニメーション

SVG リスト 4.4: 道のりに沿ったアニメーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>道のりに沿って動くアニメーション</title>
6  <defs>
7      <path id="MotionPath" d="M0,0 C0,100 100,200 200,0 z"
8          stroke-width="3" stroke="black" fill="none"/>
9      <rect id="MovingItem" x="0" y="0" width="50" height="20"
10         fill="currentColor" stroke-width="3" stroke="black"/>
11  </defs>
12  <g transform="translate(10,40)" color="red">
13      <use xlink:href="#MotionPath"/>
14      <use xlink:href="#MovingItem">
15          <animateMotion dur="10s" repeatCount="indefinite">
16              <mpath xlink:href="#MotionPath"/>
17          </animateMotion>
18      </use>
19  </g>
20  <g transform="translate(250,40)" color="blue" >
21      <use xlink:href="#MotionPath"/>
22      <use xlink:href="#MovingItem" >
23          <animateMotion dur="10s" repeatCount="indefinite" rotate="auto" >
24              <mpath xlink:href="#MotionPath"/>
25          </animateMotion>
26      </use>
27  </g>
28 </svg>

```

- 7行目から8行目でアニメーションで動くパスを定義しています。このパスは動きがわかるように表示にも使われています(13行目と21行目)。

- 14 行目から18 行目ではアニメーションで動く左側の長方形を定義しています。この長方形の内部を塗る `fill` が `currentColor` であることに注意してください。これは上位の環境で定義されている色で塗ることを意味します。ここでは12 行目にある `<g>` 要素の属性 `color` の値 (`red`) が利用されます。
- 15 行目から17 行目でこの長方形にアニメーションをつけています。`<defs>` 要素のなかで定義された道のりを参照するために `<mopath>` 要素を用いています。
- 右側の長方形についても同様のアニメーションが付きます(22 行目から26 行目)。このアニメーションには属性 `rotate` に `auto` を指定しているので道のりに接するように長方形が回転しながら移動します。`reverse-auto` という値も指定できます。

問題 4.5 リスト 4.4においてアニメーションの属性 `rotate` に `reverse-auto` を設定したときの動きを確認しなさい。

4.4 いろいろな属性に動きをつける

4.4.1 一般的の属性に変化をつける (`<animate>` 要素)

その他の属性にアニメーションをつけるのには `<animate>` 要素を用います。

色の変化 図 4.6 は円の塗り (`fill`) と縁取り (`stroke`) に個別のアニメーションをつけています¹。

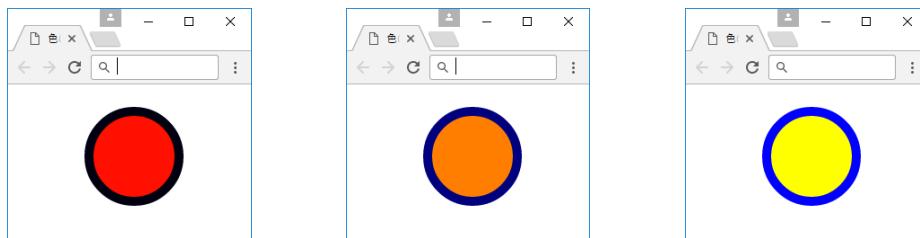


図 4.6: 色のアニメーション (開始時-左-、途中(中央)、終了時-右-)

SVG リスト 4.5: 色のアニメーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%" >
5  <title>色のアニメーション</title>
6  <g transform="translate(140,80)" >
7      <circle cx="0" y="0" r="50" stroke-width="10" stroke="black" fill="green">
8          <animate attributeName="fill" attributeType="CSS" begin="5s"

```

¹ 色に関する属性にアニメーションを付ける `<animateColor>` 要素がありますが、`<animate>` 要素で実現できるので将来的な規格ではなくなるとの記述があります (<https://www.w3.org/TR/SVG/animate.html#AnimateColorElement> の最後の部分)。

```

9      from="#ff0000" to="#ffff00" dur="10s" fill="freeze"/>
10     <animate attributeName="stroke" attributeType="CSS" begin="5s"
11       from="#000000" to="#0000ff" dur="10s" fill="freeze"/>
12   </circle>
13 </g>
14 </svg>
```

色の名前は CSS で定義されているので attributeType の属性値は CSS となります。

長方形の大きさの変化 リスト 4.6 は長方形の幅の属性 width にアニメーションをつけて形を変えています。

SVG リスト 4.6: 長方形の幅を変えるアニメーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3    xmlns:xlink="http://www.w3.org/1999/xlink"
4    height="100%" width="100%" >
5  <title>長方形の幅を変えるアニメーション</title>
6  <g transform="translate(100,50)" >
7    <rect x="0" y="0" width="100" height="50"
8      stroke-width="5" stroke="black" fill="green">
9      <animate attributeName="width" attributeType="XML"
10        from="100" to="200" dur="10s" fill="freeze"/>
11    </rect>
12  </g>
13 </svg>
```

- アニメーションは9行目から10行目の<animate>要素でつけています。
- アニメーションをつける属性名を attributeName の属性値に与え、attributeType に XML を指定します。

問題 4.6 リスト 4.3 における円のアニメーションのうち大きさを変えるアニメーションを半径で行うようにしなさい。それによりアニメーションの見え方がどのように変わるか調べなさい。

問題 4.7 <animateTransform> 要素の scale 属性にアニメーションをつけて例 4.6 と同じように長方形の形が変わるアニメーションを作成しなさい。また、この方法と例 4.6 とのアニメーションの違いがあるかどうか検討しなさい。

図形の形の変化 <path> 要素の属性 d にアニメーションをつけるためには d の属性値の構造をえてはいけません。たとえば、四角形を三角形に変化させるアニメーションでは最初に与えた点が4つならば最終の図形の三角形を4つ点で表す必要があります。

リスト 4.7 は円を正方形に変えるアニメーションです。

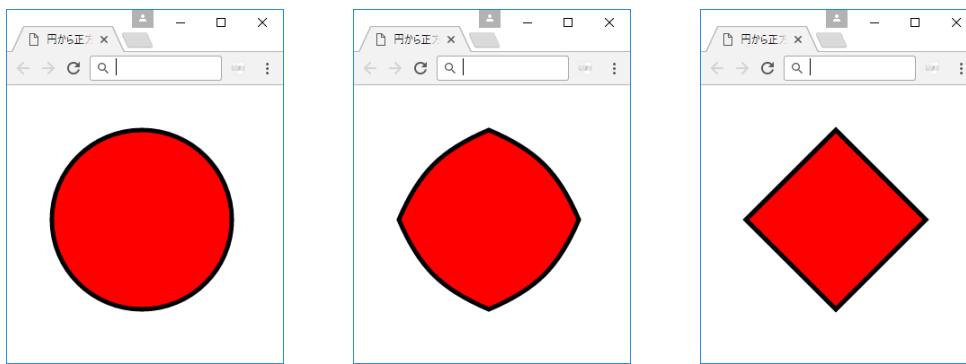


図 4.7: 円から正方形へ (開始時-左-、途中、終了時-右-)

SVG リスト 4.7: <path> 要素の属性 d にアニメーションをつける

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="300" width="300">
5      <title>円から正方形へ</title>
6      <g transform="translate(150,150)">
7          <path fill="red" stroke-width="5" stroke="black">
8              <animate attributeName="d" attributeType="XML"
9                  from="M-100,0
10                 C-100,-55.228 -55.228,-100 0,-100
11                 C55.228,-100 100,-55.228 100,0
12                 C100,55.228 55.228,100 0,100
13                 C-55.228,100 -100,55.228 -100,0z"
14                  to="M-100,0
15                 C-50,-50 -50,0,-100
16                 C50,-50 50,-50 100,0
17                 C50,50 50,50 0,100
18                 C-50,50 -50,50 -100,0z"
19                  dur="10s" fill="freeze"/>
20          </path>
21      </g>
22  </svg>
```

- 円を近似して描く解説は例 3.13 を参考にしてください。9 行目から 12 行目までの from の値はそこに現れる値を用いています。
- なお、例 3.13 では曲線の定義に対称な Bézier 曲線を定義する s を用いていますが、ここでは 4 つの独立した Bézier 曲線に直しています。
- d にアニメーションをつけるときは from で定義したデータの形を変えることができないので、to で示す正方形も Bézier 曲線で表す必要があります。ここでは途中の制御点を開始点と終了点の中点にしています。

問題 4.8 <path> 要素を用いて長方形を描き、それに形を変えるアニメーションをつけなさい。

グラデーションを横に動かす 線形グラデーションの gradientUnits の値を userSpaceOnUse にするとグラデーションの開始位置 (x1 や y1) や終了位置 (x2 や y2) を図形とは無関係な位置に指定できます。これらの属性にアニメーションをつけるとグラデーションの色が横に流れます (図 4.8)。

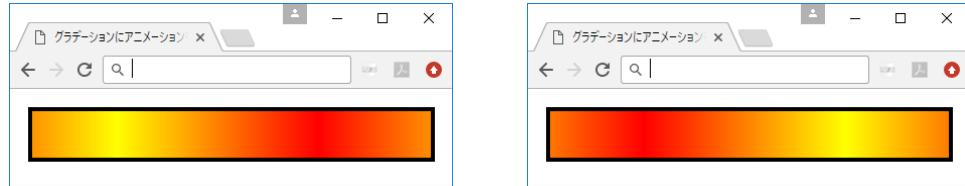


図 4.8: グラデーションにアニメーションを付ける

SVG リスト 4.8: グラデーションにアニメーションを付ける

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5      <title>グラデーションにアニメーションを付ける</title>
6      <defs>
7          <linearGradient id="Gradiation1" gradientUnits="userSpaceOnUse"
8              x1="0" y1="0" x2="800" y2="0">
9              <stop stop-color="yellow" offset="0%"/>
10             <stop stop-color="red"    offset="25%"/>
11             <stop stop-color="yellow" offset="50%"/>
12             <stop stop-color="red"    offset="75%"/>
13             <stop stop-color="yellow" offset="100%"/>
14             <animate attributeName="x1" attributeType="XML"
15                 from="0" to="-400" dur="5s" repeatCount="indefinite"/>
16             <animate attributeName="x2" attributeType="XML"
17                 from="800" to="400" dur="5s" repeatCount="indefinite"/>
18         </linearGradient>
19     </defs>
20     <g transform="translate(20,20)">
21         <rect x="0" y="0" width="400" height="50"
22             stroke="black" stroke-width="4" fill="url(#Gradiation1)"/>
23     </g>
24 </svg>
```

- 7 行目から 18 行目でアニメーションを伴った線形グラディエーションを定義しています。
 - 線形グラデーションの開始位置を変化させてるので gradientUnits の値を userSpaceOnUse にします (7 行目)。
 - 8 行目で塗る範囲を定義しています。x1 と x2 の差 (800) が線形グラディエーションを適用する長方形 (21 行目から 22 行目) の幅 (400) の 2 倍になっていることに注意してください。グラデーションが端まで行ったときに連続して変化するように見せるために、同じパターンを 2 回繰り返したものを使っているからです。

- グラデーションのパターン 赤 ⇒ 黄 ⇒ 赤が2回繰り返されています(9行目から13行目)。
- 線形グラデーションの位置を変更するために x1 と x2 にアニメーションをつけています(14行目から15行目と16行目から17行目)。
- 21行目から22行目でアニメーションが付いた線形グラディエーションを塗る長方形を定義しています。

問題 4.9 リスト 4.8 のアニメーションで stop-color にアニメーションを付けて同じように見えることができるか検討しなさい。

問題 4.10 グラデーションを構成する<stop> 要素の属性にアニメーションを付けることができます。ザバーニョの錯視を構成する長方形の線形グラディエーションの片方の端の stop-color にアニメーションを付けて見え方の変化を調べなさい。

4.4.2 属性値をすぐに変える—<set> 要素を利用したアニメーション

<animate> 要素の代わりに<set> 要素を使うと、途中は from で指定された値のままでアニメーションの終了時に to で指定された値に設定されます。ここでは図形を表示するかどうかを決める visibility 属性に利用します。色に対して<set> 要素を利用した例はリスト 4.12 にあります。

SVG リスト 4.9: 図形が 7 秒後消える

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5      <title>図形が 7 秒後消える</title>
6      <g transform="translate(150,100)">
7          <circle cx="0" cy="0" r="50" stroke-width="8" fill="lime">
8              <animateColor attributeName="stroke" attributeType="CSS"
9                  from="yellow" to="orange" dur="5s" fill="freeze"/>
10             <set attributeName="visibility" attributeType="CSS"
11                 to="hidden" fill="freeze" begin="7s" />
12         </circle>
13     </g>
14 </svg>
```

アニメーションの属性 calcMode の値を discrete にすると<set> 要素と同じ動作をします。

4.5 複数の値を指定する (keyTimes,values)

values を用いるとアニメーションの途中の値を指定することができます。この場合、与えられた値の数でアニメーションの時間が等分に分けられます。この値を変更するためには keyTimes を用います。keyTimes で与える数値はアニメーションが行われる時間に対する割合を指定します。

次の例は 4.1 に対し、初めの位置まで戻す動きを付け加えたものです

SVG リスト 4.10: 初めの位置に戻る

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%" >
5  <title>初めの位置に戻る</title>
6  <g transform="translate(100,50)" >
7      <rect x="0" y="0" width="100" height="50"
8          stroke-width="5" stroke="black" fill="green"/>
9      <g transform="rotate(45)" >
10         <rect x="0" y="50" width="100" height="50"
11             stroke-width="5" stroke="blue" fill="red"/>
12     </g>
13     <animateTransform attributeName="transform" attributeType="XML"
14         type="translate" values="100,50;200,50;100,50" keyTimes="0;0.66;1"
15         dur="15s" fill="freeze"/>
16   </g>
17 </svg>

```

- 14 行目にある `values` と `keyTimes` でアニメーションが定義されています。
- `keyTimes` が `0;0.66;1` となっているので右へ移動するときの速度が左へ移動する速度に比べて約半分の速度で移動します。

問題 4.11 正方形が同じ速度で (100,100) から (200,100), (200,200), (100,200) を経て元の位置へ戻るアニメーションを作成しなさい。

問題 4.12 問題 3.21 の画像を初めは両方表示し、一定の期間がたつたら一方だけを表示し、さらに時間が経ったらもう一方の画像を表示するアニメーションを付けなさい。

4.6 イベントを利用したアニメーション

4.6.1 イベントとは

アプリケーションの実行中にシステムからそのアプリケーションに通知される情報をイベントといいます。アプリケーション側では渡されたイベントの情報を基にして動作を変更することが可能となります(無視することも対応のひとつです)。イベントとしては「マウスボタンがクリックされた」、「一定の時間が経過した」、「別のアニメーションが終了した」などがあります。イベントを利用してアニメーションの開始や終了を指示できます。イベントを利用してより細かく SVG 文書を制御する方法については第 7 章で解説します。

4.6.2 マウスのイベントを利用したアニメーション

図 4.9 では下部の黒い長方形の上にマウスを乗せると赤い矢印が回転します。そこからマウスを離すと動きが止まります。再びマウスをその場所に乗せるとはじめの位置から再び回転します。

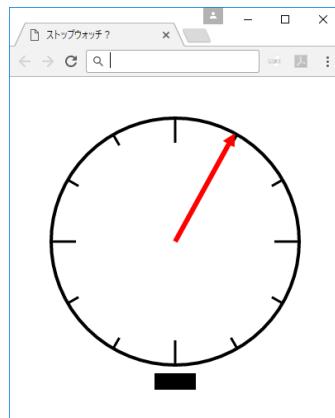


図 4.9: ストップウォッチ?

SVG リスト 4.11: ストップウォッチ?

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="400" width="400">
5  <title>トップウォッチ?</title>
6  <defs>
7      <line id="TickL" x1="120" y1="0" x2="150" y2="0"
8          stroke-width="3" stroke="black"/>
9      <line id="TickS" x1="135" y1="0" x2="150" y2="0"
10         stroke-width="3" stroke="black" />
11     <g id="Ticks4">
12         <g transform="rotate(0)"> <use xlink:href="#TickL"/></g>
13         <g transform="rotate(30)"><use xlink:href="#TickS"/></g>
14         <g transform="rotate(60)"><use xlink:href="#TickS"/></g>
15     </g>
16 </defs>
17 <g transform="translate(200,200) rotate(-90)">
18     <circle cx="0" cy="0" r="150" stroke="black" stroke-width="4" fill="none"/>
19     <g transform="rotate(0)"> <use xlink:href="#Ticks4"/></g>
20     <g transform="rotate(90)"> <use xlink:href="#Ticks4"/></g>
21     <g transform="rotate(180)"><use xlink:href="#Ticks4"/></g>
22     <g transform="rotate(270)"><use xlink:href="#Ticks4"/></g>
23     <path d="M0,-3 135,-3 135,-8 150,0 135,8 135,3 0,3z" fill="red">
24         <animateTransform attributeName="transform"
25             attributeType="XML" type="rotate"
26             from="0" to="360" dur="60s" repeatCount="indefinite"
27             begin="Face.mouseover" end="Face.mouseout" fill="freeze"/>
28     </path>
29 </g>
30     <rect x="175" y="360" width="50" height="20" fill="black" id="Face"/>
31 </svg>
```

- 6行目から16行目は時計盤の目盛りを描くためのパーツを定義しています。
 - 7行目から8行目では 90° ごとに現れる長い目盛りを、9行目から10行目では残りの目盛りを定義しています。
 - 長い目盛りを基準に短い目盛りを 30° と 60° 回転したものをひとつのものとしてまとめ定義します(11行目から15行目)。
- 18行目で時計盤の外周を描いています。
- 19行目から22行目で `<defs>` 要素内で定義した目盛りを 90° ずつ回転したものを4つ使って時計盤の目盛りを作成しています。
- 23行目では `<path>` 要素を用いて秒針を作成しています。
- この図形に対し24行目から27行目で回転のアニメーションを付けています。
 - このアニメーションは0から360の値を60秒かけて変化させるようにしているので、ちょうど一分で一回転することになります。
 - `begin` はアニメーションの開始時を指定する属性で、その値は `Face.mouseover` です。30行目の長方形の属性 `id` の値が `Face` なので `Face.mouseover` はこの長方形に「マウスカーソルが乗ったとき」を意味します。
 - `end` はアニメーションの終了時を指定する属性です。ここでは `Face.mouseout` となっているのでこの長方形から「マウスカーソルが出たとき」を意味します。

問題 4.13 リスト 4.11 に分針をつけたストップウォッチを作成しなさい。また、ボタンを2つ置き、片方がクリックされたら動き出し、他方がクリックされたら停止するものを作成しなさい。

問題 4.14 ポッケンドルフの錯視(図 2.10)の中央の長方形に `opacity` にアニメーションをつけて、その上にマウスカーソルを載せるとその長方形がだんだん消えていく、どの直線が左右につながっているかを示すようにしなさい。

4.6.3 アニメーション終了のイベントを利用する

リスト 4.12 はアニメーションの開始を他のアニメーションの終了時に設定することで信号機の明かりの変化をシミュレーションしています。

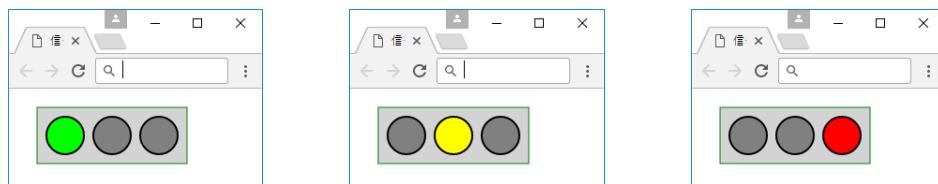


図 4.10: 信号機

SVG リスト 4.12: 信号機のシミュレーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>信号機のシミュレーション</title>
6  <defs>
7      <circle r="20" id="sign" cy="30" stroke-width="2" stroke="black"/>
8  </defs>
9  <g transform="translate(30,20)">
10     <rect x="0" y="0" width="160" height="60" fill="lightgray"
11         strok-width="2" stroke="green" id="rect"/>
12     <use xlink:href="#sign" x="130" id="Red" fill="gray" >
13         <set attributeName="fill" attributeType="CSS" id="inRed"
14             to="red" begin="rect.click;inYellow.end" dur="5s" fill="remove"/>
15     </use>
16     <use xlink:href="#sign" x="80" id="Yellow" fill="gray" >
17         <set attributeName="fill" attributeType="CSS" id="inYellow"
18             to="yellow" begin="inBlue.end" dur="2s" fill="remove"/>
19     </use>
20     <use xlink:href="#sign" x="30" id="Blue" fill="gray">
21         <set attributeName="fill" attributeType="CSS" id="inBlue"
22             to="lime" begin="inRed.end" dur="5s" fill="remove"/>
23     </use>
24 </g>
25 </svg>
```

- 7 行目で信号機の明かりの大きさを同一にするため、円を定義しています。
- 10 行目から11 行目で信号機の全体を示す長方形を定義しています。
- 12 行目から15 行目で赤色の信号を定義しています。
 - id で参照するための名前 inRed を定義しています。
 - 明かりの水平方向の位置を x で定義していることに注意してください。<use> 要素ではすでに図形が定義されているので cx では定義できないようです。
 - 信号の明かりの色を付いていない状態 (gray) に設定しています。
 - 13 行目と14 行目で fill にアニメーションをつけています。
 - * 属性 id を inRed に設定しています。
 - * アニメーションの開始を指定する begin に 0s;inYellow.end を与えています。この指定により、この SVG ファイルの開始時 (0s) と inYellow で参照されているアニメーションの終了時 (end) にこのアニメーションが開始されます。
 - * アニメーションの継続時間は dur で指定しています。
 - * アニメーションの終了時 (end) には元の値に戻す remove を指定しています。
- 16 行目と19 行目では黄色の、20 行目から23 行目では青色の信号をそれぞれ定義しています。

問題 4.15 リスト 4.12においてアニメーションの開始、終了を時間で与えるようにしたときと組む手間を比較しなさい。

4.7 アニメーションがついた錯視図形

今までに出てきた錯視図形で錯視の原因となる部分にアニメーションをつけることで見え方の変化を楽しむことができました。今までに出てきたものにアニメーションを付けることができます。

- 問題 2.4 で長方形の境界を隠すような図形をアニメーションで表示、移動させる。
- カフェウォール錯視(23 ページ、図 2.14)の細い線に黒から明るい灰色に変化するアニメーションを付ける。

問題 4.16 図 4.11 は放射状に描かれた直線群のためにその中にある正方形が歪んで見えます。この正方形が上下に移動するアニメーションを付けて形が見え方の変化を調べなさい。

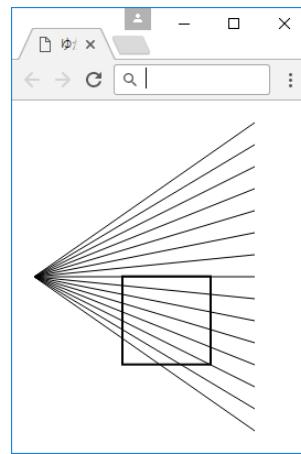


図 4.11: ゆがんだ正方形

追いかけっこをする長方形 図 4.12 は背景が黒と明るい灰色(lightgrey)の同じ幅の縦じまで塗られた背景上を明るい灰色、灰色と黒に塗られた小さな長方形が等速度で移動するものです。

長方形の両端でその色が背景の色と同じときにはその長方形は動いているように見えません(この図では黒の長方形が停止しているように見えます)。

したがって、等速度で移動して見えるものは真ん中にある灰色の長方形だけで、残りの二つの色の長方形は止まっている状態から灰色の長方形に追いつくというギクシャクした動きに見えます。

図 4.13 では背景の黒の部分を灰色に変えています。これにより等速度で動いているように見えるのは黒の長方形になります。

図 4.14 では背景の黒の部分を明るい灰色に変えています。

一番上の長方形は見えなくなり、残りの二つの長方形は等速度で移動するように見えます。

リスト 4.13 はこれら 3 つの図を順番に表示するものです。

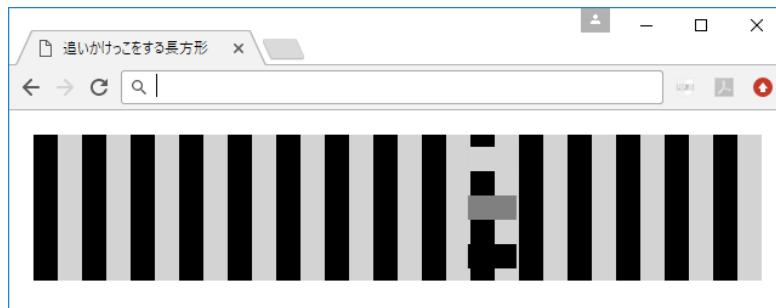


図 4.12: 追いかけっこをする長方形(その 1)

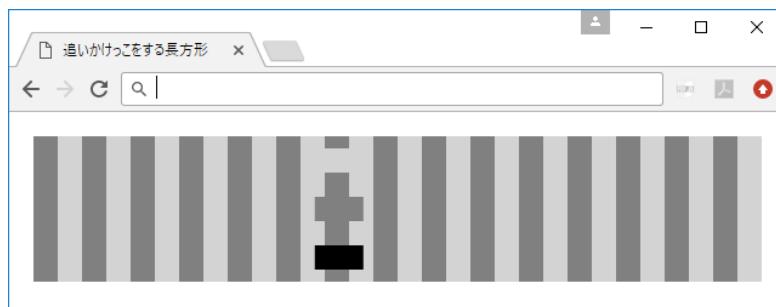


図 4.13: 追いかけっこをする長方形(その 2)

SVG リスト 4.13: 追いかけっこをする長方形

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>追いかけっこをする長方形</title>
6  <defs>
7      <pattern id="BackGround" x="0" y="0" patternUnits="userSpaceOnUse"
8          width="40" height="300">
9          <rect x="0" y="0" width="20" height="300">
10         <animate attributeName="fill" dur="10s" calcMode="discrete"
11             values="black;gray;lightgray" repeatCount="indefinite"/>
12         </rect>
13         <rect x="20" y="0" width="20" height="300" fill="lightgray"/>
14     </pattern>
15     <rect id="MoveH" width="40" height="20" fill="currentColor">
16         <animate attributeName="x" attributeType="XML"
17             values="10;560;10" keyTimes="0;0.5;1" dur="25s"
18             repeatCount="indefinite"/>
19         </rect>
20     </defs>
21     <g transform="translate(20,20)">
22         <rect x="0" y="0" width="600" height="120" fill="url(#BackGround)"/>

```

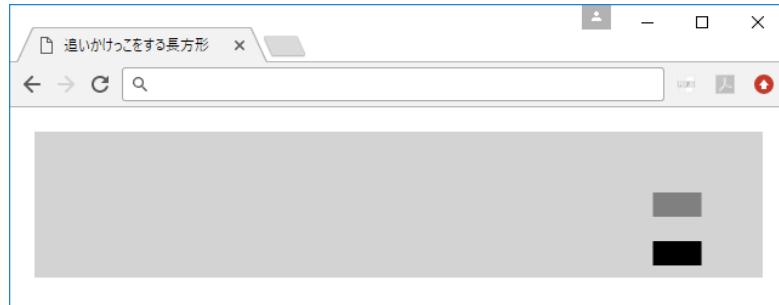


図 4.14: 追いかけっこをする長方形(その 3)

```

23   <use xlink:href="#MoveH" y="10" color="lightgray"/>
24   <use xlink:href="#MoveH" y="50" color="gray"/>
25   <use xlink:href="#MoveH" y="90" color="black"/>
26   </g>
27 </svg>
28

```

- 7 行目から14 行目で背景の作成するためのパターンを定義しています。
 - 9 行目から12 行目で黒、灰色、明るい灰色と順番に色を変える長方形を定義しています。
 - 色を変えるアニメーションは10 行目から11 行目で定義しています。
 - calcMode が discrete なので指定した時間にこれらの色に断続的に変化します。
 - 繰り返しの指定 (repeatCount) が indefinite なのでアニメーションは停止しません。
 - 13 行目では色が変化しない長方形を定義しています。
- 15 行目から19 行目では横に移動する長方形の離形を定義しています。
 - 塗りつぶしの色 (fill) は currentColor としていて引用されている先で定義されます。
 - 16 行目から18 行目で横に動く長方形のアニメーションを定義しています。
 - ここでは長方形の横位置を示す属性 x にアニメーションを付けています。
 - 右端で戻るように values の値を 10;560;10 にしています (17 行目)。
- 22 行目で背景として長方形を7 行目から14 行目で定義したパターンで塗っています。
- 23 行目から25 行目で横に動く長方形を 3 つ定義しています。それぞれの塗りつぶしの色を color で指定しています。

問題 4.17 リスト 4.13 について次のことについて調べなさい。

1. 移動する長方形の塗りつぶしの色を変えても同じように見えるかどうか
2. 長方形以外の形でも可能かどうか

第5章 文字列の表示

5.1 文字列表示の基礎

図 5.1 は SVG 画像の中に文字列を表示しています。

文字列を表示するには`<text>`要素を用います。



図 5.1: 文字列の表示

SVG リスト 5.1: 文字列の表示

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>文字列の表示</title>
6  <defs>
7      <g id="ShowPosition">
8          <line x1="-20" y1="0" x2="200" y2="0" stroke-width="1" stroke="black"/>
9          <line x1="0" y1="-20" x2="0" y2="20" stroke-width="1" stroke="black"/>
10     </g>
11 </defs>
12 <g transform="translate(100,100)">
13     <text x="0" y="0" fill="red" font-size="20px">
14         This is an example.
15     </text>
16     <use xlink:href="#ShowPosition"/>
17 </g>
18 <g transform="translate(100,150)">
19     <text x="0" y="0" fill="green" font-size="25px">

```

```

20      日本語も表示できます
21      </text>
22      <use xlink:href="#ShowPosition"/>
23  </g>
24  </svg>
```

- この表示では文字列がどこに表示されるかをわかりやすくするために (0,0) で交わる 2 直線を書く図形を定義しています (8 行目と 9 行目)。
- 初めの文字列は属性 `x` や `y` の値から表示する位置は (0,0) です。
- 文字列は `fill` を指定することで赤で表示されます。
- 表示位置は特別に指定しないので左下が基準になっています。
- 二番目の文字列は日本語を表示しています。文字列の左上が (0,0) の位置になります (19 行目から 21 行目)。

文字の表示のための属性とその属性値を表 5.1 にまとめておきます。Opera 12.17 ではこれらの属性がすべて利用できるわけではありません。特に、`dominant-baseline` はサポートされていません。

SVG は多言語に対応しているので文字列の水平方向の位置が `left`, `right` ではないことに注意してください。

問題 5.1 `left` や `right` を使わない理由は何か調査しなさい。

5.2 部分的に文字の表示を変える方法

前節で述べた `<text>` 要素には文字の表現を変えるための属性が定められています。横に並んだ文字列の一部だけ文字の表現を変えるためには `<text>` 要素だけでは変える文字列の先頭位置を指定するのが面倒です。これをするためには `<tspan>` 要素を用います。

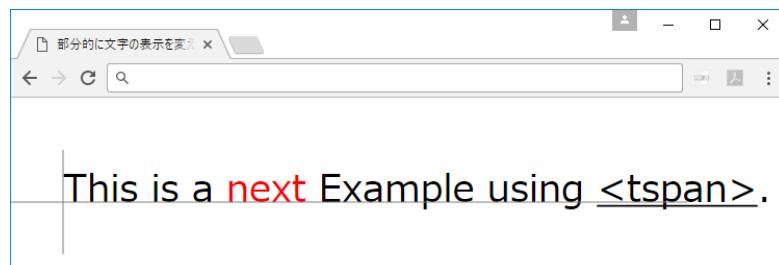


図 5.2: 部分的に文字の表示を変える

表 5.1: 文字列表示の属性と属性値

属性名	属性値	意味
text-anchor	水平方向の位置	
	start middle end	文字列の開始位置 文字列の中央 文字列の終了位置
dominant-baseline	垂直方向の位置	
	ideographic alphabetic hanging mathematical auto	文字列の一番下 文字 x の下 文字列の一番上 文字列の中央 自動
baseline-shift	文字列の上下の移動	
	baseline sub super 割合% 長さ	添え字の位置 上付きの位置
font-family	フォントの種類の指定	
	font-family serif sans-serif cursive fantasy monospace	フォント名を指定 (漢字は含めない) 文字幅等間隔
font-style	フォントの形状	
	normal italic oblique	標準 イタリック 傾けたもの
font-stretch		
font-size	数字	フォントの大きさ
text-decoration	文字列の飾り	
	none underline overline line-through	なし 下線 上線 打ち消し線

SVG リスト 5.2: <tspan> 要素の使用例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>部分的に文字の表示を変える</title>
6  <g transform="translate(50,100)">
7      <text x="0" y="0" font-size="36px">
8          This is a <tspan fill="red">next</tspan>
9          Example using
10         <tspan text-decoration="underline">&lt;tspan&gt;</tspan>.
11     </text>
12     <line stroke-width="1" x1="-50" y1="0" x2="600" y2="0" stroke="gray"/>
13     <line stroke-width="1" y1="-50" x1="0" y2="50" x2="0" stroke="gray"/>
14   </g>
15 </svg>

```

- 8行目では next の文字を<tspan>要素 ではさんで色を赤に変えています。
- 10行目では<tspan>要素を用いて文字列の下線をつけています。なお、<や>を表示するためここでは< や > というエンティティを用いていることに注意してください。

5.3 文字をグラデーションで塗る

最近の計算機に含まれるフォントはアウトラインフォントと呼ばれる形式で保持されています。アウトラインフォントとは文字の形(グリフ)を輪郭線の形で持っているものです。したがって、アウトラインフォントは図形と同等の機能を持っているのでマスクのパターンとして利用することができます。



図 5.3: 文字列をグラデーションで塗る

SVG リスト 5.3: 文字列をグラデーションで塗る

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"

```

```

4      height="100%" width="100%">
5  <title>文字列をグラデーションで塗る</title>
6  <style type="text/css">
7    .textStyle {font-size:100px; text-anchor:middle; font-family:Impact;}
8  </style>
9  <defs>
10    <linearGradient id="Gradiation1" gradientUnits="objectBoundingBox">
11      <stop stop-color="yellow" offset="0%"/>
12      <stop stop-color="red" offset="100%"/>
13    </linearGradient>
14    <mask id="text" class="textStyle" maskUnits="userSpaceOnUse"
15      x="0" y="0" width="800" height="200">
16      <text x="400" y="100" class="textStyle"
17        fill="white" >This is an Example.</text>
18    </mask>
19  </defs>
20  <g transform="translate(50,0)">
21    <rect x="0" y="0" width="800" height="200" fill="url(#Gradiation1)"
22      mask="url(#text)"/>
23    <text x="400" y="100" class="textStyle" fill="none"
24      stroke-width="4" stroke="black">This is an Example.</text>
25    <line x1="0" y1="0" x2="0" y2="150" stroke-width="1" stroke="black"/>
26    <line x1="800" y1="0" x2="800" y2="150" stroke-width="1" stroke="black"/>
27  </g>
28 </svg>

```

- 7行目で表示する文字列の属性を定義しています。
 - フォントの大きさ (font-size) を 100px
 - テキストと水平方向の位置 (text-anchor) を中央 (middle) に
 - 使用するフォント (font-family) を Impact という線の幅が広いフォントに設定¹
- 10行目から13行目でグラデーションを定義しています。
- 14行目から18行目で<mask>要素を定義しています。
 - この<mask>要素のなかに<text>要素があり、中を white で塗りつぶしています (16行目と17行目)。
 - 文字の大きさなどは CSS の要素で決定されます (7行目)。
- 21行目と22行目で長方形を10行目から13行目で定義したグラディエーション塗りつぶしています。この行にはもうひとつ mask が指定されているのでこの範囲だけ塗られることになります。
- 25行目と26行目は長方形の左と右の位置を示すためのものです。

問題 5.2 図 5.3 にグラデーションが横に流れるアニメーションを付けなさい。

¹残念ながらこのフォントは日本語フォントではありません。

5.4 道程に沿った文字列の配置

文字列を道程に沿って配置することができます。

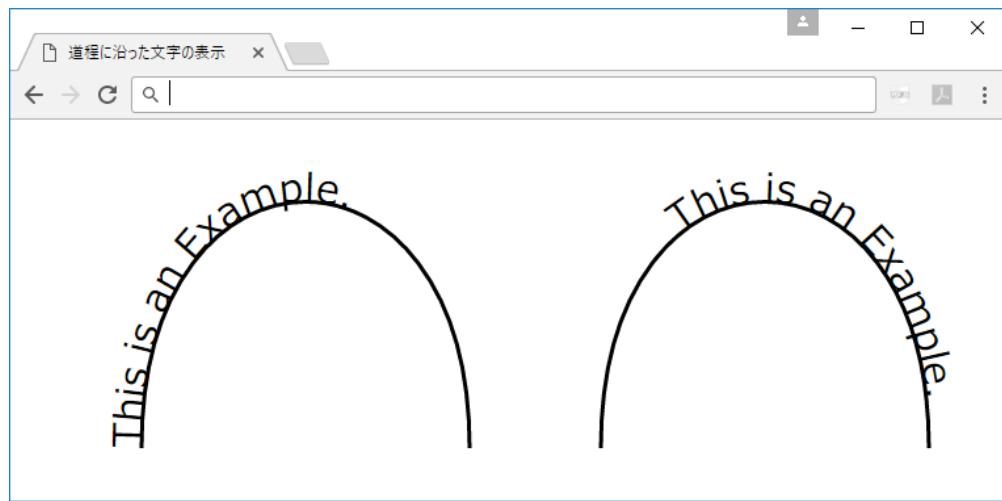


図 5.4: 道程に沿った文字の表示

SVG リスト 5.4: 道程に沿った文字の表示

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3   xmlns:xlink="http://www.w3.org/1999/xlink"
4   height="100%" width="100%">
5   <title>道程に沿った文字の表示</title>
6   <defs>
7     <path d="M0,0 C0,-250 250,-250 250,0"
8       id="TextPath" fill="none" stroke="black" stroke-width="3"/>
9   </defs>
10  <g transform="translate(100,250)">
11    <text>
12      <textPath xlink:href="#TextPath" font-size="30px">
13        This is an Example.
14      </textPath>
15    </text>
16    <use xlink:href="#TextPath"/>
17  </g>
18  <g transform="translate(450,250)">
19    <text>
20      <textPath xlink:href="#TextPath" font-size="30px">
21        This is an Example.
22        <animate attributeName="startOffset"
23          from="100%" to="0%" begin="0s" dur="10s" fill="freeze"/>
24      </textPath>
25    </text>
26    <use xlink:href="#TextPath"/>
```

```
27      </g>
28  </svg>
```

- 7 行目から 8 行目にかけてテキストを配置する道程を定義しています。ここでは 3 次の Bézier 曲線を利用します。
- この例ではテキストを 2 通りの方法で配置します。同一のテキストを利用することを強調するために行目から行目にかけてその文字列を定義しています。
- 11 行目から 15 行目にかけて一つ目の文字列を定義したパスに沿って配置しています。これは<text> 要素内に<textPath> 要素を記述することで実現できます。このタグ内で上で定義した道程を属性 xlink:href で引用し、さらに font-size でフォントの大きさを指定しています(12 行目)。
- 16 行目では文字列を配置した道のりを描いています。
- 19 行目から 25 行目でも 11 行目から 15 行目と同じことをしています。異なるのは<textPath> 要素の属性 startOffset にアニメーションをつけていることです(22 行目から 23 行目)。
- 属性 startOffset は指定された文字列を指定された道程のどの位置から配置するのかを決めるものです。長さを指定する数値か、道程に対する割合を%を用いて表すかの方法があります。
- ここでは 100%から 0%へ変化するアニメーションをつけてるので道程の終了点から道程の開始点へ文字列が移動することになります。
- 文字列の道程から外れた部分は表示されないようです。

問題 5.3 閉じた道のりに対して文字列が移動するアニメーションを作成し、文字列がどのように現れるか調べなさい。また、startOffset の値を負にしたらどうなるかも調べなさい。

5.5 円周上に沿って文字列が移動するアニメーション

道程に沿った文字列の配置では(名称から当然ですが)<path> 要素で指定された道程だけが有効のようです。したがって、円に沿って文字を配列するためには円を Bézier 曲線で近似した曲線上に配置する必要があります。

次の例はこの Bézier 曲線で近似した円周上を文字列が移動するアニメーションです。²

このアニメーションは図 5.4 と同じように<textPath> 要素の属性 startOffset にアニメーションをつけて実現しています。しかし、与えられた<textPath> 要素から外れた部分の文字は表示されないので単一の円では円周を移動するアニメーションが実現できません。これを解決するための

²これだけであれば transform の rotate にアニメーションをつければすむことですが後々の拡張もかねて少しトリッキーな方法で実現しました。

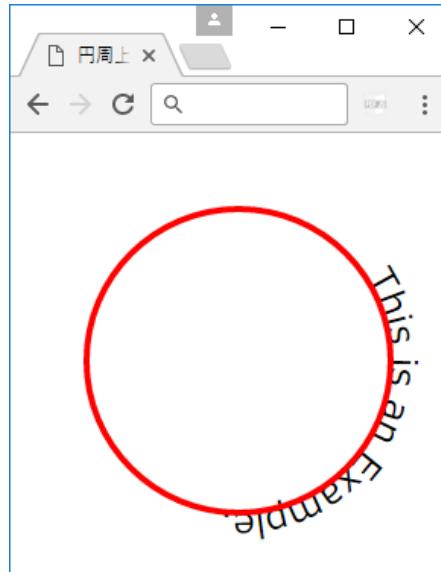


図 5.5: 円周上を文字列が移動するアニメーション

方策として開始位置が異なる二つの Bézier 曲線で近似した円周上を動く文字列のアニメーションをふたつ用意し、文字列の位置によってどちらか一方を表示することにします。

SVG リスト 5.5: 円周上を文字列が移動するアニメーション

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>円周上を文字列が移動するアニメーション</title>
6  <defs>
7      <path id="Upper"
8          d="M-100,0 C-100,-55.228 -55.228,-100 0,-100
9              S100,-55.228 100,0 S55.228,100 0,100 S-100,55.228 -100,0"
10             stroke-width="4"/>
11     <path id="Lower"
12         d="M100,0 C100,55.228 55.228,100 0,100
13             S-100,55.228 -100,0 S-55.228,-100 0,-100 S100,-55.228 100,0"
14             stroke-width="4" stroke="black"/>
15 </defs>
16 <g transform="translate(150,150)">
17     <text>
18         <textPath xlink:href="#Upper" font-size="24px">
19             This is an Example.
20             <animate attributeName="startOffset"
21                 values="50%;50%;0%" dur="30s" repeatCount="indefinite"/>
22         </textPath>
23         <animate attributeName="opacity" values="0;0;1;1"
24             keyTimes="0;0.5;0.5;1" repeatCount="indefinite" dur="30s"/>
25     </text>

```

```

26 <text>
27   <textPath xlink:href="#Lower" font-size="24px">
28     This is an Example.
29     <animate attributeName="startOffset"
30       values="50%;0%;0%" dur="30s" repeatCount="indefinite"/>
31   </textPath>
32   <animate attributeName="opacity" values="1;1;0;0"
33     keyTimes="0;0.5;0.5;1" repeatCount="indefinite" dur="30s"/>
34 </text>
35   <use xlink:href="#Upper" fill="none" stroke="red" />
36 </g>
37 </svg>

```

6行目から15行目の`<defs>`要素では開始点が異なる二つのBézier曲線で近似した円と表示する文字列を定義しています。

```

6  <defs>
7    <path id="Upper"
8      d="M-100,0 C-100,-55.228 -55.228,-100 0,-100
9        S100,-55.228 100,0 S55.228,100 0,100 S-100,55.228 -100,0"
10       stroke-width="4"/>
11    <path id="Lower"
12      d="M100,0 C100,55.228 55.228,100 0,100
13        S-100,55.228 -100,0 S-55.228,-100 0,-100 S100,-55.228 100,0"
14       stroke-width="4" stroke="black"/>
15 </defs>

```

- 7行目から10行目では中心が(0,0)で半径が100の円周を定義しています。この円周は点(-100,0)から始まり、時計回りに回っています。この値についてはリスト3.7を見てください。

この円周は円周上の右半分から上半分で文字列を表示するために利用されます。

- 11行目から14行目でも同様の円周を定義しています。この円周は開始点が(100,0)となっています。

この円周は円周上の左半分から下半分で文字列を表示するために利用されます。

17行目から25行目までは円周上を移動するアニメーションが付いた文字列の表示を定義しています。

```

17 <text>
18   <textPath xlink:href="#Upper" font-size="24px">
19     This is an Example.
20     <animate attributeName="startOffset"
21       values="50%;50%;0%" dur="30s" repeatCount="indefinite"/>
22   </textPath>
23   <animate attributeName="opacity" values="0;0;1;1"
24     keyTimes="0;0.5;0.5;1" repeatCount="indefinite" dur="30s"/>
25 </text>

```

- 18行目では7行目から始まる円周に沿って文字を表示するように指定しています(xlink:href)。
- 19行目では表示する文字列を直接記述しています。
- 20行目から21行目ではstartOffsetにアニメーションをつけています。
 - 位置の値が50%50%0\$となっているのでアニメーションは7行目で定められている<path>要素の中央つまり(100,0)の位置から開始されます。
 - アニメーションの継続時間は30秒ですが、前半は移動が起きていません。
 - 23行目から24行目で定義されているアニメーションは不透明度のものです。前半は値が0になっているのでこの文字列は見えません。
- 29行目から30行目と33行目から行目でも時間が半分ずれたアニメーションをつけています。
- 35行目では文字列が移動する円周を色を赤で描いています。

問題 5.4 リスト 5.4 について次のことを確認しなさい。

1. 円周の半分の長さになるような文字列に対してこのリストがうまく動くかどうか確認しなさい。
2. 別の曲線に対しても同じような動作をするアニメーションを作成しなさい。

第6章 フィルタ

SVG には与えられた画像にたいしてぼかしたり他の画像と合成したりするフィルターの機能があります。標準で付いてくるフィルタについて解説します。

6.1 フィルターに関する一般的な注意

一般にフィルタのある画像に対して行うと得られた画像は元の画像と大きさや位置が変わります。このことを考慮しておかないと得られた画像の一部が欠けたりする場合があります。

- フィルタをかける範囲の基準として属性 `filterUnits` があります。この属性値としてはグラデーションやマスクでも利用した `userSpaceOnUse` と `objectBoundingBox` があります。後者を用いるとフィルタをかけた画像は元の画像の範囲しか表示されません。
- フィルタをかけ始める位置を指定する `x` と `y` や範囲を指定する `width` や `height` でも同じことが言えます。この値の範囲外の部分は表示されません。

6.2 画像をぼかす (`feGaussianBlur` フィルタ)

画像をぼかす代表的なフィルタとして `feGaussianBlur` フィルタがあります。このフィルタはあるピクセルの近くにある一定の範囲のピクセルの色の情報からそのピクセルの色を決定します。このとき不透明度も計算されます。



図 6.1: `feGaussianBlur` フィルタ

SVG リスト 6.1: GaussianBlur フィルタ

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>GaussianBlur フィルタ</title>
6  <defs>
7      <rect id="Base" x="20" y="20" width="50" height="50"
8          fill="red" stroke-width="4" stroke="blue"/>
9      <filter id="blurFilter3" filterUnits="userSpaceOnUse"
10         x="0" y="0" width="100%" height="100%">
11         <feGaussianBlur stdDeviation="3"/>
12     </filter>
13     <filter id="blurFilter5" filterUnits="userSpaceOnUse"
14         x="0" y="0" width="100%" height="100%">
15         <feGaussianBlur stdDeviation="5"/>
16     </filter>
17     <filter id="blurFilter10" filterUnits="userSpaceOnUse"
18         x="0" y="0" width="100%" height="100%">
19         <feGaussianBlur stdDeviation="10"/>
20     </filter>
21 </defs>
22 <style type="text/css">
23 .textStyle {font-size:30px; text-anchor:middle; dominant-baseline:hanging}
24 </style>
25 <g transform="translate(30,30)">
26     <use xlink:href="#Base" y="0"/>
27     <text x="45" y="120" class="textStyle">0</text>
28     <use xlink:href="#Base" x="80" y="0" filter="url(#blurFilter3)"/>
29     <text x="125" y="120" class="textStyle">3</text>
30     <use xlink:href="#Base" x="160" y="0" filter="url(#blurFilter5)"/>
31     <text x="205" y="120" class="textStyle">5</text>
32     <use xlink:href="#Base" x="240" y="0" filter="url(#blurFilter10)"/>
33     <text x="285" y="120" class="textStyle">10</text>
34     <use xlink:href="#Base" x="345" y="0"/>
35     <use xlink:href="#Base" x="320" y="0" filter="url(#blurFilter10)"/>
36 </g>
37 </svg>

```

- 7行目から8行目にかけてフィルタをかける長方形を定義しています。
- 9行目から12行目でフィルタをひとつ定義しています。
 - フィルタを定義するには<filter>要素を用います。
 - フィルタはグラデーションの同様に後でフィルタをかけるオブジェクトから参照されるのでidで名称をつけます。
 - xとyでフィルタをかけ始める位置を指定します。負の値は指定できないようです。また、widthとheightで範囲を指定します。
 - ここではフィルタをかける単位としてuserSpaceOnUseを用いていますのでフィルタはかけられるオブジェクトが配置される空間が基準となります。

- 11 行目でぼかしのフィルタを feGaussianBlur フィルタで利用することを宣言しています。
- このフィルタは stdDeviation でぼかしの度合いを指定できます。0 はまったくかけないことを意味します。
- 13 行目から 16 行目と 17 行目から 20 行目でぼかしの度合いを変えたフィルタを定義しています。
- 22 行目から 24 行目では <style> 要素による属性値を定義しています。これは各画像の下にぼかしのパラメータの値を表示するテキストを表示の形式を統一するために CSS によるスタイルを採用しました。ここで定義できる属性値は CSS で定義できるものです。
 - HTML 文書では統一した表現をするために class で指定された要素の内容の属性を括して参照できる仕組みがあります。
 - class で指定された名称は <style> 要素の中ではその名称の前に.(ピリオド) をつけて定義します。定義の範囲は {} の中に書きます。
 - 定義の方法は 属性名:属性値; です。
- フィルタを作用させるにはオブジェクトの属性 filter="url()" の形で引用します (28 行目、30 行目、32 行目)。
- ぼかしの効果が下の画像にどのように影響するかを示すために元の画像を幅の半分だけ横に移動した画像の上にぼかした画像をのせています (34 行目と 35 行目)。
- ぼかしの中心では不透明度が 1 であり、外側に行くにしたがって不透明度が下がっていることがわかります。

問題 6.1 ぼかしの程度や、元の画像の色を変えて見え方がどのように変化するか調べなさい。

問題 6.2 <feGaussianBlur> 要素の属性 stdDeviation にアニメーションを付けてぼけている画像から元の画像に変化するようにしなさい。

図 6.2 はバーゲンのきらめき効果と呼ばれています ([37, 182 ページ図 17.1])。ヘルマン格子の变形です。

形から見てわかるようにヘルマン格子の場合と異なり交差点の位置が黒くなっています。交差点の位置に白い斑点がきらきら見えます。

SVG リスト 6.2: バーゲンのきらめき効果

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>バーゲンのきらめき効果</title>
6  <defs>
7      <filter id="blurFilter" filterUnits="userSpaceOnUse"
8          x="0" y="0" width="540" height="340">
```

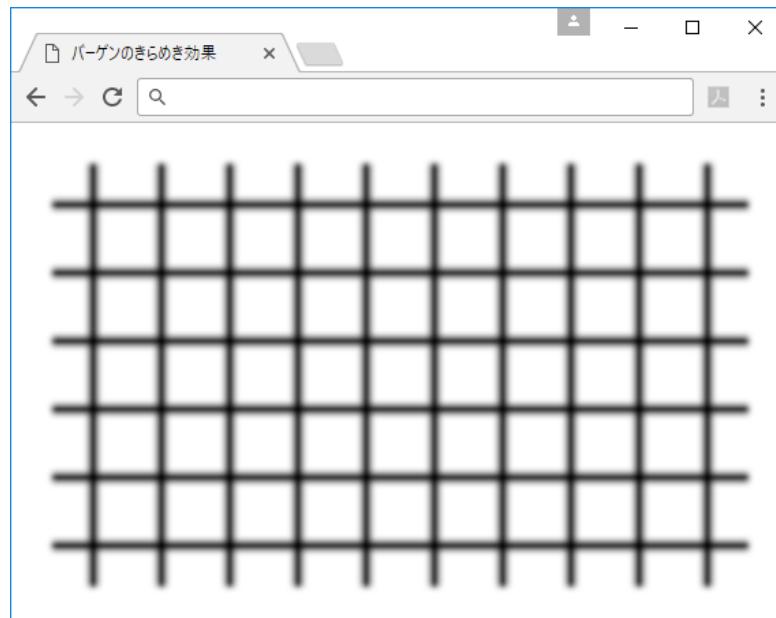


図 6.2: バーゲンのきらめき効果

```

9      <feGaussianBlur stdDeviation="2"/>
10     </filter>
11     <pattern id="pattern" patternUnits="userSpaceOnUse" width="50" height="50">
12       <rect id="Base" x="0" y="0" width="50" height="50"
13         fill="white" stroke-width="8" stroke="black"/>
14     </pattern>
15   </defs>
16   <g transform="translate(10,10)">
17     <rect x="20" y="20" width="510" height="310"
18       filter="url(#blurFilter)" fill="url(#pattern)"/>
19   </g>
20 </svg>

```

- 7行目から10行目でフィルタを定義しています。
- 9行目でガウスぼかしのパラメータを5に設定しています。
- 全体の図は前と同様に縁取りのある正方形からなるパターンで塗りつぶします。そのパターンは12行目から13行目で定義しています。
- 17行目から18行目で定義した長方形の内部を上で定義したパターンで塗りさらにフィルタをかけています。

問題 6.3 図 6.2 のぼかしの程度や、元の画像の色を変えて見え方がどのように変化するか調べなさい。

6.3 影をつける (feOffset フィルタと feMerge フィルタ)

前節の feGaussianBlur フィルタと feOffset フィルタ、feMerge フィルタを組み合わせると与えられた画像に影をつけることができます。

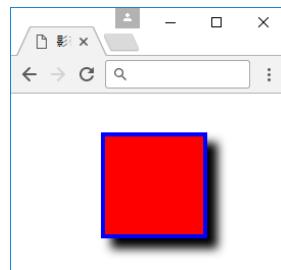


図 6.3: 影をつける

SVG リスト 6.3: 画像に影をつける

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>影をつける</title>
6  <defs>
7      <filter id="DropShadow" filterUnits="userSpaceOnUse"
8          x="-10" y="-10" width="200" height="200">
9          <feGaussianBlur stdDeviation="5" in="SourceAlpha" result="shadow"/>
10         <feOffset dx="10" dy="10" in="shadow" result="shadowMoved"/>
11         <feMerge>
12             <feMergeNode in="shadowMoved"/>
13             <feMergeNode in="SourceGraphic"/>
14         </feMerge>
15     </filter>
16 </defs>
17 <g transform="translate(50,20)">
18     <rect x="40" y="20" width="100" height="100"
19         fill="red" stroke-width="4" stroke="blue" filter="url(#DropShadow)"/>
20 </g>
21 </svg>
```

- 7 行目から 15 行目でフィルタを定義しています。
- 8 行目でフィルタの計算をする範囲を定義しています。
 - 9 行目でぼかしのフィルタを定義しています。対象とする画像は `in` で指定しています。ここでは `SourceAlpha` となっているのでフィルタを適用する画像の不透明度 (アルファチャンネル) を利用します。元の画像では画像のあるところの不透明度は 1 で画像のないところが 0 となっています。この結果の画像は灰色系になります。これ以外に定義されている `in` で利用できる値は表 6.1 を参照してください。

そのようにしてできた画像を後で参照できるようにするために result の属性値で指定しています。ここでは shadow となっています。

- feOffset フィルタは与えられた画像を移動させるフィルタです。移動量は属性 dx と dy で指定します。
 - 属性値 in に shadow が指定されているので9行目における結果が利用されます。
- 11行目から14行目で画像を重ねるフィルタ feMerge フィルタを宣言されています。
- 重ねる画像は feMergeNode フィルタで宣言します。画像は他のフィルタを作用させた結果などを in で指定します。
- ここでは12行目で影を指定し、その上に13行目でフィルタをかける画像 (in における SourceGraphic) を表示させています。
- 18行目から19行目でフィルタをかける元の画像を定義しています。ここでは正方形が指定されています。また、filter でこの画像にかけるフィルタを指定しています。

表 6.1: フィルタを適用させるもとの名称

名称	説明
SourceGraphic	フィルタを適用する画像
SourceAlpha	フィルタを適用する画像の不透明度
BackgroundImage	フィルタの適用範囲にある背景画像
BackgroundAlpha	フィルタの適用範囲にある背景画像のアルファチャンネル
FillPaint	フィルタ要素の fill 属性値
StrokePaint	フィルタ要素の stroke 属性値

6.4 画像を合成する (feImage フィルタと feBlend フィルタ)

feImage フィルタは外部のファイルを取り込んだりすでに定義済みの画像を取り込むためのフィルタです。また、feBlend フィルタは二つの画像を合わせるフィルタです。重ね合わせるときの方法を表 6.2 にまとめました。

このフィルタのモード lighten や darken を用いると加色混合や減色混合によるカラーチャート図を作成できます。この例では内部に定義した画像を feImage フィルタで取り込み feBlend フィルタで重ねています。

SVG リスト 6.4: 加色混合と減色混合によるカラーチャート図

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
```

表 6.2: feBlend フィルタのモード

名称	説明
normal	in の部分を in2 で重なった部分が置き換えられます。アルファチャンネルは両者の積の値になります。
multiply	各ピクセルごとに(値を 0 ~ 1 とみて)積がとられます。
screen	値を反転させたあと multiply をし、その後再び反転させます。
darken	各チャンネルで小さいほうの値をとります。
lighten	各チャンネルで大きいほうの値をとります。

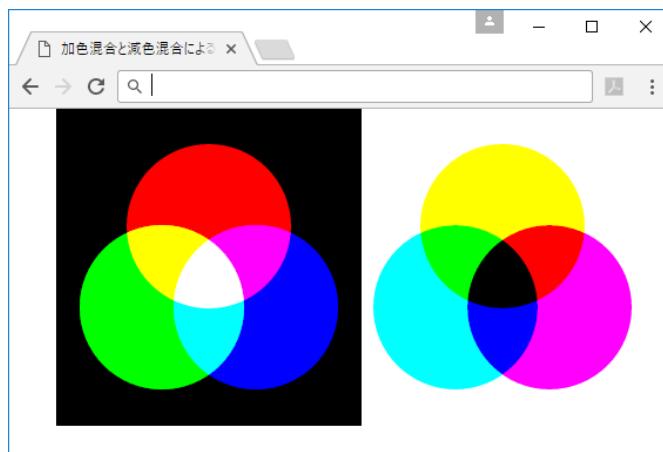


図 6.4: 加色混合と減色混合によるカラーチャート図

```

4      height="100%" width="100%">
5 <title>加色混合と減色混合によるカラーチャート図</title>
6 <defs>
7   <circle id="Circle" cx="100" cy="100" r="70"/>
8   <g id="Red">
9     <rect x="-70" y="0" width="300" height="270"/>
10    <use xlink:href="#Circle" fill="red"/>
11  </g>
12  <use id="Blue" xlink:href="#Circle" fill="Blue"/>
13  <use id="Green" xlink:href="#Circle" fill="lime"/>
14  <use id="Yellow" xlink:href="#Circle" fill="yellow"/>
15  <use id="Magenta" xlink:href="#Circle" fill="magenta"/>
16  <use id="Cyan" xlink:href="#Circle" fill="cyan"/>
17  <filter id="Lighten" filterUnits="userSpaceOnUse"
18    x="-30" y="0" width="1000" height="1000">
19    <feOffset dx="70" dy="0" in="SourceGraphic" result="RedOffset"/>
20    <feImage x="0" y="0" width="500" height="500"
21      xlink:href="#Blue" result="BlueImage"/>

```

```

22   <feOffset dx="110" dy="69" in="BlueImage" result="BlueOffset"/>
23   <feImage x="0" y="0" width="500" height="500"
24     xlink:href="#Green" result="GreenImage"/>
25   <feOffset dx="30" dy="69" in="GreenImage" result="GreenOffset"/>
26   <feBlend in="RedOffset" in2="BlueOffset" mode="lighten" result="RB"/>
27   <feBlend in="RB" in2="GreenOffset" mode="lighten"/>
28 </filter>
29 <filter id="Darken" filterUnits="userSpaceOnUse"
30   x="0" y="0" width="1000" height="1000">
31   <feOffset dx="320" dy="0" in="SourceGraphic" result="YellowOffset"/>
32   <feImage x="0" y="0" width="800" height="500"
33     xlink:href="#Magenta" result="MagentaImage"/>
34   <feOffset dx="360" dy="69" in="MagentaImage" result="MagentaOffset"/>
35   <feImage x="0" y="0" width="800" height="500"
36     xlink:href="#Cyan" result="CyanImage"/>
37   <feOffset dx="280" dy="69" in="CyanImage" result="CyanOffset"/>
38   <feBlend in="YellowOffset" in2="MagentaOffset" mode="darker" result="YM"/>
39   <feBlend in="YM" in2="CyanOffset" mode="darker" />
40 </filter>
41 </defs>
42   <use xlink:href="#Red" filter="url(#Lighten)"/>
43   <use xlink:href="#Yellow" filter="url(#Darken)"/>
44 </svg>
```

- 円の大きさを統一するために7行目で使用する円を定義しています。ここで定義している属性は円の中心位置と半径だけです。
- この円を用いて行目から16行目で fill の値を設定した6つの円を定義しています。名称は色の名前と同じにしています。Opera 12.17 で加色混合の図を正しく表示するため、赤の円だけは正方形の黒の長方形の上に描いています (LineRRedRedE)。¹
- 17行目から28行目で左側の図形を書くためのフィルタを定義しています。
 - 19行目で赤の円を移動するフィルタ (feOffset フィルタ) をかけ、結果を RedOffset としています。元の図形を呼び出すので feOffset フィルタの元画像は SourceGraphic としています。
 - 20行目から21行目では青の円を feImage フィルタを用いて BlueImage という名称で取り込んでいます。
 - 22行目では上で取り込んだ図形を feOffset フィルタで移動しています。
 - 23行目から24行目にかけて緑の円を青と同様の方法で処理しています。
 - この3つの画像を26行目と27行目で順次 feBlend フィルタを用いて合成しています。 mode の値が lighten になっているので重なった点のRGBの値は両者の図形の大きいほうの値(結果としてその点の位置の明るさが増す)になります。したがって、3つの円が重なった部分は白になります。

¹Google Chrome ではこのようなことをしなくても正しく表示されました。

- 29 行目から40 行目も同様のフィルタです。ただ、`feBlend` フィルタの `mode` の値が `darker` になっているので重なった点では二つの画像の RGB 値の小さいほうになります。したがって、3 つの円が重なった部分は黒になります。
- 42 行目から43 行目でこれらのフィルタを用いた赤と黄色の円を書かせています。

6.5 与えられた画像の色を変える

与えられた画像に対し別の色に変えるフィルタとして `feColorMatrix` フィルタと `feComponentTransfer` フィルタがあります。

6.5.1 `feColorMatrix` フィルタ

`feColorMatrix` フィルタの属性 `type` の値として `matrix`, `hueRotate`, `saturate`, `luminanceToAlpha` の4つがあります。これらの変換のパラメータは `values` で指定します。

図 6.5 はこれらのフィルタがどのように作用するかを示したものです。

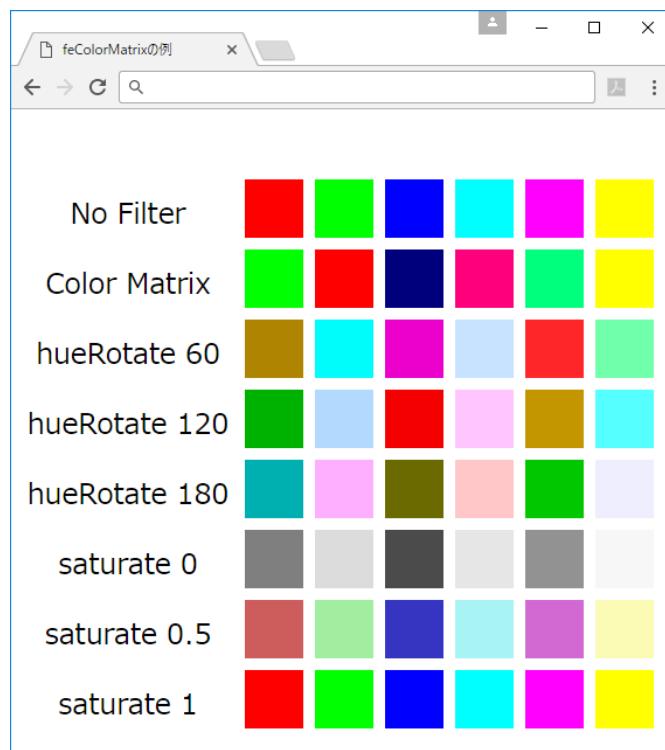


図 6.5: `feColorMatrix` フィルタの例

リスト 6.5 はこの図のリストです。

SVG リスト 6.5: feColorMatrix フィルタの例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <defs>
6      <rect id="Base" width="50" height="50" />
7      <g id="sample" height="100" width="500">
8          <use xlink:href="#Base" x="0" y="0" fill="#FF0000"/>
9          <use xlink:href="#Base" x="60" y="0" fill="#00FF00"/>
10         <use xlink:href="#Base" x="120" y="0" fill="#0000FF"/>
11         <use xlink:href="#Base" x="180" y="0" fill="#00FFFF"/>
12         <use xlink:href="#Base" x="240" y="0" fill="#FF00FF"/>
13         <use xlink:href="#Base" x="300" y="0" fill="#FFFF00"/>
14     </g>
15     <filter id="ChangeColor1" filterUnits="objectBoundingBox"
16             x="0%" y="0%" width="100%" height="100%">
17         <feColorMatrix in="SourceGraphic" type="matrix"
18             values="0 1 0 0 0, 1 0 0 0 0, 0 0 0.2 0 0, 0 0 0 1 0 " />
19     </filter>
20     <filter id="ChangeColor2" filterUnits="objectBoundingBox"
21             x="0%" y="0%" width="100%" height="100%">
22         <feColorMatrix in="SourceGraphic" type="hueRotate" values="60" />
23     </filter>
24     <filter id="ChangeColor3" filterUnits="objectBoundingBox"
25             x="0%" y="0%" width="100%" height="100%">
26         <feColorMatrix in="SourceGraphic" type="hueRotate" values="120" />
27     </filter>
28     <filter id="ChangeColor4" filterUnits="objectBoundingBox"
29             x="0%" y="0%" width="100%" height="100%">
30         <feColorMatrix in="SourceGraphic" type="hueRotate" values="180" />
31     </filter>
32     <filter id="ChangeColor5" filterUnits="objectBoundingBox"
33             x="0%" y="0%" width="100%" height="100%">
34         <feColorMatrix in="SourceGraphic" type="saturate" values="0" />
35     </filter>
36     <filter id="ChangeColor6" filterUnits="objectBoundingBox"
37             x="0%" y="0%" width="100%" height="100%">
38         <feColorMatrix in="SourceGraphic" type="saturate" values="0.5" />
39     </filter>
40     <filter id="ChangeColor7" filterUnits="objectBoundingBox"
41             x="0%" y="0%" width="100%" height="100%">
42         <feColorMatrix in="SourceGraphic" type="saturate" values="1" />
43     </filter>
44 </defs>
45 <g text-anchor="middle" font-size="24px">
46     <text x="100" y="85" dominant-baseline="mathematical">No Filter</text>
47     <use xlink:href="#sample" x="200" y="60"/>
48     <text x="100" y="145" dominant-baseline="mathematical">Color Matrix</text>
49     <use xlink:href="#sample" x="200" y="120" filter="url(#ChangeColor1)"/>
50     <text x="100" y="205" dominant-baseline="mathematical">hueRotate 60</text>
51     <use xlink:href="#sample" x="200" y="180" filter="url(#ChangeColor2)"/>
52     <text x="100" y="265" dominant-baseline="mathematical">hueRotate 120</text>
53     <use xlink:href="#sample" x="200" y="240" filter="url(#ChangeColor3)"/>

```

```

54   <text x="100" y="325" dominant-baseline="mathematical">hueRotate 180</text>
55   <use xlink:href="#sample" x="200" y="300" filter="url(#ChangeColor4)" />
56   <text x="100" y="385" dominant-baseline="mathematical">saturate 0</text>
57   <use xlink:href="#sample" x="200" y="360" filter="url(#ChangeColor5)" />
58   <text x="100" y="445" dominant-baseline="mathematical">saturate 0.5</text>
59   <use xlink:href="#sample" x="200" y="420" filter="url(#ChangeColor6)" />
60   <text x="100" y="505" dominant-baseline="mathematical">saturate 1</text>
61   <use xlink:href="#sample" x="200" y="480" filter="url(#ChangeColor7)" />
62   </g>
63 </svg>

```

- 横に並んだ正方形の離形を6行目で定義しています。
- 横に6つ並んだ正方形は7行目から14行目で定義しています。

各フィルタの解説は次の項で解説します。なお、このフィルタが使用する変換式はW3CのSVG([27])の記述からとりました。

`matrix` 色の構成要素としては3原色(赤、緑、青)のほかに不透明度(アルファチャンネルと呼ばれることもあります)があります。この4つの要素を別の要素に変換するための一般的な方法が`matrix`です。変換前の成分を (R, G, B, α) 変換後の成分を (R', G', B', α') としたとき5行5列の行列により

$$\begin{pmatrix} R' \\ G' \\ B' \\ \alpha' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \\ \alpha \\ 1 \end{pmatrix}$$

と変換します。この行列の成分を $a_{00}, a_{01} \dots$ のように横方向に20個並べたものを`values`に与えます。例6.5では変換の`values`次のようにになっています。

```
0 1 0 0 0, 1 0 0 0 0, 0 0 0.2 0 0, 0 0 0 1 0
```

このリストではわかりやすいように業の終了位置に,を入れてあります。これから変換の行列は次のようになっていることがわかりります。

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

この変換では赤の成分が緑成分に、緑の成分が赤の成分になります。青の成分が0.2倍されています(青が暗くなる)。図6.5の一番上と上記の変換後のRGB値を計算すると次のようになります。

変換前	#FF0000	#00FF00	#0000FF	#00FFFF	#FF00FF	#FFFF00
変換後	#00FF00	#FF0000	#000033	#FF0033	#00FF33	#FFFF00

図6.5にある上から2行目の色がこのようになっていることを確認してください。

`hueRotate hue` とは色相のことです。与えられた色に対し、色相が 0° から 360° の値が決められます。たとえば与えられた色相は赤が 0° 、黄色が 60° などとなっています。与えられた色相の値に与えられた分だけ角度を加えた色相に変化させるフィルタです。色相についての解説² を参考にしてください。

これを使用した例が図 6.5 の 3,4,5 行目です。リスト 6.5 の 20 行目から 31 行目がこのフィルタの記述部分です。

```

20   <filter id="ChangeColor2" filterUnits="objectBoundingBox"
21     x="0%" y="0%" width="100%" height="100%">
22     <feColorMatrix in="SourceGraphic" type="hueRotate" values="60" />
23   </filter>
24   <filter id="ChangeColor3" filterUnits="objectBoundingBox"
25     x="0%" y="0%" width="100%" height="100%">
26     <feColorMatrix in="SourceGraphic" type="hueRotate" values="120" />
27   </filter>
28   <filter id="ChangeColor4" filterUnits="objectBoundingBox"
29     x="0%" y="0%" width="100%" height="100%">
30     <feColorMatrix in="SourceGraphic" type="hueRotate" values="180" />
31   </filter>
```

このフィルタの変換式は次のようにになっています。

$$\begin{pmatrix} R' \\ G' \\ B' \\ \alpha' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 \\ a_{20} & a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \\ \alpha \\ 1 \end{pmatrix}$$

ここで θ を `hueRotate` で与えられた値とすると左上の部分の行列は次の式で与えられます。

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} +0.213 & +0.715 & +0.072 \\ +0.213 & +0.715 & +0.072 \\ +0.213 & +0.715 & +0.072 \end{pmatrix} + \cos \theta \begin{pmatrix} +0.787 & -0.715 & -0.072 \\ -0.213 & +0.285 & -0.072 \\ -0.213 & -0.715 & +0.928 \end{pmatrix} + \sin \theta \begin{pmatrix} -0.213 & -0.715 & +0.928 \\ +0.143 & +0.140 & -0.283 \\ -0.787 & +0.715 & +0.072 \end{pmatrix}$$

図 6.6 は赤、緑、青の 3 原色のそれぞれに 30° ごとのフィルタをかけて基準の色が最終的に元来の位置にくるように回転させた色相環の図です。

この図を見ると色相環が完全に作成できていないことがわかります。位相を変換したとき、元の画像の明度によって他のところの色に影響が及んでいることがわかります。

²<http://www.microsoft.com/japan/msdn/columns/hess/hess08142000.asp>



図 6.6: 色相回転の例

この図を描くための SVG のリストは次のようになっています(赤のみ)。

SVG リスト 6.6: 色相環

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3   xmlns:xlink="http://www.w3.org/1999/xlink"
4   height="100%" width="100%">
5   <title>色相環</title>
6   <defs>
7     <filter id="HRotate30" filterUnits="objectBoundingBox"
8       x="0%" y="0%" width="100%" height="100%">
9       <feColorMatrix in="SourceGraphic" type="hueRotate" values="30" />
10    </filter>
11    <filter id="HRotate60" filterUnits="objectBoundingBox"
12      x="0%" y="0%" width="100%" height="100%">
13      <feColorMatrix in="SourceGraphic" type="hueRotate" values="60" />
14    </filter>
15    <filter id="HRotate90" filterUnits="objectBoundingBox"
16      x="0%" y="0%" width="100%" height="100%">
17      <feColorMatrix in="SourceGraphic" type="hueRotate" values="90" />
18    </filter>
19    <filter id="HRotate120" filterUnits="objectBoundingBox"
20      x="0%" y="0%" width="100%" height="100%">
21      <feColorMatrix in="SourceGraphic" type="hueRotate" values="120" />
22    </filter>
23    <filter id="HRotate150" filterUnits="objectBoundingBox"
24      x="0%" y="0%" width="100%" height="100%">
25      <feColorMatrix in="SourceGraphic" type="hueRotate" values="150" />
26    </filter>
27    <filter id="HRotate180" filterUnits="objectBoundingBox"
28      x="0%" y="0%" width="100%" height="100%">
```

```
29      <feColorMatrix in="SourceGraphic" type="hueRotate" values="180" />
30  </filter>
31  <filter id="HRotate210" filterUnits="objectBoundingBox"
32      x="0%" y="0%" width="100%" height="100%">
33      <feColorMatrix in="SourceGraphic" type="hueRotate" values="210" />
34  </filter>
35  <filter id="HRotate240" filterUnits="objectBoundingBox"
36      x="0%" y="0%" width="100%" height="100%">
37      <feColorMatrix in="SourceGraphic" type="hueRotate" values="240" />
38  </filter>
39  <filter id="HRotate270" filterUnits="objectBoundingBox"
40      x="0%" y="0%" width="100%" height="100%">
41      <feColorMatrix in="SourceGraphic" type="hueRotate" values="270" />
42  </filter>
43  <filter id="HRotate300" filterUnits="objectBoundingBox"
44      x="0%" y="0%" width="100%" height="100%">
45      <feColorMatrix in="SourceGraphic" type="hueRotate" values="300" />
46  </filter>
47  <filter id="HRotate330" filterUnits="objectBoundingBox"
48      x="0%" y="0%" width="100%" height="100%">
49      <feColorMatrix in="SourceGraphic" type="hueRotate" values="330" />
50  </filter>
51  <circle id="Base" cx="80" cy="0" r="20"/>
52  <g id="Fundamental">
53      <g transform="rotate(0)">
54          <use xlink:href="#Base" />
55      </g>
56      <g transform="rotate(30)">
57          <use xlink:href="#Base" filter="url(#HRotate30)"/>
58      </g>
59      <g transform="rotate(60)">
60          <use xlink:href="#Base" filter="url(#HRotate60)"/>
61      </g>
62      <g transform="rotate(90)">
63          <use xlink:href="#Base" filter="url(#HRotate90)"/>
64      </g>
65      <g transform="rotate(120)">
66          <use xlink:href="#Base" filter="url(#HRotate120)"/>
67      </g>
68      <g transform="rotate(150)">
69          <use xlink:href="#Base" filter="url(#HRotate150)"/>
70      </g>
71      <g transform="rotate(180)">
72          <use xlink:href="#Base" filter="url(#HRotate180)"/>
73      </g>
74      <g transform="rotate(210)">
75          <use xlink:href="#Base" filter="url(#HRotate210)"/>
76      </g>
77      <g transform="rotate(240)">
78          <use xlink:href="#Base" filter="url(#HRotate240)"/>
79      </g>
80      <g transform="rotate(270)">
81          <use xlink:href="#Base" filter="url(#HRotate270)"/>
82  </g>
```

```

83   <g transform="rotate(300)">
84     <use xlink:href="#Base" filter="url(#HRotate300)"/>
85   </g>
86   <g transform="rotate(330)">
87     <use xlink:href="#Base" filter="url(#HRotate330)"/>
88   </g>
89   </g>
90 </defs>
91 <g transform="translate(150,150)">
92   <g transform="rotate(-90)" fill="red">
93     <use xlink:href="#Fundamental"/>
94   </g>
95   <text font-size="24px" text-anchor="middle" y="150">赤基準</text>
96 </g>
97 </svg>

```

- 7 行目から 50 行目で各フィルタを定義しています。全部で 11 種類あります。
- 51 行目で塗られる円を定義しています。
- 行目から 89 行目でこの円を 12 個並べた図を定義しています。
- 92 行目から 94 行目で実際の図を描いています。ここで `fill` の値を `red` に設定し、赤が一番上に来るよう画像全体を回転しています。
- 残りの色の部分も同様に回転させ、色を指定しています。

問題 6.4 黄色を基準色にして同様の図形を作成し、出来上がったものに対して色がどのようにになっているか調べなさい。

`saturate` `saturate` とは彩度のことです。`s` をこのフィルタのパラメータ `values` で与えられた値とするとこの変換は次の式で与えられます。

$$\begin{pmatrix} R' \\ G' \\ B' \\ \alpha' \\ 1 \end{pmatrix} = \begin{pmatrix} 0.213 + 0.787s & 0.715 - 0.715s & 0.072 - 0.072s & 0 & 0 \\ 0.213 - 0.213s & 0.715 + 0.285s & 0.072 - 0.072s & 0 & 0 \\ 0.213 - 0.213s & 0.715 - 0.715s & 0.072 + 0.928s & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \\ \alpha \\ 1 \end{pmatrix}$$

この式からわかるように $s = 0$ のときは `R,G,B` のそれぞれの値が他の色に同じ値が設定されるので変換後はグレイスケールになります (SVG リスト 6.5 の行目から行目のフィルタ)。

また、 $s = 1$ のときは変換行列が単位行列になるので図形の色はまったく変化しません (SVG リスト 6.5 の 44 行目から 47 行目のフィルタ)。

`luminanceToAlpha` 輝度 (luminance) を不透明度に変換するフィルタです。次の式で定義されます。単独で利用することはないでしょう。

$$\begin{pmatrix} R' \\ G' \\ B' \\ \alpha' \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.2125 & 0.7154 & 0.0721 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \\ \alpha \\ 1 \end{pmatrix}$$

6.5.2 feComponentTransfer フィルタ

`feComponentTransfer` フィルタは各色のチャンネルを直接変換する手段を与えます。この要素のなかには次の要素を書くことができます。

`feFuncR` 要素, `feFuncG` 要素, `feFuncB` 要素, `feFuncA` 要素

これらの要素には `type` 属性の値により次の属性を指定できます。なお、`C` は与えられたチャンネルの数値です。なお、属性値の `tableValues` は空白またはコンマで区切られた数字の列です。こ

表 6.3: `<feComponentTransfer>` 要素の `type` の属性と属性値の種類

type の値	とりうる属性値	意味
<code>linear</code>	<code>slope,intercept</code>	$slope \times C + intercept$
<code>gamma</code>	<code>amplitude,exponent,offset</code>	$amplitude \times C^{exponent} + offset$
<code>identity</code>	なし	<code>C</code>
<code>table</code>	<code>tableValues</code>	与えられた数値の間を直線で補間する。横軸は等間隔
<code>discrete</code>	<code>tableValues</code>	与えられた数値の階段関数にする。

のフィルタを使用した例は [27] で見ることができます。

6.6 画像を单一色で塗りつぶす (feFlood フィルタ)

`feFlood` フィルタは画像を `flood-color` で塗りつぶします。また、`flood-opacity` で不透明度も指定できます。与えられた画像に背景をつけたいときに用いられるようです。

SVG リスト 6.7: `feFlood` フィルタの例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3    xmlns:xlink="http://www.w3.org/1999/xlink"
4    height="100%" width="100%">
5  <title>背景をつける</title>

```

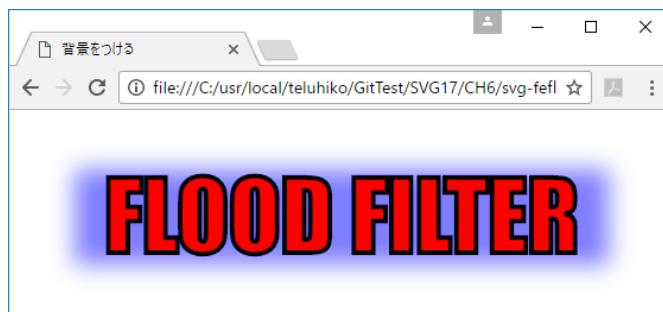


図 6.7: 背景をつける

```

6 <defs>
7   <filter id="Flood" filterUnits="userSpaceOnUse" >
8     <feFlood flood-color="blue" flood-opacity="0.5"
9       x="25" y="30" width="450" height="80"
10      in="SourceGraphic" result="flood"/>
11     <feGaussianBlur in="flood" result="floodblur" stdDeviation="10"
12       x="0" y="0" width="600" height="180"/>
13   </feMerge>
14     <feMergeNode in="floodblur"/>
15     <feMergeNode in="SourceGraphic"/>
16   </feMerge>
17 </filter>
18 </defs>
19 <style type="text/css">
20   .textStyle {font-size:80px; text-anchor:middle;
21     font-family:Impact;stroke:black; stroke-width:4; fill:red;}
22 </style>
23 <g transform="translate(30,20)" >
24   <text x="250" y="100"
25     class="textStyle" filter="url(#Flood)">FLOOD FILTER</text>
26   </g>
27 </svg>

```

- 8 行目から 10 行目で元の画像の範囲を含む長方形の範囲を `feFlood` フィルタを用いて青で塗りつぶしています。
- 塗りつぶした画像に `feGaussianBlur` フィルタをかけて周囲をぼかします (11 行目から 12 行目)
- もとの画像と上の画像を `feMerge` フィルタで重ね合わせます。
- なお、この例では左側に表示するテキストのフォントの大きさなどを CSS を用いて定義しています (19 行目から 22 行目)。ここで定義された名称は要素のほうでは `class` で指定します (25 行目)。

6.7 複雑な画像の合成(feComposite フィルタ)

feMerge フィルタのように二つのソースから新しい画像を作成する一般的なものとして feComposite フィルタがあります。

表 6.4: <feComposite> 要素の operator の属性

operator の値	意味
over	in2 で指定された画像の上に in で指定した画像を重ねる (feMerge フィルタと同じ)。
in	in2 に含まれる in の部分
out	in2 に含まれない in の部分
atop	in2 に含まれる in の部分と in に含まれない in2 の部分
xor	in2 に含まれない in の部分と in に含まれない in2 の部分
arithmetic	各ピクセルごとに in の値を A, in2 の値を B としたとき、 k1, k2, k3, k4 で指定される値に対して $k1 \times A + k2 \times B + k3 \times A + k4 \times B$ で計算される値。

これらの画像の具体的な処理は下図のようになります。ここでは赤と青の円がそれぞれ in と in2 になっています。

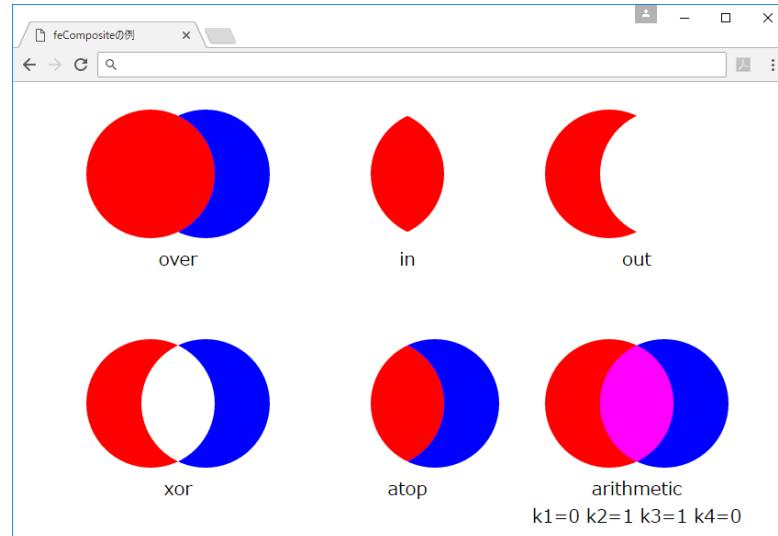


図 6.8: feComposite 要素の例

SVG リスト 6.8: feComposite 要素の例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"

```

```
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5 <title>feComposite の例</title>
6 <defs>
7   <circle id="Circle" cx="100" cy="100" r="70"/>
8   <use id="IN" x="50" xlink:href="#Circle" fill="red"/>
9   <use id="IN2" x="110" xlink:href="#Circle" fill="Blue"/>
10  <filter id="OpOVER" filterUnits="userSpaceOnUse">
11    <feImage x="0" y="0" width="400" height="200"
12      xlink:href="#IN2" result="IN2Image"/>
13    <feComposite in="SourceGraphic" in2="IN2Image" operator="over"/>
14  </filter>
15  <filter id="OpXOR" filterUnits="userSpaceOnUse"
16    x="0" y="0" width="300" height="200">
17    <feImage x="0" y="0" width="400" height="200"
18      xlink:href="#IN2" result="IN2Image"/>
19    <feComposite in="SourceGraphic" in2="IN2Image" operator="xor"/>
20  </filter>
21  <filter id="OpIN" filterUnits="userSpaceOnUse"
22    x="0" y="0" width="300" height="200">
23    <feImage x="0" y="0" width="300" height="200"
24      xlink:href="#IN2" result="IN2Image"/>
25    <feComposite in="SourceGraphic" in2="IN2Image" operator="in"/>
26  </filter>
27  <filter id="OpOUT" filterUnits="userSpaceOnUse"
28    x="0" y="0" width="300" height="200">
29    <feImage x="0" y="0" width="300" height="200"
30      xlink:href="#IN2" result="IN2Image"/>
31    <feComposite in="SourceGraphic" in2="IN2Image" operator="out"/>
32  </filter>
33  <filter id="OpATOP" filterUnits="userSpaceOnUse"
34    x="0" y="0" width="300" height="200">
35    <feImage x="0" y="0" width="300" height="200"
36      xlink:href="#IN2" result="IN2Image"/>
37    <feComposite in="SourceGraphic" in2="IN2Image" operator="atop"/>
38  </filter>
39  <filter id="OpArith" filterUnits="userSpaceOnUse"
40    x="0" y="0" width="300" height="200">
41    <feImage x="0" y="0" width="300" height="200"
42      xlink:href="#IN2" result="IN2Image"/>
43    <feComposite in="SourceGraphic" in2="IN2Image" operator="arithmetic"
44      k1="0" k2="1" k3="1" k4="0"/>
45  </filter>
46 </defs>
47 <style type="text/css">
48 .Text {text-anchor:middle; font-size:20px; }
49 </style>
50   <use xlink:href="#IN" filter="url(#OpOVER)"/>
51   <text class="Text" x="180" y="200" >over</text>
52 <g transform="translate(250,0)">
53   <use xlink:href="#IN" filter="url(#OpIN)"/>
54   <text class="Text" x="180" y="200">in</text>
55 </g>
56 <g transform="translate(500,0)">
```

```

57      <use xlink:href="#IN" filter="url(#OpOUT)"/>
58      <text class="Text" x="180" y="200">out</text>
59    </g>
60    <g transform="translate(0,250)">
61      <use xlink:href="#IN" filter="url(#OpXOR)"/>
62      <text class="Text" x="180" y="200">xor</text>
63    </g>
64    <g transform="translate(250,250)">
65      <use xlink:href="#IN" filter="url(#OpATOP)"/>
66      <text class="Text" x="180" y="200">atop</text>
67    </g>
68    <g transform="translate(500, 250)">
69      <use xlink:href="#IN" filter="url(#OpArith)"/>
70      <text class="Text" x="180" y="200">arithmetic</text>
71      <text class="Text" x="180" y="230">
72        k1=0 k2=1 k3=1 k4=0</text>
73    </g>
74  </svg>

```

- 7行目で基準となる円の大きさだけを定義しています。
- 7行目で定義した円を用いて赤と青に塗られた円をそれぞれ8行目と9行目で定義しています。
- 10行目から14行目でoperatorがoverのフィルタを定義しています(idはOpOVER)。
- 以下順にフィルタが定義されています。
 - operatorがxorのフィルタ(15行目から20行目)
 - operatorがinのフィルタ(21行目から26行目)
 - operatorがoutのフィルタ(27行目から32行目)
 - operatorがatopのフィルタ(33行目から38行目)
 - operatorがarithmeticのフィルタ(39行目から45行目)

最後のフィルタを除いてoperatorの値が異なるだけです。

- arithmeticではそれぞれのチャンネルお値を加えるパラメータが指定されています。この場合、塗られている色が赤と青なのでfeBlendフィルタのlightenと同じ効果が得られています(図6.4参照)。

問題6.5 加色混合のカラーチャート図をfeComposite要素で作成しなさい。また、feComposite要素を用いて減色混合のカラーチャート図を作成できるか検討しなさい。

6.8 光を当てる

画像に光を当て3Dのように見せる効果を出すフィルタとしてfeDiffuseLighting要素とfeSpecularLighting要素があります。これらの効果を使うと画像にこれらのフィルタが使用する光源の種類により効果

は異なります。次の図 6.9 を見てください³。なお、この図でもそうですが `feSpecularLighting` 要素は単独で使われるものではなく `bump map` として利用され、他の画像と合成されて使用します。

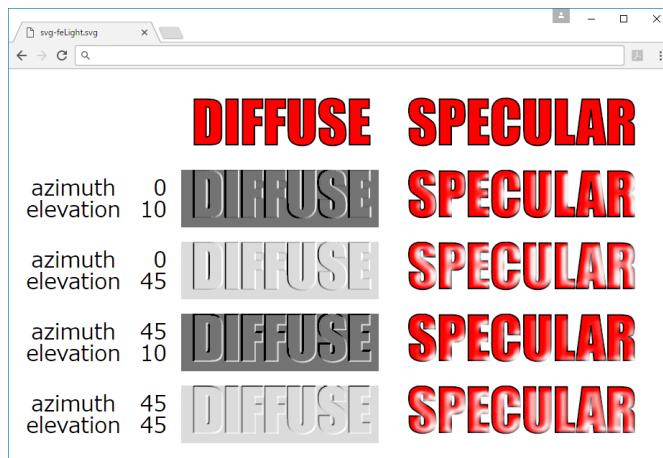


図 6.9: 光を当てる

6.8.1 光源の種類

Distant Light

Distant Light とは非常に強い遠方の光源のことです。`feDistantLight` 要素で定義します。属性として水平方向からの角度(右側から図る)`azimuth`と画面からの角度(水平方向から図る)`elevation`があります。人間の目の習性からすると左上から光が当たるようにすると飛び出したように見えます。図は散乱光を用いた Distant Light の使用例です。

Point Light

Point Light(点光源) は空間の一点から放射される光源のことです。`fePointLight` 要素で定義します。点光源の位置を示す座標の属性 `x,y,z` があります。当然のことながら `z` が大きくなれば光源が遠くなるので効果が少なくなります。

Spot Light

Spot Light(スポットライト) は点光源と似ていますが舞台でのスポットライトのように光が広がる範囲を制限する `limitingConeAngle` やスポットライトが当たる中心の位置を指定する `pointsAtX`,

³ この図を作成するに当たり Adobe 社の SVG ゾーン <http://www.adobe.com/jp/svg/basics/getstarted.html>⁴ の例にあるフィルタを参考にしました。なお、このページのフィルタで `lightColor` という属性が使われているようですがこの属性は正しくは `light-color` です。

pointsAtY, pointsAtZ の属性があります。

6.8.2 feDiffuseLighting フィルタ

次のリストは図 6.9 の一部を描くためのものです。

SVG リスト 6.9: Distant Light による散乱光の例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
3      "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
4  <svg xmlns="http://www.w3.org/2000/svg"
5      xmlns:xlink="http://www.w3.org/1999/xlink"
6      height="100%" width="100%">
7  <defs>
8      <text id="Text" class="textStyle" x="15">Lighting FILTER</text>
9      <filter id="Lighting3" filterUnits="userSpaceOnUse"
10         x="0" y="0" width="500" height="80">
11         <feDiffuseLighting in="SourceGraphic" lighting-color="white">
12             <feDistantLight azimuth="45" elevation="45"/>
13         </feDiffuseLighting>
14     </filter>
15 </defs>
16 <style type="text/css">
17 .textStyle {font-size:80px; text-anchor:start; dominant-baseline:hanging;
18             font-family:Impact;stroke:black; stroke-width:2; fill:red;}
19 .InfoStyle {font-size:30px; text-anchor:middle; dominant-baseline:mathematical;}
20 .InfoNum {font-size:30px; text-anchor:end; dominant-baseline:mathematical;}
21 </style>
22 <g transform="translate(240,40)" >
23     <use x="0" xlink:href="#Text" />
24 </g>
25 <g transform="translate(240,140)">
26     <text class="InfoStyle" x="-150" y="20">azimuth</text>
27     <text class="InfoNum"   x="-20" y="20">45</text>
28     <text class="InfoStyle" x="-150" y="50">elevation</text>
29     <text class="InfoNum"   x="-20" y="50">45</text>
30     <use xlink:href="#Text" filter="url(#Lighting3)"/>
31 </g>
32 </svg>
```

6.8.3 feSpecularLighting フィルタ

specular とは鏡面の意味です。このフィルタはもとあの画像のアルファチャンネルを bump map(凹凸をつけるための絵)として利用するものです。このフィルタで利用できる属性は surfaceScale(アルファチャンネルの値の倍数)、specularConstant(全体の強度)、specularExponent などがあります。入力の値のアルファ値をそのピクセルの高さとみなし、光が来る方向と曲面の傾きに応じて画像が構成されます。最終的には feComposite 要素を用いて与えられた画像の上に貼り付けて立体感を出すことができます。パラメータの詳しい説明は [27] を見てください。

次のリストは図 6.9 の一部を描くためのものです。

SVG リスト 6.10: Distant Light による反射光の例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
3      "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">
4  <svg xmlns="http://www.w3.org/2000/svg"
5      xmlns:xlink="http://www.w3.org/1999/xlink"
6      height="100%" width="100%">
7  <defs>
8      <text id="TextS" class="textStyle" x="15">SPECULAR</text>
9      <filter id="LightingS2" filterUnits="userSpaceOnUse"
10         x="0" y="100" width="360" height="80">
11         <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur2"/>
12         <feSpecularLight in="blur2" surfaceScale="5" specularConstant="0.9"
13             specularExponent="20" lighting-color="white" result="specularOut2">
14             <feDistantLight azimuth="45" elevation="10"/>
15         </feSpecularLight>
16         <feComposite in="SourceGraphic" in2="specularOut2" operator="arithmetic"
17             k1="0" k2="1" k3="1" k4="0"/>
18     </filter>
19 </defs>
20 <style type="text/css">
21 .textStyle {font-size:80px; text-anchor:start;
22     font-family:Impact;stroke:black; stroke-width:2; fill:red;}
23 .InfoStyle {font-size:30px; text-anchor:middle; }
24 .InfoNum {font-size:30px; text-anchor:end; }
25 </style>
26 <g transform="translate(240,90)" >
27     <use x="0" xlink:href="#TextS" />
28 </g>
29 <g transform="translate(240,140)">
30     <text class="InfoStyle" x="-150" y="20">azimuth</text>
31     <text class="InfoNum"    x="-20" y="20">45</text>
32     <text class="InfoStyle" x="-150" y="50">elevation</text>
33     <text class="InfoNum"    x="-20" y="50">10</text>
34     <use x="0" xlink:href="#TextS" filter="url(#LightingS2)"/>
35 </g>
36 </svg>
```

- 元の画像は 8 行目で定義されているテキストです (27 行目で参照しています)。この画像は全領域でアルファ値が 1 です。
- この画像のアルファ値に (`in="SourceAlpha"`) アルファ値を変化させるために `feGaussianBlur` フィルタをかけます (11 行目)。結果の画像に `blur2` という名称をつけています。
- `blur2` に右下上方 45 度の方向から白色光をあてて `feSpecularLightng` フィルタをかけます (12 行目から 15 行目)。
- ここで得られた画像を元の絵と加えて (`feComposite` フィルタの `arithmetic`) 最終的な画像を作成しています。

6.9 feTurbulence フィルタ

このフィルタは人工的なテクスチャを生成します。例を見てください。

表 6.5: feTurbulence フィルタの属性一覧

属性名	解説
type	turbulence または fractalNoise が利用可能
baseFrequency	値が大きいほど色の変化が大きくなる。正の数でなければならない。
numOctaves	ノイズ関数の数。デフォルトは 1
seed	このフィルタが使う乱数の初期値。デフォルトは 1

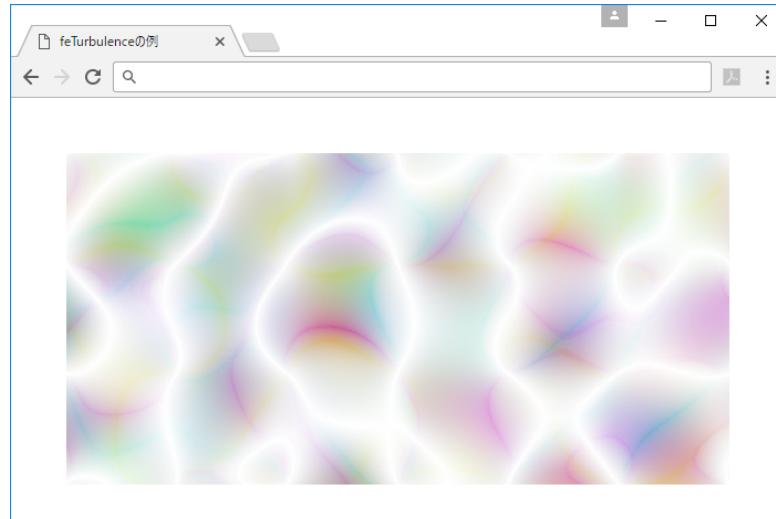


図 6.10: feTurbulence フィルタの例

SVG リスト 6.11: feTurbulence フィルタの例

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink"
4      height="100%" width="100%">
5  <title>feTurbulence の例</title>
6  <defs>
7      <filter id="Filter" filterUnits="objectBoundingBox"
8          x="0%" y="0%" width="100%" height="100%">
9          <feTurbulence type="turbulence" id="FilterfeTurbulence"
10             baseFrequency="0.01" numOctaves="1"/>
11      </filter>

```

```
12   </defs>
13   <g transform="translate(50,50)">
14     <rect x="0" y="0" width="600" height="300" id="Basic" filter="url(#Filter)"/>
15   </g>
16 </svg>
```

問題 6.6 このフィルタのパラメータ `baseFrequency` や `numOctaves` をいろいろ変えてどのような図形が作成できるか確かめなさい。

第7章 JavaScriptを利用したSVG図形の生成

この章では SVG の要素をプログラムから制御する方法を学びます。これを利用すると HTML 上で入力されたデータやユーザーの動作に反応して変化する SVG の図形を作成したり、このテキストの表紙にあるような複雑な図形をプログラムで描くことができるようになります。

なお、この章におけるプログラミングスタイルが章の始めと終わりの方で異なっています。JavaScript でより良いプログラミングをするためには気を付けなければならないことがあります。当初は理解しやすくするために簡単な記述をしています。本格的なプログラムを組むために注意する点については後の方で解説をしますので、そのプログラミングスタイルに慣れるようにしてください。

7.1 インタラクティブなSVGを作成するための準備

マウスのクリックなどに反応して図形を変えることを実現するためにはプログラミングと SVG の図形がどのように内部で扱われているかを知っておく必要があります。

利用できるプログラミング言語は JavaScript です。SVG の図形 (XML の構造) を内部で管理するためには DOM の概念が必要です。

7.1.1 JavaScript

JavaScript とは

JavaScript は HTML 文書の中に書くことができるスクリプト言語です¹。JavaScript については入門書がたくさんありますので歴史などについてはそちらを参考にしてください。一通り JavaScript のプログラムが書けるようになったら付録 [8] で計算機言語としても一度復習するとよいでしょう。

JavaScript で書かれたプログラムは HTML 文書や SVG 文書の中にあり、その処理はブラウザで行われます。文書が読み込まれるときに文書自体を作成するために `document.write` という方法でするのが一般に行われていますが、ここでの解説ではマウスのクリックなどで表示を変えたりする DOM(Document Object Model) の技法を用いて同等のことを行なうことになります。DOM の技法を取り扱う JavaScript のライブラリーもありますが、基本的なことを理解するためにこれらのライブラリーはこのテキストでは使用しません。この技法は最近話題となっている Ajax を利用するためにも欠かせないものです。

JavaScript はインタープリター型の言語です。言語を解釈しながら実行していくまで途中まで正しいプログラムを書いてあるとそこまでは実行され、その後エラーが出て停止することもあ

¹JavaScript が使えるものとしてはこのほかに pdf や svg もあります。決して HTML 専用ではありません。

ります。最近のブラウザではJavaScriptのデバッガーが付いていて、ブラウザ自身がWebアプリケーションの開発のプラットフォームになっています。プログラムが動かないときはこの機能を利用するのが一般的です。

Chromeでこの機能を使うにはアドレスバーの右端にある「⋮」をクリックし、「その他のツール」⇒「デベロッパーツール」を選択します。また、一般的なブラウザではショートカットキーとして「Cntl+Shift+i」または「F12」が利用できます。

具体的な使用例は次節以降で紹介します。

7.1.2 DOM(Document Object Model)

W3CのDOMに関するページ[22]に次のような記述があります。

The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.

Document Object Modelはプログラムやスクリプトが文書の内容、構造およびスタイルに動的にアクセスしたりアップデートするようにするプラットフォームや言語に対し中立なインターフェイスである。文書はさらに処理され、その結果は現在のページへと反映させることができます。

より具体的にいえばDOMはXMLデータをツリー状に表したオブジェクトで、このデータ構造を操作する統一的な手段がW3Cの規格として制定されています。

DOMを利用するメリットして次のことが挙げられます。

- 最近のブラウザはDOMを標準でサポートしています。したがって、ブラウザによりコードを書き分ける必要がなくなります。
- DOMを操作する手段(API – Application Program Interface)が定義されています。
- APIをJavaScriptなどの言語から利用してDOMのなかにあるデータを追加や削除、変更をすることでDOM文書が変更できます。

SVG文書もHTML文書とともにDOMとして取り扱うことができますので、ここで紹介する方法はSVG文書に限らずHTML文書に対しても応用できます。

DOMは文書をツリー状のデータ構造で管理します。

- ノードと呼ばれるものの集まりでこれらの間に上位、下位の関係で結んだものです。
- あるノードから見て下位にあるノードを子ノード、上位にあるノードを親ノードと呼びます。
- 子ノードから見ると親ノードはただひとつしかありません。

- 最上位にあるノードをルートノードと呼びます。
- ノードの種類としては要素を表す要素ノードとテキストを表すテキストノードが重要です。

図??は Opera 12.17 で

ツール → 詳細ツール → Opera Dragonfly

の順にメニューを開いた後の画面です。

- 画面の上部には SVG の画像が表示されていて、背景が水色になっています。
- 画面の下部にはいくつかのタブがあり、一番左のタブが選択されています。
 - このタブ名は「ドキュメント」です。ここに DOM の構造が示されます。
 - ここではこの画像の SVG 要素が示され、その前に「+」が付いています。
 - これは SVG 要素の下に子要素があることを示し、それらが畳まれている（表示されていない）ことを示しています。エクスプローラなどのフォルダの表示と似ていることに注意してください。

図??は図??から中央の折れ線をクリックした状態です。

Opera の開発ツールで DOM ツリーを表示する (2)opera-dragonfly2

クリックした要素に対応した要素のところまで DOM ツリーが展開され、その要素のところの背景が水色になっています。

この画面で<polyline> 要素の属性 width の属性値を 20 から 40 に変更した画面が図??です。

Opera の開発ツールで DOM ツリーを表示する (3)opera-dragonfly3

入力の過程でその値がすぐに反映されるのがわかります。

このほかのタブは左から順に次のような機能を持ちます。

- スクリプト
JavaScript のプログラムが表示されます。エラーが起きた行がここでただちに分かるほか、通常のデバッガーの機能（ブレークポイントの設定など）が用意されています。
- ネットワーク
通常 HTML 文書文書は自分自身のファイル以外にも画像ファイルなど多数のファイルにより構成されています。これらの構成されるファイルの読み込みのタイミングが示されます。これによりどのファイルの読み込みに時間がかかっているかなどパフォーマンスの向上に役立ちます。
- リソース
読み込まれたファイルの関係を示します。

- **ストレージ**

ブラウザのページ間でのデータのやり取りの情報を示します。ここにはさらに「Cookie」「ローカルストレージ」「セッションストレージ」などのタブが表示されています。あるHTML文書のデータを別のHTML文書に渡すためには従来は「Cookie」を使用するしか方法がありませんでしたが、HTML5では新たに「Webストレージ」という機能が追加されました。Webストレージには「ローカルストレージ」と「セッションストレージ」の2種類があります。

- **エラー**

JavaScriptの実行時などのエラーがまとめて表示されます。

- **コンソール(一番右)**

オンラインでJavaScriptの対話型の実行ができる(停止した場所での変数の値の確認など)ほか、プログラムからの出力を表示させることができます。

問題7.1 次の事項について調べなさい。

1. 今までに作成したSVG文書をOpera12.17で表示し、開発者ツールで図形の属性値を変化させることで表示が変わること
2. SVG文書のアニメーションの属性値を変えることができるかどうか
3. HTML文書でも同様のことができること
4. 上で説明していないタブの機能を確認すること
5. 上にあげたブラウザで同様の機能があること

7.1.3 DOMのメソッドとプロパティ

DOMの構造や属性値の操作をプログラムから可能にするためにDOMでは表7.1や7.2にあるメソッドやプロパティが規定されています。これらの手段を用いてDOMをサポートする文書にアクセスができます。

メソッドとはそのオブジェクトに対する操作を意味し、関数の形で記述します。プロパティはそのオブジェクトが持つ性質で代入により参照したり書き直したりできます。これらのメソッドやプロパティの具体的な使用方法はこの後でSVG文書を操作することで紹介します。

なお、ここでのメソッドやプロパティはDOM文書で使用可能なものですが、したがって、最近のブラウザはすべてDOMをサポートするので同様の方法でHTML文書文書も部分的に書き直すことが可能です。

表7.1: DOMのメソッド

メソッド名	使用可能要素	説明
getElementById(id)	document	属性idの値が引数idである要素を得る。

次ページへ続く

表 7.1: DOM のメソッド (続き)

メソッド名	使用可能要素	説明
getElementsByTagName(Name)	対象要素	対象要素の子要素で要素名が Name であるもののリストを得る。
getElementsByClassName(Name)	対象要素	属性 class の値が Name である要素のリストを得る。
getElementsByName(Name)	document	属性 name が Name である要素のリストを得る。
querySelector(selectors)	対象要素	selectors で指定された CSS のセレクタに該当する一番初めの要素を得る。
querySelectorAll(selectors)	対象要素	selectors で指定された CSS のセレクタに該当する要素のリストを得る。
getAttribute(Attrib)	対象要素	対象要素の属性 Attrib の値を読み出す。得られる値はすべて文字列である。
setAttribute(Attrib, Val)	対象要素	対象要素の属性 Attrib の値を Val にする。数を渡しても文字列に変換される。
hasAttribute(Attrib)	対象要素	対象要素に属性 Attrib がある場合は true を、ない場合は false を返す。
removeAttribute(Attrib)	対象要素	対象要素の属性 Attrib を削除する。
getNodeName()	対象要素	対象要素の要素名を得る。
createElement(Name)	document	Name で指定した要素を作成する。
createElementNS(NS, Name)	document	名前空間 NS で定義されている要素 Name を作成する。
createTextNode(text)	document	text を持つテキストノードを作成する。
cloneNode(bool)	対象要素	bool が true のときは対象要素の子要素すべてを、false のときは対象要素だけの複製を作る。
appendChild(Elm)	対象要素	Elm を対象要素の最後の子要素として付け加える。Elm がすでに対称要素の子要素のときは元の位置から最後の位置に移動する。
insertBefore(newElm, PElm)	対象要素	対象要素の子要素 PElm の前に newElm を子要素として付け加える。Elm がすでに対称要素の子要素のときは元の位置から指定された位置に移動する。
removeChild(Elm)	対象要素	対象要素の子要素 Elm を取り除く。
replaceChild(NewElm, OldElm)	対象要素	対象要素に含まれる子要素 OldElm を NewElm で置き換える。
setValue(value)	テキストノード	対象のテキストノードの値を value にする。

なお、表中の名前空間 (Namespace) とは、指定した要素が定義されている規格を指定するもので

す。一つの文書内で複数の規格を使用する場合、作成する要素がどこで定義されているのかを指定します。これにより、異なる規格で同じ要素名が定義されていてもそれらを区別することが可能となります。

また、要素のリストが得られるメソッドの戻り値の各要素は配列と同様に [] で参照でできます。

表 7.2 は DOM の要素に対するプロパティです。

表 7.2: DOM 要素に対するプロパティ

プロパティ名	説明
firstChild	指定された要素の先頭にある子要素
lastChild	指定された要素の最後にある子要素
nextSibling	指定された子要素の次の要素
previousSibling	現在の子要素の前にある要素
parentNode	現在の要素の親要素
hasChildNodes	その要素が子要素を持つ場合は true 持たない場合は false である。
nodeName	その要素の要素名前
nodeType	要素の種類 (1 は普通の要素、3 はテキストノード)
nodeValue	(テキスト) ノードの値
childNodes	子要素の配列
children	子要素のうち通常の要素だけからなる要素の配列 (DOM4 で定義)
firstElementChild	指定された要素の先頭にある通常の要素である子要素 (DOM4 で定義)
lastElementChild	指定された要素の最後にある通常の要素である子要素 (DOM4 で定義)
nextElementSibling	指定された子要素の次の通常の要素 (DOM4 で定義)
previousElementSibling	現在の子要素の前にある通常の要素 (DOM4 で定義)

なお、DOM4[23] とは 2015 年 11 月 19 日に Recommendation となった W3C が定める DOM の規格です。DOM の規格は今までに Level 1 から Level 3 までがあります。

- これらのプロパティのうち、nodeValue を除いてはすべて、読み取り専用です。
- ある要素に子要素がない場合にはその要素の firstChild や lastChild は null となります。
- ある要素に子要素がある場合、その firstChild.previousSibling や lastChild.nextSibling も null となります。firstElementChild などでも同様です。

付録A SVG の色名

次の表は [27] にある SVG で利用できる色名とその rgb 値の一覧表です。

色	色名	rgb 値	16進表示
aliceblue	aliceblue	rgb(240, 248, 255)	#F0F8FF
antiquewhite	antiquewhite	rgb(250, 235, 215)	#FAEBD7
aquamarine	aquamarine	rgb(127, 255, 212)	#7FFFDD
aqua	aqua	rgb(0, 255, 255)	#00FFFF
azure	azure	rgb(240, 255, 255)	#F0FFFF
beige	beige	rgb(245, 245, 220)	#F5F5DC
bisque	bisque	rgb(255, 228, 196)	#FFE4C4
black	black	rgb(0, 0, 0)	#000000
blanchedalmond	blanchedalmond	rgb(255, 235, 205)	#FFEBCD
blueviolet	blueviolet	rgb(138, 43, 226)	#8A2BE2
blue	blue	rgb(0, 0, 255)	#0000FF
brown	brown	rgb(165, 42, 42)	#A52A2A
burlywood	burlywood	rgb(222, 184, 135)	#DEB887
cadetblue	cadetblue	rgb(95, 158, 160)	#5F9EA0
chartreuse	chartreuse	rgb(127, 255, 0)	#7FFF00
chocolate	chocolate	rgb(210, 105, 30)	#D2691E
coral	coral	rgb(255, 127, 80)	#FF7F50
cornflowerblue	cornflowerblue	rgb(100, 149, 237)	#6495ED
cornsilk	cornsilk	rgb(255, 248, 220)	#FFF8DC
crimson	crimson	rgb(220, 20, 60)	#DC143C
cyan	cyan	rgb(0, 255, 255)	#00FFFF
darkblue	darkblue	rgb(0, 0, 139)	#00008B
darkcyan	darkcyan	rgb(0, 139, 139)	#008B8B
darkgoldenrod	darkgoldenrod	rgb(184, 134, 11)	#B8860B
darkgray	darkgray	rgb(169, 169, 169)	#A9A9A9
darkgreen	darkgreen	rgb(0, 100, 0)	#006400
darkgrey	darkgrey	rgb(169, 169, 169)	#A9A9A9
darkkhaki	darkkhaki	rgb(189, 183, 107)	#BDB76B
darkmagenta	darkmagenta	rgb(139, 0, 139)	#8B008B
darkolivegreen	darkolivegreen	rgb(85, 107, 47)	#556B2F
darkorange	darkorange	rgb(255, 140, 0)	#FF8C00

次のページへ

色	色名	rgb 値	16進表示
	darkorchid	rgb(153, 50, 204)	#9932CC
	darkred	rgb(139, 0, 0)	#8B0000
	darksalmon	rgb(233, 150, 122)	#E9967A
	darkseagreen	rgb(143, 188, 143)	#8FBC8F
	darkslateblue	rgb(72, 61, 139)	#483D8B
	darkslategray	rgb(47, 79, 79)	#2F4F4F
	darkslategrey	rgb(47, 79, 79)	#2F4F4F
	darkturquoise	rgb(0, 206, 209)	#00CED1
	darkviolet	rgb(148, 0, 211)	#9400D3
	deeppink	rgb(255, 20, 147)	#FF1493
	deepskyblue	rgb(0, 191, 255)	#00BFFF
	dimgray	rgb(105, 105, 105)	#696969
	dimgrey	rgb(105, 105, 105)	#696969
	dodgerblue	rgb(30, 144, 255)	#1E90FF
	firebrick	rgb(178, 34, 34)	#B22222
	floralwhite	rgb(255, 250, 240)	#FFFAF0
	forestgreen	rgb(34, 139, 34)	#228B22
	fuchsia	rgb(255, 0, 255)	#FF00FF
	gainsboro	rgb(220, 220, 220)	#DCDCDC
	ghostwhite	rgb(248, 248, 255)	#F8F8FF
	goldenrod	rgb(218, 165, 32)	#DAA520
	gold	rgb(255, 215, 0)	#FFD700
	gray	rgb(128, 128, 128)	#808080
	greenyellow	rgb(173, 255, 47)	#ADFF2F
	green	rgb(0, 128, 0)	#008000
	grey	rgb(128, 128, 128)	#808080
	honeydew	rgb(240, 255, 240)	#F0FFF0
	hotpink	rgb(255, 105, 180)	#FF69B4
	indianred	rgb(205, 92, 92)	#CD5C5C
	indigo	rgb(75, 0, 130)	#4B0082
	ivory	rgb(255, 255, 240)	#FFFFFF0
	khaki	rgb(240, 230, 140)	#F0E68C
	lavenderblush	rgb(255, 240, 245)	#FFF0F5
	lavender	rgb(230, 230, 250)	#E6E6FA
	lawngreen	rgb(124, 252, 0)	#7CFC00
	lemonchiffon	rgb(255, 250, 205)	#FFFACD
	lightblue	rgb(173, 216, 230)	#ADD8E6
	lightcoral	rgb(240, 128, 128)	#F08080
	lightcyan	rgb(224, 255, 255)	#E0FFFF

次のページへ

付録 3

色	色名	rgb 値	16進表示
lightgoldenrodyellow	lightgoldenrodyellow	rgb(250, 250, 210)	#FAFAD2
lightgray	lightgray	rgb(211, 211, 211)	#D3D3D3
lightgreen	lightgreen	rgb(144, 238, 144)	#90EE90
lightgrey	lightgrey	rgb(211, 211, 211)	#D3D3D3
lightpink	lightpink	rgb(255, 182, 193)	#FFB6C1
lightsalmon	lightsalmon	rgb(255, 160, 122)	#FFA07A
lightseagreen	lightseagreen	rgb(32, 178, 170)	#20B2AA
lightskyblue	lightskyblue	rgb(135, 206, 250)	#87CEFA
lightslategray	lightslategray	rgb(119, 136, 153)	#778899
lightslategrey	lightslategrey	rgb(119, 136, 153)	#778899
lightsteelblue	lightsteelblue	rgb(176, 196, 222)	#B0C4DE
lightyellow	lightyellow	rgb(255, 255, 224)	#FFFFE0
limegreen	limegreen	rgb(50, 205, 50)	#32CD32
lime	lime	rgb(0, 255, 0)	#00FF00
linen	linen	rgb(250, 240, 230)	#FAF0E6
magenta	magenta	rgb(255, 0, 255)	#FF00FF
maroon	maroon	rgb(128, 0, 0)	#800000
mediumaquamarine	mediumaquamarine	rgb(102, 205, 170)	#66CDAA
mediumblue	mediumblue	rgb(0, 0, 205)	#0000CD
mediumorchid	mediumorchid	rgb(186, 85, 211)	#BA55D3
mediumpurple	mediumpurple	rgb(147, 112, 219)	#9370DB
mediumseagreen	mediumseagreen	rgb(60, 179, 113)	#3CB371
mediumslateblue	mediumslateblue	rgb(123, 104, 238)	#7B68EE
mediumspringgreen	mediumspringgreen	rgb(0, 250, 154)	#00FA9A
mediumturquoise	mediumturquoise	rgb(72, 209, 204)	#48D1CC
mediumvioletred	mediumvioletred	rgb(199, 21, 133)	#C71585
midnightblue	midnightblue	rgb(25, 25, 112)	#191970
mintcream	mintcream	rgb(245, 255, 250)	#F5FFFA
mistyrose	mistyrose	rgb(255, 228, 225)	#FFE4E1
moccasin	moccasin	rgb(255, 228, 181)	#FFE4B5
navajowhite	navajowhite	rgb(255, 222, 173)	#FFDEAD
navy	navy	rgb(0, 0, 128)	#000080
oldlace	oldlace	rgb(253, 245, 230)	#FDF5E6
olivedrab	olivedrab	rgb(107, 142, 35)	#6B8E23
olive	olive	rgb(128, 128, 0)	#808000
orangered	orangered	rgb(255, 69, 0)	#FF4500
orange	orange	rgb(255, 165, 0)	#FFA500
orchid	orchid	rgb(218, 112, 214)	#DA70D6
palegoldenrod	palegoldenrod	rgb(238, 232, 170)	#EEE8AA

次のページへ

付録 4

付録 A SVG の色名

色	色名	rgb 値	16進表示
	palegreen	rgb(152, 251, 152)	#98FB98
	paleturquoise	rgb(175, 238, 238)	#AFEEEE
	palevioletred	rgb(219, 112, 147)	#DB7093
	papayawhip	rgb(255, 239, 213)	#FFEFDD
	peachpuff	rgb(255, 218, 185)	#FFDAB9
	peru	rgb(205, 133, 63)	#CD853F
	pink	rgb(255, 192, 203)	#FFC0CB
	plum	rgb(221, 160, 221)	#DDA0DD
	powderblue	rgb(176, 224, 230)	#B0E0E6
	purple	rgb(128, 0, 128)	#800080
	red	rgb(255, 0, 0)	#FF0000
	rosybrown	rgb(188, 143, 143)	#BC8F8F
	royalblue	rgb(65, 105, 225)	#4169E1
	saddlebrown	rgb(139, 69, 19)	#8B4513
	salmon	rgb(250, 128, 114)	#FA8072
	sandybrown	rgb(244, 164, 96)	#F4A460
	seagreen	rgb(46, 139, 87)	#2E8B57
	seashell	rgb(255, 245, 238)	#FFF5EE
	sienna	rgb(160, 82, 45)	#A0522D
	silver	rgb(192, 192, 192)	#COCOCO
	skyblue	rgb(135, 206, 235)	#87CEEB
	slateblue	rgb(106, 90, 205)	#6A5ACD
	slategray	rgb(112, 128, 144)	#708090
	slategrey	rgb(112, 128, 144)	#708090
	snow	rgb(255, 250, 250)	#FFFFFA
	springgreen	rgb(0, 255, 127)	#00FF7F
	steelblue	rgb(70, 130, 180)	#4682B4
	tan	rgb(210, 180, 140)	#D2B48C
	teal	rgb(0, 128, 128)	#008080
	thistle	rgb(216, 191, 216)	#D8BFD8
	tomato	rgb(255, 99, 71)	#FF6347
	turquoise	rgb(64, 224, 208)	#40E0D0
	violet	rgb(238, 130, 238)	#EE82EE
	wheat	rgb(245, 222, 179)	#F5DEB3
	whitesmoke	rgb(245, 245, 245)	#F5F5F5
	white	rgb(255, 255, 255)	#FFFFFF
	yellowgreen	rgb(154, 205, 50)	#9ACD32
	yellow	rgb(255, 255, 0)	#FFFF00

付録B 参考文献について

残念ながら SVG のまとまった解説がある日本語の本はないようです。このテキストを書来始めたころに参考にした本は [2] と [3] です。また、途中からは [7] の図書も参考にしました。これらの本のリンク先は amazon.co.jp です。平成 18 年 4 月現在で検索した書籍のページにジャンプします。

なお、SVG のフルの規格を調べるのであればやはり元の文書 [27] を参考にするしかありません。PDF 版をダウンロードしておけば文書内にリンクが張ってあるので比較的簡単に該当箇所へジャンプできます(700 ページ以上のあります)。

[2] の図書については次のような特徴があります。

- 発行年次が 2002 年と少し古く SVG1.0 に基づいた記述のようです。
- SVG の図形やテキストについての記述が詳しいです。
- フィルターについても基本的なところが詳しくかかれています。本テキストを作成するに当たっては参考になった部分が多いです。
- インターラクティブな SVG のところもかなり詳しく書かれています。

[3] の図書については次のような特徴があります。

- 似たような SVG の例が多いので前半部は飛ばして読めます。。
- JavaScript を用いた部分の解説がかなり多いのでインターラクティブな SVG を作成したい人には参考になるでしょう。
- その反面、フィルタの解説が少ないので物足りなく感じるかもしれません。

[7] の図書については次のような特徴があります。

- オーソドックスな SVG の解説書です。
- フィルターについては基本的なものだけです。
- フィルタの例については [27] のものと似ています。
- CSS の一覧表があります。

参考となる SVG のファイルや説明が日本語である Web サイトは検索するとたくさん見つかります。しかし、本書のように DOM を用いて SVG 文書を操作する方法を説明しているサイトはほとんど見つかりません。

なお、今後 HTML も含めて XML ベースの文書を操作するには DOM が基本になります。JavaScript の解説書でも最近のものは DOM を取り扱っています。

JavaScript の解説書は非常にたくさんあります。なかでも [8] は言語の解説書として手元に置いておくとよいでしょう。また、姉妹書の [9] は関数の仕様を調べるのに役に立ちます。

JavaScript は注意してコーディングをすることが必要です。プログラミングスタイルも様々ですが、大規模なアプリケーションをチームで組む必要があるときなどは [31] を参考にしてください。また、より良いコーディングスタイルを身に着けておきたい方は [10] がお勧めです。また、使用を避けるべき JavaScript の点については [4] がおすすめです。この本の記述は ECMAScript のバージョン 3 に基づいているようです。現在 ECMAScript のバージョンは 2015 年に出た 6 が最新版です [5]。バージョン 5 では配列に関するいろいろなメソッドや JSON の処理が標準で備わっています。この部分が古くなっていることに注意してください。

また、本文ではそれほど解説しませんでしたが Ajax による Web アプリケーションは標準の技術になってしましました。

2014 年に策定された HTML5 は W3C のサイト [26] その内容を見ることができます最近のブラウザは HTML5 の機能をサポートしています。また、iPhone や iPod Touch に搭載されている Safari も HTML5 に対応してるばかりでなく、特有の機能であるマルチタッチを利用する API もあります¹。

テキスト内で簡単に触れた PHP については [20, 12] などがあります。PHP は Web 時代後の言語なので、Web 関係の処理が簡単にできる機能が言語自体に備わっています。しかし、PHP のスクリプト系の言語としての面も丁寧に解説した書籍はほとんどありません。PHP の全貌を知る唯一の情報は PHP のホームページ [38] です。

¹ iPhone Javascript マルチタッチなどのキーワードで検索すれば解説しているサイトが見つかるでしょう。

関連図書

- [1] Andreas Anyuru(吉川 邦夫 訳), 実践プログラミング WebGL HTML & JavaScript による 3D グラフィック, 翔泳社, 2012 年
- [2] Kurt Cagle, *SVG Programming: The Graphical Web*, Apress 2002
- [3] Oswald Cansepato, *Fundamentals of SVG Programming: Concept to Source Code*, CHARLES RIVER MEDIA, INC. 2004
- [4] D. Crockford(水野 貴明 訳), JavaScript: The Good Parts 「良い」パートによるベストプラクティス, オライリージャパン, 2008
- [5] ECMA, ECMAScript[©] 2015 Language Specification(6th edition)
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- [6] ECMA, JSON, <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>, 2013
- [7] J. David Eisenberg, *SVG Essentials*, O'Reilly, 2002
- [8] David Flanagan(村上列 訳) JavaScript 第 6 版, オライリー・ジャパン, 2012
- [9] David Flanagan(木下哲也, 福龍興業 訳) JavaScript クイックリファレンス 第 6 版, オライリー・ジャパン, 2012
- [10] David Herman (吉川 邦夫 監修, 訳), Effective JavaScript JavaScript を使うときに知っておきたい 68 の作法, 翔泳社, 2013 年
- [11] Peter Gasston, CSS3 開発者ガイド 第 2 版 モダン Web デザインのスタイル設計, オライリー・ジャパン, 2015
- [12] Rasmus Lerdorf, Kevin Tatroe, Peter MacIntyre(高木正弘 訳) プログラミング PHP 第 2 版, オライリージャパン, 2007
- [13] Stoyan Stefanov(水野貴明, 渋川よしき 訳) オブジェクト指向 JavaScript, アスキー・メディアワークス, 2012
- [14] Document Object Model (DOM) Level 3 Events Specification, W3C Technical Reports and Publications, 2007, <http://www.w3.org/TR/DOM-Level-3-Events/>
- [15] Ray Erik T.(宮下尚、牧野聰、立堀道明訳) 入門 XML 第 2 版, オライリー・ジャパン 2004

- [16] Donald E. Knuth, *The METAFONTbook*, Computers & typesetting /C, Addison-Wesley, 2000
- [17] Hajime Ōuchi, Japanese Optical and Geometrical Art, Dover Pub., 1977
- [18] J. O. Robinson, *The Psychology of Visual Illusion*, Dover, N. Y., 1998(originally published in 1972)
- [19] Chris Shiflett (桑村 潤, 廣川 類訳) 入門 PHP セキュリティ, オライリージャパン, 2006
- [20] David Sklar (桑村 潤, 廣川 類訳) 初めての PHP5, オライリージャパン, 2005
- [21] Steve Souders, (武舎 広幸, 福地 太郎, 武舎 るみ),
ハイパフォーマンス Web サイト 高速サイトを実現する 14 のルール, オライリージャパン , 2008
- [22] W3C, Document Object Model (DOM), <http://www.w3.org/DOM>
- [23] W3C, W3C DOM4 <https://www.w3.org/TR/2015/REC-dom-20151119/>
- [24] W3C, Document Object Model(DOM) Level 2 Events Specification, W3C Recommendation, 2000
<http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/>
- [25] W3C, HTML 4.01 Specification, W3C Recommendation 24 December 1999
- [26] W3C, HTML5 A vocabulary and associated APIs for HTML and XHTML,
<http://www.w3.org/TR/html5/>
- [27] W3C, Scalable Vector Graphics (SVG) 1.1 Specification,
<http://www.w3.org/TR/2011/REC-SVG11-20110816/>
- [28] W3C, HTML & CSS , <http://www.w3.org/standards/webdesign/htmlcss>
- [29] W3C, Web Storage, <https://www.w3.org/TR/webstorage/>
- [30] Nicholas C. Zakas (水野 貴明 訳) ハイパフォーマンス JavaScript , オライリージャパン 2011
- [31] Nicholas C. Zakas(豊福 剛 訳), メンテナブル JavaScript 読みやすく保守しやすい JavaScript コードの作法, オライリー・ジャパン, 2013 年
- [32] 後藤 哲男 (編集), 田中 平八 (編集) , 錯視の科学ハンドブック, 東京大学出版会 (2005/02)
- [33] 北岡明佳, 錯視入門, 朝倉書店、2010 年
- [34] 杉原 厚吉, 錯視図鑑 脳がだまされる錯覚の世界, 誠文堂新光社, 2012 年
- [35] フィービ・マクノートン (著), 駒田曜 (翻訳), 錯視芸術 (アルケミスト双書) , 創元社, 2010
- [36] 馬場 雄二, 田中 康博, 試してナットク ! 錯視図典 CD-ROM 付, 講談社 (2004/12/17)
- [37] ジャック・ニニオ (鈴木幸太郎、向井智子訳) 錯視の世界 古典から CG 画像まで, 新曜社 2004 年
- [38] 日本 PHP ユーザー会, <http://www.php.gr.jp/>

付録 C JavaScript 入門

C.1 JavaScript とは

JavaScript というと HTML 文書文書の中で取り扱われ、主にブラウザで利用される言語と思っている人が多いと思います。しかし、JavaScript の言語仕様のもととなるものは ECMA が定めている ECMAScript と呼ばれるものです。一般的のプログラミング言語と同様に、言語の構成要素が定義されています。ブラウザなどで使用される場合には、この言語を基に、ブラウザ上で定義されたオブジェクトやそれを操作する方法を用いてプログラムを組むことになります。現に、JavaScript を網羅的に解説している [8] では第 I 部が「コア JavaScript」第 II 部が「クライアントサイド JavaScript」と分けられています。最近では「サーバーサイド JavaScript」と呼ばれるものも登場しています。

これらの違いは、プログラミングを記述するための手段として JavaScript を利用することで、利用する環境に応じて別のライブラリーが用意されているために、いろいろな環境で利用が可能となっています。これは C 言語で簡単なプログラムを記述する場合には複雑なライブラリーは必要としませんが、ウィンドウを開いて実行されるプログラムにはウィンドウを開くためのライブラリーが必要になってくるとの似ています。

とはいっても、コアの JavaScript だけを使用してプログラムを組むことを解説している参考書は皆無といえます。このテキストではコアの JavaScript によるプログラミングを解説します。とはいっても、コアの JavaScript を実行する環境はやはりブラウザとなります。ブラウザ上でどのようにコアの JavaScript を実行し、デバッグする方法を示しながら、JavaScript のプログラミング言語としての特徴をつかんでください。

C.2 データの型

JavaScript のデータ型には大きく分けてプリミティブ型と非プリミティブ型の 2 種類があります。

C.2.1 プリミティブデータ型

プリミティブ型には表 C.1 のような種類のものがあります。

変数や値の型を知りたいときは `typeof` 演算子を使います。

Number 型 JavaScript で扱う数は 64 ビット浮動小数点形式です。数を表現する方法(数値リテラル)としては次のものがあります。

表 C.1: JavaScript のプリミティブなデータ型

型	説明
Number	浮動小数点数だけ
String	文字列型、1 文字だけのデータ型はない。ダブルクオート ("") やシングルクオート ('') で囲む。
Boolean	true か false の値のみ
undefined	変数の値が定義されていないことを示す
null	null という値しか取ることができない特別なオブジェクト

- 整数リテラル 10 進整数は通常通りの形式です。16 進数を表す場合は先頭に 0x か 0X をつけます。0 で始まりそのあとに x または X が来ない場合には 8 進数と解釈される場合があるので注意が必要です。
- 浮動小数点リテラル 整数部、そのあとに必要ならば小数点、小数部その後に指数部がある形式です。

特別な Number Number 型には次のような特別な Number が定義されている。

- Infinity 無限大を表す読み出し可能な変数である。オーバーフローした場合や 1/0 などの結果としてこの値が設定されます。
- NaN Not a Number の略である。計算ができなかった場合表す読み出し可能な変数です。文字列を数値に変換できない場合や 0/0 などの結果としてこの値が設定されます。

String 型 文字列に関する情報や操作には次のようなものがあります。

表 C.2: 文字列操作のメソッド

メンバー	説明
length	文字列の長さ
indexOf(needle[, start])	needle が与えられた文字列内にあればその位置を返す。start の引数がある場合には、指定された位置以降から調べる。見つからない場合は -1 を返す。
split(separator[, limit])	separator で与えられた文字列で与えられた文字列を分けて配列で返す。セパレーターの部分は返されない。2 番目の引数はオプションで、分割する最大数を与える。
substring(start[, end])	与えられた文字列の start から end の位置までの部分文字列を返す。end がない場合には文字列の最後までがとられる。

Bool型 `true` と `false` の 2 つの値をとります。この 2 つは予約語です。論理式の結果としてこれらの値が設定されたり、論理値が必要なところでこれらの値に設定されます。

undefined 値が存在しないことを示す読み出し可能な変数である。変数が宣言されたのに値が設定されていない場合などはこの値に初期化されます。

null `typeof null` の値が "object" であることを示すように、オブジェクトが存在しないことを示す特別なオブジェクト値（であると同時にオブジェクトでもある）です。

C.2.2 配列

配列の宣言と初期化 配列を使うためには、変数を配列で初期化する必要があります。変数の宣言と同時に使うこともできます。配列の初期化は次のように行います。

```
var a = [];
var b = [1,2,3];
```

- `a` は空の配列で初期化されています。
- `b` は `b[0]=1, b[1]=2, b[2]=3` となる配列で初期化されています。

次のことに注意する必要があります。

- 配列の各要素のデータの型は同じでなくともかまいません。
- 実行時に配列の大きさを自由に変えることができます。
- 配列の要素に配列を置くことができます。たとえば、次のような配列の初期化が可能です。

```
var a=[1,[2,3,4],"a"];
```

この例では `a[0]` は数 1、`a[1]` は配列 `[2,3,4]`、`a[2]` は文字列 "a" でそれぞれ初期化されています。

配列のメソッドについては C.7 で解説します。

C.3 演算子

C.3.1 代入、四則演算

数に対しては C 言語と同様の演算子が使用できます。ただし、次のことに注意する必要があります。

- +演算子は文字列の連接にも使用できます。+演算子は左右のオペランドが Number のときだけ、ふたつの数の和をもとめます。どちらかが数でもう一方が文字列の場合は数を文字列に直して、文字列の連接を行います¹。

```
1+2 => 3
1+"2" => 12
```

- そのほかの演算子 (-*/)については文字列を数に変換してから数として計算します。
- 文字列全体が数にならない場合には変換の結果が NaN になります。次の例を見てください。

```
"2" + 3 => "23"
"2"-0 +3 => 5
"2"*3 => 6
"2""*3" => 6
"f" *2 => NaN
"0xf"*2 => 30
```

最後の例では"0xf"が 16 進数と解釈され (15), その値が 2 倍されています。

- 整数を整数で割った場合、割り切れなければ小数となります。

```
1/3 => 0.3333333333333333
```

C.3.2 比較演算子

JavaScript の比較演算子は通常の >、>= などが使えます。数と文字列の比較も文字列が数に変換されて比較されます。文字列同士を比較する場合は文字コード順になります。次の例を見てください。

```
>(11>2)? "true": "false";
"true"
>(11>"2")? "true": "false";
"true"
>("11">2)? "true": "false";
"true"
>("11">"2")? "true": "false";
"false"
```

最初の 3 つは数として比較されていますが、最後のものは文字コードで "1" のほうが "2" より小さいので判定が逆になっています。

¹一般にどのようなオブジェクトにも `toString` というメソッドが用意されていて、文字列が必要な状況ではこのメソッドによってオブジェクトが文字列に変換されます。

また、比較演算子==で文字列と数を比較すると文字列は数に直されて比較されます。値の型を含めて等しいかどうかを調べるために演算子==(等しい) や!==(等しくない) を用います。特別な事情がない限り==や!==を使いましょう [4, 127 ページ]。

```
>("1"=="1)?"true":"false";
"true"
>("1"==="1)?"true":"false";
"false"
```

C.4 制御構造

C.4.1 if 文

if 文はある条件が成立したときやしなかったときにだけ実行したい場合に使用します。一般的の形は次のようになります。

```
if(条件式)
    条件式が成立したときに実行したい式
```

実行したい式が複数ある場合にはブロック{}で囲みます。特別な場合を除いては実行したい文が一つだけであってもブロックにしておくほうが良いでしょう。

2つ目の形式は else があるものです。

```
if(条件式)
    条件式が成立したときに実行したい式
else
    条件式が成立しなかったときに実行したい式
```

else の部分もブロックにしておいたほうがバグの発生が防げます。

C.4.2 switch 文

if 文においてある条件が成立しなかった場合に、else のなかでさらに次の条件が成立するか調べたい場合があります。これが多くなってくると if 文の入れ子が深くなりすぎてプログラムが読みにくくなります。このような場合には switch 文を使うとプログラムが見やすくなります。switch 文は次のような構造をとります。

```
switch(式){
    case 値 1:
        式の値が値 1 のときに実行される
        break;
    case 値 2:
        式の値が値 2 のときに実行される
```

```
break;  
.  
.  
.  
default:  
式の値が case に現れなかった場合に実行される  
}
```

ここに現れる `break` 文は `switch` の処理を中断することを意味しています。これがないとその下の部分も実行されてしまいます。いくつかの場合に処理が同じになる場合以外には使わないほうが良いでしょう。

C.4.3 for 文と while 文

同じような処理を繰り返して行うためには `for` 文や `while` 文を用います。`for` 文は次のような構造を持ちます。

```
for(初期化 ; ループの終了条件 ; 次の繰り返しのための処理)  
繰り返しの処理
```

- 初期化のところでは繰り返しを実行する前の変数の値の初期値を設定するのが普通です。
- ループの終了条件は通常は比較演算子を書きます。
- 次の繰り返しのための準備としてはループの終了条件に現れる変数の値を変更するのが普通です。
- 初期化を実行した後、ループの終了条件がチェックされます。したがって、繰り返し処理は 1 度も実行されない場合があります。
- 初期化と次の繰り返しのための処理を複数の変数に対して行う場合にはコンマオペレータ (,) で処理を記述します。

たとえば、1 から N までの値の総和を求めるプログラムは次のように書けます。

```
for(sum = 0, i = 0; i = 1; i<= N; i++) {  
    sum += i;  
}
```

- `for` 文の初期化、終了条件、次の繰り返しのための処理の部分は全くなくてもかまいません。例えば 3 つの処理の部分がない `for(;;)` も文法上は許されます。この記述は無限ループを実現できます。
- `for` 文の繰り返しを途中で打ち切りたいときには `break` 文を使用します。

while 文は for 文と同様に繰り返しを行うためのものです。while 文では次のような構造を取ります。

```
初期化;  
while(終了条件) {  
    繰り返しの処理  
    次の繰り返しのための処理  
  
    終了条件のところを空にすることはできません。
```

C.5 関数

C.5.1 関数の定義と呼び出し

次の例は sum() という関数を定義している例です。

```
function sum(a,b) {  
    var c = a + b;  
    return c;  
}
```

関数の定義は次の部分から成り立っています。

- function キーワード
戻り値の型を記す必要はありません。
- 関数の名前
function の後にある識別名が関数の名前になります。この場合は sum が関数の名前になります。
- 引数のリスト
関数名の後に () 内にカンマで区切られた引数を記述します。この場合は変数 a と b が与えられています。引数はなくてもかまいません。
- 関数の本体であるコードブロック
{}で囲まれた部分に関数の内容を記述します。
- return キーワード
関数の戻り値をこの後に記述します。戻り値がない場合には戻り値として undefined が返されます。

実行例 次の部分はこの関数の実行例です。

```
>sum(1,2)  
3  
>sum(1)
```

```
NaN
>sum(1,2,3)
3
```

- 引数に 1 と 2 を与えれば期待通りの結果が得られます。
- 引数に 1 だけを与えた場合、エラーが起こらず、NaN となります。これは、不足している引数（この場合には b）に undefined が渡されるためです。1undefined+ の結果は通常の計算ができないので、数ではないことを示す NaN になります。
- 引数を多く渡してもエラーが発生しません。無視されるだけです。

JavaScript の関数はオブジェクト指向で使われるポリモーフィズムをサポートしていないことがわかります。さらに、次の例で見るように同じ関数を定義してもエラーにならないません。後の関数の定義が優先されます。

```
function sum(a, b){
  var c = a+b;
  return c;
}
function sum(a, b, c){
  var d = a+b+c;
  return d;
}
```

C.5.2 仮引数への代入

仮引数に値を代入してもエラーとはなりません。仮引数の値がプリミティブなときとそうでないときとでは呼び出し元における変数の値がかわります。

次の例は呼び出した関数の中で仮引数の値を変化させたときの例です。

```
function func1(a){
  a = a*2;
  return 0;
}
function func2(a){
  a[0] *=2;
  return 0;
}
```

- func1() では仮引数 a の値を 2 倍しています。これを次のように実行すると、呼び出し元の変数の値には変化がないことがわかります。つまり、プリミティブな値を仮引数で渡すと値そのものが渡されます（値渡し）。

```

>a = 4;
4
>func1(a);
0
>a;
4
\end{verbatim}
\item \verb+func2()+ の仮引数は配列が想定しています。この配列の先頭の値だけ 2 倍さ
れる関数です。これに配列を渡すと、戻ってきたとき配列の先頭の値
が変化しています。つまり、プライミティブ型以外では仮引数の渡し方が
参照渡しであることがわかります。
\begin{Verbatim}
>a = [1,2,3];
[1, 2, 3]
>func2(a);
0
>a;
[2, 2, 3]

```

C.5.3 argumentsについて

JavaScript では引数リストで引数の値などが渡されるほかに `arguments` という配列のようなオブジェクトでもアクセスできます。

- 引き渡された変数の数は `length` で知ることができます。

```

function sumN(){
    var i, s = 0;
    for(i = 0; i < arguments.length;i++) {
        s += arguments[i];
    }
    return s;
}

```

実行例は次のとおりになります。

```

>sumN(1,2,3,4);
10
>sumN(1,2,3,4,5);
15

```

- 引数があっても無視できます。

```
function sumN2(a,b,c){
    var i, s = 0;
    for(i = 0; i < arguments.length;i++) {
        s += arguments[i];
    }
    return s;
}
```

この例では引数が 3 個より少なくても正しく動きます。実行例は次のとおりです。

```
>sumN2(1,2,3,4,5);
15
```

- 仮引数と arguments は対応していて、片方を変更しても他の方も変更されます。

実行例は次のとおりです。

```
function sum2(a, b){
    var c;
    a *= 3;
    console.log(arguments[0]);
    return a + b;
}

>sum2(1,2,3,4,5);
3
5
```

C.6 変数のスコープと簡単な例

JavaScript に限らず、どのプログラミング言語でも変数のスコープという概念は重要です。変数のスコープとは、使用している変数が通用する範囲のことです。JavaScript では次のようになっています。

- 変数は宣言しないでも使用できます。この場合はグローバル変数になります。つまり、どの範囲からも参照が可能になります。
- 関数内で var を用いて宣言した変数はその関数の中で有効です。つまり、ローカル変数となります。同じ名前のグローバル変数があった場合には、そのグローバル変数にはアクセスできません。
- 関数内の変数の宣言は、どこで行ってもローカルな変数と扱われます。したがって、関数の途中で var 宣言してそこで初期化した場合、その前で同じ変数を使用するとその値は初期化されていない undefined になります。

- JavaScript では関数も通常のオブジェクトで変数に代入することができます。今まで関数を定義していた方法

```
function foo() { ...  
}
```

はグローバル変数 `foo` に対し次の形のコードと同じ意味を持ちます。

```
var foo = function(){...  
}
```

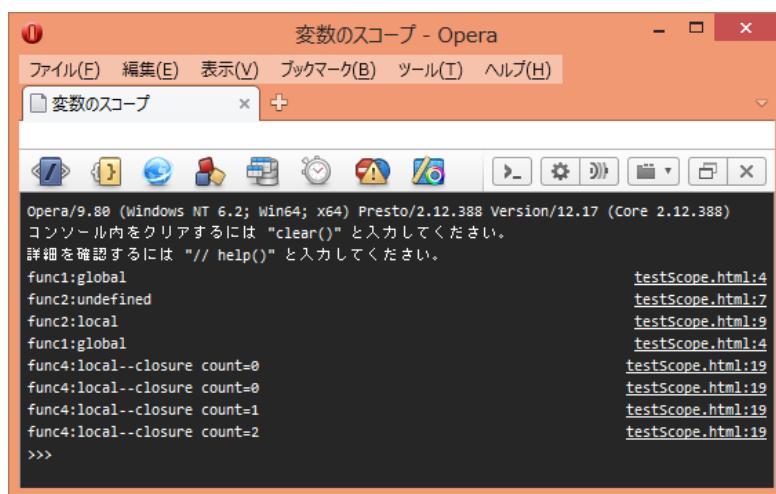


図 C.1: 変数のスコープ-コンソールを使う

HTML リスト C.1: 変数のスコープ (`testScope.html`)

```
1  <!DOCTYPE html>  
2  <html>  
3  <head>  
4  <title>変数のスコープ</title>  
5  <script type="text/ecmascript">  
6  var Variable = "global";  
7  function func1() {  
8      console.log("func1:"+Variable);  
9  }  
10 function func2() {  
11     console.log("func2:"+Variable);  
12     var Variable = "local";  
13     console.log("func2:"+Variable);  
14 }  
15 function func3(arg) {  
16     var Variable = "local";  
17     func1();  
18 }
```

```
19  function func4() {
20      var Variable = "local--closure";
21      var count = 0;
22      return function(){
23          console.log("func4:"+Variable+" count="+count);
24          count++;
25      };
26  }
27
28  func1();
29  func2();
30  func3();
31  f = func4();
32  f();
33  (func4())();
34  f();
35  f();
36  </script>
37  </head>
38  <body>
39  </body>
40  </html>
```

この HTML 文書は JavaScript の実行結果をコンソールに出力するものです。

- 6 行目でグローバル変数变数 Variable を定義して、値を"global"に定義しています。
- 関数関数 func1 は変数变数 Variable の値を関数名とともに出力します(7 行目から9 行目)。console.log() はコンソールに引数の値を表示する関数です。
- 関数関数 func2 は変数 Variable を関数内で定義しその値を"local"に定義しています。さらにその値を関数名とともに出力します(10 行目から14 行目)。
- 関数 func3 は関数 func2 と同様に変数 Variable を関数内で定義しその値を"local"に定義しています。出力は関数 func1 を呼び出すことで実現しています。
- 関数 func4 は関数 func3 と同様に変数 Variable を関数内で定義しその値を"local"に定義しています。この関数の戻り値は変数变数 Variable の値をコンソールに出力する関数です。
- 28 行目で関数 func1 が実行されています。この関数では変数 Variable は行目で定義されたものが参照されますので出力結果は func1:global となります。
- 29 行目で関数 func2 が実行されています。この関数では Variable は12 行目でローカル変数が定義されていることからグローバル変数のほうが参照されません。この時点では値が代入されていないので func2:undefined が出力されます。その後で値が設定されるので次の出力結果は func2:local となります。
- 30 行目で関数 func3 が実行されています。この関数では Variable は12 行目でローカル変数が定義されて、値が設定されています。出力は関数 func1 で行われているので、参照される Variable はグローバルに定義されているものになります。

- 31 行目で関数 func4 が実行されていて、その戻り値が変数 f に代入されています。この時点では戻り値である関数は実行されていません。関数 func4 の内でローカル変数 Variable が定義され、その値を戻り値の関数が利用しています。
- 32 行目で関数 func1 が実行されています。このとき利用される変数 Variable は関数 func4 内で定義されたものです。つまり、この関数からは関数 func4 内で定義されたローカル変数が参照できるのです。しかしながら、この関数で定義されたローカル変数を外部から直接変更や参照する手段はありません。
- 33 行目戻り値が関数オブジェクトのとき、それを変数にしまわないでそのまま実行する方法です。関数 func4 の内容を直接この () 内に記述することも可能です。この場合、関数自体に名前がありませんので無名関数と呼ばれます。
- 34 行目では31 行目で保存された関数を再び実行しています。この関数ではコンソールへの出力後、ローカル変数変数 count の値を 1 増加させているのでこの変数の値が 1 となり、その値が出力されます。この変数変数 count の値を変えることができるるのはこの関数の呼び出ししかできません。このような技術をクロージャと呼びます。クロージャについては後で詳しく説明します。
- 35 行目ではもう一度、同じ関数が実行され、変数 count の値として 2 が output されます。この変数の値は関数 f() が呼び出されたときだけ、1 増加し、これ以外の方法でこの変数の値を変更することができません。つまり、クロージャを用いることで一種のカプセル化が可能となっています。

C.6.1 JavaScript における関数の特徴

JavaScript 関数ではほかの言語では見られない関数の取り扱い方法があります。

関数もデータ

関数もデータ型のひとつなので、関数の定義を変数に代入することができます。HTML リスト C.1 の 31 行目では関数の戻り値が関数オブジェクトで、その結果を変数に代入しています。代入はいつでもできるので、実行時に関数の定義を変えることも可能です。

無名関数とコールバック関数

HTML リスト C.1 の 22 行目の関数オブジェクトは `function` の後には関数名がありません。このような関数は無名関数と呼ばれます。HTML 文書などでは、いろいろなイベント（マウスがクリックされた、一定の時間が経過した）が発生したときに、その処理を行う関数を登録する必要があります。この関数をその場で定義して、無名関数で渡すことはよく使われる技法です。なお、ある関数に引数として渡される関数は渡された関数の中で呼び出されるのでコールバック関数とよばれます。

次の例は、一定の経過時間後にある関数を呼び出す window オブジェクトの `setTimeout()` メソッドの使用例です。

```

1 var T = new Date();
2 window.setTimeout(
3   function(){
4     var NT = new Date();
5     if(NT-T<10000) {
6       console.log(Math.floor((NT-T)/1000));
7       window.setTimeout(arguments.callee,1000);
8     }
9   },1000);

```

- 1行目では実行開始時の時間を変数 `T` に格納しています。単位はミリ秒です。
- このメソッドは一定時間経過後に呼び出される関数と、実行される経過時間（単位はミリ秒）を引数に取ります。
- 実行する関数は3行目から9行目で定義されています。
- この関数内で一定の条件のときはこの関数を呼び出すために、この関数に名前はありません（3行目）。
- 4行目で呼び出されたときの時間を求め、経過時間が 10000 ミリ秒以下であれば（5行目）、経過時間を秒単位で表示します（6行目）。
- さらに、自分自身を 1 秒後に呼び出す（7行目）。`arguments` をもつ関数を `arguments.callee` で呼び出すことができます。つまり無名関数である自分自身を呼び出せます。

問題 C.1 次のプログラムは何を計算するか答えなさい。

```

var f = function(n) {
  if( n<=1) return 1;
  return n*arguments.callee(n-1);
}

```

自己実行関数

関数を定義してその場で直ちに実行することができます。次のコードを見てみましょう。

```

var i;
for(i=1;i<10;i++) {
  console.log(i+" "+i*i);
}

```

このプログラムを実行すると 1 から 9 までの値とその 2 乗の値がコンソールに出力されます。実行後に、コンソールに `i` と入力すれば 10 が出力されます。つまり、変数 `i` が存在しています。

ある関数を実行した後でその中で使用したグローバル変数を消してしまいたいことがある。それを実現するためには次のように記述する。

```
(function(){
  var i;
  for(i=1;i<10;i++) {
    console.log(i+" "+i*i);
  })();
});
```

この様に関数の定義を全体で `()` で囲み、その後に関数の呼び出しを示すための `()` を付けています。

この技法は、初期化の段階で 1 回しか実行しない事柄を記述し、かつグローバルな空間を汚さない(余計な変数などを残さない)手段として用いられています。

C.6.2 クロージャ

JavaScript をオブジェクト指向言語として使用するための基本的な概念です。ここで上げる例は実用に乏しいと思われるかもしれないが、この後に出てくるオブジェクトの項ではより実用的なものと理解できるでしょう。

スコープチェイン

関数の中で関数を定義すると、その内側の関数内で `var` で宣言された変数のほかに、一つ上の関数で利用できる(スコープにある)変数が利用できます。これがスコープチェインです。

```
var G1, G2;
function func1(a) {
  var b, c;
  function func2() [
    var G2, c;
    ...
  }
}
```

- 関数 `func1()` ではグローバル変数である `G1` と `G2`、仮引数の `a` とローカル変数 `b` と `c` が利用できます。
- 関数 `func2()` ではグローバル変数である `G1`、`func1()` の仮引数の `a` と `func1()` のローカル変数 `b`、`func2()` のローカル変数 `G2` と `c` が利用できます。

このように内側で定義された関数は自分自身の中で定義されたローカル変数があるかを探し、見つからない場合には一つ上のレベルでの変数を探します。これがスコープチェインです。JavaScript の関数のスコープは関数が定義されたときのスコープチェインが適用されます。これをレキシカルスコープと呼びます。レキシカルスコープは静的スコープとも呼ばれます。これに対して実行時にスコープが決まるものは動的スコープと呼ばれます。

クロージャ

このように関数内部で宣言された変数は、その外側から参照することができません。つまり、その関数は関数内のローカル変数を閉じ込めています。しかし、関数内部で定義された関数を外部に持ち出す（グローバルな関数にする）と、持ち出された関数のスコープチェイン内に定義された親の関数のスコープを引き継いでいることから、親の関数のローカル変数の参照が可能となります。

このような関数に対して依存する環境（変数や呼び出せる関数などのリスト）を合わせたものをその関数のクロージャと呼ばれます。

次の例は関数を定義したのちに、仮引数の値を 1 増加しています。

```
function f(arg) {
    var n = function() {
        return arg;
    }
    arg++;
    return n;
}

>var m = f(123);
undefined
>m();
124
```

定義時の arg の値ではなく、参照時の arg の値が参照されていることに注意してください。

このことが連続して関数を作成したときにバグを引き起こすことがあります。

次の例は配列の添え字を戻り値にする関数を 3 つ定義しています。配列に保存された関数が添え字の値を返すように見えますが、実際にはそのようには動きません。

```
function f() {
    var a = [];
    var i;
    for(i=0; i<3; i++) {
        a[i] = function() {
            return i;
        };
    }
}
```

```

    return a;
}

>var a = f();
undefined
>a[0]();
3
>a[1]();
3
>a[2]();
3

```

すべて同じ値が返る関数になってしまっています。これは関数 `f()` が実行されるとローカルな変数 `i` の値は `for` 文が終了した時点で値が `3` となり、戻り値の関数が実行された時点では、その値が参照されるからです。

これを避けるためには関数にその値を渡してスコープチェインを切る必要があります。

```

function f2() {
    var a = [];
    var i;
    for(i=0; i<3; i++) {
        a[i] = (function(x){
            return function() {
                return x;
            }
        })(i);
    };
    return a;
}

```

- 引数を取る無名関数を用意し、その場で与えられた引数を返す無名関数を返す関数を実行しています。
- 仮引数には、実行されたときの `i` のコピーが渡されるので、その後変数の値が変わっても呼び出された時の値が保持されます。

```

>var a = f2();
undefined
>a[0]();
0
>a[1]();
1
>a[2]();
2

```

最後の例題は同じオブジェクトを連続的に作成して、通し番号を付けたい場合に応用できます。Ajax で非同期通信を行う場合では通信が終了したときに呼び出される関数 (<コールバック関数>要素) を使用します。このとき、コールバック関数を定義したときに特別な変数の値を利用する場合があります。非同期通信を同時に複数行う場合にはその変数の値が呼び出し時と実行時で変わってしまう場合があります。定義したときの変数の値を実行時にそのまま利用するためにはこのような手法が必要になります。

C.7 配列のメソッド

配列には表 C.3 のようなメソッドが定義されています。このうちいくつかを具体的な例で示します。

表 C.3: 配列のメソッド

メンバー	説明
<code>length</code>	配列の要素の数。このメンバーに値を代入すると配列の大きさが変えられる。
<code>join(separator)</code>	配列を文字列に変換する。 <code>separator</code> はオプションの引数で、省略された場合はカンマ、である。
<code>concat(i1,i2,...)</code>	指定した引数の値をもとの配列に付け加えた配列を新たに作成する。引数が配列の場合は配列の要素を付け加える。元の配列は変化しない。
<code>sort([func])</code>	配列の要素をアルファベット順に並べ替える。 <code>func</code> は並べ替えを指定するための関数である。
<code>indexOf(value[,start])</code>	<code>start</code> 以降(指定しない場合は 0)の要素で <code>value</code> の値と等しい(==)最初のインデックスを返す。見つからない場合は-1。
<code>lastIndexOf(value[,start])</code>	<code>start</code> 以前(指定しない場合は配列の最後)の要素で <code>value</code> の値と等しい(==)最初のインデックスを返す。見つからない場合は-1。
<code>pop()</code>	配列の最後の要素を削除し、その値を返す。配列をスタックとして利用できる。
<code>push(i1,i2,...)</code>	引数で渡された要素を配列の最後に付け加える。配列をスタックやキューとして利用できる。
<code>shift()</code>	配列の最初の要素を削除し、その値を返す。配列をキューとして利用できる。
<code>unshift(i1,i2,...)</code>	引数で渡された要素を配列の最初の要素とする。
<code>reverse()</code>	与えられた配列の要素を逆順に並べ替えたものに変更する。

次ページへ続く

表 C.3: 配列のメソッド (続き)

メンバー	説明
<code>slice(start[,end])</code>	<code>start</code> から <code>end</code> の前の位置にある要素を取り出した配列を返す。 <code>end</code> がないときは配列の最後までを指定したことになる。元の配列は変化しない。
<code>splice(start,No,i1,i2,...)</code>	<code>start</code> の位置から <code>No</code> 個の要素を取り除き、その位置に <code>i1,i2,...</code> 以下の要素を付け加える。
<code>forEach(func)</code>	引数 <code>func</code> に与えられた関数を配列の各要素に対して実行する。関数の引数は配列の値、配列の <code>index</code> 、配列の順となる。与えられた関数の戻り値は無視される。途中でループの処理を中断できない。
<code>map(func)</code>	引数に与えられた関数を配列の各要素に対して実行し、関数の戻り値からなる新しい配列を作成する。引数として与えられた関数の引数は <code>forEach</code> と同じである。
<code>reduce(func[,initial])</code>	<code>func</code> は 2 つの仮引数をとる関数。 <code>initial</code> がないときは <code>func</code> が初めて呼び出されるときは配列の 1 番目と 2 番目の要素が引数として与えられる。2 回目以降の呼び出しでは 1 番目の引数はその前に呼び出された関数の戻り値が使用される。 <code>initial</code> があるときは初めての呼び出しで 1 番目の引数として使われる。
<code>reduceRight(func[,initial])</code>	<code>reduce</code> と同様のメソッド。配列の要素のアクセスが大きい方から小さい方になる。
<code>every(func)</code>	<code>func()</code> を各配列の要素に適用し、すべてが <code>true</code> と解釈される値のとき、 <code>true</code> を返す。どこか一つで <code>false</code> のときにはそこで実行が打ち切られ、 <code>false</code> の値が返る。 <code>func</code> の引数は <code>forEach</code> と同じである。
<code>some(func)</code>	<code>func()</code> を各配列の要素に適用し、どれかひとつが <code>true</code> と解釈される値のとき、 <code>true</code> を返す。どこか一つで <code>true</code> のときにはそこで実行が打ち切られる。すべての結果が <code>false</code> のときは <code>false</code> となる。 <code>func</code> の引数は <code>forEach</code> と同じである。
<code>filter(func)</code>	<code>func</code> で呼び出される関数の戻り値が <code>true</code> に変換される要素を集めて新しい配列を作成する。 <code>func</code> の引数は <code>forEach</code> と同じである。

`slice` と `splice` は似たメソッドですが、引数の意味が異なるところがあるので注意が必要です。

```
1 >A=[1,2,3,4,5,6,7,8,9];
2 [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```

3 >A.slice(2,4)
4 [3, 4]
5 >A;
6 [1, 2, 3, 4, 5, 6, 7, 8, 9]
7 >A.splice(2,4)
8 [3, 4, 5, 6]
9 >A;
10 [1, 2, 7, 8, 9]
11 >A.splice(2,0,"a","b")
12 []
13 >A;
14 [1, 2, "a", "b", 7, 8, 9]

```

- `slice` の引数は配列から取り出す開始位置と終了位置の直後までを指定します。したがって、3 行目の例では 2 番目と 3 番目の要素が切り出されます。また、5 行目の実行例からも元の配列が変化していないことがわかります。
- `splice` の 1 番目の引数は配列から取り出す開始位置で、2 番目の引数は `slice` とは異なり、取り出す個数です (7、8 行目参照)。
- `splice` を行った配列は戻り値の部分が取り除かれています (9、10 行目)。
- `splice` は追加の引数で取り除いた位置に挿入する要素を指定できます。

`forEach` と `map` の引数で与えられる関数は 3 つの引数 `value`、`index` と `array` をとることができます。したがって、次の関係が成立します。

```
array[index] = value
```

元の配列の要素を変更しないのであれば、3 つ目の引数 `array` は必要ありません。

```

>A=[1,2,3];
A.forEach(function(V, i, AA) {AA[i] = V*V;});
>A;
[1, 4, 9]

```

この例では、元の配列の各要素を 2 乗して元の配列に格納しています。

これに対し、`map` では新しい配列を作成します。

```

>A=[1,2,3];
[1, 2, 3]
>A.map(function(V) {return V*V;});
[1, 4, 9]
>A;
[1, 2, 3]

```

次の例は `reduce` の実行例です。

```
>[1,2,3].reduce(function(x,y){return x+y;});
```

```
6
```

与えられた関数は引数の和を求めており、全体としては配列の中の要素の和が求められています。

問題 C.2 次のプログラムを実行したときの結果を述べなさい。

```
A=[3,1,4,5,8,10]
A.filter(function(X) {return X%2;});
A.reduce(function(x,y){ return x+y%2;});
A.reduce(function(x,y){ return x+y%2;},0);
```

C.5.3 で `arguments` の総和をとる関数を作成しています。この部分を `reduce` で置き換えるとうまくいきません。

```
>function sum() {return arguments.reduce(function(x,y){return x+y;})}
undefined
>sum(1,2,3);
VM510:2 Uncaught TypeError: arguments.reduce is not a function(...)
```

「`arguments` には `reduce` という関数がない」というメッセージが表示されています。これからも、`arguments` は配列のようなものだけれども配列とは違うことがわかります。

これを解決するにはオブジェクトのメソッドとして関数を呼び出す `call` を使用します。

```
>function sum(){return Array.prototype.reduce.call(arguments,function(x,y){return x+y;})}
undefined
>sum(1,2,3);
6
```

`reduce` は `Array` オブジェクトの `prototype` で定義されている関数です。それを `arguments` のメソッドとして呼び出すために `call` を用いています。`reduce` の引数として関数を必要としますので、それを無名関数として 2 番目の引数に与えています。

`document.getElemensByTagName` などで得られた配列のようなものに対して `forEach` などを利用したい場合にもこの手法が利用できます。

C.8 オブジェクト

C.8.1 配列とオブジェクト

配列はいくつかのデータをまとめて一つの変数に格納しています。各データを利用するためには `foo[1]` のように数による添え字を使います。これに対し、オブジェクトでは添え字に任意の文字列を使うことができます。

次の例はあるオブジェクトを定義して、その各データにアクセスする方法を示しています。

```
var person = {
  name : "foo",
  birthday :{
    year : 2001,
    month : 4,
    day : 1
  },
  "hometown" : "神奈川",
}
```

- オブジェクトは全体を {} で囲みます。
- 各要素はキーと値の組で表されます。両者の間は : で区切れます。
- キーは任意の文字列でかまいません。キー全体を "" で囲わなくてもかまいません。
- 値は JavaScript で取り扱えるデータなあらば何でもかまいません。上の例ではキー birthday の値がまたオブジェクトとなっています。
- 各要素の値を取り出す方法は 2 通りあります。

一つは、演算子を用いてオブジェクトのキーをそのあとに書きます。もう一つは配列と同様に [] 内にキーを文字列として指定する方法です。

```
>person.name;
"foo"
>person["name"];
"foo"
```

オブジェクトの中にあるキーをすべて網羅するようなループを書く場合や変数名として利用できないキーを参照する場合には後者の方法が利用されます。

- キーの値が再びオブジェクトであれば、前と同様の方法で値を取り出せます。

```
>person.birthday;
Object {year: 2001, month: 4, day: 1}
>person.birthday.year;
2001
>person.birthday["year"];
2001
```

この例のように取り出し方は混在しても問題ありません。

- キーの値は代入して変更できます。

```
>person.hometown;
"神奈川"
>person.hometown="北海道";
"北海道"
>person.hometown;
"北海道"
```

- 存在しないキーを指定すると値として `undefined` が返ります。

```
>person.mother;
undefined
```

- 存在しないキーに値を代入すると、キーが自動で生成されます。

```
>person.mother = "aaa";
"aaa"
>person.mother;
"aaa"
```

- オブジェクトのキーをすべて渡るループは `for-in` で実現できます。

- `for(v in obj)` の形で使用します。変数 `v` はループ内でキーの値が代入される変数、`obj` はキーが走査されるオブジェクトです。
- キーの値は `obj[v]` で得られます。

```
>for(i in person) { console.log(i+" "+person[i]);}
name foo
birthday [object Object]
hometown 北海道
mother aaa
undefined
```

最後の `undefined` は `for` ループの戻り値です。

なお、オブジェクトを {} の形式で表したものをおbjectリテラルとよびます。

C.8.2 コンストラクタ関数

オブジェクトを定義する方法としてはコンストラクタ関数を使う方法がある。次の例はコンストラクタ関数を用いて、前の例と同じオブジェクト（インスタンス）を構成しています。

```
function Person(){
  this.name = "foo";
  this.birthday = {
```

```

    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
}

```

- 通常、コンストラクタ関数は大文字で始まる名前を付けます。
- そのオブジェクト内にメンバーを定義するために、`this`をつけて定義します。ここでは、前の例と同じメンバー名で同じ値を設定しています。
- この関数には`return`がないことに注意すること。
- この関数を用いてオブジェクトを作成するためには、`new`をつけて関数を呼び出します。

```
>var person = new Person();
undefined
```

- 元来、戻り値がないので`undefined`が表示されているが、オブジェクトは作成されています。
- 前と同じ文を実行すれば同じ結果が得られます。

この例はコンストラクタ関数に引数がないが、引数を持つコンストラクタ関数も定義が可能です。これにより同じメンバーを持つオブジェクトをいくつか作る必要がある場合にプログラムが簡単になります。

次の例はコンストラクタ関数を`new`を用いないで実行した場合です。

```
>p = Person();
undefined
>p;
undefined
>name;
"foo"
>>window.name;
"foo"
>birthday == window.birthday
true
```

- この関数は戻り値がないので、`undefined`が変数`p`に代入されます。
- このとき、キーワード`this`が指すのはグローバルオブジェクトです。
- 現在の実行環境はブラウザ上なので、このときのグローバルオブジェクトは`window`です。

- このとき、グローバル変数はすべてグローバルオブジェクトのメンバーとしてアクセス可能です。この例では `this.name` に値を代入した時点で変数 `name` が定義されています。
- 最後の例からも、`name` と `window.name` が同じものであることがわかります。

問題 C.3 上の実行例を次のように変えます。

```
function Person(D){
  this.name = "foo";
  this.birthday = {
    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
  return D;
}
```

これに対して次のように実行したとき、作成されるオブジェクトは何か答えなさい。

1. `p = new Person(1);`
2. `p = new Person([1,2,3]);`
3. `p = new Person({o:"1"});`
4. `p = new Person(function(){return 2;});`
5. `p = new Person(new function(){this.a = "a"});`

constructor プロパティ

オブジェクトが作成されると、`constructor` プロパティとよばれる特殊なプロパティも設定されます。このプロパティはオブジェクトを作成したときに使われたコンストラクタ関数を返します。

```
>var p = new Person();
undefined
>p.name;
"foo"
>p.constructor;
function Person(){
  this.name = "foo";
  this.birthday = {
    year : 2001,
```

```

    month : 4,
    day : 1
};

this["hometown"] = "神奈川";
}

```

このプロパティに含まれるのは関数なので、コンストラクタの名前を知らなくても、元と同じオブジェクトのコピーが作成できます。

```

>np = new p.constructor();
Person {name: "foo", birthday: Object, hometown: "神奈川"}
>np.constructor;
function Person(){
  this.name = "foo";
  this.birthday = {
    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
}

```

オブジェクトリテラルを使ってオブジェクトを作ると、組み込み関数の
\ElmJ{Object()} コンストラクタ関数がセットされます。

```

\begin{Verbatim}
>o = {}
Object {}
>o.constructor;
function Object() { [native code] }

```

このプロパティは for-in ループ内では表示されません。

instanceof 演算子

instanceof 演算子はオブジェクトを生成したコンストラクタ関数が指定されたものかを判定できます。次の結果は Opera 29 で C.8.2 を実行した後にさらに実行したものです。

```

>p instanceof Person
true
>p instanceof Object;
true
>o instanceof Object;
true

```

```
>o instanceof Person  
false
```

C.8.3 オブジェクトリテラルと JSON

JSON(JavaScript Object Notation) はデータ交換のための軽量なフォーマットです。形式は JavaScript のオブジェクトリテラルの記述法と全く同じです。

- 正しく書かれた JSON フォーマットの文字列をブラウザとサーバーの間でデータ交換の手段として利用できます。
- JavaScript 内で、JSON フォーマットの文字列を JavaScript のオブジェクトに変換できます。
- JavaScript 内のオブジェクトを JSON 形式の文字列に変換できます。

JavaScript のオブジェクトと JSON フォーマットの文字列の相互変換の手段を提供するのが JSON オブジェクトです。

次の例は 2 つの同じ形式からなるオブジェクトを通常の配列に入れたものを定義しています。

```
var persons = [ {  
    name : "foo",  
    birthday : { year : 2001, month : 4, day : 1 },  
    "hometown" : "神奈川" ,  
,  
{  
    name : "Foo",  
    birthday : { year : 2010, month : 5, day : 5 },  
    "hometown" : "北海道" ,  
}];
```

次の例はこのオブジェクトを JSON に処理させたものです。

```
>s = JSON.stringify(persons);  
"[{"name":"foo","birthday":{"year":2001,"month":4,"day":1},  
"hometown":"神奈川"},  
{"name":"Foo","birthday":{"year":2010,"month":5,"day":5},  
"hometown":"北海道"}]"  
>s2 = JSON.stringify(persons, ["name", "hometown"]);  
"[{"name":"foo","hometown":"神奈川"}, {"name":"Foo","hometown":"北海道"}]"  
>o = JSON.parse(s2);  
[Object, Object]  
>o[0];  
Object {name: "foo", hometown: "神奈川"}
```

- JavaScript のオブジェクトを文字列に変更する方法は `JSON.stringify()` を用います。このまま見ると"がおかしいように見えるが表示の関係でそうなっているだけです。なお、結果は途中で改行を入れているが実際は一つの文字列となっています。
- `JSON.stringify()` の二つ目の引数として対象のオブジェクトのキーの配列を与えることができます。このときは、指定されたキーのみが文字列に変換されます。
- ここでは、"name" と "hometown" が指定されているので "birthday" のデータは変換されません。
- JSON データを JavaScript のオブジェクトに変換するための方法は `JSON.parse()` を用います。
- ここではオブジェクトの配列に変換されたことがわかります。
- 各配列の要素が正しく変換されていることがわかります。

問題 C.4 上の実行例で

```
s3 = JSON.stringify(persons, ["year"]);
```

としたときの結果はどうなるか調べなさい。

C.8.4 ECMAScript5 のオブジェクト属性

オブジェクト指向言語におけるプロパティとメソッドの属性

オブジェクト指向言語ではオブジェクトのプロパティやメソッドは次のように分類されます。

- インスタンスフィールド
インスタンスごとに異なる値を保持できるプロパティ
- インスタンスマソッド
クラスのすべてのインスタンスで共有されるメソッド
- クラスフィールド
クラスに関連付けられたプロパティ
- クラスマソッド
クラスに関連付けられたメソッド

JavaScript では関数もデータなのでフィールドとメソッドに厳密な区別はないのでフィールドとメソッドは同一視できます。また、`prototype` を用いればクラスフィールドなども作成できます。

通常、オブジェクト指向の言語ではフィールドを勝手に操作されないようにするために、フィールドを直接操作できなくして、値を設定や取得するメソッドを用意します。そのために、フィールドにアクセスするため記述が面倒になるという欠点もあります。

一方、プロパティの代入の形をとっても実際はゲッターやセッター関数を呼ぶ形になっている言語もあります。JavaScript の最新版 1.8.1 以降ではこの方式が可能となっています。

プロパティ属性

JavaScript は Ecma International が定義している ECMAScript の仕様に基づいています。2014 年現在、最新バージョンの ECMAScript 5.1 の仕様は ECMA-262² で公開されています。

このバージョンではオブジェクトのプロパティやメソッドにプロパティ属性という機能が追加されました。オブジェクトのプロパティの属性には表 C.4 のようなものがあります。また、メソッド

表 C.4: プロパティの属性のリスト

属性名	値の型	説明	デフォルト値
value	任意のデータ	プロパティの値	undefined
writable	Boolean	false のときは value の変更ができない	false
enumerable	Boolean	true のときは for-in ループでプロパティが現れる。	false
configurable	Boolean	false のときはプロパティを消去したり、value 以外の値の変化ができない	false

に関しては表 C.5 のものがあります。

表 C.5: メソッドの属性のリスト

属性名	値の型	説明	デフォルト値
get	オブジェクトまたは未定義	関数オブジェクトでなければならない。プロパティの値が読みだされるときに呼び出される	undefined
set	オブジェクトまたは未定義	関数オブジェクトでなければならない。プロパティの値を設定するときに呼び出される	undefined
enumerable	Boolean	true のときは for-in ループでプロパティが現れる。	false
configurable	Boolean	false のときはプロパティを消去したり、value 以外の値の変化ができない	false

これらの属性を使うとオブジェクトのプロパティの呼び出しや変更に関して、いわゆるゲッターフィールドやセッターフィールドを意識しないで呼び出すことが可能となります。次の例は [8] にある 9 章の例 9.2 の Range を改良した例 9.18 と 9.21 をまとめたものです。この例はさらにコンストラクタやセッターのところで不適切な値が設定されないようにチェックを加えています。

²<http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>

```

1  function Range(from, to) {
2      if(from > to ) throw Error("Range: from must be <= to");
3      function getF() { return from;};
4      function setF(v) {
5          if(v <= to ) from = v;
6          else throw Error("Range: from must be <= to");
7      };
8      function getT() {return to;};
9      function setT(v) {
10         if(v >= from ) to = v;
11         else throw Error("Range: from must be <= to");
12     }
13     Object.defineProperty(this, "from",
14         {get: getF, set: setF, enumerable:true, configurable:false});
15     Object.defineProperty(this, "to",
16         {get: getT, set: setT, enumerable:true, configurable:false});
17 }

```

- 前と同様に、二つの引数を持つ Range 関数を作成します。
- 2 行目では下限の値が上限の値より大きくなったらエラーを発生させています。
- 3 行目から 12 行目では下限 (from) と上限 (to) のゲッターとセッターを定義しています。各セッターではコンストラクタと同様に値に矛盾が起きていたらエラーを発生させるようにしています。
- 13 行目から 16 行目で作成するインスタンスのプロパティ from のオブジェクト属性を defineProperty() を用いて定義しています。この関数の引数は次のとおりです。
 - 一番目の引数はプロパティを設定するオブジェクト
 - 2 番目の引数はプロパティの名前。ここでは文字列で与えている。
 - 3 番目の引数は設定するプロパティ属性。ここではゲッター関数とセッター関数を指定し、for-in ループで列挙可能にし、関数の置き換えや再設定ができないように設定しています。
- 15 行目から 16 行目では同様に作成するインスタンスのプロパティ to のオブジェクト属性を設定しています。

```

Range.prototype = {
    includes : function(v) {
        return this.from <= v && v <= this.to;
    },
    foreach : function(f) {

```

```

    for(var k = Math.ceil(this.from); k<= this.to; k++) f(k);
},
toString : function() { return "[" + this.from+","...,"+this.to+"]";}
};

Object.defineProperties(Range.prototype,
{ includes : {enumerable : false},
foreach : {enumerable : false},
toString : {enumerable : true}
});

```

- 18 行目から 26 行目までは前と同じく `prototype` にクラスメソッドを定義しています。
- これらのクラスメソッドの一部を列挙可能にしないために `definePropaties()` を用いて設定しています。この関数は `definePropaty()` が一つのプロパティごとに設定するのに対し、複数のプロパティに対して設定が可能です。なお、ここでは機能を確かめるため設定の値を変えています。
 - 一番目の引数は設定するオブジェクト
 - 2 ア番目の引数は設定するプロパティをキーとし、設定する属性のリストを表すオブジェクトの値

これらの設定が正しく動作しているか検証します。

- オブジェクトを作成し、プロパティを列挙します。

```

>r = new Range(1,5);
Range {from: (...), to: (...), toString: function}
>for(key in r) console.log(key+":"+r[key]);
from:1
to:5
toString:function () { return "[" + this.from+","...,"+this.to+"]";}
undefined
>r.includes;
function (v) {
    return this.from <= v && v <= this.to;
}

```

- `include` と `foreach` の `enumerable` のプロパティを `false`、`toString` の `enumerable` のプロパティを `true` に設定したので、メソッドは `toString` しか表示されません。
- 関数が存在することは確認できます。

- 各メソッドが正しく動作するか確認します。

```
>r.includes(3);
true
>r.includes(10);
false
```

以前と同じ動作をしています。

- プロパティに代入します。

```
>r.from = 10;
Uncaught Error: Range: from must be <= to
>r.from = -5;
-5
>r.from;
-5
```

- 上限より大きな値を下限に設定するとエラーが起きます。get で指定した関数が動作していることがわかります。
- 条件を満たす値を設定すれば、正しく設定されます。

念のため、オブジェクトの値がどうなっているか確認します。

```
>for(key in r) console.log(key+": "+r[key]);
from:-5
to:5
toString:function () { return "[" + this.from+",...,"+this.to+"]";}
undefined
```

- プロパティが削除できるか確認します。

```
>delete r.from;
false
>r.from
-5
```

`delete` の結果が `false` なので、取り除きに失敗しています。値は元の値のままでです。

問題 C.5 上の実行例に対して次のことをしなさい。

1. 各メソッドが正しく動くことを確認しなさい。
2. 13 行目から 16 行目にある 2 つの `Object.defineProperty()` 関数を `Object.defineProperties()` 関数で置き換えなさい。
3. 3 行目から 12 行目で定義されている関数はグローバルな関数か答えなさい。また、13 行目から 16 行目の部分は関数 `Range()` の外に記述してもよいか答えなさい。

4. 3 行目から 12 行目で定義されている関数を無名関数にしてゲッターとセッターを定義しなさい。
5. 関数 Range() 内にある変数 from と to はどこで定義されているか答えなさい。
6. 関数 Range() 内にある変数 from と to と this.from、this.to は同じものを指すか答えなさい。たとえば、3 行目の from を this.from としたら何が起こるか確認しなさい。
7. 上記の実行例で delete r.from が失敗する理由を説明しなさい。

C.8.5 エラーオブジェクトについて

前節の例ではオブジェクトの条件に合わない値を設定すると、エラーを発生するようにしています。ここではエラーオブジェクトについて詳しく説明します。次の例は前の実行例に、実行時に上限と下限の値を設定するためのテスト関数です。コンソールから var r = test() などで実行すると上限と下限の値が正しくなるまで繰り返されます。

なお、このプログラムはブラウザ上で実行されることを想定しています。

```
function test() {
  var f, t;
  for (;;) {
    try {
      f = Number(prompt("区間の下限の値を入力してください"));
      t = Number(prompt("区間の上限の値を入力してください"));
      return new Range(f,t);
    } catch(e) {
      console.log(e.name+": "+e.message);
      console.log("from:"+f+", to:"+t);
    }
  }
}
```

- 2 行目でこの関数内で使用するローカル変数を宣言しています。
- 3 行目の for(;;) は初期条件、終了条件、後処理がすべて記述されていない for ループです。これにより無限ループが構成できます。
- エラー処理をする構文が try/catch/finally 構文です。
 - try の後に書かれたブロック (ここでは 5 行目から 7 行目を含むブロック) が通常実行されます。
 - この実行の間エラーが発生しなければ catch の後のブロックは実行されません。
 - エラーが起きると catch(e) に書かれた変数 e にエラーオブジェクトが設定されています。この変数はこのブロック内でしか有効ではありません。

- try ブロック内ではキーボードから 2 つの数を読むため、window オブジェクトの `prompt()` を呼び出しています。
- この関数の戻り値（文字列）を `Number` コンストラクタで数に変換しています（5 行目と 6 行目）。
- 7 行目で作成した `Range` オブジェクトを戻り値としています。
- `Range` コンストラクタでは上限の値が下限の値より小さいときはエラーが発生させています。エラーオブジェクトではエラーが起きたコンストラクタの名前が `name` プロパティに、投げられたエラーで引数に書かれた文字列が `message` プロパティに設定されているので、その内容を 9 行目で表示させていいます。
- その後、コンストラクタを呼んだときの値を表示させています。

次の実行例は、はじめに下限値を 5、上限値を 1 に設定しエラーを表示させ、その後、下限値を 1、上限値を 5 に設定したときのものです。希望したオブジェクトが作成できていることがわかります。

```
>r = test();
Error:Range: from must be <= to
from:5, to:1
Range {from: (...), to: (...), toString: function}
>r.from
1
>r.from =10;
Error: Range: from must be <= to
```

付録D CSSについて

カスケーディングスタイルシート (CSS) は HTML 文書の要素の表示方法を指定するものです。CSS は JavaScript からも制御できます。

文書のある要素に適用されるスタイルルールは、複数の異なるルールを結合 (カスケード) したものです。スタイルを適用するためには要素を選択するセレクタで選びます。

表 D.1 は CSS3 におけるセレクタを記述したものです¹。

表 D.1: CSS3 のセレクタ

セレクタ	解説
*	任意の要素
E	タイプが E の要素
E[foo]	タイプが E で属性 "foo" を持つ要素
E[foo="bar"]	タイプが E で属性 "foo" の属性値が"bar"である要素
E[foo~="bar"]	タイプが E で属性 "foo" の属性値がスペースで区切られたリストでその一つが "bar"である要素
E[foo^="bar"]	タイプが E で属性 "foo" の属性値が"bar"で始まる要素
E[foo\$="bar"]	タイプが E で属性 "foo" の属性値が"bar"で終わる要素
E[foo*="bar"]	タイプが E で属性 "foo" の属性値が"bar"を含む要素
E[foo = "en"]	タイプが E で属性 "foo" の属性値がハイフンで区切られたリストでその一つが "en"で始まる要素
E:root	document のルート要素
E:nth-child(n)	親から見て n 番目の要素
E:nth-last-child(n)	親から見て最後から数えて n 番目の要素
E:nth-of-type(n)	そのタイプの n 番目の要素
E:nth-last-of-type(n)	そのタイプの最後から n 番目の要素
E:first-child	親から見て一番初めの子要素
E:last-child	親から見て一番最後の子要素
E:first-of-type	親から見て初めてのタイプである要素
E:last-of-type	親から見て最後のタイプである要素
E:only-child	親から見てただ一つしかない子要素

次ページへ続く

¹<http://www.w3.org/TR/selectors/>より引用。

表 D.1: CSS3 のセレクタ (続き)

セレクタ	解説
E:only-of-type	親から見てただ一つしかないタイプの要素
E:empty	テキストノードを含めて子要素がない要素
E:link, E:visited	まだ訪れたことがない (:link) が訪れたことがある (visited) ハイパーリンクのアンカーである要素
E:active, E:hover, E:focus	ユーザーに操作されている状態中の要素
E:target	参照 URI のターゲットである要素
E:enabled, E:disabled	使用可能 (:enable) か使用不可のユーザーインターフェイスの要素
E:checked	チェックされているユーザーインターフェイスの要素
E::first-line	要素のフォーマットされたはじめの行
E::first-letter	要素のフォーマットされたはじめの行
E::before	要素の前に生成されたコンテンツ
E::after	要素の後に生成されたコンテンツ
E.warning	属性 class が "warning" である要素
E#myid	属性 id の属性値が "myid" である要素
E:not(s)	単純なセレクタ s にマッチしない要素
E F	要素 E の子孫である要素 F
E > F	要素 E の子である要素 F
E F+	要素 E の直後にある要素 F
E ~ F	要素 E の直前にある要素 F

いくつか注意する点を挙げます。

- 属性 id の属性値の前に#をつけることでその要素が選ばれます。
- 属性 class の属性値の前に. をつけることでその要素が選ばれます。
- nth-child(n) には単純な式を書くことができます。詳しくは実行例 D.1 を参照してください。このセレクタは複数書いてもかまいません。
- E F と E > F の違いを理解しておくこと。たとえば div div というセレクタは途中に別の要素が挟まれていてもかまいません。また、<div>要素が 3 つある場合にはどのような 2 つの組み合わせも対象となります。

問題 D.1 次の HTML 文書において nth-child の () 内に次の式を入れた時どうなるか報告しなさい。ここでは箇条書きの開始を示す要素であり、は箇条書きの各項目を示す要素です。

```
<!DOCTYPE html>
<html>
<head>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nth-child のチェック</title>
<style type="text/css" >
li:nth-child(n){
    background:yellow;
}
</style>
</head>
<body>
<ol>
    <li>1 番目</li>
    <li>2 番目</li>
    <li>3 番目</li>
    <li>4 番目</li>
    <li>5 番目</li>
    <li>6 番目</li>
    <li>7 番目</li>
    <li>8 番目</li>
</ol>
</body>
</html>

```

[firstnumber=1]

1. n(ここでリストの設定)
2. 2n
3. n+3
4. -n+2

問題 D.2 前問のリストに対し、背景色が次のようになるように CSS を設定しなさい。

1. 偶数番目が黄色、基数番目がオレンジ色
2. 1 番目、4 番目、…のように 3 で割ったとき、1 余る位置が明るいグレー
3. 4 番目以下がピンク
4. 下から 2 番目以下が緑色

問題 D.3 次の HTML 文書を考えます。

```

<!DOCTYPE html>
<html>

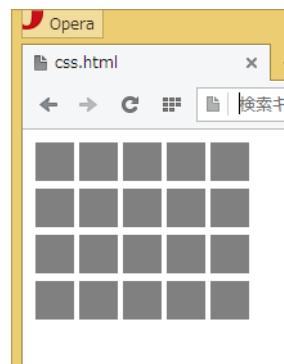
```

付録 46

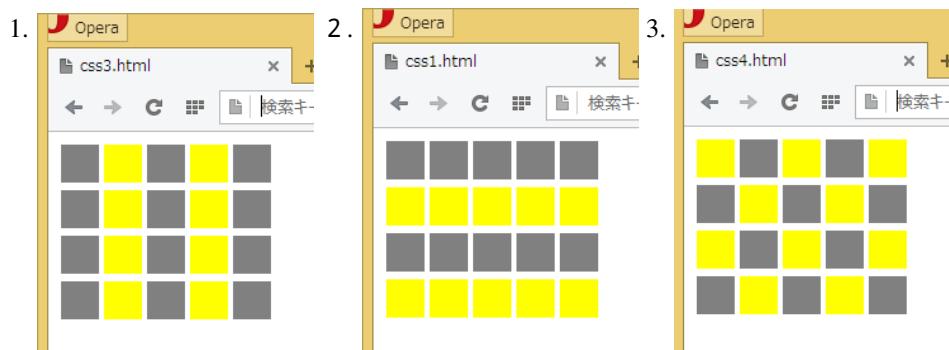
付録 D CSSについて

```
<head><style>
.Row div {
    display:inline-block;
    width:30px;
    height:30px;
    margin: 2px 2px 2px 2px;
    background: gray;
}
</style>
</head>
<body>
<div class="Row"><div></div><div></div><div></div><div></div><div></div></div>
<div class="Row"><div></div><div></div><div></div><div></div><div></div><div></div></div>
<div class="Row"><div></div><div></div><div></div><div></div><div></div><div></div></div>
<div class="Row"><div></div><div></div><div></div><div></div><div></div><div></div></div>
</body>
</html>
```

[firstnumber=1] このリストで表示されるページは次のようにになります。



下図の表示になるように CSS を設定しなさい。



索引 — SVG の用語

Symbols

#(値) 17

A

A(値-図形) 44
 a(値-図形) 44
 alphabetic(値-フォント) 93
 amplitude(フィルタ) 116
 <animate> 79, 80, 83
 <animateColor> 79
 <animateMotion> 73, 78
 <animateTransform> 74, 80
 <animatMotion> 73
 arithmetic(値-フィルタ) 118, 120, 123
 atop(値-フィルタ) 118, 120
 attributeName(アニメーション) 73, 74, 80
 attributeType(アニメーション) 73, 80
 auto(値-アニメーション) 79
 auto(値-フォント) 93
 azimuth(フィルタ) 121

B

BackgroundAlpha(値-フィルタ) 106
 BackgroundImage(値-フィルタ) 106
 baseFrequency(フィルタ) 124, 125
 baseline(値-フォント) 93
 baseline-shift(フォント) 93
 begin(アニメーション) 73, 86, 87
 bevel(値-図形) 41
 black(値-色) 14, 70
 butt(値-図形) 14

C

C(値-図形) 44
 c(値-図形) 44
 calcMode(アニメーション) 73, 83, 90
 <circle> 22, 55, 56, 77
 —— の属性 23
 class 117
 color(アニメーション) 90
 color(図形) 79
 CSS(値) 73
 currentColor(値-色) 79, 90
 cursive(値-フォント) 93
 cx(色) 33
 cx(図形) 23, 87

cy(色) 33
 cy(図形) 23

D

d(図形) 43, 45, 46, 53, 80, 81
 darken(値-フィルタ) 106, 107, 109
 <defs> 17, 18, 25, 26, 61, 79, 86, 99
 discrete(値-アニメーション) 73, 83, 90
 discrete(値-フィルタ) 116
 dominant-baseline(フォント) 92, 93
 dur(アニメーション) 73, 75, 87
 dx(フィルタ) 106
 dy(フィルタ) 106

E

elevation(フィルタ) 121
 <ellipse> 22
 —— の属性 23
 encoding 13
 end(値-フォント) 93
 end(アニメーション) 86
 end(図形) 87
 evenodd(値-色) 40
 exponent(フィルタ) 116

F

fantasy(値-フォント) 93
 feBlend(フィルター) 106–109, 120
 feColorMatrix(フィルター) iv, 109, 110
 <feComponentTransfer> 116
 feComponentTransfer(フィルター) 109, 116
 <feComposite> 118
 feComposite(フィルター) 118, 123
 feComposite(フィルタ内での要素) 118, 120, 122
 feDiffuseLighting(フィルター) 122
 feDiffuseLighting(フィルタ内での要素) 120
 feDistantLight(フィルタ内での要素) 121
 feFlood(フィルター) 116
 feFoold(フィルター) 117
 feFuncA(フィルタ内での要素) 116
 feFuncB(フィルタ内での要素) 116
 feFuncG(フィルタ内での要素) 116
 feFuncR(フィルタ内での要素) 116
 <feGaussianBlur> 103
 feGaussianBlur(フィルター) 101, 103, 105, 117, 123

feImage(フィルター) 106, 108
 feMerge(フィルター) 105, 106, 117, 118
 feMergeNode(フィルター) 106
 feOffset(フィルター) 105, 106, 108
 fePointLight(フィルタ内での要素) 121
 feSpecularLighting(フィルター) 122
 feSpecularLighting(フィルタ内での要素) .120,
 121
 feSpecularLightng(フィルター) 123
 feTurbulence(フィルター) 124
 fill(アニメーション) 73, 75
 fill(図形) 19, 23, 27, 34, 36–38, 40, 42, 45, 61, 62,
 70, 79, 87, 90, 92, 108, 115
 fill-opacity(図形) 36, 37
 fill-rule(図形) 40
 FillPaint(値-フィルタ) 106
 <filter> 102
 filter(値-フィルタ) 103
 filter(フィルタ) 106
 filterUnits(フィルタ) 101
 flood-color(フィルタ) 116
 flood-opacity(フィルタ) 116
 font-family(値-フォント) 93
 font-family(フォント) 93, 95
 font-size(フォント) 93, 95, 97
 font-stretch(フォント) 93
 font-style(フォント) 93
 fractalNoise(値-フィルタ) 124
 freeze(値-アニメーション) 73, 75
 from(アニメーション) 73, 75, 81, 83
 fx(色) 33
 fy(色) 33

G

<g> 13, 14, 17, 18, 32, 74, 77, 79
 gamma(値-フィルタ) 116
 gradientUnits(色) 26, 29, 31–33, 82
 gray(値-色) 14, 87

H

hanging(値-フォント) 93
 height(図形) 13, 19, 20, 61, 63, 70
 height(フィルタ) 101, 102
 hueRotate(値-フィルタ) 109, 112

I

identity(値-フィルタ) 116
 ideographic(値-フォント) 93
 id 17, 24–27, 61, 62, 70, 86, 87, 102
 <image> 67, 70
 Impact(値-フォント) 95
 in(値-フィルタ) 118, 120
 in(フィルタ) 105–107, 118
 in2(フィルタ) 107, 118
 indefinite(値-アニメーション) 73, 76, 90

intercept(値-フィルタ) 116
 italic(値-フォント) 93

K

k1(フィルタ) 118
 k2(フィルタ) 118
 k3(フィルタ) 118
 k4(フィルタ) 118
 keyTimes(アニメーション) 73, 83, 84

L

l(値-図形) 44
 l(値-図形) 44, 46
 lighten(値-フィルタ) 106–108, 120
 lightgrey(値-色) 88
 limitingConeAngle(フィルタ) 121
 <line> 14
 <line>
 — の属性値 14
 line-through(値-フォント) 93
 linear(値-アニメーション) 73
 linear(値-フィルタ) 116
 <linearGradient> 26, 27
 linecap(図形) 14
 linejoin(図形) 41, 42
 luminanceToAlpha(値-フィルタ) 109, 116

M

M(値-図形) 44, 56
 m(値-図形) 44
 <mask> 69–72, 95
 maskUnits(図形) 70
 mask 95
 mathematical(値-フォント) 93
 matrix(値) 60, 111
 matrix(値-フィルタ) 109, 111
 middle(値-フォント) 93, 95
 miter(値-図形) 41
 miterlimit(図形) 41
 mode(フィルタ) 108, 109
 monospace(値-フォント) 93
 mouseout(値-アニメーション) 86
 mouseover(値-アニメーション) 86
 <mpath> 79
 multiply(値-フィルタ) 107

N

none(値-色) 12, 34, 42, 45
 none(値-フォント) 93
 none(図形) 68
 normal(値-フィルタ) 107
 normal(値-フォント) 93
 numOctaves(フィルタ) 124, 125

O

- objectBoundingBox(値-色) 26, 29, 31–33, 35, 62, 70
 objectBoundingBox(値-フィルタ) 101
 oblique(値-フォント) 93
 offset(色) 27
 offset(フィルタ) 116
 opacity(色) 36, 38, 86
 operator(フィルタ) 118, 120
 out(値-フィルタ) 118, 120
 over(値-フィルタ) 118, 120
 overline(値-フォント) 93

P

- paced(値-アニメーション) 73
 <path> ... 43, 45, 46, 53, 58, 62, 80, 81, 86, 97, 100
 path(图形) 78
 <pattern> 61, 62, 66, 68–70
 patternTransform(图形) 66
 patternUnits(图形) 62
 points(图形) 39, 40
 pointsAtX(フィルタ) 121
 pointsAtY(フィルタ) 122
 pointsAtZ(フィルタ) 122
 <polygon> 39, 40, 45
 <polyline> 39, 45, 130
 preserveAspectRatio(图形) 68

Q

- Q(値-图形) 44
 q(値-图形) 44

R

- r(色) 33
 r(图形) 22, 23
 <radialGradiation> 33
 —— の属性 33
 <rect> 18, 19, 76
 —— の属性 19
 red(値) 115
 red(値-色) 11, 79
 remove(値-アニメーション) 73, 75, 87
 repeatCount(アニメーション) 73, 75, 76, 90
 result(フィルタ) 106
 reverse-auto(値-アニメーション) 79
 rgb(値-色) 11, 12, 38
 rotate(値) 13, 17, 56, 57, 60, 76, 97
 rotate(アニメーション) 79
 rotate(图形) 57
 round(値-图形) 14, 41
 rx(图形) 19, 22, 23
 ry(图形) 19, 22, 23

S

- s(値-图形) 44, 53

- sans-serif(値-フォント) 93
 saturate(値) 115
 saturate(値-フィルタ) 109
 scale(値) 13, 40, 56, 61, 76, 77, 80
 screen(値-フィルタ) 107
 seed 124
 serif(値-フォント) 93
 <set> 83
 skewX(値) 58, 61
 skewY(値) 58, 61
 slope(フィルタ) 116
 SourceAlpha(値-フィルタ) 105, 106
 SourceGraphic(値-フィルタ) 106, 108
 specularConstant(フィルタ) 122
 specularExponent(フィルタ) 122
 spline(値-アニメーション) 73
 square(値-图形) 14
 start(値-フォント) 93
 startOffset(フォント) 97, 100
 stdDeviation(フィルタ) 103
 stdDeviation(フィルタ) 103
 <stop> 27, 36, 83
 stop-color(色) 27, 83
 stop-opacity(色) 36, 38
 stroke(图形) 14, 19, 23, 36–38, 79
 stroke-linecap(图形) 12, 14
 stroke-linejoin(图形) 41, 42
 stroke-opacity(色) 36
 stroke-width(图形) 14, 19, 20, 23, 38, 46, 56
 StrokePaint(値-フィルタ) 106
 <style> 103
 sub(値-フォント) 93
 super(値-フォント) 93
 surfaceSca 122
 <svg> 13, 14

T

- T(値-图形) 44
 t(値-图形) 44
 table(フィルタ) 116
 tableValues(フィルタ) 116
 <text> 42, 91, 92, 95, 97
 text-anchor(フォント) 93, 95
 text-decoration(フォント) 93
 <textPath> 97
 to(アニメーション) 73, 75, 81, 83
 transform(値) 74, 75
 transform 13, 17, 25, 40, 56, 60, 74, 76, 97
 —— の属性値 13
 translate(値) 13, 56, 60, 75–77
 <tspan> 92, 94
 turbulence(値-フィルタ) 124
 type(アニメーション) 75, 76

type(フィルタ) 109, 116, 124

U

underline(値-フォント) 93
url(値) 27, 62
url(値-色) 62
<use> 17, 22, 32, 37, 63, 64, 87
userSpaceOnUse(値-色) 26, 29, 32–34, 62, 70, 82
userSpaceOnUse(値-フィルタ) 101, 102

V

values(アニメーション) 73, 83, 84, 90
values(フィルタ) 109, 111, 115
version 13
visibility 83

W

white(値-色) 70, 95
width(図形) 13, 19, 20, 61, 63, 70, 80, 130
width(フィルタ) 101, 102

X

x(図形) 19, 22, 64, 70, 75, 87, 90, 92
x(フィルタ) 101, 102, 121
x1(色) 28, 29, 32, 82, 83
x1(図形) 14
x2(色) 28, 29, 32, 82, 83
x2(図形) 14
xlink:href 17, 97, 100
xMaxYMid(値-図形) 68
XML(値) 73
xmlns:xlink 13
xmlns 13
xor(値-フィルタ) 118, 120

Y

y(図形) 19, 22, 64, 70, 75, 92
y(フィルタ) 101, 102, 121
y1(色) 28, 29, 32, 82
y1(図形) 14
y2(色) 28, 29, 32, 82
y2(図形) 14

Z

z(値-図形) 44, 46, 56
z(フィルタ) 121

あ**値**

..... 17
CSS 73
matrix 60, 111
red 115
rotate 13, 17, 56, 57, 60, 76, 97
saturate 115

scale 13, 40, 56, 61, 76, 77, 80

skewX 58, 61

skewY 58, 61

transform 74, 75

translate 13, 56, 60, 75–77

url 27, 62

XML 73

値-アニメーション

auto 79
discrete 73, 83, 90
freeze 73, 75
indefinite 73, 76, 90
linear 73
mouseout 86
mouseover 86
paced 73
remove 73, 75, 87
reverse-auto 79
spline 73

値-色

black 14, 70
currentColor 79, 90
evenodd 40
gray 14, 87
lightgrey 88
none 12, 34, 42, 45
objectBoundingBox 26, 29, 31–33, 35, 62, 70
red 11, 79
rgb 11, 12, 38
url 62
userSpaceOnUse 26, 29, 32–34, 62, 70, 82
white 70, 95

値-図形

A 44
a 44
bevel 41
butt 14
C 44
c 44
L 44
l 44, 46
M 44, 56
m 44
miter 41
Q 44
q 44
round 14, 41
S 44, 53
s 44
square 14
T 44
t 44
xMaxYMid 68

z 44, 46, 56
値-フィルタ
 arithmetic 118, 120, 123
 atop 118, 120
 BackgroundAlpha 106
 BackgroundImage 106
 darken 106, 107, 109
 discrete 116
 FillPaint 106
 filter 103
 fractalNoise 124
 gamma 116
 hueRotate 109, 112
 identity 116
 in 118, 120
 intercept 116
 lighten 106–108, 120
 linear 116
 luminanceToAlpha 109, 116
 matrix 109, 111
 multiply 107
 normal 107
 objectBoundingBox 101
 out 118, 120
 over 118, 120
 saturate 109
 screen 107
 SourceAlpha 105, 106
 SourceGraphic 106, 108
 StrokePaint 106
 turbulence 124
 userSpaceOnUse 101, 102
 xor 118, 120
値-フォント
 alphabetic 93
 auto 93
 baseline 93
 cursive 93
 end 93
 fantasy 93
 font-family 93
 hanging 93
 ideographic 93
 Impact 95
 italic 93
 line-through 93
 mathematical 93
 middle 93, 95
 monospace 93
 none 93
 normal 93
 oblique 93
 overline 93
 sans-serif 93

serif 93
 start 93
 sub 93
 super 93
 underline 93
アニメーション
 attributeName 73, 74, 80
 attributeType 73, 80
 begin 73, 86, 87
 calcMode 73, 83, 90
 color 90
 dur 73, 75, 87
 end 86
 fill 73, 75
 from 73, 75, 81, 83
 keyTimes 73, 83, 84
 repeatCount 73, 75, 76, 90
 rotate 79
 to 73, 75, 81, 83
 type 75, 76
 values 73, 83, 84, 90

い
色
 cx 33
 cy 33
 fx 33
 fy 33
 gradientUnits 26, 29, 31–33, 82
 offset 27
 opacity 36, 38, 86
 r 33
 stop-color 27, 83
 stop-opacity 36, 38
 stroke-opacity 36
 x1 28, 29, 32, 82, 83
 x2 28, 29, 32, 82, 83
 y1 28, 29, 32, 82
 y2 28, 29, 32, 82

こ
<コールバック関数> 付録 26

す
図形
 color 79
 cx 23, 87
 cy 23
 d 43, 45, 46, 53, 80, 81
 end 87
 fill-opacity 36, 37
 fill-rule 40
 fill 19, 23, 27, 34, 36–38, 40, 42, 45, 61, 62, 70, 79, 87, 90, 92, 108, 115
 height 13, 19, 20, 61, 63, 70

| | |
|---------------------|---------------------------------|
| linecap | 14 |
| linejoin | 41, 42 |
| maskUnits | 70 |
| miterlimit | 41 |
| none | 68 |
| path | 78 |
| patternTransform | 66 |
| patternUnits | 62 |
| points | 39, 40 |
| preserveAspectRatio | 68 |
| r | 22, 23 |
| rotate | 57 |
| rx | 19, 22, 23 |
| ry | 19, 22, 23 |
| stroke-linecap | 12, 14 |
| stroke-linejoin | 41, 42 |
| stroke-width | 14, 19, 20, 23, 38, 46, 56 |
| stroke | 14, 19, 23, 36–38, 79 |
| width | 13, 19, 20, 61, 63, 70, 80, 130 |
| x | 19, 22, 64, 70, 75, 87, 90, 92 |
| x1 | 14 |
| x2 | 14 |
| y | 19, 22, 64, 70, 75, 92 |
| y1 | 14 |
| y2 | 14 |

ふ

フィルタ

| | |
|-------------------|--------------|
| amplitude | 116 |
| azimuth | 121 |
| baseFrequency | 124, 125 |
| dx | 106 |
| dy | 106 |
| elevation | 121 |
| exponent | 116 |
| filter | 106 |
| filterUnits | 101 |
| flood-color | 116 |
| flood-opacity | 116 |
| height | 101, 102 |
| in | 105–107, 118 |
| in2 | 107, 118 |
| k1 | 118 |
| k2 | 118 |
| k3 | 118 |
| k4 | 118 |
| limitingConeAngle | 121 |
| mode | 108, 109 |
| numOctaves | 124, 125 |
| offset | 116 |
| operator | 118, 120 |
| pointsAtX | 121 |
| pointsAtY | 122 |
| pointsAtZ | 122 |

| | |
|------------------|---------------|
| result | 106 |
| slope | 116 |
| specularConstant | 122 |
| specularExponent | 122 |
| stdDeviation | 103 |
| stdDeviation | 103 |
| table | 116 |
| tableValues | 116 |
| type | 109, 116, 124 |
| values | 109, 111, 115 |
| width | 101, 102 |
| x | 101, 102, 121 |
| y | 101, 102, 121 |
| z | 121 |

フィルター

| | |
|---------------------|-------------------------|
| feBlend | 106–109, 120 |
| feColorMatrix | iv, 109, 110 |
| feComponentTransfer | 109, 116 |
| feComposite | 118, 123 |
| feDefuseLighting | 122 |
| feFlood | 116 |
| feFlood | 117 |
| feGaussianBlur | 101, 103, 105, 117, 123 |
| feImage | 106, 108 |
| feMerge | 105, 106, 117, 118 |
| feMergeNode | 106 |
| feOffset | 105, 106, 108 |
| feSpecularLighting | 122 |
| feSpecularLightng | 123 |
| feTurbulence | 124 |

フィルタ内での要素

| | |
|--------------------|---------------|
| feComposite | 118, 120, 122 |
| feDiffuseLighting | 120 |
| feDistantLight | 121 |
| feFuncA | 116 |
| feFuncB | 116 |
| feFuncG | 116 |
| feFuncR | 116 |
| fePointLight | 121 |
| feSpecularLighting | 120, 121 |

フォント

| | |
|-------------------|------------|
| baseline-shift | 93 |
| dominant-baseline | 92, 93 |
| font-family | 93, 95 |
| font-size | 93, 95, 97 |
| font-stretch | 93 |
| font-style | 93 |
| startOffset | 97, 100 |
| text-anchor | 93, 95 |
| text-decoration | 93 |

よ

要素

| | |
|-----------|------------|
| <animate> | 79, 80, 83 |
|-----------|------------|

<animateColor> 79
<animateMotion> 73, 78
<animateTransform> 74, 80
<animatMotion> 73
<circle> 22, 55, 56, 77
<defs> 17, 18, 25, 26, 61, 79, 86, 99
<ellipse> 22
<feComponentTransfer> 116
<feComposite> 118
<feGaussianBlur> 103
<filter> 102
<g> 13, 14, 17, 18, 32, 74, 77, 79
<image> 67, 70
<line> 14
<linearGradient> 26, 27
<mask> 69–72, 95
<mpath> 79
<path> .. 43, 45, 46, 53, 58, 62, 80, 81, 86, 97,
100
<pattern> 61, 62, 66, 68–70
<polygon> 39, 40, 45
<polyline> 39, 45, 130
<radialGradiation> 33
<rect> 18, 19, 76
<set> 83
<stop> 27, 36, 83
<style> 103
<svg> 13, 14
<text> 42, 91, 92, 95, 97
<textPath> 97
<tspan> 92, 94
<use> 17, 22, 32, 37, 63, 64, 87
<コールバック関数> 付録 26

索引 — HTML の用語

B

 7

C

<canvas> 6

class(属性) 103

H

<HTML> 4

そ

属性

class 103

よ

要素

 7

<canvas> 6

<HTML> 4

索引 — JavaScript の用語

Symbols

| | |
|-----------|-------|
| !== | 付録 13 |
| , | 付録 14 |
| == | 付録 13 |
| === | 付録 13 |

A

| | |
|------------------------------|------------------------|
| Ajax | 付録 26 |
| appendChild(DOM のメソッド) | 133 |
| arguments | 付録 18, 付録 17, 付録 22, v |
| argumentscallee | 付録 22 |
| Array のメソッド | |
| concat | 付録 26 |
| every | 付録 27 |
| filter | 付録 27 |
| forEach | 付録 27 |
| indexOf | 付録 26 |
| join | 付録 26 |
| lastIndexOf | 付録 26 |
| length | 付録 26 |
| map | 付録 27 |
| pop | 付録 26 |
| push | 付録 26 |
| reduce | 付録 27 |
| reduceRight | 付録 27 |
| reverse | 付録 26 |
| shift | 付録 26 |
| slice | 付録 27, 付録 28 |
| some | 付録 27 |
| sort | 付録 26 |
| splice | 付録 27, 付録 28 |
| unshift | 付録 26 |

B

| | |
|-------------|-------|
| break | 付録 14 |
|-------------|-------|

C

| | |
|----------------------------------|-------|
| childNodes(DOM のプロパティ) | 134 |
| children(DOM のプロパティ) | 134 |
| cloneNode(DOM のメソッド) | 133 |
| concat(Array のメソッド) | 付録 26 |
| console.log() | 付録 20 |
| createElement(DOM のメソッド) | 133 |
| createElementNS(DOM のメソッド) | 133 |
| createTextNode(DOM のメソッド) | 133 |

D

DOM のプロパティ

| | |
|------------------------------|-----|
| childNodes | 134 |
| children | 134 |
| firstChild | 134 |
| firstElementChild | 134 |
| hasChildNodes | 134 |
| lastChild | 134 |
| lastElementChild | 134 |
| nextElementSibling | 134 |
| nextSibling | 134 |
| nodeName | 134 |
| nodeType | 134 |
| nodeValue | 134 |
| parentNode | 134 |
| previousElementSibling | 134 |
| previousSibling | 134 |

DOM のメソッド

| | |
|------------------------------|-----|
| appendChild | 133 |
| cloneNode | 133 |
| createElement | 133 |
| createElementNS | 133 |
| createTextNode | 133 |
| getAttribute | 133 |
| getElementById | 133 |
| getElementsByClassName | 133 |
| getElementsByName | 133 |
| getElementsByTagName | 133 |
| getNodeName | 133 |
| hasAttribute | 133 |
| insertBefore | 133 |
| querySelector | 133 |
| querySelectorAll | 133 |
| removeAttribute | 133 |
| removeChild | 133 |
| replaceChild | 133 |
| setAttribute | 133 |
| setValue | 133 |

E

| | |
|--------------------------|-------|
| else | 付録 13 |
| every(Array のメソッド) | 付録 27 |

F

| | |
|---------------------------|---------------------|
| false | 付録 10, 付録 11, 付録 27 |
| filter(Array のメソッド) | 付録 27 |

firstChild(DOM のプロパティ) 134
 firstElementChild(DOM のプロパティ) ... 134
 for 付録 15, 付録 14, 付録 25, v
 forEach(Array のメソッド) 付録 27
 function 付録 21

G

getAttribute(DOM のメソッド) 133
 getElementById(DOM のメソッド) 133
 getElementsByClassName(DOM のメソッド) 133
 getElementsByName(DOM のメソッド) 133
 getElementsByTagName(DOM のメソッド) .. 133
 getnodeName(DOM のメソッド) 133

H

hasAttribute(DOM のメソッド) 133
 hasChildNodes(DOM のプロパティ) 134

I

if 付録 13, v
 indexOf 付録 10
 indexOf(Array のメソッド) 付録 26
 Infinity 付録 10
 insertBefore(DOM のメソッド) 133

J

join(Array のメソッド) 付録 26

L

lastChild(DOM のプロパティ) 134
 lastElementChild(DOM のプロパティ) 134
 lastIndexof(Array のメソッド) 付録 26
 length 付録 10, 付録 17
 length(Array のメソッド) 付録 26

M

map(Array のメソッド) 付録 27

N

NaN 付録 10, 付録 16
 nextElementSibling(DOM のプロパティ) .. 134
 nextSibling(DOM のプロパティ) 134
 nodeName(DOM のプロパティ) 134
 nodeType(DOM のプロパティ) 134
 nodeValue(DOM のプロパティ) 134
 null 付録 10

O

objec 付録 11

P

parentNode(DOM のプロパティ) 134
 pop(Array のメソッド) 付録 26
 previousElementSibling(DOM のプロパティ) 134
 previousSibling(DOM のプロパティ) 134

push(Array のメソッド) 付録 26

Q

querySelector(DOM のメソッド) 133
 querySelectorAll(DOM のメソッド) 133

R

reduce(Array のメソッド) 付録 27
 reduceRight(Array のメソッド) 付録 27
 removeAttribute(DOM のメソッド) 133
 removeChild(DOM のメソッド) 133
 replaceChild(DOM のメソッド) 133
 reverse(Array のメソッド) 付録 26

S

setAttribute(DOM のメソッド) 133
 setTimeout() 付録 22
 setValue(DOM のメソッド) 133
 shift(Array のメソッド) 付録 26
 slice(Array のメソッド) 付録 27, 付録 28
 some(Array のメソッド) 付録 27
 sort(Array のメソッド) 付録 26
 splice(Array のメソッド) 付録 27
 split 付録 10
 substring 付録 10
 switch 付録 13, 付録 14, v

T

toString 付録 12
 true 付録 10, 付録 11, 付録 27
 typeof 付録 9

U

undefined 付録 10, 付録 11, 付録 16, 付録 18
 unshift(Array のメソッド) 付録 26

V

var 付録 18, 付録 23

W

while 付録 14, 付録 15, v

<

クロージャ 付録 21

二

コールバック関数 付録 21

せ

整数リテラル 付録 10

と

特別な Number 付録 10

む

無名関数 付録 21

れ

レキシカルスコープ 付録 24

索引 — 一般

B

- Bézier 曲線 50, 97
- Bézier 曲線
 - 円を近似 55
 - 制御点の—— 51

C

- CSS 103, 117

D

- Distant Light 121
- DOM 128

E

- ECMA 付録 9
- ecmascript 付録 9

H

- HTML 4
- HTML5 5

J

- JavaScript 127

P

- PostScript 51

S

- SVG 1

W

- W3C 1
- WebStrage 5
- WHATWG 5
- World Wide Web Consortium 1

X

- XHTML 5
- XML 2

あ

- アウトラインフォント 94
- アニメーション 73
 - イベントを利用した—— 84

い

- 色の指定 11

- 色の対比 21

う

- ヴィントの錯視 44

え

- 円
 - Bézier 曲線で近似 55
 - 属性の—— 22
- エンコーディング 13

お

- 同じ色なのに周りの色で見え方が異なる 19
- 親ノード 128
- 親要素 14
- 折れ線 39

か

- 輝くヘルマン格子 64
- カニツツア錯視 48
- カフェウォール錯視 22, 64, 88

き

- 輝度 116

く

- グラデーション 25
 - 線形—— 26
 - 放射—— 33
- クレイク・オブライエン効果 35

こ

- 子ノード 128
- コフカリング 29
- コメント (SVG) 11
- 子要素 14

さ

- 彩度 115

錯視

- 色の対比 21
- ヴィントの—— 44
- 同じ色なのに周りの色で見え方が異なる 19
- 輝くヘルマン格子 64
- カニツツア—— 48
- カフェウォール—— 22, 64, 88
- クレイク・オブライエン効果 35

| | |
|----------------------------|---------|
| コフカリング | 29 |
| ザバーニョの—— | 27, 83 |
| ザヴィニーの—— | 62 |
| シェパーードの—— | 42 |
| ジャッドの—— | 76 |
| 主観的輪郭のネオン輝度現象 | 50 |
| デルブーフの—— | 25 |
| バーゲンのきらめき効果 | 103 |
| ひし形を用いたクレイク・オブライエン効果
29 | |
| ピラミッドの稜線 | 38 |
| フィックの—— | 14, 75 |
| 浮動する円 | 65 |
| ブルドンの—— | 43 |
| ヘルマン格子 | 21, 103 |
| ポッケンドルフの—— | 18, 86 |
| マッハバンド—— | 27 |
| ミューラー・ライヤーの—— | 15 |
| モーガンのねじれひも | 64 |
| ゆがんだ正方形 | 88 |
| ザバーニョの錯視 | 27, 83 |
| ザヴィニーの錯視 | 62 |
| し | |
| シェパーードの錯視 | 42 |
| 色相 | 112 |
| ジャッドの錯視 | 76 |
| 主観的輪郭のネオン輝度現象 | 50 |
| せ | |
| 制御点 | |
| —— の Bézier 曲線 | 51 |
| 線形グラデーション | 26 |
| そ | |
| 属性 | 13 |
| —— の円 | 22 |
| —— の楕円 | 22 |
| 属性値 | 13 |
| た | |
| 楕円 | 22 |
| —— の一部となる曲線 | 46 |
| 属性の—— | 22 |
| 多角形 | 39 |
| タグ | 4 |
| ち | |
| 長方形 | |
| —— の属性 | 18 |
| 直線 | 14 |
| —— の属性 | 14 |
| て | |
| テキストエディタ | 7 |

| | |
|----------------------|-------------|
| テキストノート | 129 |
| デルブーフの錯視 | 25 |
| 点光源 | 121 |
| な | |
| 名前空間 | 13, 133 |
| の | |
| ノード | 128 |
| 親—— | 128 |
| 子—— | 128 |
| テキスト—— | 129 |
| 要素—— | 129 |
| ルート—— | 129 |
| は | |
| バーゲンのきらめき効果 | 103 |
| ひ | |
| ひし形を用いたクレイク・オブライエン効果 | 29 |
| ピラミッドの稜線 | 38 |
| ふ | |
| フィックの錯視 | 14, 75 |
| フィルタ | 101 |
| 浮動する円 | 65 |
| 不透明度 | 35, 70, 105 |
| ブルドンの錯視 | 43 |
| プロパティ | 132 |
| へ | |
| ヘルマン格子 | 21, 103 |
| 変数のスコープ | 付録 18 |
| ほ | |
| 放射グラデーション | 33 |
| ポッケンドルフの錯視 | 18, 86 |
| ま | |
| マッハバンド錯視 | 27 |
| み | |
| ミューラー・ライヤーの錯視 | 15 |
| め | |
| メソッド | 132 |
| も | |
| モーガンのねじれひも | 64 |
| 文字列 | |
| —— の属性 | 92 |
| —— を表示する | 91 |
| ゆ | |
| ゆがんだ正方形 | 88 |

よ

要素ノード 129

る

ルートノード 129

ルート要素 13