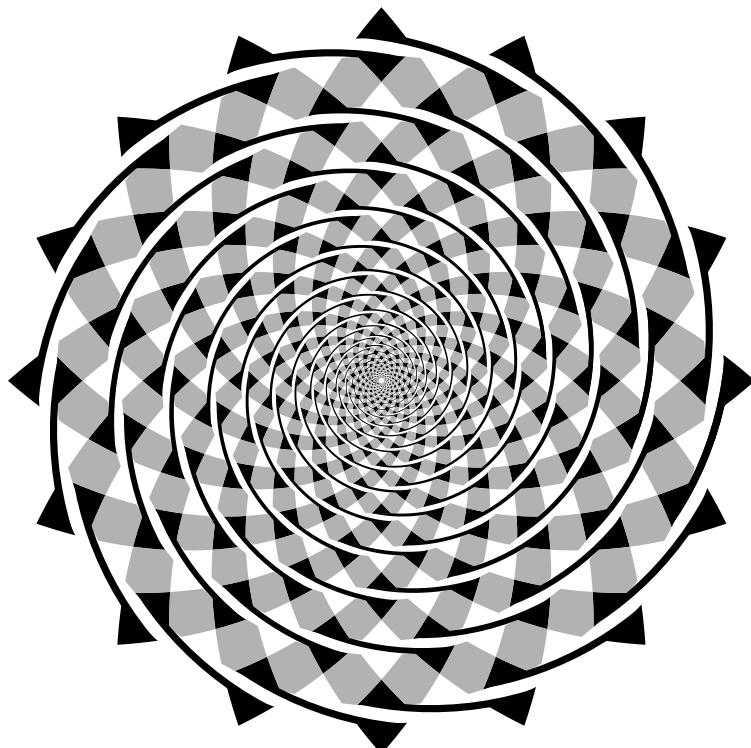


SVG ではじめる Web Graphics 2017年度版
(情報メディア専門ユニットI)
平成 29 年 3 月 24 日改訂



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	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 円と楕円	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 楕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)	77
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	87
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 画像を単一色で塗りつぶす (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	121
6.8.3 feSpecularLighting フィルタ	122
6.9 feTurbulence フィルタ	123

第 7 章 JavaScript を利用した SVG 図形の生成	125
7.1 インターラクティブな SVG を作成するための準備	125
7.1.1 JavaScript	125
7.1.2 DOM(Docunment Object Model)	126
7.1.3 DOM のメソッドとプロパティ	131
7.2 イベント	132
7.2.1 イベント概説	132
7.2.2 SVG 文書や HTML における代表的なイベント	133
7.3 JavaScript による SVG 文書のマウスイベントの処理	134
7.3.1 マウスに関するイベント処理	134
7.3.2 イベントの処理の詳細	144
7.3.3 マウスのドラッグを処理する	146
7.3.4 オブジェクトを追加する	152
7.4 起動時に JavaScript で要素を作成する	156
7.4.1 任意の形の曲線を描く	156
7.4.2 JavaScript のオブジェクトと JSON	158
7.4.3 SVG のオブジェクトを操作するための関数	161
7.4.4 ツェルナーの錯視図形	163
7.4.5 一定時間経過後に関数を呼び出す(自前でアニメーションを作成する)	168
7.5 HTML 文書とその中にある SVG 要素の間でデータ交換する	175
7.6 より進んだ JavaScript プログラミング	184
7.6.1 配列のメソッドを使う	184
7.6.2 クロージャの利用によるグローバル変数の個数の減少	187
7.6.3 WebStorage	191

付録 A SVG の色名	付録 1
---------------------	-------------

付録 B 参考文献について	付録 5
----------------------	-------------

索引	付録 10
SVG の用語	付録 10
HTML の用語	付録 17
JavaScript の用語	付録 19
一般	付録 22

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。

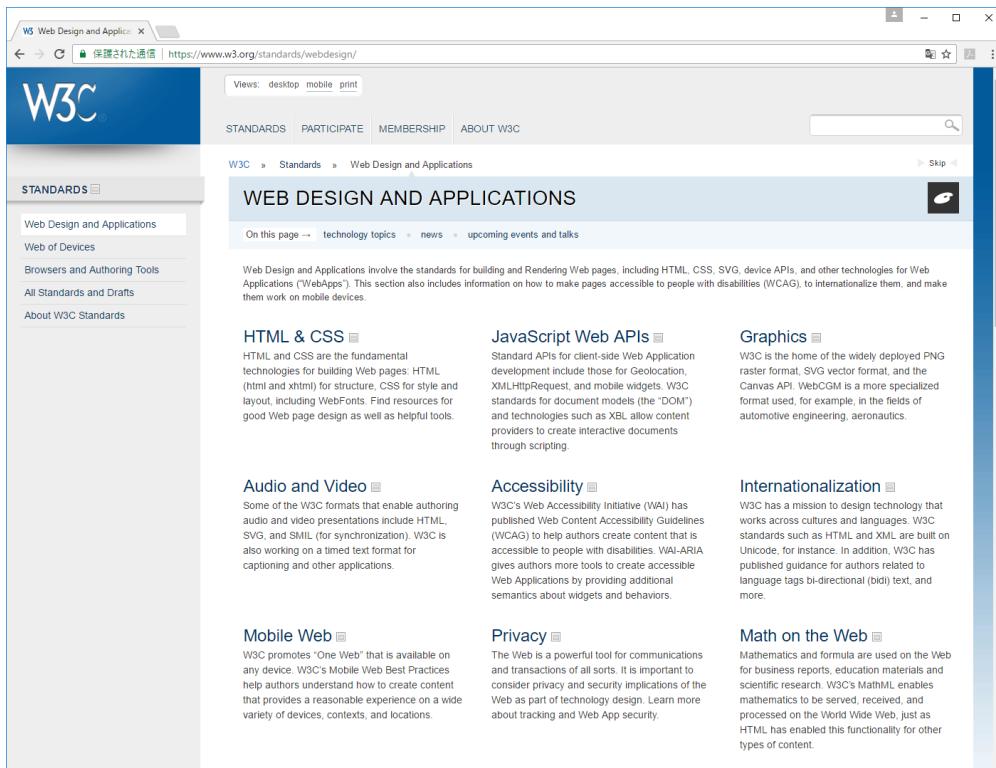


図 1.1: World Wide Web Consortium の Web Design and Application のページ (2017/2/27 参照)

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

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

make pages accessible to people with disabilities (WCAG), to internationalize them, and make them work on mobile devices.

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。

The screenshot shows the W3C Graphics page at <https://www.w3.org/standards/webdesign/graphics>. 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)

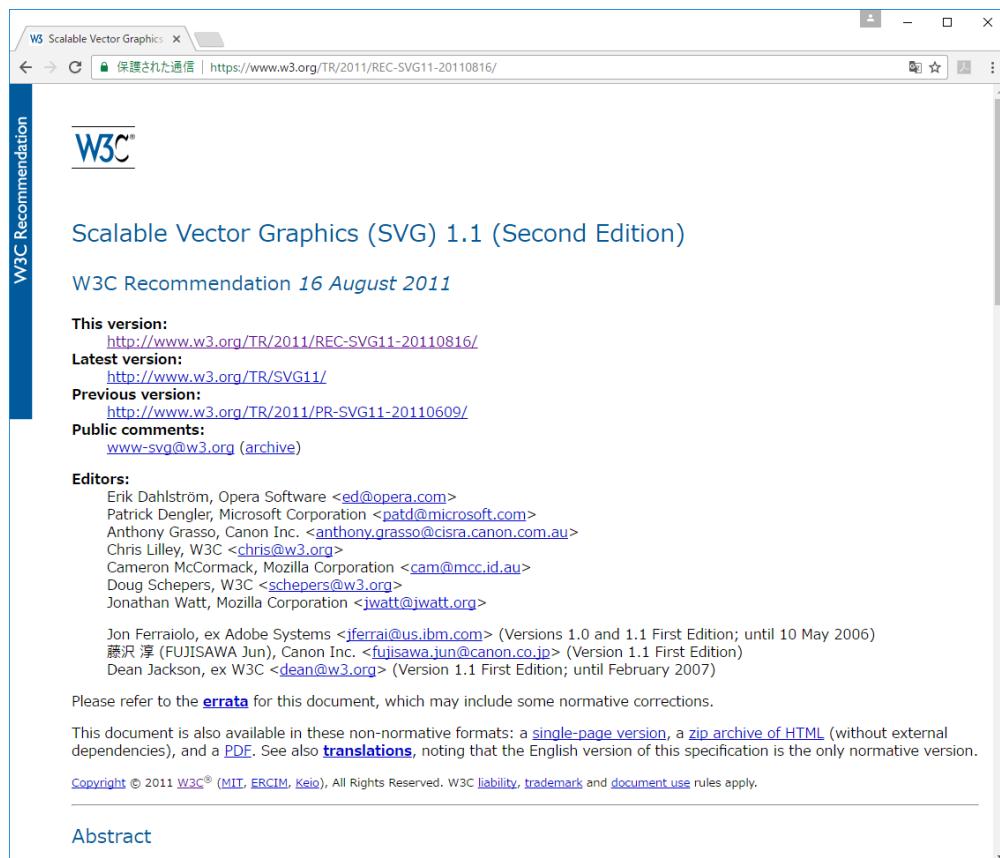


図 1.3: SVG の規格のページ (2017/2/27 参照)

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 [16] 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。

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[29] 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です。

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。

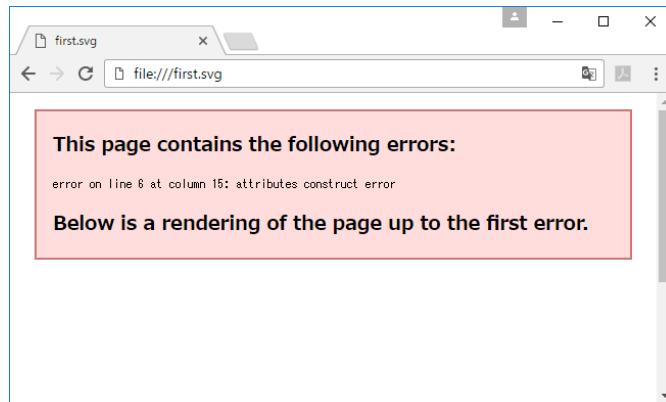


図2.1: Google Chromeにおけるエラーメッセージ

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）。

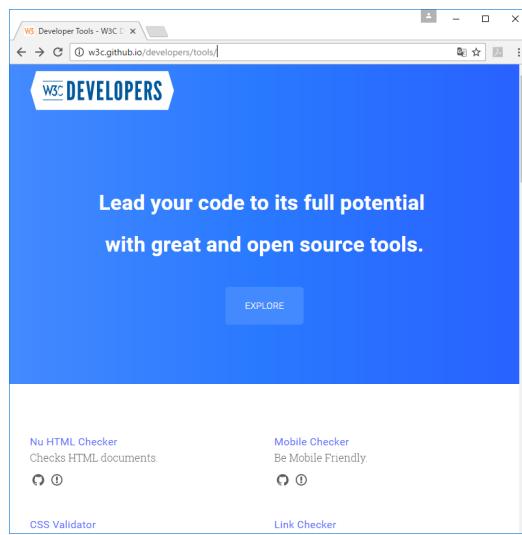


図 2.2: Validator のトップ画面

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。

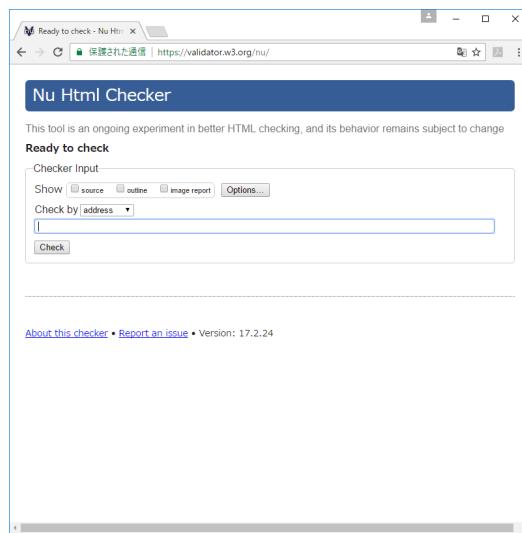


図 2.3: Nu Html Checker の画面

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。

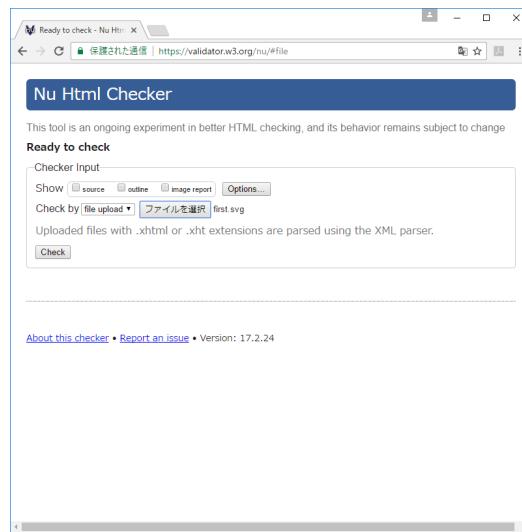


図 2.4: ファイルのアップロード

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)。

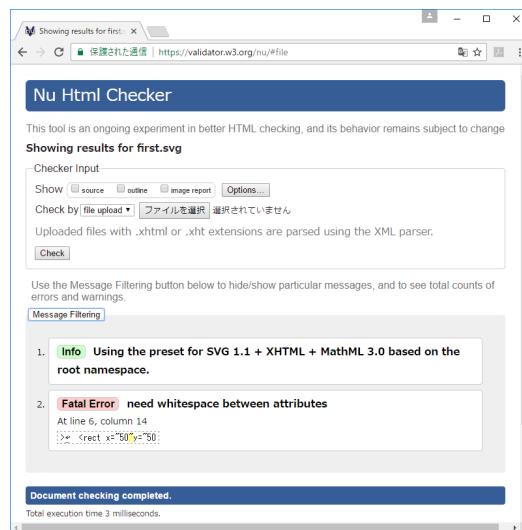


図 2.5: 検証結果の表示画面

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)。

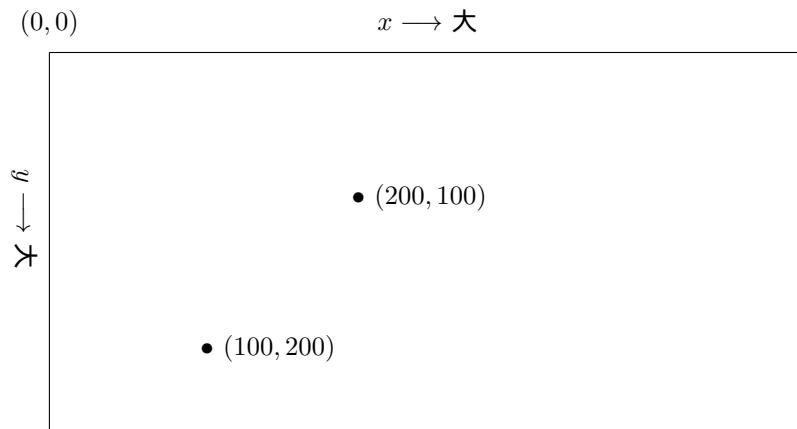


図 2.6: SVG の座標系

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。

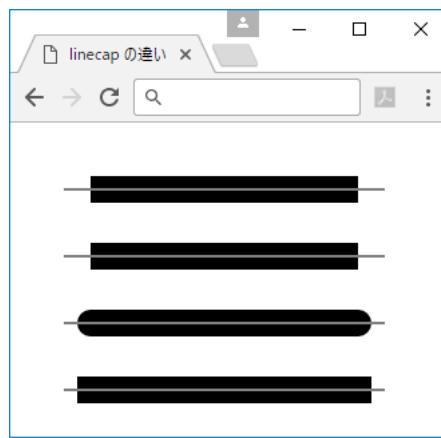


図 2.7: 直線と端の指定 stroke-linecap の違い

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によばれます。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。なくと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
<code>x1</code>	開始位置の <i>x</i> 座標	数値
<code>y1</code>	開始位置の <i>y</i> 座標	数値
<code>x2</code>	終了位置の <i>x</i> 座標	数値
<code>y2</code>	終了位置の <i>y</i> 座標	数値
<code>stroke</code>	直線を塗る色	色名または <code>rgb</code> 値で与える
<code>stroke-width</code>	直線の幅	数値
<code>stroke-linecap</code>	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 [33, 61 ページ] です (図2.8)。

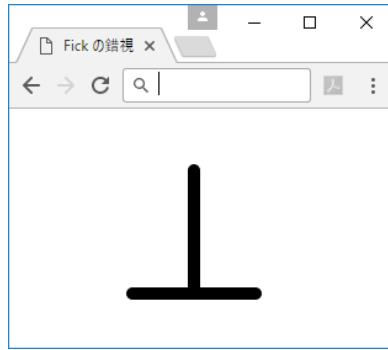


図 2.8: フィックの錯視

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 [33, 57 ページ] として知られる図形です。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。

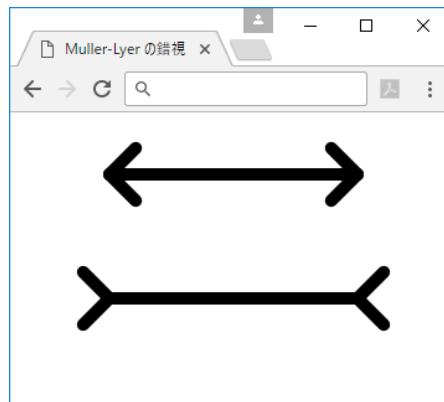


図 2.9: ミューラー・ライヤー錯視

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。

- 両端の矢印の長さを変える
- 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
- 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
- `<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。

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

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 [33, 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。

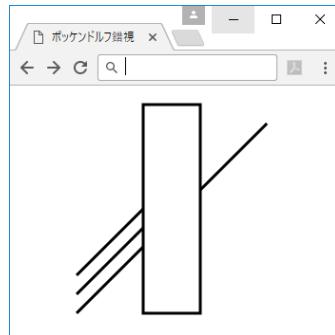


図 2.10: ポッケンドルフ錯視

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" />
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図形です。

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"
```

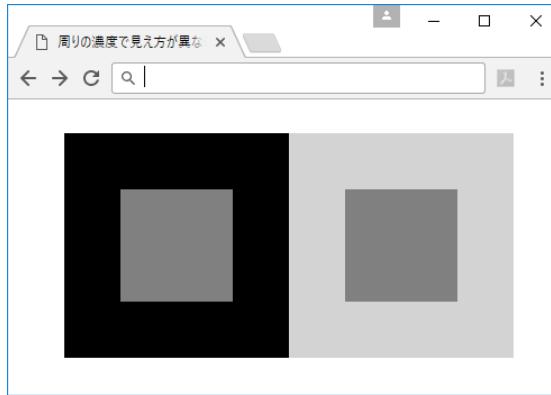


図 2.11: 周りの濃度で見え方が異なる

```

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 [38, 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。

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 [33, 180 ページ] と呼ばれるものです。

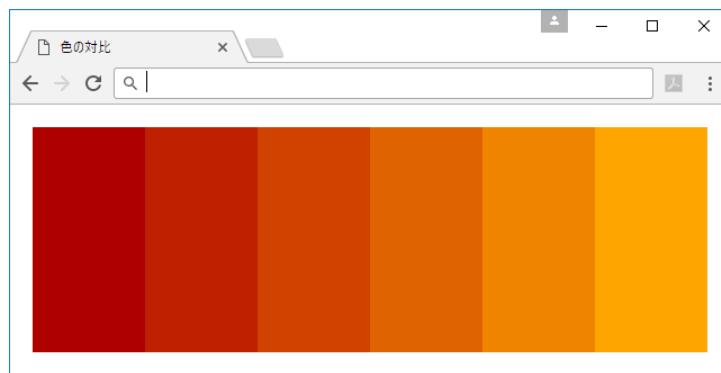


図 2.12: 色の対比

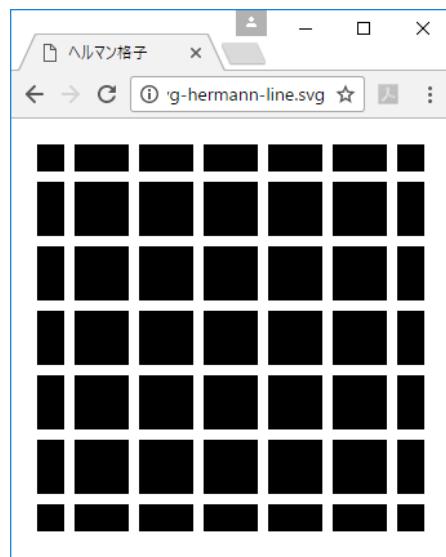


図 2.13: ヘルマン格子

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 [33, 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。

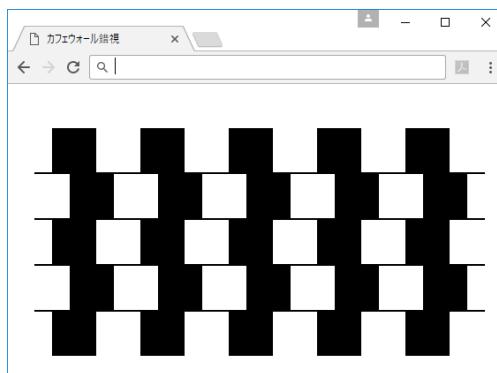


図 2.14: カフェウォール錯視

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"/>

```

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。

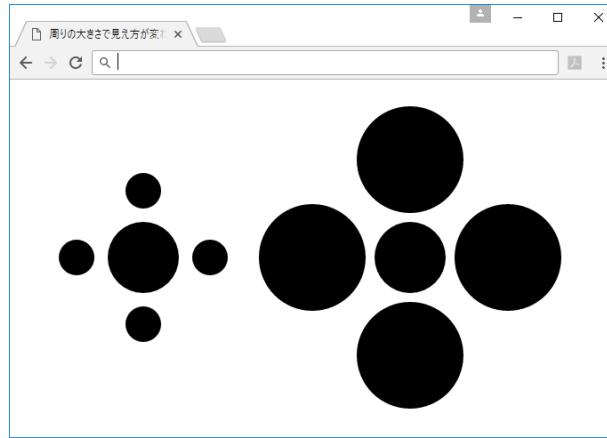


図 2.15: 周りの大きさで見え方が変わる

```
15    </g>
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 [33, 59 ページ] と呼ばれています。この図は [38, 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。

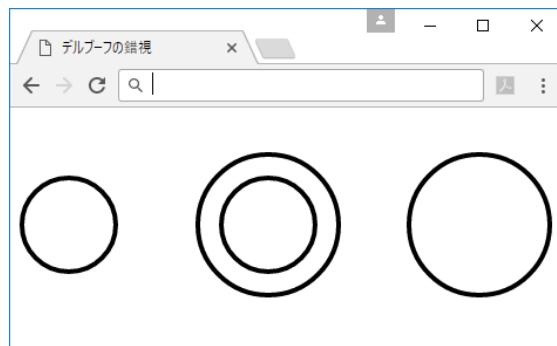


図 2.16: デルブーフの錯視

2.6 グラデーション

今まで長方形の内部を单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。

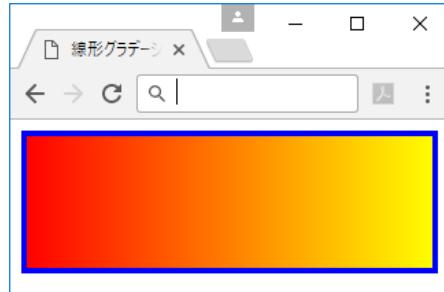


図 2.17: 線形グラデーション

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 [33, 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。

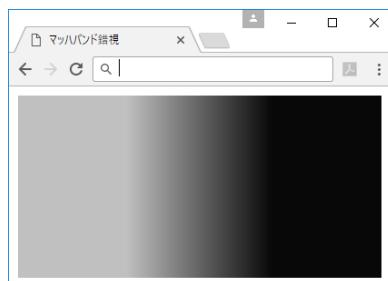


図 2.18: マッハバンド錯視

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"

```

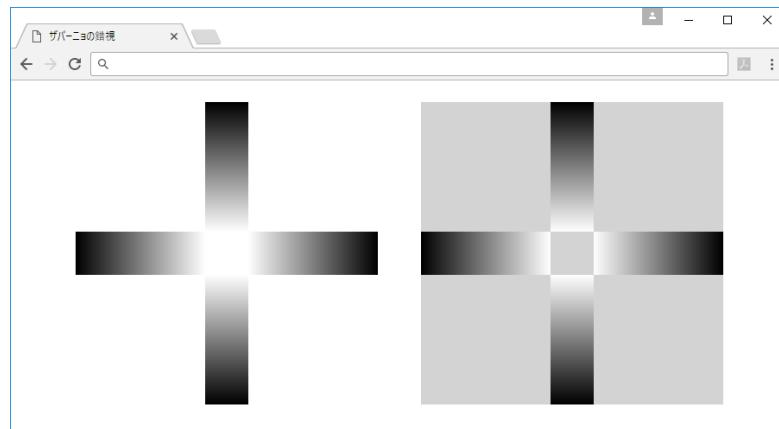


図 2.19: ザバーニョの錯視

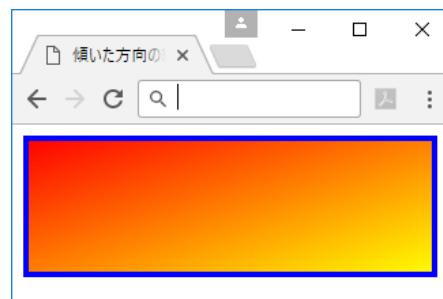


図 2.20: 傾いた方向の線形グラデーション

```

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 [33, 90 ページ] です。背景の明度の差がある部分を中心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。

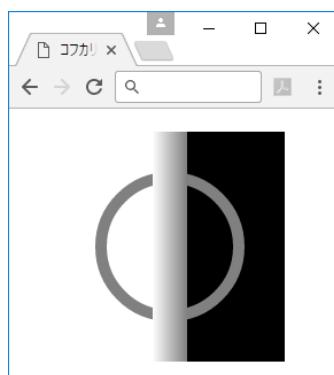


図 2.21: コフカリング

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 ([38, 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%" />
```

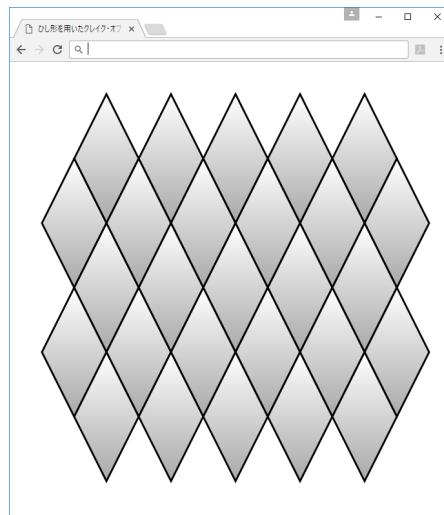


図 2.22: ひし形を用いたクレイク・オブライエン効果

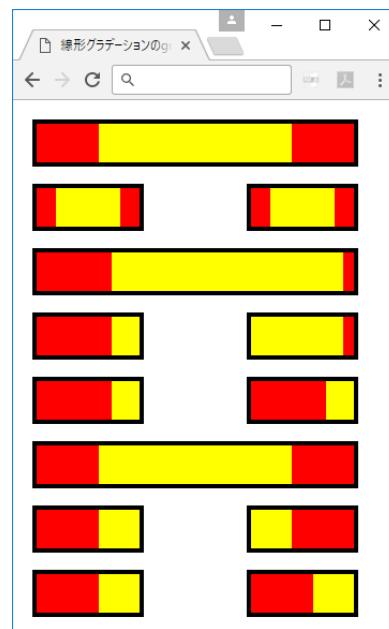


図 2.23: 線形グラデーションの gradientUnits の違い

```
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
<code>cx</code>	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
<code>cy</code>	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
<code>r</code>	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
<code>fx</code>	放射グラデーションの中心の x 座標	数値または割合
<codefy< code=""></codefy<>	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%">

```

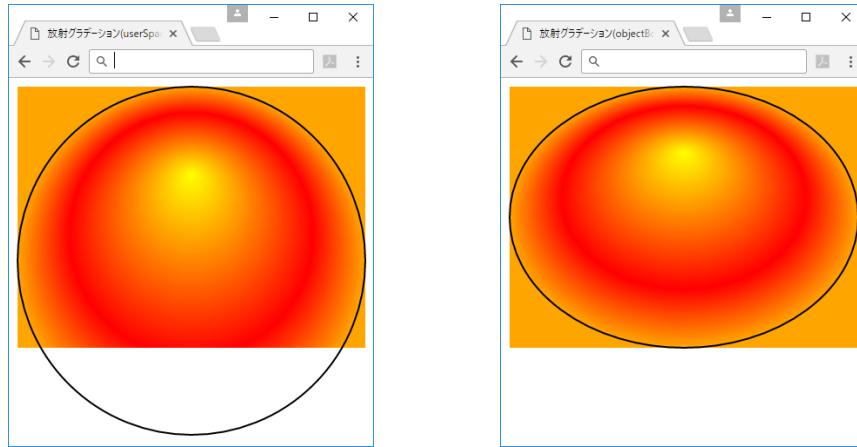


図 2.24: 放射グラデーション (userSpaceOnUse(左) と objectBoundingBox(右))

```

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。

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。

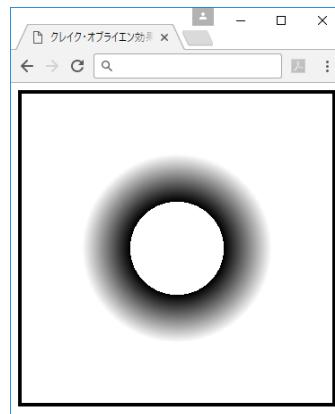


図 2.25: クレイク・オブライエン効果

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。

```

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>
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"

```

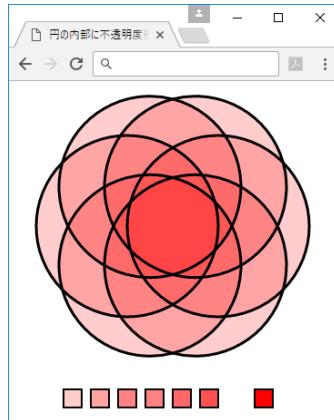


図 2.26: 円の内部に不透明度を設定した例

```

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なさい。

- `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。
- 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
- 円を放射グラデーションで塗ったらどのようになるか
- 円を放射グラデーションに `stop-opacity` を入れたらどのようになるか

問題 2.17 図 2.27 はピラミッドの稜線とよばれています ([38, 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。

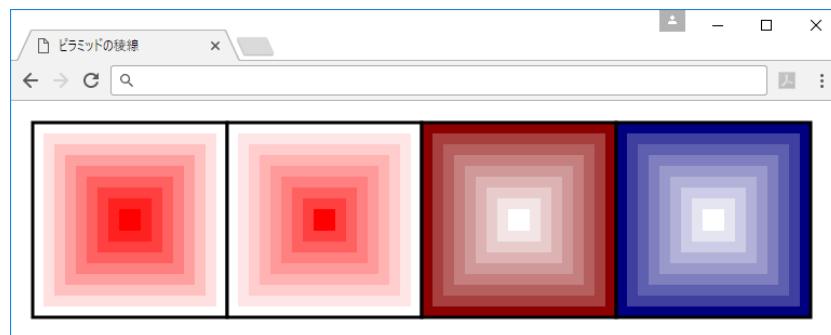


図 2.27: ピラミッドの稜線

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。

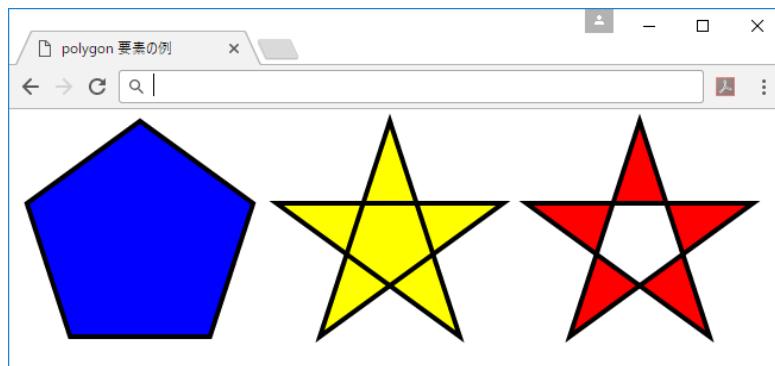


図 3.1: `<polygon>`要素の例

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。

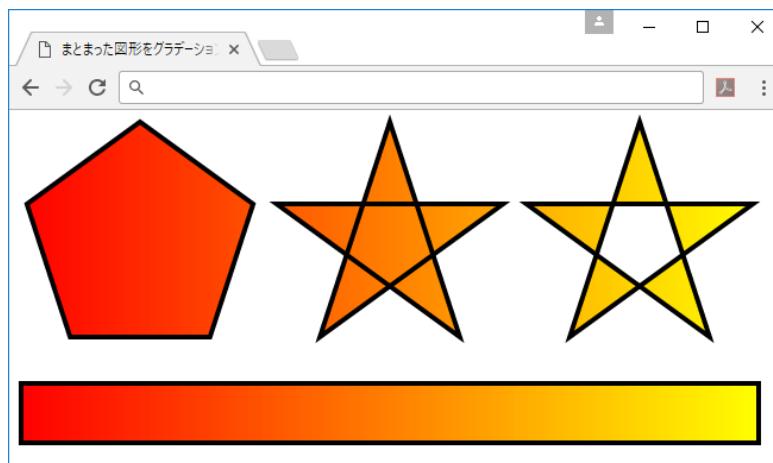


図 3.2: まとまった図形をグラデーションで図形を塗りつぶす

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。

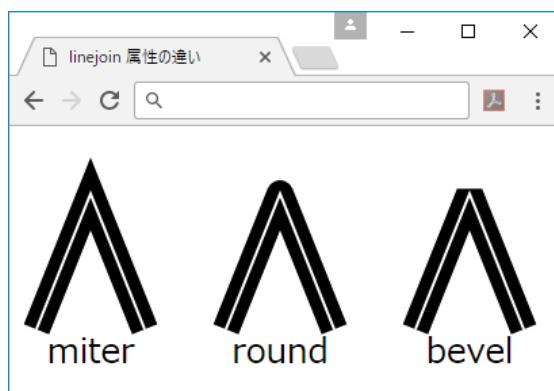


図 3.3: linejoin 属性の違い

- 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 [33, 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。

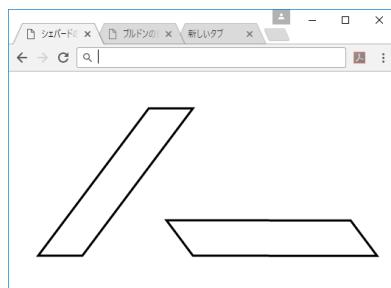


図 3.4: シェパーードの錯視

問題 3.4 図 3.5 はブルドンの錯視 [33, 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。

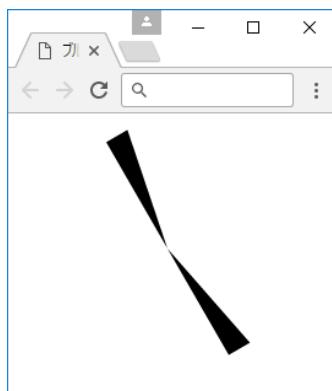


図 3.5: ブルドンの錯視

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	楕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。

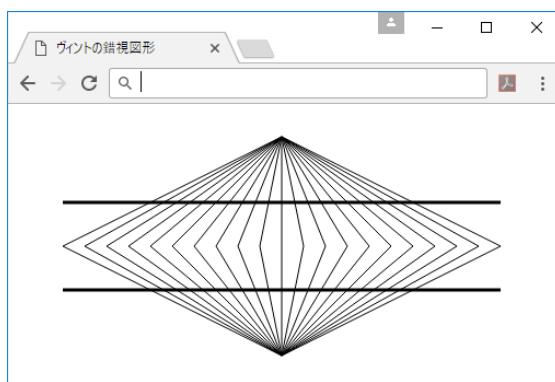


図 3.6: ヴィントの錯視図形

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` 座標の位置を少しづ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らなくなります。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 ([37, 85 ページ参照])。
3. ヴィントの錯視図形を片目で見ると見え方が変わるかどうか確かめなさい。

なお、このような図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図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"
```

と指定した場合に図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。

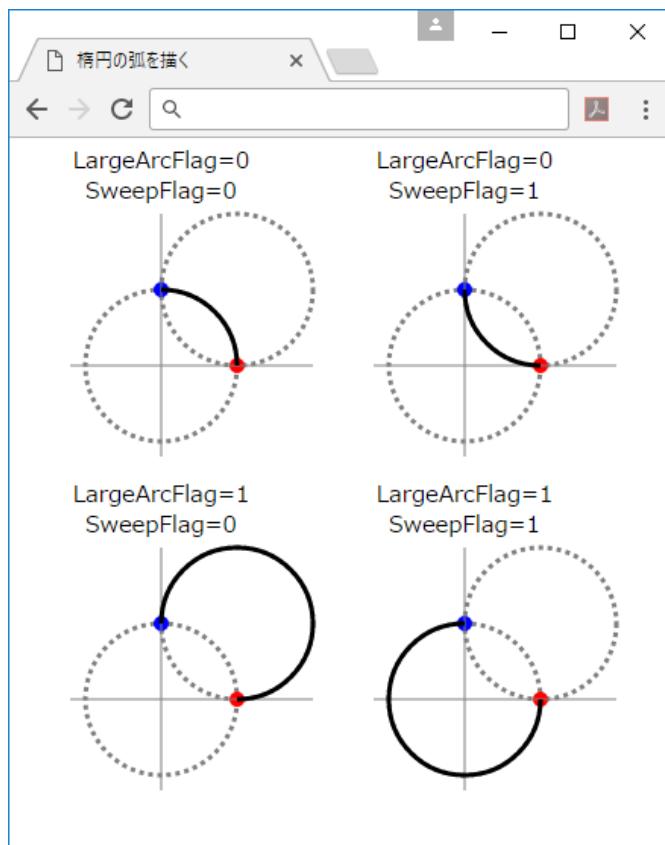


図 3.7: 橿円の弧を描く

SVG リスト 3.4: 橿円の弧を描く

```

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>橿円の弧を描く</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 [33, 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"/>
20       </g>

```

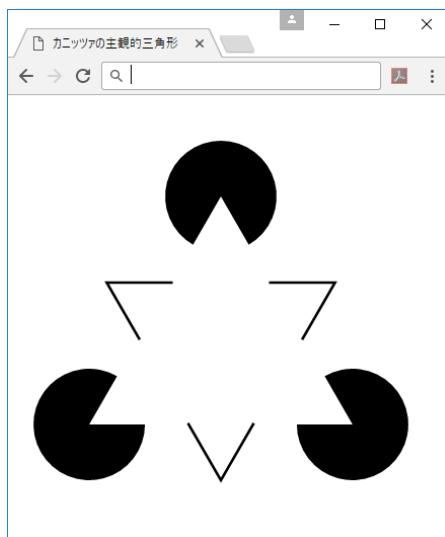


図 3.8: カニツツアの主観的三角形

```

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 のあと 1 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 [33, 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。

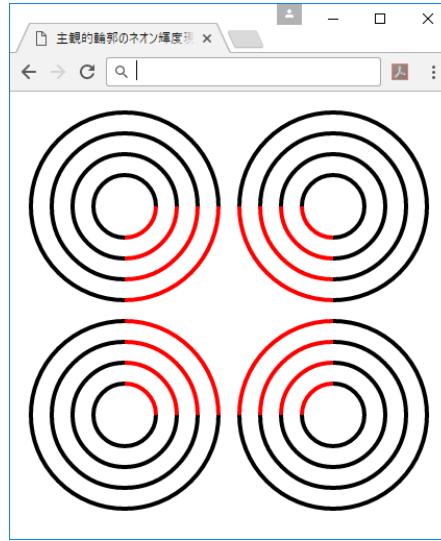


図 3.9: 主観的輪郭のネオン輝度現象

3.2.3 Bézier 曲線

Bézier 曲線の歴史については [17, p.13–14] に解説があります。なお、[4, 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。

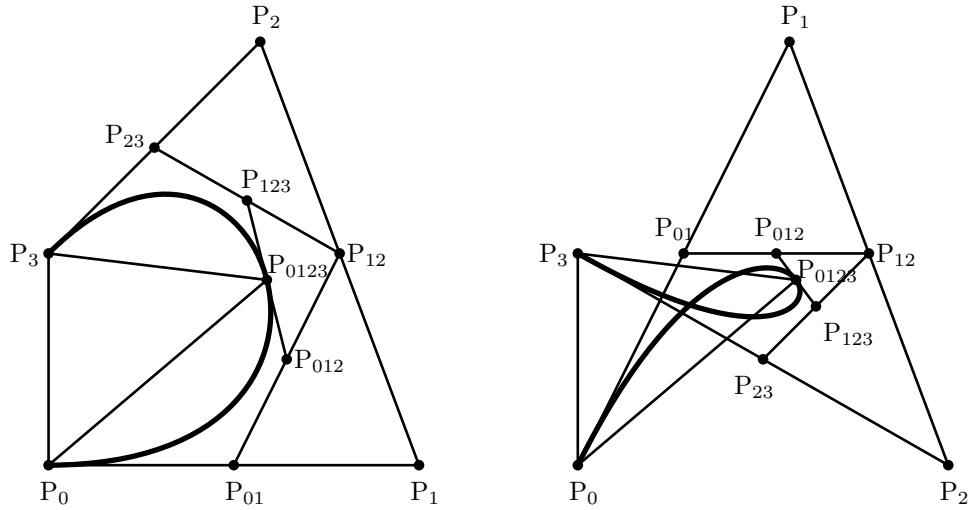


図 3.10: Bézier 曲線の解説

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* の中に *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 はカードのスケッチの絵です。この図を描きなさい。

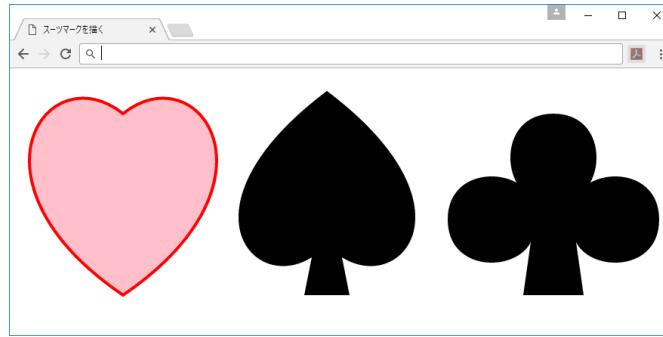


図 3.11: スーツマークを描く

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。

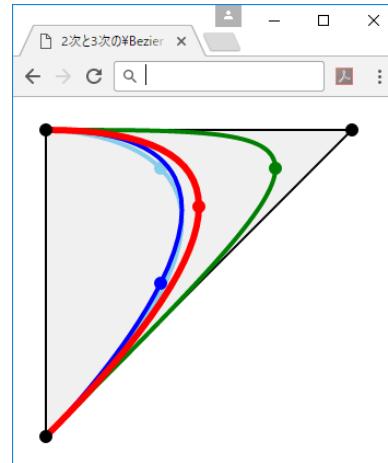


図 3.12: 2次と3次のBézier曲線の違い

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。

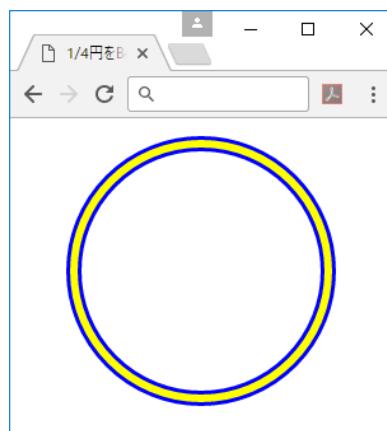


図 3.13: 1/4 円を Bézier 曲線で近似する

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。[17, 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。

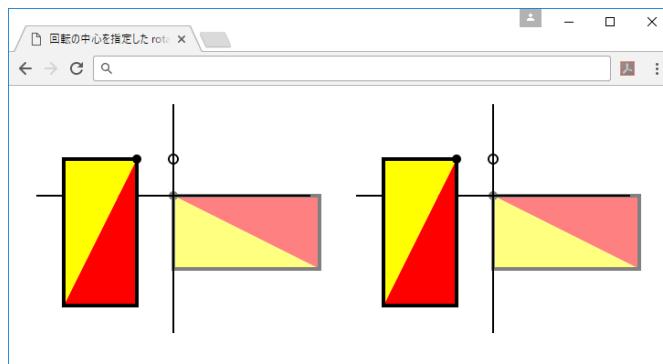


図 3.14: 回転の中心を指定した `rotate`

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 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"
```

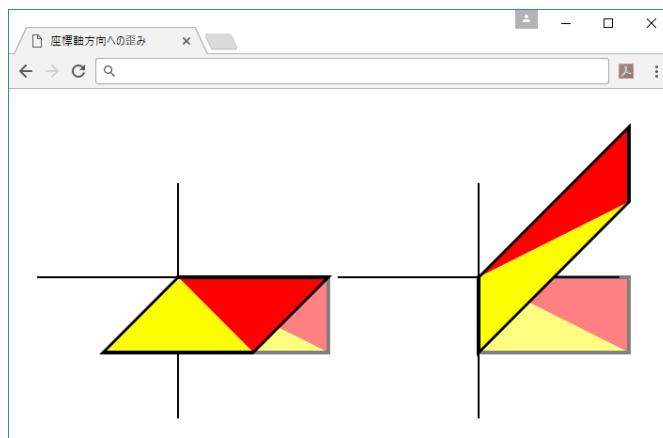


図 3.15: 座標軸方向への歪み

```

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 一般の線形変換

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。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 [38, 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。

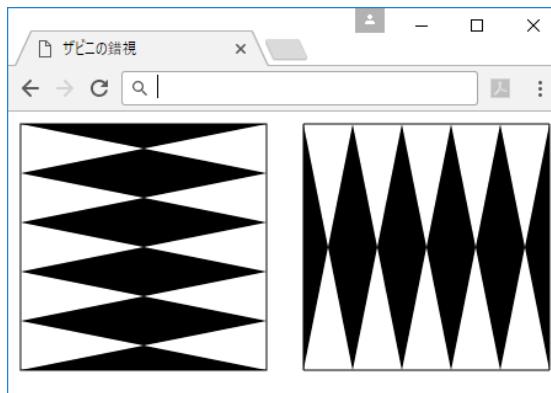


図 3.16: ザビニの錯視

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" >

```

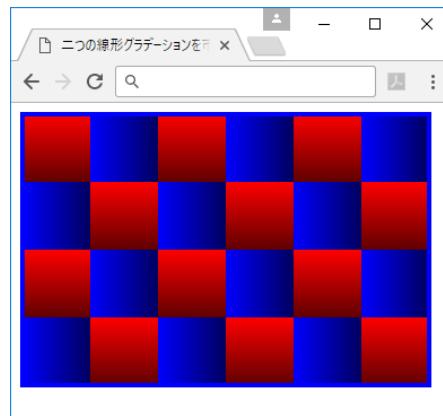


図 3.17: 二つの線形グラデーションを市松模様に並べた例

```

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 [33, 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。

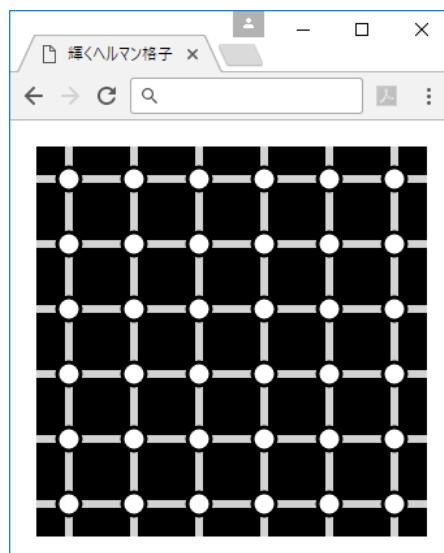


図 3.18: 輝くヘルマン格子

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 [33, 76 ページ] とよばれる錯視図形です。これを作成しなさい。

パターンを塗りつぶす図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図形を内部のパターンを簡略化した図形です ([38, 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[18, 75 ページ] も参照のこと。この本にはほかにも面白い錯視図形が載っています。

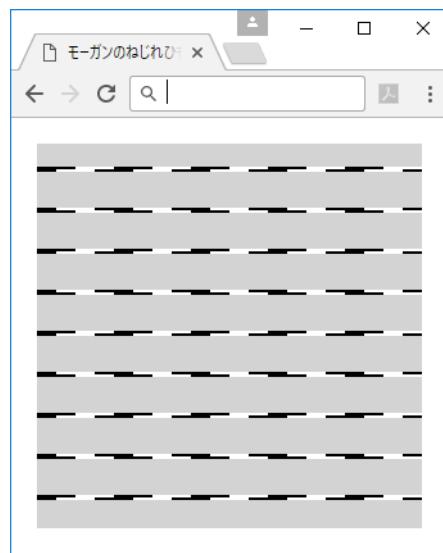


図 3.19: モーガンのねじれひも

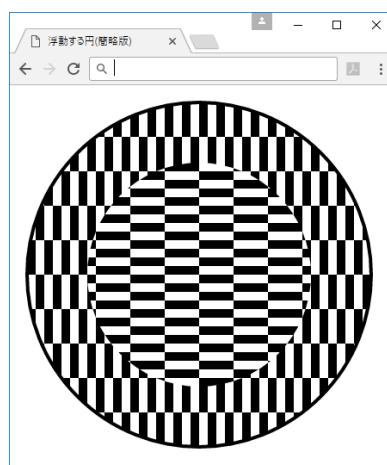


図 3.20: 浮動する円(簡略版)

```
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 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を調べること。

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。

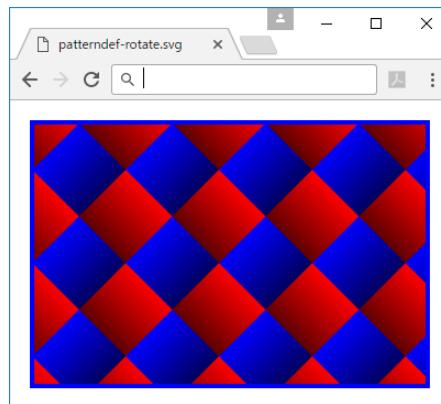


図 3.21: 図 3.17 のグラデーションのパターンを回転させる

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。

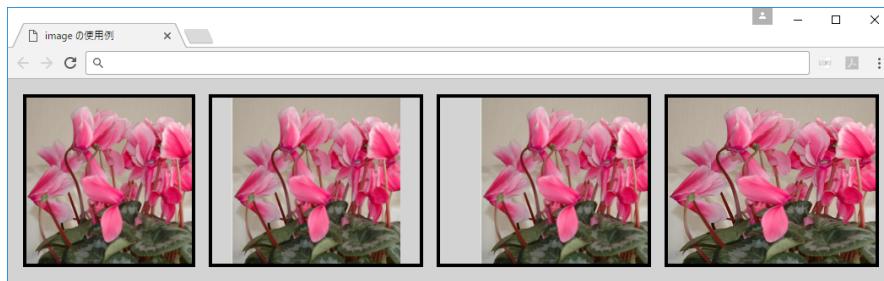


図 3.22: <image> 要素の使用例

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>

```

- 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。

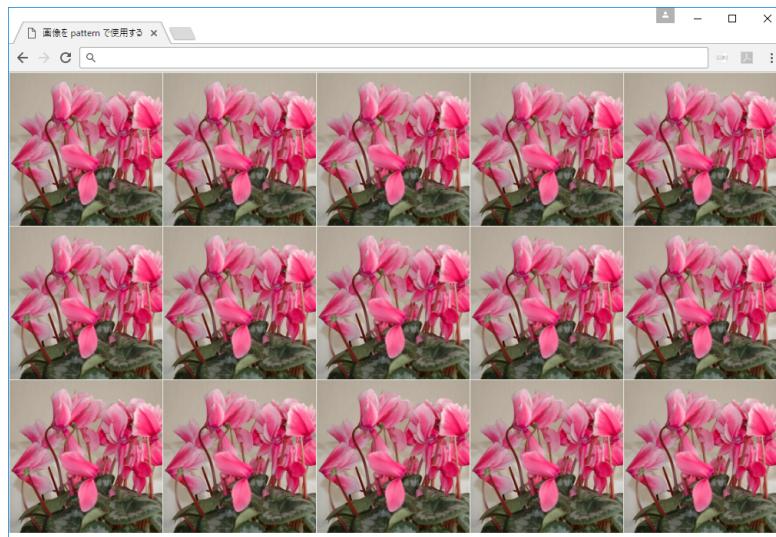


図 3.23: 画像を<pattern> 要素で使用する

- 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。

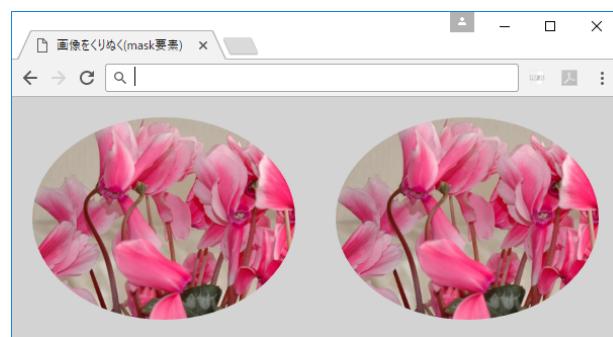


図 3.24: 画像をくりぬく (<mask> 要素)

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。
 - ここでは楕円の内部を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行目で定義された楕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>要素のなかにある楕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。

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。

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。

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。

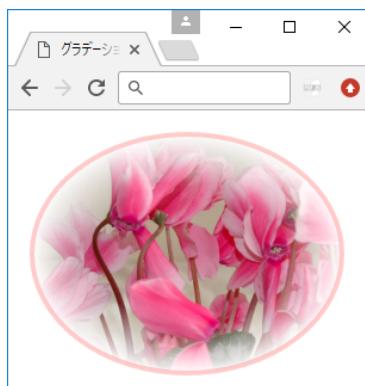


図 3.25: グラデーションを使用した<mask> 要素

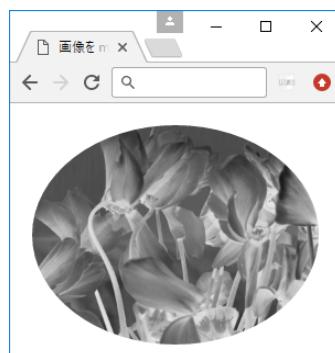


図 3.26: 画像を<mask> 要素につかう

```
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。

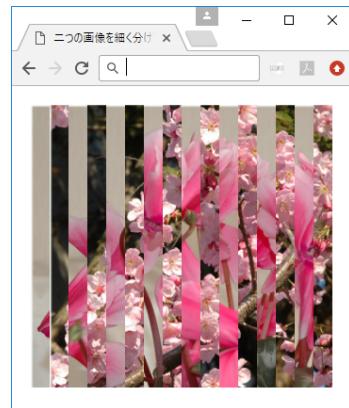


図 3.27: 二つの画像を細く分けて互い違いに並べて表示する

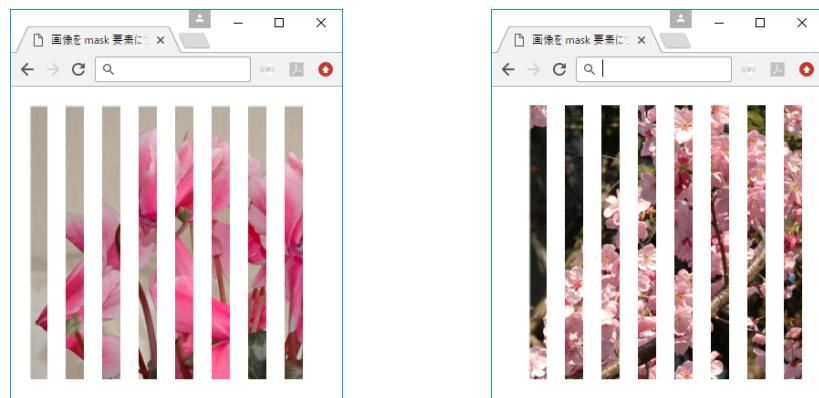


図 3.28: 画像を<mask>要素につかう (元画像)

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できます。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。

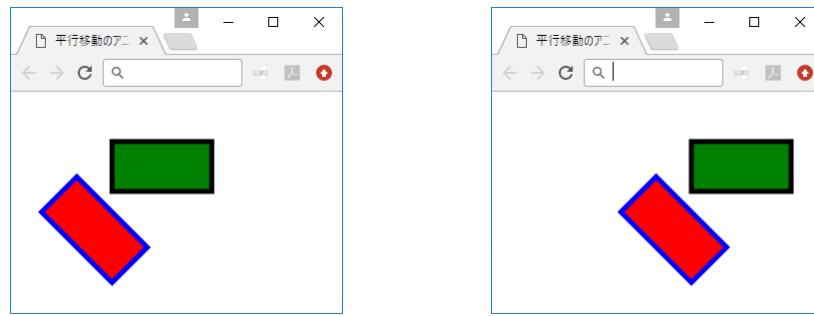


図 4.1: 平行移動のアニメーション(開始時 左と終了時 右)

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 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 フィックの錯視 (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。

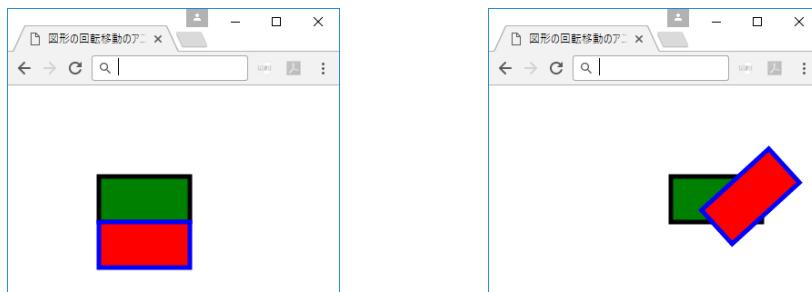


図 4.2: 図形の回転のアニメーション – 開始時 (左) と平行移動終了時 (右)

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 [33, 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。

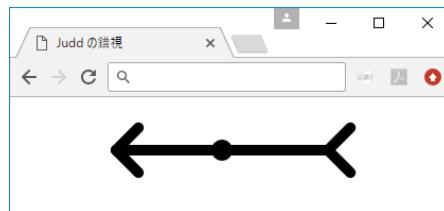


図 4.3: ジャッドの錯視

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%" >
5    <title>拡大縮小と移動のアニメーション</title>

```

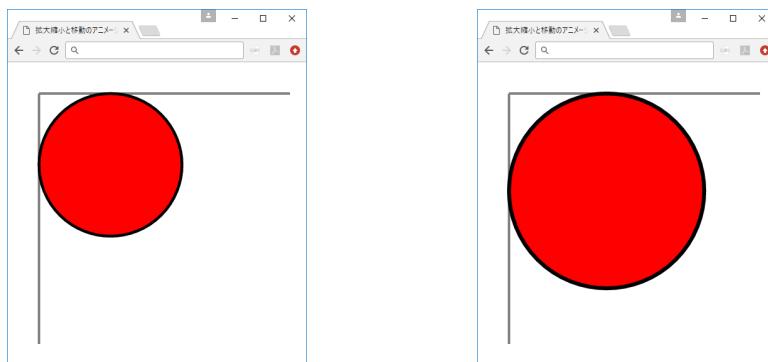


図 4.4: 拡大縮小と移動のアニメーション

```

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 (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。

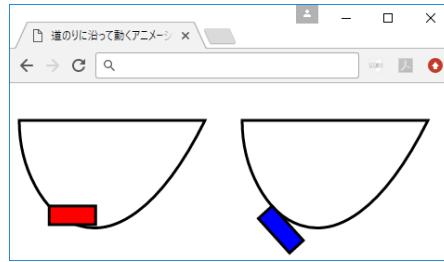


図 4.5: 道のりに沿って動くアニメーション

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<mpath> 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。

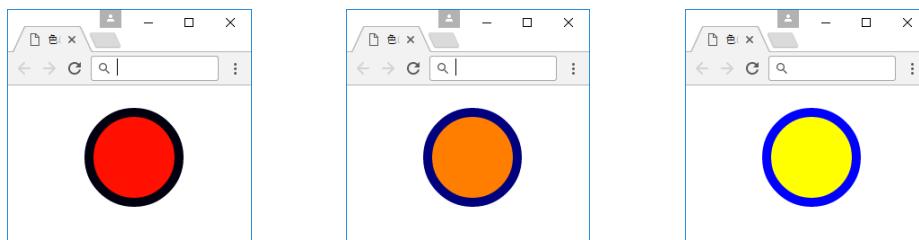


図 4.6: 色のアニメーション(開始時左、途中(中央)、終了時右)

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"
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>

```

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の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)。

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 の属性値の構造を変えてはいけません。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初めに与えた点が 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。

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
```

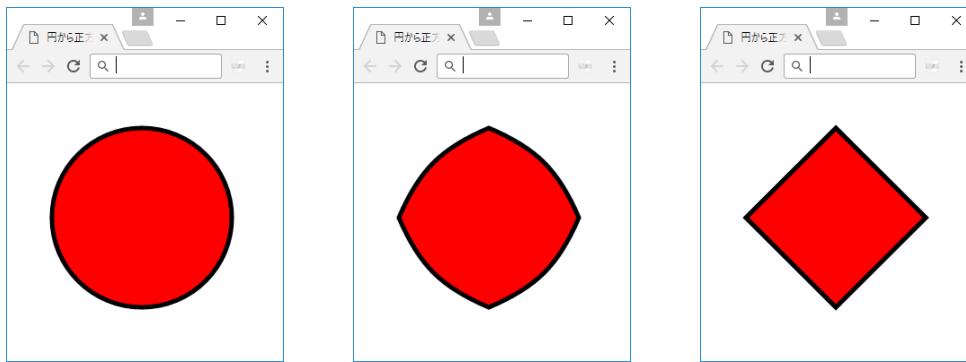


図 4.7: 円から正方形へ (開始時 左、途中、終了時 右)

```

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,-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と、`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)。

SVG リスト 4.8: グラデーション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"
```

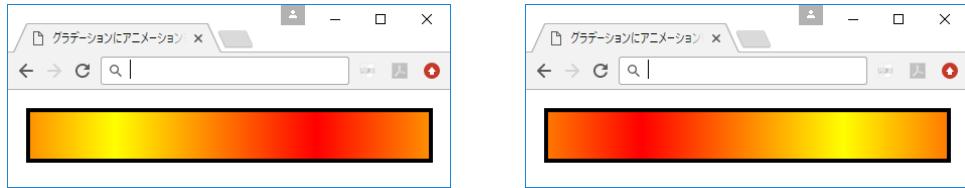


図 4.8: グラデーションにアニメーションを付ける

```

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  <linearParams 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 </linearParams>
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用意しているからです。
 - 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、さらに時間が経つ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。

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>ストップウォッチ？</title>
6  <defs>

```

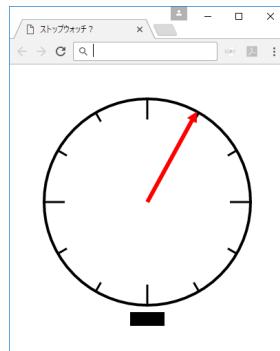


図 4.9: ストップウォッチ?

```

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トを定義しています。
 - 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° 回転したものとひとつのものとしてまとめて定義します (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Facemouseout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き、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。

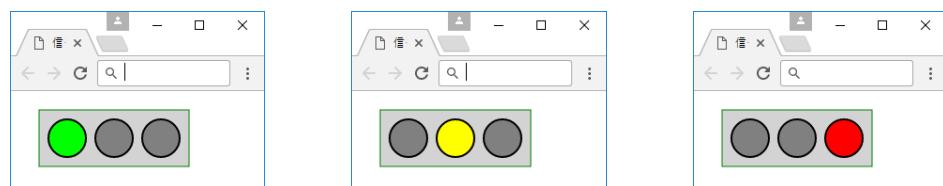


図 4.10: 信号機

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。

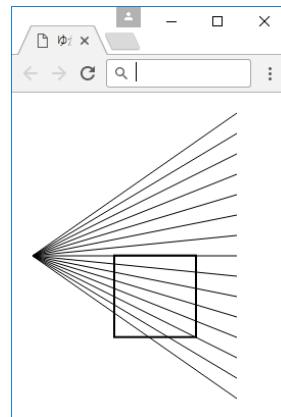


図 4.11: ゆがんだ正方形

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。

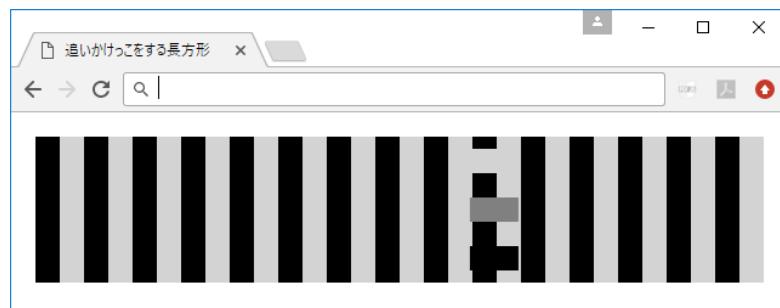


図 4.12: 追いかけっこをする長方形 (その 1)

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。

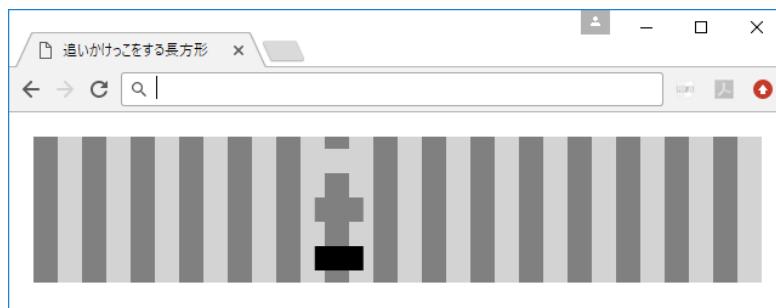


図 4.13: 追いかけっこをする長方形 (その 2)

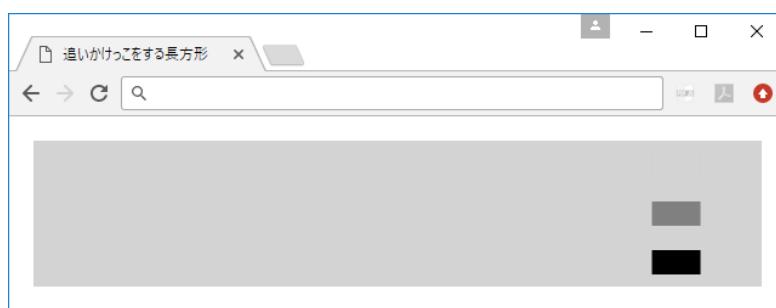


図 4.14: 追いかけっこをする長方形 (その 3)

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。

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)" />
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雛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	文字列の開始位置
	middle	文字列の中央
	end	文字列の終了位置
dominant-baseline	垂直方向の位置	
	ideographic	文字列の一番下
	alphabetic	文字 x の下
	hanging	文字列の一番上
	mathematical	文字列の中央
	auto	自動
baseline-shift	文字列の上下の移動	
	baseline	添え字の位置
	sub	上付きの位置
	super	
	割合%	
	長さ	
font-family	フォントの種類の指定	
	font-family	フォント名を指定 (漢字は含めない)
	serif	
	sans-serif	
	cursive	
	fantasy	
	monospace	文字幅等間隔
font-style	フォントの形状	
	normal	標準
	italic	イタリック
	oblique	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	なし
	underline	下線
	overline	上線
	line-through	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>を表示するためにここでは< 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。

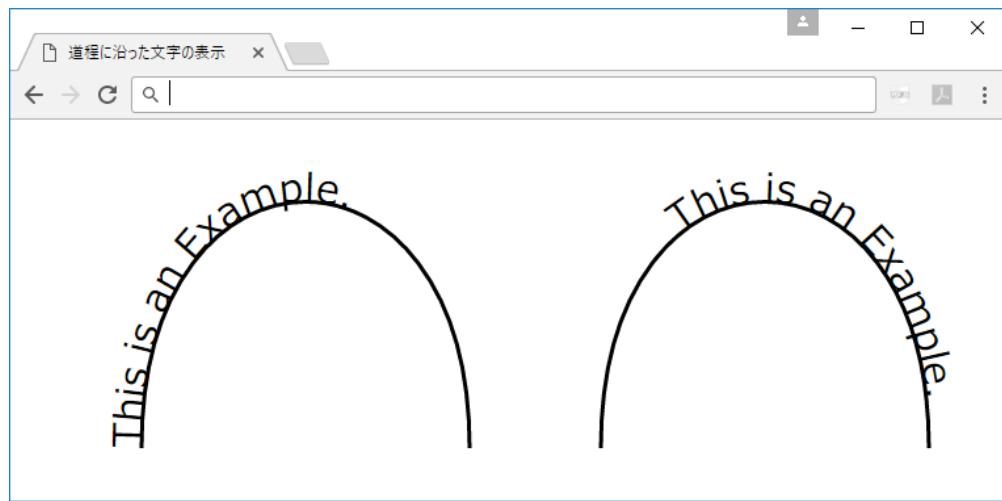


図 5.4: 道程に沿った文字の表示

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。
- 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い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单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。

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

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 ([38, 182 ページ図 17.1])。ヘルマン格子の変形です。

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">
```

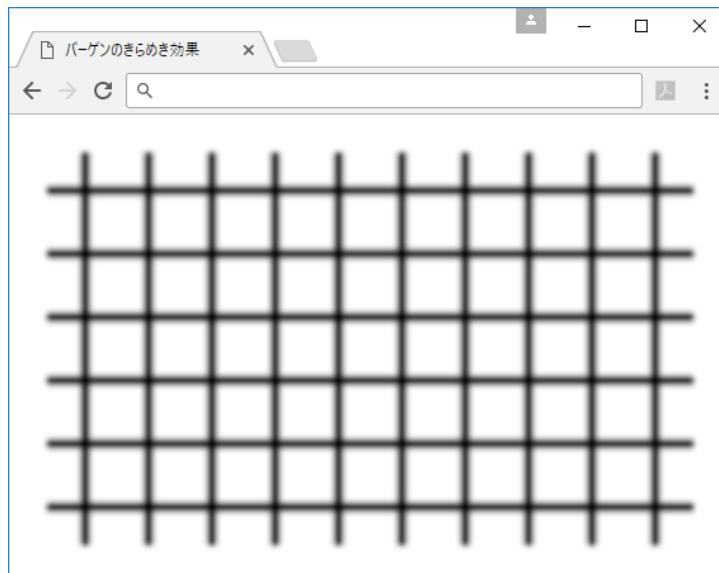


図 6.2: バーゲンのきらめき効果

```

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。

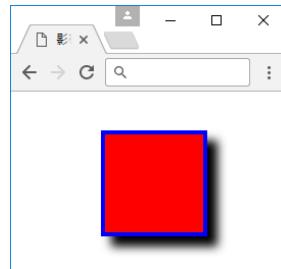


図 6.3: 影をつける

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 フィルタをが宣言されています。
- 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。

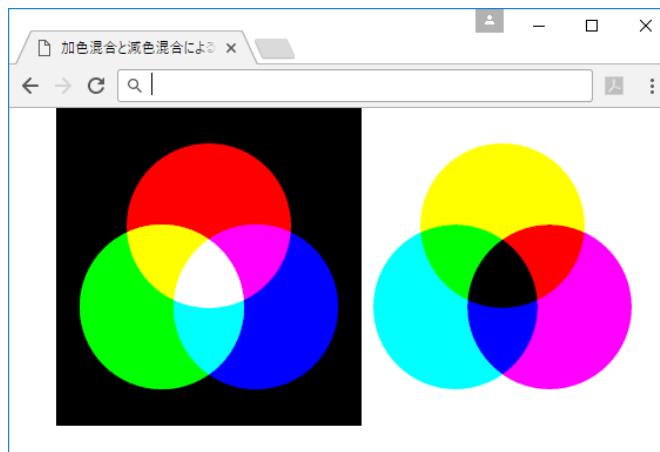


図 6.4: 加色混合と減色混合によるカラーチャート図

```

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。

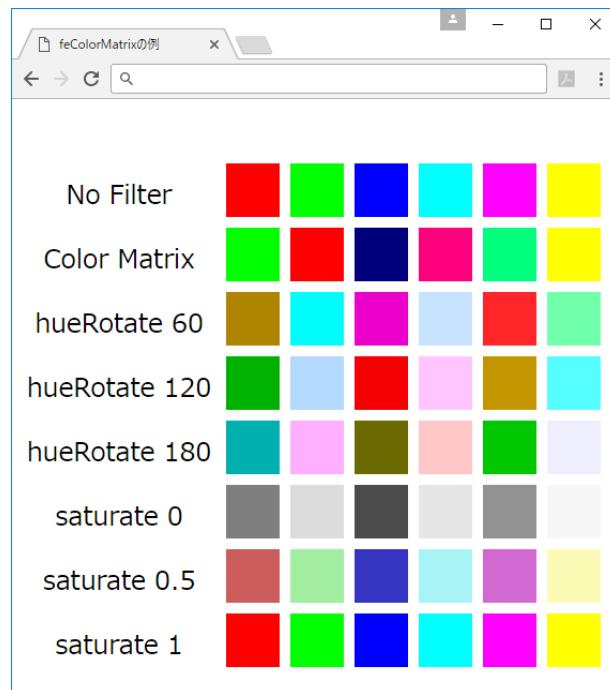


図 6.5: `feColorMatrix` フィルタの例

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雛形を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([28]) 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になっていることがわかります。

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

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。
- 52 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 32 行目から 35 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 40 行目から 43 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	C
<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 [28] 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>

```

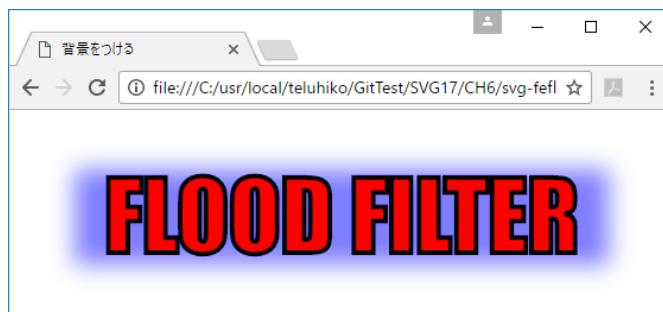


図 6.7: 背景をつける

```

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   <feMerge>
15     <feMergeNode in="floodblur"/>
16     <feMergeNode in="SourceGraphic"/>
17   </feMerge>
18 </filter>
19 </defs>
20 <style type="text/css">
21   .textStyle {font-size:80px; text-anchor:middle;
22     font-family:Impact;stroke:black; stroke-width:4; fill:red;}
23 </style>
24 <g transform="translate(30,20)" >
25   <text x="250" y="100"
26     class="textStyle" filter="url(#Flood)">FLOOD FILTER</text>
27 </g>
28 </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。

図 6.8 はこれらのフィルタの具体的な処理を示したものです。ここでは赤と青の円がそれぞれ in と in2 になっています。

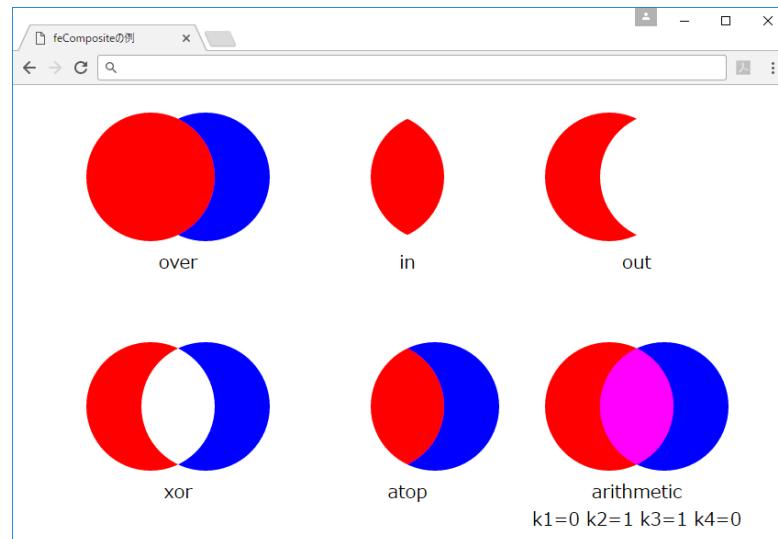


図 6.8: feComposite 要素の例

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

```

```

57    </g>
58    <g transform="translate(500, 250)">
59      <use xlink:href="#IN" filter="url(#OpArith)" />
60      <text class="Text" x="180" y="200">arithmetic</text>
61      <text class="Text" x="180" y="230">k1=0 k2=1 k3=1 k4=0</text>
62    </g>
63  </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13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(14行目から18行目)
 - operatorがinのフィルタ(19行目から22行目)
 - operatorがoutのフィルタ(23行目から26行目)
 - operatorがatopのフィルタ(27行目から30行目)
 - operatorがarithmeticのフィルタ(31行目から35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、この図もそうですがfeSpecularLighting要素は単独で使われなくてbump mapとして他の画像と合成に使用します。

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。

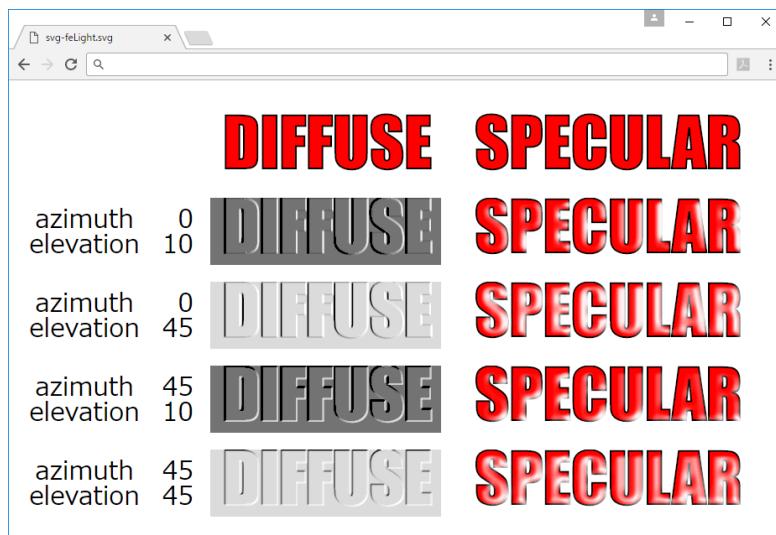


図 6.9: 光を当てる

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`, `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 [28] 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ぬ。
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

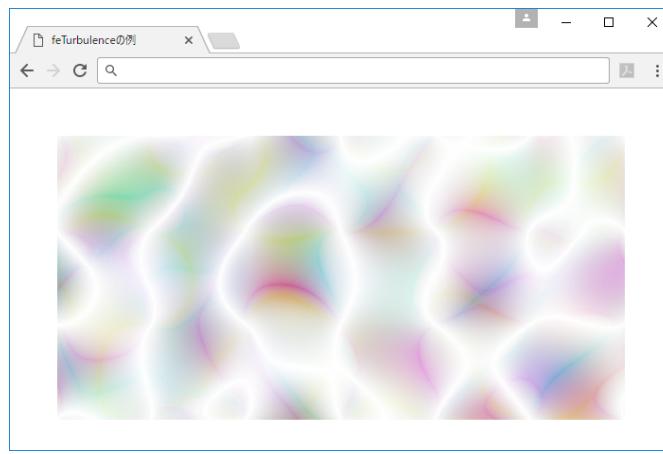


図 6.10: feTurbulence フィルタの例

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、この章で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 [9] で計算機言語としてもう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うことにします。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。

¹ JavaScript 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。

JavaScriptはインターペリター型の言語です。言語を解釈しながら実行していくの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、そのあとエラーが出て停止することもあります。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ものが一般的です。

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[23]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 7.1 は 41 ページにある図 3.3 の SVG 文書を Chrome で「デベロッパーツール」を開いたところです。

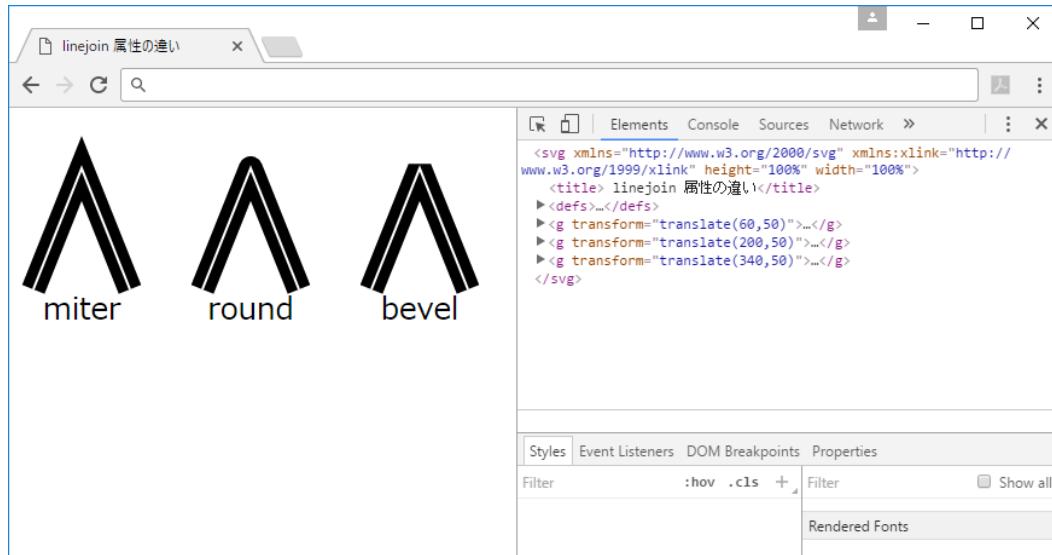


図 7.1: Chrome のデベロッパーツールで DOM ツリーを表示する (1)

- 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「Elements」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7.2は図7.1の「Elements」の中で2番目のg要素上にカーソルを置いた状態です。



図7.2: ChromeのデベロッパーツールでDOMツリーを表示する(2)

カーソル上にある要素に対応した要素の部分の背景が水色になっています。

この要素の前にある「▶」をクリックするとその要素の子要素までDOMツリーが展開されます。

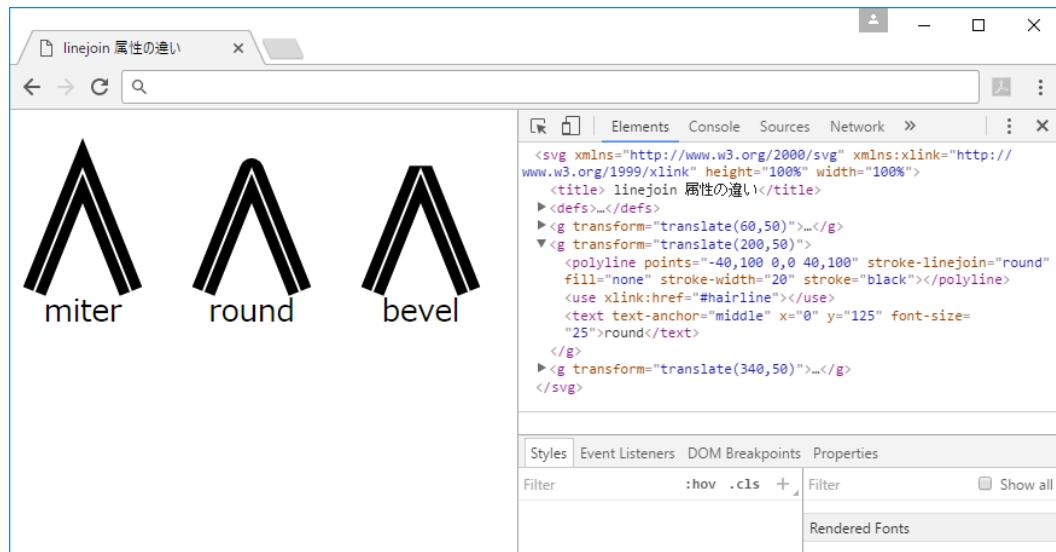


図7.3: Chromeの開発ツールでDOMツリーを表示する(3)

この画面で`<polyline>`要素の属性`width`の属性値を示すところをクリックして、属性値を変更しようとしている画面が図 7.4 です。

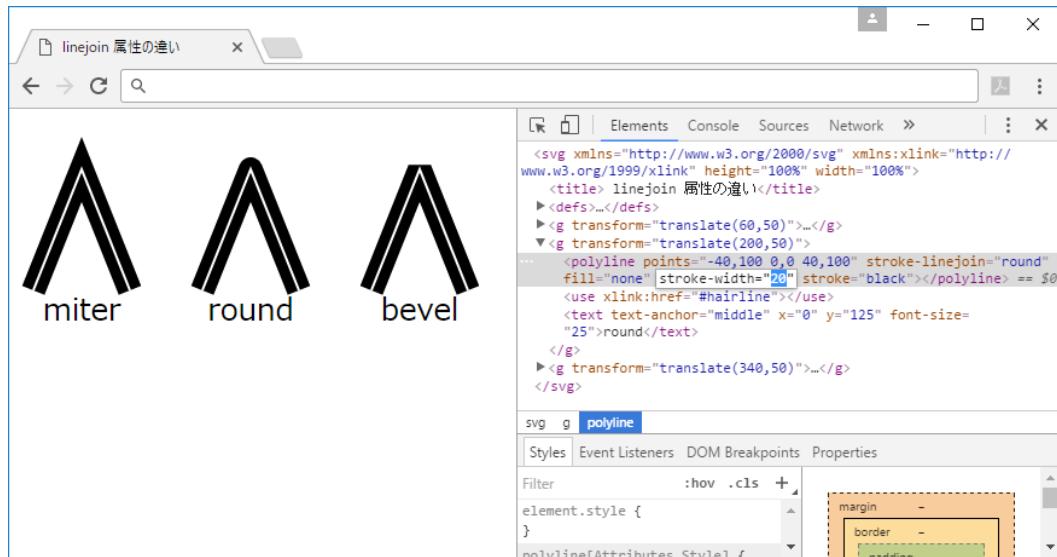


図 7.4: Chrome の開発ツールで DOM ツリーを表示する (4)

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 7.5 です。

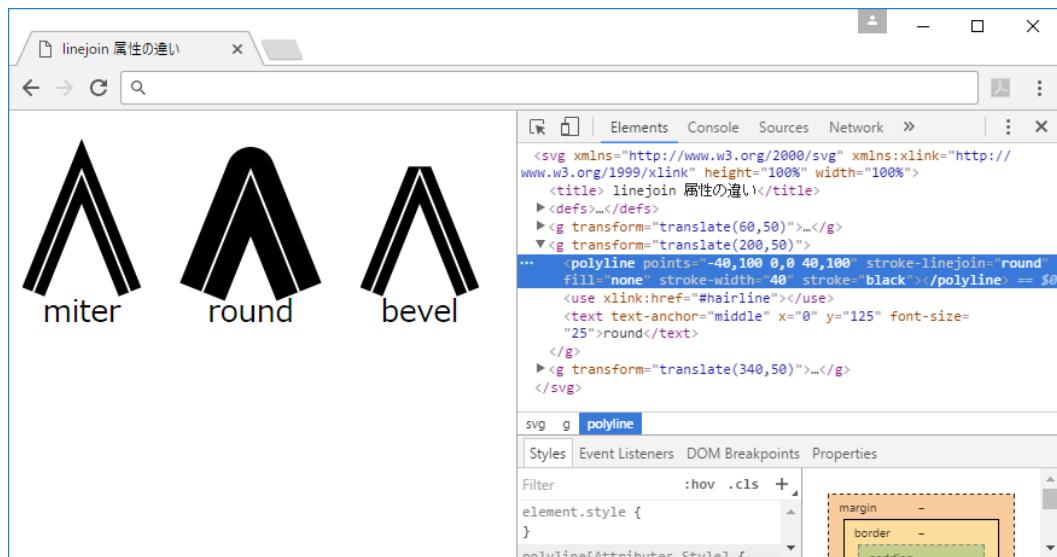


図 7.5: Chrome の開発ツールで DOM ツリーを表示する (5)

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に立ちます。

- **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 Chrome 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。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。
getElementsByName(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。
getElementsByTagName(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。

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
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 [] で参照できます。

表 7.2 は DOM の要素に対するプロパティです。

なお、DOM4[24] 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。

7.2 イベント

7.2.1 イベント概説

イベントとはプログラムに対して働きかける動作を意味します。Windows で動くプログラムにはマウスボタンが押された（クリック）などのユーザからの要求に対し反応する必要があります。このような要求を一般にイベントと呼びます。プログラムに対する要求はすべてイベントです。一定時間たったことをシステムから教えてもらうタイマーイベント、プログラムを中断することを知

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)

らせるものなどありとあらゆる行為がイベントという概念で処理されます。プログラムが開始されたということ自体イベントです。このイベントは初期化をするために利用されます。

イベントの発生する順序はあらかじめ決まっていないのでそれぞれのイベントを処理するプログラムは独立している必要があります。このようにイベントの発生を順次処理していくプログラムのモデルをイベントドリブンなプログラムといいます。

7.2.2 SVG 文書や HTML における代表的なイベント

SVG 文書や HTML 文書で発生する代表的なイベントを表 7.3 に掲げました。これらのイベントは各要素内の属性として現れます。属性に対して関数を属性値にすることでイベントの処理が行われます。

表 7.3 は代表的なイベントの例です。いくつかは SVG に固有のものもありますが、HTML 文書でも共通に使えるものもあります。

表 7.3: イベントの例

イベントの発生条件	イベントの属性名
ファイルのロード終了時	onload
ボタンがクリックされた	onclick
ボタンが押された	onmousedown
マウスカーソルが移動した	onmousemove
マウスボタンが離された	onmouseup
マウスカーソルが範囲に入った	onmouseover
マウスカーソルが範囲から出た	onmouseout
値が変化した	onchange
SVGのアニメーションが開始された	onbegin
SVGのアニメーションが終了した	onend

7.3 JavaScriptによるSVG文書のマウスイベントの処理

ユーザー側からSVG文書の图形にアクセスするきっかけとしては图形上でマウスのクリックやドラッグ、あるいはキーボードのあるキーが押された場合が主なものとして考えられます。SVGファイルの中にJavaScriptを記述するためには<script>要素内に書きます。具体的な記述方法はこの後を参考にしてください。

7.3.1 マウスに関するイベント処理

マウスイベントオブジェクトについて

マウスに関するイベントが発生すると、システムはマウスイベントオブジェクトを作成し、それをイベント処理関数の引数として渡します。表7.4はマウスイベントオブジェクトのメソッドとプロパティをまとめたものです。

クリックされたオブジェクトの属性値を表示する

与えられたオブジェクトの上でマウスボタンが押されたことを検知するSVG文書を書いてみましょう。ここでは3つの色の違う円があり、クリックされた円のfillの値をメッセージボックスで表示します。

図7.6は3つの円のうち、一番左の円をクリックした後のときのものです。

²通常は右ボタンが押されるとコンテキストメニューが表示されます。このイベントを利用するためにはJavaScriptのプログラムで制御する必要があります。

表 7.4: マウスイベントのプロパティとメソッド

プロパティ	メソッド	型	意味
target	getTarget()	EventTarget	イベントが発生したオブジェクト
currentTarget	getCurrentTarget()	EventTarget	イベントを処理しているオブジェクト
screenX	getScreenX()	long	マウスポインタの画面における <i>x</i> 座標
screenY	getScreenY()	long	マウスポインタの画面における <i>y</i> 座標
clientX	getClientX()	long	マウスポインタのクライアント領域における相対的な <i>x</i> 座標 (スクロールしている場合には pageXOffset を加える必要があります)
clientY	getClientY()	long	マウスポインタのクライアント領域における相対的な <i>y</i> 座標 (スクロールしている場合には pageYOffset を加える必要があります)
ctrlKey	getCtrlKey()	boolean	ctrl キーが押されているか
shiftKey	getShiftKey()	boolean	shift キーが押されているか
altKey	getAltKey()	boolean	alt キーが押されているか
metaKey	getMetaKey()	boolean	meta キーが押されているか
button	getButton()	unsigned short	マウスボタンの種類、0 は左ボタン、1 は中ボタン、2 は右ボタンを表します。 ²
eventPhase		unsigned short	イベント伝播の現在の段階を表す。 Event.CAPUTURING_PHASE(1)、 Event.AT_TARGET(2)、 Event.BUBBLING_PHASE(3) の値をとる
	preventDefault()		デフォルトの動作を実行しないようにする
	stopPropagation()		イベントの伝播を中止する

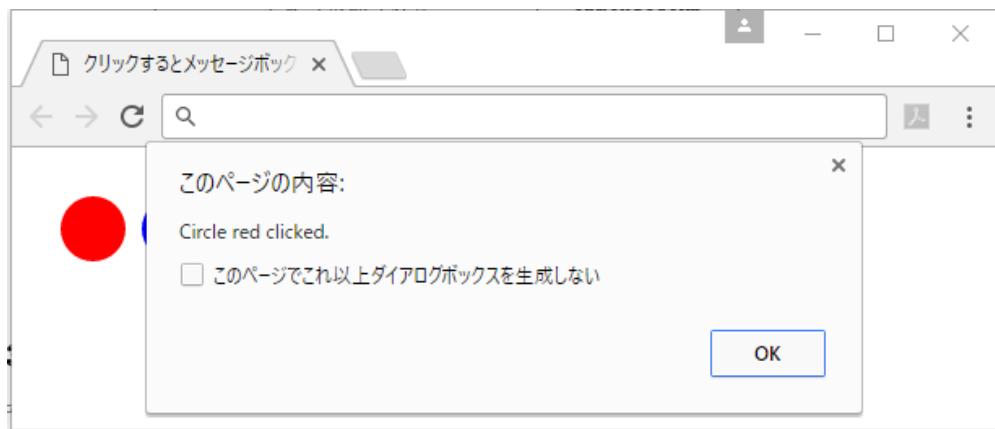


図 7.6: クリックするとメッセージボックスが表示されます

SVG リスト 7.1: マウスのクリックを検出する SVG(その 1)

```

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>クリックするとメッセージボックスが表示</title>
6      <script type="text/ecmascript">
7      //      <![CDATA[
8          function click(event) {
9              alert("Circle " +event.target.getAttribute("fill")+" clicked.");
10         }
11     //  ]]>
12   </script>
13   <circle cx="50" cy="50" r="20" fill="red"    onclick="click(evt)" />
14   <circle cx="100" cy="50" r="20" fill="blue"   onclick="click(evt)" />
15   <circle cx="150" cy="50" r="20" fill="green" onclick="click(evt)" />
16 </svg>

```

- 3つの円は13行目から15行目に定義されています。
- これらの円には onclick があり、すべてが click(evt) となっています。該当するオブジェクトの上でクリックされる (というイベントが発生する) と JavaScript 内にかけられた関数 click(evt) が呼び出されます。

ここで evt は MouseEvent という型のオブジェクトになります。このオブジェクトには次のようなプロパティとメソッドがあります (表 7.4 と [25, AppendixC] を参照)。なお、プロパティはすべて読み出しのみ可能です。

- ここに現れた変数 evt はシステムが用意している変数でこのときに発生したイベントの各種情報を格納しています。イベントは常に発生していますので発生時のイベントの情報を参照するために引数として必ず渡します。

- 上で呼ばれている関数は JavaScript で書かれています。このプログラムは6行目から11行目に書かれています。この部分は<script> 要素の中に書きます。
- <script> 要素のプログラムがどのように書かれているかを定義するのが type です。ここでは text/ecmascript となっています³。
- JavaScript 内で<が要素の開始とブラウザに解釈されないための対策としてプログラムは<! [CDATA と]]>内に書きます。⁴ この行が JavaScript として解釈されることを防ぐために //で注釈しています。現在のブラウザではこの対策はしなくてもよいようです。
- 8行目が関数の宣言です。function が先頭に付きます。関数名とその後にある () 内に仮引数名を書きます。
- 9行目の alert は画面上にメッセージボックスを表示するメソッドです（オブジェクト名が省略されています。このオブジェクトは window オブジェクトです。したがって、正式には window.alert(...) と書きます）。引数の値をメッセージボックスに表示します。
- 渡された引数は"Circle " +event.target.getAttribute("fill")+" clicked." です。ここで+は文字列をつなげる演算子です。文字列と数字を+でつなげると JavaScript では数字を文字列に直してからつなげます。
- target は event が発生したオブジェクトです。
- getAttribute() はオブジェクトの指定された属性（ここでは fill）の値を返すメソッドです。したがって、ここではクリックされた色の名称が得られ、これがメッセージボックスに表示されます。

問題 7.2 fill を red のかわりに#FF0000(赤) と定義した場合にはどのような表示になるか確認しなさい。

問題 7.3 いろいろな図形を用意してクリックしたときに別の属性値を表示しなさい。

次のリストは各オブジェクトにイベント処理の属性を定義しないで開始時にイベントの処理を設定する方法です。最近のプログラムスタイルではこのような形式が推奨されています。

SVG リスト 7.2: マウスのクリックを検出する SVG(その 2) -- イベントハンドラの登録

```

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>クリックするとメッセージボックスが表示</title>
6      <script type="text/ecmascript">
7      //    <! [CDATA[
8      window.onload = function() {
9          var Cs = document.getElementsByTagName("circle");

```

³ecmascript は JavaScript の国際的な規格のもとです。text/javascript としてもかいません。

⁴XML の規格では CDATA セクションと呼ばれます。

```
10     for( var i=0; i< Cs.length; i++) {
11         Cs[i].addEventListener("click",click, false);
12     }
13 }
14 function click(event) {
15     alert("Circle " +event.target.getAttribute("fill")+" clicked.");
16 }
17 // ]]></script>
18 <circle cx="50" cy="50" r="20" fill="red" />
19 <circle cx="100" cy="50" r="20" fill="blue"/>
20 <circle cx="150" cy="50" r="20" fill="green" />
21 </svg>
```

- 一番の違いは18行目から20行目で定義されている<circle>要素のオブジェクトに`onclick`が定義されていないことです。
- その代わりに`onload`のイベントを処理する関数が定義されています(8行目から13行目)。`onload`のイベントはSVG文書がすべて読み込まれた後、呼び出されます。
- まず、<circle>要素を持つオブジェクトのリストを得ます(9行目)。`getElementsByName`がそのメソッドです。これはオブジェクトのコレクションを返します。コレクション内のオブジェクトの数は`length`プロパティで得られます。
- このコレクションのメンバーに対して`click`イベントに対する関数を割り当てます(11行目)。そのメソッドが`addEventListener`です。引数は「イベント名」、呼び出されるイベント処理関数名とイベントバーリングを制御するフラグです。イベントバーリングについては144ページ以降で詳しく解説します。
- `addEventListener`で割り当てられたイベント処理関数はイベント情報をその引数で得ることができます(14行目)。その引数を用いて処理するのは前と同じです。

クリックされたオブジェクトの属性値を変える

前の例でSVGのオブジェクトの属性値を`getAttribute()`メソッドを用いて読み出すことができました。`getAttribute()`メソッドの代わりに`setAttribute()`メソッドを用いると属性値を設定できます。

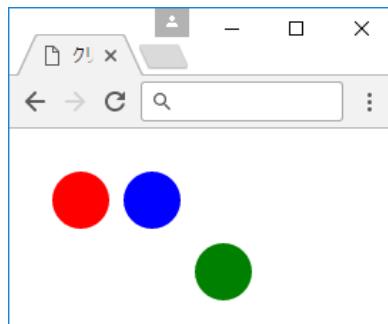


図 7.7: クリックするとその円が移動します

SVG リスト 7.3: マウスのクリックを検出する SVG(その 3)

```

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>クリックするとその円が移動</title>
6      <script type="text/ecmascript">
7      //  <![CDATA[
8          window.onload = function() {
9              var Cs = document.getElementsByTagName("circle");
10             for( var i=0; i< Cs.length; i++) {
11                 Cs[i].addEventListener("click",click, false);
12             }
13         }
14         function click(T) {
15             var Y = 150-parseInt(T.target.getAttribute("cy"));
16             T.target.setAttribute("cy",Y);
17         }
18     //  ]]></script>
19     <circle cx="50" cy="50" r="20" fill="red"/>
20     <circle cx="100" cy="50" r="20" fill="blue"/>
21     <circle cx="150" cy="50" r="20" fill="green"/>
22  </svg>

```

この例ではマウスをクリックするとクリックした円が下方に移動し、再び同じ円をクリックすると元の位置に戻るようになっています。リスト 7.2 と異なるのは `onload` を処理する関数の指定法と 14 行目から始まる `click` 関数の処理内容だけです。

- 8 行目から 13 行目で `onload` イベントが発生したときに呼び出される関数を定義しています。ここでは、リスト 7.1 と同様に、3 つの円に `click` イベント処理関数を定義しています。
- 15 行目で変数 `Y` を宣言すると同時に、クリックのイベントがおきたオブジェクトの新しい `y` 座標を計算しています。
- まず、15 行目で `getAttribute("cy")` メソッドでクリックされた円の中心の `y` 座標の値を取得します。

- この値は文字列なので、JavaScriptの関数parseIntで文字列を整数値に変換します。前の演算子が-なのでこの場合には不要ですが、いつも整数で計算するのであれば必ず変換が行われるように記述したほうがよいでしょう。

開始時の値が20なので最初にクリックされるとこの値は $150 - 20 = 130$ になり、次にクリックされると呼び出される値が130なので $150 - 130 = 20$ と最初の値に戻ります。

- 16行目では計算結果の数値を属性cyにsetAttribute()メソッドを用いてその値を設定しています。属性値は数値ではなく文字列ですが、設定するときは数値を文字列に自動的に変換するようです。

他のオブジェクトの属性値を変える

次の例では円の外でマウスをクリックするとその位置に中心が移動します。この例ではマウスのクリックした位置を取得する方法が問題になります。

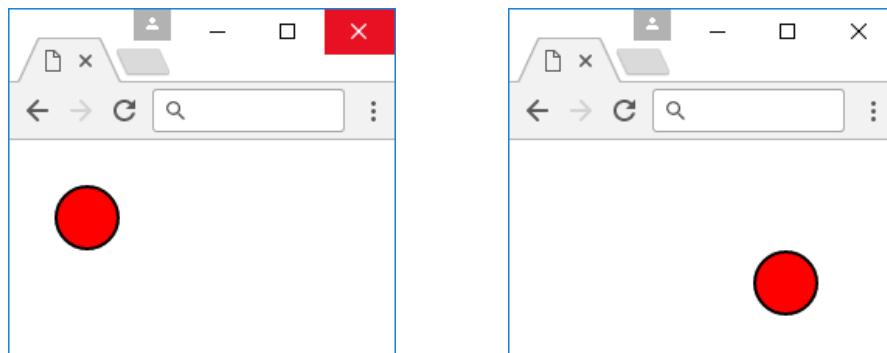


図 7.8: クリックした位置に円が移動します

SVGリスト7.4: マウスのクリックを検出するSVG(その3)

```

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>マウスをクリックした位置に円が移動</title>
6      <script type="text/ecmascript">
7 //      <![CDATA[
8      window.onload = function() {
9          document.getElementById("Canvas").addEventListener("click",click, false);
10     }
11     function click(E) {
12         var T = document.getElementById("Circle");
13         T.setAttribute("cx",E.clientX);
14         T.setAttribute("cy",E.clientY);
15     }
16 //    ]]></script>
17     <rect x="0" y="0" width="100%" height="100%" fill="white" id="Canvas" />

```

```
18 <circle id="Circle" cx="50" cy="50" r="20" fill="red"
19   stroke="black" stroke-width="2" />
20 </svg>
```

- 前回の例ではイベントが起きたオブジェクトの属性値を変更していましたが、今回の例ではイベントの起きたオブジェクトと操作するオブジェクトが異なることに注意してください。
- このために、操作されるオブジェクトに `id` で名前をつけています(19行目)。
- また、イベントを捕らえるためこのオブジェクトとして`<rect>`要素を用意しています(17行目)。
- この長方形に9行目で `click` イベントを処理する関数を割り当てています。
`getElementById("Canvas")` で得られた結果はオブジェクトなので、その後に直接、`addEventListener` を記述して、そのオブジェクトにイベント処理関数の設定を行っています。
- `click` イベントが発生したときに移動させるオブジェクトには `Circle` という名前がついていますので(19行目の `id`)これを手がかりに SVG のノードを `getElementById("Circle")` で得て、変数 `T` にセットします。この操作はイベントが発生するごとに、実行されます。
- このオブジェクトにマウスがクリックされた `x` 座標を求め(`clientX`)その値を属性値`"cx"`に `setAttribute()` を用いて設定しています(13行目)。
- ”`cy`” も同様に設定しています(14行目)。

例 7.4 では円上でクリックした場合は円の移動が起きません。円上でも動くようにするために円に対してもクリックイベントの処理関数を定義するなどの対策が必要となります。より簡単な方法については次の問を考えてください。

問題 7.4 リスト 7.4 で次のことを確かめなさい。

- 円上でクリックしても円が移動しない。
- 17行目の長方形の属性 `fill` の値を `none` にすると円が移動しない。
- 17行目と19行目を交換して、`<rect>`要素の属性として `fill-opacity="0"`を追加すると、円の上でクリックしたときも円が移動する。
- 長方形を取り除いて`<svg>`要素にイベント処理関数を割り当たらどうなるか確かめる。

クリックした位置を SVG 内に表示する

図 7.9 はクリックした位置に円が移動するだけではなく、その位置が SVG 内に表示されます。リスト 7.5 は図 7.9 のソースコードです。

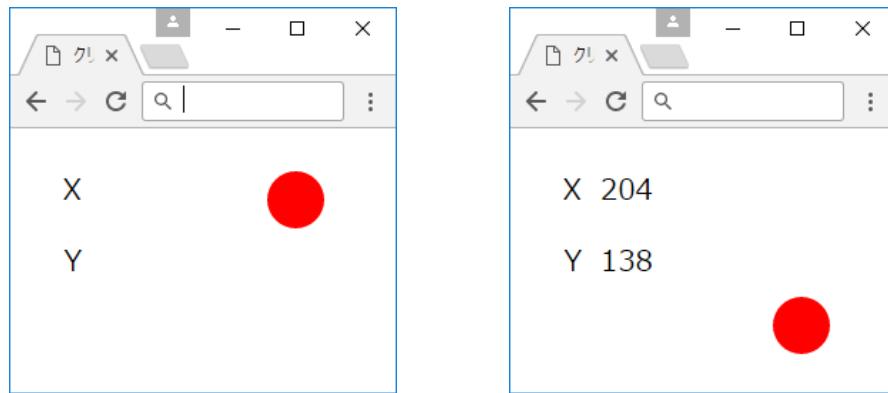


図 7.9: クリックした位置を SVG 内に表示する

SVG リスト 7.5: クリックした位置を SVG 内に表示する

```

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>クリックした位置を SVG 内に表示</title>
6      <script type="text/ecmascript">
7      //      <![CDATA[
8          window.onload = function() {
9              document.getElementById("Rect").addEventListener("click", click, false);
10         }
11         function click(event) {
12             document.getElementById("XP").firstChild.nodeValue=event.clientX;
13             document.getElementById("YP").firstChild.nodeValue=event.clientY;
14             document.getElementById("Circle").setAttribute("cx",event.clientX);
15             document.getElementById("Circle").setAttribute("cy",event.clientY);
16         }
17     //  ]]></script>
18     <style type="text/css">
19         .textStyle {
20             font-size:20px; text-anchor:end;
21         }
22     </style>
23     <circle id="Circle" cx="200" cy="50" r="20" fill="red"/>
24     <text class="textStyle" x="50" y="50"> X</text>
25     <text class="textStyle" id="XP" x="100" y="50"> </text>
26     <text class="textStyle" x="50" y="100"> Y</text>
27     <text class="textStyle" id="YP" x="100" y="100"> </text>
28     <rect id="Rect" x="0" y="0" width="100%" height="100%" opacity="0" />
29 </svg>
```

- クリックした位置を表示するためにラベルとして 2 つと x 座標と y 座標の値を表示する `<text>` 要素を作成しています (24 行目から 27 行目)。この`<text>` 要素において値を表示する部分に空白がひとつ入っていることに注意してください。

- また、文字の表示を統一するために CSS を用いてフォントの大きさや位置を指定しています(18 行目から 21 行目)。
- 画面上でクリックすると 28 行目で定義されている長方形上で click イベントが発生します。この長方形は不透明度が 0 なので下にあるオブジェクトはすべて見えます。
- この長方形には onload イベントで click イベントを処理する関数を割り当てています(9 行目)。
- 12 行目と 13 行目でそれぞれ x 座標と y 座標の値を先ほどの <text> 要素に設定しています。<text> 要素の属性としてではなく子のノードとして設定するために最初の子のノードを指定する firstChild プロパティのノードの値 (nodeValue) に位置の値を設定しています。
- 25 行目と 27 行目の <text> 要素に空白が入っていないとこれらの要素に firstChild がないことになり設定が失敗に終わります。

問題 7.5 25 行目と 27 行目の <text> 要素内の空白を取り除くと値が表示できないことを確認しなさい。

これを避けるには次のように firstChild がない場合(値が null)には新しいノードを付け加える操作が必要になります。

SVG リスト 7.6: クリックした位置を SVG 内に表示する(改良版)

```

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>クリックした位置を SVG 内に表示(改良版)</title>
6      <script type="text/ecmascript">
7      //      <![CDATA[
8          window.onload = function() {
9              document.getElementById("Rect").addEventListener("click", click, false);
10         }
11         function click(e) {
12             SetText("XP", "cx", e.clientX);
13             SetText("YP", "cy", e.clientY);
14         }
15         function SetText(name, attrib, Value) {
16             var txtNode = document.createTextNode(Value);
17             var newElement = document.getElementById(name);
18             if(newElement.firstChild) {
19                 newElement.replaceChild(txtNode, newElement.firstChild);
20             } else {
21                 newElement.appendChild(txtNode);
22             }
23             document.getElementById("Circle").setAttribute(attrib, Value);
24         }
25     //  ]]></script>
26     <style type="text/css">
27         .textStyle {

```

```

28     font-size:20px; text-anchor:end;
29   </style>
30   <circle id="Circle" cx="200" cy="50" r="20" fill="red"/>
31   <text class="textStyle" x="50" y="50"> X</text>
32   <text class="textStyle" id="XP" x="100" y="50"></text>
33   <text class="textStyle" x="50" y="100"> Y</text>
34   <text class="textStyle" id="YP" x="100" y="100"></text>
35   <rect id="Rect" x="0" y="0" width="100%" height="100%" opacity="0"/>
36 </svg>

```

- リスト7.5とは異なり、32行目と34行目の開始タグと終了タグには空白がありません。
- この例では値の表示などを関数SetText()で行っています(12行目と13行目)。
- この関数は15行目から24行目で定義されています。この関数は次のことを行います。
 - 属性名idの属性値、円の要素の属性名と数値を引数に取ります。
 - 属性名idの属性値で指定された要素の子要素として与えられた数値を表示します。
 - 円の属性値を与えられた値に設定します。
- まず16行目でcreateTextNodeメソッドで表示するためのテキストノードを作成しています。
- 17行目で指定されたノードを得ています。
- このノードに子ノードがある(18行目のnewElement.firstChildがnullにならない)場合にはそのnewElement.firstChildをreplaceChildメソッドを用いて先ほど作成したノードと置き換えます(19行目)。
- 子ノードがない場合には先ほど作成したノードをappendChildメソッドを用いて子ノードを付け加えます(21行目)。

32行目の<text class="textStyle" id="XP" x="100" y="50"></text>を<text class="textStyle" id="XP" x="100" y="50" />としても動作することも確認してください。

7.3.2 イベントの処理の詳細

DOM Level 2 のイベント処理モデル

DOM Level 2 のイベント処理モデルではあるオブジェクトの上でイベントが発生すると次の順序で各オブジェクトにイベントの発生が伝えられます。

1. 発生したオブジェクトを含む最上位のオブジェクトにイベントの発生が伝えられます。このオブジェクトにイベント処理関数が定義されていなければなにも起きません。
2. 以下順にDOMツリーに沿ってイベントが発生したオブジェクトの途中にあるオブジェクトにイベントの発生が伝えられます。

3. イベントが発生したオブジェクトまでイベントの発生が伝えられます。
4. その後、DOM ツリーに沿ってこのオブジェクトを含む最上位のオブジェクトまで再びイベントの発生が伝えられます。

DOM Level 2 のモデルではイベントが発生したオブジェクトはイベントの `target` プロパティで、イベントを処理しているオブジェクトは `currentTarget` で参照できます。

このイベント処理の前半部分である最上位のオブジェクトからイベントが発生したオブジェクトにイベントの発生が伝播する段階をイベントキャプチャリングといい、後半の部分のイベントが発生したオブジェクトから最上位のオブジェクトへ伝播する段階をイベントバーリングと呼ばれます⁵。

`addEventListener` の 3 番目の引数で `false` にするとイベント処理はイベントバーリングの段階で呼び出されます。また、`true` にするとイベント処理はイベントキャプチャリングの段階で呼び出されます。

リスト 7.2 の改良

リスト 7.2 では 3 つの円にそれぞれイベント処理の関数を付けましたが実は一番上のオブジェクトにだけイベント処理関数を付ければ十分であることがイベントキャプチャリングなどイベント処理の手順からわかります。

次のリストはそのように改良したものです。

SVG リスト 7.7: マウスのクリックを検出する SVG(その 1) の改良

```

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>クリックするとメッセージボックスが表示(改良版)</title>
6      <script type="text/ecmascript">
7      //  <![CDATA[
8          window.onload = function() {
9              var Cs = document.getElementById("Canvas");
10             Cs.addEventListener("click",click, false);
11         }
12         function click(event) {
13             alert("Circle " +event.target.getAttribute("fill")+" clicked.");
14         }
15     // ]]></script>
16     <g id="Canvas">
17         <circle cx="50" cy="50" r="20" fill="red"/>
18         <circle cx="100" cy="50" r="20" fill="blue"/>
19         <circle cx="150" cy="50" r="20" fill="green"/>
20     </g>
21 </svg>
```

- 16 行目から 20 行目で 3 つの円を含む`<g>` 要素を用意します。

⁵バブルは泡のことです。泡は下から上に上がっていくのでこう呼ばれています。

- 10行目でこの<g>要素にclickのイベントハンドラーを結び付けています。
- 13行目でクリックされた要素名をevent.target.tagNameから得ています。

問題 7.6 リスト 7.7について次のことをしなさい。

1. リスト 7.7がリスト 7.2と同じ動作をすることを確認しなさい。
2. 16行目から始まる<g>要素の代わりに<svg>要素にイベントハンドラーをつけたときの動作を確認しなさい。

7.3.3 マウスのドラッグを処理する

ドラッグとは

ドラッグとはあるオブジェクト上でマウスボタンを押したままマウスポインターを動かしてそのオブジェクトを移動させることです。この間、ユーザーは次の動作をしています()内はその間に起こるイベントです。

- オブジェクト上でマウスボタンを押す(mousedownイベントが発生する)。
- ボタンを押したままマウスポインタを動かす(mousemoveイベントが発生する)。
- オブジェクト上でマウスボタンをはなす(mouseupイベントが発生する)。

ドラッグの操作中に()内のイベントを処理する必要があります。ここで注意して欲しいのはmousemoveイベントはボタンが押されているいないにかかわらず発生していることです。したがって、一連の処理は次のようにになります。

- マウスボタンが押されたオブジェクトを記録する。
- 押された状態のままmousemoveが発生している間、そのオブジェクトを動かす。
- mouseupイベントが発生したらオブジェクトの記録をなくす。

という処理手順になります。なお、マウスボタンが押されたままの状態とはマウスボタンが押されてからmouseupイベントが発生していない間であることに注意してください。

なお、clickイベントとはmousedownとmouseupという一連の操作です。実際にはmousedownとmouseupのイベントの後にclickイベントが発生します。

オブジェクトのドラッグ処理の例

図7.10はオブジェクトをドラッグする例です。この図のリストはリスト7.8です。

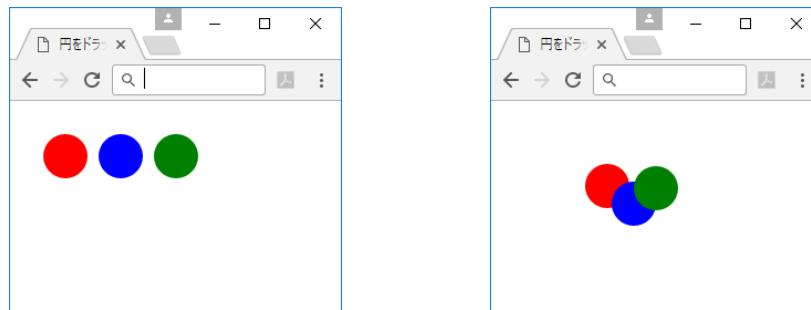


図 7.10: 3 つの円がそれぞれドラッグで移動可能 (右: 初期状態、左: 3 つの円を移動したとき)

SVG リスト 7.8: ドラッグの例

```

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>円をドラッグで移動する</title>
6  <script type="text/ecmascript">
7  //  <![CDATA[
8  var G, inDragging;
9  window.onload = function() {
10     G = document.getElementsByTagName("svg")[0];
11     G.addEventListener("mousedown", mdown, false);
12     G.addEventListener("mouseup", mup, false);
13 }
14 function mdown(event) {
15     inDragging = event.target;
16     G.addEventListener("mousemove", mmmove, false);
17 }
18 function mmmove(event) {
19     inDragging.setAttribute("cx", event.clientX);
20     inDragging.setAttribute("cy", event.clientY);
21 }
22 function mup(event) {
23     G.removeEventListener("mousemove", mmmove, false);
24 }
25 //  ]]></script>
26 <circle cx="50" cy="50" r="20" fill="red"/>
27 <circle cx="100" cy="50" r="20" fill="blue"/>
28 <circle cx="150" cy="50" r="20" fill="green"/>
29 </svg>
```

- この例は例 7.6 と同様に 3 つの円を 26 行目から 28 行目の間で定義しています。
- <svg> 要素のオブジェクトを getElementsByTagName を用いて求めています (10 行目)。このメソッドは要素のリストが得られるので (この場合は一つです)、[0] でルート要素への参照となります。

- このオブジェクトに対してmousedown, mouseup のイベントに対する関数を定義しています(11行目と12行目)。
- mousedown イベントに対してはマウスが押されたオブジェクトをグローバル変数 inDragging 变数に保存しています(15行目)。また、16行目で画像全体にmousemove のイベント処理関数を割り当てています。
- mouseup イベントに対してはグローバル変数 mousemove の設定関数を取り除いています(23行目)。したがって、イベントハンドラ-はドラッグしている間だけ処理が行われます。
- 18行目から21行目でmousemove の処理をしています。
- イベントが発生したマウスポインタの位置をドラッグ中の円の中心位置に設定しています(19行目と20行目)。
- なお、mousemove イベントはいつもドラッグ中の円で起こっているわけではありません。二つの円が重なった場合、上に表示されているほうの円で起こっています。イベントを処理する関数が<svg>要素についているので円からマウスカーソルが離れてもイベントが拾えます。
- 22行目から24行目ではマウスが離されたときの処理を定義しています。ここでは登録されたmousemove イベント処理関数を取り除いています(23行目)。

なお、このリストでは円の数を増やしてもプログラムの部分は変える必要がありません。

問題 7.7 リスト 7.8 で次のことを行いなさい。

- 円の数を増やしなさい。
- ドラッグする图形に橙円を付け加えなさい。
- ドラッグする图形に正方形を付け加えなさい。要素は<rect>要素でなくともかまいません。

図 7.11 は図 6.10 の下部にスクロールバーを付けて baseFrequency の値をインターラクティブに変更できるようにしたものです。なお、このフィルターは表示するのに時間がかかるのでスクロールバーを移動している間は图形を表示しなおず、ドラッグが終了後に図を表示するようにしています。また、baseFrequency の値はドラッグしている間も変化しています。

SVG リスト 7.9: feTurbulence フィルタフィルタのパラメータをスクロールバーで設定する

```

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>feTurbulence フィルタのパラメータをスクロールバーで設定する</title>
6  <script type="text/ecmascript">
7  //  <![CDATA[
8  var DragObj, OffsetX, FilterObj, BFVal, BF, Base;
9  window.onload = function(event) {
10     Base= document.getElementById("base");
11     Base.addEventListener("mouseup",endDrag,true);

```

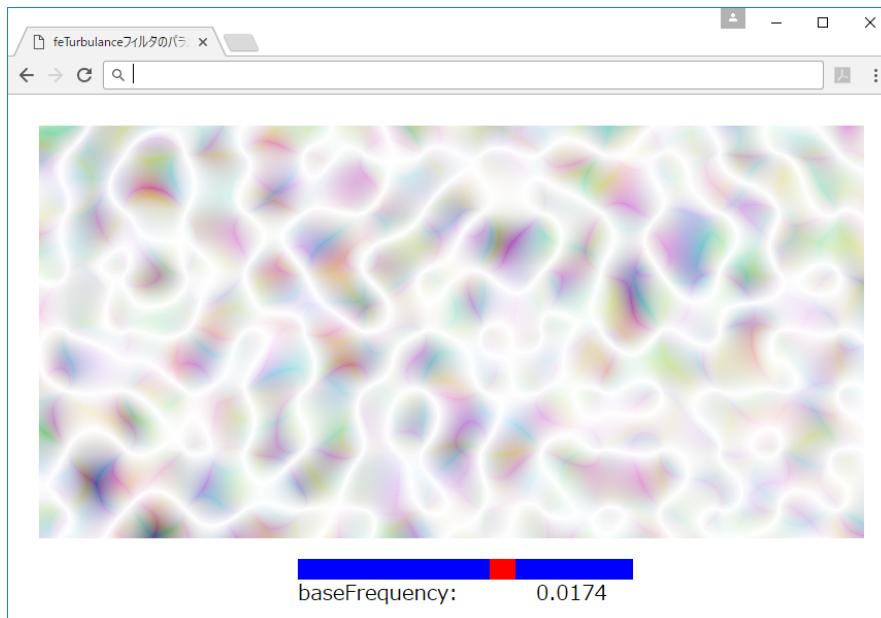


図 7.11: feTurbulence フィルタフィルタのパラメータをスクロールバーで設定

```
12     BFVal = document.getElementById("BFValue");
13     FilterObj = document.getElementById("FilterfeTurbulence");
14     DragObj = document.getElementById("Fbar");
15     DragObj.addEventListener("mousedown",beginDrag,false);
16     setBF(150);
17     FilterObj.setAttribute("baseFrequency", BF);
18 }
19 function beginDrag(event) {
20     Base.addEventListener("mousemove",Dragging,true);
21     OffsetX = DragObj.getAttribute("x")-event.clientX;
22 }
23 function Dragging(event) {
24     var PS = event.clientX+OffsetX;
25     if(PS<=0) PS =0;
26     if(PS >300) PS = 300;
27     setBF(PS);
28 }
29 function setBF(x) {
30     BF = Math.pow(10,-3+x/150.);
31     DragObj.setAttribute("x", x);
32     BFVal.firstChild.nodeValue=BF.toFixed(4);
33 }
34 function endDrag(event) {
35     FilterObj.setAttribute("baseFrequency", BF);
36     Base.removeEventListener("mousemove",Dragging,true);
37 }
38 // ]]></script>
39 <title> Scroll Bar </title>
```

```

40  <defs>
41      <filter id="Filter" filterUnits="objectBoundingBox"
42          x="0%" y="0%" width="100%" height="100%">
43          <feTurbulence type="turbulence" id="FilterfeTurbulence"
44              baseFrequency="0.005" numOctaves="1"/>
45      </filter>
46      <style type="text/css">
47          text { font-size:20px; }
48      </style>
49  </defs>
50  <g id="base">
51      <rect width="100%" height="100%" fill="white"/>
52      <rect x="30" y="30" width="800" height="400" filter="url(#Filter)" />
53      <g transform="translate(281,450)">
54          <rect x="0" y="0" width="325" height="20" fill="blue"/>
55          <rect id="Fbar" x="21" y="0" width="25" height="20" fill="red" />
56          <text y="40" >baseFrequency: </text>
57          <text id="BFValue" x="300" y="40" text-anchor="end" > </text>
58      </g>
59  </g>
60 </svg>
```

- 9行目から18行目はSVG文書が表示終了後に呼び出される初期化の関数です。图形のオブジェクトへの参照を変数に格納し、mousedownのイベント処理関数をスクロールバーの位置を示す小さな長方形に付けています。
- 19行目から22行目はスクロールバーの位置を示す小さな長方形上でmousedownが発生したときに呼び出される関数です。画面全体にmousemoveのイベント処理関数を割り当て、イベントが発生した位置とスクロールバーの位置を示す小さな長方形のxの値との差を格納します。
- 23行目から28行目はmousemoveのイベント処理関数です。スクロールバーの動く範囲を0から299の間に設定するようにしたあと、それに対応するbaseFrequencyの値を計算する関数(SetBF())を呼び出します。
- 29行目から33行目はbaseFrequencyの値を計算するための関数です。この値は指数的に変化させるほうが图形の変化が良くわかるので、スクロールバーの位置をもとに 10^{-3} から 10^{-1} まで変化するようにしています(30行目)。このあたりの数値で图形が十分に細くなってしまいます。スクロールバーの長さが325(300+小さな長方形の幅)なので $10^{-3+位置/150}$ で計算しています。

式に出てくるMath.pow(x, y)は x^y を求める関数です。ここでは 10^x を計算していることになります。Mathオブジェクトのメソッドについては表7.5を参照してください。

- 図の部分は次のようになっています。

```

50  <g id="base">
51      <rect width="100%" height="100%" fill="white"/>
52      <rect x="30" y="30" width="800" height="400" filter="url(#Filter)" />
```

```
53 <g transform="translate(281,450)">
54   <rect x="0" y="0" width="325" height="20" fill="blue"/>
55   <rect id="Fbar" x="21" y="0" width="25" height="20" fill="red" />
56   <text y="40" >baseFrequency: </text>
57   <text id="BFValue" x="300" y="40" text-anchor="end"> </text>
58 </g>
59 </g>
```

- mousemove を画面全体で拾えるように、全体を白で塗った長方形を用意します (51 行目)。このイベント自体は50 行目で定義している<g> 要素が受け取るようにしています。
- フィルターで塗りつぶした長方形を用意します (52 行目)。
- 画面の下部にスクロールバーが乗る長方形をおきます (54 行目)
- その上に小さな長方形を置きます (55 行目)。
- 値を表示するために<text> 要素を用意します (57 行目)。
この中に空白の文字列を置くことにより、テキストノードが置かれることになります。
これにより32 行目で BFVal.firstChild.nodeValue が正しく動作します。

問題 7.8 図 7.12 は図 7.11 に numOctaves の値を設定するボタンを追加したものです。numOctaves の値は整数しか意味がないようなのでこのようにしました。

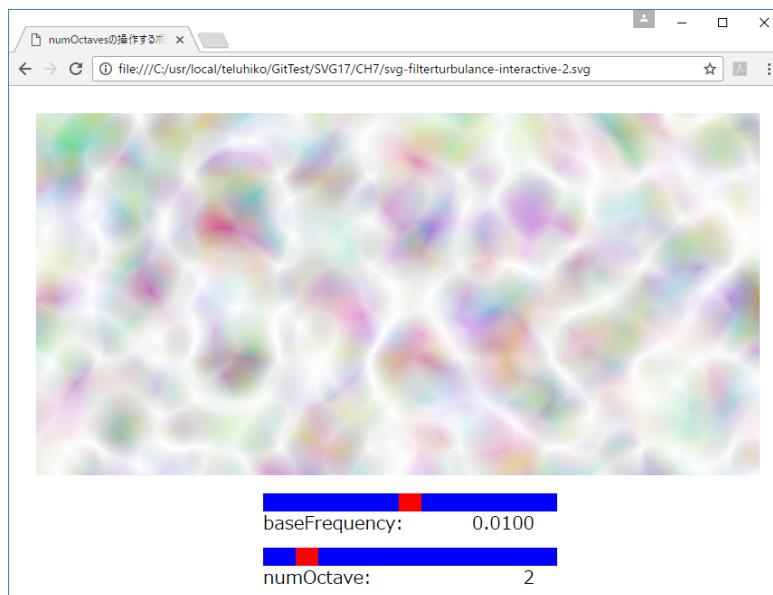


図 7.12: feTurbulence に numOctaves の操作するボタンを追加する

表 7.5: JavaScript における Math オブジェクトの種類

名称	種類	説明
Math.E	定数	自然対数の底 ($2.71828182\dots$)
Math.LN10	定数	$\log_e 10$
Math.LN2	定数	$\log_e 2$
Math.LOG2E	定数	$\log_2 e$
Math.LOG10E	定数	$\log_{10} e$
Math.PI	定数	円周率 ($\pi = 3.141592\dots$)
Math.SQRT1_2	定数	$dfrac{1}{2}$ の平方根 $\sqrt{\frac{1}{2}}$
Math.SQRT2	定数	2 の平方根 $\sqrt{2}$
Math.abs(x)	関数	x の絶対値
Math.acos(x)	関数	逆余弦関数 $\arccos x$
Math.asin(x)	関数	逆正弦関数 $\arcsin x$
Math.atan(x)	関数	逆正接関数 $\arctan x$
Math.atan2(y, x)	関数	$\arctan \frac{y}{x}$ を計算。 x が 0 のときでも正しく動く
Math.ceil(x)	関数	x 以上の整数で最小な値を返す。
Math.cos(x)	関数	余弦関数 $\cos x$
Math.exp(x)	関数	指数関数 e^x
Math.floor(x)	関数	x の値を超えない整数
Math.log(x)	関数	自然対数 $\log x$
Math.max([x1, x2, ..., xN])	関数	与えられた引数のうち最大値を返す
Math.min([x1, x2, ..., xN])	関数	与えられた引数のうち最小値を返す
Math.pow(x, y)	関数	指数関数 x^y
Math.random()	関数	0 と 1 の間の擬似乱数を返す
Math.round(x)	関数	x の値を四捨五入する
Math.sin(x)	関数	正弦関数 $\sin x$
Math.sqrt(x)	関数	平方根を求める。 \sqrt{x}
Math.tan(x)	関数	正接関数 $\tan x$

7.3.4 オブジェクトを追加する

リスト 7.8 を応用すると 2 点の間をドラッグして直線を引く SVG ファイルが作成できます。複数の直線を引くためには直線のオブジェクトが複数必要になります。いくつ必要になるかを前もってわかりませんので、実行時にオブジェクトを作成する必要があります。新しい要素を作成する DOM のメソッドは `createElement` です。このメソッドは `document` におけるメソッドです。`createElement` の代わりに新規作成する要素が定義されている名前空間を指定して新しい要素を定義する `createElementNS` を用います。

図 7.13 はいくつかの直線を引いた後のものです。

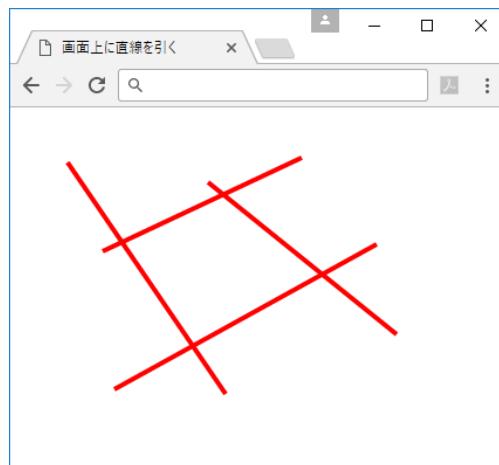


図 7.13: 画面上に直線を引く

SVG リスト 7.10: 画面上に直線を引く

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3     height="100%" width="100%">
4     <title>画面上に直線を引く</title>
5     <script type="text/ecmascript">
6 //<![CDATA[
7     var svgNS ="http://www.w3.org/2000/svg";
8     var C, NewLine = null;
9     window.onload = function() {
10         C = document.getElementById("Canvas");
11         C.addEventListener("mousedown",mdown, false);
12         C.addEventListener("mouseup",mup, false);
13     }
14     function mdown(E) {
15         NewLine = document.createElementNS(svgNS, "line");
16         NewLine.setAttribute("x1",E.clientX);
17         NewLine.setAttribute("y1",E.clientY);
18         NewLine.setAttribute("x2",E.clientX);
19         NewLine.setAttribute("y2",E.clientY);
20         NewLine.setAttribute("stroke","red");
21         NewLine.setAttribute("stroke-width","4");
22         C.appendChild(NewLine);
23         C.addEventListener("mousemove",mmove, false);
24     }
25     function mmmove(E) {
26         NewLine.setAttribute("x2",E.clientX);
27         NewLine.setAttribute("y2",E.clientY);
28     }
29     function mup(E) {
30         C.removeEventListener("mousemove",mmove, false);</pre>
```

```

31      }
32  //]]></script>
33  <g id="Canvas">
34    <rect x="0" y="0" width="100%" height="100%" fill="white"/>
35  </g>
36 </svg>

```

- SVG のロードが終了した後、ここで指定された関数が呼び出されます (9 行目から13 行目)。
 - 変数 C に id が Canvas である<g> 要素への参照を格納します (10 行目)。
 - このオブジェクトに mousedown と mouseup のイベントの処理関数を割り当てます (11 行目と12 行目)。
- 白で塗られた長方形が画面全体にあるので (34 行目)mousedown のイベントが発生するとこの長方形の親要素である Canvas で割り当てられたイベント処理関数 mdown が呼び出されます (14 行目から24 行目)。
 - 15 行目で<line> 要素を新規に作成します。 <line> 要素は SVG の要素なので名前空間に SVG の名前空間"http://www.w3.org/2000/svg"を指定します。
 - この<line> 要素に、直線の開始位置や終了位置をイベントが発生した位置に設定します (16 行目から19 行目)。これらの位置はイベントオブジェクトのプロパティ clientX や clientY で取得できます。
 - 線幅と色を設定します (20 行目と21 行目)。
 - 作成した<line> 要素を付け加えます (22 行目)。
 - この親要素に mousemove のイベント処理関数を割り当てます。
- 25 行目から28 行目はドラッグ中の処理です。マウスカーソルの位置を、追加している直線の終端の点の位置として設定しています。
- ドラッグが終了した場合の処理は29 行目から31 行目で定義しています。イベント処理関数を取り除いています。

問題 7.9 次のことを確かめたり、上記の SVG 文書の改良を行いなさい。

1. いくつか直線を引いた後で SVG の上で右クリックした場合、ソースは元のものと変わっているか調べなさい。
2. Chrome で要素が追加されていることを確認しなさい。
3. Chrome で要素の上で右クリックで現れるコンテキストメニューから「Edit as HTML」を選択するとどうなるか確認しなさい。
4. SVG の画面に色が異なる長方形をいくつか置き、そのうちのひとつをクリックした後で直線を引くとその直前にクリックした長方形の色で直線が引けるようにしなさい(図 7.14 参照)。

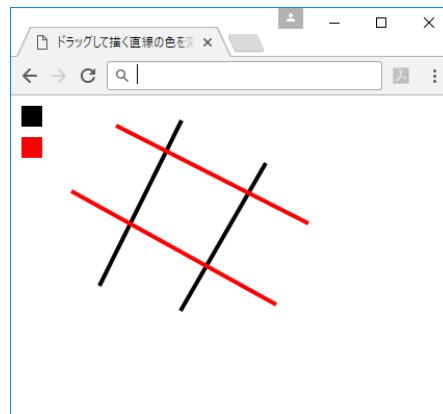


図 7.14: 色を変えて直線を引く

5. 長方形が描けるようにしなさい。width や height を負の値にすると図形が表示されないのでこれを避けるためにチェックが必要です。

リスト 7.8 ではドラッグ中の図形が他の図形の下に隠れるという使い勝手が悪い点があります。リスト 7.11 ではこの点を改良しています。

SVG リスト 7.11: ドラッグ処理の改良

```

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>円をドラッグで移動する--改良版</title>
6  <script type="text/ecmascript">
7  //  <![CDATA[
8  var G, inDragging;
9  window.onload = function() {
10     G = document.getElementsByTagName("svg")[0];
11     G.addEventListener("mousedown", mdown, false);
12     G.addEventListener("mouseup", mup, false);
13 }
14 function mdown(event) {
15     if(event.target.nodeName !== "svg") {
16         inDragging = event.target;
17         G.appendChild(inDragging)
18         G.addEventListener("mousemove", mmmove, false);
19     }
20 }
21 function mmmove(event) {
22     inDragging.setAttribute("cx", event.clientX);
23     inDragging.setAttribute("cy", event.clientY);
24     event.stopPropagation();
25 }
26 function mup(event) {

```

```

27     G.removeEventListener("mousemove", mmove, false);
28   }
29 // ]]></script>
30 <circle cx="50" cy="50" r="20" fill="red"/>
31 <circle cx="100" cy="50" r="20" fill="blue"/>
32 <circle cx="150" cy="50" r="20" fill="green"/>
33 </svg>
```

17行目ではドラッグが開始されたオブジェクトを親オブジェクトの最後の子要素にします。メソッド `appendChild` は、新規の子オブジェクトを追加するメソッドですが、すでに子要素になっているものに対しては子要素のリストの最後に移動します。したがって、一番上に表示されることになります。これにより移動中のオブジェクトが他のオブジェクトの下に来るという問題点を解消できます。

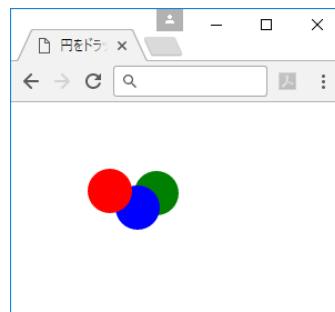


図 7.15: 3つの円がそれぞれドラッグで移動可能(改良版)

円の表示順序が変わっていることに注意してください。

問題 7.10 フィックの錯視の垂直の直線の長さをマウスのドラッグで変えるものを作成しなさい。

7.4 起動時にJavaScriptで要素を作成する

7.4.1 任意の形の曲線を描く

図 7.16 は直線上を円が滑らずに回転していくとき、その円周上に固定された点が描く軌跡です。この曲線はサイクロイドとよばれています。

半径 r の円周上に固定された点が回転する直線上に初めにあるとき、角 θ だけ円が回転したときの点の位置は

$$\begin{cases} x = r(\theta - \sin \theta) \\ y = r(1 - \cos \theta) \end{cases} \quad (7.1)$$

で与えられます。サイクロイドを描くためにはこの式で計算した点を結ぶ直線で近似することにします。これらの座標をそのまま SVG の中に直接記述するのは面倒なので、JavaScript のプログラムで計算してその結果を `<path>` 要素の属性 `d` に設定することにします。

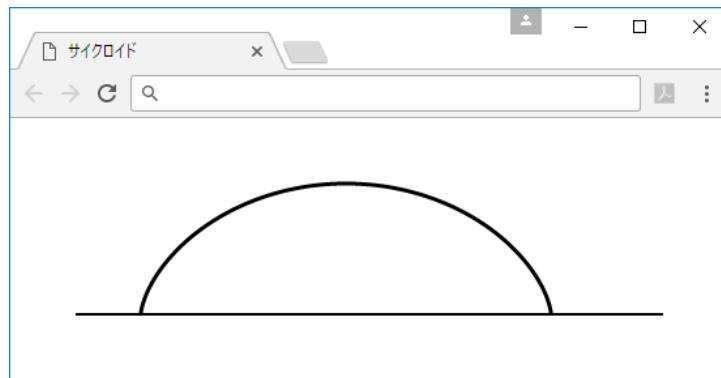


図 7.16: サイクロイド

SVG リスト 7.12: サイクロイドを描く

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <svg xmlns="http://www.w3.org/2000/svg"
3     height="100%" width="100%">
4 <title>サイクロイド</title>
5 <script type="text/ecmascript">
6 //<![CDATA[
7 var R = 50;
8 window.onload = function() {
9     var Angle, rad, pX, pY, d ="M";
10    for( Angle=0; Angle&lt;= 360; Angle++) {
11        rad = Angle/180*Math.PI;
12        pX = R*(rad-Math.sin(rad));
13        pY = R*(-1+Math.cos(rad));
14        d += pX+","+pY+" ";
15    }
16    document.getElementById("cycliod").setAttribute("d",d);
17 }
18 //]]&gt;
19 &lt;/script&gt;
20 &lt;g transform="translate(100,150)"&gt;
21   &lt;line x1="-50" y1="0" x2="400" y2="0" stroke-width="2" stroke="black"/&gt;
22   &lt;path id="cycliod" fill="none" stroke="black" stroke-width="3"/&gt;
23 &lt;/g&gt;
24 &lt;/svg&gt;</pre>
```

- 描く図形は円が転がっていく直線 (21 行目) とサイクロイドの図形 (22 行目) です。
- 転がる直線の y 座標は 0 になっています。
- 7 行目で回転する円の半径を保持する変数の値を設定します。
- 8 行目から SVG 文書がロード終了後に呼び出される関数を定義しています。

- 9行目で必要な変数の宣言をしています。サイクロイドの道のりのデータを格納する変数 d は "M" で初期化しています。
- 10行目から15行目でサイクロイドの点の座標を計算します。ここでは 0° から 1° 単位で 360° まで計算します。角度の単位がラジアンでないのはこのあのリスト 7.17 で回転する円を表示するアニメーションが付いた図形を SVG の rotate を用いて描くためです。
- 11行目で角度の単位をラジアンに直します。JavaScript では円周率の値を Math.PI で利用できます。
- 12行目と13行目で式 (7.1) に基づいて点の位置を計算しています。正弦関数 ($\sin x$) と余弦関数 ($\cos x$) は JavaScript ではそれぞれ Math.sin(x) と Math.cos(x) で利用できます。
なお、ここでは SVG の座標系の関係から y 座標の値は符号を逆にしています。
- 14行目で今までに求めた道のりのデータの後に新しく得られた点の座標を付け加えています。
- 16行目で道のりのデータを書き直しています。

問題 7.11 リスト 7.12 を参考にして正 6 角形を描く SVG ファイルを作成しなさい。

7.4.2 JavaScript のオブジェクトとJSON

JavaScript のオブジェクト

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数による添え字を使います。これに対し、オブジェクトはキーと値のペアの集まりです。

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。ここで > で始まる行はデベロッパーソールにおける入力行を表します。

```
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。
- このようなおぶじぇくをの表記をオブジェクトリテラルと呼びます。
- 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。

JSON

JSON(JavaScript Object Notation) とはオブジェクトリテラル形式で書かれた文字列です。最近では XML に代わるデータ交換のフォーマットとして利用されています。JavaScript のオブジェクトを JSON 形式に直すためには `JSON.stringify` メソッドを利用します。なお、JSON ではオブジェクトの値が通常のデータだけしか取り扱えませんのでオブジェクトの値が関数（そのオブジェクトのメソッド）の場合は変換されません。また、JSON 形式の文字列を JavaScript のオブジェクトに変換するためには `JSON.parse` メソッドが利用できます。次の例を見てください。

```
1 >A={x:10,y:20,add:function(){return(this.x+this.y);}}
2 Object {x: 10, y: 20}
3 >A.add()
4 30
5 >B=JSON.stringify(A);
6 {"x":10,"y":20}
7 >typeof B;
8 "string"
9 >JSON.parse(B);
10 ▼ Object
11     x: 10
12     y: 20
13     ►__proto__: Object
```

- 1 行目でオブジェクトを定義しています。メソッドとして `add()` があり、2 つのメンバー `x` と `y` を加えた値を返します（3 行目）
- 5 行目でオブジェクトを JSON 形式に変換しています。この中にはメソッドの `add` が含まれていません。
- 7 行目では変換されたものが文字列であることを確認しています。
- 9 行目では変換後の文字列をオブジェクトに変換しています。

7.4.3 SVG のオブジェクトを操作するための関数

今後は SVG の要素を新規に作成したり、すでに存在する要素の属性をまとめて設定しなおす必要があるのでそれらを処理する関数を作成しておくことにします。

このような関数（ヘルパー関数）を提供するライブラリーとしては `jQuery.js` が有名です。`jQuery.js` はブラウザの対応の違いも吸収してくれる有用なライブラリですが、DOM の操作に慣れるという観点から独自の関数群を用意することにします⁶。

リスト 7.13 は DOM の要素を新規に作成したり属性を設定する関数群のファイルのリストです。今後のプログラムではこのファイルを `make-svgelm.js` という名前でこれから作成する SVG 文書と同じフォルダに保存します。

この関数群は次のものからなります。

- 新規に HTML 要素を作成する関数 `MKHTMLElm`
- 新規に SVG 要素を作成する関数 `MKSVGElm`
- すでに存在する DOM 要素の属性をいくつかまとめて設定する `SetAttributes` 関数
- すでに存在する DOM 要素にイベントをいくつかまとめて設定する `AddEvents` 関数
- すでに存在する DOM 要素からイベントをいくつかまとめて取り除く `RemoveEvents` 関数

JavaScript リスト 7.13: DOM 要素を新規作成し、属性を設定する関数群 (`make-svg-elm.js`)

```

1 function MKHTMLElm(P, Elm, Attribs, Events) {
2     var HTMLNS = "http://www.w3.org/1999/xhtml";
3     return MakeElement(HTMLNS, P, Elm, Attribs, Events)
4 }
5 function MKSVGElm(P, Elm, Attribs, Events) {
6     var SVGNS = "http://www.w3.org/2000/svg";
7     return MakeElement(SVGNS, P, Elm, Attribs, Events)
8 }
9 function MakeElement(NS, P, elem, attribs, events) {
10    var Element = document.createElementNS(NS, elem);
11    SetAttributes(Element, attribs);
12    AddEvents(Element, events);
13    if(P) P.appendChild(Element);
14    return Element;
15 }
16 function SetAttributes(Elm, attribs) {
17     for( attrib in attribs) {
18         Elm.setAttribute(attrib,attribs[attrib]);
19     }
20 }
21 function AddEvents(Elm, Events) {
22     for( event in Events) {
23         Elm.addEventListener(event,Events[event][0], Events[event][1]);
24     }
25 }
```

⁶この後で SVG 要素を HTML 文書内に直接生成するときに、jQuery では対応できない場合があります。

```

26  function RemoveEvents(Elm, Events) {
27    for( event in Events) {
28      Elm.removeEventListener(event,Events[event][0], Events[event][1]);
29    }
30  }

```

- MKHTMLElm は新規に HTML 要素を作成する関数です。

```

1  function MKHTMLElm(P, Elm, Attrbs, Events) {
2    var HTMLNS = "http://www.w3.org/1999/xhtml";
3    return MakeElement(HTMLNS, P, Elm, Attrbs, Events)
4  }

```

引数は次の通りです。

- P 作成する要素の親要素。null のときは親要素がないことを示します。
- Elm 作成する要素名
- attrs 作成する要素の属性名と属性値のペアのオブジェクトです。
- events 作成する要素に付けるイベント名をキーに、値をイベント処理関数名、イベントバーピングかどうかの論理値が並んだ配列であるオブジェクトです。

この関数は与えられた引数の前に名前空間を指定して関数 MakeElement を呼び出してその戻り値を返しているだけです（3行目）。名前空間の値は直接、引数に書くことも可能ですが、ここではローカルな変数に定義して（2行目）、その値を渡しています。なお、HTML5 の名前空間については次のサイトを見てください。

<http://www.w3.org/TR/2011/WD-html5-20110525/namespaces.html#namespaces>

- MKSVGElm は新規に SVG 要素を作成する関数です。引数の意味や動作は関数 MKHTMLElm と同じです。

```

5  function MKSVGElm(P, Elm, Attrbs, Events) {
6    var SVGNS = "http://www.w3.org/2000/svg";
7    return MakeElement(SVGNS, P, Elm, Attrbs, Events)
8  }

```

- MKSVGElm は新規に DOM 要素を作成する関数です。

```

9  function MakeElement(NS, P, elem, attrs, events) {
10    var Element = document.createElementNS(NS,elem);
11    SetAttributes(Element, attrs);
12    AddEvents(Element, events);
13    if(P) P.appendChild(Element);
14    return Element;
15  }

```

- 与えられた名前空間を用いて createElementNS メソッドでオブジェクトを作成します (10 行目)。
 - 既存の要素に属性をまとめて設定する関数 SetAttributes を呼び出します (11 行目)。
 - 既存の要素にイベントをまとめて設定する関数 AddEvents を呼び出します (12 行目)。
 - 最後に、親要素が null でないときにはこの要素を子要素として付け加えます (13 行目)。
- SetAttributes はすでに存在する DOM 要素の属性をいくつかまとめて設定する関数です。属性とその属性値が組になったオブジェクトを利用して (17 行目) 指定された要素に属性値を設定します。 (18 行目)。

```

16 function SetAttributes(Elm, attrs) {
17   for( attrib in attrs) {
18     Elm.setAttribute(attrib, attrs[attrib]);
19   }
20 }
```

- SetEvents はすでに存在する DOM 要素のイベントをまとめて設定する関数です。

```

21 function AddEvents(Elm, Events) {
22   for( event in Events) {
23     Elm.addEventListener(event, Events[event][0], Events[event][1]);
24   }
25 }
```

イベント名とイベント処理関数名、イベントバーピングかどうかの論理値が並んだ配列のオブジェクトを利用して設定します (23 行目)。

- RemoveAttributes は DOM 要素のイベント処理をまとめて解除する関数です。 SetEvents と同様の配列を基にして与えられた要素からイベント処理関数を取り除きます。

```

26 function RemoveEvents(Elm, Events) {
27   for( event in Events) {
28     Elm.removeEventListener(event, Events[event][0], Events[event][1]);
29   }
30 }
```

7.4.4 ツェルナーの錯視図形

図 7.17 は古典的なツェルナーの錯視図形です [33, 58 ページ]。横にある 2 本の直線は平行なのですが斜線があるために平行に見えないというものです。

前の節の関数を用いて図 7.17 を表示する SVG 文書を作成します。



図 7.17: ツェルナーの錯視图形

SVG リスト 7.14: ツェルナーの錯視

```

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>ツェルナーの錯視图形</title>
6  <script type="text/ecmascript" xlink:href="./make-svg-elm.js" />
7  <script type="text/ecmascript">
8  //  <![CDATA[
9  window.onload = function init() {
10     var W = 1, WCol = "black";
11     var sW = 1, sWCol = "black", sL = 50/2, Ang=30, Step = 20;
12     var X1 = 50, Y1=50, X2 = 300, Y2 = 100;
13     var G1, G2, Tmp, i;
14     var Canvas = document.getElementById("canvas");
15     MKSVGElm(Canvas, "line",
16         {"x1": X1, "y1": Y1, "x2": X2, "y2": Y1,
17          "stroke-width": W, "stroke": WCol}, {});
18     MKSVGElm(Canvas, "line",
19         {"x1": X1, "y1": Y2, "x2": X2, "y2": Y2,
20          "stroke-width": W, "stroke": WCol}, {});
21     G1 = MKSVGElm(null, "g", {}, {});
22     G2 = MKSVGElm(G1, "g", {}, {});
23     MKSVGElm(G2, "line",
24         {"x1": -sL, "y1": 0, "x2": sL, "y2": 0,
25          "stroke-width": sW, "stroke": sWCol}, {});
26     SetAttributes(G2, {"transform": "rotate("+Ang+")"});
27     for(i=X1+10; i<X2-10; i+= Step) {
28         Tmp = G1.cloneNode(true);
29         SetAttributes(Tmp, {"transform": "translate("+i+", " + Y1 +" )"});
30         Canvas.appendChild(Tmp);
31     }
32     SetAttributes(G2, {"transform": "rotate("+(-Ang)+")"});
33     for(i=X1+10; i<X2-10; i+= Step) {
34         Tmp = G1.cloneNode(true);
35         SetAttributes(Tmp, {"transform": "translate("+i+", " + Y2 +" )"});
36         Canvas.appendChild(Tmp);

```

```

37      }
38  }
39 //  ]]></script>
40 <g id="canvas"/>
41 </svg>
```

- HTML 文書で外部の JavaScript ファイルを読み込むとき、ファイル名は `src` 属性で指定しますが、SVG 文書では `xlink:href` 属性を用います(6 行目)。リスト 7.13 を `make-svg-elm.js` として保存し、このファイルと同じフォルダーにおいていたことを思い出してください。
- ツエルナーの錯視の図の大きさや線の間隔などを修正しやすくするためにこれらの値を変数として定義しています(10 行目から 12 行目)。ここで変数の意味は次のとおりです。

<code>W:</code>	水平線の線幅	<code>Y1:</code>	上の水平線の縦位置	<code>SL:</code>	補助線の長さの半分
<code>WCol:</code>	水平線の色	<code>Y2:</code>	下の水平線の縦位置	<code>Ang:</code>	補助線が交わる角度
<code>X1:</code>	水平線の左位置	<code>sW:</code>	補助線の線幅	<code>Step:</code>	補助線の間隔
<code>X2:</code>	水平線の右位置	<code>sWCol:</code>	補助線の色		

- まず、40 行目で定義されている図形を保持するオブジェクトの位置を得ます(14 行目)。
- 15 行目から 17 行目と 18 行目から 20 行目で上下の水平線を図形を保持するオブジェクトの子ノードとしてセットします。
- 補助線は位置と傾きを決めるので別々の `<g>` 要素でそれぞれセットすることとします。オブジェクトが複雑になるので雛形を作ることにします(21 行目から 26 行目)。
 - G1 に `<g>` 要素を作成し(21 行目)、その子ノードとして G2 で作成した `<g>` 要素を設定します(22 行目)。
 - G2 の子ノードに補助線を設定します(23 行目と 25 行目)。引数 `[]` は空の配列です。
- G2 に上の補助線を傾けるための属性値を設定します(25 行目)。これをコピーした要素に別の角度を後でつけるために傾ける角度は指定していません。
- 27 行目から 31 行目が上の直線に斜線を付ける部分です。
 - G1 に作成した要素のコピーを作成します(28 行目)。
 - この要素を平行移動するために `transform` を設定します(29 行目)。
 - 表示するために一番外側の要素の子要素に設定します(30 行目)。
- 下の部分の補助線も同様に設定します(32 行目から 37 行目)。
- 32 行目で `(-Ang)` という記述がありますが、この部分は数値として計算されていることに注意してください。

問題 7.12 Chrome の デベロッパーツールを用いて解説のとおり SVG の DOM ができるかどうかを確認しなさい。

問題 7.13 図 7.17 の補助線に対して回転のアニメーションを付けなさい。⁷

⁷ このために上記のリストで補助線を個別の要素としました。

問題 7.14 リスト 7.10 をこの関数群を用いて書き直しなさい。

問題 7.15 図 7.18 はネックレスの糸とよばれる錯視图形です ([38, 60 ページ図 6.6 右] 参照)。この図を作成しなさい。

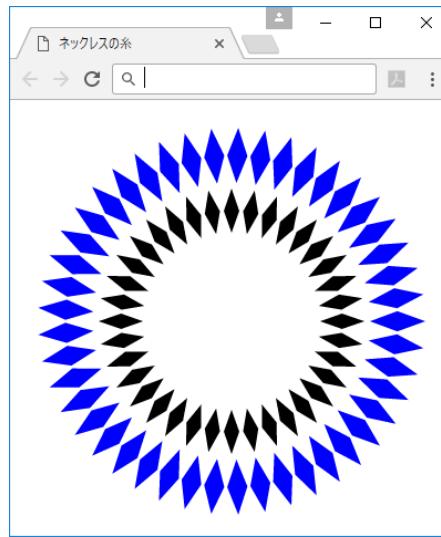


図 7.18: ネックレスの糸

次の図形はバインジオ・ピンナの錯視图形 ([38, カラー図版 9] 参照) とよばれる錯視图形です。曲がった輪郭線の間にうっすらと色がついて見えます。⁸

SVG リスト 7.15: バインジオ・ピンナの錯視

```

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>バインジオ・ピンナの錯視图形</title>
6  <script type="text/ecmascript" xlink:href=".//make-svg-elm.js" />
7  <script type="text/ecmascript">
8  <![CDATA[
9  var Canvas;
10 window.onload = function(){
11     DrawFigs("red", "green", "Canvas1");
12     DrawFigs("#C00", "#004", "Canvas2");
13 }
14 function DrawFigs(Color1, Color2, Place) {
15     var W1=8, W2=4;
16     Canvas = document.getElementById(Place);
17     DrawFigure(150, 30, W1, W2, Color1);
18     DrawFigure(144, 30, W1, W2, Color2);

```

⁸これと同じ現象は直線群の間隔が狭い場合でも起こります。身近な例は方眼紙です。工学の測定値を表すための対数方眼紙では間隔が狭いところでは色がつき、間隔が広いところでは白く見えます。

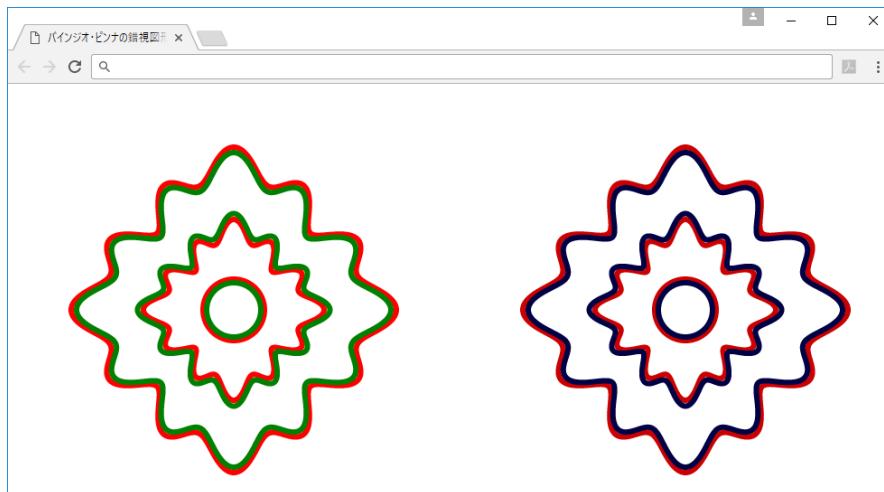


図 7.19: バインジオ・ピンナの錯視図形

```

19   DrawFigure(80, 20, W1, W2, Color1);
20   DrawFigure(86.5, 20, W1, W2, Color2);
21   DrawFigure(14, 20, 0, 0, Color1);
22   DrawFigure(10, 20, 0, 0, Color2);
23 }
24 function DrawFigure(R, sR, W, W2, Color) {
25   var d = "M", i, Ang, R0;
26   for(i=0;i<720;i++) {
27     Ang= Math.PI*i/180/2;
28     R0=R+sR*(Math.cos(W*Ang)*Math.cos(W2*Ang));
29     d += R0*Math.cos(Ang) + ","+R0*Math.sin(Ang)+" ";
30   }
31   d += "z";
32   return MKSVGElm(Canvas, "path",
33     {"d": d, "stroke-width": 6, "stroke": Color, "fill": "none"}, {});
34 }
35 ]]></script>
36 <g id="Canvas1" transform="translate(250,250)">
37 <g id="Canvas2" transform="translate(750,250)">
38 </svg>

```

- この SVG 文書では左右の図をそれぞれ描くための関数を呼び出しています (11 行目と 12 行目の `DrawFigs` 関数)。この関数は `<g>` 要素の `id` を指定し外側と内側の色を指定します。
- この `DrawFigs` 関数はそれぞれの曲線を描く関数 (`DrawFigure`) をパラメータを変えて 6 回呼び出します (17 行目から 22 行目)。
- 24 行目から 34 行目で定義されている関数 `DrawFigure` は短い直線をつなげて曲線を描きます。
- その位置は θ の方向で次の式で計算されます (28 行目)。

$$R_0 = R + R_1 \cos(W_1\theta) \cos(W_2\theta)$$

- R は基準の半径で、 R_1 はそれに変化させる幅です。 R_1 が 0 であれば（計算が無駄ですが）半径 R の円となります。
- ここで計算させた位置を文字列としてつないで<path> 要素の属性 d に設定して戻ります（29 行目）。

7.4.5 一定時間経過後に関数を呼びだす（自分でアニメーションを作成する）

SVG ではオブジェクトの属性にいろいろなアニメーションがつけられることはすでに見てきました。しかしながら次のようなことはできなかつたり制限があつたりします。

- <path> 要素のデータの並びがまったく同じ場合でない d にアニメーションをつける。
- アニメーションを途中で中断して、そこから新しいアニメーションを続けて始めること

次の例は赤い円が初めに表示されています。何もしないと画像が表示されてから 10 秒後に円の色が青に色が変わります。すでに述べたアニメーションを使えば簡単に実現できますがここでは指定した時間後に指定した関数を実行するようにしています。この方法を繰り返して利用して少しずつ図を変化させればアニメーションができることがあります。

この例では青に変わる前に円の部分をクリックすると色は緑に変わり、その後、10 秒経過しても青にはなりません。また、青に変わってからクリックしても緑にはなりません。

SVG リスト 7.16: 10 秒後に色が変わります

```

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"
4      height="100%" width="100%">
5      <title>10 秒後に色が変わります</title>
6      <script type="text/ecmascript">
7      //      <![CDATA[
8          var timeOut, Circle;
9          window.onload = function() {
10             Circle = document.getElementById("CircleRed");
11             Circle.addEventListener("click",resetColor,false);
12             timeOut = window.setTimeout(changeColor,10000);
13         }
14         function changeColor() {
15             Circle.setAttribute("fill","blue");
16             Circle.removeEventListener("click",resetColor,false);
17         }
18         function resetColor() {
19             Circle.removeEventListener("click",resetColor,false);
20             Circle.setAttribute("fill","green");
21             window.clearTimeout(timeOut);
22         }
23     //  ]]></script>
24     <circle id="CircleRed" cx="50" cy="50" r="20" fill="red"/>
25 </svg>
```

- SVG のファイルがロードされたイベントを処理するために9 行目から13 行目で処理関数を定義しています。
 - 10 行目で変数 Circle に<circle> 要素への参照を代入しています。
 - 11 行目でこの円上でのマウスのクリックを処理する関数を割り当てています。
 - 12 行目で、指定した経過時間後に指定した関数 changeColor を呼び出すことを設定しています。⁹ この関数は円の色を青にします。
 - これを設定するメソッドは window の setTimeout() です。第 1 の引数が起動する関数です。この関数に引数が必要な場合には 3 番目以降の引数で与えます。2 番目の引数が実行までに要する経過時間で、単位は ms です。ここでは $10000ms = 10$ 秒に設定しています。
 - このメソッドの戻り値は設定した setTimeout() をキャンセルするときの引数に用います。キャンセルするための関数は clearTimeout です。この関数は21 行目に現れています。
 - したがって、円をクリックすると関数 resetColor が呼び出されて色が green に変わり、10 秒後には blue に変わるように設定されたことになります。
- 関数 resetColor では、色を blue に変え (15 行目)、円上でのクリックを無効にするため、イベント処理関数を取り除いています (16 行目)。したがって、色が変わった後ではクリックしても色は green に変わりません。
- 18 行目から22 行目で円がクリックされたときにより出される関数を定義しています。
- 円上でのクリックの処理を止め (19 行目)、色を green に変え (20 行目)、経過時間に実行を予約してあった関数の実行をキャンセルしています (21 行目)。

問題 7.16 5 秒後、10 秒後に円の色が順に変化するアニメーションを作成しなさい。

この方法を使うとリスト 7.12 のサイクロイドを回転する円とその円周上の定点をアニメーションで表示しながらサイクロイドを描く SVG 文書が作成できます。

SVG リスト 7.17: サイクロイドを描く --- アニメーション版

```

1  <?xml version="1.0" encoding="UTF-8" ?>
2  <svg xmlns="http://www.w3.org/2000/svg"
3      height="100%" width="100%">
4  <title>サイクロイドを描く --- アニメーション版</title>
5  <script type="text/ecmascript">
6  //<![CDATA[
7  var R = 50, Current = 1, Step=2, d ="M0,0 ";
8  var T, Rot, C;
9  window.onload = function() {
10    C =document.getElementById("cycliod");
11    T =document.getElementById("translate");
</pre>

```

⁹何秒後かに別のページに飛ぶようになっているホームページではこのメソッドを利用して実現しています。

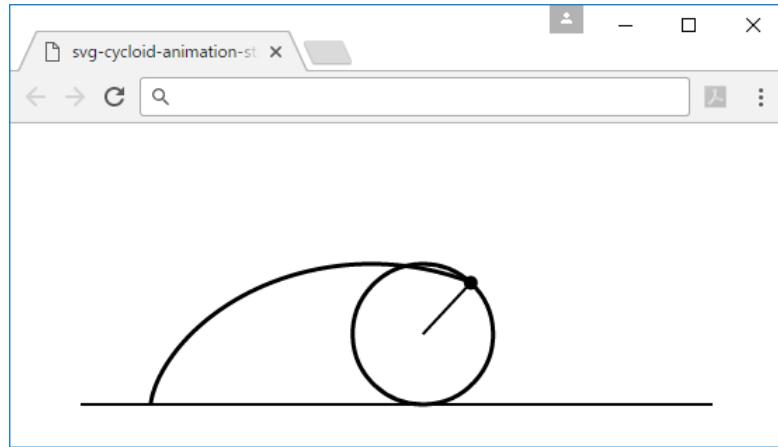


図 7.20: サイクロイドを描く — アニメーション版

```

12     Rot =document.getElementById("rotate");
13     drawCurve();
14 }
15 function drawCurve(){
16     var Next = Current + Step, rad;
17     if(Current<=360) {
18         for( ; Current< Next; Current++) {
19             rad = Current/180*Math.PI;
20             pX = R*(rad-Math.sin(rad));
21             pY = R*(-1+Math.cos(rad));
22             d += pX+", "+pY+" ";
23         }
24         C.setAttribute("d",d);
25         T.setAttribute("transform","translate(" + (R*rad) + ",-50)");
26         Rot.setAttribute("transform", "rotate("+Next+")");
27         setTimeout(drawCurve,100);
28     }
29 }
30 //]]>
31 </script>
32 <g transform="translate(100,150)">
33     <line x1="-50" y1="0" x2="400" y2="0" stroke-width="2" stroke="black"/>
34     <path id="cycloid" fill="none" stroke="black" stroke-width="3"/>
35     <g id="translate" transform="translate(0,-50)">
36         <g id="rotate">
37             <circle cx="0" cy="0" r="50" stroke-width="3" stroke="black" fill="none"/>
38             <line x1="0" y1="0" x2="0" y2="50" stroke-width="2" stroke="black" />
39             <circle cx="0" cy="50" r="5" fill="black"/>
40         </g>
41     </g>
42 </g>
43 </svg>
```

- 円周上の点が回転して描く軌跡を明示するために、円周上の点と、その点までの半径を表示しています(35行目と40行目)。
- この図形に回転と平行移動を付けるために10行目と12行目で大域変数にオブジェクトを代入しています。
- サイクロイドの図形を一定間隔の時間で描くためには現在どこまで書いたのかを記憶しておく変数が必要です。それらの変数の初期化を含めて7行目で定義しています。
- 関数 drawCurve は曲線の一部を描く関数です。
- 16行目で次の描く範囲までの位置を求め、その範囲の曲線の一部を付け足しています。描く範囲を除けば、式はリスト 7.12 と同じものです。
- 25行目で回転した図形を平行移動させています。
- 26行目では図形を回転させています。

問題 7.17 リスト 7.17 におけるアニメーションの開始時期を画面上でクリックしたときに変更したものを作成しなさい。

図 7.21 は表示されている色名の文字列と表示色が異なるため、表示色を読むときに脳が混乱するというものです。



図 7.21: 文字の表示色と文字名が異なる

このリストのかわりに表示される文字列は一時期にはひとつだけ表示されるように改造したプログラムのリストで説明します。

SVGリスト7.18: 文字の表示色と文字名が異なる---ノートレ版

```

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>文字の表示色と文字名が異なる</title>
6  <script type="text/ecmascript" xlink:href="make-svg-elm.js"></script>
7  <style type="text/css">
8      .textstyle { font-size:30px; text-anchor:middle; }
9  </style>
10 <script type="text/ecmascript">
11 //<![CDATA[
12 var Data = [
13     ["あか","赤","red"],     ["くろ","黒","black"],
14     ["みどり","緑","green"], ["あお","青","blue"],
15     ["きいろ","黄","yellow"], ["むらさき","紫","violet"]
16 ];
17 var Type = 0;
18 var xUnit = 150, xInit = 30, xMax = 6;
19 var yUnit = 50, yInit = 20, yMax = 8;
20 var Max = 10, k, Doc, textNode;
21 var PosList = [];
22 window.onload = function() {
23     var i, j;
24     for(i=0; i&lt;xMax; i++) {
25         for(j=0; j&lt;yMax; j++) {
26             PosList.push([i,j]);
27         }
28     }
29     k = Max;
30     Doc = document.getElementById("Doc");
31     ShowProb();
32 }
33 function ShowProb() {
34     var x, y, i, j, p;
35     var T;
36     if(k&gt;0) {
37         i = getRand(0, Data.length);
38         j = getRand(0, Data.length);
39         p = getRand(0, PosList.length);
40         T = PosList.splice(p,1)[0];
41         x = xInit+xUnit * T[0];
42         y = yInit + yUnit * T[1];
43         if( k == Max ) {
44             textNode = MKSVGElm(Doc,"text",
45                 {"x": x, "y": y,"class": "textstyle","fill":Data[i][2]},{} );
46             textNode.appendChild(document.createTextNode(Data[j][Type]));
47         } else {
48             SetAttributes(textNode,{"x": x, "y": y,"fill":Data[i][2]});
49             textNode.replaceChild(document.createTextNode(Data[j][Type]),
50                                 textNode.firstChild);
51         }
52         k--;
53         window.setTimeout(ShowProb,1000);
}
</pre>

```

```

54      }
55  }
56  function getRand(min, max) {
57    return Math.floor(min+(max-min)*Math.random());
58  }
59 //]]>
60 </script>
61 <g transform="translate(40,40)" id="Doc"/>
62 </svg>
```

- 12 行目から16 行目で出題する文字列とその色を配列として定義します。問題は漢字でもひらがなでも出題できるように配列として定義しています。
- 変数 Type は出題する文字列を漢字、ひらがな、英語のどれにするかをきめる値を保持します。初期値はひらがな (0) です。
- 文字を画面にランダムに配置するため画面を区域に区切ってその中にひとつだけ表示するようにします。18 行目では横方向の大きさと領域の数を定義しています。縦方向は19 行目で定義しています。
- 20 行目では大域変数を定義しています。
- 21 行目では同じ位置に問題文が 2 つ表示されないようにするための情報をしまう配列を用意しています。初期化は関数 init() で行われます。
- 22 行目と32 行目は onload イベントで呼び出される関数を定義しています。

```

22 window.onload = function() {
23   var i, j;
24   for(i=0; i<xMax; i++) {
25     for(j=0; j<yMax; j++) {
26       PosList.push([i,j]);
27     }
28   }
29   k = Max;
30   Doc = document.getElementById("Doc");
31   ShowProb();
32 }
```

- 24 行目と28 行目ではどの位置にまだ問題文を配置していないかのリストを保持する配列 PosList を初期化しています。この値は縦と横の位置を配列の値とする配列になります。
- 配列の最後に新しい要素を追加する手段として push() メソッドを用いています (26 行目)。
- 29 行目では表示する問題数を管理する変数 k を初期化しています。
- 30 行目では表示画面である<g> 要素を変数 Doc に格納しています。

- これで準備が整ったので問題を作成して表示する関数 ShowProb を呼び出します (31 行目)。
- 33 行目と 55 行目は問題を作成して表示する関数です。一定の数を出題していない場合にはもう一度この関数が呼び出されます。
 - 関数 getRand は与えられた二つの引数の間 (最小値は含み、最大値は含まない) の整数の乱数を与える関数です。
 - 変数 *i* と変数 *j* はそれぞれ問題文の色と表示の位置を与えます。これらの値は問題の種類を保持している変数 Data の配列の添え字として用いられます (37 行目と 38 行目)。
 - 表示する位置は配列 Postlist からランダムに選びます (39 行目)。
 - 選ばれた位置は配列 Postlist から取り除く必要があります。これを実現するために配列のメソッド splice を用います。splice は引数の数によりいろいろなパターンが生じますが、ここでは配列 Postlist の与えられた位置 (変数 *p*) からひとつ要素を取り除き (splice の 2 番目の引数が 1) ます。除かれた要素がこのメソッドの戻り値なのでこれを変数 *T* にしまいます。
 - この変数の一番目の要素が横方向に位置、2 番目の要素が縦方向の要素の位置なのでこれから問題の文字列を表示する位置を計算します (41 行目と 42 行目)。
 - 変数 *k* の値が変数 Max と同じであれば問題文を表示するためのオブジェクトがまだ生成されていないので、問題文を表示するためのオブジェクトを生成します。
 - 44 行目と 45 行目では <text> 要素を生成しています。属性 fill には乱数で選べた色 (Data[i][2]) を設定しています。
 - 表示する文字列はこの要素のテキストノードとして設定します (46 行目)。テキストノードを作成 (createTextNode) し、それを 44 行目と 45 行目で作成した <text> 要素の子要素として登録 (appendChild) します。
 - 2 回目以降にこの関数が呼び出されている場合は 44 行目と 45 行目で作成した <text> 要素の属性値を変えます (48 行目)。
 - 表示するテキストはテキストノードを作成してから、元のノードを置き換えます (replaceChild)。
 - 変数 *k* の値をひとつ減らします (52 行目)。
 - 次の問題文を表示するために 1 秒待機します (53 行目)。このために setTimeout をメソッド用います。
- 56 行目と 58 行目では与えられた範囲での乱数を生成する関数を定義しています。
 - 0 から 1 の間の乱数の生成には関数 Math.random() を用います。
 - この値が 0 から 1 なので最小値を初めの引数に、最大値を 2 番目の引数にするために

$$(\min+(\max-\min)*\mathbf{Math.random}())$$
 で計算しています。

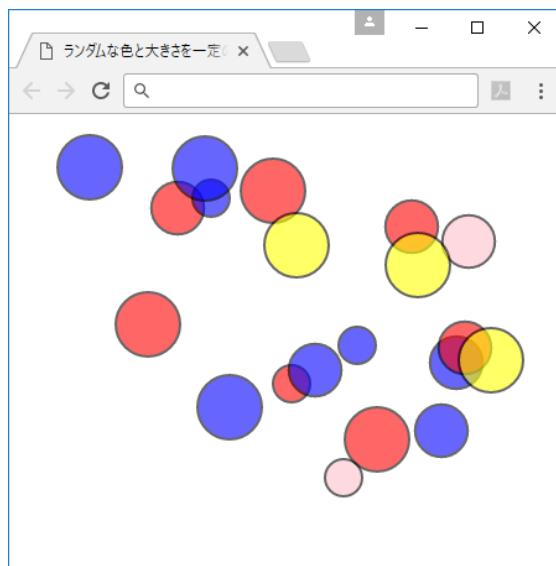


図 7.22: 色と大きさと位置をランダム決めた円を表示する

- 与えられた実数を整数にするために切り捨ての関数 `Math.floor()` を用います。

問題 7.18 図 7.22 は円の色、大きさ、位置をランダムに決めた円を表示するものです。

このプログラムは次のものが変化しています。

- 円の色 — 5 色のうちから選択
- 大きさ — 面積比で 1 から 5 まで
- 表示間隔 — `setTimeout()` の間隔を変えています。

このような図形を作成しなさい。

7.5 HTML 文書とその中にある SVG 要素の間でデータ交換する

HTML 文書は XML 文書としての類似性があり、DOM で操作することが可能です。この節では HTML 文書も今までに学んできた SVG 文書のように DOM の操作が可能であることを示します。

図 7.23 は左の部分にある SVG を用いて表示された画面上をクリックするとその位置を HTML 文書の部分に表示し、またその逆にテキストボックスなどで指定した値を SVG の画像に反映できることを示すものです。

- この図は右側のメニューをクリックした状態です。
- 左側の SVG 上の部分でクリックするとその場所に円の中心が移動し、クリックした位置の座標を SVG の画像の中だけではなく右の HTML の要素であるテキストボックスにその位置を表示します。

- テキストボックスに値を入れたり、メニューの中から色を選んだ後、下の設定ボタンを押すと、指定された位置に円が移動します。また、指定した色に変わります。

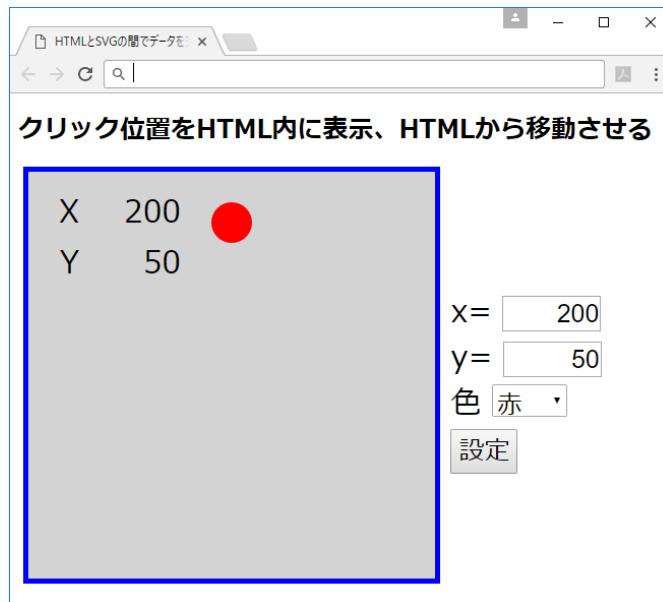


図 7.23: クリック位置を HTML で表示し、HTML のデータから SVG の図形を動かす

HTML リスト 7.19: クリック位置を HTML で表示し、HTML のデータから SVG の図形を動かす

```

1  <!DOCTYPE html>
2  <html xmlns:svg="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink">
4  <head>
5      <meta charset="UTF-8"/>
6      <script type="text/ecmascript" src="SSClickPos.js"></script>
7      <link rel="stylesheet" type="text/css" href="HTML.css">
8  <title>HTML と SVG の間でデータを交換させる</title>
9  </head>
10 <body>
11     <h1 class="display">クリック位置を HTML 内に表示、HTML から移動させる</h1>
12     <div class="Cell">
13         <svg height="410" width="410" id="canvas">
14             <g transform="translate(5,5)">
15                 <g id="field">
16                     <rect x="0" y="0" width="400" height="400" fill="lightgray"/>
17                     <circle id="Circle" cx="200" cy="50" r="20" fill="red"/>
18                     <text class="textStyle" x="50" y="50"> X</text>
19                     <text class="textStyle" id="X" x="150" y="50"> X</text>
20                     <text class="textStyle" x="50" y="100"> Y</text>
21                     <text class="textStyle" id="Y" x="150" y="100"> Y</text>
22             </g>
23             <path fill="blue" d="M-5,-5 405,-5 405,405 -5,405z M0,0 0,400 400,400 400,0z"/>

```

```
24      </g>
25    </svg>
26  </div>
27  <div class="Cell" >
28    <div><label for="XP">x=</label>
29      <input type="text" id="XP" size="3" /></div>
30    <div><label for="YP">y=</label>
31      <input type="text" id="YP" size="3" /></div>
32    <div><label for="SelectColor">色</label>
33      <select id="SelectColor"></select></div>
34    <input id="SetColor" type="button" value="設定"></input>
35  </div>
36 </body>
37 </html>
```

いくつか注意する点を述べておきます。

- 1 行目が HTML5 における DOCTYPE 宣言 です。以前のものに比べ記述が格段に簡単になっています。
- HTML 文書の<htaml> 要素 は HTML 文書のルート要素です。その属性として `svg` と `xlink` の名前空間を使用することを宣言しています (2 行目と 3 行目)。
- HTML では HTML 文書のいろいろな情報は<head> 要素内の<meta> 要素で記述します。ここでは 5 行目でこの文書の文字コードが UTF-8 であることを宣言しています。
 - 6 行目では外部の JavaScript ファイル (`SSClickPos.js`) を読み込みます。
 - 7 行目では外部の CSS ファイル (`HTML.css`) を読み込みます。
- 10 行目以降の部分に HTML 文書の本体が記述されています。
- 11 行目の<h1> 要素は最上部の表題を記述しています。
- 全体は<div> 要素が 2 つ並んだ形になっています。これらには `class` 属性が定義され、ともに `Cell` となっています。
- 12 行目から 26 行目は SVG の画像を表示しています。
 - SVG に関する要素名が直接記述されています。HTML5 では HTML 要素のように SVG の要素が記述できます (インライン SVG)。
 - 記述の内容はリスト 7.5 を利用しています。
 - 15 行目の<g> 要素は以前のものにはありません。後で見るようにこの要素に `click` イベントの処理関数を付けます。
 - この中に長方形を置いて、`click` イベントが拾えるようにしています。
 - また、23 行目で外枠をつけています。この部分は 15 行目と 22 行目の外にあるので、この上のクリックイベントは拾えません。

- 画面右のテキストボックスやプルダウンメニューに関するCSSはidを用いています(この場合のセレクタはidの前に#を付けたものになります)。

この読み込まれるCSSファイルは次のようにになっています。

CSSリスト7.20: 基本的なCSSファイル

```

1  .display {
2      font-size:25px;
3  }
4  .textStyle {
5      font-size:30px;
6      text-anchor:end;
7  }
8  .Cell {
9      font-size:30px;
10     display:inline-block;
11     vertical-align:middle;
12     padding-left:5px;
13 }
14 #XP, #YP{
15     font-size:25px;
16     text-align: right;
17 }
18 #SetColor, #SelectColor {
19     font-size:25px;
20     text-align:center;
21 }
```

- <h1>要素にはclass属性があり、その値がdisplayとなっています。これに対応するCSSのセレクタはclass属性の値の前に.(ピリオド)をつけたものになります。

CSSファイルの1行目と3行目がその部分になります。ここではフォントの大きさ(font-size)を指定しています。この値には単位が必要です。

- SVG内の<text>要素にはclass属性でtextStyleが指定されています。

CSSファイルの4行目と7行目がその部分になります。ここではフォントの大きさ(font-size)とテキストの表示位置(text-anchor¹⁰)を指定しています。

- SVGの画像とボタン群を横に並べるために、それらを含む<div>要素にclass属性でCellがしていられています。

CSSファイルの8行目と13行目がその部分になります。ここではフォントの大きさを30px、表示形式(display)をinline-block(<div>要素を一つの文字のように取り扱う)、縦方向の表示位置(vertical-align)をmiddle(中央)、左方向の空白(padding-left)を5pxに設定しています。

- 右側のテキストボックスにはidが設定されています。この属性値の前に#をつけるとそれらの要素に対してのセレクタになります。

¹⁰これはSVGにおける位置指定です。

14 行目と 17 行目がその部分になります。ここでは `<input>` 要素で `type` が `text` に対して右寄せになるように `text-align` を `right` に設定しています。SVG の場合と異なることに注意してください。また、色を選択するプルダウンメニュー (`<id>` 要素が `SelectColor`) と設定ボタン (`<id>` 要素が `SetColor`) に対しても同様の設定をしています。

リスト 7.21 はリスト 7.19 から読み込まれる JavaScript ファイルです。

JavaScript リスト 7.21: リスト 7.19 から読み込まれる JavaScript ファイル (SSClickPos.js)

```
1  var Circle, X, Y, XP, YP, oL, oT, B;
2  window.onload = function() {
3      var Colors = {"red":"赤", "yellow":"黄色", "green":"緑",
4                     "blue":"青", "gray":"灰色", "black":"黒"};
5      var tmp, tmpText, Color;
6      var SelectColor = document.getElementById("SelectColor");
7      for( Color in Colors) {
8          tmp = document.createElement("option");
9          tmp.setAttribute("value", Color);
10         tmpText = document.createTextNode(Colors[Color]);
11         tmp.appendChild(tmpText);
12         SelectColor.appendChild(tmp);
13     }
14     XP = document.getElementById("XP");
15     YP = document.getElementById("YP");
16     Circle = document.getElementById("Circle");
17     X = document.getElementById("X");
18     Y = document.getElementById("Y");
19     XP.value = Circle.getAttribute("cx");
20     YP.value = Circle.getAttribute("cy");
21     document.getElementById("field").addEventListener("click", click, false);
22     document.getElementById("SetColor").onclick = refresh;
23
24     B = document.getElementById("canvas").getBoundingClientRect();
25     oL = Math.floor(B.left)+5;
26     oT = Math.floor(B.top)+5;
27     refresh();
28 }
29 function click(event) {
30     XP.value = event.clientX-oL;
31     YP.value = event.clientY-oT;
32     refresh();
33 }
34 function refresh() {
35     SetText(X,"cx", XP.value);
36     SetText(Y,"cy", YP.value);
37     Circle.setAttribute("fill", SelectColor.value);
38 }
39
40 function SetText(Element, attrib, Value) {
41     var txtNode = document.createTextNode(Value);
42     if( Element.firstChild) {
43         Element.replaceChild(txtNode, Element.firstChild);
44     } else {
45         Element.appendChild(txtNode);
```

```
46      }
47      Circle.setAttribute(attrib, Value);
48  }
```

- 2行目からHTML文書がすべて読み終えたときに実行する関数の定義の始まりです。
- ここでは色を選択するリストボックスを作成しています。
- まず、色の選択をする選択リストへの参照を得ます(6行目)。
- 色を指定するリストボックスの内容をDOMの技法を用いて設定します。
- HTML文書の中に<select>要素がありますので、この子要素<option>要素を付け加えることをします。<option>要素は次のような形になります¹¹。

```
<option value="red">赤</option>
<option value="yellow">黄色</option>
```

- 3行目から13行目で属性値となる色の名称と日本語による表示のための文字列をオブジェクトリテラルとして定義しています。オブジェクトリテラルでは配列の引数に文字列をとることができます。
- このデータを用いて7行目から13行目でname属性がSetColorのリストボックスの内容を作成しています。
- オブジェクトリテラルのすべての要素にアクセスする方法はfor(変数名 in オブジェクト名)です(7行目)。この変数に配列の引数がセットされます。ここでは英語の文字列が値として代入されます。
- 8行目で<option>要素要素を作成し、9行目valueに色名(ここではfor内の変数であるキーの値Color)を設定しています。この値は英語における色名になっているので円の色名を設定するためにそのまま利用できます。
- <option>要素に表示する文字列を設定するために10行目でテキストノードを作成しています。この文字列はオブジェクトリテラルの値のほうを用います。
- 11行目でこのテキストノードを<option>要素要素の子ノードとして付け加えています。
- 作成された要素をリストボックスの子ノードとして付け加えます(12行目)。
- 14行目から18行目でHTML文書内のオブジェクトを参照するための変数の値を定義しています。
- その後、表示されている円の座標をテキストボックスに反映させています(19行目と20行目)。

¹¹ HTML文書を作成したことがある人は</option>はいらないのではと思うかもしれません。HTML文書をXMLの規格に合わせるXHTMLという規格では要素の開始タグに対応する終了タグは必ず記述する必要があります。

- 21 行目で定義されている関数 `click(event)` は SVG 文書上でマウスがクリックされたときに呼び出されます。イベントリスナーの定義は29 行目から33 行目で定義されています。
- 22 行目では「設定」ボタンがクリックしたとき (`onclick`) に呼び出される関数を「`refresh`」に定義しています。`function` で定義された関数名はそのままグローバル変数として使用できることに注意してください。
- この XML 文書上でクリックするとそのクリックされた位置は文書全体の位置になり、SVG 文書内の位置とは異なります。要素が配置された位置を得るための関数が `getBoundingClientRect` です (24 行目)。
- 実際にクリックできる範囲は`<svg>` 要素の中で下、右にそれぞれ 5 ピクセル移動していますので、その分も配置からずらします (25 行目と26 行目)。
なお、これらの値は状況によっては整数とはならないので、小数点以下を切り捨てています。
- 29 行目から33 行目でクリックされたときのイベント処理関数が定義されています。ここではクリックされた位置から25 行目と26 行目で求めた補正值を引いてそれぞれのテキストボックスに表示させています。

問題 7.19 リスト 7.19 について次のことを行いなさい。

1. 表題のフォントの大きさを変えてもクリックする位置は正しく求められることを確認しなさい。
2. 起動後、ウィンドウの幅を狭くして（または広くして）表題の部分の行数を変化させるとクリックした位置と円が設定される場所が異なることを確認しなさい。
3. 2 の不具合を直しなさい

図 7.24 は図 7.19 において HTML 文書から色を設定できるようにしたものです。

図形の右側に曲線を構成する色を入力するテキストボックスと、設定を図に反映させるボタンがあります。

HTML リスト 7.22: バインジオ・ピンナの錯視図形の色をテキストボックスから設定

```

1  <!DOCTYPE html>
2  <html xmlns:svg="http://www.w3.org/2000/svg"
3      xmlns:xlink="http://www.w3.org/1999/xlink">
4  <head>
5  <meta charset="UTF-8"/>
6  <script type="text/ecmascript" src=".//make-svg-elm.js" ></script>
7  <script type="text/ecmascript" src="pinna-rev.js"></script>
8  <link rel="stylesheet" type="text/css" href="pinna.css">
9  <title>バインジオ・ピンナの錯視図形</title>
10 </head>
11 <body>
12   <h1 class="display">バインジオ・ピンナの錯視図形</h1>
13   <div class="Cell">
14     <svg height="420" width="420"

```



図 7.24: バインジオ・ピンナの錯視图形をテキストボックスから設定

```

15      <g id="Canvas" transform="translate(210,210)"/>
16    </svg>
17  </div>
18  <div class="Cell" >
19    <div><label for="color1">色 1</label>
20      <input type="text" id="color1" size="5"/></div>
21    <div><label for="color2">色 2</label>
22      <input type="text" id="color2" size="5"/></div>
23    <input id="SetColor" type="button" value="設定"></input>
24  </div>
25 </body>
26 </html>
```

リスト 7.15 では一つのファイルにまとめてありました、ここでは CSS ファイルと JavaScript ファイルは外部ファイルとしています。

この HTML 文書は本質的にリスト 7.19 と同じです。図形を描いている部分がなく、座標を移動する<g> 要素があるだけです。

次のリストはリスト 7.22 で読み込む CSS ファイルです。

CSS リスト 7.23: リスト 7.22 で読み込む CSS ファイル

```

1 .display {
2   font-size:25px;
3 }
4 .Cell {
5   font-size:30px;
```

```

6   display:inline-block;
7   vertical-align:middle;
8   padding-left:5px;
9 }
10 #color1, #color2 {
11   font-size:25px;
12   text-align: right;
13 }
14 #SetColor{
15   font-size:25px;
16   text-align:center;
17 }

```

これもリスト 7.20 とほとんど変わりません。

JavaScript リスト 7.24: リスト 7.22 で読み込む JavaScript ファイル (pinna.js)

```

1 var Canvas, C1, C2, Paths =[];
2 window.onload = function(){
3   Canvas = document.getElementById("Canvas");
4   C1 = document.getElementById("color1");
5   C2 = document.getElementById("color2");
6   for(var i= 0; i<6;i++) {
7     Paths[i] = MKSVGElm(Canvas, "path", {"stroke-width": 6, "fill": "none"}, {});
8   }
9   C1.value = "red";
10  C2.value = "green";
11  document.getElementById("SetColor").addEventListener("click", DrawFigs, true);
12  DrawFigs();
13 }
14 function DrawFigs() {
15   var W1=8, W2=4;
16   var Color1 = C1.value;
17   var Color2 = C2.value;
18   DrawFigure(150, 30, W1, W2, Color1, 0);
19   DrawFigure(144, 30, W1, W2, Color2, 1);
20   DrawFigure(80, 20, W1, W2, Color1, 2);
21   DrawFigure(86.5, 20, W1, W2, Color2, 3);
22   DrawFigure(14, 20, 0, 0, Color1, 4);
23   DrawFigure(10, 20, 0, 0, Color2, 5);
24 }
25 function DrawFigure(R, sR, W, W2, Color, No) {
26   var d = "M", i, Ang, R0;
27   for(i=0;i<720;i++) {
28     Ang= Math.PI*i/180/2;
29     R0=R+sR*(Math.cos(W*Ang)*Math.cos(W2*Ang));
30     d += R0*Math.cos(Ang) + ","+R0*Math.sin(Ang)+" ";
31   }
32   SetAttributes(Paths[No], {"d": d+"z", "stroke": Color});
33 }

```

- このリストでは図形を作成して表示する関数 `DrawFigs()` とそれから 6 回呼び出される関数 `DrawFigure()` の基本的なアルゴリズムは全く同じです。

- すべてのファイルが呼び込まれた後に発生するイベントの処理関数 `window.onload` が2行目と13行目で定義されています。
 - 4行目と5行目で設定されたいろがある要素を変数に格納しています。
 - 図形を構成する4つの要素を作成して、配列に格納しています(6行目と8行目)。これは、後で色を設定しなおすときの処理を簡単にするためにです。
 - 9行目と10行目でテキストボックスの初期値を設定しています。
 - 11行目ではボタンが押されたときの処理関数を定義しています。
 - 12行目で初期値を用いて図形を表示します。
- 25行目と33行目で図形を表示する関数を定義しています。色はテキストボックスから読むので、仮引数が必要でなくなっています。
- 18行目と23行目で一周分の図形を描く関数 `DrawFigure` を呼びだしています。以前と異なり、何番目の図形なのかを示す引数が追加されています。
- この仮引数を使って、計算された図形の形と色を設定しています(32行目)。

問題 7.20 今までに示された錯視图形のパラメータを外部から設定するように書き直しなさい。

7.6 より進んだJavaScriptプログラミング

7.6.1 配列のメソッドを使う

配列には表7.6のようなメソッドやプロパティが定義されています。

表7.6: 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>fill(value[,start[,end]])</code>	与えられた配列の <code>start</code> (デフォルトは0) から <code>end</code> (デフォルトは配列の長さでこのインデックスは含まない) を <code>value</code> の値で設定する。
<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。

次ページへ続く

表 7.6: 配列のメソッド (続き)

メンバー	説明
indexOf (value[,start])	start 以降 (指定しない場合は 0) の要素で value 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。
lastIndexOf (value[,start])	start 以前 (指定しない場合は配列の最後) の要素で value 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。
pop()	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。
push(i1,[i2,...])	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。
shift()	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。
unshift(i1[,i2,...])	引数で渡された要素を配列の最初の要素とする。
reverse()	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。
slice(start[,end])	start から end の前の位置にある要素を取り出した配列を返す。end 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。
splice (from,No,i1,i2,...)	与えられた配列の from の位置から No 個の要素を取り除き、その位置に i1,i1,... 以下の要素を付け加える。
forEach(func)	引数 func 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 index、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。
map(func)	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 forEach と同じである。
reduce (func[,initial])	func は 2 つの仮引数をとる関数。initial がないときは func 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。initial 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。
reduceRight (func[,initial])	reduce 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。
every(func)	func() を各配列の要素に順に適用し、どこかで false のときにはそこで実行が打ち切られ、false の値が返る。すべてが true と解釈される値のとき、true を返す。func の引数は forEach と同じである。

次ページへ続く

表 7.6: 配列のメソッド(続き)

メンバー	説明
some(func)	func()を各配列の要素に順に適用し、どこかでtrueと解釈される値のときにはそこで実行が打ち切られ、trueを返す。すべての結果がfalseのときはfalseとなる。funcの引数はforEachと同じである。
filter(func)	関数funcの戻り値がtrueに変換される要素を集めて新しい配列を作成する。funcの引数はforEach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いるので、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 7.21 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);
```

なお、`document.getElemensByTagName` などで得られたものは配列のようにアクセスできますが、配列オブジェクトではないので `forEach` などは直接利用できません。次のような形で利用します。

```
A = document.getElementsBytagName("input");
Array.prototype.forEach.call(A,function(){...});
```

`forEach` 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 A(HTML コレクション) のメソッドとして呼び出すために `call` を用いています。`forEach` 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。

7.6.2 クロージャの利用によるグローバル変数の個数の減少

変数のスコープと仮引数への値の渡し方

変数のスコープとは使用する変数がどの範囲で参照できるか概念です。

- JavaScriptは変数は宣言なくても使用可能です。最近のJavaScriptの仕様では宣言しない変数は使えないモードを宣言できます。
- 関数などの外で宣言されたり、宣言がない変数はどこからでも参照が可能な変数（グローバル変数）になります。
- 関数内で宣言された変数はそれ以下の関数内で参照可能です。

関数の引数に対して、単純な変数は値渡し、配列は参照渡しとなります。

```

1 >foo =function(X,Y,Z) {
2   X = X*2;
3   Y[0] *= 2;
4   Z.x *=2;}
5 function (X,Y,Z) {
6   X = X*2;
7   Y[0] *= 2;
8   Z.x *=2;}
9 >A=1
10 1
11 >B=[2,3]
12 ▶[2, 3]
13 >C={x:10,y:20};
14 ▶Object {x: 10, y: 20}
15 >foo(A,B,C);
16 undefined
17 >A;
18 1
19 >B;
20 ▶[4, 3]
21 >C;
22 ▶Object {x: 20, y: 20}

```

クロージャ

今までのJavaScriptのプログラムでは、他の関数とデータのやり取りにグローバル変数を多用してきました。グローバル変数を多用することは、複数人でプログラムを開発するときなどに変数名の衝突がおき、それによってバグが発生する可能性が増大します。これを避ける方法の一つとして関数の中で別の関数オブジェクトを作成する方法があります。関数の中で新しく関数を定義すると、新しい関数ではその関数を定義する関数内のローカル変数を参照できます。関数が実行されるときの環境を含めた状態をクロージャと呼びます。

リスト7.25はリスト7.17(169ページ)を改造して、グローバル変数をまったく使わないサイクロイドをアニメーションで描くものです。

SVGリスト7.25: サイクロイドを描く --- アニメーション版(グローバル変数なし)

```

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

```

```

5 //<![CDATA[
6 window.onload = function () {
7     var C =document.getElementById("cycliod");
8     var T =document.getElementById("translate");
9     var Rot =document.getElementById("rotate");
10    var R = 50, Current = 1, Step=2, d =["M0,0"];
11    (function DrawNext(){
12        var Next = Current + Step, rad;
13        if(Current&lt;=360) {
14            for( ; Current&lt; Next; Current++) {
15                rad = Current/180*Math.PI;
16                pX = R*(rad-Math.sin(rad));
17                pY = R*(-1+Math.cos(rad));
18                d.push(pX,pY);
19            }
20            C.setAttribute("d",d.join(" "));
21            T.setAttribute("transform","translate(" + (R*rad) + ",-50)");
22            Rot.setAttribute("transform", "rotate("+Next+"')");
23            setTimeout(DrawNext,100);
24        }
25    })()
26 }
27 //]]&gt;
28 &lt;/script&gt;
29 &lt;g transform="translate(100,200)"&gt;
30     &lt;line x1="-50" y1="0" x2="400" y2="0" stroke-width="2" stroke="black"/&gt;
31     &lt;path id="cycliod" fill="none" stroke="black" stroke-width="3"/&gt;
32     &lt;g id="translate" transform="translate(0,-50)"&gt;
33         &lt;g id="rotate"&gt;
34             &lt;circle cx="0" cy="0" r="50" stroke-width="3" stroke="black" fill="none"/&gt;
35             &lt;line x1="0" y1="0" x2="0" y2="50" stroke-width="2" stroke="black" /&gt;
36             &lt;circle cx="0" cy="50" r="5" fill="black"/&gt;
37         &lt;/g&gt;
38     &lt;/g&gt;
39 &lt;/g&gt;
40 &lt;/svg&gt;</pre>

```

- すべてのグローバル変数を `onload` 後に実行する関数内に移動しています (7 行目から 10 行目)。
- その関数内でローカルな関数を定義しています (11 行目から 25 行目)。
 - 関数名は関数 `DrawNext` となっています (11 行目)。
 - `function` の前に (がついています。これに対応する) は 25 行目にあります。そのあとにある `()` はこの関数の引数リストです (ここでは引数がないので空です)。
これにより関数を定義してその場で実行できます。
 - 関数の定義位置が変わっただけで、残りのコードはほとんど同じです。
 - 一番の違いは `d` で必要となる座標データを配列として保存していることです (10 行目)。
 - また、18 行目で新たに計算した座標をこの配列に追加しています (`push` メソッド)。

- 20行目の `join` は配列の要素を、引数の文字列をはさんでつなげるメソッドです。配列内の文字列の連接を連接演算子`+`を用いるよりも簡単に記述できます。
- 23行目ではこの関数自身がこの後、呼び出せるように設定しています¹²。

JavaScript の配列のメソッドを用いて、JavaScript リスト 7.24 を書き直したものが JavaScript リスト 7.26 です。

JavaScript リスト 7.26: リスト 7.22 で読み込む JavaScript ファイル--改良版 (pinna-rev.js)

```

1  window.onload = function(){
2      var Canvas = document.getElementById("Canvas");
3      var C1 = document.getElementById("color1");
4      var C2 = document.getElementById("color2");
5      var Paths = new Array(6).fill(1).map(function() {
6          return MKSVGelm(Canvas, "path", {"stroke-width": 6, "fill": "none"}, {});
7      });
8      C1.value = "red";
9      C2.value = "green";
10     document.getElementById("SetColor").addEventListener("click", DrawFigs, true);
11     DrawFigs();
12     function DrawFigs() {
13         var W1=8, W2=4;
14         var Color1 = C1.value;
15         var Color2 = C2.value;
16         [[150, 30, W1, W2, Color1], [144, 30, W1, W2, Color2],
17          [80, 20, W1, W2, Color1], [86.5, 20, W1, W2, Color2],
18          [14, 20, 0, 0, Color1], [10, 20, 0, 0, Color2]].forEach(
19              function(Param, No) {
20                  var R=Param[0], sR = Param[1], W = Param[2],
21                      W2 = Param[3], Color = Param[4];
22                  var d = "M", i, Ang, R0;
23                  for(i=0;i<720;i++) {
24                      Ang= Math.PI*i/180/2;
25                      R0=R+sR*(Math.cos(W*Ang)*Math.cos(W2*Ang));
26                      d += R0*Math.cos(Ang) + "," + R0*Math.sin(Ang) + " ";
27                  }
28                  SetAttributes(Paths[No], {"d": d+"z", "stroke": Color});
29              })
30      }
31  }

```

- 5行目で長さが 6 の配列に图形の構成要素の`<path>` 要素を代入しています。
 - `new Array(6)` は長さが 6 の配列が作成されますが、値は `undefined` で初期化されます。
 - 値が `undefined` の要素に対しては `map` などは適用されません。
 - したがって、`fill` で 1 に初期化し、その配列に対して`<path>` 要素を代入しています。

¹² 関数の引数のリストを参照する `arguments` のメンバー `callee` を用いると、無名関数を参照できます。最新版の JavaScript では非推奨となっているので無名関数にしません。

- それぞれの<path> 要素を構成するところにはそれぞれのパラメタを一つの配列にし、それらを集めて長さが 6 の配列にしたもの(16 行目と 18 行目)に対して forEach を適用しています。
- 前の例と同じにするために配列の成分を借り引数の名前と同じ変数に代入しています(20 行目と 21 行目)

7.6.3 WebStorage

Web ページ上で作成したデータをそのままブラウザが動作しているコンピュータ上に保存することはセキュリティ上できません。しかし、ブラウザが管理する領域に保存することは可能です。この節ではそれらの方法について簡単な解説をします。

Web Storage とは HTML5 で導入された、Web ページ間のデータの共有する仕組みです。HTML5 以前の方法ではクッキーと呼ばれるデータをクライアントとサーバーの間で共有してサーバー側からデータの共有をサポートする仕組みがありました。クッキーでは小さなデータしか扱えず、サーバーとクライアントとのやりがあるたびに送られるのでセキュリティや速度の面で問題がありました。

HTML5 ではこれに代わる方法としてブラウザーのあるページ以降のページにだけ存在する sessionStorage とページが閉じられてもデータが存続できる localStorage が提案されています。なお、これらの値はすべて文字列でしか保存できません。

リスト 7.27 はリスト 7.22(181 ページ)において最後に設定した色を Storage に保存しておくように JavaScript ファイルを書き直したものです。

JavaScript リスト 7.27: Storage を利用する (pinna-storage.js)

```

1 var Storage = window.localStorage;
2 //var Storage = window.sessionStorage;
3 window.onload = function(){
4     var Canvas = document.getElementById("Canvas");
5     var C1 = document.getElementById("color1");
6     var C2 = document.getElementById("color2");
7     var Paths = new Array(6).fill(1).map(function() {
8         return MKSVGElm(Canvas, "path", {"stroke-width": 6, "fill": "none"}, {});
9     });
10    C1.value = Storage.C1?Storage.C1:"red";
11    C2.value = Storage.C2?Storage.C2:"green";
12    document.getElementById("SetColor").addEventListener("click", DrawFigs, true);
13    DrawFigs();
14    function DrawFigs() {
15        var W1=8, W2=4;
16        var Color1 = C1.value;
17        var Color2 = C2.value;
18        Storage.C1 = Color1;
19        Storage.C2 = Color2;
20        [[150, 30, W1, W2, Color1], [144, 30, W1, W2, Color2],
21         [80, 20, W1, W2, Color1], [86.5, 20, W1, W2, Color2],
22         [14, 20, 0, 0, Color1], [10, 20, 0, 0, Color2]].forEach(
23             function(Param, No) {

```

```

24     var R=Param[0], sR = Param[1], W = Param[2],
25         W2 = Param[3], Color = Param[4];
26     var d = "M", i, Ang, R0;
27     for(i=0;i<720;i++) {
28         Ang= Math.PI*i/180/2;
29         R0=R+sR*(Math.cos(W*Ang)*Math.cos(W2*Ang));
30         d += R0*Math.cos(Ang) + ","+R0*Math.sin(Ang)+" ";
31     }
32     SetAttributes(Paths[No], {"d": d+"z", "stroke": Color});
33 }
34 }
35 }

```

このリストは Storage に対応する部分が追加されているだけです。

- 1行目で変数 Storage に `window.localStorage` を代入しています。これは後で `sessionStorage` に書き直すのを簡単にするためです。
- 10行目ではテキストボックス「色1」の初期値を `Storage.C1` が定義されていたらその値を、そうでない場合には ''red'' に設定しています。
- 「色2」の方も同様です。
- 「設定」ボタンが押されたときに呼び出される関数 `DrawFigs` では色の情報を対応する Storage に格納しています (16行目と17行目)。

図7.25はデベロッパーツールで `localStorage` の値の確認をしているところです。

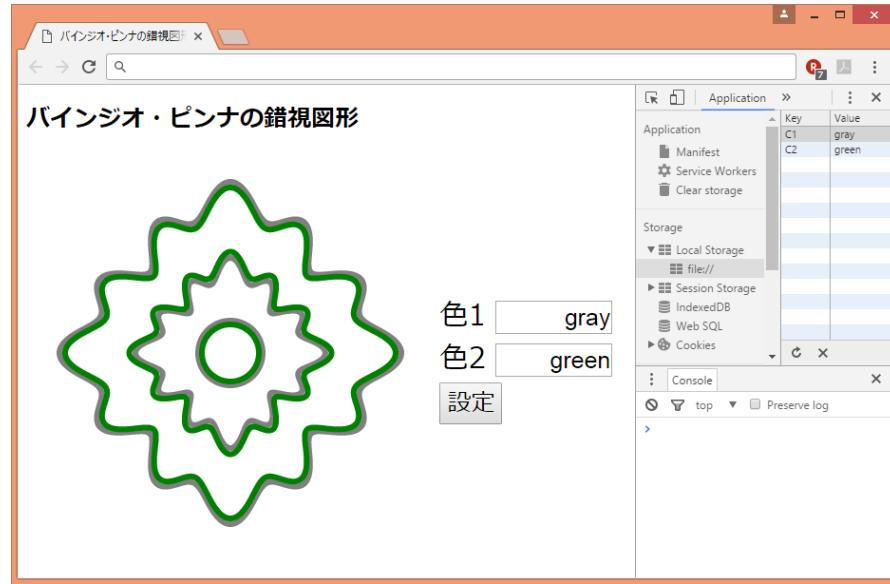


図 7.25: sessionStorage の値の確認

問題 7.22 リスト 7.27 について次のことを確認しなさい。

- localStorage の値上でダブルクリックする (または右クリック) と値が直接書き直せます。また、キーの削除もできます。別の値に書き直した後で開きなおすとその値が反映された図が描ける。
- リストの1行目をコメントアウトして、2行目のコメントを外して、別の値を設定する。
 - localStorage の方ではなく、sessionStorage の方が書き直されている。
 - 再表示すると最後に設定した値が反映されない。
 - 再表示後は sessionStorage の値が全くない。

付録A SVG の色名

次の表は [28] 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
lightblue	aliceblue	rgb(240, 248, 255)	#F0F8FF
lightbrown	antiquewhite	rgb(250, 235, 215)	#FAEBD7
lightgreen	aquamarine	rgb(127, 255, 212)	#7FFFAD
lightcyan	aqua	rgb(0, 255, 255)	#00FFFF
lightgray	azure	rgb(240, 255, 255)	#FOFFFF
lightyellow	beige	rgb(245, 245, 220)	#F5F5DC
lightorange	bisque	rgb(255, 228, 196)	#FFE4C4
black	black	rgb(0, 0, 0)	#000000
lightpink	blanchedalmond	rgb(255, 235, 205)	#FFEBBC
purple	blueviolet	rgb(138, 43, 226)	#8A2BE2
blue	blue	rgb(0, 0, 255)	#0000FF
brown	brown	rgb(165, 42, 42)	#A52A2A
lightbrown	burlywood	rgb(222, 184, 135)	#DEB887
teal	cadetblue	rgb(95, 158, 160)	#5F9EA0
lightgreen	chartreuse	rgb(127, 255, 0)	#7FFF00
darkbrown	chocolate	rgb(210, 105, 30)	#D2691E
lightorange	coral	rgb(255, 127, 80)	#FF7F50
lightblue	cornflowerblue	rgb(100, 149, 237)	#6495ED
lightyellow	cornsilk	rgb(255, 248, 220)	#FFF8DC
red	crimson	rgb(220, 20, 60)	#DC143C
lightcyan	cyan	rgb(0, 255, 255)	#00FFFF
darkblue	darkblue	rgb(0, 0, 139)	#00008B
teal	darkcyan	rgb(0, 139, 139)	#008B8B
gold	darkgoldenrod	rgb(184, 134, 11)	#B8860B
gray	darkgray	rgb(169, 169, 169)	#A9A9A9
darkgreen	darkgreen	rgb(0, 100, 0)	#006400
gray	darkgrey	rgb(169, 169, 169)	#A9A9A9
lightyellow	darkkhaki	rgb(189, 183, 107)	#BDB76B
purple	darkmagenta	rgb(139, 0, 139)	#8B008B
darkgreen	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)	#8FBBC8F
	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)	#FOFFFF
	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	rgb(250, 250, 210)	#FAFAD2
	lightgray	rgb(211, 211, 211)	#D3D3D3
	lightgreen	rgb(144, 238, 144)	#90EE90
	lightgrey	rgb(211, 211, 211)	#D3D3D3
	lightpink	rgb(255, 182, 193)	#FFB6C1
	lightsalmon	rgb(255, 160, 122)	#FFA07A
	lightseagreen	rgb(32, 178, 170)	#20B2AA
	lightskyblue	rgb(135, 206, 250)	#87CEFA
	lightslategray	rgb(119, 136, 153)	#778899
	lightslategrey	rgb(119, 136, 153)	#778899
	lightsteelblue	rgb(176, 196, 222)	#B0C4DE
	lightyellow	rgb(255, 255, 224)	#FFFFE0
	limegreen	rgb(50, 205, 50)	#32CD32
	lime	rgb(0, 255, 0)	#00FF00
	linen	rgb(250, 240, 230)	#FAF0E6
	magenta	rgb(255, 0, 255)	#FF00FF
	maroon	rgb(128, 0, 0)	#800000
	mediumaquamarine	rgb(102, 205, 170)	#66CDAA
	mediumblue	rgb(0, 0, 205)	#0000CD
	mediumorchid	rgb(186, 85, 211)	#BA55D3
	mediumpurple	rgb(147, 112, 219)	#9370DB
	mediumseagreen	rgb(60, 179, 113)	#3CB371
	mediumslateblue	rgb(123, 104, 238)	#7B68EE
	mediumspringgreen	rgb(0, 250, 154)	#00FA9A
	mediumturquoise	rgb(72, 209, 204)	#48D1CC
	mediumvioletred	rgb(199, 21, 133)	#C71585
	midnightblue	rgb(25, 25, 112)	#191970
	mintcream	rgb(245, 255, 250)	#F5FFFA
	mistyrose	rgb(255, 228, 225)	#FFE4E1
	moccasin	rgb(255, 228, 181)	#FFE4B5
	navajowhite	rgb(255, 222, 173)	#FFDEAD
	navy	rgb(0, 0, 128)	#000080
	oldlace	rgb(253, 245, 230)	#FDF5E6
	olivedrab	rgb(107, 142, 35)	#6B8E23
	olive	rgb(128, 128, 0)	#808000
	orangered	rgb(255, 69, 0)	#FF4500
	orange	rgb(255, 165, 0)	#FFA500
	orchid	rgb(218, 112, 214)	#DA70D6
	palegoldenrod	rgb(238, 232, 170)	#EEE8AA

次のページへ

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)	#FFEFB5
	peachpuff	rgb(255, 218, 185)	#FFDAB9
	peru	rgb(205, 133, 63)	#CD853F
	pink	rgb(255, 192, 203)	#FFC0CB
	plum	rgb(221, 160, 221)	#DDAODD
	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)	#FFFafa
	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 [8] 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 [28] 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。

[8] 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 [28] 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 [9] 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 [10] 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 [32] 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 [11] がお勧めです。また、使用を避けるべき JavaScript の点については [5] 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 [6]。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 [27] 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 [21, 13] 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 [39] です。

¹iPhone 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] David Cox, John Little, Donal O'Shea(落合、示野、西山、室、山本訳), グレブナ基底と代数多様体入門(上), シュプリンガー・フェアラーク東京 2000
- [5] 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
- [6] ECMA, ECMAScript[©] 2015 Language Specification(6th edition)
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- [7] ECMA, JSON, <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>, 2013
- [8] J. David Eisenberg, *SVG Essentials*, O'Reilly, 2002
- [9] David Flanagan(村上列 訳) JavaScript 第 6 版, オライリー・ジャパン, 2012
- [10] 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
- [11] 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
- [12] 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
- [13] Rasmus Lerdorf, Kevin Tattroe, 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
- [14] 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

- [15] Document Object Model (DOM) Level 3 Events Specification, W3C Technical Reports and Publications, 2007, <http://www.w3.org/TR/DOM-Level-3-Events/>
- [16] 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
- [17] Donald E. Knuth, *The METAFONTbook*, Computers & typesetting /C, Addison-Wesley, 2000
- [18] Hajime Ōuchi, Japanese Optical and Geometrical Art, Dover Pub., 1977
- [19] J. O. Robinson, *The Psychology of Visual Illusion*, Dover, N. Y., 1998(originally published in 1972)
- [20] 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
- [21] David Sklar (桑村 潤, 廣川 類 訳) 初めての PHP5, オライリージャパン, 2005
- [22] 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
- [23] W3C, Document Object Model (DOM), <http://www.w3.org/DOM>
- [24] W3C, W3C DOM4 <https://www.w3.org/TR/2015/REC-dom-20151119/>
- [25] W3C, Document Object Model(DOM) Level 2 Events Specification, W3C Recommendation, 2000 <http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/>
- [26] W3C, HTML 4.01 Specification, W3C Recommendation 24 December 1999
- [27] W3C, HTML5 A vocabulary and associated APIs for HTML and XHTML,
<http://www.w3.org/TR/html5/>
- [28] W3C, Scalable Vector Graphics (SVG) 1.1 Specification,
<http://www.w3.org/TR/2011/REC-SVG11-20110816/>
- [29] W3C, HTML & CSS , <http://www.w3.org/standards/webdesign/htmlcss>
- [30] W3C, Web Storage, <https://www.w3.org/TR/webstorage/>
- [31] 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
- [32] 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
- [33] 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)
- [34] 北岡明佳, 錯視入門, 朝倉書店、2010 年
- [35] 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

付録 9

- [36] 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
- [37] 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)
- [38] 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
- [39] 日本 PHP ユーザー会, <http://www.php.gr.jp/>

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, 77
 <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, 148, 150
 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
 blue(値-色) 169
 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, 138, 169
 —— の属性 23
 class 117
 click(値) 146
 color(アニメーション) 90
 color(図形) 78
 CSS(値) 73
 currentColor(値-色) 78, 90
 cursive(値-フォント) 93

cx(色) 33
 cx(図形) 23, 87
 cy(色) 33
 cy(図形) 23, 140

D

d(図形) 43, 45, 46, 53, 80, 81, 156, 168
 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
 d 189

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
 <feComposit> 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) 121
 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
 feSpecularLightng(フィルター) 123
 feTurbulence(フィルター) 148, 149
 feTurbulence(フィルター) 123, 124
 fill(アニメーション) 73, 75
 fill(図形) 18, 23, 27, 34, 36–38, 40, 42, 45, 61, 62, 70, 78, 79, 87, 90, 92, 108, 115, 134, 137, 174
 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, 78, 145, 146, 151, 154, 165, 167, 173, 177, 182
 gamma(値-フィルタ) 116
 gradientUnits(色) 26, 29, 31–33, 81, 82
 gray(値-色) 14, 87
 green(値-色) 169

H

hanging(値-フォント) 93
 height(図形) 13, 18, 20, 61, 63, 70, 155
 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, 141, 154, 167
 <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, 154
 <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, 151

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
- onclick 136, 138
- onload 138, 139
- opacity(色) 36, 37, 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, 156, 168, 190, 191
- path(図形) 77
- <pattern> 61, 62, 66, 68–70
- patternTransform(図形) 66
- patternUnits(図形) 62
- points(図形) 39, 40
- pointsAtX(フィルタ) 121
- pointsAtY(フィルタ) 121
- pointsAtZ(フィルタ) 121
- <polygon> 39, 40, 45
- <polyline> 39, 45, 129
- preserveAspectRatio(図形) 68

Q

- Q(値-図形) 44
- q(値-図形) 44

R

- r(色) 33
- r(図形) 22, 23
- <radialGradient> 33
- の属性 33
- <rect> 18, 76, 141, 148
- の属性 18
- red(値) 115
- red(値-色) 11, 78
- 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(図形) 57
- round(値-図形) 14, 41
- rx(図形) 18, 22, 23
- ry(図形) 18, 22, 23

S

- s(値-図形) 44, 53
- s(値-フォント) 44
- 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
- <script> 137
- 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-図形) 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(図形) 14, 18, 23, 36–38, 79
- stroke-linecap(図形) 12, 14
- stroke-linejoin(図形) 41, 42
- stroke-opacity(色) 36
- stroke-width(図形) .. 14, 18, 20, 23, 37, 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, 141, 146–148, 181

T

- t(値-図形) 44
- t(値-フォント) 44
- table(フィルタ) 116
- tableValues(フィルタ) 116

<text> .. 42, 91, 92, 95, 97, 142, 143, 151, 174,
 178
 text-anchor(フォント) 93, 95, 178
 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, 165
 —— 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
 type 137

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,
 81, 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, 18, 20, 61, 63, 70, 80, 129, 155
 width(フィルタ) 101, 102

X

x(図形) 18, 22, 64, 70, 75, 87, 90, 92, 150
 x(フィルタ) 101, 102, 121
 x1(色) 28, 29, 32, 81, 82
 x1(図形) 14
 x2(色) 28, 29, 32, 81, 82
 x2(図形) 14
 xlink:href 17, 97, 99, 165
 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) 18, 22, 64, 70, 75, 92
 y(フィルタ) 101, 102, 121
 y1(色) 28, 29, 32, 81
 y1(図形) 14

y2(色) 28, 29, 32, 81
 y2(図形) 14

Z

z(値-図形) 44, 46, 56
 z(フィルタ) 121

あ
値

..... 17
 click 146
 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
 blue 169
 currentColor 78, 90
 evenodd 40
 gray 14, 87
 green 169
 lightgrey 88
 none 12, 34, 42, 45
 objectBoundingBox 26, 29, 31–33, 35, 62,
 70
 red 11, 78
 rgb 11, 12, 38
 url 62
 userSpaceOnUse 26, 29, 32–34, 62, 70, 81,
 82
 white 70, 95

値-図形

A 44

a	44	auto	93		
bevel	41	baseline	93		
butt	14	cursive	93		
C	44	end	93		
c	44	fantasy	93		
L	44	font-family	93		
l	44, 46	hanging	93		
M	44, 56	ideographic	93		
m	44	Impact	95		
miter	41	italic	93		
Q	44	line-through	93		
q	44	mathematical	93		
round	14, 41	middle	93, 95		
S	44, 53	monospace	93		
s	44	none	93		
square	14	normal	93		
T	44	oblique	93		
t	44	overline	93		
xMaxYMid	68	sans-serif	93		
z	44, 46, 56	serif	93		
値-フィルタ					
arithmetic	118, 120, 123	start	93		
atop	118, 120	sub	93		
BackgroundAlpha	106	super	93		
BackgroundImage	106	underline	93		
darken	106, 107, 109	アニメーション			
discrete	116	attributeName	73, 74, 80		
FillPaint	106	attributeType	73, 80		
filter	103	begin	73, 86, 87		
fractalNoise	124	calcMode	73, 83, 90		
gamma	116	color	90		
hueRotate	109, 112	dur	73, 75, 87		
identity	116	end	86		
in	118, 120	fill	73, 75		
intercept	116	from	73, 75, 81, 83		
lighten	106–108, 120	keyTimes	73, 83, 84		
linear	116	repeatCount	73, 75, 76, 90		
luminanceToAlpha	109, 116	rotate	79		
matrix	109, 111	to	73, 75, 81, 83		
multiply	107	type	75, 76		
normal	107	values	73, 83, 84, 90		
objectBoundingBox	101	い			
out	118, 120	色			
over	118, 120	cx	33		
saturate	109	cy	33		
screen	107	fx	33		
SourceAlpha	105, 106	fy	33		
SourceGraphic	106, 108	gradientUnits	26, 29, 31–33, 81, 82		
StrokePaint	106	offset	27		
turbulence	124	opacity	36, 37, 86		
userSpaceOnUse	101, 102	r	33		
xor	118, 120	stop-color	27, 83		
値-フォント					
alphabetic	93	stop-opacity	36, 38		
		stroke-opacity	36		

x1 28, 29, 32, 81, 82
 x2 28, 29, 32, 81, 82
 y1 28, 29, 32, 81
 y2 28, 29, 32, 81

す

図形

color 78
 cx 23, 87
 cy 23, 140
 d 43, 45, 46, 53, 80, 81, 156, 168
 end 87
 fill-opacity 36, 37
 fill-rule 40
 fill 18, 23, 27, 34, 36–38, 40, 42,
 45, 61, 62, 70, 78, 79, 87, 90, 92, 108,
 115, 134, 137, 174
 height 13, 18, 20, 61, 63, 70, 155
 linecap 14
 linejoin 41, 42
 maskUnits 70
 miterlimit 41
 none 68
 path 77
 patternTransform 66
 patternUnits 62
 points 39, 40
 preserveAspectRatio 68
 r 22, 23
 rotate 57
 rx 18, 22, 23
 ry 18, 22, 23
 stroke-linecap 12, 14
 stroke-linejoin 41, 42
 stroke-width 14, 18, 20, 23, 37, 46, 56
 stroke 14, 18, 23, 36–38, 79
 width 13, 18, 20, 61, 63, 70, 80, 129, 155
 x 18, 22, 64, 70, 75, 87, 90, 92, 150
 x1 14
 x2 14
 y 18, 22, 64, 70, 75, 92
 y1 14
 y2 14

ふ

フィルタ

amplitude 116
 azimuth 121
 baseFrequency 124, 148, 150
 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, 151
 offset 116
 operator 118, 120
 pointsAtX 121
 pointsAtY 121
 pointsAtZ 121
 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
 feDiffuseLighting 121
 feFlood 116
 feFoold 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 148, 149
 feTurbulence 123, 124

フィルタ内の要素

feComposite 118, 120, 122
 feDiffuseLighting 120
 feDistantLight 121
 feFuncA 116

feFuncB	116
feFuncG	116
feFuncR	116
fePointLight	121
feSpecularLighting	120
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, 178
text-decoration	93

よ**要素**

<animate>	79, 80, 83
<animateColor>	79
<animateMotion>	73, 77
<animateTransform>	74, 80
<animatMotion>	73
<circle>	22, 55, 56, 77, 138, 169
<defs>	17, 18, 25, 26, 61, 79, 86, 99
<ellipse>	22
<feComponentTransfer>	116
<feComposit>	118
<feGaussianBlur>	103
<filter>	102
<g>	13, 14, 17, 18, 32, 74, 77, 78, 145, 146, 151, 154, 165, 167, 173, 177, 182
<image>	67, 70
<line>	14, 154
<linearGradient>	26, 27
<mask>	69–72, 95
<mpath>	79
<path>	43, 45, 46, 53, 58, 62, 80, 81, 86, 97, 100, 156, 168, 190, 191
<pattern>	61, 62, 66, 68–70
<polygon>	39, 40, 45
<polyline>	39, 45, 129
<radialGradiation>	33
<rect>	18, 76, 141, 148
<script>	137
<set>	83
<stop>	27, 36, 83
<style>	103
<svg>	13, 14, 141, 146–148, 181
<text>	42, 91, 92, 95, 97, 142, 143, 151, 174, 178
<textPath>	97
<tspan>	92, 94
<use>	17, 22, 32, 37, 63, 64, 87

索引 — HTMLの用語

B

 7

C

<canvas> 6
class(属性) 103, 177, 178
CSS の値

 inline-block 178
 middle 178
 right 179

CSS プロパティ

 display 178
 font-size 178
 padding-left 178
 text-align 179
 vertical-align 178

D

display(CSS プロパティ) 178
<div> 177, 178
DOCTYPE 宣言 177

F

font-size(CSS プロパティ) 178

H

<h1> 177, 178
<head> 177
<htaml> 177
<HTML> 4

I

<id> 179
id(属性) 178
inline-block(CSS の値) 178
<input> 179

M

<meta> 177
middle(CSS の値) 178

N

name(属性) 180

O

<option> 180

P

padding-left(CSS プロパティ) 178

R

right(CSS の値) 179

S

<script> 134
<select> 180
setTimeout(オブジェクトのメソッド) 174

T

text(値) 179
text-align(CSS プロパティ) 179
type(属性) 179

V

value(属性) 180
vertical-align(CSS プロパティ) 178

あ

値

 text 179

お

オブジェクトのメソッド
 setTimeout 174

そ

属性

 class 103, 177, 178
 id 178
 name 180
 type 179
 value 180

よ

要素

| | |
|----------|----------------|
|
 | 7 |
| <canvas> | 6 |
| <div> | 177, 178 |
| <h1> | 177, 178 |
| <head> | 177 |
| <htaml> | 177 |
| <HTML> | 4 |
| <id> | 179 |
| <input> | 179 |

| | |
|----------------|-----|
| <meta> | 177 |
| <option> | 180 |
| <script> | 134 |
| <select> | 180 |

索引 — JavaScript の用語

Symbols

`+` 137

A

`addEventListener(DOM のメソッド)` 141

`addEventListner(DOM のメソッド)` ..138, 145

`alert` 137

`altKey(DOM のプロパティ)` 135

`appendChild(DOM のメソッド)` .132, 144, 156,
174

`arguments` 190

`Array` 187

Array のメソッド

`concat` 184

`every` 185

`fill` 184, 190

`filter` 186

`forEach` 185–187, 191

`indexOf` 185

`join` 184

`lastIndexOf` 185

`length` 184

`map` 185, 186, 190

`pop` 185

`push` 185

`reduce` 185, 187

`reduceRight` 185

`reverse` 185

`shift` 185

`slice` 185, 186

`some` 186

`sort` 184

`splice` 174, 185, 186

`unshift` 185

B

`button(DOM のプロパティ)` 135

C

`call` 187

`callee` 190

`childNodes(DOM のプロパティ)` 133

`children(DOM のプロパティ)` 133

`clearTimeout` 169

`clientX(DOM のプロパティ)`135, 141, 154

`clientY(DOM のプロパティ)`135

`cloneNode(DOM のメソッド)` 131

`concat(Array のメソッド)` 184

`createElement(DOM のメソッド)`131, 152

`createElementNS(DOM のメソッド)` .131, 152,
163

`createTextNode(DOM のメソッド)` .. 131, 144,
174

`ctrlKey(DOM のプロパティ)`135

`currentTarget(DOM のプロパティ)` ..135, 145

D

`document` 152

DOM のオブジェクト

`window` 137, 169

DOM のプロパティ

`altKey` 135

`button` 135

`childNodes` 133

`children` 133

`clientX` 135, 141, 154

`clientY` 135

`ctrlKey` 135

`currentTarget` 135, 145

`eventPhase` 135

`firstChild` 133, 143

`firstElementChild` 133

`hasChildNodes` 133

`lastChild` 133

`lastElementChild` 133

`metaKey` 135

`nextElementSibling` 133

`nextSibling` 133

`nodeName` 133

`nodeType` 133

`nodeValue` 133

`pageXOffset` 135

`pageYOffset` 135

`parentNode` 133

`previousElementSibling` 133

`previousSibling` 133

`screenX` 135

`screenY` 135

`shiftKey` 135

`target` 135, 137, 145

DOM のメソッド

`addEventListener` 141

addEventListner 138, 145
 appendChild 132, 144, 156, 174
 cloneNode 131
 createElement 131, 152
 createElementNS 131, 152, 163
 createTextNode 131, 144, 174
 getAltKey 135
 getAttribute 131, 137–139
 getButton 135
 getClientX 135
 getClientY 135
 getCtrlKey 135
 getCurrentTarget 135
 getElementById 131, 141
 getElementsByTagName 131
 getElementsByClassName 131
 getElementsByName 131
 getElementsByTagName 131, 138, 147
 getMetaKey 135
 getNodeName 131
 getScreenX 135
 getScreenY 135
 getShiftKey 135
 getTarget 135
 hasAttribute 131
 insertBefore 132
 preventDefault 135
 querySelector 131
 querySelectorAll 131
 removeAttribute 131
 removeChild 132
 replaceChild 132, 144, 174
 setAttribute 131, 138, 140, 141
 setValue 132
 stopPropagation 135

E

Event.AT_TARGET 135
 Event.BUBBLING_PHASE 135
 Event.CAPUTURING_PHASE 135
 eventPhase(DOM のプロパティ) 135
 every(Array のメソッド) 185

F

false 145, 185, 186
 fill(Array のメソッド) 184, 190
 filter(Array のメソッド) 186
 firstChild(DOM のプロパティ) 133, 143
 firstElementChild(DOM のプロパティ) 133
 for 180
 forEach 187
 forEach(Array のメソッド) 185–187, 191
 function 137, 181, 189

G

getAltKey(DOM のメソッド) 135

getAttribute(DOM のメソッド) 131, 137–139
 getBoundingClientRect 181
 getButton(DOM のメソッド) 135
 getClientX(DOM のメソッド) 135
 getClientY(DOM のメソッド) 135
 getCtrlKey(DOM のメソッド) 135
 getCurrentTarget(DOM のメソッド) 135
 getElementById(DOM のメソッド) 131, 141
 getElementsByTagName(DOM のメソッド) 131
 getElementsByName(DOM のメソッド) 131
 getElementsByTagName(DOM のメソッド) 131, 138, 147
 getMetaKey(DOM のメソッド) 135
 getNodeName(DOM のメソッド) 131
 getScreenX(DOM のメソッド) 135
 getScreenY(DOM のメソッド) 135
 getShiftKey(DOM のメソッド) 135
 getTarget(DOM のメソッド) 135

H

hasAttribute(DOM のメソッド) 131
 hasChildNodes(DOM のプロパティ) 133

I

indexOf(Array のメソッド) 185
 insertBefore(DOM のメソッド) 132

J

join 190
 join(Array のメソッド) 184
 jQuery.js 161
 JSON 160
 JSON.parse 160
 JSON.stringify 160

L

lastChild(DOM のプロパティ) 133
 lastElementChild(DOM のプロパティ) 133
 lastIndexOf(Array のメソッド) 185
 length(Array のメソッド) 184
 length(メソッド) 138

M

map(Array のメソッド) 185, 186, 190
 Math 150, 152
 Math.abs(x) 152
 Math.acos(x) 152
 Math.asin(x) 152
 Math.atan(x) 152
 Math.atan2(y, x) 152
 Math.ceil(x) 152
 Math.cos(x) 152, 158
 Math.E 152
 Math.exp(x) 152
 Math.floor() 175

Math.floor(x) 152
 Math.LN10 152
 Math.LN2 152
 Math.log(x) 152
 Math.LOG10E 152
 Math.LOG2E 152
 Math.max([x1, x2, ..., xN]) 152
 Math.min([x1, x2, ..., xN]) 152
 Math.PI 152, 158
 Math.pow(x, y) 150, 152
 Math.random() 152, 174
 Math.round(x) 152
 Math.sin(x) 152, 158
 Math.sqrt(x) 152
 Math.SQRT1_2 152
 Math.SQRT2 152
 Math.tan(x) 152
 metaKey(DOM のプロパティ) 135

N

nextElementSibling(DOM のプロパティ) 133
 nextSibling(DOM のプロパティ) 133
 nodeName(DOM のプロパティ) 133
 nodeType(DOM のプロパティ) 133
 nodeValue(DOM のプロパティ) 133
 null 143, 162, 163

P

pageXOffset(DOM のプロパティ) 135
 pageYOffset(DOM のプロパティ) 135
 parentNode(DOM のプロパティ) 133
 parseInt 140
 pop(Array のメソッド) 185
 preventDefault(DOM のメソッド) 135
 previousElementSibling(DOM のプロパティ)
 133
 previousSibling(DOM のプロパティ) 133
 prototype 187
 push 189
 push(Array のメソッド) 185
 push()(メソッド) 173

Q

querySelector(DOM のメソッド) 131
 querySelectorAll(DOM のメソッド) 131

R

reduce(Array のメソッド) 185, 187
 reduceRight(Array のメソッド) 185
 removeAttribute(DOM のメソッド) 131
 removeChild(DOM のメソッド) 132
 replaceChild(DOM のメソッド) 132, 144, 174
 reverse(Array のメソッド) 185

S

screenX(DOM のプロパティ) 135

screenY(DOM のプロパティ) 135
 setAttribute(DOM のメソッド) 131, 138, 140,
 141
 setTimeout() 169, 175
 setValue(DOM のメソッド) 132
 shift(Array のメソッド) 185
 shiftKey(DOM のプロパティ) 135
 slice(Array のメソッド) 185, 186
 some(Array のメソッド) 186
 sort(Array のメソッド) 184
 splice(Array のメソッド) 174, 185, 186
 splice(メソッド) 174
 stopPropagation(DOM のメソッド) 135

T

target(DOM のプロパティ) 135, 137, 145
 true 145, 185, 186

U

undefined 190
 undefined(値) 190
 unshift(Array のメソッド) 185

W

window(DOM のオブジェクト) 137, 169

あ

値

undefined 190

く

グローバル変数 188

へ

変数のスコープ 187

め

メソッド

length 138
 push() 173
 splice 174

索引 — 一般

B

- Bézier 曲線 50, 97
- Bézier 曲線
 - 円を近似 55
 - 制御点の—— 51

C

- click(イベント) 139, 146, 177
- clientY(イベント) 154
- sin x (余弦関数) 158
- CSS 103, 117

D

- Distant Light 121
- DOM 126

E

- evt 変数 136

H

- HTML 4
- HTML5 5

J

- JavaScript 125
- オブジェクトリテラル 180

M

- mousedown(イベント) 146, 148, 150, 154
- mousemove(イベント) 146, 148, 150, 151, 154
- mouseup(イベント) 146, 148, 154

O

- onbegin(イベント) 134
- onchange(イベント) 134
- onclick(イベント) 134, 181
- onend(イベント) 134
- onload(イベント) 134, 139, 189
- onmousedown(イベント) 134
- onmousemove(イベント) 134
- onmouseout(イベント) 134
- onmouseover(イベント) 134
- onmouseup(イベント) 134

P

- PostScript 51

S

- sin x (正弦関数) 158
- 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

- イベント 132
 - click 139, 146, 177
 - clientY 154
 - mousedown 146, 148, 150, 154
 - mousemove 146, 148, 150, 151, 154
 - mouseup 146, 148, 154
 - onbegin 134
 - onchange 134
 - onclick 134, 181
 - onend 134
 - onload 134, 139, 189
 - onmousedown 134
 - onmousemove 134
 - onmouseout 134
 - onmouseover 134
 - onmouseup 134
- クリック 132
- イベントキャプチャリング 145
- イベントドリップ 133
- イベントバーリング 138, 145
- 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
- オブジェクトリテラル
 - JavaScript における—— 180
- 親ノード 126
- 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
- クリック 132
- クリックの処理 (JavaScript) 134
- クレイク・オブライエン効果 35
- クロージャ 188

こ

- 子ノード 126
- コフカリング 29
- コメント (SVG) 11
- 子要素 14

さ

- サイクロイド 156, 169
- 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
 - シェパードの—— 42
 - ジャッドの—— 76
 - 主観的輪郭のネオン輝度現象 50
 - ツェルナーの—— 163

デルブーフの—— 25

ネットレスの糸 166

バーゲンのきらめき効果 103

バインジオ・ピンナの—— 166

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, 156

浮動する円 65

ブルドンの—— 43

ヘルマン格子 20, 103

ポッケンドルフの—— 19, 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

正弦関数 158

線形グラデーション 26

そ

属性 13

—— の円 22

—— の橢円 22

属性値 13

た

橢円 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

つ

ツェルナーの錯視 163

て

テキストエディタ 7

- テキストノード 127, 144
 デルブーフの錯視 25
 点光源 121
- と**
 ドラッグの処理 (JavaScript) 146
- な**
 名前空間 13, 131, 152, 154
- ね**
 ネックレスの糸 166
- の**
 ノード 126
 親—— 126
 子—— 126
 テキスト—— 127
 要素—— 127
 ルート—— 127
- は**
 バーゲンのきらめき効果 103
 バインジオ・ピンナの錯視 166
- 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, 156
 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 131
- へ**
 ヘルマン格子 20, 103
- ほ**
 放射グラデーション 33
 ポッケンドルフの錯視 19, 86
- ま**
 マウスイベントのプロパティとメソッド 135
 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 131
 メッセージボックス 137

- も**
 モーガンのねじれひも 64
 文字列
 —— をつなげる 137
 —— 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 127
 余弦関数 158
- る**
 ルートノード 127
 ルート要素 13