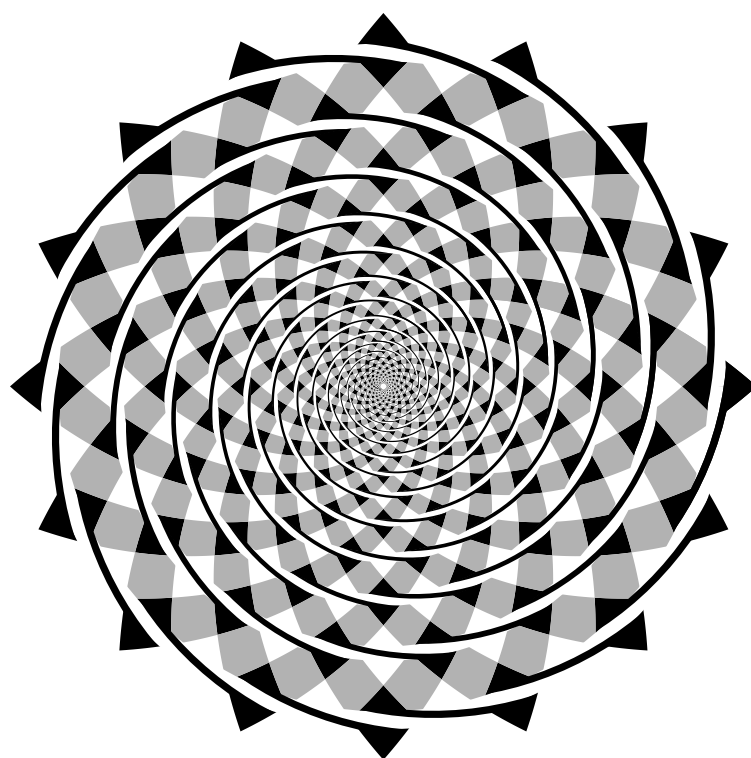


錯視図形を描こう
平成30年3月29日改訂



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 – 2018 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 |
| 3.3 SVG パターン | 45 |
| 第 4 章 アニメーション | 51 |
| 4.1 アニメーションにおける属性 | 51 |
| 4.2 位置を動かす (<animateTransform> 要素) | 52 |
| 4.3 道のりに沿ったアニメーション (<animateMotion> 要素) | 55 |
| 4.4 いろいろな属性に動きをつける | 57 |
| 4.4.1 一般の属性に変化をつける (<animate> 要素) | 57 |
| 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 | 60 |

| | | |
|--------------|-----------------------------------|--------------|
| 4.5 | 複数の値を指定する (keyTimes, values) | 60 |
| 4.6 | アニメーションがついた錯視図形 | 61 |
| 第 5 章 | 文字列の表示 | 65 |
| 5.1 | 文字列表示の基礎 | 65 |
| 5.2 | 部分的に文字の表示を変える方法 | 66 |
| 5.3 | 道程に沿った文字列の配置 | 68 |
| 第 6 章 | JavaScript を利用した SVG 図形の生成 | 71 |
| 6.1 | インタラクティブな SVG を作成するための準備 | 71 |
| 6.1.1 | JavaScript | 71 |
| 6.1.2 | DOM(Document Object Model) | 72 |
| 6.1.3 | DOM のメソッドとプロパティ | 77 |
| 6.2 | イベント | 78 |
| 6.2.1 | イベント概説 | 78 |
| 6.2.2 | SVG 文書や HTML における代表的なイベント | 79 |
| 6.3 | JavaScript による SVG 文書のマウスイベントの処理 | 80 |
| 6.3.1 | マウスに関するイベント処理 | 80 |
| 6.3.2 | イベントの処理の詳細 | 90 |
| 6.3.3 | マウスのドラッグを処理する | 92 |
| 6.3.4 | オブジェクトを追加する | 94 |
| 6.4 | 起動時に JavaScript で要素を作成する | 98 |
| 6.4.1 | 任意の形の曲線を描く | 98 |
| 6.4.2 | SVG のオブジェクトを操作するための関数 | 100 |
| 6.4.3 | ツェルナーの錯視図形 | 103 |
| 6.4.4 | 一定時間経過後に関数をよびだす (自前でアニメーションを作成する) | 107 |
| 6.5 | HTML 文書とそこにある SVG 要素の間でデータ交換する | 114 |
| 付 録 A | SVG で利用できる色名 | 付録 1 |
| 付 録 B | 関連図書 | 付録 5 |
| 付 録 C | CSS について | 付録 7 |
| 索引 | | 付録 10 |
| | SVG の用語 | 付録 10 |
| | HTML の用語 | 付録 15 |
| | JavaScript の用語 | 付録 17 |
| | 一般 | 付録 20 |

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。

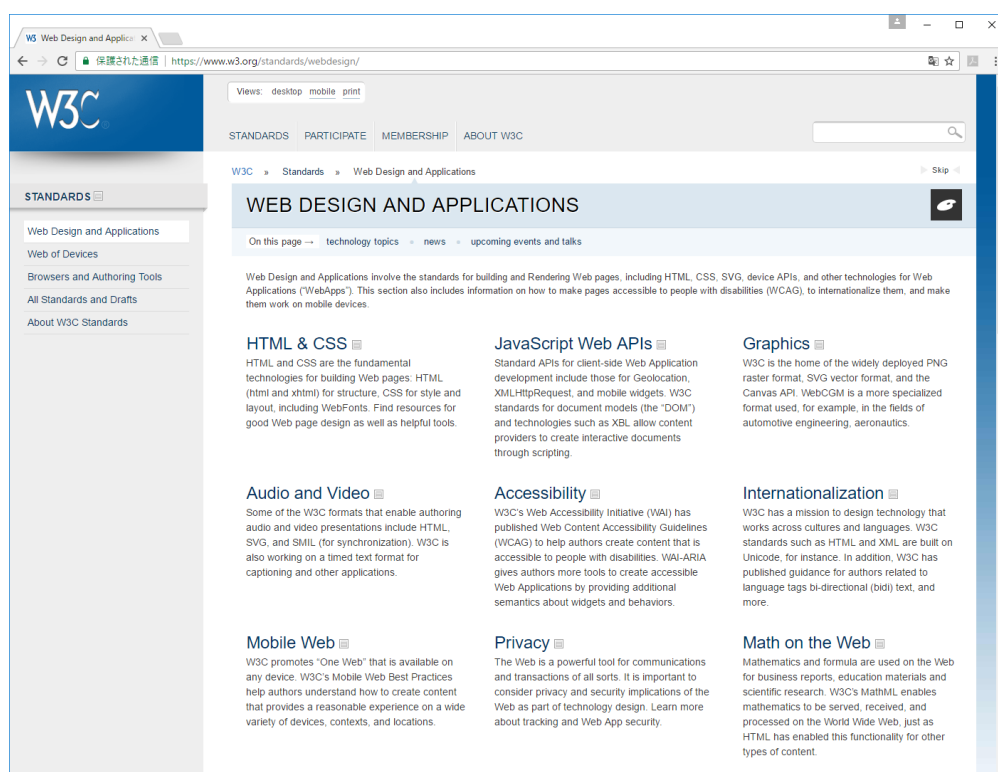


図 1.1: World Wide Web Consortium の Web Design and Application のページ (2017/2/27 参照)

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 (または長さや方向)、線の幅、色の情報などをそのまま持ちます。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。

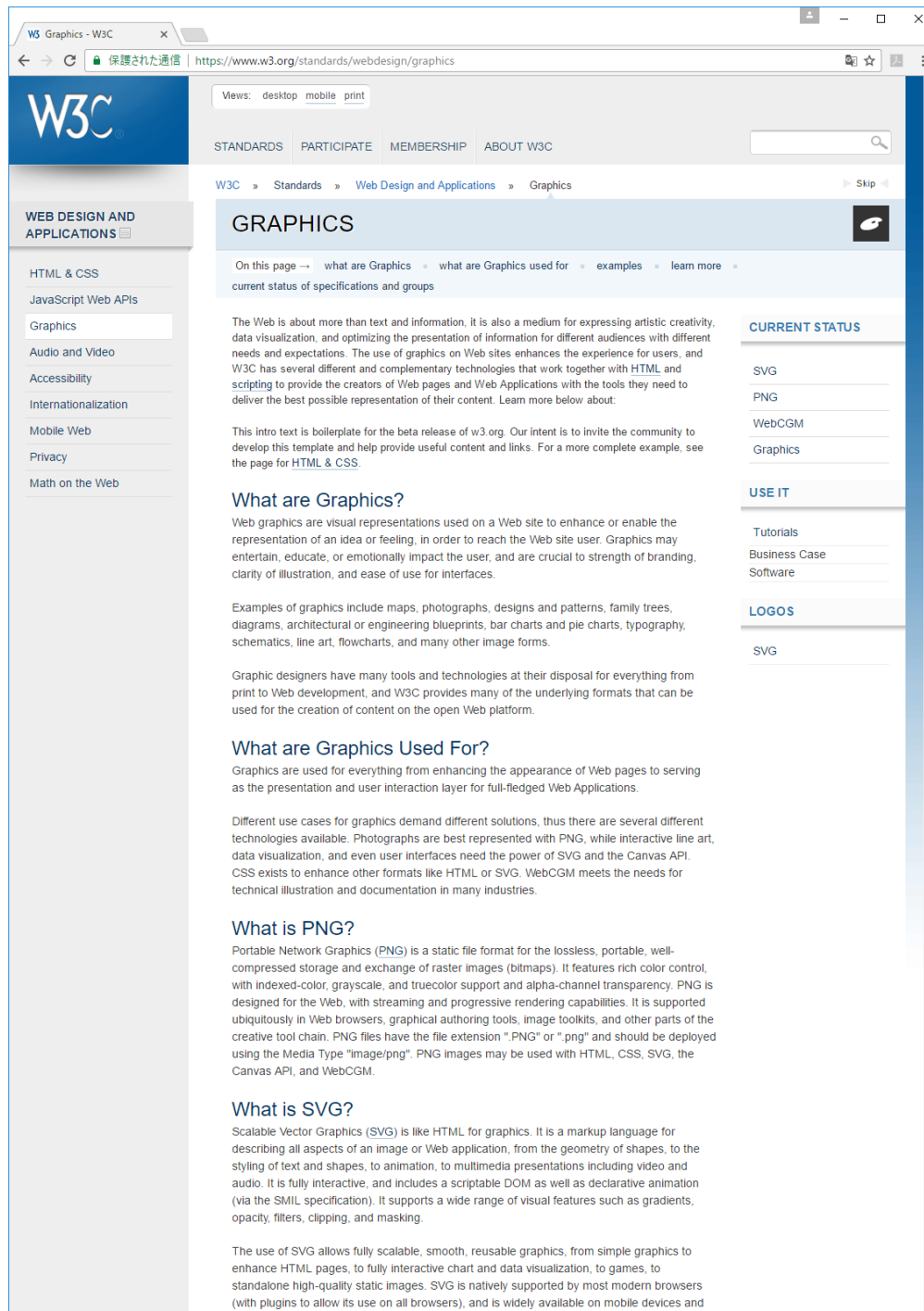


図 1.2: W3C の SVG に関するトップページ (2017/2/27 参照)



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行います。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ます。ユーザが勝手にタグを定義することができません。一方、XML では extensible という用語が示すようにタグは自由に定義できます。したがって、XML を用いることでいろいろな情報を構造化して記述することが可能になっています。XML についてもっと知りたい場合には文献 [10]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可能です。

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[19] 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ビットマップ方式で、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ため、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ます。また、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。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おきません。
 - － 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認しましょう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。

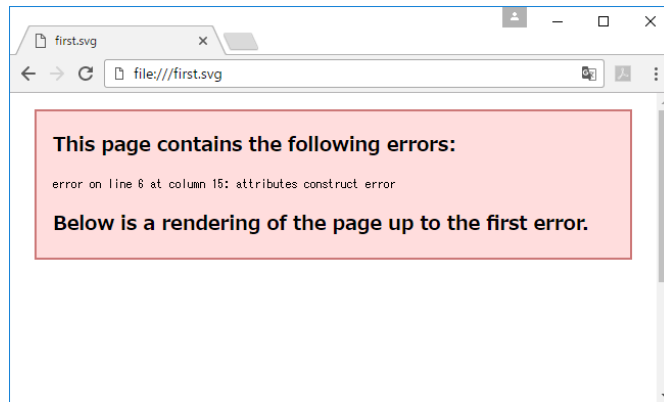


図 2.1: Google Chrome におけるエラーメッセージ

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)。

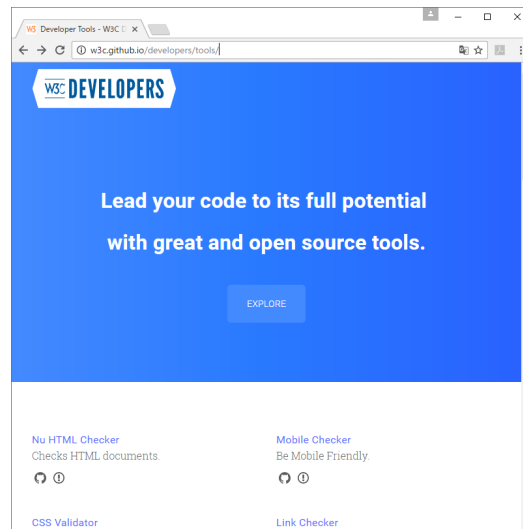


図 2.2: Validator のトップ画面

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。

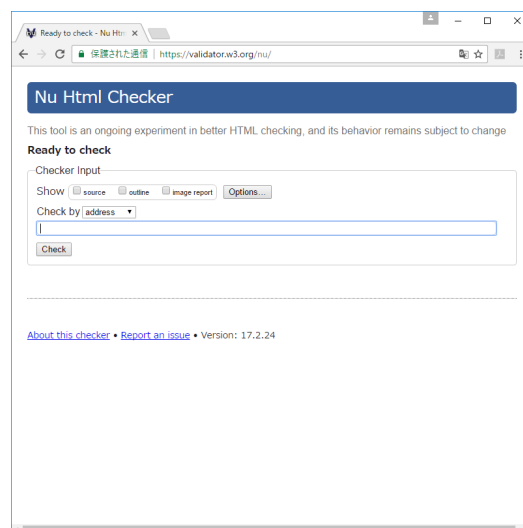


図 2.3: Nu Html Checker の画面

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。

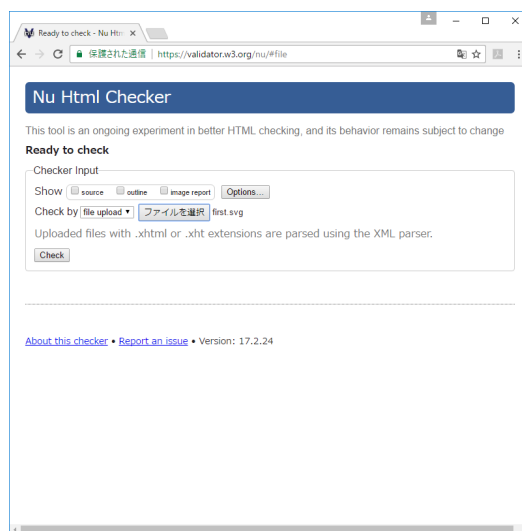


図 2.4: ファイルのアップロード

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)。

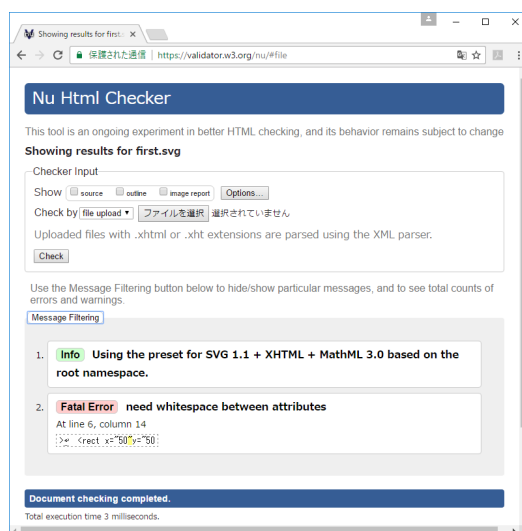


図 2.5: 検証結果の表示画面

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)。

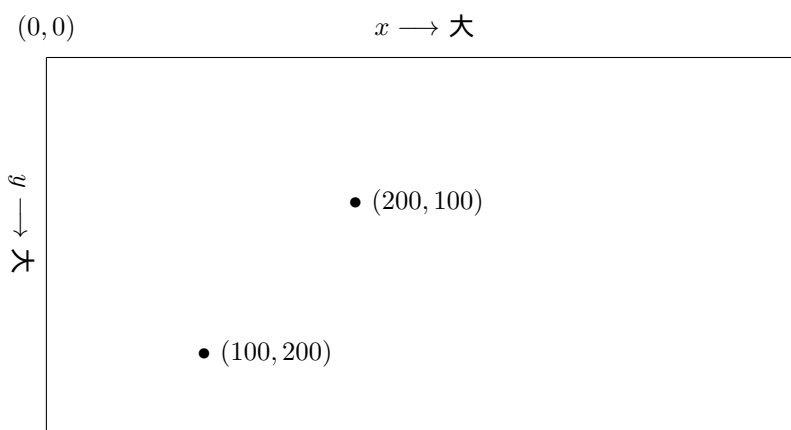


図 2.6: SVG の座標系

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、回転させることが可能です。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 `-->` で囲みます。このテキスト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 をご覧ください。

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 $0 \sim 255$ の値を指定します。なお、rgb 関数は $0 \sim 255$ 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。

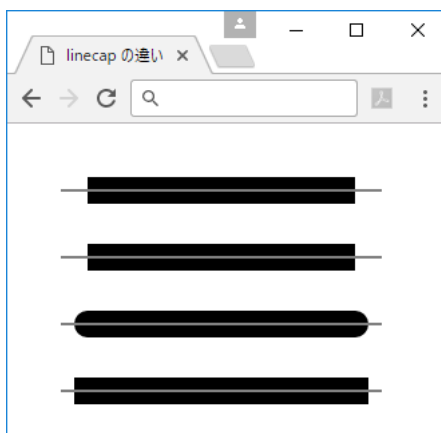


図 2.7: 直線と端の指定 stroke-linecap の違い

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

表 2.1: transform の属性値

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

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

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

- 7行目から13行目で `stroke-linecap` で指定される直線の両端の形が異なるものを4つ定義しています。
 - `<line>` 要素は直線を定義します。この要素は`<g>` 要素の内側にあるので`<line>` 要素は`<g>` 要素の子要素であるといい、`<g>` 要素は`<line>` 要素の親要素であるといいます。
 - 属性は表 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 x 座標 | 数値 |
| <code>y1</code> | 開始位置の y 座標 | 数値 |
| <code>x2</code> | 終了位置の x 座標 | 数値 |
| <code>y2</code> | 終了位置の y 座標 | 数値 |
| <code>stroke</code> | 直線を塗る色 | 色名または rgb 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 | <div>butt (default) 何もつけ加えない</div> <div>round 半円形にする</div> <div>square 長方形にする</div> |

- 最初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。
- 最初の二つが同じ形なので `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行目では最初の直線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 [21, 61 ページ] です (図 2.8)。

水平と垂直の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錯視図形です。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。

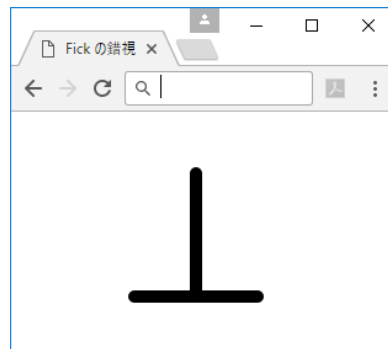


図 2.8: フィックの錯視

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>
```

- 線分の長さが直感的にわかるために<g> 要素で座標系を平行移動しています (6 行目)。
- 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ラー・ライヤーの錯視 [21, 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。

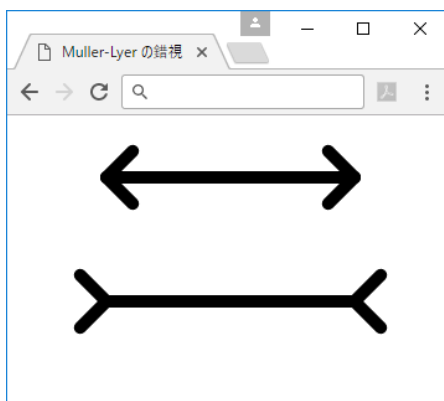


図 2.9: ミューラー・ライヤー錯視

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雛形を定義することにします。雛形は<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し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 行目の間が上の図形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 (22 行目の transform="translate(200,0)"), 23 行目で transform="scale(-1,1)" で矢印の向きを反転させています。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>

```

- 左の矢印は向きが上のものと逆なので図形を左右反転する属性を持つ32行目の<g> 要素の子要素にして実現しています。
- 右の矢印は35行目の平行移動をする属性を持つ<g> 要素の子要素とすることで図形を平行移動しています。

問題 2.3 ミューラー・ライヤーの錯視図形で次のことを調べなさい。

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

2.4 長方形

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

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 [21, 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。

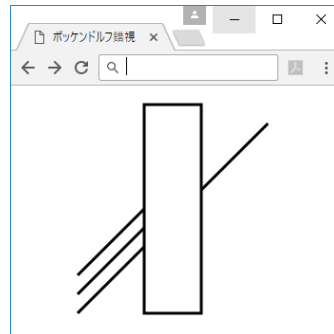


図 2.10: ポッケンドルフ錯視

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。

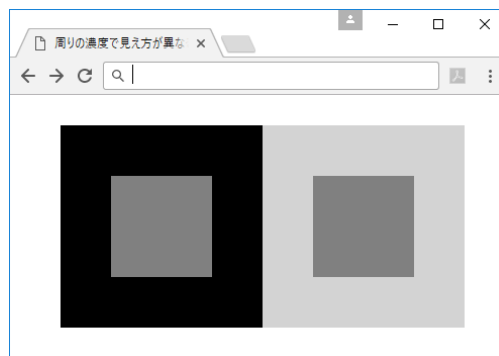


図 2.11: 周りの濃度で見え方が異なる

SVG リスト 2.6: 周りの濃度で見え方が異なる

```

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

```

この図形は左の部分は二つの正方形を重ねて描いています。右の部分は縁取りの幅を大きくして同じような大きさの図形になる用の表示位置や長方形の大きさを調整しています。同じ図形を描くので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[24, カラー図版 3]。

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

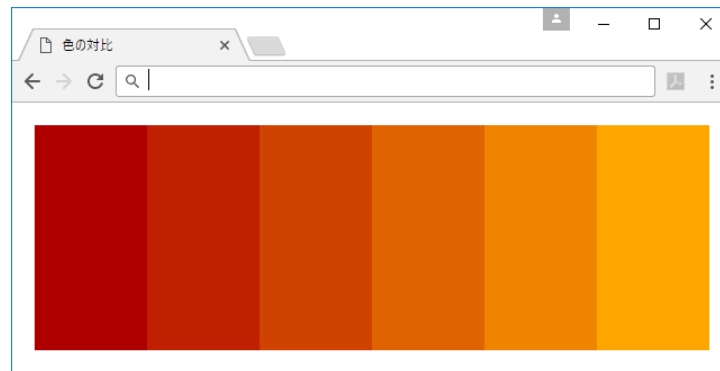


図 2.12: 色の対比

ヘルマン格子 図 2.13 は白い線の交差している位置に灰色のちらつきが表れるというヘルマン格子 [21, 180 ページ] と呼ばれるものです。

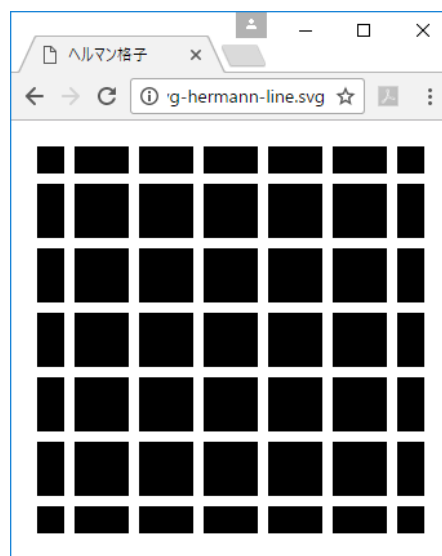


図 2.13: ヘルマン格子

SVG リスト 2.7: ヘルマン格子

```

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

```

```

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

```

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

問題 2.5 リスト 2.7 について次の事柄を検討しなさい。

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

問題 2.6 図 2.14 はカフェウォール錯視 [21, 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 円と楕円

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

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

³水平線が黒の場合にはミュンスターベルグ錯視と呼ばれます。

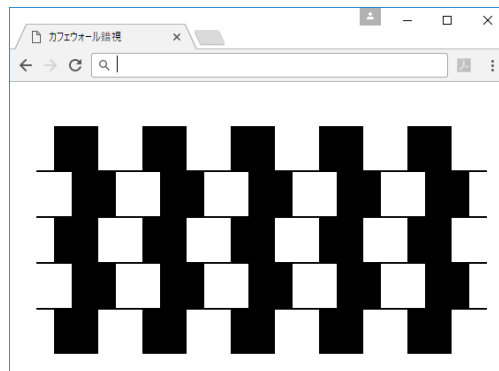


図 2.14: カフェウォール錯視

表 2.4: 円と楕円の属性

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

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

SVG リスト 2.8: 周りの大きさで見え方が変わる

```

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

```

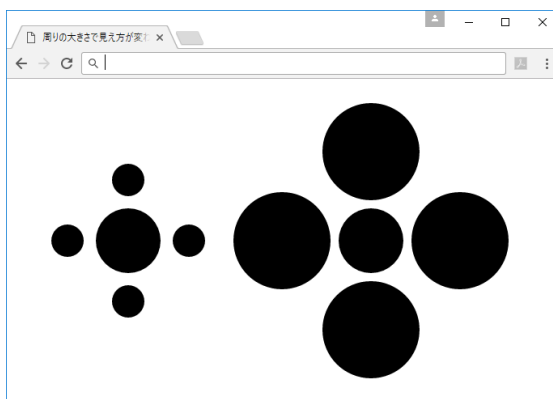


図 2.15: 周りの大きさで見え方が変わる

```

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

```

- 7 行目で中央にある円を定義し、それに CCircle という id を付けています。
- 8 行目で右側の周りに置く円のひとつを定義し、それに LCircle という id を付けています。
- 9 行目で左側に描く円を定義しています。この円には LCircle という id を付けています。
- 11 行目から25 行目で左側の図形を定義しています。(0, 0) の位置に中央の円を描き、その周りに SCircle で参照する小さな円を 4 つ付けています。小さな円を描く方法として16 行目

で `transform` の値を `rotate(90),translate(0,75)` と二つ指定しています。これは次のように記述したものと同じです。

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

- 右側の図形も26行目から40行目で同様に描いています。

問題 2.7 リスト 2.15 において次のことをしなさい。

1. `<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。
2. 中心の円や周りの円の色を変えたときに見え方が変わるかどうか調べなさい。
3. 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 [21, 59 ページ] と呼ばれています。この図は [24, 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。

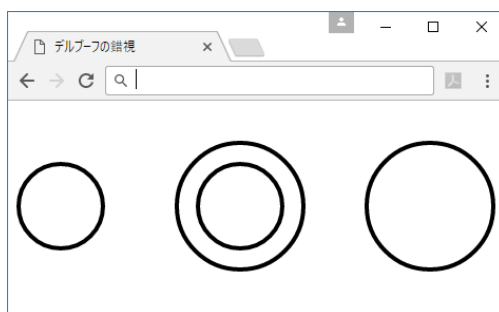


図 2.16: デルブーフの錯視

2.6 グラデーション

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

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

2.6.1 線形グラデーション

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

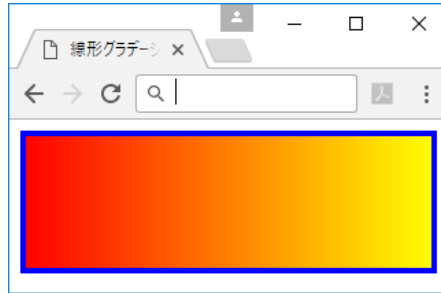


図 2.17: 線形グラデーション

SVG リスト 2.9: 線形グラデーション

```

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

```

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

- 8行目と9行目にグラデーションの特定の位置に対する場所 (offset) とそこでの色 (stop-color) を<stop> 要素で定義しています。ここでは開始位置 (offset="0%") の色を赤に、終了位置 (offset="100%") の色を黄色に指定しています。
したがって、このグラデーションは左の赤から右に行くにしたがい黄色へと変化することになります。なお、<stop> 要素は途中の値も指定できるので2つより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 [21, 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。

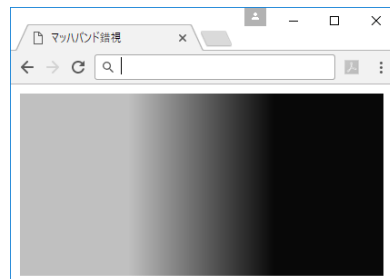


図 2.18: マッハバンド錯視

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">
```

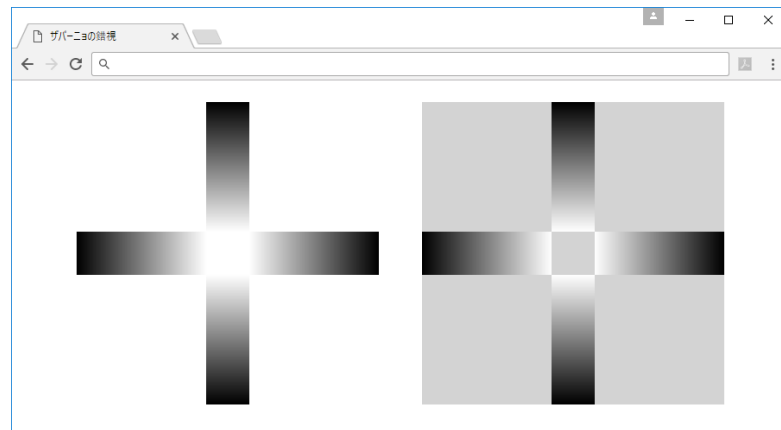


図 2.19: ザバーニョの錯視

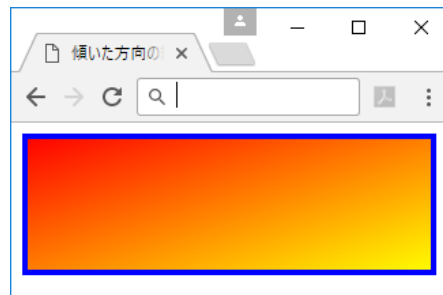


図 2.20: 傾いた方向の線形グラデーション

```

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. x_1 、 y_1 、 x_2 と y_2 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 [21, 90 ページ] です。背景の明度の差がある部分を中央のグラデーションの部分が隠れているので、同じ色の円の縁取りが明るさの違う色に見えます。縁取りと同じ色でない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。

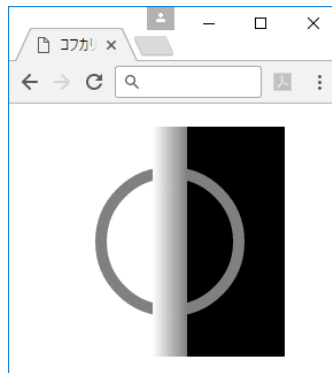


図 2.21: コフカリング

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 ([24, 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れ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%" />

```

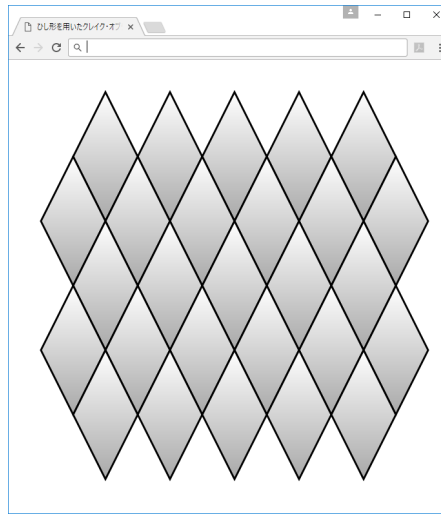


図 2.22: ひし形を用いたクレイク・オブライエン効果

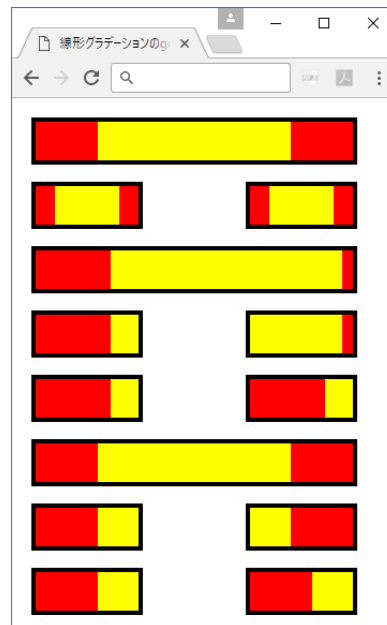


図 2.23: 線形グラデーションの gradientUnits の違い

```

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>

```

- 色の变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变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变化位置は同じです。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变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变化がつく放射グラデーションを表す<radialGradient> 要素があります。放射グラデーションには次の属性があります。

表 2.5: 放射グラデーションの属性

| 属性名 | 意味 | 値 |
|-----|-----------------------------|---------|
| cx | 放射グラデーションの終了位置の円の中心の x 座標 | 数値または割合 |
| cy | 放射グラデーションの終了位置の円の中心の y 座標 | 数値または割合 |
| r | 放射グラデーションの終了位置の円の半径 | 数値または割合 |
| fx | 放射グラデーションの中心の x 座標 | 数値または割合 |
| fy | 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%">
5   <title>放射グラデーション (userSpaceOnUse の場合)</title>

```

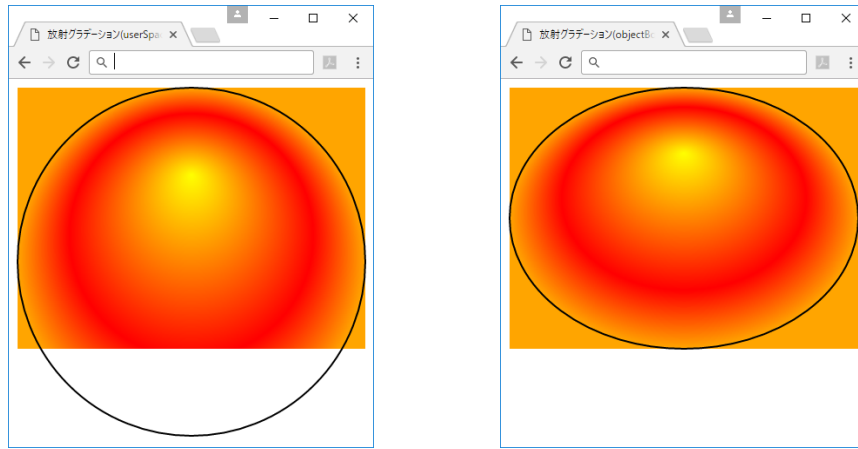


図 2.24: 放射グラデーション (userSpaceOnUse(左) と objectBoundingBox(右))

```

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ーションで塗っています。終了位置の円の外側は最後の色をそのまま使うのがデフォルトです。
- 放射グラデーションの端を示すために16 行目から17 行目 で円の縁を描いています (fill の値が none になっていることに注意すること)。

リスト 2.12 は図 2.24 の右側の図のものです。

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

```

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

```

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

問題 2.15 図 2.25 はクレイク・オブライエン効果とよばれるものです。内側にある境界部分の黒が内部の白を外部よりより白く見えさせています。放射グラデ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な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、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と呼ばれることがあります。

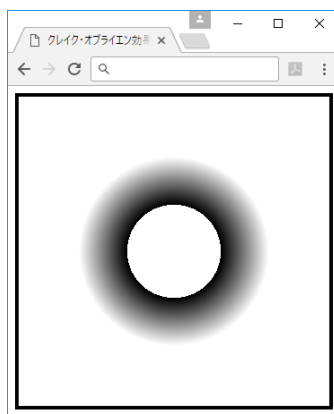


図 2.25: クレイク・オブライエン効果

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 stroke に設定される。 |
| 図形一般 | fill-opacity | 図形の属性 fill 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"

```

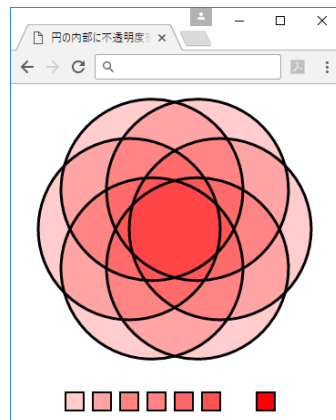



図 2.26: 円の内部に不透明度を設定した例

```

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雛形を作成します (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

$$\text{opacityが定義された図形の色} \times \text{この図形の色} + (1 - \text{この図形の opacity}) \times \text{この図形の下にある色}$$

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 に関して次の問いに答えなさい。

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

問題 2.17 図 2.27 はピラミッドの稜線とよばれています ([24, 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。これらの要素の点の位置は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。

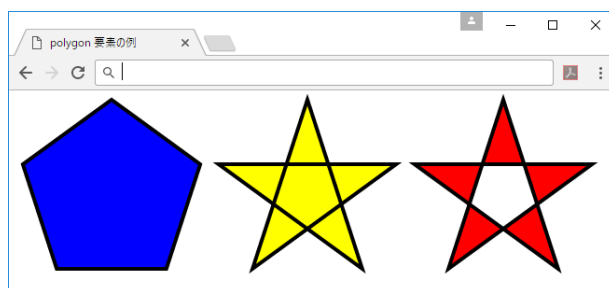


図 3.1: <polygon> 要素の例

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y座標の上方が正となるように直しています。
- 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、点の座標は前もって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、 $(100 \cos(90 + 72)^\circ, 100 \sin(90 + 72)^\circ)$ などの式で計算できます。

- 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。



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

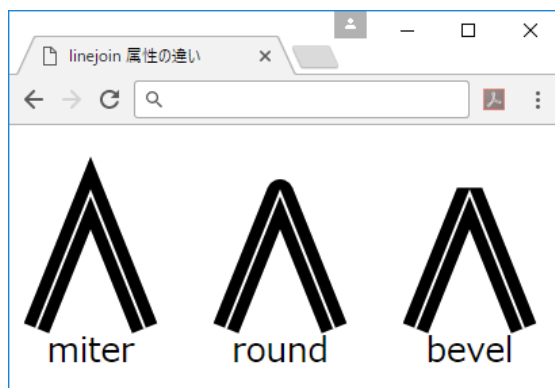


図 3.3: linejoin 属性の違い

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。

- 線分がつなが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ことです。
- その他の値として頂点を丸くする `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が、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"/>

```

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。

```

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 はシェパードの錯視 [21, 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。

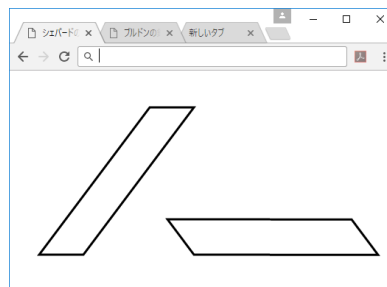


図 3.4: シェパードの錯視

問題 3.4 図 3.5 はブルドンの錯視 [21, 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。

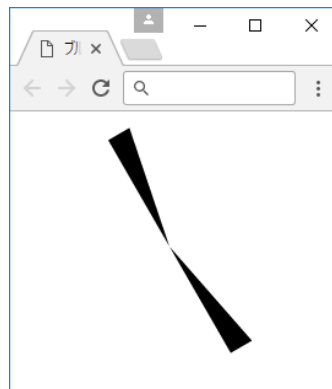


図 3.5: ブルドンの錯視

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。

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。 |

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。

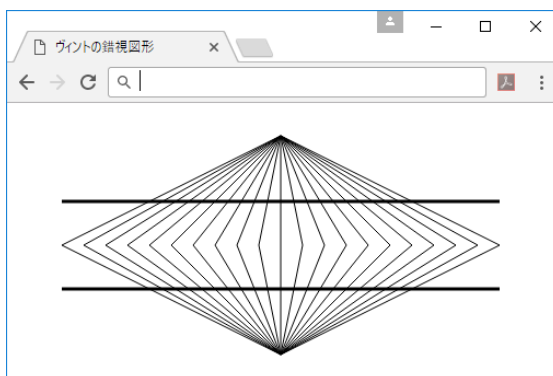


図 3.6: ヴィントの錯視図形

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行目で上部の一点から放射状に広がって下部の一点へ集まる図形を<path> 要素を用いて作成しています (この図形は<polyline> 要素で描くことも可能です)。
- <path> 要素の形状を示すための属性は d です。
- 点の座標はすでに見てきた<polyline> 要素や<polygon> 要素と同様の方法で記述します。
- 7行目の d の先頭にある M で開始点への移動を指示しています。この位置は上部の線分が集中している場所の位置 (0, -100) です。
 - 次は中央部の一番左の位置 (-200, 0) です。M などの指定がありません。この場合には L を指定したものとみなされます。
 - 次は下部の線分が集中している位置 (0, 100) です。
 - 次は中央部の右端の位置 (0, 200) です。
 - 次は上部の線分が集中している位置に戻っています。

– その後は x 座標の位置を少しずつ中央に近づくように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 `<path>` 要素で描いています。この属性 `d` の値に `M` が 2 回現れている (13 行目) のでこの前後で直線が繋がらないので、2 本の平行線をひとつの `<path>` 要素で定義できます。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. 放射状の線の位置や水平線の間隔を変えたときの図形の見え方に变化があるか確かめなさい。
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 ([23, 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。

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ので面倒です。プログラムから点の位置を指定する方法については第 6 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次のようになります。

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

初めの移動先の位置を一度だけ `l` で移動を定義しておくとも残りの移動量の指定になります。

問題 3.6 正方形を次のように最後の `z` を省略すると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)。

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

3.3 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 3.4: 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 3.7 図 3.7 はザヴィニの錯視 [24, 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大きさが異なる別の長方形に見えます。この図を描きなさい。

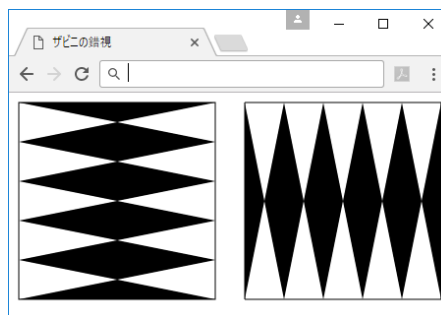


図 3.7: ザヴィニの錯視

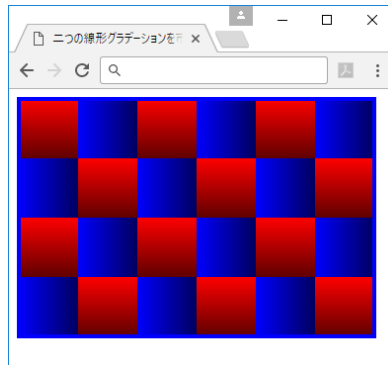


図 3.8: 二つの線形グラデーションを市松模様に並べた例

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 3.8 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 3.5: 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" >
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行目は上から下へ向かう線形グラデーションを、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 3.8 図 3.9 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 [21, 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。

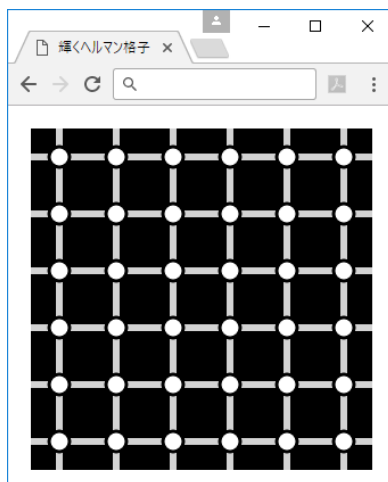


図 3.9: 輝くヘルマン格子

問題 3.9 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 3.10 図 3.10 はモーガンのねじれひも [21, 76 ページ] とよばれる錯視図形です。これを作成しなさい。

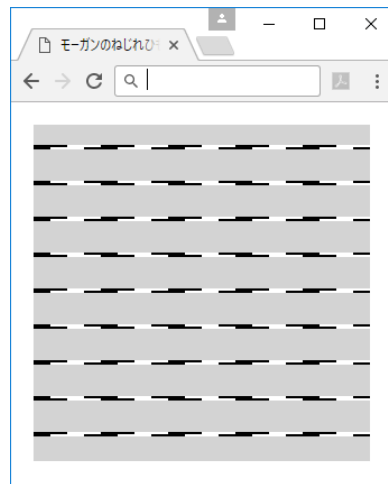


図 3.10: モーガンのねじれひも

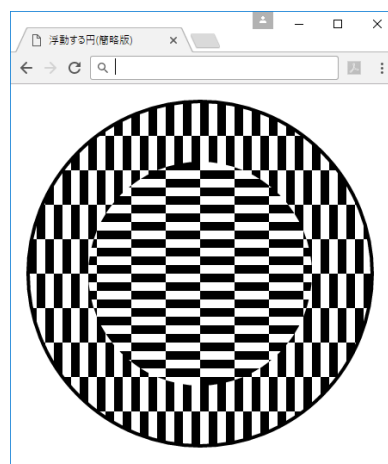


図 3.11: 浮動する円 (簡略版)

パターンを塗りつぶす図形は長方形でなくともかまいません。

図 3.11 は大内元が作成した錯視図形の内部のパターンを簡略化した図形です ([24, 74 ページ図 7.5])。²中央の円がページの画面から浮き上がって揺れているように見えます。

SVG リスト 3.6: 浮動する円

```

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>浮動する円 (簡略版)</title>

```

²[11, 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。

```

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。
- パターンは全体を白で塗りつぶし長方形(8行目)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。
- この円の上に同じパターンで塗りつぶされた小さな円(16行目)を90°回転(15行目)して描いています。

問題 3.11 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 3.12 図 3.12 は図 3.8 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。<pattern> 要素に属性 patternTransform を用いてこの図を作成しなさい。

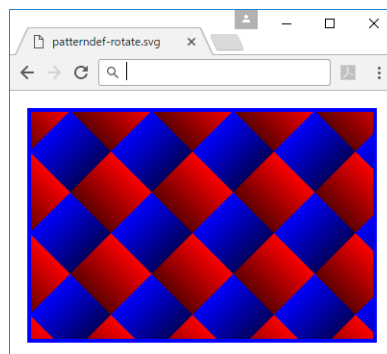


図 3.12: 図 3.8 のグラデーションのパターンを回転させる

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 (<animateMotion> 要素 のデフォルト) spline 3 次の Bézier で値を補間 |

4.2 位置を動かす (<animateTransform> 要素)

平行移動 グループ化されたオブジェクト<g> 要素は属性 transform で位置の移動ができました。

<animateTransform> 要素を用いると transform に対してアニメーションがつきます。

次の例は二つの長方形をグループ化し、それに平行移動のアニメーションをつけています。

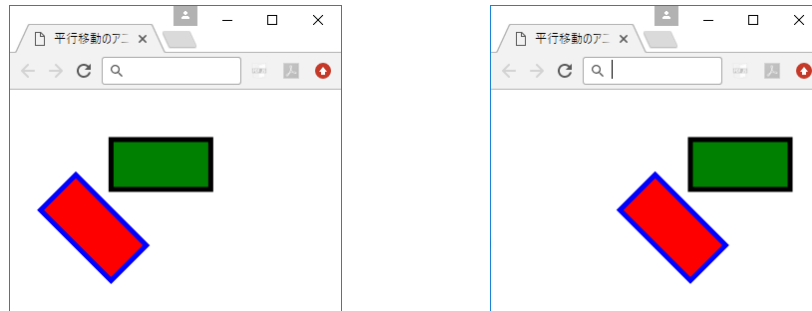


図 4.1: 平行移動のアニメーション (開始時-左-と終了時-右-)

SVG リスト 4.1: 図形の平行移動のアニメーション

```

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

```

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

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

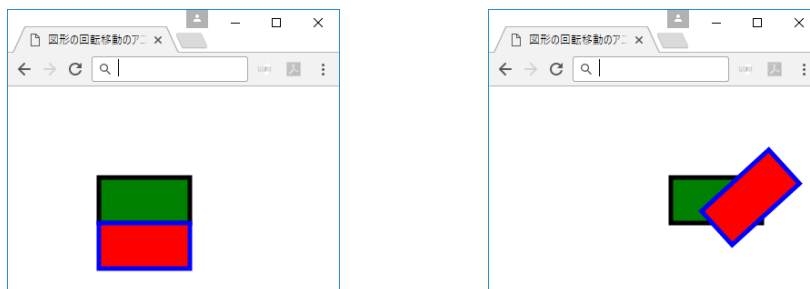


図 4.2: 図形の回転のアニメーション – 開始時 (左) と平行移動終了時 (右)

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 行目が変わっています。
- 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 [21, 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>

```

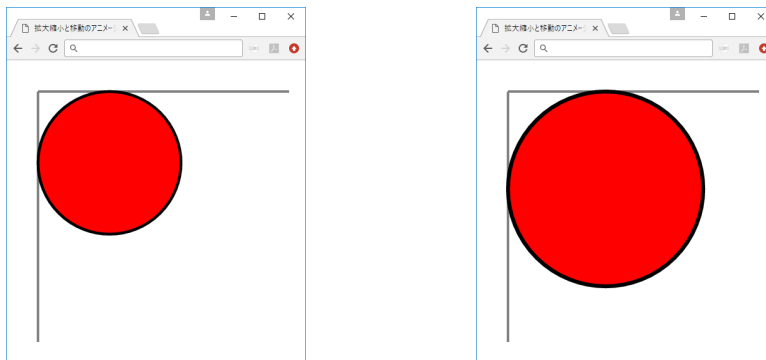


図 4.4: 拡大縮小と移動のアニメーション

```

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 (<animateMotion> 要素)

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

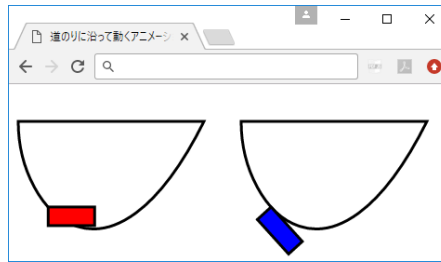


図 4.5: 道のりに沿って動くアニメーション

SVG リスト 4.4: 道のりに沿ったアニメーション

```

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

```

- 7行目から8行目でアニメーションで動くパスを定義しています。このパスは動きがわかるように表示にも使われています(13行目と21行目)。
- 14行目から18行目ではアニメーションで動く左側の長方形を定義しています。この長方形の内部を塗る fill が currentColor であることに注意してください。これは上位の環境で定義されている色で塗ることを意味します。ここでは12行目にある<g>要素の属性 color の値 (red) が利用されます。

- 15 行目から 17 行目でこの長方形にアニメーションをつけています。<defs> 要素のなかで定義された道のりを参照するために<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。

色の变化 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。

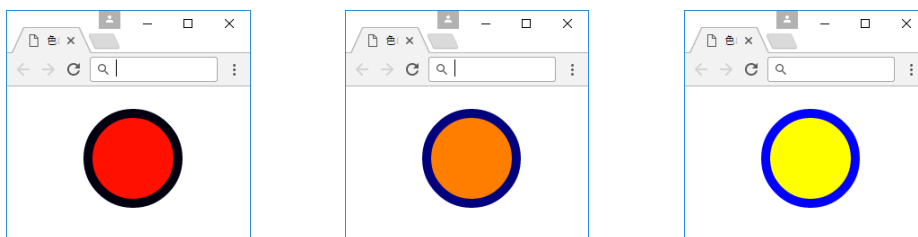


図 4.6: 色のアニメーション (開始時-左-, 途中-中央-, 終了時-右-)

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。

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 (x_1 や y_1) や終了位置 (x_2 や y_2) 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 4.7)。

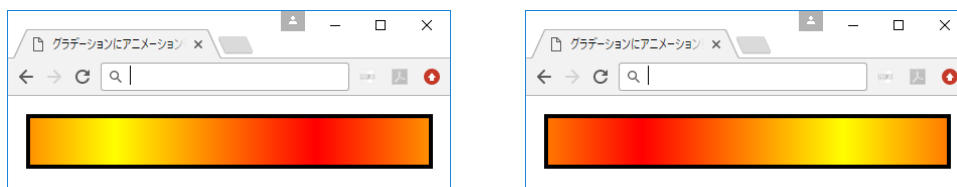


図 4.7: グラデーションにアニメーションを付ける

SVG リスト 4.7: 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"
4      height="100%" width="100%">
5      <title>グラデーションにアニメーションを付ける</title>
6      <defs>
7          <linearGradient id="Gradiation1" gradientUnits="userSpaceOnUse"
8              x1="0" y1="0" x2="800" y2="0">
9              <stop stop-color="yellow" offset="0%" />
10             <stop stop-color="red" offset="25%" />
11             <stop stop-color="yellow" offset="50%" />
12             <stop stop-color="red" offset="75%" />
13             <stop stop-color="yellow" offset="100%" />
14             <animate attributeName="x1" attributeType="XML"
15                 from="0" to="-400" dur="5s" repeatCount="indefinite" />
16             <animate attributeName="x2" attributeType="XML"
17                 from="800" to="400" dur="5s" repeatCount="indefinite" />
18         </linearGradient>
19     </defs>
20     <g transform="translate(20,20)">
21         <rect x="0" y="0" width="400" height="50"
22             stroke="black" stroke-width="4" fill="url(#Gradiation1)" />
23     </g>
24 </svg>

```

- 7行目から18行目でアニメーションを伴った線形グラディエーションを定義しています。
 - 線形グラデーションの開始位置を変化させるので `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いています (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 4.8 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。

問題 4.9 リスト 4.7 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ることができると検討しなさい。

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。

SVG リスト 4.8: 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 4.9: 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 4.10 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。

4.6 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 4.11 図 4.8 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。

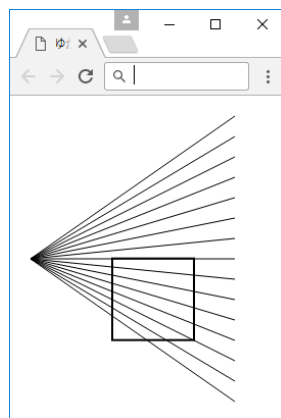


図 4.8: ゆがんだ正方形

追いかっこをする長方形 図 4.9 は背景が黒と明るい灰色 (lightgrey) の同じ幅の縦じまに塗られた背景上を明るい灰色、灰色と黒に塗られた小さな長方形が等速度で移動するものです。

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

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

図 4.10 では背景の黒の部分を変えています。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。

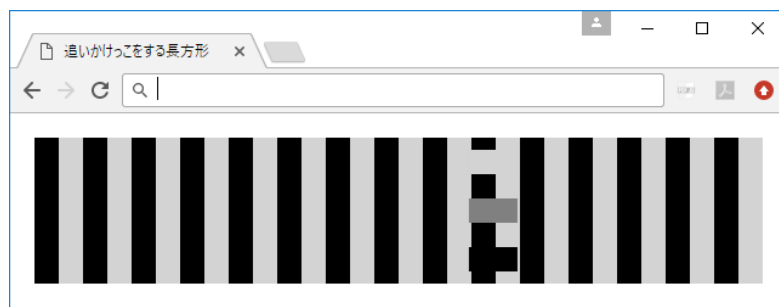


図 4.9: 追いかけてくる長方形 (その 1)

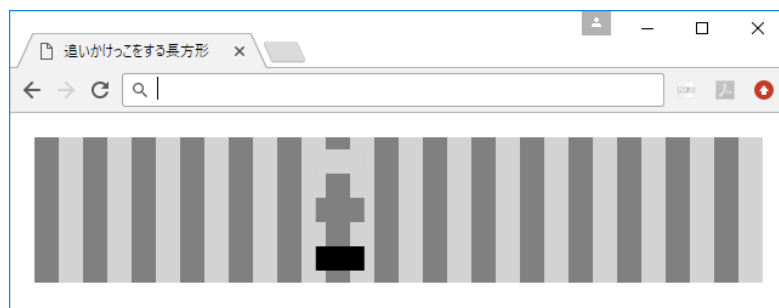


図 4.10: 追いかけてくる長方形 (その 2)

図 4.11 では背景の黒の部分をもるい灰色に変えています。
一番上の長方形は見えなくなり、残りの二つの長方形は等速度で移動するように見えます。
リスト 4.10 はこれら 3 つの図を順番に表示するものです。

SVG リスト 4.10: 追いかけてくる長方形

```

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>追いかけてくる長方形</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"/>

```

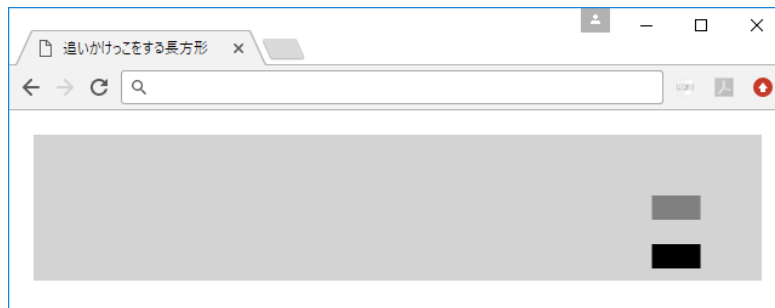


図 4.11: 追いかけてくる長方形 (その 3)

```

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 4.12 リスト 4.10 について次のことについて調べなさい。

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

第5章 文字列の表示

5.1 文字列表示の基礎

図 5.1 は SVG 画像の中に文字列を表示しています。
文字列を表示するには<text> 要素を用います。

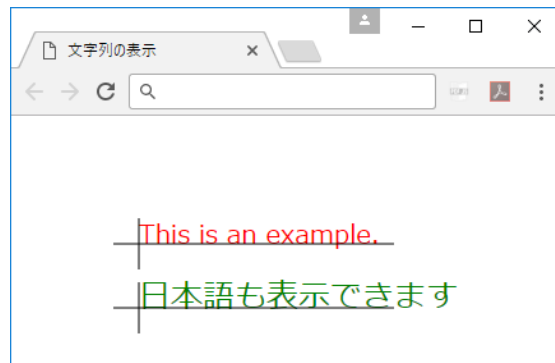


図 5.1: 文字列の表示

SVG リスト 5.1: 文字列の表示

```

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

```

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

```

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

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

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

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

5.2 部分的に文字の表示を変える方法

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

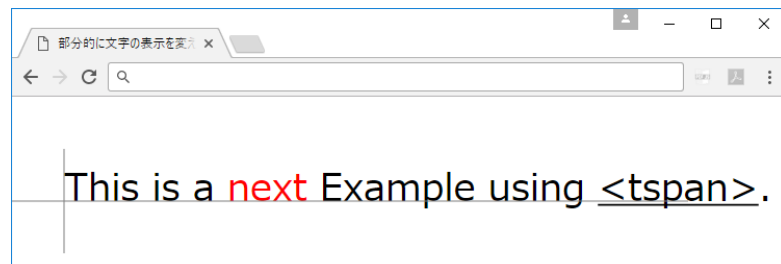


図 5.2: 部分的に文字の表示を変える

表 5.1: 文字列表示の属性と属性値

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

SVG リスト 5.2: <tspan> 要素の使用例

```

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

```

- 8 行目では next の文字を<tspan> 要素 ではさんで色を赤に変えています。
- 10 行目では<tspan> 要素を用いて文字列の下線をつけています。なお、<や>を表示するためにはここでは< 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 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。

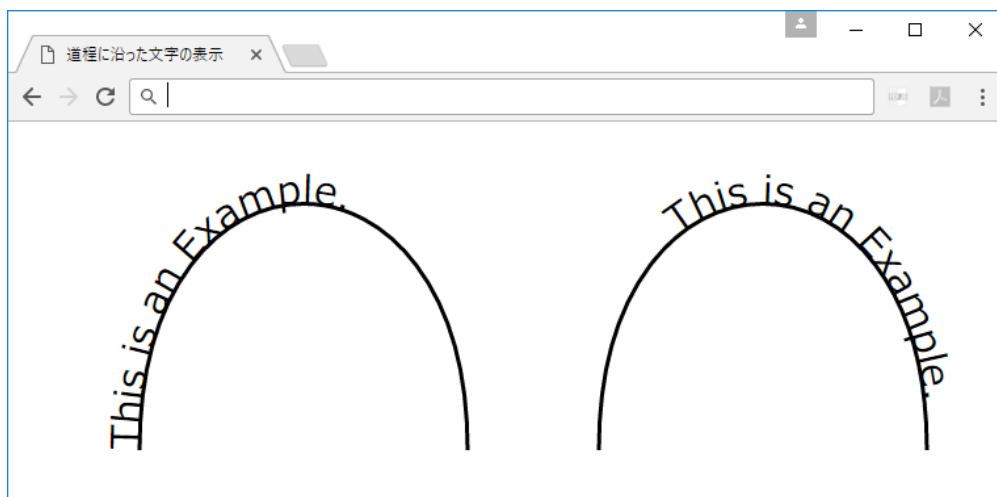


図 5.3: 道程に沿った文字の表示

SVG リスト 5.3: 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 5.2 閉じた道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。

第6章 JavaScript を利用した SVG 図形の生成

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

なお、この章におけるプログラミングスタイルが章の始めと終わりの方で異なっています。JavaScript でより良いプログラミングをするためには気を付けなければならないことが多くあります。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。

6.1 インターラクティブな 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。

6.1.1 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 [5] 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` という方法であ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。

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

JavaScript はインタープリター型の言語です。言語を解釈しながら実行していきますので途中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、一般のブラウザではショートカットキーとして「Ctrl+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。

6.1.2 DOM(Document Object Model)

W3C の DOM に関するページ [13] 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る。

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と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 6.1 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。

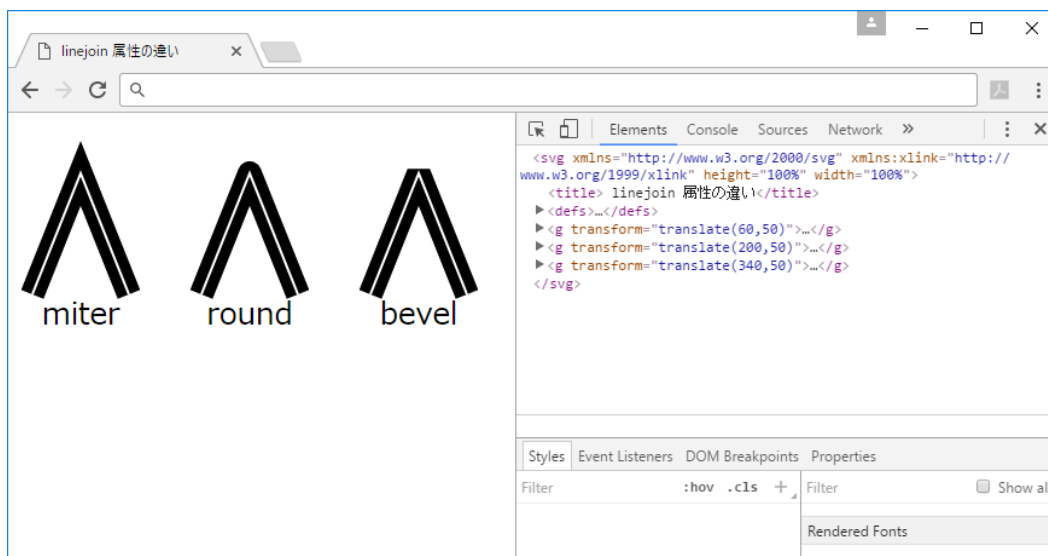


図 6.1: Chrome のデベロッパーツールで DOM ツリーを表示する (1)

- 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 6.2 は図 6.1 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。

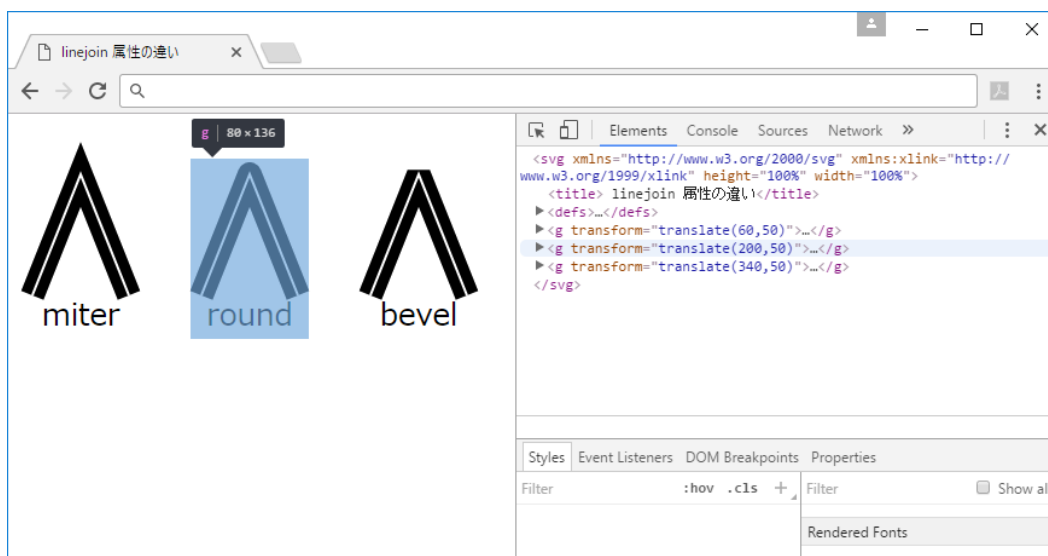


図 6.2: Chrome のデベロッパーツールで DOM ツリーを表示する (2)

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。

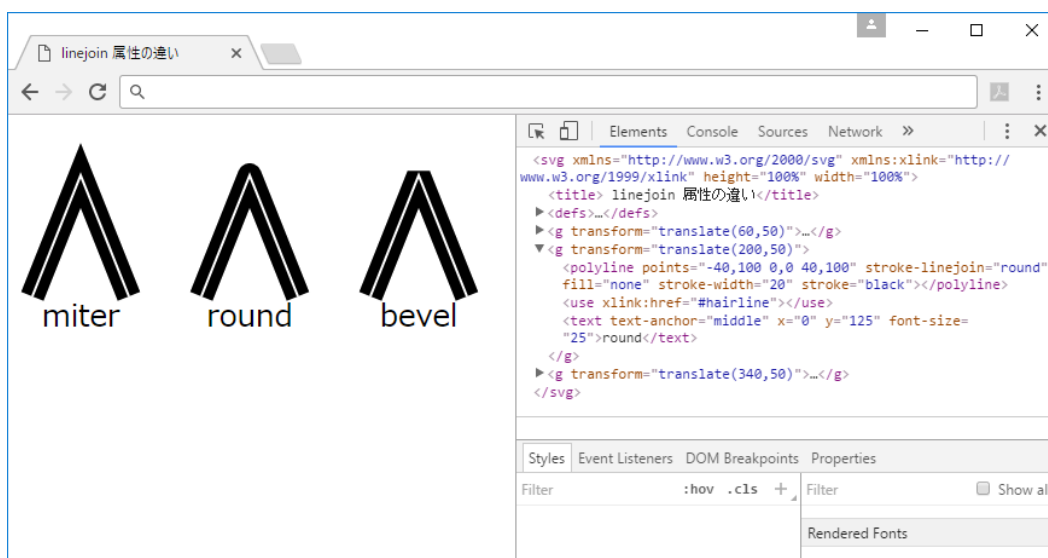


図 6.3: Chrome の開発ツールで DOM ツリーを表示する (3)

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 6.4 です。

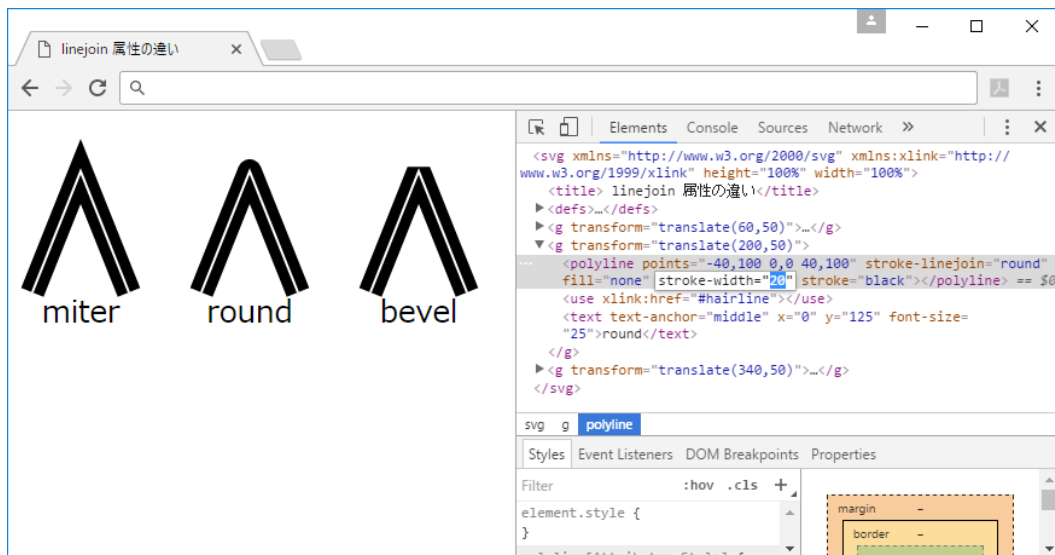


図 6.4: Chrome の開発ツールで DOM ツリーを表示する (4)

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 6.5 です。

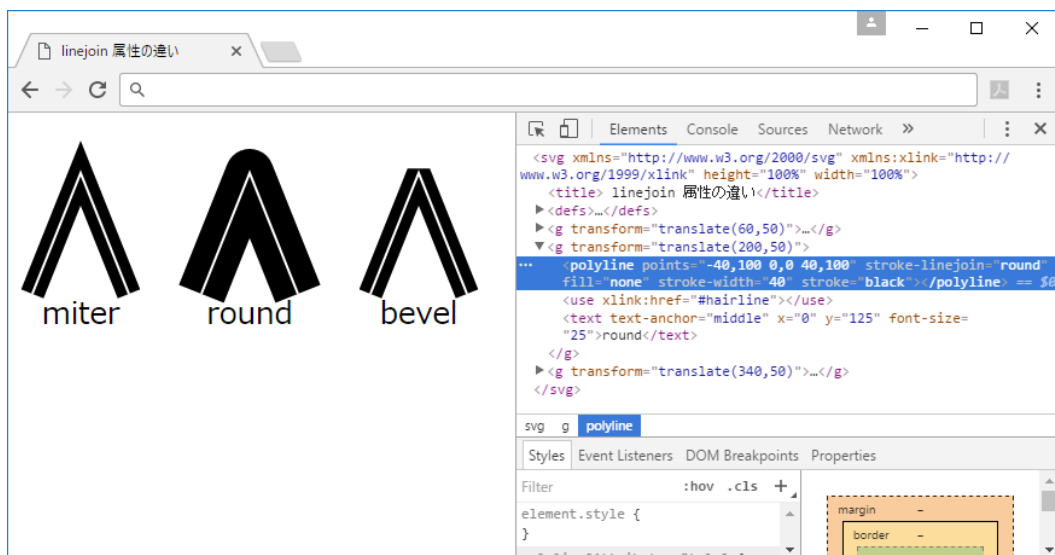


図 6.5: Chrome の開発ツールで DOM ツリーを表示する (5)

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 6.1 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

6.1.3 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 6.1 や 6.2 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 文書进行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 6.1: DOM のメソッド

| メソッド名 | 使用可能要素 | 説 明 |
|---|----------|--|
| <code>getElementById(id)</code> | document | 属性 <code>id</code> の値が引数 <code>id</code> である要素を得る。 |
| <code>getElementsByTagName(Name)</code> | 対象要素 | 対象要素の子要素で要素名が <code>Name</code> であるもののリストを得る。 |
| <code>getElementsByClassName(Name)</code> | 対象要素 | 属性 <code>class</code> の値が <code>Name</code> である要素のリストを得る。 |
| <code>getElementsByName(Name)</code> | document | 属性 <code>name</code> が <code>Name</code> である要素のリストを得る。 |
| <code>querySelector(selectors)</code> | 対象要素 | <code>selectors</code> 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。 |
| <code>querySelectorAll(selectors)</code> | 対象要素 | <code>selectors</code> 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。 |
| <code>getAttribute(Attrib)</code> | 対象要素 | 対象要素の属性 <code>Attrib</code> の値を読み出す。得られる値はすべて文字列である。 |
| <code>setAttribute(Attrib, Val)</code> | 対象要素 | 対象要素の属性 <code>Attrib</code> の値を <code>Val</code> にする。数を渡しても文字列に変換される。 |
| <code>hasAttribute(Attrib)</code> | 対象要素 | 対象要素に属性 <code>Attrib</code> がある場合は <code>true</code> を、ない場合は <code>false</code> を返す。 |
| <code>removeAttribute(Attrib)</code> | 対象要素 | 対象要素の属性 <code>Attrib</code> を削除する。 |
| <code>getNodeName()</code> | 対象要素 | 対象要素の要素名を得る。 |
| <code>createElement(Name)</code> | document | <code>Name</code> で指定した要素を作成する。 |
| <code>createElementNS(NS, Name)</code> | document | 名前空間 <code>NS</code> で定義されている要素 <code>Name</code> を作成する。 |
| <code>createTextNode(text)</code> | document | <code>text</code> を持つテキストノードを作成する。 |
| <code>cloneNode(bool)</code> | 対象要素 | <code>bool</code> が <code>true</code> のときは対象要素の子要素すべてを、 <code>false</code> のときは対象要素だけの複製を作る。 |

次ページへ続く

表 6.1: DOM のメソッド (続き)

| メソッド名 | 使用可能要素 | 説 明 |
|---|---------|---|
| <code>appendChild(Elm)</code> | 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。 |
| <code>insertBefore(newElm, PElm)</code> | 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。 |
| <code>removeChild(Elm)</code> | 対象要素 | 対象要素の子要素 Elm を取り除く。 |
| <code>replaceChild(NewElm, OldElm)</code> | 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。 |
| <code>setValue(value)</code> | テキストノード | 対象のテキストノードの値を value にする。 |

いくつか注意をします。

- 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。

6.5 節では HTML の要素と SVG の要素を同時に取り扱います。

- CSS セレクタをまとめたものは付録 C を参照してください。
- 要素のリストが得られるメソッドの戻り値の各要素は配列と同様に [] で参照でできます。

表 6.2 は DOM の要素に対するプロパティです。

なお、DOM4[14] 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。

6.2 イベント

6.2.1 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。

表 6.2: 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) |

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。

6.2.2 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 6.3 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 6.3 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 6.3: 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 |

6.3 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 文書の図形にアクセスするきっかけとしては図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。具合的な記述方法はこの後を参考にしてください。

6.3.1 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 6.4 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 6.6 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 6.4: 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 x 座標 |
| screenY | getScreenY() | long | マウスポインタの画面における y 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 x 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 y 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 | cntrl 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.CAPTURING_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 |

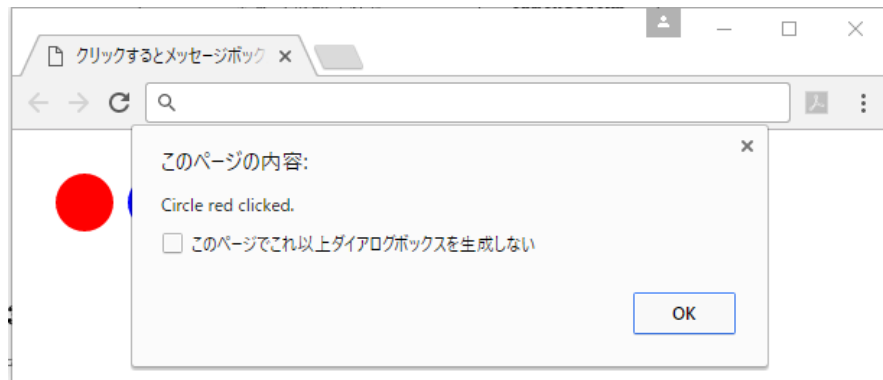


図 6.6: クリックするとメッセージボックスが表示されます

SVG リスト 6.1: 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。
- こららの円には 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 内にかかれた関数 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 6.4 と [15, 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。

なお、新しい EcmaScript ではテンプレートリテラルが追加されました。これを用いると文字列の中に式の結果を直接入れることができます。テンプレートリテラルは ` (バッククォート) で囲み、置き換える式を {} 内に記述し、その前に\$を付けます。上の例では次のようになります。

```
‘Circle ${event.target.getAttribute("fill")} clicked.’
```

問題 6.2 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 6.3 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 6.2: 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"
```

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

⁴XML の規格では CDATA セクションと呼ばれます。

```

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          let Cs = document.getElementsByTagName("circle");
10         for( let 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。
- (9行目) では<circle> 要素を持つオブジェクトのリストを得て (getElementsByTagName メソッド)、その結果を変数 Cs に代入しています。let は変数の宣言方法です。従来は var で宣言をしていましたが、いくつかの問題点があったので新しい EcmaScript で導入されました。

getElementsByTagName メソッドは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ブリングを制御するフラグです。イベントバブリングについては 90 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。

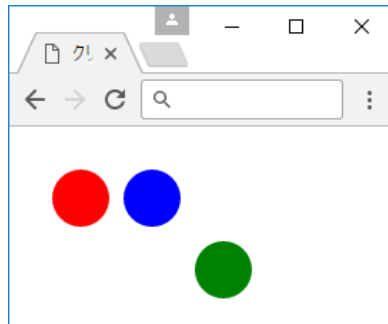


図 6.7: クリックするとその円が移動します

SVG リスト 6.3: 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              let Cs = document.getElementsByTagName("circle");
10             for( let i=0; i< Cs.length; i++) {
11                 Cs[i].addEventListener("click",click, false);
12             }
13         }
14         function click(T) {
15             let 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した。リスト 6.2 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 6.1 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 なので初め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 $150 - 130 = 20$ と初めの値に戻ります。

- 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。

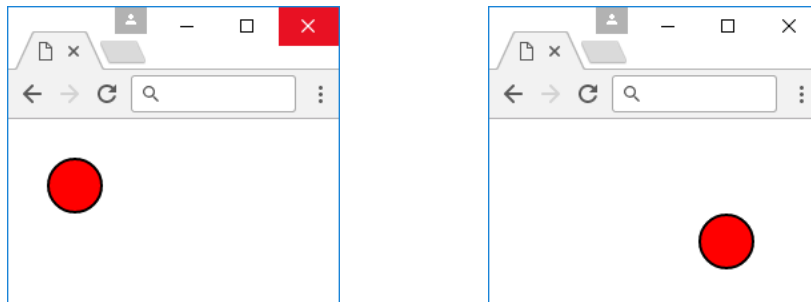


図 6.8: クリックした位置に円が移動します

SVG リスト 6.4: 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             let 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、そのあとに直接、`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しています。
- `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 6.4 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は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 6.4 リスト 6.4 で次のことを確かめなさい。

1. 円上でクリックしても円が移動しない。
2. 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。
3. 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。
4. 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て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 6.9 はクリックした位置に円が移動するだけでなく、その位置が SVG 内に表示されます。リスト 6.5 は図 6.9 のソースコードです。

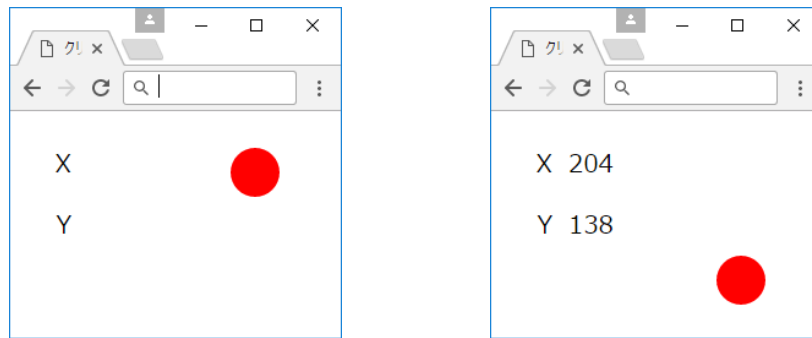


図 6.9: クリックした位置を SVG 内に表示する

SVG リスト 6.5: 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 6.5 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 6.6: 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             let txtNode = document.createTextNode(Value);
17             let 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 6.5 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。

6.3.2 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ブ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ブ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 6.2 の改良

リスト 6.2 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。

次のリストはそうように改良したものです。

SVG リスト 6.7: 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              let 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 6.6 リスト 6.7 について次のことをしなさい。

1. リスト 6.7 がリスト 6.2 と同じ動作を確認しなさい。
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。

イベントの伝播のキャンセル

DOM Level 2 のイベントモデルではイベントの発生が DOM の構造を伝わってオブジェクトに伝えられます。構造が複雑になればシステムに負担がかかります。このイベントの発生の伝播を途中で打ち切るのが `stopPropagation()` です。

6.3.3 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たから `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。

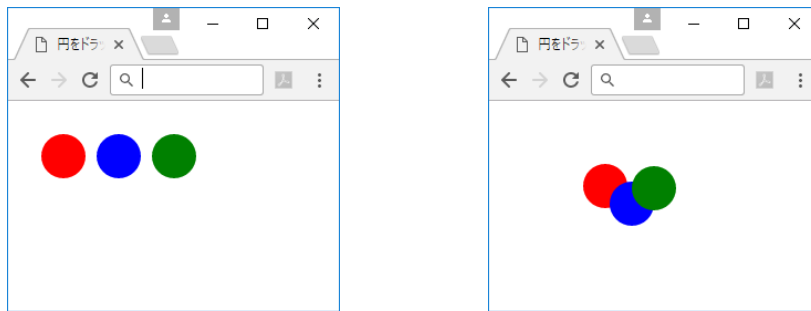


図 6.10: 3 つの円がそれぞれドラッグで移動可能 (右: 初期状態、左: 3 つの円を移動したとき)

オブジェクトのドラッグ処理の例

図 6.10 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 6.8 です。

SVG リスト 6.8: 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          let 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", mmove, false);
17         }
18         function mmove(event) {
19             inDragging.setAttribute("cx", event.clientX);
20             inDragging.setAttribute("cy", event.clientY);
21         }
22         function mup(event) {
23             G.removeEventListener("mousemove", mmove, 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 6.6 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` 変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 6.7 リスト 6.8 で次のことを行いなさい。

1. 円の数を増やしなさい。
2. ドラッグする図形に楕円を付け加えなさい。
3. ドラッグする図形に正方形を付け加えなさい。要素は`<rect>` 要素でなくてもかまいません。

6.3.4 オブジェクトを追加する

リスト 6.8 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 6.11 はいくつかの直線を引いた後のものです。

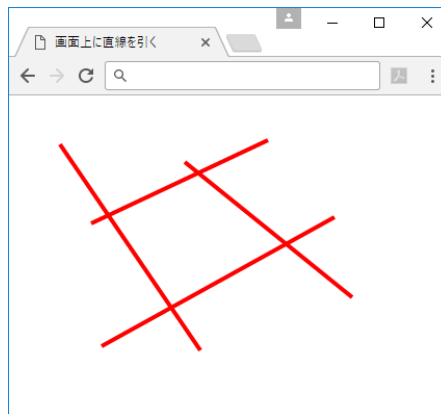


図 6.11: 画面上に直線を引く

SVG リスト 6.9: 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  //
7      let svgNS = "http://www.w3.org/2000/svg";
8      let 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 mmove(E) {
26         NewLine.setAttribute("x2", E.clientX);
27         NewLine.setAttribute("y2", E.clientY);
28     }
29     function mup(E) {
30         C.removeEventListener("mousemove", mmove, false);
31     }
32 //]]&gt;&lt;/script&gt;
33 &lt;g id="Canvas"&gt;
34     &lt;rect x="0" y="0" width="100%" height="100%" fill="white"/&gt;
</pre>
</div>
```

```

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 6.8 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 6.12 参照)。
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のでこれを避けるためにチェックが必要です。

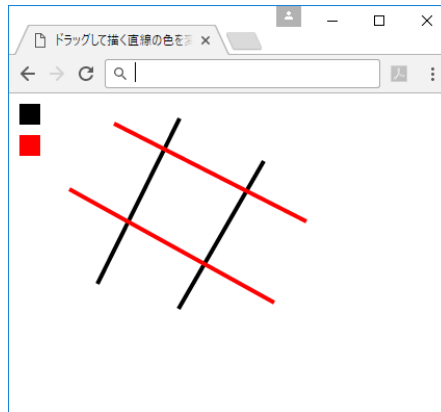


図 6.12: 色を変えて直線を引く

リスト 6.8 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 6.10 ではこの点を改良しています。

SVG リスト 6.10: 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      let 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", mmove, false);
19         }
20     }
21     function mmove(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。

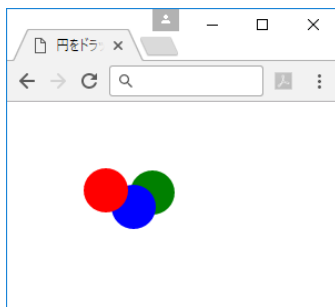


図 6.13: 3 つの円がそれぞれドラッグで移動可能 (改良版)

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 6.9 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。

6.4 起動時に JavaScript で要素を作成する

6.4.1 任意の形の曲線を描く

図 6.14 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

$$\begin{cases} x &= r(\theta - \sin \theta) \\ y &= r(1 - \cos \theta) \end{cases} \quad (6.1)$$

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なります。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` に設定することになります。

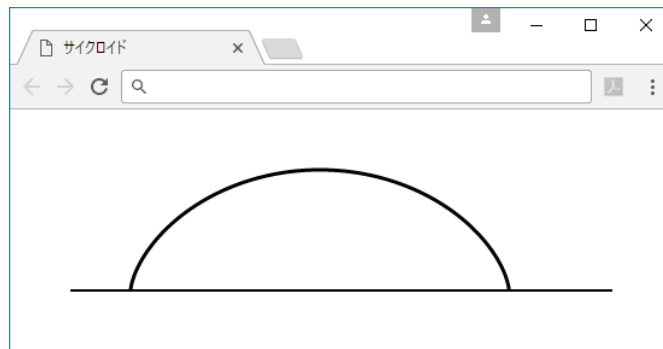


図 6.14: サイクロイド

SVG リスト 6.11: 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  //
7  let R = 50;
8  window.onload = function() {
9      let 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>
</div>
<div data-bbox="141 721 823 858" data-label="List-Group">
<ul>
<li>● 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。</li>
<li>● 転がる直線の <math>y</math> 座標は 0 になっています。</li>
<li>● 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。</li>
<li>● 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。</li>
<li>● 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 <math>d</math> は "M" で初期化しています。</li>
</ul>
</div>
```

- 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後のリスト 6.16 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 (6.1) 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 6.10 リスト 6.11 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。

6.4.2 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进行处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 6.12 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 6.12: 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   let HTMLNS = "http://www.w3.org/1999/xhtml";
3   return MakeElement(HTMLNS, P, Elm, Attribs, Events)
4 }
```

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。


```

5  function MKSVGElm(P, Elm, Attribs, Events) {
6      let 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     let 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 }
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, Attribs, Events) {
2      let HTMLNS = "http://www.w3.org/1999/xhtml";
3      return MakeElement(HTMLNS, P, Elm, Attribs, 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
- attribs 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、イベントバブリ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, Attribs, Events) {  
6   let SVGNS = "http://www.w3.org/2000/svg";  
7   return MakeElement(SVGNS, P, Elm, Attribs, 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, attribs, events) {  
10  let Element = document.createElementNS(NS,elem);  
11  SetAttributes(Element, attribs);  
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, attribs) {  
17   for( attrib in attribs) {  
18     Elm.setAttribute(attrib,attribs[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、イベントバブリ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 }

```

6.4.3 ツェルナーの錯視図形

図 6.15 は古典的なツェルナーの錯視図形です [21, 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。



図 6.15: ツェルナーの錯視図形

前の節の関数を用いて図 6.15 を表示する SVG 文書を作成します。

SVG リスト 6.13: 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>ツェルナーの錯視図形</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     let W = 1, WCol = "black";
11     let sW = 1, sWCol = "black", sL = 50/2, Ang=30, Step = 20;
12     let X1 = 50, Y1=50, X2 = 300, Y2 = 100;
13     let G1, G2, Tmp, i;
14     let 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 6.12 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たことを思い出してください。
- ツェ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)。ここでの変数の意味は次のとおりです。

| | | | | | |
|-------|---------|--------|-----------|-------|-----------|
| W: | 水平線の線幅 | Y1: | 上の水平線の縦位置 | SL: | 補助線の長さの半分 |
| WCol: | 水平線の色 | Y2: | 下の水平線の縦位置 | Ang: | 補助線が交わる角度 |
| X1: | 水平線の左位置 | sW: | 補助線の線幅 | Step: | 補助線の間隔 |
| X2: | 水平線の右位置 | sWCol: | 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とします (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 6.11 Chrome の デベロッパーツールを用いて解説のとおり SVG の DOM ができているかどうかを確認しなさい。

問題 6.12 図 6.15 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。

問題 6.13 リスト 6.9 をここの関数群を用いて書き直しなさい。

問題 6.14 図 6.16 はネックレスの糸とよばれる錯視図形です ([24, 60 ページ図 6.6 右] 参照)。
この図を作成しなさい。

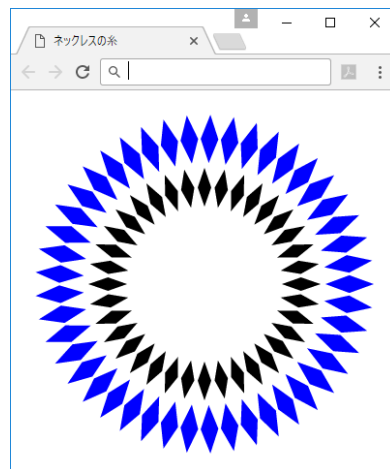


図 6.16: ネックレスの糸

次の図形はバインジオ・ピンナの錯視図形 ([24, カラー図版 9] 参照) とよばれる錯視図形です。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

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。

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。

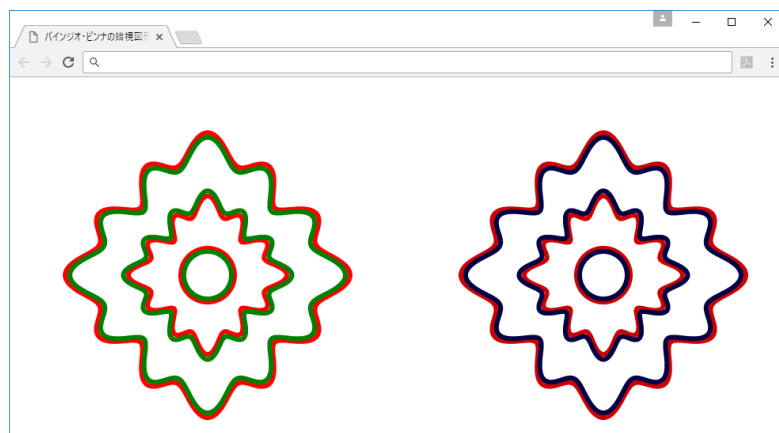


図 6.17: バインジオ・ピンナの錯視図形

SVG リスト 6.14: 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・ピンナの錯視図形</title>
6  <script type="text/ecmascript" xlink:href="./make-svg-elm.js" />
7  <script type="text/ecmascript">
8  <![CDATA[
9  let Canvas;
10 window.onload = function(){
11     DrawFigs("red", "green", "Canvas1");
12     DrawFigs("#C00", "#004", "Canvas2");
13 }
14 function DrawFigs(Color1, Color2, Place) {
15     let W1=8, W2=4;
16     Canvas = document.getElementById(Place);
17     DrawFigure(150, 30, W1, W2, Color1);
18     DrawFigure(144, 30, W1, W2, Color2);
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     let 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 はそれに变化させる幅です。 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)。

6.4.4 一定時間経過後に関数をよびだす (自前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ったり制限があったりします。

- <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。これは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になります。

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 6.15: 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        let 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。

⁹何秒後かに別のページに飛ば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。

- 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 6.15 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 6.11 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。

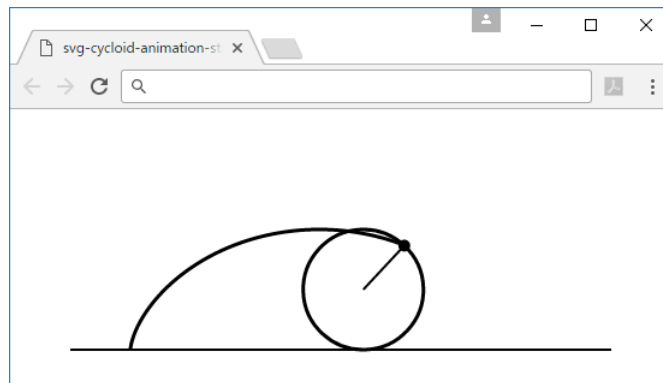


図 6.18: サイクロイドを描く — アニメーション版

SVG リスト 6.16: 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  //
7  let R = 50, Current = 1, Step=2, d ="M0,0 ";
8  let T, Rot, C;
9  window.onload = function() {
10     C =document.getElementById("cycliod");
11     T =document.getElementById("translate");
12     Rot =document.getElementById("rotate");
13     drawCurve();
14 }
15 function drawCurve(){
16     let Next = Current + Step, rad;
17     if(Current&lt;=360) {
18         for( ; Current&lt; Next; Current++) {
19             rad = Current/180*Math.PI;
20             d += `${R*(rad-Math.sin(rad))},${R*(-1+Math.cos(rad))} `;
21         }
22         C.setAttribute("d",d);
23         T.setAttribute("transform",`translate(${R*rad},-50)`);
24         Rot.setAttribute("transform",`rotate(${Next})`);
25         setTimeout(drawCurve,100);
</pre>
</div>
```

```

26     }
27 }
28 //]]>
29 </script>
30 <g transform="translate(100,150)">
31   <line x1="-50" y1="0" x2="400" y2="0" stroke-width="2" stroke="black"/>
32   <path id="cycliod" fill="none" stroke="black" stroke-width="3"/>
33   <g id="translate" transform="translate(0,-50)">
34     <g id="rotate">
35       <circle cx="0" cy="0" r="50" stroke-width="3" stroke="black" fill="none"/>
36       <line x1="0" y1="0" x2="0" y2="50" stroke-width="2" stroke="black" />
37       <circle cx="0" cy="50" r="5" fill="black"/>
38     </g>
39   </g>
40 </g>
41 </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 (33 行目と38 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 6.11 と同じものです。
- 23 行目で回転した図形を平行移動させています。
- 24 行目では図形を回転させています。

問題 6.16 リスト 6.16 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 6.19 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。

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 6.17: 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>

```

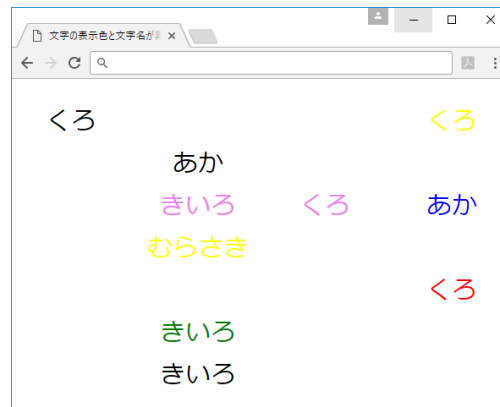


図 6.19: 文字の表示色と文字名が異なる

```

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 //
12 let 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 let Type = 0;
18 let xUnit = 150, xInit = 30, xMax = 6;
19 let yUnit = 50, yInit = 20, yMax = 8;
20 let Max = 10, k, Doc, textNode;
21 let PosList = [];
22 window.onload = function() {
23     let 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     let x, y, i, j, p;
35     let 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];
</pre>
</div>
```

```

41     x = xInit+xUnit * T[0];
42     y = yInit + yUnit * T[1];
43     if( k == Max ) {
44         textNode = MKSVGElem(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);
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     let 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 行目ではまだ問題文を配置していない位置を保持する配列 `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ために `push()` メソッドを用います (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。
 - 準備が終了したので、問題を表示する関数関数 `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)。
 - 選ばれた位置を配列 `Postlist` から取り除く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` にしまします。
 - 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` には乱数で選ばれた色 (`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1 秒後に表示するために `setTimeout` をメソッド用います (53 行目)。
 - 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。

- この値は 0 から 1 の間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次の式で計算します。

$$(\text{min} + (\text{max} - \text{min}) * \text{Math.random}())$$

- 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 6.17 図 6.20 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。

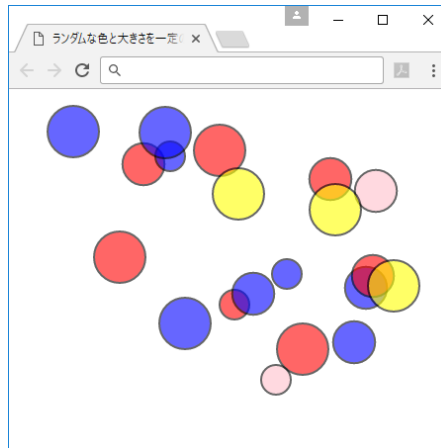


図 6.20: 色と大きさと位置をランダム決めた円を表示する

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。

6.5 HTML 文書とそこにある 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 6.21 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したときの位置を 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。

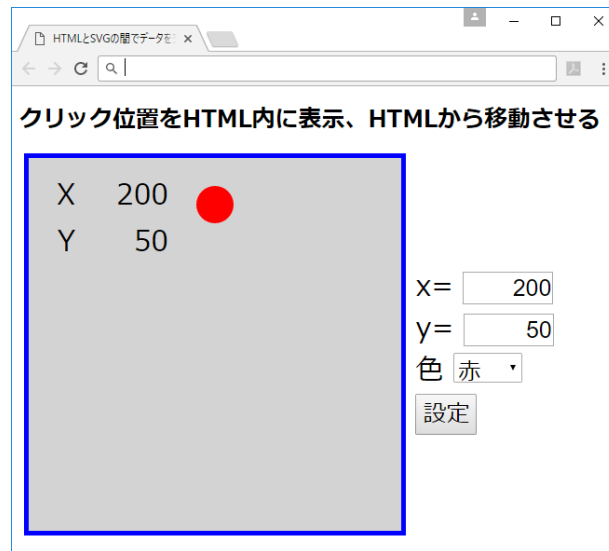


図 6.21: クリック位置を HTML で表示し、HTML のデータから SVG の図形を動かす

HTML リスト 6.18: 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"></text>
20           <text class="textStyle" x="50" y="100"> Y</text>

```

```

21         <text class="textStyle" id="Y" x="150" y="100"></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。以前と比べ記述が格段に簡単になっています。
- HTML文書の<html>要素はHTML文書のルート要素です。その属性としてsvgとxmlns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6.5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。
 - この要素内に長方形を置いて、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行目から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 (この場合のセレクトは 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なっています。

CSS リスト 6.19: 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 のセレクトは 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 が使われています。

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トになります。

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 6.20 はリスト 6.18 から読み込まれる JavaScript ファイルです。

JavaScript リスト 6.20: リスト 6.18 から読み込まれる JavaScript ファイル (SSClickPos.js)

```
1  let Circle, X, Y, XP, YP, oT, oL, B;
2  window.onload = function() {
3      let Colors = {"red": "赤", "yellow": "黄色", "green": "緑",
4                  "blue": "青", "gray": "灰色", "black": "黒"};
5      let tmp, tmpText, Color;
6      let 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     let 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の表示に使用する文字列をオブジェクトリテラルで定義しています。オブジェクトリテラルでは配列の添え字に文字列が使用できます。
- 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。この値は英語の色名なので円の色の設定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の参照を変数に代入しています。
- 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)。
- 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。

¹¹HTML 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が。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。

- 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。
- ブラウザ上でクリックするとその位置は文書全体の位置になり、SVG 文書内で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 行目で求めた補正値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 6.18 リスト 6.18 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 6.22 は図 6.17 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 6.22: バインジオ・ピンナの錯視図形をテキストボックスから設定

右側に曲線の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 6.21: 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.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"
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 6.14 では一つのファイルにまとめてありましたが、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 6.18 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 6.21 で読み込む CSS ファイルです。

CSS リスト 6.22: リスト 6.21 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 6.19 とほとんど変わりません。

JavaScript リスト 6.23: リスト 6.21 で読み込む JavaScript ファイル (pinna.js)

```

1  let 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(let 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     let W1=8, W2=4;
16     let Color1 = C1.value;
17     let 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     let 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 行目で設定された色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 6.19 今までに示された錯視図形のパラメータを外部から設定するように書き直さない。








































付 録 A SVG で利用できる色名

次の表は SVG で利用できる色名とその rgb 値の一覧表です [18]¹。








































| 色 | 色名 | rgb 値 | 16 進表示 |
|---|----------------|---------------------|---------|
|  | aliceblue | rgb(240, 248, 255) | #F0F8FF |
|  | antiquewhite | rgb(250, 235, 215) | #FAEBD7 |
|  | aquamarine | rgb(127, 255, 212) | #7FFFD4 |
|  | aqua | rgb(0, 255, 255) | #00FFFF |
|  | azure | rgb(240, 255, 255) | #F0FFFF |
|  | beige | rgb(245, 245, 220) | #F5F5DC |
|  | bisque | rgb(255, 228, 196) | #FFE4C4 |
|  | black | rgb(0, 0, 0) | #000000 |
|  | blanchedalmond | rgb(255, 235, 205) | #FFEBCD |
|  | blueviolet | rgb(138, 43, 226) | #8A2BE2 |
|  | blue | rgb(0, 0, 255) | #0000FF |
|  | brown | rgb(165, 42, 42) | #A52A2A |
|  | burlywood | rgb(222, 184, 135) | #DEB887 |
|  | cadetblue | rgb(95, 158, 160) | #5F9EA0 |
|  | chartreuse | rgb(127, 255, 0) | #7FFF00 |
|  | chocolate | rgb(210, 105, 30) | #D2691E |
|  | coral | rgb(255, 127, 80) | #FF7F50 |
|  | cornflowerblue | rgb(100, 149, 237) | #6495ED |
|  | cornsilk | rgb(255, 248, 220) | #FFF8DC |
|  | crimson | rgb(220, 20, 60) | #DC143C |
|  | cyan | rgb(0, 255, 255) | #00FFFF |
|  | darkblue | rgb(0, 0, 139) | #00008B |
|  | darkcyan | rgb(0, 139, 139) | #008B8B |
|  | darkgoldenrod | rgb(184, 134, 11) | #B8860B |
|  | darkgray | rgb(169, 169, 169) | #A9A9A9 |
|  | darkgreen | rgb(0, 100, 0) | #006400 |
|  | darkgrey | rgb(169, 169, 169) | #A9A9A9 |
|  | darkkhaki | rgb(189, 183, 107) | #BDB76B |
|  | darkmagenta | rgb(139, 0, 139) | #8B008B |
|  | darkolivegreen | rgb(85, 107, 47) | #556B2F |

次のページへ








































¹ # で始まる 3 桁の 16 進表示は個々の数値を繰り返した 16 進表示と同じです。たとえば #fb8 は #ffbb88 を意味します (<https://www.w3.org/TR/2011/REC-SVG11-20110816/types.html#DataTypeColor>)。

| 色 | 色名 | rgb 値 | 16 進表示 |
|---|---------------|---------------------|---------|
|  | darkorange | rgb(255, 140, 0) | #FF8C00 |
|  | darkorchid | rgb(153, 50, 204) | #9932CC |
|  | darkred | rgb(139, 0, 0) | #8B0000 |
|  | darksalmon | rgb(233, 150, 122) | #E9967A |
|  | darkseagreen | rgb(143, 188, 143) | #8FBC8F |
|  | darkslateblue | rgb(72, 61, 139) | #483D8B |
|  | darkslategray | rgb(47, 79, 79) | #2F4F4F |
|  | darkslategrey | rgb(47, 79, 79) | #2F4F4F |
|  | darkturquoise | rgb(0, 206, 209) | #00CED1 |
|  | darkviolet | rgb(148, 0, 211) | #9400D3 |
|  | deeppink | rgb(255, 20, 147) | #FF1493 |
|  | deepskyblue | rgb(0, 191, 255) | #00BFFF |
|  | dimgray | rgb(105, 105, 105) | #696969 |
|  | dimgrey | rgb(105, 105, 105) | #696969 |
|  | dodgerblue | rgb(30, 144, 255) | #1E90FF |
|  | firebrick | rgb(178, 34, 34) | #B22222 |
|  | floralwhite | rgb(255, 250, 240) | #FFFAF0 |
|  | forestgreen | rgb(34, 139, 34) | #228B22 |
|  | fuchsia | rgb(255, 0, 255) | #FF00FF |
|  | gainsboro | rgb(220, 220, 220) | #DCDCDC |
|  | ghostwhite | rgb(248, 248, 255) | #F8F8FF |
|  | goldenrod | rgb(218, 165, 32) | #DAA520 |
|  | gold | rgb(255, 215, 0) | #FFD700 |
|  | gray | rgb(128, 128, 128) | #808080 |
|  | greenyellow | rgb(173, 255, 47) | #ADFF2F |
|  | green | rgb(0, 128, 0) | #008000 |
|  | grey | rgb(128, 128, 128) | #808080 |
|  | honeydew | rgb(240, 255, 240) | #F0FFF0 |
|  | hotpink | rgb(255, 105, 180) | #FF69B4 |
|  | indianred | rgb(205, 92, 92) | #CD5C5C |
|  | indigo | rgb(75, 0, 130) | #4B0082 |
|  | ivory | rgb(255, 255, 240) | #FFFFFF |
|  | 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 |

次のページへ

| 色 | 色名 | rgb 値 | 16 進表示 |
|---|----------------------|---------------------|---------|
|  | lightcyan | rgb(224, 255, 255) | #E0FFFF |
|  | 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 |

次のページへ

| 色 | 色名 | rgb 値 | 16 進表示 |
|---|---------------|---------------------|---------|
|  | palegoldenrod | rgb(238, 232, 170) | #EEE8AA |
|  | palegreen | rgb(152, 251, 152) | #98FB98 |
|  | paleturquoise | rgb(175, 238, 238) | #AFEEEE |
|  | palevioletred | rgb(219, 112, 147) | #DB7093 |
|  | papayawhip | rgb(255, 239, 213) | #FFefd5 |
|  | peachpuff | rgb(255, 218, 185) | #FFDAB9 |
|  | peru | rgb(205, 133, 63) | #CD853F |
|  | pink | rgb(255, 192, 203) | #FFC0CB |
|  | plum | rgb(221, 160, 221) | #DDA0DD |
|  | powderblue | rgb(176, 224, 230) | #B0E0E6 |
|  | purple | rgb(128, 0, 128) | #800080 |
|  | red | rgb(255, 0, 0) | #FF0000 |
|  | rosybrown | rgb(188, 143, 143) | #BC8F8F |
|  | royalblue | rgb(65, 105, 225) | #4169E1 |
|  | saddlebrown | rgb(139, 69, 19) | #8B4513 |
|  | salmon | rgb(250, 128, 114) | #FA8072 |
|  | sandybrown | rgb(244, 164, 96) | #F4A460 |
|  | seagreen | rgb(46, 139, 87) | #2E8B57 |
|  | seashell | rgb(255, 245, 238) | #FFF5EE |
|  | sienna | rgb(160, 82, 45) | #A0522D |
|  | silver | rgb(192, 192, 192) | #C0C0C0 |
|  | 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 関連図書

- [1] Kurt Cagle, *SVG Programming: The Graphical Web*, Apress 2002
- [2] Oswald Cansepatto, *Fundamentals of SVG Programming: Concept to Source Code*, CHARLES RIVER MEDIA, INC. 2004
- [3] D. Crockford(水野 貴明 訳), JavaScript:The Good Parts 「良いパーツ」によるベストプラクティス, オライリー・ジャパン, 2008
- [4] ECMA, ECMAScript[®] 2015 Language Specification(6th edition)
<http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- [5] David Flanagan(村上列 訳) JavaScript 第 6 版, オライリー・ジャパン, 2012
- [6] 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
- [7] 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
- [8] 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
- [9] Document Object Model (DOM) Level 3 Events Specification, W3C Technical Reports and Publications, 2007, <http://www.w3.org/TR/DOM-Level-3-Events/>
- [10] Ray Erik T.(宮下尚、牧野聡、立堀道明訳) 入門 XML 第 2 版, オライリー・ジャパン 2004
- [11] Hajime Ōuchi, Japanese Optical and Geometrical Art, Dover Pub., 1977
- [12] Ethan Brown (著), (武舎 広幸、武舎 るみ訳) 初めての JavaScript 第 3 版 ES2015 以降の最新ウェブ開発, オライリー・ジャパン (2017)
- [13] W3C, Document Object Model (DOM), <http://www.w3.org/DOM>
- [14] W3C, W3C DOM4 <https://www.w3.org/TR/2015/REC-dom-20151119/>
- [15] W3C, Document Object Model(DOM) Level 2 Events Specification, W3C Recommendation, 2000 <http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/>
- [16] W3C, HTML 4.01 Specification, W3C Recommendation 24 December 1999

- [17] W3C, HTML5 A vocabulary and associated APIs for HTML and XHTML,
<http://www.w3.org/TR/html5/>
- [18] W3C, Scalable Vector Graphics (SVG) 1.1 Specification,
<http://www.w3.org/TR/2011/REC-SVG11-20110816/>
- [19] W3C, HTML & CSS , <http://www.w3.org/standards/webdesign/htmlcss>
- [20] 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
- [21] 後藤 倬男 (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)
- [22] 北岡明佳, 錯視入門, 朝倉書店、2010 年
- [23] 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)
- [24] 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

付 録 C CSS について

カスケーディングスタイルシート (CSS) は HTML 文書の要素の表示方法を指定するものです。CSS は JavaScript から制御できます。

文書のある要素に適用されるスタイルルールは、複数の異なるルールを結合 (カスケード) したものです。スタイルを適用するためには要素を選択するセレクトクで選びます。

表 C.1 は CSS3 におけるセレクトクを記述したものです¹。

表 C.1: CSS3 のセレクトク

| セレクトク | 解説 |
|-----------------------|--|
| * | 任意の要素 |
| E | タイプが E の要素 |
| E[foo] | タイプが E で属性 "foo" を持つ要素 |
| E[foo="bar"] | タイプが E で属性 "foo" の属性値が"bar"である要素 |
| E[foo~="bar"] | タイプが E で属性 "foo" の属性値がスペースで区切られたリストでその一つが "bar"である要素 |
| E[foo^="bar"] | タイプが E で属性 "foo" の属性値が"bar"で始まる要素 |
| E[foo\$="bar"] | タイプが E で属性 "foo" の属性値が"bar"で終わる要素 |
| E[foo*="bar"] | タイプが E で属性 "foo" の属性値が"bar"を含む要素 |
| E[foo = "en"] | タイプが E で属性 "foo" の属性値がハイフンで区切られたリストでその一つが "en"で始まる要素 |
| E:root | document のルート要素 |
| E:nth-child(n) | 親から見て n 番目の要素 |
| E:nth-last-child(n) | 親から見て最後から数えて n 番目の要素 |
| E:nth-of-type(n) | そのタイプの n 番目の要素 |
| E:nth-last-of-type(n) | そのタイプの最後から n 番目の要素 |
| E:first-child | 親から見て一番初めの子要素 |
| E:last-child | 親から見て一番最後の子要素 |
| E:first-of-type | 親から見て初めてのタイプである要素 |
| E:last-of-type | 親から見て最後のタイプである要素 |
| E:only-child | 親から見てただ一つしかない子要素 |

次ページへ続く

¹<http://www.w3.org/TR/selectors/>より引用。

表 C.1: CSS3 のセレクタ (続き)

| セレクタ | 解説 |
|----------------------------|--|
| E:only-of-type | 親から見てただ一つしかないタイプの要素 |
| E:empty | テキストノードを含めて子要素がない要素 |
| E:link, E:visited | まだ訪れたことがない (:link) か訪れたことがある (visited) ハイパーリンクのアンカーである要素 |
| E:active, E:hover, E:focus | ユーザーに操作されている状態中の要素 |
| E:target | 参照 URI のターゲットである要素 |
| E:enabled, E:disabled | 使用可能 (:enable) か使用不可のユーザーインターフェイスの要素 |
| E:checked | チェックされているユーザーインターフェイスの要素 |
| E::first-line | 要素のフォーマットされたはじめの行 |
| E::first-letter | 要素のフォーマットされたはじめの行 |
| E::before | 要素の前に生成されたコンテンツ |
| E::after | 要素の後に生成されたコンテンツ |
| E.warning | 属性 class が "warning" である要素 |
| E#myid | 属性 id の属性値が "myid" である要素 |
| E:not(s) | 単純なセレクタ s にマッチしない要素 |
| E F | 要素 E の子孫である要素 F |
| E > F | 要素 E の子である要素 F |
| E F+ | 要素 E の直後にある要素 F |
| E ~ F | 要素 E の直前にある要素 F |

いくつか注意する点を挙げます。

- 属性 id の属性値の前に#をつけることでその要素が選ばれます。
- 属性 class の属性値の前に.をつけることでその要素が選ばれます。
- nth-child(n) には単純な式を書くことができます。詳しくは実行例 C.1 を参照してください。このセレクタは複数書いてもかまいません。
- E F と E > F の違いを理解しておくこと。たとえば div div というセレクタは途中で別の要素が挟まれていてもかまいません。また、<div>要素が3つある場合にはどのような2つの組み合わせも対象となります。

問題 C.1 次の HTML 文書において nth-child の () 内に次の式を入れた時どうなるか報告しなさい。ここでは箇条書きの開始を示す要素であり、は箇条書きの各項目を示す要素です。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```



```

<title>nth-child のチェック</title>
<style type="text/css" >
li:nth-child(n){
    background:yellow;
}
</style>
</head>
<body>
    <ol>
        <li>1 番目</li>
        <li>2 番目</li>
        <li>3 番目</li>
        <li>4 番目</li>
        <li>5 番目</li>
        <li>6 番目</li>
        <li>7 番目</li>
        <li>8 番目</li>
    </ol>
</body>
</html>

```

[firstnumber=1]

1. n (ここでのリストの設定)
2. $2n$
3. $n+3$
4. $-n+2$

問題 C.2 前問のリストに対し、背景色が次のようになるように CSS を設定しなさい。

1. 偶数番目が黄色、基数番目がオレンジ色
2. 1 番目、4 番目、... のように 3 で割ったとき、1 余る位置が明るいグレー
3. 4 番目以下がピンク
4. 下から 2 番目以下が緑色

索引 — SVG の用語

Symbols

#(値) 17

A

A(値-図形) 43
 a(値-図形) 43
 alphabetic(値-フォント) 67
 <animate> 57, 58, 60
 <animateColor> 57
 <animateMotion> 51, 55
 <animateTransform> 52, 58
 <animatMotion> 51
 attributeName(アニメーション) 51, 52, 58
 attributeType(アニメーション) 51, 58
 auto(値-アニメーション) 57
 auto(値-フォント) 67

B

baseline(値-フォント) 67
 baseline-shift(フォント) 67
 begin(アニメーション) 51
 bevel(値-図形) 41
 black(値-色) 14
 blue(値-色) 108
 butt(値-図形) 14

C

C(値-図形) 43
 c(値-図形) 43
 calcMode(アニメーション) 51, 60, 63
 <circle> 22, 55, 84, 108
 — の属性 23
 click(値) 92
 color(アニメーション) 63
 color(図形) 56
 CSS(値) 51
 currentColor(値-色) 56, 63
 cursive(値-フォント) 67
 cx(色) 33
 cx(図形) 23
 cy(色) 33
 cy(図形) 23, 86

D

d(図形) 43-45, 98, 107
 <defs> 17, 18, 25, 26, 46, 57

discrete(値-アニメーション) 51, 60, 63
 dominant-baseline(フォント) 66, 67
 dur(アニメーション) 51, 53

E

<ellipse> 22
 — の属性 23
 encoding 13
 end(値-フォント) 67
 evenodd(値-色) 40

F

fantasy(値-フォント) 67
 fill(アニメーション) 51, 53
 fill(図形) . 18, 23, 27, 34, 36-38, 40, 41, 45, 46,
 56, 57, 63, 66, 80, 83, 113
 fill-opacity(図形) 36, 37
 fill-rule(図形) 40
 font-family(値-フォント) 67
 font-family(フォント) 67
 font-size(フォント) 67, 69
 font-stretch(フォント) 67
 font-style(フォント) 67
 freeze(値-アニメーション) 51, 53
 from(アニメーション) 51, 53, 60
 fx(色) 33
 fy(色) 33

G

<g> 13-15, 17, 18, 32, 52, 55, 56, 91, 92, 96, 104,
 107, 113, 116, 121
 gradientUnits(色) 26, 29, 31-33, 58, 59
 gray(値-色) 14
 green(値-色) 108, 109

H

hanging(値-フォント) 67
 height(図形) 13, 18, 20, 46, 48, 96

I

ideographic(値-フォント) 67
 id 17, 24-27, 46, 87, 96, 107
 indefinite(値-アニメーション) 51, 54, 63
 italic(値-フォント) 67

K

keyTimes(アニメーション) 51, 60, 61

L

L(値-図形) 43
 l(値-図形) 43, 45
 lightgrey(値-色) 61
 <line> 14, 96
 <line>
 — の属性値 14
 line-through(値-フォント) 67
 linear(値-アニメーション) 51
 <linearGradient> 26, 27
 linecap(図形) 14
 linejoin(図形) 41

M

M(値-図形) 43
 m(値-図形) 43
 mathematical(値-フォント) 67
 middle(値-フォント) 67
 miter(値-図形) 41
 miterlimit(図形) 41
 monospace(値-フォント) 67
 <mpath> 57

N

none(値-色) 12, 34, 41, 45
 none(値-フォント) 67
 normal(値-フォント) 67

O

objectBoundingBox(値-色) .26, 29, 31–33, 35, 46
 oblique(値-フォント) 67
 offset(色) 27
 onclick 82, 84
 onload 84, 85
 opacity(色) 36, 37
 overline(値-フォント) 67

P

paced(値-アニメーション) 51
 <path> 43–45, 98, 107
 path(図形) 55
 <pattern> 46, 50
 patternTransform(図形) 50
 patternUnits(図形) 46
 points(図形) 39, 40
 <polygon> 39, 40, 44
 <polyline> 39, 44, 75

Q

Q(値-図形) 43
 q(値-図形) 43

R

r(色) 33
 r(図形) 22, 23

<radialGradient> 33
 — の属性 33
 <rect> 18, 54, 87, 94
 — の属性 18
 red(値-色) 11, 56
 remove(値-アニメーション) 51, 53
 repeatCount(アニメーション) 51, 53, 54, 63
 reverse-auto(値-アニメーション) 57
 rgb(値-色) 11, 12, 38
 rotate(値) 13, 17, 54
 rotate(アニメーション) 57
 round(値-図形) 14, 41
 rx(図形) 18, 22, 23
 ry(図形) 18, 22, 23

S

S(値-図形) 43
 s(値-図形) 43
 sans-serif(値-フォント) 67
 scale(値) 13, 40, 54, 55, 58
 <script> 83
 serif(値-フォント) 67
 <set> 60
 spline(値-アニメーション) 51
 square(値-図形) 14
 start(値-フォント) 67
 startOffset(フォント) 69, 70
 <stop> 27, 36
 stop-color(色) 27, 59
 stop-opacity(色) 36, 38
 stroke(図形) 14, 18, 23, 36–38, 57
 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, 45
 sub(値-フォント) 67
 super(値-フォント) 67
 <svg> 13, 14, 87, 92–94, 120

T

T(値-図形) 43
 t(値-図形) 43
 <text> 41, 65, 66, 69, 88, 89, 113, 117
 text-anchor(フォント) 67, 117
 text-decoration(フォント) 67
 <textPath> 69
 to(アニメーション) 51, 53, 60
 transform(値) 52, 53
 transform 13, 17, 25, 40, 52, 54, 105
 — の属性値 13
 translate(値) 13, 53–55
 <tspan> 66, 68
 type(アニメーション) 53, 54
 type 83

U

underline(値-フォント) 67
 url(値) 27, 46
 url(値-色) 46
 <use> 17, 22, 32, 37, 48
 userSpaceOnUse(値-色) 26, 29, 32-34, 46, 58, 59

V

values(アニメーション) 51, 60, 61, 63
 version 13
 visibility 60

W

width(図形) 13, 18, 20, 46, 48, 58, 75, 96

X

x(図形) 18, 22, 48, 53, 63, 66
 x1(色) 28, 29, 32, 58, 59
 x1(図形) 14
 x2(色) 28, 29, 32, 58, 59
 x2(図形) 14
 xlink:href 17, 69, 104
 XML(値) 51
 xmlns:xlink 13
 xmlns 13

Y

y(図形) 18, 22, 48, 53, 66
 y1(色) 28, 29, 32, 58
 y1(図形) 14
 y2(色) 28, 29, 32, 58
 y2(図形) 14

Z

z(値-図形) 43, 45

あ

値

..... 17
 click 92
 CSS 51
 rotate 13, 17, 54
 scale 13, 40, 54, 55, 58
 transform 52, 53
 translate 13, 53-55
 url 27, 46
 XML 51

値-アニメーション

auto 57
 discrete 51, 60, 63
 freeze 51, 53
 indefinite 51, 54, 63
 linear 51
 paced 51
 remove 51, 53

reverse-auto 57
 spline 51

値-色

black 14
 blue 108
 currentColor 56, 63
 evenodd 40
 gray 14
 green 108, 109
 lightgrey 61
 none 12, 34, 41, 45
 objectBoundingBox ... 26, 29, 31-33, 35, 46
 red 11, 56
 rgb 11, 12, 38
 url 46
 userSpaceOnUse ... 26, 29, 32-34, 46, 58, 59

値-図形

A 43
 a 43
 bevel 41
 butt 14
 C 43
 c 43
 L 43
 l 43, 45
 M 43
 m 43
 miter 41
 Q 43
 q 43
 round 14, 41
 S 43
 s 43
 square 14
 T 43
 t 43
 z 43, 45

値-フォント

alphabetic 67
 auto 67
 baseline 67
 cursive 67
 end 67
 fantasy 67
 font-family 67
 hanging 67
 ideographic 67
 italic 67
 line-through 67
 mathematical 67
 middle 67
 monospace 67
 none 67
 normal 67

oblique 67
 overline 67
 sans-serif 67
 serif 67
 start 67
 sub 67
 super 67
 underline 67

アニメーション

attributeName 51, 52, 58
 attributeType 51, 58
 begin 51
 calcMode 51, 60, 63
 color 63
 dur 51, 53
 fill 51, 53
 from 51, 53, 60
 keyTimes 51, 60, 61
 repeatCount 51, 53, 54, 63
 rotate 57
 to 51, 53, 60
 type 53, 54
 values 51, 60, 61, 63

い
色

cx 33
 cy 33
 fx 33
 fy 33
 gradientUnits 26, 29, 31–33, 58, 59
 offset 27
 opacity 36, 37
 r 33
 stop-color 27, 59
 stop-opacity 36, 38
 stroke-opacity 36
 x1 28, 29, 32, 58, 59
 x2 28, 29, 32, 58, 59
 y1 28, 29, 32, 58
 y2 28, 29, 32, 58

す
図形

color 56
 cx 23
 cy 23, 86
 d 43–45, 98, 107
 fill-opacity 36, 37
 fill-rule 40
 fill 18, 23, 27, 34, 36–38, 40, 41, 45, 46,
 56, 57, 63, 66, 80, 83, 113
 height 13, 18, 20, 46, 48, 96
 linecap 14

linejoin 41
 miterlimit 41
 path 55
 patternTransform 50
 patternUnits 46
 points 39, 40
 r 22, 23
 rx 18, 22, 23
 ry 18, 22, 23
 stroke-linecap 12, 14
 stroke-linejoin 41, 42
 stroke-width 14, 18, 20, 23, 37, 45
 stroke 14, 18, 23, 36–38, 57
 width 13, 18, 20, 46, 48, 58, 75, 96
 x 18, 22, 48, 53, 63, 66
 x1 14
 x2 14
 y 18, 22, 48, 53, 66
 y1 14
 y2 14

ふ
フォント

baseline-shift 67
 dominant-baseline 66, 67
 font-family 67
 font-size 67, 69
 font-stretch 67
 font-style 67
 startOffset 69, 70
 text-anchor 67, 117
 text-decoration 67

よ
要素

<animate> 57, 58, 60
 <animateColor> 57
 <animateMotion> 51, 55
 <animateTransform> 52, 58
 <animatMotion> 51
 <circle> 22, 55, 84, 108
 <defs> 17, 18, 25, 26, 46, 57
 <ellipse> 22
 <g> 13–15, 17, 18, 32, 52, 55, 56, 91, 92, 96,
 104, 107, 113, 116, 121
 <line> 14, 96
 <linearGradient> 26, 27
 <mpath> 57
 <path> 43–45, 98, 107
 <pattern> 46, 50
 <polygon> 39, 40, 44
 <polyline> 39, 44, 75
 <radialGradiation> 33
 <rect> 18, 54, 87, 94
 <script> 83

| | |
|-------------------------------------|----------------------------------|
| <code><set></code> | 60 |
| <code><stop></code> | 27, 36 |
| <code><svg></code> | 13, 14, 87, 92–94, 120 |
| <code><text></code> | 41, 65, 66, 69, 88, 89, 113, 117 |
| <code><textPath></code> | 69 |
| <code><tspan></code> | 66, 68 |
| <code><use></code> | 17, 22, 32, 37, 48 |

索引 — HTML の用語

B

 7

C

<canvas> 6

class(属性) 116, 117

CSS の値

 inline-block 117

 middle 117

 right 118

CSS プロパティ

 display 117

 font-size 117

 padding-left 117

 text-align 118

 vertical-align 117

D

display(CSS プロパティ) 117

<div> 116, 117

DOCTYPE 宣言 116

F

font-size(CSS プロパティ) 117

H

<h1> 116, 117

<head> 116

<html> 116

<html> 4

I

<id> 118

id(属性) 117

inline-block(CSS の値) 117

<input> 118

M

<meta> 116

middle(CSS の値) 117

N

name(属性) 119

O

<option> 119

P

padding-left(CSS プロパティ) 117

R

right(CSS の値) 118

S

<script> 80

<select> 119

setTimeout(オブジェクトのメソッド) 113

T

text(値) 118

text-align(CSS プロパティ) 118

type(属性) 118

V

value(属性) 119

vertical-align(CSS プロパティ) 117

あ

値

 text 118

お

オブジェクトのメソッド

 setTimeout 113

そ

属性

 class 116, 117

 id 117

 name 119

 type 118

 value 119

よ

要素

 7

 <canvas> 6

 <div> 116, 117

 <h1> 116, 117

 <head> 116

 <html> 116

 <html> 4

 <id> 118

 <input> 118

| | |
|-----------------------------------|-----|
| <code><meta></code> | 116 |
| <code><option></code> | 119 |
| <code><script></code> | 80 |
| <code><select></code> | 119 |

索引 — JavaScript の用語

Symbols

+ 83

A

addEventListener(DOM のメソッド) 87
 addEventListener(DOM のメソッド) 84, 91
 alert 83
 altKey(DOM のプロパティ) 81
 appendChild(DOM のメソッド) .. 78, 90, 98, 113
 Array のメソッド
 splice 113

B

button(DOM のプロパティ) 81

C

childNodes(DOM のプロパティ) 79
 children(DOM のプロパティ) 79
 clearTimeout 108
 clientX(DOM のプロパティ) 81, 87, 96
 clientY(DOM のプロパティ) 81
 cloneNode(DOM のメソッド) 77
 createElement(DOM のメソッド) 77, 94
 createElementNS(DOM のメソッド) . 77, 94, 102
 createTextNode(DOM のメソッド) .. 77, 90, 113
 ctrlKey(DOM のプロパティ) 81
 currentTarget(DOM のプロパティ) 81, 91

D

document 94
 DOM のオブジェクト
 window 83, 108

DOM のプロパティ

altKey 81
 button 81
 childNodes 79
 children 79
 clientX 81, 87, 96
 clientY 81
 ctrlKey 81
 currentTarget 81, 91
 eventPhase 81
 firstChild 79, 89
 firstElementChild 79
 hasChildNodes 79
 lastChild 79

lastElementChild 79
 metaKey 81
 nextElementSibling 79
 nextSibling 79
 nodeName 79
 nodeType 79
 nodeValue 79
 pageXOffset 81
 pageYOffset 81
 parentNode 79
 previousElementSibling 79
 previousSibling 79
 screenX 81
 screenY 81
 shiftKey 81
 target 81, 83, 91

DOM のメソッド

addEventListener 87
 addEventListener 84, 91
 appendChild 78, 90, 98, 113
 cloneNode 77
 createElement 77, 94
 createElementNS 77, 94, 102
 createTextNode 77, 90, 113
 getAltKey 81
 getAttribute 77, 83-85
 getButton 81
 getClientX 81
 getClientY 81
 getCtrlKey 81
 getCurrentTarget 81
 getElementById 77, 87
 getElementsByClassName 77
 getElementsByName 77
 getElementsByTagName 77, 84, 93
 getMetaKey 81
 getNodeName 77
 getScreenX 81
 getScreenY 81
 getShiftKey 81
 getTarget 81
 hasAttribute 77
 insertBefore 78
 preventDefault 81
 querySelector 77
 querySelectorAll 77

- removeAttribute 77
 - removeChild 78
 - replaceChild 78, 90, 113
 - setAttribute 77, 84, 86, 87
 - setValue 78
 - stopPropagation 81, 92
- E**
- Event.AT_TARGET 81
 - Event.BUBBLING_PHASE 81
 - Event.CAPTURING_PHASE 81
 - eventPhase(DOM のプロパティ) 81
- F**
- false 91
 - firstChild(DOM のプロパティ) 79, 89
 - firstElementChild(DOM のプロパティ) 79
 - for 119
 - function 83, 120
- G**
- getAltKey(DOM のメソッド) 81
 - getAttribute(DOM のメソッド) 77, 83–85
 - getBoundingClientRect 120
 - getButton(DOM のメソッド) 81
 - getClientX(DOM のメソッド) 81
 - getClientY(DOM のメソッド) 81
 - getCtrlKey(DOM のメソッド) 81
 - getCurrentTarget(DOM のメソッド) 81
 - getElementById(DOM のメソッド) 77, 87
 - getElementsByClassName(DOM のメソッド) 77
 - getElementsByName(DOM のメソッド) 77
 - getElementsByName(DOM のメソッド) 77, 84, 93
 - getMetaKey(DOM のメソッド) 81
 - getNodeName(DOM のメソッド) 77
 - getScreenX(DOM のメソッド) 81
 - getScreenY(DOM のメソッド) 81
 - getShiftKey(DOM のメソッド) 81
 - getTarget(DOM のメソッド) 81
- H**
- hasAttribute(DOM のメソッド) 77
 - hasChildNodes(DOM のプロパティ) 79
- I**
- insertBefore(DOM のメソッド) 78
- J**
- jQuery.js 100
- L**
- lastChild(DOM のプロパティ) 79
 - lastElementChild(DOM のプロパティ) 79
 - length(メソッド) 84
- let 84
- M**
- Math.cos(x) 100
 - Math.floor() 114
 - Math.PI 100
 - Math.random() 113
 - Math.sin(x) 100
 - metaKey(DOM のプロパティ) 81
- N**
- nextElementSibling(DOM のプロパティ) 79
 - nextSibling(DOM のプロパティ) 79
 - nodeName(DOM のプロパティ) 79
 - nodeType(DOM のプロパティ) 79
 - nodeValue(DOM のプロパティ) 79
 - null 89, 101, 102
- P**
- pageXOffset(DOM のプロパティ) 81
 - pageYOffset(DOM のプロパティ) 81
 - parentNode(DOM のプロパティ) 79
 - parseInt 86
 - preventDefault(DOM のメソッド) 81
 - previousElementSibling(DOM のプロパティ) 79
 - previousSibling(DOM のプロパティ) 79
 - push()(メソッド) 113
- Q**
- querySelector(DOM のメソッド) 77
 - querySelectorAll(DOM のメソッド) 77
- R**
- removeAttribute(DOM のメソッド) 77
 - removeChild(DOM のメソッド) 78
 - replaceChild(DOM のメソッド) 78, 90, 113
- S**
- screenX(DOM のプロパティ) 81
 - screenY(DOM のプロパティ) 81
 - setAttribute(DOM のメソッド) .. 77, 84, 86, 87
 - setTimeout() 108, 114
 - setValue(DOM のメソッド) 78
 - shiftKey(DOM のプロパティ) 81
 - splice(Array のメソッド) 113
 - splice(メソッド) 113
 - stopPropagation(DOM のメソッド) 81, 92
- T**
- target(DOM のプロパティ) 81, 83, 91
 - true 91
- V**
- var 84

W

window(DOM のオブジェクト) 83, 108

め

メソッド

length 84
push() 113
splice 113

索引 — 一般

C

click(イベント) 85, 92, 116
 clientY(イベント) 96
 $\sin x$ (余弦関数) 100

D

DOM 72

E

evt 変数 82

H

html 4
 HTML5 5

J

JavaScript 71
 オブジェクトリテラル 119

M

mousedown(イベント) 92, 94, 96
 mousemove(イベント) 92, 94, 96
 mouseup(イベント) 92, 94, 96

O

onbegin(イベント) 80
 onchange(イベント) 80
 onclick(イベント) 80, 120
 onend(イベント) 80
 onload(イベント) 80, 85
 onmousedown(イベント) 80
 onmousemove(イベント) 80
 onmouseout(イベント) 80
 onmouseover(イベント) 80
 onmouseup(イベント) 80

S

$\sin x$ (正弦関数) 100
 SVG 1

W

W3C 1
 WebStrage 5
 WHATWG 5
 World Wide Web Consortium 1

X

XHTML 5
 XML 2

あ

アニメーション 51

い

イベント 78
 click 85, 92, 116
 clientY 96
 mousedown 92, 94, 96
 mousemove 92, 94, 96
 mouseup 92, 94, 96
 onbegin 80
 onchange 80
 onclick 80, 120
 onend 80
 onload 80, 85
 onmousedown 80
 onmousemove 80
 onmouseout 80
 onmouseover 80
 onmouseup 80
 クリック 78

イベントキャプチャリング 91

イベントドリブン 79

イベントバブリング 84, 91

色の指定 11

色の対比 21

う

ヴィントの錯視 43

え

円 22

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 20

オブジェクトリテラル
 JavaScript における—— 119

親ノード 72

親要素 14

折れ線 39

か

輝くヘルマン格子 48
 カフェウォール錯視 22, 48, 61

く

グラデーション 25
 線形—— 26
 放射—— 33
 クリック 78
 クリックの処理 (JavaScript) 80
 クレイク・オブライエン効果 35

こ

子ノード 72
 コフカリング 29
 コメント (SVG) 11
 子要素 14

さ

サイクロイド 98, 109
 ザヴィニの錯視 46
 錯視

 色の対比 21
 ヴィントの—— 43
 同じ色なのに周りの色で見え方が異なる .. 20
 輝くヘルマン格子 48
 カフェウォール—— 22, 48, 61
 クレイク・オブライエン効果 35
 コフカリング 29
 ザヴィニの—— 46
 ザバーニョの—— 27, 59
 シェパードの—— 42
 ジャッドの—— 54
 ツェルナーの—— 103
 デルブーフの—— 25
 ネックレスの糸 105
 バインジオ・ピンナの—— 105
 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, 53, 98
 浮動する円 49
 ブルドンの—— 42
 ヘルマン格子 21
 ポッケンドルフの—— 19
 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 48
 ゆがんだ正方形 61
 ザバーニョの錯視 27, 59

し

シェパードの錯視 42
 ジャッドの錯視 54

せ

正弦関数 100
 線形グラデーション 26

そ

属性 13
 —— の円 22
 —— の楕円 22
 属性値 13

た

楕円 22
 属性の—— 22
 多角形 39
 タグ 4

ち

長方形
 —— の属性 18
 直線
 —— の属性 14

つ

ツェルナーの錯視 103

て

テキストエディタ 7
 テキストノード 73, 90
 デルブーフの錯視 25
 テンプレートリテラル 83

と

ドラッグの処理 (JavaScript) 92

な

名前空間 13, 77, 94, 96

ね

ネックレスの糸 105

の

ノード 72
 親—— 72
 子—— 72
 テキスト—— 73
 要素—— 73
 ルート—— 73

は

バインジオ・ピンナの錯視 105

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, 53, 98 |
| 浮動する円 | 49 |
| 不透明度 | 35 |
| ブルドンの錯視 | 42 |
| プロパティ | 77 |

へ

| | |
|--------------|----|
| ヘルマン格子 | 21 |
|--------------|----|

ほ

| | |
|------------------|----|
| 放射グラデーション | 33 |
| ポッケンドルフの錯視 | 19 |

ま

| | |
|--------------------------|----|
| マウスイベントのプロパティとメソッド | 81 |
| 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 | 77 |
| メッセージボックス | 83 |

も

| | |
|------------------|----|
| モーガンのねじれひも | 48 |
| 文字列 | |
| —— をつなげる | 83 |
| —— の属性 | 66 |
| —— を表示する | 65 |

ゆ

| | |
|---------------|----|
| ゆがんだ正方形 | 61 |
|---------------|----|

よ

| | |
|-------------|-----|
| 要素ノード | 73 |
| 余弦関数 | 100 |

る

| | |
|--------------|----|
| ルートノード | 73 |
| ルート要素 | 13 |