M A S - I C T

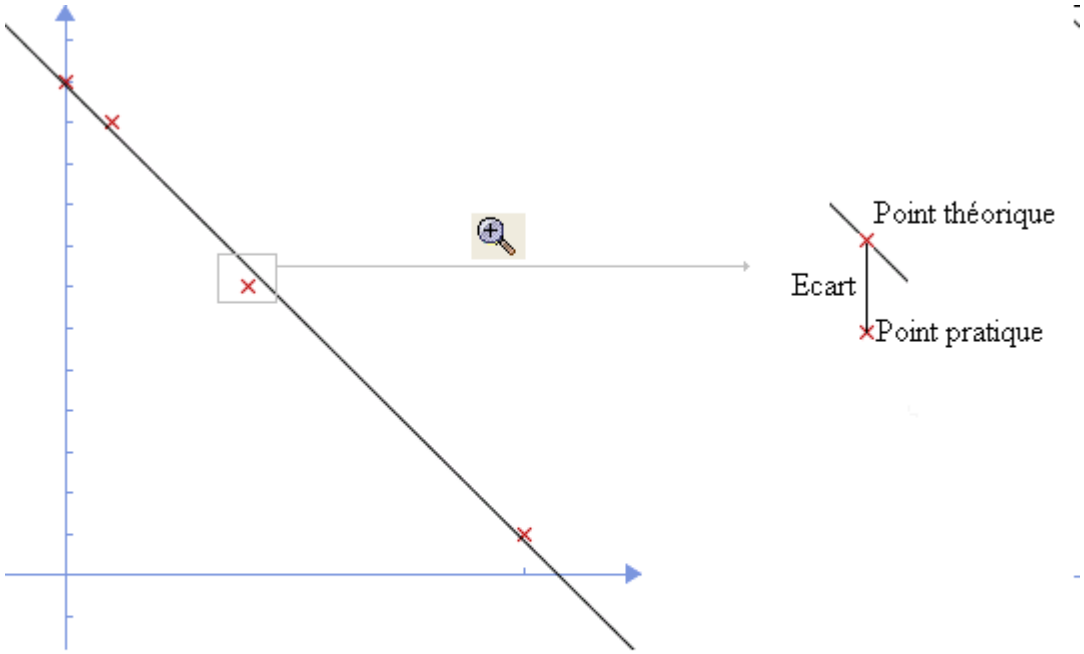
C O U R S S V G

Bienvenue

Plan du 9 octobre

- Régression linéaire, clonage
- Animation avec SMIL / Javascript
- Textes, textes sur chemin
- Les filtres
- Dégradés, textures
- SVG ⇔ HTML
- Accès aux bases de données avec php

Régression linéaire 1



En rouge, on a dessiné les points expérimentaux, et en noir, on a tracé une droite de régression. Le point théorique qui correspond au point pratique est celui situé sur la droite à la même abscisse. La méthode des moindres carrés consiste à prendre la somme des écarts au carré, et à la minimiser.

Régression linéaire 2

- Exemple tiré de www.bibmath.net

$$J(a,b) = \sum_{i=1}^{n} (y_i - ax_i - b)^2.$$

$$\begin{cases} \dfrac{\partial J}{\partial a} &= -2\displaystyle\sum_{i=1}^{n} x_i(y_i - ax_i - b) = 0, \\ \dfrac{\partial J}{\partial b} &= -2\displaystyle\sum_{i=1}^{n} y_i - ax_i - b = 0. \end{cases}$$

$$a = \frac{n\displaystyle\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\displaystyle\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \text{ et } b = \frac{\displaystyle\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n\displaystyle\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}.$$

Clonage

Création d'un élément par copie (cloneNode):

<element id="nom" …….>

```
var oldobjet = svgDocument.getElementById("nom");
var newobjet = oldobjet.cloneNode(true);
newobjet.setAttribute("??","??");
parent.appendChild(newobjet);
```

'parent' est l'élément (balise) parent de 'nom'

Animation

Deux méthodes permettent l'animation:

- SMIL

  Une recommandation du W3C pour la synchronisation multimédia.

- Javascript

  Utilisation de fonctions prédéfinies qui permettent la gestion du temps.

SMIL 1

Langage d'Intégration Multimédia Synchronisé

Eléments de SMIL animation:

‘animate’: modifie la valeur d’attributs au cours du temps

‘set’: raccourci pour ‘animate’ (une valeur)

‘animateMotion’: animation le long d’un tracé

‘animateColor’: animation de la couleur

SMIL 2

Deux extensions SVG à SMIL animation:

‘animateTransform’: permet une modification de l’attribut transform au cours du temps

‘animateMotion’: l’attribut rotate peut être utilisé afin de mieux contrôler la rotation

Les balises d’animation sont imbriquées dans l’élément à animer.

Attributs 1

Attributs pour identifier l'animation:

- attributeName
- attributeType = "**CSS | XML | auto**"

Attributs définissant une valeur:

- from : valeur de depart
- to : valeur d'arrivée
- additive = "**replace | sum**"

Attributs 2

Attributs de gestion de l'animation:

- begin – end : son début et sa fin
- dur : sa durée
- restart : la relance ou non
- repeatCount : nombre de répétitions
- repeatDur : temps pendant on répète
- fill : état à la fin (freeze | remove)

Exemple 1

Animation de la couleur :

```
<animateColor
  attributeName="fill"
  attributeType="CSS"
  from="rgb(0,0,255)"
  to="rgb(128,0,0)"
  begin="3s" dur="6s"
  fill="freeze" />
```

Exemple 2

Effet de zoom (changement d'échelle) :

```
<animateTransform
  attributeName="transform"
  attributeType="XML"
  type="scale"
  from="1" to="3" additive="sum"
  begin="3s" dur="6s"
  fill="freeze" />
```

Exemple 3

Set permet la modification d'un seul attribut :

```
<set
  attributeName="visibility"
  attributeType="CSS"
  to="visible"
  begin="3s"
  fill="freeze" />
```

Textes

Une première façon d'afficher un texte consiste à définir la position du coin supérieur gauche (x,y), la police, la taille et la couleur des caractères

```
<text x="250" y="150"
      font-family="Verdana"
      font-size="55" fill="blue">
      Bonjour vous
</text>
```

Animation 1

Animation à l'aide d'un script:

setInterval et setTimeout (une seule fois) permettent de créer une alarme stockée dans une variable.
clearInterval et clearTimeout arêtent l'alarme.

1. Déclarer une variable globale
2. Déclencher l'animation (setInterval)
3. Arrêt de l'animation (clearInterval)

Animation 2

Exemple : la fonction anime est exécutée toutes les 100 millisecondes.

```
var alarme;
function anime() {…….}
```

Dans une autre fonction :

```
alarme = setInterval("anime() ",100);
```

Dans éventuellement une autre fonction:

```
clearInterval(alarme);
```

Enfants 1

Méthodes concernant les enfants d'un noeud :

`hasChildNodes` : vrai si Nodes a un enfant

`firstChild` : premier enfant

`lastChild` : dernier enfant

`childNodes` : liste d'enfants

`childNodes[n]` : nième enfant de la liste

`childNodes[0]` : **`firstChild`**

`childNodes[-1]` : **`lastChild`**

Enfants 2

Méthodes permettant une action sur les enfants d'un éléments :

`getFirstChild()`: recherche le premier enfant

`replaceChild(…)`: remplace un enfant

`appendChild(…)`: ajoute un enfant

`removeChild(…)`: supprime un enfant

textes

Remplacement d'un texte:

<text id="nom" …….> </text>

```
texte =
 svgDocument.createTextNode("toto");
place =
 svgDocument.getElementById("nom");

place.replaceChild
 (texte,place.getFirstChild());
```

Image bitmap

L'insertion d'une image bitmap dans un document svg :

```
<image   x="400" y="400"
         width="50" height="50"
         xlink:href="fichier.png"/>
```

Les visionneurs SVG reconnaissent au moins les fichiers .png et .jpeg

Portable Network Graphic inspiré de GIF

Les filtres

Les filtres permettent de :

- Donner du relief à une figure 2D
- Donner un éclairage à une scène
- Donner des règles de composition entre 2 figures
- Créer des images ‘artificielles’ (ciel)
- Transformer des figures par calcul

Filtres en SVG

L'effet voulu est souvent obtenu en combinant plusieurs filtres appliqués à la suite.

```
<filter id="effet">
    <filtre_1 ……>
      ……
    <filtre_n ……>
</filter>
<g filter="#effet">
    ici vient la description du dessin
</g>
```

Ombre portée

Filtres permettant de créer une ombre portée :

- feGaussianBlur
  crée un flou gaussien : projection ‘floue’ en niveaux de gris de la figure.
- feOffset
  décalage d’une figure, ici le ‘flou’
- feMerge
  mise ensemble de la figure et du ‘flou’.

Ombre portée 2

Filtre complet :

```
<filter id="ombre">
  <feGaussianBlur in="SourceAlpha"
        stdDeviation="1" result="flou"/>
  <feOffset in="flou" dx="1" dy="1"
        result="flouDecale"/>
  <feMerge>
    <feMergeNode in="flouDecale"/>
    <feMergeNode in="SourceGraphic"/>
  </feMerge>
</filter>
```

Ombre portée 3

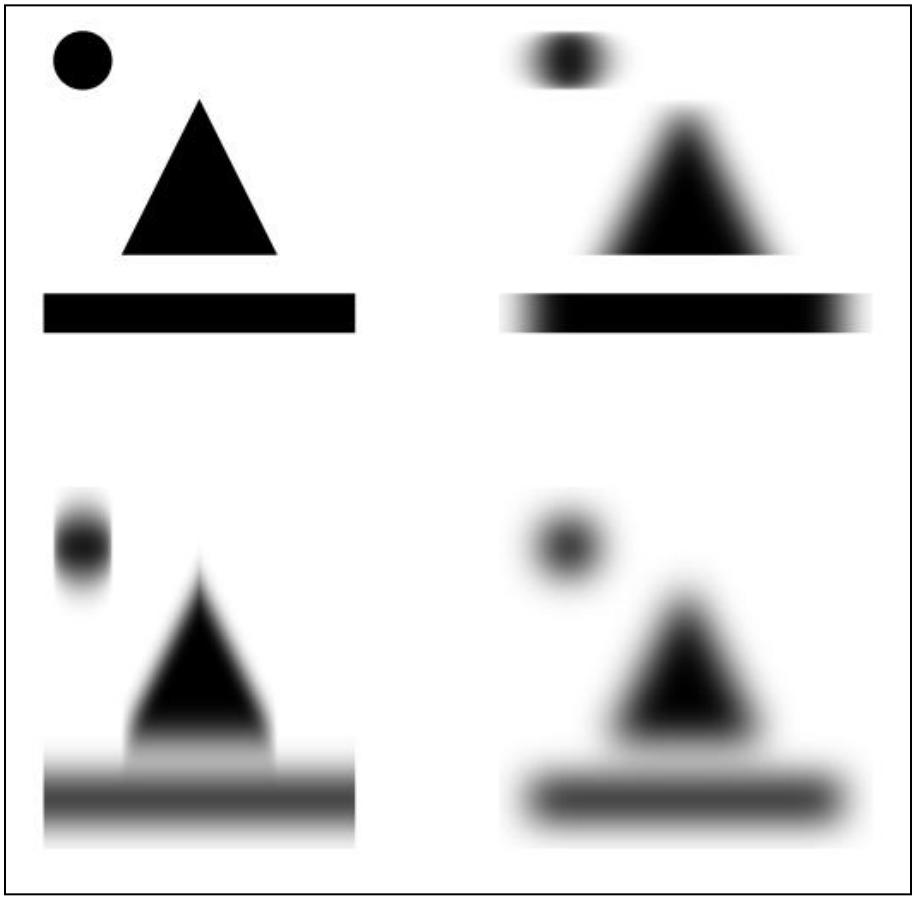
GaussianBlur

Valeurs du Paramètre stdDeviation

0 2,0

0,2 2

exemple



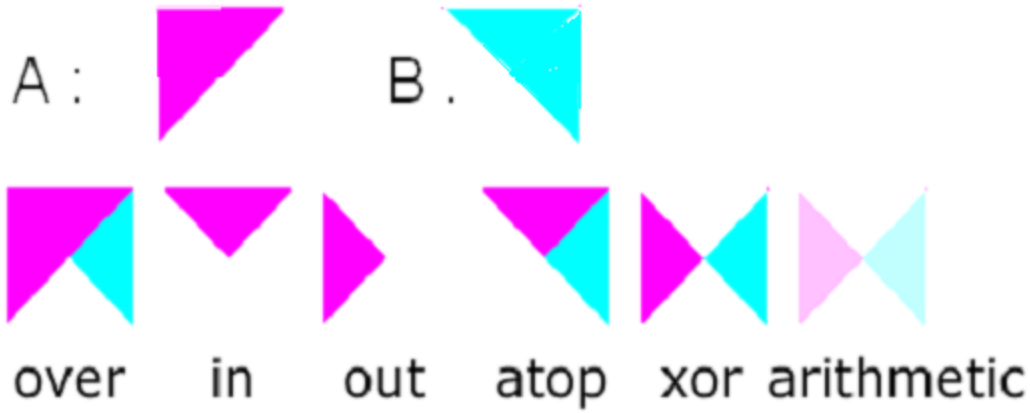
Composition

La composition est une généralisation de la fusion (merge), elle en donne des règles :

- A over B : identique à merge (A sur B)
- A in B : partie de A dans B
- A out B : partie de A qui n'est pas dans B
- A atop B : (A in B) U (B out A)
- A xor B : (A out B) U (B out A)
- Arithmétique : $k_1AB + k_2A + k_3B + k_4$

Composition 2

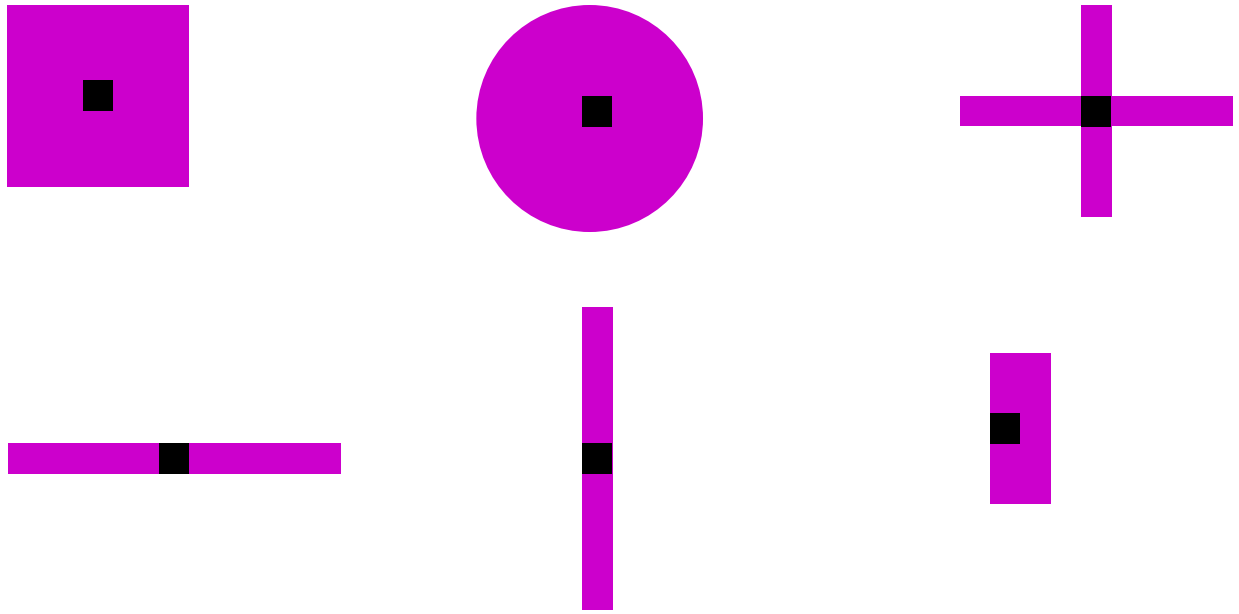
Exemple :



arithmétique : 0.5*A + 0.5*B

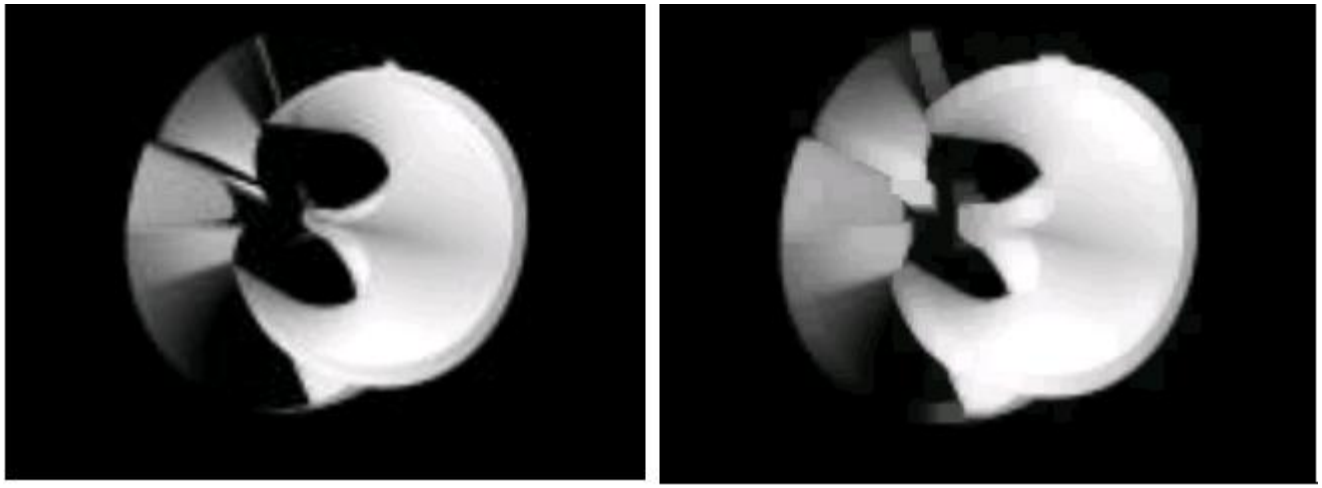
Morphologie

Eléments structurants



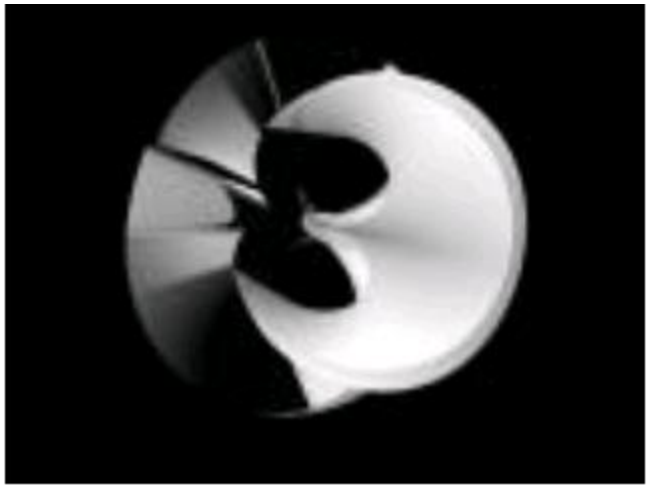
Morphologie : dilatation

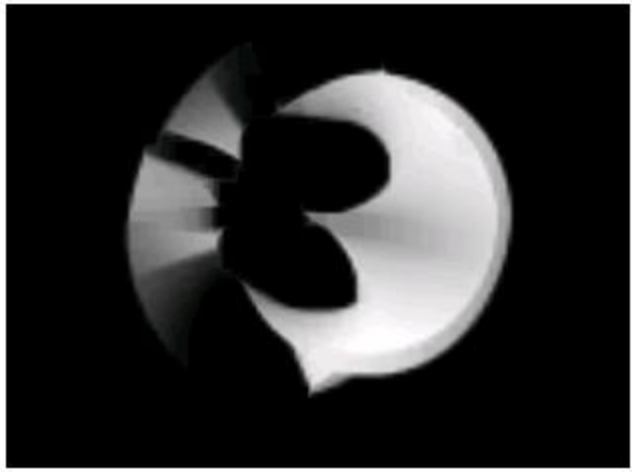
La dilatation est un opérateur de morphologie mathématique qui dilate les éléments blancs d'une image noir-blanc :



Morphologie : érosion

L'érosion est un opérateur de morphologie mathématique qui érode les éléments blancs d'une image noir-blanc :





Convolution

Pour chacun des pixels, on calcule une moyenne pondérée de sa valeur et des valeurs de ses 8 voisins, par exemple:

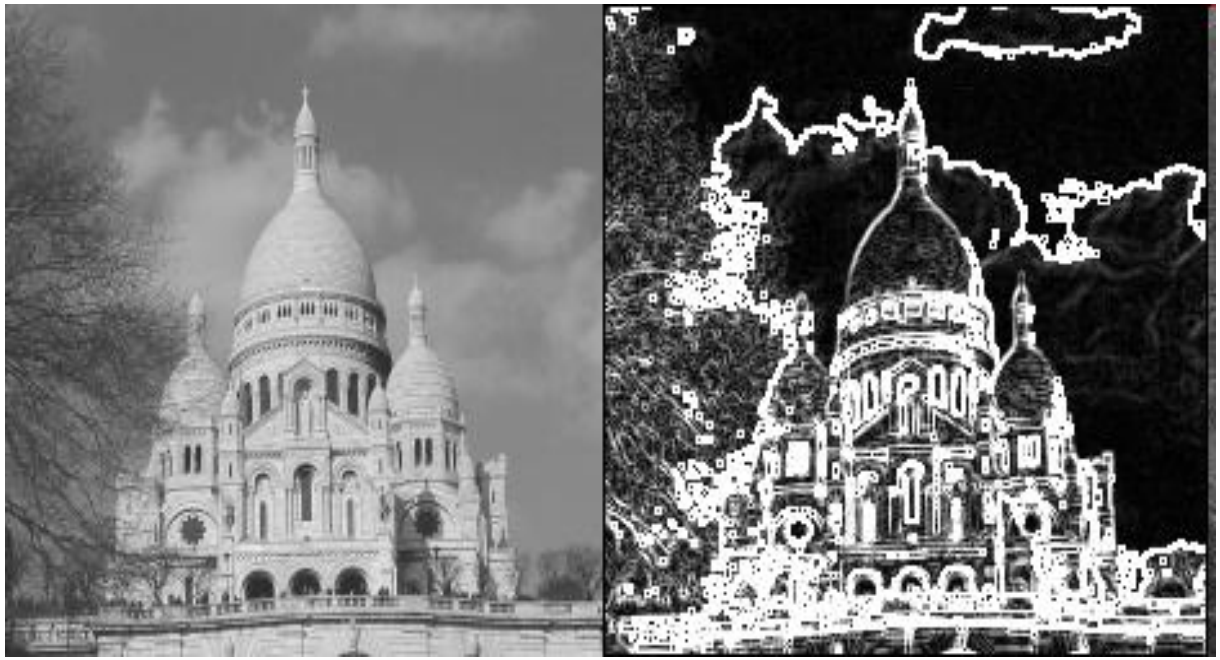
A	**B**	**C**	**1**	**2**	**1**
D	**P**	**E**	**2**	**4**	**2**
F	**G**	**H**	**1**	**2**	**1**

$$P' = (A+2B+C+2D+4P+2E+F+2G+H)/16$$

Convolution : contours

Matrice

```
 0  -1   0
-1   4  -1
 0  -1   0
```



Dégradés de couleurs

Les dégradés de couleur permettent une transition douce entre plusieurs couleurs lors du remplissage d'une forme géométrique

- Dégradé linéaire :

  Transition le long d'un vecteur

- Dégradé radial :

  Transition radiale par rapport à un point

Dégradé linéaire

Le gradient linéaire est défini de gauche à droite

stop : permet d'indiquer les changements de couleur

offset : position en % du changement de couleur

stop-color : définit la couleur à cette position

Exemple du passage de rouge à bleu :

```
<linearGradient id="un_nom">
    <stop offset="0%" stop-color="red"/>
    <stop offset="100%" stop-color="blue"/>
</linearGradient>
```

Sens du dégradé

Changement de sens du dégradé

De droite à gauche :

```
<linearGradient id="dg" xlink:href="#un_nom"
      x1="100%"  y1="0%"  x2="0%" y2="0%" />
```

De haut en bas :

```
      x1="0%"  y1="0%"  x2="0%" y2="100%"
```

En diagonale :

```
      x1="0%"  y1="0%"  x2="100%" y2="100%"
```

Dégradé radial

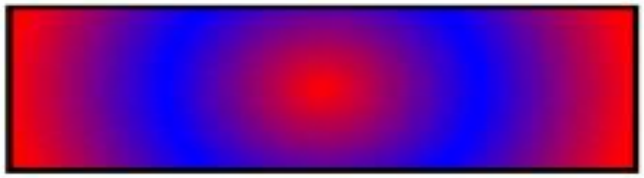
Le dégradé radial permet de définir des cercles concentriques changeant de façon continue de couleur.

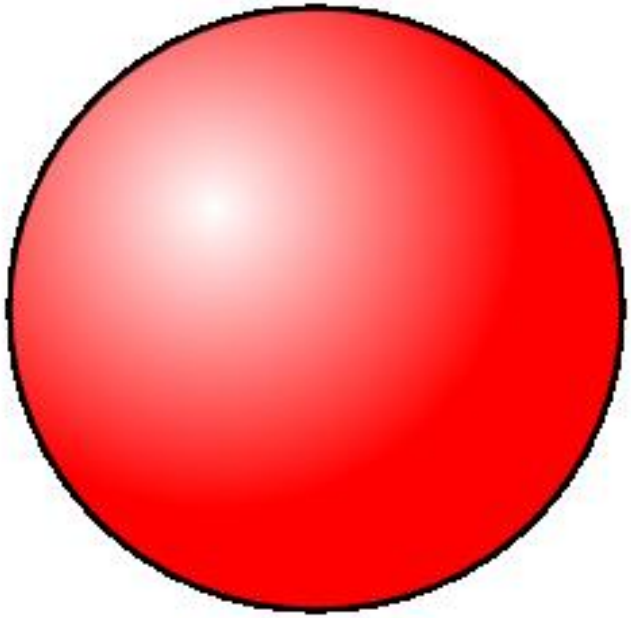
Exemple de rouge à vert en passant pas bleu :

```
<radialGradient id="un_nom"
                    cx ="50%"  cy="50%"  r="100%">
   <stop offset="0%" stop-color="red"/>
   <stop offset="50%" stop-color="blue"/>
   <stop offset="100%" stop-color="green"/>
</radialGradient>
```

Exemple de dégradés

radial





linéaire

Dégradé radial: boule

```
<radialGradient id="MonDegrade" gradientUnits="userSpaceOnUse"
                    cx="300" cy="300" r="300" fx="300" fy="300">
    <stop offset="0%" stop-color="white" />
    <stop offset="100%" stop-color="red" />
  </radialGradient>
……
<circle fill="url(#MonDegrade)" stroke="black" stroke-width="5"
                cx="400" cy="400" r="300"/
```

Texture

Une texture permet de remplir une surface, c'est un motif qui se répète autant verticalement que horizontalement.

Exemple: un petit cercle qui se répète

```
<pattern id="montagne" x="0" y="0" width="10"
        height="10" patternUnits="userSpaceOnUse">
    <rect x="0" y="0" width="10" height="10" fill="#ffffff"/>
    <circle cx="5" cy="5" r="3" fill="#000000"/>
</pattern>
```

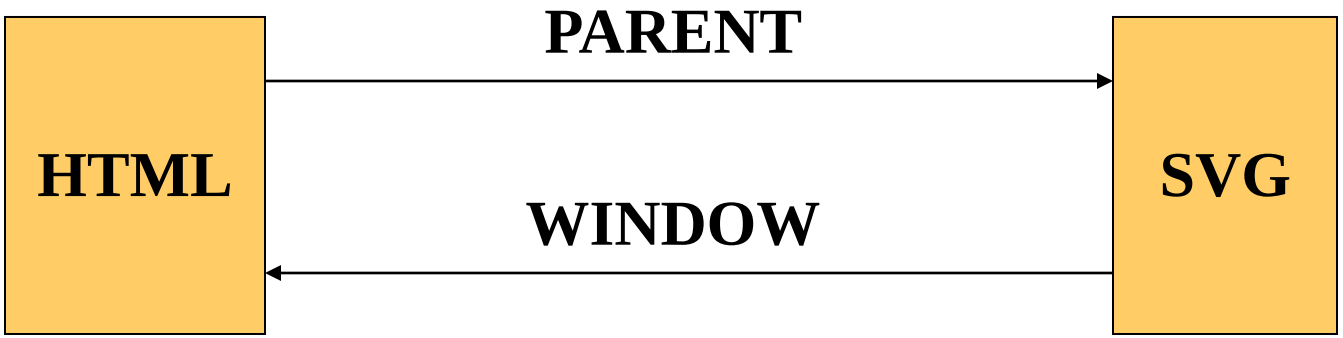
SVG dans HTML

La balise **<embed>** permet d'intégrer un document SVG dans un document HTML :

```
<div align="center">
 <H2>SVG dans HTML</H2>
 <embed src="fichier.svg"
        type="image/svg+xml"
        width="?" height="?"/>
</div>
```

SVG ⇔ HTML

A l'ouverture du document, un lien doit être fait entre les fonctions 'javascript'.



Parent permet d'accéder à une fonction côté HTML depuis SVG, **Window** inversément.

HTML => SVG

Des données d'un formulaire mettent à jour un graphique directement

Côté SVG :

```
onload="init()"
function init() {
    parent.majSVG = changeColHtml;…….}
function changeColHtml(val) {…….}
```

Côté HTML :

```
onchange="majSVG(this.value)"
```

SVG => HTML

Des données obtenues sur un graphique sont transmises à un formulaire

Côté SVG :

```
parent.majFormulaire(val);
```

Côté HTML :

```
function majFormulaire (val) {
    document.formulaire["name"].value = val; }
```

Base de données

Accès à une base de données (ici PostgreSQL) avec PHP :

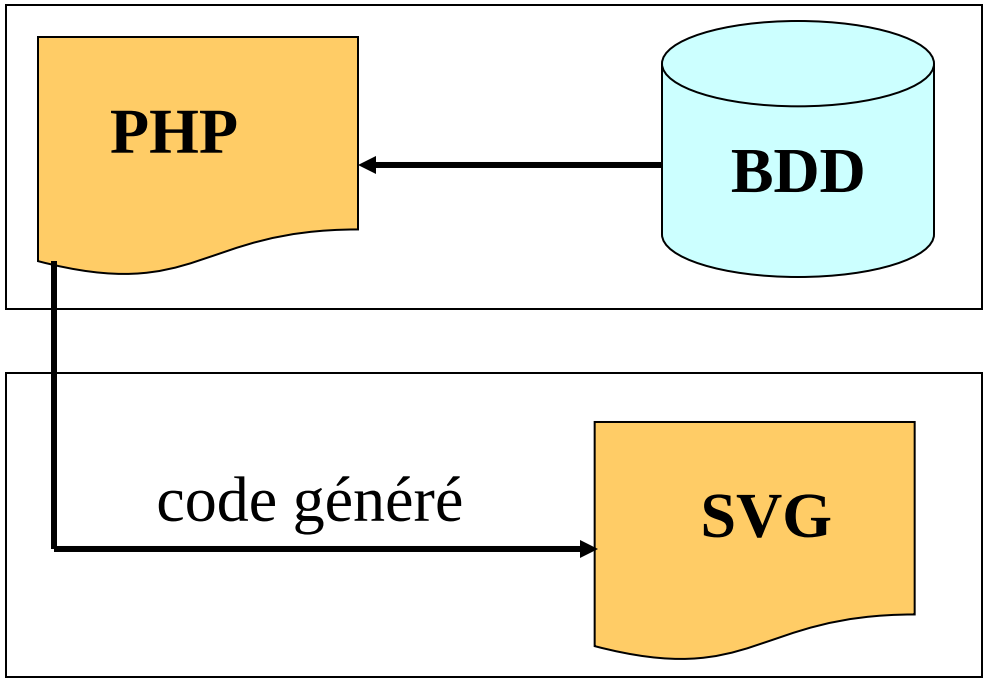
- Connection: pg_connect
- Ecriture d'une requête:
  $requete="select ……. ";
- Exécution de la requête:
  $resultat=pg_query($requete);
- Récupérarion des résultats:
  $variable=pg_fetch_all($resultat);

PHP => SVG

Affichage graphique de données d'une BDD

Côté serveur :

Côté client :



PHP => SVG 2

Une méthode:

- PHP intégré dans du HTML

  balise <embed> dans HTML

```
<?php
.....
?>
<embed src='fichier.svg' ……>
```

PHP => SVG 3

Une seconde méthode:

```
<?php
  header("Content-type: image/svg+xml");
  echo '<?xml version="1.0" encoding="iso-8859-1"?>';
?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN" "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xml:space="default" width="500" height="400">
  ……
</svg>
```

SVG => PHP

Mise à jour d'une BDD à partir de SVG

Côté serveur :

Côté client :