Standard Vector Graphics (SVG):

un linguaggio standard per la grafica 2D ed il web

Multimedia – Prof. Sebastiano Battiato

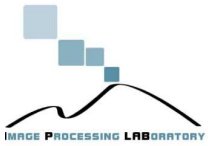
Outline

- ✓ SVG overview e RoadMap (fonte w3c)
- ✓ Cosa è SVG?
- ✓ Primitive di Base
- ✓ Path, il testo
- ✓ Le Animazioni
- ✓ I filtri (Filter Effects) e l'interattività

SVG Overview (1/2)

SVG is a platform for two-dimensional graphics. It has two parts: an XML-based file format and a programming API for graphical applications. Key features include shapes, text and embedded raster graphics, with many different painting styles. It supports scripting through languages such as ECMAScript and has comprehensive support for animation.

SVG is used in many business areas including Web graphics, animation, user interfaces, graphics interchange, print and hardcopy output, mobile applications and high-quality design.



SVG Overview (2/2)

SVG is a royalty-free vendor-neutral open standard developed under the W3C Process. It has strong industry support; Authors of the SVG specification include Adobe, Apple, Canon, Corel, Ericsson, HP, IBM, Kodak, Macromedia, Microsoft, Nokia, Sharp, Sun Microsystems. SVG viewers are deployed to over 100 million desktops, and there is a broad range of support in many authoring tools.

SVG builds upon many other successful standards such as XML (SVG graphics are text-based and thus easy to create), JPEG and PNG for image formats, DOM for scripting and interactivity, SMIL for animation and CSS for styling.

SVG is interoperable. The W3C release a test suite and implementation results to ensure conformance.

SVG Roadmap (OLD)

Document	FWD	*Next WD*	LC	CR	PR	REC
SVG 1.0	11 Feb 1999	-	03 Mar 2000	02 Aug 2000	19 July 2001	5 Sep 2001
SVG 1.1	30 Oct 2001	-	15 Feb 2002	30 Apr 2002	15 Nov 2002	14 Jan 2003
SVG Mobile Profiles	30 Oct 2001	-	15 Feb 2002	30 Apr 2002	15 Nov 2002	14 Jan 2003
SVG Mobile1.2	9 Dec 2003	-	13 Aug 2004	*[Nov 2004]*	*[Mar 2005]*	*[May 2005]*
SVG 1.2	11 Nov 2002	-	27 Oct 2004	*[Dec 2004]*	*[Mar 2005]*	*[May 2005]*
DOM Level 3 Events	01 Sep 2000	-	31 Mar 2003	*[Dec 2004]*	*[Mar 2005]*	*[May 2005]*
DOM Level 3 XPath	18 Jun 2001	-	28 Mar 2002	31 Mar 2003	*[Mar 2005]*	*[May 2005]*
sXBL	01 Sep 2004	22 Nov 2004	*[Feb 2005]*	*[Apr 2005]*	*[Sep 2005]*	*[Nov 2005]*
SVG Print	15 July 2003	*[Jan 2005]*	*[Mar 2005]*	*[Jun 2005]*	*[Sep 2005]*	*[Nov 2005]*
Authoring Tool Guidelines	*[Feb 2005]*	-	-	-	-	-
Accessibility Techniques	*[May 2005]*	-	-	-	-	-

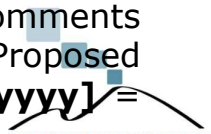
Legend: **FWD** = First working draft; **LC** = last call for comments (i.e., last WD); **CR** = Candidate Recommendation; **PR** = Proposed Recommendation; **REC** = W3C Recommendation. **[mmm yyyy]** = expected date.

SVG Roadmap (recent)

Roadmap of Deliverables

Specification	FPWD	LC	CR	PR	Rec
Compositing	February 2008	July 2008	August 2008	February 2009	May 2009
Filters	May 2007	March 2008	May 2008	October 2008	February 2009
Gradients	see Paint Servers				
Layout Requirements and Use Cases	May 2008	September 2008			
Layout	December 2008	July 2009	September 2009	November 2009	February 2010
Masking and Clipping	July 2008	December 2008	February 2009	May 2009	July 2009
Media Access Events	October 2006	January 2008	March 2008	October 2008	December 2008
Paint Servers	April 2009	August 2009	January 2010	April 2010	May 2010
Print	May 2007	January 2008	March 2008	October 2008	December 2008
Transformations	June 2008	November 2008	January 2009	February 2009	June 2009
Vector Effects	September 2008	January 2009	March 2009	October 2009	December 2009
WebFonts	June 2008	December 2008	February 2009	May 2009	July 2009
SVG 1.1 Full 2nd Edition					January 2009
SVG 1.2 Tiny	December 2003	September 2008	November 2008	November 2008	December 2008
SVG 1.2 Full Modular	November 2008	April 2009	July 2009	December 2009	June 2010
SVG 2.0	January 2009	April 2009	July 2009	December 2009	June 2010

Legend: **FWD** = First working draft; **LC** = last call for comments (i.e., last WD); **CR** = Candidate Recommendation; **PR** = Proposed Recommendation; **REC** = W3C Recommendation. **[mmm yyyy]** = expected date.

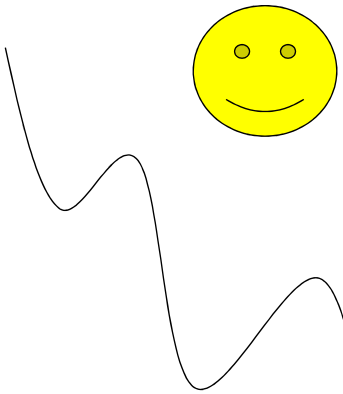


SVG: Alcuni chiarimenti..

- SVG is a language for describing two-dimensional graphics and graphical applications in XML.
  - **SVG 1.1 is a W3C Recommendation and is the most recent version of the full specification**.
  - **SVG Tiny 1.2 is a W3C Recommendation, and targets mobile devices.**
- There are various SVG modules under development which will extend previous versions of the specification, and which will serve as the core of future SVG developments.
- The SVG Working Group is currently working in parallel on a set of modules, for extending prior specifications, and a new specification, **SVG 2.0**, which will combine those modules with the rest of the SVG framework to work across the full range of devices and platforms.
- Older specifications include SVG 1.0 and the SVG Mobile Profiles: SVG Basic and SVG Tiny which were targeted to resource-limited devices and are part of the 3GPP platform for third generation mobile phones.
- SVG Print is a set of guidelines to produce final-form documents in XML suitable for archiving and printing.

Cosa è SVG ?

- SVG (Scalable Vector Graphics) è un'applicazione di XML per la rappresentazione di **oggetti grafici** in forma *compatta* e *portabile.*
- (**Esigenza**) Esiste un forte interesse a creare uno standard comune di grafica vettoriale utilizzabile sul web.

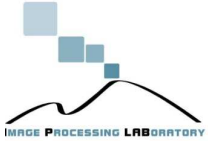


Cosa è SVG

- SVG è un linguaggio per descrivere oggetti grafici bidimensionali in XML.
- SVG ammette tre tipi di oggetti grafici:
  - Forme grafiche vettoriali (Paths formati da linee e curve)
  - Immagini (raster)
  - Testo
- Gli oggetti grafici possono:
  - Essere raggruppati
  - Avere uno stile (legami con CSS)
  - Essere trasformati (ruotati, ridimensionati, traslati)

Cosa è SVG

- I testi possono
  - Essere strutturati in linguaggio XML (favorendo accessibilità e ricerca)
- I Disegni SVG possono essere dinamici e interattivi attraverso:
  - la gestione degli eventi di HTML
  - Un linguaggio di scripting (ECMA script = Javascript)
  - Il DOM



Alcuni Esempi

Stick's Computed Dream:

http://www.dotuscomus.com/svg/SCD.svg

Rolex:

http://www.gnote.org/svg-images/Rolex.svg

Eluzions' Optical Illusions:

http://eluzions.com/Illusions/Distortion/Cafèwall.svg

*SVG & Net Art (*Luigia Cardarelli)

http://xoomer.virgilio.it/lcardar/

Other: Adobe Filter Demo, Eye-Chart, Dale's SVG Game

Vantaggi di SVG

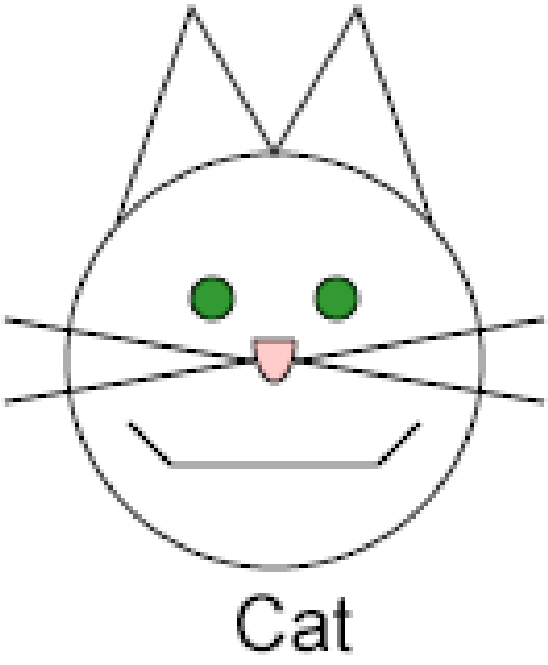
- ✓ **Open Source**: I file sono scritti in un linguaggio comune a tutti (niente formati binari incomprensibili)
- ✓ **Web Based**: SVG apre il web alla grafica vettoriale (occupa poco spazio)
- ✓ **XML Based**: Integrazione con altri linguaggi XML (Xhtml, MathML, ecc.) ed estensibilità

Un esempio concreto di scalabilità

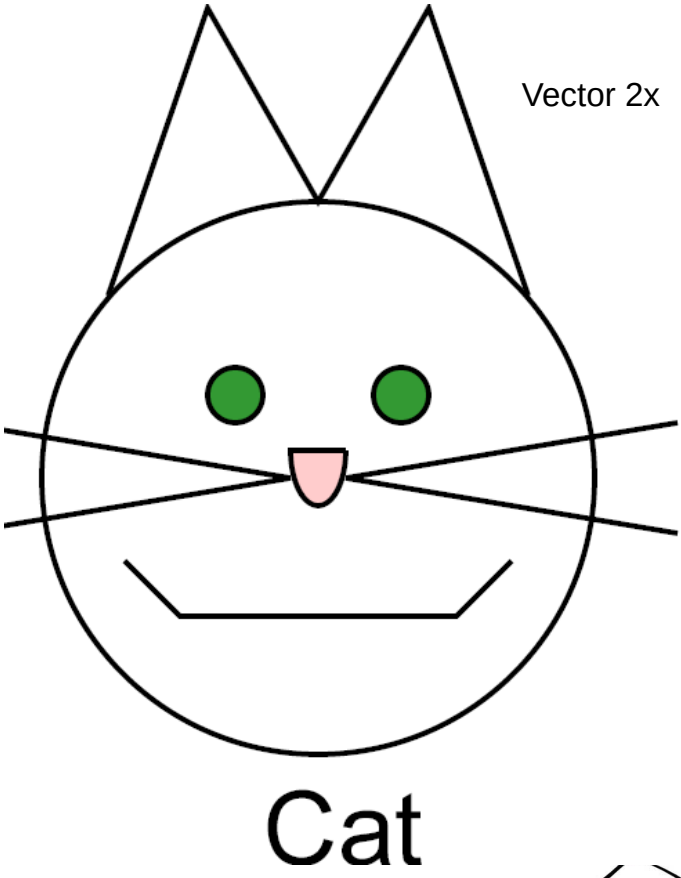
Raster
(140x170)

Vector
(140x170)

Raster 2x



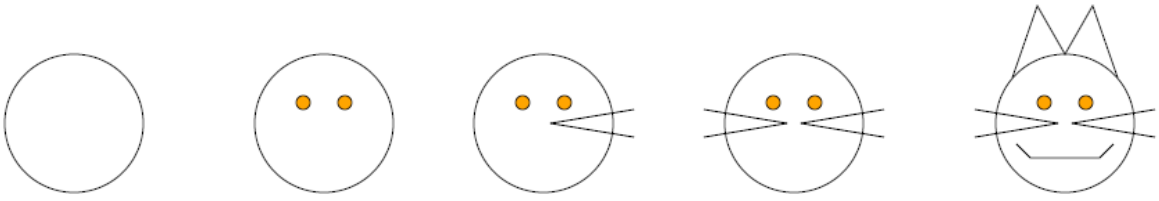
Vector 2x

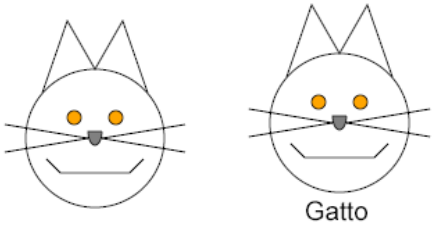


Struttura di un documento SVG

```
<?xml version=“1.0”?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG
   1.0//EN" "http://www.w3.org/TR/2001/REC-SVG-
   20010904/DTD/svg10.dtd">
<svg width="140" height="170">
<title>Titolo</title>
 <desc>Descrizione</desc>
</svg>
```

L’esempio del gatto (7 passi)

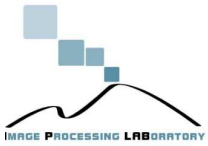




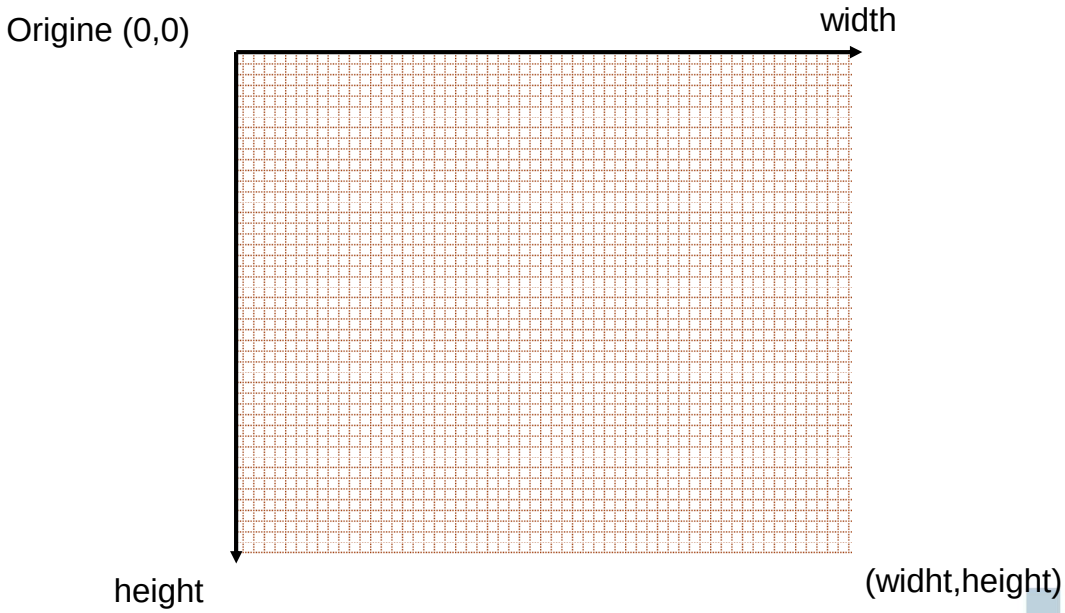

```
<svg width="140" height="170">
<title>Cat</title>
<desc>Stick Figure of a Cat</desc>
<circle cx="55" cy="55" r="50" id="head" />
<ellipse cx="35" cy="45" rx="10" ry="10" id="eye1" />
<ellipse cx="75" cy="45" rx="10" ry="10" id="eye2" />
<line x1="55" y1="55" x2="55" y2="105" />
<polyline points="25,110 45,120 95,120 105, 110" />
</svg>
```

Il sistema di riferimento

- ✓ Il “mondo” di SVG è un foglio infinito.
- ✓ Tuttavia occorre stabilire le dimensioni della tela (canvas) e il sistema di riferimento in quest’area.
- ✓ L’area di lavoro in SVG si chiama **viewport**.
- ✓ Le dimensioni del viewport sono settate dagli attributi *width* e *height* del tag di apertura <svg>
- ✓ Es. <svg width=“400” height=“400”>



Il sistema di riferimento di default (in pixels)



Primitive grafiche di base

Le primitive grafiche di SVG sono:

- ✓Linee
- ✓Rettangoli
- ✓Cerchi ed ellissi
- ✓Poligoni
- ✓Polylines

Gli stili in SVG

In SVG gli stili possono essere specificati in quattro modi (dal più importante al meno importante)

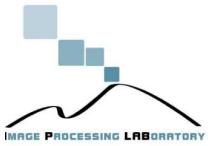
1. Inline styles
2. Internal stylesheets
3. External stylesheets
4. Attributi

Stili *inline*

- ✓Lo stile dell'oggetto viene definito nell'elemento tramite l'attributo *style;*

- ✓Lo schema dello stile è:

proprietà1:valore1; proprietà2:valore2;…



Stili interni

- ✓Vengono specificati dentro l'elemento <svg> e dentro uno speciale elemento <**defs**> che vedremo in seguito.

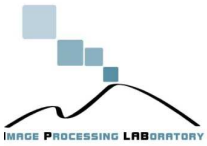
- ✓Lo stile viene definito da

<**style** type="text/css"><! [CDATA [circle { fill:blue; fill-opacity: 0.5; }]]</style>

Tutti i cerchi definiti saranno riempiti di blu, e con opacità del 50%

Fogli di stile esterni

- ✓I fogli di stile esterni sono file di testo che raccolgono definizioni di stile che vanno applicate a più documenti SVG.
- ✓L'estensione di un foglio di stile è .css (*Cascading Style Sheets*)
- ✓Viene richiamato tra la definizione del documento .xml e il DOCTYPE



Esempio

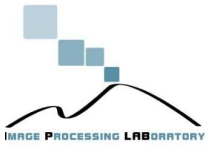
```
<?xml version="1.0"?>
<?xml-stylesheet href="ext_style.css"
    type="text/css"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
   "http://www.w3.org/TR/2001/REC-SVG-
    20010904/DTD/svg10.dtd">
<svg width="200px" height="200px" viewBox="0 0 200
    200"
   preserveAspectRatio="xMinYMin meet">

<line x1="10" y1="10" x2="40" y2="10"/>
<rect x="10" y="20" width="40" class="yellow" cx="70"
    cy="20" r="10"/>
<polygon class="thick" points="60 50, 60 80,  90 80"/>
<polygon height="30"/>
<circle class="thick semiblue"
   points="100 30, 150 30, 150 50, 130 50"/>
</svg>
```

```
{ fill:none; stroke: black; }
/* default for all elements */
rect { stroke-dasharray: 7 3;}
circle.yellow { fill: yellow; }
.thick
{ stroke-width: 5; }
.semiblue
{ fill:blue; fill-opacity: 0.5; }
```

Raggruppare gli elementi: **g**

- ✓ Gli oggetti SVG possono essere raggruppati e richiamati.
- ✓ Il tag **<g id="nome_gruppo">** apre la definizione degli elementi del gruppo.
- ✓ Dentro l'elemento **g** è possibile inserire i tags <title> e <desc>.
- ✓ È possibile stabilire uno stile che viene ereditato dagli elementi interni.



Richiamare i gruppi: **use**

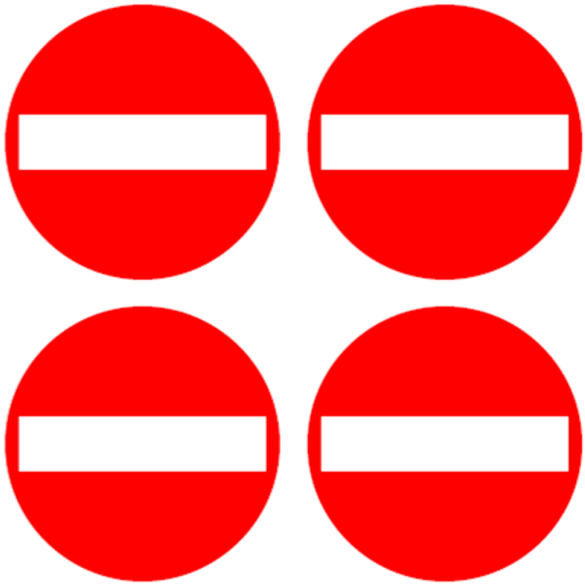
- ✓ Tramite il tag **<use>** definito da:

<**use** xlink:href="#id_elemento" x="" y=""/>

- ✓ Vengono ridisegnati tutti gli elementi di **<g>** nella posizione traslata rispetto all'originale di **x** pixels in orizzontale e **y** pixels in verticale (x, y possono assumere anche valori negativi)

Esempio

```
<svg width="500" height="500">
<title>Divieto di accesso</title>
<desc>Un segnale stradale</desc>
<g id="divietoaccesso">
<circle cx="50" cy="50" r="50"
    style="fill:red"/>
<rect x="5" y="40" width="90" height="20"
    style="fill:white"/>
</g>
<use xlink:href="#divietoaccesso" x="110"
    y="0"/>
<use xlink:href="#divietoaccesso" x="0"
    y="110"/>
<use xlink:href="#divietoaccesso" x="110"
    y="110"/>
</svg>
```



Definizioni: <defs>

È possibile definire degli elementi generici mettendoli tra <**defs**> e </**defs**>.

Tali elementi:

- ✓Non vengono immediatamente disegnati;
- ✓Possono “subire” futuri cambiamenti di stile;
- ✓Riutilizzo più semplice.

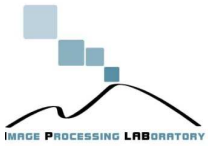
L’elemento <**use**> può far riferimento a qualsiasi risorsa disponibile in rete. È possibile quindi creare un file svg con sole definizioni e poi richiamarli negli altri file svg.

Trasformazioni del sistema di coordinate

In SVG è possibile effettuare delle trasformazioni sul piano:

- ✓Traslazioni;
- ✓Scaling;
- ✓Rotazioni;

Tutte queste azioni vengono eseguite tramite l'attributo **trasform** da applicare ai relativi elementi (gruppi, primitive, ecc.).



Traslazioni

✓Le traslazioni le abbiamo già viste quando nell'elemento <**use**> abbiamo specificato gli attributi x e y che indicavano lo spostamento orizzontale e verticale dell'oggetto da riutilizzare.

✓In generale la sintassi della traslazione è:

<… **transform**="translate(x-value,y-value)"

x-value e **y-value** rappresentano (in pixels) lo spostamento orizzontale e verticale del sistema di riferimento dell'elemento a cui è applicato.

Scaling

- ✓ Lo **scaling** permette il ridimensionamento delle unità di misura del sistema di riferimento.
- ✓ La conseguenza di tale trasformazione è che l'oggetto risulta ingrandito o rimpicciolito o riflesso.

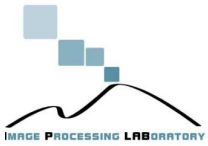
- ✓ La sintassi della trasformazione è:

<.. transform="scale(r)">

se si vogliono ridimensionare entrambi gli assi in modo eguale

<.. transform="scale(rx,ry)">

se il fattore di scala è diverso per i due assi.



Rotazione

- ✓ La trasformata rotazione permette di ruotare il sistema di riferimento dell'oggetto secondo un centro di rotazione e un angolo.
- ✓ Se il centro di rotazione non viene espresso la rotazione viene effettuata rispetto all'origine.
- ✓ L'angolo viene espresso in gradi.

Sintassi

<… transform="rotate(angle,centerx,centery);"

<… transform="rotate(angle);"

Combinare le trasformazioni

- ✓ Dentro l'attributo **transform** è possibile effettuare qualsiasi combinazione di trasformazioni separandole con lo spazio

Esempio:

<… **transform**="rotate(30) translate(120,130) scale(2)">

- ✓ Ovviamente l'ordine delle trasformazioni è determinante. E' inoltre possibile piegare gli assi di un certo angolo tramite le trasformate

skewX(angolo) **skewY**(angolo)

Esempio

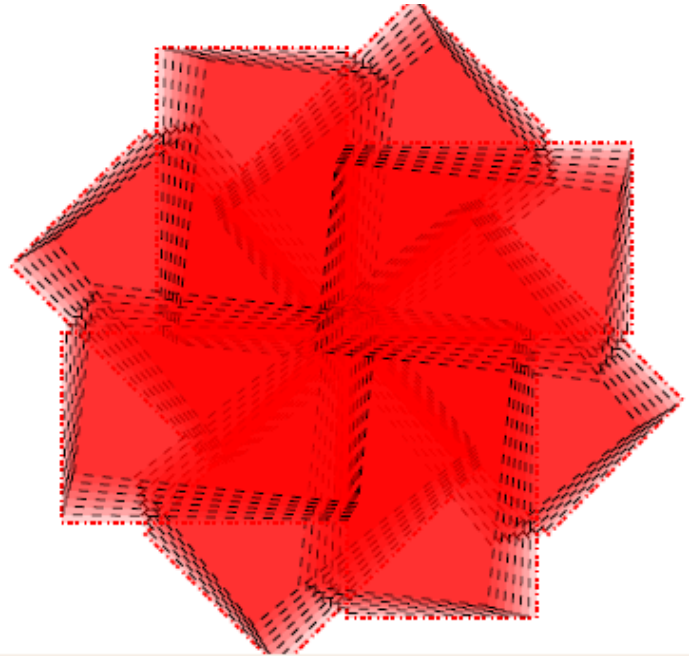
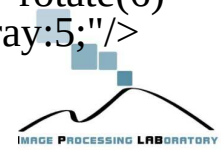


IMAGE PROCESSING LABORATORY

Esempio: codice SVG (1/2)

```
<defs> <g id="petalo">
<rect x="100" y="50" width="150" height="100" style="fill:red;fill-
    opacity:0.2; stroke:red;stroke-width:2;stroke-dasharray:3,2,1;"/>
<rect x="100" y="50" width="150" height="100" transform="rotate(1)"
    style="fill:red;fill-opacity:0.2; stroke:black;stroke-dasharray:5;"/>
<rect x="100" y="50" width="150" height="100" transform="rotate(2)"
    style="fill:red;fill-opacity:0.2; stroke:black;stroke-dasharray:5;"/>
<rect x="100" y="50" width="150" height="100" transform="rotate(3)"
    style="fill:red;fill-opacity:0.2; stroke:black;stroke-dasharray:5;"/>
<rect x="100" y="50" width="150" height="100" transform="rotate(4)"
    style="fill:red;fill-opacity:0.2; stroke:black;stroke-dasharray:5;"/>
<rect x="100" y="50" width="150" height="100" transform="rotate(5)"
    style="fill:red;fill-opacity:0.2; stroke:black;stroke-dasharray:5;"/>
<rect x="100" y="50" width="150" height="100" transform="rotate(6)"
    style="fill:red;fill-opacity:0.2; stroke:black;stroke-dasharray:5;"/>
</g> </defs>
```

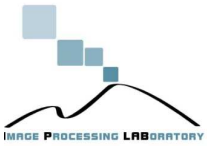


Esempio: codice SVG (2/2)

```
<use xlink:href="#petalo" transform="rotate(-135,100,150)"/>
<use xlink:href="#petalo" transform="rotate(-45,100,150)"/>
<use xlink:href="#petalo" transform="rotate(-90,100,150)"/>
<use xlink:href="#petalo" transform="rotate(45,100,150)"/>
<use xlink:href="#petalo" transform="rotate(135,100,150)"/>
<use xlink:href="#petalo" transform="rotate(90,100,150)"/>
<use xlink:href="#petalo" transform="rotate(180,100,150)"/>
<use xlink:href="#petalo"/>
```

PATHS

- Tutti le primitive descritte finora sono *scorciatoie* per la nozione più generale di <**path**>
- L'elemento PATH crea forme tramite linee, curve e archi che possono essere adiacenti o partire da un punto arbitrario del canvas.
- L'elemento PATH ha un solo attributo (escluso lo stile) denominato **d** (che sta per data) che contiene una serie di:
  - Comandi (spostati_la,disegna una linea, crea un arco)
  - Coordinate
- È possibile riempire un PATH con l'attributo di stile **fill:color.** Valgono le stesse regole descritte per i poligoni tramite l'attributo fill-rule

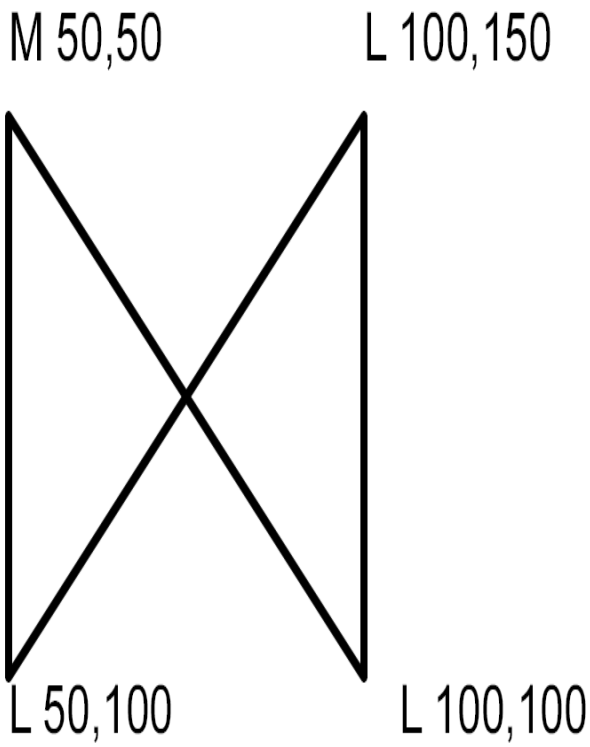


PATHS: M, L, Z.

- Il comando **M** x y (**moveto**) sposta il pennino sul punto di coordinate (x,y);
- Ogni PATH deve iniziare con un **moveto**;
- Il comando **L** x' y' (**lineto**) disegna una linea dal punto corrente fino a (x',y')
- Il comando **Z** (**closepath**) chiude il cammino unendo il punto corrente con quello iniziale.
- Inserendo una nuova **M** tra i comandi del **path**, si crea un nuovo sottocammino, cioè viene alzato il pennino e spostato nella nuova posizione pronto a ricevere altri comandi.

Esempio

<path d="
M 50,50
L 50,100
L 100,150
L 100,100
Z "
style=""">



Esercizio:

- ✓ Scrivere il path generico relativo alle primitive grafiche:
  - ✓ Rect
  - ✓ Polyline
  - ✓ Polygon

Soluzione: Rect

<rect x="startx" y="starty" width="larghezza"
height="altezza"/>

Diventa

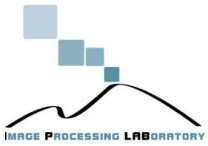
<path d="M startx starty

L startx+larghezza,starty

L startx+larghezza,starty+altezza

L startx,starty+altezza

Z>



Coordinate relative

- Esiste una versione "relativa" dei comandi **M** ed **L**. Per relativo si intende rispetto al punto precedente. I comandi "relativi" sono gli stessi ma in forma minuscola **m** ed **l**. Le coordinate dopo **m** e **l** sono espresse in pixels e possono anche assumere valori negativi.
- Esistono poi dei veri e propri "*Shortcuts*" per linee verticali ed orizzontali:
  - **H**,(**h**) x' : disegna una linea orizzontale dalla posizione corrente al punto (x-attuale+x', y-attuale);
  - **V**,(**v**) y': equivalente verticale;

Il rettangolo “relativo”

<rect x=“startx” y=“starty” width=“larghezza”
height=“altezza”/>

Diventa:

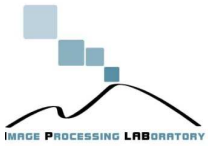
<path d=“M startx starty

l larghezza,0

l 0,altezza

l –larghezza,0

Z>



Il rettangolo: parte terza

<rect x=“startx” y=“starty” width=“larghezza”
height=“altezza”/>

Diventa:

<path d=“M startx starty

h larghezza

v altezza

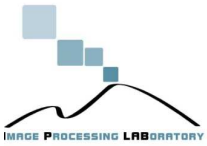
h –larghezza

Z>

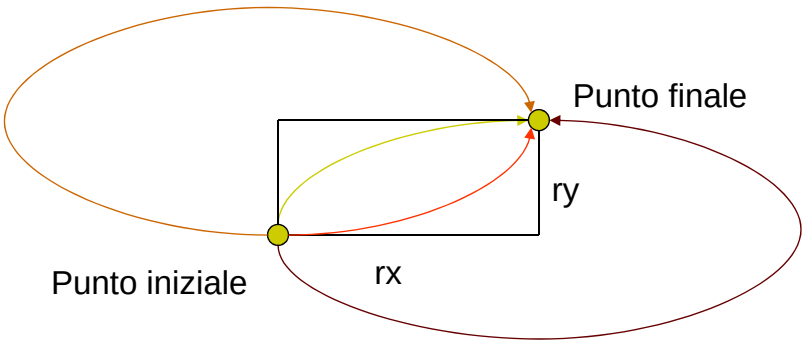
PATH: Archi ellittici

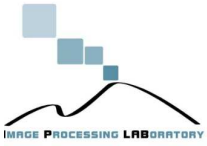
- Gli archi rappresentano il tipo più semplice di curva.Vengono dichiarati con il comando **A**

- Un arco è definito da:
  - Raggio-orizzontale
  - Raggio-verticale
  - Large-arc-flag
  - Sweep-flag
  - Endx
  - Endy

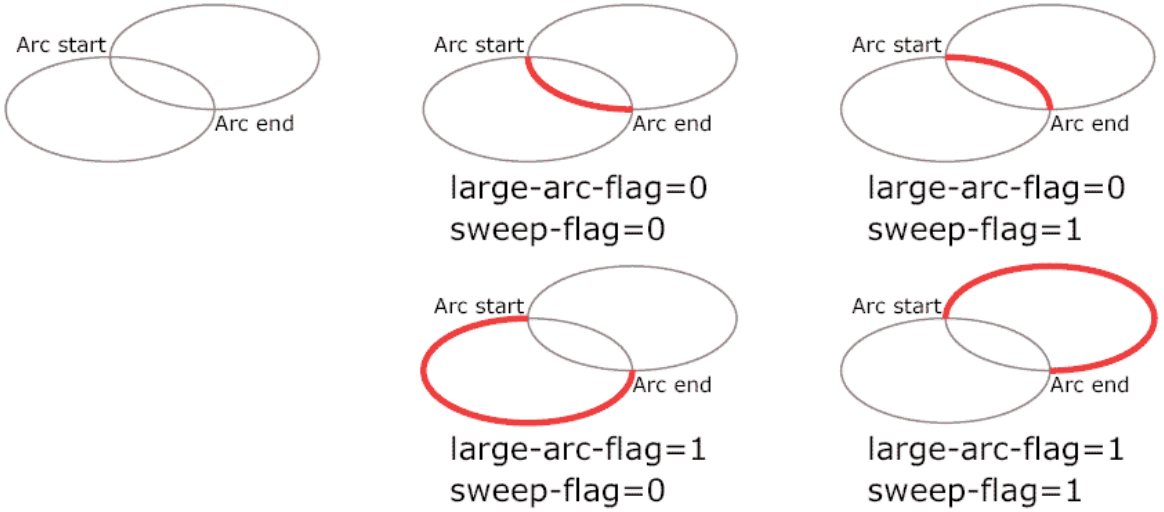


Archi: Teoria





Archi ellittici



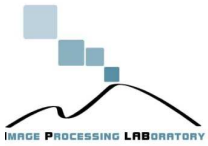
Curve di Bezier

- ✓ Le curve di Bezier (da Pierre Bezier della Renault e indipendemente anche da Paul de Casteljau della Citroen) introducono un modo computazionalmente conveniente di creare curve parametriche di secondo e terzo grado.
- ✓ Sono costituite dai punti di inizio e fine curva, più una serie di punti di controllo.
- ✓ Metaforicamente un punto di controllo è come un magnete che attira la linea (come se fosse deformabile) con + forza tanto quanto + il punto è vicino.

Curve di Bezier quadratiche

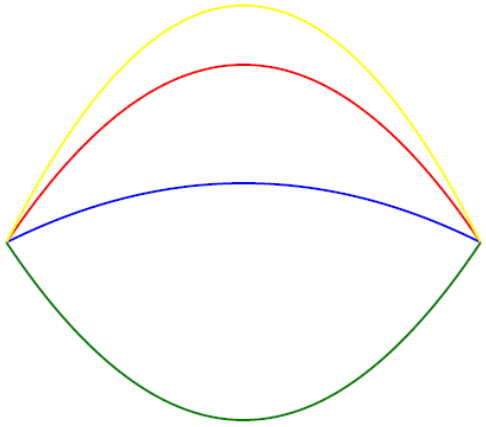
- Sono caratterizzate da:
  - Punto iniziale
  - Punto finale
  - Un punto di controllo
- La sintassi è (dentro un PATH)

  <.. **Q** controlpointx controlpointy endx endy>
- Esiste anche la versione **q** (con coordinate relative)



Esempi

```
<path d="M 100 250 Q 200 100 300 250" style="stroke:red; fill:none;" />
<path d="M 100 250 Q 200 200 300 250" style="stroke:blue; fill:none;"/>
<path d="M 100 250 Q 200 50 300 250" style="stroke:yellow; fill:none;"/>
<path d="M 100 250 Q 200 400 300 250" style="stroke:green; fill:none;"/>
```

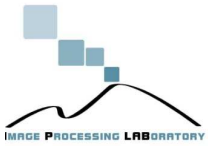


Polybezier

- ✓ Si ottengono specificando altre coppie di control point e punto finale

<path… **Q** cpx1 cpy1 endx1 endx1 cpx2 cpy2 endx2 endx2>

- ✓ Se si vuole ottenere uno smoothing della curva si può usare T dopo la prima coppia.

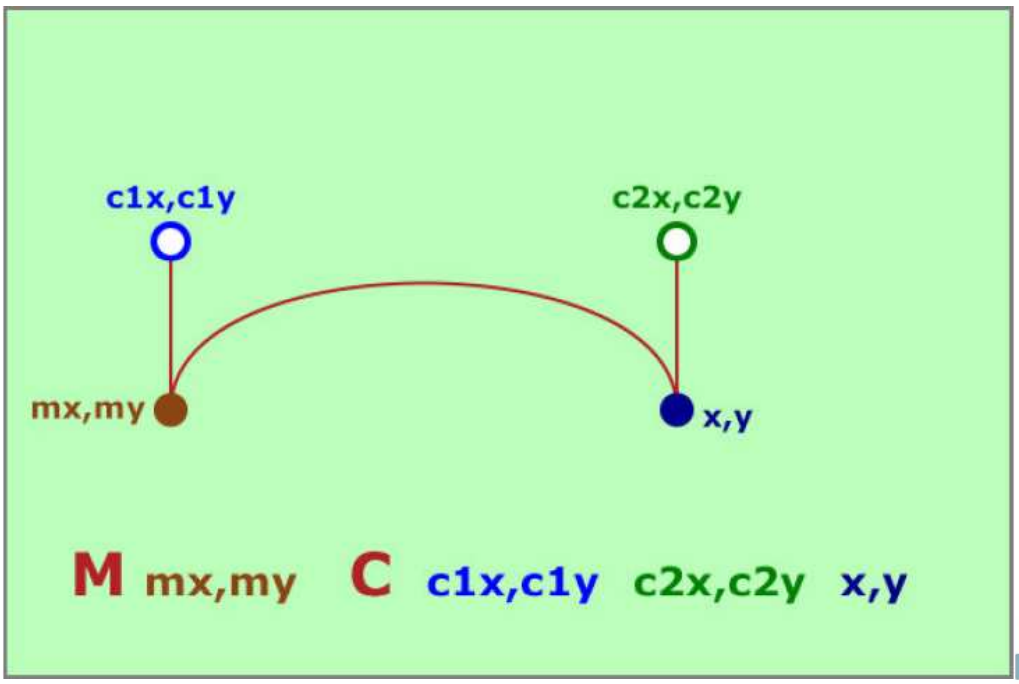


Curve di Bezier cubiche

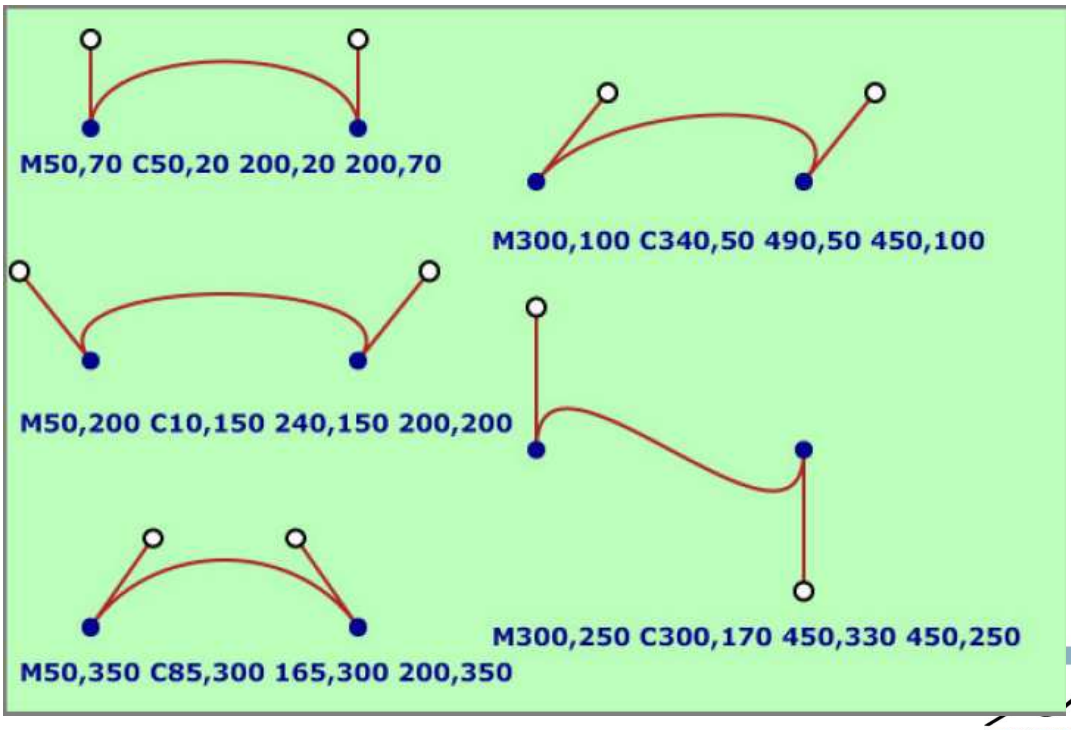
- ✓ Le curve di Bezier cubiche hanno due punti di controlli piuttosto che uno.
- ✓ La sintassi è:

<… **C** cpx1 cp1y cp2x cp2y endx endy >

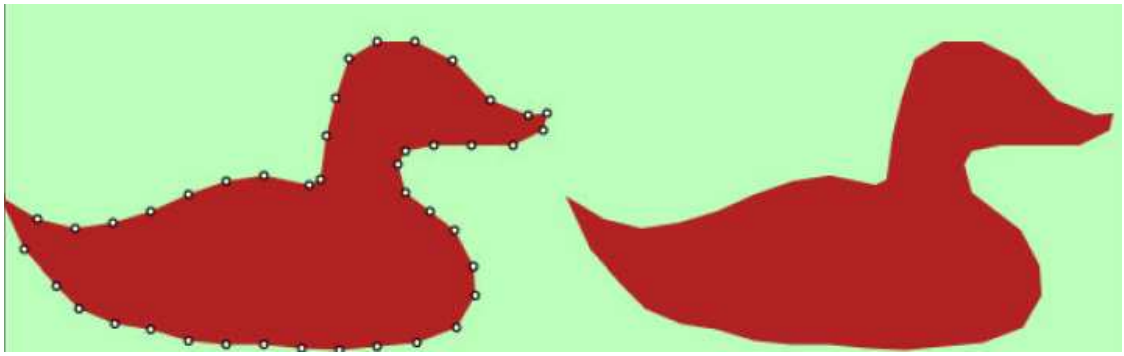
- ✓ Esiste la versione relativa c
- ✓ Esistono le polybezier cubiche con e senza smooth (comando S)



I Punti di controllo

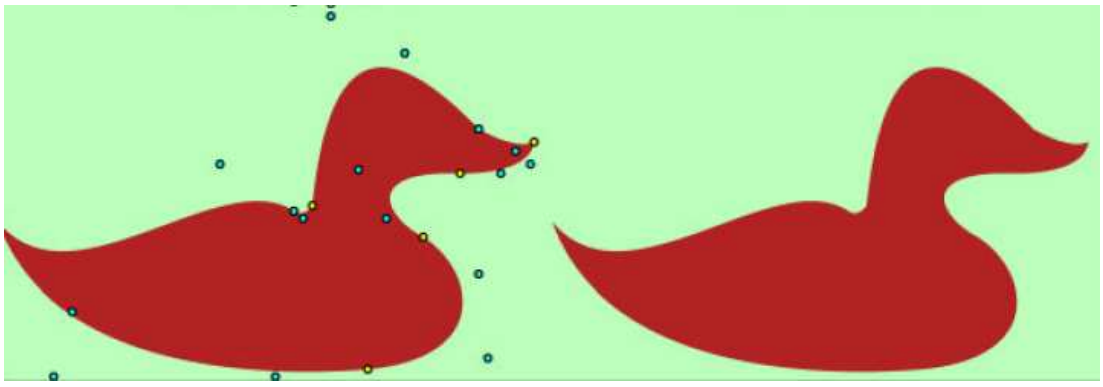


Red Duck!!! (1/2)



<path d="M 0 112 L 20 124 L 40 129 L 60 126 L 80 120 L 100 111 L 120 104 L 140 101 L 164 106 L 170 103 L 173 80 L 178 60 L 185 39 L 200 30 L 220 30 L 240 40 L 260 61 L 280 69 L 290 68 L 288 77 L 272 85 L 250 85 L 230 85 L 215 88 L 211 95 L 215 110 L 228 120 L 241 130 L 251 149 L 252 164 L 242 181 L 221 189 L 200 191 L 180 193 L 160 192 L 140 190 L 120 190 L 100 188 L 80 182 L 61 179 L 42 171 L 30 159 L 13 140 Z"/>

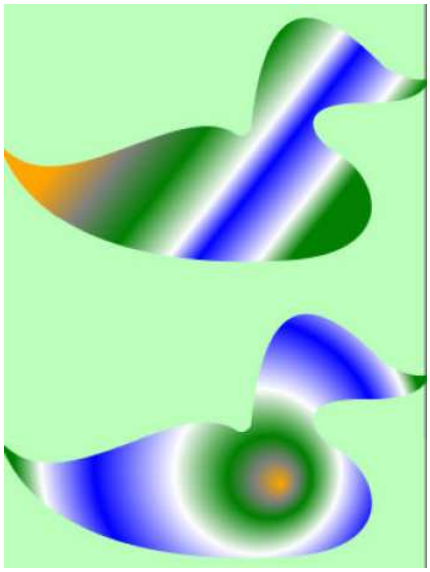
Red Duck!!! (2/2)



<path d="M 0 312
C 40 360 120 280 160 306 C 160 306 165 310 170 303
C 180 200 220 220 260 261 C 260 261 280 273 290 268
C 288 280 272 285 250 285 C 195 283 210 310 230 320
C 260 340 265 385 200 391 C 150 395 30 395 0 312 Z"/>

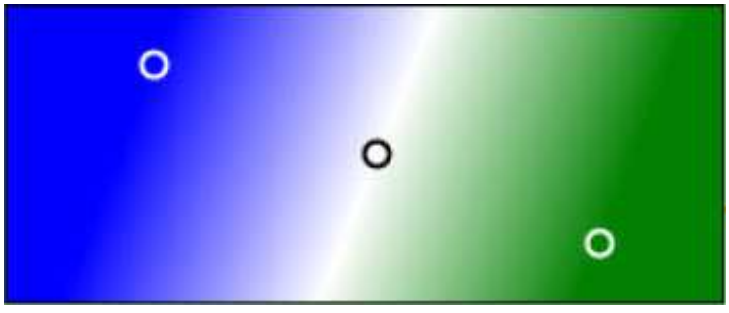
I Color Gradient

- La proprietà <**fill**> permette inoltre di realizzare effetti colore particolari tramite l'utilizzo dei gradienti. E' possibile riempire un oggetto con delle gradazioni di colore tra due estremi in maniera sia ***lineare*** che ***radiale.***



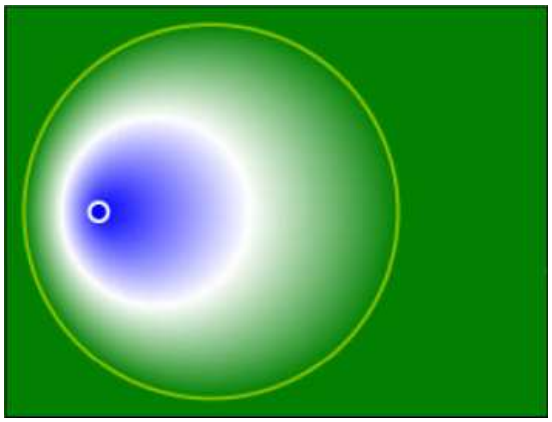
<linearGradient>

<**linearGradientid**="MyGradient" gradientUnits="userSpaceOnUse"
x1="80" y1="44“ x2="260" y2="116">
<stop offset="0" style="stop-color:blue"/>
<stop offset="0.5" style="stop-color:white"/>
<stop offset="1" style="stop-color:green"/>
</linearGradient>
<rect x="20" y="20" width="290" height="120"
style="fill:url(#MyGradient)"/>



<radialGradient>

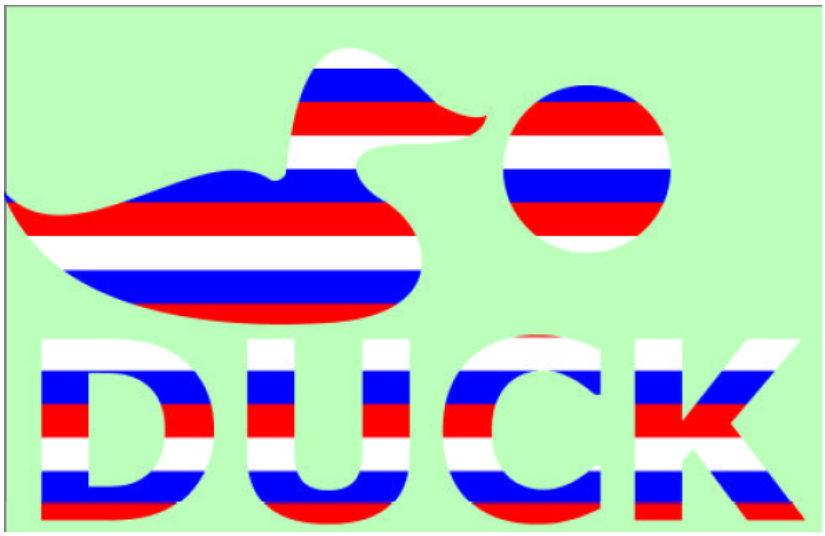
```
<radialGradient id="MyGradient2" gradientUnits="userSpaceOnUse“
 cx="130“ cy="270" r="100" fx="70" fy="270">
<stop offset="0" style="stop-color:blue"/>
<stop offset="0.5" style="stop-color:white"/>
<stop offset="1" style="stop-color:green"/>
</radialGradient>
<rect x="20" y="160" width="290" height="220" style="fill:url(#MyGradient2)"/>
```



La circonferenza di centro ***(cx,cy)*** *e raggio* ***r*** *si riferisce all'offset=1*

Il fuoco ***(fx,fy)*** *individua il punto con offset=0*

Clipping



E' possibile clippare un "disegno" (un gruppo di elementi) rispetto ad un dato oggetto grafico mendiante la primitiva **<clipPath>**

Sintassi <clipPath>

```
<clipPath id="myClip">
<circle cx="350" cy="100" r="50"/>
</clipPath>

<g style="stroke:none;clip-path:url(#myClip)">
<rect style="fill:red" x="0" y="0" width="500" height="20" />
<rect style="fill:white" x="0" y="20" width="500" height="20" />
<rect style="fill:blue" x="0" y="40" width="500" height="20" />
<rect style="fill:red" x="0" y="60" width="500" height="20" />
<rect style="fill:white" x="0" y="80" width="500" height="20" />
<rect style="fill:blue" x="0" y="100" width="500" height="20" />
<rect style="fill:white" x="0" y="120" width="500" height="20" />
<rect style="fill:blue" x="0" y="160" width="500" height="20" />
</g>
```

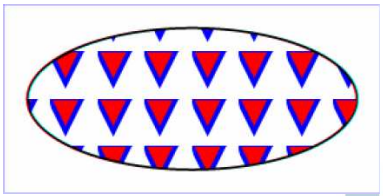

Pattern

- ✓ SVG permette di riempire (**fill**) gli oggetti con *pattern* grafici. La definizione di un pattern segue la sintassi classica:

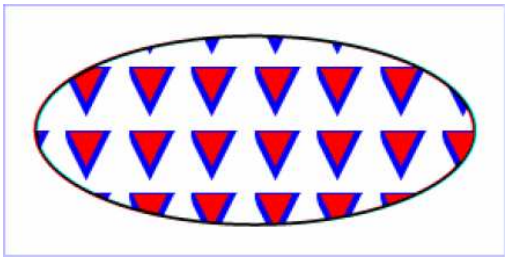
<**pattern** id=“” x=“”, y=“”, width=“”, heigth=“”,
patternTrasform=“”,
patternUnits=“UserSpaceOnUse”>

….

</**pattern**>



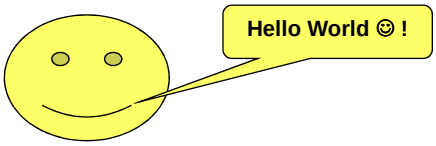
Esempio



```
<defs>
<pattern id="TrianglePattern" patternUnits="userSpaceOnUse"
x="0" y="0" width="100" height="100" >
<path d="M 0 0 L 7 0 L 3.5 7 z" fill="red" stroke="blue" />
</pattern>
</defs>

<ellipse fill="url(#TrianglePattern)" stroke="black" stroke-width="5"
cx="400" cy="200" rx="350" ry="150" />
```

Adding Text to SVG



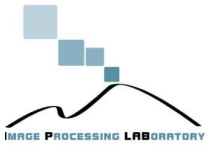
- ✓ Se è vero che ogni disegno racconta una storia, è perfettamente giusto usare le parole per aiutare a raccontare la storia. SVG gestisce il testo tramite l'elemento <text> ed i suoi numerosi attributi di stile. La sintassi::

<text x="" y="" style=""> Testo </text>

fill:color (crea un font filled)
stroke: color (crea un font outlined)
font-size: x
font-weight: (bold,normal)
font-style: (italic, normal)
word-spacing: x
letter-spacing: x
text-decoration: (underline, overline, line-trought)
font-family: (serif, sans-serif, monospace)

Esempio

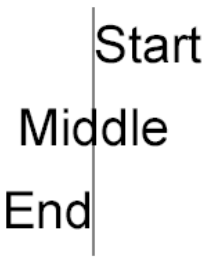
Testo normale
Testo con stroke black, width .5, fill:none
Testo con stroke black, width .5, fill:red
Testo Famiglia Serif
Testo Famiglia Sans-Serif
Testo Famiglia Monospace
Testo Font size 50
Testo font-weight:bold
Testo font-style:italic
Testo text-decoration:overline
Testo text-decoration:underline
~~Testo text-decoration:line-through~~
Testo con spaziatura caratteri 30
T e s t o c o n s p a z i a



Allineamento del testo

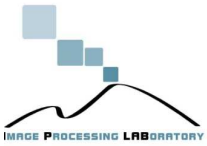
- ✓ Il punto finale del testo inserito non è noto *a priori*.
- ✓ L'allineamento del testo viene eseguito intorno al punto iniziale tramite l'attributo di stile:

text-anchor: start | middle | end



Cambio di stile inline

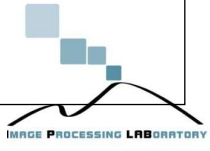
- Tramite l'elemento <**tspan** style=“”> è possibile cambiare lo stile del testo in una sessione <text …> </text>.
- **tspan** è analogo di span in HTML. Gli attributi di **tspan** sono:
  - dx, dy: spostano i caratteri interni di dx e dy pixel rispetto ai precedenti
  - x,y: spostano i caratteri interni nella posizione x,y
  - baseline-shift: sub | super



Esempio

Questo e' un unico blocco di testo
un po' italico o anche normale o **Grassetto**
c
a
d
e e ritorna normale
H_2

```
<g font-size="24pt">
<text x="0" y="24">
Questo e' un unico blocco di testo
<tspan x="0" y="48" style="font-
style:italic">
un po' italico</tspan>
o anche normale o
<tspan style="font-weight:bold">
Grassetto</tspan>
<tspan x="0" y="64" dy="10 20 30
40">cade e ritorna normale</tspan>
<tspan x="0" y="200">H
<tspan style="baseline-
shift:sub;">2</tspan></tspan>
</text>
```



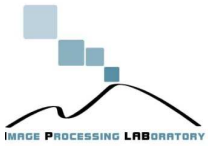
Testo verticale

- ✓ Il testo in verticale si può ottenere in 3 modi:
- ✓ Con una rotazione di 90 gradi

  <**text** x="" y="" transform(rotate(90))
- ✓ Con la proprietà di stile writing-mode: tb

  (effetto analogo al precedente)
- ✓ Se si vuole che il testo sia scritto in verticale occorre aggiungere la proprietà

  glyph-orientation-vertical:0;

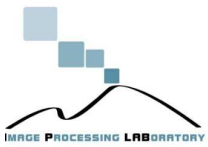


Testi che seguono un path

- ✓ I testi non devono essere necessariamente in orizzontale o verticale ma possono seguire una forma arbitraria.
- ✓ In SVG ciò si ottiene tramite l'elemento

  <**textPath** xlink:href="#nomepath"> </textPath>

  che punta un path definito precedentemente.
- ✓ Il testo avrà in ogni punto la baseline perpendicolare al path che sta seguendo.
- ✓ E' possibile aggiustare l'inizio del text aggiungendo l'attributo startOffset con un valore in percentuale.

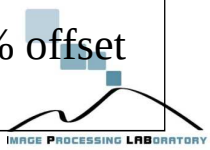


Esempio

Testo in curva di Bezier

Testo in curva di Bezier

```
<defs>
<path id="path1" d="M 100 100 C 200 50 300 150 400 100"/>
</defs>
<text style="font-size:24;">
<textPath xlink:href="#path1">
Testo in curva di Bezier </textPath>
</text>
<text style="font-size:24;" transform="translate(0,100)" >
<textPath xlink:href="#path1" startOffset="20%">
Testo in curva di Bezier 20% offset
</textPath> </text>
```



Animazioni in SVG

- ✓ SVG permette di creare animazioni basandosi su SMIL2 (Synchronized Multimedia Integration Language Level2).
- ✓ In questo sistema vengono specificati
  - ✓ i valori iniziali e finali di attributi, colori e trasformazioni che si vogliono animare;
  - ✓ Il momento di inizio dell'animazione
  - ✓ La durata dell'animazione

Esempio Basic

```
<rect x="10" y="10"
    width="500" height="200"
    style="stroke: black; fill:
    none;">
    <animate
        attributeName="width"
        attributeType="XML"
        from="500" to="100"
        begin="0s" dur="5s"
        fill="freeze" />
</rect>
```

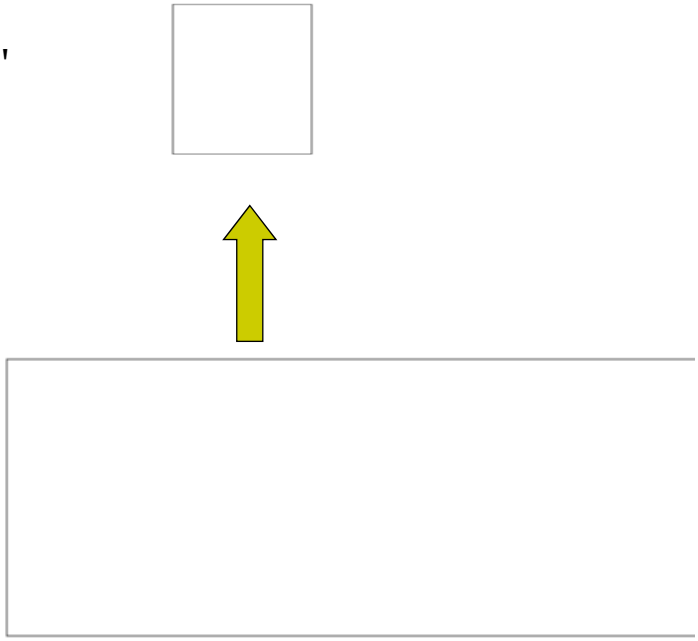


IMAGE PROCESSING LABORATORY

Animazioni...

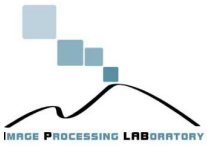
Alias: Animazioni multiple su un singolo oggetto

- ✓ In SVG è possibile creare animazioni multiple a diversi attributi dell'elemento.
- ✓ Le animazioni multiple si ottengono specificando dentro l'elemento la sequenza delle animazioni tramite il tag **<animate>**
- ✓ Gli attributi degli elementi possono essere di tipo:
  - ✓ XML (se si riferiscono alla natura di un oggetto)
  - ✓ CSS (se si riferiscono ad elementi di stile)

IMAGE PROCESSING LABORATORY

Animazione di più oggetti

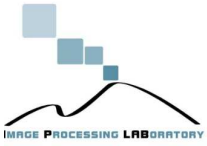
- ✓ Ogni “oggetto” di SVG può essere animato tramite l’attributo <**animate**> dentro l’elemento.
- ✓ Le animazioni avvengono “contemporaneamente” in base alle indicazioni temporali dell’animazione stessa.
- ✓ Il “cronometro” di SVG parte quando il documento è interamente caricato e si stacca quando la finestra del visualizzatore viene chiusa.
- ✓ Il tempo 🕐 si esprime in:
  - ✓ hh:mm:ss
  - ✓ mm:ss
  - ✓ Valore_numerico unità_di_tempo (1h, 3.5min)
  - ✓ Le unità di tempo sono: h, m, s, ms



E gli attributi non numerici ?

- ✓ Fino ad adesso abbiamo visto come creare delle animazioni facendo variare i valori numerici di alcuni attributi degli elementi (XML, CSS).
- ✓ Esistono attributi che hanno valori alfanumerici (tipo: **visibility: hidden|visible**)
- ✓ Questi attributi vengono animati da:

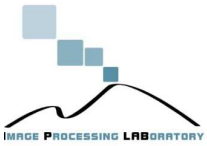
<**set** attributeName=“” attributeType=“” to=“” begin=“” dur=“” fill=“”>



Animazioni: Colori

- ✓ L'elemento <**animate**> non funziona con i colori e quindi con gli attributi a valore colore.
- ✓ Questa animazione si esegue con l'elemento

<**animateColor** attributeName=“”
begin=“”
dur=“”
from=“*red*”
to=“*yellow*” fill=“”>



Animazioni: Trasformazioni

- ✓ Similmente a quanto detto sui colori, le trasformazioni necessitano di un tag ad hoc, detto <**animateTransform**>. La sintassi è la seguente:

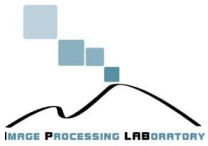
  <**animateTransform** attributeType=“XML” attributeName=“transform” type=“scale” from=“” to=“” begin=“” dur=“” fill=“”>

- ✓ Se si vogliono comporre più trasformazioni occorre specificare l'attributo additive=“sum” in <animateTransform>. Di default l'attributo additive=“replace”, cioè la trasformazione rimpiazza la precendente.

Animazioni: Motion

- Fino ad adesso le animazioni “spaziali” sono state effettuate in linea retta (cambiando i parametri x,y o con translate>
- Il tag <**animateMotion**> permette di spostare oggetti lungo un PATH arbitrario. È possibile specificare l’attributo rotate=“auto” che automaticamente ruota l’oggetto rispetto al path che segue. La relativa sintassi è la seguente:

  <**animateMotion** path=“” dur=“” fill=“”>



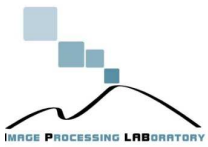
Usare path definiti in precedenza.

- È possibile richiamare dei path definiti precedentemente nella sezione <defs> in questo modo:

<animateMotion dur=“” begin=“” fill=“”>

<mpath xlink:href=“#id_path”/>

</animateMotion>



Animation Control

Come visto in SVG è possibile animare delle primitive grafiche specificando i valori iniziali e finali degli attributi coinvolti in un dato intervallo di tempo. L'animazione, per default, si realizza in modo ***lineare.***

E' possibile specificare un modo diverso mediante la primitiva <**calcMode**>

Interattività

In SVG si può fare riferimento ad eventi esterni, avviati dall'utente (es: il **click** del mouse, il **mouseover**, il **mouseout**, ecc.)

Gli eventi possono essere gestiti associandogli degli script (si utilizza il linguaggio ECMAscript) che vengono mandati in esecuzione allo scatenarsi dell'evento stesso.

Lista Eventi

- Gli eventi principali gestiti da SVG sono:

click: evento scatenato dal click del mouse su di un elemento grafico;

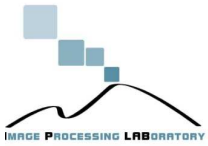
mousemove: evento associato al movimento del mouse;

mouseover: evento scatenato dal passaggio del mouse su di un oggetto grafico;

mouseout: evento scatenato quando il puntatore del mouse abbandona l'area di un elemento grafico;

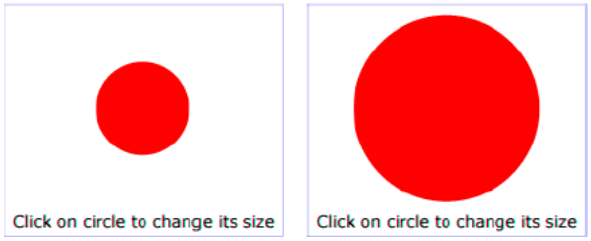
mousedown: evento associato alla pressione del tasto del mouse su di un elemento grafico;

load: evento scatenato quando il documento SVG viene caricato dal visualizzatore.



Gli script

Circle1.svg

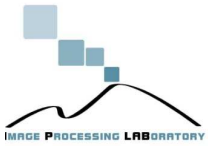


E' possibile utilizzare un vero e proprio linguaggio di scripting ECMAScript (standardizzato e basato su JavaScript) per la gestione degli eventi. Esempio:

```
<script type="text/ecmascript"> <![CDATA[
function circle_click(evt) {
var circle = evt.target;
var currentRadius = circle.getAttribute("r");
if (currentRadius == 100)
   circle.setAttribute("r", currentRadius*2);
     else
   circle.setAttribute("r", currentRadius*0.5);
} ]]> </script>
```

Evt, target

- ✓ **Evt** permette di identificare l'elemento grafico a cui abbiamo associato l'evento scatenato e tale informazione viene passata come parametro alle funzioni relative alla gestione degli eventi.
- ✓ Attraverso il metodo **target** siamo in grado di ottenere un riferimento all'elemento grafico a cui è associato l'evento.
- ✓ L'uso di evt.target ci permette quindi di memorizzare facilmente all'interno di una variabile, un riferimento all'oggetto su cui è stato scatenato l'evento.



Scripting:qualche dettaglio

Come succede per un file XML o HTML, quando un documento SVG viene caricato dal visualizzatore, viene creata una struttura interna ad albero che rappresenta il documento.

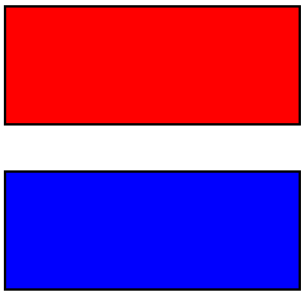
Ad esempio:

```
<svg width="100" height="100">
        <rect x="10" y="10" width="10" height="10"/>
        <circle cx="50" cy="50" r="10"/>
</svg>
```

Ad ogni tag SVG corrisponde un nodo della struttura. I nodi **<rect>** e **<circle>**, essendo definiti all'interno del tag <**svg**>, vengono chiamati nodi figli di **<svg>**, mentre **<svg>** stesso è detto nodo padre. Inoltre il nodo principale del documento **<svg>** viene chiamato nodo root (radice).

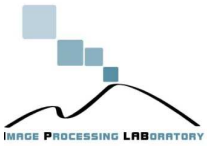
SVG mette a disposizione una serie di metodi, che costituiscono l'interfaccia **DOM** (Document Object Model), per accedere e manipolare i nodi della struttura.

Un Esempio



```
<svg id="elementoRadice" width="300" height="300"
    onload="aggiungiRect()">
      <script type="text/ecmascript"><![CDATA[

       function aggiungiRect(){
           var svgdoc=document.getElementById("elementoRadice");
           var newrect=document.createElement("rect");
           newrect.setAttribute("x",10);
           newrect.setAttribute("y",150);
           newrect.setAttribute("width",250);
           newrect.setAttribute("height",100);
           newrect.setAttribute("style","fill:blue;stroke:black;stroke-width:2;");
           svgdoc.appendChild(newrect);
        }
      ]]></script>
      <rect x="10" y="10" width="250" height="100"
    style="stroke:black;fill:red;stroke-width:2"/>
    </svg>
```



Onde....

Onde.svg

E possibile animare una curva di Bezier, facendone variare in maniera pseudo-casuale i punti di controllo. Nell'esempio il path iniziale da animare è:

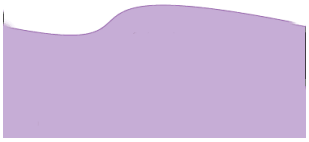
```
<path id="line" style="fill:#905CA8;fill-opacity:0.5;stroke:#905CA8"

d="M 0,150 h 400"/>
```

L'animazione ha inizio al verificarsi dell'evento **onload** del tag **<svg>.** Viene quindi salvato un "puntatore" all'intero documento SVG in una variabile globale. Inoltre viene chiamata la funzione di sistema **setInterval,** che a sua volta si occupa di richiamare iterativamente la funzione **next_frame.**

```
function on_load (event){
svgdoc = event.getCurrentNode().getOwnerDocument();
setInterval ('next_frame()', 100);}
```

Il codice delle onde..1/3



L'animazione è realizzata facendo variare i 2 punti di controllo tenendo comunque fermi i punti iniziali e finali. Una volta generati in valori "target" per i punti di controllo le coordinate correnti vengono gradualmene aumentate o diminuite fino a coincidere esattamente con essi.

Le coordinate correnti dei punti di controllo vengono indicate nel codice con le variabili (x0, y0)e (x1, y1) mentre i valori "target" vengono invece indicati con (tx0, ty0) e (tx1, ty1) .

La funzione **next_frame** innanzitutto si occupa di recuperare, tramite I metodi DOM relativi, l'handler del path da animare:

```
var linenode = svgdoc.getElementById ('line');
if (!linenode)    return;
```

Il codice delle onde..2/3

A questo punto se necessario vengono generati nuovi valori "target" per le coordinate dei punti di controllo. In particolare ciò sara vero alla prima invocazione (valori "target posti ad -1) oppure quando tutte le coordinate correnti hanno raggiunto i valori "target"

```
if (tx0 < 0  || (tx0 == x0 && ty0 == y0 && tx1 == x1 && ty1 ==
y1))  {
  tx0 = Math.floor (400*Math.random());
  ty0 = Math.floor (300*Math.random());
  tx1 = Math.floor (400*Math.random());
  ty1 = Math.floor (300*Math.random());  }
```

Il codice delle onde..3/3

Le coordinate correnti vengono indirizzate verso i valori “target”, ad intervalli di +/- 10 pixel attraverso una ulteriore funzione **change_coord**():

```
x0 = change_coord (x0, tx0);
y0 = change_coord (y0, ty0);
x1 = change_coord (x1, tx1);
y1 = change_coord (y1, ty1);
```

Possono quindi essere cambiati gli elementi dell’atributo “d” del path in questione utilizzando le nuove coordinate:

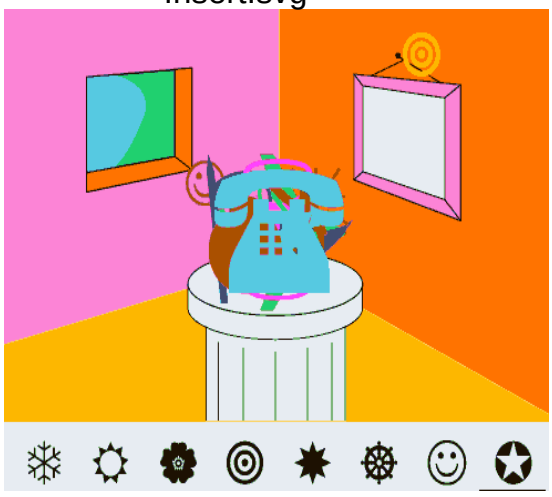
linenode.setAttribute('d', 'M 0, 300 L 0, 150 C‘+x0+','+y0+','+x1+','+y1+', 400,150 L 400, 300 z');

Inserimento dinamico

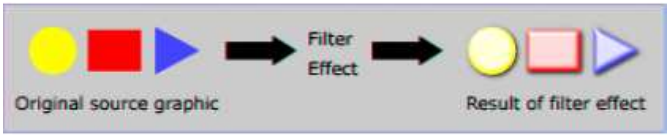
Abbiamo già accennato alla intrinseca struttura ad albero di un documento SVG. Tramite opportuni metodi DOM è possibile accedere alle singole parti.

E’ possibile “stravolgere” la struttura ad albero di un documento SVG operando delle operazioni di pruning, di inserimento, utilizzando solo degli script.

Insert.svg



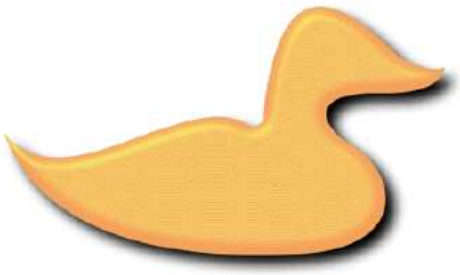
I Filtri in SVG



- ✓ In SVG è possibile utilizzare un insieme di *declarative feature's set* in grado di generare e descrivere *effetti grafici* anche complessi.
- ✓ La possibilità di applicare filtri ad elementi grafici di qualsiasi natura (anche testi) li rende uno strumento flessibile e potente.
- ✓ Un filtro per SVG è costituito da una serie di operazioni che applicate ad una data **sorgente grafica** la modificano, mostrando il risultato direttamente sul *device* finale.

IMAGE PROCESSING LABORATORY

Potenzialità



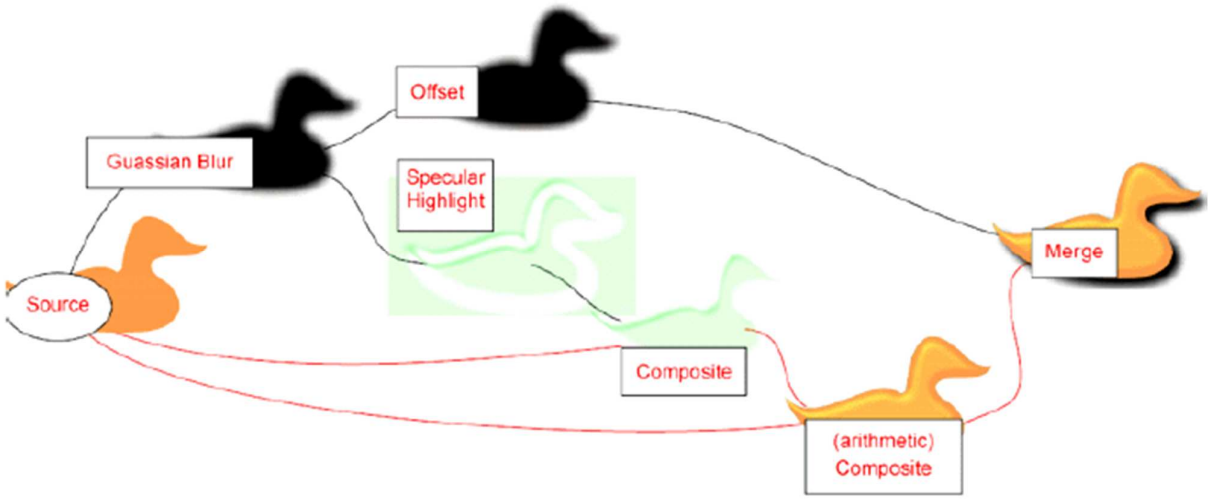
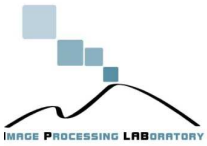


IMAGE PROCESSING LABORATORY

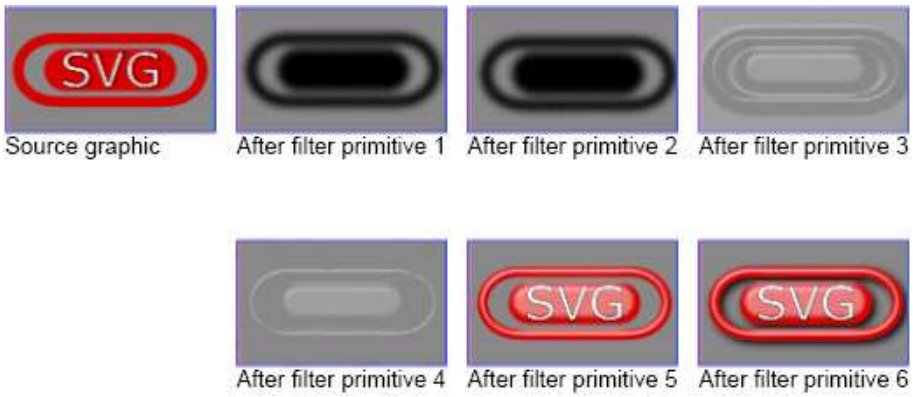
<filter>

- ✓ I filtri si definiscono mediante la primitiva <**filter**>. Il loro utilizzo effettivo si ottiene mediante un **idfilter.**
- ✓ Ciascun elemento <**filter**> contiene poi delle primitive "figlie" che compiono le vere e proprie operazioni grafiche.

- ✓ Il grafico sorgente o l'output di un filtro può essere riutilizzato come input.



Un primo esempio



Attributi di <**filter**>

- La sintassi specifica di ***filter*** è la seguente:

  <**filter** id="…",
  filterUnits="*userSpaceOnUse|objectBoundingBox*",
  primitiveUnits="*userSpaceOnUse|objectBoundingBox*"
  x='',**y**='', **width**='', **heigth**='',
  filterRes='' **xlink:href**=""
  >
  <**fe**…….. >

</**filter**>

Il filtro verrà poi applicato ad un qualsiasi oggetto grafico mediante l'attributo di stile **filter:url(#.....)**

Attributi comuni…

- La sintassi specifica di ciascun ***primitive filter*** o ***filter effect*** è la seguente:

  <**fe**…..
  x='',y='',
  width='', heigth='',
  in='' result=''
  >

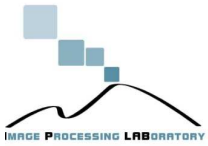
…..insieme ai singoli parametri che ciascuno di essi richiede.

Filter effects: *Gaussian Blur*

Realizza la classica sfocatura regolare, utile ad esempio per creare l'ombreggiatura di primitive grafiche:

<**feGaussianBlur** in="SourceGraphic", stdDeviation="", result="">

Si può utilizzare il valore dell'attributo result come input per ulteriori filtri.



<feOffset> & <feMerge>

- ✓ L'elemento <**feOffset**> trasla l'input:
  <**feOffset** in="" dx="" dy="" result="">

- ✓ L'elemento <**feMerge**> realizza il merging grafico di più sorgenti:

```
<feMerge>
   <feMergeNode in="">
   <feMergeNode in="">
</feMerge>
```

Blending

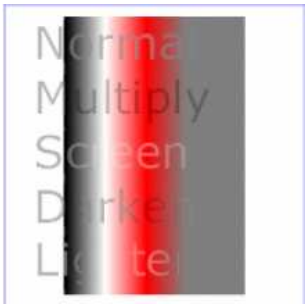
- La primitiva <**feBlend**> sovrappone due oggetti grafici, pixel per pixel, secondo diverse modalità:

normal | *multiply* | *screen* | *darken* | *lighten*

Sintassi:

<**feBlend** in2="", in="", mode="">

Blending (2)



L'opacità dell'immagine risultante *qr,*
date le opacità *qa* e *qb* delle immagini di input A e B, si ottiene:

qr = 1 - (1-qa)(1-qb)*

Il colore finale pixel per pixel *cr*, si ottiene a partire dai colori *ca* e *cb* nelle diverse modalità:

normal	cr = (1 - qa) * cb + ca
multiply	cr = (1-qa)*cb + (1-qb)*ca + ca*cb
screen	cr = cb + ca - ca * cb
darken	cr = Min ((1 - qa) * cb + ca, (1 - qb) * ca + cb)
lighten	cr = Max ((1 - qa) * cb + ca, (1 - qb) * ca + cb)

Turbulence

- ✓ Una delle primitive filtro che nonostante la sua semplicità permette di realizzare interessanti effetti visivi è <**feTurbulence**>, che utilizza la funzione di turbolenza di Perlin:

<**feTurbulence**

in=“”

type=“fractalnoise|turbulence”

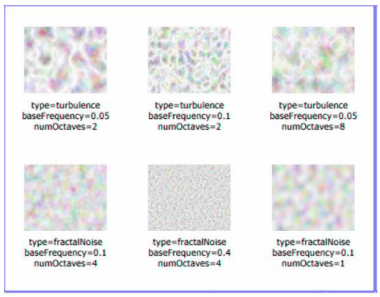
baseFrequency=“”

numOctaves=“”

Seed=“”

</**feTurbulence**>

Esempi di <feTurbulence>



<feMorphology>

La primitiva <**feMorphology**> realizza le classiche operazioni di **Dilation** e di **Erosion**. E' possibile specificare come parametro il raggio (o i raggi) del kernel: è un rettangolo di dimensioni 2*x_radius x 2*y_radius.

La sintassi:

<**feMorphology in=“”**

operator=“dilate|erode”

radius=“”

result=“”/>

<feConvolveMatrix>

Il classico operatore di convoluzione può essere generato attraverso l'utilizzo della primitiva <**feConvolveMatrix**>. E' quindi possibile generare effetti di edge detection, sharpening, blurring, ecc.

La sintassi:

<**feConvolveMatrix in=“”**

order=“” **kernelMatrix=“”**

edgeMode=“duplicate|wrap|none”

preserveAlpha = “false|true”

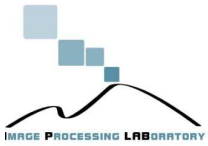
result=“”/>

Breve panoramica

<**feColorMatrix**> permette di applicare delle matrici di trasformazione ai colori;

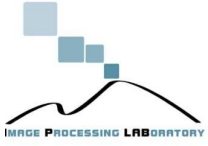
<**feComponentTransfer**> come sopra ma agisce solo su una componente.

<**feImage**> permette di utilizzare una sorgente esterna come input per un filtro.



L'intera gamma

- ✓ feBlend
- ✓ feColorMatrix
- ✓ feComponentTransfer
- ✓ feComposite
- ✓ feConvolveMatrix
- ✓ feDiffuseLighting
- ✓ feDisplacementMap
- ✓ feFlood
- ✓ feGaussianBlur
- ✓ feImage
- ✓ feMerge
- ✓ feMorphology
- ✓ feOffset
- ✓ feSpecularLigthing
- ✓ feTile
- ✓ feTurbulence



Further SVG topics

- ✓ SVG Mobile
- ✓ SVG Print
- ✓ SVG & Multimedia Databases
- ✓ SVG Software