HTML5 + canvas + 2d e JavaScript

HTML5

HTML5 estende e migliora le versioni precedenti e introduce delle API (Application Programming Interface) per applicazioni WEB complesse.

Nativamente include e gestisce contenuti multimediali e grafica; sono stati aggiunti i nuovi elementi <video>, <audio> e <canvas> oltre al supporto per SVG (Scalable Vector Graphics) e MathML per le formule matematiche

- cross-platform multimedia library
- fornisce accesso (di livello sufficientemente basso) a
  - audio e video
  - keyboard, mouse, touch screen, ecc.
  - grafica 2D
  - grafica 3D accelerata via WebGL
- gira su tutti i browser (Chrome, Firefox, InternetExplorer, Safari, ecc.)

HTML5

Un documento HTML5:

```
<!doctype html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Example</title>
        <style> body{ color: #222222; } </style>
        <script>
                ...
        </script>
    </head>
    <body>
        <p>Some text</p>
        <script>
                … // codice JavaScript
        </script>
    </body>
</html>
```

Elemento <canvas>

Per la programmazione grafica con un browser, la differenza più importante rispetto alle versioni precedenti di HTML è l'elemento <canvas>.
Questo nuovo elemento permette un rendering all'interno di un browser.
L'area interna al canvas può essere gestita da un codice in linguaggio JavaScript.

I browser supportano l'elemento <canvas> con un markup del tipo:

```
<canvas id="my-canvas" width="600" height="400">
    Your browser does not support the HTML5 canvas element
</canvas>
```

L'elemento canvas ha gli attributi id, width e height.
Il testo all'interno del tag viene visualizzato solo se il browser non supporta il tag <canvas>.

Contesto

Poiché un elemento <canvas> supporta più di un API per la grafica, per iniziare il rendering, si deve prima specificare l'API che si vuole usare. Questo si fa con il metodo

getContext(contextId, args...)

Dove il primo argomento è il nome del contesto che può essere:

'2d', 'webgl' o ' webgl2'

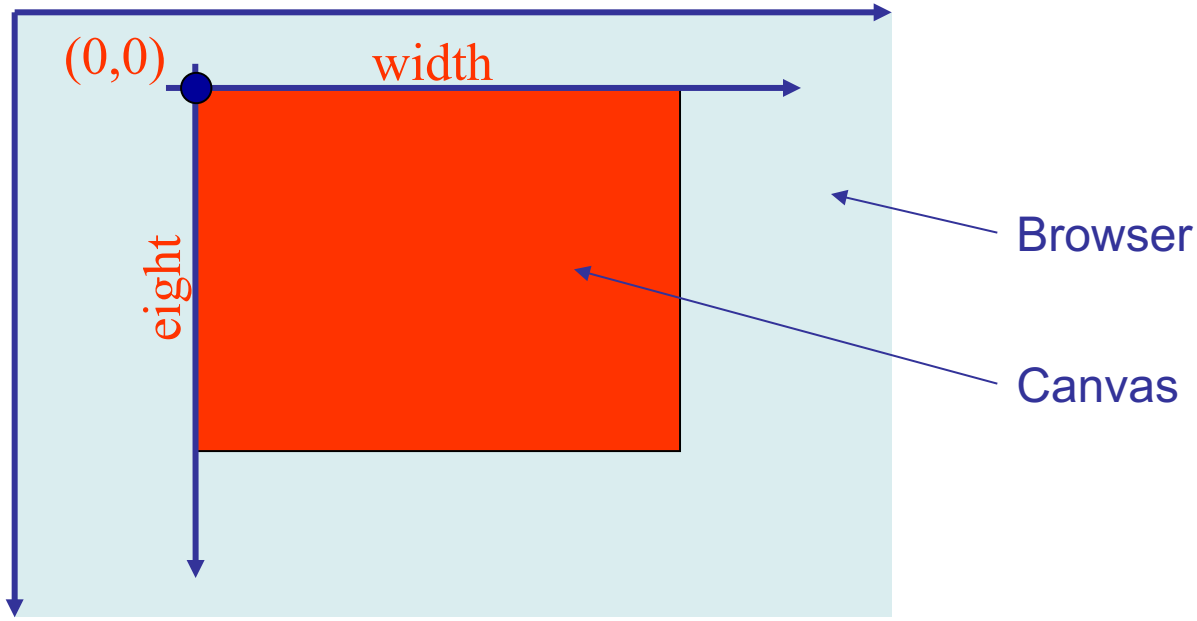
e argomenti aggiuntivi opzionali che variano a seconda dell'API che si sta definendo di usare.

```
<canvas id="my-canvas" width="600" height="400">
   Your browser does not support the HTML5 canvas element
</canvas>
<script>
    // codice JavaScript
    var canvas = document.getElementById("my-canvas");
    var context = canvas.getContext('2d');
    // ora si può disegnare qualcosa
</script>
```

Sistema di Coordinate

All'elemento <canvas> (area rettangolare all'interno del browser) è associato un sistema di coordinate cartesiane ortogonali.

Il vertice alto a sinistra del canvas è l'origine (0,0), l'asse x punta a destra, l'asse y punta in basso.



Attenzione. Anche alla pagina del browser viene associato un sistema di coordinate cartesiane.

Elemento <canvas>

Dimensione del <canvas> e dimensione della superficie di disegno. Utilizzare CSS per dimensionare l'elemento <canvas> non è la stessa cosa di impostare gli attributi di larghezza e altezza.

```
<!DOCTYPE html>
  <head>
    <title>Canvas element size: 600 x 300,
       Canvas drawing surface size: 300 x 150
    </title>
    <style>
       body {
         background: #dddddd;
       }
       #canvas {
         margin: 20px;
         padding: 20px;
         background: #ffffff;
         border: thin inset #aaaaaa;
         width: 600px;
         height: 300px;
       }
    </style>
  </head>
```

```
  <body>
    <canvas id='canvas'>
      Canvas not supported
    </canvas>


    <script src='example.js'></script>
  </body>
</html>
```

Esempio in cui si imposta la dimensione dell'elemento <canvas> a valori diversi dalla dimensione della superficie di disegno.

Elemento <canvas>

Quando si impostano gli attributi width e height dell'elemento <canvas>, si imposta sia la dimensione dell'elemento sia la dimensione della superficie di disegno dell'elemento;

tuttavia, quando si usa CSS per dimensionare l'elemento <canvas>, si impostano solo le dimensioni dell'elemento e non la superficie di disegno.

Per default, sia la dimensione dell'elemento <canvas> sia la dimensione della sua superficie di disegno sono 300 x 150 pixel. Nell'esempio riportato, che utilizza CSS, la dimensione dell'elemento <canvas> è impostata a 600 x 300 pixel, ma la dimensione della superficie di disegno rimane al valore di default di 300 x 150 pixel.

Come conseguenza il browser mappa la superficie di disegno di 300 x 150 pixel a 600 x 300 pixel. Questo darà conseguenze?

Elemento <canvas>

Chiariamo meglio la relazione fra le dimensioni dell'elemento definito in CSS e le dimensioni della superficie di disegno.

Per quanto si evince da prove (le sole che danno certezze) definendo le dimensioni dell'elemento in CSS si apre un canvas di quelle dimensioni in pixel, mentre impostando gli attributi width e height dell'elemento canvas si va ad impostare il range delle coordinate di disegno che verranno mappate nell'area del canvas.

Se quindi il canvas è più piccolo della superficie di disegno, punti a coordinate differenti potrebbero essere visualizzati come lo stesso pixel;

Se invece il canvas è più grande della superficie di disegno, il disegno di un punto potrebbe essere reso come più pixel;

Elemento <canvas>

Se sul browser usiamo la combinazione di tasti Ctrl+, chiediamo che le dimensioni dell'elemento canvas aumentino, mentre resteranno uguali le dimensioni della superficie di disegno e l'effetto è quello di avere un disegno più grande, ma anche meno nitido, perché ogni pixel viene rappresentato più grande con una tecnica di interpolazione bilineare.

L'ideale è avere le dimensioni del canvas e della superficie di disegno uguali, cosa che si può fare

definendo solo le dimensioni della superficie di disegno.

A questo punto bisognerebbbe aggiungere la richiesta di disabilitare il Ctrl+ del canvas.

Elemento <canvas>

L'elemento <canvas> non è un API per fare grafica, ma è solo un contenitore di grafica; espone solo tre metodi e a noi interesserà soprattutto il primo:

getContext()	ritorna il contesto grafico associato con il canvas. Ogni elemento canvas ha almeno un contesto ed ogni contesto è associato con un elemento canvas.
toDataURL(type, quality)	vedi documentazione
toBlob(callback, type, args...)	vedi documentazione

Primitive Grafiche

Le primitive grafiche sono i mattoni più piccoli con cui si può disegnare. Le primitive a nostra disposizione dipendono dal contesto utilizzato e possono essere punti, linee, poligoni come triangoli e quadrilateri o forme in alcuni casi anche di livello superiore.

In queste slide ci riferiremo al contesto '2d' e alle sue proprietà e metodi.

Praticamente i metodi esposti permettono di accedere al "frame buffer" in memoria video e modificarne/leggerne il contenuto.

Nota: i contesti per l'elemento <canvas> in HTML5 non espongono metodi/primitive grafiche per disegnare un singolo pixel.
Detto questo, vedremo un esempio di come si possa disegnare un singolo pixel; vedi slide Gestione Immagini e Pixel.

Rettangoli 2d

I seguenti metodi permettono di disegnare un rettangolo:

rect()	definisce un rettangolo, ma non lo disegna
fillRect()	disegna un rettangolo pieno (filled)
strokeRect()	disegna un rettangolo vuoto (no fill)
clearRect()	cancella i pixel di un dato rettangolo

```
var mc = document.getElementById( "myCanvas" );
var ctx = mc.getContext( "2d" );
ctx.fillRect(10, 10, 110, 50);
```

Percorso/path 2d

Metodi per la definizione e/o disegno di forme più complesse:

fill()	riempie il path disegnato
stroke()	disegna il path definito
beginPath()	inizia un path, o resetta il path corrente
moveTo()	muove il path al punto specificato nel canvas, ma senza disegnare
closePath()	crea un path dal punto corrente al punto iniziale
lineTo()	aggiunge un nuovo punto e crea una linea dall'ultimo punto specificato a questo
clip()	taglia una regione di forma e dimensione generica
quadraticCurveTo()	crea una curva di Bézier quadratica
bezierCurveTo()	crea una curva di Bézier cubica
arc()	crea un arco (viene usato per cerchi o archi di cerchio)
arcTo()	crea un arco con tangenti agli estremi
isPointInPath()	ritorna true se il punto specificato è nel path corrente, altrimenti ritorna false

Codici di Esempio

```
var mc = document.getElementById( "myCanvas” );
var ctx = mc.getContext( "2d” );
ctx.beginPath( );
ctx.rect(10, 10, 110, 50);
ctx.stroke( );
```

```
var mc = document.getElementById( "myCanvas” );
var ctx = mc.getContext( "2d” );
ctx.beginPath( );
ctx.moveTo(0, 0);
ctx.lineTo(100, 150);
ctx.stroke( );
```

Codici di Esempio

```
var mc = document.getElementById( "myCanvas” );
var ctx = mc.getContext( "2d” );

ctx.beginPath( );
ctx.lineWidth = "5";
ctx.strokeStyle = "green” ; // Green path
ctx.moveTo(0, 50);
ctx.lineTo(150, 50);
ctx.stroke( ); // Draw

ctx.beginPath( );
ctx.strokeStyle = ”blue” ; // Blue path
ctx.moveTo(50, 0);
ctx.lineTo(150, 150);
ctx.stroke( ); // Draw
```

Colori, Stili e altro

I seguenti metodi permettono di definire colori e stili di disegno:

fillStyle()	setta il colore, gradiente o pattern di disegno pieno
strokeStyle()	setta il colore, gradiente o pattern di disegno
lineWidth()	setta lo spessore di disegno

Si consulti la documentazione per i tanti altri metodi esposti per il setting dei parametri di disegno.

Si guardi il seguente link per informazioni sui colori:

http://en.wikipedia.org/wiki/RGBA_color_space

Codici di Esempio

cartella: HTML5_2d_1

Codici: open_canvas.html
draw_on_canvas0.html
draw_on_canvas1.html
draw_on_canvas2.html
read_value.html
polygon.html
polygon.js

Gestione Eventi

La tastiera e il mouse generano eventi.

Le applicazioni HTML5 sono guidate dagli eventi. Si registrano gli eventi accaduti su elementi HTML e si implementano delle funzioni che rispondono a tali eventi.

Se si definisce un elemento canvas, verrà gestita una coda degli eventi associata a questo specifico elemento.

Quasi tutte le applicazioni basate su canvas gestiscono eventi del mouse o eventi touch o entrambi e molte applicazioni gestiscono anche vari eventi come la pressione dei tasti e il drag and drop.

Gestione Eventi: keyboard

La tastiera genera eventi e per la precisione:
onkeydown, onkeypress, onkeyup

```
canvas.onkeydown = function (e) {
    // Reagisce all'evento "pressione di un tasto della keyboard"
};
```

In alternativa, è possibile utilizzare il metodo più generico addEventListener():

```
canvas.addEventListener('keydown', function (e) {
    // Reagisce all'evento "pressione di un tasto della keyboard"
});
```

Assegnare una funzione a onkeydown, onkeypress, ecc., è leggermente più semplice di usare addEventListener(); tuttavia, addEventListener() è necessario quando si ha bisognao di associare più listeners ad un singolo evento.

Si veda il codice: move_box_keyboard.html

Gestione Eventi: Mouse

Gli eventi del mouse sono:

onclick, ondbclick, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onwheel

```
canvas.onmousedown = function (e) {
   // Reagisce all'evento "pressione di un button del mouse"
};
```

In alternativa, è possibile utilizzare il metodo più generico addEventListener():

```
canvas.addEventListener('mousedown', function (e) {
   // Reagisce all'evento "pressione di un button del mouse"
});
```

Assegnare una funzione a onmousedown, onmousemove, ecc., è leggermente più semplice di usare addEventListener(); tuttavia, addEventListener() è necessario quando si ha bisognao di associare più listeners ad un singolo evento del mouse.

Gestione Eventi Mouse

Attenzione: bisogna sempre trasformare le coordinate del mouse in coordinate della superficie di disegno.

Si noti che le coordinate del mouse nell'oggetto evento, che il browser passa al listener di eventi, sono coordinate del sistema associato al browser anziché essere relative alla superficie di disegno.

Si veda il codice: mouse_event.html

Gestione Altri Eventi

HTML5 gestisce anche altri eventi, non associati a dispositivi di input, come:

Window Event Attributes

Form Events

Drag Events

Clipboard Events

Media Events

Gestione Testo

Con il contesto ’2d’ si può gestire il disegno del testo su una superficie di disegno.

Proprietà	Descrizione
font	setta o ritorna le proprietà del font corrente per il contenuto del testo
textAlign	setta o ritorna l’allineamento corrente per il contenuto del testo
textBaseline	setta o ritorna il corrente baseline del testo usato quando si disegna testo

Metodi	Descrizione
fillText()	disegna testo pieno (filled) sul canvas
strokeText()	disegna testo sul canvas (no fill)
measureText()	ritorna un oggetto che contiene il width del testo specificato

Codice di Esempio

```
var mc = document.getElementById( "myCanvas" );
var ctx = mc.getContext( "2d" );

ctx.font = "bold 32px Arial";
ctx.textAlign = "center";
ctx.textBaseline = "middle";
ctx.strokeStyle = "orange";
ctx.strokeText( "Welcome to CG LAB! " , 200, 100);
```

Si veda il codice: text_on_canvas1.html

Gestione Immagini

Per disegnare immagini su un canvas, HTML5 espone il metodo

drawImage() disegna un'immagine, ma anche un canvas o un video sul canvas

```
window.onload = function( ) {
  var mc = document.getElementById( "myCanvas" );
  var ctx = mc.getContext( "2d" );
  var img = document.getElementById( "myimage" );
  ctx.drawImage(img, 10, 10);
};

<img id="myimage" src="star.png">
```

Si veda il codice: image_on_canvas2.html

Altri metodi per Immagini

createImageData()	crea una nuovo oggetto vuoto ImageData
getImageData()	ritorna un oggetto ImageData che copia i valori dei pixel di un dato rettangolo su una canvas
putImageData()	rimette i dati di una immagine (da uno specificato oggetto ImageData) sulla canvas

Proprietà	Descrizione
width	ritorna il width di un oggetto ImageData
height	ritorna il height di un oggetto ImageData
data	ritorna un oggetto che contiene dati di uno specificato oggetto ImageData

Questi metodi ci permettono di gestire i singoli pixel di una immagine o di un canvas;

si veda il codice: set_pixels.html

Esempio di Animazione

L'idea alla base di una animazione è il ridisegno in posizioni differenti di un oggetto. Il ridisegno deve avvenire almeno 50-60 volte al secondo (si dice 50-60 fps, frame per secondo) che è il giusto compromesso per far sì che l'utente veda un'animazione fluida e il sistema non stia ricalcolando troppe volte.

Inoltre, l'utente dovrebbe vedere solo le immagini disegnate e non la fase di disegno dell'immagine; questo viene solitamente ottenuto con il double buffer, ossia con la possibilità di disegnare su due superfici di disegno differenti e mentre si disegna su una si mostra l'altra e viceversa.

Se poi l'utente vuole interagire con l'animazione, bisognerà prevedere di reagire ad eventuali eventi.

Si veda il codice: ball.html

Codici di Esempio

Nei seguenti codici si possono trovare altri esempi di animazione con un po' di interattività:

ball2.html

ball3.html

ball3_2.hml

analizziamo insieme ball3_2.html

Riferimenti, Tutorial ed Esempi

https://www.html.it/pag/45343/introduzione-a-javascript/

https://www.homeandlearn.co.uk/javascript/javascript.html

https://developer.mozilla.org/en-US/docs/Web/JavaScript

https://www.tutorialspoint.com/html5/

https://www.w3schools.com/graphics/canvas_intro.asp

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Giulio Casciola

Dip. di Matematica

giulio.casciola at unibo.it