SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3197

INTERAKTIVNI GRAFIČKI OBJEKTI U STANDARDU HTML5

Damir Ciganović-Janković

Zagreb, lipanj 2013.

Sadržaj

1. Uvod

Znajući to ili ne, svaki današnji korisnik Interneta susreo se s prezentacijskim jezikom za izradu internetskih stranica HTML (HyperText Markup Language) ili barem onime što je njime prikazano na našim ekranima. Početak jezika seže u 1991. godinu kada ga njegov osnivač Tim Berners Lee[1] po prvi put prezentira, a prva verzija je objavljena 1993. godine. HTML funkcionira tako da izrađuje hipertekst dokument te daje uputu internetskom pregledniku kako da ga prikaže. Sintaksa je zbog svoje jednostavnosti i poprilično intuitivnog načina korištenja dovela do velike popularnosti i raširenosti HTML-a među svim korisnicima. Ona se temelji na oznakama(engl. tags) koje omogućuju korisniku da strukturno gradi internetske stranice, slažući svaki dio internet stranice posebno. Broj oznaka, te same mogućnosti raznih oznaka, mijenja se iz verzije u verziju jezika te se time moć i korisnost jezika konstantno povećava. Iako je službeni standard HTML-a HTML 4.01, ovaj rad će biti posvećen mogućnostima HTML5 standarda.

HTML5 je standard čija je namijena da bude "ono što je HTML trebao biti od početka"[2]. On objedinjuje HTML 4 i XHTML u jedan prezentacijski jezik koji dopušta sintaksu oba svoja prethodnika. Prva verzija HTML5 standarda je izašla 2008., no sam standard još uvijek nije u potpunosti implementiran te postoje pretpostavke da bi tijekom 2014. godine mogao biti dovršen do kraja. Unatoč tomu, HTML5 već sada nudi mnoge mogućnosti te se na njega gleda kao na budućnost onoga što nam Internet donosi. Vrlo dobar primjer onoga što HTML5 donosi je značajan napredak u pomaganju internet tražilicama kako bi što bolje mogle odrediti rangove i relevantnost pojedinih stranica te tako preciznije odrediti prioritete[3]. No, u fokusu ovoga rada su mogućnosti HTML5 standarda u području grafike.

Mi ljudi od svih svojih osjetila najviše informacija upijamo putem oka. Oku ugodne stvari nas najviše privlače te je više vjerojatno da ćemo se zadržati na internetskoj stranici koja nam pruža vizualno ugodnije okruženje, nego ona koja nas manje privlači, pa bila ona sadržajno i bogatija. HTML5 pruža potpunu

[1] Uz to, izumitelj je World Wide Weba te kasnije osnivač World Wide Web Consortium organizacije koja se bavi standardizacijom tehnologija korištenih na webu, http://www.w3.org/People/Berners-Lee/

[2] Aitken, , What is HTML5?, 2013., http://www.html-5-tutorial.com/about-html5.htm, pristup 10.06.2010.

[3] Primjer su nove, uvedene oznake: <nav>, <header>, <footer>, itd.

podršku za CSS3 stiski jezik koji u sebi sadrži mogućnosti prikaza 2D i 3D elemenata, no osim toga, među novim oznakama standarda nalazimo potporu za dva tipa grafičkih formata koje možemo stvoriti, bitmape i vektore. Jedan je usmjeren na slikovne elemente (enlg. pixel-oriented), a drugi na oblike (engl. shape-oriented). Korištenje prvog tipa je kroz HTML5 standard omogućeno preko oznake <canvas>, a drugoga preko oznake <svg> te ćemo osnove i mogućnosti obje oznake, uz CSS3, promotriti tijekom ovog završnog rada.

Kao zadnje poglavlje ovog rada, upoznat ćemo se s definicijom proširene stvarnosti te prikazati nekoliko primjera ostvarenih uz pomoć HTML5 standarda.

Slika 1. Logo HTML5 standarda

2. CSS3

Cascading Style Sheets ili kraticom CSS je stilski jezik kojim definiramo kako se HTML elementi prikazuju. U početnoj fazi korištenja HTML-a naglasak je bio na strukturi stranice te definiciji sadržaja unutar dokumenta, a ne kako i kojim stilom je taj sadržaj prikazan. Kako su porasli apetiti, tako se javila i potreba za formatom prikaza. Rješenje je bio CSS. Inačica na kojoj će biti naglasak u ovom radu je posljednji CSS standard CSS3. CSS3 daje mogućnost prikaza uobičajenih HTML5 elemenata kao 2D i 3D objekte na kojima je moguće obavljati razne geometrijske transformacije te ih animirati.

2.1. Geometrijske transformacije

Geometrijske transformacije se unutar CSS3 jezika obavljaju korištenjem svojstva (eng. property) *transform.* To je svojstvo različito podržano ovisno o pregledniku kojeg koristimo pa će tako u slučaju korištenja Google Chrome i Safari preglednika kao prefiks svojstvu biti potrebno dodati "-webkit-", a u slučaju korištenja Internet Explorer-a "-ms-", dok za Firefox i Operu neće biti potrebno dodati prefiks za ime svojstva. Pošto većina današnjih internet korisnika koristi Google Chrome[4], primjeri kodova bit će prilagođeni tom pregledniku radi čitljivosti koda. Unatoč tome, uzmimo u obzir da je u praksi preporučljivo napisati sve varijacije kako bi napisani kod vrijedio za svaki preglednik.

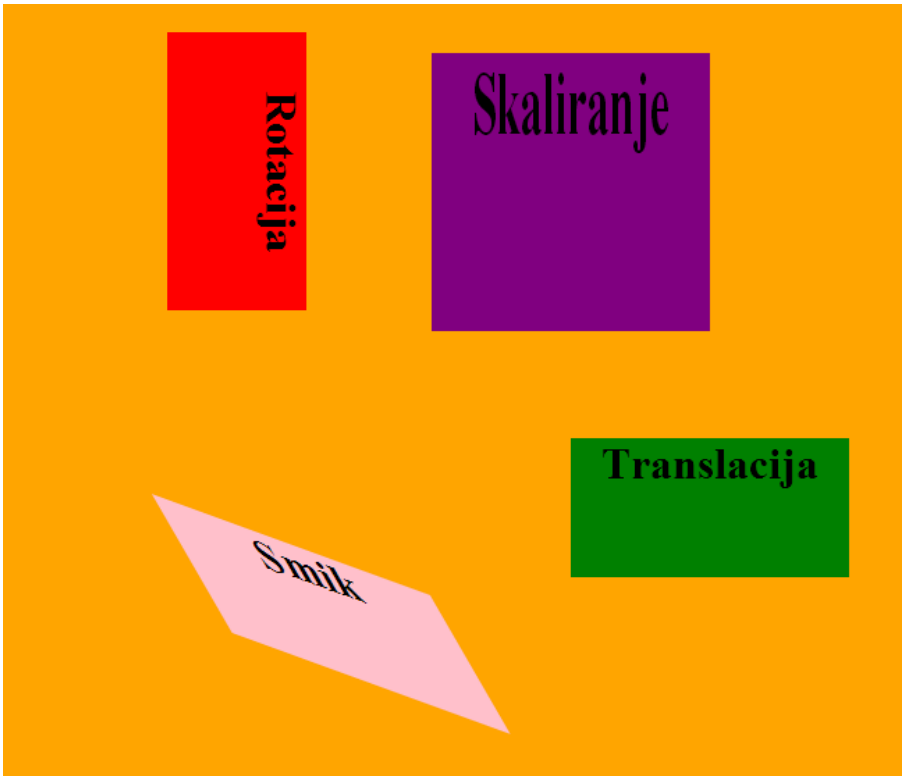
Transformacije koje imamo na raspolaganju su: translacija, rotacija, smik, skaliranje i transformacija preko matrice kojom se mogu primijeniti varijacije navedenih transformacija. Slika 2. prikazuje kod za primjer kojim ćemo pokazati funkcionalnost navedenih transformacija. U ovom primjeru modificirani HTML element je oznaka . Za svaku od transformacija napravljena je po jedna div klasa. Nakon svojstva *transform* navodimo naziv funkcije kojom želimo transformirati element. Metoda *translate* obavlja translataciju za vrijednost x i y koordinate u slikovnim elementima. Početna točka (0, 0) nalazi se u gornjem lijevom kutu. Metoda *rotate* obavlja rotaciju za vrijednost u stupnjevima. Metoda *skew* obavlja smik tako da prvi argument određuje kut za koji se vertikalni okvir elementa otklanja od y-osi, a drugi argument određuje kut za koji se horizontalni

[4] Browser Statistics, 2013., http://www.w3schools.com/browsers/browsers_stats.asp, 10.06.2013.

okvir elementa otklanja od x-osi. Na posljetku, *scale* metoda skalira x koordinate okvira s prvim argumentom, a y koordinate s drugim i tako povećava, odnosno smanjuje element. Svaki od elemenata je uz osnovnu transformaciju i translatiran kako bi se mogle jasnije uočiti razlike među elementima (Slika 3.) te se na tom primjeru odmah može uočiti ispravan način na koji se nižu razne transformacije. Ukoliko se transformacije ne pišu tako, u jednom retku, nego svaka transformacija u svojem retku, bit će izvršena samo posljednja transformacija. Metoda *matrix()* prima 6 argumenata kojima imamo mogućnosti zadavanja transformacija odjednom u redoslijedu: rotaciju, skaliranje, translaciju pa smik. Osim ovih oblika, postoje i varijacije spomenutih transformacija.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>HTML5 - CSS3 transformacije </title>
      <style>
        div
        {
        width: 200px;
        height: 100px;
        }
        div.rotacija
        {
        background-color: red;
        -webkit-transform: translate(60px, 50px) rotate(90deg);
        }
        div.translacija
        {
        background-color: green;
        -webkit-transform: translate(400px,170px);
        }
        div.smik
        {
        background-color: pink;
        -webkit-transform: skew(30deg,20deg) translate(70px, 100px);
        }
        div.skaliranje
        {
        background-color: purple;
        -webkit-transform: translate(-200px, -300px)
                           scale(1,2)
                           translate(500px, 0px);
        }
        h1
        {
        text-align: center;
        }
      </style>
  </head>
  <body style="background-color: orange;">
            <div class="rotacija"><h1>Rotacija</h1></div>
            <div class="translacija"><h1>Translacija</h1></div>
            <div class="smik"><h1>Smik</h1></div>
            <div class="skaliranje"><h1>Skaliranje</h1></div>
  </body>
</html>
```

Slika 2. Primjer 2D transformacija, izvorni kod



Slika 3. Prikaz koda sa Slike 2. u internet pregledniku

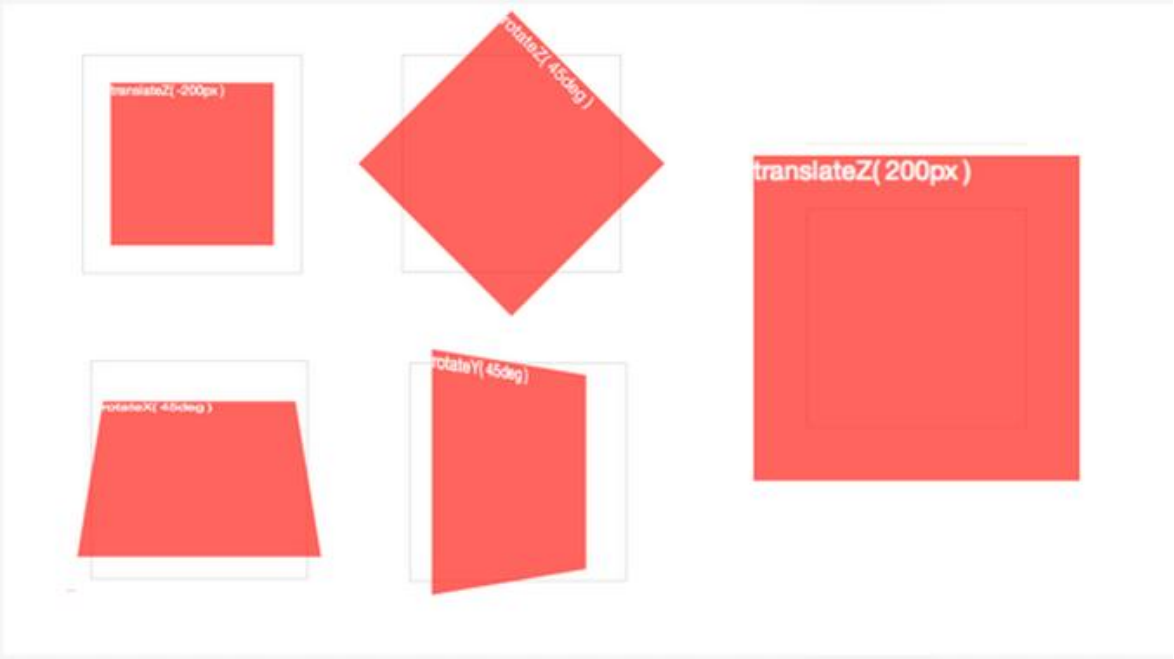
Dosadašnji primjeri prikazivali su transformacije u dvije dimenzije, no CSS3 nam isto tako omogućava i transformacije u 3D prostoru. Kako bi omogućili 3D transformacije nad HTML5 elementom, potrebno je postaviti perspektivu na element. Metodom *perspective* postavljamo vrijednost perspektive, a ona je određena udaljenosti elementa od promatrača u slikovnim elementima. Nakon što smo postavili vrijednost perspektive, koristimo *transform* metodu kako bi obavljali transformacije u 3D prostoru. Neke od metoda su *matrix3d*, *scale3d*, *rotate3d*, *translate3d* te posebno po kordinatama *scaleX*, *rotateZ*, *translateY* itd. U danom primjeru (Slika 4.) promatrali smo element s određene perspektive te na unutarnji element <iframe>[5] primijenili neke od navedenih 3D transformacija.

[5] <iframe> je oznaka HTML standarda koja omogućuje da otvorimo HTML dokument unutar HTML dokumenta u prozoru kao elementu HTML dokumenta

```
<div style="-webkit-perspective: 800;  ">
   <iframe src="http://www.fer.unizg.hr//"
      style="-webkit-transform:
            rotate3d(1, 1, 0, 60deg) translateX(300px) ;"></iframe>
</div>
<div style="-webkit-perspective: 800;  ">
   <iframe src="http://www.html5rocks.com/"
      style="-webkit-transform: rotateX(30deg) rotateY(-5deg)
            translateZ(200px) translateX(100px);"></iframe>
</div>
```

Slika 4. Dio HTML dokumenta, primjer 3D transformacije[1]

Slika 5. Prikaz izvođenja koda sa Slike 4.



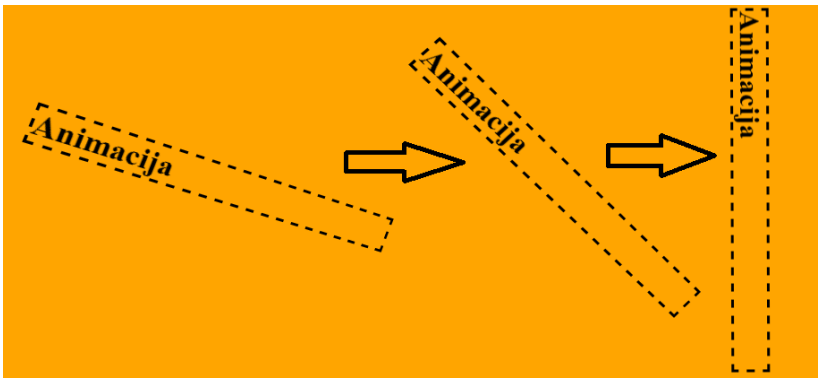
Slika 6. Primjeri 3D transformacija [2]

2.2. Animacija

Postoji nekoliko načina na koje možemo animirati elemente HTML5 standarda uz pomoć CSS3-a, a u sklopu ovog rada bit će prikazana dva. Prvi način je preko svojstva *transition.* Svojstvo *transition* u potpunosti određuju 4 parametra. Prvi je *transition-property* koji određuje naziv svojstva na kojeg će tranzicija imati efekt. U našem primjeru (Slika 7.) naziv svojstva je –*webkit-transform*. Drugi parametar je *transition-duration* kojim se određuje koliko drugo će tranzicija između stanja trajati, u primjeru je taj parameter jednak tri sekunde. Sljedeći parametar je *transition-timing-function* kojim određujemo o kojoj funkciji će ovisiti brzina u datom trenutku kojom element prelazi iz jednog u drugo stanje. U primjeru ta funkcija je *ease-in-out* kojom određujemo da će vrijednost brzine kretanja na početku i na kraju animacije biti veća. Osim toga, mogli smo izabrati i linearnu, kubnu Bezierovu funkciju, spori početak te druge varijacije. Zadnji argument *transition-delay* određuje odgodu početka same animacije. Taj parametar u primjeru nije naveden pa mu se pridodaje početna vrijednost od nula sekundi. Primjer funkcionira tako da kada miš stavimo na element u kojem piše “Animacija” (*hover* svojstvo), izvodi se animacija tog elementa zbog zadane tranzicije koja se izvodi kada se nad elementom obavlja transformacija.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      #anim1 {
        -webkit-perspective: 800;
        margin: 300px 300px 200px 50px;
      }
      #anim1 h1 {
        -webkit-transition: -webkit-transform 3s ease-in-out;
        -webkit-transform: translate(50px, 100px) rotate3d(0, 1, 1, 30deg);
      }
      #anim1 h1:hover {
        -webkit-transform: translate(50px, 100px) rotate3d(0, 0, 1, 50deg);
      }
    </style>
  </head>
  <body style="background-color: orange">
    <div id="anim1">
      <h1 style="border: dashed; width: 40%;">Animacija</h1>
    </div>
  </body>
</html>
```

Slika 7. Animacija pomoću *transition* svojstva



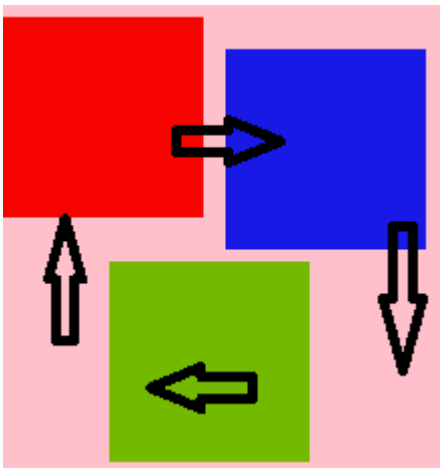
Slika 8. Prikaz izvođenja koda sa Slike 7.

Drugi način animiranja HTML5 elemenata je preko CSS3 *animation* svojstva. Animacija se određuje s nekoliko parametara. Parametar *animation-name* određuje ime za *@keyframes* animaciju. Animacija *@keyframes* je pravilo kojim određujemo tijek animacije tako da slijedno upisujemo postotak dovršenosti animacije te vrijednosti svojstava koje element ravnomjerno mijenja u odnosu na te postotke. Dani primjer (Slika 9.) prikazuje kvadrat koji kruži u smjeru kazaljke na satu te putujući od jednog stanja, određenog postotkom do drugog, mijenja svoju pozadinsku boju kako bi u sljedećem stanju dosegnuo zadanu. Time smo definirali

@keyframe pravilo čije ime je u primjeru *mymove*. Osim toga parametra, svojstvu *animation* postavili smo i parametar *animation-duration*, koji određuje duljinu trajanja animacije, na 5 sekundi te parametar *animation-iteration-count* kojeg smo postavili na *infinite*, odnosno beskonačan broj iteracija.

```
<!DOCTYPE html>
<html>
<head>
        <style>
            div
            {
            width:100px;
            height:100px;
            background:red;
            position:relative;
            -webkit-animation:mymove 5s infinite;
            }

            @-webkit-keyframes mymove
            {
            0%   {top:0px; left:0px; background:red;}
            25%  {top:0px; left:100px; background:blue;}
            50%  {top:100px; left:100px; background:yellow;}
            75%  {top:100px; left:0px; background:green;}
            100% {top:0px; left:0px; background:red;}
            }
        </style>
</head>
<body style="background-color: pink">
        <div></div>
</body>
</html>
```

Slika 9. Animacija pomoću *animation* svojstva



Slika 10. Prikaz izvođenja koda sa Slike 9.

3. SVG kao oznaka HTML5

Scalable Vector Graphics ili kraće SVG je jezik koji opisuje dvodimenzionalnu vektorsku grafiku unutar jezika XML, no HTML5 standard koristi taj jezik kao jednu od oznaka standarda - <svg>. SVG je jezik koji koristi vektorsku grafiku, a to znači da grafičke oblike crta koristeći objekte pa je tako moguće svakom objektu posebno izmijeniti pojedini atributi. Iz toga proizlazi da objekti, pa čak i oni veliki, zauzimaju vrlo malo memorije zato toga što računalo u memoriju zapisuje samo vrijednosti atributa objekata te su i datoteke koje sadrže elemente vektorske grafike u principu male. Potrebno je napomenuti da promjenom veličine ili zumiranja objekata, slika ne gubi na kvaliteti što znači da je SVG neovisan o rezoluciji ekrana. Mana vektorske grafike je u tome da slike ponekad ne izgledaju stvarno nego više nalikuju onima iz crtanih filmova. SVG za sada omogućuje prikaz samo 2D elemenata i transformacija nad njima, no uz pomoć CSS3 standarda, možemo raditi I 3D transformacije, stoga će u nastavku biti prikazan način na koji se iscrtavaju jednostavni 2D objekti, transformacije koje se na njima mogu obavljati te animacije takvih objekata.[4]

3.1. Jednostavni objekti

SVG sadrži unaprijed definirane elemente koji predstavljaju određene oblike. Ti elementi su: <rect>, <circle>, <ellipse>, <line>, <path>, <polyline>, <text> i <polygon>. Svaki od elemenata je objekt s vlastitim atributima čije vrijednosti zadajemo prilikom crtanja. Oni se ponašaju kao i svaki drugi element HTML5 standarda, no da bi se dobio očekivani rezultat oznake se moraju staviti unutar <svg> oznaka. Pokazat ćemo kako koristiti spomenute elemente u nekoliko primjera. Prvi primjer (Slika 11. i Slika 12.) koristi <circle> i <line> elemente. Crtajući liniju odredili smo koordinate prve i druge točke, a crtajući krug središte kruga te njegov radijus. Svojstvima *stroke-width* i *stroke* odredili smo debljinu i boju linije, a sa *fill* svojstvom boju kojom smo ispunili krug. U drugom primjeru (Slika 13. i Slika 14.) se koriste <rect>, <polygon> i <text> elementi. Za <rect> element definiramo položaj, duljinu i visinu, a za <polygon> element koordinate točaka poligona preko svojstva *points*.

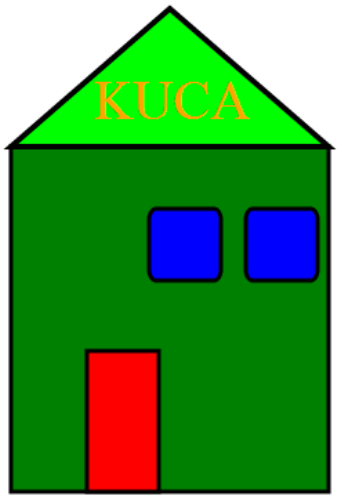
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1">
      <style>
      line
      {
      stroke-width:5;
      stroke:black;
      }
      </style>
      <line x1="60" y1="440" x2="100" y2="320"/>
      <line x1="140" y1="440" x2="100" y2="320"/>
      <line x1="100" y1="220" x2="100" y2="320"/>
      <!--ruke-->
      <line x1="100" y1="220" x2="82" y2="275"/>
      <line x1="82" y1="275" x2="60" y2="308"/>
      <line x1="100" y1="220" x2="156" y2="225"/>
      <line x1="156" y1="225" x2="185" y2="180"/>
      <!--glava-->
      <circle cx="100" cy="180" r="40"
              style="stroke-width:5; stroke:black; fill:white;"/>
</svg>
  </body>
</html>
```

Slika 11. <svg> oznake <circle> i <line>

Slika 12. Prikaz izvođenja koda sa Slike 11.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1">
      <style>
          rect
          {
          stroke-width:3;
          stroke:black;
          }
      </style>
      <rect x="360" y="190" width="230" height="250"
      style="fill:green;"/>
      <!--prozori-->
      <rect x="460" y="235" rx="5" ry="8" width="52" height="52"
      style="fill:blue;"/>
      <rect x="530" y="235" rx="5" ry="8" width="52" height="52"
      style="fill:blue;"/>
      <!--vrata-->
      <rect x="415" y="338" width="52" height="102" style="fill:red;"/>
      <!--krov-->
      <polygon points="360,190 475,90 590,190"
      style="fill:lime;stroke:black;stroke-width:4"/>
      <text x="420" y="170" fill="orange" style="font-size:40">KUCA</text>
    </svg>
  </body>
</html>
```

Slika 13. <svg> oznake <rect>, <polygon> i <text>[6]



Slika 14. Prikaz izvođenja koda sa Slike 13.

[6] prozori nacrtani uz korištenje [5]

3.2. Geometrijske transformacije nad SVG objektima

Slično kao što su se koristile 2D transformacije uz pomoć CSS3 standarda, koristit ćemo transformacije nad SVG elementima. Uz pomoć svojstva *transform* možemo koristiti nekoliko vrsta transformacija. Metode koje su u službi tih transformacija su: *matrix, translate, scale, rotate, skewX* i *skewY* . Pa tako ako primjerice unutar <circle> elementa primjera na Slici 11. dodamo svojstvo *trasform* te metodu *translate(85, -43),* dobit ćemo rezultat na Slici 15. Vrijednosti unutar metode *translate* određuju za koliko slikovnih elemenata će se objekt nad kojim se radi transformacija pomaknuti. Ako bi uz ovu promjenu dodali nad desnom nogom čovječuljka s primjera na Slici 12. primijenili rotaciju izvođenjem naredbe *transform* = "*rotate(-75, 100, 320)"* dobili bismo rezultat na Slici 16. Prvi parametar metode *rotate* određuje kut za koji se element otklanja u smjeru kazaljke na satu, a ostala dva određuju oko koje točke će se element vrtjeti, ukoliko se te dvije vrijednosti izostave, element se vrti oko ishodišta trenutnog koordinatnog sustava, tj. u našem slučaju oko točke s koordinatama (0,0).

```
<circle cx="100" cy="180" r="40" trasform
style="stroke-width:5; stroke:black; fill:white;"
transform = "translate(85, -43)";/>
```

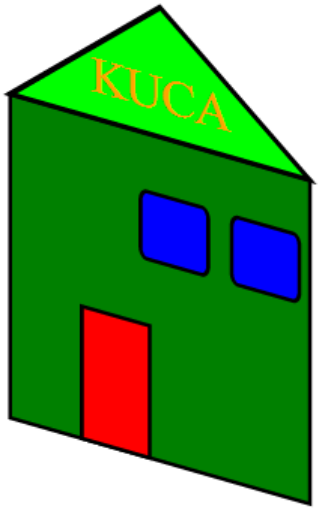
Slika 15. Primjer sa Slike 12. uz korištenje trasnslatacije <circle> elementa

```
<line x1="140" y1="440" x2="100" y2="320"
      transform = "rotate(-75 100 320)";/>
```

Slika 16. Primjer sa Slike 15. uz rotaciju <line> elementa (“desna noga”)

3.3. Transformacije u 3D prostoru uz pomoć CSS3 standarda

Iako SVG sam ne omogućuje transformacije u 3D prostoru, možemo iskoristiti to što se uz pomoć CSS3 standarda takve transformacije mogu obaviti. Kao što smo vidjeli u poglavlju 2 ovoga rada, CSS3 djeluje nad elementima HTML5 standarda pa tako i nad <svg> elementom te svim oznakama koje se koriste unutar SVG-a. Primjer transformacije je na Slici 17. Na istoj slici napisan je i redak koda kojeg smo primijenili na svaku od korištenih SVG oznaka u primjeru sa Slike 13.



Slika 17. 3D transformacija nad SVG elementima uz pomoć CSS3 standarda

3.4. Animacija u SVG-u

Animacija se u SVG-u ostvaruje uz pomoć oznaka <animate>, <animateMotion> i <animateTransform>. Ispravno se koriste tako da oznaku stavimo unutar oznake jednog od jednostavnih objekata, primjerice <animate> oznaku unutar <circle></circle> oznaka.

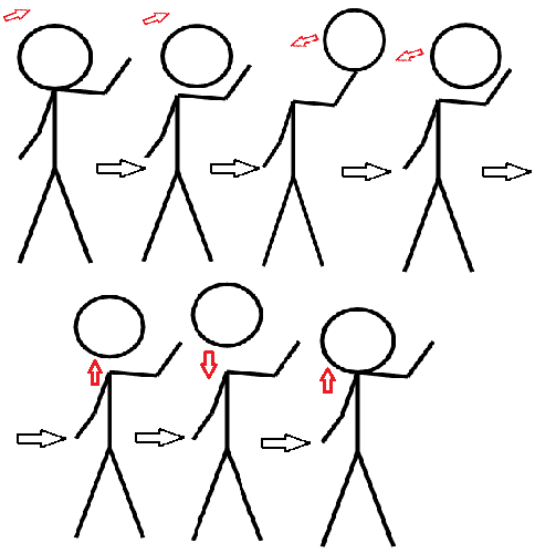
3.4.1. Animacija preko oznake <animate>

Oznakom <animate> možemo ostvariti animaciju nakon što specificiramo vrijednosti svojstava te oznake. Nazivi svojstava kojima moramo pridružiti vrijednost su *atributeName, values* i *keyTimes* (ili *from* i *to*), *dur,* te *repeatTime.* Svojstvom *atributeName* određujemo ime atributa koje će se prilikom animacije mijenjati te zbog kojeg ćemo i imati dojam animacije. U primjeru (Slika 18.) koji se osvrće na primjer sa Slike 12. koristimo svojstva *values* i *keyTimes,* iako se mogu koristiti i svojstva *from* i *to.* Prednost prve opcije pred drugom je u tome što koristeći *values* svojstvo možemo navesti jednu, dvije ili više vrijednosti (u drugoj opciji samo početnu i konačnu vrijednost) atributa kojeg mijenjamo te zatim postavljamo jednak broj vrijednosti *keyTimes* atributa s uzlaznim vrijednostima od 0 do 1 kojima postavljamo vrijeme u koje će se ostvariti pripadajuća vrijednost. Tako u primjeru unutar prve <animate> oznake *cx* atribut će na početku poprimiti vrijednost 100, nakon što prođe pola tijeka animacije vrijednost 185 te se zatim

vratiti na vrijednost 100. Svojstvo *dur* određuje trajanje animacije te je ono isto za obje animacije kako bi promjena vrijednosti atributa bila usklađena pošto je primjer napravljen tako da bude cikličan. Posljednje svojstvo je svojstvo *repeatCount* kojim određujemo broj ponavljanja animacija. Prva animacija će se ponoviti jednom, a drugoj smo postavili vrijednost *indefinite* koja označava da će se animacija izvršavati dok je ne odlučimo prekinuti. Izvršenje koda prikazano je na Slici 19. gdje crvene strelice uz krug pokazuju kretanje kruga.

```
...

<line x1="100" y1="220" x2="156" y2="225"/>
<line x1="156" y1="225" x2="185" y2="180"/>
<!--glava-->
<circle cx="100" cy="180" r="40"
       style="stroke-width:5; stroke:black; fill:white;">
      <animate   attributeName="cx"
                 values="100; 185; 100"
                 keyTimes="0;0.5;1"
                 dur="3s"
                 repeatCount="1"/>

<animate   attributeName="cy"
                 values="180; 137; 180"
                 keyTimes="0;0.5;1"
                 dur="3s"
                 repeatCount="indefinite"/>
</circle>

...
```

Slika 18. Animacija preko oznake <animate>



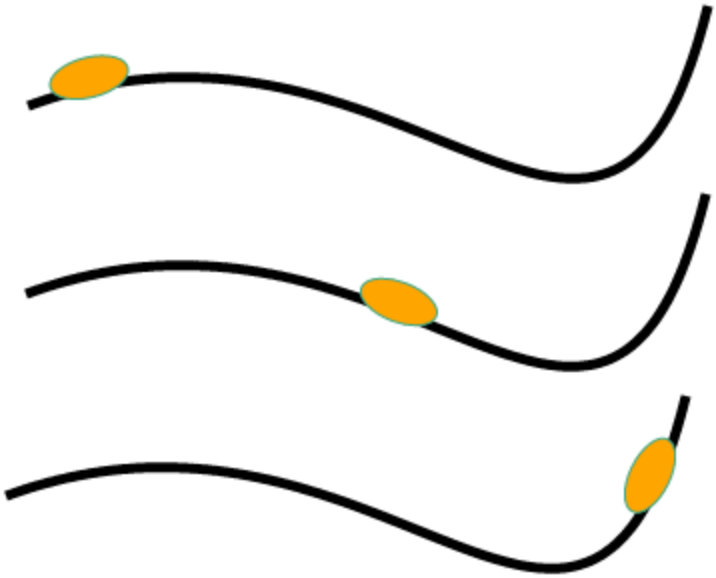
Slika 19. Izvršavanje koda sa Slike 18.

3.4.2. Animacija preko oznake <animateMotion>

Oznakom <animateMotion> animacija se ostvaruje tako da odredimo putanju po kojoj će se referencirani element kretati [6]. Kako bi se odredila putanja, koristi se jedan od primitivnih objekata SVG-a, a to je element <path>. Tom elementu preko svojstva *d* određujemo put. Svojstvo *d* kao vrijednosti prima naredbe i točke kao argumente naredbi. U našem primjeru (Slika 20.) prilikom određivanja putanje koriste se naredbe *M* i *C*. Naredbom *M* određujemo početnu točku puta, a naredbom *C* dajemo parametre za kubnu Bezierovu funkciju tako da se zadaju dvije kontrolne točke te točka u kojoj želimo da krivulja završi. Osim tih metoda postoje još i A kojom definiramo luk, Q kojom definiramo Bezierovu funkciju 4. stupnja te druge. Svojstva koja smo odredili unutar <animateMotion> oznake su *dur* kojim određujemo trajanje animacije, *rotate* koje određuje orijentaciju objekta prilikom kretanja po putanji, te *repeatCount* koji se ponaša isto kao i kod <animate> oznake. Svojstvo *rotate* je u primjeru postavljeno na *auto* kako bi elipsa prilagođavala svoju orijentaciju, u odnosu na koordinatne osi, po putanji za vrijeme animacije. Oznaka <mpath> nam omogućuje da koristimo izvanjski put unutar <animateMotion> oznake pa tako u primjeru koristimo već prije definiranu putanju.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <svg>
            <path id="curve" d="M 10 100 C 200,30 300,250 350,50"
            stroke="black" fill="none" stroke-width="5" />
            <ellipse cx="7" cy="-5" rx="20" ry="10"
                        fill="orange" stroke="#44aa88">
                <animateMotion dur="2s"
                rotate="auto" repeatCount="indefinite" >
                <mpath xlink:href="#curve"/>
                </animateMotion>
            </ellipse>
      </svg>
</body>
</html>
```

Slika 20. Animacija preko oznake <animateMotion>

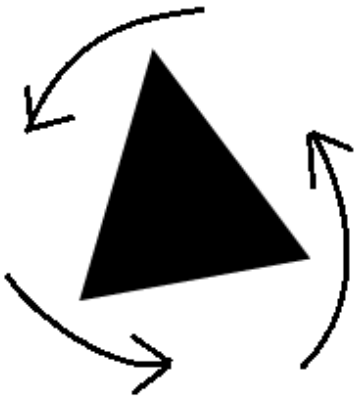


Slika 21. Izvršavanje koda sa Slike 20.

3.4.3. Animacija preko oznake <animateTransform>

Animacija preko oznake <animateTransform> se u principu provodi na jednak način kao kod oznake <animate> samo što je atribut kojega će se mijenjati uvijek atribut *transform* koji je obrađen u odjeljku 3.2. Primjer na Slici 22. prikazuje transformaciju preko te oznake.

```
<polygon points="120,60 180,180 60,180">
    <animateTransform attributeName="transform"
                      attributeType="XML"
                      type="rotate"
                      from="0 120 140"
                      to="360 120 140"
                      dur="10s"
                      repeatCount="indefinite"/>
</polygon>
```



Slika 21. Prikaz animacije pomoću oznake <animateTransform>

4. *Canvas* kao oznaka HTML5

Canvas (engl. platno) je oznaka HTML5 standarda uz pomoć koje možemo koristiti grafiku. Element sam po sebi ne crta ništa, ali je spremnik za grafiku koju crtamo upotrebom skripti (obično JavaScript jezika) i tako omogućava crtanje. Kao što je u uvodu spomenuto, način crtanja upotrebom canvas elementa je usmjeren na slikovne elemente, odnosno korištenjem mape bitova (engl. bitmap), tj. rastersku grafiku. Rasterska grafika sastoji se od mnogo slikovnih elemenata kojima je pridjeljena vrijednost pomoću koje se određuje boja pojedinog slikovnog elementa. Moguće je mijenjati vrijednosti svakoga slikovnog elementa. Pošto se slike često sastoje i od više tisuća slikovnih elemenata, količina memorije koju slika zahtjeva je velika. Promjenom veličine ili zumiranjem slika gubi na kvaliteti [4].

Crtanje pomoću <canvas> elementa ostvariti ćemo tako da prvo kao oznaku smjestimo u HTML dokument, pristupimo mu preko JavaScript jezika, stvorimo kontekst te iskoristimo HTML5 Canvas API[7] kako bi se iscrtali grafički oblici[7]. Razlika između <canvas> elementa i canvas konteksta je u tome da je <canvas> element DOM[8] čvor ugrađen u HTML, a canvas kontekst je objekt koji sadrži svojstva i metode koje možemo koristit kako bi iscrtavali grafiku unutar <canvas> elementa. Kontekst može biti 2D samostalno, a 3D uz pomoć WebGL-a[9]. Svaki <canvas> element može imati najviše jedan kontekst. Primjer JavaScript koda koji dinamički stvara jedan <canvas> element te pridružuje kontekst tome elementu prikazan je na Slici 22.

Kako bismo isprobali mogućnosti <canvas> elementa pogledat ćemo na koji način se mogu iscrtavati 2D objekti te kako ih animirati.

```
function createCanvas(h,w)
{
var c = document.createElement("canvas");
c.width = w;
c.height = h;
return c;
}
```

```
var c=createCanvas(100,100);
var ctx = c.getContext("2d");
document.body.appendChild(c);
```

Slika 22. Primjer stvaranja jednog <canvas> s pripadajućim kontekstom

[7] application programming interface (hrv. aplikacijsko programsko sučelje)
[8] HTML DOM introduction, http://www.w3schools.com/htmldom/dom_intro.asp, 11.06.2013.
[9] An Introduction to WebGL, kolovoz 2012., http://dev.opera.com/articles/view/an-introduction-to-webgl/, pristupljeno 11.06.2013.

4.1. Crtanje 2D objekata uz pomoć <canvas> oznake

Kako bi smo pokazali mogućnosti crtanja 2D uz pomoć <canvas> elementa pogledat ćemo nekoliko primjera. Pregled svih metoda i svojstava <canvas> elementa možemo pronaći u [8]

4.1.1. Crtanje puta

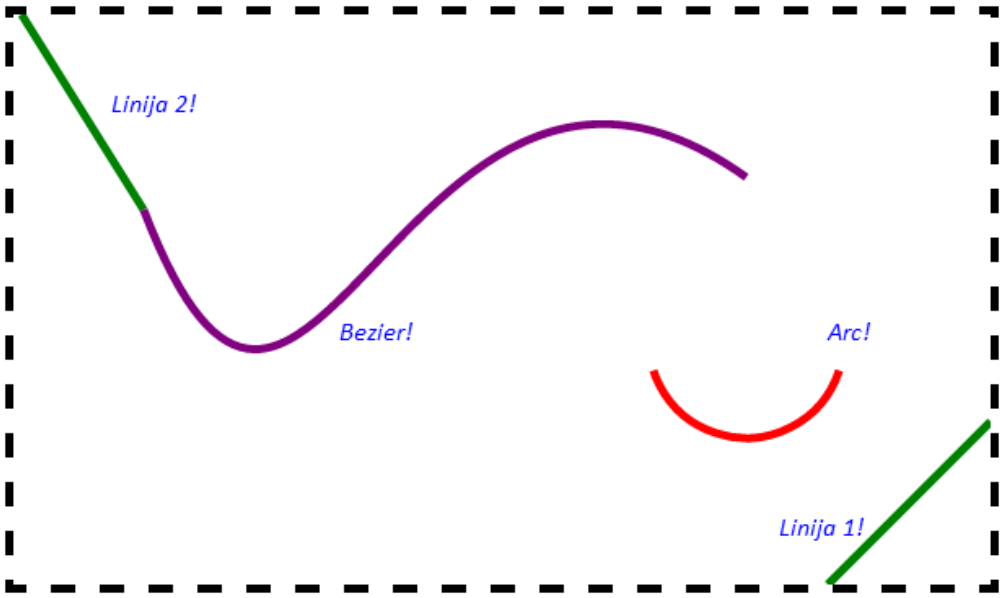
Put se unutar canvas konteksta crta uz pomoć metode beginpath() koja započinje put ili resetira do sada stvoreni put. Iako metoda nije nužna za iscrtavanje, dobro ju je koristiti jer otvara mogućnost stvaranja različitih putova s različitim vrijednostima svojstava. Primjerice, bez korištenja te metode ne bismo bili u mogućnosti nacrtati jednu liniju s crnom bojom, a drugu pokraj nje s crvenom, nego bi obje bile povučene jednakom bojom. Neke od metoda kojima stvaramo put su: *moveTo*, *lineTo*, *quadricCurveTo*, *bezierCurveTo*, *arcTo*, i *arc*. Metodom *stroke* iscrtavamo sve ono što smo naveli kao put pomoću navedenih metoda. Pa tako ukoliko unutar istog puta kojeg stvaramo, dva puta pozovemo naredbu *stroke*, drugo pozivanje će ponovno iscrtavati sve ono što je unutar prvog pozivanja bilo već iscrtano i to onim vrijednostima svojstava koje svojstva imaju u trenutku izvođenja tog poziva. Primjer iscrtavanja puta dan je na Slikama 23. i 24.

Na početku primjera postavljamo širinu i visinu <canvas> elementa te svojstvo *id.* Vrijednost tog svojstva referenciramo u skripti tako da preko njega pomoću metode *document.getElementById* dohvaćamo <canvas> element u varijablu *canvas_element.* Nakon toga u varijablu *kontekst* stavljamo kontekst <canvas> elementa koristeći metodu *getContext.* Sada krećemo uređivati kontekst. Boju puta određujemo metodom *strokeStyle* pa će tako prvi put koji se sastoji dvije linije biti obojan zelenom, drugi put koji sadrži Bezierovu krivulju ljubičastom, a treći put koji sadrži luk crvenom bojom. Metodom *moveTo* se pozicioniramo unutar prozora. Ishodište se nalazi u gornjem lijevom kutu, a vrijednosti koordinata za ovu metodu, a tako i za ostale metode zadajemo u slikovnim elementima. Metoda *lineTo* povlači označava kontekstu da treba povući liniju od trenutnih koordinata na koje smo se pozicionirali metodom *moveTo* do koordinate koje dajemo kao argument metode. Ta metoda samo označava, dok *stroke* metoda iscrtava. Metoda *bezierCurveTo* crta Bezierovu krivulju krenuvši iz točke na koju smo pozicionirani uz pomoć dvije kontrolne točke čije koordinate zadajemo kao prva četiri

argumenta metode do konačne točke čije koordinate zadajemo kao zadnja dva argumenta. Za crtanje luka koristili smo *arc* metodu kojoj kao argumente redom zadajemo konačnu točku, radijus, početni i završni kut te istinitosnu vrijednost kojom odlučujemo hoće li luk biti iscrtan u smjeru obrnutom od onog kojim ide kazaljka na satu ili u suprotnom. Na kraju kako bi se crtež lakše mogao pratiti ispisan je tekst uz svaku liniju. To smo ostvarili uz pomoć metoda *font, fillStyle* i *fillText* kojima smo odredili font kojim će se pisati, boju teksta te položaj ispisa teksta na ekranu.

```
<!DOCTYPE html>
<html>
<body>
<canvas id="primjer1" width="600" height="350"
style="border:5px dashed black;"></canvas>
<script>
    var canvas_element=document.getElementById("primjer1");
    var kontekst=canvas_element.getContext("2d");

    kontekst.beginPath();
    kontekst.lineWidth="5";
    kontekst.strokeStyle="green";
    kontekst.moveTo(500,350);
    kontekst.lineTo(600,250);
    kontekst.moveTo(5,0);
    kontekst.lineTo(80,120);
    kontekst.stroke();

    kontekst.beginPath();
    kontekst.strokeStyle="purple";
    kontekst.moveTo(80,120);
    kontekst.bezierCurveTo(180,380 , 240,-50 , 450,100);
    kontekst.stroke();

    kontekst.beginPath();
    kontekst.strokeStyle="red";
    kontekst.arc(450, 200, 60, 0.1*Math.PI, 0.9 * Math.PI, false);
    kontekst.stroke();

    kontekst.font = 'italic 12pt Calibri';
    kontekst.fillStyle = 'blue';
    kontekst.fillText('Linija 1!', 470, 320);
    kontekst.fillText('Linija 2!', 60, 60);
    kontekst.fillText('Bezier!', 200, 200);
    kontekst.fillText('Arc!', 500, 200);
</script>
</body>
</html>
```

Slika 23. Crtanje puta pomoću <canvas> oznake



24. Izvršavanje koda sa Slike 23.

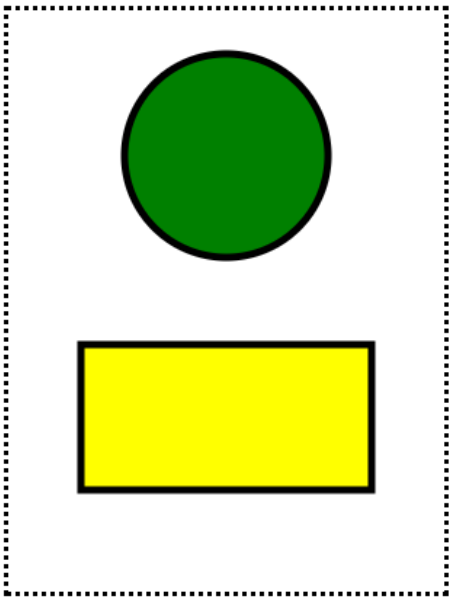
4.1.2. Crtanje geometrijskih likova

<canvas> nema mogućnost iscrtavanja geometrijskih likova osim za pravokutnik, no zbog potpore koje daju metode za iscrtavanje puta možemo iscrtati svaki lik koji nam je potreban. Slike 25. i 26. prikazuju primjer korištenja *rect* metode kojom iscrtavamo pravokutnik te *arc* metode kojom iscrtavamo krug. Metodi *rect* predajemo koordinate gornjeg lijevog kuta te dužinu i širinu pravokutnika. Pomoću *fillStyle* metode određujemo boju kojom će pravokutnik biti ispunjen te ga iscrtavamo metodom *fill.* Ako želimo iscrtati samo unutrašnjost pravokutnika, bez obruba, tada nije potrebno pozvati *stroke* metodu, ali za iscrtavanje obruba jest. Krug smo nacrtali uz pomoć već spomenute metode *arc* tako da smo kao početni kut uzeli vrijednost od nula stupnjeva, a kao završni 2π vrijednost, odnosno 360 stupnjeva.

Pravokutnik možemo crtati i uz pomoć *fillRect* metode koja prima jednake argumente kao i *rect* metoda te mijenja metodu *fill.* Takvim načinom stvaranja pravokutnika iscrtavamo samo unutrašnjost pravokutnika, bez obruba.

```
<!DOCTYPE HTML>
<html>
  <body>
    <canvas id="myCanvas" width="300" height="400"
      style="border:dotted; border-color:black;"></canvas>
    <script>
      var canvas = document.getElementById('myCanvas');
      var context = canvas.getContext('2d');

      <!--pravokutnik->
      context.beginPath();
      context.rect(50, 230, 200, 100);
      context.fillStyle = 'yellow';
      context.fill();
      context.lineWidth = 5;
      context.strokeStyle = 'black';
      context.stroke();

      <!--krug->
      var centerX = canvas.width / 2;
      var centerY = canvas.height / 4;
      var radius = 70;

      context.beginPath();
      context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
      context.fillStyle = 'green';
      context.fill();
      context.stroke();
    </script>
  </body>
</html>
```

Slika 25. Crtanje geometrijskih likova pomoću <canvas> oznake [7]

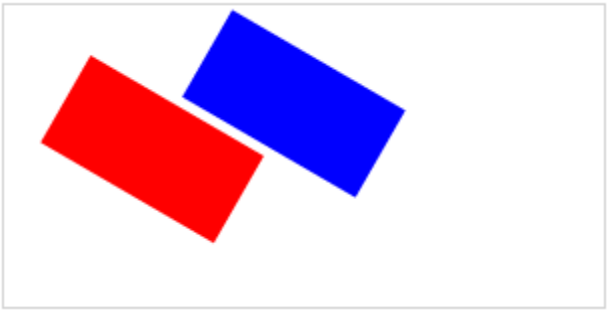


26. Izvršavanje koda sa Slike 25.

4.1.3. 2D transformacije nad canvas kontekstom

Za razliku od transformacija korištenjem oznake <svg> gdje smo transformacije obavljali nad objektima, ovdje transformacije obavljamo nad kontekstom. Ovisno o tome u kojem dijelu koda definiramo koju transformaciju, djelovanje na elemente će biti različito jer transformacija utječe samo na crteže napravljene nakon izvršene transformacije (Slika 27.). Možemo gledati na njih kao da su u sloju iznad crteža koji će se iscrtati nakon daljnjih transformacija. Transformacije koje <canvas> element pruža su *translate, rotate, scale* i *transform.* Metoda *transform* ima šest parametara od kojih prvi skalira kontekst horizontalno, a četvrti parametar vertikalno. Drugi parametar radi smik horizontalno, a treći vertikalno. Zadnja dva parametra obavljaju translaciju u *x* i *y* smjeru. Primjer korištenja *transform* metode možemo vidjeti na Slikama 28. i 29.

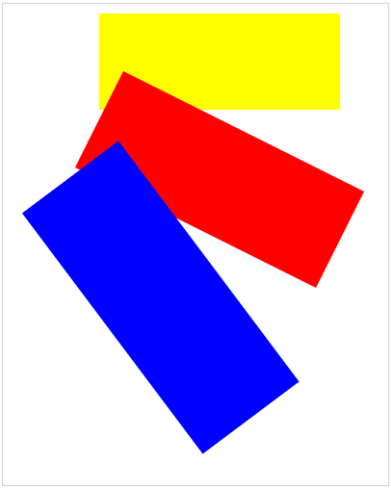
```
ctx.rotate(Math.PI/6);
ctx.fillStyle="red";
ctx.fillRect(50,0,100,50);

ctx.translate(50,-55);
ctx.fillStyle="blue";
ctx.fillRect(50,0,100,50);
```



Slika 27. Metoda *translate* utjecala je samo na plavi pravokutnik

```
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="400" height="500"
style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>

<script>

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

    ctx.fillStyle="yellow";
    ctx.fillRect(100,10,250,100)

    ctx.transform(1,0.5,-0.5,1,30,10);
    ctx.fillStyle="red";
    ctx.fillRect(100,10,250,100);

    ctx.transform(1,0.5,-0.5,1,30,10);
    ctx.fillStyle="blue";
    ctx.fillRect(100,10,250,100);

</script>

</body>
</html>
```

Slika 28. Transformacija nad canvas kontekstom uporabom metode *transform* [7]



29. Izvršavanje koda sa Slike 28.

4.2. Animacija uz pomoć <canvas> oznake

Korištenje animacije uz pomoć oznake <canvas> prikazat ćemo kroz primjer (Slike od 30. do 33.). Metoda *requestAnimationFrame* govori pregledniku da obavi animaciju i traži od njega da ponovno iscrta prozor s novim okvirom (engl. frame) animacije. Pri pozivu metode kao argument predajemo funkciju koju će metoda pozvati prilikom ponovnog iscrtavanja prozora. Funkcija setTimeout poziva funkciju predanu kao prvi argument nakon odgode koja ima trajanje vrijednosti drugoga argumenta. U našem primjeru predali smo definiciju funkcije koja uzima trenutno vrijeme te ga kao jedan od argumenata predaje funkciji *animate.*

```
window.requestAnimFrame = (function(callback) {
      return window.requestAnimationFrame ||
      window.webkitRequestAnimationFrame ||
      function(callback) {
          window.setTimeout(callback, 1000 / 60);
        };
      })();

//deklaracija funkcija

var canvas = document.getElementById('myCanvas');
var context = canvas.getContext('2d');
var myCircle = {
        x: 250,
        y: 70,
        borderWidth: 5
      };

drawCircle(myCircle, context);
// wait one second before starting animation
setTimeout(function() {
        var startTime = (new Date()).getTime();
        animate(myCircle, canvas, context, startTime);
      }, 1000);
```

Slika 30. Prvi dio koda primjera animacije canvas konteksta [7]

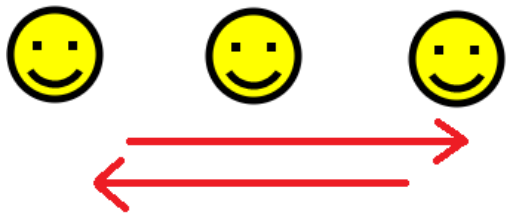
Metoda *animate* prima 4 argumenta. Prvi je struktura *myCircle* koju smo napravili kako bi mogli čuvati podatke za krugove koje iscrtavamo. Struktura u sebi sadrži *x* i *y* koordinatu te debljinu obruba. Animaciju obavljamo tako da mijenjamo *x* koordinatu kruga. Novu *x* koordinatu računamo pomoću sinusne funkcije unutar metode *animate* te nakon što izračunamo pomoću *clearRect* metode brišemo sadržaj konteksta te pozivamo *drawCircle* metodu kojom iscrtavamo lik koji želimo animirati. Metode *animate* i *drawCircle* nalaze se na Slikama 31. i 32.

```
function animate(myCircle, canvas, context, startTime) {
        // update
        var time = (new Date()).getTime() - startTime;
        var amplitude = 150;
        // in ms
        var period = 2000;
        var centerX = canvas.width / 2 - 30 / 2;
        var nextX = amplitude * Math.sin(time * 2 * Math.PI / period)
                              + centerX;
        myCircle.x = nextX;
        // clear
        context.clearRect(0, 0, canvas.width, canvas.height);
        // draw
        drawCircle(myCircle, context);
        // request new frame
        requestAnimFrame(function() {
          animate(myCircle, canvas, context, startTime);
        });
      }
```

Slika 31. Drugi dio koda primjera animacije canvas konteksta, metoda *animate* [7]

```
function drawCircle(myCircle, context) {
    context.beginPath();
    context.arc(myCircle.x, myCircle.y, 30, 0, 2 * Math.PI, false);
    context.fillStyle = 'yellow';
    context.fill();
    context.lineWidth = myCircle.borderWidth;
    context.strokeStyle = 'black';
    context.stroke();
    context.beginPath();
    context.arc(myCircle.x, myCircle.y,
                20, -7.1*Math.PI/6, 1.1*Math.PI/6, true);
    context.stroke();
    context.fillStyle = 'black';
    context.fillRect(myCircle.x-15, myCircle.y-9,6,6);
    context.fillRect(myCircle.x+9, myCircle.y-9,6,6);
}
```

Slika 32. Treći dio koda primjera animacije *canvas* konteksta, metoda *drawCircle* [7]



Slika 33. Izvršavanje koda sa Slika 30.-32.

5. Proširena stvarnost i HTML5

Proširena stvarnost (engl. augmented reallity) je područje računarske znanosti koje se bavi dodavanjem osjetilnih elemenata u stvarnost koje zapravo u stvarnosti ne postoje nego je, kao što sam naziv kaže, stvarnost proširena računalom stvorenim unosom. Na taj način okruženje postaje digitalizirano te uz koristeći tehnologije za prepoznavanje objekata možemo s proširenom okolinom obavljati interakciju. U nastavku će biti prikazano nekoliko primjera implementacije proširene stvarnosti uz pomoć HTML5 standarda.

5.1. Primjeri

Prvi primjer [10] radi uz pomoć biblioteke koja implementira algoritam za prepoznavanje lica *CCV*[10] te <canvas> i <video> oznake te implementira virtualne naočale koje prate kretanje osobe koja stoji ispred web kamere. Program radi tako da svakih 200 milisekudni uzima jedan okvir (engl. frame) videa te predaje to algoritmu za detektiranje lica. U skladu s time kako se mijenja položaj lica, naočale se iscrtavaju na drugom mjestu na ekranu. Primjer izvođenja algoritma za otkrivanje lica prikazan je na Slici 34., a primjer izvođenja programa na Slici 35.

Slika 34. Primjer izvođenja *CCV* algoritma za otkrivanje lica [11]

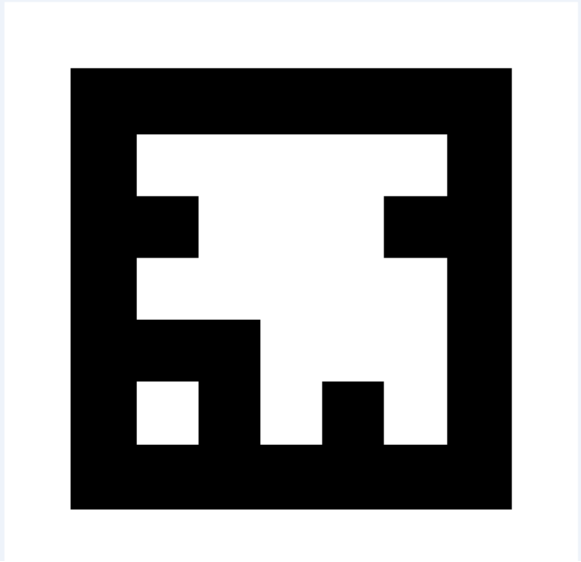
[10] punim imenom CCV JavaScript Face Detection [9]

Slika 35. Primjer izvođenja primjera stvarnosti proširene virtualnim naočalama [11][12]

Drugi primjer [13] isto tako uz pomoć <canvas> i <video> oznake sve što je na slici zeleno pretvara u crvenu boju. Program radi tako da pronalazi zelene slikovne elemente na okviru kojeg prima preko video toka (engl. video stream) te iscrtava na tom mjestu crvene slikovne elemente. Primjer izvođenja programa prikazan je na Slici 36.

Slika 36. Prikaz izvođenja drugog primjera proširene stvarnosti [13]

Treći primjer [15] prikazuje stvaranje slike iz prirode na A4 papiru na kojem se nalazi marker (Slika 37.). Primjer koristi JSARToolKit[11] [14] biblioteku koja radi nad <canvas> elementom. <canvas> element šaljemo JSARToolKit biblioteci, ona nakon obrade elementa vraća listu markera koje pronalazi na slici. Algoritam programa radi tako da ako unutar liste bude marker koji odgovara definiranom markeru u kodu, područje na videu unutar kojeg se nalazi marker se mijenja s jednom od slika. Kako bi primjer radio, moramo isprintati zadani marker te ga postaviti ispred web kamere. Dva prikaza izvođenja primjera su na Slikama 38. i 39.



Slika 37. Izgled markera trećeg primjera proširene stvarnosti

[11] JavaScript biblioteka za proširenu stvarnost

Slika 38. Prikaz izvođenja trećeg primjera proširene stvarnosti broj 1

Slika 39. Prikaz izvođenja trećeg primjera proširene stvarnosti broj 2

6. Zaključak

HTML5 standard doseže veliku popularnost u svome korištenju iako još nije u potpunosti implementiran. S pravom ga nazivaju “budućnost Interneta” jer svojim karakteristikama daje korisniku visoke mogućnosti izradi Internet stranica. Ovaj rad se bazirao na mogućnosti standarda u području računalne grafike te su promotreni standard CSS3, oznake <canvas> i <svg> te neki primjeri proširene stvarnosti implementirani uz pomoć HTML5.

CSS3 standard daje stil HTML elementima, odnosno određuje način na koji se HTML elementi, predstavljeni HTML oznakama, prikazuju na ekranu. Kroz rad je pokazano kako CSS3 može utjecati na HTML5 oznake mijenjajući ih grafičkim transformacijama poput skaliranja, rotiranja, translatiranja i smika. Mogućnost prikaza inače 2D HTML elemenata u 3D sustavu otvara nove mogućnosti u izradi i kompoziciji Internetskih stranica, a mogućnost animiranja će ih oživjeti i pridodati im na dinamičnosti.

Oznaka <svg> daje nam mogućnost da unutar HTML5 standarda koristimo SVG jezik za dvodimenzionalnu grafiku. Zbog činjenice da SVG koristi vektorsku grafiku crteži nacrtani preko te oznake ne gube kvalitetu pri promjeni rezolucije te zauzimaju malo memorije. Oznaku <svg> vrlo je lako koristiti jer su elementi koje sadrži objekti kojima za iscrtavanje moramo samo odrediti vrijednosti svojstava. Takav način organizacije daje nam mogućnost laganog korištenja grafičkih transformacija u 2D prostoru te animacija SVG elemenata. Transformacije nad <svg> oznakom se u 3D prostoru mogu ostvariti, kao što je i pokazano, uz pomoć CSS3 standarda.

Oznaka <canvas> unutar HTML5 standarda daje iznimno velike mogućnosti kako u grafici, tako i u području proširene stvarnosti. <canvas> oznaka puni svoj *Canvas* kontekst te iscrtava sve što je u njemu. Transformacije se vrše nad *cavas* kontekstom te je potrebno pripaziti na redoslijed kojim naredbe upisujemo kako ne bi dobili pogrešan rezultat. Animacije se vrše tako da se ciklično poziva funkcija za iscrtavanje s promjenom nekih parametara.

Proširena stvarnost predstavlja područje računarske znanosti koje, iako napreduje vrlo brzo, je relativno mlado te je zasigurno budućnost računalne

tehnologije i područje u kojem će uvijek biti mjesta za napredak jer je algoritme koje koristi za svoju realizaciju gotovo nemoguće napraviti 100% nepogrešivim.

7. Literatura

[1] Kinlan, 3D and CSS, 07.07.2010., http://www.html5rocks.com/en/tutorials/3d/css/, pristupljeno 10.06.2010.

[2] DeSandro, An introduction to CSS 3-D transformations, 14.12.2010, http://24ways.org/2010/intro-to-css-3d-transforms/, pristupljeno 10.06.2010.

[3] CSS3 @keyframes Rule, http://www.w3schools.com/cssref/tryit.asp?filename=trycss3_keyframes4 , pristupljeno 10.06.2010.

[4] Bitmap and vector graphics, http://www.bbc.co.uk/schools/gcsebitesize/dida/graphics/bitmapvectorrev1.shtml, pristupljeno 10.06.2013.

[5] SVG <rect>, http://www.w3schools.com/svg/svg_rect.asp, pristupljeno 10.06.2013.

[6] animateMotion, https://developer.mozilla.org/en-US/docs/Web/SVG/Element/animateMotion, pristupljeno 10.06.2010.

[7] Rowell, 08.04.2013., HTML5 Canvas Element Tutorial, http://www.html5canvastutorials.com/tutorials/html5-canvas-element/, 11.06.2013.

[8] Seidelin, HTML5 Canvas Cheat Sheet, 22.02.2009., http://blog.nihilogic.dk/2009/02/html5-canvas-cheat-sheet.html, pristupljeno 11.06.2013.

[9] Liu,CCV, 10.09.2012., http://libccv.org/, pristupljeno 12.06.2012.

[10] Bos, JavaScript Fave Detection + Canvas + Video = HTML5 Glasses!, 27.06.2011., http://wesbos.com/html5-video-face-detection-canvas-javascript/, pristupljeno 12.06.2012.

[11] Liu, A Not-so-slow JavaScript Face Detector, http://liuliu.me/ccv/js/nss/, pristupljeno 12.06.2013.

[12] Neave, Face Detection, 24.04.2012., http://neave.github.io/face-detection/, pristupljeno 12.06.2013.

[13] Frost, Web Gestures With getUserMedia:Part1, 14.,09.,2012., http://40win.com/blog/2012/11/14/web-gestures-with-getusermedia-part1/, pristupljeno 12.06.2013.

[14] JSARToolKit, https://github.com/kig/JSARToolKit, 13.06.2013.

[15] Heikkinen, Writing Augmented Reality Applications Using JSARToolKit, 28.02.2012., http://www.html5rocks.com/en/tutorials/webgl/jsartoolkit_webrtc/, pristupljeno 13.06.2013.

[16] Gorner, Animate your HTML5, 25.02.2013., http://animateyourhtml5.appspot.com, pristupljeno 12.06.2013.

[17] Bennett, Lubbers, McAnlis, HTML5 Game Development Course, https://www.udacity.com/course/cs255, pristupljeno 12.06.2013.

[18] Elliot, Getting started with SVG for HTML5, 28.10.2011., http://www.i-programmer.info/programming/graphics-and-imaging/2063-getting-started-with-svg-html5.html, pristupljeno 12.06.2013.
[19] Wikipedia, CSS, 05.04.2013., http://hr.wikipedia.org/wiki/CSS, pristupljeno 12.06.2013.
[20] Firdaus, A Look Into: Scalable Vector Graphics (SVG) Animation, http://www.hongkiat.com/blog/scalable-vector-graphics-animation/, pristupljeno 10.06.2013.
[21] SMIL animations embedded in SVG, http://srufaculty.sru.edu/david.dailey/svg/intro/PartC.htm, pristupljeno 10.06.2013.

8. Sažetak

Naslov: **Interaktivni grafički objekti u standardu HTML5**

U radu su promotrene mogućnosti HTML5 standarda u području računalne grafike. Promatrali smo kako HTML5 koristi svojstva CSS3 standarda koji uređuje prikaz HTML oznaka. Pri tome dotaknute su teme prikaza 2D i 3D grafičkih objekata, transformacije nad njima te animacija tih objekata. Promotrene su mogućnosti SVG jezika kao <svg> oznake te <canvas> oznake HTML5 standarda. Prikazani su osnovni primjeri korištenja 2D transfomacija nad grafičkim objektima tih oznaka te animacije crteža u 2D prostoru. Na kraju rada, dotaknuta je tema proširene stvarnosti unutar HTML5 standarda te je prikazano nekoliko primjera.

Ključne riječi: HTML5, CSS3, SVG, Canvas, proširena stvarnost

9. Abstract

Title: **Interactive grafical objects in HTML5 standard**

In this work we examined possibilities of the HTML5 standard in the field of computer graphics. We saw how HTML5 uses the properties of CSS3 standard which govern the display of HTML tags. Then we touched upon the topics of 2D and 3D graphical objects, transformations over them and animation of these objects in CSS3. We examined possibilities of the <canvas> tag and the SVG language as <svg> tag inside HTML5 standard. Some basic examples of using 2D transfomations on graphical objects and animation of drawings in 2D space are shown. In the end, we addressed the topic of augmented reality in the HTML5 standard and presented a few examples.

Ključne riječi: HTML5, CSS3, SVG, Canvas, proširena stvarnost