Razvoj filtera za slike u HTML5 web sučelju

Prlić, Matija

SVEUČILIŠTE U ZAGREBU
GRAFIČKI FAKULTET

ZAVRŠNI RAD

Matija Prlić

Sveučilište u Zagrebu
Grafički fakultet

Smjer: Tehničko tehnološki

ZAVRŠNI RAD

RAZVOJ FILTERA ZA SLIKE U HTML5 WEB SUČELJU

Mentor:

Prof.dr.sc. Klaudio Pap

Student:

Matija Prlić

Zagreb, 2015.

SAŽETAK

U ovom završnom radu objašnjava se princip rada manipulacije slikama putem programskog jezika JavaScript i novog prezentacijskog jezika za izradu web stranica HTML5. Završni rad sadrži informacije potrebne za shvaćanje manipulacije slikama, od piksela, RGB paleta boja, struktura, sadržaj i mogućnosti HTML dokumenta, pa sve do same izrade filtera na slikama u HTML5 web okruženju. Također se opširnije objašnjava korištenje kodova u JavaScriptu preko kojih se može manipulirati slikom i stvaranje različitih vrsta filtera, poput crno-bijelog, negativa, „šuma“ i drugih efekata. Cilj samoga rada jest saznati i obrazložiti koje sve mogućnosti nudi za manipulaciju slika nova tehnologija HTML5 sa novim elementom „canvas“, kao i razvoj vlastitoga filtera. Za izradu završnoga rada poslužili su različiti materijali, od knjiga povezanih sa HTML5 tehnologijom i različitih poveznica na Internetu.

Istraživanje se bazira na razvoju jednostavnih i složenih filtera za sliku, koji se postižu novim web tehnologijama. Element "canvas" u HTML5 jeziku kroz skriptiranje nudi mogućnost manipulacije slikom. Kreirat će se novi filteri za prikaz piksel grafike na web-u, kroz programiranje u Javascriptu. Cilj istraživanja je pronaći nova rješenja u manipulaciji piksel grafikom, koja će odgovarati različitim web sučeljima. Filteri će se testirati na različitim platformama i uređajima (Windows, MacOS, Tablet, Smartphone) kako bi se utvrdila podrška uređaja za nove tehnologije. Kreiranje filtera će se provoditi u programu Adobe Dreamweaver pomoću HTML5, CSS3, Javascript tehnologije.

HTML5, JAVASCRIPT, CANVAS, PIKSEL

SADRŽAJ

1. UVOD

HTML je kratica za HyperText Markup Language, što znači prezentacijski jezik za izradu web stranica. Hipertekst dokument stvara se pomoću HTML jezika. HTML jezikom oblikuje se sadržaj i stvaraju se hiperveze hipertekst dokumenta [1]. HTML je jednostavan za uporabu i lako se uči, što je jedan od razloga njegove opće prihvaćenosti i popularnosti. Svoju raširenost zahvaljuje jednostavnosti i tome što je od početka bio zamišljen kao besplatan i tako dostupan svima. Prikaz hipertekst dokumenta omogućuje web preglednik. Temeljna zadaća HTML jezika jest uputiti web preglednik kako prikazati hipertekst dokument. Pri tome se nastoji da taj dokument izgleda jednako bez obzira o kojemu je web pregledniku, računalu i operacijskom sustavu riječ. HTML nije programski jezik niti su ljudi koji ga koriste programeri. Njime ne možemo izvršiti nikakvu zadaću, pa čak ni najjednostavniju operaciju zbrajanja ili oduzimanja dvaju cijelih brojeva. On služi samo za opis naših hipertekstualnih dokumenata. HTML datoteke su zapravo obične tekstualne datoteke; ekstenzija im je .html ili .htm. Osnovni građevni element svake stranice su oznake (tags) koji opisuju kako će se nešto prikazati u web pregledniku. Povezice unutar HTML dokumenata povezuju dokumente u uređenu hijerarhijsku strukturu i time određuju način na koji posjetitelj doživljava sadržaj stranica.

Prvi javno dostupan opis HTML-a je dokument zvan HTML „tags“ (oznake), prvi put se spominje na internetu od strane Tim Berners-Leeja krajem 1991 [2]. Taj opis se sastoji od 20 elemenata početnog, relativno jednostavnog dizajna HTML-a. Trinaest tih elemenata još uvijek postoji u HTML4 verziji. Postanak mnogih svojih oznaka duguje jednom od ranih jezika za formatiranje teksta, „runoff-u“. „Runoff“ je razvijen u ranim 1960-im za CTSS (Kompatibilni Time-Sharing System) operacijski sustav. „Runoff“ je kasnije inkorporiran u UNIX operativni sustav u naprednije formatirajuće programe kao što su „roff“, „nroff“ i „troff“. Svaka nova verzija HTML-a je razvijana tako da ostane čitljiva na svim web preglednicima. Tim Berners-Lee je, nakon što je u listopadu 1994. napustio CERN (Europsku organizaciju za nuklearno istraživanje), osnovao organizaciju World Wide Web Consortium koja se bavi standardizacijom tehnologija korištenih na webu poznatija kao W3C.

Prva verzija HTML jezika objavljena je 1993. godine. U to je vrijeme bio još poprilično ograničen, pa nije bilo moguće čak ni dodati slike u HTML dokumente. Razvoj HTML-a nastavljen je prvom "imenovanom" verzijom 2.0, no ni ona nije postala standardom.

U ožujku 1995. W3C objavljuje verziju 3.0, koja donosi mogućnosti definicije tablica. Daljnji razvoj ove verzije HTML-a označilo je prihvaćanje "specifičnih" oznaka podržanih u tada najvećim i najprihvaćenijim web preglednicima. Tako su nastale mnoge duplikacije, pa je postojalo više oznaka koje su imale istu funkciju. Podebljani tekst, primjerice bilo je moguće definirati oznakom `<b>`, ali i oznakom `<strong>`.

HTML4 predstavljen je u prosincu 1997., nastavio je s prihvaćanjem oznaka nametnutih od strane proizvođača različitih web preglednika, no istovremeno je pokrenuto i "čišćenje" standarda i proglašavanje nekih od njih suvišnim. Manje promjene u specifikaciji ovog standarda predstavljene su u prosincu 1999., kada je predstavljena konačna verzija ovog jezika HTML 4.01.

HTML5 je prva nova revizija standarda od HTML 4.01, koji je izdan 1999. Nastao u suradnji World Wide Web Consortium (W3C) i Web Hypertext Application Technology Working Group (WHATWG). Do 2006. godine su ove dvije grupe radile odvojeno, WHATWG je radio sa web formama i aplikacijama, a W3C sa XHTML 2.0. Tokom vremena odlučili su udružiti snage i kreirati novu verziju HTML-a. Izdavanje konačnih specifikacija standarda HTML5 u suprotnosti je s inicijativom Web Hypertext Application Technology Working Group (WHATWG) prema kojoj bi HTML trebao biti "živi" standard koji se stalno nadograđuje, bez oznake verzije specifikacija. Danas preglednici imaju impementiranu podršku za većinu HTML5 elemenata [3].

HTML5 donosi brojne nove mogućnosti koje HTML 4.01 i XHTML 1.x nisu imali, kao što je mogućnost reprodukcije videa na stranicama bez korištenja Adobe flasha ili Microsoftovog silverlighta, mogućnost upravljanja pomoću tipkovnice i opcijama za bilo koju vrstu manipulacija, „drag and drop“, „canvas“ kao i ostali novi elementi.

Svaki HTML dokument sastoji se od osnovnih građevnih blokova - HTML elemenata. Svaki, pak, HTML element sastoji se od para HTML oznaki (engl. tag). Također, svaki element može imati i atribute kojim se definiraju svojstva tog elementa. Na samom početku HTML dokumenta preporučljivo je postaviti `<!DOCTYPE>` element, kojim se

označava DTD (engl. Document Type Declaration), čime se definira točna inačica standarda koja se koristi za izradu HTML dokumenta. Nakon `<!DOCTYPE>` elementa, `<html>` elementom označava se početak HTML dokumenta. Unutar `<html>` elementa nalaze se i `<head>` element te `<body>` element. `<head>` element predstavlja zaglavlje HTML dokumenta u kojemu se najčešće specificiraju jezične značajke HTML dokumenta kao i sam naslov (engl. title) stranice. Pomoću određenih HTML elemenata unutar zaglavlja dodaju se i stilska obilježja stranice, bila ona direktno ugrađena (engl. embedded) ili dodana kao referenca na vanjsku CSS datoteku. Često se unutar zaglavlja još definiraju i skripte kreirane u JavaScript jeziku. U `<body>` elementu kreira se sadržaj HTML dokumenta, odnosno, stranice koju on reprezentira.

Svaka HTML oznaka (koja u paru kreira HTML element) počinje znakom < (manje od), a završava znakom > (više od). Zatvarajuća HTML oznaka kreira se na isti način kao i otvarajuća, ali se prije završnog znaka > dodaje i kosa crta / (engl. slash).

Primjer jednog HTML elementa: `<body></body>`

Osim navedenih, standardnih HTML elemenata, postoje i samozatvarajući HTML elementi. Kod takvih elemenata nema zatvarajuće oznake.

Primjer samozatvarajućeg elementa: `<link rel="stylesheet" type="text/css" href="stil.css" />`

Iako nije nužno, prema preporuci W3C-a, kod samozatvarajućih elemenata poželjno je ostaviti razmak između definiranih atributa i njihovih vrijednosti i zatvarajućih znakova (/>).

Svaki HTML dokument moguće je kreirati u bilo kojem uređivaču teksta. Ipak, za neke naprednije funkcije, kao i za olakšavanje repetitivnih aktivnosti kod izrade HTML dokumenata te vizualni pretpregled kreirane web stranice, preporučljivo je koristiti neke od funkcionalnijih alata, poput Adobeovog Dreamweavera ili Microsoftovog Expression Weba.

Primjer jednostavnog HTML dokumenta:

```
<html>
<head>
```

```
<title> </title>
</head>
<body>
<p> </p>
<img src="image.jpg" />
</body>
</html>
```

Objašnjenje:

`<html> </html>` su prvi i posljednji tag u web stranici i između njih se nalazi se cijela struktura web stranice. Može se primjetiti kako je drugi tag isti kao prvi samo što ima kosu crtu ispred tag naziva `</html>`. Tagovi dolaze u parovima kako bi označili početak i završetak njihovog djelovanja.

U četvrtom retku je tag koji dolazi u paru. `<title>` označava početak navođenja teksta koji će biti ispisan u naslovnoj traci a `</title>` označava kraj.

`<body> </body>` ovdje je mjesto gdje se nalazi većina stranice html dokumenta. Sve što se može vidjeti u prozoru preglednika je prikazano i sadržano u tom elementu, uključujući odlomke teksta, poveznice, slike, tablice i mnogo ostalih stvari. Kako će izgledati stranica u prozoru preglednika isključivo ovisi o sadržaju kako je ispunjen unutar tog elementa.

`<img src="image.jpg" />` se koristi za umetanje slika u html dokumentu. Osim umetanja slike pomoću „img“ (slika, eng. image) elementa potrebno je i odrediti izvor slike gdje se nalazi pomoću naredbe „src“ (izvor, eng. source). Nakon što se ispravno unese rezultat tog elementa je da se na prozoru preglednika dobije željena slika koju smo unijeli unutar elementa.

Postoje još mnogi dodatni parametri koji omogućuju manipuliranje slikama, tako imamo opciju „poravnanje“ (eng. align) s kojom možemo odrediti poziciju slike u odnosu na tekst (lijevo, centrirano, desno) i slika će se pojaviti na poziciji u prozoru preglednika ovisno koju opciju smo odabrali, također postoji i horizontalni odnosno „vertikalni razmak“ (eng. hspace, vspace) i rezultat svega toga jest da ćemo dobiti razmak između slike horizontalno odnosno vertikalno u iznosu koji smo unijeli sa ostatkom sadržaja koji se nalazi zajedno sa slikom u html dokumentu. Postoji i opcija

dodavanja obruba (eng. border) s kojom možemo manipulirati debljinu obruba slike koja će biti one debljine koju smo definirali u html dokumentu. Ako slika nije pronađena od strane preglednika, atribut „alt“ će prikazati tekst na web stranici koji smo zadali. Za određivanje visine ili širine slike koristimo „width“ i „height“, gdje width označava horizontalnu veličinu, a height vertikalnu veličinu.

Osim ovakvog načina prikazivanja slika u HTML-u imamo također i mogućnost umetanja slika preko SVG-a i canvas-a. SVG je skalirana vektorska grafika (eng. Scalable Vector Graphics), koja koristi XML jezik za prikazivanje dvodimenzionalne vektorske grafike, bilo nepomične ili animirane. Canvas je element unutar kojeg se mogu crtati razni geometrijski oblici (kvadrat, krug, elipsa), izrađivati grafove, procesuirati slike, može se dodavati tekst ili izraditi animacije.

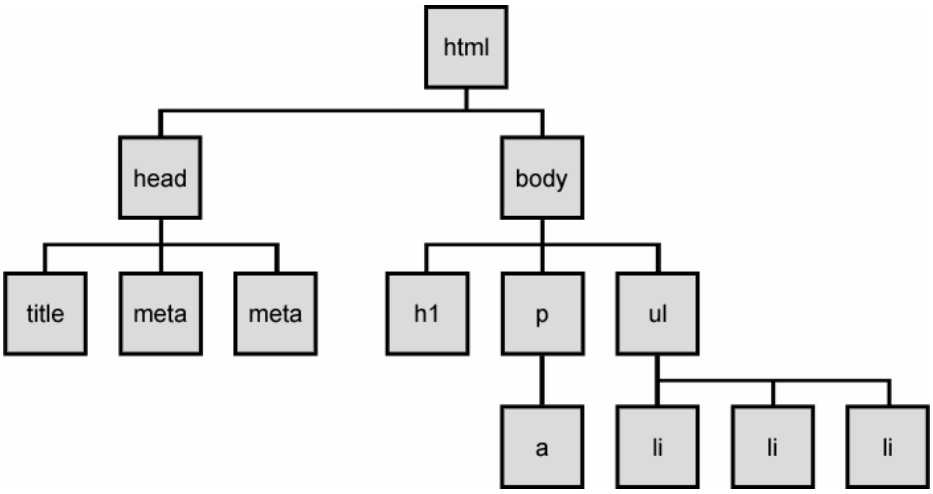
2. TEORIJSKI DIO

2.1 Struktura web dokumenta za prikaz slikovnog sadržaja

DOM (Document object model)

DOM ili Document Object Model, objektni model dokumenta, je više-platformsko aplikacijsko programsko sučelje (API) za HTML, neovisno o programskom jeziku. DOM definira logičku strukturu dokumenta i način na koji se dokumentu pristupa i manipulira s njim. Uz pomoć DOM-a se stvaraju, ažuriraju ili brišu dokumenti i pruža strukturirani prikaz dokumenata (u obliku stabla) te omogućava promjenu njegovog sadržaja i izgleda [4].

Postoji više razina DOM-a koji se međusobno razlikuju po objektima koje sadrže kao i mogućnostima mijenjanja svojstava i prikaza tih objekata. Primjer pojednostavljene sheme elemenata DOM-a dan je na slici 1. gdje čvorovi u „stablu“ imaju hijerarhijski odnos, tako HTML ima dva čvora head i body, a opet svako od njih sadrži još čvorova sačinjavajući sve zajedno „stablo“ za DOM.



Slika 1. – Izgled Document Object Modul „stabla“

slika preuzeta sa:

https://courses.cs.washington.edu/courses/cse190m/07sp/lectures/slides/images/dom_tree.gif

HTML5 kao i prethodne verzije HTML-a ima osnovnu funkciju povezivanja na Internetu, s time da se do sada povezivanje vršilo uglavnom putem hiperteksta sa statičnim slikama, a novom verzijom omogućena je potpuna ugradnja multimedijskih elemenata. Time su „audio“ i „video“ elementi samo nastavak „object“ elementa koji je bio dio prethodne inačice HTML-a. Ta dva elemenata su zamijenila element „object“ koji su ljudi godinama koristili za izradu web stranica, ponajviše koristeći za povezivanje videa sa stranica poput YouTube ili Vimeo. Ti elementi imaju funkciju prikazivanja videa na web stranici i zadržavanje korisnika na istoj, a ne preusmjeravati na web stranicu od koje potječe izvor toga videa. U tom slučaju, čak i „img“ elementi su na neki način ništa više nego dio HTML elementa koji vuče sadržaj sa web stranice na drugoj lokaciji. Sve se svodi na povezivanje što je pojam i svrha HTML-a. Sada se mogu povlačiti slike, audio i video direktno u dokument, a iz JavaScript-a se mogu manipulirati sami audio i video. HTML5 dozvoljava da se integriraju svi elementi u jedan običan dokument i na jedno mjesto. Osim „audio“ i „video“ elemenata tu se još nalazi mnogo ostalih elemenata koji se također mogu upravljati isto kao i navedena dva [5].

Postoji mnogo razloga zašto je HTML5 kvalitetnija inačica od prethodnih. Adobe Flash je oduvijek bio fokusiran na animacije i bolje vizualne efekte i time donio mnogo više integracije na sučelju web stranice i izbjegavanje JavaScripta. Danas u HTML5 imamo bolju povezanost JavaScripta i elemenata HTML dokumenta, gdje JavaScript nudi sve što je nudio i Flash (poput audio, vizualnih i sličnih interakcija).

Iako su web programeri i dizajneri povlačili slike sa ostalih stranica i od ostalih autora, web stranice su i dalje vrlo fleksibilni u korištenju i preuzimanju autorskih prava. Web stranice danas imaju na jednom mjestu tekstualni sadržaj, slike, multimediju.

HTML5 danas sve više posvjećuje pažnju na pametne mobitele, ali postoji i rizik u svemu tome, jer je dosta često slučaj da ono što radi na osobnom ili prijenosnom računalu ne radi i na mobilnim uređajima (pametni mobiteli, tableti). HTML5 ima vrlo

dobru podlogu za korištenje na bilo kojim uređajima koji su danas ljudima nadohvat ruke poput tableta, pametnih mobitela.

HTML5 nudi JavaScript-u vrlo solidan set alata i efekata, usporedivih i sa najvećim zahtjevnim Flash stranicama. Time dobivamo višestruke mogućnosti poput toga da se tekst može odabrati, označavati bez previše zahtjeva, animirati i naravno vlasnici stranica mogu bez problema ažurirati svoje stranice, bez pomoći programera i web dizajnera [6].

Rezultat svega toga jest taj da je HTML5 opet najviše korišten alat koji je dostupan za web sadržaje, s time da je sada još bogatiji nego prije.

HTML5 se puno više fokusira na JavaScript od svojih ostalih inačica. HTML5 dodaje nekoliko novih važnih elemenata (već spomenutih i ključnih audio i video) poput `<canvas>` elementa koji se obrađuje u ovom radu. Pokazat će se da nije potrebno puno rada sa JavaScriptom i ID stukturom za povlačenje elemenata na web stranici. JavaScript-om možemo povlačiti video, kontrolirati njihov prikaz, mijenjati njegovu poveznicu i uspješno pokretati neki drugi video, čak i podešavati njegov zvuk. Što se tiče manipulacije slikom Javascriptom se potpuno kontrolira prikaz slike, od visine, širine i pozicije do manipulacije pikselima i njihovim bojama [7].

CSS (eng. Cascading Style Sheets) predstavlja jednostavan mehanizam za dodavanje forme informacijama u HTML stranici. CSS je kao tehnologija nastala iz nekoliko različitih rješenja kako bi se u osnovi odvojio izgled informacije od samog opisa i same informacije. Nakon nekoliko prezentacija na međunarodnim skupovima 1996. Je donijeta specifikacija CSS-a 1.0.

S obzirom na stanje zastupljenosti HTML preglednika i općih odnosa na tom tržištu prava zastupljenost i prihvatljivost uporabe CSS-a počela je oko 2005. godine kada je većina preglednika u većoj mjeri podržavala CSS 3.0 (ova verzija je u razvojnoj fazi od 1998.).

Uporaba CSS-a u stvaranju današnjih Web sadržaja je postala uobičajena praksa, ne samo zbog svoje jednostavnosti već i zbog činjenice da se Web sadržaji danas dohvaćaju s različitih uređaja koji imaju i različita fizička ograničenja po pitanju

prikazivanja informacija i njihovih opisa pa nam CSS omogućava da Web sadržaj odmah prilagodimo i tim uređajima bez dodatnog izrade novih i specifičnih HTML stranica [8].

U osnovi, CSS je moguće dodati unutar HTML-a na tri načina:

- pozivanjem vanjske datoteke s CSS kodom - najprihvatljiviji način koji ujedno omogućava laganu zamjenu i manipulaciju izgledom:

<link rel="stylesheet" href="css/thickbox.css" type="text/css" media="screen" />

Na ovaj se način ujedno mogu iskoristiti dodatne funkcije kao što je atribut media te omogućiti selektivno učitavanje ovisno o vanjskim parametrima preglednika.

- uključivanjem CSS koda unutar HTML stranice u zasebnom bloku - blok počinje HTML oznakom *<style type="text/css"*> i završava HTML oznakom *</style>*
- definiranje izgleda unutar HTML oznake - za ovu potrebu koristi se atribut style unutar kojega se specificira izgled samo za tu HTML oznaku

Format CSS zapisa je sljedeći:

selector [, selector2, ...][:pseudo-class] {

property: value;

[property2: value2;

...]

}

Kod dodavanja CSS koda za pojedinu HTML oznaku preskaču se opisi selektora, te se u jednom retku upisuje sve, po definiciji između vitičastih zagrada *({ })*:

<tag style="property: value;[property2: value2;...]" ...>

Primjer jednog CSS opisa:

```
img {
   margin:0;
   border:0;
}
```

Objašnjenje:

img selektor predstavlja element na koji dodajemo CSS atribute, gdje margin označava razmak između slike i sadržaja unutar elementa na kojem se slika nalazi, a border *debljinu obruba same slike.*

2.2 Element <canvas> i skriptiranje slikovnog sadržaja Javascriptom

Skriptiranje sadržaja dokumenta Javascript-om

JavaScript je skriptni programski jezik, koji se izvršava u pregledniku na strani korisnika. JavaScript je najpopularniji skriptni jezik na Internetu kojeg danas podržavaju svi poznatiji preglednici (Internet Explorer, Mozilla Firefox, Chrome, Opera). Cilj kreiranja JavaScript jezika je dodati interaktivnost HTML stranicama. Skriptni jezici su programski jezici manjih mogućnosti, koji se sastoje od ~~od~~ izvršnog računalnog koda, obično ugrađenog u HTML stranice. JavaScript je interpreter, što znači da se skripta izvršava odmah naredbu po naredbu, bez prethodnog prevođenja cijelog programa i kreiranja izvršne datoteke. JavaScript omogućava programiranje u okviru HTML stranica. Pretvaranje dinamičkog teksta u HTML stranicu se može putem raznih skriptnih naredbi. Čitanje i pisanje HTML elemenata kao i reagiranje na događaje su uključeni u JavaScriptu. To je moguće postaviti tako da se skirpta izvršava kada se dogodi neki događaj (npr. prilikom učitavanja stranica ili kada korisnik klikne na određenu poveznicu HTML elementa). Provjera ispravnosti i vjerodostojnosti podataka izvršava JavaScript slanjem podataka na server. JavaScript može detektirati preglednik kojeg korisnik upotrebljava tako da se učita stranica koja je posebno dizajnirana za taj preglednik. Osim toga JavaScript-om se mogu napraviti „kolačići“ (eng. cookies), gdje se pohranjuju i učitavaju informacije o korisnikovim podacima [9].

Da bi se izvršio JavaScript kod koji se pokreće na pretraživaču prilikom otvaranja dokumenta potrebno je sljedeće:

1. Pisanje koda – Izrada HTML i JavaScript koda koji će biti spremljeni zasebno u datotekama, npr. index.html za HTML i index.js za JavaScript ili unutar istog HTML dokumenta.
2. Učitavanje koda - Kako naiđe na JavaScript kod, Internet preglednik analizira i provjerava točnost, a zatim izvršava zadanu Javascript naredbu. Internet preglednik gradi interni model za HTML koji se naziva DOM.

3. Pokretanje koda - JavaScript nastavlja sa izvršavanjem naredbi koristeći DOM, te se primaju i dobivaju naredbe iz njega za izvršavanje ostalih datoteka sa servera.

Javascript skripte se programiraju u posebnom dijelu HTML dokumenta omeđenim `<script>` elementom. Pomoću JavaScripta možemo davati obavijesti, umetati nove elemente, izvršavati više stvari od jednom, donositi odluke, stvarati varijable.

Da bi koristili JavaScript potrebno ga je svrstati unutar web stranice. Najbolja lokacija za pozicionirati skriptu jest u samoj „glavi“ (eng. head) web stranice. Time čim otvaramo preglednik odmah učitavamo skriptu, a zatim ostatak sadržaja web stranice. Također, skriptu možemo povezati sa datotekom ako dodamo poveznicu te datoteke na kojoj se nalazi JavaScript kod ili ju možemo pozicionirati unutar „tijela“ (eng. body) elementa.

Za programiranje u JavaScriptu dovoljan je običan tekst editor poput Notepada i internet preglednika u kojemu se može provjeriti kako rade aplikacije. Sam kod JavaScript aplikacije se nalazi u HTML kodu stranice, pa, kako su HTML datoteke obične tekstualne datoteke, može se za unos JavaScript *apleta* (male skripte koja izvršava jednostavne programe) upotrijebiti i običan tekst editor. U ovom radu za skriptiranje i izradu HTML dokumenata korišten je program Adobe Dreamweaver namijenjen razvoju web sadržaja. Za provjeravanje funkcionalnosti aplikacija preglednik mora podržavati JavaScript. Zato se JavaScript program koristi u sljedećem obliku:

U HTML kod stranice treba unijeti element `<SCRIPT language="JavaScript">` koji će pregledniku pokazati da počinje JavaScript kod. Završetak JavaScript koda označavamo sa `</SCRIPT>` čime pregledniku pokazujemo da je kod JavaScript završen.

Kod programiranja u JavaScriptu koristi se objektni način razmišljanja, tj. postoje objekti i njihova svojstva i metode. Preciznije rečeno promjenom nekog svojstva objekta pozvat će se metoda pridružena tom svojstvu, npr. kad dođemo kursorom miša nad nekim aktivnim poljem na ekranu pozvat će se procedura definirana u svojstvu *onMouseOver* tog aktivnog polja na ekranu. Takav način definiranja svojstava proteže se i na same dokumente. Na svakoj JavaScript stranici postoji određen skup objekata sa

svojim karakteristikama koji će se procesuirati i donijeti događaj ovisno kakve oni parametre sadrže. To mogu biti: *anchor, button, checkbox, Date, document, elements array, form, frame, hidden, history, link, location, Math, navigator, password, radio, reset, select, string, submit, text, textarea, window*. Svaki od njih sadrži svoje parametre i metode pomoću kojih je moguće programski upravljati cijelom stranicom, na način sličan navedenom, koji je različit od proceduralnog načina razmišljanja, gdje se stvari odvijaju onim redom kojim je programer zamislio i onim redom kojim je to navedeno u kodu. Ovdje je korisnik i njegova akcija taj koji određuje što će se točno dogoditi.

Prije svega, važna je napomena da JavaScript, za razliku od ponekih programa, razlikuje verzalna i kurentna slova, tako da je važno točno upisati naredbe.

U ovom radu Javascript skriptni jezik omogućuje pristup svim slikovnim elementima kao što su visina, širina slike te podaci o boji i transparenciji pomoću kojih se kreiraju željeni filteri slike. Prednosti ovakvog direktnog manipuliranja slikom je to što se Javascriptom jedna slika može umnažati više puta unutar „canvas“ prostora tako da je prikaz samih manipuliranih slika višestruko brži nego kada korisnik mora lokalno „preuzeti“ pojedinačne slike. Javascriptom se nudi i mogućnost interakcije korisnika sa web sučeljem gdje je stvoren prostor za izradu brzih web aplikacija za manipulaciju slikom.

Element <canvas>

HTML5 specifikacija uključuje brojne nove značajke, od kojih je jedan `<canvas>` element. HTML5 canvas nudi jednostavan i moćan način stvaranja grafike koristeći JavaScript. Za razliku od nekih drugih tehnologija, za prikaz <canvas>-a nisu potrebne dodatne skripte, već HTML5 kompatibilan preglednik i običan tekst uređivač**.** „Canvas“ element se može opisati kao područje na kojem se može iscrtavati vektorska i piksel grafika [10]. To područje ima svoju visinu i duljinu (fiksna veličina), te koristi JavaScript kao medij koji crta i animira kompleksnu grafiku kao što su grafovi, kompozicije slika i izrada animacija. Stvaranje „canvas“ konteksta u HTML dokument:

```
<canvas id="canvas" width="300" height="300"></canvas>
```

Zadaje se ID elementa kako bi ga mogli pozvati u JavaScript kodu, te visina i duljina područja na kojem se grafika iscrtava. Sada imamo prazan „papir“ za kreiranje grafike.

Element `<canvas>` stvara fiksnu površinu za crtanje koja predstavlja jedan ili više „kontekst“ za ostvarivanje prikaza (eng. rendering contexts) u koji se izdaju JavaScript naredbe za crtanje i manipuliranje prikazanog sadržaja. Preglednici mogu implementirati više „canvas“ konteksta gdje se osigurava funkcionalnost crtanja.

Inicijalno <*canvas*> element je prazan. Kako bi prikazali nešto, skripta prvo mora pristupiti kontekstu i crtati na njemu. „Canvas“ element ima DOM (eng. Document Object Model) metodu *getContext* koja se koristi za dobivanje konteksta i funkcija za crtanje. *getContext()* prima jedan parametar – tip konteksta (u sljedećem primjeru je to 2D, čime se definira kreiranje dvodimenzionalne vektorske grafike).

```
var canvas = document.getElementById('canvas');

var ctx = canvas.getContext('2d');
```

U prvoj liniji se preuzima „canvas“ objekt koristeći *getElementById()* metodu. Nakon toga možemo pristupiti kontekstu za crtanje koristeći *getContext* metodu.

Element <*canvas*> koristi mrežu koordinatnog sustava web preglednika kako bi iscrtavao grafiku. Jedna jedinica u mreži odgovara jednom slikovnom elementu

(pikselu) na „canvasu“. Ishodište sustava pozicionirano je u gornjem lijevom kutu, koordinata (0,0). Svi elementi su pozicionirani relativno s obzirom na ishodište.

„Canvas“ podržava samo jedan primitivni oblik – pravokutnik. Svi ostali oblici se stvaraju kombiniranjem jedne ili više staza (eng. path). Postoji izbor različitih staza kao npr. kružni isječci, elipse, linije, poli-linije itd, koje omogućuju kompoziciju vrlo kompleksnih oblika [11].

Funkcija *beginPath* započinje novu stazu dok se *moveTo, lineTo, arcTo, arc* i slične metode koriste za dodavanje segmenata stazi. Segmenti se spajaju u listu pod-staza koje zajedno stvaraju oblik.

Kada je trenutna staza prazna i tek je pozvana funkcija *beginPath*, neovisno o kojoj je sljedećoj naredbi riječ, tretira se kao *moveTo()* – specificiranje početne pozicije.

Staza se zatvara koristeći *closePath*. Ova metoda zatvara oblik tako da crta ravnu liniju od trenutne pozicije do početne. Ako je oblik već zatvoren ili je samo jedna točka u listi, funkcija ne radi ništa.

Kada se oblik stvori, mogu se koristiti *fill* ili *stroke* za crtanje staze (oblik) na „canvas“, „stroke“ se koristi za crtanje obruba, dok „fill“ crta puni oblik. Kada se poziva „fill“ metoda, staza će se automatski zatvoriti pa nije potrebno koristiti *closePath* metodu.

Vrlo važna funkcija koja ne crta je *moveTo* funkcija. Ovu funkciju možemo zamisliti kao podizanje olovke s papira s jednog mjesta na drugo.

moveTo(x, y) – prima dva argumenta, x i y, koji su koordinate nove pozicije.

Kada se „canvas“ inicijalizira ili se pozove *beginPath* metoda, potrebno je upotrijebiti *moveTo* metodu za pomicanje početne točke crtanja na novu poziciju. *moveTo* metoda se također koristi za crtanje nepovezanih staza.

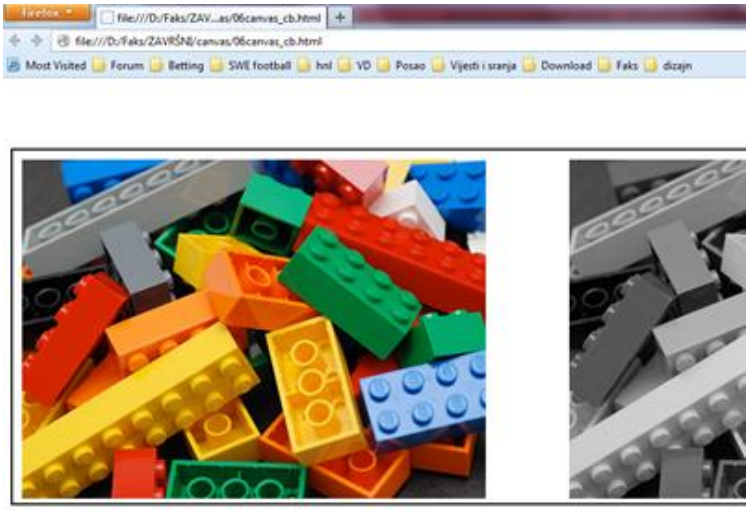
Unutar elementa <*canvas*> rasterska grafika se može dohvaćati iz postojećeg <*img*> elementa metodom *drawImage()*. Ova metoda traži tri, pet ili devet argumenata kojima određujemo poziciju, širinu, visinu i rez originalne grafike.

- *drawImage(image, dx, dy)* – prvi argument „image“ predstavlja izvor slike koju želimo prikazati. Koordinate (*dx*, *dy*) označavaju gornji lijevi kut pozicije dotične slike. Koordinate (0, 0) će stvoriti sliku u gornjem lijevom kutu *<canvas>* elementa.
- *drawImage(image, dx, dy, dw, dh)* – argument „image“ kao i u prethodnom slučaju predstavlja izvor slike koju želimo prikazati, povećamo/smanjimo do širine (*dw*) i visine (*dh*), i zatim iscrtava na „canvasu“ koordinatama (*dx, dy*).
- *drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)* prikazuje sliku, oblikuje ju na pravokutnik veličine (*sx, sy, sw, sh*), poveća/smanji na (*dw, dh*), i iscrtava na „canvas“ sa koordinatama (*dx, dy*).

Također, slika se kao objekt može kreirati isključivo pomoću Javascripta gdje koristimo objekt *Image()* sa zadanim izvorom slike:

```
var slika = new Image();                    // pozivanje naredbe za novu sliku
slika.src = "images/slika.png";             // povlačenje slike
slika.onload = function() {                 // pozivanje funkcije
context.drawImage(slika, 0, 0); ...  // iscrtavanje slike
```

Na slici 2. prikazano je iscrtavanje slike iz prethodnog programa. Dimenzije *canvasa* su su omeđene crnim obrubom kreiranim CSS-om, a unutar njega se pomoću JavaScripta učitava slika.



Slika 2. – Prikaz „canvasa“ sa preglednika, sa svojim crnim obrubom

3. EKSPERIMENTALNI DIO

3.1 Razvoj filtera za slike u HTML5 web okruženju

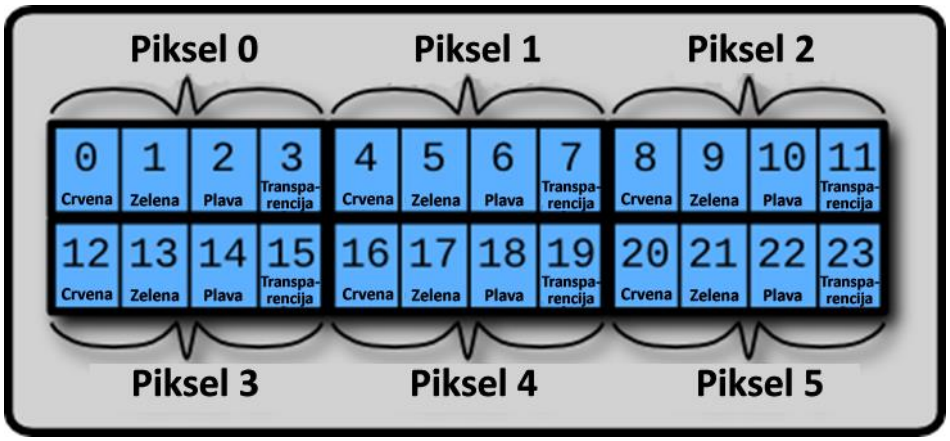
Struktura slike

Da bi prikazalo sliku, računalo je pretvara u određeni broj točaka. Svaka točka može biti točno određene boje koja se sastoji od tri osnovne komponente (crvena, zelena i plava) što je tzv. RGB paleta boja (engl. RedGreenBlue). Četvrta komponenta koja se može definirati označava transparenciju (eng. alpha) i označava se slovom A – RGBA. Slika se na zaslonu crta pomoću niza točkica (elementa slike) raspoređenih u kvadratnu mrežu, a te točkice nazivamo pikselima.

Da bi se uspješno odradila manipulacija pikselima dovoljno je učiniti sljedeće stvari:

- Otvoriti niz praznih piksela
- Dobiti niz piksela iz već postojećeg „canvasa“
- Postaviti piksele unutar „canvasa“ od dobivenih niza piksela

Nakon što se dobio niz piksela, njima je moguće manipulirati pomoću standardnih pristupnika (eng. Accessor). Pikseli u polju su raspoređeni u redovima, te poprimaju vrijednosti boja između 0 i 255, gdje svake četiri cijele grupe predstavljaju četiri kanala boja jednoga piksela: crvena, zelena, plava i alfa (RGBA). Da bi se dobio vizualni dojam manipulacije pikselima uzmimo npr. sliku koja je tri piksela široka i dva piksela visoka gdje svaki piksel ima svoju ulogu u prikazivanju slike. [12]



Slika 3. - Izgled grupe piksela veličine tri piksela u širini i dva u visini. Svaki piksel uzima četiri elemenata u svojoj grupi za crvenu, zelenu, plavu i alfa boju i imamo ukupno 24 elemenata ili od 0 do 23.

Dohvaćanje i manipulacija RGBA vrijednosti elemenata slike

Za dohvaćanje slikovnog podataka koristi se 2D funkcija *getImageData(x, y, width, height)*

Primjer:

```
var canvas  = document.getElementById("ex1");
var context = canvas.getContext("2d");
var imageData = context.getImageData(x, y, width, height);
```

X i Y koordinata predstavljaju početak čitanja vrijednosti piksela unutar <*canvas*> elementa. Širina i visina (*width i height*) ove funkcije postavlja širinu i visinu u pikselima od pikselnog područja koja je zastupljena pomoću stvorenoga *ImageData* objekta. Manipulirana slika se postavlja unutar „canvasa“ funkcijom putImageData() i ona ima tri svojstva: širinu, visinu i podatke o pikselima. Širina i visina sadrže širinu i visinu područja grafičkih podataka dok je podatak niz koji sadržava RGBA vrijednosti svakog piksela.

Svaki piksel sadrži vrijednost od četiri niza boja. Jedna vrijednost je za crvenu boju, druga za zelenu, treća za plavu boju i jedna vrijednost za alfa kanal. Boja piksela je

određena aditivnim mješanjem crvene, zelene i plave boje zajedno čineći konačnu boju. Alfa kanal predstavlja koliko je transparentan piksel i u brojevima ta vrijednost varira od 0 do 255.

Primjer koda koji određuje boju i alfa vrijednost prvog piksela kojem je određena crvena boja (prva vrijednost – R – u nizu od četiri je maksimalna. dok su druge dvije – R i G – nula):

```
var pixelIndex = 0;
imageData.data[pixelIndex    ] = 255;  // crvena boja
imageData.data[pixelIndex + 1] =   0;  // zelena boja
imageData.data[pixelIndex + 2] =   0;  // plava boja
imageData.data[pixelIndex + 3] = 255;  // transparencija
```

Za čitanje vrijednosti piksela, ispisujemo sljedeće:

```
var pixelIndex = 0;
var red   = imageData.data[pixelIndex    ];  // crvena boja
var green = imageData.data[pixelIndex + 1];  // zelena boja
var blue  = imageData.data[pixelIndex + 2];  // plava boja
var alpha = imageData.data[pixelIndex + 3];  // transparencija
```

Za pristupanje vrijednosti naknadnih piksela, povećava se za 4 vrijednost pikselovog indeksa (svaki piksel je stvoren od četiri niza elemenata kao što je već objašnjeno)

```
var index = 4 * (x + y * width);
```

U izračunu x i y jesu koordinate piksela da bi se izračunao njegov indeks. Pikseli u nizu podataka su organizirani kao jedan dugi niz piksela, od gornjeg lijevog piksela kreću se vertikalno do desno. Kada se dosegnuo završetak linije pikselov niz se nastavlja od piksela koji se nalazi najviše lijevo, a prvom linijom ispod prethodnog. Stoga, da bi se izračunao indeks piksela smještenog na x, y potrebno je pomnožiti y koordinatu sa brojem piksela po liniji i dodati vrijednost x za njega.

Finalan program koji postavlja sliku na canvas te dohvaća vrijednosti pojedinačnih piksela prikazan je sljedećim kodom:

```
window.onload = function(){
var canvas  = document.getElementById('mojCanvas');
var context = canvas.getContext('2d');

var imageObj = new Image();
imageObj.src = slika.jpg';

function postaviSliku(context, canvas) {
var imgData = context.getImageData(10, 10, 500, 366);
var pixels  = imgData.data;
for (var pixelIndex = 0, n = pixels.length; pixelIndex < n; pixelIndex
+= 4) {

red = pixels[pixelIndex];       // crvena
green =  pixels[pixelIndex +1]  // zelena
blue = pixels[pixelIndex +2];   // plava
alpha = pixels[pixelIndex + 3]  // transparencija
}

context.putImageData(imgData, 600, 10);
}
//add the function call in the imageObj.onload
imageObj.onload = function(){
context.drawImage(imageObj, 10, 10);
postaviSliku(context, canvas);
};
}
```

Metodom *putImageData()* stvoreno je novo područje na <*canvas*> objektu na kojem se iscrtava nova slika. Podaci o RGBA vrijednostima piksela inicijalne slike preuzeti metodom *getImageData()* koriste se kako bi se konstruirao novi set pikselskih vrijednosti na drugom području koordinatnog sustava unutar istog <*canvas*> elementa. Ovaj program je osnova na kojoj se grade daljnji algoritmi koji matematičkim putem manipuliraju uzete RGBA vrijednosti piksela. Stvaranje filtera za sliku jest manipulacija osnovnih RGBA varijabli svakog pojedinog piksela na način da se stvaraju nove

vrijednosti boja. Nove boje se zatim zamjenjuju originalnim vrijednostima ili se umeću na novo koordinatno područje <*canvas*> elementa.

HTML5 „canvas“ element se može koristiti za pisanje filtera slika. Ono što se treba učiniti je nacrtati sliku unutar „canvasa“, pročitati pikselnu vrijednost „canvasa“, i pokrenuti filter unutar njega.

3.2 Projektiranje jednostavnih filtera slike

Ovo poglavlje bavi se izradom baznih filtera koji u sebi sadrže samo jedan efekt, bilo da se taj efekt javlja u jednom kanalu slike ili u više njih. Efektom se ovdje naziva određena matematička manipulacija koja rezultira promijenjenom bojom originalnog piksela. Jednostavni filteri uključuju manipulaciju svjetlinom, naglašavanje pojedinih tonova, negativ efekt [13], crno bijeli efekt [14][15] i dodavanje „šuma“ (eng. noise). Iz njih se izvodi složeni filter u poglavlju 3.3.

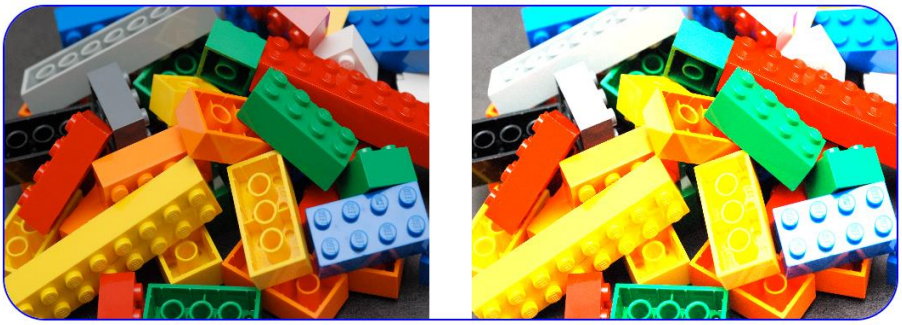
3.2.1 Filter za manipulaciju svjetline slike

Prvi testirani filter na uniforman način povećava ili smanjuje RGB vrijednosti piksela čineći tako sliku tamnijom ili svjetlijom. To znači da sve vrijednosti povećava za isti iznos. U osnovni program kreiranja slike umetnuta je matematička formula koja dijeli ili množi originalne vrijednosti boja slike sa jednakim brojem te na taj način postiže efekt svjetline. Varijable red, green i blue u primjeru na slici 4, prikazuju množenje preuzetih originalnih vrijednosti kanala R, G i B sa faktorom *f*, što rezultira jednoličnim smanjenjem ili povećanjem svjetline cjelokupne slike. Faktor *f* je varijabla koja promjenom vrijednosti mijenja svojstva slike. Faktor *f* koji se kreće u intervalu od 0 – 1 rezultirat će smanjenjem svjetline dok vrijednost *f* iznad 1 rezultira većom svjetlinom slike.

```
window.onload = function(){
var canvas  = document.getElementById('mojCanvas');
var context = canvas.getContext('2d');
var imageObj = new Image();
imageObj.src = 'slika.jpg';
function postaviSliku(context, canvas) {
var imgData = context.getImageData(0, 0, 500, 366);
var pixels  = imgData.data;
var f = 0.5;
for (var i = 0, n = pixels.length; i < n; i += 4) {
Red = pixels[i]* f;     // množenjem faktorom f
Green = pixels[i+1]*f;  // slika se osvjetljuje
```

```
Blue =  pixels[i+2]*f;  // ili tamni
pixels[i]=Red;
pixels[i+1]=Green;
pixels[i+2]=Blue;
}
context.putImageData(imgData, 550, 0);
}
imageObj.onload = function(){
context.drawImage(imageObj, 0, 0);
postaviSliku(context, canvas);};}
```

a)



b)

c)

d)

Slika 4. – Lijevo: originalna slika, desno: krajnja promjena svjetline uniformno za sve kanale u ovisnosti o faktoru f: a) f=2, b) f=1.5, c) f= 0.6, d) f=0.3

3.2.2 Filter za naglašavanje pojedinih boja slike

Filter za naglašavanje pojedinih boja na slici dobivamo zahvaljujući matematičkoj formuli koju koristimo kao u prethodnom primjeru gdje dijeljenjem odnosno množenjem možemo naglasiti pojedinu boju, bilo to crvena, plava, zelena ili neka druga. U ovom slučaju se vrijednosti svakog kanala boje množe ili dijele različitim faktorom. Množenjem crvenog kanala i dijeljenjem zelene i plave rezultira naglašavanje crvene boje. Tako vrijedi i za ostale kanale, gdje matematičkim operacijama utječemo na rezultat slike i naglašavanje određene boje. Na slici 5. je vidljivo različito naglašavanje pojedinih boja. Naglašavanje boja kao što su cijan, magenta i žuta dobiva se naglašavanjem dvaju kanala, a reduciranjem jednog. Ako želimo naglasiti magenta boju slike naglasit ćemo crveni i plavi kanal množenjem nekim postotkom, a reducirati zeleni kanal dijeleći određenim postotkom. Isti princip vrijedi i za naglašavanje cijana i žute. U ovisnosti o postotku povećanja ili smanjenja vrijednosti kanala možemo napraviti naglašavanje bilo koje nijanse spektra.

Program za naglašavanje plavog kanala:

```
…
function postaviSliku(context, canvas) {
var imgData = context.getImageData(10, 10, 500, 366);
var pixels  = imgData.data;
```

```
for (var i = 0, n = pixels.length; i < n; i += 4) {
Red = pixels[i]/1.2;
Green = pixels[i+1]/1.2;
Blue =  pixels[i+2]*1.5;

pixels[i]= Red;
pixels[i+1]= Green;
pixels[i+2]= Blue;
}
context.putImageData(imgData, 600, 10);
}
imageObj.onload = function(){
context.drawImage(imageObj, 10, 10);
postaviSliku(context, canvas);};
}
```

a)

b)

c)

d)

e)

f)

Slika 5. a) lijevo: originalna slika, desno: naglašen crveni kanal, b)naglašen zeleni kanal, c)naglašen plavi kanal, d) naglašen cijan, kombinacijom zelenog i plavog kanala, e) naglašenamagenta, kombinacijom crvenog i plavog kanala, f) naglašena žuta, kombinacijom crvenog i zelenog kanala

3.2.3 Filter za sliku u negativu

Kod filtera za pretvaranje slike u negativu koristimo funkciju oduzimanja vrijednosti boja i to tako da vrijednost koja je najveća moguća (255) oduzmemo za postojećom prisutne boje na slici matematičkom operacijom oduzimanja i to za svaku boju: crvenu, zelenu i plavu. Rezultat programa daje inverzne, komplementarne boje što je vidljivo na slici ispod samoga programa. Za smanjivanje utjecaja negativa, potrebno je promjeniti i vrijednost koja se oduzima sa postojećom vrijednosti boja i to mora biti niže od vrijednosti 255. Slika 6.a) Prikazuje originalnu sliku i rezultat programa koji daje negativ efekt zahvaljujući oduzimanju 255 od postojeće vrijednosti boja na original slici pomoću funkcije i varijable (negRed = 255 - pixels[i];) za crvenu boju, (negGreen = 255 - pixels[i+1];) za zelenu i (negBlue = 255 - pixels[i+2];) za plavu boju. Da bismo dobili „reducirani negativ“ kakav prikazuje slika 6.b) koristila se apsolutna vrijednost razlike 50% sive (vrijednost 128) i originalne vrijednosti. Formula prevedena u Javascriptu za svaki kanal glasi:

`nRed = Math.abs(128 – pixels[i]);` za crveni kanal

`nGreen = Math.abs(128 – pixels[i+1]);` za zeleni kanal i

`nBlue = Math.abs(128 – pixels[i+2]);` za plavi kanal.

Program za ispis slike u negativu:

```
…

var imageObj = new Image();
imageObj.src = 'slika.jpg';

function postaviSliku(context, canvas) {
var imgData = context.getImageData(10, 10, 500, 366);
var pixels  = imgData.data;

for (var i = 0, n = pixels.length; i < n; i += 4) {
negRed = 255 - pixels[i];      // na ovom mjestu se vrši
negGreen = 255 - pixels[i+1];  // oduzimanje i kao razultat
negBlue = 255 - pixels[i+2];   // dobivamo sliku u negativu

pixels[i]= negRed;
pixels[i+1]= negGreen;
pixels[i+2]= negBlue;
}

context.putImageData(imgData, 600, 10);
```

```
}
imageObj.onload = function(){
context.drawImage(imageObj, 10, 10);
postaviSliku (context, canvas);
};
}
```

a)

b)

Slika 6. – a) lijevo: originalna slika, desno: slika u negativu, b) lijevo: originalna slika, desno: reducirani negativ – oduzimanje originalnih vrijednosti od vrijednosti manje od 255

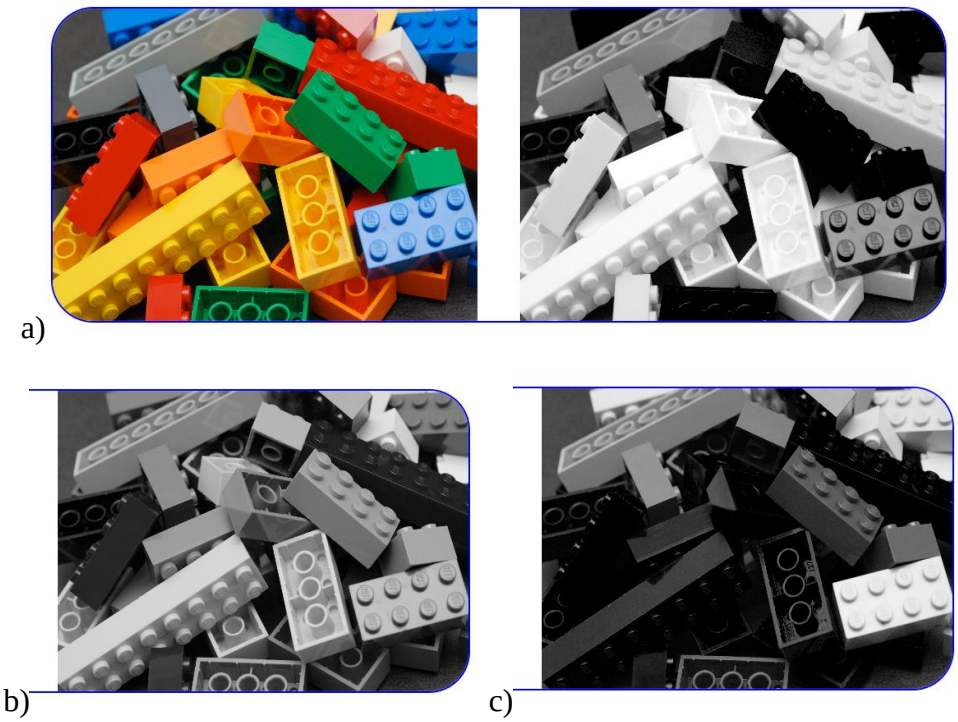
3.2.4 Filter crno-bijele slike

Da bi se stvorila crno bijela slika sva tri RGB kanala moraju imati jednaku vrijednost. Može se definirati crno bijela slika od vrijednosti jednog kanala: npr. sva tri kanala poprimaju vrijednost crvenog kanala (ili zelenog ili plavog) kao što je prikazano na slici 7. Na taj način možemo filtrirati pojedini kanal i prikazati njegovu svjetlinu. Unutar samoga programa možemo utjecati i filtrirati određeni kanal boje koje želimo, npr. pomoću funkcije (var grayscale = pixels[i];) utječemo na filtriranje crvenoga kanala, a za zelenu odnosno plavu potrebno je izmjeniti vrijednost unutar uglate zagrade na [i+1] odnosno [i+2].

Program crno bijele slike izvedene iz crvenog kanala:

…

```
var imageObj = new Image();
imageObj.src = 'slika.jpg';

function crnoBijela(context, canvas) {
var imgData = context.getImageData(10, 10, canvas.width,
canvas.height);
var pixels  = imgData.data;
for (var i = 0, n = pixels.length; i < n; i += 4) {
var grayscale = pixels[i];      // filtrira se samo crveni kanal
pixels[i  ] = grayscale;        // crvena
pixels[i+1] = grayscale;        // zelena
pixels[i+2] = grayscale;        // plava
}
context.putImageData(imgData, 600, 10);
}
imageObj.onload = function(){
context.drawImage(imageObj, 10, 10);
crnoBijela(context, canvas);
};
}
```



Slika 7. – prikaz efekta crno-bijelo i to filtriranjem određenoga kanala boja, a) lijevo: originalna slika, desno: crno bijeli efekt od izdvojenog crvenog kanala, b) filtriranje zelenog kanala, c) filtriranje plavog kanala

Crno bijeli filter može koristiti i druge matematičke formule koje uzimaju različite udjele RGB kanala kako bi se stvorio realniji dojam svjetline pojedinih boja. Postoji nekoliko standardnih metoda kao npr.:

- izračunavanje srednje vrijednosti (slika 8.a)– (R+G+B)/3

```
grayscale = (pixels[i]+pixels[i+1]+pixels[i+2])/3;
```

- formula koju koriste programi za procesiranje slika, kao npr. Adobe Photoshop (slika8.b) - R * 0.3 + G * 0.59 + B * 0.11

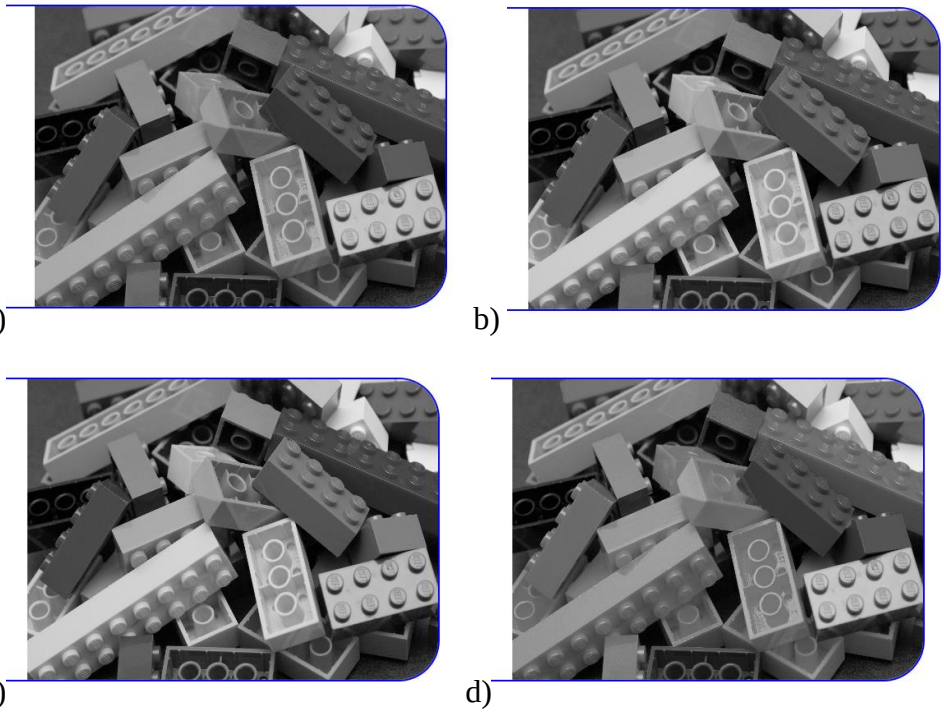
```
grayscale = pixels[i]*0.3+pixels[i+1]*0.59+pixels[i+2]*0.11;
```

- Luma formula (slika 8.c) - R * 0.2126 + G * 0.7152 + B * 0.0722

```
grayscale = 0.2126*pixels[i]+0.7152* pixels[i+1]+0.0722*
pixels[i+2]
```

- desaturacija (slika 8.d) – [Max(R, G, B) + Min(R, G, B)] / 2

```
grayscale = (Math.max(pixels[i], pixels[i+1],
pixels[i+2])+Math.min(pixels[i], pixels[i+1], pixels[i+2]))/2
```



Slika 8 – različiti crno bijeli efekti u ovisnosti o formuli: a) srednja vrijednost, b) Adobe Photoshop, c) Luma, d) desaturacija

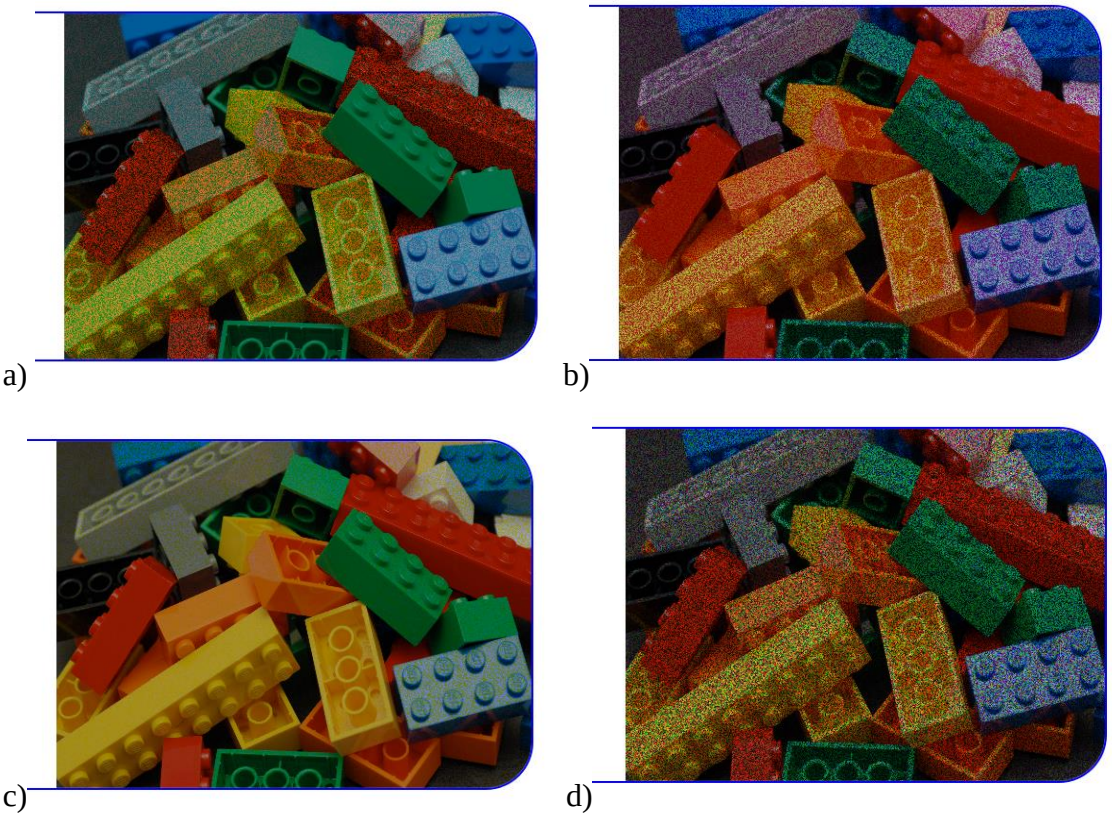
3.2.5 Filter za dodavanje „šuma“ na slici

Kod filtera za dodavanja „šuma“ (eng. noise) potrebno je dodati funkciju koja umeće slučajne vrijednosti piksela u željeni kanal. Da bi postigli takav efekt u Javascriptu množimo željeni kanal boje sa funkcijom *Math.random(),* dok je za ostale boje potrebno unijeti matematičku operaciju dijeljenja i to sa vrijednosti 1.5 da bi se intenzitet boje smanjio. Tako je za zelenu boju potrebno unijeti funkciju Math.radnom() na poziciju gdje se manipulira zelenom bojom što je u ovom primjeru na poziciji pixels[i+1], a na ostalim mjestima se dodaje matematička operacija dijeljenja sa 1.5 i time se dobiva naglašeni „šum“ za zeleni kanal boje. Isto pravilo vrijedi i za crveni i plavi kanal boje. Naglašeni „šum“ za sva tri kanala su prikazani na slici 9. ispod programa u kojem je prikazan primjer za naglašavanje „šuma“ kod crvenoga kanala.

Program za naglašavanje šuma u crvenom kanalu:

…

```
var imageObj = new Image();
imageObj.src = 'slika.jpg';

function postaviSliku(context, canvas) {
var imgData = context.getImageData(10, 10, 500, 366);
var pixels  = imgData.data;
for (var i = 0, n = pixels.length; i < n; i += 4) {
Red = pixels[i]* Math.random();  //funkcija šuma je dodana za crvenu
boju
Green = pixels[i+1]/1.5;
Blue =  pixels[i+2]/1.5;

pixels[i]=Red;
pixels[i+1]=Green;
pixels[i+2]=Blue;
}
context.putImageData(imgData, 600, 10);
}
imageObj.onload = function(){
context.drawImage(imageObj, 10, 10);
postaviSliku(context, canvas);
};
}
```



Slika 9. – Prikazuje rezultat naglašavanja „šuma“ za određeni kanal boje putem funkcije Math.random() i dijeljenjem ostale dvije, a) šum u crvenom kanalu, b) šum u zelenom kanalu, c) šum u plavom kanalu, d) Math.random() funkcija u svim kanalima

3.3 Stvaranje složenog filtera slike

U primjeru je stvoren složeni filter slike, u kojem je upotrijebljeno više funkcija i efekata u jednom. Tako imamo gradaciju od jedne nijanse boje prema drugoj po visini slike zahvaljujući funkciji i matematičkim operacijama, konkretno u crvenom kanalu u kojem koristimo formulu *(i/n)*visinaSlike* da bi dobili ravnomjernu gradaciju od maksimalne crvene prema 0. U formuli varijabla *i* predstavlja indeksni broj piksela, a varijabla *n* ukupan broj piksela u slici. Na slici 10. je prikazan filter u cijelosti te sa efektima u pojedinačnim kanalima. U zelenom kanalu (slika 10b) uvodi se uvjetovanje metodom *if/else* pomoću koje se mogu dodatno definirati koje vrijednosti boja će ući u izračun formula a koji će biti izostavljeni. Ovdje su vrijednosti zelene ograničene samo na 0 ili 255, tako da sve tamnije nijanse ispod 150 postaju 0, a sve iznad 150 postaje maksimum – 255. Zeleni kanal je dodatno i postavljen u negativ koji je stvoren zahvaljujući oduzimanjem na način da maksimalna vrijednost 255 oduzima postojeću vrijednost zelenog kanala. Kod kanala plave boje (slika 10c) imamo naglašeni „šum“ određene vrijednosti što je rezultat množenja pikselne vrijednosti plave boje sa funkcijom *Math.random()* i koeficijentom 1.5 što kao rezultat daje naglašeni „šum“ za plavi kanal boje. Ovdje je također vrijednost plave obrnuta u negativ. Osim navedenih efekata utjecalo se i na transparenciju što je učinjeno varijablom *v* koja je množila vrijednosti crvenog i plavog kanala boje i sadrži svoje uvjete. Varijabla *v* ovisna je o vrijednostima crvenog i plavog kanala prema formuli *R+B/3* te joj je *if/else* metodom postavljen maksimum opaciteta od 255.

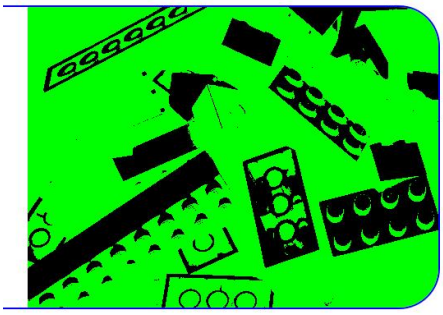
Program:

```
window.onload = function() {
var canvas = document.getElementById('mojCanvas');
var context = canvas.getContext('2d');
canvas.style.border = "2px solid black";

var imageObj = new Image();
imageObj.src = 'slika.jpg';
var width = 500;
var height = 366;
function mojFilter(context, canvas) {
var imgData = context.getImageData(10, 10, width, height);
var pixels = imgData.data;
```

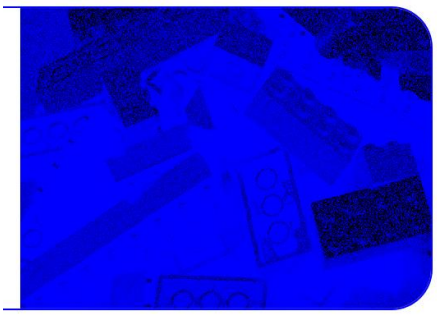
```
for (var i = 0, n = pixels.length; i < n; i += 4) {
Red =  (i / n) * 366;
// i/n - broj između 0 i 1. množi se sa visinom slike da bi dobili
ravnomjeran gradijent. ako množimo s manjim ili većim brojem od
"height" linija gradijenta mijenja horizontalnu poziciju. Taj broj
možemo oduzeti od 255 kako bi dobili inverzni gradijent

Green = pixels[i + 1];
if (Green >= 150) {Green = 255;} else {Green = 0;}
// zeleni kanal postize samo 2 vrijednosti - maksimalnu 255 i
minimalnu 0. Očitavaju se zelene vrijednosti svakog piksela, if else
metoda postavlja uvijet: za sve vrijednosti iznad 150, zelena poprima
maksimalnu vrijednost, a za manje od 150 poprima vrijednost 0

Blue = pixels[i + 2] * Math.random()*1.5;
// vrijednosti plavog kanala množe se sa funkcijom Math.random() koja
generira slučajnu vrijednost od 0 do 1. Sve zajedno se množi
koeficijentom koji određuje jačinu zrnatosti kanala

var v = pixels[i] * pixels[i + 2] /3;
// uvodi se i varijabla v koja množi originalne vrijednosti crvenog i
plavog kanala

if (v > 255) {v = 255}
// ukoliko vrijednost množenja plave i crvene premašuje maksimalnu
vrijednost 255 - varijabla v se postavlja na maksimalnu 255 vrijednost

pixels[i] =  Red;
pixels[i + 1] =  255 - Green;
//postavlja se inverzna vrijednost da bi se dobio obrnuti efekt slike

pixels[i + 2] = 255 - Blue;
pixels[i + 3] = v;
// transparencija koja ovisi o varijabli v - dakle o crvenom i plavom
kanalu

}
context.putImageData(imgData, 600, 10);
}
imageObj.onload = function() {
context.drawImage(imageObj, 10, 10);
mojFilter(context, canvas);
};
}
```

a)

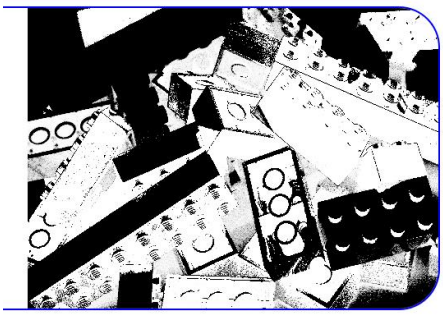
b)



c)



d)



e)

Slika 10. – Prikaz rezultata složenog filtera. a) lijevo: originalna slika, desno: rezultat filtera, b) efekt u crvenom kanalu, c) efekt u zelenom kanalu, d) efekt u plavom kanalu e) kanal alfa - transparencija

Na slici 10.a) odnosno rezultatu programa je vidljiv utjecaj svih funkcija koje se nalaze unutar programa, od gradijenta koji se proteže po visini slike, negativ efekt, naglašavanje „šuma“ za plavi kanal boje kao i transparencija koja je ovisna o varijabli v.

4. ZAKLJUČAK

Gledajući kroz povijest HTML tehnologija je u vrlo kratkom vremenu ispisala velike uspjehe i olakšala današnji život koji bi bio vrlo teško zamisliv bez toga. Zahvaljujući HTMLu, JavaScriptu, CSSu i sami koncept, dizajn, funkcionalnost, interakcija web stranice se izmjenila i poboljšala na gotovo svim mogućim segmentima. To se prvenstveno odnosi na kvalitetu slika i njezinih efekata koje je moguće dobiti zahvaljujući stilizaciji putem CSSa, a isto tako i programiranjem u JavaScriptu što je prikazano različitim primjerima i u samome završnome radu. Sami tekst i struktura HTMLa je dobila drugu dimenziju najnovijom tehnologijom u kojoj je moguće doslovce iscrtavati različite slike, grafove, dijagrame sa unosom različitih funkcija, varijabli, elemenata i sl., čemu je najviše pridonio element canvas. Kako bude vrijeme odmicalo tako će se vjerujem raditi i na još jednostavnijem i boljem pristupu elementu canvasu i ostalim funkcijama zahvaljujući kojima je danas olakšan rad sa HTMLom i čini ga iz dana u dan još jednostavnijim alatom za izradu web sadržaja, kao što je to slučaj za canvas koji olakšava manipuliranje slikama. Danas je iz dana u dan sve više web programera, dizajnera te samim time i sve više povratnih informacija i preporuka koji mogu pridonijeti jednostavnijem i boljem radu na HTMLu, JavaScriptu i CSSu. U svakom slučaju svjetla budućnost za ovu vrstu tehnologija je neupitna u skorijoj budućnosti i za očekivati je još naprednije oblike i veću dostupnost kao i njihovo korištenje od strane programera i web dizajnera.

Prilikom izrade završnoga rada izvršena su testiranja na tri različita preglednika: Mozilla Firefox, Internet Explorer i Google Chrome. Kod Firefox-a i Internet Explorer-a nisu bili nikakvi problemi sa prikazivanjem slika sa svojim filtrom, dok je kod Google Chrome-a bilo problema gdje je originalna slika bila vidljiva, ali slika sa svojim umetnutim filterom nije.

5. LITERATURA

1. Henick, B. (2010) *HTML & CSS: The Good Parts,* O'Reilly Media
2. ***http://www.w3.org/History/19921103_hypertext/hypertext/WWW/MarkUp/Tags.html
3. ***http://caniuse.com/#compare=ie+11,firefox+34,chrome+39,safari+8,opera+24,ios_saf+8,op_mini+5.0-7.0,android+4.4.3,op_mob+22,bb+10,and_chr+35,and_ff+30,ie_mob+10&compare_cats=HTML5
4. ***http://ptgmedia.pearsoncmg.com/images/9780137001316/samplechapter/JavaScriptFP_10_DHTMLObjColl.pdf
5. McLaughlin B., (2011). *What is HTML5,* O'Reilly Media
6. Pilgrim M., (2011). *Dive into HTML5,* http://diveintohtml5.info, 14. Kolovoz 2013.
7. Freeman E., Robson E. (2011) *Head first HTML5 Programming,* O'Reilly
8. Sawyer McFarland D., (2009). *CSS: The Missing Manual, Second Edition,* O'Reilly Media
9. Flanagan, D. (2006) *Javascript: The Definitive Guide, 5th ed.* O'Reilly Media
10. Fulton S., Fulton J., (2011). *HTML5 Canvas,* O'Reilly Media
11. Hall B., (2010). *HTML5's canvas Part II: Pixel Manipulation,* http://beej.us/blog/data/html5s-canvas-2-pixel/, 14. Kolovoz 2013.
12. Jenkov J., (2011). *HTML5 Canvas: Pixel Manipulation,* http://tutorials.jenkov.com/html5-canvas/pixels.html, 15. Kolovoz 2013.
13. Gravelle R., (2013). *Create Negative, Blur, and Rotate Image Effects Using the HTML5 Canvas,* http://www.htmlgoodies.com/html5/other/create-negative-blur-and-rotate-image-effects-using-the-html5-canvas.html#fbid=vS66anaxdI-, 18. Kolovoz 2013.
14. Lisci M., Scarlata L., (2011). *HTML5 Canvas Image Effects: Black & White,* http://spyrestudios.com/html5-canvas-image-effects-black-white/, 16. Kolovoz 2013

15. Kanan C, Cottrell GW (2012). *Color-to-Grayscale: Does the Method Matter in Image Recognition?*
http://tdlc.ucsd.edu/SV2013/Kanan_Cottrell_PLOS_Color_2012.pdf