Mogućnosti HTML-a 5 pri izradi modela i simulacija

Sambolek, Petar

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Stručni studij

Mogućnosti HTML-a 5 pri izradi modela i simulacija

Završni rad

Petar Sambolek

Osijek, 2016.

SADRŽAJ

1. UVOD

HTML[1], odnosno prezentacijski jezik za izradu web stranica, u početku je bio revolucionaran. Izrada neke stranice koja se mogla postaviti na Internet kako bi svi vidjeli Vaš rad je bila u najmanju ruku čudo. Evolucijom računala, odnosno hardvera, evoluira i softver.

Tako je krajem 1999. godine predstavljen HTML 4.01, pročišćena inačica HTML4 standarda, koja je donijela mnogo novosti, no još su uvijek nedostajale mogućnosti za implementaciju multimedijalnog sadržaja. U osnovi, to je i dalje bio tekstualni dokument. Iako se u to vrijeme pojavilo mnoštvo dodataka (*Eng. plugin*), jedan se isticao u implementaciji videa, zvuka, te raznih animacija – Flash player. Proizvod je to tvrtke Macromedia koji omogućava prikaz muldimedijskih datoteka stvorenih drugim Macromedijinim alatom - Flashom. Flash je u to vrijeme izazvao svojevrsni „boom“, te postao okosnica modernijih web stranica sa raznim efektima i korisnim funkcijama. Za primjer se može uzeti YouTube, kojeg ne bi bilo da Flash nije postojao.

No, s vremenom su nedostaci Flasha isplivali na vidjelo – kako su se na tržištu počeli pojavljivati pametni telefoni, primjetno je bilo da Flash nije optimiziran za izvođenje na slabijim sustavima.

Uz to, Flash datoteke bile su znatno veće veličine, što je usporavalo učitavanje i pregled web odredišta.

Sve je to potaknulo W3C[2] da razmisli o budućnosti takvog alata. Tako su došli na ideju implementacije multimedijalnih mogućnosti i animacija kao nativnih elemenata novog standarda za izradu web stranica – HTML5.

HTML5, iako još nije službeno predstavljen kao standard, opće je prihvaćen jer sadrži sve potrebne elemente za izradu interaktivnih stranica bez dodataka potrebnih za rad – sve što je potrebno je kompatibilan preglednik. HTML5 trebao bi raditi na pametnim telefonima, tabletima, te računalima na isti način. Prije je bilo nezamislivo igrati igru ili gledati animiranu stranicu na pametnom telefonu, no sada to postaje stvarnost.

U ovom radu predstavljene su neke od mogućnosti HTMLa 5 koje pomažu pri izradi modela i simulacija. Iako HTML5 službeno podržava samo 2D crtanje[1, str. 7], korištenjem biblioteke

[1] *Hypertext Markup Language*

[2] *World Wide Web Consortium*, konzorcij koji razvija web standarde

koja se oslanja na WebGL API – koji koristi većina modernih preglednika, može se uvelike olakšati crtanje 3D modela.

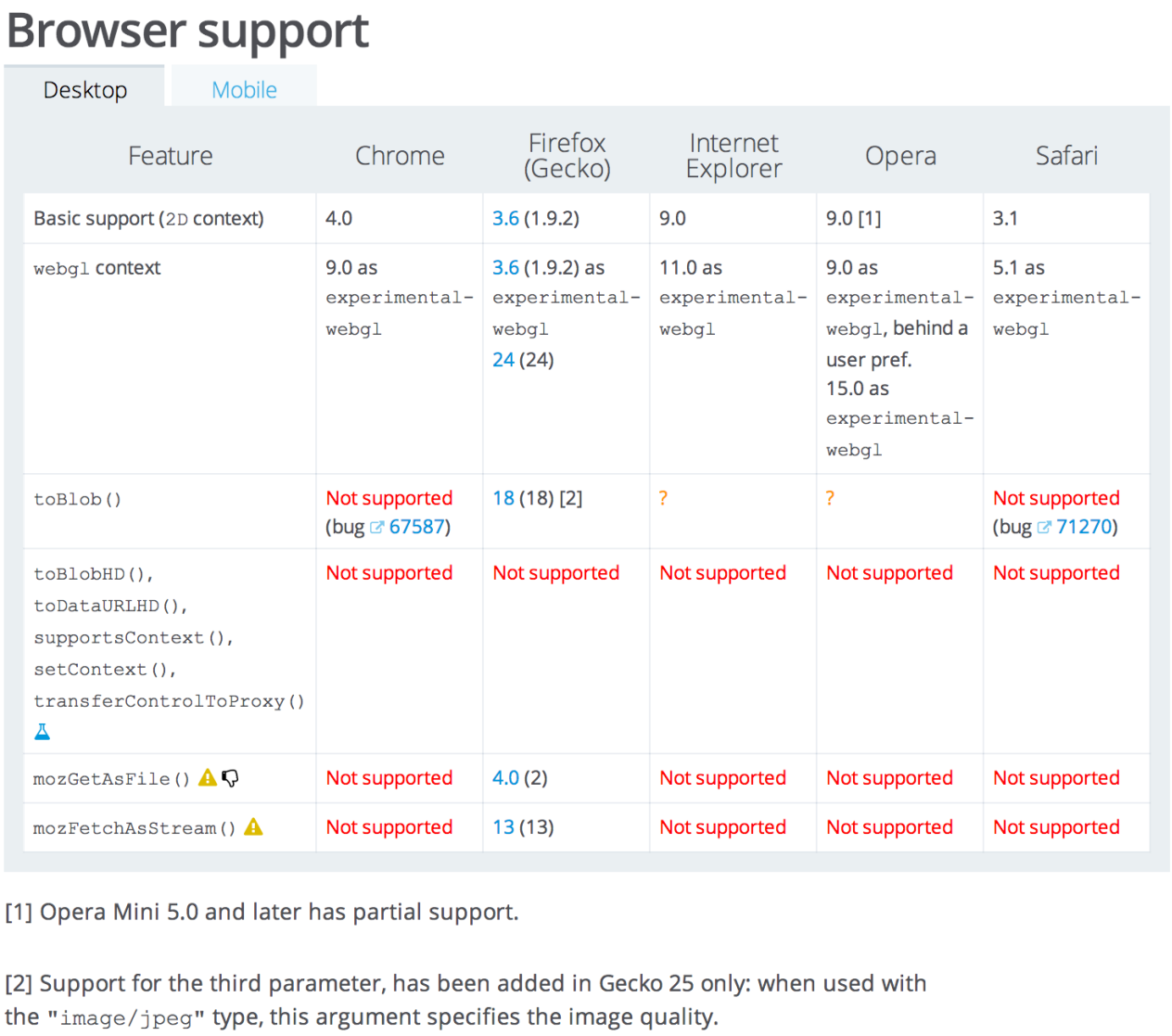
1.1. Zadatak završnog rada

HTML5 novi je standard za izradu web stranica. Ova peta revizija HTML standarda namijenjena je uvođenju novih tehnologija za podršku multimediji. Cilj ovoga rada je prikazati novosti u petoj verziji HTML standarda i na jednom primjeru, tehničkog modela, njegovu mogućnosti pri izradi simulacije.

2. PODRŠKA U INTERNET PREGLEDNICIMA

2.1. Podrška u Internet preglednicima na računalima

Element koji najviše utječe na primjenu HTMLa 5 pri izradi grafika je podrška preglednika za određene elemente standarda. Dok je 2D crtanje podržano čak i u starijim inačicama internet preglednika, podrška za 3D (webgl) kontekst, odnosno crtanje WebGL APIjem trenutno postoji samo za novije inačice preglednika, što je vidljivo iz slike 2.1.



Browser support

Desktop | Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support (2D context)	4.0	3.6 (1.9.2)	9.0	9.0 [1]	3.1
webgl context	9.0 as experimental-webgl	3.6 (1.9.2) as experimental-webgl 24 (24)	11.0 as experimental-webgl	9.0 as experimental-webgl, behind a user pref. 15.0 as experimental-webgl	5.1 as experimental-webgl
toBlob()	Not supported (bug 67587)	18 (18) [2]	?	?	Not supported (bug 71270)
toBlobHD(), toDataURLHD(), supportsContext(), setContext(), transferControlToProxy()	Not supported	Not supported	Not supported	Not supported	Not supported
mozGetAsFile()	Not supported	4.0 (2)	Not supported	Not supported	Not supported
mozFetchAsStream()	Not supported	13 (13)	Not supported	Not supported	Not supported

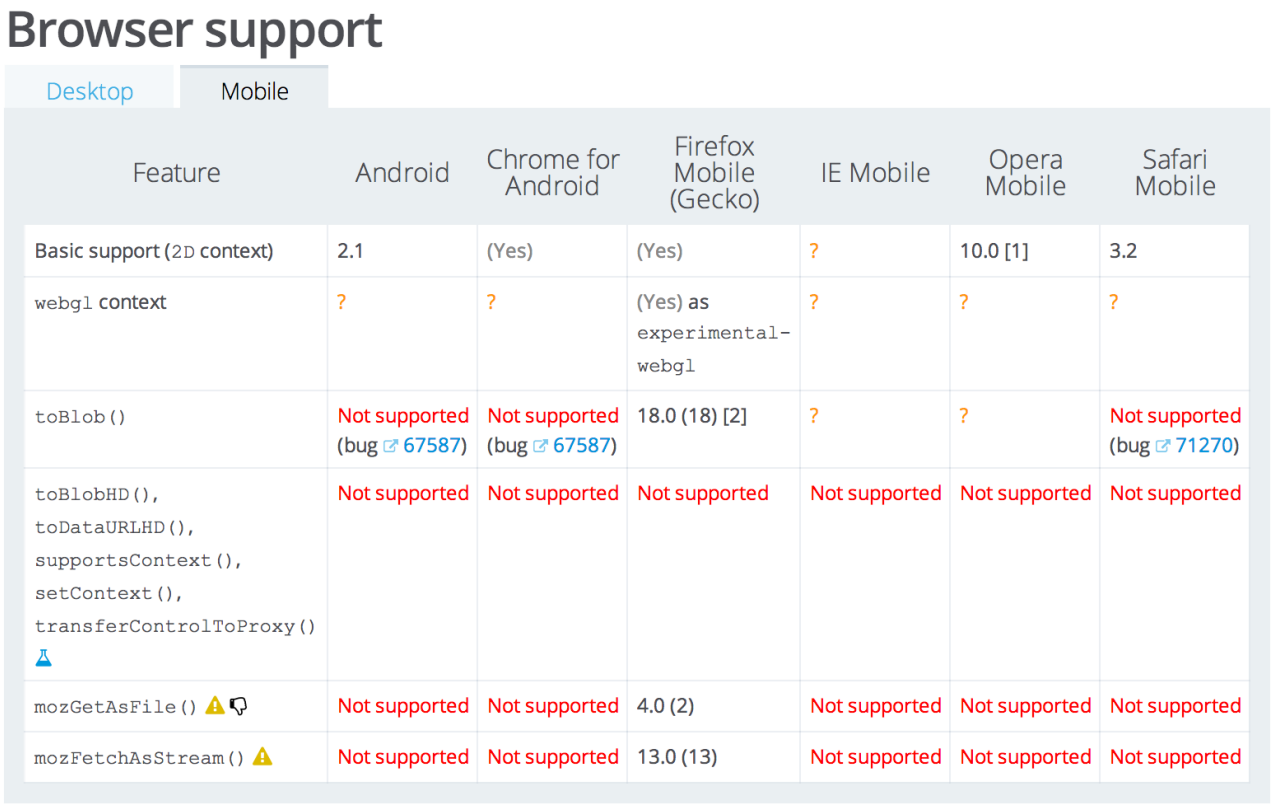
[1] Opera Mini 5.0 and later has partial support.

[2] Support for the third parameter, has been added in Gecko 25 only: when used with the "image/jpeg" type, this argument specifies the image quality.

Slika 2.1. Podrška WebGL APIja u različitim preglednicima na računalima [2]

2.2. Podrška u Internet preglednicima na mobilnim telefonima

Istovjetno kao za računala, postoji i tablica podrške WebGL konteksta za preglednike na mobilnim telefonima.



Browser support

Desktop | Mobile

Feature	Android	Chrome for Android	Firefox Mobile (Gecko)	IE Mobile	Opera Mobile	Safari Mobile
Basic support (2D context)	2.1	(Yes)	(Yes)	?	10.0 [1]	3.2
webgl context	?	?	(Yes) as experimental-webgl	?	?	?
toBlob()	Not supported (bug 67587)	Not supported (bug 67587)	18.0 (18) [2]	?	?	Not supported (bug 71270)
toBlobHD(), toDataURLHD(), supportsContext(), setContext(), transferControlToProxy()	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
mozGetAsFile()	Not supported	Not supported	4.0 (2)	Not supported	Not supported	Not supported
mozFetchAsStream()	Not supported	Not supported	13.0 (13)	Not supported	Not supported	Not supported

[1] Opera Mini 5.0 and later has partial support.

[2] Support for the third parameter, has been added in Gecko 25 only: when used with the "image/jpeg" type, this argument specifies the image quality.

Slika 2.2. Podrška WebGL APIja u različitim preglednicima na mobilnim telefonima

Iako je u početku HTML5 canvas element, odnosno 2D i 3D crtanje, zamišljeno tako da radi na velikoj većini uređaja, trenutno je samo 2D crtanje podržano na većini mobilnih uređaja, izuzev podrške za 3D crtanje. Kao što je vidljivo na slici 2.2., ona trenutno postoji isključivo u Firefox pregledniku.

3. CRTANJE OSNOVNIH OBLIKA U HTMLU 5

3.1. Inicijalizacija HTML5 canvas elementa i crtanje linije

Kako bi se po HTML dokumentu moglo crtati, prvo se mora inicijalizirati *canvas* element koji će služiti kao svojevrsno polje za crtanje. U njemu je definiran identifikator, širina od, primjerice, 500 točaka (*Eng. pixel*), te visina od 200 točaka, po propisanom HTML standardu za definiciju elemenata. Struktura HTML dokumenta u kojem se inicijalizira HTML5 stranica i *canvas* element nalazi se na slici 3.1.

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4  </head>
5  <body>
6      <canvas id="platno" width="500" height="200">
7          Vaš preglednik ne podržava canvas element!
8      </canvas>
9  </body>
10 </html>
11
```

Slika 3.1. Inicijalizacija HTML5 dokumenta, te *canvas* elementa

Kako bi se moglo crtati po *canvas* elementu, na kraj dokumenta dodan je *script* element u kojem je napisan JavaScript[3] kod pomoću kojeg je zapravo sve nacrtano [1, str. 3].

```
12 <script>
13  var canvas = document.getElementById('platno');
14  var context = canvas.getContext('2d');
15
16   context.beginPath();
17   context.moveTo(10, 20);
18   context.lineTo(20, 90);
19   context.lineWidth = 1;
20   context.stroke();
21 </script>
```

Slika 3.2. Inicijalizacija *script* elementa kojim crtamo po *canvas* elementu

Na linijama 13 do 21 slike 3.2. definirani su:

- identifikator *canvas* elementa po kojem se crta
- kontekst crtanja

[3] Računalni programski jezik razvijen 1995. godine od strane Brendana Eicha [3]

- početak putanje koja se crta
- točka od koje se počinje crtati, definirana kao koordinate u *canvas* elementu
- točka na kojoj se završava crtanje, također definirana kao koordinate
- širina linije koja se crta

Na liniji broj 20 slike 3.2. nalazi se metoda koja crta liniju – *stroke()*. Konačni rezultat može se vidjeti na slici 3.3.

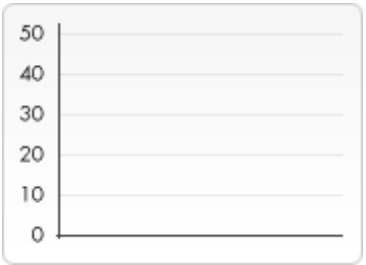
Slika 3.3. Linija nacrtana kodovima iz primjera 3.1. i 3.2.

Kao što je vidljivo, crtanje jednostavnih oblika u HTMLu 5 ne predstavlja problem. Sintaksa je prilično jednostavna, te je malo kodiranja potrebno da bi se dobio rezultat. Već uz pomoć osnovnih metoda definiranih standardom mogu se napraviti razni oblici, te prikazati slika.

3.2. Crtanje slike

Kao što se mogu crtati oblici, tako se mogu crtati i slike u *canvas* elementu. Ova metoda može biti korisna kod, primjerice, izrade 2D igara, gdje postoji neka slika koja se mijenja, a predstavlja pozadinu. Procedura za crtanje slike nije teška, što je vidljivo iz narednog primjera. Primjerice, ukoliko je potrebno prikazati pozadinu s vrijednostima za graf koji će se crtati, to bi se radilo na sljedeći način.

U narednom primjeru, slika 3.4. je pozadina koja će se iscrtati u *canvas* elementu s nazivom datoteke *bg.png*.

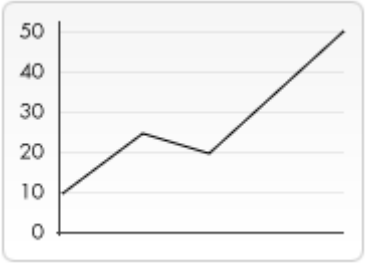


Slika 3.4. Pozadina za graf koji će se crtati

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4  </head>
5  <body onload="draw();">
6      <canvas id="platno" width="500" height="200">
7          Vaš preglednik ne podržava canvas element!
8      </canvas>
9  </body>
10 </html>
11
12 <script>
13
14 function draw() {
15     var canvas = document.getElementById('platno');
16     var context = canvas.getContext('2d');
17
18     var img = new Image();
19     img.src = 'bg.png';
20
21     img.onload = function(){
22         context.drawImage(img,0,0);
23
24         context.beginPath();
25         context.moveTo(30,96);
26         context.lineTo(70,66);
27         context.lineTo(103,76);
28         context.lineTo(170,15);
29         context.lineWidth = 1;
30         context.stroke();
31     }
32 }
33
34 </script>
35
```

Slika 3.5 Postupak koji iscrtava pozadinu i graf koji je proizvoljno određen

Kao što je vidljivo na linijama 18 i 19 slike 3.5., dovoljno je instancirati objekt *Image* u neku varijablu, te definirati njen izvor (*Eng: source)*. Nakon toga, već definiranom *drawImage* metodom [4] *context* objekta crta se slika direktno iz izvora. Cijeli postupak izvršava *draw* metoda koja enkapsulira cijeli postupak, a poziva se pri učitavanju *body* dijela HTML dokumenta. Rezultat postupka vidljiv je na slici 3.6.



Slika 3.6. Rezultat izvršavanja postupka na slici 3.5.

4. IZRADA MODELA

Canvas element vrlo je koristan kad je potrebno crtanje 2D grafika, što je vidljivo iz prethodnih primjera. Po W3C specifikaciji canvas elementa[5], crtanje po canvasu za sada je službeno podržano samo u 2D obliku.

Unatoč tome, postoji nekoliko zajednica koje rade na svojim implementacijama crtanja po *canvasu* u 3 dimenzije. Mozilla zajednica tako razvija webgl kontekst. Kako je trenutno u razvoju, kontekst u skripti definira se kao *experimental-webgl*[1, str. 316], dok bi završena inačica koristila kontekst *webgl*.

Korištenje standarda koji je u razvoju nije nimalo jednostavno. Specifikacija se brzo mijenja, tako da ono što je razvijeno danas, za nekoliko mjeseci ne mora uopće raditi. U demonstraciji mogućnosti HTMLa 5 u sljedećem primjeru, koristit će se WebGL API[1, str. 270]. Budući da je WebGL API iznimno kompleksan, te je za jednostavni 3D model potrebno više stotina linija koda, zbog jednostavnosti implementacije metode APIja[1, str. 270] enkapsulirane su u vlastite metode, koje se, za potrebe ovog primjera, koriste pri izradi modela.

U sljedećem primjeru iscrtan je kvadrat u 3 dimenzije pomoću WebGL APIja. Na taj način demonstrirane su mogućnosti HTMLa 5 pri izradi modela. U HTML dokument potrebno je uključiti dvije Javascript datoteke: *gl-matrix* (WebGL API), te *WebGL* (enkapsulirane metode WebGL APIja).

Treba imati na umu da se navedeni kod može izvršavati samo na web preglednicima koji podržavaju WebGL, kao što je prikazano na slikama 2.1 i 2.2.

Kao prvi korak, uključuju se Javascript datoteke:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta http-equiv="content-type" content="text/html; charset=utf-8">
5  </head>
6  <body>
7      <canvas id="platno" width="600" height="250" style="border:1px solid black;"></canvas>
8      <script type="text/javascript" src="gl-matrix.js"></script>
9      <script type="text/javascript" src="WebGL.js"></script>
```

Slika 4.1. Uključivanje Javascript datoteka i izrada strukture HTML dokumenta

Nakon što je postavljeno početno okruženje, može se započeti s izradom skripte koja izrađuje model kocke.

```
<script>
    function initBuffers(gl){
        var cubeBuffers = {}
        cubeBuffers.positionBuffer = gl.createArrayBuffer([
            // Front face
            -1, -1, 1,
            1, -1, 1,
            1, 1, 1,
            -1, 1, 1,
            // Back face
            -1, -1, -1,
            -1, 1, -1,
            1, 1, -1,
            1, -1, -1,
            // Top face
            -1, 1, -1,
            -1, 1, 1,
            1, 1, 1,
            1, 1, -1,
            // Bottom face
            -1, -1, -1,
            1, -1, -1,
            1, -1, 1,
            -1, -1, 1,
            // Right face
            1, -1, -1,
            1, 1, -1,
            1, 1, 1,
            1, -1, 1,
            // Left face
            -1, -1, -1,
            -1, -1, 1,
            -1, 1, 1,
            -1, 1, -1
        ]);
```

Slika 4.2. Definicija međuspremnika pozicije

Međuspremnik pozicije (*Eng. position buffer*)[1, str. 284] predstavlja polje vrijednosti koje određuju pozicije vektora koji čine kocku. Nakon što se odrede pozicije vektora, izrađuje se međuspremnik boje.

```
45
46          // build color Vertices
47          var colors = [
48              [1, 0, 1, 1], // Front face - Pink
49              [0, 1, 0, 1], // Back face - Green
50              [0, 0, 1, 1], // Top face - Blue
51              [0, 1, 1, 1], // Bottom face - Turquoise
52              [1, 1, 0, 1], // Right face - Yellow
53              [1, 0, 0, 1] // Left face - Red
54          ];
55
56          var colorVertices = [];
57
58          for (var n in colors) {
59              var color = colors[n];
60              for (var i=0; i < 4; i++) {
61                  colorVertices = colorVertices.concat(color);
62              }
63          }
64
65          cubeBuffers.colorBuffer = gl.createArrayBuffer(colorVertices);
```

Slika 4.3. Definicija međuspremnika boje

Međuspremnik boje[1, str. 292] određuje koje će boje biti pojedina stranica kocke, što je definirano u *colors* polju. U *for* petlji spojene su vrijednosti pojedinog elementa *colors* polja. Primjerice, za prvi element polja – [1, 0, 1, 1] - vrijednosti unutar zagrada pretvorene su u RGBA[6] reprezentaciju boje. Kod prve 3 znamenke broj 1 predstavlja broj 255, dok broj 0 predstavlja 0. Zadnja znamenka predstavlja zasićenost, 1 se odnosi na zasićenost od 100%. Ovim postupkom vrlo lako se može zaključiti da je prva boja RGBA(255, 0, 255, 1) – Magenta[7]

Sljedeći korak je definicija međuspremnika indeksa.

```
66
67          cubeBuffers.indexBuffer = gl.createElementArrayBuffer([
68              0, 1, 2, 0, 2, 3, // Front face
69              4, 5, 6, 4, 6, 7, // Back face
70              8, 9, 10, 8, 10, 11, // Top face
71              12, 13, 14, 12, 14, 15, // Bottom face
72              16, 17, 18, 16, 18, 19, // Right face
73              20, 21, 22, 20, 22, 23 // Left face
74          ]);
75
76          return cubeBuffers;
77      }
```

Slika 4.4. Definicija međuspremnika indeksa

Pomoću međuspremnika indeksa[1, str. 292] izrađene su pojedine stranice kocke. Budući da se crtanje 3D oblika u računalnom svijetu uvijek radi pomoću vektora koji čine trokute, ni u WebGL API nije iznimka. Kako bi se izradila jedna stranica kocke, potrebno je definirati dva trokuta koji međusobno tvore kvadrat.

Primjerice, za prvu stranicu kocke koja se definira prvim elementom polja sa slike 4.4. - [0, 1, 2, 0, 2, 3], prve 3 vrijednosti (0, 1, 2) predstavljaju vektore koji određuju prvi, dok se druge 3 (0, 2, 3) odnose na vektore drugog trokuta. Treba napomenuti da su vrijednosti u ovom polju samo reference na vrijednosti polja u međuspremniku pozicije na slici 4.2. Tako vrijednost *0* predstavlja koordinate *-1, -1, -1*, dok *3* predstavlja koordinate *-1, 1, 1*.

Za kraj, preostaje samo postavljanje kuta gledanja, te pozivanje metoda koje će iscrtati kocku.

```
79        function stage(gl, cubeBuffers, angle){
80            // set field of view at 45 degrees
81            // set viewing range between 0.1 and 100.0 units away.
82            gl.perspective(45, 0.1, 100);
83            gl.identity();
84
85            // translate model-view matrix
86            gl.translate(0, 0, -5);
87            // rotate model-view matrix about x-axis (tilt box downwards)
88            gl.rotate(Math.PI * 0.15, 1, 0, 0);
89            // rotate model-view matrix about y-axis
90            gl.rotate(angle, 0, 1, 0);
91
92            gl.pushPositionBuffer(cubeBuffers);
93            gl.pushColorBuffer(cubeBuffers);
94            gl.pushIndexBuffer(cubeBuffers);
95            gl.drawElements(cubeBuffers);
96        }
97
98        window.onload = function(){
99            var gl = new WebGL("platno", "experimental-webgl");
100           gl.setShaderProgram("VARYING_COLOR");
101           var cubeBuffers = initBuffers(gl);
102           var angle = 0;
103           gl.setStage(function(){
104               this.clear();
105               stage(this, cubeBuffers, angle);
106           });
107           gl.start();
108       };
109   </script>
110 </body>
111 </html>
```

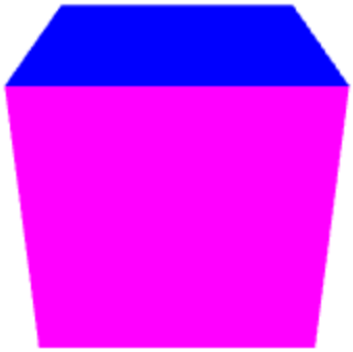
Slika 4.5. Pozivanje metoda koje iscrtavaju kocku

Na linijama 82 do 90 slike 4.5. postavlja se kut gledanja, te se kocka pomiče prema dolje kako bi trodimenzionalno svojstvo kocke bilo više izraženo. U protivnom, vidjeti bi se mogao samo kvadrat.

Linije 92 do 94 postavljaju sva svojstva koja su definirana ranije. Linija 95 crta kocku ovisno o parametrima koji su postavljeni.

Cijeli dio koda koji je obrađen u ovom potpoglavlju izvršava se u trenutku kada se prozor Internet preglednika u potpunosti učita, što linije 98 do 108 daju natuknuti.

Konačni rezultat je kocka na slici ispod.



Slika 4.6. Izrađeni 3D model kocke

5. IZRADA SIMULACIJE

HTML5 sa svojim canvas elementom može poslužiti i za izradu simulacija. Simulacije se izvode Javascript kodom, dok canvas element služi samo kao mjesto na kojem će se simulacija iscrtavati.

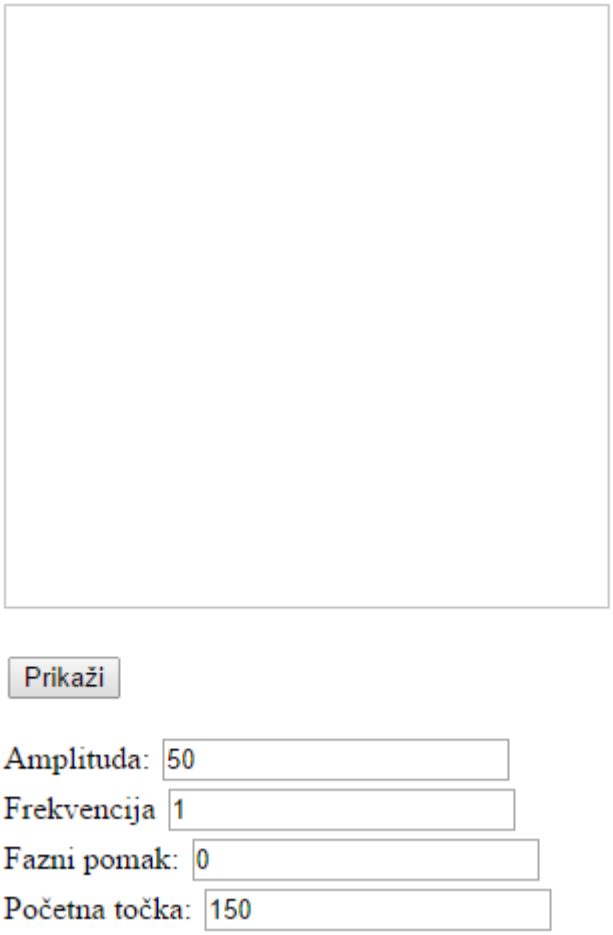
Za izradu simulacija nisu nam potrebne vanjske biblioteke, osim ako se izrađuje simulacija u 3 dimenzije. U sljedećoj demonstraciji prikazat će se simulacija sinusoidne krivulje koja se crta ovisno o parametrima koji su odabrani pri pokretanju crtanja.

Prvi korak je izrada sučelja u kojem se krivulja crta.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta http-equiv="content-type" content="text/html; charset=utf-8">
5  </head>
6  <body>
7      <p>
8          <canvas id="platno" width="1000" height="300" style="border:1px solid #c3c3c3;">
9              <br>Vaš preglednik ne podržava canvas element.<br>
10         </canvas>
11         <br>
12         <br>
13         <button id="calc">Prikaži</button>
14     </p>
15
16     <p>
17         Amplituda: <input type="text" id="amplitude" value="50"><br>
18         Frekvencija <input type="text" id="frequency" value="1"><br>
19         Fazni pomak: <input type="text" id="phase" value="0"><br>
20         Početna točka: <input type="text" id="baseline" value="150">
21     </p>
22 </body>
23 </html>
24
```

Slika 5.1. Izrada sučelja za simulaciju u HTMLu

Na ovaj način definirani su osnovni elementi, kao što su polja za unos, tipka koja pokreće iscrtavanje, te sam *canvas* element u kojemu se krivulja iscrtava.



Slika 5.2. Izgled sučelja za simulaciju u HTMLu

Demonstracija se nastavlja izradom Javascript datoteke koja omogućuje da se sinusoidna krivulja crta. Potrebno je definirati početne vrijednosti za polja za unos, te izraditi metodu koja računa položaj točke koja se crta, pri čemu se koristi geometrijska *sin* operacija.

```
23  <script type="text/javascript">
24      function $(id) {
25          return document.getElementById(id);
26      }
27      var c;
28      var cxt;
29      var x;
30      var y;
31      var baseline = 150;
32      var amp = 50;
33      var phase = 0;
34      var freq = 1;
35      var time = 0;
36      var drawWave;
37      var wave = 0;
38      var colors = ["#FF0000","#00FF00","#0000FF","#FFFF00","#FF00FF","#00FFFF"];
39      function init() {
40          c=document.getElementById("platno");
41          $("calc").addEventListener("click",waveInit,false);
42      }
43      function sinePoint(b,t,a,f,p) {
44          return b + (a * Math.sin(f*t*Math.PI+p));
45      }
```

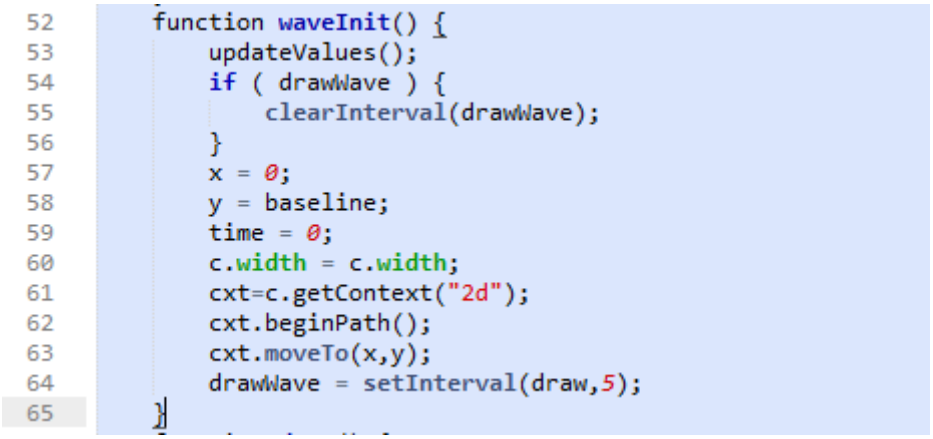
Slika 5.3. Definicija početnih metoda varijabli skripte koja crtaju sinusoidu

Kao što je navedeno ranije, prvo su definirane varijable i njihove početne vrijednosti na linijama 27 do 38 slike 5.3. Zatim je definirana metoda *init* koja prati događaj klika na gumb s identifikatorom *init*, te pokreće metodu *waveInit* koju se definira nešto kasnije. Naposljetku, izrađena je metoda koja računa vrijednost točke sinusoide koju se crta.

```
46      function updateValues() {
47          amp = parseFloat($("amplitude").value);
48          freq = parseFloat($("frequency").value) / 500;
49          phase = parseFloat($("phase").value);
50          baseline = parseFloat($("baseline").value);
51      }
```

Slika 5.4. updateValues metoda

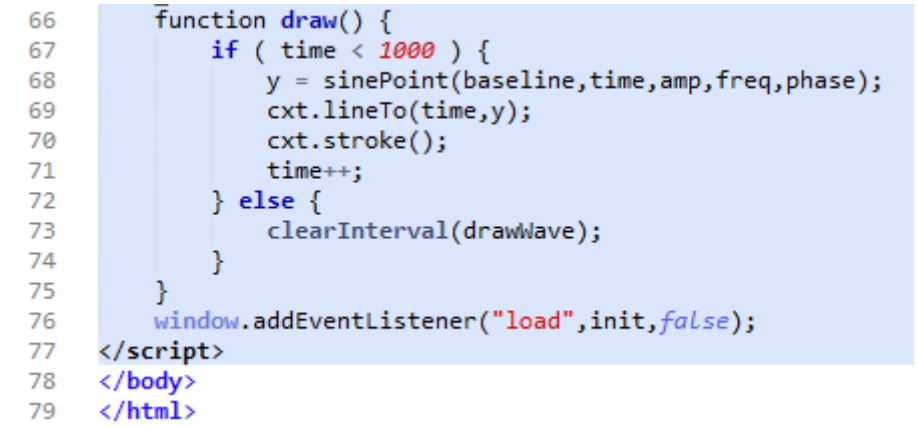
U drugom koraku definirana je *updateValues* metoda, kao što je vidljivo na slici 5.4., pomoću koje se dohvaćaju vrijednosti iz polja za unos koja su prethodno definirana. Ova metodu poziva se svaki puta kada se započinje crtanje sinusoide, kako bi se uvijek koristile najnovije vrijednosti.



```
52      function waveInit() {
53          updateValues();
54          if ( drawWave ) {
55              clearInterval(drawWave);
56          }
57          x = 0;
58          y = baseline;
59          time = 0;
60          c.width = c.width;
61          cxt=c.getContext("2d");
62          cxt.beginPath();
63          cxt.moveTo(x,y);
64          drawWave = setInterval(draw,5);
65      }
```

Slika 5.5. Izgled sučelja za simulaciju u HTMLu

Metoda *waveInit* već je spomenuta pri objašnjenju koda na slici 5.3. Ova metoda zaslužna je za inicijalizaciju konteksta *canvas* elementa, te pozivanje metode *draw*, koja će, u stvarnosti, crtati sinusoidu.



```
66      function draw() {
67          if ( time < 1000 ) {
68              y = sinePoint(baseline,time,amp,freq,phase);
69              cxt.lineTo(time,y);
70              cxt.stroke();
71              time++;
72          } else {
73              clearInterval(drawWave);
74          }
75      }
76      window.addEventListener("load",init,false);
77  </script>
78  </body>
79  </html>
```

Slika 5.6. Izgled sučelja za simulaciju u HTMLu

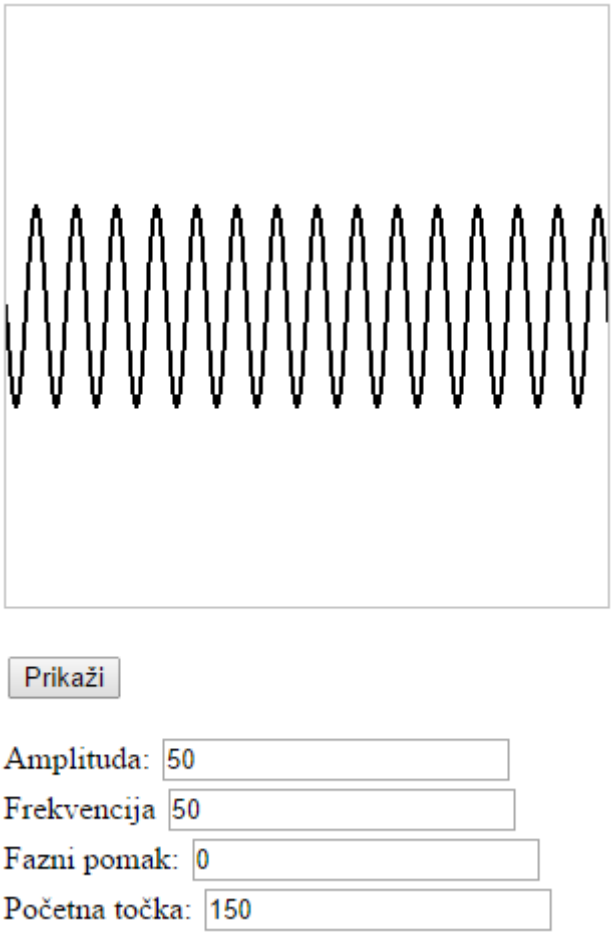
Za kraj preostaje definicija *draw* metode. Naime, ova metoda poziva metodu *sinePoint*, koja određuje *y* koordinatu, odnosno vrijednost točke ovisne o vremenu (linija 68 slike 5.6.). Zatim se poziva metoda *lineTo*, koja crta liniju do točke koja je definirana. Kao *x* koordinata uzima se vrijeme u milisekundama. Nakon toga, poziva se metoda *stroke* koja crta liniju, te se povećava vrijednost varijable *time*, pomoću koje se određuje *x* koordinata.

Ako je prošlo vise od 1000 milisekundi od početka izvršavanja, pomoću metode *clearInterval* zaustavlja se izvršavanje skripte.

Sav kod koji koji je napisan u ovom poglavlju ne izvršava se bez povezanog događaja (*Eng: event)* koji se promatra, te metode koja se poziva kada se taj događaj dogodi. Metoda *addEventListener[8]* poziva se na *window* object, odnosno prozor Internet preglednika, te prima

3 parametra[4]. Prvi parametar odnosi se na događaj koji se promatra, a drugi na metodu koju se poziva pozvati kada se događaj dogodi. Treći parametar je neobavezan, ovu demonstraciju.

Krajnji rezultat izvršavanja ovog programa vidljiv je na slici 5.7.



Slika 5.7. Krajnji rezultat izvršavanja koda u ovom poglavlju

[4] document.addEventListener(event, function, useCapture)

6. ZAKLJUČAK

HTML5 pruža mnogo mogućnosti kad je u pitanju iscrtavanje 2D crteža i manipulacija animacijama. 2D crteže vrlo je lako crtati i animirati ako poznajemo barem osnovne mogućnosti Javascript jezika, što je vidljivo iz demonstracija u prethodnim poglavljima.

Kada se radi o 3D modelima i animiranju istih, proces postaje mnogo kompliciraniji. Budući da specifikacija za 3D iscrtavanje ne postoji, odnosno 3D kontekst još uvijek nije službeno podržan, najčešće je potrebno oslanjati se na uključivanje dodatnih biblioteka koje implementiraju WebGL API. Uz to, kako bi proces bio jednostavniji, preporučljivo je WebGL metode implementirati u vlastite metode koje se pozivaju pri crtanju.

Dakle, iako HTML5 specifikacija napreduje u dobrom smjeru, HTML5 još uvijek nije zreo za jednostavno crtanje modela i simulacija u usporedbi sa, još uvijek najpopularnijim, Flash formatom. Može se reći da je HTML5 sa svojim canvas elementom trenutno idealan za crtanje jednostavnih, čak i kompleksnih 2D crteža. No kada se priča o 3D modelima i animacijama, HTML5 nije jednostavan za korištenje koliko bi mogao biti. Tu u priču najčešće ulazi WebGL API, koji je izuzetno kompleksno programsko sučelje s vrlo strmom krivuljom učenja, što obeshrabruje mnoge koji bi se rado upustili u 3D animiranje bez potrebe za uključivanjem vanjskih biblioteka.

Ono što je također nedostatak WebGL programskog sučelja je nekompatibilnost sa starijim preglednicima, što znači da, ukoliko preglednik koji korisnik koristi ne podržava WebGL, korisnik vidi samo poruku koja ga o tome obavještava. Također treba uzeti u obzir i to da većina korisnika koristi zastarjele preglednike, što dodatno obeshrabruje programere pri odluci za korištenje HTMLa 5 i Javascripta za iscrtavanje modela i simulacija.

Budući da HTML5 standard još uvijek nije finaliziran, pred njim je svijetla budućnost. Ako ne u 3D modelima i simulacijama, svakako će unijeti promjene u iscrtavanju grafika na web stranicama.

LITERATURA

[1] E. Rowell, HTML5 Canvas Cookbook, Packt Publishing, UK, 2011.

[2] Mozilla Foundation, HTMLCanvasElement, 25.8.2014.

https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement , 9.2.2014.

[3] Wikipedia, JavaScript, 8.9.2014., http://en.wikipedia.org/wiki/JavaScript , 9.9.2014

[4] W3Schools, HTML canvas drawImage() Method,

http://www.w3schools.com/tags/canvas_drawimage.asp , 2.9.2014.

[5] W3C, HTML Canvas 2D Context, 21.8.2014., http://www.w3.org/TR/2014/CR-2dcontext-20140821/ , 2.9.2014.

[6] Webflux, RGBA Colors, http://www.css3.info/preview/rgba/ 2.9.2014.

[7] Wikipedia, Magenta, 11.3.2013., http://hr.wikipedia.org/wiki/Magenta , 2.9.2014.

[8] W3Schools, HTML DOM addEventListener() Method, http://www.w3schools.com/jsref/met_document_addeventlistener.asp , 2.9.2014.

[9] Gary Longworth, HTML5 Canvas – Sine waves, http://thestratusquo.com/html5-canvas-sine-waves/ , 2.9.2014.

SAŽETAK

U radu su prikazane mogućnosti HTML5 standarda, te njegovih novih elemenata pri crtanju. Prikazano je koji Internet preglednici podržavaju 2D i 3D crtanje po novom elementu HTML5 standarda – canvasu (platnu). Također, prikazana je i kompatibilnost s Internet preglednicima na mobilnim uređajima. Za demonstraciju mogućnosti crtanja 2D oblika, nacrtana je linija, te graf uporabom samo canvas HTML elementa i Javascript funkcija. Kod demonstracije mogućnosti crtanja 3D modela u HTMLu 5, nacrtana je kocka. Kako bi se olakšao razvoj, enkapsulirane su neke metode WebGL APIja. Rezultat je odličan, te dokazuje kako HTML5 može bez imalo problema iscrtati 3D model, uz odlične performanse. Za programere je, ipak, postupak relativno težak. Također je izrađena simulacija sinusiodne krivulje. Ovisno o ulaznim parametrima koji se odrede, crta se sinusoidna krivulja. Dakle, HTML5 ima odlične performanse pri 2D crtanju, te ne zahtjeva razne dodatke za preglednik kako bi mogao crtati. Uz to, 3D crtanje je također moguće, unatoč nedovršenoj specifikaciji.

KLJUČNE RIJEČI: crtanje, HTML 5, modeliranje, platno, WebGL

ABSTRACT

HTML5 capabilities in model and simulation drawing

In this paper, drawing capabilities of HTML5 standard and its new elements were tested. Internet browser support for 2D and 3D drawing under the new HTML5 element - canvas - was showed, as well as HTML5 browser compatibility on mobile devices. To demonstrate the possibilities of drawing 2D shapes, both line and a graph were drawn using only HTML canvas element and Javascript functions. For demonstration of the drawing possibilities of 3D models in HTML5, a cube was drawn. To facilitate development, WebGL API methods were wrapped with a wrapper functions. The result is excellent and proves that HTML5 can be used to draw a 3D model without any issues, with great performance. For developers, however, the process is still relatively difficult. A simulation of a sine wave was also developed. The sine wave is drawn depending on the input parameters. HTML5 has excellent performance in 2D drawing, and does not require any add-on or plug-in for the browser to enable drawing. In addition, 3D drawing is also possible, despite unfinished specification.

KEYWORDS: canvas, drawing, HTML 5, modelling, WebGL

ŽIVOTOPIS

Petar Sambolek rođen je u Zagrebu 23. rujna 1991. Osnovnu školu završio je u Kutini. Srednju školu završava 2010. u Tehničkoj Školi Kutina, te stječe zvanje tehničara za računalstvo. 2010. godine upisuje stručni studij Informatike na Elektrotehničkom fakultetu u Osijeku.

Od 2012. godine radi u sektoru informacijsko-komunikacijskih tehnologija, gdje u praksi primjenjuje svoja znanja.

Služi se engleskim, te njemačkim jezikom.

Petar Sambolek

(vlastoručni potpis)

PRILOZI

Na optičkom disku priloženom ovom radu nalaze se digitalna inačica ovog rada i sve datoteke.

Enkapsulacija WebGL APIja[5]

```
var WebGL = function(canvasId){
        this.canvas = document.getElementById(canvasId);
        this.context = this.canvas.getContext("experimental-webgl");
        this.stage = undefined;
 // Animation
 this.t = 0;
 this.timeInterval = 0;
 this.startTime = 0;
 this.lastTime = 0;
 this.frame = 0;
 this.animating = false;
 // provided by Paul Irish
 window.requestAnimFrame = (function(callback){
        return window.requestAnimationFrame ||
        window.webkitRequestAnimationFrame ||
        window.mozRequestAnimationFrame ||
        window.oRequestAnimationFrame ||
        window.msRequestAnimationFrame ||
```

[5] [1, str. 270]

```
        function(callback){
                window.setTimeout(callback, 1000 / 60);
        };
})();
/*
* encapsulte mat3, mat4, and vec3 from
* glMatrix globals
*/
this.mat3 = mat3;
this.mat4 = mat4;
this.vec3 = vec3;

// shader type constants
this.BLUE_COLOR = "BLUE_COLOR";
this.VARYING_COLOR = "VARYING_COLOR";
this.TEXTURE = "TEXTURE";
this.TEXTURE_DIRECTIONAL_LIGHTING = "TEXTURE_DIRECTIONAL_LIGHTING";

this.shaderProgram = null;
this.mvMatrix = this.mat4.create();
this.pMatrix = this.mat4.create();
this.mvMatrixStack = [];
this.context.viewportWidth = this.canvas.width;
```

```
 this.context.viewportHeight = this.canvas.height;


 // init depth test
 this.context.enable(this.context.DEPTH_TEST);
};
WebGL.prototype.getContext = function(){
        return this.context;
};
WebGL.prototype.getCanvas = function(){
        return this.canvas;
};
WebGL.prototype.clear = function(){
        this.context.viewport(0, 0, this.context.viewportWidth, this.context.viewportHeight);
        this.context.clear(this.context.COLOR_BUFFER_BIT                                    |
this.context.DEPTH_BUFFER_BIT);
};
WebGL.prototype.setStage = function(func){
        this.stage = func;
};
WebGL.prototype.isAnimating = function(){
        return this.animating;
};
WebGL.prototype.getFrame = function(){
```

```
	return this.frame;
};
WebGL.prototype.start = function(){
	this.animating = true;
	var date = new Date();
	this.startTime = date.getTime();
	this.lastTime = this.startTime;

	if (this.stage !== undefined) {
		this.stage();
	}

	this.animationLoop();
};

WebGL.prototype.stopAnimation = function(){
	this.animating = false;
};
WebGL.prototype.getTimeInterval = function(){
	return this.timeInterval;
};
WebGL.prototype.getTime = function(){
	return this.t;
```

```
};
WebGL.prototype.getFps = function(){
	return this.timeInterval > 0 ? 1000 / this.timeInterval : 0;
};
WebGL.prototype.animationLoop = function(){
	var that = this;

	this.frame++;
	var date = new Date();
	var thisTime = date.getTime();
	this.timeInterval = thisTime - this.lastTime;
	this.t += this.timeInterval;
	this.lastTime = thisTime;

	if (this.stage !== undefined) {
		this.stage();
	}

	if (this.animating) {
		requestAnimFrame(function(){
			that.animationLoop();
		});
	}
```

```
};
WebGL.prototype.save = function(){
	var copy = this.mat4.create();
	this.mat4.set(this.mvMatrix, copy);
	this.mvMatrixStack.push(copy);
};

WebGL.prototype.restore = function(){
	if (this.mvMatrixStack.length == 0) {
		throw "Invalid popMatrix!";
	}
	this.mvMatrix = this.mvMatrixStack.pop();
};
WebGL.prototype.getFragmentShaderGLSL = function(shaderType){
	switch (shaderType) {
		case this.BLUE_COLOR:
		return "#ifdef GL_ES\n" +
		"precision highp float;\n" +
		"#endif\n" +
		"void main(void) {\n" +
		"gl_FragColor = vec4(0.0, 0.0, 1.0, 1.0);\n" +
		"}";
		case this.VARYING_COLOR:
```

```
return "#ifdef GL_ES\n" +
"precision highp float;\n" +
"#endif\n" +
"varying vec4 vColor;\n" +
"void main(void) {\n" +
"gl_FragColor = vColor;\n" +
"}";
case this.TEXTURE:
return "#ifdef GL_ES\n" +
"precision highp float;\n" +
"#endif\n" +
"varying vec2 vTextureCoord;\n" +
"uniform sampler2D uSampler;\n" +
"void main(void) {\n" +
"gl_FragColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));\n" +
"}";
case this.TEXTURE_DIRECTIONAL_LIGHTING:
return "#ifdef GL_ES\n" +
"precision highp float;\n" +
"#endif\n" +
"varying vec2 vTextureCoord;\n" +
"varying vec3 vLightWeighting;\n" +
```

```
"uniform sampler2D uSampler;\n" +
"void main(void) {\n" +
"vec4 textureColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));\n" +
"gl_FragColor = vec4(textureColor.rgb * vLightWeighting, textureColor.a);\n" +
"}";
}
};
WebGL.prototype.getVertexShaderGLSL = function(shaderType){
switch (shaderType) {
case this.BLUE_COLOR:
return "attribute vec3 aVertexPosition;\n" +
"uniform mat4 uMVMatrix;\n" +
"uniform mat4 uPMatrix;\n" +
"void main(void) {\n" +
"gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);\n" +
"}";
case this.VARYING_COLOR:
return "attribute vec3 aVertexPosition;\n" +
"attribute vec4 aVertexColor;\n" +
"uniform mat4 uMVMatrix;\n" +
"uniform mat4 uPMatrix;\n" +
"varying vec4 vColor;\n" +
```

```
"void main(void) {\n" +
"gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);\n" +
"vColor = aVertexColor;\n" +
"}";
case this.TEXTURE:
return "attribute vec3 aVertexPosition;\n" +
"attribute vec2 aTextureCoord;\n" +
"uniform mat4 uMVMatrix;\n" +
"uniform mat4 uPMatrix;\n" +
"varying vec2 vTextureCoord;\n" +
"void main(void) {\n" +
"gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);\n" +
"vTextureCoord = aTextureCoord;\n" +
"}";
case this.TEXTURE_DIRECTIONAL_LIGHTING:
return "attribute vec3 aVertexPosition;\n" +
"attribute vec3 aVertexNormal;\n" +
"attribute vec2 aTextureCoord;\n" +
"uniform mat4 uMVMatrix;\n" +
"uniform mat4 uPMatrix;\n" +
"uniform mat3 uNMatrix;\n" +
"uniform vec3 uAmbientColor;\n" +
"uniform vec3 uLightingDirection;\n" +
```

```
                "uniform vec3 uDirectionalColor;\n" +
                "uniform bool uUseLighting;\n" +
                "varying vec2 vTextureCoord;\n" +
                "varying vec3 vLightWeighting;\n" +
                "void main(void) {\n" +
                "gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);\n" +
                "vTextureCoord = aTextureCoord;\n" +
                "if (!uUseLighting) {\n" +
                "vLightWeighting = vec3(1.0, 1.0, 1.0);\n" +
                "} else {\n" +
                "vec3 transformedNormal = uNMatrix * aVertexNormal;\n" +
                "float directionalLightWeighting = max(dot(transformedNormal, uLightingDirection), 0.0);\n" +
                "vLightWeighting = uAmbientColor + uDirectionalColor * directionalLightWeighting;\n" +
                "}\n" +
                "}";
        }
};
WebGL.prototype.initShaders = function(shaderType){
        this.initPositionShader();

        switch (shaderType) {
                case this.VARYING_COLOR:
```

```
            this.initColorShader();
            break;
            case this.TEXTURE:
            this.initTextureShader();
            break;
            case this.TEXTURE_DIRECTIONAL_LIGHTING:
            this.initTextureShader();
            this.initNormalShader();
            this.initLightingShader();
            break;
        }
};
WebGL.prototype.setShaderProgram = function(shaderType){
        var fragmentGLSL = this.getFragmentShaderGLSL(shaderType);
        var vertexGLSL = this.getVertexShaderGLSL(shaderType);

        var fragmentShader = this.context.createShader(this.context.
            FRAGMENT_SHADER);
        this.context.shaderSource(fragmentShader, fragmentGLSL);
        this.context.compileShader(fragmentShader);

        var vertexShader = this.context.createShader(this.context.VERTEX_SHADER);
        this.context.shaderSource(vertexShader, vertexGLSL);
```

```
        this.context.compileShader(vertexShader);

-       this.shaderProgram = this.context.createProgram();
        this.context.attachShader(this.shaderProgram, vertexShader);
        this.context.attachShader(this.shaderProgram, fragmentShader);
        this.context.linkProgram(this.shaderProgram);

        if (!this.context.getProgramParameter(this.shaderProgram,
                this.context.LINK_STATUS)) {
                alert("Could not initialize shaders");
}

this.context.useProgram(this.shaderProgram);

 // once shader program is loaded, it's time to init the shaders
 this.initShaders(shaderType);
};
WebGL.prototype.perspective = function(viewAngle, minDist,
        maxDist){
        this.mat4.perspective(viewAngle, this.context.viewportWidth /
this.context.viewportHeight, minDist, maxDist, this.pMatrix);
};
WebGL.prototype.identity = function(){
```

```
        this.mat4.identity(this.mvMatrix);
};
WebGL.prototype.translate = function(x, y, z){
        this.mat4.translate(this.mvMatrix, [x, y, z]);
};
WebGL.prototype.rotate = function(angle, x, y, z){
        this.mat4.rotate(this.mvMatrix, angle, [x, y, z]);
};
WebGL.prototype.initPositionShader = function(){
        this.shaderProgram.vertexPositionAttribute =
this.context.getAttribLocation(this.shaderProgram, "aVertexPosition");
        this.context.enableVertexAttribArray(this.shaderProgram.vertexPositionAttribute);
        this.shaderProgram.pMatrixUniform =
this.context.getUniformLocation(this.shaderProgram, "uPMatrix");
        this.shaderProgram.mvMatrixUniform =
this.context.getUniformLocation(this.shaderProgram, "uMVMatrix");
};
WebGL.prototype.initColorShader = function(){
        this.shaderProgram.vertexColorAttribute =
this.context.getAttribLocation(this.shaderProgram, "aVertexColor");
        this.context.enableVertexAttribArray(this.shaderProgram.vertexColorAttribute);
};
WebGL.prototype.initTextureShader = function(){
```

```
        this.shaderProgram.textureCoordAttribute =
this.context.getAttribLocation(this.shaderProgram, "aTextureCoord");

        this.context.enableVertexAttribArray(this.shaderProgram.textureCoordAttribute);

        this.shaderProgram.samplerUniform =
this.context.getUniformLocation(this.shaderProgram, "uSampler");

};

WebGL.prototype.initNormalShader = function(){

        this.shaderProgram.vertexNormalAttribute =
this.context.getAttribLocation(this.shaderProgram, "aVertexNormal");

        this.context.enableVertexAttribArray(this.shaderProgram.vertexNormalAttribute);

        this.shaderProgram.nMatrixUniform =
this.context.getUniformLocation(this.shaderProgram, "uNMatrix");

};

WebGL.prototype.initLightingShader = function(){

        this.shaderProgram.useLightingUniform =
this.context.getUniformLocation(this.shaderProgram, "uUseLighting");

        this.shaderProgram.ambientColorUniform =
this.context.getUniformLocation(this.shaderProgram, "uAmbientColor");

        this.shaderProgram.lightingDirectionUniform =
this.context.getUniformLocation(this.shaderProgram, "uLightingDirection");

        this.shaderProgram.directionalColorUniform =
this.context.getUniformLocation(this.shaderProgram, "uDirectionalColor");

};

WebGL.prototype.initTexture = function(texture){

        this.context.pixelStorei(this.context.UNPACK_FLIP_Y_WEBGL, true);
```

```
        this.context.bindTexture(this.context.TEXTURE_2D, texture);
        this.context.texImage2D(this.context.TEXTURE_2D, 0, this.context.RGBA, this.context.RGBA, this.context.UNSIGNED_BYTE, texture.image);
        this.context.texParameteri(this.context.TEXTURE_2D, this.context.TEXTURE_MAG_FILTER, this.context.NEAREST);
        this.context.texParameteri(this.context.TEXTURE_2D, this.context.TEXTURE_MIN_FILTER, this.context.LINEAR_MIPMAP_NEAREST);
        this.context.generateMipmap(this.context.TEXTURE_2D);
        this.context.bindTexture(this.context.TEXTURE_2D, null);
};
WebGL.prototype.createArrayBuffer = function(vertices){
        var buffer = this.context.createBuffer();
        buffer.numElements = vertices.length;
        this.context.bindBuffer(this.context.ARRAY_BUFFER, buffer);
        this.context.bufferData(this.context.ARRAY_BUFFER, new Float32Array(vertices), this.context.STATIC_DRAW);
        return buffer;
};
WebGL.prototype.createElementArrayBuffer = function(vertices){
        var buffer = this.context.createBuffer();
        buffer.numElements = vertices.length;
        this.context.bindBuffer(this.context.ELEMENT_ARRAY_BUFFER, buffer);
        this.context.bufferData(this.context.ELEMENT_ARRAY_BUFFER, new Uint16Array(vertices), this.context.STATIC_DRAW);
```

```
        return buffer;
};
WebGL.prototype.pushPositionBuffer = function(buffers){
        this.context.bindBuffer(this.context.ARRAY_BUFFER, buffers.positionBuffer);
        this.context.vertexAttribPointer(this.shaderProgram.vertexPositionAttribute, 3, this.context.FLOAT, false, 0, 0);
};
WebGL.prototype.pushColorBuffer = function(buffers){
        this.context.bindBuffer(this.context.ARRAY_BUFFER, buffers.colorBuffer);
        this.context.vertexAttribPointer(this.shaderProgram.vertexColorAttribute, 4, this.context.FLOAT, false, 0, 0);
};
WebGL.prototype.pushTextureBuffer = function(buffers, texture){
        this.context.bindBuffer(this.context.ARRAY_BUFFER, buffers.textureBuffer);
        this.context.vertexAttribPointer(this.shaderProgram.textureCoordAttribute, 2, this.context.FLOAT, false, 0, 0);
        this.context.activeTexture(this.context.TEXTURE0);
        this.context.bindTexture(this.context.TEXTURE_2D, texture);
        this.context.uniform1i(this.shaderProgram.samplerUniform, 0);
};
WebGL.prototype.pushIndexBuffer = function(buffers){
        this.context.bindBuffer(this.context.ELEMENT_ARRAY_BUFFER, buffers.indexBuffer);
};
```

```
WebGL.prototype.pushNormalBuffer = function(buffers){
    this.context.bindBuffer(this.context.ARRAY_BUFFER, buffers.normalBuffer);
    this.context.vertexAttribPointer(this.shaderProgram.vertexNormalAttribute, 3, this.context.FLOAT, false, 0, 0);
};
WebGL.prototype.setMatrixUniforms = function(){
    this.context.uniformMatrix4fv(this.shaderProgram.pMatrixUniform, false, this.pMatrix);
    this.context.uniformMatrix4fv(this.shaderProgram.mvMatrixUniform, false, this.mvMatrix);

    var normalMatrix = this.mat3.create();
    this.mat4.toInverseMat3(this.mvMatrix, normalMatrix);
    this.mat3.transpose(normalMatrix);
    this.context.uniformMatrix3fv(this.shaderProgram.nMatrixUniform, false, normalMatrix);
};
WebGL.prototype.drawElements = function(buffers){
    this.setMatrixUniforms();

 // draw elements
 this.context.drawElements(this.context.TRIANGLES, buffers.indexBuffer.numElements, this.context.UNSIGNED_SHORT, 0);
};
WebGL.prototype.drawArrays = function(buffers){
```

```
        this.setMatrixUniforms();


 // draw arrays
 this.context.drawArrays(this.context.TRIANGLES, 0, buffers.positionBuffer.numElements / 3);
};
WebGL.prototype.enableLighting = function(){
        this.context.uniform1i(this.shaderProgram.useLightingUniform, true);
};
WebGL.prototype.setAmbientLighting = function(red, green, blue){
        this.context.uniform3f(this.shaderProgram.ambientColorUniform, parseFloat(red),
parseFloat(green), parseFloat(blue));
};
WebGL.prototype.setDirectionalLighting = function(x, y, z, red,
        green, blue){
 // directional lighting
 var lightingDirection = [x, y, z];
 var adjustedLD = this.vec3.create();
 this.vec3.normalize(lightingDirection, adjustedLD);
 this.vec3.scale(adjustedLD, -1);
 this.context.uniform3fv(this.shaderProgram.lightingDirectionUniform, adjustedLD);


 // directional color
 this.context.uniform3f(this.shaderProgram.directionalColorUniform, parseFloat(red),
parseFloat(green),
```

```
        parseFloat(blue));
};
```

Izrada simulacije[9]

```
<!DOCTYPE html>
<html>
<head>
        <meta http-equiv="content-type" content="text/html; charset=utf-8">
</head>
<body>

        <p>
                <canvas id="platno" width="1000" height="300" style="border:1px solid #c3c3c3;">
                        <br>Vaš preglednik ne podržava canvas element.<br>
                </canvas>
                <br>
                <br>
                <button id="calc">Prikaži</button>
        </p>

        <p>
                Amplituda: <input type="text" id="amplitude" value="50"><br>
                Frekvencija <input type="text" id="frequency" value="1"><br>
```

```
        Fazni pomak: <input type="text" id="phase" value="0"><br>
        Početna točka: <input type="text" id="baseline" value="150">
    </p>

    <script type="text/javascript">
        function $(id) {
            return document.getElementById(id);
        }
        var c;
        var cxt;
        var x;
        var y;
        var baseline = 150;
        var amp = 50;
        var phase = 0;
        var freq = 1;
        var time = 0;
        var drawWave;
        var wave = 0;
        var colors =
["#FF0000","#00FF00","#0000FF","#FFFF00","#FF00FF","#00FFFF"];
        function init() {
            c=document.getElementById("platno");
```

```
        $("calc").addEventListener("click",waveInit,false);
}
function sinePoint(b,t,a,f,p) {
        return b + (a * Math.sin(f*t*Math.PI+p));
}
function updateValues() {
        amp = parseFloat($("amplitude").value);
        freq = parseFloat($("frequency").value) / 500;
        phase = parseFloat($("phase").value);
        baseline = parseFloat($("baseline").value);
}
function waveInit() {
        updateValues();
        if ( drawWave ) {
                clearInterval(drawWave);
        }
        x = 0;
        y = baseline;
        time = 0;
        c.width = c.width;
        cxt=c.getContext("2d");
        cxt.beginPath();
        cxt.moveTo(x,y);
```

```
                drawWave = setInterval(draw,5);
        }
        function draw() {
                if ( time < 1000 ) {
                        y = sinePoint(baseline,time,amp,freq,phase);
                        cxt.lineTo(time,y);
                        cxt.stroke();
                        time++;
                } else {
                        clearInterval(drawWave);
                }
        }
        window.addEventListener("load",init,false);
    </script>
</body>
</html>
```