Pretvaranje 2D SVG modela u 3D model

Sviben, Nika

SVEUČILIŠTE U ZAGREBU
GRAFIČKI FAKULTET

ZAVRŠNI RAD

Nika Sviben

Sveučilište u Zagrebu
Grafički fakultet

Smjer: Tehničko-tehnološki

ZAVRŠNI RAD

Pretvaranje 2D SVG modela u 3D model

Mentor:

prof. dr. sc. Klaudio Pap

Student:

Nika Sviben

Zagreb, 2014

U radu se istražuje pretvaranje SVG 2D modela u 3D model. 2D vektorski model može postati 3D model upotrebom virtualnog izvora svjetlosti ili mijenjanjem projekcije modela tako da model dobiva treću dimenziju u vidu matematičke reprezentacije kolekcije podataka o točkama u 3D prostoru i pripadnih drugih informacija. Eksperimentalni dio je izveden na primjeru 2D modela u SVG formatu koji se pomoću JavaScripta transformira u 3D model. Kako prilikom pretvaranja 2D modela u 3D model nedostaje informacija o trećoj dimenziji potrebno ju je odrediti. Program omogućava unošenje dubine , odnosno iznos proširenja u treću dimenziju. 3D model se prikazuje unutar pretraživača, te se rotira i translatira kako bi se vidjela trodimenzionalnost. Kako bi se 3D model mogao prikazati potrebno je definirati matematičke operacije pomoću kojih računalo izračunava poziciju točaka ovisno o pomaku. Navedeno, računalo izračunava putem matrica transformacije. Svaka točka modela je prikazana u obliku matrice. Sam model se sastoji od vektora. Matematičke operacije koje se koriste su objašnjene u radu.

KLJUČNE RIJEČI: SVG, JavaScript, 2D, 3D, matrice, vektori

1. UVOD

SVG (Scalable Vector Graphics) je raširena grafički format za vektorsku grafiku stvoren od strane W3C. Posebno je pogodan za internet jer omogućuje stvaranje manjih, bržih i interaktivnijih dokumenata. SVG omogućuje stvaranje 2D vektorskih modela, a to može biti ograničavajuće. Međutim, u SVG model moguće je integrirati skripte potpunog programskog jezika poput ECMAScripta (standardizirani JavaScript) te na taj način proširiti mogućnosti. Dodane skripte mogu pristupati elementima SVG-a i mijenjati ih. Također, SVG je moguće i animirati te u potpunosti prikazati sve tri dimenzije modela. JavaScript je programski jezik napravljen za Internet te je i najkorišteniji na internetu. Velik broj internetskih stranica koristi JavaScript te svi pretraživači na raznim uređajima podržavaju JavaScript. U ovom radu će se pomoću JavaScripta dodati treću dimenziju 2D SVG modelu, te će se animirati nastali 3D model.

Teorijski dio rada sastoji se od četiri poglavlja. Prvo poglavlje bavi se SVG-om te XML-om jer je SVG aplikacija XML-a za vektorsku grafiku. U drugom poglavlju objašnjene su osnove JavaScripta i DOM-a. Javascript može pristupati elementima XML dokumenta i mijenjati ih putem DOM-a, aplikacijskog korisničkog sučelja. U trećem i četvrtom poglavlju objašnjeni su vektori i matrice. Vektori se u vektorskoj grafici koriste kako bi se njima prikazale točke u prostoru, kao što su lokacije objekata ili vrhovi nekog poligona. Također, njima se može opisati smjer u prostoru za orijentaciju kamere ili normale površine. U 3D modeliranju matrice se koriste kako bi preračunale točke objekta u prostoru ovisno o promjeni perspektive gledanja objekta. S obzirom da 3D objekt nije realan, odnosno nalazi se u memoriji računala, potrebno ga je prikazati na 2D zaslonu računala. Za svaki pomak potrebno je ponovo preračunati gdje se nalaze točke. Računalo se pri tome koristi matricama transformacije.

Eksperimentalni dio rada bavi se programom koji pretvara 2D SVG model u 3D model. Program je se sastoji od dva dokumenta. U dokumentu *Math3D.js* definirane su klase objekata koji će se koristiti kod izrade programa. JavaScript je objektno orijentiran jezik te

postoji mogućnost određivanja 'nacrta' prema kojima se kreiraju novi objekti slijedeći definirana pravila. U drugom dokumentu 2Du3D.svg nalaze se 3 SVG 2D modela koji će se putem JavaScript skripte koja se nalazi u istom dokumentu pretvarati u 3D modele. Program se nalazi u prilogu rada.

2. TEORIJSKI DIO

2.1. SVG

Extensible Markup Language (XML) je jednostavan i vrlo fleksibilan tekstualni format koji je proizašao od SGML-a (Standard Generalized Markup Language) i zapravo je podskup SGML-a. HTML je aplikacija SGML-a. Napravljen je za elektroničko izdavaštvo, ali igra sve važniju ulogu u razmjeni podataka putem interneta i drugdje. [2]

Markup označava način izražavanja strukture dokumenta unutar samog dokumenta. XML je stvoren kako bi pružio dogovorenu sintaksu te s time omogućio da svi dokumenti postanu dostupni svim korisnicima bez obzira na format. Za razliku od HTML, u XML-u nije puno toga unaprijed definirano. Oznake <> označavaju elemente kao i kod HTML-a, ali nema set definiranih elemenata kao što su glava, tijelo itd.. XML je zapravo set pravila koje omogućavaju pisanje drugih jezika. [2]

Kako znakovi <,>,'," i & imaju značenje, u XML-u postoje zamjenski znakovi kako bi se mogli koristiti i ne narušiti ispravnost dokumenta. To su &lt, &gt, &apos, &guot i &amp. Također, XML dopušta stvaranje vlastitih zamjenskih znakova. Također, moguće je napisati bilo koji Unicode znak putem njegove numeričke pozicije. Jedan od osnovnih zahtjeva za XML procesore je podrška svih Unicode standardnih znakova. Unicode 3.0. ima više od 57 700 znakova. Što je više nego što je potrebno kod pisanja dokumenta koji se obično piše na jednom jeziku. Kako bi se mogao kodirati svaki znak trebalo bi 4 okteta za svaki znak. Zato postoji UTF-8 i UTF-16. Na taj način aplikacije mogu procesuirati dokumente u 8- ili 16- bitnim segmentima. Ako znak nije dostupan u jednom segmentu onda će se procesuirati sljedeći znak što je puno lakše za memoriju i spremište procesora. XML podržava i druge načine kodiranja znakova. [2]

Osim što mora biti ispravan, XML dokument mora biti i valjan. To znači da se moraju definirati elementi i atributi koji će se koristiti i kojim redom. To se definira sa DTD-om (*Document Type Definition*). DTD je obično sastavljen od jednog dokumenta u kojem su

navedene deklaracije koje definiraju tipove elemenata i liste atributa. DTD se dodaje XML-u poslije XML deklaracije sa <!DOCTYPE...> te linkom na dokument. DTD deklaracija specificira korijenski element dokumenta. DTD se može dodati za više elemenata. XML procesor će provjeriti valjanost dokumenta prema navedenom DTD-u. Ako nije valjan dokument će biti odbijen. [2]

Kako XML dopušta kreiranje elemenata i atributa moguće je preklapanje naziva. Zato postoji način da se definiraju specifični rječnici koristeći URI (*Uniform Resource Identifiers*). SVG koristi URI http://www.w3.org/2000/svg i uključen je u SVG DTD. URI je identifikator pomoću kojega se otvara stranica na XML specifikacije. On omogućava programima koji procesuiraju dokumente prepoznavanje kojim vokabularom raspolažu. [2]

Scalable Vector Graphics (SVG) je jezik kojim se opisuje dvodimenzionalna grafika u XML-u. SVG dopušta tri osnovna grafička objekta: vektorsku grafiku, rastersku grafiku i tekst. Objekti se mogu grupirati, transformirati, stilizirati i dodavati u već napravljene objekte. SVG crteži mogu biti interaktivni i dinamički. Animacije se mogu definirati i pokretati deklarativno ili putem skripte. [3]

1998. godine World Wide Web Consortium (W3C) formirao je radnu grupu kako bi se napravila aplikacija XML-a za prikaz vektorske grafike. Kako je SVG aplikacija XML-a informacije o slici se spremaju u tekstualnom obliku te ima sve prednosti XML-a (otvorenost, prenosivost i interoperabilnost). Također, SVG se može koristiti uz druge XML aplikacije. SVG je i W3C preporuka te neki program koriste SVG format za eksportiranje crteža. Za pregled u pretraživaču potrebno je instalirati *plug-in* koji omogućuje slične mogućnosti skriptiranja i animacije kao i Flash. [2]

Stvaranje SVG dokumenta započinje sa standardnim XML uputama za procesuiranje i deklaracijom dokumenta.

```
<?xml version=’’1.0’’?>
<!DOCTYPE svg PUBLIC “-//W3C//DTD SVG 1.0//EN”
      "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
```

Zatim, se definiramo površinu prikaza ‘platna’ na kojem će se prikazati nacrtani objekt. Površinu definiramo unutar korijenskog <svg> elementa sa atributima *width* i *height* u pikselima. Kada površina ne bi bila definirana onda bi cijeli prostor pretraživača bila površina na kojoj bi se prikazivao objekt. Također, na početku možemo dati neki naziv objektu sa atributom <title> te ga opisati sa atributom <desc>. [2]

Kada je definirano sve potrebno može se početi crtati objekt. SVG podržava više osnovnih oblika. Najjednostavniji element je <line> . S njime se može nacrtati ravna linija, koja se definira sa početnom i završnom točkom u koordinatnom sustavu. Zatim, sa elementom <rectangle> može se nacrtati pravokutnik. Definiran je sa x i y koordinatama gornjeg lijevog kuta te njegovom širinom i visinom. Također, može se nacrtati i pravokutnik sa zaobljenim vrhovima, a tada još dodatno treba definirati x i y radijus zakrivljenosti vrhova. Mogu se nacrtati zaobljeni objekti; krug i elipsa. Krug <circle> se definira sa koordinatama središta i radijusom. Elipsa <ellipse> se dodatno definira sa x i y radijusom. Sa elementom <polygon> mogu se nacrtati zahtjevniji oblici po želji kao što su heksagoni, zvijezde ili bilo koji zatvoreni oblik. Potrebno je definirati seriju točaka koje opisuju oblik . Točke su x i y koordinate koje su odvojene zarezom, a dvije točke su odvojene razmakom. Objekt će se zatvoriti spajajući početnu točku i zadnju napisanu točku. Zadnji podržani oblik je <polyline> koji omogućuje crtanje više spojenih točaka koje nisu zatvorene. [2]

Navedeni oblici su brži načini crtanja željenih oblika, ali sve ih je moguće nacrtati i sa elementom <path> jer svi ti oblici jesu staze. Sa navedenim elementom moguće je nacrtati obrise željenog oblika opisivanjem seriju spojenih linija, lukova i krivulja. Crtanje staze počinje sa *moveto* komandom, veliko slovo M, te x i y koordinate. Ova komanda određuje početak puta. Zatim slijede jedna ili više lineto komandi označene sa velikim slovom L te x i y koordinatama. Putem ove komande crta se linija od početka puta do točke definirane

komandom. Može se koristiti i closepath komanda s kojom se zatvara put od zadnje navedene točke do početne. Označava se sa slovom Z. Ako bi komande napisali sa malim slovima to bi koordinate će biti relativne poziciji na kojoj se nalazi početak puta. [2]

SVG omogućuje razdvajanje strukture dokumenta od prezentacije. Struktura bi bili svi oblici i njihova pozicija, prezentacija je boja, debljina linija te ispuna. Prezentaciju je moguće definirati na četiri načina. Prvi način je unutar nekog atributa tako da se nakon naredbe za crtanje nekog oblika definira njegov izgled sa atributom <style> i zatim nizom vizualnih svojstva i njihovim vrijednostima. Drugi način je pomoću unutarnjih *stylesheet*-ova. Putem *stylesheet*-a možemo unaprijed definirati vizualna svojstva koja želimo za neki oblik i zatim ih samo primijenimo te ih ne moramo svaki puta ispisivati. Treći način je putem vanjskih *stylesheet*-ova. Na ovaj način možemo spremiti neki *stylesheet* kao poseban dokument formata .css koji možemo primijeniti na veći broj SVG dokumenata. Četvrti način, omogućuje definiranje vizualnih svojstava putem prezentacijskih atributa. Na taj način vizualna svojstva postaju dio strukture jer se ne upotrebljava atribut <style>. Prilikom iscrtavanja nekog objekta uvijek će prioritet imati stylesheet, te će se zanemariti prezentacijski atributi ako su navedeni. Ovaj način se koristi ako se SVG pregledava u okruženju koje ne podržava stylesheet-ove. [2]

Osim crteža SVG podržava i tekst pa je moguće dodavanje teksta nekoj grafici sa <text> atributom. Za ispisivanje jednostavnog teksta potrebno je samo definirati jednu točku u koordinatnom sustavu, a ta točka predstavljat će početak pismovne linije. Tekst će biti ispisan crnom bojom bez ovojnice. Putem atributa <style> moguće je odrediti željeni font, stil i boju ispisa. Dodano je moguće odrediti smjer ispisa te dužinu teksta. Također, moguće je dodati i rastersku grafiku sa <image> atributom. Potrebno je navesti poveznicu na kojoj se nalazi grafika. [2]

Kako bi SVG dokument bio razumljiviji i bolje strukturiran moguće je grupirati neke oblike. <g> element omogućuje grupiranje odabranih u elemenata i dodavanje id atributa kako bi se ta grupa nazvala. <g> element postaje roditeljski element, a svi elementi koji su grupirani postaju elementi djeca. Također, omogućuje upisivanje vizualnih svojstva jednom

na početku, a ta se svojstva primjenjuju na sve oblike koji su u grupi. Nakon što se grupiraju objekti se mogu iskoristiti sa <use> elementom putem xlink:href atributa te x i y koordinata na koje želimo smjestiti te objekte. Element <defs> omogućuje definiranje prije nacrtanih objekata bez da ih se odmah nacrta. To je također i pogodno za bolje procesuiranje podataka. Kada želimo objektima možemo promijeniti boju ispune ili debljinu linije po želji. Element <simbol> dodatno omogućuje određivanje prozora u kojem će biti prikazani objekti i očuvanje omjera proporcija širine i visine. [2]

SVG omogućuje i mijenjanje koordinatnog sustava. Objekte je moguće pomicati na novu lokaciju (translate), rotirati (rotate) i mijenjati veličinu (scale). Pri tome se ne mijenja sam objekt nego se mijenja cijeli koordinatni sustav. Sve to je moguće napraviti sa *transform* atributom. [2]

SVG omogućuje animaciju objekta i interakciju sa objektom. Animacija je kontrolirana od strane autora tog svg dokumenta i ne može se mijenjati, ali uz korištenje skripte omogućuje se upravljanje animacijom korisniku. [2]

Animacija u SVG je bazirana na SMIL2 WWW3 specifikacijama. U tom sustavu trebaju se odrediti početna i završna vrijednost atributa, boja, kretanje ili transformacija, vrijeme kada bi animacija trebala početi i vrijeme trajanja animacije. Također, moguće je primijeniti više animacija u jednom SVG dokumentu. [2]

Elementom <a> omogućuje se interakcija sa SVG dokumentom. Kada se grafika nalazi unutar <a> elementa postaje aktivna. Kada se odabere otvara se URL koji je napisan u xlink:href atributu. URL može biti od nekog drugog SVG dokumenta ili na neku web stranicu. Moguće je napraviti neki objekt i odrediti mu boju i ostala vizualna svojstva te odabirom tog objekta otvoriti poveznicu. [2]

Još veću interakciju omogućuju skripte. ECMAScript, poznatiji kao JavaScript omogućuje interakciju sa SVG grafikom. Interakcija se događa kada grafički objekt reagira na događaje. Objekti mogu reagirati na događaje povezane s pritiskom miša: *click, mousedown* ili *mouseup;* događaje povezane sa pomicanjem miša: *mouseover, mouseout* i *mousemove;*

događaje povezane sa statusom objekta*: load;* te nestandardizirani događaji povezani sa pritiskom na tipkovnicu: *keydown i keyup*. Kako bi objekt reagirao na događaj dodaje se atribut *oneventName* elementu koji želimo. Vrijednost atributa biti će definirana u skripti, obično u pozivu neke funkcije. *Evt* je rezervirana riječ koja u skripti opisuje svojstva i metode koji opisuju događaj. Najčešće metode su *getTarget()*koji vraća referencu na grafički objekt koji će reagirati na događaj, te *getClientX()* i *getClientY()*koji vraćaju x i y koordinate miša kada se dogodi događaj. Početak skripte se označava sa <script> i to znači da se više ne radi o SVG/XML-u nego ECMAScript okruženju. Atribut *type* određuje koji programski jezik se koristi. Kako ECMAScript nije XML <![CDATA[govori XML da taj dio ne tumači kao posebne znakove. Sintaksa XML-a i ECMAScript-a je različita te znak < u XML-u označava početak oznake dok u ECMAScript-u ma drugo značenje. Znakovi]]> označavaju prestanak CDATA konstrukcije i ponovnu interpretaciju XML-a. Atributi grafičkog objekta mogu se mijenjati sa funkcijama *setAttribute* i *getAttribut*. Sa prvom se mijenjaju vrijednosti atributa, a sa drugom se dobivaju trenutne vrijednosti atributa. Ove funkcije su dio DOM-a (Document Object Model) koji je standardni API za pristupanje, mijenjanje i premještanje elemenata u XML dokumentu. [2]

SVG grafiku je moguće ugraditi u HTML dokument odnosno web stranicu. To se može napraviti sa <embed> elementom. Važni atributi koji se mogu odrediti su *src* kojim se određuje URL grafike*, width i height* koji određuju visinu i širinu, te *type* atribut koji će biti *image/svg+xml*. Nakon što je ugrađen SVG, moguće je dodati skriptu i SVG dokumentu i HTML dokumentu kako bi mogli komunicirati. [2]

Animaciju i skriptu moguće je koristiti zajedno, pa se ne primjer nakon završetka animacije poveziva neka funkcija. Također, moguće je animaciju vršiti putem skripte jer se na taj način dobiva više mogućnosti. Prednost animacije integrirane u SVG je jednostavnost čitanja za ljude i za XML alate. Animacija putem skripte omogućuje da animacija ovisi o poziciji miša ili raznim uvjetima sa više varijabli. [2]

2.2. JAVASCRIPT

JavaScript je programski jezik interneta. Većina modernih web stranica koristi JavaScript i svi moderni web pretraživači podržavaju JavaScript. JavaScript je programski jezik visoke razine što znači da je apstraktniji, lakši i bolje prenosiv po različitim platformama od jezika nižih razina. Također, JavaScript je dinamičan, interpretativni programski jezik. Prilagođen je za objektno orijentirano i funkcionalno programiranje. Sintaksa dolazi od programskog jezika Jave, a funkcije visoke razine dolaze od programskog jezika Scheme. JavaScript je stvorio Netscape na početku razvoja interneta i JavaScript je naziv implementacije jezika za korištenje u Netscape-u (danas Mozilla). Netscape je poslao jezik za standardizaciju ECMA-i (European Computer Manufacturer's Association) te je jezik dobio naziv ECMAScript zbog pravnih razloga. Microsoft-ova verzija jezika zove se JScript, ali u praksi gotovo svi jezik zovu JavaScript. Naziv ECMAScript koristi se kada se referira na standard jezika. Do nedavno svi pretraživači su imali implementiranu podršku za verziju 3 ECMAScript-a, ali je dostupna nova verzija jezika ECMAScript verzija 5 te sve veći broj pretraživača implementira podršku i za noviju verziju. Također, Mozilla ima svoje verzije koje uključuju i nestandardizirane dodatke jeziku, a i interpreteri jezika imaju svoje verzije, pa je tako zadnja Google-ova verzija interpretera V8 broj 3. [4]

Svaki jezik mora imati platformu ili standardiziranu biblioteku ili API (aplikacijsko korisničko sučelje) u kojima su određena pravila i specifikacije za unos i ispis podataka. Srž JavaScript-a je definiranje minimalnog API-a za rad sa tekstom, nizovima, datumima i izrazima ali ne uključuje funkcije unosa i ispisa. Unos i ispis kao i neke druge kompleksnije značajke su dio okruženja u kojem je ugrađen JavaScript. Obično je to okruženje pretraživač. [4]

JavaScript programi pisani su koristeći Unicode set znakova. Također, jezik je osjetljiv na mala i velika slova te se identifikatori uvijek moraju pisati na isti način. Inače će se tumačiti kao različiti. Kako se često koristi unutar HTML dokumenta koji nije osjetljiv potrebno je pripaziti kod pisanja. JavaScript zanemaruje razmake između znakova te se može formatirati program na pregledan način po želji autora. Specijalne znakove koje ne

podržavaju svi programi moguće je napisati koristeći 6 ASCII znakova kojima je moguće napisati bilo koji 16-bitni Unicode znak. Komentare je moguće pisati na dva načina. Sav tekst između znakova // i kraja linije JavaScript tumači kao komentar i ignorira ih. Također, sav tekst između /* */ se tretira kao komentar, ali takvi komentari mogu se protezati kroz nekoliko linija. Literali su vrijednosti koje se pojavljuju izravno u programu. To mogu biti cijeli brojevi, decimalni brojevi, stringovi, Boolean vrijednosti (true/false) ili null koji označava da nema vrijednosti. Identifikatori su nazivi varijabli ili funkcija te određenih petlji. Identifikatori moraju početi sa slovnim znakom, donjom crtom (_) ili znakom za dolar ($). Nisu dopušteni brojevi kako bi se mogli raspoznati identifikatori od brojeva. Postoje određene rezervirane riječi koje se ne smiju koristiti jer su ključne riječi u samom programu ili bi mogle to postati. Točka i zarez se koriste za odvajanje naredbi, ali ih je moguće izostaviti ako su naredbe napisane u različitim linijama. [4]

JavaScript podržava primitivne i objektne tipove vrijednosti. Primitivne vrijednosti su brojevi, stringovi teksta i Boolean vrijednosti. Posebne vrijednosti su *null* i *undefiend* primitivne vrijednosti jer nisu brojevi, stringovi ili boolean vrijednosti. Svaka vrijednost koja nije jedna od navedenih je objekt. Objekti su skup svojstava, gdje svako svojstvo ima naziv i vrijednost. Posebna skupina objekata je niz, uređeni skup brojeva. Za niz postoji posebna sintaksa i njihovo ponašanje je drugačije od ostalih objekata. Također, funkcije su objekti koje mogu izvršavati naredbe koje su s njima povezane. Kao i nizovi, funkcije se ponašaju drugačije od ostalih objekata. [4]

DOM je osnovni API kojim se prezentira i manipulira sadržajem HTML i XML dokumenta. Ugniježđeni elementi HTML ili XML dokumenta su prikazani u DOM-u kao stablo objekata. Takav prikaz sadrži čvorove koji predstavljaju elemente HTML ili XML dokumenta i tekst, a također i komentare. Svaki čvor koji se nalazi izravno ispod nekog drugog je dijete, a čvor iznad je roditelj. Na vrhu je čvor koji predstavlja cijeli dokument. Postoji više podtipova čvorova koje predstavljaju različite tipove elemenata, a svaki definira JavaScript svojstva koja preslikavaju HTML ili XML atribute specifičnog elementa ili grupe elemenata. [4]

Većina JavaScript programa radi na način da nekako manipulira jednim ili više elemenata dokumenta. Kada program započinje koristi se varijabla *document* kako bi se referirali na dokument. Kako bi mogli manipulirati elementima moraju ih nekako odabrati. DOM definira više načina na koje je to moguće: pomoću *id* atributa, pomoću *name* atributa, pomoću oznaka, pomoću CSS klasa ili sparivanjem CSS selektora. Najčešći način je pomoću *id* atributa. Svaki element može imati *id* atribut sa jedinstvenom vrijednosti unutar dokumenta. Element se može odabrati pomoću metode *getElementByID()*. [4]

2.3. Vektori

Vektori su ključni za grafičke programe. Koriste se kako bi se njima prikazale točke u prostoru, kao što su lokacije objekata ili vrhovi nekog poligona. Također, njima se može opisati smjer u prostoru za orijentaciju kamere ili normale površine. [6]

Vektor opisuje dužinu i smjer. Vektor n-dimenzije može se napisati kao:

$$V = \langle V_1, V_2, \dots, V_n \rangle \quad (1)$$

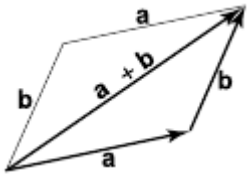
gdje su V_i komponente vektora **V**. Vektor se može napisati i kao matrica sa jednim stupcem i n brojem redova:

$$V = \begin{bmatrix} V_1 \\ V_2 \\ \vdots \\ V_n \end{bmatrix} \quad (2)$$

Također, mogu se prikazati i 3D ili 4D vektori u Kartezijskom koordinatnom sustavu. Kod 3D vektora koristi se baza vektora z koja je ortogonalna sa x i y. [7]

Vektorima je moguće raditi uobičajene aritmetičke operacije kao i kod realnih brojeva. Dva vektora su jednaka samo i samo onda kada imaju istu dužinu i smjer. Dva vektora zbrajaju se prema pravilu paralelograma. Sumu možemo dobiti spajanjem kraja jednog vektora sa

početkom drugog. Suma će biti vektor koji spaja ta dva vektora i s njima stvara trokut. Paralelogram dobivamo uzimanjem sume u bilo kojem redu. [8]



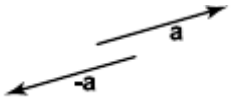
Slika 1. Pravilo paralelograma, preuzeto iz [5].

Zbrajanje vektora je komutativno, te vrijedi:

$$a + b = b + a \ (3)$$

Vektori se mogu i oduzimati. Vektor –a je vektor iste dužine kao i vektor a, ali mu je smjer suprotan te vrijedi:

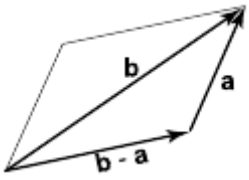
$$b - a \equiv -a + b \ (4)$$



Slika 2. Vektori iste dužine, ali različitog smjera, preuzeto iz [5]

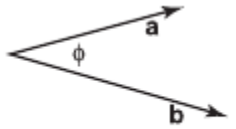
Vektori se mogu oduzimati također po pravilu paralelograma, te vrijedi:

$$a + (b - a) = b \ (5)$$



Slika 3. Oduzimanje vektora, preuzeto iz [5]

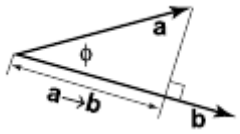
Također, moguće je i množiti sa vektorima. Tako je skalarni produkt:



$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\phi \quad (6)$$

Slika 4. Skalarni produkt, preuzeto iz [5]

Skalarni produkt daje vrijednost povezanu sa dužinom i kutom između vektora, te se najčešće koristi za izračunavanje kuta između dva vektora. Također, koristi se za pronalaženje projekcije jednog vektora na drugi. [5]



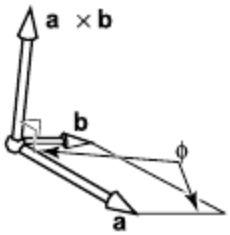
Slika 5. Projekcija vektora na drugi vektor, preuzeto iz [5]

Svojstva skalarnog produkta su komutativnost i distributivnost. Produkt skalarnog množenja vektora je uvijek skalar. [5]

Vektorski produkt je:

$$\|a \times b\| = \|a\|\|b\| \sin \emptyset \quad (7)$$

Vektorski produkt se obično koristi kod trodimenzionalnih vektora. Vektorski produkt daje vrijednost 3D vektora koji je okomit na dva argumenta vektorskog produkta. Dužina dobivenog vektora je povezana sa sinØ. Veličina površine $\|a \times b\|$ je jednaka površini paralelograma koji stvaraju vektori a i b. [5]



Slika 6. Vektorski produkt, preuzeto iz [5]

2.4. Matrice

Grafički API je kolekcija standardnih funkcija kojima se izvršavaju operacije kao što su crtanje modela i 3D površina na zaslonu računala. Gotovo svako računalo danas ima mogućnost iscrtavanja 3D modela na zaslonu. Osnovne operacije pri tome su određivanje vrhova 3D modela na 2D zaslonu na način da se zadrži njihov odnos u dubini kako bi se dobio 3D doživljaj. Nekada je najveći problem predstavljalo upravo navedeno, ali se sada to rješava sa z-međuspremnikom. Sve grafičke manipulacije rade se u 4D koordinatnom sustavu gdje postoje 3 klasične koordinate i dodatnom četvrtom dimenzijom koja služi kao pomoć kod perspektive. Navedenim koordinatama manipulira se pomoću 4x4 matrica i 4-vektorima. [5]

U 3D modeliranju matrice se koriste kako bi preračunale točke objekta u prostoru ovisno o promjeni perspektive gledanja objekta. S obzirom da 3D objekt nije realan odnosno nalazi se u memoriji računala potrebno ga je prikazati na 2D zaslonu računala, te je za svaki pomak potrebno ponovo preračunati gdje se nalaze točke. Računalo se pri tome koristi matricama transformacije. [6]

Općenito, matrica je pravokutna tablica brojeva ili općenito tablica koja se sastoji od apstraktnih objekata koji se mogu zbrajati i množiti. Koriste se za opisivanje linearnih jednadžbi, za praćenje koeficijenata linearnih transformacija te za čuvanje podataka koji ovise o dva parametra. Matrice se mogu zbrajati, množiti i razlagati na razne načine. Horizontalne linije u matrici zovu se retcima, a vertikalne stupcima matrice. Matematički

matrica se definira: neka su m i n prirodni brojevi. Matrica A tipa (reda ili formata) mxn je svaka pravokutna tablica elemenata (brojeva) poredanih u m redaka i n stupaca. Simbolički:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{mn} \end{bmatrix} \quad (8)$$

Ili kraće:

$$A = [a^{ij}], i = 1,2,\dots,m;\ j = 1,2,\dots,n. \quad (9)$$

Matrice se označavaju sa A, B, C,... A matrični elementi sa a_{ij}, b_{kl}, c_{if}. Skup svih matrica označava se sa M_{mn} ili M_{mxn} ili R_{mxn}. Ako je $m = n$ matrica se naziva kvadratna matrica reda n. Također postoji kvadratna matrica za koju vrijedi $m \neq n$. Nul-matrica je matrica čiji su svi elementi jednaki nuli a označava se sa O ili O_{mn}. Matrice su jednake ako i samo ako su istog reda i svi odgovarajući elementi su im međusobno jednaki. Ako u matrici zamijenimo retke i stupce (prvi redak pišemo kao prvi stupac zatim drugi redak kao drugi stupac…) dobivamo transponiranu matricu A^T ili A'. Kvadratna matrica je simetrična ako vrijedi $A^T = A$, te antisimetrična ako vrijedi $A^T = -A$. Matrice se zbrajaju ako su istog reda i to tako da zbrajamo odgovarajuće elemente. [10]

$$A = |a_{ij}|, B = |b_{ij}| \in M_{mn} \Rightarrow A + B = |a_{ij} + b_{ij}| \in M_{mn} \quad (10)$$

Matricu možemo množiti brojem tako da njime pomnožimo svaki njezin element.

$$A = |a_{ij}| \in M_{mn}, \lambda \in R \quad (11)$$

Za $\lambda = 0$ je $0 \cdot A = 0$ (12) (nul-matrica).

Za $\lambda = -1$ je $-1 \cdot A = -A$ (13) (suprotna matrica).

Oduzimanje matrica je zbrajanje sa suprotnom matricom , $A - B = A + (-B)$, te vrijedi:

$$A - A = A + (-A) = 0 \text{ (14).}$$

Dvije matrice mogu se pomnožiti međusobno samo ako su ulančane (prva matrica ima toliko stupaca koliko druga redaka). Produkt je matrica sa toliko redaka kao prva, a stupaca kao druga matrica u umnošku. [7]

$$A(m \cdot n) \cdot B(n \cdot p) = C(m \cdot p) \text{ (15)}$$

2.4.1. 2D transformacije

Možemo koristiti matricu 2x2 kako bismo promijenili, tj. transformirali 2D vektor:

$$\begin{bmatrix} a_{11} a_{12} \\ a_{21} a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix} \text{ (16)}$$

Ova operacija, gdje se uzima jedan vektor i dobiva se drugi vektor jednostavnim množenjem matrice, naziva se linearna transformacija. Ova jednostavna formula može se koristiti za razne korisne transformacije kao što je pomicanje po x i y osima. [5]

Jedna od osnovnih transformacija je povećavanje ili smanjivanje po koordinatnim osima. Ovom transformacijom može se promijeniti dužina i smjer. Kod navedene transformacije kod vektora sa Kartezijskim komponentama (x,y) događa se [5]:

$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix} \text{ (17)}$$

Ako želimo rotirati vektor **a** za neki kut Ø suprotno od smjera kazaljke na satu kako bismo dobili vektor **b,** isto možemo napraviti putem matrice:

$$\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \quad (18)$$

Kako je zbroj svakog reda matrice 1 ($\sin^2 \emptyset + \cos^2 \emptyset = 1$) (19) i redovi su ortogonalni ($\cos\emptyset(-\sin\emptyset) + \sin\emptyset\cos\emptyset = 0$) (20), matrice rotacije su ortogonalne matrice [5].

2.4.2. 3D transformacije

Linearne 3D transformacije su nastavak na 2D linearne transformacije. Pomoću njih možemo raditi rotacije, smanjivanje i povećavanje, rotacije ili pomicanje u sve 3 dimenzije. [5]

3D rotacije su znatno kompliciranije nego 2D rotacije, ali ako se želi rotirati samo oko jedne osi onda se mogu koristiti 2D transformacije bez ikakvih operacija na osi oko koje se želi rotirati:

$$z(\emptyset) = \begin{bmatrix} \cos\emptyset & -\sin\emptyset & 0 \\ \sin\emptyset & \cos\emptyset & 0 \\ 0 & 0 & 1 \end{bmatrix}, (21)$$

$$x(\emptyset) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\emptyset & -\sin\emptyset \\ 0 & \sin\emptyset & \cos\emptyset \end{bmatrix}, (22)$$

$$y(\emptyset) = \begin{bmatrix} \cos\emptyset & 0 & \sin\emptyset \\ 0 & 1 & 0 \\ \sin\emptyset & 0 & \cos\emptyset \end{bmatrix}. (23)$$

Kao i 2D, 3D rotacije su ortogonalne matrice. To znači da su 3 reda matrice Kartezijske koordinate 3 zajednički ortogonalna vektora. Stupci su 3 potencijalno različita, međusobno ortogonalna vektora. Postoji neograničen broj takvih rotacijskih matrica. [5]

$$R_{uvw} = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix} (24)$$

Vrijedi, $u = x_u\boldsymbol{x} + y_u\boldsymbol{y} + z_u\boldsymbol{z}$ te isto za v i w. Kako su 3 vektora ortonormalna znamo da vrijedi:

$$\boldsymbol{u} \cdot \boldsymbol{u} = \boldsymbol{v} \cdot \boldsymbol{v} = \boldsymbol{w} \cdot \boldsymbol{w} = 1 \quad (25)$$

$$\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{v} \cdot \boldsymbol{w} = \boldsymbol{w} \cdot \boldsymbol{u} = 0 \quad (26)$$

Možemo zaključiti kako će se ponašati matrica kada ju upotrijebimo na vektorima $\boldsymbol{u}$, $\boldsymbol{v}$ i $\boldsymbol{w}$. Na primjer:

$$R_{uvw}u = \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix} \begin{bmatrix} x_u \\ y_u \\ z_u \end{bmatrix} = \begin{bmatrix} x_u x_u + & y_u y_u + & z_u z_u \\ x_v x_u + & y_v y_u + & z_v z_u \\ x_w x_u + & y_w y_u + & z_w z_u \end{bmatrix} \quad (27)$$

3 reda matrice su skalarni produkti:

$$R_{uvw}u = \begin{bmatrix} u \cdot u \\ v \cdot u \\ w \cdot u \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = x \quad (28)$$

Slično, $R_{uvw}v = y$ i $R_{uvw}w = z$. To znači da R_{uvw} prenosi $\boldsymbol{uvw}$ do odgovarajućih Kartezijskih osi putem rotacije. [5]

Ako je R_{uvw} rotacijska matrica sa ortonormalnim redovima onda za R_{uvw}^T vrijedi isto. Znači da ako R_{uvw}prenosi u do x onda R_{uvw}^Tprenosi x do u. Isto vrijedi i za v te y:

$$R_{uvw}^{\mathrm{T}}y = \begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} = v \quad (29)$$

Uvijek možemo stvarati rotacijske matrice iz ortonormalnih baza.

Ako želimo rotirati oko nekog vektora a, možemo stvoriti ortonormalnu bazu sa w=a, rotirati tu bazu oko x, y i z osi, zatim rotirati oko z-osi te rotirati osi x,y i z natrag u uvw bazu. [5]

Rotacija oko w-osi za kut Ø:

$$\begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix} \quad (30)$$

Kod mijenjanja vektora putem matrice u dvije dimenzije transformacije imaju formu:

$$x' = m_{11}x + m_{12}y \quad (32)$$

$$y' = m_{21}x + m_{22}y \quad (33)$$

Kod promijene veličine i rotacije središte (0,0) uvijek ostaje isto, pa se ne može isto primijeniti za pomicanje. [5]

Za pomicanje, translataciju, objekta potrebno je pomaknuti sve točke za isti iznos, na način:

$$x' = x + x_t \quad (34)$$

$$y' = y + y_t \quad (35)$$

Navedeno nije moguće izračunati množenjem (x,y) sa 2x2 matricom, ali je moguće sa 3x3 matricom kod koje je točka prikazana sa x i y, a 3D vektor sa $[xy1]T$. [5]

$$\begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix} \quad (36)$$

Treći red je fiksan i služi kako bi se prenijela jedinica u transformirani vektor, te kako bi svi vektori imali 1 na zadnjem mjestu. Prva dva reda izračunavaju x' i y' kao linearnu kombinaciju x,y i 1. [5]

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + x_t \\ m_{21}x + m_{22}y + y_t \\ 1 \end{bmatrix} (37)$$

Navedeno je dobiveno pomoću homgenih koordinata. Također, gore napisano množenje matrice predstavlja afinu transformaciju i ima svojstva da čuva paralelnost pravaca, ali ne čuva duljine i kutove. [5]

Definicija homogenih koordinata je :

Skup svih trojki $(x1, x2, x3)$ u kojoj nisu svi xi nula su homogene koordinate u realnoj projektivnoj ravnini, gdje su (x_1, x_2, x_3) koordinate (a) Euklidske točke (x,y), ako je $x = x1/x3\ i\ y = xx, x3, x3 \neq 0$ (38), (b) Idealne točke s nagibom $x2/x1, ako\ je\ x3 = 0\ i\ x1 \neq 0$. (39) i (c) Idealne točke s beskonačnim nagibom (vertikala) ako je $x1 = x3 = 0$ (40).

Navedeno je moguće generalizirati na više dimenzionalne projektivne ravnine.

Za točke izražene u homogenim koordinatama, sve tri transformacije se mogu prikazati kao množenja matrica i vektora. Kod homogenih koordinata dodaje se treća koordinata.

$$(x, y) \rightarrow (x, y, w) (41)$$

Dvije trojke homogenih koordinata (x, y, w) i (x', y', w') reprezentiraju istu točku ako i samo ako za $t \neq 0$ vrijedi:

$$(x', y', w') = (t \cdot x, t \cdot y, t \cdot w)\ ili\ (x, y, w) = \left(\frac{1}{t} \cdot x', \frac{1}{t} \cdot y', \frac{1}{t} \cdot w'\right) (42)$$

Barem jedna od homogenih koordinata mora biti $\neq 0$; $(0,0,0)$ nije dozvoljeno.

Ako je $w \neq 0$ tada (x, y, w) (43) reprezentira istu točku kao i $\left(\frac{x}{w}, \frac{y}{w}, 1\right)$ (44). Pri čemu se x/w i y/w nazivaju Kartezijvim koordinatama homogene točke. Točke sa $w = 0$ nazivaju se točkama u beskonačnosti. Sve trojke $(tx, ty, tw), t \neq 0$ (45) koje reprezentiraju istu točku, čine pravac u 3D prostoru. Ako se homogeniziraju trojke u pravcu (podijele sa w koordinatom), dobit će se točka oblika $(x, y, 1)$. Homogenizirane točke oblikuju ravninu sa jednadžbom $w = 1\ u\ (x, y, w)$ – prostoru (46). [9]

Problem nastaje ako se želi transformirati vektore koji predstavljaju smjer ili pomak između mjesta. Takvi vektori ne bi se smjeli mijenjati kada se napravi translatacija. Zato se u tom slučaju treća koordinata upisuje kao 0. [5]

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} (47)$$

Ako se vrši rotacija ili promjena veličine u gornjem lijevom dijelu 2x2 matrice biti će primijenjena na vektor, ali translatacija se množi sa 0 i ne izvršava se. Homogena koordinata može imati i druge vrijednosti osim 0 i 1, a to se koristi kod određivanje perspektive gledanja. [5]

Kod 3D vektora translatacija se vrši preko 4x4 matrice uz proširenje vektora uz korištenje homogene koordinate. 3D točka je proširena u 4 dimenzije dodavanjem četvrte koordinate, koja je jednaka 1. 4x4 matrica se konstruira na način da se napravi 3x3 matrica m uz 3D translataciju t.

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & t_x \\ m_{21} & m_{22} & m_{23} & t_y \\ m_{31} & m_{32} & m_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} (48)$$

Ovdje također vrijedi da je homogena koordinata 1 ako se radi o prostornom vektoru, a 0 ako se radi o vektoru koji predstavlja smjer u 3D prostoru. [6]

3. EKSPERIMENTALNI DIO

Program koji pretvara 2D model u 3D model sastoji se od dva dokumenta. U prvom dokumentu koji se zove *Math3D.js* definirne su klase objekata. U drugom dokumentu *2Du3D.svg* nalaze se 3 SVG 2D modela te skripta koja pretvara 2D modele u 3D modele. Pretvorba se vrši putem skripte koja se nalazi u istom dokumentu. Programski jezik koji se koristi je JavaScript.

3.1. Math 3D skripta

Kako je JavaScript objektno orijentirani jezik moguće je kao i kod ostalih objektno orijentiranih jezika definirati klase objekata koji dijele slična svojstva. Definira se prototipni objekt te svi objekti koji nasljeđuju njegova svojstva pripadaju određenoj klasi. Kada se definira klasa moguće je napraviti objekt koji će nasljediti sve metode i funkcije definirane za tu klasu objekata. Tako je moguće brzo stvoriti više istih objekata. U dokumentu *Math3D.js* definirane su klase prema kojima će se kreirati objekti u skripti unutar SVG-a. Skripta koja se koristi za definiranje klasa preuzeta je sa stranice *http://www.kevlindev.com/*. Autor dopušta korištenje koda uz mogućnost modifikacije.

Unutar skripte definirane su 4 klase: *Point3D, Vertex, Mesh3D* i *Transform*. *Point3D* definira metode pomoću kojih se stvara 3D vektor i definira unutar 4x4 matrice. Metode koje su definirane su *transform* sa kojim se stvaraju 3 točke, te se svaka od točaka definira kao matrica, *project* koji služi za fokusiranje na jedan element, ali neće biti korišten u programu te *toString* koja vraća vrijednost dvije točke u tekstualnom obliku te također neće biti korišten.

```
function Point3D(x, y, z) {
        this.x3 = x;
        this.y3 = y;
        this.z3 = z;

        this.x2 = 0;
        this.y2 = 0;
```

```
}

Point3D.prototype.transform = function(transform, focus) {
	var matrix = transform.matrix;
	var x = this.x3;
	var y = this.y3;
	var z = this.z3;

	var x3 = x * matrix[0] + y * matrix[4] + z * matrix[8]  + matrix[12];
	var y3 = x * matrix[1] + y * matrix[5] + z * matrix[9]  + matrix[13];
	var z3 = x * matrix[2] + y * matrix[6] + z * matrix[10] + matrix[14];

	this.x2 = focus * x3 / z3;
	this.y2 = focus * y3 / z3;
};

Point3D.prototype.project = function(focus) {
	this.x2 = focus * this.x3 / this.z3;
	this.y2 = focus * this.y3 / this.z3;
};

Point3D.prototype.toString = function() {
	return this.x2 + "," + this.y2;
};
```

Vertex definira metode za iscrtavanje vrhova. Definirana je metoda *draw* koja stvara nove linije koje će činiti 3D model u SVG dokumentu te definira atribute novih linija.

```
function Vertex(v1, v2) {
	this.v1    = v1;
	this.v2    = v2;
	this.node  = null;
	this.color = "black";
}

Vertex.prototype.draw = function(parent, vertices) {
	var vertex1 = vertices[this.v1];
	var vertex2 = vertices[this.v2];

	if (this.node == null) {
		var SVGDoc  = parent.ownerDocument;
		var line    = SVGDoc.createElementNS(svgns, "line");
```

```
            line.setAttributeNS(null, "stroke", this.color);

            parent.appendChild(line);
            this.node = line;
        }

        this.node.setAttributeNS(null, "x1", vertex1.x2);
        this.node.setAttributeNS(null, "y1", vertex1.y2);
        this.node.setAttributeNS(null, "x2", vertex2.x2);
        this.node.setAttributeNS(null, "y2", vertex2.y2);
};
```

Mesh3D definira metode pomoćukojih će se vrhovi i rubovi 3D modela definirati kao nizovi i koje će vraćati njihovu vrijednst. Ta vrijednost koristiti će se za iscrtavanje 3D modela. Metode su *add_vertex* koji definiraju vrhove 3D modela i vraća njihovu vrijednost, *add_edge* dodaje rubove objekta u klasu *Vertex* te *draw* koja služi za dodavanje vrijednosti vrhova u matricu transformacije te iscrtava rubove.

```
function Mesh3D(parent) {
        this.vertices = new Array();
        this.edges    = new Array();

        this.parent   = parent;

        this.z_projection = 90;
}

Mesh3D.prototype.add_vertex = function(x, y, z) {
        var length = this.vertices.length;

        this.vertices[length] = new Point3D(x, y, z);

        return length;
};

Mesh3D.prototype.add_edge = function(v1, v2) {
        this.edges[this.edges.length] = new Vertex(v1, v2);
};

Mesh3D.prototype.draw = function(matrix) {
        var vertices = this.vertices;
```

```
        var edges    = this.edges;

        for (var i = 0; i < vertices.length; i++) {
                vertices[i].transform(matrix, this.z_projection);
        }

        for (var i = 0; i < edges.length; i++) {
                edges[i].draw(this.parent, vertices);
        }
};
```

Transform prvo definira dijagonalnu matricu kao novi niz, a nakon toga dvije metode, *translate* i *rotate*. *Translate* sadrži matricu translatacije. Za matricu translatacije potrebne su 3 točke kao što je napisano u formuli 48. Kako je matrica dijagonalna, homogena koordinata imati će vrijednost 1. Točke koje se koriste u matrici definirane su u klasi *Point3D*. *Rotate* sadrži matricu rotacije. Kako bi se matrica mogla izvršavati potrebno je definirati matematičke varijable koje će se koristiti, a to su sin i cos za z, x i y točke. Nakon toga se varijable uvrštavaju u matricu. Matrica će se rotirati oko proizvoljne osi te će prema tome biti uvrštene varijable na odgovarajuće pozicije u matrici.

```
function Transform() {
        this.matrix = new Array(
                1, 0, 0, 0,
                0, 1, 0, 0,
                0, 0, 1, 0,
                0, 0, 0, 1
        );
}

Transform.prototype.translate = function(x, y, z) {
        var matrix = this.matrix;

        matrix[12] = x;
        matrix[13] = y;
        matrix[14] = z;
};

Transform.prototype.rotate = function(x, y, z) {
        var matrix = this.matrix;
```

```
        var cosx = Math.cos(x);
        var sinx = Math.sin(x);
        var cosy = Math.cos(y);
        var siny = Math.sin(y);
        var cosz = Math.cos(z);
        var sinz = Math.sin(z);

        matrix[0]  = cosy*cosz + siny*sinx*sinz;
        matrix[1]  = -cosy*sinz + siny*sinx*cosz;
        matrix[2]  = -siny*cosx;
        matrix[4]  = cosx*sinz;
        matrix[5]  = cosx*cosz;
        matrix[6]  = sinx;
        matrix[8]  = siny*cosz - cosy*sinx*sinz;
        matrix[9]  = -siny*sinz - cosy*sinx*cosz;
        matrix[10] = cosy*cosx;
};
```

3.2. SVG dokument

U ovom dokumentu se nalazi SVG model te skripta koja upravlja animacijom i pretvara 2D model u 3D model.

Dokument započinje sa XML deklaracijom i SVG DTD-om. Zatim je definirana visina i širina prikaza modela koji će iznositi 800 x 800 piksela.

Nakon toga slijede dvije skripte. Prva skripta ne nalazi se unutar SVG dokumenta nego u drugom dokumentu, a poziva se sa *xlink:href="math3d.js"* gdje je *math3d.js* naziv drugog dokumenta.

```
<script
   xlink:href="math3d.js"
   type="text/ecmascript"
   a3:scriptImplementation="Adobe"
   id="script4" />
```

Druga skripta se nalazi unutar dokumenta i pomoću nje se kreira 3D model te se upravlja njegovom animacijom. Na početku je definiran programski jezik koji se koristi, a to je JavaScript.

```
<script
  type="text/ecmascript"
  a3:scriptImplementation="Adobe"
  id="script6">
```

Nakon toga definirane su varijable koje će se koristiti. Varijable se definiraju sa ključnom riječi *var*. Prve dvije varijable su *ploca* i *objekt*. To su SVG objekti na kojima će se izvršavati skripta. Kako se 3 modela pretvaraju iz 2D u 3D modele definirana su 3 objekta. *Ploca* je zaslon na kojem će se prikazivati 3D modeli, a *objekt* je 2D model koji se transformira. Zatim su definirane matematičke varijable. *Mreza* je 3D model, *dubina* je iznos produženja u treću dimenziju 2D modela, a *promjena* je matrica transformacije. *Promjena* će biti novi objekt, ovdje će to biti 3 nova objekta, koji je napravljen prema *Transform* prototipu ili klasi koji je definiran u skripti *Math3D.js*.

```
var ploca;

var objekt1;
var objekt2;
var objekt3;

var mreza1;
var mreza2;
var mreza3;

var dubina = 10;

var promjena1 = new Transform();
var promjena2 = new Transform();
var promjena3 = new Transform();
```

Sljedeće 3 varijable služe za animaciju 3D modela. *Pokrenuto* određuje hoće li animacija prvi prvom pokretanju programa biti pokrenuta ili neće. Vrijednost varijable je *true* što znači da hoće, *rot* određuje početni kut, a *korak* koliki će se pomak napraviti sa svakim novom slikom.

```
var pokrenuto = true;
var rot = 0
var korak = 0.01
```

Zatim slijedi funkcija koja se pokreće na početku, a to je definirano sa *startup(evt)*. Prvo se treba dohvatiti sami dokument u kojem je sve smješteno, a zatim SVG elementi prema ID-u. Nakon toga se definira zaustavljanje i pokretanje animacije sa klikom miša. Potom slijedi stvaranje pretvaranje 2D modela u 3D model, te na kraju animacija unutar koje se crta i rotira 3D model. *Mreza* je novi objekt koji je napravljen prema Mesh3D prototipu.

```
function startup(evt) {
    t = evt.target;
    svgDocument = t.ownerDocument;
    ploca   = svgDocument.getElementById(&amp;quot;ploca&amp;quot;);
    objekt   = svgDocument.getElementById(&amp;quot;objekt3d&amp;quot;);

t.setAttribute(&amp;quot;onmousedown&amp;quot;,&amp;quot;running=!running;petlja()&amp;quot;);
    mreza = new Mesh3D(ploca);
    napravi_3d(objekt);
    petlja();
}
```

Funkcija za animaciju definirana je nakon toga i zove se *petlja()*. Koristi se if petlja koja osigurava da se funkcija izvršava samo ako je pokrenuta animacija. Ako je pokrenuta animacija onda će se prvo nacrtati *mreza*, a varijabla *rot* će se mijenjati za svaki korak. Nakon toga pričekati će 10 ms i ponovno pokrenuti funkciju.

```
function petlja(){
  if (!pokrenuto) return
  draw(rot);
  rot = rot + korak;
  window.setTimeout(&amp;quot;petlja()&amp;quot;,10);
}
```

Funkcija koja poziva crtanje *mreze* zove se *draw*. Prije nego što se nacrta *mreza* prvo se rotira i translatira. Na taj način se za svako pokretanje funkcije *petlja()* crta rotirana i translatirana *mreza*, a to je svakih 10 ms. Kako se trebaju nacrtati tri 3D modela za svaki je definirana translatacija i rotacija te će se svaki drugačije kretati.

```
        function draw(rot) {
            promjena1.translate(50, 50, 50);
            promjena1.rotate(0, rot, rot/2);
            mreza1.draw(promjena1);

            promjena2.translate(-50, 100, 50);
            promjena2.rotate(0, rot, rot/3);
            mreza2.draw(promjena2);
promjena1.translate(100, 100, 50);
            promjena1.rotate(0, rot, rot/4);
            mreza1.draw(promjena1);

        }
```

Funkcija koja radi transformaciju 2D modela u 3D model zove se *napravi_3d(objekt)*. Definirana je varijabla *path* koja uzima vrijednosti atributa SVG elementa kao string. To će biti vrijednosti puta koji je definiran u SVG dokumentu. Zatim je definirana varijabla *tocke* koja razdvaja string iz varijable *path* po razmacima i na taj način dobivaju se točke. Dodatno se brišu slova m i z koja se nalaze na početku i kraja puta jer nisu potrebni te nastaje polje. Potom su definirane pomoćne varijable koje će se koristiti, p i r postaju nova polja, te varijable p_tmp, r_tmp, pp, rr, p_point i r_point koje nemaju vrijednost. Nakon toga slijedi for petlja koja se izvršava sve dok je dužina polja varijable *tocke* manja od i. Prvo se dodaje dubina r točkama, dok p točkama dubina ostaje 0. Potom se razdvajaju u 3 broja i dodaju se u *mrezu* kao vrhovi koji se spajaju i dodaju u polje točaka r i p. Tada su točke povezane po dubini i potrebno ih je povezati na način na koji su bile povezane u 2D modelu, te se to izvršava sa drugom for petljom koja povezuje redom točke sve dok je dužina polja p manja od i. Dodatno se još povezuje prva i zadnja točka. Isto je ponovljeno i za druga dva modela koji se pretvaraju u 3D modele, ali su nazivi pomoćnih varijabli drugačiji.

```
        function napravi_3d(objekt) {
            var path = objekt.getAttribute(&amp;quot;d&amp;quot;);
            var tocke = path.split(&amp;quot; &amp;quot;);
            tocke = tocke.slice(1,tocke.length-1);

            var p = new Array();
            var r = new Array();
```

```
            var p_tmp; var r_tmp;
            var pp; var rr;
            var p_point; var r_point;

            for ( var i=0; i&lt;tocke.length; i++ ){
                  p_tmp = &quot;0,&quot;+dots[i];
                  r_tmp = depth+&quot;,&quot;+dots[i];
                  pp = p_tmp.split(&quot;,&quot;);
                  rr = r_tmp.split(&quot;,&quot;);
                  p_point = mreza.add_vertex( pp[0],pp[1],pp[2] );
                  r_point = mreza.add_vertex( rr[0],rr[1],rr[2] );
                  mreza.add_edge(p_point, r_point);
                  p.push(p_point);
                  r.push(r_point);
            }
.
            for ( i=0; i&lt;p.length-1; i++ ){
                  mreza.add_edge( p[i], p[i+1] );
                  mreza.add_edge( r[i], r[i+1] );
            }
            mreza.add_edge( p[0], p[i] );
            mreza.add_edge( r[0], r[i] );
         }
   </script>
```

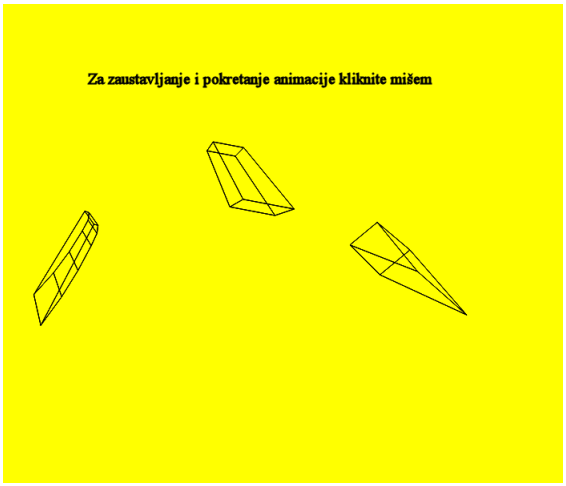
Nakon skripti definirani su SVG 2D modeli. Svaki 2D model svakako treba biti definiran kao put kako bi se mogla izvršiti skripta. Također, ne smiju biti zakrivljene linije kako bi se skripta mogla izvršiti. Zakrivljenost bi se mogla postići definiranjem većeg broja točaka unutar puta kao što je djelomično prikazan u lijevom objektu. 2D modeli trebaju imati ID *objekt3dx* prema kojem se pozivaju u skripti. Dodano je definiran i zaslon na kojem se prikazuje 3D model. Definiran je kao kvadrat i treba imati ID *ploca*. Također, na zaslonu su ispisane upute za korisnika.

```
 <g
   id="ploca"
   transform="translate(200,100)" >
  <rect
     x="-200"
     y="-100"
     width="700"
```

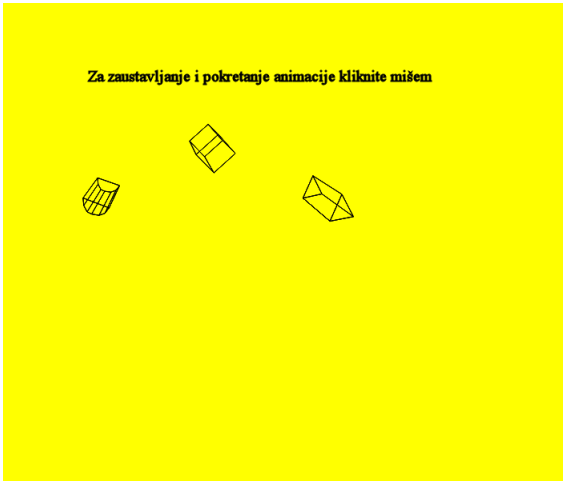
```
      height="600"
      style="fill: yellow"
      id="rect11" />
<text x="120" y="0" style="stroke:black;text-anchor:middle;font-size:15pt">Za
zaustavljanje i pokretanje animacije kliknite mišem</text>

 </g>
 <g
   transform="translate(0,0)"
   id="g13" />
 <path
   style="fill:none;fill-opacity:1;stroke:none"
   d="M 0,0 20,0 20,20 0,20 Z"
   id="objekt3d1" />
 <path
   style="fill:none;fill-opacity:1;stroke:none"
   d="M 7.5,20 5,15 5,10 7.5,5 10,2.5 15,2.5 20,5 Z"
   id="objekt3d2" />
  <path
   style="fill:none;fill-opacity:1;stroke:none"
   d="M 10,10 30,10 20,20 10,10 Z"
   id="objekt3d3" />
```

4. REZULTATI I RASPRAVA

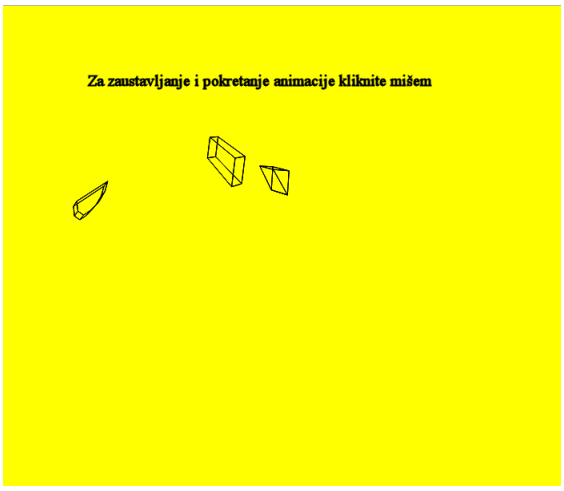
Kako bi program radio potrebno ga je pokrenuti u pretraživaču koji podržava JavaScript, a to je većina novih verzija pretraživača. Prikazati će se 3D model koji se rotira i translatira unutar kvadrata 'zaslona' koji je definiran. Moguće je mijenjati oblik 2D modela, a time i 3D modela, boju i veličinu 'zaslona' i linija 3D i 2D modela, te dubinu treće dimenzije 3D modela. Na slikama 7, 8 i 9 prikazana su tri 3D modela u različitim pozicijama.



Slika 7. Prikaz 3D modela – pozicija 1



Slika 8. Prikaz 3D modela – pozicija 2



Slika 9. Prikaz 3D modela – pozicija 3

Kod pretvaranja 2D modela u 3D model javlja se problem jer ne postoji informacija o trećoj dimenziji, pa ju je potrebno definirati. To je moguće na više načina. Odabran je način kod kojeg se povlači linija okomito na ravninu modela iz njegovih vrhova. Krajevi linija se spajaju na isti način kao i kod 2D modela. Moguće je mijenjati dužinu linija te tako mijenjati 3D model. Zbog takvog načina stvaranja 3D modela potrebno je 2D model definirati kao path. Program će prepoznati vrhove prema točkama koje su definirane u path-u. Nije moguće koristiti zakrivljene linije, ali je moguće postići prividnu zakrivljenost definiranjem većeg broja točaka. Dodana je i animacija kako bi se vidjele sve 3 dimenzije. Kod rotacije i translatacije program koristi matrice te za svaki pomak izračunava točke koje će biti prikazane na 2D zaslonu računala. Dokument je moguće otvoriti i pregledati u gotovo svakom pretraživaču.

5. ZAKLJUČAK

SVG je format za vektorsku grafiku baziran na XML-u. Prednosti SVG-a su mala veličina datoteke zbog tekstualnog zapisa, s obzirom na rasterske formate. Iz istog razloga prednost je lakše pretraživanje jer tražilice na internetu indeksiraju sadržaj putem ključnih riječi. SVG podržavaju gotovo svi preglednici. Jedna od najvećih prednosti je kvaliteta grafike. Grafika se može povećavati i smanjivati bez utjecaja na kvalitetu. Također, prednost je mogućnost prikaza teksta i rasterske grafike uz vektorsku grafiku. SVG se brzo razvija i sve više upotrebljava. Dodatna prenost SVG-a je mogućnost integracije skripti napisanih na drugim programskim jezicima čime se otvaraju dodatne mogućnosti kod kreiranja grafike. Time se omogućuje animacija i manipulacija grafikom koja nije moguća kod korištenja samog SVG-a. JavaScript je jedan od najkorištenijih programskih jezika na internetu zbog mogućnosti jednostavne integracije u HTML i XML dokumente čime se znatno povećavaju funkcionalnosti. Također, jedan od razloga je i izvršavanje na procesoru korisnika zbog čega se olakšava rad servera i povećava brzina. Također, prednost je što je podržan na svim pretraživačima. U SVG-u vektori s koriste kako bi se odredile početne i krajnje točke linija te njihov smjer. Općenito, vektori opisuju smjer i dužinu te se često koriste u računalnoj grafici. Matrice se koriste kod 3D modeliranja kako bi se lakše manipuliralo točkama u prostoru. U radu se koriste matrice transformacije za translataciju i rotaciju. Pomoću matrica olakšava se izračunavanje pomaka trodimnzionalne točke u prostoru, jer računalo za svaki pomak treba izračunati kako prikazati točku na 2D zaslonu računala. Pri tome najveći problem predstavlja ispravno prikazivanje dubine točke te pozicija točaka ovisno o perspektivi gledanja.

U eksperimentalnom dijelu napravljen je SVG dokument koji sadrži tri 2D modela. U SVG dokument dodana je skripta napisana na programskom jeziku JavaScript koja pristupa elementima 2D modela te ih transformira kako bi nastao 3D model. Nastali 3D modeli dodatno su animirani kako bi se pokazale sve tri dimenzije. Radom je pokazano kako je moguće pretvoriti 2D model u 3D model uz određena ograničenja. Kod pretvaranja 2D modela u 3D model nedostaje informacija o trećoj dimenziji te ju je potrebno odrediti. Izabrano je spuštanje okomice za određeni iznos te spajanje novonastalih vrhova na način

na koji su spojen i u 2D modelu. Zbog toga što program radi na način da traži vrhove modela nije moguće raditi transformacije na zakrivljenim tijelima, ali je to moguće postići definiranjem većeg broja točaka koji daju dojam zakrivljenosti. Dobiveni 3D model mogao bi pronaći primjenu na webu jer danas gotovo svaki pretraživač ima ugrađenu podršku za JavaScript , a i sami programski jezik je kreiran za web. SVG se može ugraditi u HTML te se na taj način model može jednostavno dodati na web stranicu.

6. LITERATURA

1. *** http://www.w3.org/XML/ - Introduction, 15.07.2014

2. Eisenberg J. D.(2002). SVG Essentials, O'Reilly, Sebastopol

3. *** http://www.w3.org/TR/SVG/intro.html – Introduction, 16.07.2014

4. Flanagan D. (2011). JavaScript: The Definitive Guide, O'Reilly, Sebastopol

5. Shirley P., Marschner S. (2009). Fundamentals of Computer Graphics, Taylor & Francis Group, Boca Raton

6. Lengyel E. (2012). Mathematics for 3D Game Programming and Computer Graphics, Cengage Learning, Boston

7. Gortler S.J., (2011). Foundations of 3D Computer Graphics, MIT Press, Cambridge

8. Rogers D. F., Adams J. A. (1989). Mathematical elements for computer graphics, McGraw Hill

9. Elezović N., Aglić A. (1995). Linearna algebra – Zbirka zadataka, Element, Zagreb

10. Bakić D. (2008). Linearna algebra, Školska knjiga, Zagreb

7. PRILOZI

Prilog 1 - Math3d.js skripta

Prilog 2 – 2Dto3D.svg skripta

Prilog 1 - Math3d.js skripta

```
/**
 * KEL_3D.js
 *
 * @author Kevin Lindsey
 * @version 1.0
 * @copyright 2000, Kevin Lindsey
 * @license http://www.kevlindev.com/license.txt
 */

/*
 * globals
 */
var svgns = "http://www.w3.org/2000/svg";

/**
 * Point3D
 *
 * @constructor
 * @param {Number} x
 * @param {Number} y
 * @param {Number} z
 */
function Point3D(x, y, z) {
	this.x3 = x;
	this.y3 = y;
	this.z3 = z;

	this.x2 = 0;
	this.y2 = 0;
}

/**
 * transform
 *
 * @param {Array} transform
 * @param {Number} focus
 */
Point3D.prototype.transform = function(transform, focus) {
	var matrix = transform.matrix;
	var x = this.x3;
	var y = this.y3;
	var z = this.z3;
```

```
        var x3 = x * matrix[0] + y * matrix[4] + z * matrix[8]  + matrix[12];
        var y3 = x * matrix[1] + y * matrix[5] + z * matrix[9]  + matrix[13];
        var z3 = x * matrix[2] + y * matrix[6] + z * matrix[10] + matrix[14];

        this.x2 = focus * x3 / z3;
        this.y2 = focus * y3 / z3;
};

/**
 * project
 *
 * @param {Number} focus
 */
Point3D.prototype.project = function(focus) {
        this.x2 = focus * this.x3 / this.z3;
        this.y2 = focus * this.y3 / this.z3;
};

/**
 * toString
 *
 * @return {String}
 */
Point3D.prototype.toString = function() {
        return this.x2 + "," + this.y2;
};


/*
 * Vertex.js
 */

/**
 * Vertex
 *
 * @constructor
 * @param {Number} v1
 * @param {Number} v2
 */
function Vertex(v1, v2) {
        this.v1    = v1;
        this.v2    = v2;
        this.node  = null;
        this.color = "black";
```

```
}

/**
 * draw
 *
 * @param {SVGElement} parent
 * @param {Array} vertices
 */
Vertex.prototype.draw = function(parent, vertices) {
	var vertex1 = vertices[this.v1];
	var vertex2 = vertices[this.v2];

	if (this.node == null) {
		var SVGDoc  = parent.ownerDocument;
		var line    = SVGDoc.createElementNS(svgns, "line");

		line.setAttributeNS(null, "stroke", this.color);

		parent.appendChild(line);
		this.node = line;
	}

	this.node.setAttributeNS(null, "x1", vertex1.x2);
	this.node.setAttributeNS(null, "y1", vertex1.y2);
	this.node.setAttributeNS(null, "x2", vertex2.x2);
	this.node.setAttributeNS(null, "y2", vertex2.y2);
};

/*
 * Mesh3D.js
 */

/**
 * Mesh3D
 *
 * @param {SVGElement} parent
 */
function Mesh3D(parent) {
	this.vertices = new Array();
	this.edges    = new Array();

	this.parent   = parent;

	this.z_projection = 90;
}
```

```
/**
 * add_vertex
 *
 * @param {Number} x
 * @param {Number} y
 * @param {Number} z
 */
Mesh3D.prototype.add_vertex = function(x, y, z) {
	var length = this.vertices.length;

	this.vertices[length] = new Point3D(x, y, z);

	return length;
};

/**
 * add_edge
 *
 * @param {Vertex} v1
 * @param {Vertex} v2
 */
Mesh3D.prototype.add_edge = function(v1, v2) {
	this.edges[this.edges.length] = new Vertex(v1, v2);
};

/**
 * draw
 *
 * @param {Array} matrix
 */
Mesh3D.prototype.draw = function(matrix) {
	var vertices = this.vertices;
	var edges    = this.edges;

	for (var i = 0; i < vertices.length; i++) {
		vertices[i].transform(matrix, this.z_projection);
	}

	for (var i = 0; i < edges.length; i++) {
		edges[i].draw(this.parent, vertices);
	}
};
```

```
/*
 * Transform.js
 */

/**
 * Transform
 *
 * @constructor
 */
function Transform() {
	this.matrix = new Array(
		1, 0, 0, 0,
		0, 1, 0, 0,
		0, 0, 1, 0,
		0, 0, 0, 1
	);
}

/**
 * translate
 *
 * @param {Number} x
 * @param {Number} y
 * @param {Number} z
 */
Transform.prototype.translate = function(x, y, z) {
	var matrix = this.matrix;

	matrix[12] = x;
	matrix[13] = y;
	matrix[14] = z;
};

/**
 * rotate
 *
 * @param {Number} x
 * @param {Number} y
 * @param {Number} z
 */
Transform.prototype.rotate = function(x, y, z) {
	var matrix = this.matrix;

	var cosx = Math.cos(x);
	var sinx = Math.sin(x);
```

```
        var cosy = Math.cos(y);
        var siny = Math.sin(y);
        var cosz = Math.cos(z);
        var sinz = Math.sin(z);

        matrix[0]  = cosy*cosz + siny*sinx*sinz;
        matrix[1]  = -cosy*sinz + siny*sinx*cosz;
        matrix[2]  = -siny*cosx;
        matrix[4]  = cosx*sinz;
        matrix[5]  = cosx*cosz;
        matrix[6]  = sinx;
        matrix[8]  = siny*cosz - cosy*sinx*sinz;
        matrix[9]  = -siny*sinz - cosy*sinx*cosz;
        matrix[10] = cosy*cosx;
};
```

Prilog 2 – 2Du3D.svg skripta

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
  xmlns:a3="http://ns.adobe.com/AdobeSVGViewerExtensions/3.0/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:cc="http://creativecommons.org/ns#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd"
  width="800"
  height="800"
  onload="startup(evt)"
  a3:scriptImplementation="Adobe"
  id="svg2"
  version="1.1"
  sodipodi:docname="2Du3D.svg" >
 <metadata
   id="metadata20" >
  <rdf:RDF>
   <cc:Work
     rdf:about="" >
    <dc:format>image/svg+xml</dc:format>
    <dc:type
      rdf:resource="http://purl.org/dc/dcmitype/StillImage" />
    <dc:title />
   </cc:Work>
  </rdf:RDF>
 </metadata>
 <defs
   id="defs18" />
 <sodipodi:namedview
   pagecolor="#ffffff"
   bordercolor="#666666"
   borderopacity="1"
   objecttolerance="10"
   gridtolerance="10"
   guidetolerance="10"
   inkscape:pageopacity="0"
   inkscape:pageshadow="2"
   inkscape:window-width="640"
   inkscape:window-height="480"
```

```
    id="namedview16"
    showgrid="false"
  />
  <script
    xlink:href="math3d.js"
    type="text/ecmascript"
    a3:scriptImplementation="Adobe"
    id="script4" />
  <script
    type="text/ecmascript"
    a3:scriptImplementation="Adobe"
    id="script6">
          var ploca;
          var objekt1;
          var objekt2;
          var objekt3;

          var mreza1;
          var mreza2;
          var mreza3;
          var dubina = 10;
          var promjena1 = new Transform();
          var promjena2 = new Transform();
          var promjena3 = new Transform();

          var pokrenuto = true;
          var rot = 0
          var korak = 0.01

       function startup(evt) {
            t = evt.target;
            svgDocument = t.ownerDocument;
            ploca = svgDocument.getElementById(&quot;ploca&quot;);
            objekt1 = svgDocument.getElementById(&quot;objekt3d1&quot;);
            objekt2 = svgDocument.getElementById(&quot;objekt3d2&quot;);
            objekt3 = svgDocument.getElementById(&quot;objekt3d3&quot;);
t.setAttribute(&quot;onmousedown&quot;,&quot;pokrenuto=!pokrenuto;petlja()&quot;);

            mreza1 = new Mesh3D(ploca);
            mreza2 = new Mesh3D(ploca);
            mreza3 = new Mesh3D(ploca);

            napravi_3d(objekt1);
            napravi_3d(objekt2);
            napravi_3d(objekt3);
```

```
    petlja();
    }

function petlja(){

    if (!pokrenuto){
          return
    }

    else
    draw(rot);
    rot = rot + korak;
    window.setTimeout(&amp;quot;petlja()&amp;quot;,,15);
}

function draw(rot) {

    promjena1.translate(50, 50, 50);
    promjena1.rotate(0, rot, rot/2);
    mreza1.draw(promjena1);

    promjena2.translate(-50, 100, 50);
    promjena2.rotate(0, rot, rot/3);
    mreza2.draw(promjena2);

    promjena3.translate(100, 100, 50);
    promjena3.rotate(0, rot, rot/4);
    mreza3.draw(promjena3);
}

function napravi_3d(objekt) {

    var path1 = objekt1.getAttribute(&amp;quot;d&amp;quot;);
    var tocke1 = path1.split(&amp;quot; &amp;quot;);
    tocke1 = tocke1.slice(1,tocke1.length-1);
    var p = new Array();
    var r = new Array();
    var p_tmp; var r_tmp;
    var pp; var rr;
    var p_point; var r_point;

    for ( var i=0; i&amp;lt;tocke1.length; i++ ){
          p_tmp = &amp;quot;0,&amp;quot;+tocke1[i];
          r_tmp = dubina+&amp;quot;,&amp;quot;+tocke1[i];
          pp = p_tmp.split(&amp;quot;,&amp;quot;);
```

```
            rr = r_tmp.split(&quot;,&quot;);
            p_point = mreza1.add_vertex( pp[0],pp[1],pp[2] );
            r_point = mreza1.add_vertex( rr[0],rr[1],rr[2] );
            mreza1.add_edge(p_point, r_point);
            p.push(p_point);
            r.push(r_point);
        }

        for ( i=0; i&lt;p.length-1; i++ ){
            mreza1.add_edge( p[i], p[i+1] );
            mreza1.add_edge( r[i], r[i+1] );
        }

        mreza1.add_edge( p[0], p[i] );
        mreza1.add_edge( r[0], r[i] );

var path2 = objekt2.getAttribute(&quot;d&quot;);
        var tocke2 = path2.split(&quot; &quot;);
        tocke2 = tocke2.slice(1,tocke2.length-1);
        var t = new Array();
        var m = new Array();
        var t_tmp; var m_tmp;
        var tt; var mm;
        var t_point; var m_point;

        for ( var i=0; i&lt;tocke2.length; i++ ){
            t_tmp = &quot;0,&quot;+tocke2[i];
            m_tmp = dubina+&quot;,&quot;+tocke2[i];
            tt = t_tmp.split(&quot;,&quot;);
            mm = m_tmp.split(&quot;,&quot;);
            t_point = mreza2.add_vertex( tt[0],tt[1],tt[2] );
            m_point = mreza2.add_vertex( mm[0],mm[1],mm[2] );

            mreza2.add_edge(t_point, m_point);

            t.push(t_point);
            m.push(m_point);
        }

        for ( i=0; i&lt;t.length-1; i++ ){
            mreza2.add_edge( t[i], t[i+1] );
            mreza2.add_edge( m[i], m[i+1] );
        }

        mreza2.add_edge( t[0], t[i] );
```

```
            mreza2.add_edge( m[0], m[i] );

            var path3 = objekt3.getAttribute(&amp;quot;d&amp;quot;);
            var tocke3 = path3.split(&amp;quot; &amp;quot;);

            tocke3 = tocke3.slice(1,tocke3.length-1);
            var n = new Array();
            var o = new Array();
            var n_tmp; var o_tmp;
            var nn; var oo;
            var n_point; var o_point;

            for ( var i=0; i&amp;lt;tocke3.length; i++ ){

                  n_tmp = &amp;quot;0,&amp;quot;+tocke3[i];
                  o_tmp = dubina+&amp;quot;,&amp;quot;+tocke3[i];
                  nn = n_tmp.split(&amp;quot;,&amp;quot;);
                  oo = o_tmp.split(&amp;quot;,&amp;quot;);
                  n_point = mreza3.add_vertex( nn[0],nn[1],nn[2] );
                  o_point = mreza3.add_vertex( oo[0],oo[1],oo[2] );
                  mreza3.add_edge(n_point, o_point);
                  n.push(n_point);
                  o.push(o_point);
            }

            for ( i=0; i&amp;lt;p.length-1; i++ ){
                  mreza3.add_edge( n[i], n[i+1] );
                  mreza3.add_edge( o[i], o[i+1] );
            }

            mreza3.add_edge( n[0], n[i] );
            mreza3.add_edge( o[0], o[i] );
         }

  </script>
<rect
   width="100%"
   height="100%"
   style="fill:white"
   id="rect8" />
<g
   id="ploca"
   transform="translate(200,100)" >
  <rect
     x="-200"
```

```
      y="-100"
      width="700"
      height="600"
      style="fill: yellow"
      id="rect11" />
<text x="120" y="0" style="stroke:black;text-anchor:middle;font-size:15pt">Za
zaustavljanje i pokretanje animacije kliknite mišem</text>

 </g>
 <g
   transform="translate(0,0)"
   id="g13" />
 <path
   style="fill:none;fill-opacity:1;stroke:none"
   d="M 0,0 20,0 20,20 0,20 Z"
   id="objekt3d1" />
 <path
   style="fill:none;fill-opacity:1;stroke:none"
   d="M 7.5,20 5,15 5,10 7.5,5 10,2.5 15,2.5 20,5 Z"
   id="objekt3d2" />
  <path
   style="fill:none;fill-opacity:1;stroke:none"
   d="M 10,10 30,10 20,20 10,10 Z"
   id="objekt3d3" />
</svg>
```