SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1852

Vizualna analitika primijenjena na grupiranje velikih skupova podataka

Anteo Ivankov

Zagreb, Lipanj 2019.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 7. ožujka 2019.

DIPLOMSKI ZADATAK br. 1852

Pristupnik: **Anteo Ivankov (0036485518)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Vizualna analitika primijenjena na grupiranje velikih skupova podataka**

Opis zadatka:

U sustavima u kojima postoji veliki broj računalnih čvorova i veliki broj razmijenjenih podataka, postoji izrazita potreba za skaliranim prikazom čvorova i poruka u tri prostorne dimenzije. Nasuprot klasičnim metodama vizualizacije, potrebno je iskoristiti prostornu dimenziju te preglednije prikazati komunikaciju između čvorova u kontekstu vizualnog grupiranja podataka, izražavanja velikog broja slijednih poruka te analizi povezanosti čvorova u mrežnoj topologiji.
Izraditi odgovarajući programski proizvod. Koristiti WebGL tehnologiju 3D prikaza te three.js biblioteku. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Zadatak uručen pristupniku: 15. ožujka 2019.
Rok za predaju rada: 28. lipnja 2019.

Mentor:
Prof. dr. sc. Željka Mihajlović

Djelovođa:
Izv. prof. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:
Doc. dr. sc. Marko Čupić

Sadržaj

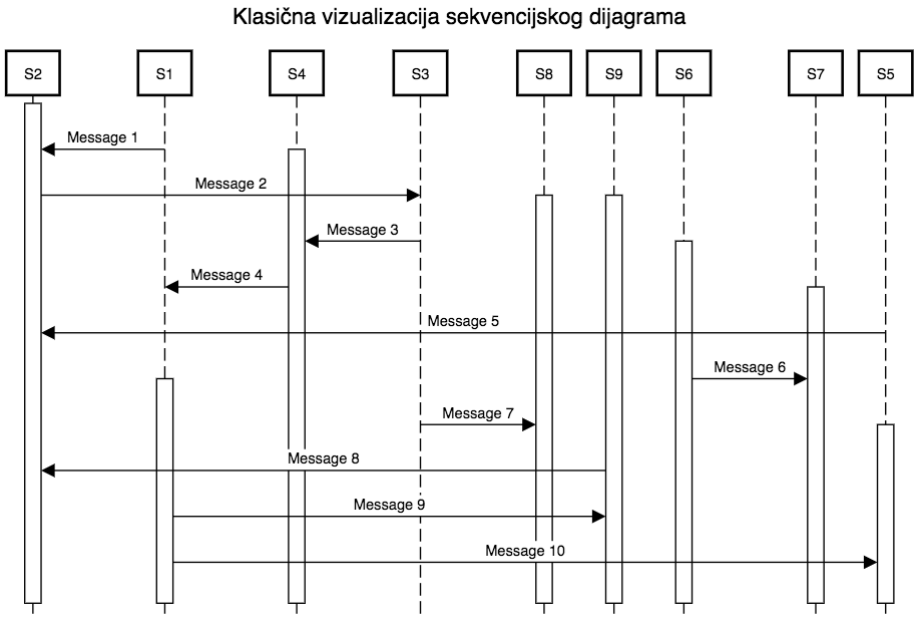
Popis tablica

Popis slika

1. Uvod

U sustavima u kojima postoji velik broj računalnih čvorova i velik broj razmijenjenih podataka postoji potreba za drukčijim metodama vizualizacije komunikacije naspram klasičnih.

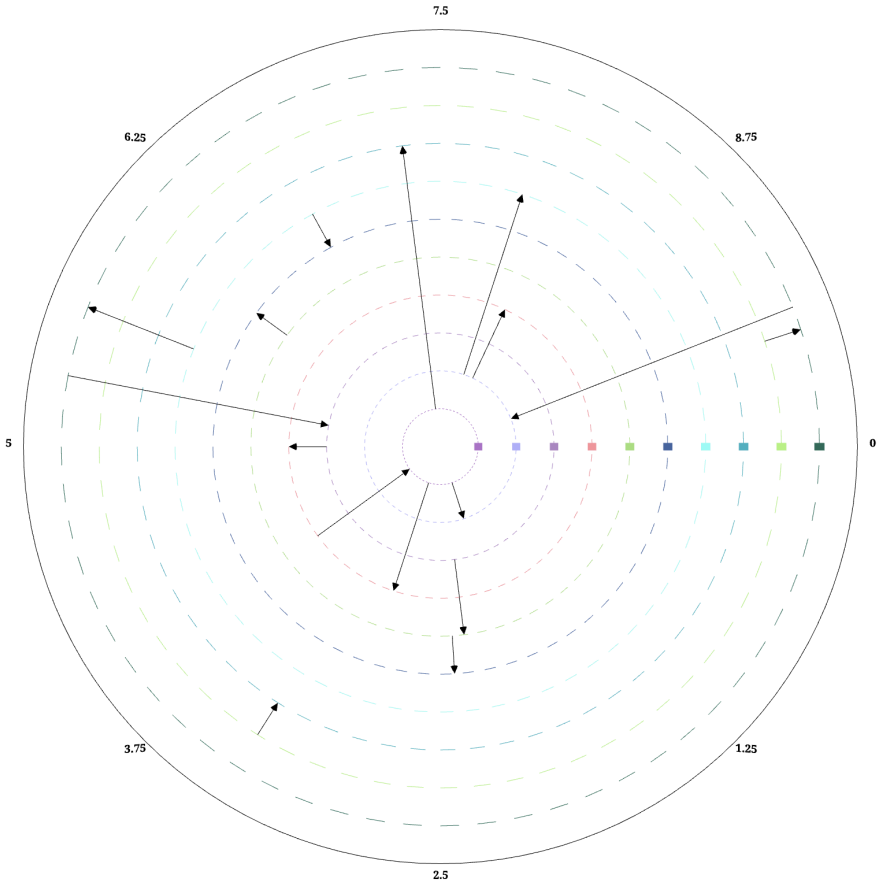
Primjerice, kod klasičnog prikaza komunikacije korištenjem sekvencijskog dijagrama, problem nastaje kada broj sudionika u komunikaciji postane velik. U takvom slučaju da bi se vidjela sva komunikacija potrebno je pomicati pogled prema desno kako bi se vidjeli svi sudionici komunikacije te prema dolje kako bi se vidjela čitava komunikacija u vremenu što zna biti nespretno i nepregledno. Slika 1 prikazuje klasični sekvencijski dijagram na kojoj je prikazana problematika vizualizacije velike količine podataka na dijagramu.



Slika 1. Klasični sekvencijski dijagram

Zbog prikazanog problema, u radu je predložena i opisana drukčija vizualizacija u kojoj vrijeme nije prikazano od vrha prema dnu, a sudionici komunikacije s lijeva na desno, već je vrijeme prikazano kružno te je svaki sudionik u komunikaciji prikazan kao jedna koncentrična kružnica. Tom

idejom pokušava se ukloniti problem u kojem cijeli dijagram nije odmah vidljiv na ekranu te omogućiti bolju preglednost. Također, pokušava se ostvariti cilj da je na prvi pogled moguće shvatiti o kakvoj se komunikaciji radi. Slika 2 prikazuje opisani izgled jednog takvog dijagrama.



Slika 2. Alternativna vizualizacija sekvencijskog dijagrama

2. Korištene tehnologije i alati

2.1. JavaScript

JavaScript je skriptni programski jezik koji se izvršava u web pregledniku te je implementacija ECMAScript specifikacije. Neka od svojstava programskog jezika JavaScript su dinamičko tipiziranje, objektno orijentirana paradigma temeljena na prototipovima, funkcije prvog reda itd. Zajedno sa prezentacijskim jezikom za izradu web stranica HTML te stilskim obrascima CSS za oblikovanje HTML dokumenata, JavaScript je glavna tehnologija koja pokreće World Wide Web. JavaScript omogućuje interaktivnost web aplikacija te svi poznatiji web preglednici sadrže JavaScript pokretač (engl. engine) koji je zadužen za njegovo izvođenje. Inicijalno je JavaScript bio implementiran jedino u web preglednicima te se izvršavao isključivo na strani klijenta. Danas su JavaScript pokretači prisutni unutar brojnih programa pa se tako jezik može koristiti i za implementaciju poslužitelja kao i u programima koji nisu povezani s web tehnologijom primjerice u okruženjima u kojima se JavaScript koristi za izradu mobilnih i desktop aplikacija.

2.2. HTML i DOM

HTML je kratica za Hypertext Markup Language, jezik kojim se opisuje sadržaj koji se prezentira web preglednicima. Trenutačno najnovija inačica je HTML5. HTML5 je uveo mnogo novina u sam jezik. Primjerice dodana je podrška za element *<canvas>* koji podržava vektorsku grafiku što je omogućilo brže izvođenje grafički zahtjevnijih aplikacija unutar web preglednika. Podržan je u svim korištenijim web preglednicima kao što su Google Chrome, Microsoft Edge, Firefox, Safari, Opera i drugi.

DOM je kratica za Document Object Model. To je aplikacijsko programsko sučelje za HTML dokumente. DOM definira način na koji je moguće pristupiti HTML elementima te način na koji se oni mijenjaju. Glavna mu je uloga pružiti standardizirano programsko sučelje te je neovisan o programskom jeziku.

2.3. Three.js biblioteka i WebGL

Three.js je JavaScript biblioteka koja ima podršku za različite web preglednike, a koristi se za izradu 2D i 3D interaktivne računalne grafike. Biblioteka je bazirana na tehnologiji WebGL te omogućava lakše rukovanje nego li izravno koristeći WebGL. WebGL je aplikacijsko programsko sučelje (engl. Application Programming interface API) koje se koristi za izradu grafičkih programa te podržava izvođenje programa na grafičkoj kartici koristeći OpenGL ES programski jezik za sjenčanje (engl. shading language). WebGL je integriran sa ostalim web standardima. To znači da se može koristiti za iscrtavanje na HTML element <*canvas*> te se može koristiti zajedno s ostalim HTML elementima prilikom izrade web sadržaja.

2.4. Pokretač Node.js i npm

NodeJS je JavaScript pokretač koji izvodi JavaScript kod izvan web preglednika. NodeJS omogućava korištenje JavaScript programskog jezika u okruženju za koje nije inicijalno bio namjenjen primjerice u komandnoj liniji ili izvođenje skripti na poslužiteljskoj strani kako bi se dobila dinamička web stranica prije nego je ona poslana klijentu. Programski paket npm je kratica za Node Package Manager. To je program za upravljanje paketima koji se instalira zajedno s programom Node.js.

3. Opis implementacije

3.1. Postavljanje razvojnog okruženja

Budući da JavaScript ima sigurnosno ograničenje nazvano pravilo istog podrijetla (engl. same-origin policy) koje onemogućuje učitavanje eksternih datoteka unutar JavaScript koda, te kako Three.js biblioteka zahtjeva učitavanje geometrije, tekstura i ostalih datoteka, potrebno je postaviti lokalnog poslužitelja kako bi sve datoteke dolazile do klijenta sa istog mjesta. U tu svrhu je korišten npm paket http-server jer on omogućava jednostavno stvaranje lokalnog poslužitelja putem komandne linije te je zato idealan za testiranje i lokalni razvoj. instalira se preko npm-a korištenjem komandne linije naredbom *npm install http-server -g*. Svi paketi koji se instaliraju preko npm-a automatski se dodaju u jedan direktorij nazvan *node_modules*. U datoteci *package.json* su izlistani svi paketi koje program koristi. Na taj način se jednostavno upravlja paketima. Još jedna prednost npm-a je što je za svaki program samo potrebno pokrenuti naredbu *npm init* te će npm automatski instalirati sve pakete koje program koristi. Nakon što je instaliran paket *http-server* potrebno je pokrenti naredbu u komandnoj liniji: *http-server ./app -a localhost -p 8000*. Tom naredbom podižemo lokalni poslužitelj na vratima 8000 te sve datoteke koje dohvaćamo se trebaju nalaziti u trenutnom direktoriju u *app* folderu. Na ovaj način je napravljeno sve da bi se moglo početi raditi s Three.js bibliotekom.

3.2. Datoteka index.html

Pristupanjem lokalnom poslužitelju na adresi *http://localhost:8000* dohvaća se *index.html* datoteka. sadržaj te datoteke prikazan je u nastavku.

```
<html>
    <head>
        <title>
            Sequence Diagram
        </title>
    </head>
</html>

<body>
    <canvas id="canvas"></canvas>
    <script src="./js/lib/three.min.js"></script>
    <script type="text/javascript" src="./js/lib/dat.gui.min.js"></
script>
    <script src="./js/lib/OrbitControls.js"></script>
   <script src="./js/components/Text.js"></script>
    <script src="./js/components/Cube.js"></script>
    <script src="./js/utils/utils.js"></script>
    <script src="./js/components/Message.js"></script>
    <script src="./js/components/Server.js"></script>
    <script src="./js/components/SequenceDiagram.js"></script>
    <script src="./js/components/Arrow.js"></script>
    <script src="./js/components/Circle.js"></script>
    <script src="./js/SceneManager.js"></script>
    <script src="./js/app.js"></script>
</body>
```

Ispis programskog koda 1

<*html*> element daje do znanja web pregledniku da se radi o HTML dokumentu. To je uvijek vršni element unutar dokumenta te su u njemu sadržani svi ostali elementi.

<*head*> element sadrži metapodatke te služi kao mjesto u koji se smještaju ostali elementi.

<*title*> element definira naslov dokumenta primjerice u tab-u web preglednika.

<*body*> element definira tijelo dokumenta te sadrži sve ostale elemente poput teksta, slike, tablice, liste i drugih.

Unutar <*body*> elementa nalazi se element <*canvas*> te više <*script*> elemenata. <*canvas*> sadrži i atribut *id* preko kojeg iz JavaScript koda referenciramo upravo taj element. To je potrebno budući da se cijelo crtanje u Three.js biblioteci događa unutar <*canvas*> elementa pa je potrebno imati referencu na taj element. Također, budući da je nespretno cijeli kod pisati u jednoj datoteci, onda se u <*body*> elementu svaka pojedinačna datoteka sa programskim kodom mora učitati preko <*script*> elementa koji sadrži atribut *src* koji pokazuje na lokaciju datoteke koja se učitava. Nakon što se HTML dokument dohvatio sa poslužitelja, pokreće se izvođenje koda koje je definirano u <*script*> elementima. Izvođenje se izvršava onim redom kojim su <*script*> elementi poredani pa je zato važno i kojim se redom skripte učitavaju. Iz tog razloga sve datoteke koje sadrže kod vezan za biblioteke kao i definicije funkcija i struktura podataka moraju biti ispred kako bi onda ostali kod mogao koristiti te podatke.

3.3. Datoteka app.js

datoteka *app.js* je ulazna točka u program koja inicijalizira sve potrebne varijable i strukture podataka. isječak koda iz datoteke *app.js* prikazan je u nastavku.

```
function init() {
    canvas = document.getElementById("canvas");
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
    sceneManager = new SceneManager(canvas);
    addEventListeners();
    const sequenceDiagram = new SequenceDiagram(generateMessages(),
      0, 10);
    sceneManager.addSceneObject(sequenceDiagram) };
```

Ispis programskog koda 2

Funkcija *init* dohvaća element <*canvas*> preko njegovog *id* atributa te se postavljaju širina i visina u ovisnosti o veličini prozora u web pregledniku. Potom se učitava objekt razreda *SceneManager*. Instanca tog razreda zadužena je za inicijalizaciju Three.js scene, kamere i ostalog. Na kraju *init* funkcije se stvara objekt razreda *SequenceDiagram* koji se dodaje u scenu.

3.4. Razred SceneManager

Instanca razreda *SceneManager* zadužena je za inicijalizaciju Three.js scene, kamere te svega ostalog potrebnog kako bi se scena pravilno iscrtala. Osim toga razred implementira sve funkcionalnosti za manipulaciju događajima te sadrži sve objekte koji se trebaju iscrtati u sceni. Konstruktor razreda prikazan je u nastavku.

```
constructor(canvas) {
  this.screenDimensions = {
       width: canvas.width,
       height: canvas.height
   }
   this.mouse = new THREE.Vector2(0, 0);
   this.canvas = canvas;
   this.scene = this.createScene();
   this.renderer = this.createRenderer(this.screenDimensions);
   this.camera = this.createCamera(this.screenDimensions);
   this.sceneObjects = this.createSceneObjects(this.scene);
}
```

Ispis programskog koda 3

Konstruktor prima referencu na instancu HTML elementa <*canvas*>. Na taj način razred *SceneManager* ne mora brinuti o manipulaciji DOM elementima. U konstruktoru se inicijalizira pozicija kursora, scena, način iscrtavanja i kamera.

3.4.1. Inicijalizacija kamere

Three.js biblioteka sadrži implementacije dviju vrsta kamera: perspektivnu i ortografsku. Inicijalizacija perspektivne kamere vrši se naredbom:

```
const camera = new THREE.PerspectiveCamera(fieldOfView, aspectRatio, nearPlane, farPlane);
```

Ispis programskog koda 4

Prvi parametar predstavlja opseg vidljivog svijeta, ovisno o rezoluciji. Drugi parametar definira omjer visine i širine prozora u kojem se prikazuje scena. Treći i četvrti parametri definiraju bližu i dalju odsijecajuću ravninu. Zajedno su ova četiri podatka dovoljna za definiranje krnje piramide pogleda (engl. frustum) te je time određeno da su samo oni objekti koji se nalaze unutar piramide vidljivi.

3.4.2. Iscrtavanje

Three.js podržava više načina iscrtavanja scene. Najčešće korišteni je *WebGLRenderer*. Inicijalizacija se vrši naredbom:

```
const renderer = new THREE.WebGLRenderer({canvas: canvas});
```

Ispis programskog koda 5

Konstruktoru se predaje objekt koji ima definiran *canvas* atribut. Na taj način se scena iscrtava upravo na tom predanom platnu. Razred *SceneManager* zadužen je i za iscrtavanje scene koje se vrši naredbom:

```
renderer.render(scene, camera);
```

Ispis programskog koda 6

Naredbom se automatski čiste svi spremnici (engl. buffers) vezani za iscrtavanje. Predani argument scena iscrtava se koristeći drugi predani argument kameru.

3.4.3. Događaji

Još jedna uloga razreda *SceneManager* je manipuliranje događajima. Ideja je da se u tom razredu nalaze sve metode koje upravljaju događajima kako bi se sve nalazile na istom mjestu. Potrebne metode se tako definiraju unutar razreda te se automatski pozivaju svaki put kada je to potrebno to jest onda kada je određeni događaj nastupio. Kada se HTML dokument učita u web preglednik moguće mu je pristupiti preko *document* objekta. Objekt *document* predstavlja vršni čvor HTML dokumenta. Taj objekt sadrži metodu *addEventListener(type, listener)* gdje prvi parametar označava vrstu nekog događaja, a *listener* je funkcija koja se pri tom događaju poziva. Primjer kako je moguće definirati reakciju na događaj pomaka miša dan je u nastavku.

```
document.addEventListener('mousemove', (event) => {
    sceneManager.onMouseMove(event);
});
```

Ispis programskog koda 7

3.5. Komponente

Svaki objekt koji se iscrtava u sceni modeliran je kao jedna komponenta. To znači da svaki objekt kao svoj dio može sadržavati neku drugu komponentu. Primjerice, komponenta sekvencijskog dijagrama sadrži komponentu kružnice, strelice i slično. Na taj način dobivamo modularan kod u kojem je jednostavno zamijeniti određenu komponentu nekom drugom ili izmjeniti postojeću. Sve komponente smještene su u direktoriju naziva

components. Najmanje što svaka komponenta sadrži je geometrija (engl. geometry), materijal (engl. material) i mreža poligona (engl. mesh).

Primjer definiranja komponente *Circle* koja se koristi pri iscrtavanju sekvencijskog dijagrama dan je u nastavku.

```
class Circle {
    constructor(radius, identifier, color=0x0000ff) {
        this.radius = radius;
        this.identifier = identifier;
        this.geometry = new THREE.CircleGeometry(radius, 128);
        this.geometry.vertices.shift();
        this.material = new THREE.LineBasicMaterial({color: color});
        this.mesh = new THREE.LineLoop(this.geometry,
        this.material);
}
```

Ispis programskog koda 8

Komponenta u konstruktoru prima radijus, identifikator i boju. U konstruktoru se inicijalizira geometrija kružnice s definiranim radijusom te se kao materijal definira objekt instance *LineBasicMaterial* (jer na kružnicu neće utjecati osvjetljenje). Na kraju se kao mrežu poligona definira instanca razreda *LineLoop* koja spaja sve prethodno generirane točke geometrije. Bitno je naglastiti da svaka komponenta mora sadržavati atribut *mesh* jer se pretpostavlja da se prilikom dodavanja objekata u scenu koristi upravo atribut tog imena.

3.5.1. Bazni razred Geometrija

Geometry je bazni razred koji se koristi za spremanje vrhova (engl. vertices). instance te klase čuvaju informacije o vrhovima kao što su pozicija,

boja, strana (engl. face) i slično. Primjer korištenja *Geometry* klase koji stvara novu geometriju na osnovu tri vrha prikazan je u nastavku:

```
var geometry = new THREE.Geometry();
geometry.vertices.push(
      new THREE.Vector3( -10,  10, 0 ),
      new THREE.Vector3( -10, -10, 0 ),
      new THREE.Vector3(  10, -10, 0 ));
```

Ispis programskog koda 9

Ostala svojstva uključuju identifikaciju, naziv te svojstva koja služe za obavještavanje o promjeni podataka kao što su *verticesNeedsUpdate*, *colorsNeedsUpdate*, *normalsNeedsUpdate* i druge zastavice.

3.5.2. Bazni razred Materijali

Materials je apstraktni bazni razred kojim se definira izgled objekata. Materijali su definirani na način da su neovisni o načinu iscrtavanja. Neki od konkretnih razreda koji nasljeđuju razred *Materials* su: *LineBasicMaterial* i *MeshLambertMaterial*. *LineBasicMaterial* razred koristi se za iscrtavanje žične mreže (engl. wireframe) geometrije. Na ovaj materijal ne utječe osvjetljenje već se samo definira boja materijala.

```
const material = new THREE.LineBasicMaterial({color: 0xff0000});
```

Ispis programskog koda 10

MeshLambertMaterial razred koristi se za iscrtavanje površina koje nemaju sjaj to jest nemaju zrcalnu komponentu. Materijal koristi Lambertov model za izračun osvjetljenja. Pogodan je za simulaciju površina poput drva ili kamena, ali ne može simulirati sjajne predmete.

3.5.3. Mreža poligona

Mesh je razred koji reprezentira mrežu poligona. Instanca razreda *Mesh* se inicijalizira s dva argumenta od kojih je prvi geometrija objekta, a drugi materijal.

```
const mesh = new THREE.Mesh(geometry, material);
```

Ispis programskog koda 11

Nakon što se objekt razreda *Mesh* inicijalizira sve što je potrebno napraviti je dodati objekt u scenu. To se radi pomoću metode *add* razreda *Scene*.

```
scene.add(mesh);
```

Ispis programskog koda 12

3.6. Implementacija sekvencijskog dijagrama

3.6.1. Struktura i prikaz poruka

Svaka poruka opisana je sa četiri podatka: identifikator računala koji šalje poruku, identifikator računala koji prima poruku, vrijeme slanja poruke i tip poruke. Prva dva podatka su proizvoljni nizovi znakova, vrijeme je decimalni broj, a tip poruke je protokol kojim se poruka šalje primjerice TCP (engl. Transmission Control Protocol ili UDP (engl. User Datagram Protocol). Primjer stvaranja poruke:

```
const message = new Message("name1", "name2", 2, “tcp”);
```

Ispis programskog koda 13

Poruka je na dijagramu reprezentirana sa strelicom. Strelica počinje na kružnici koja predstavlja poslužitelja koji šalje poruku, a završava u kružnici koja predstavlja poslužitelja koji prima poruku. Primjer stvaranja komponente strelice:

```
constructor(startPoint, endPoint, color = 0xff0000) {
    const length = startPoint.distanceTo(endPoint);
    const direction = new THREE.Vector3();
    direction.subVectors(endPoint, startPoint).normalize();
    this.mesh = new THREE.ArrowHelper(direction, startPoint,
    length, color, 1, 1);
}
```

Ispis programskog koda 14

Konstruktor razreda strelice prima početnu i krajnju točku te boju. Početna i krajnja točka su instance razreda *Vector3*. U konstruktoru se na početku izračunava udaljenost između početne i krajnje točke kao i jedinični vektor smjera. Na kraju se konstruira mreža poligona preko pomoćnog razreda *ArrowHelper*. Taj razred čini mrežu poligona strelice, a strelica je implementirana kao kombinacija linije i stošca.

3.6.2. Font i tekst komponenta

Budući da je na dijagramu potrebno koristiti tekst na nekoliko mjesta, prvo je potrebno učitati font. Biblioteka Three.js sadrži nekoliko predefiniranih fontova koje je moguće koristiti. Učitavanje fonta obavlja se naredbama:

```
const loader = new THREE.FontLoader();
  loader.load('https://threejs.org/examples/fonts/droid/
droid_serif_bold.typeface.json', (res) => {
      font = res;
  });
```

Ispis programskog koda 15

Prvom naredbom se inicijalizira objekt tipa *FontLoader* koji omogućava učitavanje fonta u JSON formatu. JSON je kratica od JavaScript Object Notation. To je datoteka čiji je format razumljiv ljudima, a sastoji se od parova atribut-vrijednost te polja. To je najčešće korišteni format koji se koristi za asinkronu komunikaciju između poslužitelja i web preglednika te je neovisan o programskom jeziku. Nakon toga se učitava font pomoću metode *load*. Metoda prima kao prvi parametar URL koji pokazuje na datoteku koja sadrži font, a drugi parametar je funkcija koja sprema učitani font u varijablu. Ta funkcija se poziva asinkrono kada učitavanje fonta završi. Na taj način je implementirano asinkrono izvođenje jer program ne treba čekati na učitavanje fonta. To je dobro jer učitavanje može biti dugotrajna operacija.

Nakon učitavanja fonta potrebno ga je iscrtati na ekran. Za to je zadužena komponenta *Text*. Konstruktor tog razreda je:

```
constructor(text, textGeometryProperties, color = 0xff0000) {
   this.text = text;
   this.textGeometry = new THREE.TextGeometry(text,
   textGeometryProperties);
   this.textGeometry.computeBoundingBox();
   this.basicMaterial = new THREE.MeshBasicMaterial({color:
   color});
   this.mesh = new THREE.Mesh(this.textGeometry,
   this.basicMaterial);
}
```

Ispis programskog koda 16

Konstruktor prima niz znakova kao prvi parametar, objekt s atributima koji opisuju kako će se tekst iscrtati na ekranu i boju. atributi opisuju koji font će se koristiti, veličinu fonta, debljinu fonta i slično. U konstruktoru se potom inicijalizira objekt razreda *TextGeometry* kojim se stvara geometrija na osnovu niza znakova i parametara te se izračunava omeđujući volumen dobivene geometrije. Na kraju se stvara osnovni materijal i mreža poligona.

3.6.3. Rotacija teksta prema kameri

Budući da kamera u sceni može promijeniti poziciju ili rotaciju, potrebno je omogućiti da tekst uvijek bude orijentiran prema kameri. Kako bi se to ostvarilo potrebno je prilikom svakog iscrtavanja promijeniti rotaciju teksta na način da tekst bude orijentiran prema kameri. Iz tog razloga komponenta sekvencijskog dijagrama sadrži metodu *update*. Tu metodu poziva instanca razreda *SceneManager* koja upravlja scenom. Instanca razreda *SceneManager* prilikom svakog ponovnog iscrtavanja scene poziva metodu *update* nad svim objektima koje iscrtava te im šalje objekt kamere.

```
update() {
   for(let i = 0; i < this.sceneObjects.length; i++) {
        this.sceneObjects[i].update(camera);
 }
```

Ispis programskog koda 17

Sada će se prilikom svakog ponovnog iscrtavanja scene, pozvati metoda *update* sekvencijskog dijagrama kojeg iscrtavamo. U toj metodi je onda potrebno promijeniti rotaciju svih komponenti teksta prema kameri.

```
update(elapsedTime, camera) {
        this.textComponents.forEach( (textComponent) => {
        textComponent.mesh.lookAt(camera.position.x,
        camera.position.y, camera.position.z);
        });
}
```

Ispis programskog koda 18

Promjena rotacije teksta obavlja se pozivom metode *lookAt* nad mrežom poligona koju se želi rotirati. Metoda kao parametar prima x, y i z koordinatu točke prema kojoj se treba zarotirati.

3.7. Razred SequenceDiagram

Ovaj razred predstavlja komponentu sekvencijskog dijagrama. Uloga ovog razreda je da izgradi sekvencijski dijagram na osnovu ulaznih podataka te da omogući jednostavno aplikacijsko programsko sučelje za modificiranje dijagrama. Konstruktor razreda je:

```
constructor(messages, startTime, endTime)
```

Ispis programskog koda 19

Prvi parametar predstavlja niz poruka dok drugi i treći parametar definiraju vremenski interval koji se promatra. Poruke izvan danog vremenskog intervala se ne uzimaju u obzir. Konstruktor na osnovu niza poruka stvara strukturu podataka mapu. Kod koji to radi prikazan je u nastavku.

```
messages.forEach((msg) => {
            if(!this.serverMessages.has(msg.NAME_src)) {
                const server = new Server(msg.NAME_src);
                server.addMessage(msg);
                this.serverMessages.set(msg.NAME_src, server);
          } else {
                const server =
                this.serverMessages.get(msg.NAME_src);
                server.addMessage(msg);
            }
            if(!this.serverMessages.has(msg.NAME_dest)) {
                const server = new Server(msg.NAME_dest);
                this.serverMessages.set(msg.NAME_dest, server);
            }
        });
```

Ispis programskog koda 20

Mapa ima onoliko elemenata koliko je bilo različitih imena poslužitelja koji komuniciraju. Svaki element mape sadrži i niz poruka koji poslužitelj šalje drugima. Na ovaj način je dobivena struktura podataka iz koje je lakše konstruirati dijagram jer iz samog niza poruka nije jasno koliko poslužitelja ukupno komunicira. Sljedeći korak u izgradnji dijagrama je iscrtavanje kružnica i strelica. Za svaki element rječnika potrebno je nacrtati jednu kružnicu.

```
this.serverMessages.forEach((server, key) => {
  const circle = new Circle(radius, server.name);
  this.mesh.add(circle.mesh);
  const serverTextName = new Text(key, this.textGeometryProperties);
  serverTextName.setPosition(0, radius - 2, 0);
  this.mesh.add(serverTextName.mesh);
  server.sentMessages.forEach((msg) => {
    const startRadius = this.serverRadiuses.get(msg.NAME_src);
    const endRadius =  this.serverRadiuses.get(msg.NAME_dest);
    const startPoint = new THREE.Vector3(startRadius, 0, 0);
    const endPoint = new THREE.Vector3(endRadius, 0, 0);
    const axis = new THREE.Vector3(0, 0, -1);
    const angle = getAngleFromTimestamp(1, 10, msg.timestamp);
    const angleInRadians = THREE.Math.degToRad(angle);
    startPoint.applyAxisAngle(axis, angleInRadians);
    endPoint.applyAxisAngle(axis, angleInRadians);
    const arrow = new Arrow(startPoint, endPoint);
    this.mesh.add(arrow.mesh);
   });
});
```

Ispis programskog koda 21

Prikazani programski kod iterira po svakom elementu mape te za svakog instancira novu komponentu kružnice. Također se instancira i pripadajuća tekst komponenta koja prikazuje identifikator poslužitelja. Mreže poligona teksta i kružnice dodaju se kao djeca mreži poligona sekvencijskog dijagrama. Zatim se za svaku poruku koju je trenutni poslužitelj poslao

instancira jedna strelica. To se radi na način da se prvo odrede radijusi kružnica koje strelica povezuje, zatim se uzmu točke na kružnici koje odgovaraju vremenskom trenutku u kojem je poruka poslana. Točke na kružnici određuje pomoćna funkcija *getAngleFromTimestamp* koja prima početni i krajnji trenutak promatranja te trenutak u kojem je poruka poslana. Funkcija vraća kut u stupnjevima za koliko je dani trenutak udaljen u odnosu na početni.

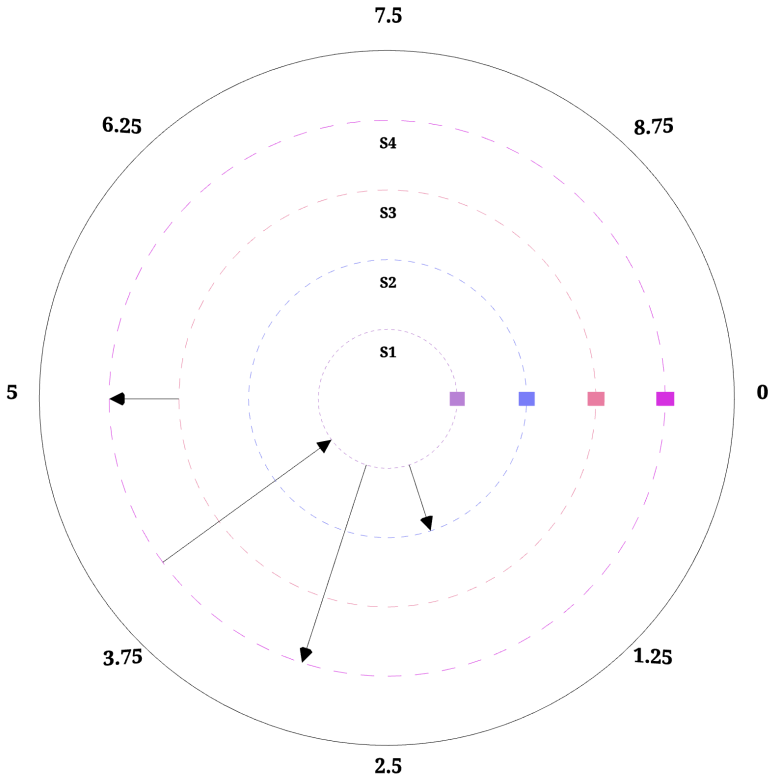
```
function getAngleFromTimestamp(timeStart, timeEnd, timestamp) {
    const interval = timeEnd - timeStart;
    const angle = ((timestamp - timeStart) / interval) * 360.0;
    return angle;
}
```

Ispis programskog koda 22

Na slici 3 je prikazan primjer dijagrama kojeg generira ovaj razred kad mu se u konstruktoru preda niz poruka sa vrijednostima prikazanim u tablici 1. te promatranim vremenskim intervalom od nulte do desete jedinice vremena.

Tablica 1. Primjer detalja poruka.

	Izvorište	Odredište	Vrijeme
Poruka 1	S1	S2	2
Poruka 2	S1	S4	3
Poruka 3	S4	S1	4
Poruka 4	S3	S4	5

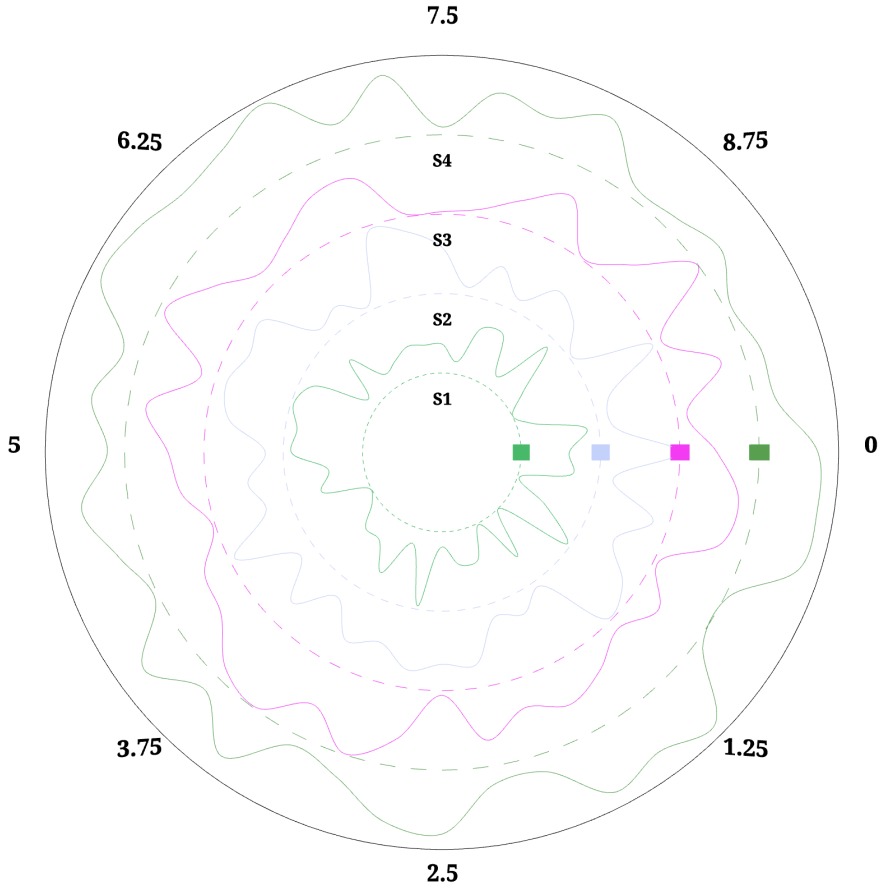


Slika 3. Sekvencijski dijagram

Iz tablice 1 vidljivo je da u komunikaciji sveukupno sudjeluju četiri poslužitelja. Zbog toga dijagram na slici 3 sadrži četiri iscrtkane kružnice gdje svaka predstavlja jednog poslužitelja. Nadalje, tablica sadrži četiri poruke pa su zbog toga na dijagramu vidljive i četiri strelice. Zadnja puna kružnica predstavlja rub dijagrama na kojem su iscrtane oznake vremena svakih 45 stupnjeva kako bi bilo lakše vidjeti u kojem trenutku je svaka poruka poslana.

3.7.1. Prikaz procesorske aktivnosti na dijagramu

Osim prikaza komunikacije između poslužitelja, na dijagramu je prikladno prikazati i neke druge parametre poslužitelja koji se mijenjaju tijekom vremena, a koji mogu biti korisni za različite primjene. Primjerice, na dijagramu se može prikazati aktivnost procesora svakog poslužitelja. Razred *SequenceDiagram* tako na osnovu danih podataka također generira i graf koji prikazuje aktivnost procesora. Primjer te vizualizacije prikazan je na slici 4.



Slika 4. Dijagram s prikazom aktivnosti procesora.

Slika 4 prikazuje procesorsku aktivnost za četiri poslužitelja na način da se crta krivulja između dvije kružnice. Krivulja uvijek prikazuje aktivnost procesora za poslužitelja koji je predstavljen manjom od dvije kružnice koje omeđuju krivulju. Graf se percipira na način da što je krivulja bliža kružnici s manjim radijusom to je postotak aktivnosti manji, a što je bliža kružnici s većim radijusom to je postotak aktivnosti procesora u promatranom trenutku veći.

Razred *SequenceDiagram* prima podatke u obliku liste kako bi iscrtao dijagram procesorske aktivnosti. Svaki element liste je podatak u JSON obliku koji sadrži identifikator poslužitelja te listu vrijednosti koje predstavljaju postotak aktivnosti procesora u određenom trenutku. Svaki podatak u listi vrijednosti se onda može transformirati u koordinatu točke na grafu. Budući da su podaci diskretni te postotak aktivnosti procesora ne mora biti definiran

u svakom trenutku, potrebno je povezati dane točke kako bi se dobio cjeloviti graf. To je moguće obaviti na više načina primjerice povezivanjem točaka s pravcem ili krivuljom. U radu je korištena centripetalna Catmull-Rom krivulja iz razloga što takva krivulja ne stvara presjeke unutar jednog segmenta krivulje te zato što prolazi kroz sve kontrolne točke krivulje. Za iscrtavanje krivulje zadužena je komponenta *Spline*. Komponenta *Spline* generira točke krivulje na sljedeći način:

```
constructor(points, color) {
        const curve = new THREE.CatmullRomCurve3(points);
        curve.closed = true;
        curve.curveType = "centripetal"
        const p = curve.getPoints(100);
        this.geometry = new THREE.BufferGeometry().setFromPoints(p);
        this.material = new THREE.LineBasicMaterial({color: color});
        this.mesh = new THREE.Line(this.geometry, this.material);
    }
```

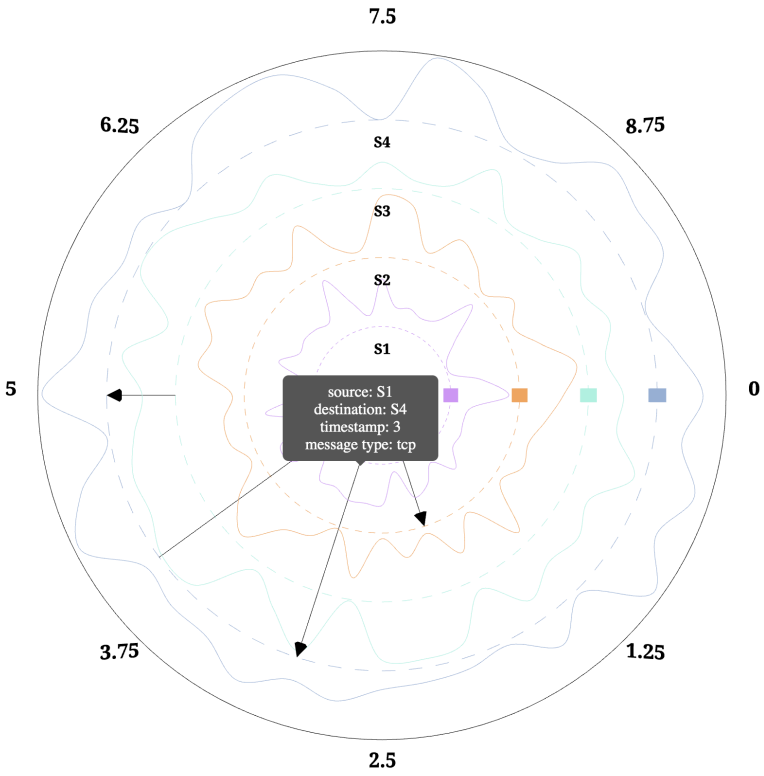
Ispis programskog koda 23

Prvo se instancira objekt razreda *CatmullRomCurve3* preko točaka dobivenih iz podataka. Potom se postavi da je krivulja zatvorena čime se spajaju krajnja i početna točka. Nakon toga se iz primljenih podataka generira sto točaka na krivulji te se instancira mreža poligona na sličan način kao i kod ostalih komponenti. Pretpostavka u ovakvom načinu generiranja točaka je da je broj podataka preko kojih se generira krivulja, dovoljno velik i dobro raspodijeljen kako bi se dobila približno kružna krivulja jer u protivnom krivulja može sjeći ostale elemente na dijagramu.

3.7.2. Iscrtavanje detalja poruke

S obzirom da svaka poruka nosi određenu informaciju u komunikaciji između poslužitelja, potrebno je omogućiti način prikaza detalja svake poruke. Budući da bi prikaz trebao biti takav da ne umanjuje preglednost

samog dijagrama, korišten je način prikaza u kojem se klikom miša na pojedinu strelicu pojavljuje prozor unutar kojeg se prikazuju detalji poruke. Ponovnim klikom na bilo koji dio scene taj prozorčić prestaje biti vidljiv. Na slici 5 je dan primjer prikaza dodatnih informacija jedne poruke.



Slika 5. Prikaz detalja jedne poruke

Budući da je potrebno prepoznati svaki puta kada korisnik klikne mišem, razred *SequenceDiagram* sadrži metodu *onMouseClick*. Ta metoda je pretplaćena na događaj klika miša te se poziva kada je to potrebno. Metoda koristi Three.js razred *Raycaster* kako bi provjerila je li korisnik kliknuo na određenu strelicu. Kod koji to radi je prikazan u nastavku:

```
   const raycaster = new THREE.Raycaster();
  raycaster.setFromCamera(mouse, camera);
  this.arrowComponents.forEach(arrow => {
      const intersections = raycaster.intersectObject(arrow.mesh,
true);
      if(intersections.length > 0) {
            // user clicked on arrow …
      }
```

Ispis programskog koda 24

Prvo se instancira objekt razreda *Raycaster*. Potom se metodom *setFromCamera* postavi novo izvorište i smjer zrake na način da je izvorište u kameri a smjer prema mjestu gdje je korisnik kliknuo mišem. Nakon toga se provjerava je li zraka presjekla bilo koju strelicu. To radi metoda *intersectObject*. Ta metoda kao prvi parametar prima mrežu poligona za koju se želi napraviti provjera presjeka, a drugi parametar označava hoće li metoda provjeravati i sve mreže poligona koji su djeca prvog parametra. U ovom slučaju se to radi jer je strelica sastavljena od stošca i linije pa se moraju provjeriti sjecišta i za jedno i za drugo. Metoda vraća listu objekata koje je zraka presjekla te ako je u toj listi barem jedan element, onda znamo da je korisnik kliknuo na strelicu.

Budući da biblioteka Three.js nema mogućnost kojim bi se izravno mogao definirati ovakav prikaz detalja, za prikaz je korišten HTML i CSS. To je moguće jer web preglednik iscrtava sve elemente unutar HTML dokumenta kojeg dobije od poslužitelja. Budući da je sav dijagram iscrtan unutar elementa <*canvas*>, moguće je koristiti i ostale HTML elemente te pomoću njih dobiti navedeni prikaz detalja poruke.

```
<div class="popup" id="popupWindow" style="position: absolute;">
          <div class="popuptext" id="messageInfoPopup">
                <div id="sourcePlaceholder"></div>
```

```
            <div id="destinationPlaceholder"></div>
            <div id="timestampPlaceholder"></div>
            <div id="typePlaceholder"></div>
        </div>
    </div>
```

Ispis programskog koda 25

Prikazani programski kod prikazuje dio HTML dokumenta u kojem je definirana samo struktura i izgled prozora koji prikazuje detalje poruke bez teksta. Bitno je uočiti da vršni element ima pridjeljeno svojstvo kojim mu se pozicija određuje na apsolutan način, to jest omogućuje se da se prozor postavi bilo gdje na ekranu neovisno o drugim elementima. Prozor je na početku nevidljiv te je njegov izgled definiran u odvojenom CSS dokumentu. Kada smo izgled prozora definirali na ovakav način, sve što je još potrebno napraviti je umetnuti željeni tekst i pozicionirati prozor na željeno mjesto kada ga želimo prikazati. Programski kod koji to radi prikazan je u nastavku:

```
const popup = document.getElementById(“popupWindow");const
msgInfo = document.getElementById("messageInfoPopup");const
target = new THREE.Vector3();
arrow.mesh.getWorldPosition(target);
target.project(camera);
target.x = Math.round((target.x + 1) * window.innerWidth  /
2 );
target.y = Math.round((-target.y + 1) * window.innerHeight / 2
);
target.z = 0;
popup.style.left = String(target.x) + "px";
popup.style.top = String(target.y) + “px";

const source = document.getElementById("sourcePlaceholder");
source.textContent = "source: " + messageInfo.NAME_src;
const dest =document.getElementById("destinationPlaceholder");
```

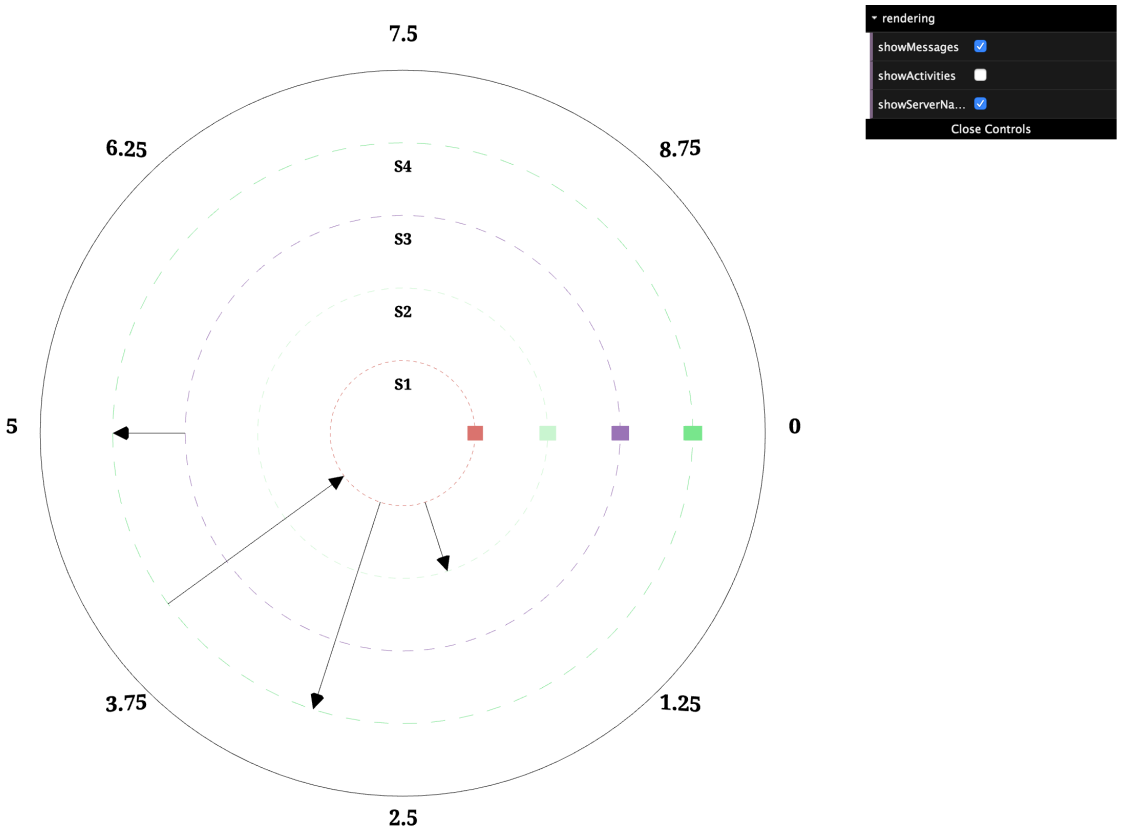
```
dest.textContent = "destination: " + messageInfo.NAME_dest;
const time = document.getElementById("timestampPlaceholder");
time.textContent = "timestamp: " + messageInfo.timestamp;
const type = document.getElementById("typePlaceholder");
type.textContent = "message type: " + messageInfo.msg_type;
msgInfo.classList.toggle("show");
this.popupOnScreen = true;
```

Ispis programskog koda 26

HTML elemente dohvaćamo metodom *getElementById.* Nakon toga definiramo vektor *target* koji definira gdje treba smjestiti prozor. Potom se računaju koordinate strelice metodom *getWorldPosition(target)*. Na ovaj se način u vektoru *target* nalazi pozicija strelice. Međutim, budući da te koordinate nisu koordinate ekrana, potrebno je taj vektor projicirati u ovisnosti o kameri te ga pretvoriti u koordinate ekrana. Na kraju, elementima pridružimo tekst koji želimo prikazati te prikažemo prozor na način da dodamo CSS klasu koja će promjeniti izgled prozora i učiniti ga vidljivim.

3.7.3. Biblioteka dat.gui

Biblioteka dat.gui pruža jednostavno grafičko sučelje za mijenjanje vrijednosti varijablama unutar JavaScript programskog jezika. Biblioteka je korištena kako bi se jednostavno uključilo ili isključilo prikazivanje pojedinih dijelova dijagrama. Konkretno, koristi se kako bi se uključio ili isključio prikaz strelica, identifikatora poslužitelja ili aktivnosti procesora. Slika 6 prikazuje izgled sučelja generiranog bibliotekom dat.gui te je ispod slike naveden programski kod koji stvara prikazano sučelje.



Slika 6. Prikaz grafičkog korisničkog sučelja

```
    const gui = new dat.GUI();
    const renderGUI = gui.addFolder("rendering");
    const showMessagesController = renderGUI.add(sequenceDiagram,
'showMessages');
    showMessagesController.onChange(value => {
        sequenceDiagram.showMessagesValueChanged(value);
    });

    const showActivitiesController = renderGUI.add(sequenceDiagram,
'showActivities');
    showActivitiesController.onChange(value => {
        sequenceDiagram.showActivitiesValueChanged(value);
    });

    const showServerNamesController = renderGUI.add(sequenceDiagram,
'showServerNames');
    showServerNamesController.onChange(value => {
```

```
        sequenceDiagram.showServerNamesValueChanged(value);
    });
```

Ispis programskog koda 27

Nakon inicijalizacije sučelja sve što je potrebno napraviti je pozvati nad inicijaliziranim sučeljem metodu *add* kojoj kao prvi parametar predajemo objekt, a kao drugi parametar ime varijable čiju vrijednost želimo moći mijenjati putem sučelja. Opcionalno, metodom *onChange* možemo konfigurirati biblioteku na način da svaka promjena varijable kroz sučelje inicira poziv funkcije koju predajemo kao parametar metodi. To je napravljeno u prikazanom isječku koda jer je na svaku promjenu varijable potrebno uključiti ili isključiti prikaz nekog dijela dijagrama.

4. Zaključak

Biblioteka Three.js znatno olakšava upotrebu WebGL tehnologije u izradi grafičkih aplikacija za web. Biblioteka definira razrede i funkcije koje implementiraju mnoge korisne funkcionalnosti za korištenje u grafičkim aplikacijama. Osim toga, omogućuje jednostavniju implementaciju interakcije miša i tipkovnice s programom. U radu je opisan nov način vizualizacije klasičnog sekvencijskog dijagrama kojim se htjelo postići jednostavnije i brže shvaćanje dijagrama kao i prikazati cijeli dijagram na zaslonu bez potrebe za pomicanjem pogleda. Budući da se radi o novom načinu vizualizacije teško je reći je li takav način bolji od postojećeg te postoji li konkretna primjena u kojoj se on može bolje iskoristiti nego li klasična vizualizacija. Također je pokazan primjer proširenja dijagrama na način da osim komunikacije, dijagram prikazuje i procesorsku aktivnost u pojedinom trenutku. U svakom slučaju, prikazana vizualizacija daje jedan novi pogled na dijagram koji se često susreće u praksi što uvijek može biti korisno kao ideja za nove načine vizualizacije ili poboljšanja postojećih.

Kao nastavak za budući rad dijagram bi se mogao integrirati u neke postojeće sustave. Primjerice, dijagram bi se mogao koristiti za vizualizaciju podataka dobivenih putem programa za praćenje mrežnog prometa Wireshark. Moguće je i proširenje kojim bi se pratili podaci u stvarnom vremenu te bi tada dijagram dinamički kroz vrijeme mogao prikazivati komunikaciju.

5. Literatura

[1] Nicholas C. Zakas, Professional JavaScript for Web Developers, Wrox, 2012

[2] HTML5 Introduction, http://www.w3schools.com/html/html5_intro.asp, 20.04.2019.

[3] What is HTML DOM, HTML DOM, https://www.w3schools.com/whatis/whatis_htmldom.asp, 20.04.2019

[3] HTML5 New Elements, https://www.w3schools.com/html/html5_new_elements.asp, 20.04.2019.

[4] HTML5 Canvas, http://www.w3schools.com/html/html5_canvas.asp, 20.04.2019.

[5] GitHub, Three.js, https://github.com/mrdoob/three.js, 28.03.2019

[6] ThreeJs, ThreeJs, http://threejs.org, 28.03.2019.

[7] Learning Three.js, Learning Three.js, http://learningthreejs.com, 28.03.2019

[8] Github, dat.gui, https://github.com/dataarts/dat.gui, 20.5.2019

[9] Node.js, https://nodejs.org/en/, 28.3.2019

[10] Node.js Tutorial, Node.js, https://www.w3schools.com/nodejs/, 28.3.2019

[11] http-server-npm, https://www.npmjs.com/package/http-server, 28.3.2019

Vizualna analitika primijenjena na grupiranje velikih skupova podataka

Sažetak

U radu je prikazana metoda vizualizacije sekvencijskog dijagrama. Opisana je ideja iza takvog prikaza te je napravljena usporedba s klasičnom metodom vizualizacije sekvencijskog dijagrama. Izrečene su prednosti i nedostaci metode. Programska implementacija napravljena je u programskom jeziku JavaScript uporabom biblioteke Three.js. Opisana je struktura programskog rješenja te je dan uvid kako takav dijagram proširiti da pokazuje još informacija koje mogu biti korisne pri upotrebi.

Ključne riječi: Sekvencijski dijagram, vizualizacija dijagrama, Three.js, JavaScript, HTML.

Visual analytics applied to grouping large data sets

Abstract

The paper describes method for visualizing sequence diagram. It describes the idea behind this type of visualization and comparison is made to classical method that is used for visualizing sequence diagram. Advantages and disadvantages of the method are outlined in the paper. Program implementation is made in the JavaScript programming language using the Three.js library. The structure of the software solution is described and the example is given on how to expand the usage of diagram to show more information that could be useful.

Key words: Sequence diagram, diagram visualization, Three.js, JavaScript, HTML.