SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2345

Proceduralno generiranje trave i niskog raslinja

Mihael Međan

Zagreb, siječanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 9. listopada 2020.

DIPLOMSKI ZADATAK br. 2345

Pristupnik:	**Mihael Međan (0036487393)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Željka Mihajlović

Zadatak: **Proceduralno generiranje trave i niskog raslinja**

Opis zadatka:

Proučiti tehnike generiranja trave i niskog raslinja uz različite razine detaljnosti u prikazu (LOD). Proučiti utjecaj fizikalnih veličina kao što je vjetar na izradu simuliranih i animiranih prikaza. Razraditi i ostvariti fizikalno temeljen simulacijski model prikaza trave i niskog raslinja iz različite razine detaljnosti prikaza. Diskutirati utjecaj raznih parametara. Načiniti ocjenu rezultata i implementiranih algoritama. Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 5. veljače 2021.

SADRŽAJ

1. Uvod

Proceduralno generiranje 3D modela je izrazito široko područje s velikim brojem različitih pristupa rješavanju tog problema ovisno o primjeni. Generiranje modela se u novije vrijeme sve češće radi skeniranjem stvarnih objekata. Ovakav način generiranja modela daje impresivne rezultate, ali takvi rezultati su često izrazito kompleksni s jako puno geometrije i neupotrebljivi su u interaktivnim sustavima poput video igara.

U video igrama je potreban minimalan broj detalja kako bi brzina izvođenja aplikacije bila što veća, ali dovoljno velik da daje dojam stvarnosti. Današnje igre sve više teže prema otvorenom svijetu (engl. *open-world*). Kod takvih igara igraču je prepuštena potpuna sloboda kretanja na velikim površinama. Sa željom da povećaju brzinu izvođenja, programeri koriste tehniku iscrtavanja prikaza s različitom razinom detalja. Kod ovakvog pristupa, objekti koji su daleko od kamere koriste modele s izrazito malo detalja, a kod objekata koji su bliže kameri koriste detaljne modele. Budući da su modeli koji su udaljeni od kamere na ekranu mali, igrač ne može vidjeti gubitak detalja i dojam stvarnosti mu ostaje velik.

Još jedan od problema koji se susreće kod igara s otvorenim svijetom je vegetacija. Pojedine lokacije igrinog svijeta imaju potpuno različite vegetacije. Ovo iziskuje vrlo velik napor od umjetnika koji moraju modelirati različite vegetacije za različite lokacije. Osim velikog broja različitih biljaka, zbog tehnike iscrtavanja u različitoj razini detalja često je za isti model potrebno napraviti nekoliko različitih modela s različitom razinom detalja, što dodatno rezultira vremenom potrebnim za izradu igre. Igre su interaktivni sustav i statični objekti koji su u stvarnom svijetu dinamički razbijaju osjećaj stvarnosti. Zbog toga je sve te modele potrebno i animirati.

Kroz ovaj rad istražena je tehnika generiranja modela vegetacije u različitim razinama detalja. Osim generiranja modela, istražen je i način simulacije vjetra kako bi se i taj dio mogao automatizirati i time skratiti vrijeme razvoja igara.

2. Generiranje modela trave i niskog raslinja

2.1. Osnovni pristup generiranju modela

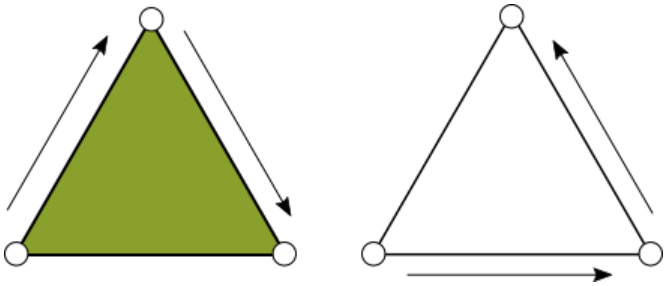
Grafička programska sučelja (engl. *graphics application programming interface*) su skup bibloteka korištena za interakciju sa grafičkom karticom u svrhu postizanja sklopovski ubrzanog iscrtavanja (engl. *hardware-accelerated rendering*). Najpoznatija grafička programska sučelja su OpenGL, DirectX, Vulkan te verzija OpenGL-a napisana za korištenje na webu - WebGL.

Grafička programska sučelja omogućuju pristup resursima grafičke kartice. Grafička procesorska jedinica višestruko je brža od centralne procesorske jedinice u operacijama koje su potrebne za rasterizaciju prikaza. Rasterizacija je postupak transformacije podataka grafičkih primitiva u virtualnom prostoru u reprezentaciju pomoću slikovnih elemenata (engl. *pixels*). Postupak rasterizacije se većim dijelom svodi na operacije nad matricama i vektorima. Instrukcije grafičkoj procesorskoj jedinici kako rasterizirati prikaz opisuju se u programu za sjenčanje. Program za sjenčanje (engl. *shader*) se izvršava na grafičkoj kartici.

Izvršavanje takvih programa na grafičkoj kartici je izrazito brzo ali je i jako ograničeno u opsegu zadaća koje takav program može izvoditi. Zbog toga se za sve ostale zadaće osim rasterizacija koristi centralna procesorska jedinica, a grafička kartica izvodi samo rasterizaciju. Centralna procesorska jedinica se mora brinuti o zadaćama poput pripreme i slanja podataka prema grafičkoj kartici, alokaciji i čišćenju memorije te obradi korisničkih ulaza.

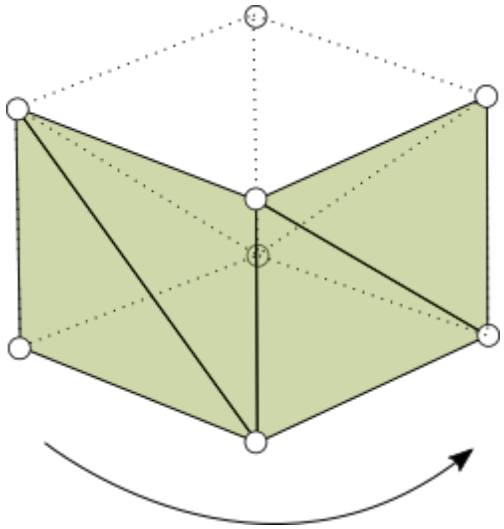
Podaci koje centralna procesorska jedinica mora pripremiti i poslati na obradu grafičkoj kartici uključuju pozicije točaka, popis lica koje te točke modeliraju, normale tih

lica, pozicije i intenziteta svijetla, informacije o načinu mapiranja tekstura na model (engl. *UV texture maps*), podatke o statusu animacije i druge opcionalne naprednije parametre.



Slika 2.1: Ilustracija redoslijeda točaka u smjeru kazaljke na satu i u smjeru obrnutom od kazaljke na satu

Za generiranje modela u 3D prostoru koji će se uspješno rasterizirati potrebno je osigurati da su točke koje tvore lica poredane u smjeru kazaljke na satu zbog optimizacija prilikom rasteriziranja. Slika 2.1 prikazuje moguće redoslijede definiranja točaka koje tvore lice s točkom u donjem lijevom kutu kao početnom točkom za definiranje lica. Zelena boja označava lice koje će se rasterizirati iz trenutnog kuta gledišta.



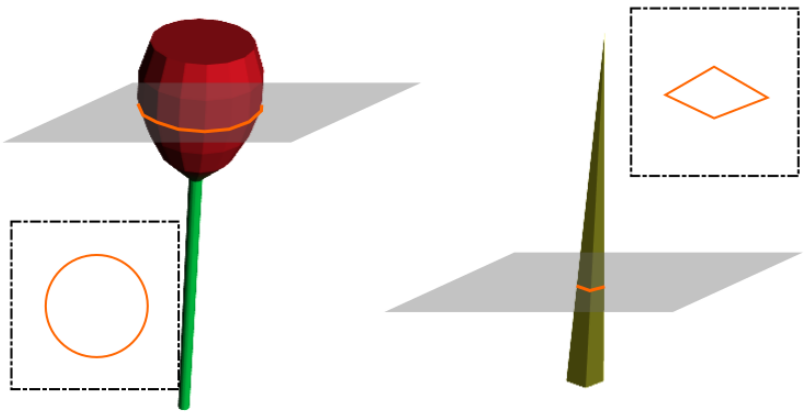
Slika 2.2: Šetnja oko modela za popunjavanje lica

Uzimajući u obzir redoslijed točaka unutar lica, potrebno je generirati sva lica 3D modela kako bi svako lice bilo vidljivo iz kuta gledišta u kojem treba biti vidljivo. Jednostavan način za vizualizaciju tog postupka prikazan je na slici 2.2. Krenuvši od

početne točke generiramo samo ona lica koja bi bila vidljiva iz neke točke gledišta. Rotiramo točku gledišta oko modela i dodajemo lica koja su vidljiva iz točke gledišta, a nisu prije bila dodana. Nakon vraćanja u početni položaj, sva lica vidljiva iz kuta gledišta su popunjena. Proces je potrebno ponoviti na različitim visinama točke gledišta kako bi se popunila i lica koja omeđuju model s gornje i donje strane.

2.2. Generiranje točaka i lica modela biljaka

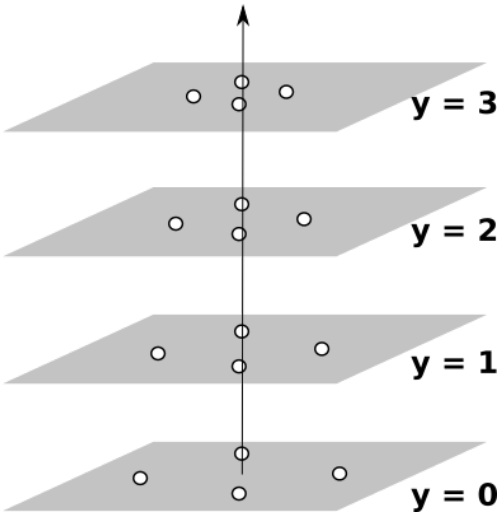
Generiranje točaka biljaka vrši se ekstrapolacijom osnovnog oblika biljke na različite visine. Osnovni oblik biljke je presjek biljke sa horizontalnom ravninom. Za primjer trstike osnovni oblik je krug, a trave paralelogram. Slika 2.3 prikazuje vizualizaciju dobivanja osnovnog oblika biljke za tulipan i travku.



Slika 2.3: Dobivanje osnovnog oblika biljke

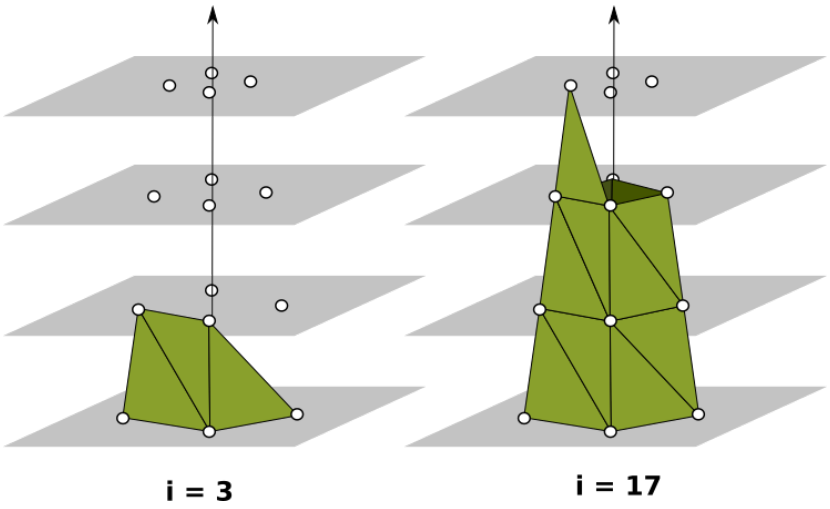
Nakon postavljanja početnog oblika, točke osnovnog oblika se postavljaju kao korijen biljke i dodaje se novi prsten osnovnog oblika na fiksnim intervalima visine. Svaki prsten ima svoju određenu širinu oko centralne točke koja se dobiva iz opisne funkcije biljke. Konstruiranje opisne funkcije biljke opisano je u 2.5. Slika 2.4 prikazuje ekstrapolaciju točaka trave, kojima je osnovni oblik paralelogram.

Generirane točke spajaju se u model dodavanjem lica. Dodavanje lica se vrši šetnjom gledišta oko generiranih točaka. Postupak dodavanja lica na generirane točke



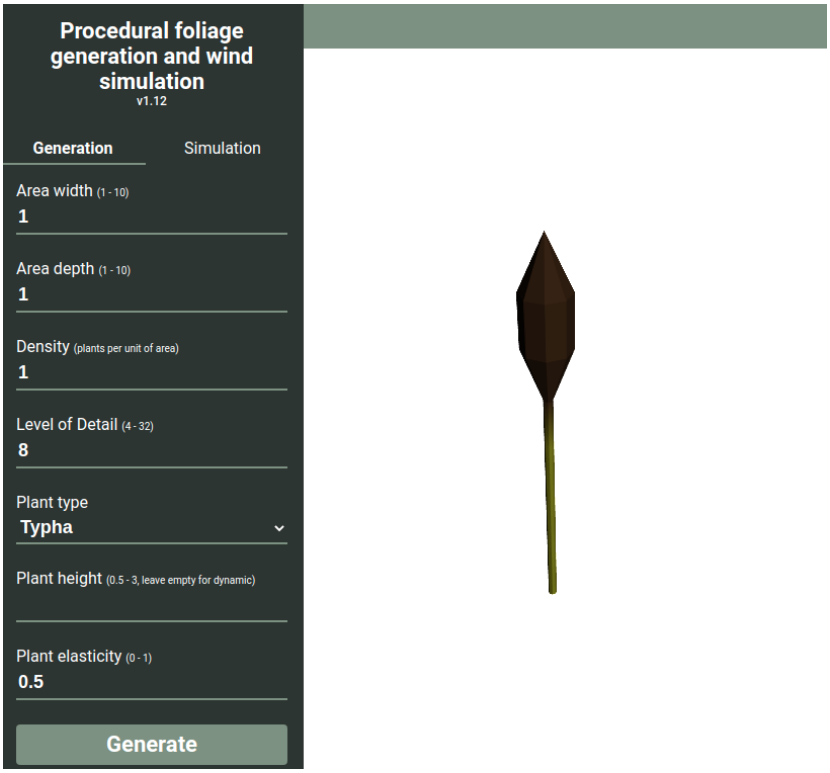
Slika 2.4: Ekstrapolacija točaka trave

opisan je u poglavlju 2.1. Slika 2.5 prikazuje postupak omatanja u dvije različite iteracije. Tamnije obojana lica reprezentiraju lica koja gledaju u smjeru suprotnom od trenutnog gledišta.



Slika 2.5: Spajanje točaka u model kroz iteracije

Neke osnovne biljke i njihove generacijske funkcije uključene su u prezentirano rješenje i mogu se generirati preko korisničkog sučelja. Korisničko sučelje za generiranje modela biljaka prikazano je na slici 2.6.

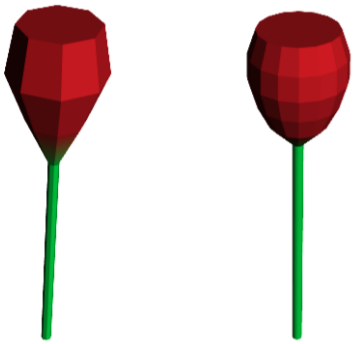


Slika 2.6: Korisničko sučelje za generiranje modela biljaka

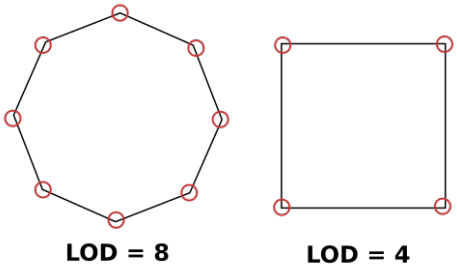
2.3. Generiranje modela uz različitu razinu detalja

Različite razina za sve biljke detalja uvjetovane su brojem vertikalnih segmenata na koje je biljka razdvojena po svojoj visini. Veći broj segmenata rezultira većom razinom detalja. Razdvajanje biljaka po vertikalnim segmentima prikazano je na slici 2.7. Dodavanjem većeg broja segmenata biljka dobiva sve više točaka te na taj način objekt dobiva i više detalja. Sijenčanje takvog modela rezultira postepenim prijelazom između boja. Kod modela s malim brojem segmenata, promjene u intenzitetu osvjetljenja su mnogo izraženije što je također vidljivo na slici 2.7.

Osim razine detalja po vertikalnim segmentima, moguće je osnovni oblik biljke uvjetovati po količini detalja te time dobiti dodatne karakteristike kod prikaza s više detalja. Primjer generiranja osnovnog oblika u većoj ili manjoj razini detalja za osnovni oblik kruga ilustrirano je slikom 2.8, a implementacija tog primjera prikazana je isječkom 2.7.



Slika 2.7: Generirane biljke sa različitim brojem vertikalnih segmenata



Slika 2.8: Prikaz ovisnosti baze kruga o razini detalja prikaza

2.4. Programska implementacija generiranja i prikaza

U programskoj implementaciji generiranje modela razdvojeno je u dva dijela. Prvi dio, prikazan u isječku koda 2.1, je generiranje gradivnih točaka modela. Gradivne točke modela su svojevrsni kostur samog modela. Funkcija koja vrši generiranje gradivnih točaka kao parametre uzima visinu biljke i razinu detalja za generiranje. Iz opisne funkcije biljke uzima se osnovni oblik. Na temelju visine biljke i razine detalja izračunat je inkrement visine kod generiranja te korak između dvije razine.

Program prolazi kroz sve segmente biljke i dodaje dodatan osnovni oblik na visini povezanoj s trenutnim segmentom. Prsten osnovnog oblika se skalira vrijednošću za taj visinski faktor. Vrijednost skaliranja definirana je u funkciji generiranja biljke.

Primjer konstruiranja funkcije generiranja biljke opisan je u poglavlju 2.5.

```
generateVertices(height, lod) {
  let vertices = [];
  let baseShape = fn.getBaseShape(lod);
  if (!height) {
    height = fn.generateHeight();
  }

  const constructionVertices = [];
  for (const v of baseShape) {
    constructionVertices.push(v.map(a => a * fn.calculateCurvePoint
        (0)));
  }

  const heightIncrement = height / lod;
  const offsetStep = 1 / lod;

  let currentHeight = heightIncrement;
  let currentOffset = offsetStep;

  for (let i = 1; i < lod + 1; i++) {
    const newBaseShape = [];
    for (const v of baseShape) {
      newBaseShape.push(
        v.map(a => a * fn.calculateCurvePoint(currentOffset))
      );
    }

    for (const v of newBaseShape) {
      v[1] = currentHeight;
      constructionVertices.push(v);
    }

    currentHeight += heightIncrement;
    currentOffset += offsetStep;
  }
}
```

Isječak koda 2.1: Generiranje gradivnih točaka modela biljke

Nakon generiranja gradivnih blokova algoritam prelazi na sljedeći korak. U drugom koraku, prikazanom u isječku koda 2.2, algoritam radi šetnju oko modela postupkom opisanim u 2.2 za svaki visinski segment biljke. Važno je napomenuti da šetnja oko modela samo vizualno tumačenje koda koji se zapravo izvršava.

```
const vertices = [];
for (let level = 0; level < lod + 1; level++) {
    for (let i = 0; i < baseShape.length; i++) {
        const next = i + 1 < baseShape.length ? i + 1 : 0;
        const rectVertices = [
            constructionVertices[baseShape.length * level + i],
            constructionVertices[baseShape.length * level + i +
                baseShape.length],
            constructionVertices[baseShape.length * level + next],
            constructionVertices[baseShape.length * level + next +
                baseShape.length]
        ]

        vertices.push(
            rectVertices[0],
            rectVertices[2],
            rectVertices[1],
            rectVertices[1],
            rectVertices[2],
            rectVertices[3],
        )
    }
}
```

Isječak koda 2.2: Generiranje lica od gradivnih točaka

Nakon generiranja svih potrebnih podataka, potrebno ih je poslati na grafičku karticu kako bi se ti podaci mogli iscrtati. Slanje podataka na grafičku karticu otvara se otvaranjem kanala (engl. *buffer*). Isječak koda **??** prikazuje način povezivanja kanala sa atributima u programu za sjenčanje. Isječak koda za kreiranje kanala je identičan za sve vrste podataka koje šaljemo na grafičku karticu i razlikuje se samo u parametrima.

```
createAttributeBuffer(shaderBinding, size, type) {
    if (!type) {
        type = this.gl.FLOAT;
```

```
4       }
5
6       const buffer = this.gl.createBuffer();
7       this.gl.bindBuffer(this.gl.ARRAY_BUFFER, buffer);
8
9       this.attributes[shaderBinding] = {
10          buffer: buffer,
11          attribLocation: this.gl.getAttribLocation(this.program,
                shaderBinding),
12          currentData: [],
13          numComponents: size,
14          type: type
15      };
16
17      const attr = this.attributes[shaderBinding];
18      this.gl.enableVertexAttribArray(attr.attribLocation);
19      this.gl.vertexAttribPointer(
20          attr.attribLocation,
21          attr.numComponents,
22          attr.type,
23          false,
24          0,
25          0
26      );
27  }
```

Isječak koda 2.3: Kreiranje kanala za slanje podataka prema grafičkoj kartici

Program za sjenčanje kojem šaljemo podatke sastoji se od dva dijela. Programa za sjenčanje točaka (engl. vertex shader) i programa za sjenčanje slikovnih elemenata (engl. fragment shader). Program za sjenčanje točaka se izvršava za svaku točku u prostoru, a program za sjenčanje slikovnih elemenata za svaki slikovni element prikaza. Na početku svakog dijela programa definiramo atribute, uniformne elemente i varirajuće elemente. Atributi su svojstva neke točke u prostoru poput pozicije, boje i normale. Uniformni elementi su jednaki u jednom iscrtavanju za sve točke odnosno slikovne elemente. Varirajući elementi svoju vrijednost dobivaju interpolacijom između točaka koji čine lice od kuda im i naziv. Svaki slikovni element ima svoju jedinstvenu vrijednost varirajućih elemenata.

Isječak koda 2.2 prikazuje primjer jednostavnog programa za sjenčanje točaka korištenog u prezentiranom rješenju. U nastavku rada ovaj program je nadograđen moguč-

nošću animacije.

```
attribute vec4 a_position;
attribute vec3 a_normal;
attribute vec4 a_color;

uniform mat4 u_worldViewProjection;
uniform mat4 u_worldInverseTranspose;
uniform mat4 u_bone;

varying vec3 v_normal;
varying vec4 v_color;

void main() {
    gl_Position = u_worldViewProjection * a_position(
    v_normal = mat3(u_worldInverseTranspose) * a_normal;
    v_color = a_color;
}
```

Isječak koda 2.4: Program za sjenčanje točaka

Izvršavanjem programa za sjenčanje točaka nad svim točkama prelazimo na sjenčanje slikovnih elemenata. Program za sjenčanje slikovnih elemenata ima sličnu strukturu kao i program za sjenčanje točaka. U isječku koda 2.5 prikazan je jednostavan program za sjenčanje slikovnih elemenata koji pomoću normale slikovnog elementa računa njegovu osvijetljenost i primjenjuje je uz boju slikovnog elementa.

```
precision mediump float;
varying vec3 v_normal;
uniform vec3 u_reverseLightDirection;

varying vec4 v_color;

void main() {
    vec3 normal = normalize(v_normal);
    float light = max(dot(normal, u_reverseLightDirection), 0.1);
    gl_FragColor = v_color;
    gl_FragColor.rgb *= light;
}
```

Isječak koda 2.5: Program za sjenčanje slikovnih elemenata

2.5. Primjer korištenja razvijenog programa za generiranje jednostavne proizvoljne biljke

Generiranje biljke prezentiranim rješenjem je jednostavno. Potrebno je definirati svega nekoliko svojstava. Oblik baze, visinu biljke, širinu na pojedinim dijelovima te opcijanalno boju na dijelovima biljke. Slika 2.9 prikazuje završeni model tulipana objašnjenog u ovom primjeru.

Slika 2.9: Prikaz završenog modela tulipana

Prije opisa biljke potrebno je osigurati apstraktna svojstva biljke nasljeđivanjem razreda $BaseGenerationFunction$. Postupak je vidljiv u isječku koda 2.6.

```
1 import {BaseGenerationFunction} from "base-generation-function.js";
2 export class TulipFunction extends BaseGenerationFunction {
3 }
```

Isječak koda 2.6: Nasljeđivanje razreda BaseGenerationFunction

Nakon naslijeđivanja, potrebno je generirati osnovni oblik biljke. Osnovni oblik biljke možemo zamisliti kao presjek biljke i horizontalne ravnine. U slučaju tulipana to je krug. Skup točaka koje reprezentiraju krug možemo izračunati kretanjem po jediničnoj kružnici, i na svakom pomaku trigonometrijskim funkcijama sin i cos izračunati koordinate na trenutnom koraku. Koordinatu y fiksiramo na 0 za dobivanje točaka u istoj horizontalnoj ravnini. Isječak koda 2.7 prikazuje taj postupak. Opis baznog oblika vrši se u metodi $getBaseShape$. Metoda $getBaseShape$ prima parametar lod koji opisuje u kojoj razini detalja generiramo biljku.

```
1  getBaseShape(lod) {
2      if (!lod || lod < 4) {
3      lod = 4;
4    }
5
6    const vertices = [];
7    let i = 0;
8    while (i < lod) {
9      const offset = i / lod;
10     vertices.push([
11       -Math.cos(offset * Math.PI * 2) * 0.3,
12       0,
13       Math.sin(offset * Math.PI * 2) * 0.3
14     ]);
15
16     i++;
17   }
18
19   return vertices;
20 }
```

Isječak koda 2.7: Generiranje osnovnog oblika biljke

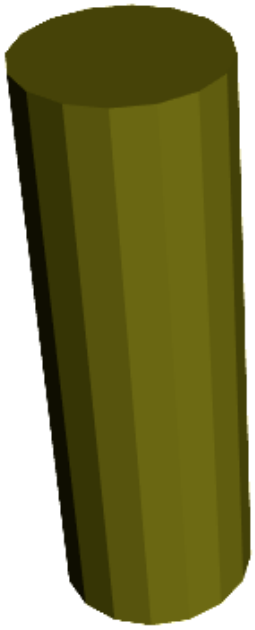
Nakon generiranja osnovnog oblika, potrebno je definirati pretpostavljenu visinu biljke. U ovom primjeru je ta visina nasumična između 1.2 i 2.2 jedinica u trodimenzijskom prostoru. Isječak koda 2.9 prikazuje definiranje pretpostavljene visine biljke.

```
1  generateHeight() {
2    return Math.random() + 1.2;
```

```
3 }
```

Isječak koda 2.8: Generiranje visine biljke

Slika 2.10 pokazuje rezultat generiranja sa definiranim osnovnim oblikom i pretpostavljenom visinom.



Slika 2.10: Rezultat generiranja osnovnog oblika i pretpostavljene visine

Nakon opisa osnovna dva svojsta, potrebno je definirati zakrivljenost modela. Zakrivljenost modela se postavlja u metodi $calculateCurvePoint$. Metoda $calculateCurvePoint$ prima parametar $offset$ koji nam govori za koju visinu biljke tražimo širinu. Parametar $offset$ je vrijednost između 0 i 1, gdje 0 simbolizira dno biljke, a 1 simbolizira vrh biljke. Tulipan je biljka tanke stabljike i zvonolikog cvijeta. Za tanku stabljiku vraćamo vrijednost širine 0.1, a zvono modeliramo sa pola periode spljoštene sinusoide i bazne širine na koju dodajemo amplitudu takve sinusoide. Isječak koda 2.9 pokazuje opis biljke tulipana, a slika 2.11 rezultat pokretanja nakon opisa oblika.

```
1 calculateCurvePoint(offset) {
2   if (offset < 0.75) {
3     return 0.1;
4   }
5
```

```
6     const flowerOffset = (offset - 0.75) / 0.3 * Math.PI;
7     return 0.5 + Math.sin(flowerOffset) / 3;
8   }
```

Isječak koda 2.9: Postavljanje oblika biljke u ovisnosti o visinu od tla

Slika 2.11: Rezultat generiranja sa definiranim oblikom biljke

Dodavanje boje se odvija na sličan način kao definiranje oblika. Dodavanje boje definira se u metodi $calculateColorAtPoint$ koja također prima parametar $offset$. Tulipan ima zelenu stabiljku i cvijet neke boje. U primjeru je korištena crvena boja cvijeta. Isječak koda 2.10 prikazuje postavljanje boje, a rezultat generiranja i prikazan je na slici 2.1.

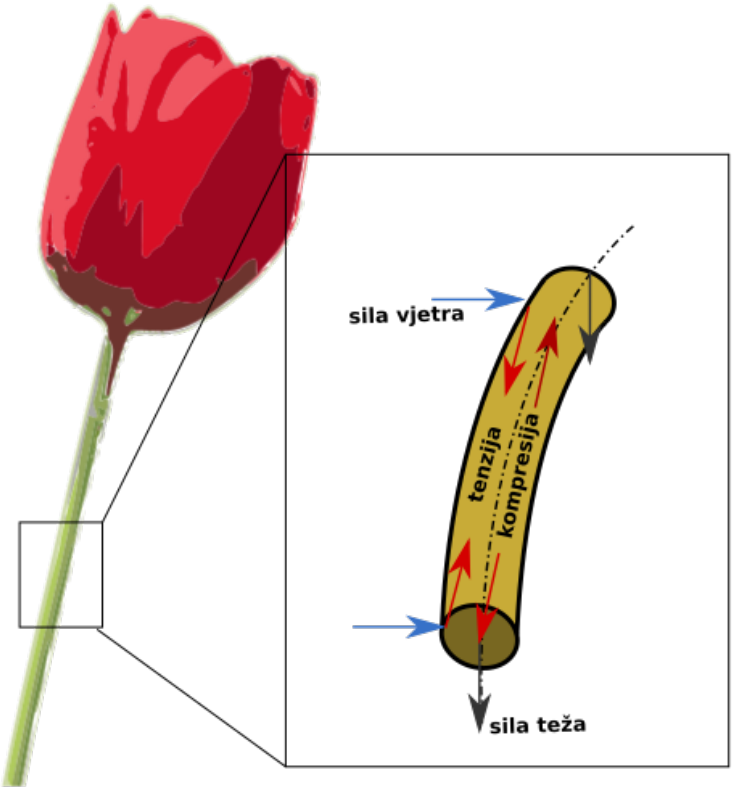
```
1  calculateColorAtPoint(offset) {
2    if (offset < 0.75) {
3      return [0.2, 0.85, 0.2, 1];
4    }
5
6    return [0.8, 0.1, 0.1, 1];
7  }
```

Isječak koda 2.10: Dodavanje boja modelu

3. Simulacijski model

3.1. Fizički model ponašanja biljke

Osnovne sile koje utječu na ponašanje biljke su sila teža, unutrašnji otpor biljke i vanjski utjecaji. Sila teža konstantnim intenzitetom gura sve dijelove biljke prema dolje. Vanjski utjecaji poput vjetra, djeluju u svim smjerovima različitim intenzitetima na različite dijelove biljke. Te unutrašnji otpor biljke, koji uvijek djeluje u suprotnom smjeru od preostale dvije sile. Na slici 3.1 su slikovito prikazane sile koje djeluju na svaki mikroskopski malen dio biljke. Važno je napomenuti da su za tenziju i kompresiju prikazane sile koje one uzrokuju, a ne smjer kompresije odnosno tenzije.



Slika 3.1: Prikaz sila koje djeluju na biljku

Usred djelovanja te tri sile, biljka u svakom trenutku pokušava doći u stanje gdje se te tri sile poništavaju. Svako gibanje biljke prouzrokovano je promjenom vanjskih utjecaja kojima se biljka pokuša prilagoditi kako bi ukupna sila koja djeluje na biljku bila nula.

Čak i naizgled jednostavan problem kao što je simuliranje ponašanja biljke je u svojoj naravi izrazito kompleksan. Sila teža djeluje na svaki mikroskopski dio biljke. Na isti način djeluju i vanjske sile. Vanjske sile i sila teže u biljci uzrokuju napetosti i kompresije koje uzrokuju silu otpora biljke. To sve se događa na mikroskopskoj razini na svakom dijelu površine biljke. Savršena simulacija ovakvog sustava bila bi izrazito računalno zahtjevna.

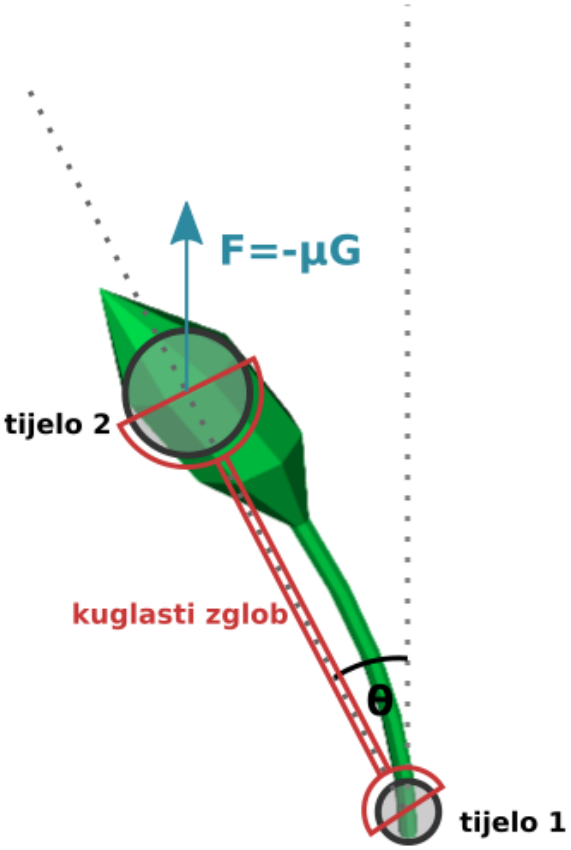
3.2. Programski model fizičkog ponašanja

Zbog računalne zahtjevnosti i problema preciznog definiranja svih sila, potrebno je pronaći matematički model koji bi rezultirao istim (ili barem vrlo sličnim) ponašanjem, a imao bi mnogo manju računalnu složenost i bio bi jednostavniji za definirati.

U računalnom programiranju se fizičke interakcije obično modeliraju preko tri osnovna građevna bloka. Tijela, ograničenja i sile. Tijela su geometrijski oblici u trodimenzijskom prostoru koja zauzimaju volumen i podložna su djelovanju sila. Ograničenja su skup pravila koja određuju kako se neka dva tijela mogu ponašati relativno jedno prema drugom. Sile djeluju na tijela i uzrokuju promjene koje se tada na temelju ograničenja rješavaju.

Primjer tijela je kocka u trodimenzijskom prostoru. Primjer sile je sila teža koja djeluje konstantnim intenzitetom prema negativnoj y osi. Primjer ograničenja je ako se volumen jednog tijela nađe unutar volumena drugog tijela, na oba tijela se primjenjuje sila u smjeru suprotnom od središta mase drugog tijela. Ovo ograničenje je naivni pristup rješavanju kolizija između tijela.

Stvarnu simulaciju jednostavne biljke možemo vrlo dobro aproksimirati korištenjem svega dva tijela, jednog ograničenja i jedne sile koja djeluje na jedno od tijela. Ovaj jednostavni model prikazan je slikom 3.2.



Slika 3.2: Ilustracija prikaza fizičkog modela stvarnog gibanja

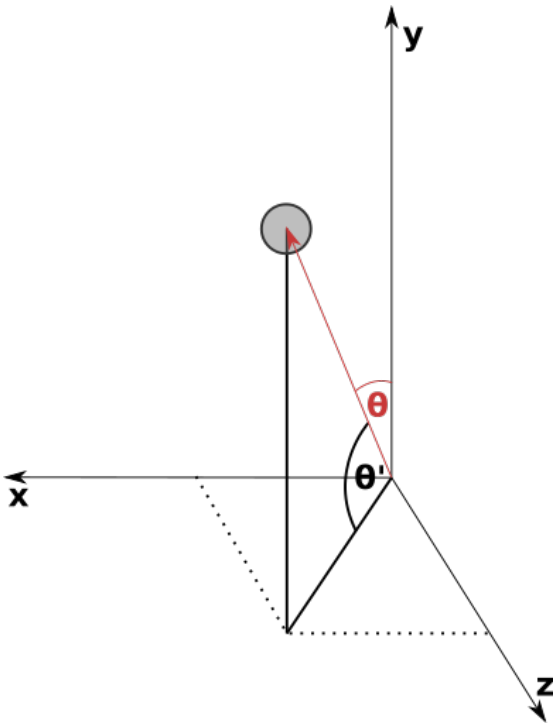
Tijelo 1 u fizičkom modelu predstavlja korijen biljke i u simulaciji je u cijelosti statično. Tijelo 2 predstavlja vrh biljke i dinamičko je, odnosno može reagirati na utjecaj sila. Ta dva tijela spojena su kuglastim zglobom. Kuglasti zglob je ograničenje na tijelo 2 koje dozvoljava tijelu da se slobodno giba i rotira u prostoru oko bilo koje osi pod uvjetom da je uvijek jednako udaljeno od tijela 1.

Na tijelo 2 također djeluje sila F koja je svojim smjerom obrnuta od smjera sile teže, a njezin iznos je određen kutom otklona od neutralne pozicije. Neutralna pozicija ima tijelo 2 direktno iznad tijela 1 i u takvom slučaju je faktor μ jednak 0.

Računanje otklona θ u trodimenzijskom prostoru radimo kao omjer duljine stranica trokuta. Jedna od stranica je projekcija pozicije tijela 2 na ravninu xz, a druga stranica je visina tijela 2 iznad te ravnine. Zbog kuglastog zgloba očekujemo da je duljina vektora pozicije tijela 2 od ishodišta uvijek jednaka, normalizacijom pozicije tijela 2, duljina projekcije na xz ravninu ima vrijednost cos kuta θ'. Zanimljiva informacija nam je otklon od y osi, koristimo $sin(\theta')$ te vrijednost kako bi dobili suprotan kut $(\pi/2-\theta')$.

U takvom slučaju nam je dovoljna informacija uzeti y vrijednost normalizirane pozicije i možemo precizno izračunati otklon. Računanje kuta otklona prikazano je formulom 3.1.

$$\theta = asin(normalized(position).y) \tag{3.1}$$



Slika 3.3: Prikaz izračuna kuta otklona

Nakon izračuna kuta otklona, računanje sile koja djeluje na tijelo 2 je sada jednostavno i prikazano formulom 3.2. Računanje sile je također proporcionalno sa sinusom kuta na intervalu između 0 i 90 stupnjeva. Sila koja djeluje na tijelo kod otklona većeg od 90 stupnjeva veća je od sile koja djeluje na tijelo na otklonu manjem od 90 stupnjeva. Kod jednostavnog modela s jednim zglobom ovo je nerealan slučaj jer se tijelo 2 ne može naći u takvoj poziciji (tijelo 2 bi tada bilo ispod tijela 1, i prošlo bi kroz površinu poda), ali ga je bitno spomenuti zbog skaliranja na model s većim brojem zglobova.

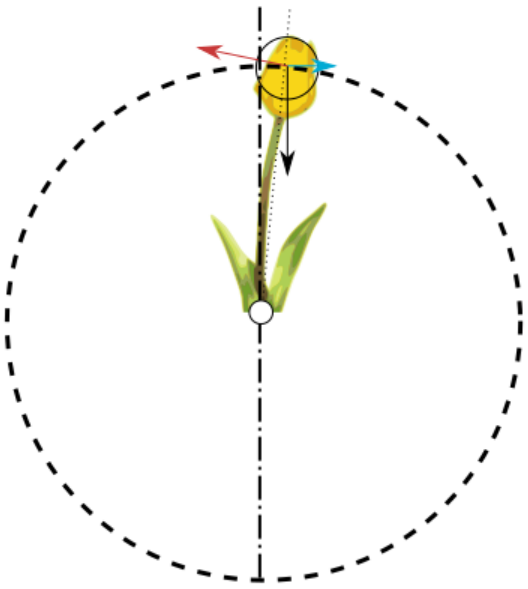
$$\vec{F} = sin(\theta) \cdot (1 + elasticity) \cdot \vec{G} \cdot (-1) \tag{3.2}$$

Rezultirajuća sila $\vec{F}$ djeluje u negativnom y smjeru (suprotno od smjera sile teže). Takav smjer je rezultat faktora $\vec{G} \cdot (-1)$. U formuli se javlja i faktor $(1 + elasticity)$.

Njegova zadaća je osigurati da biljke koje imaju veći faktor elastičnosti stoje bliže centru ravnoteže čak i kod malih otklona od središta. Primjer biljke koja ima velik faktor elastičnosti je rogoz, koji stoji potpuno uspravno u neutralnoj poziciji. Primjer male elastičnosti biljke je zvončić, koji nema dovoljno sile kako bi se vratio u neutralni (uspravni) položaj. Ime faktora je elastičnost jer odgovara na pitanje kolikom silom će se biljka pokušati vratiti u neutralni položaj nakon što je otklonjena od centra ravnoteže.

3.2.1. Usporedba sila programskog i fizičkog modela

Zbrajanjem svih komponenata sila po cijeloj površini biljke dobivamo jednu reprezentativnu silu koja dijeluje na biljku. Tu reprezentativnu silu možemo razdvojiti na tri komponente - horizontalnu, vertikalnu i silu na tangenti kružnice oko centra rotacije. Horizontalna sila je razlika između sile vjetra i horizontalne komponente unutrašnje sile otpora biljke koja se opire toj sili vjetra. Vertikalna sila je razlika između sile teže koja djeluje na biljku i vertikalne komponente unutarnje sile otpora biljke koja se odupire sili teži. Treća komponenta sile je sila tangencijalna na centar rotacije biljke. Tangencijalna sila je prividna sila koja pokušava vratiti biljku u početni položaj. Na slici 3.4 prikazane su sume sila koje djeluju na biljku.



Slika 3.4: Komponente rezultatne sile koje djeluju na gornji dio biljke

U programskom modelu fizičkog ponašanja biljke imamo samo jednu aktivnu silu koja je zadužena za simuliranje ponašanja unatrašnjeg otpora. Iako ta sila ima isključivo vertikalni smjer, fizički model i dalje pokazuje horizontalno ponašanje. Razlog ponašanja koje prati fizičko ponašanje čak i slučaju kada ne simuliramo sve sile je ograničenje kuglastog zgloba. Kuglasti zglob implicitno primjenjuje silu preko svojih pravila. Primjenom sile prema gore, tijelo koje reprezentira vrh biljke pokušava se gibati prema gore, gibanjem prema gore udaljava se od tijela korijena biljke. Kuglasti zglob osigurava jednako udaljenost između tijela pomicanjem tijela vrha biljke u horizontalnom smjeru.

Fizički pokretač ne djeluje silom da bi pomaknuo tijelo u horizontalnom smjeru, već mijenja poziciju direktno. Ali čak i takvo rješenje daje uvjeriljivu simulaciju.

3.3. Programska implementacija fizičkog modela

Fizički pokretač korišten u prezentiranom rješenju je OimoPhysics. Taj pokretač je napisan u programskom jeziku Haxe i preveden u programski jezik Javascript. Izbor ovog fizičkog pokretača prvenstveno je potaknut njegovom relativnom jednostavnošću i njegovom mogućnosti izvršavanja direktnog izvođenja unutar preglednika. Zbog jednostavnog modela kojeg simuliramo, nije predana velika težina performansama koje zbog izvođenja u interpretiranom jeziku mogu negativno utjecati na brzinu izvođenja programskog rješenja.

Osnova svakog fizičkog pokretača je svijet simulacije (engl. *world*) koji sadržava sve osnovne gradivne blokove fizičke simulacije objašnjene u sekciji 3.2. Svijet implicitno definira ograničenja za rješavanje kolizija između objekata. Inicijalno postavljanje vrijednosti svijeta u OimoPhysics pokretaču je iznimno jednostavno i prikazano je isječkom koda 3.1.

```
1 const world = new OIMO.World(1, new Vec3(0, -9.81, 0))
```

Isječak koda 3.1: Postavljanje simulacijskog svijeta

Prvi parametar kod inicijalizacije je izbor algoritma rješavača kolizija (engl. *collision-solver*. Različiti algoritmi pridaju različite važnosti pojedinim elementima simulacije.

Za znanstvene radove izabrali bi algoritam koji manju pažnju pridaje brzini izvođenja simulacije, a veću pažnju točnosti same simulacije. Za igre i ostale sustave koji se izvršavaju u stvarnom vremenu, manje nam je bitna točnost, a bitnija je brzina izvođenja. Algoritmi unutar OimoPhysics pokretača su označeni od 0 do 2, gdje vrijednost implicira kolika se težina pridaje točnosti simulacije. U prezentiranom radu korištena je postavka 1, koja postiže balansirani odnos između točnosti i brzine izvođenja.

Nakon kreiranja simulacijskog svijeta dodajemo fizički kostur svakoj biljci. Fizički kostur biljke sastoji se od 3 dijela. tijela 1 koje je nepomično i opisuje korijen biljke, tijela 2 koje je vrh biljke i podložno je utjecaju sila te kuglastog zgloba koji povezuje tijelo 1 i tijelo 2.

Tijelo u simulacijskom svijetu opisano je s dva konfiguracijska objekta. Svaki konfiguracijski objekt objašnjava zaseban dio simulacije. Prvi konfiguracijski objekt su općenita svojstva tijela poput njegove inicijalne pozicije u simulacijskom svijetu i vrsta ponašanja tijela. Konfiguracijski razred za općenita svojstva tijela u OimoPhysics pokretaču je `RigidBodyConfig`. U korištenom fizičkom pokretaču postoje tri vrste ponašanja tijela - statično, dinamički i kinematičko. Na statična tijela ne utječe sila i ne mijenjaju svoju poziciju i rotaciju čak ni prilikom kolizije s drugim tijelima. Na dinamička tijela utječe sila i ona se ponašaju kao svi objekti u stvarnom svijetu - prilikom kolizije se odbijaju i rotiraju, a prilikom djelovanja sile ubrzavaju ili usporavaju u smjeru sile. Dinamička tijela reagiraju na koliziju sa statičnim i kinematičkim tijelima. Kinematička tijela ne reagiraju na kolizije, ali reagiraju na sile i njihovim utjecajem mogu mijenjati svoju poziciju i rotaciju.

```
1  const origin = new OIMO.Vec3(
2    plant.pos[0], plant.pos[1], plant.pos[2]
3  );
4
5  const baseConfig = new OIMO.RigidBodyConfig();
6  baseConfig.position = origin;
7  baseConfig.type = OIMO.RigidBodyType.STATIC;
8
9  let shapeConfig = new OIMO.ShapeConfig();
10 shapeConfig.geometry = new OIMO.BoxGeometry(
11   new OIMO.Vec3(0.1, 0.1, 0.1)
12 );
13
```

```
14 const base = new OIMO.RigidBody(baseConfig);
15 base.addShape(new OIMO.Shape(shapeConfig));
16
17 this.world.addRigidBody(base);
```

Isječak koda 3.2: Postavljanje tijela korijena biljke i njegovih svojstava

Za tijelo koje simulira korijen biljke pozicija tijela odgovara poziciji tijela u 3D prikazu. Vrsta tog tijela je statična. Ne mijenja svoju poziciju i na rotira se. Ovakvo ponašanje prati ponašanje stvarnih biljaka u normalnim atmosferskim uvjetima.

Drugi konfiguracijski razred je `ShapeConfig`. `ShapeConfig` opusuje geometriju modela u 3D prostoru. U fizičkim pokretačima uobičajeno je pojednostaviti geometriju modela što je više moguće kako bi brzina izvođenja bila što veća. Iz tog razloga su geometrije tijela obično jednostavni geometrijski oblici poput kvadra, kugle, piramide i stošca. Složeniji oblici mogu se dobiti dodavanjem dodatnih jednostavnih oblika na tijelo.

Tijelo koje simulira korijen biljke je statično i nema interakciju s ostalim tijelima, pa je u prezentiranom rješenju opisan samo kao mala kocka na dnu biljke. Postavljanje svojstava tijela korijena biljke i njegove geometrije prikazano je u isječku koda 3.2.

Tijelo koje simulira vrh biljke iako složenije, također ima vrlo jednostavan proces postavljanja početnih vrijednosti. Pozicija tijela je točno iznad tijela korijena biljke na y vrijednosti koja odgovara visini biljke. Vrsta ovog tijela je dinamičko, jer osim reagiranja na sile vjetra reagira i kod potencijalnih interakcija s ostalim biljkama. Promjene koje sadrži postavljanje početnih vrijednosti tijela vrha biljke u odnosu na tijelo korijena prikazan je isječkom koda 3.3.

```
1 ...
2 tipConfig.position = origin.add(new OIMO.Vec3(0, plant.height, 0));
3 tipConfig.type = OIMO.RigidBodyType.DYNAMIC;
4
5 ...
6 shapeConfig.geometry = new OIMO.SphereGeometry(
7   plant.collisionRadius
8 );
```

Isječak koda 3.3: Postavljanje tijela vrha biljke

Vrh biljke umjesto kocke ima kuglu. Iako neprecizno, kugla dobro opisuje ponašanje interakcije vrha biljaka i pogodno je za brzinu izvođenja.

Definicija kuglastog zgloba u OimoPhysics pokretaču također je jednostavna. Sastoji se od jednog konfiguracijskog objekta koji opisuje sva svojstva ovog ograničenja. Taj konfiguracijski razred je `SphericalJointConfig`. `init` metodi ovog konfiguracijskog objekta predajemo tijela na koja će se ovo ograničenje primjenjivati. U prezentiranom rješenju to su tijela `base` i `tip`. Osim tijela na koja se ograničenje primjenjuje potrebno je definirati i referentnu točku za to ograničenje. Kod simulacije se vrh biljke rotira oko korijena biljke, pa poziciju korijena biljke uzimamo kao referentnu točku.

```
1 const jointConfig = new OIMO.SphericalJointConfig();
2 jointConfig.init(base, tip, baseConfig.position);
3 jointConfig.springDamper = new OIMO.SpringDamper().setSpring(0, 0);
4 jointConfig.breakForce = 0;
5 jointConfig.breakTorque = 0;
6
7 const joint = new OIMO.SphericalJoint(jointConfig);
8 this.world.addJoint(joint);
```

Isječak koda 3.4: Postavljanje kuglastog zgloba

U ostatku konfiguracije definiran je i prigušivač opruge. Prigušivač opruge simulira ograničenje kao oprugu umjesto čvrstu vezu između tijela. U prezentiranom rješenju postavke prigušivača su postavljene na nulu. Ovakve postavke postavljaju ograničenje kao čvrstu vezu između dva tijela. Biljke možemo zamisliti kao opruge pa ovakav izbor djeluje nelogično. Odabir baš takvih parametara rezultat je ručne implementacije vrlo sličnog ponašanja biljke. Korištenje oba sustava za simulaciju elastičnosti rezultira fizičkim nekonzistencijama u ponašanju biljke.

Parametri `breakForce` i `breakTorque` opisuju silu i obrtni moment pod kojim bi ograničenje popustilo, odnosno prestalo raditi. Vrijednost nula u konfiguraciji signalizira fizičkom pokretaču da je ograničenje konstantno, odnosno da neće prestati vrijediti bez obzira na sile koje djeluju na tijela. Cijelovito postavljanje vrijednosti ograničenja kuglastog zgloba prikazano je isječkom koda 3.4.

Nakon početnog postavljanja kostura biljke potrebno je definirati ponašanje biljke u simulaciji. Definicija ponašanja prikazana je u isječku koda 3.5. U simulaciji prolazimo kroz sve kuglaste zglobove u kosturu biljke i izvršavamo simulaciju za svaki od njih. U jednostavnim primjerima postoji samo jedan zglob za svaku od biljaka, ali na ovaj način je sačuvana mogućnost simuliranja mnogo složenijih konfiguracija.

Na početku jedne iteracije simulacije uzimamo tijela na koja je primijenjeno ograničenje kuglastog zgloba. Na drugo tijelo (vrh biljke) primjenjujemo silu vjetra. Sila vjetra je vektor neke duljine u 3D prostoru. Duljina vektora odgovara snazi koju vjetar ima u nekom trenutku. Nakon primjene sile vjetra, na tijelo se primjenjuje izrazito mala sila koja djeluje suprotno od vektora brzine tijela. Tijelo zbog te sile konstantno usporava i time rješava problem beskonačne simulacije čak i za vrlo male i kratkotrajne početne sile.

```
1  simulateMovement() {
2    for (const joint of this.skeleton.joints) {
3      const rigid1 = joint.getRigidBody1();
4      const rigid2 = joint.getRigidBody2();
5
6      this.applyWind(rigid2);
7      this.applyInnerFriction(rigid2);
8
9      // we need relative position of joint
10     const pos1 = rigid1.getPosition();
11     const pos2 = pos1.sub(rigid2.getPosition());
12
13     const force = Math.abs(this.forceFactor(pos2));
14     rigid2.applyForceToCenter(new OIMO.Vec3(0, force, 0));
15   }
16 }
```

Isječak koda 3.5: Fizička simulacija ponašanja biljke.

Nakon završene simulacije vjetra računamo silu koja objašnjava ponašanje biljke na vjetru. Sila je detaljno objašnjena u poglavlju 3.2. Funkcija `forceFactor` računa iznos sile kojom djelujemo, a opisana je formulama 3.1 i 3.2.

4. Simulacija ponašanja generiranog modela

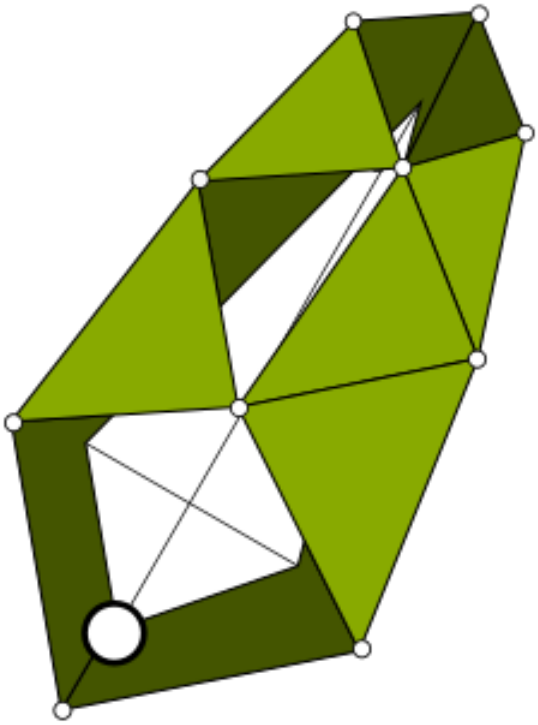
4.1. Povezivanje fizičkog modela i modela prikaza

Do sada su definirana dva odvojena sustava, sustav fizičke simulacije i sustav prikaza. Fizički sustav je definiran preko dva tijela i jednog zgloba koji imaju odnose roditelj- dijete. Tijelo koje ima ulogu korijena biljke je roditelj, tijelo koje simulira vrh biljke je dijete tog tijela, a ograničenje koje ih povezuje je veza između njih. U računalnoj grafici postoji sličan koncept zapisa poze nekog složenog sustava. Takav način hijerarhijskog zapisa naziva kostur (engl. *skeleton*), u kojem su pohranjeni podaci o djeci - kostima (engl. *bones*).

Fizički sustav opisan u cjelini 3 možemo opisati kosturom od dvije kosti. Prva kost nema roditelja, pozicionirana je na referentnoj točki fizičkog sustava i ima konstantnu rotaciju koja uvijek gleda prema pozitivnoj y osi (prema gore). Druga kost je dijete prve kosti i rotirana je tako da uvijek gleda u smjeru vektora između tijela 1 i tijela 2.

Nakon definiranja kostura, potrebno je definirati način na koji će se grafički model mijenjati prema promjenama pozicija i rotacija kostiju. Ovaj postupak se zove omatanje (engl. *skinning*). Omatanje se vizualno može objasniti kao dodavanje utjecaja kosti na pojedine točke u prostoru, tako da je kost uvijek bude unutar volumena koje te točke zatvaraju. Primjer ilustracije omatane mreže točaka prikazan je na slici 4.1.

Svaka kost je definirana preko matrice. U matrici su podaci o poziciji kosti te njenoj orijentaciji. Način na koji točka unutar prikaza poprima te karakteristike je jednostavnim množenjem pozicije točke s matricom kosti.



Slika 4.1: Ilustracija omotane mreže točaka oko kosti

Kod savršenog omatanja utjecaj na jednu točki ima samo jedna kost. U praksi, kao i u prikazanom rješenju to nije uvijek slučaj. Korištenjem savršenog omatanja dobili bi biljku koja se u korijenu uvijek rotira prema smjeru vjetra. Takva biljka bila bi ravna i ne bi djelovala prirodno.

Način na koji se rješava taj problem je dodavanjem težina utjecaja kosti na pojedinu točku u modelu. Težina je vektor decimalnih brojeva između jedan i nula, gdje jedinica označava potpun utjecaj neke kosti na točku, a nula označava da ta kost nema nikakav utjecaj na točku.

U primjeru simuliranja biljke na vjetru, točke koje su bliže korijenu su pod manjim utjecajem kosti koja uzrokuje rotaciju od točaka koje su bliže vrhu. Rezultat takve raspodjele težina jasno je vidljiv na slici 3.2.

Zbog jednostavnosti problema, način na koji su težine raspodijeljene je linearan s obzirom na visinu biljke. Točke na samom dnu biljke imaju težinu prema rotirajućoj kosti nula, a prema statičnoj kosti jedan. Rastom visine, odnosno y vrijednosti pozicije pojedine točke, težina kojom statična kost utječe na točku opada, a težina kojom

rotirajuća kost utječe raste. Izračun težine za pojedinu točku prikazan je formulom 4.1.

$$w = vertexHeight/plantHeight \tag{4.1}$$

4.2. Programska implementacija modela

Izračun pozicije točke u prostoru pod utjecajem kosti izvršava se na grafičkoj kartici. Prošireni program sjenčanja s omatanjem prikazan je u isječku 4.1.

```
1  ...
2  attribute vec4 a_weights;
3
4  ...
5  uniform mat4 u_bone;
6
7  void main() {
8    float counterWeight = 1.0 - a_weights[0];
9    gl_Position = u_worldViewProjection * (
10     u_bone * a_position * a_weights[0] +
11     a_position * counterWeight
12   );
13
14   ...
15 }
```

Isječak koda 4.1: Proširenja programa za sjenčanje omatanjem

Kost korijena biljke je konstantna pa ju zbog jednostavnosti ne šaljemo prema grafičkoj kartici.

`a_weights` je vektor utjecaja kostiju na točku u prostoru. Koristimo samo prvu vrijednost iz vektora jer imamo samo jednu kost koja vrši animaciju. Korištenje vektora je zadržano zbog potencijalnih nadogradnji u budućnosti. Također, korištenje vektora ne utječe na performanse jer se prema grafičkoj kartici šalje samo jedan podatak, a ostala tri su pretpostavljene vrijednosti. Ovakav vektor u teoriji podržava 4 različite kosti.

`u_bone` je matrica rotirajuće kosti u prikazu. U glavnom dijelu programa izračunava se protu-težina. Odnosno, koliko utjecaja na točku u prostoru ima statična kost. Statična kost je matrica identiteta i njenim množenjem pozicije točke u prostoru ne dobivamo nikakvu razliku pa je to množenje u programu za sjenčanje izostavljeno.

Kada bi statična kost mijenjala svoju poziciju ili rotaciju, nju bi također morali poslati prema grafičkoj kartici i izračun pozicija točke bi izgledao kao što je prikazano u isječku koda 4.2.

```
gl_Position = u_worldViewProjection * (
  u_bone_1 * a_position * a_weights[0] +
  u_bone_2 * a_position * a_weights[1]
);
```

Isječak koda 4.2: Prošireno računanje pozicije točke na dvije kosti

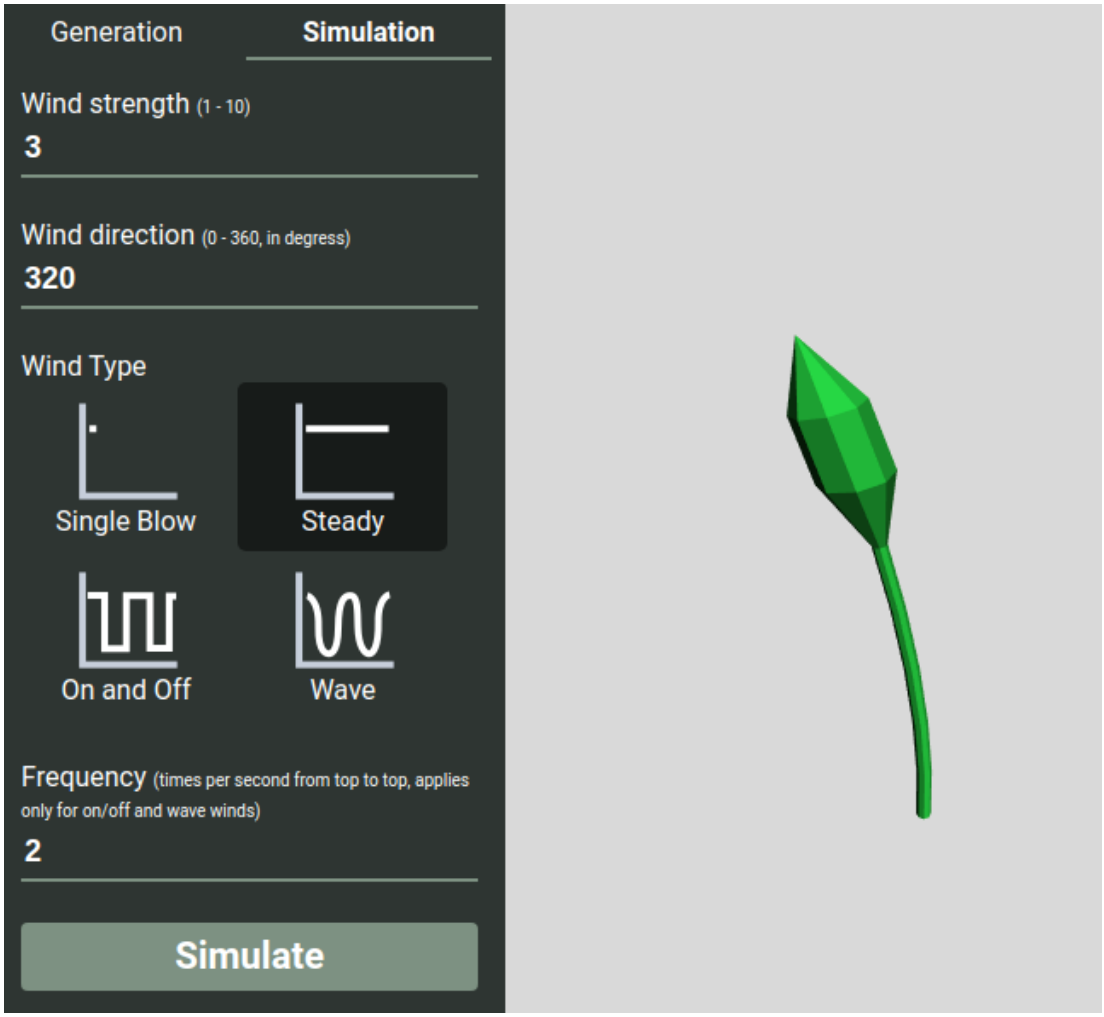
Kako bi grafička kartica ispravno izračunala pozicije točaka, potrebno joj je u svakoj iteraciji poslati matricu rotirajuće kosti. Slanje matrice prema grafičkoj kartici prikazano je isječkom koda 4.3.

```
let boneMatrix = mat4.xRotation(angles[0]);
boneMatrix = mat4.zRotate(boneMatrix, angles[2]);

const uBone = this.gl.getUniformLocation(
  this.program, shaderBinding
);

this.gl.uniformMatrix4fv(
  uBone,
  false,
  boneMatrix
);
```

Isječak koda 4.3: Slanje dodatnih informacija prema grafičkoj kartici

4.3. Različiti parametri simulacije

Za bolju ocjenu rezultata potrebno je testirati više različitih parametara simulacije i usporediti njihov utjecaj na simulaciju s ponašanjem u stvarnom svijetu. U simulaciji možemo utjecati na snagu vjetra, smjer vjetra, vrstu puhanja vjetra i za neke vrste puhanja možemo mijenjati frekvenciju puhanja. Prikaz sučelja za podešavanje parametara simulacije prikazan je na slici 4.2.



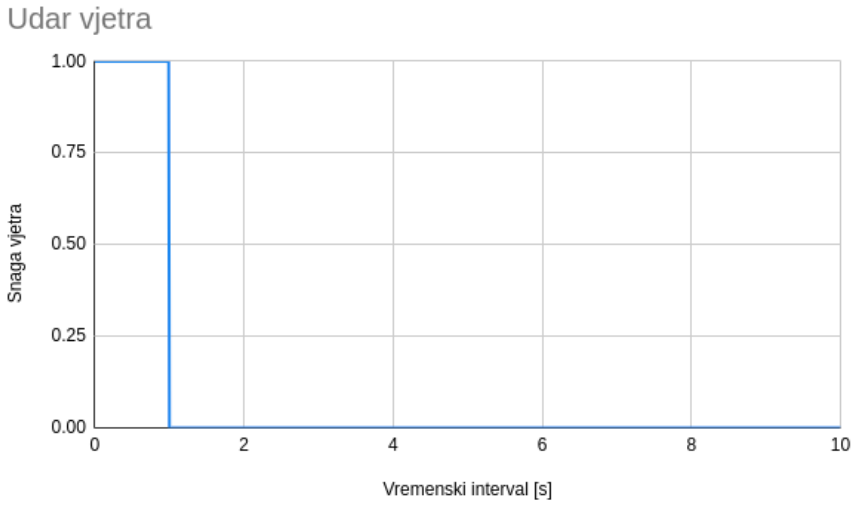
Slika 4.2: Prikaz sučelja za simuliranje vjetra

Snaga vjetra objašnjava kojim intenzitetom djelujemo na tijelo vrha biljke u jedinici vremena. Jedinica vremena u fizičkom pokretaču se naziva korak i traje 32 ms. Snaga vjetra 10 odgovara sili od 10 N.

Smjer vjetra je predstavljen kutem iz kojeg dolazi u odnosu na biljku. Vjetar pod kutem 0 stupnjeva djeluje od negativne prema pozitivnoj z osi. Vjetar pod kutem 90

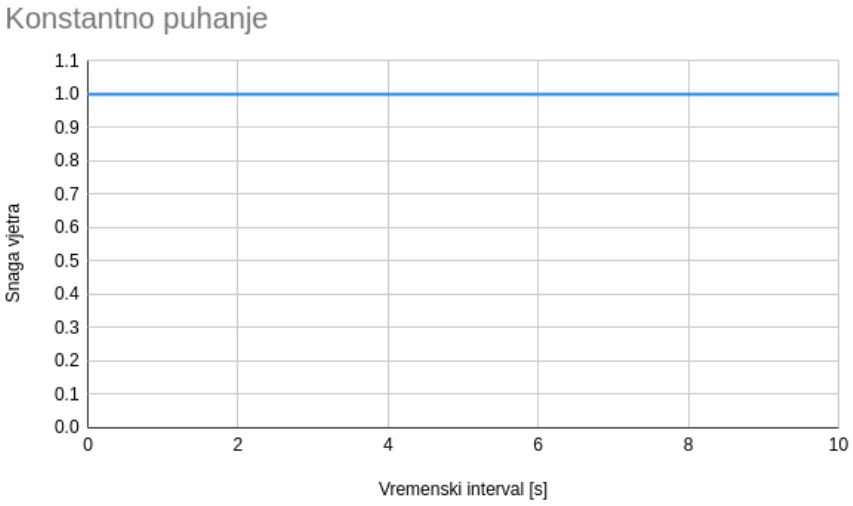
stupnjeva djeluje od negativne x osi prema pozitivnoj x osi.

U simulaciji postoje 4 vrste vjetra. Udar vjetra puše kratko vrijeme u jednom smjeru i nakon toga je njegovo puhanje gotovo. Graf snage puhanja kod udara vjetra prikazan je na slici 4.3.



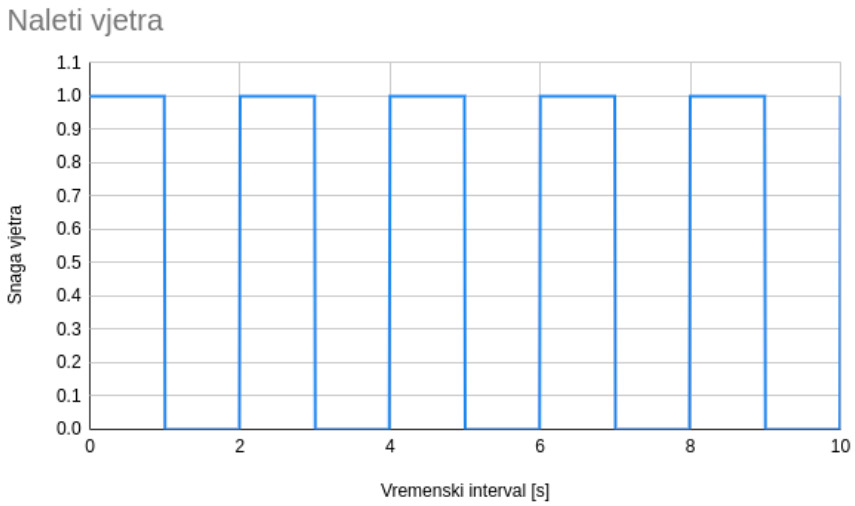
Slika 4.3: Funkcija udara vjetra uz parametar snage 1

Konstantno puhanje djeluje istim inzetitetom u jednom smjeru bez prestanka. Graf konstantnog puhanja prikazan je na slici 4.4. Konstantno puhanje i udar vjetra su jednostavne simulacije korištene u razvoju i ne simuliraju dobro stvarnu pojavu vjetra.



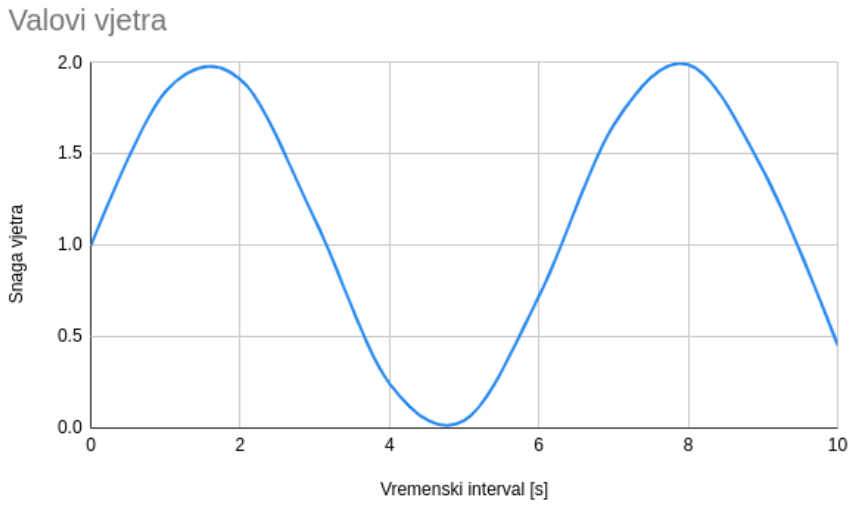
Slika 4.4: Funkcija konstantog puhanja vjetra uz parametar snage 1

Puhanje u naletima vjetra jedan je od dinamičkih načina puhanja. Ovaj vjetar naivno modelira vjetre poput udara bure. Naivno modelira jer radi pretpostavku da vjetar potpuno stane između svakog udara. Na ovaj model utječe i parametar frekvencije. Na slici 4.5 prikazan je graf jednog takvog vjetra.



Slika 4.5: Funkcija vjetra u udarima uz parametar snage 1 i frekvencije 0.5

Puhanje u valovima najrealističnija je simulacija vjetra. U ovakvom načinu vjetar polako puše sve većom snagom oko nazivne snage i kada dostigne dvostruku nazivnu snagu krene lagano opadati. Ovo je modelirano sinusoidom oko nazivne snage vjetra s amplitudom snage vjetra. Na ovaj model također utječe parametar frekvencije.



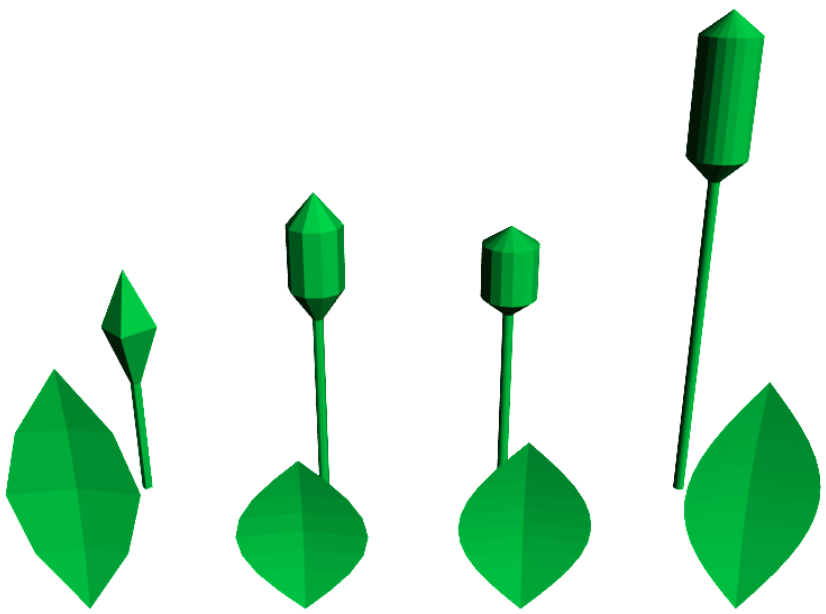
Slika 4.6: Funkcija vjetra u valovima uz parametar snage 1 i frekvencije 0.16

Utjecaj parametara na simulaciju opisan je u poglavlju 5.2.

5. Ocjena rezultata

5.1. Realističnost prikaza

Realističnost prikaza nije bila primarna zadaća ovog rada. Generirane biljke ne izgledaju osobito prirodno bez obzira na razinu detalja kojom su generirane. Primjer generiranih biljaka vidljiv je na slici 5.1.



Slika 5.1: Prikaz modela u različitoj razini detalja

Prvi razlog je primjena konstantnog sjenčanja poligona. U ovakvom načinu sjenčanja, normale za pojedine vrhove računamo kao normalu površine (poligona) kojeg oni zatvaraju. Izračunata vrijednost normale se pripisuje svim vrhovima poligona. Korištenjem konstantnog sjenčanja poligona postižemo jasno vidljivu granicu između pojedinih površina. Takav način prikaza, osim što je vrlo brz u izvođenju, znatno je

olakšao razvoj prezentiranog programskog rješenja jer daje programeru jasan uvid u poziciju pojedinih vrhova na ekranu i njihovo ponašanje.

Bez obzira na prednosti u razvoju koje ovakav pristup implicira, prikazani modeli izgledaju umjetno i izrazito su pravilni. Pravilnost modela je najveći uzrok neprirodnog izgleda generiranih modela, jer na biljkama rijetko očekujemo savršenu simetriju oko geometrijskih osi.

Drugi razlog je nedostatak detalja na generiranim modelima biljaka. Ovaj razlog dodatno naglašava pravilnost biljaka i smanjuje osjećaj realnosti prikaza. Smanjenje broja detalja donijelo je iste prednosti kao i prethodni pristup - lakši razvoj i bolje performanse po cijenu realističnosti prikaza.

Treći razlog je izolacija prikaza. Biljke su prikazane kao centralni i jedini dio prikaza. Ovakav vakuum u prostoru dodatno naglašava oba prije spomenuta problema. Nedostatak simulacije atmosfere (prašina, distorzija zraka svjetlosti kao posljedica vlage u zraku itd.) uzrokuje dojam statičnosti biljaka bez obzira na animirani prikaz ponašanja vjetra. Izostavak navedenih značajki, kao i prethodna pojednostavljenja, olakšalo je razvoj programskog rješenja i oslobodilo vrijeme za ostvarenje kvalitetnije simulacije vjetra.

5.1.1. Mogućnosti poboljšanja

Problem sjenčanja može se riješiti primjenom nekog drugog načina sjenčanja modela korištenjem Gouraudovog sjenčanja. Korištenje ove metode sjenčanja eliminiralo bi vidljivost pojedinih vrhova na modelu i interpolacijom između vrhova osiguralo gladak prijelaz između osjenčanih i ne osjenčanih dijelova modela.

Biljke u stvarnosti rijetko imaju ikakve oštre bridove između svojih strana i zaglađivanje intenziteta osvjetljenja Gouraudovog sjenčanja između vrhova bi rezultiralo uvjerljivijim prikazom kod modela generiranih u većoj razini detalja. Kod modela generiranih manjom razinom detalja, rubovi samog modela bi ostali oštri iako je model zaglađen. Sraz između rubova i unutrašnjosti modela može uzrokovati smanjenje realističnosti prikaza.

Dodavanje malih varijacija kod generiranja modela poboljšalo bi realističnost prikaza. Mali istupi točaka od centra simetrije dali bi biljkama prirodniji izgled uvođenjem nesavršenosti i raznovrsnosti biljaka. Osim dodavanja istupa u poziciji točaka u modelu, istupi se mogu dodati i na boje pojedinih površina, gdje bi neke površine bile više ili manje intenzivne boje od drugih. Dodavanje boja u kombinaciji s Gouraudovim sjenčanjem dodatno bi pojačalo realističnost prikaza.

Dodavanje varijacija je programski izrazito jednostavno za implementaciju ali uzrokuje usporenje izvođenja, jer se svaki model zbog svoje unikatnosti mora preslikati u memoriju grafičke kartice umjesto korištenja istog modela za sve biljke iste vrste. Kompromis se može postići na sredini - imati limitiran broj različitih modela iste vrste biljke koji se nasumično odabere za svaku biljku prilikom njenog kreiranja.

Dodavanje atmosferskih efekata na simulaciju bi uvelike pomoglo realističnosti prikaza i dojmu živosti biljaka. Ovaj pristup rješavanju nerealističnosti prikaza je najkompleksniji od ponuđenih alternativa i zahtjeva razvoj nekolicine popratnih sustava, ali dao bi najveći doprinos realističnosti prikaza. Primjeri popratnih sustava koje je potrebno razviti uključuju: sustav ukrasnih čestica, sustav efekata nakon iscrtavanja (engl. *post-processing effects*) i druge.

5.2. Realističnost fizičke simulacije

Bez obzira na jednostavnost implementacije fizičke simulacije i snažne pretpostavke uključene u nju, fizička simulacija daje zadovoljavajuće rezultate. Ponašanje prati stvarno prirodno gibanje u tri dimenzije i ne potiče osjećaj umjetnosti ili ograničenosti prolaskom kroz prostor. Simulacija dobro modelira utjecaj visine biljke na njezinu simulaciju na vjetru.

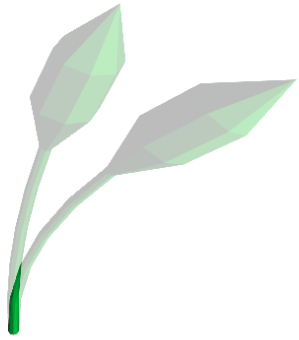
Na udarima vjetra visoke biljke rade velike i snažne zamahe i polagano gube energiju za nastavak daljnjeg osciliranja. Niske biljke rade kraće zamahe ali većom frekvencijom.

Kod konstantnog puhanja, biljke osciliraju prema točki konvergencije sile puhanja, sile teže i sila otpora unutar stabljike. Visoke biljke očekivano imaju veću amplitudu i manju frekvenciju oko te točke od niskih biljaka.

Puhanje s naletima vjetra također daje realistične rezultate. Niske biljke u naletu vjetra se brzo poravnavaju prema točki konvergencije i osciliraju oko nje, a po prestanku puhanja osciliraju oko centra ravnoteže velikom frekvencijom. Visoke biljke imaju veću tromost i rade veće oscilacije oko točke konvergencije vjetra i otpora biljke prilikom naleta vjetra. Kad vjetar prestane, naprave manje oscilacija oko centra ravnoteže prije nego vjetar ponovno počne.

Slika 5.2: Kaotično ponašanje niske biljke prilikom vjetra u valovima

Kad je simulacija vjetra realistična (takva da vjetar dolazi i prolazi u valovima - postepeno raste u snazi, i nakon toga postepeno opada u snazi) također imamo realističnu simulaciju. Visoke biljke održavaju smjer i neprestano osciliraju prema središtu između točaka centra ravnoteže otpora biljke, i konvergentne točke sile vjetra, ravnoteže i unutarnjeg otpora biljke. Niske biljke kod ovakvog vjetra djeluju kaotično i osciliraju velikom frekvencijom i amplitudom, sa središtem oscilacije koji vidno varira između konvergentne točke i centra ravnoteže biljke. Razlika u ponašanju je vidljiva na slikama 5.2 i 5.3



Slika 5.3: Stabilno ponašanje visoke biljke prilikom vjetra u valovima

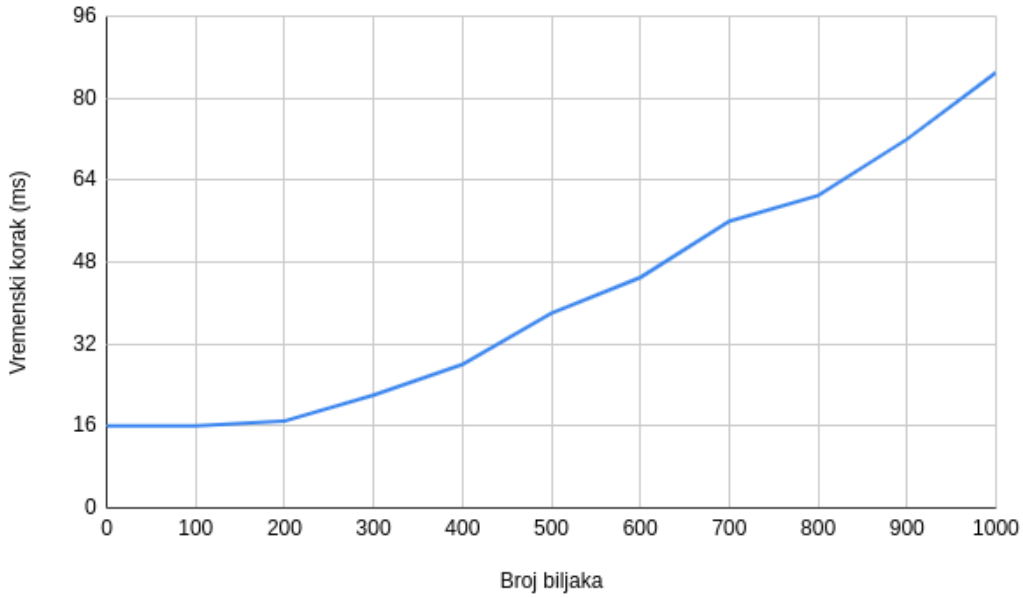
5.2.1. Poboljšanje realističnosti simulacije

Još realističnija simulacija se može postići dodavanjem većeg broja zglobova u fizički kostur biljke. Dodavanjem većeg broja zglobova dobili bi mogućnost simulacije pregiba u stabljici biljke. U prirodi kod jednostavnih biljaka ovaj fenomen nije uvijek jasno vidljiv, ali možemo ga uočiti kad je vjetar u rezonantnoj frekvenciji sa stabljikom biljke.

Dodatnu kontrolu možemo postići dodavanjem težina pojedinim zglobovima. Tako da na neke dijelove kostura vjetar ima jači utjecaj nego na druge. Primjer takve biljke je zvončić, kod kojeg je utjecaj vjetra puno jasnije vidljiv na cvijetu nego na stabljici biljke.

5.3. Brzina izvođenja

Brzina izvođenja linearno opada s brojem simuliranih biljaka. Najveći utjecaj na brzinu izvođenja ima fizička simulacija. Generiranje biljaka se odvija na samom početku i nema nikakvog utjecaja na brzinu izvođenja nakon početnog perioda generiranja. Iscrtavanje je zbog jednostavnosti prikaza vrlo jeftino i usporava izvođenje tek na velikom broju biljaka ili na vrlo visokoj razini detalja. Graf trajanja vremenskog koraka vidljiv je na 5.4.



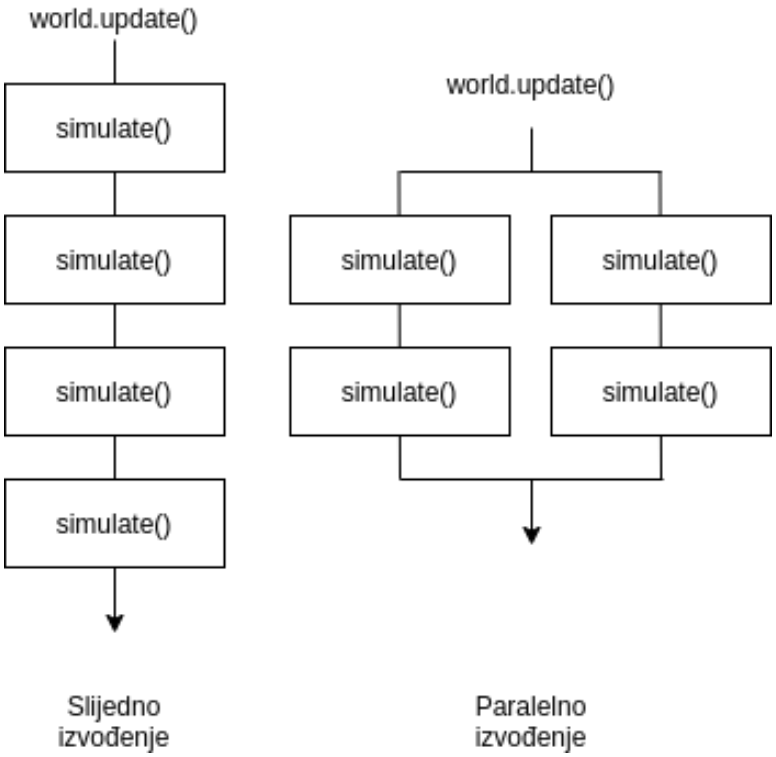
Slika 5.4: Trajanje vremenskog koraka u ovisnosti o broju biljaka u simulaciji

5.3.1. Pristupi poboljšanju brzine izvođenja

Fizička simulacija ima najveći utjecaj na brzinu izvođenja pa su prijedlozi za ubrzanje programskog rješenja fokusirani na taj dio.

Paralelno izvođenje fizičke simulacije

Trenutna programska implementacija u obzir uzima i kolizije samih biljaka. Fizička simulacija se odvija slijedno za svaku od biljaka i eventualni dodir nakon simulacije neke od narednih biljaka može imati utjecaj na onu prvu. Ako ne marimo za interakciju između samih biljaka i odlučimo je zanemariti, jednostavan način za ubrzanje fizičke simulacije je paralelizam. Na 5.5 je prikazan dijagram slijednog i paralelnog izvođenja (na dvije dretve) i vidljivo je da paralelni primjer završava u dva koraka dok slijedni algoritam završava u jednom koraku.



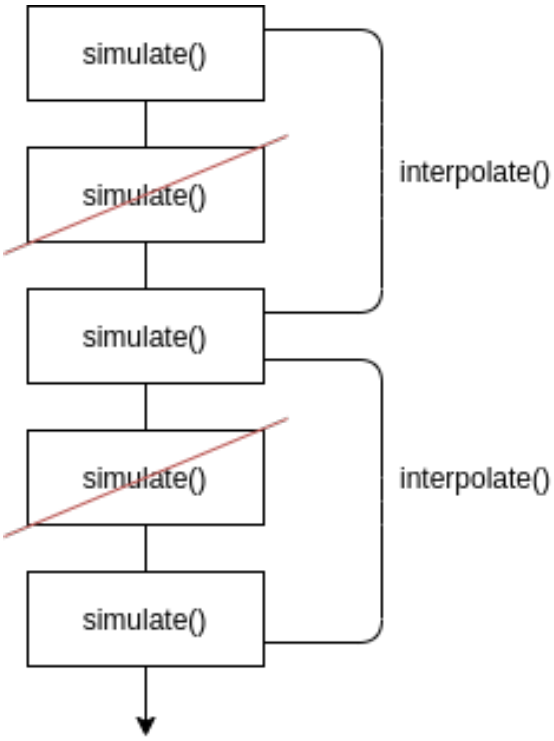
Slika 5.5: Dijagram slijednog i paralelnog izvođenja

Na ovaj način svaka biljka (ili grupa biljaka) može imati svoj fizički procesor koji će se brinuti o njezinoj simulaciji i cijeli proces se završava u manje koraka.

Smanjenje rezolucije simulacije i zaglađivanje rezultata

Brzinu izvođenja možemo i povećati tako da ne ažuriramo fizičku simulaciju biljke u svakom koraku. Kod ovakvog pristupa stanje svake biljke izračunavamo nakon nekoliko koraka umjesto na svakom, a rezultate u međukoracima zagladimo. U primjeru 5.6 vidimo slijed simulacije jedne biljke kroz vremenske korake. Prilikom simulacije trebamo osigurati da je vremenski razmak za koji simuliramo točno onoliko koraka koliko će trajati do sljedeće simulacije i to uzrokuje smanjenje rezolucije simulacije. Ako bi vremenski korak simulacije ostao jednak kao da simuliramo biljku u svakom simulacija bi se odvijala usporeno.

Bitno je ostvariti da se nikad istovremeno ne simuliraju i zaglađuju svi modeli nego da je postupak naizmjeničan. Za primjer dvije biljke, dok se jedna biljka simulira druga se zaglađuje i obratno.



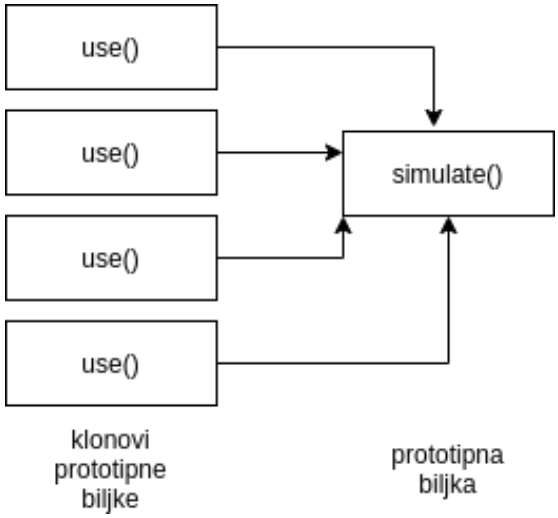
Slika 5.6: Dijagram simulacije sa zaglađivanjem

Problem kod ovakvog pristupa je kašnjenje prikaza nad stvarnim stanjem fizičke simulacije. Fizička simulacija je uvijek barem jedan korak ispred prikaza. Ovo može uzrokovati naizgledno podrhtavanje ako želimo zadržati simulaciju interakcija između biljaka, a biljka se pomaknula nakon što je dobivena zadnja točka za interpolaciju. Ako

ne marimo za interakciju između biljaka, ovaj problem možemo zanemariti. Iako će animacija i dalje kasniti za stvarnim fizičkim stanjem, promatraču to neće biti vidljivo.

Izračunavanje sličnosti modela i grupiranje izračuna

Ako nam nije bitna interakcija između biljaka i želimo drastično smanjiti utjecaj fizičke simulacije na brzinu izvođenja možemo izdvojiti prototipne biljke iz simulacije i simulirati samo njih. Sve ostale biljke, ovisno o sličnosti će koristiti podatke simulacije te biljke kao svoje. Primjer takvog sustava vidljiv je na slici 5.7



Slika 5.7: Dijagram simulacije sa grupiranjem

Na ovaj način potrebno je simulirati samo nekoliko biljaka umjesto svih ali ovakav pristup će rezultirati time da simulacija izgleda nerealistično jer će sličnosti između ponašanja biljaka biti identične u isto vrijeme i na istim uvjetima što narušava dojam realizma. Potencijalno rješenje ovog problema je da kod nekih biljaka unesemo kašnjenje nad prototipnom biljkom. Tako smo eliminirali problem dojma da se sve ponavlja, ali smo uveli problem da simulacija ne djeluje toliko responzivno.

6. Zaključak

Generiranje i simulacija trave i niskog raslinja je moguća, ali i takav način izrade modela iziskuje veliku količinu vremena. Nakon početnog uloženog vremena, proces same izrade je ubrzan ali zahtjeva daljnji razvoj infrastrukture jer je za svaku novu karakteristiku neke biljke potrebno vrijeme kako bi se ona matematički opisala i razvila. Rezultati dobiveni ovom tehnikom mogu izgledati siromašno i potrebno je uložiti dodatan trud kako bi se takvi modeli efektivno iskoristili.

Ovakav model generiranja i simuliranja modela bi ipak, uz neka poboljšanja, mogao biti isplativ u izrazito velikim igrama ili kao rješenje koje bi se koristilo u više igara. Ulaganjem vremena u ovakav sustav generiranja modela moguće je dostignuti razinu gdje je gotovo svaku biljku moguće vrlo jednostavno opisati s par pravila. Model simulacije vjetra koji je istražen je jednostavan i uz dodatne optimizacije bi se efektivno mogao koristiti u igrama, čak i sa ručno modeliranim modelima biljaka.

LITERATURA

B. Bywalec D. M. Bourg. *Physics for Game Developers: Science, math, and code for realistic effects*. O'Reilly, Sebastopol, California, SAD, u drugom izdanju, 2001.

R. Lea K. Matsuda. *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL*. Addison-Wesley, Boston, Massachusetts, SAD, u prvom izdanju, 2013.

POPIS SLIKA

POPIS ISJEČAKA KODA

Proceduralno generiranje trave i niskog raslinja

Sažetak

Proceduralno generiranje modela trave i niskog raslinja te simulacija njihovog ponašanja na različitim atmosferskim utjecajima je kompleksan matematički problem. Promatranjem izgleda biljaka i njihovog ponašanja moguće je uočiti pravilnosti. Korištenjem tih pravilnosti kao pretpostavki u izgradnji modela prikaza i fizičkog modela simulacije, moguće je vrlo jednostavnim matematičkim konstrukcijama doći do modela koji vjerno opisuje stvarno ponašanje. U ovom radu su istražene i implementirane tehnike za generiranje 3D modela trave i niskog raslinja. Ponuđen je razvojni okvir za izradu proizvoljnih 3D modela te osnovni sustav simulacije vjetra. Ocijenjeni su i prezentirani rezultati. Istražene su neke mogućnosti za nastavak istraživanja.

Ključne riječi: proceduralno generiranje, simulacija vjetra, omatanje modela, fizička simulacija

Procedural generation of grass and low vegetation

Abstract

Procedural generation of grass and low vegetation models, with simulated behavior under different atmospheric conditions is a complex mathematical problem. By observing plants it is possible to deduct some regularities about their appearance and behavior. Using those regularities as assumptions while building graphics and physics model it is possible to achieve level of realism with just simple mathematical constructions. This paper researched and implemented some of the techniques used for generating 3D models of grass and low vegetation. It offers a framework for creating a wide range of different plants and basic system for wind simulation. Results were measured and are presented. Paper explored some of the possibilities for the continuation of research.

Keywords: procedural generation, wind simulation, model skinning, physics simulation