SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1598

Algoritmi za proceduralno generiranje i prikaz drveća

Hrvoje Nuić

Zagreb, lipanj 2018

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1598

Algoritmi za proceduralno generiranje i prikaz drveća

Hrvoje Nuić

Zagreb, lipanj 2018

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

Zagreb, 9. ožujka 2018.

DIPLOMSKI ZADATAK br. 1598

Pristupnik: **Hrvoje Nuić (0036470074)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Algoritmi za proceduralno generiranje i prikaz drveća**

Opis zadatka:

Proučiti postupke generiranja biljaka, posebice drveća u računalnoj grafici. Proučiti mehanizme upravljanje postupkom generiranja s ciljem postizanja željenog oblika krošnje. Razraditi proučene postupke i ostvariti algoritme za proceduralno generiranje i prikaz drveća temeljeno na ciljnom željenom obliku. Simulirati utjecaj djelovanje sile na stvorene modele drveća te animirati prikaz. Na različitim primjerima prikazati ostvarene rezultate. Diskutirati utjecaj različitih parametara. Načiniti ocjenu rezultata i implementiranih algoritama.
Izraditi odgovarajući programski proizvod. Koristiti programski jezik C++ i grafičko programsko sučelje OpenGL. 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.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 29. lipnja 2018.

Mentor:

Prof. dr. sc. Željka Mihajlović

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Sadržaj

1 Popis slika

2 Uvod

Modeliranje i animiranje stabala za moderne filmove, računalne igre i simulacije je iscrpan posao. Stabla su često reprezentirana kao statični objekti s kojima likovi i ostala okolina imaju minimalnu interakciju. Glavni razlog je kompleksnost izrade strukture stabla i upravljanja njome na način da odgovara stvarnosti. Procijenjeno je da odraslo stablo hrasta lužnjaka može imati i do 100,000 grana, a visina im doseže 40 metara što govori o kompleksnosti potrebnih modela.

Na tržištu postoje alati koji olakšavaju izradu modela i rezultati su uglavnom foto realistični, ali proces je i dalje dugotrajan. Postoje proceduralni algoritmi koji se mogu prilagoditi na način da popunjavaju željeni model krošnjom. Također, u proučenim alatima nedostaje mogućnost prirodnog savijanja grana kojima bi se i dalje olakšalo upravljanje modelom.

Cilj ovoga rada je istražiti mogućnosti i implementirati algoritme pomoću kojih se stabla mogu brzo i kvalitetno izgraditi, algoritme pomoću kojih se stablo može prikazati te algoritme koji omogućavaju korisniku upravljanje strukturom stabla.

3 Opis implementacije

3.1 Korištene tehnologije i alati

U izradi programskog proizvoda korišteno je programsko okruženje Microsoft Visual Studio 2017. Program je implementiran pomoću C++ programskog jezika, te je korišteno grafičko programsko sučelje OpenGL verzije 4.5. Korišten je OpenGL jezik za sjenčanje na grafičkoj kartici.

Korištena je biblioteka OpenGL Mathematics (GLM) koja olakšava uporabu vektora, matrica i operacijama nad njima. Korištena je biblioteka DevIL 1.8. za učitavanje slika iz datoteka koje se koriste za teksturiranje stabla i okruženja. Za olakšavanje stvaranja prozora, komunikaciju s vanjskim uređajima računala, korištenje tajmera i pripremu konteksta za OpenGL sučelje korištena je biblioteka GLFW 3.2.1.

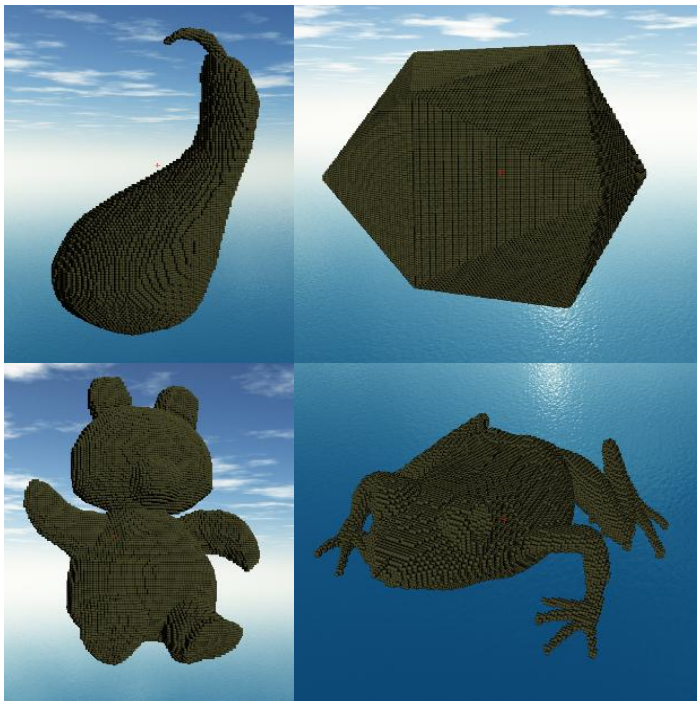
3.2 Pregled funkcionalnosti programa

Proces izgradnje stabla počinje odabirom željenog modela iz datoteke. Model se prosljeđuje dijelu programa zaduženog za vokselizaciju. Svaki model se vokselizira kako bi se ubrzalo izvođenje generatora. Vokselizirani model se zajedno sa željenim parametrima prosljeđuje generatoru stabla. Generiranom stablu se zatim unutar programske petlje osvježava pozicija grana na temelju utjecaja sile te se prikazuje na ekran korisniku. Svaki generator stabla je implementiran tako da se može pozvati sa željenom veličinom krošnje, visinom na kojoj se krošnja treba nalaziti, te vokseliziranim modelom. Na temelju tih parametara generator izrađuje stablo. Kako bi se stablo stavilo u prirodni kontekst iscrtava se pozadina s oblacima i morem.

Korisnik se može pomoću tipkovnice kretati u svim smjerovima unutar scene. Pomoću miša omogućeno je rotiranje i zumiranje pogleda kamere. Na ekranu se prikazuje kursor koji označava središte ekrana i pomoću njega korisnik može odabrati i povući granu u željenom smjeru. Korisniku je omogućeno ponovno generiranje stabla pritiskom na tipku R. Pomoću tipki X i C omogućeno je mijenjanje debljine grana na cijelom stablu. Unutar programa omogućeno je mjerenje broja sličica po sekundi i taj podatak se ispisuje korisniku.

4 Učitavanje i vokselizacija modela

Kako bi generatori stabla znali koji oblik krošnje stablo mora poprimiti, potrebno je učitati ili izraditi podatke o modelu. Za jednostavna geometrijska tijela poput kvadra ili elipsoida implementirano je proceduralno generiranje modela koji se mogu koristiti za izradu krošnje. Za kompliciranе modele omogućeno je učitavanje podataka o modelu iz datoteke te daljnja obrada.



Slika 1. Vokselizirani modeli

Prilikom generiranja krošnje željenog oblika najviše je upita nalazi li se neka točka unutar modela. Vokselizirani model je savršen oblik podataka za taj zadatak jer se ispitivanje izvršava u konstantnom vremenu. Negativna strana vokseliziranog modela je ta što se gubi preciznost jer svaka ćelija koja može imati samo jednu vrijednost reprezentira sve točke u njenoj okolini. Druga negativna strana je to što jednostavni modeli koji bi inače zauzeli jako malo prostora u memoriji reprezentirani pomoću ploha u ovom slučaju zauzimaju puno više memorijskog prostora.

Cilj vokselizacije je pretvoriti model reprezentiran pomoću ploha u model reprezentiran kockama određenog volumena. U radu sam odabrao da se svaki model reprezentira pomoću

rešetke veličine 256x256x256. Svaka ćelija je logički tip podatka koja zauzima jedan bajt, te ako je vrijednost istinita to označava da se na tom mjestu nalazi model, a inače je prazan prostor. Prilikom vokselizacije se odbacuju podatci vezani uz teksturiranje i sjenčanje modela. Neovisno o složenosti vokseliziranog modela on će na kraju biti veličine od 16MB. Memorijski prostor bi se mogao smanjiti na 2MB ukoliko bi se umjesto logičkog tipa podataka koristio tip podataka bitset. Korištenjem bitset podataka se ne dobiva na brzini jer procesori u modernim računalima ne mogu adresirati podatke manje od bajta, a kako je glavni razlog korištenja vokseliziranog modela brzina testiranja pripadnosti neke točke modelu nema potrebe za korištenjem bitset tipa podatka.

Postupak algoritma je takav da za svaku ćeliju ispita nalazi li se unutar modela ili ne. Ispitivanje se provodi tako da se iz ćelije povuče polupravac i izračuna kroz koliko ploha taj polupravac prolazi. Kako su plohe reprezentirane pomoću trokuta testiranje siječe li polupravac trokut se provodi pomoću baricentričnih koordinata trokuta. Odabir polupravca je bitan jer želimo smanjiti broj operacija za izradu modela.

```
for (svaki trokut modela) {
	Površina = 0.5*(-By*Cx + Ay*(-Bx+Cx) + Ax*(By-Cy) +Bx*Cy)

	s = 1/(2*Površina)*(Ay*Cx - Ax*Cy + (Cy-Ay)*Točka.x + (Ax-Cx)* Točka.y)
	t = 1/(2*Površina)*(Ax*By - Ay*Bx + (Ay-By)*Točka.x + (Bx-Ax)* Točka.y)

	if (s>0 && t>0 && 1 - s - t>0) {
		spremi trokut za testiranje visine
	}
for (svaka ćelija u stupcu) {
	int brojač=0;
	for (svaki spremljeni trokut modela) {
		if( pravac probada trokut na većoj visini od trenutne ćelije)
			brojac++;
		}
	if (brojač % 2 == 1)
		Ćelija se nalazi unutar modela
	}
```

Programski odsječak 1. Testiranje da li se ćelija nalazi unutar modela

4.1 Ubrzanje algoritma

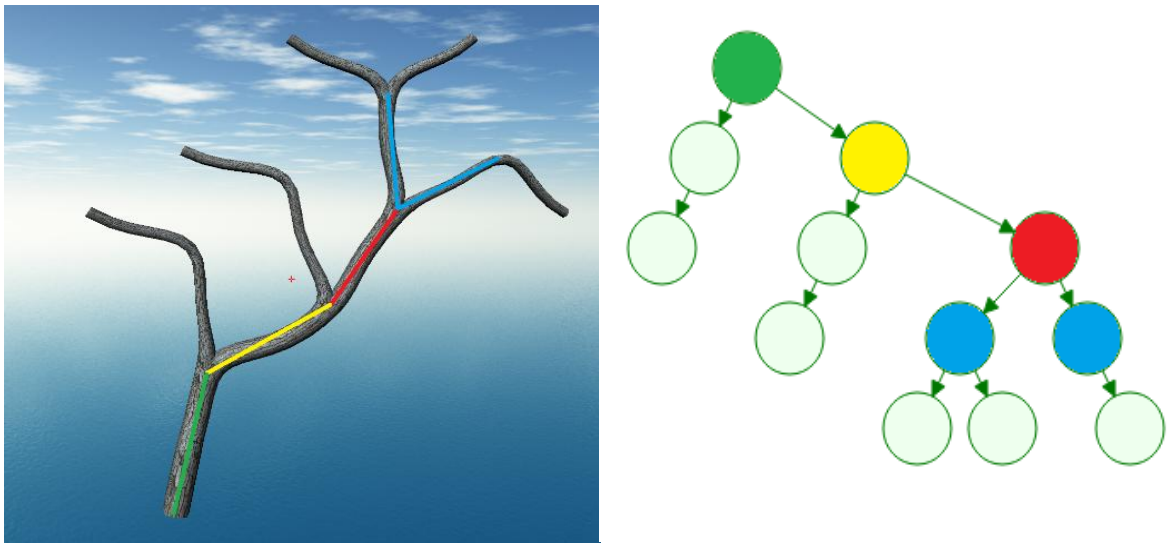
Postoje dvije optimizacije koje ubrzavaju izvođenje algoritma. Prva je grupiranje ćelija u stupce kako bi jednim pravcem mogli testirati koje plohe trebamo uzeti u obzir za zadani

stupac. Druga optimizacija je ta da zbog odabira pravca koji je paralelan s jednom osi možemo ignorirati jednu koordinatu trokuta i svesti problem prolaska pravca kroz trodimenzionalni trokut na problem pripadnosti točke dvodimenzionalnom trokutu. Na kraju se pobroji koliko ima ploha iznad određene ćelije te se, ako je taj broj neparan, ćelija nalazi unutar modela. Obje optimizacije su implementirane u programu.

Postoje algoritmi koji ubrzavaju izvođenje vokselizacije tako da se algoritam izvodi na grafičkoj kartici. Jedan takav algoritam je predstavljen u [9]. U tom radu se dodatno smanjuje korištenje prostora pomoću oktalnog stabla. Oktalno stablo se generira direktno iz žičanog modela bez potrebe za stvaranjem potpune rešetke u međukoracima.

5 Algoritmi za generiranje stabla

Rezultat algoritama za generiranje stabla je apstraktna struktura koja se dalje može koristiti prilikom izračunavanja utjecaja sila na stablo te se može lako prikazati pomoću žičanog modela i teksturirati. Ta struktura podataka se zove stablo gdje svaki čvor reprezentira jednu granu. Svaki čvor posjeduje pokazivače na njegovu djecu i na roditelja. Čvor također sadrži poziciju završetka grane u lokalnom prostoru stabla. Na temelju te pozicije i pozicije roditelja računa se vektor smjera grane koji se koristi za algoritam utjecaja sila.



Slika 2. Primjer jednostavnog stabla i njena apstraktna struktura

Na slici 2. je prikazana struktura jednostavnog stabla. Ukoliko je crvena grana trenutna koju promatramo onda su plavo obojane grane njegova djeca, žuta grana je roditelj, a zelenom bojom je označen djed. Boje na lijevoj slici odgovaraju bojama na desnoj.

Postoje različiti pristupi kojima se stablo može proceduralno generirati, a u tekstu će se obraditi tri načina koji reprezentiraju svoju skupinu. To su algoritam kolonizacije prostora, algoritam struje čestica i fraktalni algoritmi. Algoritam kolonizacije prostora i algoritam generiranja pomoću struje čestica su konceptualno slični jer se generiraju privlačne točke, tj. čestice koje međusobnom interakcijom generiraju nove grane. Razlika je u tome što algoritam kolonizacije prostora počinje iz debla i nove grane stvara do listova, dok algoritam struje čestica prvo stvara listove te međusobnim privlačenjem stvara nove grane prema deblu. Fraktalni

```
class Branch {
	static const float m_minThickness;

	mat4 m_modelMatrix;
	int m_modelMatrixUpdated;

	int m_renderDataIndex;

	glm::vec3 m_primDirection, m_currentDirection;

	float m_length, m_thickness;
	int m_depth;

	glm::vec3 m_endPosition;
	glm::vec3 m_force;

	Branch *m_parent;
	std::vector<Branch*> m_children;
	}
```

Programski odsječak 2. Korišteni podatci za jedan čvor unutar stabla

algoritam i generiranje pomoću L-sustava su slični jer prilikom izrada novih grana svaki put provjeravamo nalazi li se grana unutar željene krošnje. Ukoliko se grana ne nalazi unutar krošnje ona se samo odbaci i generiranje se nastavlja dalje. Prednosti fraktalnih algoritama su te što će sve grane biti točno unutar željene krošnje, dok će kod algoritma kolonizacije prostora i struje čestica krošnja podsjećati na željeni model, ali grane ne moraju nužno biti unutar modela.

U programskom odsječku 2. su popisani svi podatci koji su potrebni kako bi se mogla koristiti grana unutar programa. Unutar varijable `m_modelMatrix` je zapisana matrica transformacije tako da se vektor [0, 1, 0] transformira u vektor koji opisuje trenutnu granu. Ovaj podatak je potreban prilikom izračunavanja utjecaja sile na stablo. Varijabla `m_renderDataIndex` označava indeks grane, a potreban je kada se osvježavaju podaci o svakoj grani za prikaz na ekranu. Varijabla `m_endPosition` predstavlja poziciju vrha grane u globalnom koordinatnom sustavu. Varijable `m_parent` i `m_children` su pokazivači koji ostvaruju podatkovnu strukturu stabla.

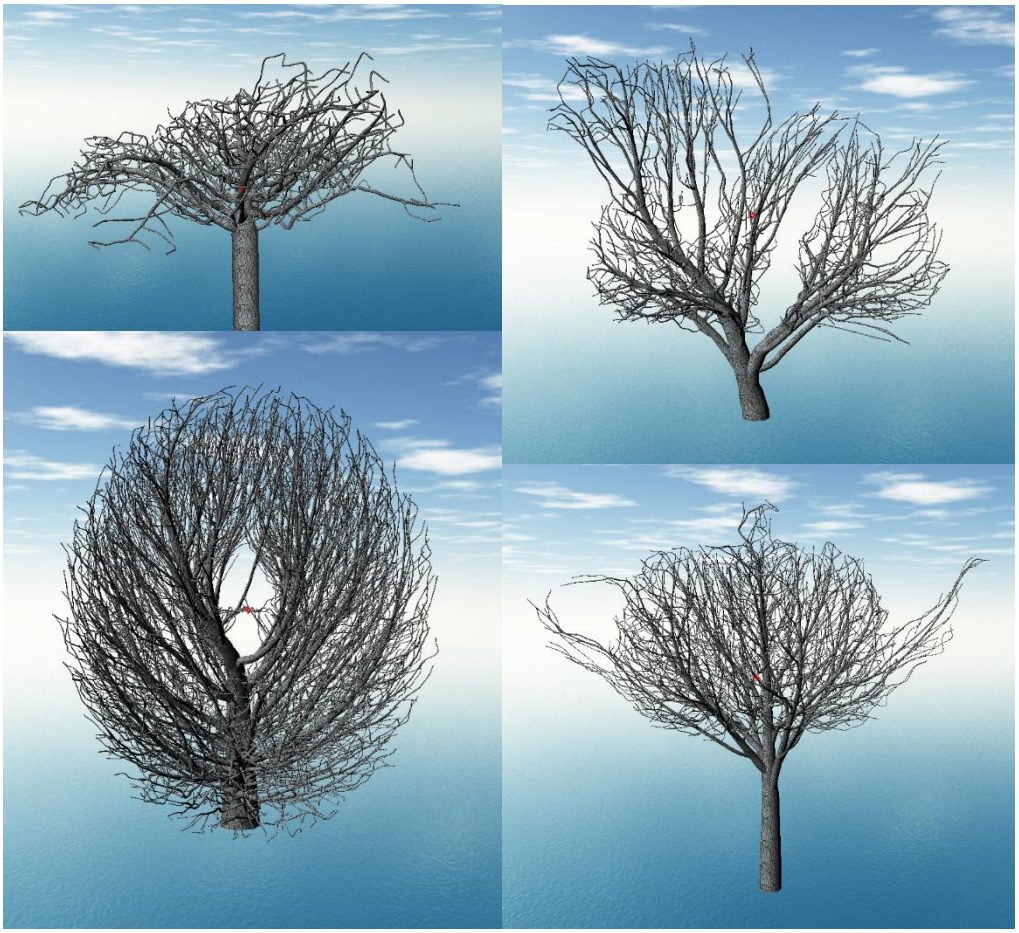
5.1 Algoritam kolonizacije prostora

Prva razrada i prijedlog ovog algoritma je napravljena u[1], ali algoritam u tom radu je bio ograničen na 2D prostor. Na temelju tog rada je algoritam proširen u 3D prostor unutar[5]. U ovome radu implementiran je navedeni algoritam uz dodatne mogućnosti koje omogućavaju korisniku odabir izgleda krošnje i naknadno uređivanje.

Algoritam se sastoji od dva dijela. Prvo se prostor popunjava pomoću privlačnih točaka, a onda se pomoću odabranih točaka iterativnim postupkom izvršava generiranje grana.

Slika 3. Privlačne točke u obliku modela čajanke i krajnji rezultat

Rezultat prvog dijela algoritma je popis privlačnih točaka. Korisniku je omogućeno da generatoru proslijedi vokselizirani model pomoću kojeg se dodaju privlačne točke u željeni prostor. Broj točaka kojima će se popuniti model je jedan od parametara koje korisnik može



Slika 4. Stabla nastala od modela žabe, zmaja, krafne s rupom i čajanke

odabrati, a razuman broj privlačnih točaka se nalazi unutar [200, 50000]. Taj broj ovisi i o drugim parametrima i korisnik mora voditi računa o kompleksnosti algoritma kako se generator ne bi previše usporio. Očekivani broj točaka kojima se popuni model je u razini 1000 točaka. Kako model reprezentira samo krošnju, potrebno je na neki način pripremiti privlačne točke da se generira i deblo. To je ostvareno tako da se privlačne točke odgovorne za nastanak krošnje translatiraju na željenu visinu te da se ispod krošnje u jednakim razmacima dodaju privlačne točke koje reprezentiraju deblo.

Kako bi se algoritam uspješno pokrenuo potrebno je ručno izraditi početnu granu, to jest korijen cijeloga stabla. Korijen se odabire na mjestu na kojem su dodane privlačne točke za deblo, a uvijek je usmjeren prema gore. Algoritam se izvršava u iteracijama sve dok postoje aktivne privlačne točke. Također, dodan je parametar za maksimalan broj grana koji služi kao gornja granica generatoru da ne stvori prekompliciran model i zaguši program.

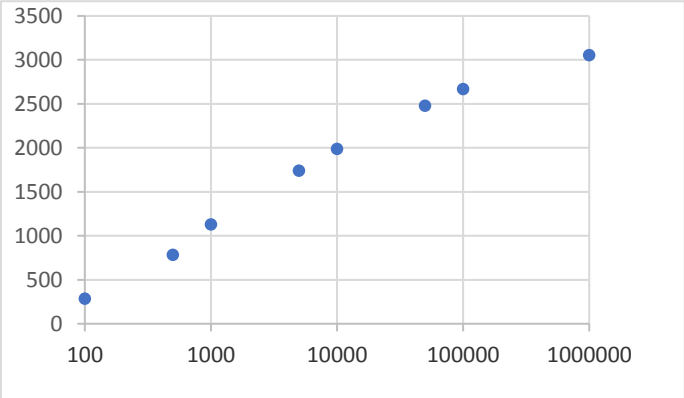
U jednoj se iteraciji za svaku aktivnu privlačnu točku traže najbliže grane te se, ukoliko se točka nalazi dovoljno blizu neke grane, ona se zapisuje u popis točaka za tu granu. Taj popis služi kako bi se izračunao vektor rasta nove grane. Ukoliko se točka nalazi preblizu nekoj grani, ona se izbacuje iz popisa aktivnih točaka. Na kraju iteracije se za svaku granu koja u popisu sadrži neku privlačnu točku napravi nova grana u smjeru aritmetičke sredine svih privlačnih točaka i to po formuli 1. Gdje je „$\vec{n}$" vektor koji reprezentira smjer rasta nove grane, „s“ je pozicija trenutne privlačne točke, a „v“ je pozicija vrha trenutne grane.

$$\vec{n} = \sum_{s}^{S} \frac{s - v}{||s - v||} \tag{1}$$

5.1.1 Analiza parametara algoritma

Duljina nove grane se računa prema formuli 2., gdje je maxDuljina najveća željena duljina grane u stablu, a minDuljina najmanja. Razina predstavlja udaljenost grane od debla na način da deblo ima razinu 1, njegovo dijete razinu 2 i tako dalje. Ovakvim odabirom duljine grane je omogućeno da grane koje su daleko od debla imaju manju duljinu, pa se zbog toga i više grana stvori pri vrhu krošnje. Duljina grane izravno utječe na kompleksnost stabla jer je potrebno generirati više grana kako bi prošle jednaku udaljenost unutar željenog modela u odnosu na dulje grane.

$$\text{duljinaGrane} = \text{minDuljina} + \frac{\text{maxDuljina} - \text{minDuljina}}{1 + e^{razina-10}} \tag{2}$$



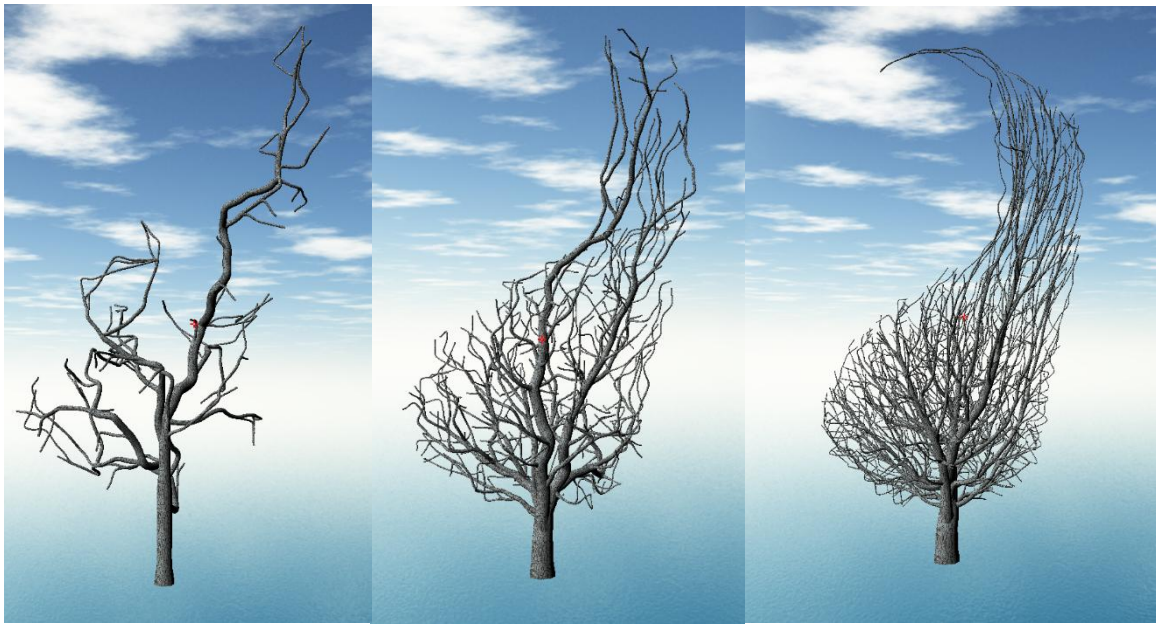
Tablica 1. Utjecaj broja privlačnih točaka na krajnji broj grana

Može se dogoditi da pri kraju generiranja stabla postoje točke koje su predaleko od svih grana, pa se zato te točke ne mogu maknuti iz popisa aktivnih točaka. Zato je potrebno dodati umjetno zaustavljanje generatora prilikom nesmanjivanja broja aktivnih točaka kako algoritam ne bi ušao u beskonačnu petlju.

Minimalna udaljenost između grane i privlačne točke služi da se točke koje su dovoljno blizu nekoj grani izbace iz daljnjeg računanja jer se stablo dovoljno proširilo u tom smjeru. Iz tablice 1. se može zaključiti da postoji logaritamska veza između broja privlačnih grana i krajnjeg broja grana.

5.1.2 Daljnje poboljšanje algoritma

Složenost algoritma ovisi među ostalim o načinu na koji tražimo najbliže točke za svaku granu. Implementirano je obično pretraživanje tako da se za svaku granu prođe kroz popis aktivnih privlačnih točaka i provjeri kolika je udaljenost grane od te točke. Složenost tog pristupa je $O(n * p)$, gdje je n završni broj grana, a p početni broj privlačnih točaka. Pretraživanje se može smanjiti na složenost od $O(n * \log(p))$ ili $O(p * \log(n))$, ovisno o trenutnim veličinama parametara, a postigla bi se pomoću korištenja strukture podataka oktalnog stabla[4]. Pomoću te strukture podataka moguće je kod traženja točaka brzo odbaciti



Slika 5. Izgled stabla za 100, 1000 i 100,000 privlačnih točaka

sve točke koje nisu dovoljno blizu trenutnoj grani, pa se ispitivanje provodi samo nad onim točkama koje su od interesa.

Kako se tijekom izvođenja programa rijetko generira stablo i kako je brzina izvođenja rješenja bez navedene optimizacije bila zadovoljavajuća, nije bilo potrebe za implementaciju poboljšanja.

5.2 Fraktalni algoritmi

Najranije korišteni načini generiranja stabala su pomoću fraktalnih algoritama. Razlog tome je što je to najintuitivniji pristup, a matematička podloga je već bila razrađena.

Prednost korištenja fraktalnih algoritama je lako postizanje pravilnih i vizualno ugodnih oblika kod grananja. Stablo nastalo korištenjem fraktalnih algoritama pokazuje svojstvo samosličnosti. Samosličnost je takvo svojstvo da, ukoliko se neka grana i sva njegova djeca izoliraju od ostatka stabla, oni izgledaju slično početnom stablu.

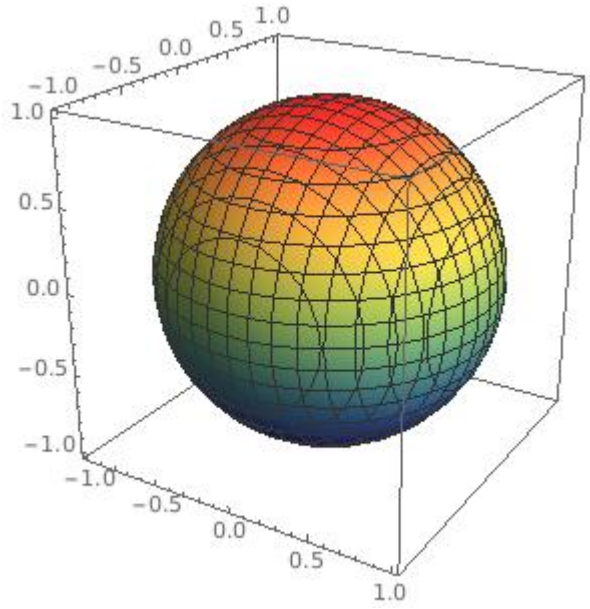
Problem koji nastaje prilikom korištenja fraktalnih algoritama je teško kontroliranje konačnog oblika krošnje stabla. Baš zbog toga je unutar ovog rada razrađen i implementiran fraktalni algoritam koji kao jedan od parametara prilikom generiranja uzima konačni željeni oblik te na temelju njega stvara stablo sličnog oblika.

Generiranje stabla počinje odabirom točke unutar modela. Do te točke se izgradi deblo željene visine, te se pokrene algoritam. Algoritam se izvršava u unaprijed zadanom broju iteracija. One grane koje nemaju djece se smatraju kao aktivne grane. Unutar iteracije se odabere jedna aktivna grana te se odredi koliko djece ta grana treba imati. Potom se za svaku novonastalu granu odabere vektor rasta te grane na način da ne izlazi izvan modela.

Odabir broja djece za svaku granu se računa prema formuli 3. koja predstavlja eksponencijalnu distribuciju. Parametar λ se može ugađati te je unutar programa odabrana vrijednost λ=3.5.

$$p(x|\lambda) = \lambda e^{-\lambda x} \tag{3}$$

Odabir smjera novonastalih grana se provodi na način da se na smjer u kojem se kreće roditelj doda slučajan pomak u stranu. Zbog toga ne postoje nagla skretanja grana i neprirodni oblici. Ukoliko vrh roditeljeve grane postavimo u centar koordinatnog sustava, a vektor smjera roditeljeve grane bude paralelan s y osi onda je na slici 6. pomoću boja prikazana gustoća razdiobe smjera novonastale grane. Crvena boja označava područje u kojem je najvjerojatnije da se stvori novi vektor, a plavom bojom najmanje vjerojatno područje.

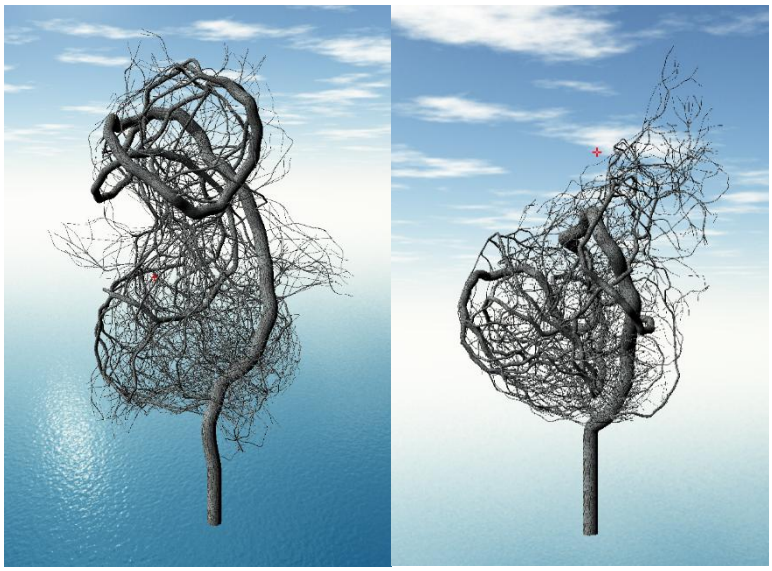


Slika 6. Prikaz gustoće razdiobe novonastalih vektora grana

5.2.1 Varijacije algoritma

Fraktalni algoritam je moguće ostvariti na više načina, pomoću reda i polja. Također, moguće je korištenje rekurzije prilikom generiranja stabla, ali ono je ekvivalentno korištenju strukture podataka stog. Unutar programa implementirana su sve 3 varijante fraktalnog algoritma.

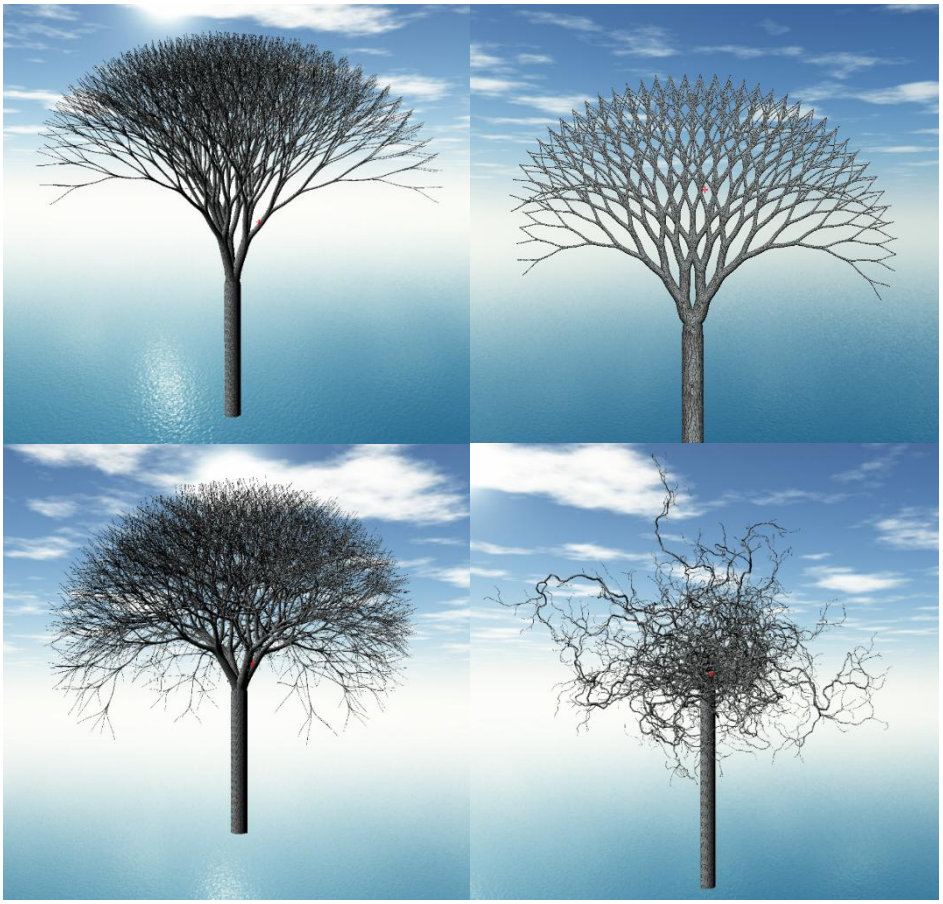
Ono što razlikuje svaki pristup je redoslijed kojim se odabiru grane za daljnje grananje. Kod stoga se odabire ona grana koja je zadnja napravljena, kod reda se odabire ona grana koja je napravljena najranije, a kod polja se nasumično odabire redoslijed grana.



Slika 7. Stabla nastala od modela medvjedića i kapljice

Prilikom korištenja fraktalnog algoritma želimo ispitati nalazi li se neka grana unutar željenog oblika. Kako bi omogućili generatoru da nastavi širenje stabla unutar željnog modela, potrebno je korijen stabla namjestiti točno na onu poziciju gdje počinje model, a to se napravi transformacijama modela.

Na početku izvršavanja algoritma postoji mali broj grana i zbog toga je velika vjerojatnost da se algoritam zaustavi prerano. Da se izbjegne problem prebrzog zaustavljanja, pokrećemo algoritam sve dok ne dobijemo željeni broj grana. Unutar programa odabrano je 100 iteracija pokretanja sve dok ne nastane minimalno 30 novih grana, a nakon toga se generator više ne pokreće.



Slika 8. Primjeri generirani fraktalnim algoritmima

Slika 8. prikazuje rezultate izvršavanja fraktalnog algoritma bez popunjavanja nekog modela. Gore lijevo je prikazano stablo u kojemu svaki sloj ima tri puta više grana nego prošli. Gore desno je prikazano stablo u kojemu svaki sloj ima dva puta više grana nego prošli te se stablo grana u dvije dimenzije. Dolje lijevo je prikazano stablo u kojemu svaki sloj ima 4 puta više novih grana. Dolje desno je prikazano stablo kojemu se tijekom izvođenja slučajno odabire nova grana za grananje, a odabir količine djece se izvršava pomoću eksponencijalne distribucije između jedne i tri grane.

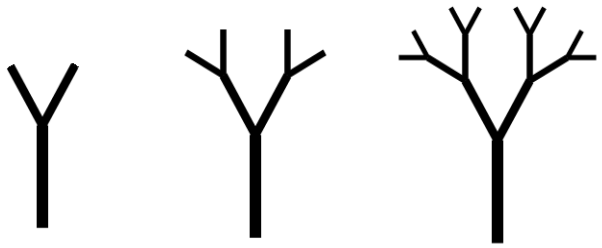
5.2.2 Usporedba s Lindenmayer sustavima

L-sustav je vrsta formalne gramatike, a rezultati podsjećaju na fraktalne algoritme. Ispravnim parametriziranjem L-sustava moguće je dobiti mnogo oblika koji izgledaju identično različitim biljkama[7] . Kao i svaka formalna gramatika, sastoji se od liste znakova V, početnog simbola ω i pravila produkcija P. Lista znakova može sadržavati završne znakove za koje dalje ne postoje pravila produkcije. Početni simbol kod implementiranog fraktalnog algoritma je početna grana. Pravila produkcije označavaju korak kada se iz jedne aktivne grane stvori jedna ili više novih grana. Završni znakovi predstavljaju grane koje su obrađene i više se iz njih ne mogu stvoriti nove grane.

Rezultat L-sustava je niz znakova, a postupak generiranja se provodi tako da u iteracijama odjednom sve aktivne znakove zamijenimo pomoću produkcijskih pravila novim kombinacijama znakova. Bitan korak kod dizajniranja takvog sustava je interpretacija niza znakova.

V = { A, B, [,]}
ω= A
$P = \{(B \rightarrow BB), (A \rightarrow B[A]A)\}$

Početni znak: A
1. Iteracija: B[A]A
2. Iteracija: BB[B[A]A]B[A]A
3. Iteracija: BBBB[BB[B[A]A]B[A]A]BB[B[A]A]B[A]A



Slika 9. Primjer L-sustava i grafički prikaz prvih nekoliko iteracija

Slika 9. prikazuje grafički interpretaciju navedenog primjera L-sustava. Niz znakova moguće je interpretirati tako da znakovi A i B označavaju crtanje grane određene duljine. Svi

znakovi koji se nalaze unutar para zagrada [i] se smatraju djecom one grane koja se nalazi odmah lijevo od tog para zagrada. Prvu iteraciju algoritma zato interpretiramo kao:

- B – stvori novu granu
- [A] - stvori novu granu i označi roditelja kao trenutnu granu
- A - stvori novo dijete trenutne grane

Ono što razlikuje L-sustav od implementiranih fraktalnih algoritama je to što se grane u implementiranim algoritmima obrađuju jedna po jedna i redoslijed obrade grana se može slučajno odabrati.

5.3 Generiranje stabla pomoću struje čestica

Implementirani algoritam je verzija algoritma predstavljenog u [6]. U tom radu se opisuje kako se na temelju slike i pomoći korisnika mogu izvući podatci o usmjerenosti grana stabla unutar trodimenzionalnog prostora. Na temelju podataka o usmjerenosti se može izgraditi trodimenzionalno polje vektora koje opisuje kretanje grana. Potom se polje vektora koristi prilikom simulacije struje čestica gdje vektor smjera u pojedinoj točci utječe na gibanje čestice. Na kraju simulacije se na temelju prijeđenog puta svake čestice izgradi stablo. Razlika između opisanog algoritma i algoritma implementiranog u ovom radu je to što se u ovom radu ne koristi polje vektora koje opisuje rast grane.

Slika 10. Modeli stabala u obliku ptice i zmaja

Slično kao i kod algoritma kolonizacije prostora, ovaj algoritam se sastoji od dva dijela. Prvo se prostor željenog oblika popuni česticama te se potom na temelju početnih pozicija i masa čestica iterativno izračunavaju nove pozicije čestica ovisno o vremenu.

Svaka čestica je reprezentirana strukturom koja u sebi sadrži podatke o poziciji, brzini, sili i masi pojedine čestice. Dodatno se nalazi i pokazivač na tip podataka „Grana“ koji u sebi sadrži trenutnu apstraktnu strukturu stabla vezanu uz tu česticu. Tip podataka „vec3“ reprezentira trodimenzionalni vektor.

```
struct Čestica{
	vec3 pozicija;
	vec3 brzina;
	vec3 sila;
	float masa;
	Grana *grana;
	};
```

Programski odsječak 3. Podatkovna struktura svake čestice

Odabir početnih pozicija čestica se može parametrizirati funkcijama gustoće. Uloga funkcije gustoće je da program zna na koji način određen model treba biti popunjen s česticama. Primjeri takvih funkcija su one koje preferiraju stvaranje čestice koje se nalaze uz rub modela ili one koje su na većoj ili manjoj visini. Korisnik može odabrati početni broj čestica, a najbolji rezultati se postižu odabirom broja 500. Na temelju početnih pozicija čestica se izrade vrhovi grana te se daljnjim pomicanjem i međusobnom interakcijom grade nove grane. Grane koje su nastale na početku izvođenja algoritma su listovi u strukturi podataka, a grane koje nastaju na kraju izvođenja algoritma su bliže deblu.

```
Unutar svake iteracije {
1. Izračunaj silu na svaku česticu
2. Osvježi poziciju svake čestice
3. Napravi nove grane
4. Spoji čestice i pripadajuće grane
}
```

Programski odsječak 4. pseudo kod jedne iteracije algoritma struje čestica

Nakon odabira početnih pozicija čestica slijedi iteriranje algoritma za fizičku simulaciju kretanja čestica. Svaka iteracija se sastoji od računanja sile na svaku česticu te potom pomicanja čestice za neki vremenski period. Nakon pomicanja svake čestice testiraju se

međusobne udaljenosti i one grupe čestica koje su dovoljno blizu se spajaju u novu česticu veće mase.

Sila na svaku pojedinu česticu se računa pomoću formule 4. Svaka čestica utječe na sve ostale čestice i to proporcionalno svojom masom (m_i, m_j) i obrnuto proporcionalno njihovom udaljenošću. Koeficijent k utječe na ukupnu jačinu kojom se čestice privlače, a x_i i x_j su pozicije trenutno promatranih čestica.

$$\vec{F_i} = \sum_{j}^{N} k * m_i * m_j * \frac{x_j - x_i}{\|x_j - x_i\|^2} \tag{4}$$

Formule 5. opisuju promjenu pozicije i brzine svake čestice u vremenu. Prilikom pokretanja algoritma potrebno je odabrati promjenu vremena tako da simulacija bude dovoljno precizna, ali ne i prespora. Unutar programa odabrano je da je vremenska razlika prilikom svake iteracije algoritma uvijek jednaka 10ms.

$$v_i = v_{i-1} + \Delta t * \frac{\vec{F_i}}{m_i}$$
$$\mathrm{x_i} = x_{i-1} + \Delta t * v_i \tag{5}$$

Slika 11. Primjeri generiranih stabala s 400, 1000 i 2000 grana

Grupu čestica predstavljaju sve čestice koje se nalaze unutar neke kugle radijusa željene konstante. Radijus je unutar programa namješten na 10cm. Na temelju tog uvjeta i pozicija čestica moguće je pronaći različita točna rješenja grupiranja čestica, tako da se odabire prvo koje odgovara. To nam odgovara jer se na taj način unosi slučajnost prilikom grananja stabla. Za svaku novonastalu grupu čestica izradi se jedna nova grana kojoj završetak odgovara trenutnoj poziciji nove grupe. Trenutna pozicija grupe se računa kao aritmetička sredina svih čestica unutar te grupe po formuli 6. Slično se po formuli 7. računa i nova brzina grupe.

$$x = \frac{\sum m_i * x_i}{\sum m_i} \tag{6}$$

$$\vec{v} = \frac{\sum m_i * \overrightarrow{v_i}}{\sum m_i} \tag{7}$$

Grane koje su nastale u prijašnjim iteracijama, a pripadaju česticama koje se spajaju, upišu se kao djeca novonastale grane.

Kako bi se postigao standardni oblik stabla s krošnjom, u svakom trenutku na čestice djeluje i gravitacijska sila. Rezultat utjecaja gravitacije je deblo usmjereno prema tlu.

Slika 12. Lijevo je gravitacija isključena, a desno pojačana

Moguće je napraviti varijaciju na navedeni algoritam tako da na čestice utječu i sve grane koje su do nekog trenutka napravljene. Kako je zbog toga masa koja privlači čestice više distribuirana u prostoru, izbjegava se učinak da se sve čestice nakon nekog vremena počnu kretati prema centru mase svih čestica.

Negativna strana korištenja ove metode je teško implementiranje novih pravila na način da novonastalo drvo izgleda drugačije u odnosu na početno stablo. Razlog tomu su kompleksne interakcije između čestica i statični parametri za svaku razinu grana. Kako algoritam stablo gradi od listova prema deblu, nije u ni jednom trenutku poznato na kojoj dubini će određena grana na kraju biti. Zbog nepoznavanja dubine određene grane unutar stabla, teško je implementirati algoritme koji bi mogli smanjivati kompleksnost stabla prilikom izvođenja algoritma. Također nije moguće postići da korisnik odabere koliko maksimalno novih grana se može stvoriti iz jedne grane.

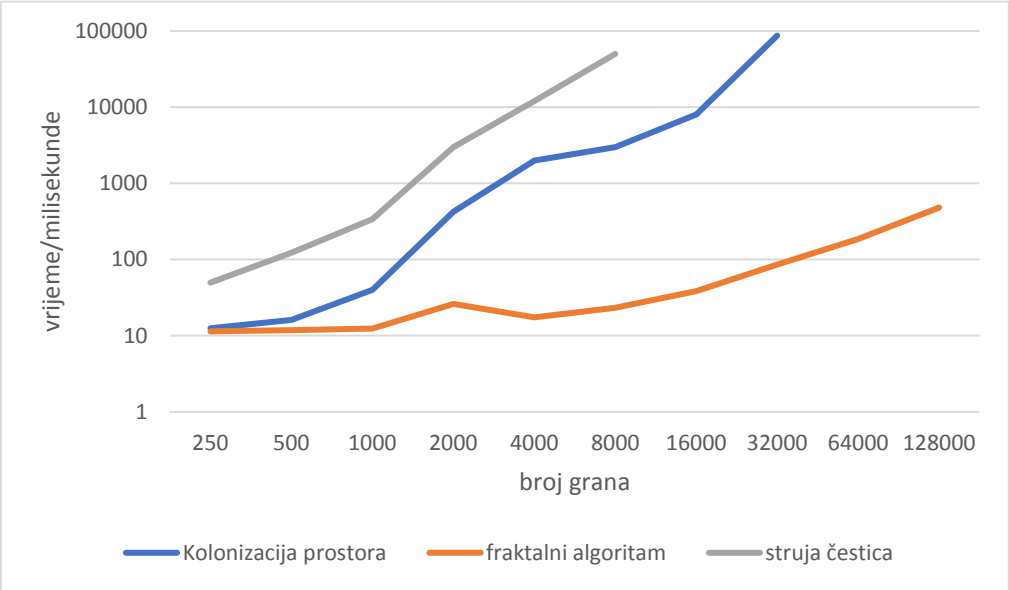
Algoritam posjeduje puno parametara koji se moraju podesiti kako bi generirano stablo izgledalo realno. U tablica 2. su navedeni glavni parametri i njihove vrijednosti koje su korištene prilikom generiranja stabala.

Naziv parametra	Vrijednost	Opis
numberOfParticles	500	Broj čestica s kojima algoritam započinje
maxIteration	500	Najveći broj iteracija prije nego se algoritam zaustavi služi za izbjegavanje beskonačne petlje
combineDistance	0.1m	Udaljenost unutar koje dvije čestice moraju biti da se kombiniraju u zajedničku čestice
attractionCoef	0.25	Parametar s kojim se množi utjecaj sile na svaku česticu
timeCoef	0.01s	Vremenski pomak između svake iteracije simulacije
gravityCoef	$10m/s^2$	Utjecaj gravitacije na svaku česticu
minBranchLength	0.1m	Kako se čestica pomiče ne želimo svaki korak pretvoriti u novu granu nego samo kada čestica prođe dovoljnu udaljenost

tablica 2. Popis parametara algoritma struje čestica

5.4 Usporedba proceduralnih algoritama

Razvijeni program se testirao na računalu srednje jačine. Računalo je opremljeno četverojezgrenim procesorom radnog takta od 3.2GHz. Program se izvršava na jednoj jezgri procesora. Glavna memorija računala je veličine 20GB. Grafički procesor radi na taktu od 1GHz, posjeduje ukupno 2GB memorije i 320 procesora za izvršavanje programa sjenčara.



tablica 3. Vrijeme potrebno algoritmima za izradu određenog broja grana

Iz tablice 3. se vidi da svi algoritmi na testiranom rasponu imaju linearnu vremensku ovisnost broja izrađenih grana i potrebnog vremena. Najbrži algoritam je fraktalni jer je potrebno izvršiti minimalan broj operacija kako bi se dodala nova grana. Prilikom izvršavanja algoritma kolonizacije prostora, svaka novonastala grana mora dodatno ispitati udaljenost između nje i svih privlačnih točaka. Algoritam struje čestica je najsporiji jer prilikom stvaranja nove grane mora ispitati udaljenost između trenutne čestice i svih ostalih čestica te mora još proći i dovoljan broj iteracija kako bi se čestica odmakla od vrha prijašnje grane.

Rezultate algoritma možemo usporediti po subjektivnom osjećaju ljepote. Po tome algoritam kolonizacije izrađuje najprirodnija i najljepša stabla. Fraktalni algoritam je najbrže i najlakše implementirati, te se uz minimalne promjene unutar programskog koda mogu dobiti veoma različiti i zanimljivi rezultati.

6 Prikazivanje stabla

Prilikom izrade stabla potrebno je odrediti debljinu grana. Prema pravilu koje je da' Vinci otkrio[3], radijus grana kod različitih vrsta biljaka se ponaša uvijek po istoj formuli uz mala odstupanja. Svakom listu u stablu se pridijeli osnovna debljina, te se rekurzivno za svakog roditelja računa njegova debljina po formuli 8., gdje je r radijus određene grane, a x potencija unutar $[1.8, 2.3]$. Odabir potencije ovisi o vrsti stabla, a unutar programa je zbog praktičnosti odabrana potencija 2.

$$r_{roditelj}^{x} = \sum_{svako\ dijete} r_{dijete}^{x} \tag{8}$$

Zadatak algoritama za prikazivanje stabla je da generiranu apstraktnu strukturu stabla pretvori u žičani model i prikaže na ekran. Generiranje modela moguće je ostvariti na centralnom procesoru ili na grafičkom procesoru. Unutar programa ostvareno je generiranje modela na grafičkom procesoru. Prednosti tog pristupa su da se model brže generira zbog paralelizacije i nije potrebno na grafičku karticu prenositi toliko podataka koliko bi bilo potrebno da se prenosi cijeli žičani model. Na primjer, ukoliko generirano drvo sadrži 50000 grana, potrebno je između prikazivanja svake sličice na grafičku karticu prenijeti 0.5MB podataka o koordinatama i polumjerima grana. Na temelju tih podataka grafička kartica generira 17MB podataka o koordinatama žičanog modela. Izračunavanje žičanog modela na grafičkoj kartici je asinkrono u odnosu na centralni procesor, pa tako glavni program može nastaviti izvođenje i pripremati podatke za novi prikaz. Mana ovakvog načina prikazivanja je ta da se žičani model ne može naknadno lako obraditi i uljepšati. Primjer obrade koji se može napraviti je spajanje žičanih modela grana koji imaju istog roditelja čime bi se smanjio ukupan broj podataka stabla. Zbog navedenih razloga grafički procesor je bolje koristiti u slučajevima kada nam je potreban prikaz drveta u realnom vremenu za simulacije, a centralni procesor je

bolje koristiti u slučajevima kada želimo jednom izraditi model drveta te ga koristiti u drugim programima ili naknadno uljepšati.

```
currentPoint= bezierPoints[0] * (1-T)*(1-T)*(1-T)+
              bezierPoints[1] * 3*T*(1-T)*(1-T)  +
              bezierPoints[2] * 3*T*T*(1-T) +
              bezierPoints[3] * T*T*T;

tangent=      bezierPoints[0] * (-3*T*T + 6 *T - 3) +
              bezierPoints[1] * ( 9*T*T - 12*T + 3) +
              bezierPoints[2] * (-9*T*T + 6 *T) +
              bezierPoints[3] * ( 3*T*T);

normal=       bezierPoints[0] * (-6*T + 6) +
              bezierPoints[1] * (18*T - 12) +
              bezierPoints[2] * (-18*T + 6) +
              bezierPoints[3] * (6*T);
```

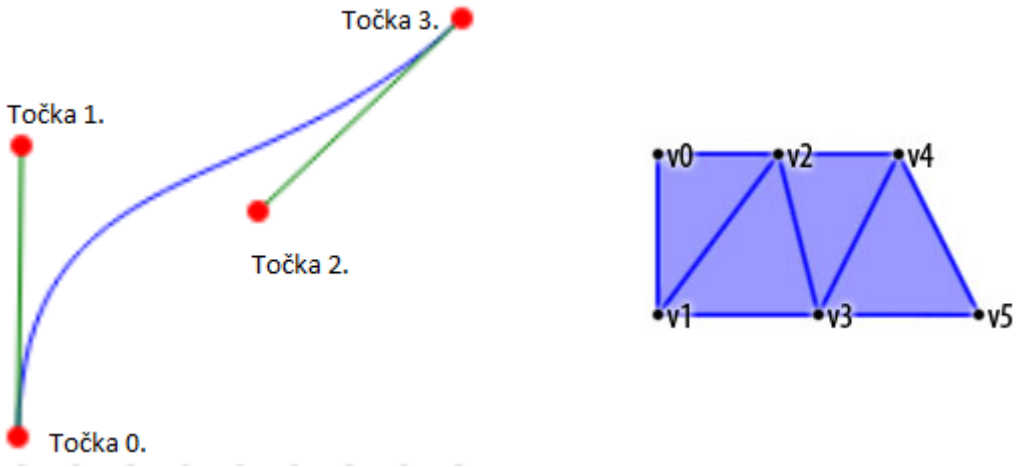
Programski odsječak 5. Način računanja pozicije, normale i tangente na Bezierovu krivulju

Prilikom izrade žičanog modela za svaku granu je potrebno znati poziciju vrha trenutne grane, poziciju roditelja, poziciju djeda te radijus početka grane i radijus završetka grane. To je ukupno 11 podataka tipa float. Izrada žičanog modela svake grane odvija se unutar geometrijskog sjenačara na grafičkom procesoru. Sjenčar ima ograničen broj točaka po grani koje može poslati na izlaz, pa je potrebno ravnomjerno rasporediti broj točaka žičanog modela. Odabrano je da svaku granu reprezentira 8 slojeva po 8 točaka na kružnici. Kako je najbolji zapis žičanog modela u obliku trake trokuta, krajnji broj točaka iznosi 128, pa je tako ukupan broj podataka nakon geometrijskog sjenčara po grani 384 podatka tipa float.

Prvi korak u algoritmu generiranja žičanog modela je izračunavanje Bézierove krivulje koja prolazi kroz 4 točke. Točke su odabrane tako da se prijelazi između susjednih grana ne vide.

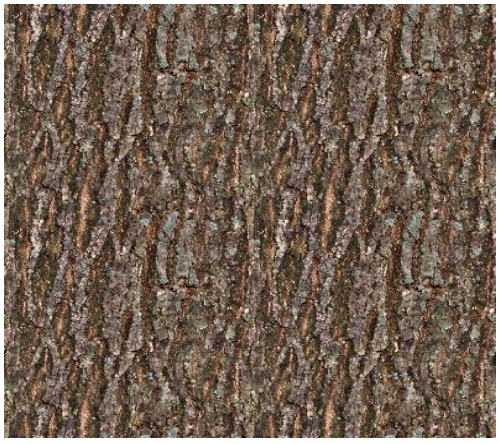
- Točka 0. je vrh grane roditelja
- Točka 1. je točka nastala translacijom točke roditelja u smjeru rasta grane roditelja
- Točka 2. je točka nastala negativnom translacijom točke trenutne grane za vektor rasta trenutne grane
- Točka 3. je vrh trenutne grane

Nakon što se definira Bézierova krivulja potrebno je naći pozicije točaka u jednakim razmacima koje predstavljaju slojeve svake grane. U svakom tom sloju će biti niz točaka poredanih u kružnicu. Kako bi se kružnica mogla orijentirati u pravom smjeru potrebno je



Slika 13. Bézierova krivulja kroz 4 točke i popunjavanje pomoću trake trokuta

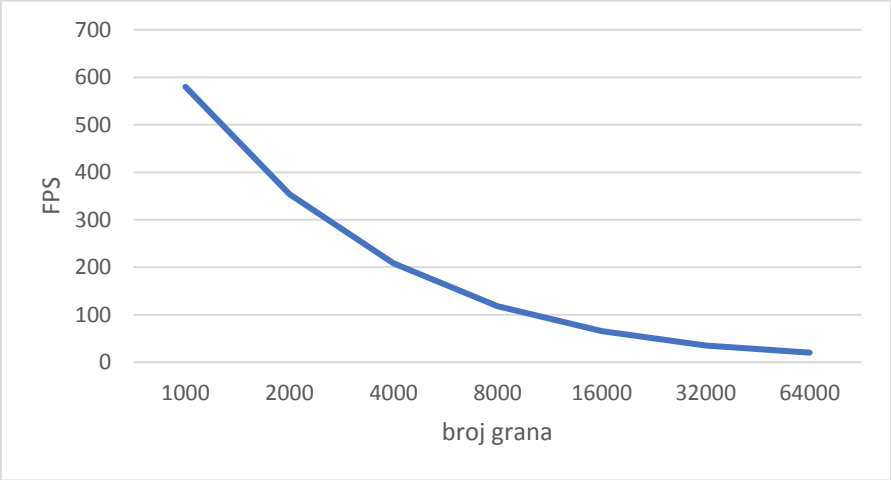
izračunati normalu i tangentu na Bézierovu krivulju u odabranim točkama. Tangenta na krivulju je prva derivacija formule za krivulju dok je normala druga derivacija formule. Zadnji korak je da se izračunate točke poredaju u niz kako bi tvorile strukturu trake trokuta. Svaka traka predstavlja jedan sloj u grani.



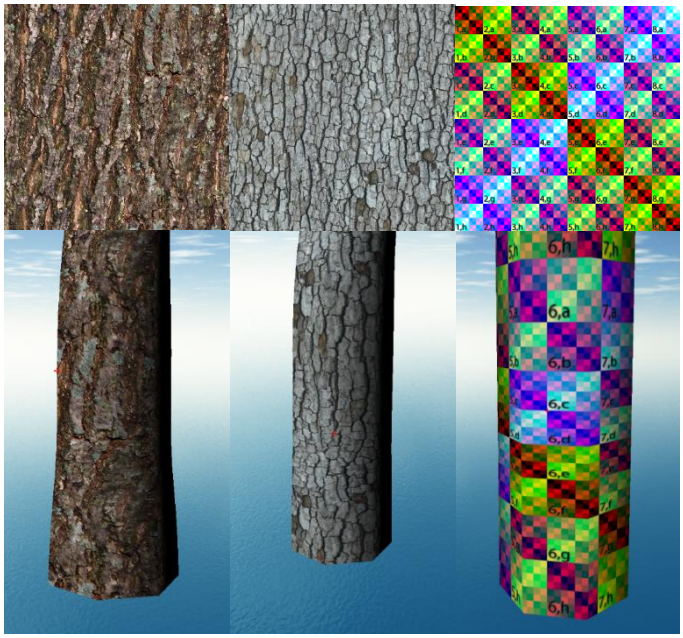
Slika 14. Primjer toroidne teksture

Na kraju je napravljeno preslikavanje teksture na granu i osvjetljavanje. Kako bi prijelazi između grana bili što manje vidljivi potrebno je koristiti toroidne teksture. Kod njih se ne vidi prijelaz između kraja grane roditelja i grane djeteta. Osvjetljavanje je ostvareno jednostavnim

modelom s ambijentalnom i difuznom komponentom. Svakoj grani se može podesiti razina detalja kojom je prikazana. Podešava se po broju horizontalnih i vertikalnih slojeva oplošja grane. Maksimalan broj točaka koje je testirana grafička kartica mogla dati po grani je 250.



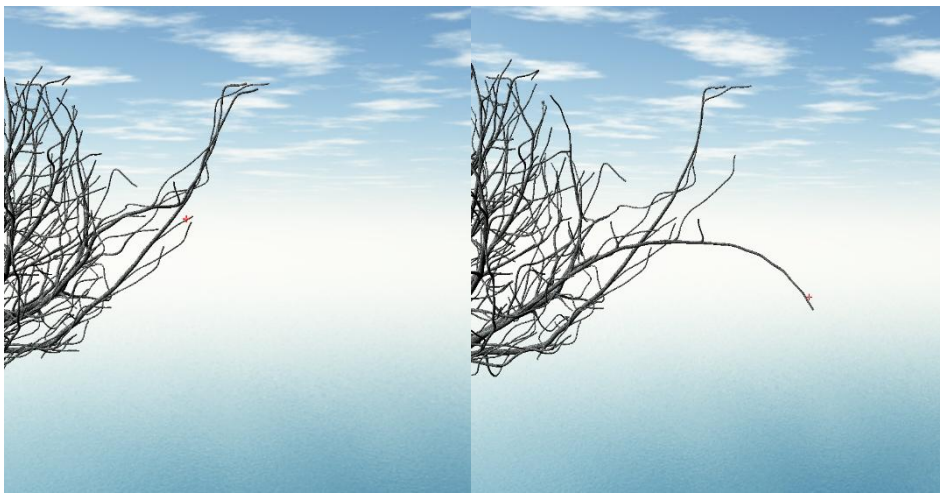
tablica 4. Ovisnost broja sličica u sekundi o broju prikazanih grana



Slika 15. Teksture korištene za grane na stablu

7 Utjecaj sile na grane

Kada na stvarnom stablu pokušamo potegnuti granu očekujemo da će se savinuti u našem smjeru i to na takav način da se tanji dio grane više savija. Ta sila se treba propagirati skroz do korijena stabla. Ona grana koja nije između točke hvatišta i korijena se ne savija nego samo prati pomicanje svog roditelja. Cilj izračunavanja utjecaja sile je ostvariti što sličniji učinak propagacije sile. Primjena algoritma može biti prilikom modeliranja stabla ili kada postoji potreba da je stablo u interakciji s okolinom. U programu su ostvarena dva načina na koji sila utječe na stablo i korisnik može odabirati koji mu više odgovara u nekom trenutku.



Slika 16. Promjena izgleda stabla prilikom utjecaja sile

Prvi način računanja utjecaja sile je pogodan za modeliranje jer mijenja smjer i poziciju zahvaćene grane i svih grana do korijena, dok one grane koje nisu pod utjecajem sile ne mijenjaju svoj globalni smjer. Drugi način je onaj realan jer u obzir uzima i to da se osim pozicije mijenja i smjer svake grane u drvetu prilikom utjecaja sile. Drugi način je prezentiran u [8], gdje su opisane sve sile koje utječu na stablo prilikom savijanja grane. Propagacija sila se u tom radu koristi za simulaciju utjecaja vjetra na stablo.

Prije svake obrade sličice se poziva funkcija koja osvježava pozicije svake grane na temelju aktivnih sila i trenutnih pozicija grana. Korisnik pomoću pokazivača generira vektor sile kojim djeluje na neku granu. Taj vektor se kod rekurzivnog osvježavanja dijeli na dio koji rotira trenutnu granu oko roditelja i vektor koji se prosljeđuje grani roditelja. Taj proslijeđeni vektor

se zbraja s ostalim prisutnim silama te se dalje rastavlja. Na taj način je omogućena propagacija sile do samog debla stabla. Bitno je napomenuti da svaka grana ima elastičnu silu koja djeluje na nju ukoliko je grana odmaknuta od početne pozicije u odnosu na roditelja. Ta sila je proporcionalna samom odmaku i debljini grane. Ukoliko je grana deblja, veća sila djeluje na nju da se vrati u početni položaj.

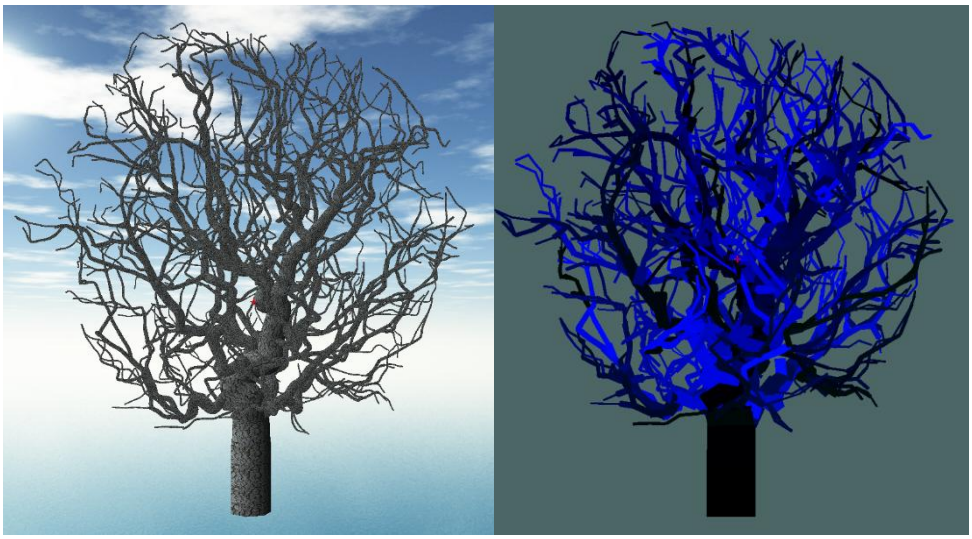
8 Hvatanje i pomicanje grane pomoću kursora

Korisniku je unutar programa omogućeno da na interaktivan način pomiče grane. Na sredini ekrana se prikazuje kursor u obliku križa kako bi korisnik znao koju granu će označiti pritiskom na gumb miša. Moguće je kursorom pritisnuti na granu, a pomicanjem miša se stvara vektor sile koji utječe na označenu granu. Postoje dva načina na koji se može praktično izvesti odabir grane.

Prvi način je pomoću izračunavanja pravca koji izlazi iz središta pogleda. To je moguće napraviti jer je poznata pozicija kamere u sceni i vektor smjera pogleda čime je jedinstveno definiran pravac. Kako bi se pojednostavilo testiranje prolazi li pravac kroz neku granu, izgrade se omeđujuće kutije oko svake grane. Stranice omeđujućih kutija mogu biti poravnate s osima koordinatnog sustava, čime se još više pojednostavljuje testiranje presjeka pravca i kutije. Problem koji nastaje korištenjem takvih kutija u sceni s puno objekata međusobno male relativne udaljenosti je što raste nepreciznost označavanja.

Drugi način je taj da se prilikom pritiska gumba izradi prikaz scene u pozadini na način da se svaka grana oboji različitom bojom. Sliku izgrađene scene možemo prenijeti s grafičke kartice nazad u glavnu memoriju računala, te potom testiramo koje boje je piksel koji se nalazi u sredini prenesene slike. Tom metodom moguće je prikazati 16 milijuna različitih objekata što je i više nego dovoljno za ovakvu primjenu. Kako korisnik u odnosu na broj prikazanih sličica u sekundi rijetko označava grane, ne dolazi do zagušenja na grafičkoj kartici.

Unutar programa je odabran drugi način za označavanje grana, a glavni razlog je taj što su podaci uvijek već pripremljeni za normalno iscrtavanje scene, pa nije potrebno prenositi nove podatke na grafičku karticu za ponovno iscrtavanje pomoću jednostavnijih oblika.



Slika 17. Usporedba prikaza normalnog i jednostavnog stabla

Na slici 17. je prikazan izgled normalne scene u kojoj se koriste Bézierove krivulje i scene prikazane pomoću kvadara. Na slikama je prikazano 3077 grana te je svaka grana označena

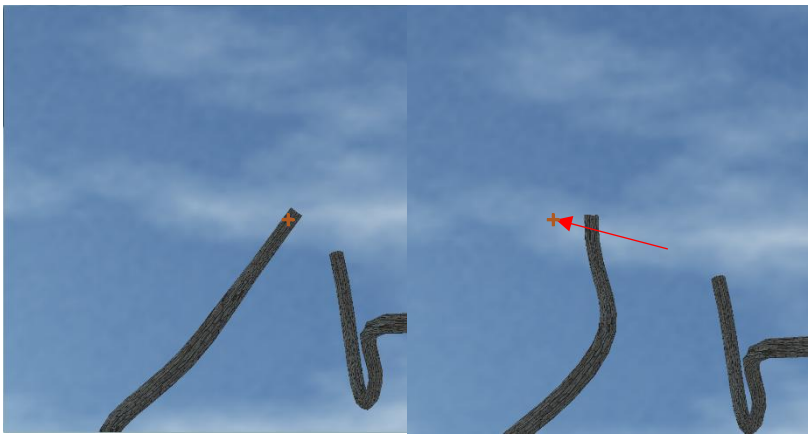
```
Crvena = (index / 65536) mod 256
Zelena = (index / 256) mod 256
Plava = index mod 256
```

Programski odsječak 6. Odabir boje na temelju indeksa

različitom bojom. Odabir boje grane se izvodi po formuli napisanoj unutar programskog odsječka 6., gdje je preciznost podataka za svaku boju 8 bita. Operacija dijeljenja u formuli je cjelobrojno dijeljenje.

Kad se u glavnoj memoriji učita vrijednost označenog piksela, potrebno je izvršiti inverzne operacije kako bi se došlo do jedinstvenog broja grane. Taj broj se zatim prosljeđuje dijelu programa zaduženom za nadziranje pomaka kursora.

U trenutku kada korisnik pomakne kursor izračuna se normala i pozicija plohe koja je paralelna s vektorom pogleda kamere, a središte plohe se nalazi na vrhu označene grane. Korisnik kursorom pomiče sjecište pravca pogleda s plohom. Iz podataka o početnoj i završnoj poziciji točke sjecišta se izračuna vektor sile koji se prosljeđuje grani koja je označena. Na slici 19. je prikazan vektor sile nastao iz početne i završne pozicije kursora (strelica je dodana naknadno na sliku).



Slika 18. Prikaz generirane sile nad granom

9 Zaključak

U ovom radu su predstavljeni algoritmi za proceduralno generiranje stabala.

Ukoliko je potrebno stvoriti model stabla koji najviše podsjeća na prirodno stablo najbolje je koristiti algoritam kolonizacije prostora. Algoritmom kolonizacije prostora je moguće fino podešavati ključna svojstva stabla. Zbog ta dva razloga bi se taj algoritam koristio u situacijama kad je potrebna preciznost i kad se stablo promatra iz blizine.

Ukoliko je potrebno brzo stvoriti veliki broj sličnih stabala fraktalni algoritam je najbolji zbog brzine izvođenja. Fraktalni algoritam je odličan ukoliko je potrebno izraditi stablo s puno ponavljajućih oblika. Fraktalni algoritam nije dobar ukoliko želimo da krošnja stabla poprimi oblik modela s razvedenom površinom.

Algoritam struje čestica izgleda kao najlošiji jer po navedenim kriterijima uvijek ispada na drugom ili trećem mjestu. Ideja algoritma struje čestica je zanimljiva, ali ne generira dovoljno dobre modele i izvršava se dugo. Bolja primjena bi možda bila kod popunjavanja stabla s lišćem, ali u ovom radu to nije obrađeno.

Predstavljen je algoritam vokselizacije modela. Rezultat i vrijeme izvršavanja algoritma su zadovoljavajući. Postoje algoritmi koji paraleliziraju izvođenje ovog algoritma, pa se može još više ubrzati.

Predstavljen je algoritam utjecaja sile. Prilikom izvođenja najviše vremena uzimaju osvježavanje pozicija grana i prenošenje novih vrijednosti na grafičku karticu. Zbog ta dva razloga realno je koristiti algoritam samo na jednom stablu normalne veličine od oko 10000 grana. Ukoliko postoji potreba u realnom vremenu simulirati utjecaj sile na više stabala može se smanjiti kompleksnost svakog stabla.

Primarna primjena opisanih algoritama je za generiranje modela stabla i upravljanje njime, a mogu se iskoristiti u simulacijama, filmovima i video igrama.

10 Literatura

[1] Runions, A., Fuhrer, M., Lane, B., Federl, P., Rolland-Lagan, A. G., & Prusinkiewicz, P. (2005). Modeling and visualization of leaf venation patterns. *ACM Transactions on Graphics (TOG)*, *24*(3), 702-711.

[2] Huang, J., Yagel, R., Filippov, V., & Kurzion, Y. (1998, October). An accurate method for voxelizing polygon meshes. In Proceedings of the 1998 IEEE symposium on Volume visualization (pp. 119-126). ACM.

[3] Minamino, R., & Tateno, M. (2014). Tree branching: Leonardo da Vinci's rule versus biomechanical models. *PloS one*, *9*(4), e93535.

[4] Meagher, D. (1982). Geometric modeling using octree encoding. *Computer graphics and image processing*, *19*(2), 129-147.

[5] Runions, A., Lane, B., & Prusinkiewicz, P. (2007). Modeling Trees with a Space Colonization Algorithm. *NPH*, *7*, 63-70.

[6] Neubert, B., Franken, T., & Deussen, O. (2007, August). Approximate image-based tree-modeling using particle flows. In *ACM Transactions on Graphics (TOG)* (Vol. 26, No. 3, p. 88). ACM.

[7] Prusinkiewicz, P., & Lindenmayer, A. (2012). *The algorithmic beauty of plants*. Springer Science & Business Media.

[8] Pirk, S., Niese, T., Hädrich, T., Benes, B., & Deussen, O. (2014). Windy trees: computing stress response for developmental tree models. ACM Transactions on Graphics (TOG), 33(6), 204.

[9] Schwarz, M., & Seidel, H. P. (2010). Fast parallel surface and solid voxelization on GPUs. ACM Transactions on Graphics (TOG), 29(6), 179.

Sažetak

Algoritmi za proceduralno generiranje i prikaz drveća

U ovom radu su prezentirane varijante proceduralnih algoritama pomoću kojih se mogu izraditi stabla. Svaki algoritam je modificiran kako bi se omogućilo stvaranje modela stabla kojemu krošnja odgovara željenom obliku. Prezentirani su algoritmi utjecaja sile na stablo i opisan je način na koji se stablo može prikazati pomoću Bézierovih krivulja. Objašnjen je postupak vokselizacije modela i njena primjena. Objašnjen je postupak na koji korisnik može uhvatiti granu i pomicati ju. Izrađen je programski proizvod koji omogućuje interakciju s prikazanim stablom.

Ključne riječi: stablo, proceduralno generiranje, kolonizacija prostora, fraktali, strujanje čestica, vokselizacija, simulacija utjecaja sile, Bézierova krivulja, C++, OpenGL

Summary

Algorithms for procedural generation and display of trees

In this thesis, a variety of procedural algorithms capable of generating a tree model is presented. Each algorithm is modified so as to be able to generate a tree model whose shape resembles the required model. Algorithms for force propagation to the tree structure are presented, and the method of representing the tree using Bézier's curves is described. The voxelization procedure of the model and its application are explained. The procedure of grabbing a branch and moving it, intended for the user, is explained. A program product which enables interaction with the displayed tree is created.

Key words: tree, procedural generation, space colonization, fractals, particle flow, voxelization, force propagation, Bézier curve, C++, OpenGL