



УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ
ФАКУЛТЕТ
ДЕПАРТМАН ЗА
МАТЕМАТИКУ И ИНФОРМАТИКУ



Немања Стојановић

Системи итеративних функција,
Линденмајерови системи, фракционо
Брауново кретање и примена у графичком
креирању природних појава

МАСТЕР РАД

Ментор:
Проф. др Данијела Рајтер-Ћирић

Нови Сад, 2015

Садржај

Увод у фракталну геометрију	5
Манделбровов скуп	7
Julia скуп.....	8
Системи итеративних функција (IFS)	11
Афине трансформације	12
Хауздорфова метрика.....	13
Контракција.....	14
Системи итеративних функција	15
n-димензионални системи итеративних функција.....	20
Дводимензионални системи итеративних функција	21
Кондензациони скупови	22
Инверзан проблем – Колаж теорема	24
IFS алгоритми.....	26
Линденмајерови системи	33
Системи преписивања.....	34
DOL-системи	35
Алгоритам корњаче	36
Преписивање ивица.....	39
Преписивање чворова.....	40
Однос између преписивања чворова и преписивања ивица	42
Моделирање у три димензије.....	43
Рачвасте структуре	45
OL-системи са заградама	45
Стохастички L-системи.....	48
Контекст-сензитивни L-системи	50
Параметарски L-системи.....	54
Интерпретација параметарских речи	57

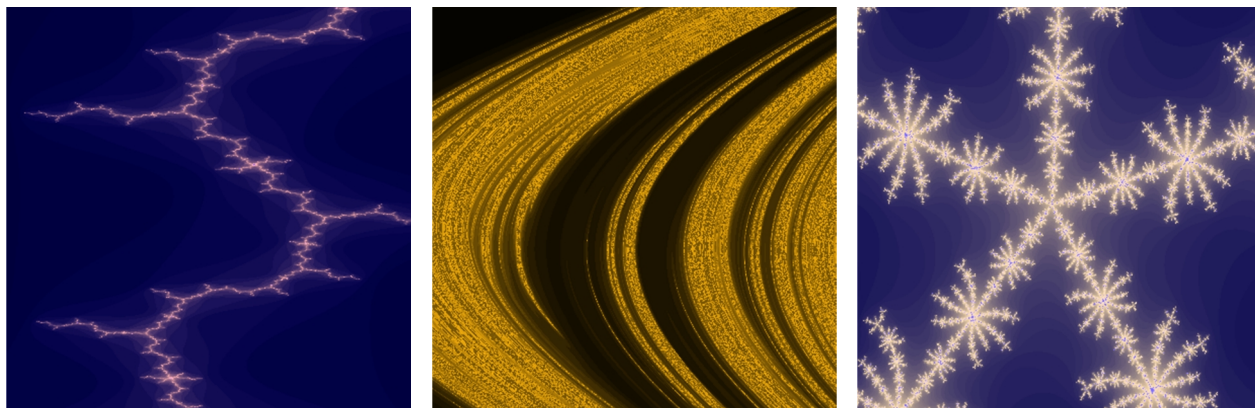
Моделирање дрвећа	57
Симболи.....	66
Симболи који узрокују да се корњача креће напред и црта	66
Симболи који контролишу корњачу у простору	66
Симболи за моделирање структура са гранама	66
Симболи за креирање и укључивање површина	66
Симболи који мењају особине цртања	66
Фракционо Брауново кретање	69
Брауново кретање	69
Фракционо Брауново кретање	73
Фракционе Браунове површине	77
Алгоритми за креирање терена.....	81
Heightmap-е	81
Перлинов шум	81
Прављење Перлиновог шума	86
Примена Перлиновог шума	102
Генерисање текстура са Перлиновим шумом	103
Fault formation алгоритам	106
Diamond-square алгоритам.....	111
Закључак	117
Литература	119
Софтвери коришћени у изради	121

1

Увод у фракталну геометрију

Већина физичких система природе и људских предмета нису регуларни геометријски облици стандардне Еуклидске геометрије. Биолози су традиционално моделирали природу користећи Еуклидско представљање природних објеката. Представљали су откуцаје срца синусним таласима, четинарско дрвеће купама, животињска станишта као просте области и ћелијске мембране као криве или просте површине. Али научници су спознали да се многе природне конструкције боље карактеришу коришћењем фракталне геометрије. Биолошки системи и процеси се типично карактеришу по више нивоа подструктура, са истим општим шаблоном који се понавља. Фрактална геометрија нам даје скоро неограничен број начина за описивање, мерење и предвиђање природних феномена. Али да ли је могуће дефинисати цео свет користећи ове математичке једначине? Фрактали су најбоља могућа дескрипција многих природних форми, као што су обале, планине или делови живих организама.

Фрактална геометрија је прожета у многим областима науке, као што су астрофизика, биолошке науке, и она је једна од најважнијих техника у компјутерској графици.



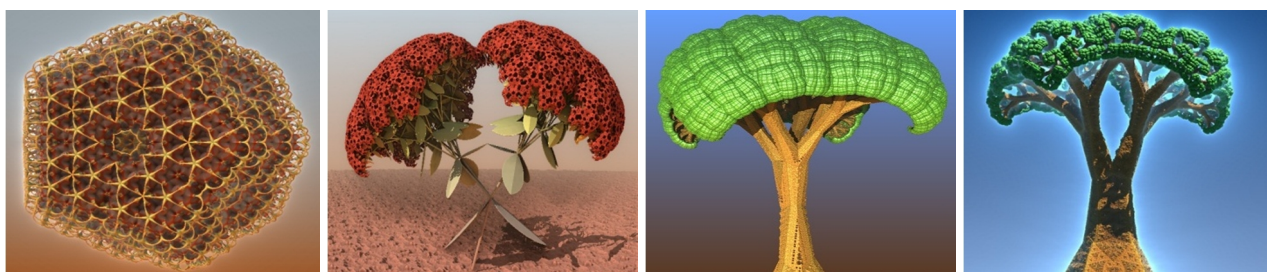
Слика. “Дводимензионални” фрактали муње, Сатурнових прстенова и снежне пахуље креирани у софтверу GIMP

Нико не зна колико звезда светли на нашем небу, али да ли се икад питамо како су настале и како су на крају понашле свој дом у Универзуму? Астрофизичари сматрају да је кључ овог проблема у фракталној природи међузвезданог гаса. Фрактална расподела је хијерархијска, као трагови дима или таласасти облици на небу. Турбуленција обликује и облаке на небу и облике у свемиру дајући им неправилну шему која се понавља, коју би било немогуће описати без фракталне геометрије.

1 Увод у фракталну геометрију

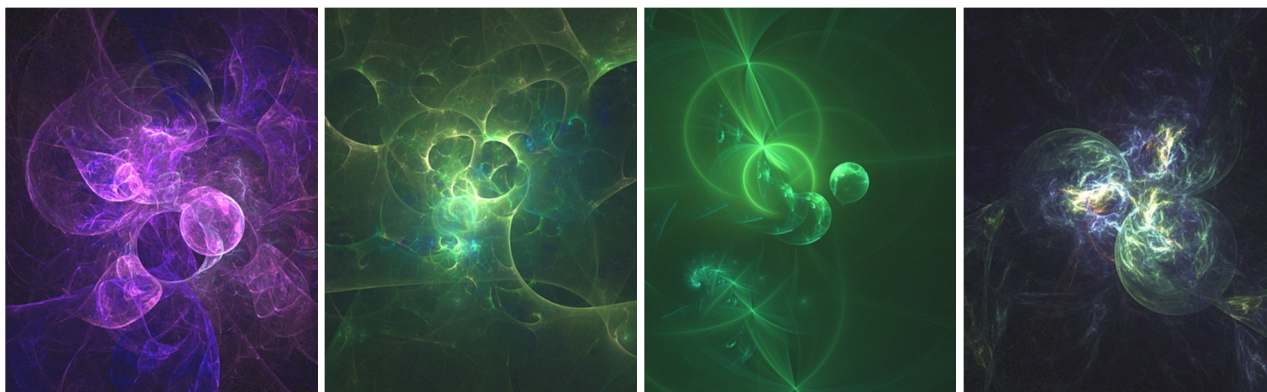
Научници су открили да је основна архитектура хромозома налик структури дрвета. Сваки хромозом се састоји од више „мањих хромозома“ па их стога можемо сматрати фракталима.

Најчешћа примена фрактала у свакодневном животу је у компјутерским наукама. Уметници компјутерске графике користе многе фракталне форме да би креирали текстуре пејзажа и друге комплексне моделе. Могуће је креирати разне врсте разних „фракталних фалсификата“ слика природних сцена као што су: пејзаж месеца, планински венци и обале. Али фрактали могу такође бити кориштени за моделирање природних објеката, омогућавајући нам да математички дефинишемо нашу околину са већом прецизношћу него икада раније



Слика. „Тродимензионални“ фрактали креирани у софтверу Mandelbulb 3D

Многи научници су схватили да је фрактална геометрија моћан алат за откривање тајни за широку разноликост система и решавање важних проблема у примењеној науци. Листа познатих физичких фракталних система је велика и расте брзо. Фрактали побољшавају нашу прецизност у описивању и класификовању „случајних“ органских објеката иако нису перфектни. Можда су само ближи нашем природном свету али не и исти. Неки научници још увек верују да права случајност постоји и да ниједна математичка једначина никад неће моћи да је опише перфектно. Не може се са сигурношћу рећи ко је у праву.



Слика. Фрактали који представљају појаве у свмиру креирани у софтверу Chaotica

Британски картографи су имали проблем са мерењем дужине Британске обале. Обала мерена на обичним мапама је била скоро дупло краћа од обале мерене на детаљним мапама. Што су ближе гледали, више детаља и веће дужине обале је било. Нису знали да су пронашли једну од најважнијих особина фрактала.

1 Увод у фракталну геометрију

Две најважније особине фрактала су:

1. Самосличност – код папрати сваки мали лист (део већег) има исти облик као и цео лист папрати. Лист папрати је самосличан.



Слика. Самосличност код биљака

2. Нецелобројну димензију – класична геометрија се бави објектима са целобројним димензијама: 0D-тачка, 1D-линије и криве, 2D-квадрати и кругови, 3D-купе и сфере...

Многи природни феномени су боље описани користећи димензије између два цела броја. Фрактална крива ће имати димензију између 1 и 2 у зависности од тога колико простора заузима. Фрактал брежуљкастог терена ће имати димензију између 2 и 3. Фрактални пејзаж прекривен великим планинама са заобљеним врховима бити ближе 2 димензије, док ће груба површина састављена од пуно планина средње величине бити ближе 3 димензије. Димензијама фрактала се нећемо бавити у овом раду.

Манделбровов скуп

Развој рачунара седамдесетих година прошлог века омогућио је математичару Манделброту да исцрта хиљаде тачака једноставних итеративних функција. Тако је дошло до стварања Манделбрововог скупа који представља зачетак Фракталне геометрије. Приметио је да се од веома простих итеративних функција могу добити јединствени бесконачни објекти који подсећају на разне облике у природи и који могу да опишу разне комплексне појаве. Манделбровов чувени цитат: „Облаци нису сфере, планине нису купе, обале нису кругови, кора дрвета није глатка, нити се муње крећу правим линијама“ говори о немогућности еуклидске геометрије да дефинише појаве и облике у природи.

Манделбровов скуп је скуп тачака у комплексној равни. Да бисмо га направили користимо алгоритам заснован на рекурзивној формули:

$$Z_n = Z_{n-1}^2 + C .$$

Раздвајамо тачке у комплексној равни у две категорије:

- Тачке унутар Манделбрововог скупа
- Тачке ван Манделбрововог скупа

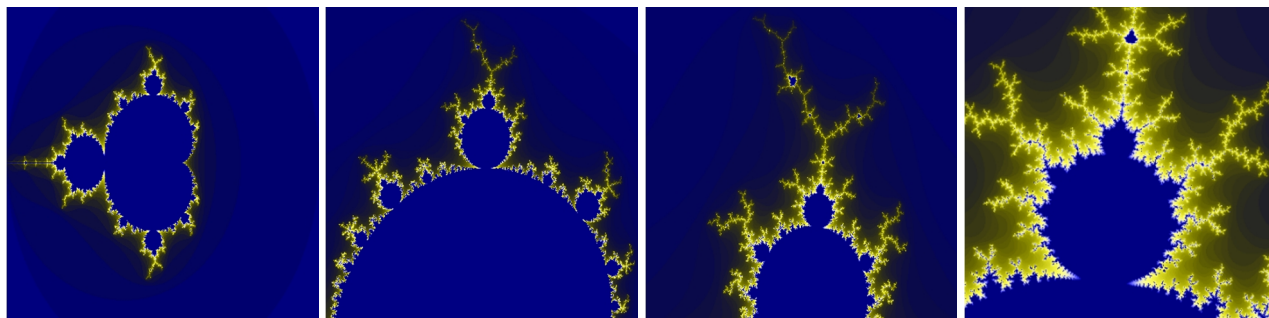
1 Увод у фракталну геометрију

Нпр. ако почињемо са $Z_0 = 0$ и $C = 1$ добијамо низ $0, 1, 2, 5, 26, \dots$ која тежи бесконачности. Како је овај низ неограничен, 1 није елемент Манделбрововог скупа. Са друге стране, $C = -1$ даје низ $0, -1, 0, -1, 0, \dots$ који је ограничен, па је -1 елемент Манделбрововог скупа.

Можемо доделити боју за тачке ван Манделбрововог скупа. Њихове боје зависе од тога колико је итерација неопходно да се утврди да су оне ван Манделбрововог скупа.

Како креирамо Манделбровов скуп?

Да бисмо креирали Манделбровов скуп морамо да изаберемо тачку C на комплексној равни. Комплексан број који одговара овој тачки је $C = a + b \cdot i$. Када израчунавамо вредност првог израза: $Z_1 = Z_0^2 + C$ користећи нулу као вредност од Z_0 , добијамо C као резултат. Следећи корак се састоји од убацивања резултата у Z_1 и понављања калкулације. Сада је резултат комплексан број $C_2 + C$. Затим треба да доделимо вредност за Z_2 и понављамо процес опет. Овај процес може бити представљен као кретање почетне тачке C по равни. Шта се дешава са тачком када понављамо итерације функције? Да ли ће остати близу координатног почетка или ће се удаљавати без границе? У првом случају, ми кажемо да C припада Манделбрововом скупу. Иначе кажемо да иде у бесконачност и додељујемо боју за C у зависности од брзине којом тачка „бежи“ од координатног почетка.



Слика. Приказан је Манделбровов скуп, а затим и три слике увеличаног Манделбрововог скупа. Увећаван је „горњи“ део скупа на свакој слици

Julia скуп

Julia скуп је стриктно повезан са Манделбрововим скупом. Итеративна функција која се користи за добијање Julia скупа је иста као и за Манделбровов скуп. Једина разлика је начин на који се ова функција користи. Како бисмо нацртали Манделбровов скуп, итерирамо функцију за сваку тачку C комплексне равни, увек почињујући од $Z_0 = 0$. Ако хоћемо да креирамо слику Julia скупа, C мора бити константа током целог процеса генерисања, док вредност Z_0 варира. Вредност C дефинише облик Julia скупа (свака тачка комплексне равни је повезана са посебним (партикуларним) Julia скупом).

1 Увод у фракталну геометрију

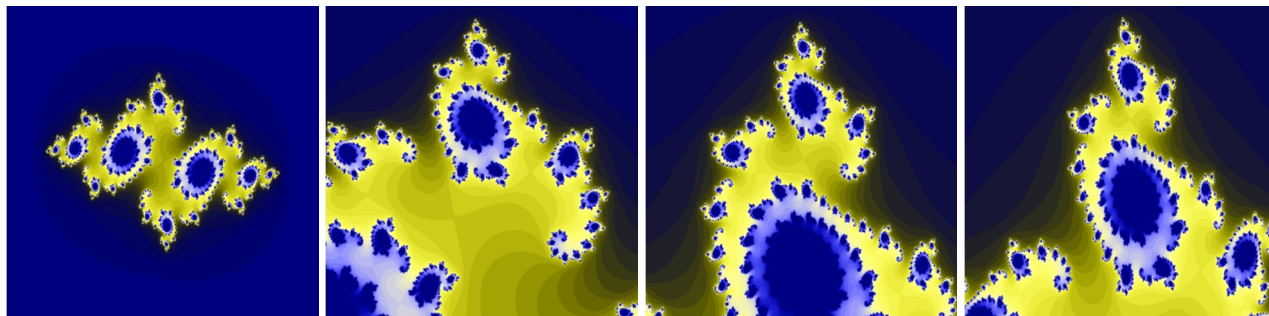
Како се креира Julia скуп?

Морамо да изаберемо тачку C у комплексној равни. Следећи алгоритам одређује хоће ли или неће нека тачка у комплексној равни Z припадати Julia скупу повезаним са C , и одређује боју која би требало да буде додељена. Да бисмо проверили да ли Z припада скупу, морамо да итерирамо функцију $Z_1 = Z_0^2 + C$ користећи $Z_0 = Z$. Шта се дешава са почетном тачком Z када се функција итерира? Да ли ће остати близу координатног почетка или ће се удаљавати од њега, повећавајући раздаљињу од координатног почетка без границе? У првом случају она припада Julia скупу. А у другом иде у бесконачност и дајемо јој боју за Z у зависности од брзине којом тачка „бежи“ од координатног почетка. Да бисмо креирали слику целог Julia скупа повезаним са C , морамо понављати овај процес за све тачке Z чије су координате у следећем интервалу:

$$-2 < a < 2; -1.5 < y < 1.5$$

Најважнија веза између Julia скупа и Манделбрововог скупа је:

Манделбровов скуп је повезан (он је једна целина), а Julia скуп је повезан само ако је повезан са тачком унутар Манделбрововог скупа.



Слика. Приказан је Julia скуп а затим и три слике увеличаног Julia скупа. Увећаван је „горњи“ део скупа на свакој слици

2

Системи итеративних функција (IFS)

За системе итеративних функција ћемо често користити скраћеницу IFS од Iterated Function System. Системи итеративних функција су поступци конструисања фрактала, при чему су добијене конструкције самосличне. Фрактал је начињен од унија неколико копија самог себе, при чему је свака копија трансформисана функцијом (одатле и систем функција). Опис IFS-а даје потенцијалну нову методу за истраживање облика слике и текстуре. Системи итеративних функција се базирају на математичким темељима Хачкинсона, док су истраживања која је вршио Барнслеј довела до значајних метода за разумевање слика. Теорија система итеративних функција је такође и одличан алат за математичко дефинисање хаоса и неправилности.

IFS фрактали имају елегантну рекурзивну дефиницију: Фрактал је конструисан од колажа трансформисаних копија себе; самосличан је и бесконачно скалабилан. Примењујући трансформације којима је фрактал генерисан на исти тај фрактал не доводи до његових промена. Ова обсервација, да је фрактал фиксна тачка система итеративних функција нам омогућава да нађемо трансформације које генеришу дату фракталну криву проналажењем скупа трансформација које мапирају фрактал у себе као унију колекција мањих самосличних копија.

IFS фрактали су креирани простим трансформацијама равни: скалирањем, дислокацијом, ротацијом... Креирање IFS фрактала се састоји од следећих корака:

1. Дефинисање скупа трансформација равни
2. Цртање почетног образаца у равни (било каквог)
3. Трансформисање почетног обрасца користећи трансформације дефинисане у првом кораку.
4. Трансформисање нове слике (коришћење почетних и трансформисаних образаца) користећи исти скуп трансформација.
5. Понављање четвртог корака што је више пута могуће (у теорији ова процедура може да се понавља бесконачан број пута).

Афине трансформације

Дефиниција: Трансформација $f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ облика:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = f \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

где су $(x, y), (x', y') \in \mathbb{R}^2$ било које тачке у равни и $a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2 \in \mathbb{R}$, назива се дводимензионална афина трансформација.

Матрица:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

је комбинација ротације, скалирања, смицања, и других трансформација.

Вектор:

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

дефинише транслациони део трансформације.

Матрицу A увек можемо представити и у следећем облику:

$$\begin{pmatrix} r_1 \cos \theta_1 & -r_2 \sin \theta_2 \\ r_1 \sin \theta_1 & r_2 \cos \theta_2 \end{pmatrix}$$

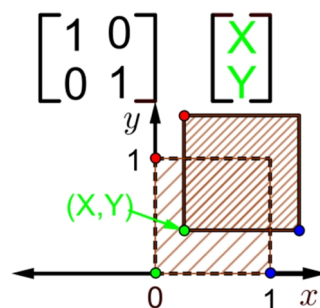
где су r_1, r_2 фактори скалирања примењени на X и Y осе, а θ_1 и θ_2 се односе на ротацију оса.

На следећем примеру ћемо показати како се налази афина трансформација.

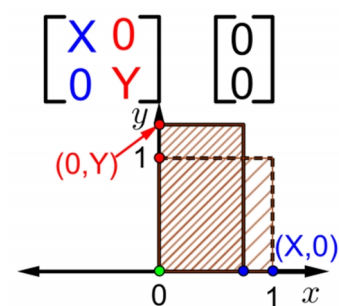
Потребно је од великог листа да приближно добијемо мали лист (Слика 1). Желимо да нађемо $a_{11}, a_{12}, a_{21}, a_{22}, b_1, b_2$ тако да је f (великог листа) приближно једнако малом листу.

Прво што треба да урадимо је да означимо x и y осе. Затим означимо по 3 тачке које се налазе на истим „локацијама“ на листу

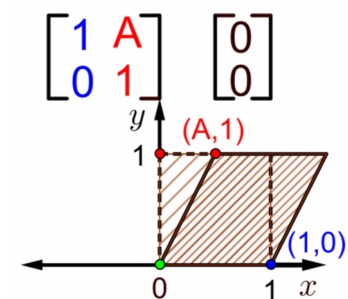
Транслирање



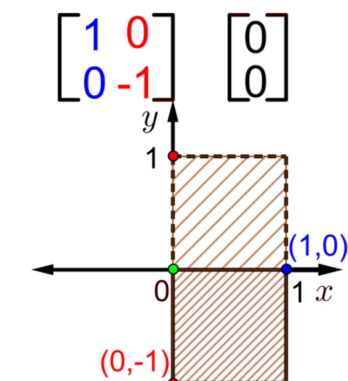
Скалирање



Смицање по x



Рефлексација око x -осе



2 Системи итеративних функција

и одредимо им координате.

a_{11}, a_{12}, b_1 добијамо решавајући систем:

$$x_1 a_{11} + x_2 a_{12} + b_1 = x_1'$$

$$y_1 a_{11} + y_2 a_{12} + b_1 = y_1'$$

$$z_1 a_{11} + z_2 a_{12} + b_1 = z_1'$$

а a_{21}, a_{22}, b_2 , добијамо решавајући:

$$x_1 a_{21} + x_2 a_{22} + b_2 = x_2'$$

$$y_1 a_{21} + y_2 a_{22} + b_2 = y_2'$$

$$z_1 a_{21} + z_2 a_{22} + b_2 = z_2'$$

Системи итеративних функција у равни су дати са коначном колекцијом ових афиних трансформација које делују на метрички простор.

За даљи рад неопходно је и да дефинишемо Хауздорову метрику.

Хауздорова метрика

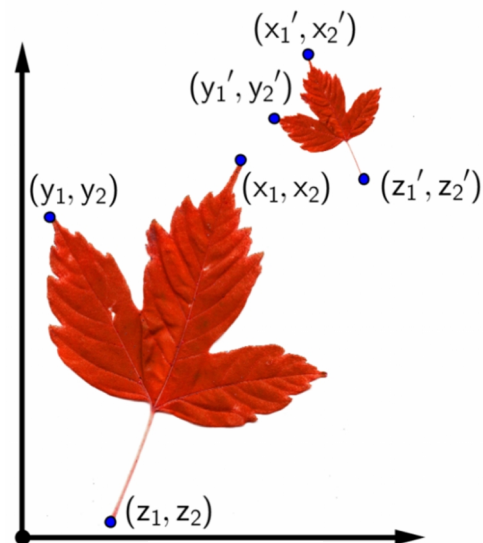
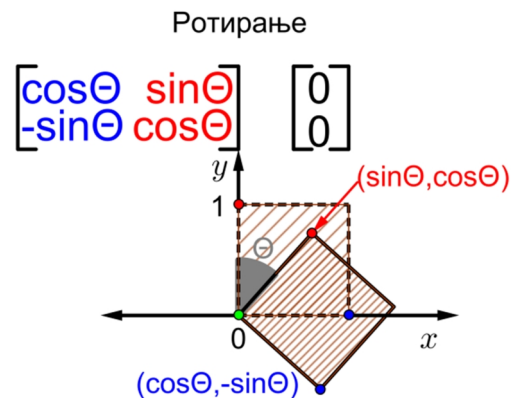
Дефиниција: Хауздорово растојање између два подскупа A и B од X је дефинисано на следећи начин:

$$d_H(A, B) = \max \left\{ \sup_{x \in A} \inf_{y \in B} d(x, y), \sup_{y \in B} \inf_{x \in A} d(x, y) \right\}.$$

Претпоставимо да су A и B компактни подскупови равни. Онда је Хауздорово растојање између A и B

$$h(A, B) = \inf \{ \varepsilon \geq 0 : A \subseteq B_\varepsilon \wedge B \subseteq A_\varepsilon \}.$$

Нпр, претпоставимо да је A црвени квадрат димензије 2, а B правоугаоник са ширином 2



Слика 1

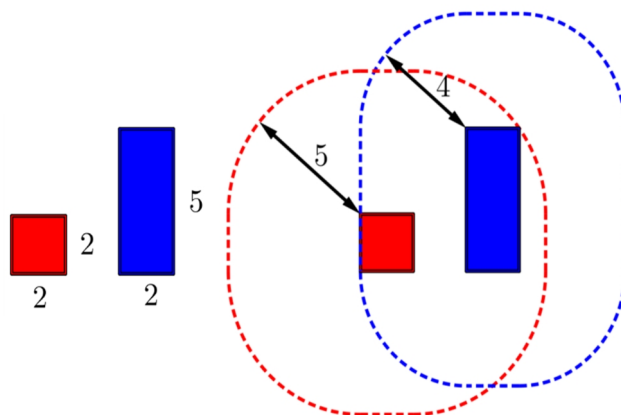


График. Скуп A , скуп B и њихове ε околине

2 Системи итеративних функција

и висином 4 и нека је растојање између њих такође 2. Шта је $h(A, B)$?

Са X_ε означавамо ε околину слике X .

Видимо да је: A садржано у B_4 , а B садржано у A_5 .

Ово су најмање околине која ће да функционишу:

За $\varepsilon < 5$, A_ε не садржи горњу десну страну од B , а за $\varepsilon < 4$, B_ε не садржи леви део од A .

Па је $h(A, B) = 5$.

Нећемо показивати да је Хауздорфова метрика заиста метрика. Претпоставићемо да важи да је метрика d фундаментална метрика за метрику $h(d)$ и да контракције индукују контракције у Хауздорфовој метрици.

Контракција

Дефиниција: Трансформација $f: X \rightarrow X$ на (комплетном) метричком простору (X, d) се назива контракција или контрактивно мапирање ако постоји константа $0 \leq s < 1$ таква да важи

$$d(f(x), f(y)) \leq s \cdot d(x, y), \forall x, y \in X.$$

Сваки такав број s се зове контрактивни фактор трансформације f .

Најмањи број s који задовољава ову неједначину се назива Липшицова константа од f .

Број d представља Еуклидско растојање између тачака:

$$d((x, y), (x', y')) = ((x - x')^2 + (y - y')^2)^{\frac{1}{2}}.$$

У $\mathbb{R}^2$, контракциони фактор од f је најмање s које задовољава:

$$d(f(x, y), f(x', y')) \leq s \cdot d((x, y), (x', y')),$$

за сваки пар тачака $(x, y), (x', y')$.

Контракција је трансформација f која редукује растојање између било којег пара тачака.

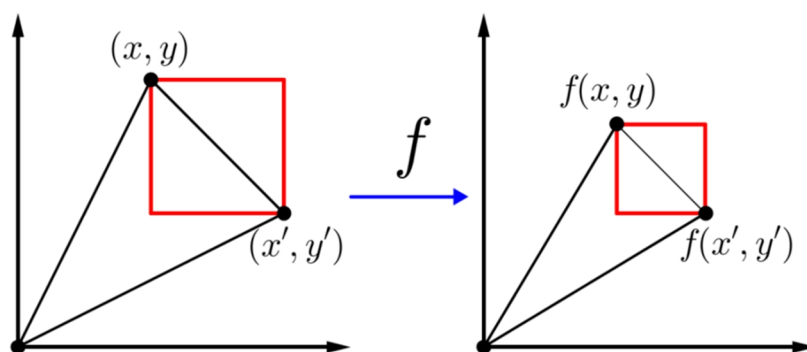


График. Контрактивно пресликавање

2 Системи итеративних функција

Контракције могу да редукују растојање између тачака за различите износе, у зависности од позиције тачака.

Системи итеративних функција

Дефиниција: Нека је (X, d) комплетан метрички простор. Нека је $f_i : X \rightarrow X$ колекција мапирања $(f_i; i = 1, 2, \dots, n)$. Онда се

$$\Omega = \{X; f_i, i = 1, 2, \dots, n\}$$

зове систем итеративних функција.

Теорема: (Теорема контрактивног мапирања – Банахова теорема о фиксној тачки). Нека је $f : X \rightarrow X$ контрактивно мапирање на комплетном метричком простору (X, d) . Онда f поседује тачно једну фиксну тачку $A \in X$ и, штавише, за сваку тачку $x \in X$, низ $\{f^{\circ k}(x) : k = 0, 1, 2, \dots\}$ конвергира ка A . Односно:

$$\lim_{k \rightarrow \infty} f^{\circ k}(x) = A, \quad \forall x \in X$$

Доказ: Нека је $x \in X$ и нека је $0 \leq s < 1$ контрактивни фактор за f . Онда је

$$d(f^{\circ k}(x), f^{\circ m}(x)) \leq s^{m \wedge k} d(x, f^{\circ |k-m|}(x)) \quad (1)$$

за свако $k, m = 0, 1, 2, \dots$, где имамо фиксно $x \in X$. Нотација $u \wedge v$ представља минимум пара реалних бројева u и v . За $i = 0, 1, 2, \dots$, имамо

$$\begin{aligned} d(x, f^{\circ i}(x)) &\leq d(x, f(x)) + d(f(x), f^{\circ 2}(x)) + \dots + d(f^{\circ(i-1)}(x), f^{\circ i}(x)) \\ &\leq (1 + s + s^2 + \dots + s^{i-1}) d(x, f(x)) \\ &\leq (1 - s)^{-1} d(x, f(x)), \end{aligned}$$

па убацавањем у једначину (1) добијамо

$$d(f^{\circ k}(x), f^{\circ m}(x)) \leq s^{m \wedge k} \cdot (1 - s)^{-1} \cdot d(x, f(x)),$$

одакле следи да је $\{f^{\circ k}(x)\}_{k=0}^{\infty}$ Кошијев низ. Пошто је X комплетан, овај Кошијев низ је ограничен са $A \in X$, и имамо

$$\lim_{k \rightarrow \infty} f^{\circ k}(x) = A.$$

2 Системи итеративних функција

Сада ћемо показати да је A фиксна тачка од f . Пошто је f контрактивна она је и непрекидна па следи

$$f(A) = f(\lim_{k \rightarrow \infty} f^{\circ k}(x)) = \lim_{k \rightarrow \infty} f^{\circ(n+1)}(x) = A.$$

Треба још да проверимо јединственост фиксне тачке. Претпоставимо да су A и B две фиксне тачке од f . Онда је $A = f(A)$, $B = f(B)$, и

$$d(A, B) = d(f(A), f(B)) \leq sd(A, B),$$

где је $(1-s)d(A, B) \leq 0$, одакле следи $d(A, B) = 0$, односно $A = B$.

Ова теорема тврди да свако контрактивно мапирање на непразном комплетном метричком простору има јединствену фиксну тачку и да за свако $x \in X$ низ итеративних функција $x, f(x), f(f(x)), f(f(f(x))), \dots$ конвергира ка фиксној тачки.

Теорема: (Контрактивно мапирање са фиксном тачком). Нека је (X, d) комплетан метрички простор. Нека је $f_i: X \rightarrow X$ колекција мапирања $(f_i; i=1, 2, \dots, n)$. Онда $\Omega = \{X; f_i, i=1, 2, \dots, n\}$ има јединствену фиксну тачку која се назива атрактор A .

Дефиниција: (Хиперболични) систем итеративних функција (IFS) се састоји од комплетног метричком простора (X, d) заједно са коначним скупом контрактивних мапирања $f_i: X \rightarrow X$, којима одговарају контрактивни фактори s_i . Његов контрактивни фактор је $s = \max\{s_i : i=1, 2, \dots, n\}$.

Ако је X комплетан метрички простор и све мапе f_i су контрактивне онда се за IFS каже да је хиперболичан.

Реч „хиперболичан“ смо ставили у заграду јер се често у пракси изоставља. Често ћемо користити скраћеницу IFS када мислимо на коначан скуп мапа које делују на метрички простор, без икаквих посебних услова наметнутих на мапе.

Дефиниција: Ако је IFS $\Omega = \{X; f_i, i=1, 2, \dots, n\}$ контрактиван, онда постоји јединствени скуп

$$A = f(A) = \bigcup_{i=1}^n f_i(A)$$

Који се назива фиксна тачка (атрактор) од Ω .

2 Системи итеративних функција

У наставку ћемо радити на простору фрактала $(H(X), h(d))$. $(H(X), h(d))$ је простор непразних компактних подскупова метричког простора (X, d) са Хаздорфовом метриком $h(d)$.

За доказ наредне теореме неопходно је да докажемо следећу Лему.

Лема: Нека је $f: X \rightarrow X$ непрекидно пресликавање на метричком простору (X, d) . Онда f пресликава $H(X)$ на $H(X)$.

Доказ: Нека је S непразни компактан подскуп од X . Онда је $f(S) = \{f(x) : x \in S\}$ исто непразан. Хоћемо да покажемо да је $f(S)$ компактан. Нека је $\{y_n = f(x_n)\}$ бесконачан низ тачака у S . Онда је $\{x_n\}$ бесконачан низ тачака у S . Пошто је S компактан, постоји подниз $\{x_{N_n}\}$ који конвергира ка тачки $x' \in S$. Али онда непрекидност од f имплицира да је $\{y_{N_n} = f(x_{N_n})\}$ подниз од $\{y_n\}$ који конвергира ка $y' = f(x') \in f(S)$.

Теорема: Нека је (X, d) метрички простор. Нека је $\{f_i : i = 1, 2, \dots, n\}$ контрактивно мапирање на $(H(X), h(d))$. Нека је контрактивни фактор за f_k означен са s_k за свако k . Дефинишемо $F: H(X) \rightarrow H(X)$ као:

$$F(B) = f_1(B) \cup f_2(B) \cup \dots \cup f_n(B) = \bigcup_{i=1}^n f_i(B), \text{ за свако } B \in H(X).$$

Онда је F контрактивно мапирање са контрактивним фактором $s = \max \{s_i : i = 1, 2, \dots, n\}$.

Доказ: Доказаћемо теорему за $n = 2$. Нека је $B, C \in H(X)$. Имамо

$$\begin{aligned} h(F(B), F(C)) &= h(f_1(B) \cup f_2(B), f_1(C) \cup f_2(C)) \\ &\leq h(f_1(B), f_1(C)) \vee h(f_2(B), f_2(C)) \quad (\text{Лема}) \\ &\leq s_1 h(B, C) \vee s_2 h(B, C) \leq s h(B, C). \end{aligned}$$

Следећа теорема сумира главне чињенице о хиперболичним IFS.

Теорема: Нека је $\{X; f_i, i = 1, 2, \dots, n\}$ хиперболичан систем итеративних функција са контрактивним фактором s . Онда је трансформација $F: H(X) \rightarrow H(X)$ дефинисана као:

2 Системи итеративних функција

$$F(B) = \bigcup_{i=1}^n f_i(B), \quad \forall B, C \in H(X)$$

контрактивно мапирање на комплетном метричком простору $(H(X), h(d))$ са контрактивним фактором s . Односно:

$$h(F(B), F(C)) \leq s \cdot h(B, C), \quad \forall B, C \in H(X).$$

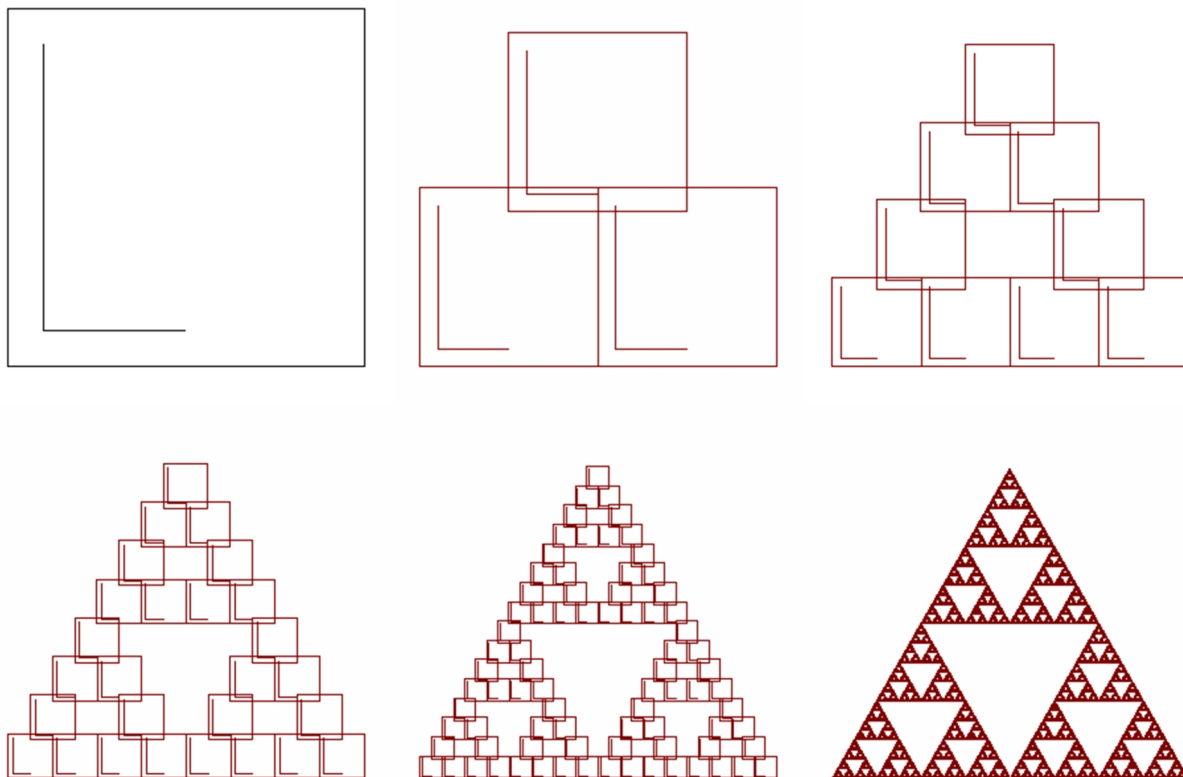
Њена јединствена фиксна тачка, $A \in H(X)$ задовољава:

$$A = F(A) = \bigcup_{k=1}^n f_k(A)$$

И дата је са $A = \lim_{k \rightarrow \infty} F^{\circ k}(B)$ за свако $B \in H(X)$.

Фиксна тачка $A \in H(X)$ у теорему се назива атрактор IFS-а.

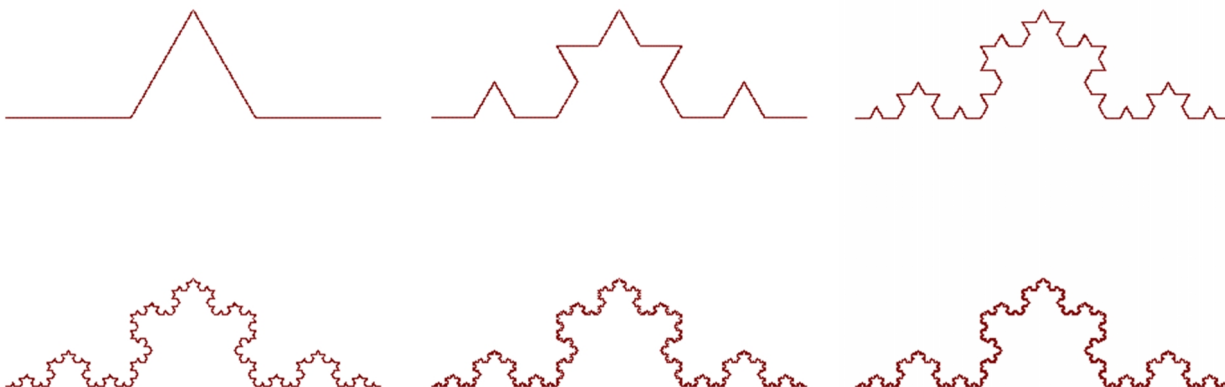
На следећем графику је приказан атрактор добијен трансформацијама јединичног квадрата. Следећи пример, као и већину наредних примера система итеративних функција креирао сам у софтверу IFS kit.



Слика. Дат је приказ креирања фрактала под називом троугао СIERПИНСКОГ – фрактала коначног обима са површином једнаком нула

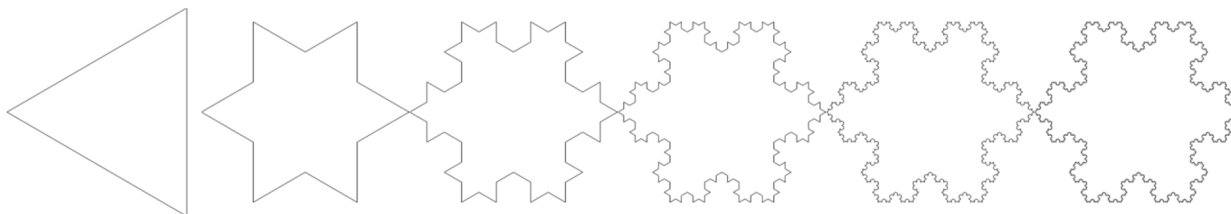
2 Системи итеративних функција

Приказан је и атрактор који се назива Кочова пахуља. Следећа слика представља првих шест итерација за креирање Кочове криве.



Слика. Првих шест итерација Кочове криве

Ако поновимо поступак креирања Кочове криве на троуглу добијамо Кочову пахуљу.



Слика. Приказано је првих неколико корака креирања фрактала, Кочове пахуље – фрактала коначне површине, а бесконачног обима

Дефиниција: Дат је систем итеративних функција Ω у равни са коначном колекцијом контрактивних афино трансформација $f_i: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ за свако $(f_i; i=1, 2, \dots, n)$. IFS са вероватноћама је IFS такав да је сваком f_i додељена вероватноћа p_i , $p_1 + p_2 + \dots + p_n = 1$:

$$\Omega = \{ \mathbb{R}^2, (f_i, p_i); i=1, 2, \dots, n \}$$

Дефиниција: Нека је систем итеративних функција $\Omega = \{ \mathbb{R}^2, (f_i, p_i); i=1, 2, \dots, n \}$ нехиперболични IFS са вероватноћама. Нека су контрактивни фактори IFS-а $(s_i; i=1, 2, \dots, n)$. IFS модел је контрактиван и поседује јединствену фиксну тачку која је њен атрактор A када је просечан контракциони услов задовољен:

$$s_1 p_1 \cdot s_2 p_2 \cdot \dots \cdot s_N p_N < 1$$

n-димензионални системи итеративних функција

n-димензионалне системе итеративних функција ћемо скраћено означавати са n-D IFS. Класични системи итеративних функција функционишу у равни. Међутим, модерни модели компјутерске графике су често приказани као апстрактни n-димензионални векторски простори укључујући три-димензионалне просторе слика и три-димензионалан простор боја. Да бисмо оценили те приказе у фракталном моделирању слика, уводимо n-димензионалне системе итеративних функција. n-D IFS ће се састојати од афиних трансформација које делују на $\mathbb{R}^n$ простор.

Аналогно као и код IFS у равни, свака афина трансформација $f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ n-D простора има форму:

$$\begin{pmatrix} x' \\ y' \\ \vdots \\ n' \end{pmatrix} = f \begin{pmatrix} x \\ y \\ \vdots \\ n \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x \\ y \\ \vdots \\ n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

где је $(x, y, \dots, n) \in \mathbb{R}^n$ било која тачка у n-D простору.

Матрица

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

је комбинација различитих оператора. Све оне делују на простор слика (x, y) , где лако можемо да препознамо ротације, скалирање и смицање.

Вектор

$$(b_1, b_2, \dots, b_n)$$

одређује транслације у равни и дефинише основне вредности за друге компоненте.

Дефиниција: Нека је $(\mathbb{R}^n, d)$ комплетан метрички простор. Нека је $f_i: \mathbb{R}^n \rightarrow \mathbb{R}^n$ колекција мапирања $(f_i; i=1, 2, \dots, n)$ и нека сви f_i имају одговарајуће вероватноће p_i , где је $p_1 + p_2 + \dots + p_n = 1$. Онда се

$$\Omega = \{ \mathbb{R}^n, (f_i, p_i); i=1, 2, \dots, n \}$$

2 Системи итеративних функција

назива n -D систем итеративних функција.

Дефиниција: За мапирање $f_i: \mathbb{R}^n \rightarrow \mathbb{R}^n$ из метричког простора $(\mathbb{R}^n, d)$ се каже да је контрактивно ако постоји позитиван реалан број $0 \leq s < 1$ тако да важи:

$$\forall (\alpha_1, \alpha_2) \in \mathbb{R}^n, d(f_i(\alpha_1), f_i(\alpha_2)) \leq s_i d(\alpha_1, \alpha_2).$$

где су (α_1, α_2) тачке у $\mathbb{R}^n$ и d је растојање у n -D простору.

Веома је тешко задржати све s_i факторе испод јединице за n -димензионални IFS модел.

Целокупан проблем процењивања s_i -ова је много компликованији за n -D случајеве.

Због практичне апликације, можемо искористити просечан контрактивни услов:

Дефиниција: Нека је систем итеративних функција $\Omega = \{ \mathbb{R}^2, (f_i, p_i); i = 1, 2, \dots, n \}$ нехиперболични IFS са вероватноћама. Нека IFS има контрактивне факторе $(s_i; i = 1, 2, \dots, n)$. IFS је контрактиван и поседује јединствену фиксну тачку која се назива атрактор A када је услов просечне контракције задовољен $s_1 p_1 \cdot s_2 p_2 \cdot \dots \cdot s_n p_n < 1$

Нехиперболични n -D IFS има атрактор чак и у неким случајевима када су контрактивни односи $s_i > 1$. Ако је само један довољно мали, $s_j < 1$, онда је услов просечне контракције задовољен и n -D систем итеративних функција има фиксну тачку у n -D простору. Такав IFS има посебне карактеристике које допуштају директну манипулацију n -димензионалних фрактала.

Дводимензионални системи итеративних функција

Дводимензионални IFS је специјалан случај n -D IFS-а. За компактан метрички простор $X = \mathbb{R}^2$, имаћемо IFS скуп $\Omega \{ X; f_i, i = 1, 2, \dots, n \}$ који се састоји од n афиних мапа f_i . Због једноставности, не разматрамо IFS са вероватноћама, односно претпостављамо да су све повезане вероватноће једнаке и сума им је једнака један. Свака афина трансформација $f: X \rightarrow X$ или $f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ има форму:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = f \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

где су $(x, y), (x', y') \in \mathbb{R}^2$ било које тачке у равни.

Матрица дефинисана са $a_{11}, a_{12}, a_{21}, a_{22}$ и вектор који се састоји од b_1, b_2 могу се конструисати од обе врсте, и линеарних и нелинеарних функција.

2 Системи итеративних функција

2-D бинарни IFS је најкласичнија имплементација система итеративних функција. Ако претпоставимо да су матрица која се састоји од $a_{11}, a_{12}, a_{21}, a_{22}$ и вектор b_1, b_2 дефинисани са линеарном функцијом, онда је могуће оценити контрактивни фактор за сваку трансформацију. IFS не мора бити хиперболичан и неки контракциони фактори могу бити близу или чак већи од један. За такве случајеве добићемо нехиперболичне IFS-ове, али могу и даље да садрже фиксне тачке (атракторе), докле год је контракција задовољена. Такође је могуће да уведемо нелинеарност у IFS код.

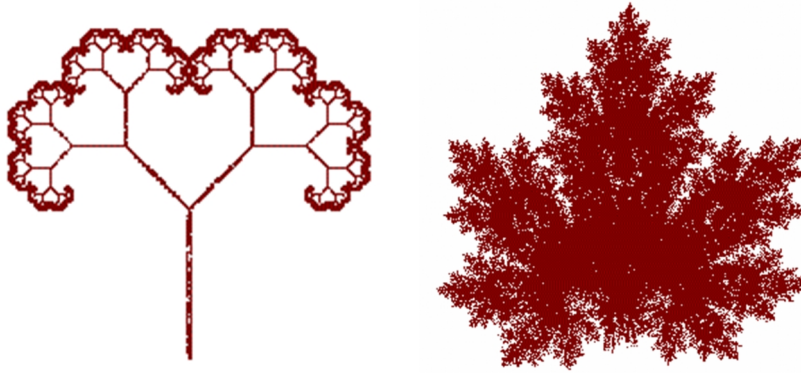
У класичном систему итеративних функција, матрица описана са $a_{11}, a_{12}, a_{21}, a_{22}$ представља комбинацију ротације, рескалирања и смицања у равни. IFS параметри такође могу бити представљени и у облику тригонометријских функција. Такав приказ је веома користан у конструкцији. Можемо да извршимо ротације, искосимо, скалирамо и рефлектујемо. Вектор дефинисан са b_1, b_2 представља просте транслације у равни према центру од површинских координата оног што се креира. Овај прост али ефикасан модел је веома популаран у IFS моделирању. Иако постоје нелинеарне екстензије тог модела, укључујући поларни IFS, чак и најлакши случај је позадина за неколико интересантних имплементација.

Кондензациони скупови

Постоји још једна врста фрактала која је релативно честа: фрактали са кондензационим скуповима. Разматрамо фрактално дрво на следећој слици. Овај фрактал није баш само-сличан. Фрактално дрво садржи два мања дрвета, али стабло дрвета не припада ни једном од мањих дрвећа. Стабло фракталног дрвета се назива кондензациони скуп. Ако се фрактал састоји од самосличних делова заједно са додатним скупом, додатни скуп се назива кондензациони скуп. Према томе, да бисмо генерисали фрактал, морамо да спецификаујемо три ствари: скуп трансформација (IFS), базни случај (у којем почињемо итерацију) и кондензациони скуп (могуће је да буде празан скуп).

Постоје два начина како да се бавимо кондензационим скуповима. Један приступ је да узмемо базни случај да буде кондензациони скуп и укључимо јединичну трансформацију у IFS-у.

Предност ове технике је проста, али хендикеп је да функција идентитета мора бити примењена у сваком нивоу. Штавише, у овом приступу фрактал није више независан од базног случаја. Алтернативни метод је да просто додамо кондензациони скуп назад у криву у сваком нивоу након примењивања трансформације. Предност овог приступа је да не трошимо време сувишним примењивањем функције идентитета. Такође можемо и да користимо било који базни случај; Нисмо више ограничени тако да морамо да почнемо са кондензационим скупом.



Слика 2.

На слици 2 кондензациони скуп представља стабло фракталног дрвета док би код јаворовог листа кондензациони скуп представљала граница уколико би је додали.

Дефиниција: Нека је (X, d) метрички простор и нека је $C \in H(X)$. Дефинишемо трансформацију $f_0 : H(X) \rightarrow H(X)$ са $f_0(B) = C$ за свако $B \in H(X)$. Тада се f_0 зове кондензациона трансформација и C се зове кондензациони скуп.

Кондензациона трансформација $f_0 : H(X) \rightarrow H(X)$ је контрактивно мапирање на метричком простору $(H(X), h(d))$, са контрактивним фактором једнаким нула. Ова трансформација поседује јединствену фиксну тачку која се назива кондензациони скуп.

Теорема: Нека је $\{X; f_i; i = 0, 1, 2, \dots, n\}$ хиперболичан систем итеративних функција са кондензационим скупом C и са контрактивним факторима $(s_i; i = 1, 2, \dots, n)$, $0 \leq s < 1$. Онда је трансформација $F : H(X) \rightarrow H(X)$ дефинисана са

$$F(B) = \bigcup_{i=0}^n f_i(B) \quad \forall B \in H(X)$$

контрактивно мапирање на комплетном метричком простору $(H(X), h(d))$ са контрактивним фактором s таквим да:

$$h(F(B), F(C)) \leq s \cdot h(B, C) \quad \forall B, C \in H(X).$$

Њена јединствена фиксна тачка, $A \in H(X)$ задовољава:

$$A = F(A) = \bigcup_{i=0}^n f_i(A)$$

и дата је са $A = \lim_{k \rightarrow \infty} F^{\circ k}(B)$ за свако $B \in H(X)$.

2 Системи итеративних функција

Дефиниција: Нека је систем итеративних функција $\Omega = \{\mathbb{R}^2; f_i, i = 1, 2, \dots, n\}$ хиперболични IFS. Нека IFS има контраktivне факторе $(s_i; i = 1, 2, \dots, n)$ $0 \leq s < 1$. Нека је $f_0 : H(X) \rightarrow H(X)$ кондензациона трансформација. Онда се $\Omega = \{\mathbb{R}^2; (f_0, f_i), i = 1, 2, \dots, n\}$ назива хиперболичан IFS са кондензацијом и често се обележава са IFS_c .

Кондензационе трансформације могу бити корисне моделирајући неке структуре биљака или могу да дају илузију изгледа IFS фрактала. Међутим, кондензационе трансформације нису баш интуитивне и изазивају проблеме у примени Колаж теореме.

Инверзан проблем – Колаж теорема

Инверзан проблем за системе итеративних функција може бити изложен на следећи начин: За дату почетну слику I , ми треба да нађемо контраktivан IFS са атрактором који репродукује ову слику. Са практичне тачке гледишта, ми желимо да IFS атрактор буде што је ближе могуће датој слици за дефинисану функцију растојања. Пошто фрактални скупови обично садрже бесконачан број тачака са веома комплексном организацијом, није могуће тачно спецификовати где свака тачка у њему лежи. Апроксимациони скуп је дефинисан по односима између делова. Делови су мале копије оригиналне слике атрактора. Колаж теорема је веома користан алат за упрошћавање инверзног проблема за дводимензионалне системе итеративних функција.

Следећа теорема је централна за дизајн IFS-ова чији су атрактори близу датих скупова.

Теорема (Колаж теорема): Нека је (X, d) комплетан метрички простор и нека је дато $e \geq 0$ и слика $I \in H(X)$. Изаберимо IFS (или IFS са кондензацијом) $\{X; (f_0), f_1, f_2, \dots, f_n\}$ са контраktivним фактором $0 \leq s < 1$, тако да је

$$h(I, \bigcup_{i=0}^n f_i(I)) \leq e,$$

где је $h(d)$ Хауздорфова метрика. Онда је $h(I, A) \leq \frac{e}{1-s}$, где је A атрактор IFS-а.

Еквивалентно,

$$h(I, A) \leq (1-s)^{-1} h(I, \bigcup_{i=0}^n f_i(I)) \text{ за свако } I \in H(X).$$

Теорема нам говори следеће: Да бисмо нашли IFS чији је атрактор „близу“ или „личи“ на дати скуп, морамо настојати да нађемо скуп трансформација - контраktivних мапирања на погодном простору унутар којег дати скуп лежи, тако да је унија, или колаж, слика датог скупа под трансформацијама близу датог скупа. „Близина“ се мери користећи Хауздорфову метрику.

2 Системи итеративних функција

Колаж теорема нам омогућава приступ инверзном проблему: За дат скуп I , пронаћи IFS за који је I атрактор. Основни математички принцип је веома лак: Доказ следеће Леме је скица доказа Колаж теореме.

Лема: Нека је (X, d) комплетан метрички простор. Нека је $f : X \rightarrow X$ контрактивно мапирање са контрактивним фактором $0 \leq s < 1$, и нека је фиксна тачка од f , $x_f \in X$. Онда је

$$d(x, x_f) \leq (1-s)^{-1} \cdot d(x, f(x)), \text{ за свако } x \in X$$

Доказ: Функција растојања $d(a, b)$, за фиксирано $a \in X$, је непрекидна у $b \in X$. Па је

$$d(x, x_f) = d(x, \lim_{n \rightarrow \infty} f_n(x)) = \lim_{n \rightarrow \infty} d(x, f_n(x))$$

$$\leq \lim_{n \rightarrow \infty} \sum_{m=1}^n d(f_{m-1}(x), f_m(x))$$

$$\leq \lim_{n \rightarrow \infty} d(x, f(x))(1 + s + \dots + s^{n-1}) \leq (1-s)^{-1} d(x, f(x)).$$

Ово комплетира доказ.

Ако можемо да направимо мале копије неког почетног објекта I и нађемо скуп трансформација – контрактивних мапирања на простор у којем објекти леже са минималним преклапањем афиних мапирања, онда можемо да апроксимирамо ту слику са одговарајућим IFS скупом. Морамо пронаћи редукцијски фактор и транслацију за сваку копију. Такође, угао ротације (ако је неопходно) мора бити одређен. Ми такође морамо и да одлучимо да ли је копија рефлексија оригинала или није. Да бисмо добили тачну унију или колаж, морамо да минимизирамо Хауздорфову раздаљину између оригиналног објекта и копија које је преслијавају. Требало би да минимизирамо преклапање копија и укупног броја IFS трансформација.

Постоје неке тешкоће код колаж теореме:

Ако су неке трансформације слабо контрактивне (s_i је близу јединице), онда десни део неједначине $d_H(I, A) \leq \frac{\varepsilon}{(1-s)}$ може постати екстремно велики. Ограничење губи поенту.

Контрактивни фактор може лако да се одржи испод јединице за дводимензионални IFS који се састоји само од линеарних функција. За n -димензионални IFS и нелинеарне IFS, контрактивни фактор не мора бити униформан. То га чини немогућим за процену.

Хауздорфова раздаљина је често контраинтуитивна. За визуелно различите објекте, можемо добити сличне $h(d)$ вредности. Израчунавање Хауздорфове раздаљине одузима

2 Системи итеративних функција

много времена. Могуће је разматрати друге погодне функције као што је Еуклидско растојање.

Тешко је пронаћи одговарајући скуп IFS трансформација које су добре апроксимације оригиналне слике. Ово је нарочито видљиво у моделирању биљака, где непрекидан процес раста биљака приказ у IFS моделу. Генетски програмерски алгоритми засновани на Еуклидским метрикама су предложени за решавање овог проблема.

Такође је могуће пронаћи колаж ручно. Ово захтева вежбу и тренинг у примећивању само-сличности у сликама. Да би се развио ручни IFS дизајнер, неопходно нам је мало интерактивности у апликацији. Кондензацијским скуповима је неопходна специјална пажња у процесу IFS конструкције. Наредна теорема нам говори да можемо непрекидно да мењамо атрактор IFS-а, подешавајући параметре у трансформацијама.

Теорема: Нека је (X, d) метрички простор. Нека је $f_i: X \rightarrow X$ контраktivно мапирање. Нека је $\varepsilon > 0$ дато. Нека је $\Omega = \{X; f_i, i=1, 2, \dots, n\}$ хиперболичан IFS са контраktivним факторима $0 \leq s_i < 1$. За $i=1, 2, \dots, n$, нека f_i зависи непрекидно од параметра $k \in X$, где је X компактан метрички простор. Онда IFS атрактор $A_k \subset X$ зависи непрекидно од k са Хауздорфовим растојањем $h(d)$.

Мале промене у параметрима који описују сваку IFS трансформацију f_i ће довести до малих промена у одговарајућем атрактору. Нећемо изгубити контролу над атрактором ако мало изменимо почетни модел. Системи итеративних функција су потпуно детерминистички; стога, сваки пут када креирамо дати скуп од IFS параметара добићемо идентичан или статистички сличан (за алгоритам случајних итерација) објекат.

IFS алгоритми

У претходном делу смо описали неколико репрезентација система итеративних функција. Под претпоставком да је задовољена контраktivност, све оне би требале да резултирају константном тачком која се назива атрактор IFS-а. Постоји више метода за исцртавање IFS атрактора. Најпростији приступ је лак за имплементацију, али је често нетачан и неефикасан.

Најпознатија процедура је метода са вероватноћама (алгоритам случајне итерације). Овај алгоритам грубо апроксимира атрактор веома брзо али захтева значајну количину времена да ви се завршио. Барнслеј је развио алгоритам детерминистичке итерације (DIA). DIA метод може бити адаптиран да креира широку разноликост фрактала.

Приказаћемо 2 алгоритма за креирање слика атрактора IFS-а. То су: Детерминистички алгоритам и алгоритам случајних итерација (Random iteration algorithm - RIA). Детерминистички алгоритам је заснован на идеји директног израчунавања низа скупова $\{A_n = F_n(A)\}$. Због једноставности ограничићемо нашу пажњу на хиперболичне IFS у

2 Системи итеративних функција

форми $\{\mathbb{R}^2; f_i, i = 1, 2, \dots, n\}$ где је свако мапирање афина трансформација. Илустроваћемо алгоритам за IFS чији је атрактор троуглао Сьерпинског. Ево примера таквог IFS-а:

$$f_1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$f_2 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 50 \end{bmatrix}$$

$$f_3 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 25 \\ 50 \end{bmatrix}$$

Општи облик ће бити:

$$f_i(x) = f_i \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} e_i \\ g_i \end{bmatrix} = A_i x + t_i$$

Уреднији начин за запис је у облику табеле:

Табела 1

IFS код за троугао Сьерпинског							
f	a	b	c	d	e	g	p
1	0.5	0	0	0.5	1	1	0.33
2	0.5	0	0	0.5	1	50	0.33
3	0.5	0	0	0.5	50	50	0.34

У табели је дат број p_i који одговара f_i за $i=1,2,3$. Ови бројеви представљају вероватноће да се примене трансформације f_i . У општијем случају IFS-а $\{X; f_i, i=1,2,\dots,n\}$, постојало би n таквих бројева $\{p_i : i=1,2,\dots,n\}$ за које важи:

$$p_1 + p_2 + p_3 + \dots + p_n = 1 \text{ и } p_i \geq 0 \text{ за } i=1,2,\dots,n.$$

Ове вероватноће играју важну улогу у изграчувању слика атрактора IFS-а користећи алгоритам случајних итерација, међутим немају никакву улогу у детерминистичком алгоритму. За сада ћемо их користити само као помоћ око израчунавања у конекцији са алгоритмом случајних итерација. У том циљу додељиваћемо им вредности апроксимативно на следећи начин:

$$p_i \approx \frac{|\det A_i|}{\sum_{i=1}^N |A_i|} = \frac{|a_i d_i - b_i c_i|}{\sum_{i=1}^N |a_i d_i - b_i c_i|} \text{ за } i=1,2,\dots,N.$$

Ако је за неко i , $\det A_i = 0$, онда за p_i треба да доделимо мали позитиван број, као нпр 0.001. Друге ситуације треба третирати емпиријски.

2 Системи итеративних функција

Алгоритам (Детерминистички алгоритам): Нека је $\{X; f_1, f_2, \dots, f_n\}$ хиперболички IFS. Изаберемо компактан скуп $A_0 \subset \mathbb{R}^2$, а затим сукцесивно израчунавамо $A_k = F^{\circ k}(A)$ према:

$$A_{n+1} = \bigcup_{j=1}^n f_j(A_n) \text{ за } n = 1, 2, \dots$$

Тако креирамо низ $\{A_n : n = 0, 1, 2, 3, \dots\} \subset H(X)$ за који знамо да конвергира ка атрактору IFS у Хаусдорфовој метрици.

Алгоритам детерминистичке итерације (DIA) потпуно прати дефиницију система итеративних функција као скуп копија самог себе. У детерминистичком алгоритму све трансформације су примењене у свакој итерацији па нам нису неопходне вероватноће.

Илустроваћемо имплементацију алгоритма. Израчунавамо и цртамо сукцесивне скупове A_{n+1} почињајући од почетног скупа A_0 , који у овом случају представља слику листа, користећи IFS код у табели 2. Све наредне слике сам такође генерисао у софтверу IFS kit.

Табела 2

IFS код за дрво 1							
f	a	b	c	d	e	g	p
1	0.050	0.000	0.000	0.400	-0.060	-0.470	0.030
2	-0.050	0.000	0.000	-0.400	-0.060	-0.470	0.030
3	0.030	-0.140	0.000	0.260	-0.160	-0.010	0.010
4	-0.030	0.140	0.000	-0.260	-0.160	-0.010	0.010
5	0.560	0.440	-0.370	0.510	0.300	0.150	0.590
6	0.190	0.070	-0.010	0.150	-0.200	0.280	0.040
7	-0.330	-0.340	-0.330	0.340	-0.540	0.390	0.290



Слика. Прва слика представља почетни скуп (слику листа). Друга представља слику након првог скупа трансформација, затим након другог и последња слика представља слику након петог скупа трансформација.

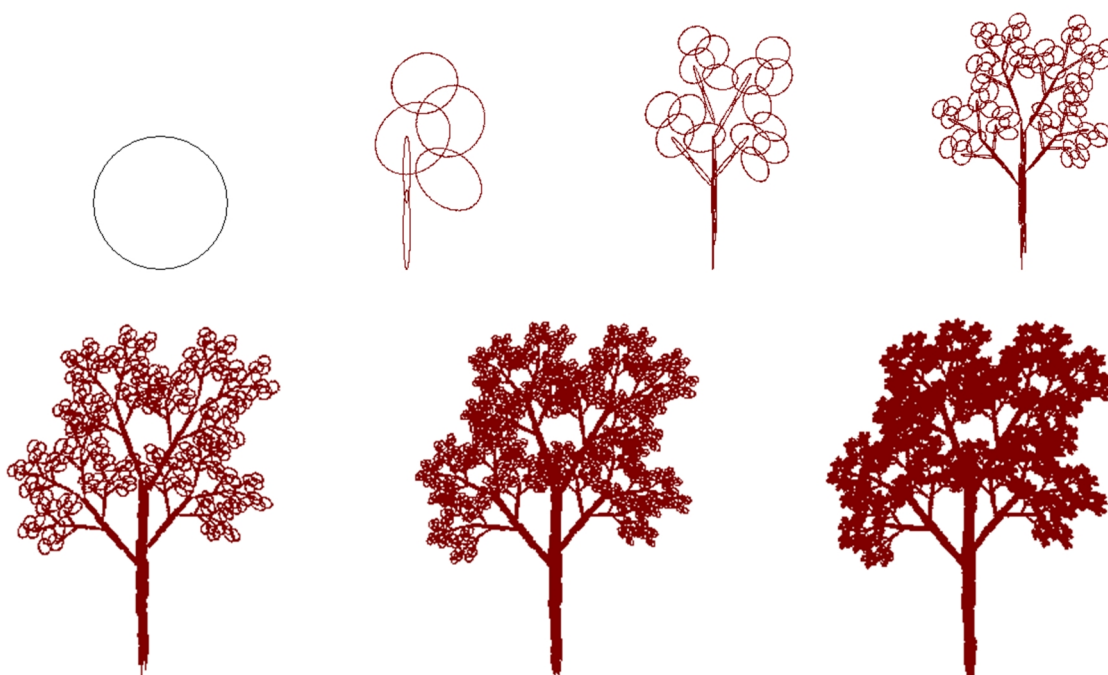
Без обзира коју базу узмемо за започињање итерације, за фиксни IFS облици које генеришемо ће аутоматски да конвергирају ка идентичном фракталу.

2 Системи итеративних функција

Сада ћемо да креирамо нови фрактал трансформишући јединичну кружницу.

Табела 3

IFS код за дрво 2							
f	a	b	c	d	e	g	p
1	0.050	0.000	0.000	0.600	0.000	0.000	0.030
2	0.050	0.000	0.000	-0.500	0.000	1.000	0.020
3	0.459	-0.320	0.385	0.383	0.000	0.600	0.280
4	0.469	-0.153	0.171	0.422	0.000	1.100	0.210
5	0.560	0.440	-0.370	0.510	0.000	0.150	0.590
6	0.433	0.275	-0.250	0.476	0.000	1.000	0.260
7	0.421	0.257	-0.353	0.306	0.000	0.700	0.200



Слика. Приказано је првих шест итерација
детерминистичког алгоритма

Алгоритам (алгоритам случајне итерације): Нека је $\{X; f_1, f_2, \dots, f_n\}$ хиперболички IFS, где је вероватноћа $p_i > 0$ додељена за f_i за $i = 1, 2, \dots, n$, где је $\sum_{i=1}^n p_i = 1$. Изаберемо $x_0 \in X$, а онда бирамо рекурзивно, независно,

$$x_n \in \{f_1(x_{n-1}), f_2(x_{n-1}), \dots, f_n(x_{n-1})\} \text{ за } n = 1, 2, 3, \dots$$

2 Системи итеративних функција

где је вероватноћа догађаја $x_n = f_i(x_{n-1})$ једнака p_i . Према томе, конструишемо низ $\{x_i : i = 0, 1, 2, 3, \dots\} \subset X$.

Ако је $\{X, f_0, f_1, f_2, \dots, f_n\}$ IFS са кондензационом мапом f_0 и повезаним кондензационим скупом $C \subset H(X)$, онда је алгоритам модификован:

(а) додељивање вероватноће $p_0 > 0$ на f_0 , тако да је $\sum_{i=0}^n p_i = 1$;

(б) кад год је $f_0(x_{n-1})$ селектовано за неко n , изаберемо x_n насумице из C . Према томе, у овом случају такође конструишемо низ $\{x_n : n = 0, 1, 2, \dots\}$ од тачака из X .

Низ $\{x_n\}_{n=0}^{\infty}$ конвергира ка атрактору IFS-а.

Сада ћемо илустровати имплементацију алгоритма.

Табела 4

IFS код за папрат (fern leaf)							
f	a	b	c	d	e	g	p
1	0.000	0.000	0.000	0.200	0.000	-0.120	0.010
2	0.845	0.035	-0.035	0.820	0.000	1.600	0.850
3	0.200	-0.310	0.255	0.245	0.000	0.290	0.070
4	-0.150	0.240	0.250	0.200	0.000	0.680	0.070



Слика. Дате су три слике папрати где се слике састоје редом од 1500, 5000 и 50000 тачака

Тачке добијене првом функцијом смо обележили црвеном бојом, другом - зеленом, трећом - жутом и четвртом - плавом.

2 Системи итеративних функција

Табела 5

IFS код							
f	a	b	c	d	e	g	p
1	0.000	0.243	0.000	0.305	0.000	0.000	0.010
2	0.724	0.033	-0.025	0.742	0.206	0.253	0.710
3	0.158	-0.130	0.355	0.367	0.138	0.175	0.140
4	0.338	0.369	0.222	-0.075	0.068	0.068	0.140



Слика. Слика је добијена IFS кодом из табеле 5 где су све тачке зелене боје



Слика. Две модификације папрати креиране алгоритмом случајне итерације

3

Линденмајерови системи

Лепота биљака је привлачила пажњу математичара вековима. Изражене геометријске карактеристике као што су симетрије листова, ротациона симетрија цвећа и спирални аранжмани борових шишарки су студирани опширно.

У овој секцији се бавимо са два фактора која организују структуре биљака и доприносе њиховој лепоти. Први је елеганција и релативна једноставност развојних алгоритама, односно правила која описује развој биљака у времену. Други је самосличност коју је увео Манделброт. У многим процесима раста живих организама, нарочито код биљака, регуларно понављајуће појаве од неких вишећелијских структура су лако уочљиве. Нпр. код листа неки од режњева који су делови листа у напредној фази, имају исти облик као и цео лист у својој ранијој фази. Према томе, самосличност код биљака је резултат развојног процеса.

L-системи су уведени 1968. од стране Линденмајера као теоретски оквир за изучавање развоја простих вишећелијских организама, а затим у примени за истраживање биљака и биљних органа. Након инкорпорације геометријских карактеристика, модели биљака приказани користећи L-системе постају довољно детаљни да омогуће примену компјутерске графике за реалистичну визуелизацију структура биљака и развојних процеса.

Примена L-система за опис биљака је истраживана од стране биолога, и укључује различите методе опште математике. Самосличност доводи у везу структуре биљака са геометријом фрактала. Софтвери за визуелизацију и креирање ових структура спајају науку са уметношћу.

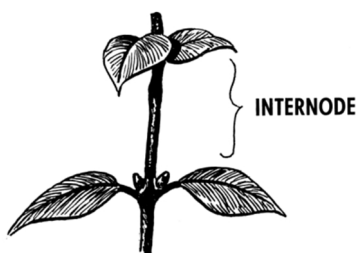
Енергичан развој математичке теорије L-система је праћен њеном применом у моделирању биљака. Наредни развој је следећи:

- Увођење корњачиног алгорита и пречишћавање програмског језика базираног на L-системима, који олакшава спецификацију модела за симулацијске сврхе и промовише коришћење L-система као језик за опис модела,
- Препознавање фракталних карактера у структурама генерисаним L-системима, који их везују са динамичким развојем науке фрактала.
- Увећање интереса у примени компјутерске симулације за разумевање животних процес и структура,

3 Линденмајерови системи

- Повећање разумевања процеса моделирања, који нам дају методологију за конструкцију модела према биолошким обсервацијама и мерењима.

L-системи су оригинално уведени за моделирање развоја простих вишећелијских организама (нпр, алги) у погледу, дељења, раста, и смрти индивидуалних ћелија. Интервал примене L-система се затим увећао на веће биљке и комплексније структуре, нарочито код цвећа, описаних као конфигурације модула у простору. У контексту L-система, термин модул представља директну конструкцију јединицу која се понавља док биљка расте, нпр као internode, врх, цвет, грана. Циљ моделирања на модуларном нивоу је опис конструкције биљке као целине.

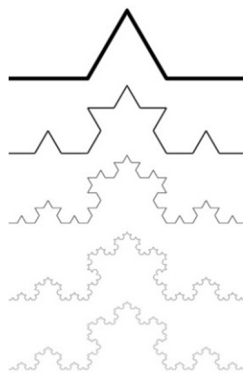


Слика. Значење речи *internode*

Системи преписивања

Централни концепт L-система је преписивање. Преписивање је техника за дефинисање комплексних објеката sukcesивним замењивањем делова простог почетног објекта користећи скуп правила преписивања или израде. Класичан пример графичког објекта дефинисаног у смислу правила преписивања је крива пахуље измишљена 1905. од стране Koch-а. Манделброт излаже ову конструкцију на следећи начин:

Почињемо са две фигуре, иницијатор и генератор. Имамо оријентисану изломљену линију сачињену од n једнаких страна дужине l . Дакле, свака страна конструкције почиње са изломљеном линијом и састоји се од замена сваког равног интервала са копијом генератора, редукованим и премештеним тако да има исте крајње тачке као и у интервалу који је био замењен. На следећој слици је приказана конструкција криве пахуље



Слика. Иницијатор је хоризонтална дуж, а генератор представља прву изломљену линију

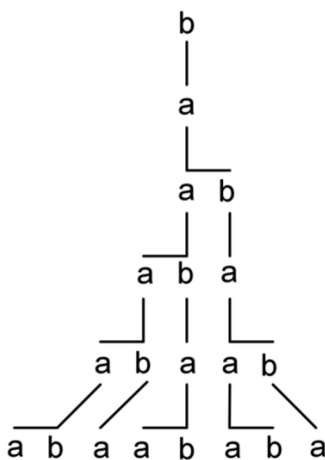
3 Линденмајерови системи

Најобимније истражен и најбоље схваћен систем преписивања ради на стринговима сачињеним од слова.

D0L-системи

Ова секција приказује најједноставнију класу L-система, оних који су детерминистички и без додатака, које зовемо D0L-системима. Дискусија започиње са примером који уводи главну идеју у интуитивним терминима.

Разматрамо стрингове (речи) који се састоје од два слова a и b , који могу да се појављују више пута у стрингу. Свако слово је повезано са правилом преписивања. Правило $a \rightarrow ab$ значи да ће слово a бити замењено стрингом ab , а правило $b \rightarrow a$ значи да ће слово b бити замењено са a . Процес преписивања почиње од карактеристичног стринга који се зове аксиом. Претпоставимо да се он састоји од једног слова b . У првом кораку преписивања аксиома b је замењено са a користећи израду $b \rightarrow a$. У другом кораку a је замењено са ab користећи израду $a \rightarrow ab$. Реч ab се састоји од два слова, оба која су симултано замењена у следећем кораку. Дакле a је замењено са ab , b је замењено са a и стринг aba је резултат. На сличан начин стринг aba производи $abaab$ који даље производи $abaababa$, а затим $abaababaabaab$ и тако даље



Слика. Пример деривације у D0L-систему

Формална дефиниција која описује D0L-системе и њене операције је:

Дефиниција: Нека V представља алфабет, V^* скуп свих речи над V , и V^+ скуп непразних речи над V . 0L-систем је уређена тројка $G = \langle V, \omega, P \rangle$ где је V алфабет система, $\omega \in V^+$ непразна реч која се зове аксиом и $P \subset V \times V^*$ је коначан скуп продукција. Продукција $(a, \chi) \in P$ се записује као $a \rightarrow \chi$. Слово a и реч χ се називају **претходник** и **наследник** ове продукције. Претпостављамо да за било које слово $a \in V$

3 Линденмајерови системи

постоји бар једна реч $\chi \in V^*$ таква да је $a \rightarrow \chi$. Ако ниједна продукција није експлицитно одређена за дати претходник $a \in V$, претпостављамо да продукција идентитета $a \rightarrow a$ припада скупу продукција P .

Дефиниција: *OL-систем* је детерминистички (популарно се зове *DOL-систем*) ако за свако $a \in V$ постоји тачно један $\chi \in V^*$ тако да $a \rightarrow \chi$.

Дефиниција: (деривација) Нека је $\mu = a_1 \dots a_m$ произвољна реч над V . Продукција $p: a \rightarrow \chi$ одговара слову a_i , $1 \leq i \leq m$, ако $a = a_i$. Одговарајућа продукција p може бити примењена на слово a_i , производећи реч χ . Ако слово a_i производи реч χ као резултат примене продукције, то онда обележавамо као $a_i \mapsto \chi$. Реч $\nu = \chi_1 \dots \chi_m \in V^*$ је **директно изведена** (генерисана) од μ што означавамо $\mu \Rightarrow \nu$, ако $a_i \mapsto \chi_i$ за $\forall i = 1, \dots, m$. Реч ν је генерисана са G у деривацији дужине n ако постоји **развојни низ** речи $\mu_0, \mu_1, \dots, \mu_n$ тако да је $\mu_0 = \omega$, $\mu_n = \nu$ и $\mu_0 \Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n$.

Следећи пример даје још једну илустрацију операције DOL-система. Симболи a и b представљају цитолошка стања ћелија (њихову величину и спремност на деобу). Индекси l и r указују на ћелијску поларност, специфицирајући позицију у којој ће ћерка ћелија од врсте a и b бити произведена. Развој је описан у следећем L-систему

$$\begin{aligned} w &: a_r \\ p_1 &: a_r \rightarrow a_l b_r \\ p_2 &: a_l \rightarrow b_l a_r \\ p_3 &: b_r \rightarrow a_r \\ p_4 &: b_l \rightarrow a_l \end{aligned}$$

Започињајући од једне ћелије a_r (аксиома), низ речи је генерисан.

$$\begin{aligned} &a_r \\ &a_l b_r \\ &b_l a_r a_r \\ &a_l a_l b_r a_l b_r \\ &b_l a_r b_l a_r a_r b_l a_r a_r \end{aligned}$$

Алгоритам корњаче

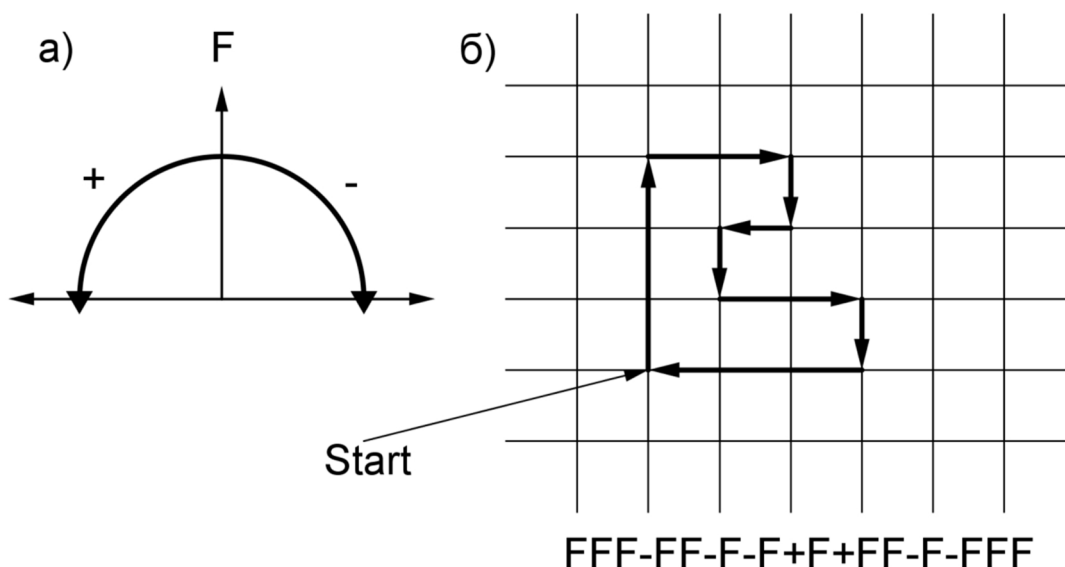
Слова L-систем алфавета су представљена графички као краћи или дужи правоугаоници са заобљеним угловима. Генерисане структуре су једнодимензионални ланци правоугаоника, који одражава низ симбола у одговарајућим стринговима.

3 Линденмајерови системи

Основна идеја корњачиног алгоритма је:

Дефиниција: *Стање корњаче је дефинисано као тројка (x, y, α) , где координате (x, y) представљају корњачину **позицију**, и угао α који се тумачи као смер у коме је корњача окренута. За дату **дужину корака** d и **прирастај угла** δ , корњача може да одговара на команде дате са следећим симболима:*

- F Крећи се напред корак, дужине d . Стање корњаче се мења у (x', y', α) , где је $x' = x + d \cos \alpha$ и $y' = y + d \sin \alpha$. Сегмент линије између (x, y) и (x', y') се црта.
- f Крећи се напред корак дужине d без цртања линије.
- $+$ Скрени лево за угао δ . Следеће стање корњаче је $(x, y, \alpha + \delta)$. Позитивна оријентација углова је у супротном смеру казаљке на сату.
- $-$ Скрени десно за угао δ . Следеће стање корњаче је $(x, y, \alpha - \delta)$.



Слика. а) Интерпретација стринг симбола F , $+$, $-$. б) Интерпретирање стринга. Угао δ је 90° . У старту је корњача окренута ка „горе“.

За дат стринг v , почетно стање корњаче (x_0, y_0, α_0) и фиксирани параметри d и δ , интерпретација од v је фигура (скуп линија) нацртана од стране корњаче као одговор на стринг v (б). Конкретно, овај метод може бити примењен да тумачи стрингове који су генерисани са L-системима.

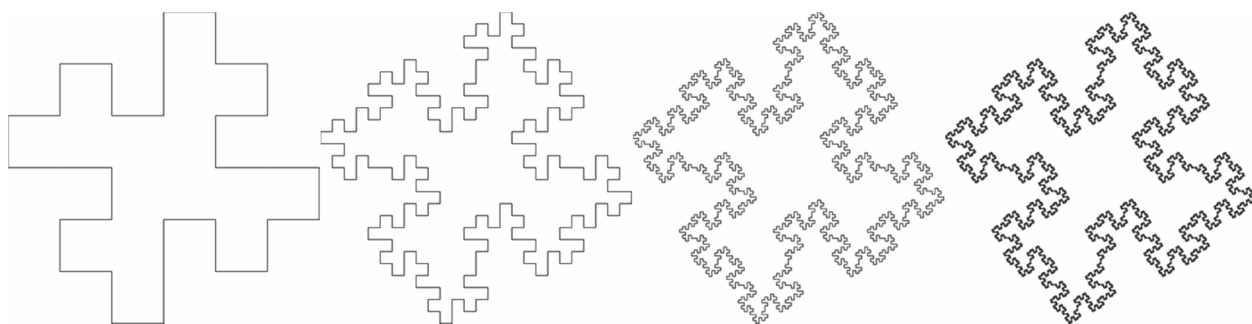
3 Линденмајерови системи

Следећа слика представља квадратно Koch-ово острво. Ове фигуре су добијене интерпретирањем стрингова генерисаних следећим L-системом:

$$\omega: F - F - F - F$$

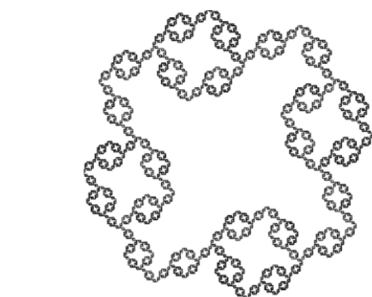
$$p: F \rightarrow F - F + F + FF - F - F + F$$

Слике одговарају стринговима добијеним у извођењу од дужина 0 до 3. Прирастај угла је $\delta = 90^\circ$. Дужина корака d је смањена четири пута за време следећих слика, чинећи растојање између крајњих тачака од полигона наследника једнаким дужини претходниковог сегмента. Све слике у овом делу сам креирао у L-systems Turtle Graphics Renderer-у.



Слика. Генерисање квадратног Кочовог острва. Сlike сам генерисао редом за $n = 0, 1, 2, 3$

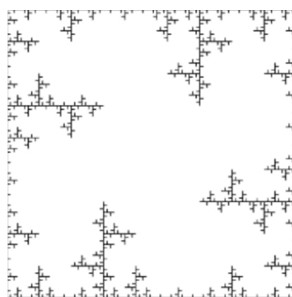
На следећа три примера су приказани записи L-система и слике коју корњача генерише. У првом реду су број итерација и угао, у другом аксиом, а у трећем продукција:



а) $n = 4, \delta = 90^\circ$

$F - F - F - F$

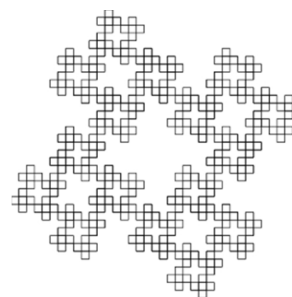
$F \rightarrow FF - F - F - F - F + F$



б) $n = 4, \delta = 90^\circ$

$F - F - F - F$

$F \rightarrow FF - F - F - F - F$

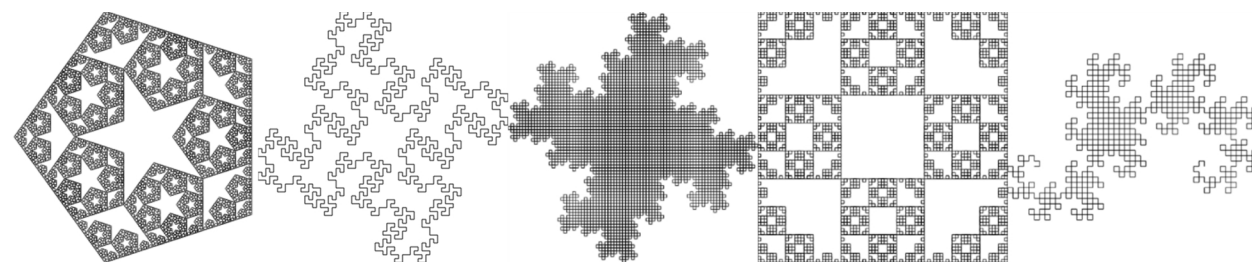


в) $n = 4, \delta = 90^\circ$

$F - F - F - F$

$F \rightarrow F - F + F - F - F$

Приказано је још неколико примера креирано на сличан начин:

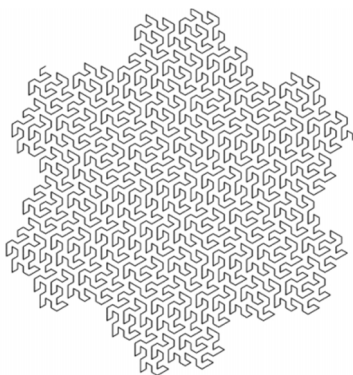


3 Линденмајерови системи

Модификација продукција даје мали увид унутар односа између L-система и фигура које генеришу. Међутим, ми често желимо да конструишемо L-систем који обухвата дату структуру или низ структура које представљају развојни процес. Ово се назива закључни проблем у теорији L-система. Методи које уводимо доле су интуитивнији у природи него модели које смо раније навели. Они користе два начина операције за L-системе са корњачиним алгоритмом, који се називају преписивање ивица и преписивање чворова. У случају преписивања ивица, продукције замењују фигуре за ивице полигона, док у преписивању чворова, продукције раде на теменима полигона. Иако су концепти илустровани користећи апстрактне криве, они се примењују на гранајуће структуре које се појављују код биљака.

Преписивање ивица

Преписивање ивица можемо да посматрамо као екстензију Koch-ове конструкције. Оба, и F_l и F_r симболи представљају ивице креиране „корњачом“ која извршава „крећи се напред“ командом. Продукције смењују F_l и F_r ивице по паровима линија формирајући лева или десна скретања. Многе интересантне криве могу бити добијене претпостављајући две врсте ивица, „леве“ и „десне“.

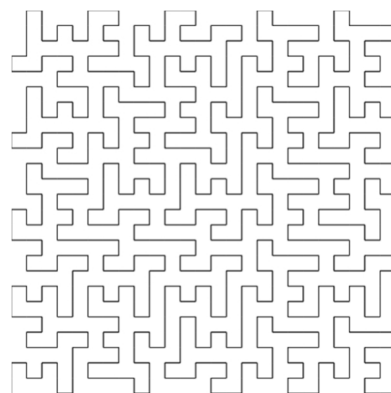


а) $n = 4$, $\delta = 60^\circ$

F_l

$F_l \rightarrow F_l + F_r + +F_r - F_l - -F_l F_l - F_r +$

$F_r \rightarrow -F_l + F_r F_r + +F_r + F_l - -F_l - F_r$



б) $n = 2$, $\delta = 90^\circ$

$-F_r$

$F_l \rightarrow F_l F_l - F_r - F_r + F_l + F_l - F_r - F_r F_l +$

$F_r + F_l F_l F_r - F_l + F_r + F_l F_l +$

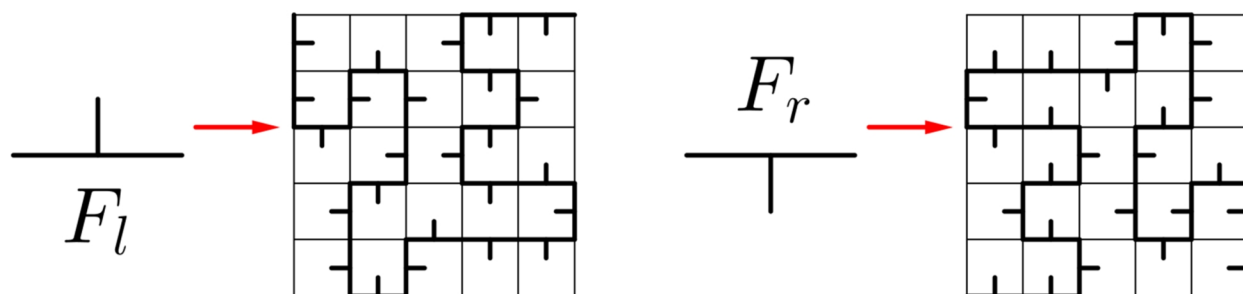
$F_r - F_l F_r - F_r - F_l + F_l + F_r F_r -$

$F_r \rightarrow +F_l F_l - F_r - F_r + F_l + F_l F_r + F_l -$

$F_r F_r - F_l - F_r + F_l F_r F_r - F_l -$

$F_r F_l + F_l + F_r - F_r - F_l + F_l + F_r F_r$

Слика. Примери L-система генерисаних преписивањем ивица



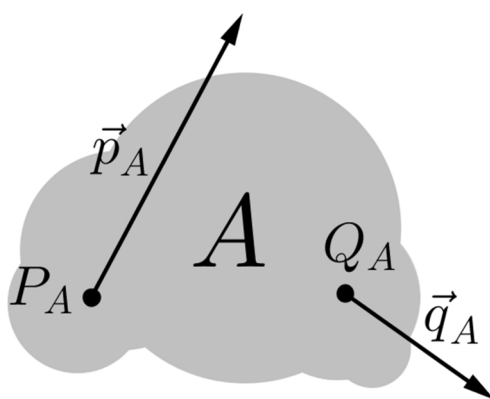
Слика. Рекурзивна конструкција криве на квадратној мрежи методом преписивања ивица.

Преписивање чворова

Идеја преписивања чворова је да заменимо нове полигоне чворовима криве претходника. Како би ово било могуће, корњачин алгоритам је проширен са симболима који представљају произвољне подфигуре. Као што је приказано на слици, свака подфигура A из скупа подфигура A је представљена са:

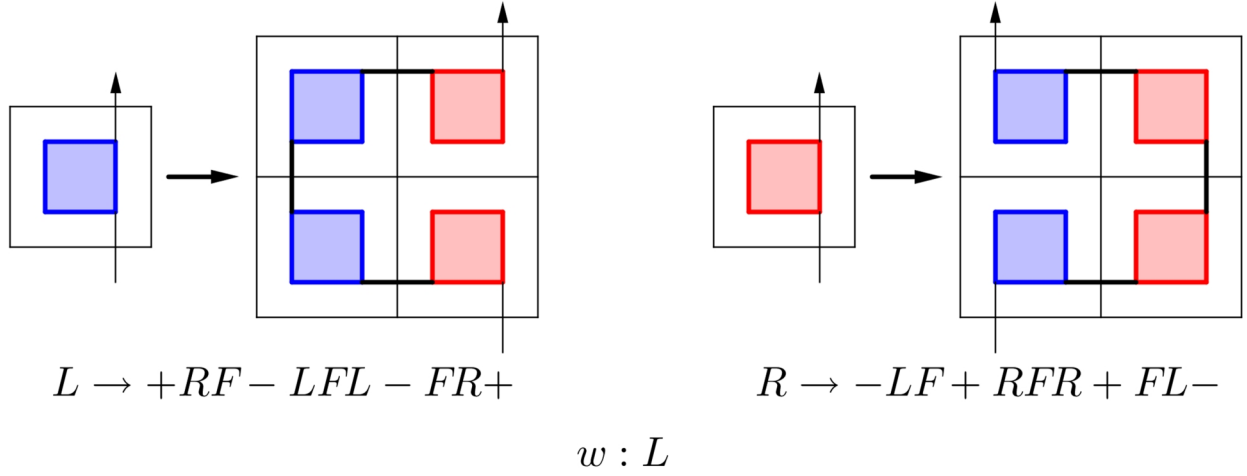
- две контакт тачке, које се називају улазна тачка P_A , и излазна тачка Q_A , и
- два вектора правца, који се називају улазни вектор $\vec{p}_A$ и излазни вектор $\vec{q}_A$.

За време интерпретације стринга v , симбол $A \in A$ укључује одговарајуће подфигуре у слику. У том циљу, A је транслирана и ротирана како би ускладила улазну тачку P_A и правац $\vec{p}_A$ са тренутном позицијом и оријентацијом корњаче. Пошто је постављено A , корњачи је додењена резултујућа излазна позиција Q_A и правац $\vec{q}_A$.



Слика. Опис подфигуре A

3 Линденмајерови системи



Слика. Рекурзивна конструкција Хилбертове криве смењивањем чворова

Нпр, претпоставимо да су контакт тачке и правци подфигура L_n и R_n као на слици. Фигуре L_{n+1} и R_{n+1} добијамо на основу следећих формула:

$$\begin{aligned} L_{n+1} &= +R_n F - L_n F L_n - F R_n + \\ R_{n+1} &= -L_n F + R_n F R_n + F L_n \end{aligned}$$

Претпоставимо да су криве L_0 и R_0 дате. Један начин да израчунамо стринг L_n (или R_n) за $n > 0$ је да генеришемо сукцесивне стрингове рекурзивно, по редоследу смањења вредности од индекса n . Нпр, израчунавање L_2 ће изгледати:

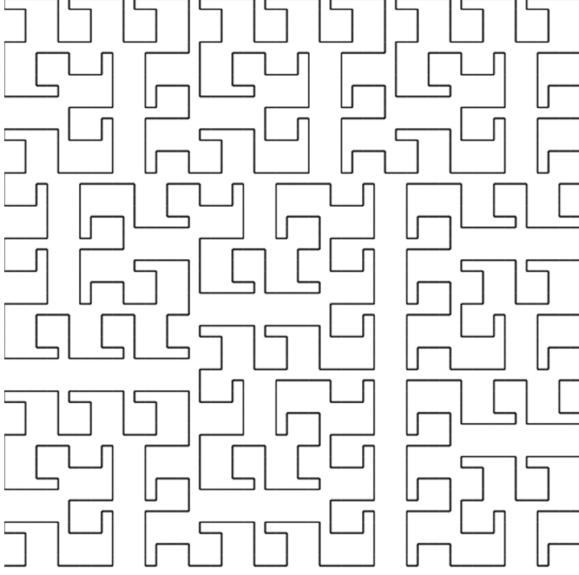
$$\begin{aligned} L_2 &= +R_1 F - L_1 F L_1 - F R_1 + \\ &= +(-L_0 F + R_0 F R_0 + F L_0) F - (+R_0 F - L_0 F L_0 - F R_0) \\ &\quad F(+R_0 F - L_0 F L_0 - F R_0) - F(-L_0 F + R_0 F R_0 + F L_0) + \end{aligned}$$

Према томе, генерисање стринга L_n можемо сматрати за механизам стринг-преписивања, где су симболи са леве стране рекурзивних формула смењени са одговарајућим стринговима на десној страни. Смена поступа на паралелан начин са F , $+$ и $-$ који смењују сами себе. Пошто сви индекси у било ком стрингу имају исту вредност, они могу бити избачени, под претпоставком да је глобалан број деривацијских корака сачуван. Сходно томе, стринг L_n можемо добити у деривацији дужине n користећи следећи L-систем:

$$\begin{aligned} \omega : L \\ p_1 : L &\rightarrow +RF - LFL - FR + \\ p_2 : r &\rightarrow -LF + RFR + FL - \end{aligned}$$

3 Линденмајерови системи

Како бисмо комплетирали дефиницију криве, неопходно је да спецификујемо подфигуре представљене са симболима L и R . У специјалном случају кривих ове подфигуре су редуковане на тачке. Према томе, можемо претпоставити да су симболи L и R обрисани (замењени са празним стринговима на крају деривације). Алтернативно, они могу бити остављени у стринговима и игнорисани од корњаче за време стринг интерпретације. Овај други приступ је конзистентан са претходном дефиницијом корњачиног алгоритма.

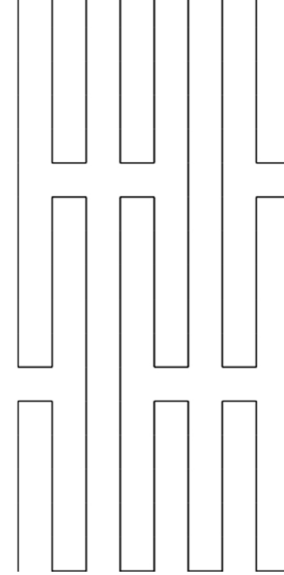


$n = 3, \delta = 90^\circ$

$-L$

$L \rightarrow LF + RFR + FL - F - LFLFL - FRFR +$

$R \rightarrow -LFLF + RFRFR + F + RF - LFL - FR$



$n = 3, \delta = 90^\circ$

L

$L \rightarrow LFRFL - F - RFLRF + F + LRFRFL$

$R \rightarrow RFLFR + F + LFRFL - F - RFLFR$

Однос између преписивања чворова и преписивања ивица

Класе кривих које могу бити генерисане користећи технике преписивања чворова и преписивања ивица нису дисјунктне. Нпр, размотримо L-систем који генерише „dragon curve“ користећи замену ивица:

$$\omega : F_2$$

$$p_1 : F_l \rightarrow F_l + F_r +$$

$$p_2 : F_r \rightarrow -F_l - F_r$$

Претпоставимо привремено да продукција претходника може да садржи више од једног слова; Према томе цела подреч може бити смењена наследником једне продукције (формализација овога се назива псеудо L-систем). Генерисани L-систем може бити записан као:

$$\omega : F_l$$

$$p_1 : Fl \rightarrow Fl + rF +$$

$$p_2 : rF \rightarrow -Fl - rF$$

где се симболи l и r не интерпретирају. Продукција p_1 замењује l са стрингом $l + rF$ – док прво слово F остаје недирнуто. На сличан начин, продукција p_2 замењује слово r са стрингом $-Fl - r$ и оставља F недирнуто. Према томе, L-систем може бити трансформисан у преписивање-чворова на следећи начин :

$$\omega : Fl$$

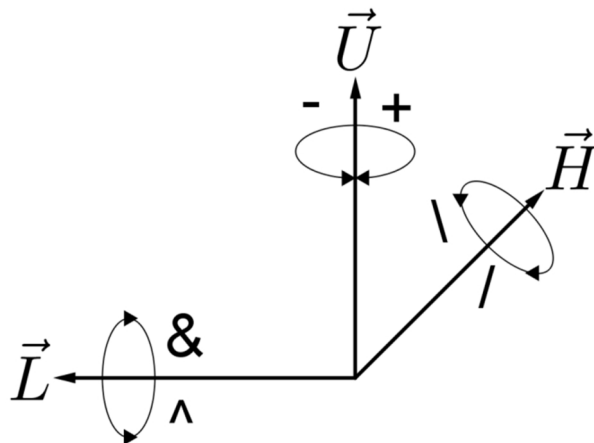
$$p_1 : l \rightarrow l + rF +$$

$$p_2 : r \rightarrow -Fl - r$$

У пракси, избор између преписивања ивица и преписивања чворова је ствар погодности. Ниједан приступ не даје аутоматски и општи метод за конструkcију L-система који дају дате структуре. Међутим, разлика између преписивања ивица и преписивања чворова нам омогућава да лакше разумемо замршеност L-система операције, и у овом смислу асистира у моделирању задатака.

Моделирање у три димензије

Интерпретација L-система може бити проширена на три димензије. Кључни концепт је да представимо тренутну оријентацију корњаче у простору са три вектора $\vec{H}$, $\vec{L}$, $\vec{U}$, што указује на корњачин правац право, правац на лево (left) и правац на горе (up). Ови вектори имају јединичну дужину и нормални су један на други и задовољавају једначину $\vec{H} \times \vec{L} = \vec{U}$.



Слика. Контролисање корњаче у три димензије са $-$, $+$, $\&$, $\wedge$, $\backslash$ и $/$

Ротације корњаче се онда могу изразити са једначинама:

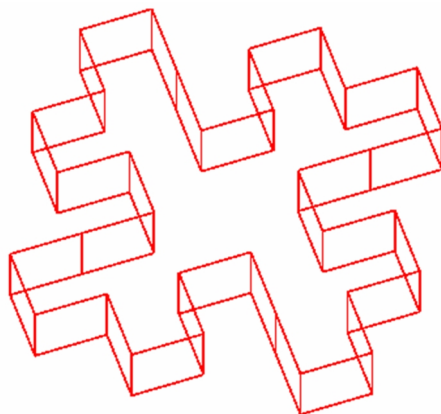
3 Линденмајерови системи

$$\begin{bmatrix} \vec{H}' & \vec{L}' & \vec{U}' \end{bmatrix} = \begin{bmatrix} \vec{H} & \vec{L} & \vec{U} \end{bmatrix} R,$$

где је R 3×3 ротациона матрица. Конкретно, ротације по углу α око $\vec{U}$, $\vec{L}$ и $\vec{H}$ су представљене са матрицама

$$R_U(\alpha) = \begin{pmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad R_L(\alpha) = \begin{pmatrix} \cos \alpha & 0 & -\sin \alpha \\ 0 & 1 & 0 \\ \sin \alpha & 0 & \cos \alpha \end{pmatrix} \quad R_H(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{pmatrix}$$

Приказана је квадратна Кочова крива креирана са следећим L-системом у три димензије. Пример сам креирао у софтверу L-studio:



$$\begin{aligned} n &= 2, \delta = 90^\circ \\ \omega &: \& - XS - XS - XS - XS \\ p_1 &: X \rightarrow XS + XS - XS - XSXS + XS + XS - X \\ p_2 &: S \rightarrow F \wedge F \wedge F \wedge F \wedge F \end{aligned}$$

Следећи симболи контролишу корњачину оријентацију у простору:

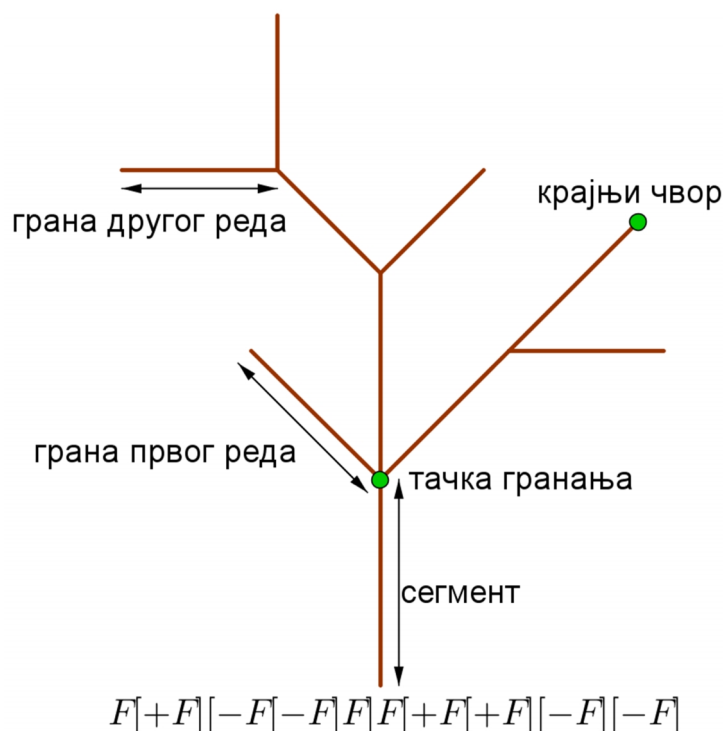
- + Скени лево за угао δ , користећи ротациону матрицу $R_U(\delta)$
- Скени десно за угао δ , користећи ротациону матрицу $R_U(-\delta)$
- & Нагни се доле за угао δ , користећи ротациону матрицу $R_L(\delta)$
- ^ Нагни се горе за угао δ , користећи ротациону матрицу $R_L(-\delta)$
- \ „Котрљај“ се лево за угао δ , користећи ротациону матрицу $R_H(\delta)$
- / „Котрљај“ се десно за угао δ , користећи ротациону матрицу $R_H(-\delta)$
- | Окрените се, користећи ротациону матрицу $R_U(180^\circ)$

Рачвасте структуре

На основу правила приказаних досад, тумаче се слова стринга као низ линијских сегмената. У зависности од дужине сегмента и углова између њих, резултујућа линија је самопресецајућа или није, може бити више или мање замршена, може да има неке сегменте нацртане неколико пута а неке да учини невидљивим, али увек остаје само као једна линија. Међутим, међу биљкама доминирају рачвасте структуре; па су према томе математички опис облика попут дрвета и методе за њихово генерисање неопходне у сврхе моделирања.

Дрво има ивице које су означене и усмерене. Низ ивица формира стазе од различитог чвора, који се назива база до крајњих чворова. У биолошком контексту, ове ивице се односе на сегменте гране. Сегмент којег прати бар још један сегмент у некој стази се назива *internode*. Крајњи чвор (без наредних ивица) се назива врх.

Грана (стабло) који почиње у корену целе биљке има ред нула. Грана која почиње као бочни сегмент гране-родитеља n -тог реда има ред $n+1$. Ред гране је једнак реду његовог најмањег реда или главне гране (стабла).



0L-системи са заградама

Дефиниција 0L-система не спецификује структуру података за представљање дрвећа. Дрвеће може бити представљено користећи стрингове са заградама. Слична одлика може бити примећена у Кочовој конструкцији, која може бити имплементирана или преписивањем ивица и полигона или њихово представљање стринга. Екстензија

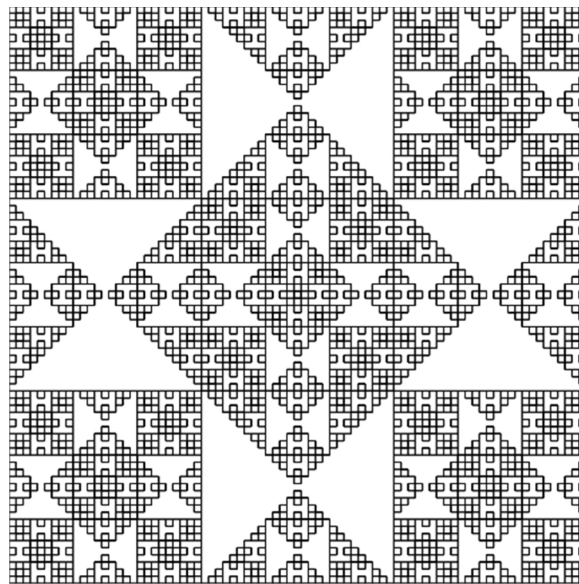
3 Линденмајерови системи

корњачиног алгоритма са заградама и операција L-система са заградама је описана доле.

Два нова симбола су уведена да би се ограничила грана. Они су интерпретирани на следећи начин:

- [Сачувај тренутно стање корњаче, укључујући позицију, правац и боју.
-] Поврати стање корњаче тачно тамо где је била када је претходна команда [процесуирана. Број команди [и] треба да буде једнак.

Деривације у 0L-системима са заградама поступају као и 0L-системи без заграда. Заграде смењују саме себе. Приказан је пример L-система са заградама. Све примере у овом делу сам креирао у L-systems Turtle Graphics Renderer-у.



$$n = 4, \delta = 90^\circ$$

$$F - F - F - F$$

$$F \rightarrow F[F] - F + F[- - F] + F - F$$

Примери дводимензионалних рачвастих структура генерисаних са 0L-системима са заградама се приказани на слици 1.



3 Линденмајерови системи

$$n = 5, \delta = 25.7^\circ$$

F

$$F \rightarrow F[+F]F[-F]F$$

$$n = 7, \delta = 25.7^\circ$$

X

$$X \rightarrow F[+X][X]FX$$

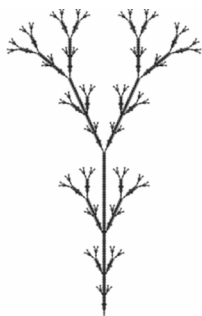
$$F \rightarrow FF$$

$$n = 4, \delta = 22.5^\circ$$

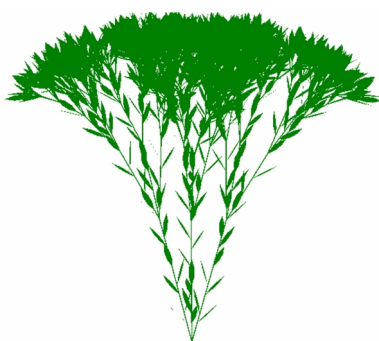
F

$$F \rightarrow FF - [-F + F + F] + [+F - F - F]$$

Приказана су још четири примера креирана на сличан начин:



Следећи пример је пример тродимензионалне структуре налик жбуну генерисан са L-системом са заградама. Примере сам креирао у софтверу L-studio. Продукција p_1 креира три нове гране од врха старе гране. Грана се састоји од ивице F која формира почетни унутрашњи чвор, листа L и врха A (који ће затим креирати три нове гране). Продукције p_2 и p_3 спецификују раст Internode-а. У следећем деривацијском кораку унутрашњи чвор постаје дужи и стиче нове листове. Продукција p_4 спецификује лист као попуњен полигон са 6 ивица. Његова граница је формирана од ивица f ограђених у заградама $\{ , \text{ и } \}$. Символи $!$ и $'$ се користе да умањују дијаметар сегмената и увећавају тренутан индекс боја (мењају боју), респективно.



$$n = 7, \delta = 22.5^\circ$$

$\omega : A$

$$p_1 : A \rightarrow [\&FL!A] [\&FL!A] [\&FL!A]$$

$$p_2 : F \rightarrow S [\&FL!A]$$

$$p_3 : S \rightarrow FL$$

$$p_4 : L \rightarrow [\&FL!A] [\&FL!A] [\&FL!A]$$

$$n = 5, \delta = 18^\circ$$

$\omega : P$

$$p_1 : P \rightarrow I + [P + O] - - [- - E] I [+ + E]$$

$$- [P O] + + P O$$

$$p_2 : I \rightarrow FS [\&FL!A] [\&FL!A] FS$$

$$p_3 : S \rightarrow SFS$$

$$\begin{aligned}
 p_4 : E &\rightarrow ['\{+f - ff - f + | +f - ff - f\}] \\
 p_5 : O &\rightarrow [\&\&\&p' w//// w//// w//// w//// w] \\
 p_6 : p &\rightarrow FF \\
 p_7 : w &\rightarrow [\wedge F][\{\&\&\&-f + f | -f + f\}]
 \end{aligned}$$



Слика. Претходни пример са испрограмираним бојама и осветљењем. Слика је преузета из књиге “The Algorithmic beauty of plants”

Стохастички L-системи

Све биљке генерисане истим детерминистичким L-системима су идентичне. Покушај да их комбинујемо у исту слику би произвео упадљиву вештачку правилност. Како бисмо спречили овај ефекат, неопходно је да уведемо варијације које ће сачувати опште аспекте биљке али ће јој модификовати детаље.

Варијација може бити постигнута увођењем случајности у корњачин алгоритам, у L-систем, или у оба. Увођењем случајности само у интерпретацију има ограничине ефекат. Док су геометријски аспекти биљке – као што су дужине стабљика и углови грана – модификовани, топологија остаје иста. У супротном, примена стохастике продукција може имати ефекат и на топологију и на геометрију биљке.

Дефиниција: Стохастички 0L-систем је уређена четворка $G_\pi = \langle V, \omega, P, \pi \rangle$. Алфабет V , аксиом ω и скуп продукција P су дефинисани као и у 0L-систему. Функција $\pi : P \rightarrow (0,1]$, се назива расподела вероватноће и мапира скуп продукција у скуп расподела вероватноћа. Претпоставља се да је за свако слово $a \in V$, сума вероватноћа свих продукција са претходником a једнака јединици.

Зваћемо деривацију $\mu \Rightarrow \nu$ стохастичка деривација у G_π ако је за сваки случај слова a у речи μ вероватноћа примењивања продукције p са претходником a једнака $\pi(p)$.

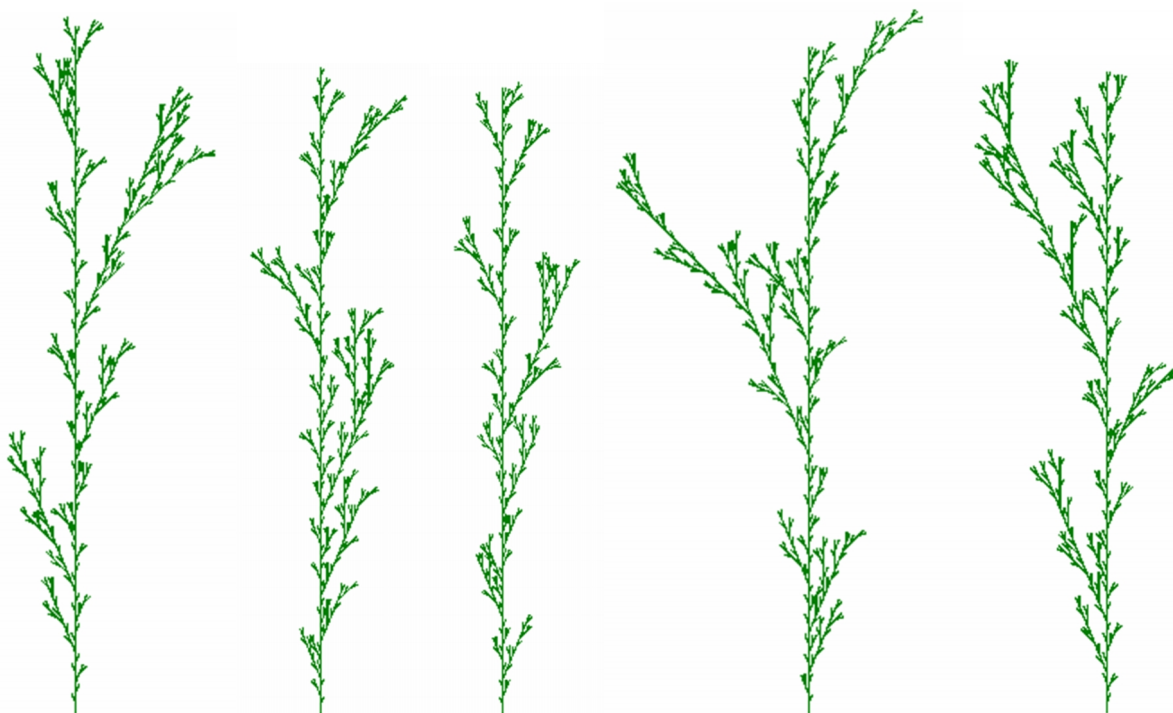
3 Линденмајерови системи

Према томе, различите продукције са истим претходником могу бити примењене на различите случајеве истог слова у деривацијском кораку.

Пример стохастичког L-система:

$$\begin{aligned}\omega &: F \\ p_1 &: F \xrightarrow{0.33} F[+F]F[-F]F \\ p_2 &: F \xrightarrow{0.33} F[+F]F \\ p_3 &: F \xrightarrow{0.34} F[-F]F\end{aligned}$$

Продукцијске вероватноће су поређане изнад деривацијског симбола $\rightarrow$. Свака продукција може бити изабрана са апроксимативно истом вероватноћом од $\frac{1}{3}$. Примери гранајуће структуре генерисане овим L-системом са деривацијама дужине 5 су приказани на следећој слици.



Слика. Стохастичке гранајуће структуре. Свих 5 слика је генерисано истим L-системом

Суштина ове модификације је да се замени оригинална продукција p_3 са следеће три продукције:

$$p_3' : segment \xrightarrow{0.33} segment[//\&\&list][//^{\wedge}list]Fsegment$$

3 Линденмајерови системи

$$p_3'' : \overset{0.33}{segment} \rightarrow segment F segment$$

$$p_3''' : \overset{0.33}{segment} \rightarrow segment$$

Према томе, у сваком кораку деривације, сегмент стабљике *segment* може или да расте и производи нове листове (продукција p_3'), да расте без производње нових листова (продукција p_3''), или да не расте уопште (продукција p_3'''). Резултирајуће поље изгледа да се састоји од разноврсних узорака исте врсте биљака. Ако би се исти L-систем употребио опет добили бисмо варијацију ове слике.

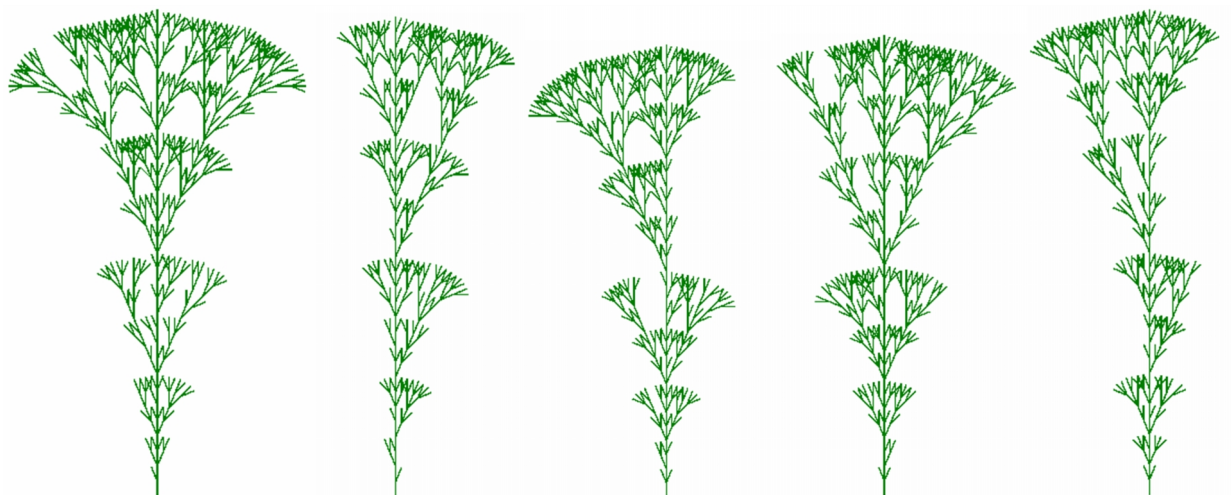
Благом модификацијом претходног примера добијамо следећи L-система који се поприлично разликује од претходног:

$$\omega : F$$

$$p_1 : F \xrightarrow{0.50} F[+F][-F]F$$

$$p_2 : F \xrightarrow{0.30} F[-F]F$$

$$p_3 : F \xrightarrow{0.20} F[+F]F$$



Слика. 5 слика генерисаних истим L-системом

Контекст-сензитивни L-системи

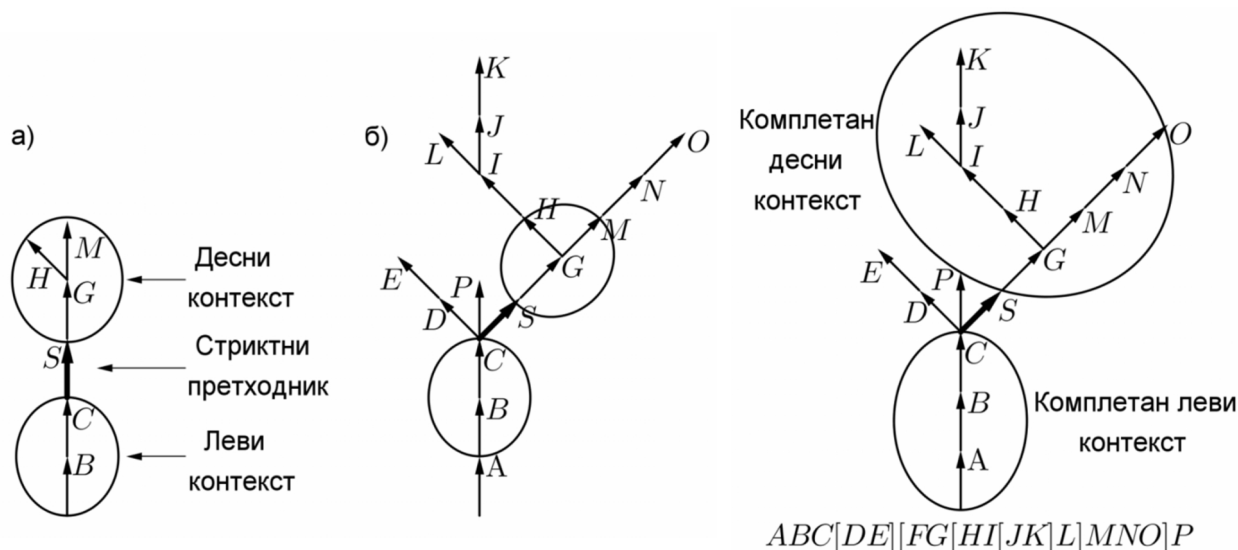
Производње у 0L-систему су без контекста, односно примењиве без обзира на контекст у којем се претходник појављује. Међутим примена производње може такође и да зависи од претходниковог контекста. Овај ефекат је користан у симулирању односа између делова биљака, због примера протока хранљивих састојака и хормона. Различите врсте контекст-сензитивних екстензија од L-система је било предложено и изучавано у прошлости. 2L-системи користе производње у форми $a_l < a > a_r \rightarrow \chi$ где слово a (назива се стриктни

3 Линденмајерови системи

претходник) може да произведе реч χ ако и само ако му је претходник a_l и следи му a_r . Према томе, слова a_l и a_r формирају леви и десни контекст од a у овој продукцији. Продукције у 1L-системима имају само једностране контексте; Сходно томе, оне су или у форми $a_l < a \rightarrow \chi$ или $a > a_r \rightarrow \chi$. 0L-системи, 1L-системи и 2L-системи припадају широкој класи IL-система, који се још називају и (k, l) -системи. У (k, l) -систему, леви контекст је реч дужине k , а десни контекст је реч дужине l .

Да бисмо одржали спецификације L-система малим, уобичајен појам IL-система је овде модификован дозвољавајући продукцијама са различитим дужинама контекста да коегзистирају унутар једног (истог) система. Штавише, претпоставља се да контекст-сензитивне продукције имају предност над продукцијама без контекста са истим стриктним претходником. Сходно томе, ако се на дато слово примењују и продукције без контекста и контекст-сензитивне продукције, требало би изабрати контекст-сензитивну.

Ако се ниједна продукција не примењује, ово слово се замењује самим собом као што је претходно претпостављено за 0L-системе



Слика. Претходник од контекст сензитивне продукције дрвета а) одговара ивици S у дрвету б)

Слика. Стринг са заградама и одговарајућа структура гранања. Комплетни контексти за слово S су означени

Следећи прости 1L-систем користи контекст да симулира сигнал размножавања кроз стринг од симбола:

$$\begin{aligned} \omega &: baaaaaaaaa \\ p_1 &: b < a \rightarrow b \\ p_2 &: b \rightarrow a \end{aligned}$$

3 Линденмајерови системи

Првих пар речи генерисаних са овим L-системом су дати доле:

*b*aaaaaaaa
a**b**aaaaaaaa
aa**b**aaaaaaaa
aaa**b**aaaaaa
...

Слово *b* се помера од леве до десне стране стринга.

Контекст-сензитивна екстензија L-система дрвета захтева комшијске ивице од замењене ивице да буду тестиране за одговарање контексту. Претходник контекст-сензитивне продукције *p* се састоји од 3 компоненте: стазе *l* која формира леви контекст, ивице *S* која се назива стриктни претходник, дрво *r* које представља десни контекст. Асиметрија између левог и десног контекста рефлектује чињеницу да постоји само једна стаза од корена дрвета до дате ивице, док може постојати више стаза од ове ивице до различитих крајњих чворова. Продукција *p* одговара датом случају ивице *S* у дрвету *T* ако је *l* стаза у *T* која се завршава у почетном чвору *S*. Продукција онда може бити примењена смењивањем *S* са осовинским дрветом спецификованим као продукцијски наследник.

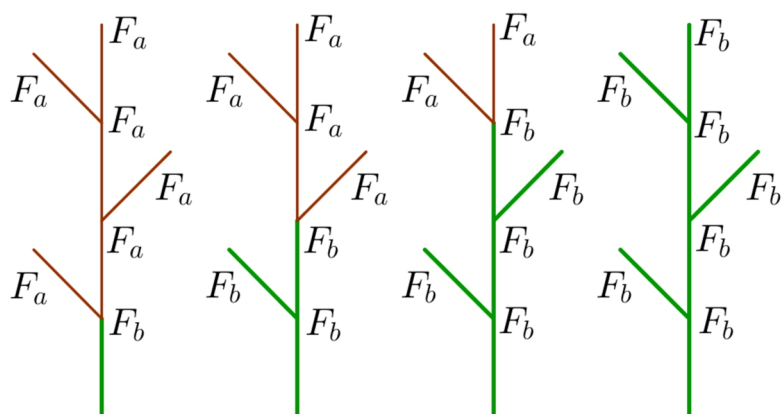
Увођење контекста у L-системе са заградама је много теже неко код L-система без заграда, јер представљање дрвећа са стринговима са заградама не чува сегментов „комшилук“. Сходно томе, одговарајућа контекст процедура можда треба да прескочи преко симбола који представљају гране или делове грана. Нпр, претходна слика указује да продукција са претходником $BC < S > G[H]M$ може бити примењена на симбол *S* у стрингу $ABC[DE][SG[HI[JK]L]MNO]$, који укључује прескакање преко симбола $[DE]$ у претраживању за леви контекст, и $I[JK]L$ у претраживању за десни контекст.

У формализму L-система са заградама, леви контекст може да се користи за симулацију контролних сигнала који се шире од корена или базичног лишћа до врха моделирања биљке, док десни контекст представља сигнале који се шире од врха до корена. Нпр. следећи 1L-систем симулира ширење сигнала од корена до врха у гранајућој сгруктури која не расте:

$$\begin{aligned} \#ignore : + - \\ \omega : F_b[+F_a]F_a[-F_a]F_a[+F_a]F_a \\ p_1 : F_b < F_a \rightarrow F_b \end{aligned}$$

Симбол F_b представља сегмент који је већ досегнут сигналом, док F_a представља сегмент који још увек није досегнут. *#ignore* назначује да би требало геометријске симболе + и – сматрати непостојећим за време укапања контекста. Сlike које представљају узастопне нивое ширења сигнала (одговарајући узастопним речима генерисаним са L-системом који разматрамо) су приказане на следећој слици:

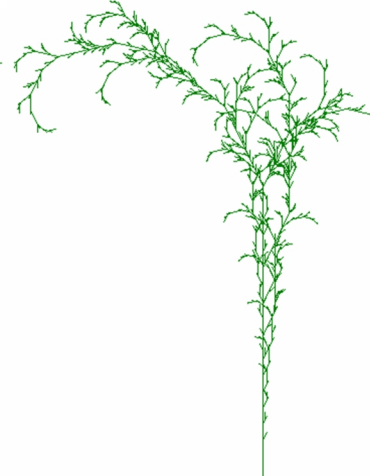
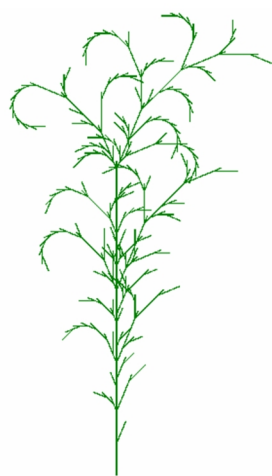
3 Линденмајерови системи



Слика. Ширење сигнала у гранајућој структури

Операција контекст-сензитивних L-система се даље обрађује користећи примере добијене од Hogeweg-а и Hesper-а. Примере сам креирао у софтверу L-studio.

$n = 30, \delta = 22.5^\circ$	$n = 30, \delta = 22.5^\circ$	$n = 24, \delta = 25.75^\circ$	$n = 26, \delta = 22.5^\circ$
#ignore: + - F	#ignore: + - F	#ignore: + - F	#ignore: + - F
F1F1F1	F1F1F1	F0F1F1	F1F1F1
$0 < 0 > 0 \rightarrow 0$	$0 < 0 > 0 \rightarrow 1$	$0 < 0 > 0 \rightarrow 1$	$0 < 0 > 0 \rightarrow 0$
$0 < 0 > 1 \rightarrow 1[+F1F]$	$0 < 0 > 1 \rightarrow 1[-F1F1]$	$0 < 0 > 1 \rightarrow 0$	$0 < 0 > 1 \rightarrow 1[-F1F1]$
$0 < 1 > 0 \rightarrow 1$	$0 < 1 > 0 \rightarrow 1$	$0 < 1 > 0 \rightarrow 0$	$0 < 1 > 0 \rightarrow 1$
$0 < 1 > 1 \rightarrow 1$	$0 < 1 > 1 \rightarrow 1$	$0 < 1 > 1 \rightarrow 1F1$	$0 < 1 > 1 \rightarrow 1$
$1 < 0 > 0 \rightarrow 0$	$1 < 0 > 0 \rightarrow 0$	$1 < 0 > 0 \rightarrow 1$	$1 < 0 > 0 \rightarrow 0$
$1 < 0 > 1 \rightarrow 1F1$	$1 < 0 > 1 \rightarrow 1F1$	$1 < 0 > 1 \rightarrow 1[+F1F1]$	$1 < 0 > 1 \rightarrow 1F1$
$1 < 1 > 0 \rightarrow 0$	$1 < 1 > 0 \rightarrow 1$	$1 < 1 > 0 \rightarrow 1$	$1 < 1 > 0 \rightarrow 1$
$1 < 1 > 1 \rightarrow 0$	$1 < 1 > 1 \rightarrow 0$	$1 < 1 > 1 \rightarrow 0$	$1 < 1 > 1 \rightarrow 0$
$* < + > * \rightarrow -$	$* < + > * \rightarrow -$	$* < + > * \rightarrow -$	$* < + > * \rightarrow -$
$* < - > * \rightarrow +$	$* < - > * \rightarrow +$	$* < - > * \rightarrow +$	$* < - > * \rightarrow +$



Параметарски L-системи

Иако L-системи са корњачиним алгоритмом омогућавају генерисање различитих интересантних објеката, од апстрактних фрактала до структуре које се гранају попут биљака, њихова моћ моделирања је веома ограничена.

Линденмајер је предложио да се нумерички параметри придруже симболима L-система. тако су осмишљени параметарски L-системи који су формулисани од стране Prusinkiewicz-a и Hanan-a.

Параметарски L-системи функционишу над параметарским речима, који су стрингови модула који се састоје од слова са придруженим параметрима. Слова припадају алфабету V , и параметри припадају скупу реалних бројева $\mathbb{R}$. Модул са словом $A \in V$ и параметрима $a_1, a_2, \dots, a_n \in \mathbb{R}$ означавамо са $A(a_1, a_2, \dots, a_n)$. Сваки модул припада скупу $M = V \times \mathbb{R}^*$, где је $\mathbb{R}^*$ скуп свих коначних низова параметара. Скуп свих стрингова модула и скуп свих непразних стрингова означавамо са $M^* = (V \times \mathbb{R}^*)^*$ и $M^+ = (V \times \mathbb{R}^*)^+$.

Актуелни параметри који се појављују у речима одговарају формалним параметрима коришћеним у спецификацији продукција L-система. Ако је Σ скуп формалних параметара, онда $C(\Sigma)$ означава логички израз са параметрима из Σ , и $E(\Sigma)$ је аритметички израз са параметрима из истог скупа. Обе врсте израза се састоје од формалних параметара и нумеричких константи, комбинованих користећи аритметичке операторе $+$, $-$, $*$, $/$; оператора степеновања $^{\wedge}$, релација $>$, $<$, $=$; логичких оператора $!$, $\&$, $|$ (негација, и, или); и заграда $(,)$. Стандардна правила за конструкцију синтактички правилних израза и за правила предности оператора су размотрена.

Код релационих и логичких израза нула је нетачно, а јединица тачно. За логичку изјаву спецификовану као празан стринг сматрамо да има вредност један. Скупови свих коректно конструисаних логичких и аритметичких израза са параметрима из Σ се ознавају са $C(\Sigma)$ и $E(\Sigma)$.

Дефиниција: *Параметарски 0L-систем се дефинише као уређена четворка $G = \langle V, \Sigma, \omega, P \rangle$, где је*

- V алфабет система,
- Σ скуп формалних параметара
- $\omega \in (V \times \mathbb{R}^*)^+$ непразна параметарска реч која се назива аксиом
- $P \subset (V \times \mathbb{R}^*) \times C(\Sigma) \times (V \times E(\Sigma))^*$ је коначан скуп продукција.

3 Линденмајерови системи

Симболи : и $\rightarrow$ се користе да одвоје три компоненте у продукцији: претходник, услов и наследник. Нпр, продукција са претходником $A(t)$, условом $t > 5$ и наследником $B(t+1)CD(t \wedge 0.5, t-2)$ се пише као:

$$A(t) : t > 5 \rightarrow B(t+1)CD(t \wedge 0.5, t-2) \quad (1)$$

Дефиниција. Продукција p_i одговара модулу у параметарској речи ако су следећи услови задовољени:

- Слово у модулу и слово у продукцији претходника је исто,
- Број актуелних параметара у модулу је једнак броју формалних параметара у продукцији претходника, и
- Услов израчунава тачно ако су вредности актуелног параметра замењене формалним параметрима у продукцији.

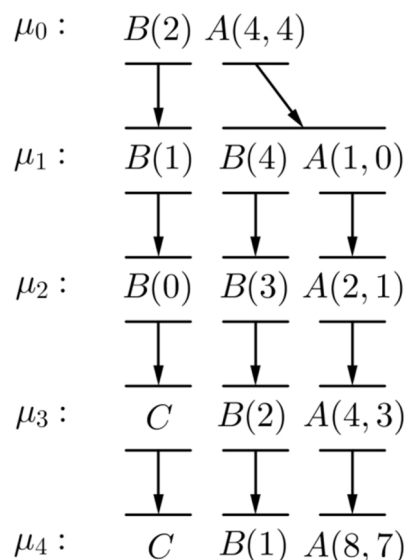
Одговарајућа продукција може бити примењена на модул, креирајући стринг модула спецификованих са продукцијом наследника. Вредности актуелних параметара су смењене формалним параметрима према њиховој позицији. Нпр, продукција (1) одговара модулу $A(9)$, пошто је слово A у модулу исто као и у продукцији претходника, постоји један актуелни параметар у модулу $A(9)$ и један формалан параметар у претходнику $A(t)$, и логички израз $t > 5$ је тачан за $t=9$. Резултат примене ове продукције је параметарска реч $B(10)CD(3, 7)$.

Ако модул a производи параметарску реч χ као резултат примене продукције у L-систему G , пишемо $a \mapsto \chi$. За дату параметарску реч $\mu = a_1 a_2 \dots a_m$, кажемо да је реч $\nu = \chi_1 \chi_2 \dots \chi_m$ директно изведена од речи μ (генерисана од μ) и пишемо $\mu \Rightarrow \nu$ акко је $a_i \mapsto \chi_i$ за свако $i = 1, 2, \dots, m$. Параметарска реч ν је генерисана са G у деривацији дужине n ако постоји низ речи $\mu_0, \mu_1, \dots, \mu_n$ тако да је $\mu_0 = \omega$, $\mu_n = \nu$, $\Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n$.

Пример параметарског L-система:

$$\begin{aligned} \omega &: B(2)A(4, 4) \\ p_1 &: A(x, y) : y \leq 3 \rightarrow A(x*2, x+y) \\ p_2 &: A(x, y) : y > 3 \rightarrow B(x)A(x/y, 0) \\ p_3 &: B(x) : x < 1 \rightarrow C \\ p_4 &: B(x) : x \geq 1 \rightarrow B(x-1) \end{aligned} \quad (2)$$

Као и у случају непараметарских L-система, претпоставља се да модул замењује сам себе ако није пронађена ниједна одговарајућа продукција у скупу P . Речи добијене у првих неколико деривацијских корака су дате на следећој слици:



Слика. Почетни низ стрингова генерисан са параметарским L-системом спецификованим у једначини (2)

Продукције у параметарским OL-системима су без контекста односно примењиве без обзира на контекст у којем се претходник појављује. Контекст-сензитивне екстензије су неопходне за моделирање размене информација између суседних модула. У параметарском случају, сваки компонент продукцијског претходника (леви контекст, стриктни претходник и десни контекст) је параметарска реч са словима из алфавета V и формалним параметрима из скупа Σ . Било који формални параметри могу да се појаве у услову и продукционом наследнику.

Пример контекст-сензитивне продукције је дат доле:

$$A(x) < B(y) > C(z) : x + y + z > 10 \rightarrow E((x + y)/2)F((y + z)/2)$$

Он може бити примењен на модул $B(5)$ који се појављује у параметарској речи

$$\dots A(4)B(5)C(6)\dots \quad (3)$$

Пошто је низ слова A , B , C у продукцији и у параметарској речи (3) исти, бројеви од формалних параметара и актуелних параметара се подударају, и услов $4 + 5 + 6 > 10$ је тачан. Као резултат продукцијске примене, модул $B(5)$ ће бити замењен са паром модула $E(4.5)F(5.5)$. Природно, модули $A(4)$ и $C(6)$ ће бити замењени са другим продукцијама у истом деривацијском кораку.

Интерпретација параметарских речи

Ако су један или више параметара удружени са симболима које интерпретирамо, вредност првог параметра контролише корњачино стање. Ако ниједан параметар не прати симбол, уобичајена вредност спецификована ван L-система се користи у непараметарском случају. Основни скуп симбола погођених уводом параметара је дат доле

$F(a)$ Крећи се напред кораком дужине $a > 0$. Позиција корњаче се помера на (x', y', z') где је $x' = x + a\vec{H}_x$, $y' = y + a\vec{H}_y$ и $z' = z + a\vec{H}_z$. Лينيјски сегмент је нацртан између (x, y, z) и (x', y', z') .

$f(a)$ Крећи се напред за корак дужине a без цртања линије.

$+(a)$ Ротирај се око $\vec{U}$ за угао од a степени. Ако је a позитивно, онда се корњача окреће на лево, а ако је a негативно, онда се окреће на десно.

$\&(a)$ Ротирај се око $\vec{L}$ за угао од a степени. Ако је a позитивно, корњача се нагиње на доле, а ако је a негативно, корњача се нагиње на горе.

$/(a)$ Ротирај се око $\vec{H}$ за угао од a степени. Ако је a позитивно, корњача се ваља на десно, а ако је a негативно, корњача се ваља на лево.

Примети да се симболи $+$, $\&$, $\wedge$ и $/$ користе и као слова у алфабету V и као оператори у логичким и аритметичким изразима. Њихово значење зависи од контекста.

Моделирање дрвећа

Користићемо следеће претпоставке:

- Дужине две сегмент-ћерке од сегмент-мајке се скраћују са константним размерама, r_1 и r_2 ,
- Сегмент-ћерке формирају константне углове гранања a_1 и a_2 ,
- Основа гране је фиксирана према смеру гравитације тако да је што више хоризонтална. Изузетак су гране које расту из главног стабла. У овом случају, константни дивергентни угао α између узастопних бочних сегмената је одржан (не мења се).

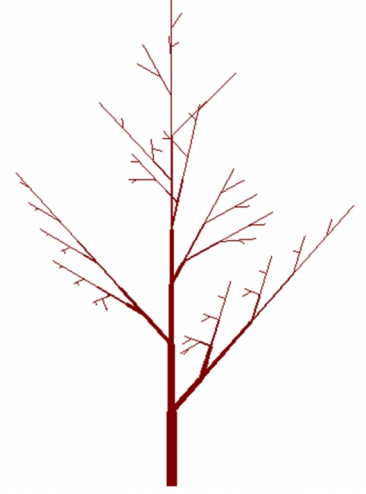
Наредне примере сам генерисао у софтверу L-studio.

3 Линденмајерови системи

```

n = 10
#define r1 0.9      /*контракциони однос између стабала*/
#define r2 0.6      /*контракциони однос за гране*/
#define a0 45      /*угао гранања од стабла*/
#define a2 45      /*угао гранања од бочних врхова*/
#define d 137.5     /*угао дивергенције*/
#define ωr 0.707    /*стопа смањења ширине*/
ω : A(1,10)
p1 : A(l, ω) : * → !(ω)F(l)[&(a0)B(l * r2, ω * ωr)]/(d)A(l * r1, ω * ωr)
p2 : B(l, ω) : * → !(ω)F(l)[- (a2)$C(l * r2, ω * ωr)]C(l * r1, ω * ωr)
p3 : C(l, ω) : * → !(ω)F(l)[+ (a2)$B(l * r2, ω * ωr)]B(l * r1, ω * ωr)

```

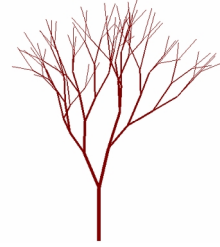


Наравно, осмишљени су и другачији начини за креирање дрвећа:

```

n = 6
ω : !(1)F(1)A(1)
p1 : !(x) → !(x + 1)
p2 : A(s) → [/(137) & (25)!(1)F(0.85s)A(0.85s)]
          /(43) & (25)!(1)F(0.75s)A(0.75s)

```



Према продукцији p_1 , врх главне осе A производи унутрашњи чвор F и бочни врх B у сваком деривацијском кораку. Константе r_1 и r_2 спецификују контракционе односе за равне и бочне сегменте, a_0 и a_2 су углови гранања и d је угао дивергенције. Модул $!(\omega)$ поставља ширину линије за ω , па према томе продукција p_1 смањује ширину ћерки-сегмената са фактором $\omega_r = 0.707$. Ова константа задовољава постулат Леонарда да Винчија, по коме: „Све гране дрвета на сваком нивоу његове висине када се саберу су једнаке дебљини ставла испод њих“. У случају када мајка-грана дијаметра ω_1 даје раст за две ћерке-гране једнаких дијаметара ω_2 , овај поступак рађа једначину $\omega_1^2 = 2\omega_2^2$, која даје вредност за ω_r једнаку $\omega_2 / \omega_1 = 1/\sqrt{2} \approx 0.707$.

Продукције p_2 и p_3 описују подниз развоја бочних грана. У сваком деривацијском кораку, раван врх (или B или C) издаје бочни врх следећег реда на углу a_2 или $-a_2$. Две продукције се употребљавају за креирање бочних врхова наизменично на лево и десно. Симбол $\$$ „увија“ корњачу око своје осе тако да вектор $\vec{L}$ који показује на лево од корњаче је доведен до хоризонталне позиције. Стога, основа гране је најближа

3 Линденмајерови системи

горизонталној равни. Са техничке тачке гледишта, $\vec{L}$ модификује корњачину оријентацију у простору према формули:

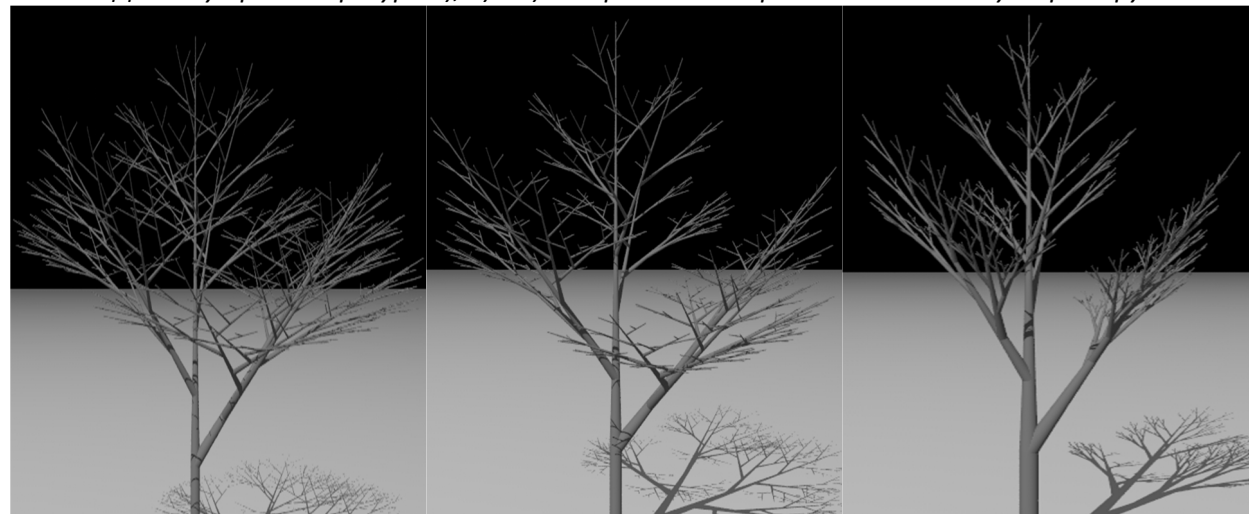
$$\vec{L} = \frac{\vec{V} \times \vec{H}}{|\vec{V} \times \vec{H}|} \text{ и } \vec{U} = \vec{H} \times \vec{L},$$

где су вектори $\vec{H}$, $\vec{L}$ и $\vec{U}$ вектори повезани са корњачом, $\vec{V}$ је правац супротан гравитацији, и $|\vec{A}|$ означава дужину вектора $\vec{A}$. Структуре налик дрвету приказане на наредној слици су генерисане користећи вредности константи које су дате у следећој табели

Фигура	r_1	r_2	a_1	a_2
<i>a</i>	0.9	0.9	45	45
<i>b</i>	0.9	0.8	45	45
<i>c</i>	0.9	0.7	30	-30



Слика. Дати су прикази фигура *a*), *b*) и *c*) генерисаних из претходне табеле у софтверу L-studio



Слика. Сlike из претходног примера креиране у софтверу POV-Ray

3 Линденмајерови системи

Најважније код креирања природних, реалистичних, слика је случајност, па тако и дрвеће можемо да креирамо користећи стохастичке L-системе. У следећем примеру продукције 2 и 3 имају једнаке вероватноће.

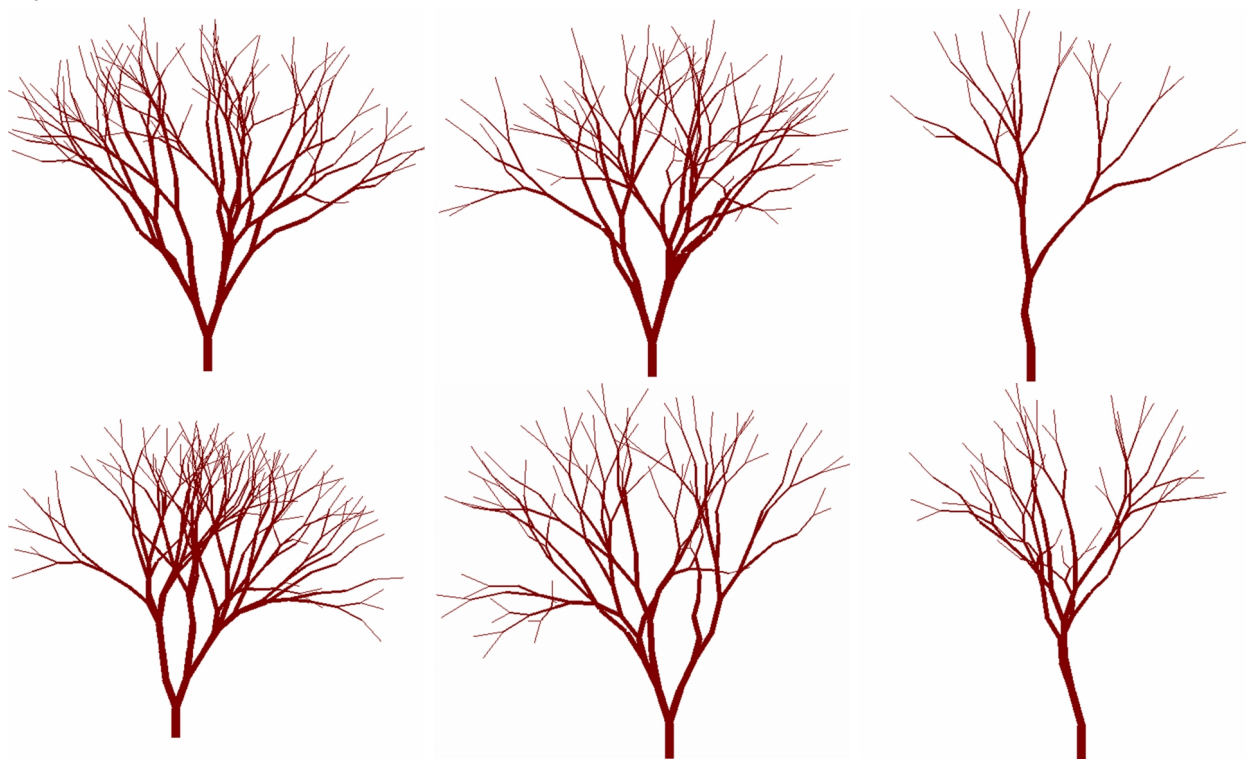
$$n = 10$$

$$\omega : !(1)FA$$

$$p_1 : !(n) \rightarrow !(n+1)$$

$$p_2 : A \rightarrow /(137)[+(20)!(1)FA] - (20)!(1)FA : 1/2$$

$$p_3 : A \rightarrow /(137) - (20)!(1)FA : 1/2$$



Слика. Шест слика креираних истим L-системом

Можемо лако да модификујемо L-системе и правимо од њих стохастичке како би добијали реалније слике. Изменићемо мало L-систем за креирање дрвећа од раније на следећи начин:

$$n = 6$$

$$\omega : !(1)F(1)A(1)$$

$$p_1 : !(x) \rightarrow !(x+1)$$

$$p_2 : A(s) \rightarrow [/(137) \& (25)!(1)F(0.85s)A(0.85s)] / (43) \& (25)!(1)F(0.75s)A(0.75)$$

Ћемо претворити у

$$n = 7$$

$$\omega : !(1)F(1)A(1)$$

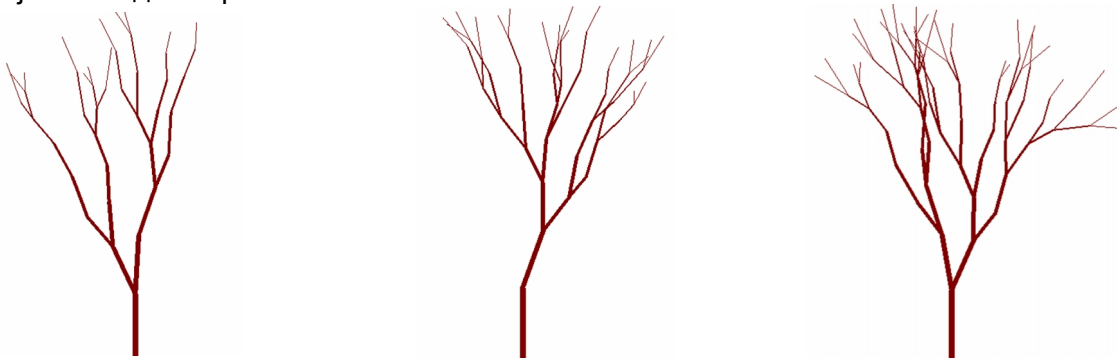
3 Линденмајерови системи

$$p_1 : !(x) \rightarrow !(x+1)$$

$$p_2 : A(s) \rightarrow [(137) \& (25)!(1)F(0.85s)A(0.85s)] / (43) \& (25)!(1)F(0.75s)A(0.75s) : 0.5$$

$$p_3 : A(s) \rightarrow [(120) \& (20)!(1)F(0.85s)A(0.85s)] : 0.5$$

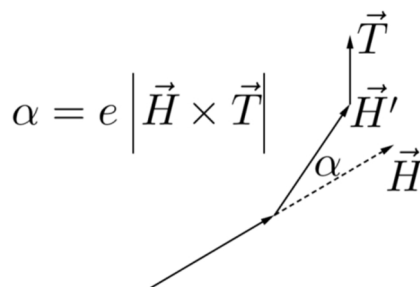
И добијамо следеће три слике стохастичког L-система



Код L-система досад свим сегментима је додељена њихова финална дужина у моменту креације и никаква даља продужења нису постојала. Сличне структуре могу бити добијене креирајући нове сегменте константне дужине и увећавањем дужина претходно креираних сегмената са константним фактором у сваком кораку.

Целокупна структура дрвета је дефинисана са продукцијом p_1 . У сваком кораку, врх A производи три нове гране које се завршавају својим врховима. Параметар ω и константа v_r доводе у везу ширину гране-мајке ω_1 са ћеркама-грананама ω_2 . Према да Винчијевом поступку $\omega_1^2 = 3\omega_2^2$ па је према томе $v_r = \omega_1 / \omega_2 = \sqrt{3} \approx 1.732$. Продукције p_2 и p_3 доводе до постепеног продужења грана и увећање у њиховим параметрима током времена.

Савијање грана се симулира малом ротацијом корњаче у смеру предефинисаном вектором тропизма $\vec{T}$ након цртања сваког сегмента. Подешавање оријентације α се израчунава користећи формулу $\alpha = e |\vec{H} \times \vec{T}|$, где је e параметар који представља осину осетљивост на савијање. Ако $\vec{T}$ интерпретирамо као силу примењену на крајње тачке вектора $\vec{H}$, и $\vec{H}$ може да се ротира око своје почетне тачке, обртни моменат је једнак $\vec{H} \times \vec{T}$



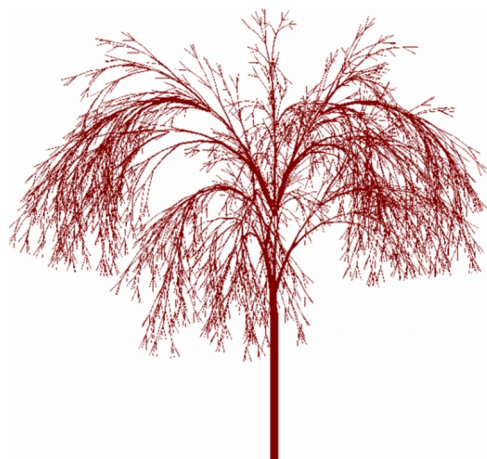
Слика. Корекција α од оријентације сегмента $\vec{H}$ услед тропизма $\vec{T}$

3 Линденмајерови системи

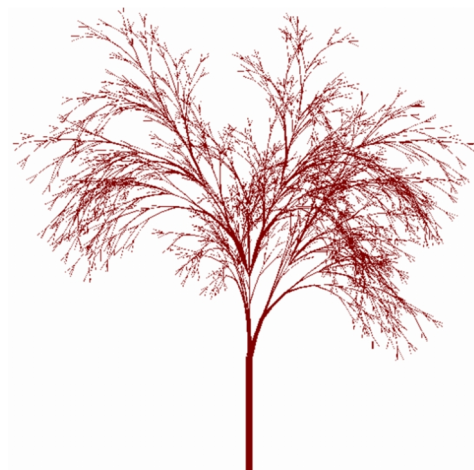
Дата су следећа два примера која сам креирао у софтверу L-studio.

Фигура	d_1	d_2	a	l_r	$\vec{T}$	e	n
a	94.74	132.63	18.95	1.109	0.00, -1.00, 0.00	0.22	6
b	137.50	137.50	18.95	1.109	0.00, -1.00, 0.00	0.14	6

```
#define d1 94.74 /*угао дивергенције 1*/
#define d2 132.63 /*угао дивергенције 2*/
#define a 18.95 /*угао гранања*/
#define lr 1.109 /*стопа продужења*/
#define vr 0.707 /*стопа увећања ширине*/
ω:!(1)F(200)/(45)A
p1:A:*→!(vr)F(50)[&(a)F(50)A]/(d1)[&(a)F(50)A]
/(d2)[&(a)F(50)A]
p2:F(1):*→F(l*lr)
p3:!(ω):*→!(ω*vr)
```

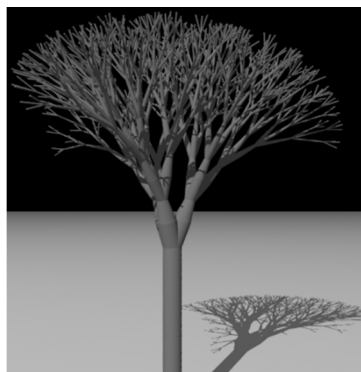


```
#define d1 94.74 /*угао дивергенције 1*/
#define d2 132.63 /*угао дивергенције 2*/
#define a 18.95 /*угао гранања*/
#define lr 1.109 /*стопа продужења*/
#define vr 0.707 /*стопа увећања ширине*/
ω:!(1)F(200)/(45)A
p1:A:*→!(vr)F(50)[&(a)F(50)A]/(d1)[&(a)F(50)A]
/(d2)[&(a)F(50)A]
p2:F(1):*→F(l*lr)
p3:!(ω):*→!(ω*vr)
```

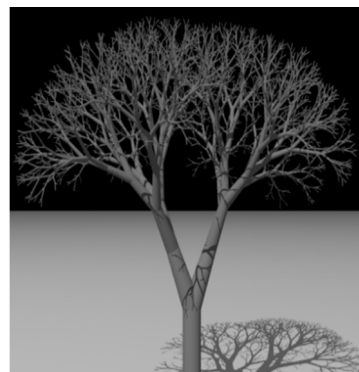


Дата су још два примера за креирање дрвета L-системом који користе различите, најчешће моделе. Оба примера сам креирао у софтверу POV-Ray.

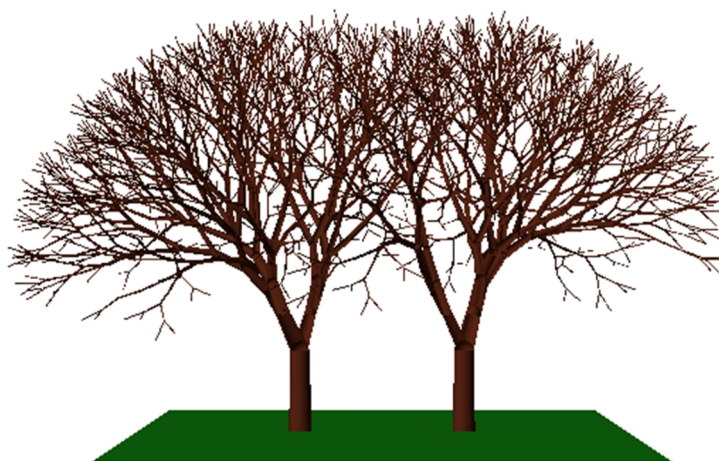
3 Линденмајерови системи



$n = 6$
 $\omega : !(0.01)!(0.01)F(2)/(45)A$
 $p_1 : A \rightarrow !(0.01732)F(0.50)$
 $[\&(18.95)F(0.50)A]/(94.74)$
 $[\&(18.95)F(0.5)A]/(132.63)$
 $[\&(18.95)F(0.5)A]$
 $p_2 : F(l) \rightarrow F(l*1.109)$
 $p_3 : l(w) \rightarrow !(w*1.732)$



$n = 12$
 $\#define \ r_1 = 0.8$
 $\#define \ r_2 = 0.8$
 $\#define \ a_1 = 30$
 $\#define \ a_2 = -30$
 $\#define \ \gamma_1 = 137$
 $\#define \ \gamma_2 = 137$
 $\#define \ q = 0.5$
 $\#define \ e = 0.5$
 $\#define \ min = 0$
 $\omega : !(15)A(100,15)$
 $p_1 : A(s, w) : s > min \rightarrow !(w)F(s)$
 $[+(a_1)/(\gamma_1)A(s*r_1, w*q^e)]$
 $[+(a_2)/(\gamma_2)A(s*r_2, w*(1-q)^e)]$

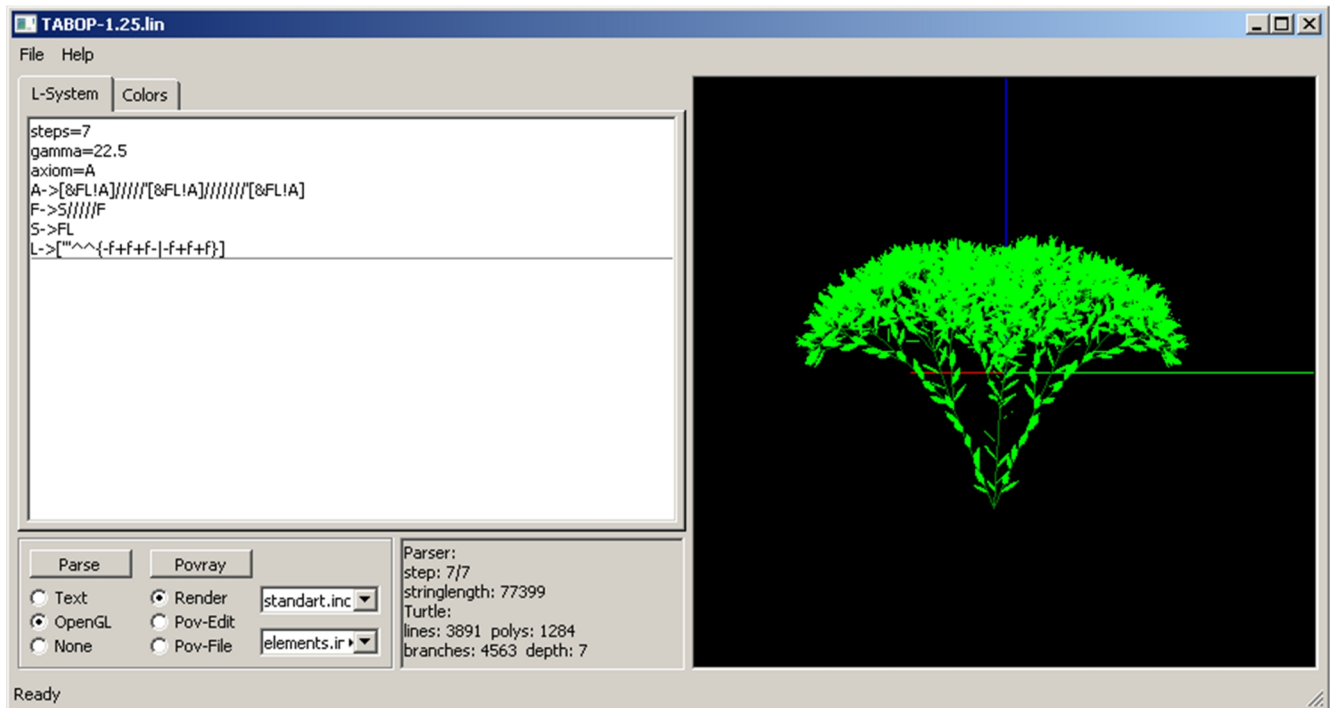


Слика. Два дрвета креирана у софтверу L-studio

Ево како примена L-система изгледа у два софтвера.

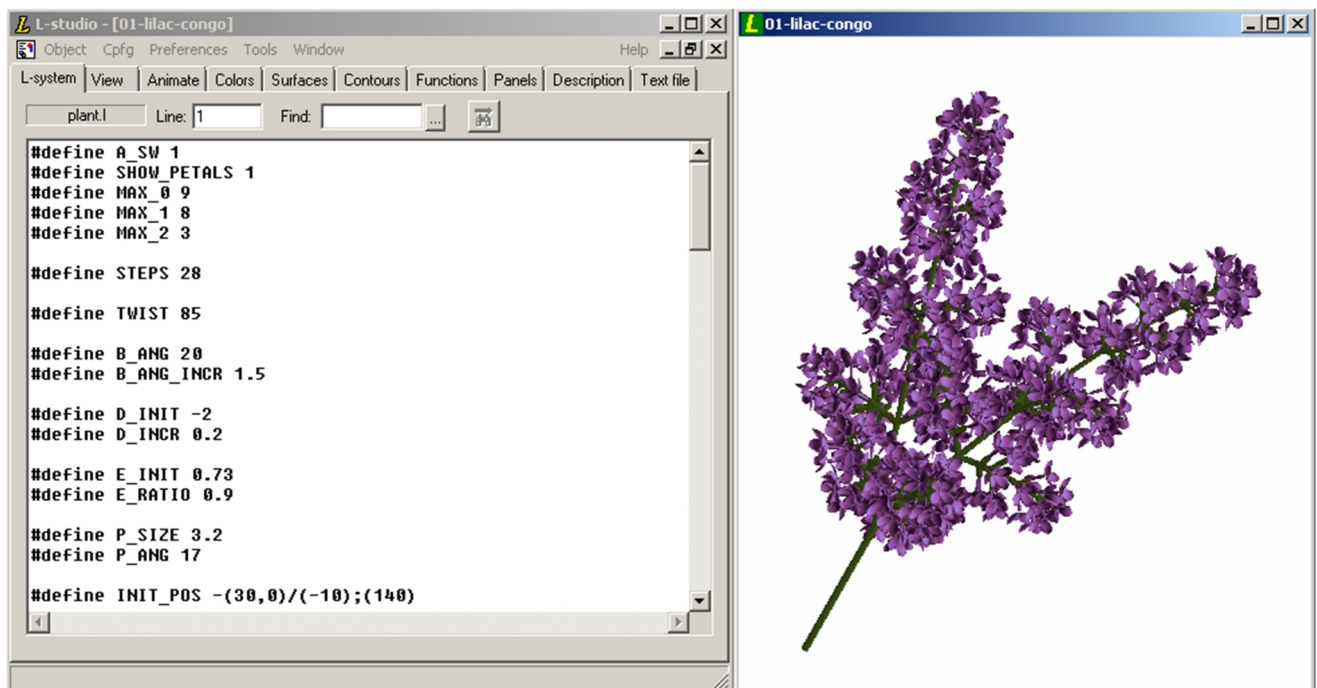
3 Линденмајерови системи

У првом софтверу креирамо жбунасту биљку коју смо раније дефинисали:



Слика. Приказан је програм XLinden и креирање L-система у њему

У другом софтверу се креира комплексан пример биљке: Syringa vulgaris Congo



Слика. Софтвер L-studio и пример креиран у њему

3 Линденмајерови системи

Приказан је код у софтверском пакету L-studio за креирање биљке *Mycelis muralis*:

```
Lsystem: 1
derivation length: 180
ignore: EF/^&\,;+-012345678f|9pmZ#!
Axiom: E///T(0)FFFA(0)

T(a) : a==60 --> b
T(a) --> T(a+1)
b < a --> b
d < a --> b
b < A(a) --> U(0) [;B(0)~w]
d < A(a) --> U(0) [;B(0)~w]
A(a) : a==3 --> a [&FFFx] [&&!{-F+F+F-|-F+F+F-}
]FFF/////////A(0)
A(a) --> A(a+1)
U(a) : a==2 --> cd
U(a) --> U(a+1)
b > c --> t(0)
c < x --> A(0)
c > d --> u(0)
t(a) : a==0 --> c
t(a) --> t(a+1)
u(a) : a==1 --> d
u(a) --> u(a+1)
B(a) : a==3 --> C(0)%
B(a) --> B(a+1)
C(a) : a==0 --> C(1)~z
C(a) : a==3 --> D(0)%
C(a) --> C(a+1)
D(a) : a==0 --> ,D(1)[FF,/[,^F][!{&&&&-F+F-|-F+F}]////
[,^F][!{&&&&-F+F-|-F+F}]////
[,^F][!{&&&&-F+F-|-F+F}]////
[,^F][!{&&&&-F+F-|-F+F}]////
[,^F][!{&&&&-F+F-|-F+F}]
D(a) : a==20 --> D(a+1)%
D(a) : a>20 --> ;;~f
D(a) --> D(a+1)
E --> _E
Endlsystem
```



Битно је напоменути да се у засебном табу софтвера пише угао, који је у овом случају 21° , правац тропизма $-0.005, 1.0, 0.0$ и еластичност: -0.060 . Треба обратити и пажњу и на то да се пуполци и цветови креирају засебно па се само позивају у одређеном моменту. Они су у коду обележени словима **w**, **z** и **f**.

Сви неопходни симболи за креирање L-система:

Симболи

Симболи који узрокују да се корњача креће напред и црта

$F(s), G(s)$	Крећи се напред за корак дужине s и цртај линијски сегмент од почетне до нове позиције корњаче.
$f(s), g(s)$	Крећи се напред за корак дужине s без цртања линије.
$@O(r)$	Нацртај сферу радијуса r у тренутној позицији.

Симболи који контролишу корњачу у простору

$+(\theta)$	Скрени лево за угао θ око $\vec{U}$.
$-(\theta)$	Скрени десно за угао θ око $\vec{U}$.
$\&(\theta)$	Нагни се на доле за угао θ око $\vec{L}$ осе.
$\wedge(\theta)$	Нагни се на горе за угао θ око $\vec{L}$ осе.
$/(\theta)$	Смотај се на лево за угао θ око $\vec{H}$ осе.
$\backslash(\theta)$	Смотај се на десно за угао θ око $\vec{H}$ осе.
$ $	Окрени се за 180° око $\vec{U}$ осе. Ово је еквивалентно са $+(180^\circ)$ или $-(180^\circ)$.

Симболи за моделирање структура са гранама

$[$	Сачувај тренутно стање корњаче, укључујући позицију, правац и боју.
$]$	Поврати стање корњаче тачно тамо где је била када је претходна команда $]$ процесуирана.

Симболи за креирање и укључивање површина

$\{$	Почни да сачуваваш следеће позиције корњаче као темена полигона који се попуњава.
$\}$	Попуни сачуван полигон.
$\sim X(s)$	Нацртај површину идентификовану са симболом X , скалирану са s , у корњачиној тренутној ситуацији и оријентацији.

Симболи који мењају особине цртања

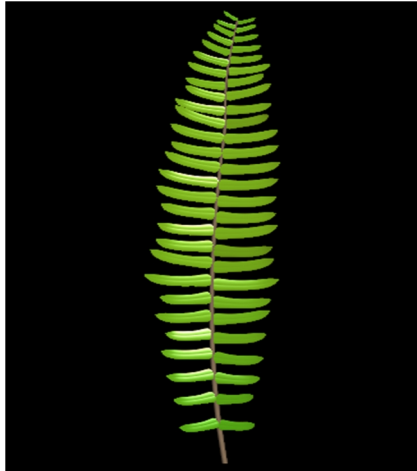
$\#(\omega)$	Постави за ширину линије ω , или увећај вредност ширине тренутне линије за уобичајено повећање ширине ако није дат параметар.
$!(\omega)$	Постави за ширину линије ω , или умањи вредност ширине тренутне линије за уобичајено смањење ако није дат параметар.
$;(n)$	Постави индекс боје на n , или увећај вредност тренутног индекса за уобичајено увећање боје ако није дат параметар.

3 Линденмајерови системи

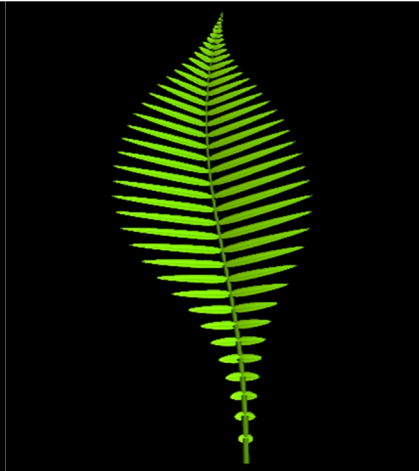
,(n) Постави индекс боје на n , или смањи вредност тренутног индекса за уобичајено смањење боје ако није дат параметар

Уобичајена дужина линија представљена симболима F без параметра је 1, а уобичајена величина углова представљених симболима $+$ и $-$ је 45° .

Приказани су неки од комплекснијих примера добијени употребом L-система креирани у софтверу L-studio:



Pellaea falcata



nanpam



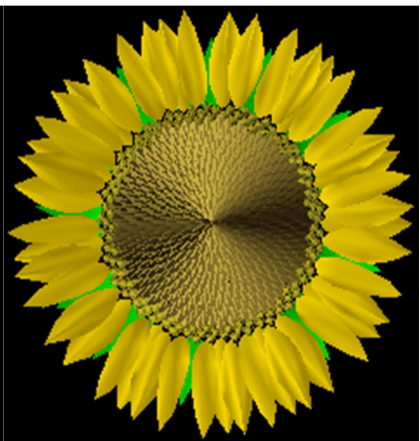
Indian paintbrush.



љилијан



gaillardia clump



сунцокретов цвет

4

Фракционо Брауново кретање

1827. године Шкотски ботаничар Роберт Браун је приметио да се мале честице течности и гасу крећу неправилним стазама. Овај и сличан феномен честица гаса у ваздуху су објашњени као резултат молекулараног бомбардовања честица. Течност и гас су сачињени од молекула који се непрекидно крећу и ударају мале честице из различитих смерова. Научници су покушали да предвиде кретање честице у односу на време али су схватили да се честице крећу „на најлуђи могући начин“ и да њихово кретање не тежи некој граници како се време узето за обсервацију смањује. Ајнштајн је публикувао математичку студију овог кретања, који је евентуално довео до Периновог освајања Нобелове награде за израчунавање Авогадровог броја.

Брауново кретање

Прво ћемо дефинисати Брауново кретање у једној димензији а затим ћемо проширити дефиницију на вишедимензионалне случајеве.

Да бисмо дефинисали Брауново кретање, разматраћемо честицу која изводи случајан ход на реалној линији. Претпоставимо да у малим временским интервалима τ честица прелази мала растојања δ , насумично лево или десно.

Нека $X_\tau(t)$ представља позицију честице у временском тренутку t . За дату позицију $X_\tau(k\tau)$ у временском тренутку $k\tau$, позиција $X_\tau((k+1)\tau)$ у временском тренутку $(k+1)\tau$ има исту вероватноћу да буде $X_\tau(k\tau) + \delta$ или $X_\tau(k\tau) - \delta$. Претпостављајући да честица креће у координатном почетку у временском тренутку 0, позиција од t , за $t > 0$ описана је са случајном промењивом

$$X_\tau(t) = \delta(Y_1 + \dots + Y_{\lfloor t/\tau \rfloor})$$

Где су $Y_1, Y_2, \dots$ независне случајне промењиве, које са вероватноћом $1/2$ узимају вредност 1 и вероватноћом $1/2$ узимају вредност -1 . $\lfloor t/\tau \rfloor$ представља највећи цео број мањи или једнак од t/τ . Нормализујемо корак постављајући $\delta = \sqrt{\tau}$ тако да је

$$X_\tau(t) = \sqrt{\tau}(Y_1 + \dots + Y_{\lfloor t/\tau \rfloor}).$$

4 Фракционо Брауново кретање

Централна гранична теорема нам говори да за фиксирано t и мало τ , расподела случајне промењиве $X_\tau(t)$ има апроксимативно нормалну расподелу са средином 0 и варијансом $\tau \lfloor t/\tau \rfloor \approx t$, пошто Y_i има средину 0 и варијансу 1. На исти начин, ако су t и h фиксирани и τ је довољно мало, онда је $X_\tau(t+h) - X_\tau(t)$ апроксимативно нормално са средином 0 и варијансом h . Такође примећујемо да ако је $0 \leq t_1 \leq t_2 \leq \dots \leq t_{2m}$, онда су прираштаји $X_\tau(t_2) - X_\tau(t_1), X_\tau(t_4) - X_\tau(t_3), \dots, X_\tau(t_{2m}) - X_\tau(t_{2m-1})$ независне случајне промењиве.

Нека је (Ω, F, P) простор вероватноћа. X зовемо случајним процесом или случајном функцијом од $[0, \infty)$ ка $\mathbb{R}$ ако је $X(t)$ случајна промењива за свако t са $0 \leq t < \infty$. Понекад, уместо тога, посматрамо случајне функције на коначном интервалу $[t_1, t_2]$.

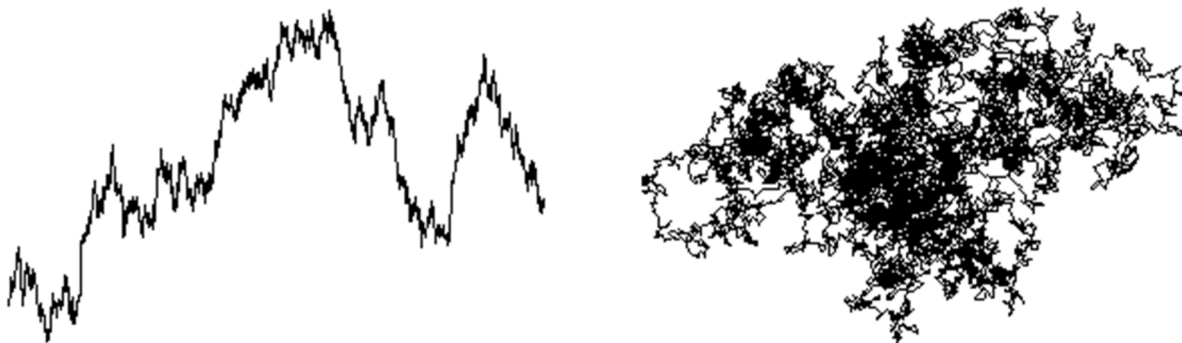
Дефиниција. Брауново кретање је случајан процес X тако да је:

- 1) Са вероватноћом 1, $X(0) = 0$ (односно, процес почиње у координатном почетку) и $X(t)$ је непрекидна функција од t ;
- 2) За свако $t \geq 0$ и $h > 0$ прираштај $X(t+h) - X(t)$ има нормалну расподелу са средином 0 и варијансом h , па према томе:

$$P(X(t+h) - X(t) \leq x) = (2\pi h)^{-1/2} \int_{-\infty}^x \exp\left(-\frac{u^2}{2h}\right) du;$$

- 3) Ако је $0 \leq t_1 \leq t_2 \leq \dots \leq t_{2m}$, прираштаји $X(t_2) - X(t_1), X(t_4) - X(t_3), X(t_{2m}) - X(t_{2m-1})$ су независни.

Приметимо да непосредно из 1) и 2) следи да $X(t)$ има нормалну расподелу са средином 0 и варијансом t за свако t . Прираштаји од X су *стационарни*; односно, $X(t+h) - X(t)$ има расподелу независну од t .



Слика. Симулација Брауновог кретања у једној и две димензије

Прво питање које се поставља је да ли постоји заправо случајна функција која задовољава услове Брауновог кретања. Доста је тешко показати да Брауново кретање

4 Фракционо Брауново кретање

заправо постоји, и то нећемо показивати овде. Доказ користи специјалне особине нормалне расподеле. Нпр, с обзиром да су $X(t_2) - X(t_1)$ и $X(t_3) - X(t_2)$ независни и имају нормалне расподеле са срединама 0 и варијансама $t_2 - t_1$ и $t_3 - t_2$, сума $X(t_3) - X(t_1)$ обавезно има нормалну расподелу са средином 0 и варијансом $t_3 - t_1$.

Уместо доказивања постојања, поменућемо две методе за симулирање Браунових узорачких функција.

Прва метода користи апроксимацију случајног хода $X_\tau(t) = \sqrt{\tau}(Y_1 + \dots + Y_{\lfloor t/\tau \rfloor})$. Вредности од 1 до -1 су додељене за Y_i „бацањем новчића“ за $1 \leq i \leq m$ где је m велико и $X_\tau(t)$ је исцртан према томе. Ако је τ мало у односу на t , онда ово даје тачну апроксимацију Браунове узорачке функције.

Друга метода, „random midpoint displacement“ метода може да се користи за добијање узорачке функције $X: [0,1] \rightarrow \mathbb{R}$. Дефинишемо вредности од $X(k2^{-j})$ где је $0 \leq k \leq 2^j$ са индукцијом по j . Постављамо $X(0) = 0$ и бирамо $X(1)$ случајно из нормалне расподеле са средином 0 и варијансом 1. Затим изаберемо $X\left(\frac{1}{2}\right)$ из нормалне расподеле са средином $\frac{1}{2}(X(0) + X(1))$ и варијансом $\frac{1}{2}$. У наредном кораку бирамо $X\left(\frac{1}{4}\right)$ и $X\left(\frac{3}{4}\right)$ и тако даље. У j -том кораку вредности $X(k2^{-j})$ за непарно k се бирају независно из нормалне расподеле са средином $\frac{1}{2}(X((k-1)2^{-j}) + X((k+1)2^{-j}))$ и варијансом 2^{-j} . Ова процедура даје $X(t)$ за све тачке $t = k2^{-j}$. Претпостављајући да је X непрекидно, X је потпуно одређено. Може се показати, користећи особине нормалне расподеле, да функције према томе генерисане имају расподелу дату са Брауновим кретањем.

Лако је проширити дефиницију Браунове узорачке функције са $\mathbb{R}$ на $\mathbb{R}^n$: само дефинишемо Брауново кретање на $\mathbb{R}^n$ тако да су координате компоненте независне једнодимензионална Браунова кретања. Према томе $X: [0, \infty) \rightarrow \mathbb{R}^n$ дато са $X(t) = (X_1(t), \dots, X_n(t))$ је n -димензионално Брауново кретање на неком простору вероватноћа ако је случајан процес $X_i(t)$ једнодимензионално Брауново кретање за свако i , и $X_1(t_1), \dots, X_n(t_n)$ су независни за скупове времена $t_1, \dots, t_n$.

По дефиницији, пројекција $X(t)$ -а на било коју од координатних оса је једнодимензионално Брауново кретање. Ипак, координатне осе нису специјалне у овом погледу: n -димензионално Брауново кретање има исте карактеристике у свим смеровима. Да бисмо видели ово, размотрићемо због погодности, случај дводимензионалног

4 Фракционо Брауново кретање

Брауновог кретања $X(t) = (X_1(t), X_2(t))$. Пројекција $X(t)$ -а на линију L_θ под углом θ кроз координатни почетак је $X_1(t)\cos\theta + X_2(t)\sin\theta$. За $t \geq 0$ и $h > 0$ случајне промењиве $X_1(t+h) - X_1(t)$ и $X_2(t+h) - X_2(t)$ су независне и имају нормалне расподеле са срединама 0 и варијансама h . Па према томе прираштаји пројекције на L_θ , дате са

$$(X_1(t+h) - X_1(t))\cos\theta + (X_2(t+h) - X_2(t))\sin\theta$$

имају нормалну расподелу са средином 0 и варијансом $h\cos^2\theta + h\sin^2\theta = h$. На сличан начин, прираштаји пројекције су независни, па је пројекција од $X(t)$ на L_θ једнодимензионално Брауново кретање, за све углове θ .

Ако је $\gamma > 0$, замењивањем h са γh и x са $\gamma^{1/2}x$ не мења десну страну од $P(X(t+h) - X(t) \leq x) = (2\pi h)^{-1/2} \int_{-\infty}^x \exp\left(\frac{-u^2}{2h}\right) du$ (заменом $u_1 = u\gamma^{-1/2}$ у интеграл). Према томе

$$P(X_i(t+h) - X_i(t) \leq x_i) = P(X_i(\gamma t + \gamma h) - X_i(\gamma t) \leq \gamma^{1/2}x_i)$$

за свако x_i . Следи да $X(t)$ и $\gamma^{-1/2}X(\gamma t)$ имају исте расподеле. Према томе мењање временске скале фактором γ и просторне скале фактором $\gamma^{1/2}$ даје процес који се не разликује од оригинала па Брауново кретање $X(t)$ и његов граф зовемо *статистички самоафиним*, а Браунове трагове зовемо *статистички самослични* због тога што се просторна расподела трагова $\{X(t): 0 \leq t \leq T\}$ и $\{X(t): 0 \leq t \leq \gamma T\}$ не разликују осим у фактору $\gamma^{1/2}$.

Претпоставимо да је $X(t) = (X_1(t), \dots, X_n(t))$ n -димензионално Брауново кретање. Пошто $X_i(t+h) - X_i(t)$ има нормалну расподелу за свако i , следи из

$$P(X(t+h) - X(t) \leq x) = (2\pi h)^{-1/2} \int_{-\infty}^x \exp\left(\frac{-u^2}{2h}\right) du \quad \text{да ако су } [a_i, b_i] \text{ интервали, онда}$$

$$P(X_i(t+h) - X_i(t) \in [a_i, b_i]) = (2\pi h)^{-1/2} \int_{a_i}^{b_i} \exp\left(-\frac{x_i^2}{2h}\right) dx_i.$$

Дакле, ако је E координата паралелопипеда $[a_1, b_1] \times \dots \times [a_n, b_n]$

$$P(X(t+h) - X(t) \in E) = \prod_{i=1}^n \left[(2\pi h)^{-1/2} \int_{a_i}^{b_i} \exp\left(-\frac{x_i^2}{2h}\right) dx_i \right]$$

4 Фракционо Брауново кретање

$$= (2\pi h)^{-n/2} \int_E \exp\left(-\frac{|x|^2}{2h}\right) dx$$

Где је $x = (x_1, \dots, x_n)$. Апроксимирајући скупе са унијама таквих паралелопипеда, следи

да $= (2\pi h)^{-n/2} \int_E \exp\left(-\frac{|x|^2}{2h}\right) dx$ важи за сваки Борелов скуп E . Према томе, узимање за E

лопту $B(0, \rho)$, и конвертујући у поларне координате,

$$P(|X(t+h) - X(t)| \leq \rho) = ch^{-n/2} \int_{r=0}^{\rho} r^{n-1} \exp\left(-\frac{r^2}{2h}\right) dr$$

Где је $c = 2\pi^{-n/2} a_n$ са a_n $(n-1)$ -димензионална област од површи од $B(0,1)$.

Фундаментална особина Брауновог кретања је та, да са вероватноћом 1 узорачка функција задовољава Холдеров услов од експонента λ за свако $\lambda < \frac{1}{2}$.

Теорема: Са вероватноћом 1, Браунова узорачка функција $X : [0, \infty) \rightarrow \mathbb{R}^n$ имају вишеструке тачке као што следи:

$n = 2$: има тачке вишеструкости k за сваки позитиван цео број k ;

$n = 3$: постоје двоструке тачке али не троструке;

$n \geq 4$: не постоје вишеструке тачке.

Фракционо Брауново кретање

Брауново кретање $X : [0, \infty) \rightarrow \mathbb{R}$ је Гаусов процес. Ово значи да је за $0 \leq t_1 \leq t_2 \leq \dots \leq t_m$ и скаларе $\lambda_1, \dots, \lambda_m$, случајна промењива $\lambda_1 X(t_1) + \dots + \lambda_m X(t_m)$ нормална. Заправо Брауново кретање има непрекидне прираштаје који су стационарни и имају коначне варијансе. Да бисмо добили узорачке функције са различитим карактеристикама неопходно је ослабити један или два од ових услова.

Постоје две уобичајене варијације. *Фракционо Брауново кретање* има прираштаје који имају нормалну расподелу али нису више независни. *Левиеви процеси*, са друге стране, изостављају услов са нормалном расподелом и ово може да доведе до испрекиданих функција.

Класично Брауново кретање је стохастички процес који је самосличан и има стационарне прираштаје. Он је заправо једини непрекидни Гаусовски процес који има ове две особине које се често примећују у „стварном животу“, нпр у кретању честица у течности, или у

4 Фракционо Брауново кретање

понашању цена финансијских актива. Уопште, природно је питати се да ли постоји стохастички процес који је истовремено Гаусовски, са стационарним прираштајима и самосличан, али не обавезно са индексом $1/2$ као у случају Брауновог кретања. Такав процес заправо постоји, уведен је од стране Колмогорова у четрдесетим годинама прошлог века за моделирање турбуленције у течностима. Име фракционо Брауново кретање (скраћено fBm), које је термин који сви данас користимо, су увели у својој студији Манделброт и Ван Нес. У теорији вероватноће, фракционо Брауново кретање, које се још зове и фрактално Брауново кретање је генерализација Брауновог кретања.

fBm-а се заснива на једном параметру H који узима вредност између нуле и јединице. Фракционо Брауново кретање је интересантно за моделирање, јер омогућава моделару да подешава вредности од H .

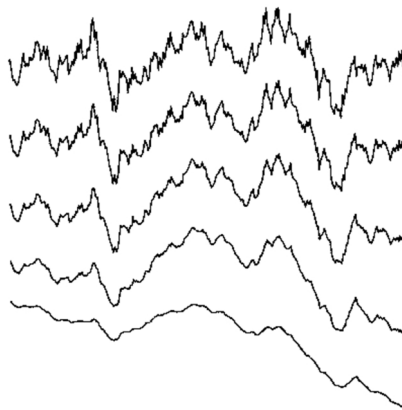
Дефиниција: Нека је $H \in (0,1]$. Фракционо Брауново кретање (fBm) од Хурстовог параметра H је непрекидни (по времену) Гаусовски процес $B_H(t)$, који почиње у нули, има очекивање нулу за свако t из $[0,T]$ и има следећу функцију коваријансе:

$$E[B_H(t)B_H(s)] = \frac{1}{2} \left(|t|^{2H} + |s|^{2H} - |t-s|^{2H} \right).$$

Где је H реалан број из $(0,1)$ који се назива Хурстов индекс или индекс самосличности. Хурстов експонент описује храпавост резултујућег кретања, са већом вредности води ка глађем кретању.

Вредност од H описује какав је fBm процес:

- 1) Ако је $H = 1/2$ онда је процес заправо Брауново кретање или Винеров процес;
- 2) Ако је $H > 1/2$ онда су прираштаји процеса позитивно корелисани;
- 3) Ако је $H < 1/2$ онда су прираштаји процеса негативно корелисани.



Слика. Једнодимензионално фракционо Брауново кретање са Хурстовим параметром 0.1, 0.3, 0.5, 0.7 и 0.9

4 Фракционо Брауново кретање

Кад да је $H > \frac{1}{2}$ лако се може проверити да је његова функција варијансе:

$$\frac{1}{2}(t^{2H} + s^{2H} - |t-s|^{2H}) = H(2H-1) \int_0^t du \int_0^s dv |v-u|^{2H-2}.$$

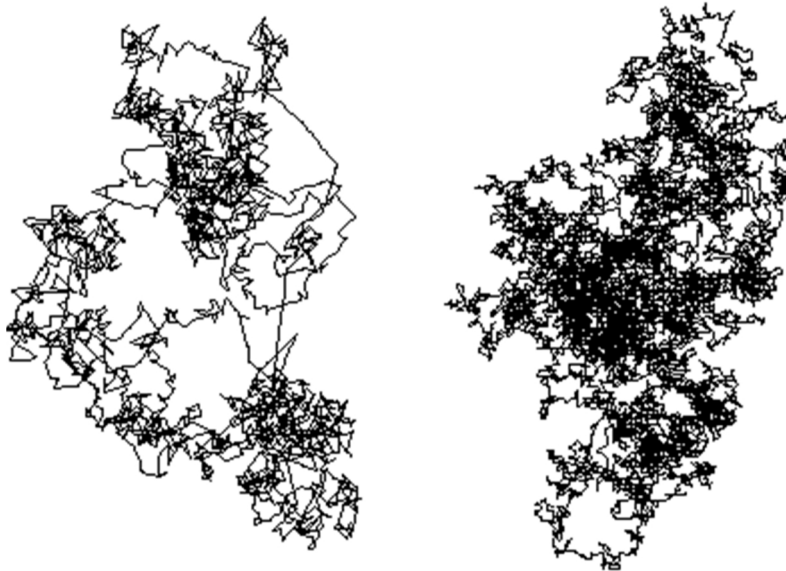
Нажалост, идентитет $\frac{1}{2}(t^{2H} + s^{2H} - |t-s|^{2H}) = H(2H-1) \int_0^t du \int_0^s dv |v-u|^{2H-2}$ не важи кад је

$H \leq \frac{1}{2}$ пошто у овом случају, $|v-u|^{2H-2}$ није интеграбилно.

Следећа пропозиција наглашава две посебне вредности од H , и показује да је случај $H=1$ на неки начин тривијалан. Као последица, од сада па надаље ми ћемо увек сматрати да је $0 < H < 1$.



Слика. Дводимензионално фракционо Брауново кретање



Слика. Разлика између фракционог Брауновог кретања и Брауновог кретања у две димензије

4 Фракционо Брауново кретање

Генерализација фракционог Брауновог кретања: **фракционо Брауново кретање n -тог реда**, скраћено **n -fBm**. n -fBm је Гаусовски, самосличан, нестационарни процес чији су прираштаји реда n стационарни. За $n = 1$, n -fBm је класични fBm.

Главна разлика између фракционог Брауновог кретања и обичног Брауновог кретања је да су прираштаји у Брауновом кретању независни, док су у фракционом Брауновом кретању прираштаји зависни. Ова зависност значи да ако постоји растућа шема у претходним корацима, онда је вероватно да ће и овај корак бити растући (ако је $H > 1/2$).

Дефиниција: Фракционо Брауново кретање са индексом H , $0 < H < 1$, је Гаусовски процес $X : [0, \infty) \rightarrow \mathbb{R}$ на неком простору вероватноћа тако да је:

- 1) Са вероватноћом 1, $X(t)$ је непрекидно и $X(0) = 0$;
- 2) За свако $t \geq 0$ и $h > 0$ прираштај $X(t+h) - X(t)$ има нормалну расподелу са средином 0 и варијасном $h^{2\alpha}$, тако да је

$$P(X(t+h) - X(t) \leq x) = (2\pi)^{-1/2} h^{-\alpha} \int_{-\infty}^x \exp(-u^2 / 2h^{2\alpha}) du.$$

Може се показати да, за $0 < \alpha < 1$, процес који задовољава ова два услова постоји.

Имплицитно је у горњој дефиницији да су прираштаји $X(t+h) - X(t)$ стационарни; односно, имају расподелу вероватноћа независну од t . Међутим, расподела функција одређена са 1) и 2) не може имати независне прираштаје осим у Брауновом случају $\alpha = \frac{1}{2}$. За услове 1) и 2), $E(X(t)^2) = t^{2\alpha}$ и

$$E(X(t+h) - X(t))^2 = h^{2\alpha}$$

Од којих

$$E(X(t)X(t+h)) = \frac{1}{2} [t^{2\alpha} + (t+h)^{2\alpha} - h^{2\alpha}]$$

Тако да

$$E(X(t)(X(t+h) - X(t))) = \frac{1}{2} [(t+h)^{2\alpha} - t^{2\alpha} - h^{2\alpha}]$$

Који је нунула ако је $\alpha \neq \frac{1}{2}$. Отуда је $E((X(t) - X(0))(X(t+h) - X(t)))$ позитивно или

негативно у зависности од тога да ли је $\alpha > \frac{1}{2}$ или $\alpha < \frac{1}{2}$. Према томе, прираштаји су

независни – ако је $\alpha > \frac{1}{2}$ онда $X(t) - X(0)$ и $X(t+h) - X(t)$ теже да буду истог знака.

Слично ако је $\alpha < \frac{1}{2}$ онда $X(t) - X(0)$ и $X(t+h) - X(t)$ теже да буду супротног знака.

4 Фракционо Брауново кретање

Приметимо да из $P(X(t+h) - X(t) \leq x) = (2\pi)^{-1/2} h^{-\alpha} \int_{-\infty}^x \exp(-u^2 / 2h^{2\alpha}) du$ следи да је процес

самоафин, односно да скалиране стазе $\gamma^{-\alpha} X(\gamma t)$ имају исту статистичку расподелу као и $X(t)$ за $\gamma > 0$.

Симулација фракционог Брауновог кретања је много тежа него симулација Брауновог кретања, јер прираштаји нису независни. Међутим, са модерном рачунарском снагом, тачна симулација је могућа.

Пропозиција: Нека је B^H фракционо Брауново кретања са Хурстовим параметром $H \in (0,1)$. Онда је процес:

- само сличан: $B_H(at) \sim |a|^H B_H(t)$
- има стационарне прираштаје: $B_H(t) - B_H(s) \sim B_H(t-s)$

Фракционе Браунове површине

Фракционе Браунове површине се користе веома ефективно у креирању компјутерски генерисаних пејзажа.

Замењујемо временску варијаблу t са координатама (x, y) тако да случајну промењиву $X(x, y)$ можемо да посматрамо као висину површине у тачки (x, y) .

За $0 < \alpha < 1$ дефинишемо индекс- α фракционо Брауново кретање $X: \mathbb{R}^2 \rightarrow \mathbb{R}$ да буде Гаусова случајна функција таква да је:

- 1) Са вероватноћом 1, $X(0,0) = 0$, и $X(x, y)$ је непрекидна функција од (x, y) .
- 2) За $(x, y), (h, k) \in \mathbb{R}^2$ висински прираштаји $X(x+h, y+k) - X(x, y)$ имају нормалну расподелу са средином нула и варијансом $(h^2 + k^2) = |(h, k)|^{2\alpha}$, тако

$$P(X(x+h, y+k) - X(x, y) \leq z)$$

$$= (2\pi)^{-1/2} (h^2 + k^2)^{-\alpha/2} \int_{-\infty}^z \exp\left(\frac{-r^2}{2(h^2 + k^2)^\alpha}\right) dr.$$

Мало напора је неопходно да бисмо демонстрирали постојање процеса који задовољава ове услове. Корелације између случајне промењиве $X(x, y)$ и између различитих тачака су прилично укључене.

Означавамо са $\{(x, y, X(x, y)) : (x, y) \in \mathbb{R}^2\}$ индекс- α фракциону Браунову површину.

4 Фракционо Брауново кретање

Упоређивањем $(2\pi)^{-1/2} (h^2 + k^2)^{-\alpha/2} \int_{-\infty}^z \exp\left(\frac{-r^2}{2(h^2 + k^2)^\alpha}\right) dr$ са расподелом

$$P(X(t+h) - X(t) \leq x) = (2\pi)^{-1/2} h^{-\alpha} \int_{-\infty}^x \exp(-u^2 / 2h^{2\alpha}) du$$
 видимо да је граф добијен

пресецањем $X(x, y)$ са било којом вертикалном равни једнодимнезионална индекс- α Браунова функција (после додавања константе да бисмо обезбедили $X(0) = 0$). Можемо често да добијемо информацију о површинама узимајући у обзир такве вертикалне секције.

Проблеми генерисања случајних површина за индекс- α Браунове функције су значајни, и не идемо у детаље сада. Међутим, примећујемо да су аналогно од

$$X(t) = \sum_{k=1}^{\infty} c_k \lambda^{-\alpha k} \sin(\lambda^k t + A_k) \text{ за индекс-}\alpha \text{ површине:}$$

$$X(x, y) = \sum_{k=1}^{\infty} C_k \lambda^{-\alpha k} \sin(\lambda^k (x \cos B_k + y \sin B_k) + A_k),$$

где су C_k независни са нормалном расподелом са средином нула и варијансом један, а A_k и B_k су независни са униформном расподелом на $[0, 2\pi)$. Такве функције обезбеђују један могући приступ за компјутерско генерисање случајних површина.

Идеја може бити проширена у много смерова и комбинована на много начина. Фракционо Брауново кретање може бити дефинисано од $\mathbb{R}^n$ до $\mathbb{R}^m$ за свако n, m и постоје многе друге варијације. Питање скупова нивоа, вишеструких тачака, пресека са фиксираним скуповима, сликама $X(F)$ за различите фрактале F и тако даље, појављују се у овим ситуацијама. Анализа таквих проблема често захтева софистициране технике вероватноће уз разне геометријске аргументе.

Браунова површина је фрактална површина генерисана са фракталном функцијом висине

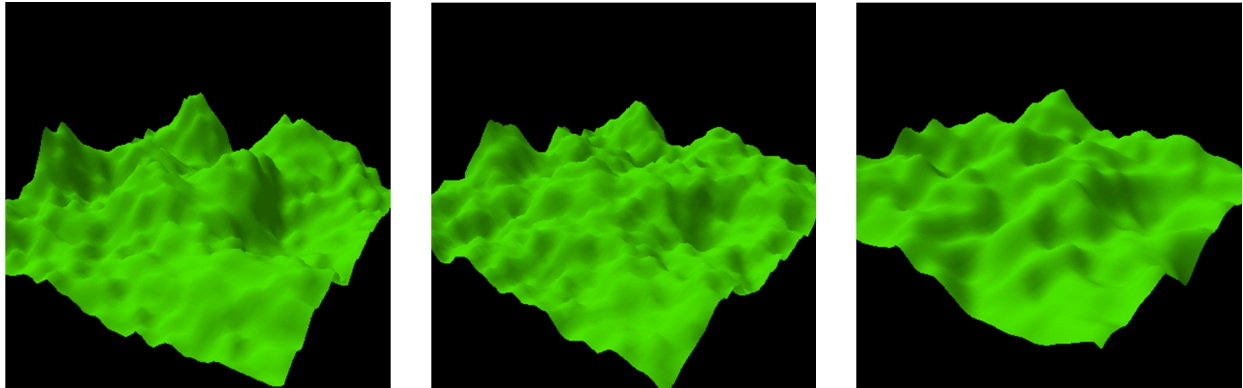
У тродимензионалном простору, где су две променљиве X и Y дате координате, функција висине између било које две тачке (x_1, y_1) и (x_2, y_2) може бити постављена да буде средина или очекивана вредност која се повећава као вектор растојања између (x_1, y_1) и (x_2, y_2) . Међутим, постоје многи начина за дефинисање функције висине. Фракционо Брауново кретање или различите функције ротације могу бити искоришћени да би се остварио природнији изглед површина.

Као и Брауново кретање, Браунове површине су исто назване по биологу Роберту Брауну из 19. Века.

Ефикасно генерисање фракционих Браунових површина захтева значајне изазове. Пошто Браунова површина представља Гаусов процес са нестационарном функцијом

4 Фракционо Брауново кретање

коваријансе, можемо да користимо методу декомпозиције Чолеског. Ефикаснија метода је Стајнова метода, која генерише помоћни стационарни Гаусов процес користећи приступ циркуларног уграђивања а затим подешава овај помоћни процес да би добио жељени нестационарни Гаусов процес. Фигура доле приказује три типичне реализације фракционих Браунових површина за различите вредности храпавости или Хурстовог параметра. Хурстов параметар је увек између нуле и јединице, са вредностима ближе један одговара глађим површинама (што је већи параметар глађа је површина).



Слика. Приказане су три слике фракционих Браунових површина са Хурстовим параметром 0.2, 0.5 и 0.8 редом. Сlike су испрограмиране користећи C++ и OpenGL

5

Алгоритми за креирање терена

Heightmap-е

Heightmap-а је мапа која дефинише вредности висина за наш терен. Односно, за мрежу у XY равни мапа висина дефинише вредности које ће продужити мрежу у Z осу.

Heightmap-а се најчешће представља као grayscale слика, односно слика у нијансама сиве боје (црно-белој скали) у којој сваки пиксел представља вредност висине (висине имају вредности од 0 до 255 што је еквивалентно броју нијанси сиве боје). Тамније боје представљају мање висина, а светлије представљају веће висине.

Сада ћемо да разматрамо како да процедурално генеришемо heightmap-е користећи три технике фракталног генерисања. Први је Перлинов шум.

Перлинов шум

Многи људи користе генераторе случајних бројева у својим програмима како би креирали непредвидивост, чинећи кретање и понашање објеката да изгледају природније, или да генеришу текстуре. Генератор случајних бројева сигурно има своје користи, али у неким случајевима њихов аутпут може бити превише груб да би изгледао природно. Ми ћемо приказати функцију која има веома широку примену било где где нам треба да нешто изгледа природно. Штавише, њен аутпут може лако бити модификован да одговара нашим потребама.

Многи објекти у природи су фрактали. Имају разне нивое детаља. Основни пример је контура планинских венаца. Оне садржи велике варијације у висини (планине), средње варијације (брда) и мале варијације (стене) и сићушне варијације (камење)... можемо ићи и даље. Расподела траве на ливади, таласи на мору, кретање мравца, кретање грана дрвета, шаблони у кликеру, ветрови... Сви ови феномени излажу исту шему великих и малих варијација. Функција Перлинов шума рекреира ово просто додавајући функције шума у опсегу различитих скала.

Да бисмо креирали функцију Перлиновог шума, требају нам две ствари, функција шума и интерполациона функција.

5 Алгоритми за креирање терена

Функција шума је у суштини генератор случајних бројева који узима цео број као параметар, и враћа случајан број који је заснован на том параметру. Ако убацимо тај исти параметар два пута, добићемо исти број. Веома је важно да се овако понаша наша функција, иначе би Перлинова функција производила бесмислице.

Дат је график који показује пример функције шума. Случајна вредност између 0 и 1 је додељена свакој тачки на X оси.

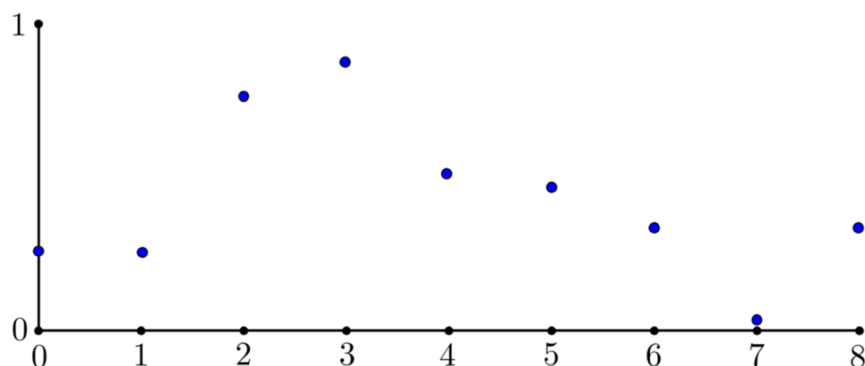


График. Случајне тачке на интервалу 0 до 8

Глатким интерполирањем између тачака, можемо да дефинишемо непрекидну функцију која узима не-целобројне параметре.

■

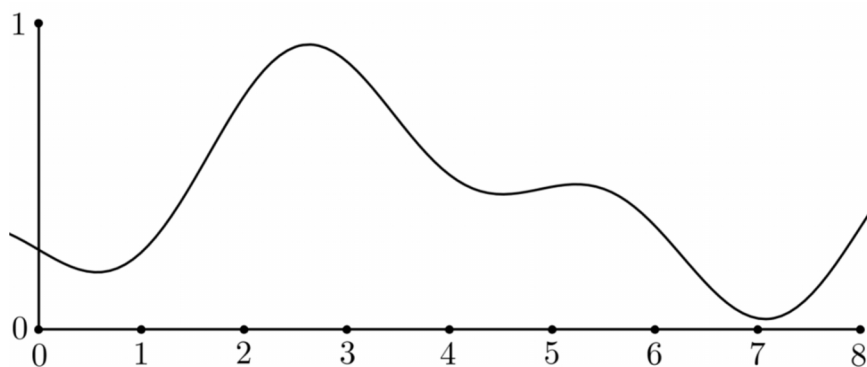


График. Косинусна интерполација над случајним тачкама

Пре него што одемо даље, прво ћемо да дефинишемо амплитуде и фреквенције.

Фреквенција представља број тачака који стаје у простор који смо креирали. Па је тако за креирање планина неопходно само пар тачака, међутим уколико желимо да креирамо и ситне детаље, стене и камење, неопходно ће нам бити на стотине тачака. Дакле, почињемо са малом фреквенцијом (за велике објекте) и идемо ка великим фреквенцијама које производе мање детаље.

5 Алгоритми за креирање терена

Амплитуда представља висину објекта. Фреквенцијама дефинише ширину објеката, а амплитуда одређује висину. Свака октава амплитуду скупља, што не мора увек да буде случај.

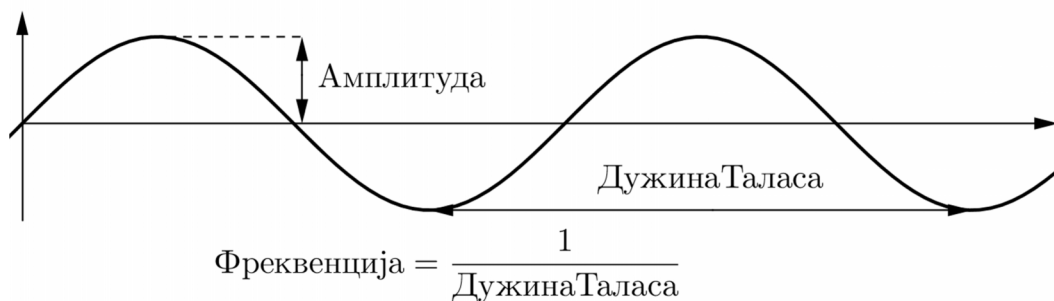


График. Синусни талас

Дужина синусног таласа представља растојање од једног врха до другог. Амплитуда је висина таласа. Фреквенција је дефинисана да буде $1/\text{дужинаталаса}$.

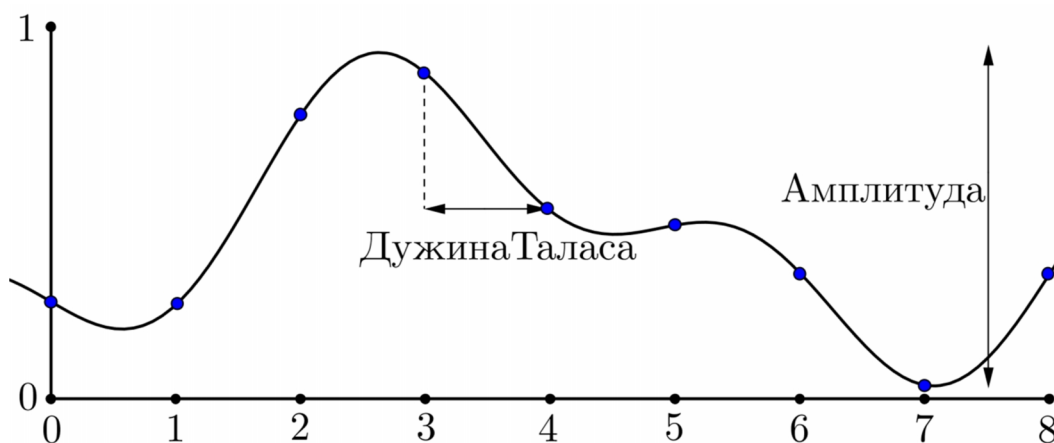
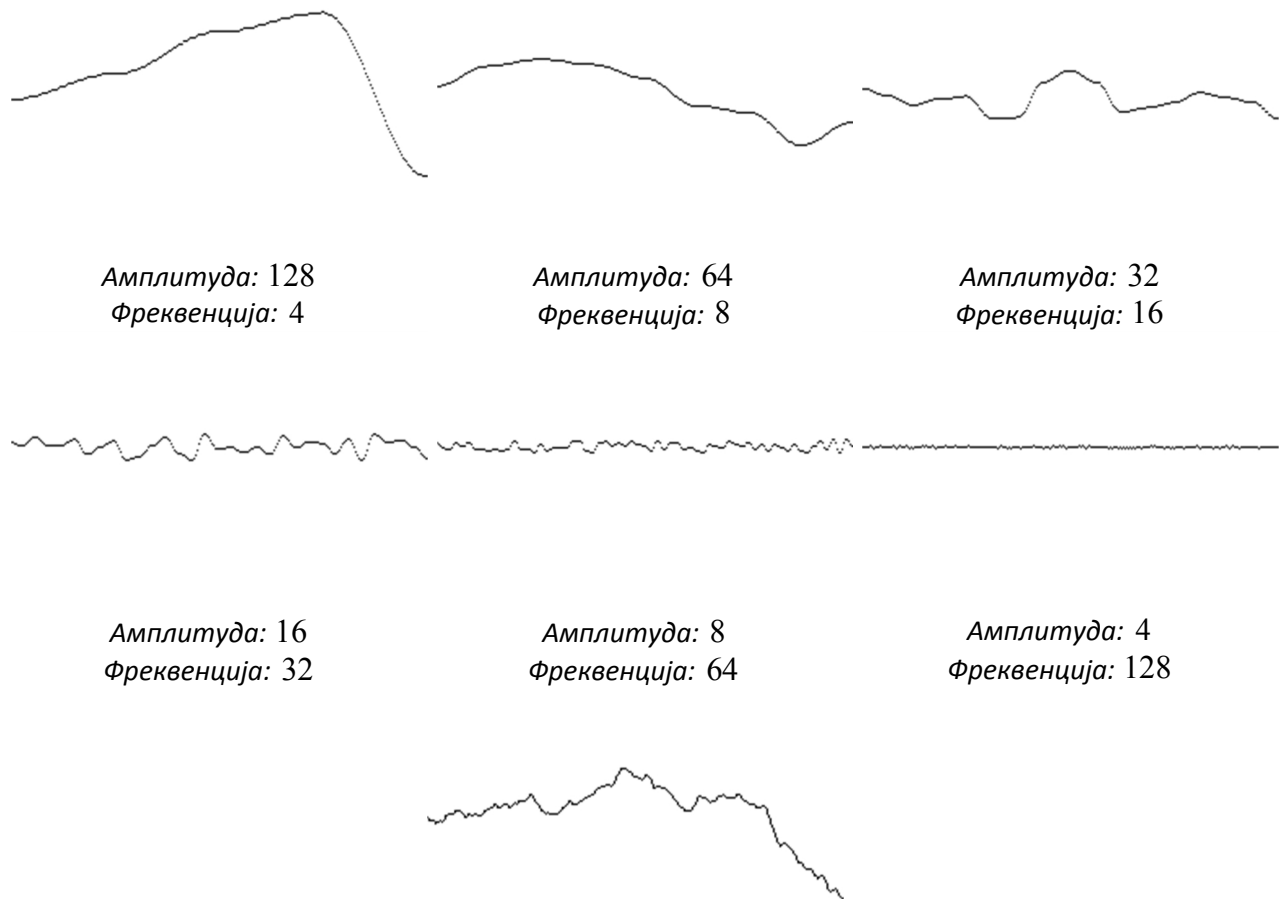


График. Функција шума

На графику овог примера функције шума, плаве тачке представљају случајне вредности дефинисане дуж димензије функције. У овом случају, амплитуда је разлика између минималних и максималних вредности које функција може да има. Дужина таласа је растојање од једне црвене тачке до следеће. Опет фреквенција је дефинисана да буде $1/\text{дужинаталаса}$.

Сада, ако узмемо пуно таквих глатких функција, са различитим фреквенцијама и амплитудама, можемо да их додамо све заједно да бисмо креирали лепу функцију шума.

5 Алгоритми за креирање терена



Перлинов шум

Слика. Једнодимензионални Перлинов шум

Можемо да видимо да је ова функција има велике, средње и мале варијације. Можемо и да је замислимо као планински венац. Заправо многи компјутерско генерисани пејзажи су направљени овом методом. Наравно они користе дводимензионални шум.

Сад ћемо урадити исто у две димензије.

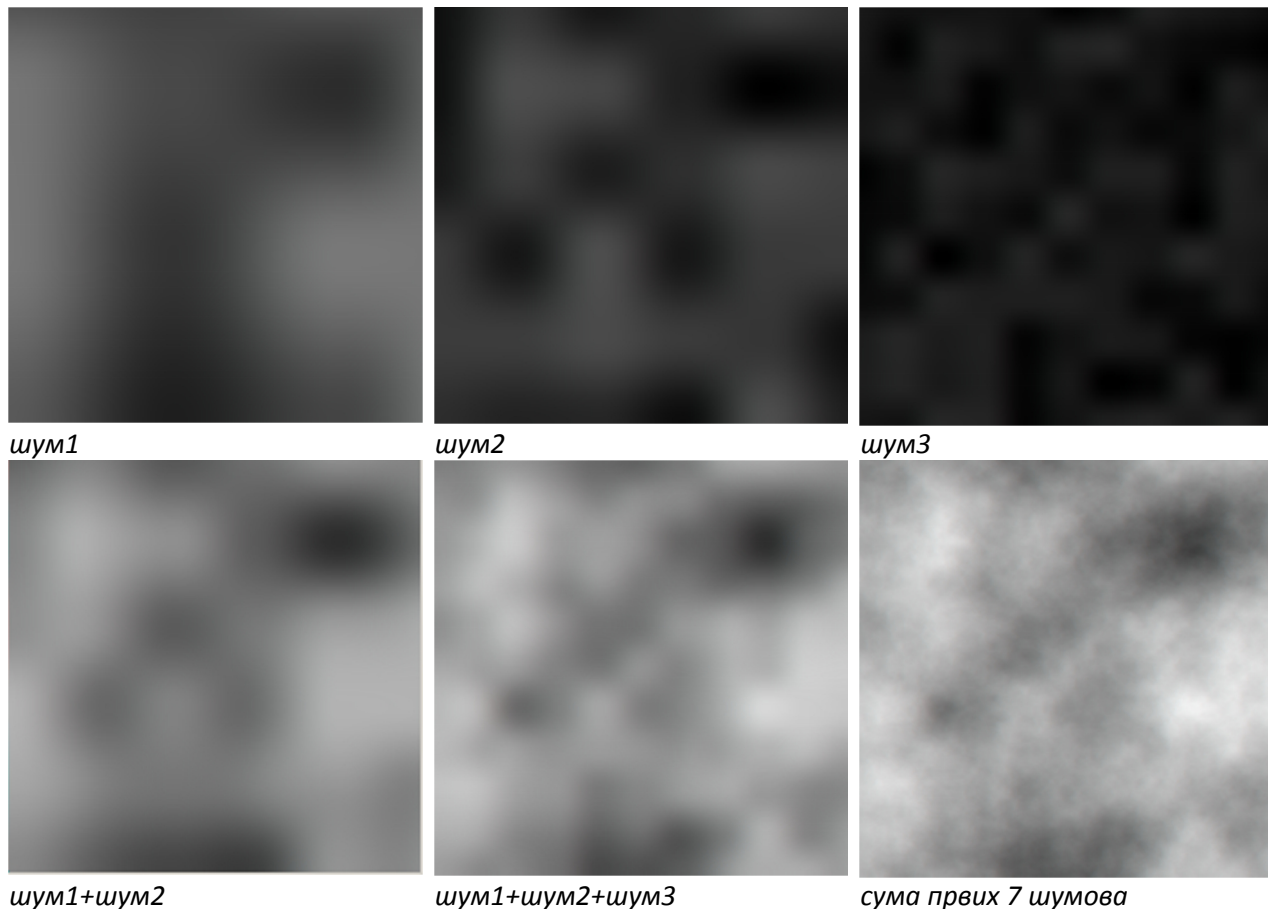
Неопходно је да дефинишемо октаве како бисмо наставили са дводимензионалним Перлиновим шумом

Октаве представљају колико слојева спајамо заједно. Ако кренемо са великим карактеристикама, број октава одређује колико ће детаљно мапа да изгледа. Свака сукцесивна функција шума коју додајемо је једна октава. Разлог за ово је тај што свака функција има дуплу фреквенцију од претходне функције. У музици, октаве имају ово

5 Алгоритми за креирање терена

својство. Колико тачно октава додајемо је на нама. Можемо да додамо колико год желимо.

Фракционо Брауново кретање је збир сукцесивних октава шума, сваки са вишом фреквенцијом и нижом амплитудом. У суштини није важно који врста шума, свака ће проћи. Фракционо Брауново кретање није увек неопходно и знатно успорава програм. Међутим, већина шума није фрактал по природи. Фракционо Брауново кретање је оно што их чини фракталним.



Када смо креирали шум, лако је креирати fBm фрактал комбинујући или множећи резултате од других шума. Међутим, пре него што одемо даље, важно је нагласити да можемо користити и друге вредности да бисмо контролисали шум који је генерисан. Већ смо поменули фреквенцију, али можемо да користимо и октаве, амплитуде и Н параметар.

Вредност октаве одређује колико вредности шума се додају или множи заједно. Вредност амплитуде подешава колико су високо вредности шума (или ако користимо фрактално генерисање за heightmap-у, ова вредност повећава просечну висину генерисаних вредности). Н параметар контролише колико се амплитуда мења за октаву.

Прављење Перлиновог шума

За овај и све наредне програме ја користим Windows, C++ i OpenGL.

Да бисмо почели, креираћемо функцију која ће да генерише вредност у неком интервалу; нпр, наћи ће случајну вредност између 1 и 52. То је база за нашу fBm имплементацију. Функција изгледа овако:

```
double Generator(int x)
{
    x=(x<<13)^x;
    return (((x*(x*x*15731+789221)+1376312589)& 0x7fffffff)/2147483648.0);
}
```

У суштини ова функција је генератор случајних бројева. За разлику од других генератора случајних бројева (који дају другачији број сваки пут када су позвани), ова функције шума даје исти резултат кад год унесемо исти број у функцију шума. Нећемо разматрати зашто ова функција изгледа овако. Ови велики бројеви у њој су прости бројеви и функција може да ради и са другим простим бројевима приближне величине ових. Једино што нам је битно је да нам ова функција генерише случајне бројеве.

Да бисмо креирали функцију шума неопходно је да интерполирамо ове случајне вредности. Стандардна функција интерполације узима три инпута, a и b , вредности између којих се интерполира, и x узима вредности између 0 и 1. Функција интерполације враћа вредности између a и b која је заснована на вредности x . Када је x једнако нула, враћа a , а када је x једнако један, така враћа b . Када је x између вредности 0 и 1, враћа неку вредност између a и b .

Линеарна интерполација није лош избор код једнодимензионалног цртања ивица планинских венаца, међутим за дводимензионалну интерполацију ће нам дати јако лош резултат. За разлику од ње кубна интерполација нам даје баш оно што желимо међутим много кошта временски да се испрограмира па ћемо да се задовољимо са косинусном интерполацијом која даје одличну слику а кошта временски мало више од линеарне интерпоалције.

Функција косинусне интерполације је дата на следећи начин:

```
double InterPolacija(double a, double b, double c)
{
    double d=c*3.1415927;
    double e=(1.0-cos(d))*0.5;
    return a*(1.0-e)+b*e;
}
```

Као што можемо да видимо, прво добијемо случајну вредност, а након тога добијамо вредности од три тачке које је окружују. Затим, желимо да интерполирамо вредности из

5 Алгоритми за креирање терена

прве две калкулације, интерполирамо вредности од треће и четврте калкулације и интерполирамо две интерполиране вредности

```
ceo_broj_x=(int)(x);
ceo_broj_y=(int)(y);
razlomak_x=x-ceo_broj_x;
razlomak_y=y-ceo_broj_y;
baza=ceo_broj_x+ceo_broj_y*num;
zajedno=(baza+seed);
a=InterPolacija(Generator(zajedno), Generator(zajedno+1), razlomak_x);
b=InterPolacija(Generator(zajedno+num), Generator(zajedno+1+num), razlomak_x);
vrednost=InterPolacija(a, b, razlomak_y)*amplituda;
return vrednost;
```

и имамо вредности шума за ту тачку! Ова функција сама неће урадити много, па морамо да креирамо функцију да креира цео fBm фрактал!

Креирање вредности fBm фрактала је заправо доста лако сада када имамо све неопходне базне функције. Функција фракталног креирања треба да узима аргументе за све вредности које смо разматрали до сад: октаве, аплитуду, фреквенцију и Н. Такође желимо да узмемо скуп (x, y) координата да бисмо могли да израчунамо вредности fBm-а у свакој тачки.

Сада када знамо све то, време је да спојимо све и да креирамо функцију Перлиновог шума. Она је само неколико интерполираних функција шума додатих заједно. Перлинов шум је у ствари само функција. Убацимо један или више параметара и она одговори са бројем. Једнодимензионални Перлинов шум нас не интересује па се са њим нећемо ни бавити.

Сада ћемо да прикажемо програм који креира Перлинов шум у две димензије.

```
#include <iostream>
#include <glut.h> //ukljucujemo OpenGL fajl zaglavlja
#include <stdlib.h>
#include <stdio.h>
#include <math.h> //uvedeno zbog kosinusa u funkciji kosinusne interpolacije

#define SirinaSlike 512 //definiseмо sirinu slike
#define VisinaSlike 512 //definiseмо visinu slike
GLubyte Podaci[SirinaSlike][VisinaSlike][3]; //definiseмо matricu Podaci koja ce
//cuvati visine za svaki piksel

static GLint visina; //definiseмо celobrojnu promenjivu visine
//Definiseмо funkciju kosinusne interpolacije
double InterPolacija(double a, double b, double c)
{
    double d=c*3.1415927;
    double e=(1.0-cos(d))*0.5;
    return a*(1.0-e)+b*e;
}
//Definiseмо funkciju koja ce nam davati slucajan broj
double Generator(int x)
{
```

5 Алгоритми за креирање терена

```
x=(x<<13)^x;
return (((x*(x*x*15731+789221)+1376312589)& 0x7fffffff)/2147483648.0);
}
//Definisemo funkciju koja kreira shum
double Shum(double x, double y, int korektor, int seed, double duzina_talasa)
{
    double a, b, vrednost, frekv, tam_pas, razlomak_x, razlomak_y;
    int baza, num, ceo_broj_x, ceo_broj_y;
    int amplituda=150; //dobra vrednost za amplitudu s obzirom na to da su vrednosti
                       //za grayscale sliku koju zelimo da dobijemo od 0 do 255
    int zajedno;
    frekv=1/duzina_talasa; //period cemo umanjivati dvostruko za svaki shum
                          //pa ce frekv rasti dvostruko
    num=(int)(korektor*frekv);
    //Izracunavanja vrednosti koje cemo unositi u funkciju interpolacije
    ceo_broj_x=(int)(x*frekv);
    ceo_broj_y=(int)(y*frekv);
    razlomak_x=x*frekv-ceo_broj_x;
    razlomak_y=y*frekv-ceo_broj_y;
    baza=ceo_broj_x+ceo_broj_y*num; //vrednost koju cemo unositi u generator
    zajedno=(baza+seed); //dodajemo vrednost seed-a kako bi smo mogli da promenom
                        //jedne vrednosti menjamo izgled perlinovog shuma
    //Dobijamo vrednosti a i b funkcijom interpolacije
    a=InterPolacija(Generator(zajedno), Generator(zajedno+1), razlomak_x);
    b=InterPolacija(Generator(zajedno+num), Generator(zajedno+1+num), razlomak_x);
    vrednost=InterPolacija(a, b, razlomak_y)*amplituda;
    return vrednost;
}
void makeImage(void)
{
    int x, y; //definisemo celobrojne x i y
    int seed; //definisemo seed, cijom promenom dobijamo razlicite krajnje rezultate
    int korektor; //definisemo sirinu
    //Definisemo shumove kao realne brojeve
    double shum1, shum2, shum3, shum4, shum5, shum6, shum7, skaliranje;
    //Sa dve petlje cemo proci kroz sve piksele slike
    for (y=0; y<SirinaSlike; y++)
    {
        for (x=0; x<VisinaSlike; x++)
        {
            skaliranje=1; //daje nam mogucnost skaliranja slike
            korektor=12413; //sto je veci manja je sansa da dobijemo istu sliku
            seed=134; //promenom seed vrednosti dobijamo drugaciji Perlinov shum
            //prvi shum
            shum1=Shum(x*skaliranje, y*skaliranje, korektor, seed, 100);
            //drugi shum
            shum2=Shum(x*skaliranje, y*skaliranje, korektor, seed, 50);
            //treci shum
            shum3=Shum(x*skaliranje, y*skaliranje, korektor, seed, 25);
            //cetvrti shum
            shum4=Shum(x*skaliranje, y*skaliranje, korektor, seed, 12.5);
            //peti shum
            shum5=Shum(x*skaliranje, y*skaliranje, korektor, seed, 6.25);
            //sesti shum
            shum6=Shum(x*skaliranje, y*skaliranje, korektor, seed, 3.125);
            //sedmi shum
            shum7=Shum(x*skaliranje, y*skaliranje, korektor, seed, 1.56);
```

5 Алгоритми за креирање терена

```
        //Sada cemo za svaki piksel da sacuvamo njegovu dobijeno visinu
        //[0], [1] i [2] predstavljaju red(crvenu), green(zelenu) i
        //blue(plavu) boju u kojoj ce pikseli da se cuvaju. Ako sacuvamo na
        //ovaj nacin pozivajuci sva tri dobicemo crno-belu sliku (grayscale
        //image) kao sto smo i zeleli
        Podaci[x][y][1]=(int)shum1+(int)(shum2*0.5)+(int)(shum3*0.25)+(int)(shum4*0.125)+
            (int)(shum5*0.0625)+(int)(shum6*0.03125)+(int)(shum7*0.0156);
        Podaci[x][y][2]=(int)shum1+(int)(shum2*0.5)+(int)(shum3*0.25)+(int)(shum4*0.125)+
            (int)(shum5*0.0625)+(int)(shum6*0.03125)+(int)(shum7*0.0156);
        Podaci[x][y][3]=(int)shum1+(int)(shum2*0.5)+(int)(shum3*0.25)+(int)(shum4*0.125)+
            (int)(shum5*0.0625)+(int)(shum6*0.03125)+(int)(shum7*0.0156);
    }
}

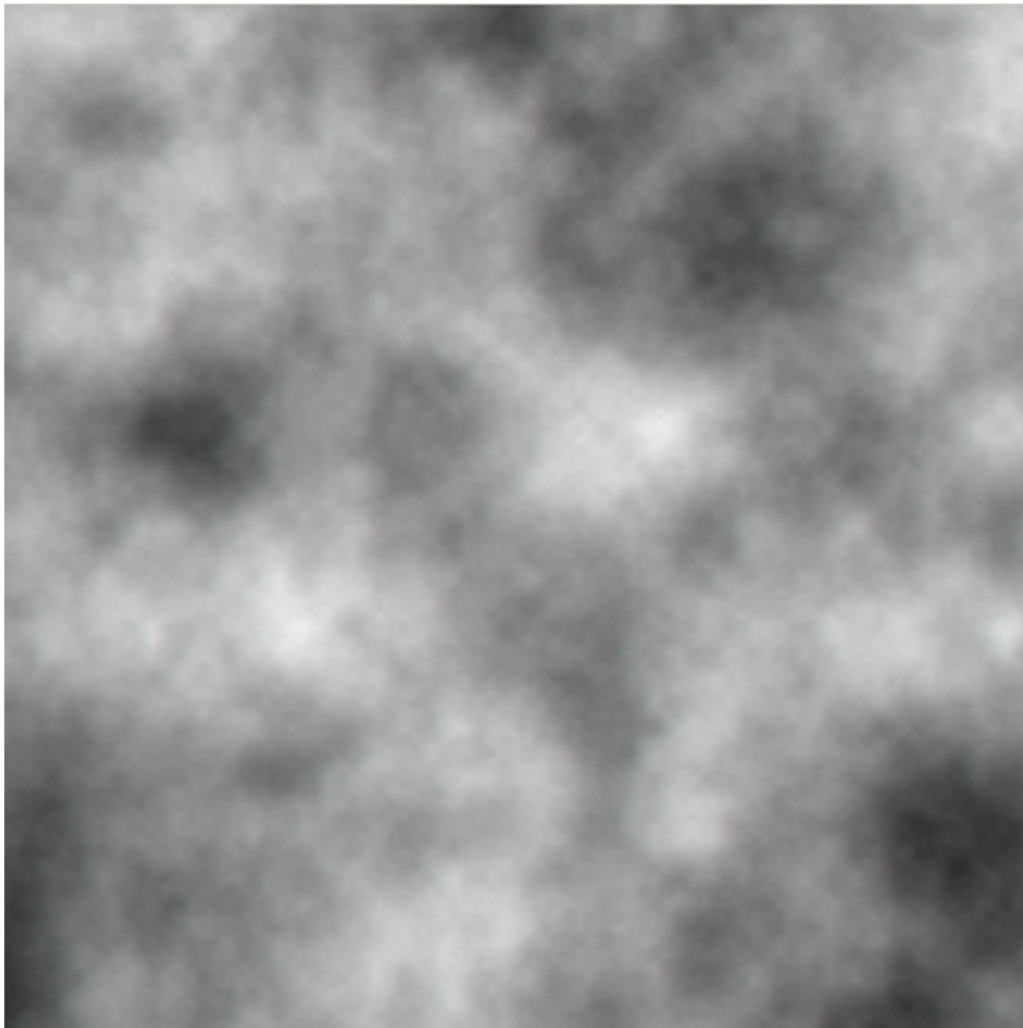
void init(void)//inicijalizuje 3D renderovanje
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);
    makeImage();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
}

void display(void)//Crtaj 3D scenu
{
    //obrisi informacije of poslednje brisanja
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glDrawPixels(SirinaSlike, VisinaSlike, GL_RGB, GL_UNSIGNED_BYTE, Podaci);
    glFlush(); //izvršenje GL komandi
}

void reshape(int w, int h) //poziva se kada prozor menja dimenzije
{
    //Govori OpenGL-u kako da konvertuje sa kordinatnih vrednosti na vrednosti piksela
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    visina=(GLint)h;
    glMatrixMode(GL_PROJECTION); //postavljanje perspektive kamere
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);
    glMatrixMode(GL_MODELVIEW);
    //Podesi perspektivu kamere
    glLoadIdentity(); //resetuj kameru
}

int main(int argc, char**argv)
{
    //Inicijalizacija GLUT-a
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(512, 512); //Podesavanje dimenzija prozora
    glutInitWindowPosition(100, 100);
    //Kreiraj prozor
    glutCreateWindow("Perlinov shum");
    init(); //Inicijalizuje renderovanje
    glutDisplayFunc(display); //crtamo sve sto je u funkciji display
    glutReshapeFunc(reshape);
    glutMainLoop(); //Zapocni main petlju
    return 0;
}
```

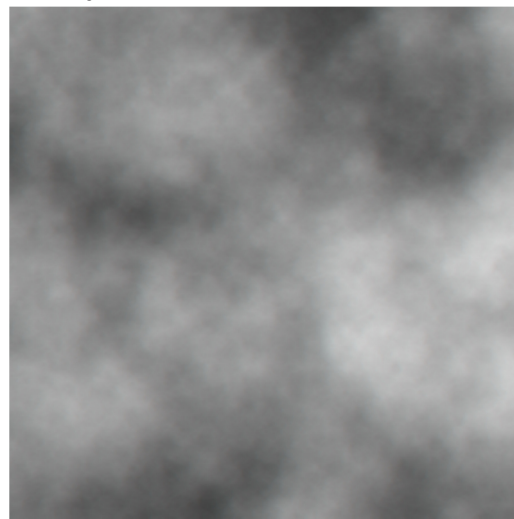
Датим кодом испрограмираћемо следећу слику Перлиновог шума:



Програм је лако модификовати за наше потребе. Прво ћемо умањити димензије слике на 256×256 пиксела, а затим са следећим подацима добијамо наредне слике:



seed=135



seed=140

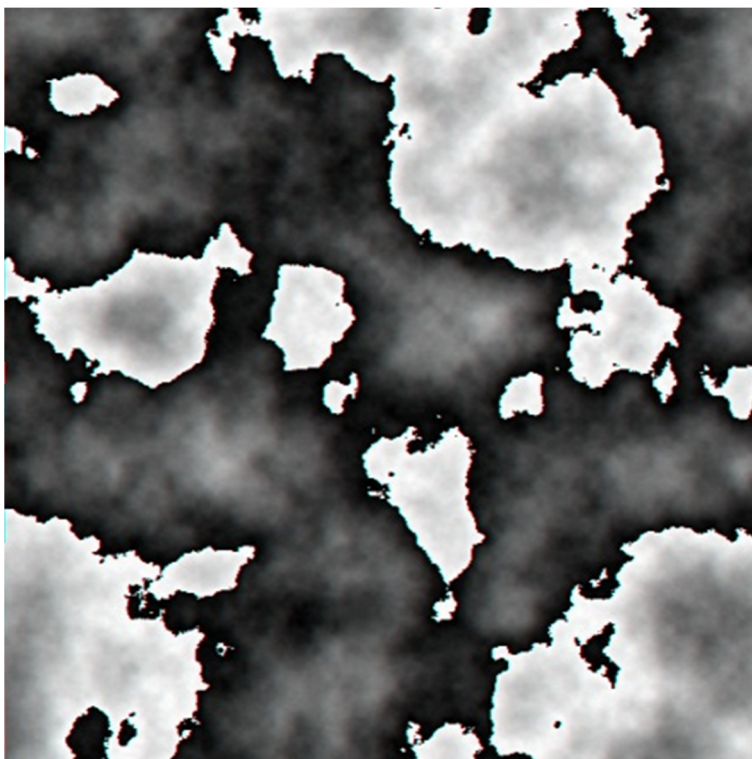
5 Алгоритми за креирање терена

Да бисмо могли да модификујемо функцију Перлиног шума и са Хурстовим параметром како бисмо скоро у потпуности могли да сматрамо да радимо са фракционим Брауновим кретањем неопходна је следећа измена кода:

За $H = 0.7$:

```
1) double PerlinNoise(double x, double y, int width, int seed, double period, double H)
2) int amplituda=100*H;
3) shum1=PerlinNoise(x*scale, y*scale, width, seed, 100, 1);
   shum2=PerlinNoise(x*scale, y*scale, width, seed, 50, 0.7);
   shum3=PerlinNoise(x*scale, y*scale, width, seed, 25, 0.7*0.7);
   shum4=PerlinNoise(x*scale, y*scale, width, seed, 12.5, 0.7*0.7*0.7);
   shum5=PerlinNoise(x*scale, y*scale, width, seed, 6.25, 0.7*0.7*0.7*0.7);
   shum6=PerlinNoise(x*scale, y*scale, width, seed, 3.125, 0.7*0.7*0.7*0.7*0.7);
   shum7=PerlinNoise(x*scale, y*scale, width, seed, 1.56, 0.7*0.7*0.7*0.7*0.7*0.7);
```

Битно је да напоменемо и да са повећавањем аплитуде толико да нам вредности излазе из интервала од 0 до 255 можемо да добијамо сјајне текстуре користећи Перлинов шум. Па ћемо тако за исте параметре као и у почетном коду уз измену да амплитуда узима почетну вредност 270 добити следећу слику:

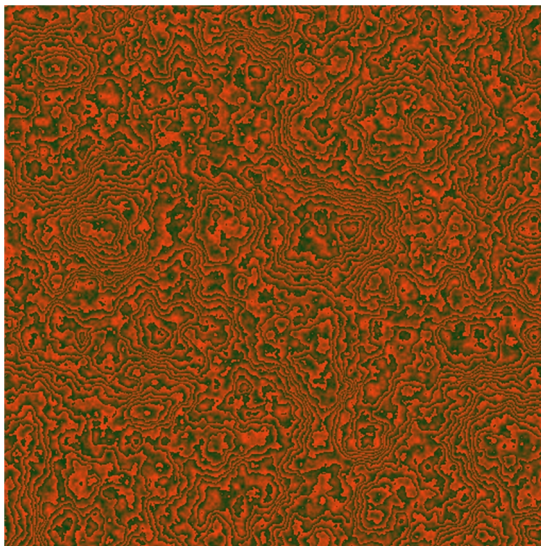


изменом боја пиксела:

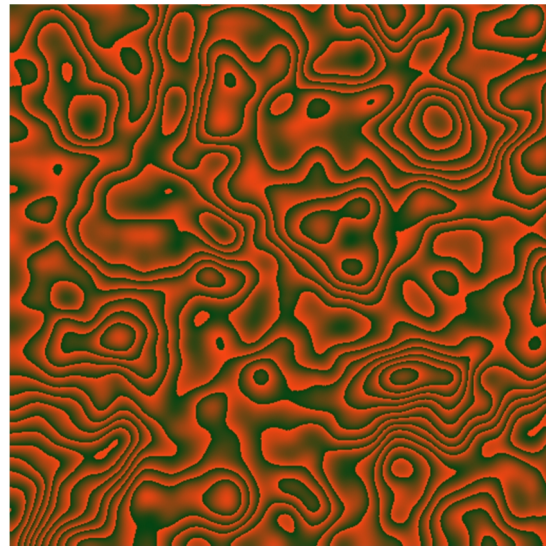
```
int k=(int)shum1+(int)(shum2*0.5)+(int)(shum3*0.25)+(int)(shum4*0.125)+(int)
      (shum5*0.0625)+(int)(shum6*0.03125)+(int)(shum7*0.0156);
Podaci[x][y][1]=69;
Podaci[x][y][2]=19;
Podaci[x][y][3]=k;
```

5 Алгоритми за креирање терена

и узимајући за амплитуду вредност од 4000 и 1500 у наредна два примера добијамо следеће слике које подсећају на текстуре дрвета:



$H = 0.5$

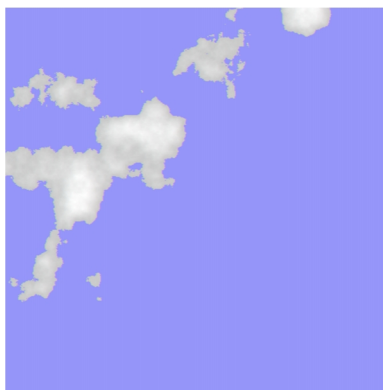
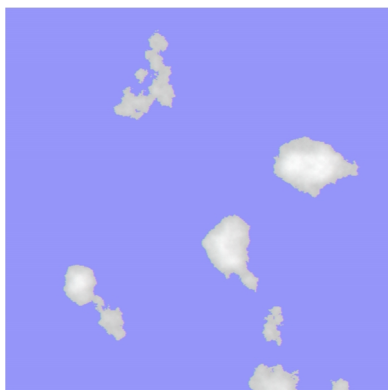


$H = 0.8$

Сличне слике могу да се користе и као текстуре и као ивице и границе различитих објеката због своје случајности.

Модификацијом Перлиновог шума можемо да добијемо слике облака. Једино што треба да урадимо је да одредимо висину и да све вредности испод ње обојимо плавом бојом. Модификација у коду је следећа:

```
if (k<200){ //odabrana je vrednost 200
//vrednosti 150/250/150 daju nijansu plave koja mi se najvise svidela
    Podaci[x][y][1]=150;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=150;}
else{
    Podaci[x][y][1]=k;
    Podaci[x][y][2]=k;
    Podaci[x][y][3]=k;}
```



Слика. Три текстуре облака креиране од Перлиновог шума

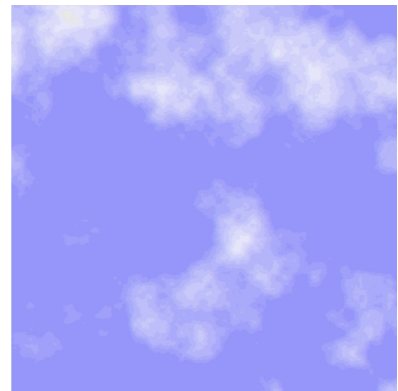
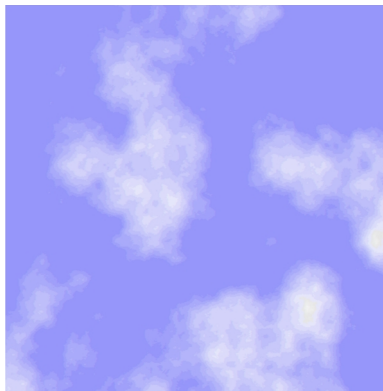
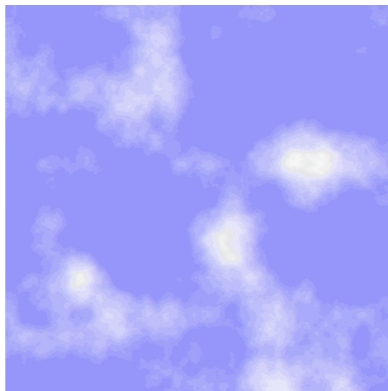
5 Алгоритми за креирање терена

Што је висина коју узимамо за границу мања, већа је површина коју на слици заузимају облаци а мања површина неба и обрнуто. Сlike изгледају реалистичније ако се примени филтер за замагљење или осветле ивице облака али са тиме се нећемо бавити. Направићемо благи прелаз са бојама на следећи начин:

```
if (k<150)
{
    Podaci[x][y][1]=150;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=150;
}
else if (k<160)
{
    Podaci[x][y][1]=160;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=160;
}
else if (k<170)
{
    Podaci[x][y][1]=170;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=170;
}
else if (k<180)
{
    Podaci[x][y][1]=180;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=180;
}
else if (k<190)
{
    Podaci[x][y][1]=190;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=190;
}
else if (k<200)
{
    Podaci[x][y][1]=200;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=200;
}
else if (k<210)
{
    Podaci[x][y][1]=210;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=210;
}
else if (k<220)
{
    Podaci[x][y][1]=220;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=220;
}
else if (k<230)
```


5 Алгоритми за креирање терена

```
{
    Podaci[x][y][1]=230;
    Podaci[x][y][2]=250;
    Podaci[x][y][3]=230;
}
else
{
    Podaci[x][y][1]=k;
    Podaci[x][y][2]=k;
    Podaci[x][y][3]=k;
}
```



И добијамо потпуно реалистичне облаке генерисане рачунаром.

За разлику од heightmap-а које сам морао самостално да испрограмирам, за креирање терена од већ постојећих heightmap-а постоји пуно доступних кодова на интернету. Ми ћемо користити следећи код који сам пронашао. Код је у малој мери измењен и скраћен за делове који нам нису неопходни. Цео код, заједно са класама, може да се прекопира у .cpp фајл. Једина ствар на коју треба да обратимо пажњу је да код ради за heightmap-е мапе bitmap формата и да морамо да унесемо назив тог фајла у наш код како би могао да га пронађе.

```
#include <iostream>
#include <stdlib.h>
#include <glut.h>
#include <assert.h>
#include <fstream>
#include <math.h>
using namespace std;
//definise se klasa za vektore. Definisani su sabiranje, oduzimanje,
//mnozenje i deljenje vektora, skalarni i vektorski proizvod i normalizacija vektora
#ifndef VEC3F_H_INCLUDED
#define VEC3F_H_INCLUDED
class Vec3f {
private:
    float v[3];
public:
    Vec3f();
    Vec3f(float x, float y, float z); //izgled konstruktora
    float &operator[](int index);
```

5 Алгоритми за креирање терена

```
float operator[](int index) const; //pristup individualnim elementima
                                //vektora koristeći zagrade
Vec3f operator*(float scale) const; //množenje vektora
Vec3f operator/(float scale) const; //deljenje vektora
Vec3f operator+(const Vec3f &other) const; //sabiranje vektora
Vec3f operator-(const Vec3f &other) const; //oduzimanje vektora
Vec3f operator-() const;
const Vec3f &operator*=(float scale);
const Vec3f &operator/=(float scale);
const Vec3f &operator+=(const Vec3f &other);
const Vec3f &operator-=(const Vec3f &other);
//skalarni i vektorski proizvod, normalizacija, intenzitet
float magnitude() const;
float magnitudeSquared() const;
Vec3f normalize() const;
float dot(const Vec3f &other) const;
Vec3f cross(const Vec3f &other) const;};

Vec3f operator*(float scale, const Vec3f &v);
std::ostream &operator<<(std::ostream &output, const Vec3f &v);
#endif

Vec3f::Vec3f() {}
Vec3f::Vec3f(float x, float y, float z) {
    v[0] = x;
    v[1] = y;
    v[2] = z;}

float &Vec3f::operator[](int index) {
    return v[index];}

float Vec3f::operator[](int index) const {
    return v[index];}

Vec3f Vec3f::operator*(float scale) const {
    return Vec3f(v[0] * scale, v[1] * scale, v[2] * scale);}

Vec3f Vec3f::operator/(float scale) const {
    return Vec3f(v[0] / scale, v[1] / scale, v[2] / scale);}

Vec3f Vec3f::operator+(const Vec3f &other) const {
    return Vec3f(v[0] + other.v[0], v[1] + other.v[1], v[2] + other.v[2]);}

Vec3f Vec3f::operator-(const Vec3f &other) const {
    return Vec3f(v[0] - other.v[0], v[1] - other.v[1], v[2] - other.v[2]);}

Vec3f Vec3f::operator-() const {
    return Vec3f(-v[0], -v[1], -v[2]);}

const Vec3f &Vec3f::operator*=(float scale) {
    v[0] *= scale;
    v[1] *= scale;
    v[2] *= scale;
    return *this;}

const Vec3f &Vec3f::operator/=(float scale) {
    v[0] /= scale;
    v[1] /= scale;
    v[2] /= scale;
    return *this;}

const Vec3f &Vec3f::operator+=(const Vec3f &other) {
    v[0] += other.v[0];
    v[1] += other.v[1];
    v[2] += other.v[2];
    return *this;}

const Vec3f &Vec3f::operator-=(const Vec3f &other) {
    v[0] -= other.v[0];
    v[1] -= other.v[1];
```

5 Алгоритми за креирање терена

```
        v[2] -= other.v[2];
        return *this;}
float Vec3f::magnitude() const {
    return sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);}
float Vec3f::magnitudeSquared() const {
    return v[0] * v[0] + v[1] * v[1] + v[2] * v[2];}
Vec3f Vec3f::normalize() const {
    float m = sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);
    return Vec3f(v[0] / m, v[1] / m, v[2] / m);}
float Vec3f::dot(const Vec3f &other) const {
    return v[0] * other.v[0] + v[1] * other.v[1] + v[2] * other.v[2];}
Vec3f Vec3f::cross(const Vec3f &other) const {
    return Vec3f(v[1] * other.v[2] - v[2] * other.v[1],
                v[2] * other.v[0] - v[0] * other.v[2],
                v[0] * other.v[1] - v[1] * other.v[0]);}
Vec3f operator*(float scale, const Vec3f &v) {
    return v * scale;}
ostream &operator<<(ostream &output, const Vec3f &v) {
    cout << '(' << v[0] << ", " << v[1] << ", " << v[2] << ')';
    return output;}
#ifdef IMAGE_LOADER_H_INCLUDED
#define IMAGE_LOADER_H_INCLUDED
//predstavlja sliku
class Image {
public:
    Image(char* ps, int w, int h);
    ~Image();
    char* pixels;
    int width;
    int height;};
//cita bitmap sliku iz fajla
Image* loadBMP(const char* filename);
#endif
using namespace std;
Image::Image(char* ps, int w, int h) : pixels(ps), width(w), height(h) {}
Image::~Image() {
    delete[] pixels;}
namespace {
    int toInt(const char* bytes) {
        return (int)((((unsigned char)bytes[3] << 24) |
                    ((unsigned char)bytes[2] << 16) |
                    ((unsigned char)bytes[1] << 8) |
                    (unsigned char)bytes[0]));}
    short toShort(const char* bytes) {
        return (short)((((unsigned char)bytes[1] << 8) |
                    (unsigned char)bytes[0]));}
    int readInt(istream &input) {
        char buffer[4];
        input.read(buffer, 4);
        return toInt(buffer);}
    short readShort(istream &input) {
        char buffer[2];
        input.read(buffer, 2);
        return toShort(buffer);}
    template<class T>
    class auto_array {
    private:
```

5 Алгоритми за креирање терена

```
T* array;
mutable bool isReleased;
public:
    explicit auto_array(T* array_ = NULL) :
        array(array_), isReleased(false) {}
    auto_array(const auto_array<T> &aarray) {
        array = aarray.array;
        isReleased = aarray.isReleased;
        aarray.isReleased = true;}
    ~auto_array() {
        if (!isReleased && array != NULL) {
            delete[] array;}}
    T* get() const {
        return array;}
    T &operator*() const {
        return *array;}
    void operator=(const auto_array<T> &aarray) {
        if (!isReleased && array != NULL) {
            delete[] array;}
        array = aarray.array;
        isReleased = aarray.isReleased;
        aarray.isReleased = true;}
    T* operator->() const {
        return array;}
    T* release() {
        isReleased = true;
        return array;}
    void reset(T* array_ = NULL) {
        if (!isReleased && array != NULL) {
            delete[] array;}
        array = array_;}
    T* operator+(int i) {
        return array + i;}
    T &operator[](int i) {
        return array[i];}};

Image* loadBMP(const char* filename) {
    ifstream input;
    input.open(filename, ifstream::binary);
    assert(!input.fail() || !"Could not find file");
    char buffer[2];
    input.read(buffer, 2);
    assert(buffer[0] == 'B' && buffer[1] == 'M' || !"Not a bitmap file");
    input.ignore(8);
    int dataOffset = readInt(input);
    //cita zaglavlje
    int headerSize = readInt(input);
    int width;
    int height;

    width = readInt(input);
    height = readInt(input);

    //iscitava podatke
    int bytesPerRow = ((width * 3 + 3) / 4) * 4 - (width * 3 % 4);
    int size = bytesPerRow * height;
    auto_array<char> pixels(new char[size]);
    input.seekg(dataOffset, ios_base::beg);
    input.read(pixels.get(), size);
    auto_array<char> pixels2(new char[width * height * 3]);
```


5 Алгоритми за креирање терена

```
for(int y = 0; y < height; y++) {
    for(int x = 0; x < width; x++) {
        for(int c = 0; c < 3; c++) {
            pixels2[3 * (width * y + x) + c] =
                pixels[bytesPerRow * y + 3 * x + (2 - c)];}}
input.close();
return new Image(pixels2.release(), width, height);}
//Predstavlja teren, cuvajuci skup visina i normala u 2D lokacijama
class Terrain {
private:
    int w; //sirina (broj tacaka mreze na X osi)
    int l; //duzina (broj tacaka mreze na Z osi)
    float** hs; //visine (visina terena u svakoj tacki mreze)
    Vec3f** normals; //cuva normale u svakoj tacki
    bool computedNormals; //prati da li smo izracunali normale
public:
    Terrain(int w2, int l2) {
        w = w2;
        l = l2;
        hs = new float*[l];
        for(int i = 0; i < l; i++) {
            hs[i] = new float[w];}
        normals = new Vec3f*[l];
        for(int i = 0; i < l; i++) {
            normals[i] = new Vec3f[w];}
        computedNormals = false;}
    ~Terrain() {
        for(int i = 0; i < l; i++) {
            delete[] hs[i];}
        delete[] hs;
        for(int i = 0; i < l; i++) {
            delete[] normals[i];}
        delete[] normals;}
    int width() {
        return w;}
    int length() {
        return l;}
    //postavlja visinu u tacki (x, z) na y
    void setHeight(int x, int z, float y) {
        hs[z][x] = y;
        computedNormals = false;}
    //Vraca visinu u tacki (x, z)
    float getHeight(int x, int z) {
        return hs[z][x];}
    //izracunava normale ako nisu jos uvek izracunate
    void computeNormals() {
        if (computedNormals) {
            return;}
        //izracunava normale
        Vec3f** normals2 = new Vec3f*[l];
        for(int i = 0; i < l; i++) {
            normals2[i] = new Vec3f[w];}
        for(int z = 0; z < l; z++) {
            for(int x = 0; x < w; x++) {
                Vec3f sum(0.0f, 0.0f, 0.0f);
                Vec3f out;
                if (z > 0) {
```

5 Алгоритми за креирање терена

```
        out = Vec3f(0.0f, hs[z - 1][x] - hs[z][x], -
                    1.0f);}
    Vec3f in;
    if (z < 1 - 1) {
        in = Vec3f(0.0f, hs[z + 1][x] - hs[z][x],
                    1.0f);}
    Vec3f left;
    if (x > 0) {
        left = Vec3f(-1.0f, hs[z][x - 1] - hs[z][x],
                     0.0f);}
    Vec3f right;
    if (x < w - 1) {
        right = Vec3f(1.0f, hs[z][x + 1] - hs[z][x],
                     0.0f);}
    if (x > 0 && z > 0) {
        sum += out.cross(left).normalize();}
    if (x > 0 && z < 1 - 1) {
        sum += left.cross(in).normalize();}
    if (x < w - 1 && z < 1 - 1) {
        sum += in.cross(right).normalize();}
    if (x < w - 1 && z > 0) {
        sum += right.cross(out).normalize();}
    normals2[z][x] = sum;}}
const float FALLOUT_RATIO = 0.5f;
for(int z = 0; z < 1; z++) {
    for(int x = 0; x < w; x++) {
        Vec3f sum = normals2[z][x];
        if (x > 0) {
            sum += normals2[z][x - 1] * FALLOUT_RATIO;}
        if (x < w - 1) {
            sum += normals2[z][x + 1] * FALLOUT_RATIO;}
        if (z > 0) {
            sum += normals2[z - 1][x] * FALLOUT_RATIO;}
        if (z < 1 - 1) {
            sum += normals2[z + 1][x] * FALLOUT_RATIO;}
        if (sum.magnitude() == 0) {
            sum = Vec3f(0.0f, 1.0f, 0.0f);}
        normals[z][x] = sum;}}
for(int i = 0; i < 1; i++) {
    delete[] normals2[i];}
delete[] normals2;
computedNormals = true;}
//Vraca normalu u tacki (x, z)
Vec3f getNormal(int x, int z) {
    if (!computedNormals) {
        computeNormals();}
    return normals[z][x];}}
//Ucitavamo teren od heightmapa. Visine terena se kreću od -height / 2 do height / 2.
Terrain* loadTerrain(const char* filename, float height) {
    Image* image = loadBMP(filename);
    Terrain* t = new Terrain(image->width, image->height);
    for(int y = 0; y < image->height; y++) {
        for(int x = 0; x < image->width; x++) {
            unsigned char color =
                (unsigned char)image->pixels[3 * (y * image->width + x)];
            float h = height * ((color / 255.0f) - 0.5f);
            t->setHeight(x, y, h);}}
```

5 Алгоритми за креирање терена

```
        delete image;
        t->computeNormals();
        return t;};
float _angle = 60.0f;
Terrain* _terrain;
void cleanup() {
    delete _terrain;
}
void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glShadeModel(GL_SMOOTH);}
void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (double)w / (double)h, 1.0, 200.0);}
void drawScene() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -10.0f);
    glRotatef(30.0f, 1.0f, 0.0f, 0.0f);
    glRotatef(-_angle, 0.0f, 1.0f, 0.0f);
    GLfloat ambientColor[] = {0.4f, 0.4f, 0.4f, 1.0f};
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientColor);
    GLfloat lightColor0[] = {0.6f, 0.6f, 0.6f, 1.0f};
    GLfloat lightPos0[] = {-0.5f, 0.8f, 0.1f, 0.0f};
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightColor0);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos0);
    float scale = 5.0f / max(_terrain->width() - 1, _terrain->length() - 1);
    glScalef(scale, scale, scale);
    glTranslatef(-(float)(_terrain->width() - 1) / 2, 0.0f, -(float)(_terrain->length() - 1) / 2);
    glColor3f(0.3f, 0.9f, 0.0f);
    for(int z = 0; z < _terrain->length() - 1; z++) {
        //Crta trouglove od svaka tri uzastopna temena
        glBegin(GL_TRIANGLE_STRIP);
        for(int x = 0; x < _terrain->width(); x++) {
            Vec3f normal = _terrain->getNormal(x, z);
            glNormal3f(normal[0], normal[1], normal[2]);
            glVertex3f(x, _terrain->getHeight(x, z), z);
            normal = _terrain->getNormal(x, z + 1);
            glNormal3f(normal[0], normal[1], normal[2]);
            glVertex3f(x, _terrain->getHeight(x, z + 1), z + 1);}
        glEnd();}
    glutSwapBuffers();}
void update(int value) {
    _angle += 1.0f;
    if (_angle > 360) {
        _angle -= 360;}
    glutPostRedisplay();
    glutTimerFunc(25, update, 0);}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
```

5 Алгоритми за креирање терена

```
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize(400, 400);
glutCreateWindow("Teren");
initRendering();
_terrain = loadTerrain("heightmap.bmp", 20); //ime fajla heightmape
glutDisplayFunc(drawScene);
glutReshapeFunc(handleResize);
glutTimerFunc(25, update, 0);
glutMainLoop();
return 0;}
```

Сада ћемо овим кодом да испрограмирамо терен од heightmap-е Перлиног шума који смо добили раније:

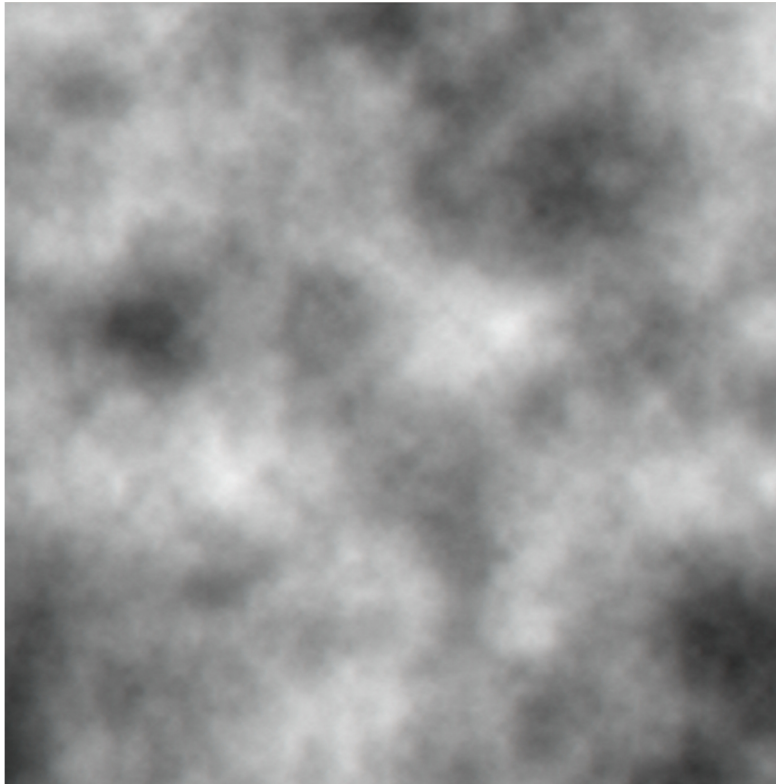


График. Перлинов шум креиран из програма за Перлинов шум не мењајући параметре

Прво што треба да урадимо је да умањимо нашу слику. Програм одлично ради за димензије слике мање од 100×100 , па ћемо је смањити негде у том интервалу.

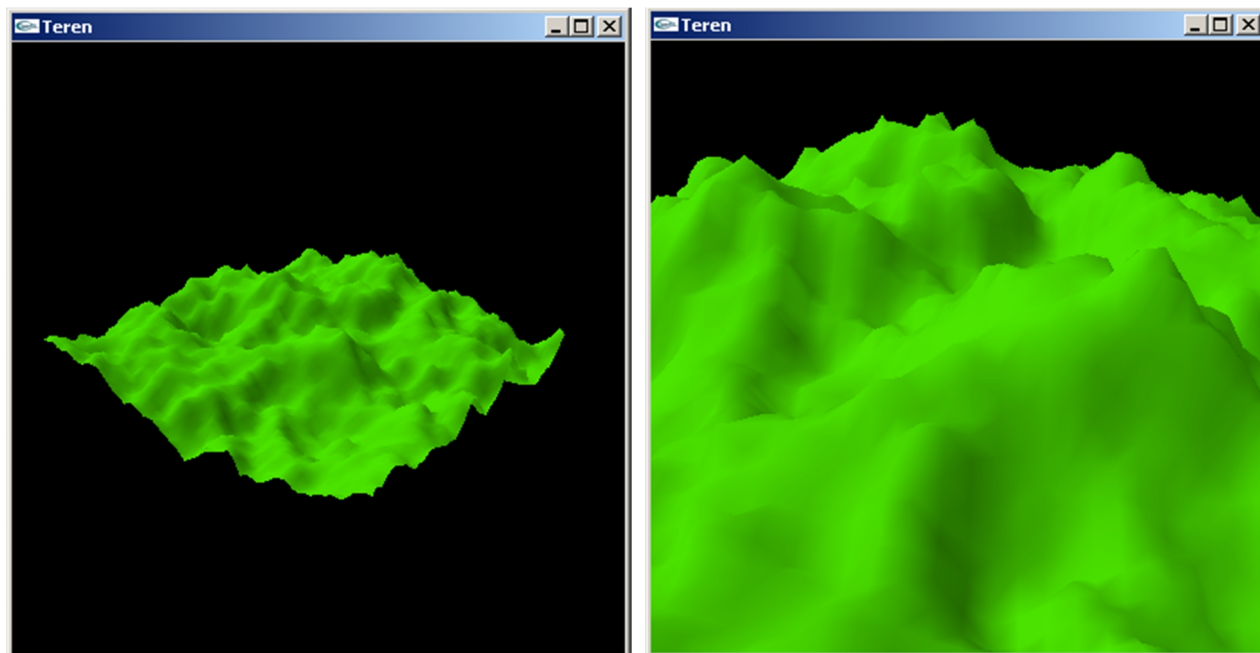
Ево је слика у димензијама 64×64 :



График. Умањена слика претходно креираног Перлиновог шума

Затим слику назовемо "Perlin.bmp", а након тога у нашем коду уместо "heightmap.bmp" унесемо "Perlin.bmp" и добијамо следећи терен:

5 Алгоритми за креирање терена



Слика. Терен креиран од Перлиновог шума у Visual C++, и приказ истог увећаног терена

Примена Перлиновог шума

Перлинов шум има веома широку употребу:

Контролисање виртуелних објеката: Перлинов шум се користи за константно подешавамо позиције виртуелног живог бића, у игрици на пример да би изгледао што живље.

Цртање линија скице: Компјутерске линије су увек равне, што их чини неприродним и непријатељским. Можемо да користимо Перлинов шум да бисмо увели неправилност у алгоритам цртања линија да би оне изгледале као да су нацртане руком.

Пејзажи: Ово је перфектна примена дводимензионалног Перлиног шума. Не морамо да складиштимо пејзаж нигде у меморију, висина сваке тачке у пејзажу може бити израчуната веома лако. Штавише, земљиште се простире у бесконачност (скоро), и може бити израчунато до сићушних детаља.

Облаци: Креирање облака је веома једноставно користећи Перлинов шум; неопходно је само да ограничимо висину на нашој слици Перлиновог шума и да све вредности испод те обојимо плавом бојом.

Генерисање текстура: Различите врсте текстура могу бити генерисане Перлиновим шумом. Текстури генерисане могу да иду у бесконачност без понављања, што их чини много пријатније за гледање него у поплочане текстури које се понављају.

Тродимензионални облаци: Можемо Перлинов шум да користимо и за креирање тродимензионалних облака.

5 Алгоритми за креирање терена

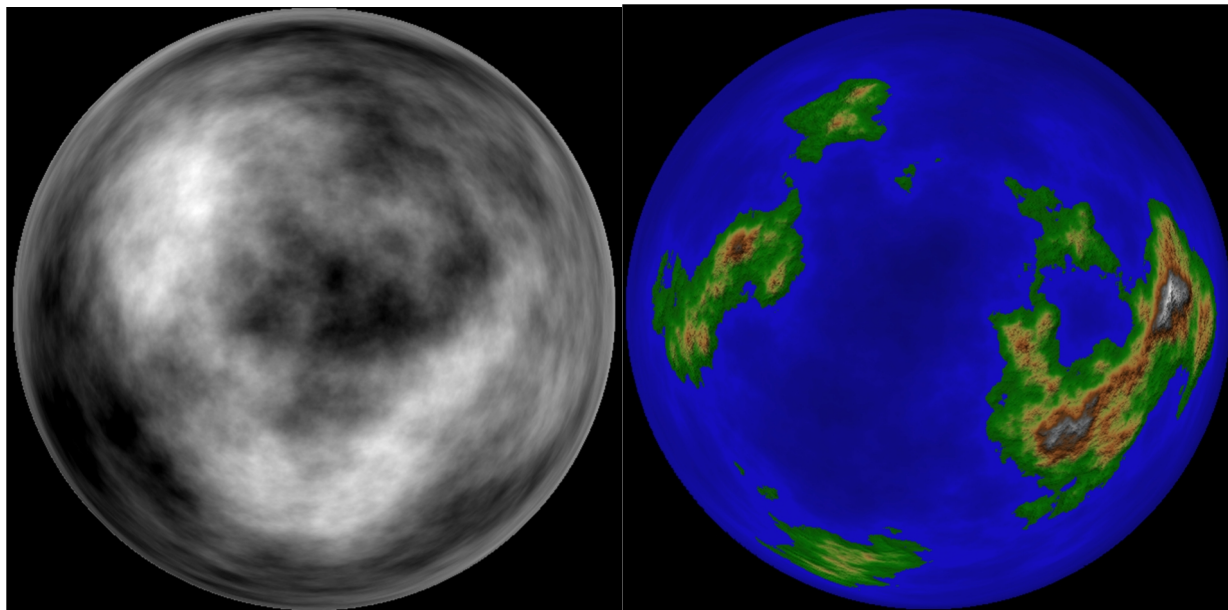
Анимирани облаци: Можемо да произведемо анимирани дводимензионалне облаци са тродимензионалним Перлиновим шумом, ако претпоставимо да је једна димензија време.

Анимирани тродимензионалне текстуре и облаци: Одласком у „више“ димензије, можемо лако да произведемо анимирани облаци и чврсте текстуре. Додатна димензија је време.

Генерисање текстура са Перлиновим шумом

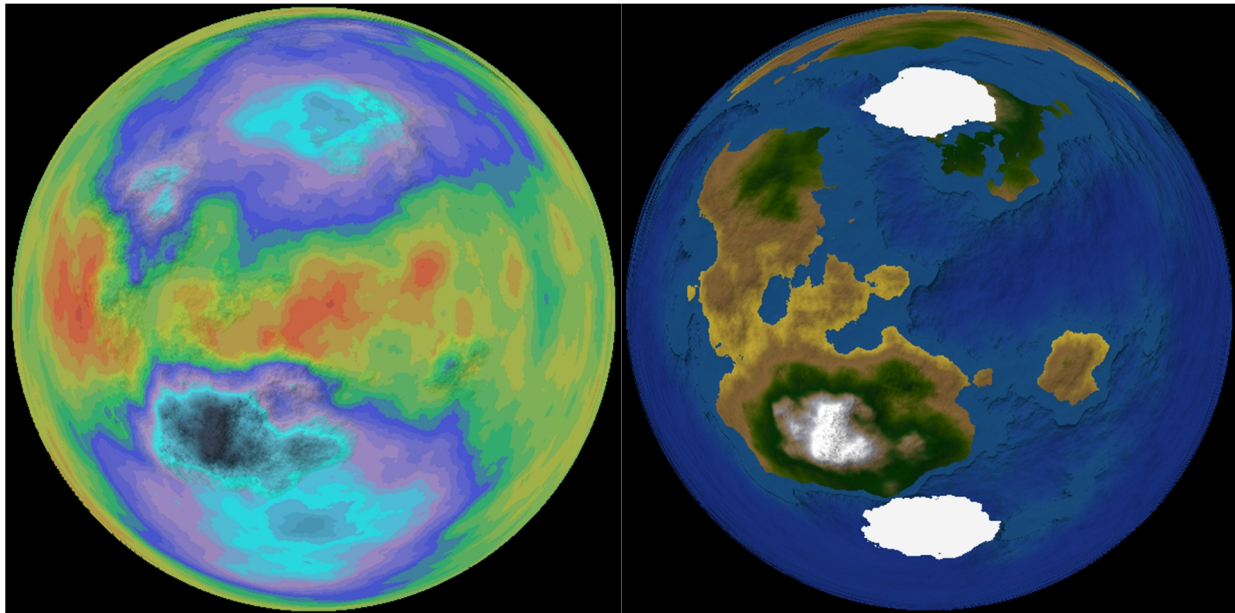
Перлинов шум је фантастичан за генерисање текстура. Можемо правити текстуре које су бесконачно велике, али не користе скоро уопште меморије. Можемо креирати слике кликера, дрва, скоро било шта ако би пробали. Можемо такође и да креирамо тродимензионалне текстуре. Можемо размишљати о овоме као о чврстом блоку материјала од којег ћемо да креирамо објекат. Ово нам омогућава да креирамо текстуре које могу бити примењене на сваки објекат. Може да захтева много имагинације, размишљања и експериментисања да би добили текстуре које изгледају добро, али резултати могу бити заиста импресивни.

Мењањем параметара Перлиновог шума, комбинујући различите Перлинове шумове, мењањем боја и комбиновањем истог са различитим функцијама можемо да добијемо разне текстуре. Ево их неки примери добијени примењивањем тих текстура на сфере креирајући тако интересантен облике кликера или планета:

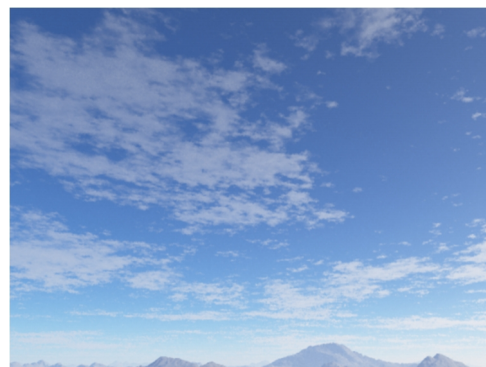
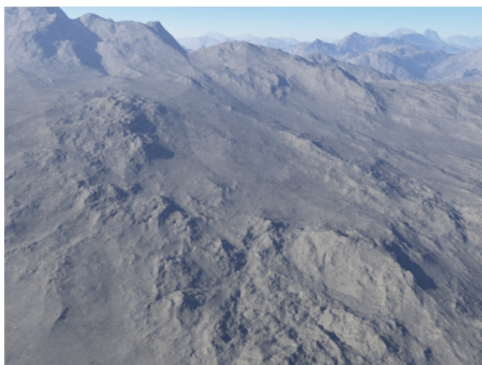


Слика. Тесктура Перлиновог шума примењена на сферу. На слици лево видимо неизмењени Перлинов шум док га на десној слици користимо како бисмо креирали мапу висина планете

5 Алгоритми за креирање терена

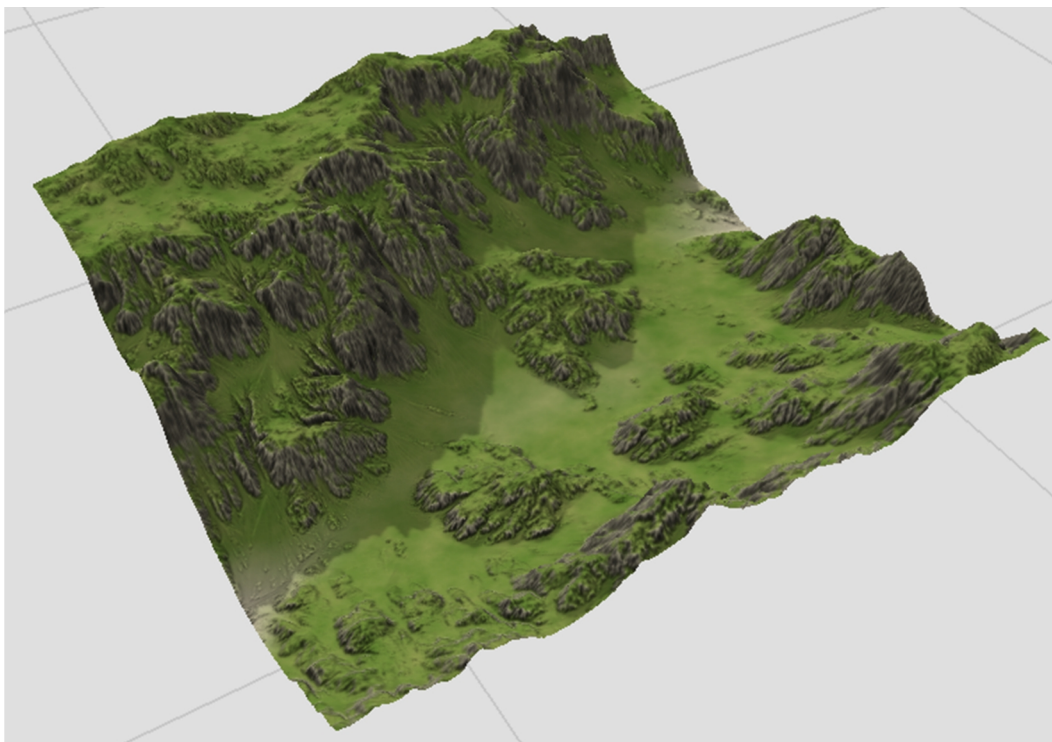


Слика. Текстура Перлиновог шума примењена на сферу. На слици лево користимо Перлинов шум како бисмо приказали температурну мапу планете а на десној слици како бисмо приказали рељеф планете

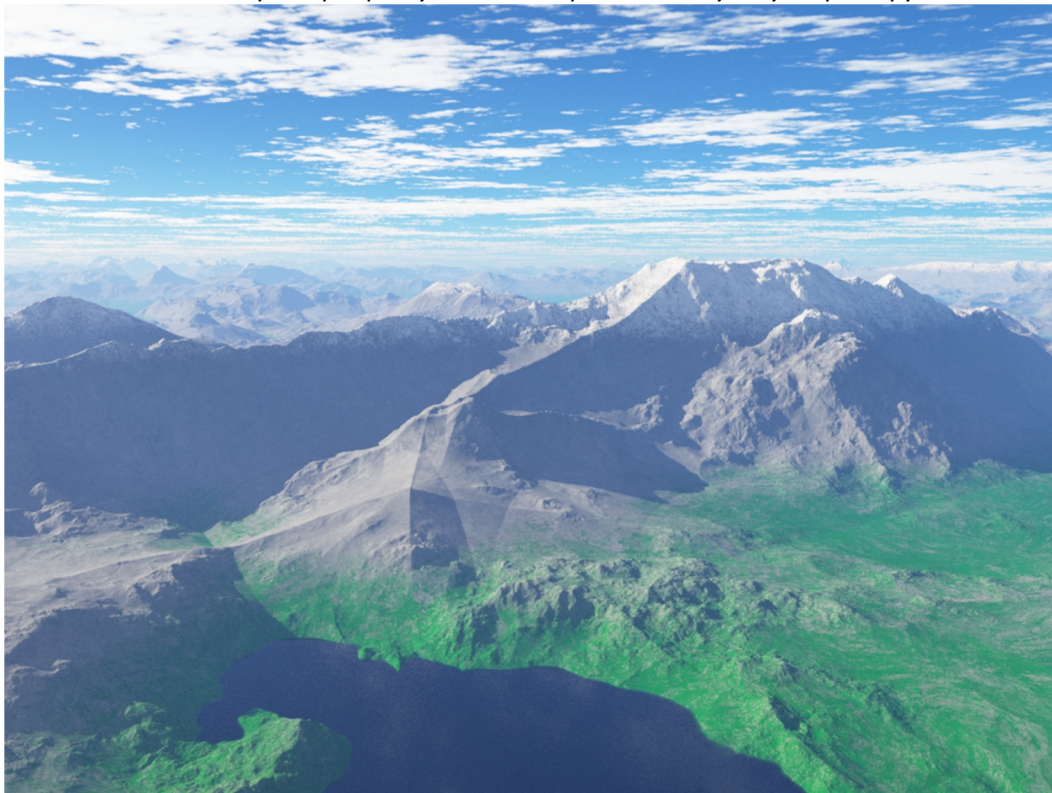


Слика. 2 примера терена и 2 примера облака генерисаних у софтверу Terragen који се у потпуности заснивају на Перлиновом шуму

5 Алгоритми за креирање терена



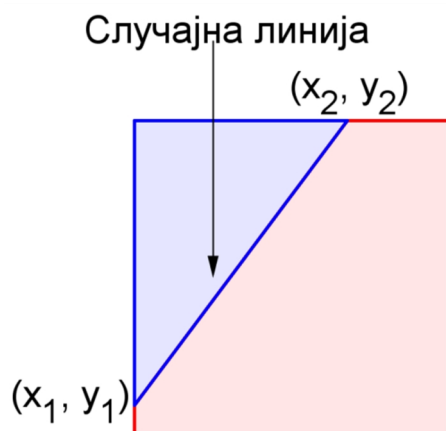
Слика. Реалистични терен креиран уз помоћ Перлиновог шума у софтверу World Machine



Слика. Пејзаж креиран у Terragen-у. Креирање терена и облака се у потпуности заснива на Перлиновом шуму

Fault formation алгоритам

Овај алгоритам је веома једноставан, поприлично брз, и ако му резултат није добар као Перлинов шум често је задовољавајућ. Да бисмо започели алгоритам морамо да имамо heightmap-у чије су све висине једнаке нули. Затим правимо случајну линију која дели терен на два дела. Тачкама са једне стране ћемо повећати висину, а тачкама са друге смањити. Следећа слика илуструје прву итерацију овог алгоритма.



Слика. Случајна линија која раздваја податке heightmap-е на оне чију виисину повећавамо и оне чију смањујемо

Први проблем је како да креирамо случајну линију која дели терен. То ћемо урадити тако што ћемо наћи две случајне тачке (x_1, y_1) и (x_2, y_2) од којих ћемо да направимо вектор $v_1 = (x_2 - x_1, y_2 - y_1)$. Затим ћемо за сваку тачку терена (x, y) да израчунамо још један вектор $v = (x - x_1, y - y_1)$. Након тога y компонента векторског производа се израчунава и она ће нам дати правац по којем увећавамо и смањујемо.

Овај алгоритам има много приступа. Можемо да почнемо са висинама које су једнаке нула па да повећавамо у свакој итерацији једну страну од случајне линије. Можемо једну страну да смањујемо а другу да повећавамо за константе вредности, случајне вредности или неком функцијом. Ми ћемо приступити овом алгоритму тако што ћемо свим тачкама доделити вредност 127 која представља сиву боју која је средина интервала grayscale слике са распоном боја од 0 до 255, а затим ћемо вредности умањивати и смањивати за 10. Овако изгледа алгоритам са првих неколико итерација:

5 Алгоритми за креирање терена



Почетна слика



Једна итерација



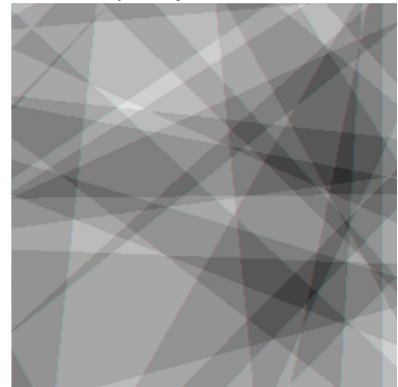
Две итерације



Три итерације



Четири итерације



Педесет итерација

Сада ћемо показати како изгледа наш програм за 300 итерација у којем увећавамо и смањујемо вредности пиксела за вредност 2 :

```
#include <math.h>
#define ImageHeight 512
#define ImageWidth 512
GLubyte heightmap[ImageWidth][ImageHeight][3];
static GLint height;

void FaultFormation(void)
{
    int x, y, i, j, k;
    for (i=0; i<ImageWidth; i++)
    {
        for (j=0; j<ImageHeight; j++)
        {
            //postavljamo pocetne vrednosti svih elemenata
            //heightmapa na 127, odnosno sivu boju
            heightmap[i][j][1]=127;
            heightmap[i][j][2]=127;
            heightmap[i][j][3]=127;
        }
    }
    //300 iteracija
    for (k=0; k<300; k++)
```

5 Алгоритми за креирање терена

```
{
    int h=ImageHeight;
    int w=ImageWidth;
    //koordinata x1,x2 i y1,y2 dobijamo kao slucajne vrednosti
    //iz intervala 0 do 255
    int x1=rand()*255;
    int x2=rand()*255;
    int y1=rand()*255;
    int y2=rand()*255;
    for (x=0; x<ImageWidth; x++)
    {
        for (y=0; y<ImageHeight; y++)
        {
            if ((x2-x1)*(y-y1)-(y2-y1)*(x-x1)>127.0){
                //uvecavamo vrednosti elemenata za 2
                heightmap[x][y][1]+=2;
                heightmap[x][y][2]+=2;
                heightmap[x][y][3]+=2;}
            else{
                //umanjujemo vrednosti elemenata za 2
                heightmap[x][y][1]-=2;
                heightmap[x][y][2]-=2;
                heightmap[x][y][3]-=2;}
        }
    }
}

void inicijalizacija(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glShadeModel(GL_SMOOTH);
    FaultFormation();
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glRasterPos2i(0, 0);
    glDrawPixels(ImageWidth, ImageHeight, GL_RGB, GL_UNSIGNED_BYTE, heightmap);
    glFlush();
}

void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    height=(GLint)h;
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

int main(int argc, char**argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(512, 512);
    glutInitWindowPosition(100, 100);
```

5 Алгоритми за креирање терена

```
glutCreateWindow("Fault formation");  
inicijalizacija();  
glutDisplayFunc(display);  
glutReshapeFunc(reshape);  
glutMainLoop();  
return 0;}
```

и добијамо следећу слику:

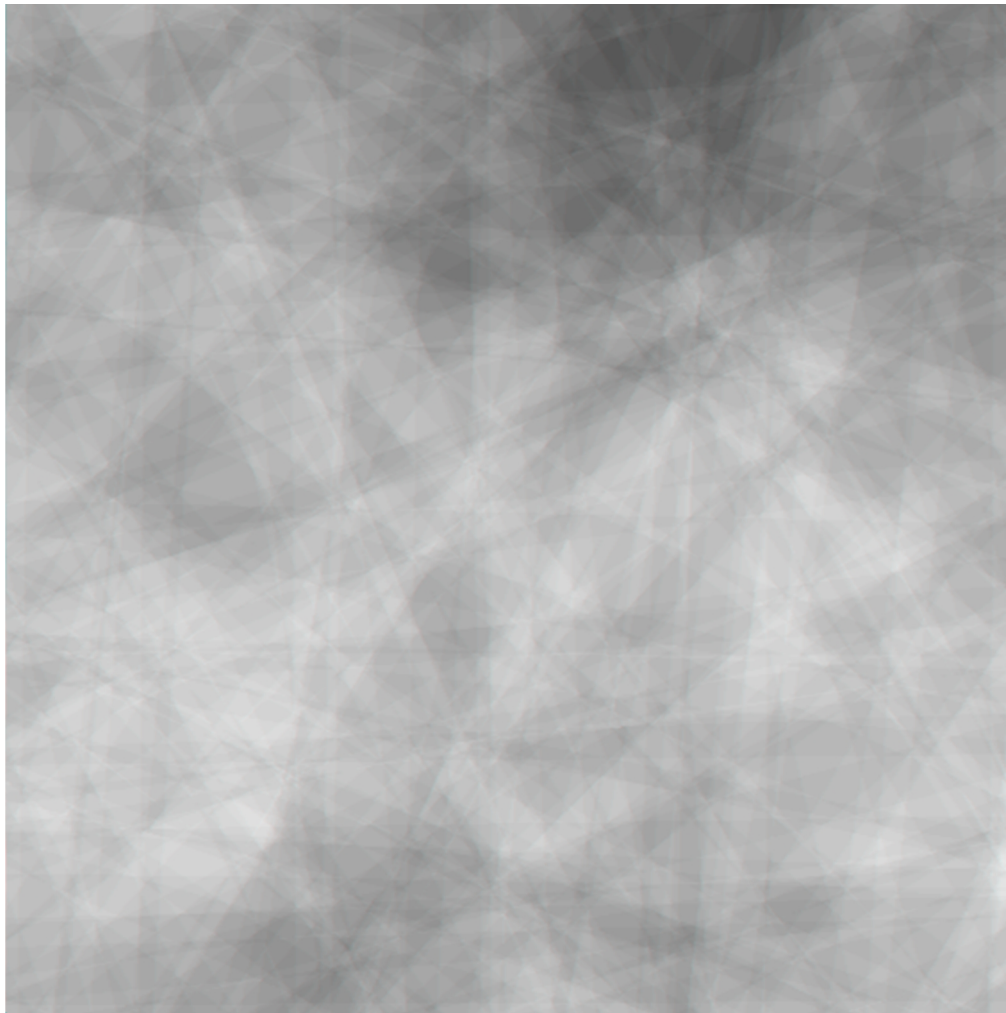


График. *Heightmap*-а креирана претходним кодом

Добијена слике не изгледа лоше, међутим приметно је лошија од Перлиновог шума, али повећањем броја итерација добијамо све бољу и бољу слику.

Умањена слика изгледа овако

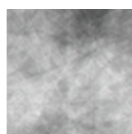
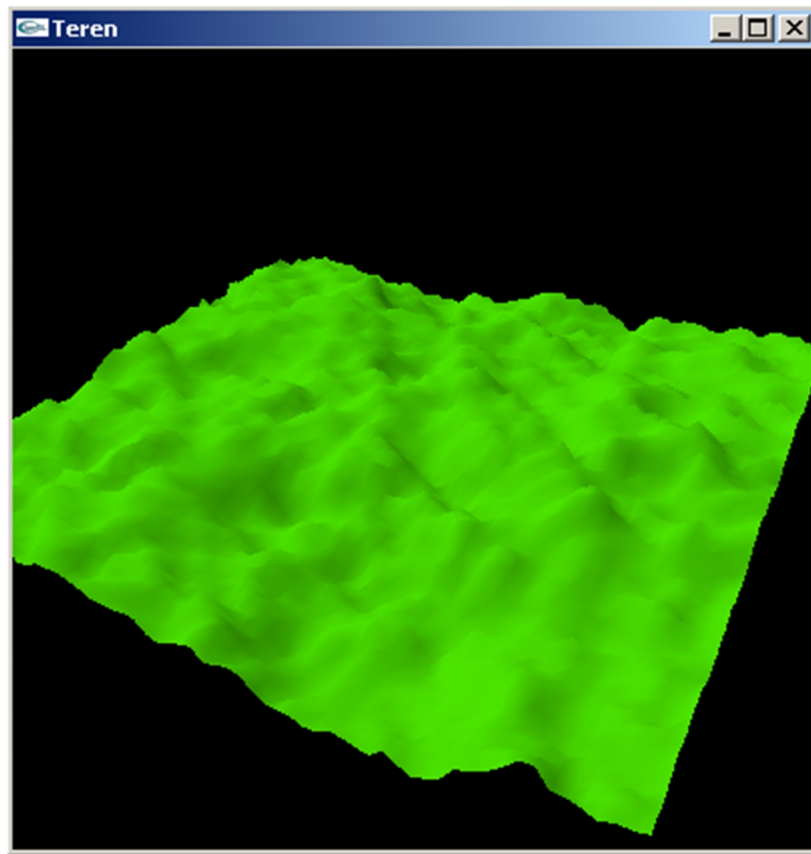


График. Умањена слика претходно креираног шума

5 Алгоритми за креирање терена

а добијени терен:



Слика. Терен генерисан *heightmap*-ом која је добијена *fault formation* алгоритмом

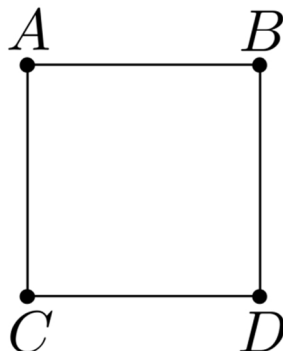
Diamond-square алгоритам

Fault formation је одличан за мале сцене састављене од малих брда (планина), али шта ако хоћемо нешто хаотичније од тога, нпр планинске венце?

Diamond-square је одговор. Овај алгоритам је такође познат и као плазма фрактал и midpoint displacement алгоритам. Лоша страна овог алгоритма је што димензије морају бити степена 2, што није случај код fault formation алгоритма где могу да буду било које димензије.

Суштински, све што радимо у овом алгоритму је да узмемо средњу тачку неке линије и преместимо је.

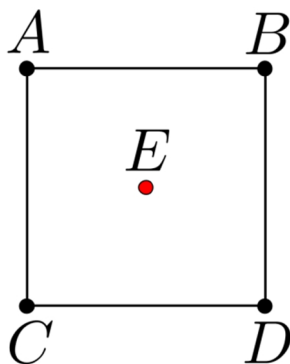
diamond-square алгоритам је метод за генерисање heightmap-а. Алгоритам је веома прост. Број тачака мреже расте са сваком итерацијом. У старту алгоритам почиње са мрежом 2×2 . Висине у четири угла су најчешће постављене на нулу или неку случајну вредност.



Први корак је да нађемо висину центра терена. Та вредност се израчунава као просек висина четири угла плус случајна вредност. Овај корак је познат као diamond корак.

$$E = (A + B + C + D) / 4 + \text{RAND}(d)$$

где је $\text{RAND}(d)$ случајна вредност из интервала $[-d, d]$. Вредност d представља максималну промену висине у тренутној итерацији.



5 Алгоритми за креирање терена

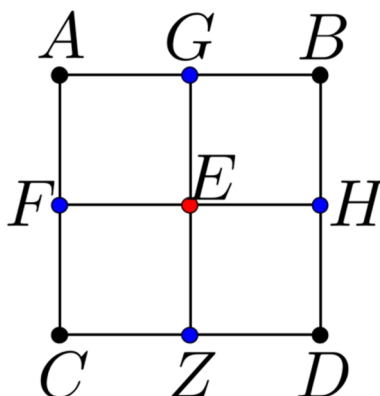
Затим следи square корак. Средње тачке све четири ивице (плаве тачке) се израчунавају као просек повезаних тачака плус случајна вредност.

$$F = (A + C + E + E) / 4 + \text{RAND}(d)$$

$$G = (A + B + E + E) / 4 + \text{RAND}(d)$$

$$H = (B + D + E + E) / 4 + \text{RAND}(d)$$

$$Z = (C + D + E + E) / 4 + \text{RAND}(d)$$



Ако оставимо $F = H$ доћи ће до понављања (симетричности), иако то у неким случајевима није лоше, ми хоћемо неправилности у нашем терену. Ово ћемо избећи тако што ћемо F израчунавати као:

$$F = (A + C + E) / 3 + \dots$$

уместо

$$F = (A + C + E + E) / 4 + \dots$$

Прва итерација је готова. Сада је неопходно помножити вредност d на следећи начин:

$$d *= \text{pow}(2, -r)$$

где је r константа храпавости са којом смо се већ сусрећали.

Сада имамо четири квадрата. Процес се понавља за сваки квадрат. Прво примењујемо diamond корак:

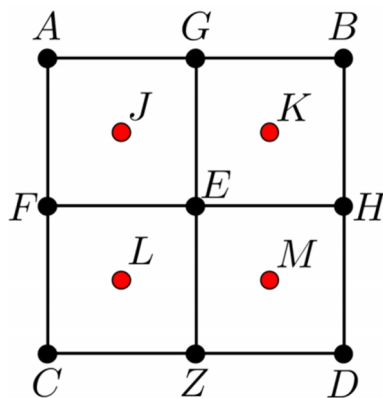
$$J = (A + G + F + E) / 4 + \text{RAND}(d)$$

$$K = (G + B + E + H) / 4 + \text{RAND}(d)$$

$$L = (F + E + C + Z) / 4 + \text{RAND}(d)$$

$$M = (E + H + I + D) / 4 + \text{RAND}(d)$$

5 Алгоритми за креирање терена



Након примењивања diamond корака на све квадрате, време је да применимо square корак. На следећој слици тачке првог квадрата су плаве тачке. У верзији која има понављања имамо:

$$N = (K + A + J + F) / 4 + RAND(d)$$

$$O = (L + A + G + J) / 4 + RAND(d)$$

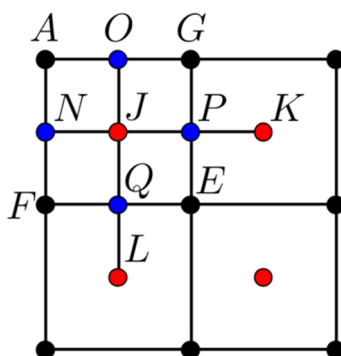
$$P = (J + G + K + E) / 4 + RAND(d)$$

$$Q = (F + J + E + L) / 4 + RAND(d)$$

А у верзији без понављања ирачунавање за N и O је другачије:

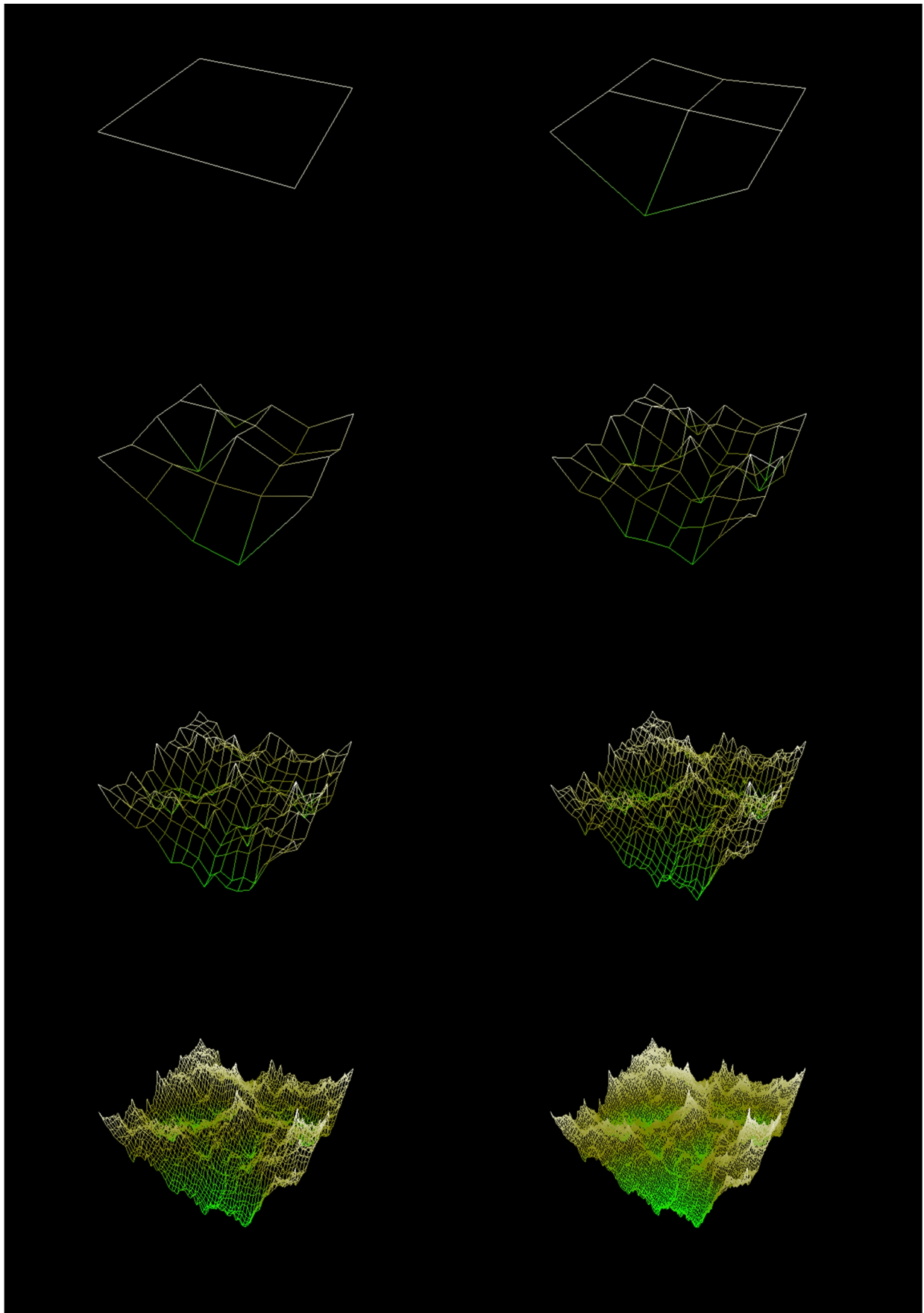
$$N = (A + J + F) / 3 + RAND(d)$$

$$O = (A + G + J) / 4 + RAND(d)$$



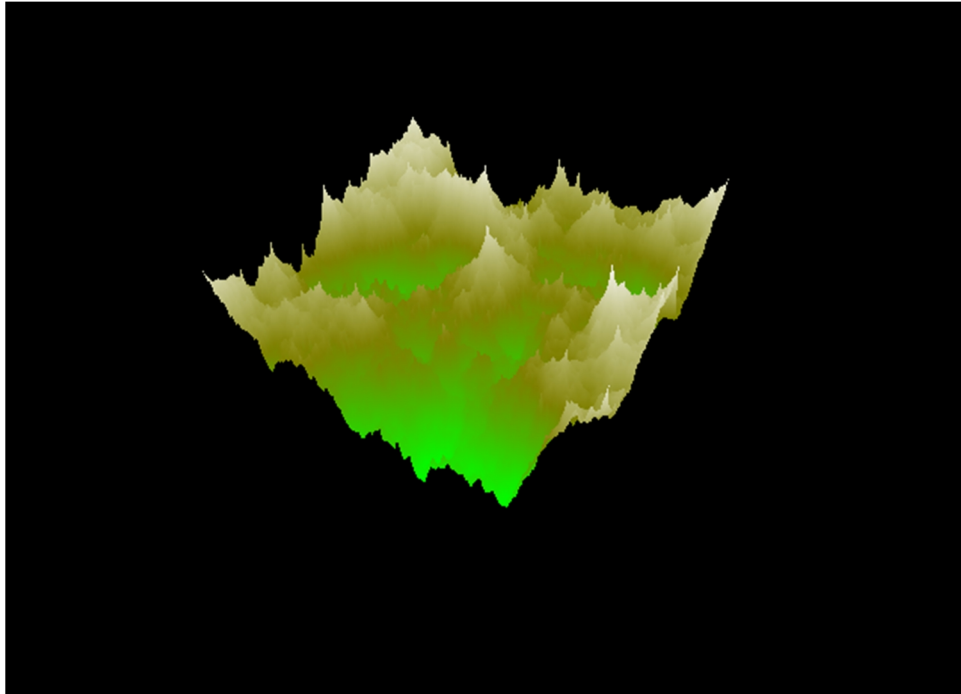
Приказаћемо рад алгоритма на следећој слици у 7 итерација:

5 Алгоритми за креирање терена

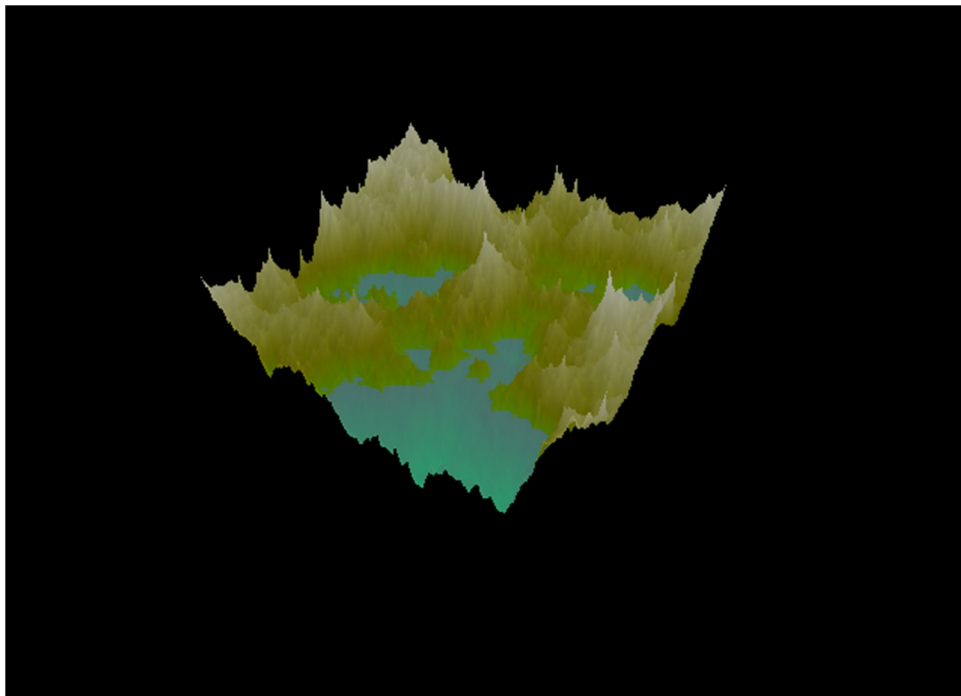


5 Алгоритми за креирање терена

Бојањем површине добијамо следећу слику пејзажа:



Увођењем осветљења и нивоа мора добијамо следећи реалистични пејзаж:



Закључак

Фрактална геометрија, иако млада научна дисциплина, представља моћан алат за дефинисање и моделирање разних природних појава. Системи итеративних функција, Линденмајерови системи и фракционо Брауново кретање представљају само један део ове дисциплине. Данас фрактали и фрактална геометрија имају огромну примену. Користе се у астрономији, предвиђању промена климе, мапирању DNK, гранању биљака, развијању облака, процесу ерозије, анализи саобраћаја, анализи можданих таласа, анализи откуцаја срца, анализи слика, анализи природних граница, анализи хемијских реакција, специјалним ефектима, игрицама, креирању шума, квантној теорији, симулацији терена, израчунавањима трендова и варијација на берзама, флуктуацијама температуре, подводним истраживањима, понашањима животиња... Све ово имплицира да је фрактална геометрија за веома кратко време постала математика будућности и да њено познавање и разумевање представља основу за развој науке.

Због широке примене фракталне геометрије у графичком креирању природних појава, заинтересовала ме је симулација примера, па сам стога скоро све слике у раду креирао самостално у савременим софтверима.

Литература

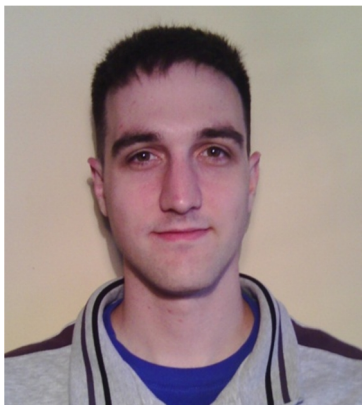
- [1] Benoit B. Mandelbrot, *The fractal geometry of nature*, W. H. Freeman and company, New York, 1983
- [2] Kenneth Falconer: *Fractal geometry – Mathematical Foundation and Applications*, Second edition, John Wiley & Sons Ltd, the Atrium, Southern Gate, Chichester, England, 2003
- [3] Michael F. Barnsley, *Fractals everywhere*, Morgan Kaufmann, New York, 2000
- [4] Slawomir Nikiel, *Iterated Function Systems for Real-Time Image Syntesis*, Springer, London, 2007
- [5] Przemyslaw Prusinkiewicz, Aristid Lindenmayer: *The Algorithmic Beauty of Plants*, Springer, New York, 1990
- [6] Ivan Nourdin, *Selected Aspects of Fractional Brownian Motion*, Springer, Milano, 2012
- [7] Ton Dieker, *Simulation of fractional Brownian motion*, University of Twente, Department of Mathematical Sciences, Netherlands, 2004
- [8] Takeyuki Hida, *Stochastic Analysis: Classical and Quantum – Perspectives of White Noise Theory*, Meijo University, Nagoya, Japan, 2004
- [9] Yuliya S. Mishura, *Stochastic Calculus for Fractional Brownian Motion and Related Processe*, Springer, Belin, Germany, 2008
- [10] Lewis R., *Fractals in Your Future*, Chapter 1. Ontario, 2000
- [11] Turner, M.J.: *Modeling Nature With Fractals*, Leicester, 2000
- [12] Clifford A. Pickover: *Computers, Pattern, Chaos and Beauty: graphics from an unseen world by Clifford A Pickover*. Stroud: Sutton, 1990.
- [13] Przemyslaw Prusinkiewicz, James Hanan: *Lecture Notes in Biomathematics: Lindenmayer Systems, Fractals, and Plants*, Springer – Verlag, Regina, March, 1989
- [14] Radomir Mech: *CPFG Version 4.0 User's Manual*, May 31, 2005
- [15] Cedric Loi, Paul Henry Cournede: *Generating Functions of Stochastic L-Systems and Application to Models of Plant Development*. Ecole Centrale Paris, Laboratory of Applied Mathematics and Systems, 92290 Chatenay Malabry, France, 2008
- [16] A. Takenaka, paper “A simulation model of tree architecture development based on growth response to local light environment”, *Journal of Plant Research* 107 (1994), pp. 321-330
- [17] R. Mech, P. Prusinkiewicz, *Visual models of plants interacting with their environment*, *Procedings of Siggraph 96*, pp. 397-410.
- [18] Ashok K. Samal, Brian Peterson, David J. Holliday, *Recognising Plants Using Stochastic L-systems*, University of Nebraska – Lincoln, Department of Computer Science and Engineering, 1994
- [19] Knut Arild Erstad, *L-systems, Twinning Plants, Lisp: Modeling of Twinning plants using Environmentally Sensitive L-systems, and an Extensible Framework for L-systems in Common Lisp*, Department of Informatics, University of Bergen, 2002
- [20] <http://classes.yale.edu/fractals/>
- [21] <http://www.gameprogrammer.com/fractal.html>
- [22] <http://www.math.harvard.edu/library/sternberg/slides/1180910.pdf>

- [23] <http://mrl.nyu.edu/~perlin/doc/oscar.html>
- [24] https://code.google.com/p/fractalterraingeneration/wiki/Value_Noise
- [25] https://code.google.com/p/fractalterraingeneration/wiki/Perlin_Noise
- [26] https://code.google.com/p/fractalterraingeneration/wiki/Midpoint_Displacement
- [27] http://www.videotutorialsrock.com/opengl_tutorial/terrain/home.php
- [28] https://en.wikipedia.org/wiki/Diamond-square_algorithm
- [29] <http://lighthouse3d.com/opengl/terrain/index.php?mpd2>
- [30] <http://www.lighthouse3d.com/opengl/terrain/index.php3?fault>
- [31] <http://www.lighthouse3d.com/opengl/terrain/index.php?impdetails>
- [32] <http://www.cgjennings.ca/toybox/lsystems/>
- [33] <https://en.wikipedia.org/wiki/L-system>
- [34] <http://www.yekaterinasatanina.com/tag/computer-science/>
- [35] <http://paulbourke.net/fractals/>
- [36] http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
- [37] <https://www.clear.rice.edu/comp360/lectures/FractalIFSTEXTNew.pdf>
- [38] http://www.wahl.org/fe/HTML_version/link/FE3W/c3.htm
- [39] <http://natureofcode.com/book/chapter-8-fractals/>

Софтвери коришћени у изради

1. Visual C++
2. IFS Construction Kit: <http://ecademy.agnesscott.edu/~lriddle/ifskit/>
3. Chaotica: <http://www.chaoticafractals.com/>
4. FractInt: <http://www.fractint.org/>
5. Mandelbulb 3D: <http://mandelbulb.com/2014/mandelbulb-3d-mb3d-fractal-rendering-software/>
6. ultra fractal 5: <http://www.ultrafractal.com/>
7. XenoDream: <http://www.xenodream.com/>
8. L-Systems Turtle Graphics Renderer: <http://www.kevs3d.co.uk/dev/lsystems/>
9. Lindenmayer System Generator: <http://nolandc.com/sandbox/fractals/>
10. XLinden L-System Parser: <http://www.ph2.net/XLinden/>
11. POV-Ray – The Persistence of Vision Raytracer: <http://www.povray.org/>
12. L-studio: http://algorithmicbotany.org/virtual_laboratory/
13. Terragen: <http://planetside.co.uk/>
14. Gimp: <http://www.gimp.org/>
15. Geogebra: <https://www.geogebra.org/>
16. MDTerrain by Mathias Grumet: <http://www.embege.com/fractals/mdterrain/>
17. Fractal Terrains 3: <https://secure.profantasy.com/products/ft.asp>
18. World Machine: <https://www.world-machine.com>

Биографија



Немања Стојановић је рођен 15.10.1991. у Сомбору. Завршио је Основну школу „Никола Вукићевић“ у Сомбору као носилац Вукове дипломе. Исте године је уписао „Средњу економску школу“ у Сомбору коју завршава 2010. године, такође као носилац Вукове дипломе. По завршетку средње школе уписује основне академске студије примењене математике на Природно-математичком факултету у Новом Саду које завршава 2013. године. Исте године у октобру уписује мастер академске студије примењене математике. Све испите предвиђене планом и програмом полаже закључно са јунским испитним роком 2015. године и тиме

стиче услов за одбрану овог мастер рада.

Нови Сад, септембар 2015. године

Немања Стојановић

УНИВЕРЗИТЕТ У НОВОМ САДУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ
КЉУЧНА ДОКУМЕНТАЦИЈА ИНФОРМАЦИЈА

Редни број:

RBR

Идентификациони број:

IBR

Тип документације: Монографска документација

TD

Тип записа: Текстуални штампани материјал

TZ

Врста рада: Мастер рад

VR

Аутор: Немања Стојановић

AU

Ментор: Проф. др Данијела Рајтер-Ћирић

MN

Наслов: Системи итеративних функција, Линденмајерови системи, фракционо Брауново кретање и примена у графичком креирању природних појава

NR

Језик публикације: Српски (ћирилица)

JP

Језик извода: српски/енглески

JI

Земља публикавања: Република Србија

ZP

Уже географско подручје: Војводина

UGP

Година: 2015.

GO

Издавач: Ауторски репринт

IZ

Место и адреса: Нови Сад, Департман за математику и информатику, Природно-математички факултет, Универзитет у Новом Саду, Трг Доситеја Обрадовића 4
МА

Физички опис рада: (5/130/1/7/160/46/0)
(број поглавља / страна / лис. цитата / табела / слика / графика / прилога)
FO

Научна област: Математика
NO

Научна дисциплина: Примењена математика
ND

Предметна одредница / Кључне речи: Фрактал, Системи итеративних функција, Контракција, Хауздорфова метрика, Кондензациони скуп, Колаж теорема, Линденмајерови системи, Брауново кретање, фракционо Брауново кретање, фракционе Браунове површине, heightmap-е, Перлинов шум.
PO
UDK

Чува се: Библиотека Департмана за математику и информатику Природно-математичког факултета, Универзитета у Новом Саду.
ЋУ

Важа напомена:
VN

Извод:
У овом раду су изложени основни појмови везани за фракталну геометрију, самосличност и димензије фрактала, као и различити модели за креирање дводимензионалних и тродимензионалних фрактала. Дефинисани су појмови попут Система итеративних функција, Линденмајерових система, Перлиновог шума и фракционог Брауновог кретања. У првом делу рада смо дефинисали системе итеративних функције и приказали два алгорита којима се креирају фрактали. Приказали смо рад оба алгорита на неколико примера биљака. У другом делу рада смо дефинисали Линденмајерове системе и корњачин алгорита и њиме измоделирали разне врсте биљака у две и три димензије. У трећем делу рада смо дефинисали фракционо Брауново кретање и објаснили његову примену. Затим смо дефинисали heightmap-е и три алгорита за њихово креирање. Приказане су модификације алгоритама са циљем измена heightmap-а ради добијања различитих текстура које се појављују у природи као и за креирање терена. Приказани су примери у савременим софтверима попут Terragen-а и World Machine-а
IZ

Датум прихватања теме од стране NN већа: 21.04.2015.
DP

Датум одбране: октобар 2015.

DO

Чланови комисије:

КО

Председник: др Сања Рапајић, ванредни професор,
Природно-математички факултет, Универзитет у Новом Саду,

Ментор: др Данијела Рајтер Ћирић, редовни професор,
Природно-математички факултет, Универзитет у Новом Саду,

Члан: др Дора Селеши, ванредни професор,
Природно-математички факултет, Универзитет у Новом Саду.

UNIVERSITY OF NOVI SAD
FACULTY OF SCIENCE AND MATHEMATICS
KEY WORD DOCUMENTATION

Accession number:

ANO

Identification number:

INO

Documentation: Monograph type

DT

Type of record: Printed text

TR

Content Code: Master's thesis

CC

Author: Nemanja Stojanović

AU

Mentor: Danijela Rajter-Ćirić. Ph.D.

MN

Title: Iterated function system, Lindenmayer systems, fractional Brownian motion and implementation in creating nature

TI

Language of text: Serbian

LT

Language of abstract: English

LA

Contry of publication: Serbia

CP

Locality of publication: Vojvodina

LP

Publication year: 2015

PY

Publisher: Author's reprint

PU

Publication place: Novi Sad, Department of Mathematics and Informatics, Faculty of Science and Mathematics, University of Novi Sad, Trg Dositeja Obradovića 4

PP

Physical description: (5/130/1/7/160/46/0)

(number of sections / pages / references / tables / pictures / graphs / appendices)

PD

Scientific field: Mathematics

SF

Scientific Discipline: Applied mathematics

SD

Subject / Key words: Fractal, Iterated Function System, Contraction, Hausdorff distance, Condensation set, Collage theorem, Lindenmayer system, Brownian motion, fractional Brownian motion, Brownian surfaces, heightmap, Perlin noise

SKW

UC

Holding data: The Library of the Department of Mathematics and Informatics, Faculty of Science and Mathematics, University of Novi Sad

HD

Note:

N

Abstract:

This paper presents basic concepts related to fractal geometry, self-similarity and fractal dimension, as well as models for creating two-dimensional and three-dimensional fractals. We defined terms such as Iterated Function System, Lindenmayer System, Perlin Noise and fractional Brownian motion.

In the first part of the paper we have defined Iterated Function System and displayed two algorithms for creating fractals. We explained how both algorithms work on several examples of plants. In the second part we defined Lindenmayer systems and turtle algorithm with whom we created various plant species in two and three dimensions. In the third part of this paper we defined fractional Brownian motion and its implementation. Then we defined heightmaps and three algorithms for creating them. We presented modifications of these algorithms with the aim of amendment heightmaps with the purpose of obtaining different textures that appear in nature as well as to create landscapes. We presented examples in modern softwares such as Terragen and World Machine.

AB

Accepted by the Scientific Board on: 21.04.2015.

ASB

Defended: october 2015.

DE

Thesis defend board:

DB

President: Sanja Rapajić, Ph.D., Associate Professor,
Faculty of Science and Mathematics,
University of Novi Sad,

Supervisor: Danijela Rajter-Ćirić, Ph.D., Full Professor,
Faculty of Science and Mathematics,
University of Novi Sad,

Member: Dora Seleši, Ph.D., Associate Professor,
Faculty of Science and Mathematics,
University of Novi Sad.