МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра прикладної математики та моделювання складних систем

Допущено до захисту

Завідувач кафедри ПМ та МСС

_____ Коплик І.В.
(підпис)

«__» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»

спеціальність 113 «Прикладна математика»

освітньо-професійна програма «Прикладна математика»

тема роботи **«МОДЕЛЮВАННЯ ФОРМУВАННЯ НАНОСТРУКТУРИ ПЛІВКОВИХ ПОКРИТТІВ НІТРИДІВ ПЕРЕХІДНИХ МЕТАЛІВ»**

Виконавець

студент факультету ЕлІТ

Левчинський Антон Володимирович
(прізвище, ім’я, по батькові)

_____
(підпис)

Науковий керівник

професор, доктор фіз.-мат. наук
(науковий ступінь, вчене звання)

Гончаров Олександр Андрійович _____
(прізвище, ім’я, по батькові) (підпис)

Суми – 2023 р

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Факультет	**електроніки та інформаційних технологій**
Кафедра	**прикладної математики та моделювання складних систем**
Рівень вищої освіти	бакалавр
Галузь знань	**11 Математика та статистика**
Спеціальність	**113 Прикладна математика**
Освітня програма	**освітньо- професійна** «Прикладна математика»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМ та МСС
Коплик І.В. _____
«___» _____2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Левчинському Антону Володимировичу

1. Тема роботи: Моделювання формування наноструктури плівкових покриттів нітридів перехідних металів.

Керівник роботи Гончаров О.А., професор, доктор фіз.-мат. наук.
затверджено наказом по факультету ЕлІТ від «___» ___ 2023 р. № _____
2. Термін подання роботи студентом від «11» червня 2023 р.
3.Вихідні дані до роботи: навчальні посібники, методичні вказівки, монографії, ресурси інтернету, періодичні видання з теми дослідження.
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): методи зростання плівкових покриттів, фізична модель росту плівок нітридів перехідних металів, математична модель, комп’ютерна програма, аналіз результатів.

5.Перелік графічного матеріалу: осадження атомного шару на різних підкладках, механізм зростання, напрямки зростання, фізичні процеси, зростання в 3D форматі.

6. Консультанти до проекту (роботи), із значенням розділів проекту:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Літературний огляд	Гончаров О.А., професор кафедри ПМ та МСС.		
Модель формування плівкового покриття	Гончаров О.А., професор кафедри ПМ та МСС.		

7. Дата видачі завдання від «15» травня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування роботи, заходи	Термін виконання	Примітка
1	Провести аналіз механізмів зростання нітридних покриттів.	15.05.23-20.05.23	
2	Побудувати фізичну модель, що описує ріст плівок нітридів перехідних металів.	21.05.23-26.05.23	
3	Побудвати математичну модель задачі дослідження.	27.05.23-01.06.23	
4	На базі математичної моделі створити комп’ютерну програму.	02.06.23-07.06.23	
5	Провести аналіз отриманих результатів	08.06.23-10.06.23	

Здобувач вищої освіти _____ Левчинський А.В.
(підпис)

Керівник роботи _____ Гончаров О.А.
(підпис)

РЕФЕРАТ

Кваліфікаційна робота: 45 стор., 22 рис., 16 джерел.

Мета роботи: розробити математичну модель процесу утворення плівкової наноструктури нітридів перехідних металів. Створити комп’ютерну програму та зробити розрахунки методом Монте-Карло. Дослідити вплив параметрів при зростанні плівкової наноструктури.

Об'єкт дослідження: процес формування плівкової наноструктури на прикладі формування нітриду перехідного металу.

Предмет дослідження: комп’ютерна модель формування плівок нітридів перехідних металів.

Методи навчання: метод молекулярної динаміки та метод випадкових випробувань (метод Монте-Карло).

Для побудови нової моделі брались до уваги роботи інших дослідників. Основна відмінність полягає у використанні 3D формату за допомогою якого більш детально розібрані властивості тонких плівок. Також були додані нові напрямки зростання з метою отримання більш точних розрахунків. Серед інших переваг даного експерименту є можливість бачити зростання в режимі реального часу.

Отримана модель зростання плівкових покриттів нітридів перехідних металів може бути використана для дослідження та покращення структури матеріалів. У свою чергу це допоможе отримати більшу твердість та ефективність, матиме позитивний вплив на термін використання у виробництві та промисловості.

Ключові слова: ПЛІВКОВА НАНОСТРУКТУРА, МЕТОД МОНТЕ-КАРЛО, МАТЕМАТИЧНА МОДЕЛЬ, ОСАДЖЕННЯ АТОМІВ, НІТРИДИ ПЕРЕХІДНИХ МЕТАЛІВ, 3D МОДЕЛЮВАННЯ.

ЗМІСТ

ВСТУП

Тонкі плівки зазвичай покращують властивості багатьох матеріалів за рахунок формування шару з необхідними фізичними та хімічними характеристиками. Завдяки цьому вони мають широке застосування в електроніці, космічній техніці, сучасному машинобудівництві, металургії та інших сферах [1].

Деякі плівки мають у своєму складі нітриди, а саме сполуки металів з азотом [2]. Вони представляють великий інтерес через їх твердість, високу температуру плавлення, металічний блиск, надпровідність, а також чудову хімічну стійкість в різних агресивних середовищах.

В залежності від характеру зв’язку нітриди, зазвичай, поділяють на три основні групи:

1.Іонні нітриди лужних металів, лужноземельних металів та металів, які належать до групи 3 періодичної таблиці.

2.Ковалентні нітриди, які формуються з елементами груп 13-15, наприклад BN, Si_3N_4 та інші. Ці сполуки, зазвичай, дуже міцні, а також не є провідниками або напівпровідниками.

3.Нітриди перехідних металів, де атоми азоту додаються в кристалічні решітки основного матеріалу.

У цій роботі найбільше описуються нітриди перехідних металів. Результати досліджень мікроструктури плівки цих сполук шляхом експерименту допоможуть отримати більшу твердість та ефективність.

Для спостереження за змінами мікроструктури часто необхідно враховувати значення різних параметрів та факторів. Зі збільшенням кількості таких факторів можна отримати більш точні результати. У реальних експериментах часто неможливо побачити ріст плівки, де враховуючи деякі фізичні властивості, можна зробити детальний аналіз.

З огляду на велику кількість розрахунків та неможливість проведення деяких експериментів в реальному житті, зручно використовувати метод Монте-Карло. Перевага цього методу полягає у використанні великого обсягу даних, які генеруються випадковим чином. При цьому найголовніше враховувати ймовірнісні характеристики процесу, який ми досліджуємо. Отримані результати будуть дуже схожі на теоретичні, хоча кількість ітерацій набагато зменшиться. Для моделювання зростання тонкої плівки із використанням цього методу, беруться випадкові точки зі своїми фізичними властивостями. Таким чином можна побачити комп’ютерну реалізацію методу реактивного магнетронного напилення з подіями осадження та дифузії.

РОЗДІЛ 1

ЛІТЕРАТУРНИЙ ОГЛЯД

1.1 Структурні особливості перехідних металів

Нітриди перехідних металів з електронними d- і f- оболонками характеризуються більш або менш широкими областями гомогенності, переважно металічними властивостями, високою електропровідністю, високими температурами плавлення, твердістю. Результатом цього, є такий перерозподіл валентних електронів металу і азоту, який призводить до утворення максимально статистичної ваги атомів, які мають стійкі конфігурації локалізованої частини валентних електронів. Для нітридів перехідних металів характерна наявність сильного ковалентного зв'язку між атомами металу і азоту, що дозволяє віднести їх до класу ковалентно-металічних нітридів. Діапазон змін фізичних якостей всередині цього класу нітридів доволі широкий [3].

При утворенні нітридів перехідних металів мається на увазі утворення у атомів азоту як s^2p^6-, так і sp^3– конфігурацій, відношення статистичних вагів яких залежить від особливостей партнера по з'єднанню. По мірі підвищення статистичної ваги d^5–конфігурацій атомів перехідного металу в металічному кристалі повинна зменшуватись можливість передачі нелокалізованих електронів атомам азоту з утворенням останніми s^2p^6–конфігурацій, тобто збільшується статистична вага d^5– і s^2p^3-, але зменшується статистична вага d^5– і s^2p^6– конфігурацій атомів металу і азоту, які входять в склад нітриду [3].

Зі зменшенням складу азоту в межах області гомогенності нітридних фаз перехідних металів твердість знижується. Зменшення відбувається в швидшому темпі, ніж для карбідів, що пояснюється появою у нітридів долі іонного зв'язку, який збільшується зі зменшенням кількості азоту в межах області гомогенності, тобто зі збільшенням локалізації електронів на зв'язках Ме-Ме і зі збільшенням статистичної ваги s^2p^6– конфігурацій атомів азоту. Зменшення твердості супроводжується появою

і ростом енергетичного розриву між станами атомів металу і азоту зовнішньо виглядає як ріст ширини забороненої зони [3].

В залежності від енергетичних умов формування плівки може знаходитись в різних енергетичних станах: від аморфноподібного до нанокристалічного з переважно з волокнистою структурою. Для нітридних покриттів перехідних металів зі структурою NaCl можливі два напрямки переважного росту (111) або (200).

Показано [4], що густина TiN дорівнює 5.22 г/м^3, температура плавління 3203K із атмосферним тиском. Механічні характеристики TiN та HfN сильно залежать від фаз, які має плівка. Однофазні плівки TiN мають можливість не деформуючись витримувати тиск між 14.7 ± 4.9 ГПа та 34.3 ± 4.9 ГПа у той час як не стехіометричні плівки демонструють досить низькі показники лише в 3.3 ГПа. TiN має достатньо високий модуль Юнга, який становить 350 ГПа. HfN має густину 11.7 г/м^3, температуру плавління 3578K, але поступається TiN в стійкості до деформацій. Слід відмітити, що механічні властивості плівок можуть варіюватись у разі зміни орієнтації пластини при напиленні [5].

TiN та HfN поводять себе як метали. Ti та Nf є перехідними металами, які мають три валентні електрони. Структура та слабо закріплені основні електрони в Ti та Nf змушують TiN та HfN стати провідниками.

TiN та HfN зазвичай використовують як міцні, захисні, зносостійкі покриття завдяки їх твердості, хімічній, термічній та механічній стабільності. Їх також використовують як дифузійний бар’єр у виробництві мікроелектроніки із-за їх електричної провідності.

1.2 Моделювання формування плівкових наноструктур перехідних металів

У цій роботі буде найбільше розглянутий механізм зростання Вольмера-Вебера [6]. Цей механізм також відомий як “3D острівцевий механізм”. Згідно з цим механізмом, атоми осідають на частинках субстрату і формують початкові острівці. Як тільки атоми потрапляють на субстрат, вони кріпляться до вже існуючих острівців. Таким чином острівці ростуть швидше, а вже потім склеюються із сусідніми острівцями. Товщина плівки не збільшується до тих пір, доки осідаючі атоми повністю не заповнять прогалини між острівцями та нова поверхня повністю не сформується.

Існує два відомих методи моделювання за допомогою яких можна реалізувати схоже зростання, а саме метод молекулярної динаміки та метод Монте-Карло. Слід відмітити, що є багато комбінованих методів, які поєднують переваги цих двох.

З метою дослідження зародження плівок TiN на різних підкладках було виконане спостереження за осадженням атомного шару [8].

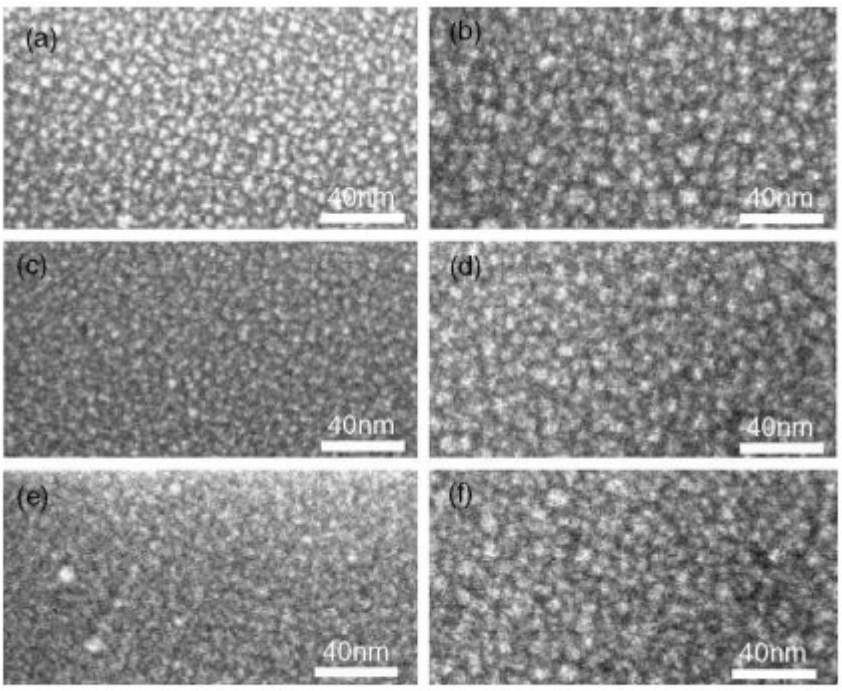


Fig. 1. Surface coverage of TiN on dielectric surfaces, (a) 5 cycles on SiO_2, (b) 100 cycles on SiO_2, (c) 5 cycles on HfSiO, (d) 100 cycles on HfSiO, (e) 5 cycles on HfO_2, (f) 100 cycles on HfO_2.

рис.4

Після п’яти циклів було з’ясовано, що густина зерен найнижча на підкладці SiO_2 у порівнянні з іншими діелектричними підкладками. Така закономірність зберігається і після ста циклів.

Швидкість росту плівки на підкладці HfO_2 майже не змінюється. З іншого боку, швидкість зростання на підкладці SiO_2 поступово зменшується. Після двадцяти циклів можна чітко побачити таку тенденцію на рис.5. Також наявність точки зміни швидкості зростання говорить про завершеність покриття діелектричної поверхні.

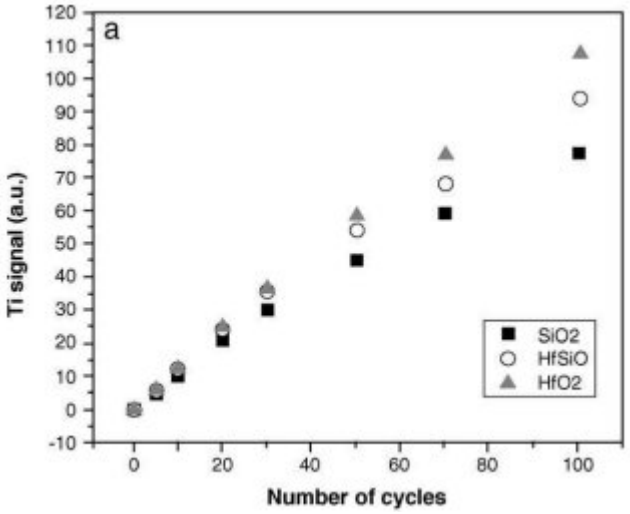


рис.5

Виходячи із попередніх спостережень можна сказати, що процес формування на підкладці SiO_2 потребує найбільше часу. Разом з тим, для склеювання острівців та утворення суцільного шару висота стовпців на підкладці SiO_2, у середньому, має бути в чотири більшою ніж у випадку з HfO_2.

Наступний дослід полягав у отриманні плівок методом реактивного магнетронного розпилення, враховуючи різні значення кутів під якими це відбувається [9].

Катод представляє собою диск діаметром 75 мм, закріплений під фіксованим кутом 25 градусів. Покриття наносились під кутами 5°, 35°, 65°, 75° та 85° як показано на рис.6.

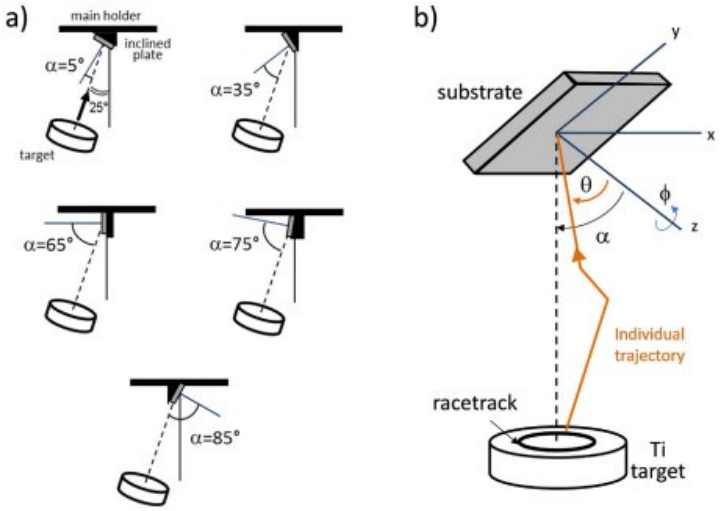


Fig. 1. a) Schematic illustration of the substrate holder configurations used to produce TiN thin films at various tilt angles α. The normal to the target surface (cylindrical cathode) makes an angle of 25° with respect to the vertical axis (solid line). The angle α is referred with respect to the normal of the substrate surface. b) Schematic showing that the incident angle θ of a given particle reaching the substrate may differ from the geometrical tilt angle α due to collisions during transport in the gas phase.

рис.6

Проводилось декілька дослідів під різним робочим тиском та були отримані наступні розподіли

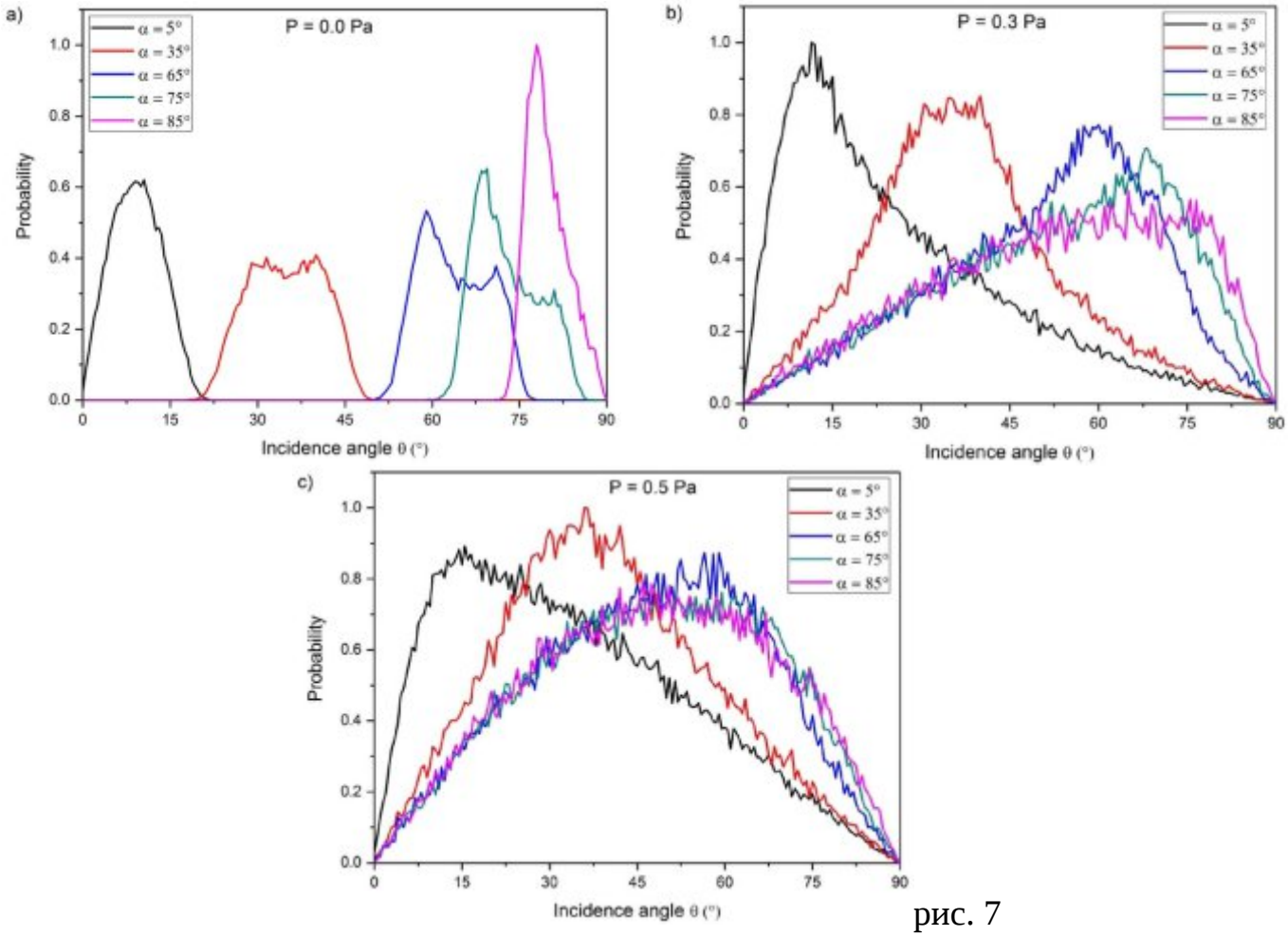


рис. 7

У випадку, коли не було робочого тиску, частина потоку втрачалась, осідаючи на стінках камери. Цей ефект врегульовується за допомогою збільшення тиску всередині камери до 0.3 Па або навіть до 0.5 Па. Таким чином, більше частинок можуть брати участь у формуванні плівки. Для кращого уявлення були зроблені зображення плівок після формування на рис.8.

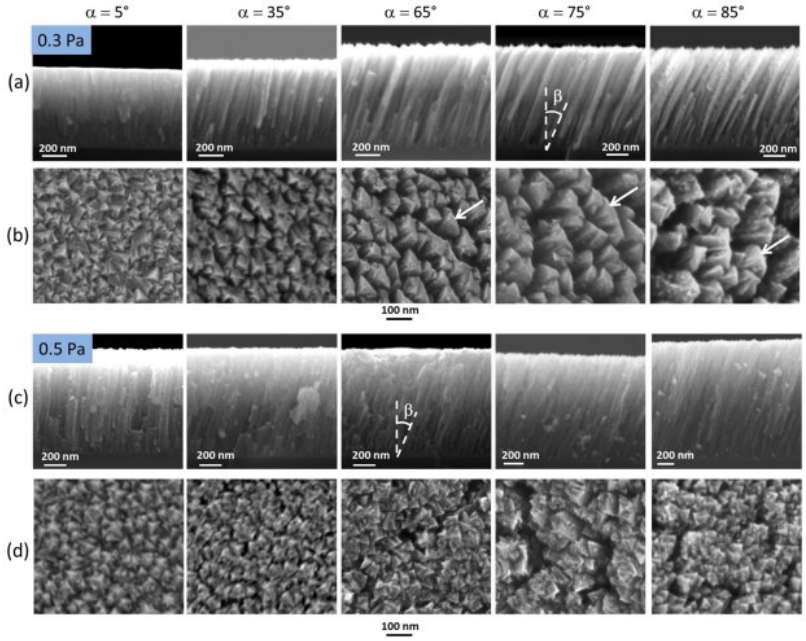


рис.8

Можна виявити ще деякі морфологічні особливості зі збільшенням товщини плівки, якщо окрім кута напилення враховувати ще й температуру, а також зробити дослідження в 3D [10].

За звичайних умов осадження, еволюція мікроструктури зазвичай зображається як функція гомологічної температури

$$T_h = \frac{T}{T_m},$$

де T_m–температура плавлення матеріалу та якісно описується моделями структурних зон. За умов напилення під деяким кутом очікується, що поверхнева дифузія зможе протидіяти ефекту затінення, оскільки згладжує поверхню плівки та скорочує пористість. При розгляді моделей структурних зон для різних кутів осадження були виявлені чіткі області формування наностержнів, наностовпців біля граней та виступів.

На рис. 9 показано, що на ранніх етапах росту зі збільшенням температури субстрату до 500K, плівка має великі, відокремлені острівці порівняно з ситуацією при 300K, де кожний адатом вдаряється та закріплюється зверху до вже осаджених атомів, що призводить до утворення рельєфу із стохастичною шорсткістю. Також у випадку з T=500K виявлена нижча густина ядер, яка призводить до більш нахилених стовпців, які в подальшому роблять плівку товстішою через ефект затінення.

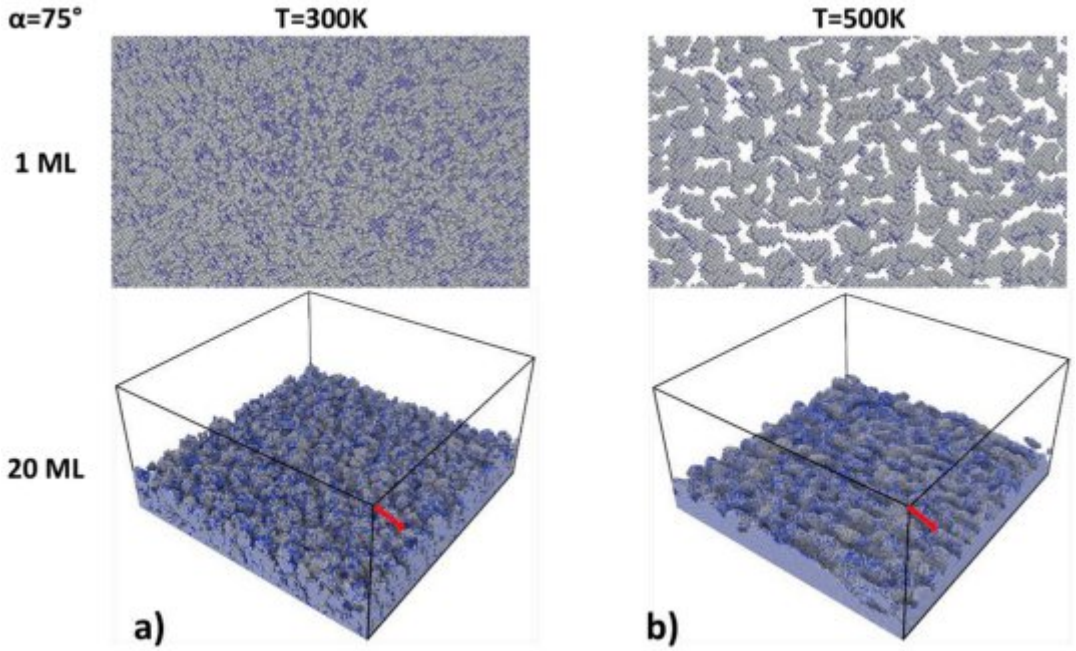


рис.9

Ще одними авторами був проведений детальний дослід ефекту парціального тиску азоту на характеристики плівок TiN [13]. На рис.10 показана залежність зміни катодного потенціалу від парціального тиску азоту. Слід відмітити, що в цьому досліді сила струму дорівнює 0.35 А, а постійний тиск $8 * 10^{-3}$ мілібар. Катодний потенціал демонструє поступове зростання при збільшенні парціального тиску азоту.

У разі катодного розряду із постійною силою струму, катодний потенціал V_c може бути описаний формулою

$$V_c = \frac{V_i}{\gamma \eta_i \phi_e}$$

У цій формулі V_i – середня напруга іонізації в розряді, η_i – ефективність збору іонів на мішені, ϕ_e – середня доля енергії, яка необхідна вторинному електрону при падінні катода, що видаляється в процесі іонізації перед тим як електрон вийде із системи.

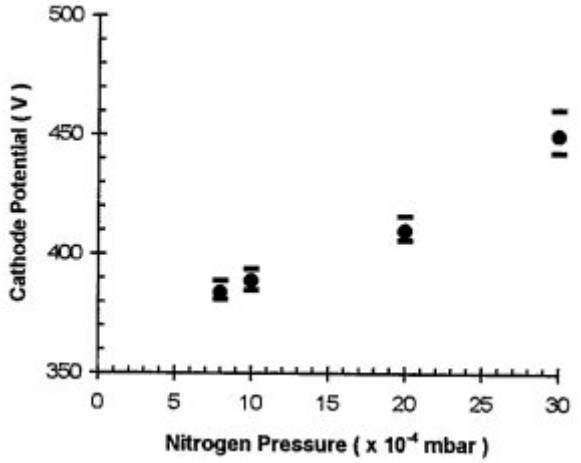


рис.10

Передбачається, що коефіцієнт емісії вторинного електрона γ повинен знижуватись при збільшенні парціального тиску азоту.

Цими ж самим авторами було помічене зменшення швидкості осадження при збільшенні парціального тиску рис.11.

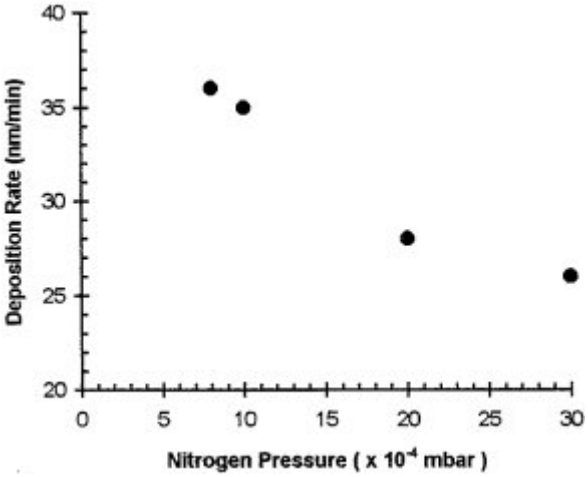


рис.11

Дифракційні дослідження вказують, що всі плівки мають переважну орієнтацію (111) рис.12. Інтенсивність такої орієнтації зменшується при збільшенні парціального

тиску азоту. Якщо тиск досягає значення в $3 * 10^{-3}$ мілібар, то починає з’являтись орієнтація (200). Це означає, що кристалографічна орієнтація плівки змінюється із лише (111) до двох станів (111) та (200).

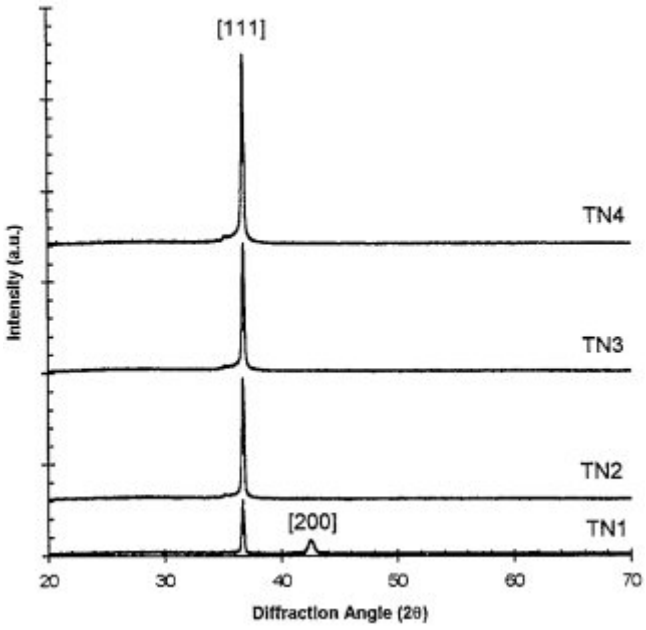


рис.12

Підсумовуючи всі досліди можна сказати, що на формування плівки сильно впливає матеріал з якого зроблена підкладка. Підкладка HfO_2 найкраще підходить у разі, коли нам потрібно отримати готову плівку за найменший час. Інші підкладки сприяють швидшому росту стовпців, але завершений шар довго не може сформуватись. Також має значення кут під яким робиться розпилення. Якщо він буде дуже великим, то відбуватиметься марнування матеріалу, оскільки його частина осідатиме на стінках камери. Для того, щоб запобігти такому ефекту, потрібно збільшити тиск всередині камери. Ще одну важливу роль грає змочування підкладки. У разі, коли підкладка суха, ми отримаємо менші зерна і це негативно вплине на міцність структури. Окрім цього була виявлена можливість плівки змінювати структуру росту при різних значеннях парціального тиску азоту. Було розглянуто вплив температури субстрату. При збільшенні температури збільшується вірогідність появи ефекту затінення, який призводить до збільшення товщини плівки, але негативно впливає на міцність.

З урахуванням літературного огляду в подальшому буде побудована фізична модель та проведене вдосконалення математичної моделі. Для її чисельної реалізації краще за все використовувати метод Монте-Карло, оскільки він суттєво зберігає часові затрати та не поступається в ефективності іншим методам, а також враховує ймовірнісні характеристики процесів, які відбуваються при формуванні.

РОЗДІЛ 2 МОДЕЛЬ ФОРМУВАННЯ ПЛІВКОВОГО ПОКРИТТЯ

2.1 Процес формування

У цій роботі найбільше буде розглянутий механізм формування Вольмера-Вебера рис.13(б). Згідно з цим механізмом, матеріал при осадженні не змочує підкладку, тому частинки субстрату сильніше взаємодіють між собою. У результаті цього ріст відбувається прямо на поверхні підкладки із зародженням монокристалічних 3D острівців. Цей процес продовжується доки менші стовпці не почнуть об’єднуватись один з одним та не почнеться формування більших стовпців. Таким чином готовий шар довго не може сформуватись, хоча стовпці стають доволі високими. Наприкінці росту з’явиться суцільна полікристалічна плівка.

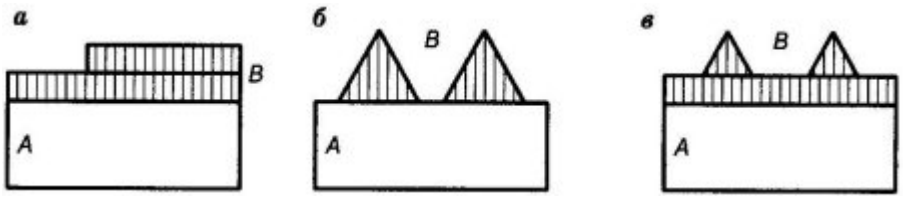


рис.13

Усі чіткі етапи росту супроводжуються кінетично керованими процесами, впливаючи не тільки на морфологію поверхні, розмір зерен та текстуру, але й на внутрішні деформації.

Існують дослідження, які описують немонотонний розвиток сил, що впливають на деформацію зі збільшенням товщини плівки. Загалом це може бути описане як CTC (compressive-tensile-compressive) послідовність або стиснення-розтягнення-стиснення. Така послідовність пов’язана із подіями перед зрощуванням, у момент зрощування та після зрощування.

Також важливе значення має енергія частинок у порівнянні з енергією підкладки, а також кут під яким формуються стовпці. Слід виокремити три види енергій, які впливають на формування, а саме γ_o – енергія на поверхні готового шару або стовпця, γ_i – енергія між готовим шаром або стовпцем та субстратом, γ_s – енергія на поверхні субстрату у разі, коли речовина ще не потрапила на підкладку. Схематично ці енергії можна побачити на рис.14.

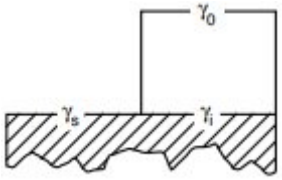


рис.14

На основі цього можна точно порахувати кут під яким формується пірамідоподібний стовпець з текстурою росту (111)

рис.15

використовуючи наступну формулу

$$cos\,\theta = \frac{\gamma_s - \gamma_i}{\gamma_o}$$

2.2 Фізичні процеси

На рис.16 зображені типові дифузійні процеси, які беруть участь у атомарно-променевій епітаксії. Атоми осідають на субстрат із термальною енергією 0.1eV та потоком F, який виражений в моношарах за секунду.

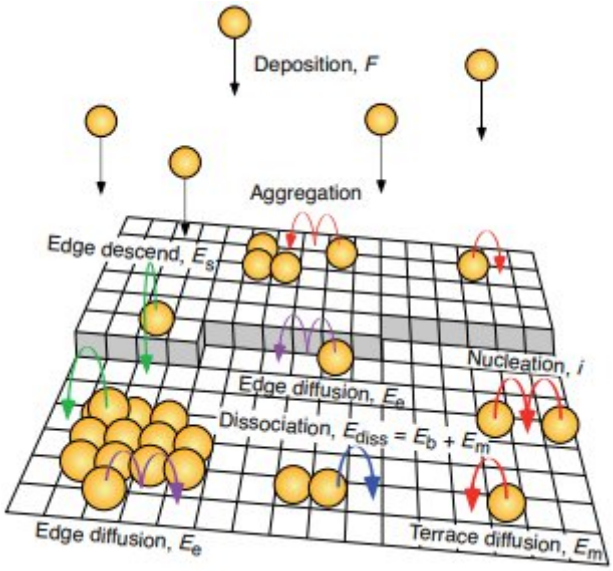


рис.16

Енергія, яка накопичується при формування зв’язку між абсорбатом та субстратом, миттєво розсіюється на решітку, тому атоми прямують до термічної рівноваги у місці удару. Це можна бачити в експериментах, які досліджують розмір кластерів або місця поглинання атомів при низьких температурах, виходячи із ймовірнісних характеристик та припускаючи, що дифузії взагалі не буде. Тим не менш, винятком є перехідні стрибки, де для дифузії може використовуватись частина зв’язуючої енергії поглинання. Варто відмітити різницю між перехідним нетермічним рухом та термічним рухом з малими енергетичними бар’єрами в напрямку до найближчих сусідів та кластерів. Для певних комбінацій елементів, енергія поглинання могла б використовуватись для запуску перехідних процесів обміну, навіть при низькій температурі. Однак, при відсутності обмінів та достатньо великій відстані від кластерів та адатомів, атоми зазвичай абсорбуються, де вони приземляться з парової фази.

Дифузійними процесами, які беруть участь в епітаксіальному зростанні, є термічно активовані стрибки, переважно у вигляді руху єдиного адатому між вузлами решітки. Однак, також може бути залучений узгоджений рух декількох атомів. Теорія перехідного стану припускає, що до появи термічної рівноваги, атоми, між двома послідуючими стрибками, доволі довго залишаються на своїх місцях поглинання. Також припускається, що перетин сідлової точки буде незначним. Ці припущення справджуються, якщо різниця енергій зв’язку та відокремлення при дифузії задовольняє наступний вираз

$$E \ll k_B T$$

Імовірність стрибків в процесі дифузії описується формулою

$$\nu_n = \nu_{0,n} e^{\frac{-E_n}{k_B T}}$$

Тут $\nu_{0,n}$ – частота спроб, яка зазвичай знаходиться в діапазоні 10^{12} – 10^{13}Гц.

Процес міграції одиничних адатомів у місцях, де ще немає скупчення є найбільш фундаментальним серед усіх процесів. Це дає початок зародженню острівців або стовпцевому зростанню при підвищених температурах, де довжина вільного пробігу адатомів більша ніж ширина канавки, тобто “вільного” місця між острівцями. В залежності від густини одночасно дифундуючих частинок, виділяють “колективний” коефіцієнт дифузії ансамблю взаємодіючих частинок та внутрішній коефіцієнт дифузії, який характеризує середньоквадратичне відхилення ізольованої випадкової частинки в кожний момент часу. При типовій швидкості росту, густина дифундуючих частинок доволі невелика. Для адатомів металу на металічному субстраті характерна досить велика відстань між частинками у порівнянні з типовими діапазонами взаємодії, а густина кластерів визначається за коефіцієнтом дифузії D, який в свою чергу описується наступною формулою

$$D = D_0 e^{\frac{-E_m}{k_B T}}$$

Де $D_0 = \frac{1}{4}\nu_0$, а D виражається в клітинках субстрату за секунду. Дріб $\frac{1}{4}$ в D_0 стосується дифузії на квадратній підкладці в 2D розмірності.

Дифузії уздовж тераси (вільного місця між острівцями) завершується як тільки адатоми зіткнуться з одним або більшим числом своїх сусідів, або коли вони зіткнуться зі структурним або хімічним дефектом. Якщо такі дефекти дуже рідкісні, або якщо вони не мають впливу на процеси дифузії, то густина острівців визначається першими двома процесами (дифузією та осадженням), тобто однорідним зародкоутворенням. У разі коли дефекти більш рясні ніж однорідна острівцева густина, або якщо вони мають сильний вплив на дифузію адатомів, то таке зародкоутворення називають неоднорідним.

У разі однорідного зародкоутворення, сформовані кластери можуть як залишатись стабільними, так і розпадатися знову. Усе залежить від енергії бокового зв’язку E_B рис.16 та кількості сусідів. Стабільний зародок являє собою кластер, який має достатньо великі розміри для того, щоб рости більш швидко ніж розпадатись під

час дифузії. Критичний розмір кластеру i рис.16 визначається числом атомів в найменшому стабільному зародку мінус один, іншими словами, приєднання ще одного атому обертає критичний кластер в стабільний.

Форма кластеру в 2D форматі визначається рухомістю скупчення адатомів уздовж його границі (E_e рис.16). Слід зауважити, що це не єдиний фактор, оскільки дифузія може відбуватись у різноманітних напрямках. Низька рухомість призводить до розгалужених кластерів з фрактальною розмірністю схожих на льодяний кристал. Нерівномірна дифузія, яка спричинена за рахунок симетричного субстрату та нерівномірного приклеювання до границь, може призвести до формування моноатомних дротів. У разі, якщо швидкість дифузії біля границі доволі висока у порівнянні зі швидкістю росту, то формуються густі 2D кластери. При цьому, термодинамічна рівновага з’являється, коли активується перетин кутів або перегинів.

Ще одним важливим процесом для морфології плівок є адатомний спуск на острівцевих сходинках. Аналогічно до кутового перетину в двох вимірах, який визначав напрямок кроків, міжшарова дифузія визначає, які плівки матимуть пласку форму, а які зростатимуть в 3D форматі. Швидкість або вірогідність такого роду дифузії визначає, які із осаджених атомів спустяться до рівня субстрату. Якщо атоми не можуть спуститись, то вони почнуть рухатись в напрямку інших атомів, і у разі i=1, стабільні зародки на вершині острівця спричинять збільшення висоти 3D кластера. У разі, коли атоми матимуть достатньо велику швидкість спуску, наприклад у випадку знаходження єдиного атому на вершині, острівці залишатимуться в 2D розмірності, стимулюючи пошарове зростання.

Досягаючи низхідної сходинки, адатом стикається з бар’єром, де E_s зазвичай більша ніж E_m. Міжшарова дифузія також може відбуватись за рахунок обміну атомів, коли атом, який знаходиться нижче, виштовхується нагору, а його місце займає інший.

Розрахунки, які базуються лише на основних принципах квантової механіки, без використання даних із експериментів, дають основне уявлення про те, чому

дифузія з обміном атомів характерна лише для деяких орієнтацій та комбінацій елементів. У той час як для інших випадків, міжшарова дифузія є лише простим “скочуючим” процесом.

Якщо узагальнити, то можна виділити деякі основні процеси при формуванні, такі як дифузія або осадження. Дифузія може відбуватись в різних напрямках в залежності від енергії, яка накопичується в місцях формування острівців або на субстраті. Також дифузію можна поділити на багато видів, як наприклад, міжшарова дифузія, рух окремого атому в напрямку до острівця, дифузія з обміном та інші. До того ж дифузії може взагалі не бути, якщо атоми, після осадження, досягнуть термодинамічної рівноваги.

На основі цього буде побудована математична модель та її подальша реалізація за допомогою методу Монте-Карло.

2.3 Математична модель

Однією з найпростіших моделей з точним розрахунком є модель випадкового осадження. У даній моделі шорсткість не має великого прояву, а плівка росте нескінченно. Головне, щоб кожна частинка осаджувалась на випадковому місці сітки розміру L, швидко займаючи позицію на поверхні, як показано на рис.17.

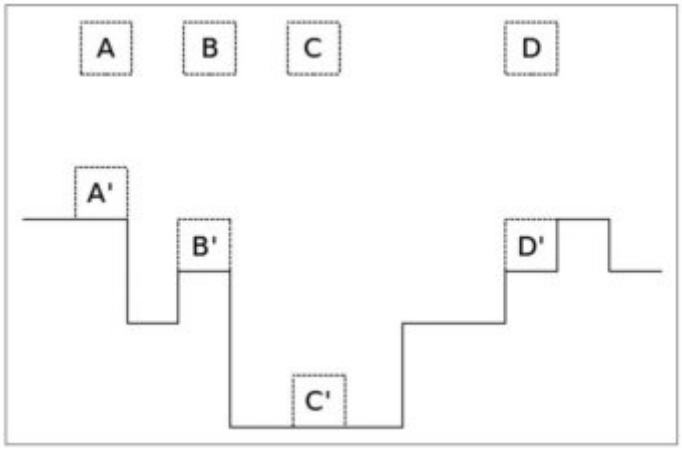


рис.17

Для цієї моделі припускається відсутність пор на поверхні, а для зображення еволіції плівки у кожний момент часу використовується стохастичне рівняння. При цьому зміна висоти $h(x,t)$ може виражати статистичні характеристики:

$$\frac{\partial h(x,t)}{\partial t} = \Phi(x,t) \tag{1}$$

$\Phi(x,t)$ – число частинок, що генеруються на позицію x в одиницю часу t. Вираз $\Phi(x,t)$, у зв'язку з функцією випадковості, може бути виражений як сума двох виразів і набуде вигляду:

$$\frac{\partial h(x,t)}{\partial t} = F + \eta(x,t) \tag{2}$$

F-середнє число частинок в одиницю часу, які додаються на позицію x та "шум" $\eta(x,t)$, який виражає відхилення випадкової величини:

$$< \eta(x,t) >= 0 \tag{3}$$

Наступний момент такого "шуму":

$$< \eta(x,t)\,\eta(x',t') >= C\delta(x - x')(t - t') \tag{4}$$

Інтегруючи рівняння (2) за часом:

$$h(x,t) = Ft + \int_0^1 \eta(x,t')\,dt' \tag{5}$$

Середня висота плівки матиме вигляд:

$$< h(x,t) >= Ft \tag{6}$$

Якщо рівняння (5) піднести до квадрату, то отримаємо:

$$< h^2(x,t) >= F^2t^2 + Ct \tag{7}$$

Тут ширина поверхні $\omega^2(t) = Ct$. Звідси випливає:

$$< \omega^2(t) > \sim t^{\frac{1}{2}} \tag{8}$$

Загалом, ця модель далека від реальних умов, а також нехтує шорсткістю плівки. Варто додати функцію, яка буде враховувати можливість руху частинок та взаємодію осілих атомів з вільними клітинками або сусідами. При осадженні частинка з більшою вірогідністю потрапить на вищий стовпець, але також може зміститись на одну клітинку рис.18.

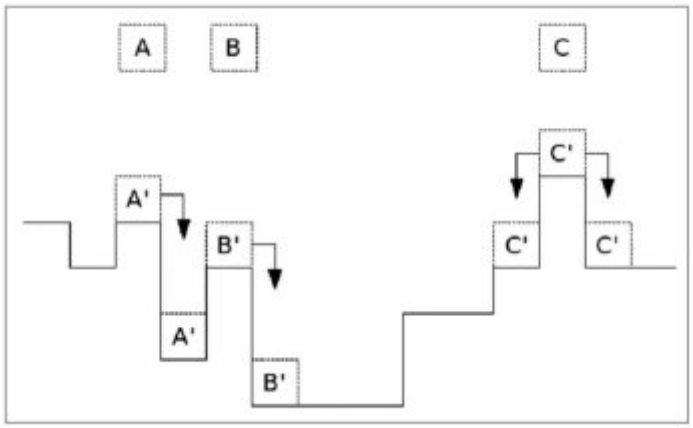


рис.18

Якщо стовпці мають однакову висоту, то частинка рівновірогідно потрапить на один із них. Модель не має точного розв’язку, оскільки неможливо передбачити, що відбудеться у випадковому процесі, але її можливо описати стохастичним диференціальним рівнянням. Із цього рівняння можливо визначити показники моделі зростання. Також ця модель не зображує пористість в плівці.

Диференціальне рівняння, яке описує умови осадження матиме наступний вигляд

$$\frac{\partial h(x,t)}{\partial t} = G(h,x,t) + \eta(x,t) \qquad (9)$$

У зв’язку з урахуванням максимальної висоти шорсткості плівки до $G(h,x,t)$ додається аргумент h. Залежність від позиції x зумовлена кореляцією між точкою осадження та сусідніми точками. Також розглядаються такі умови осадження як:

1.Часова інваріантність: $T \to t + dt$ при $G = G(h,x)$.

2.Поступальна інваріантність (напрямок зростання): $h \to h + \delta h$ у разі, коли $G = G\left(\tilde{N}hn,x\right)$

3.Поступальна інваріантність (у напрямку ортогональному до зростання): $x \to x + \delta x$, у цьому випадку $G = G\left(\tilde{N}h^n\right)$.

4.Інверсивна або обертальна симетрія навколо напрямку зростання: щоб задовольнити таку умову, мають бути відсутні похідні непарних порядків, тому $G = G\left(\tilde{N}^{2K}h\right)$ або $G = G(\nabla^{2K+1}h)^{2l}$.

5.Симетрія вгору/вниз, $h \to - h$.

З урахуванням перелічених умов симетрії, рівняння, яке описує модель, набуває стохастичної форми:

$$\frac{\partial h(\vec{x},t)}{\partial t} = \nabla^2 h + \nabla^4 h + ... + \nabla^{2n} h + \nabla^2 h\left(\nabla h^2\right) + \left(\nabla^{2k} h\right)(\nabla h)^{2y} + \eta(\vec{x},t) \qquad (10)$$

Зважаючи на гідродинамічні характеристики зростання плівки, можна відкинути члени вищих порядків, тоді рівняння матиме вигляд:

$$\frac{\partial h(\vec{x},t)}{\partial t} = \nu\nabla^2 h + \eta(\vec{x},t) \qquad (11)$$

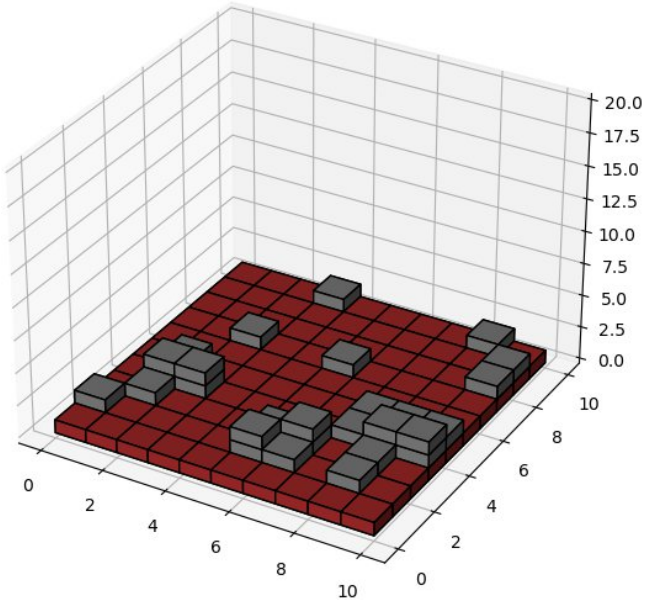
Вираз $\nu\nabla^2 h$–поверхневий натяг, а $\eta(\vec{x},t)$ - “білий шум”.

2.4 Програмна реалізація

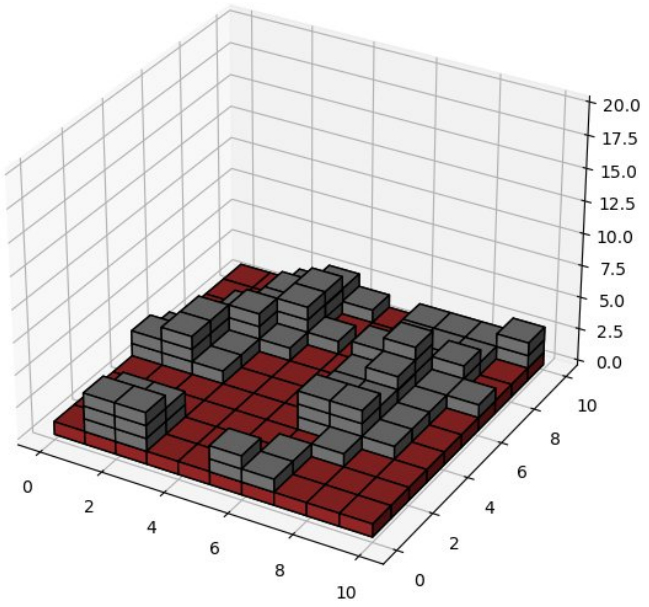
Для того, щоб зобразити процес осадження, випадковим чином генеруються точки на поверхні. Вірогідності визначаються згідно з енергією вже осілих точок або кластерів. Якщо точка (атом) знаходиться окремо від інших, то він починає рух (процес дифузії), а в разі зіткнення зі своїм сусідом зупиняється (початок зародкоутворення). Після того як з’явились перші острівці, ті атоми, які знаходяться на найвищих точках, спускаються на одну сходинку нижче (пошарова дифузія). Якщо

процес повторити багато разів, то можна побачити 3D зростання плівки за механізмом Вольмера-Вебера.

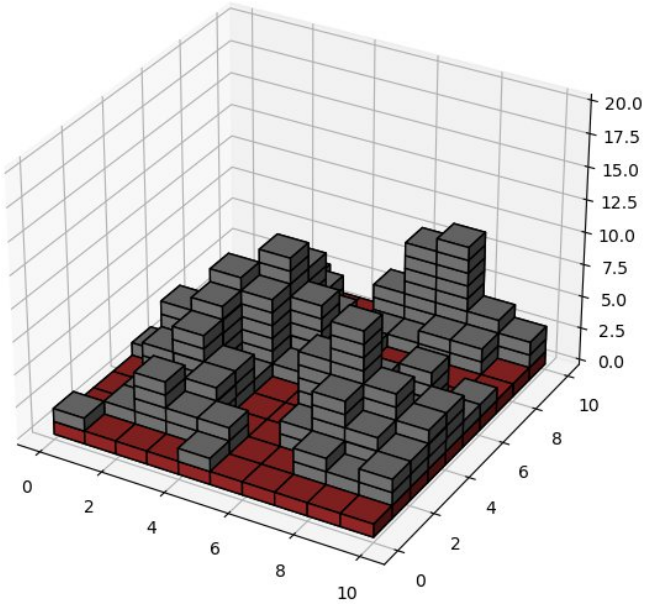
Для початку проведемо дослід, де розглянутий лише процес осадження.



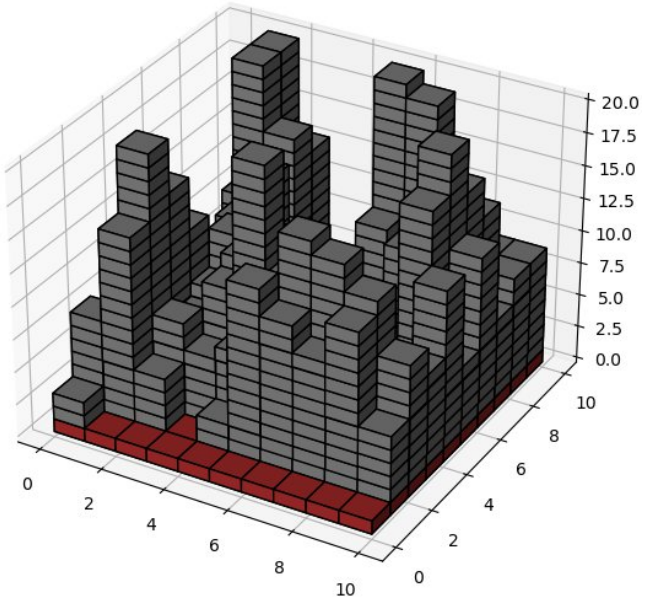
30 ітерацій



100 ітерацій



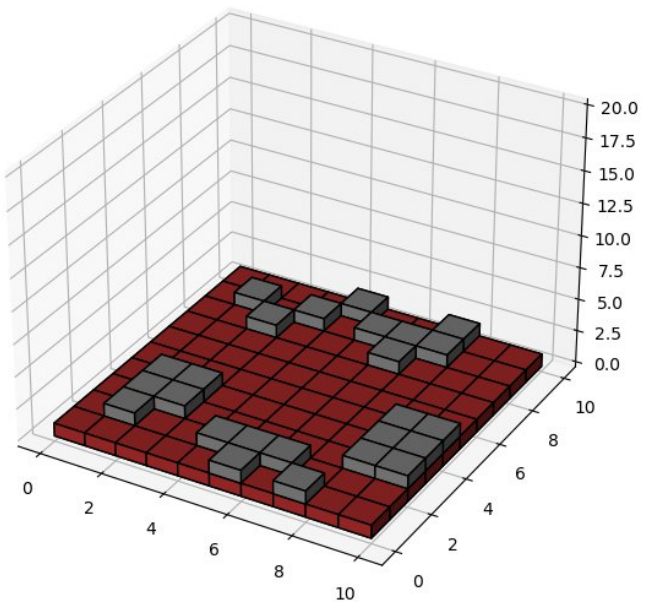
200 ітерацій



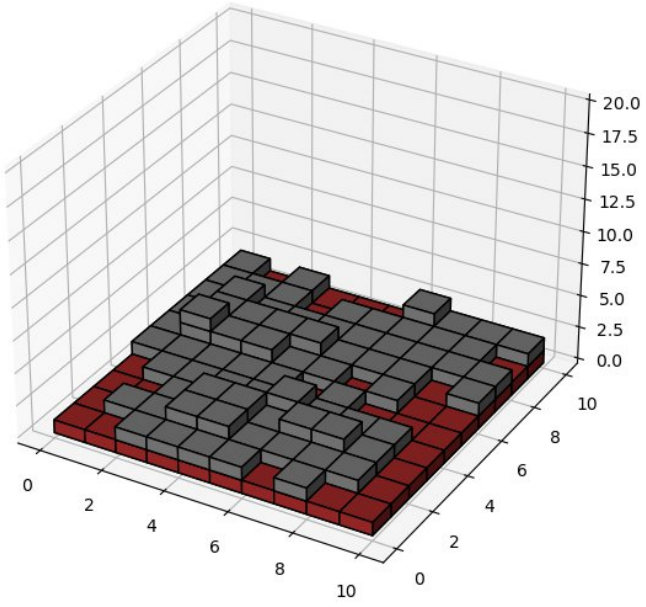
1000 ітерацій

рис.19

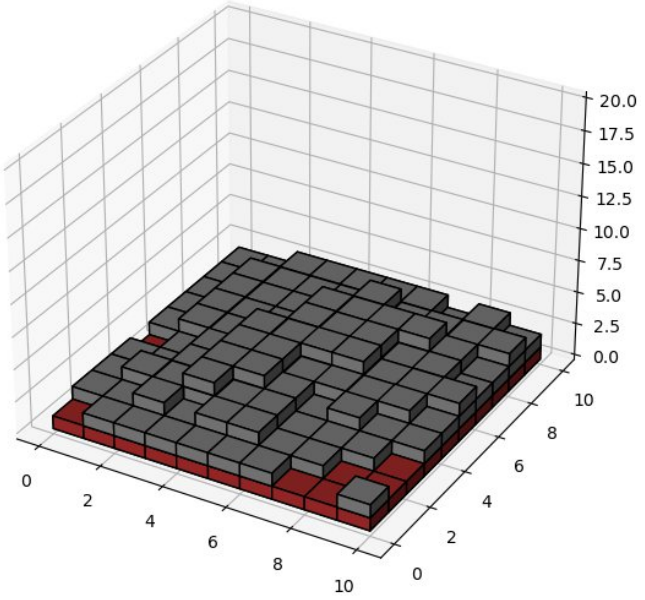
Бачимо появу стовпців, які потім починають зростатись. Для того, щоб отримати більш точні результати та наблизити процес моделювання до реальних умов додамо подію міжшарової дифузії, тобто можливість атомів спускатись на нижчу сходинку.



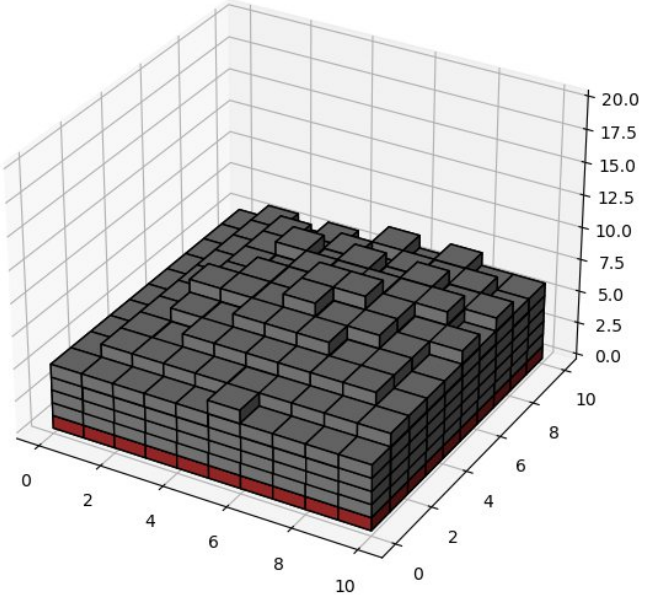
30 ітерацій



100 ітерацій



200 ітерацій

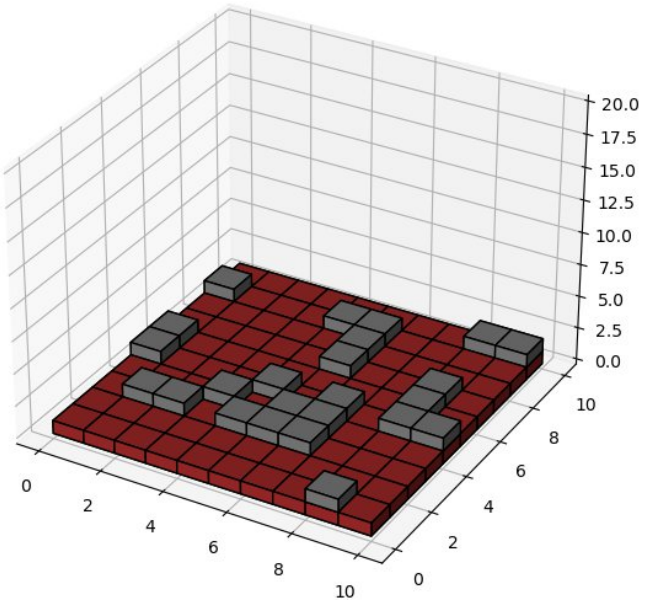


1000 ітерацій

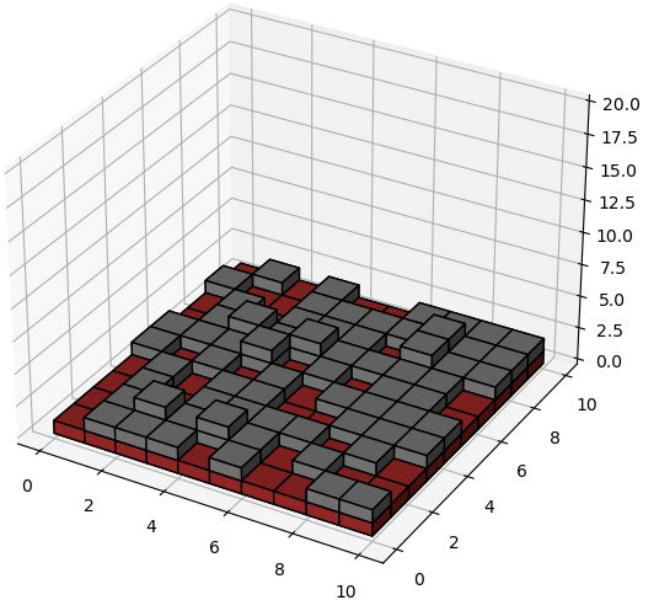
рис.20

Графіки почали суттєво відрізнятись. Стовпці стали більше набувати пірамідоподібної форми. Проте принцип, який стосується поєднання окремих стовпців в єдиний, залишається незмінним.

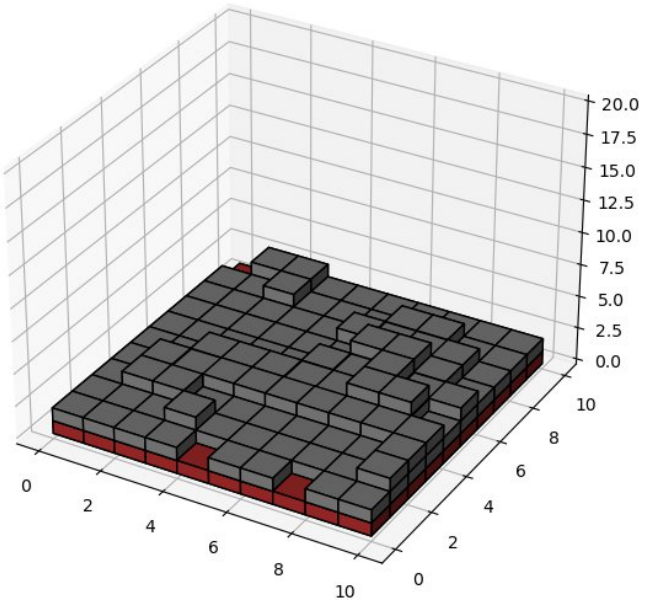
Окрім цього атомам слід додати можливість не лише спускатись, але й рухатись уздовж тераси. Спробуємо це зробити та поглянемо, що вийде.



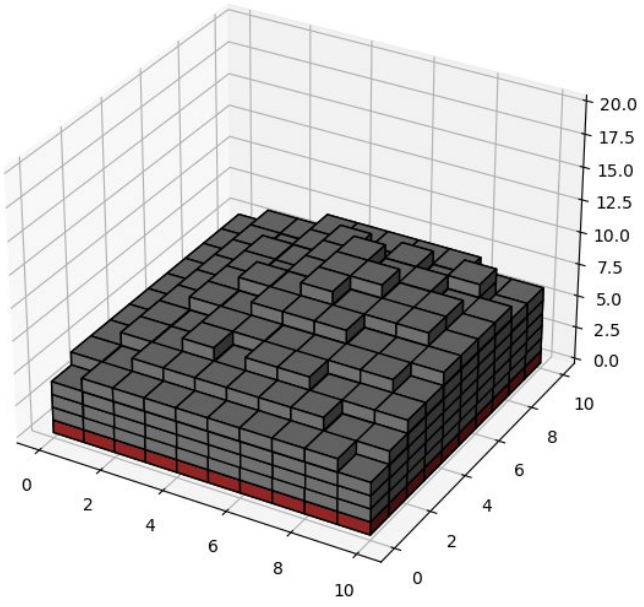
30 ітерацій



100 ітерацій



200 ітерацій

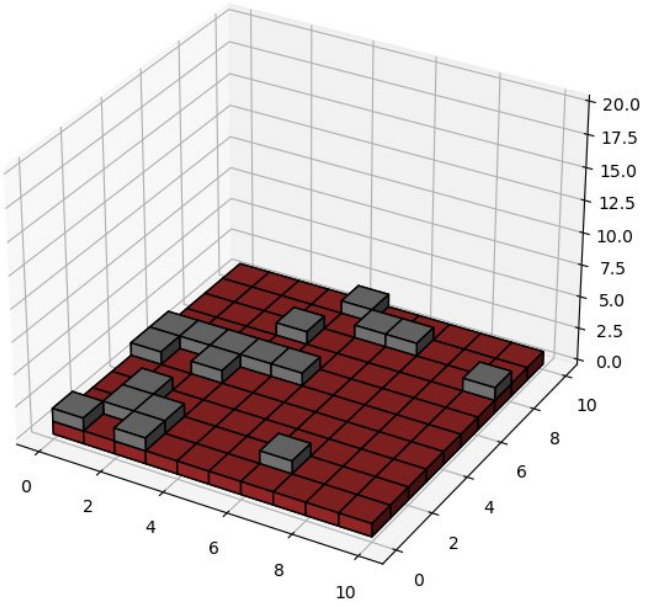


1000 ітерацій

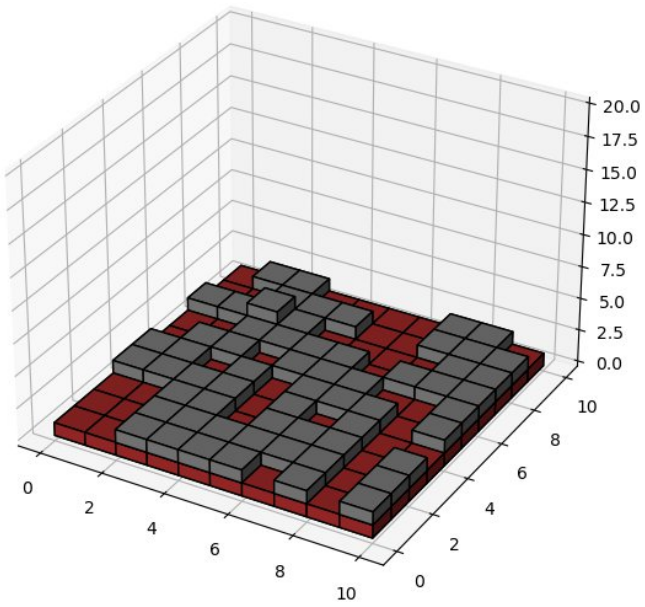
рис.21

На початку формування атоми щільніше заповнюють поверхню. Це можна побачити на сотій ітерації, де ще й досі продовжується утворення другого поверху, оскільки стовпці почали поєднуватись на дуже ранніх етапах.

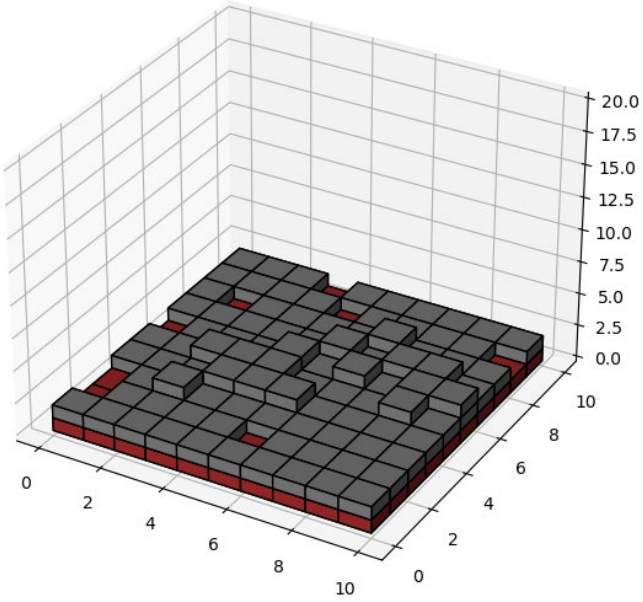
Окрім самої дифузії має значення і її інтенсивність. Збільшимо інтенсивність в 5 разів та прослідкуємо за результатом.



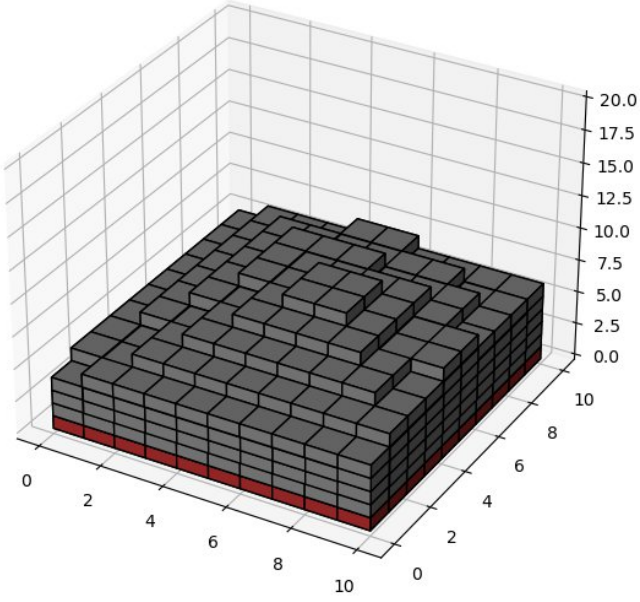
30 ітерацій



100 ітерацій



200 ітерацій



1000 ітерацій

рис.22

Другий поверх став формуватись ще пізніше. Збільшується вірогідність зміни механізму зростання Вольмера-Вебера на механізм Странського-Кристанова.

ВИСНОВКИ

1.Проведений аналіз фізичних процесів формування плівкових наноструктур нітридів перехідних металів, а саме: осадження, зародкоутворення, агрегації, дисоціації, дифузії уздовж бар’єру, дифузії уздовж тераси, а також міжшарової дифузії. При цьому проаналізована математична модель, де враховані деякі із наведених процесів.

2.Проведене вдосконалення математичної моделі та її реалізації, а саме доданий процес міжшарової дифузії. Атоми “скочуються” на нижчі сходинки, що наближує комп’ютерний експеримент до реальних умов та дає більш точні результати у порівнянні з наявністю лише самого осадження.

3.У результаті моделювання та програмної реалізації було досліджене формування плівки нітридів перехідних металів. Були отримані стовпці згідно з механізмом зростання Вольмера-Вебера. Спочатку відбувається утворення окремих стовпців, які потім поєднуються в більші та поступово завершують появу готових моношарів. Досліджений вплив дифузії уздовж тераси, тобто руху окремих атомів між сформованими кластерами, коли перший моношар ще не утворився. При наявності такого процесу зародки починають з’являтись ще на дуже ранніх етапах.

4.Проведені дослідження впливу не лише наявності дифузії, але і її інтенсивності, тобто поведінки зростання плівки при збільшенні швидкості переміщення осаджених атомів перед появою нових. Якщо збільшити інтенсивність, то збільшиться ймовірність зміни механізму зростання Вольмера-Вебера на механізм Странського-Кристанова.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕЛЕЛ

1.“Thin Films/Properties and Applications”, Edwin Acosta.

2.“Nitrides”, Peter Ettmayer, Walter Lengauer.

3.“Structural regularities of the formation of nitride and boride coatings based on transition metals”, Alexander Goncharov, Andrei Yunda, Ivan Kolinko, Olga Maksakova.

4.“Texture evolution in metal nitride (aluminum nitride, titanium nitride, hafnium nitride) thin films prepared by off-normal incidence reactive magnetron sputtering”, Darya Deniz.

5.“Self-patterned oxide nanostructures grown by post-deposition thermal annealing on stepped surfaces”, R Bachelet, S Cottrino, G Nahelou, V Coudert, A Boulle, ´ B Soulestin, F Rossignol, R Guinebretiere and A Dauger.

6.“Molecular Dynamic Simulation of Thin Film Growth Stress Evolution”, Haifeng Zheng.

7.“STRESSES AND FAILURE MODES IN THIN FILMS AND MULTILAYERS”, John W.Hutchinson.

8.“Growth mechanism of TiN film on dielectric films and the effects on the work function”, K. Choia,T, P. Lysaghta , H. Alshareef b , C. Huffmanb , H.-C. Wena , R. Harrisc , H. Luand , P.-Y. Hunga , C. Sparksa , M. Cruza , K. Matthewsa , P. Majhie , B.H. Lee.

9.“Nanocolumnar TiN thin film growth by oblique angle sputter-deposition: Experiments vs. simulations”, Boudjemaa Bouaouina, Cédric Mastail, Aurélien Besnard, Rubenson Mareus, Florin Nita, Anny Michel, Grégory Abadias.

10.“Effect of temperature on the growth of TiN thin films by oblique angle sputter deposition: A three-dimensional atomistic computational study”, Rubenson Mareus, Cedric Mastail, Florin Nita, Anny Michel, Gregory Abadias.

11.“Gold Nitride: Preparation and Properties”, L.Alves, A.C. Brieva, Yu.V.Butenko, M.R.C. Hunt.

12.“Simulation at high temperature of atomic deposition, islands coalescence, Ostwald and inverse Ostwald ripening with a general simple kinetic Monte Carlo code”, S. Lucas, P. Moskovkin.

13.“Characterization of titanium nitride films prepared by d.c. reactive magnetron sputtering at different nitrogen pressures», Li-Jian Meng, M.P. dos Santos.

14.“Volmer-Weber growth stages of polycystalline metal films probed by situ and real-time optical diagnostics”, Gregory Abadias, Lionel Simonot, Anny Michel, Sophie Camelio.

15. “Epitaxial Growth of Thin Films”, Harald Brune.

16. “A brief review of mathematical models of thin film growth and surfaces: A possible route to avoid defects in stents”, Fabricio Luchesi Forgerini, Roberto Marchiori.

ДОДАТОК А. Код програми, де не враховується процес дифузії

```
import random as rn
import numpy as np
import matplotlib.pyplot as plt

class Sizes:
    def __init__(self):
        self.borders = 3
        self.width = 8 + 2*self.borders
        self.length = 8 + 2*self.borders
        self.height = 8
    def set_sizes(self, width, length, height):
        self.width = width + 2*self.borders
        self.length = length + 2*self.borders
        self.height = height
class Lining:
    def __init__(self):
        self.borders = sizes.borders
        self.width = sizes.width
        self.length = sizes.length
        self.height = sizes.height
        self.lining = np.zeros((self.width,  self.length, self.height))
        self.lining[:, :, 0:1] = 1
class Field:
    def __init__(self):
        self.borders = sizes.borders
        self.width = sizes.width
        self.length = sizes.length
        self.height = sizes.height
        self.field = np.zeros((self.width,  self.length, self.height
        self.Ei = 1.68e-19
        self.k = 1.38e-23
        self.T = 1000
        self.kT = self.k*self.T
        self.attempt = 0
    def new_particle(self):
        x = rn.randint(self.borders, self.width - self.borders)
        y = rn.randint(self.borders, self.length - self.borders)
        P0 = self.probability(x, y)
        Pfront = self.probability(x - 1, y)
        PfrontRight = self.probability(x - 1, y + 1)
        Pright = self.probability(x, y + 1)
        PrightBack = self.probability(x + 1, y + 1)
        Pback = self.probability(x + 1, y)
        PbackLeft = self.probability(x + 1, y - 1)
        Pleft = self.probability(x, y - 1)
        PleftFront = self.probability(x - 1, y - 1)
        P = P0 + Pfront + PfrontRight + PrightBack + Pback + PbackLeft + Pleft + PleftFront
        P0 = P0 / P
        Pfront = Pfront / P
        PfrontRight = PfrontRight / P
```

```
        Pright = Pright / P
        PrightBack = PrightBack / P
        Pback = Pback / P
        PbackLeft = PbackLeft / P
        Pleft = Pleft / P
        PleftFront = PleftFront / P
        r = rn.random()
        if 0 <= r < P0:
            self.set_point(x, y)
        if P0 <= r < P0 + Pfront:
            self.set_point(x-1, y)
        if P0 + Pfront <= r < P0 + Pfront + PfrontRight:
            self.set_point(x-1, y+1)
        if P0 + Pfront + PfrontRight <= r < P0 + Pfront + PfrontRight + Pright:
            self.set_point(x, y+1)
        if P0 + Pfront + PfrontRight + Pright <= r < P0 + Pfront + PfrontRight + Pright + PrightBack:
            self.set_point(x+1, y+1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack <= r < P0 + Pfront + PfrontRight + Pright + PrightBack +
Pback:
            self.set_point(x+1, y)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback <= r < P0 + Pfront + PfrontRight + Pright +
PrightBack + Pback + PbackLeft:
            self.set_point(x+1, y-1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback + PbackLeft <= r < P0 + Pfront + PfrontRight +
Pright + PrightBack + Pback + PbackLeft + Pleft:
            self.set_point(x, y-1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback + PbackLeft + Pleft <= r < P0 + Pfront +
PfrontRight + Pright + PrightBack + Pback + PbackLeft + Pleft + PleftFront:
            self.set_point(x-1, y-1)
    def set_point(self, i, j):
        h = self.get_height(i, j) + 1
        if self.check_height(h):
            self.attempt = 0
            self.field[i, j, h] = 1
        else:
            if 0 <= self.attempt < 100:
                self.attempt += 1
                self.new_particle()
            if 100 <= self.attempt < 900:
                x = rn.randint(self.borders, self.width - self.borders)
                y = rn.randint(self.borders, self.length - self.borders)
                self.attempt += 1
                self.set_point(x, y)
            else:
                self.attempt = 0
    def probability(self, i, j):
        e = 0
        h = self.get_height(i - 1, j)
        e += h if self.check_height(h) else 0
        h = self.get_height(i, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j)
```

```
        e += h if self.check_height(h) else 0
        h = self.get_height(i, j - 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i - 1, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j - 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i - 1, j - 1)
        e += h if self.check_height(h) else 0
        return np.exp(e*self.Ei/self.kT)
    def get_height(self, i, j):
        return sum(self.field[i, j, :] == 1)
    def check_height(self, h):
        return h < self.height
    def check_sum(self):
        return sum(sum(self.field[:, :, self.height - 1])) < (self.width - 5) * (self.length - 5)
class Draw:
    def __init__(self):
        self.ax = plt.figure().add_subplot(projection='3d')
    def show_lining(self):
        self.ax.voxels(
            lining.lining[lining.borders:lining.width-lining.borders, lining.borders:lining.length-lining.borders, :],
            facecolors='brown', edgecolor='k')
    def show_field(self):
        self.ax.voxels(field.field[field.borders:field.width-field.borders, field.borders:field.length-field.borders, :],
                facecolors='grey', edgecolor='k')
sizes = Sizes()
sizes.set_sizes(width=30, length=30, height=30)
lining = Lining()
field = Field()
draw = Draw()
draw.show_lining()
n = 0
while True:
    if field.check_sum():
        field.new_particle()
        if n % (((field.width-2*field.borders)*(field.length-2*field.borders))//4) == 0:
            draw.show_field()
            plt.draw()
            plt.pause(1e-16)
        n += 1
    else:
        break
draw.show_field()
plt.show()
```

ДОДАТОК Б. Код програми, де враховується процес міжшарової дифузії

```
import random as rn
import numpy as np
import matplotlib.pyplot as plt

class Sizes:
    def __init__(self):
        self.borders = 3
        self.width = 8 + 2*self.borders
        self.length = 8 + 2*self.borders
        self.height = 8
    def set_sizes(self, width, length, height):
        self.width = width + 2*self.borders
        self.length = length + 2*self.borders
        self.height = height
class Lining:
    def __init__(self):
        self.borders = sizes.borders
        self.width = sizes.width
        self.length = sizes.length
        self.height = sizes.height
        self.lining = np.zeros((self.width,  self.length, self.height))
        self.lining[:, :, 0:1] = 1
class Field:
    def __init__(self):
        self.borders = sizes.borders
        self.width = sizes.width
        self.length = sizes.length
        self.height = sizes.height
        self.field = np.zeros((self.width,  self.length, self.height))
        self.Ei = 1.68e-19
        self.k = 1.38e-23
        self.T = 1000
        self.kT = self.k*self.T
        self.attempt = 0
    def diffusion(self):
        for h in range(self.height-2, 0, -1):
            indices = np.where(self.field[:, :, h] == 1)
            if indices[0].any() and indices[1].any():
                indices_xy = list(zip(indices[0], indices[1]))
                rn.shuffle(indices_xy)
                for xy in indices_xy:
                    if self.field[xy[0], xy[1], h+1] == 0:
                        h0 = self.get_height(xy[0], xy[1])
                        hs = []
                        columns = []
                        for i in range(-1, 2):
                            for j in range(-1, 2):
                                if i != 0 and j != 0:
                                    try:
                                        h1 = self.get_height(xy[0]+i, xy[1]+j)
```

```
                    except IndexError:
                        h1 = h0
                    if h0 > 1:
                        if h1 < h0-1:
                            hs.append(h1+1)
                            columns.append([xy[0]+i, xy[1]+j])
            if hs:
                probabilities = self.get_diffusion_probabilities(hs)
                column_number = self.choose_column_number(probabilities)
                h_chose = self.get_height(columns[column_number][0], columns[column_number][1])
                self.field[columns[column_number][0], columns[column_number][1], h_chose + 1] = 1
                self.field[xy[0], xy[1], h0] = 0
                break
    def choose_column_number(self, probabilities):
        p = list(probabilities)
        r = rn.random()
        if p[0] and \
                r < p[0]: return 0
        if p[0] and p[1] and \
                p[0] <= r < p[0]+p[1]: return 1
        elif p[0] and p[1] and p[2] and \
                p[0]+p[1] <= r < p[0]+p[1]+p[2]: return 2
        elif p[0] and p[1] and p[2] and p[3] and \
                p[0]+p[1]+p[2] <= r < p[0]+p[1]+p[2]+p[3]: return 3
        elif p[0] and p[1] and p[2] and p[3] and p[4] and \
                p[0]+p[1]+p[2]+p[3] <= r < p[0]+p[1]+p[2]+p[3]+p[4]: return 4
        elif p[0] and p[1] and p[2] and p[3] and p[4] and p[5] and \
                p[0]+p[1]+p[2]+p[3]+p[4] <= r < p[0]+p[1]+p[2]+p[3]+p[4]+p[5]: return 5
        elif p[0] and p[1] and p[2] and p[3] and p[4] and p[5] and p[6] and \
                p[0]+p[1]+p[2]+p[3]+p[4]+p[5] <= r < p[0]+p[1]+p[2]+p[3]+p[4]+p[5]+p[6]: return 6
        elif p[0] and p[1] and p[2] and p[3] and p[4] and p[5] and p[6] and p[7] and \
                p[0]+p[1]+p[2]+p[3]+p[4]+p[5]+p[6] <= r < p[0]+p[1]+p[2]+p[3]+p[4]+p[5]+p[6]+p[7]: return 7
        return 0
    def get_diffusion_probabilities(self, hs):
        probabilities = []
        s = sum(hs)
        for h in hs:
            probabilities.append(h/s)
        return probabilities
    def new_particle(self):
        x = rn.randint(self.borders, self.width - self.borders)
        y = rn.randint(self.borders, self.length - self.borders)
        P0 = self.energy(x, y)
        Pfront = self.energy(x - 1, y)
        PfrontRight = self.energy(x - 1, y + 1)
        Pright = self.energy(x, y + 1)
        PrightBack = self.energy(x + 1, y + 1)
        Pback = self.energy(x + 1, y)
        PbackLeft = self.energy(x + 1, y - 1)
        Pleft = self.energy(x, y - 1)
        PleftFront = self.energy(x - 1, y - 1)
        P = P0 + Pfront + PfrontRight + PrightBack + Pback + PbackLeft + Pleft + PleftFront
```

```
        P0 = P0 / P
        Pfront = Pfront / P
        PfrontRight = PfrontRight / P
        Pright = Pright / P
        PrightBack = PrightBack / P
        Pback = Pback / P
        PbackLeft = PbackLeft / P
        Pleft = Pleft / P
        PleftFront = PleftFront / P
        r = rn.random()
        if 0 <= r < P0:
            self.set_point(x, y)
        if P0 <= r < P0 + Pfront:
            self.set_point(x-1, y)
        if P0 + Pfront <= r < P0 + Pfront + PfrontRight:
            self.set_point(x-1, y+1)
        if P0 + Pfront + PfrontRight <= r < P0 + Pfront + PfrontRight + Pright:
            self.set_point(x, y+1)
        if P0 + Pfront + PfrontRight + Pright <= r < P0 + Pfront + PfrontRight + Pright + PrightBack:
            self.set_point(x+1, y+1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack <= r < P0 + Pfront + PfrontRight + Pright + PrightBack +
Pback:
            self.set_point(x+1, y)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback <= r < P0 + Pfront + PfrontRight + Pright +
PrightBack + Pback + PbackLeft:
            self.set_point(x+1, y-1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback + PbackLeft <= r < P0 + Pfront + PfrontRight +
Pright + PrightBack + Pback + PbackLeft + Pleft:
            self.set_point(x, y-1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback + PbackLeft + Pleft <= r < P0 + Pfront +
PfrontRight + Pright + PrightBack + Pback + PbackLeft + Pleft + PleftFront:
            self.set_point(x-1, y-1)
    def set_point(self, i, j):
        h = self.get_height(i, j) + 1
        if self.check_height(h):
            self.attempt = 0
            self.field[i, j, h] = 1
        else:
            if 0 <= self.attempt < 100:
                self.attempt += 1
                self.new_particle()
            if 100 <= self.attempt < 900:
                x = rn.randint(self.borders, self.width - self.borders)
                y = rn.randint(self.borders, self.length - self.borders)
                self.attempt += 1
                self.set_point(x, y)
            else:
                self.attempt = 0
    def energy(self, i, j):
        e = 0
        h = self.get_height(i - 1, j)
        e += h if self.check_height(h) else 0
```

```
        h = self.get_height(i, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j)
        e += h if self.check_height(h) else 0
        h = self.get_height(i, j - 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i - 1, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j - 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i - 1, j - 1)
        e += h if self.check_height(h) else 0
        return np.exp(e*self.Ei/self.kT)
    def get_height(self, i, j):
        return sum(self.field[i, j, :] == 1)
    def check_height(self, h):
        return h < self.height
    def check_sum(self):
        return sum(sum(self.field[:, :, self.height - 1])) < (self.width - 5) * (self.length - 5)
class Draw:
    def __init__(self):
        self.ax = plt.figure().add_subplot(projection='3d')
    def show_lining(self):
        self.ax.voxels(
            lining.lining[lining.borders:lining.width-lining.borders, lining.borders:lining.length-lining.borders, :],
            facecolors='brown', edgecolor='k')
    def show_field(self):
        self.ax.voxels(field.field[field.borders:field.width-field.borders, field.borders:field.length-field.borders, :],
                facecolors='grey', edgecolor='k')
sizes = Sizes()
sizes.set_sizes(width=10, length=10, height=20)
lining = Lining()
field = Field()
draw = Draw()
draw.show_lining()
n = 1
while True:
    if field.check_sum():
        field.diffusion()
        field.new_particle()
        if n % (((field.width-2*field.borders)*(field.length-2*field.borders))//4) == 0:
            draw.show_field()
            plt.draw()
            plt.pause(1e-16)
        n += 1
    else:
        break
draw.show_field()
plt.show()
```

ДОДАТОК В. Код програми, де враховуються процеси міжшарової дифузії та дифузії уздовж тераси

```
import random as rn
import numpy as np
import matplotlib.pyplot as plt

class Sizes:
    def __init__(self):
        self.borders = 3
        self.width = 8 + 2*self.borders
        self.length = 8 + 2*self.borders
        self.height = 8
    def set_sizes(self, width, length, height):
        self.width = width + 2*self.borders
        self.length = length + 2*self.borders
        self.height = height
class Lining:
    def __init__(self):
        self.borders = sizes.borders
        self.width = sizes.width
        self.length = sizes.length
        self.height = sizes.height
        self.lining = np.zeros((self.width,  self.length, self.height))
        self.lining[:, :, 0:1] = 1
class Field:
    def __init__(self):
        self.borders = sizes.borders
        self.width = sizes.width
        self.length = sizes.length
        self.height = sizes.height
        self.field = np.zeros((self.width,  self.length, self.height))
        self.Ei = 1.68e-19
        self.k = 1.38e-23
        self.T = 1000
        self.kT = self.k*self.T
        self.attempt = 0
    def diffusion(self):
        for h in range(self.height-2, 0, -1):
            indices = np.where(self.field[:, :, h] == 1)
            if indices[0].any() and indices[1].any():
                indices_xy = list(zip(indices[0], indices[1]))
                rn.shuffle(indices_xy)
                for xy in indices_xy:
                    if self.field[xy[0], xy[1], h+1] == 0:
                        h0 = self.get_height(xy[0], xy[1])
                        hs = []
                        columns = []
                        for i in range(-1, 2):
                            for j in range(-1, 2):
                                if i != 0 and j != 0:
                                    try:
```

```
                        h1 = self.get_height(xy[0]+i, xy[1]+j)
                    except IndexError:
                        h1 = h0
                    if h0 > 1:
                        if h1 < h0-1:
                            hs.append(h1+1)
                            columns.append([xy[0]+i, xy[1]+j])
                    else:
                        if h1 < h0:
                            hs.append(h1 + 1)
                            columns.append([xy[0] + i, xy[1] + j])
            if hs:
                probabilities = self.get_diffusion_probabilities(hs)
                column_number = self.choose_column_number(probabilities)
                h_chose = self.get_height(columns[column_number][0], columns[column_number][1])
                self.field[columns[column_number][0], columns[column_number][1], h_chose + 1] = 1
                self.field[xy[0], xy[1], h0] = 0
                break
    def choose_column_number(self, probabilities):
        p = list(probabilities)
        r = rn.random()
        if p[0] and \
                r < p[0]: return 0
        if p[0] and p[1] and \
                p[0] <= r < p[0]+p[1]: return 1
        elif p[0] and p[1] and p[2] and \
                p[0]+p[1] <= r < p[0]+p[1]+p[2]: return 2
        elif p[0] and p[1] and p[2] and p[3] and \
                p[0]+p[1]+p[2] <= r < p[0]+p[1]+p[2]+p[3]: return 3
        elif p[0] and p[1] and p[2] and p[3] and p[4] and \
                p[0]+p[1]+p[2]+p[3] <= r < p[0]+p[1]+p[2]+p[3]+p[4]: return 4
        elif p[0] and p[1] and p[2] and p[3] and p[4] and p[5] and \
                p[0]+p[1]+p[2]+p[3]+p[4] <= r < p[0]+p[1]+p[2]+p[3]+p[4]+p[5]: return 5
        elif p[0] and p[1] and p[2] and p[3] and p[4] and p[5] and p[6] and \
                p[0]+p[1]+p[2]+p[3]+p[4]+p[5] <= r < p[0]+p[1]+p[2]+p[3]+p[4]+p[5]+p[6]: return 6
        elif p[0] and p[1] and p[2] and p[3] and p[4] and p[5] and p[6] and p[7] and \
                p[0]+p[1]+p[2]+p[3]+p[4]+p[5]+p[6] <= r < p[0]+p[1]+p[2]+p[3]+p[4]+p[5]+p[6]+p[7]: return 7
        return 0
    def get_diffusion_probabilities(self, hs):
        probabilities = []
        s = sum(hs)
        for h in hs:
            probabilities.append(h/s)
        return probabilities
    def new_particle(self):
        x = rn.randint(self.borders, self.width - self.borders)
        y = rn.randint(self.borders, self.length - self.borders)
        P0 = self.energy(x, y)
        Pfront = self.energy(x - 1, y)
        PfrontRight = self.energy(x - 1, y + 1)
        Pright = self.energy(x, y + 1)
        PrightBack = self.energy(x + 1, y + 1)
```

```
        Pback = self.energy(x + 1, y)
        PbackLeft = self.energy(x + 1, y - 1)
        Pleft = self.energy(x, y - 1)
        PleftFront = self.energy(x - 1, y - 1)
        P = P0 + Pfront + PfrontRight + PrightBack + Pback + PbackLeft + Pleft + PleftFront
        P0 = P0 / P
        Pfront = Pfront / P
        PfrontRight = PfrontRight / P
        Pright = Pright / P
        PrightBack = PrightBack / P
        Pback = Pback / P
        PbackLeft = PbackLeft / P
        Pleft = Pleft / P
        PleftFront = PleftFront / P
        r = rn.random()
        if 0 <= r < P0:
            self.set_point(x, y)
        if P0 <= r < P0 + Pfront:
            self.set_point(x-1, y)
        if P0 + Pfront <= r < P0 + Pfront + PfrontRight:
            self.set_point(x-1, y+1)
        if P0 + Pfront + PfrontRight <= r < P0 + Pfront + PfrontRight + Pright:
            self.set_point(x, y+1)
        if P0 + Pfront + PfrontRight + Pright <= r < P0 + Pfront + PfrontRight + Pright + PrightBack:
            self.set_point(x+1, y+1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack <= r < P0 + Pfront + PfrontRight + Pright + PrightBack +
Pback:
            self.set_point(x+1, y)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback <= r < P0 + Pfront + PfrontRight + Pright +
PrightBack + Pback + PbackLeft:
            self.set_point(x+1, y-1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback + PbackLeft <= r < P0 + Pfront + PfrontRight +
Pright + PrightBack + Pback + PbackLeft + Pleft:
            self.set_point(x, y-1)
        if P0 + Pfront + PfrontRight + Pright + PrightBack + Pback + PbackLeft + Pleft <= r < P0 + Pfront +
PfrontRight + Pright + PrightBack + Pback + PbackLeft + Pleft + PleftFront:
            self.set_point(x-1, y-1)
    def set_point(self, i, j):
        h = self.get_height(i, j) + 1
        if self.check_height(h):
            self.attempt = 0
            self.field[i, j, h] = 1
        else:
            if 0 <= self.attempt < 100:
                self.attempt += 1
                self.new_particle()
            if 100 <= self.attempt < 900:
                x = rn.randint(self.borders, self.width - self.borders)
                y = rn.randint(self.borders, self.length - self.borders)
                self.attempt += 1
                self.set_point(x, y)
            else:
```

```
            self.attempt = 0
    def energy(self, i, j):
        e = 0
        h = self.get_height(i - 1, j)
        e += h if self.check_height(h) else 0
        h = self.get_height(i, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j)
        e += h if self.check_height(h) else 0
        h = self.get_height(i, j - 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i - 1, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j + 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i + 1, j - 1)
        e += h if self.check_height(h) else 0
        h = self.get_height(i - 1, j - 1)
        e += h if self.check_height(h) else 0
        return np.exp(e*self.Ei/self.kT)
    def get_height(self, i, j):
        return sum(self.field[i, j, :] == 1)
    def check_height(self, h):
        return h < self.height
    def check_sum(self):
        return sum(sum(self.field[:, :, self.height - 1])) < (self.width - 5) * (self.length - 5)
class Draw:
    def __init__(self):
        self.ax = plt.figure().add_subplot(projection='3d')
    def show_lining(self):
        self.ax.voxels(
            lining.lining[lining.borders:lining.width-lining.borders, lining.borders:lining.length-lining.borders, :],
            facecolors='brown', edgecolor='k')
    def show_field(self):
        self.ax.voxels(field.field[field.borders:field.width-field.borders, field.borders:field.length-field.borders, :],
                facecolors='grey', edgecolor='k')
sizes = Sizes()
sizes.set_sizes(width=10, length=10, height=20)
lining = Lining()
field = Field()
draw = Draw()
draw.show_lining()
n = 1
while True:
    if field.check_sum():
        field.diffusion()
        field.new_particle()
        if n % (((field.width-2*field.borders)*(field.length-2*field.borders))//4) == 0:
            draw.show_field()
            plt.draw()
            plt.pause(1e-16)
        n += 1
```

```
    else:
        break
draw.show_field()
plt.show()
```