ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут телекомунікацій

Кафедра системного аналізу

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр
на тему: **«МЕТОДИ ТА МОДЕЛІ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ВІРТУАЛЬНИХ ПРОСТОРІВ»**

Виконав: студент 4 курсу, групи _____

спеціальності _____

_____

Керівник _____

Рецензент _____

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
Навчально-науковий інститут телекомунікацій

Кафедра системного аналізу
Ступінь вищої освіти – «Бакалавр»
Спеціальність – 124 Системний аналіз

ЗАТВЕРДЖУЮ
Завідувач кафедри
системного аналізу
_____ Т.Б. Гордієнко
«_____» _____ 2023 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ
Цинковському Д.В.

1. Тема роботи: «Аналіз методів та моделі процедурної генерації віртуальних просторів»

Керівник роботи Резник Сергій Юрійович, асистент кафедри системного аналізу

затверджені наказом вищого навчального закладу від «___» _____ 20__ року №___

2. Строк подання студентом роботи _____

3. Вхідні дані до роботи: метод процедурної генерації Best Fit; науково-технічна література з питань, пов’язаних з процедурною генерацією віртуальних просторів.

4. Зміст розрахунково-пояснювальної записки:

4.1. Основні методи та моделі процедурної генерації віртуальних просторів;

4.2. Методи та моделі процедурної генерації для побудови віртуальних будівель;

4.3. Методи та моделі генерації тривимірної сітки.

5. Перелік графічного матеріалу:

Рисунок 1.1 – Схеми побудови віртуального простору в спрощеному природному порядку (а) та в оптимізованому процедурному (б)

Рисунок 1.2 – Приклад «берегової» лінії

Рисунок 1.3 – Приклад результату алгоритму Форчуна

Рисунок 1.4 – Згенерована текстура методом клітинних автоматів

Рисунок 1.5 – Клітинний автомат із комірками, що зображені у вигляді шестикутника

Рисунок 2.1 – Університет Мангейма у відкритій карті вулиць і відповідному представленні XML OSM

Рисунок 2.2 – Модель даних будівлі

Рисунок 2.3 – Розрахунок зони перетину (червоний полігон) трьох полігонів

Рисунок 2.4 - Оптимальне розміщення U-подібних сходів в кімнаті

Рисунок 2.5 – Розташування кімнат за допомогою алгоритму Best Fit

Рисунок 2.6 – Покрокове розширення кімнат

Рисунок 2.7 - Зернистість типу поверхні

Рисунок 2.8 - Поверхні набору текстур кімнати відображені на гранях сітки

Рисунок 2.9 – Алгоритм підключених компонентів та Dijkstra для побудови коридорів

Рисунок 3.1 – Створення підлоги, стін та стелі на сітці

Рисунок 3.2 – Інтерполяція між краями перил сходів

Рисунок 3.3 – Три типи перил: a) палі; b) суцільні; c) палі у висоту стіни

Рисунок 3.4 – Процес двовимірного створення вікна та віконної рами

Рисунок 3.5 – Ідентична будівля на трьох рівнях LOD

6. Дата видачі завдання: _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури		
2	Теоретико-інформаційний огляд методів процедурної генерації		
3	Процедурна генерація віртуальних будівель		

4	Процедурна генерація тривимірних сіток		
5	Вступ, висновки, реферат		
6	Попередній захист роботи		

Студент _____ Д. Цинковський

Керівник роботи _____ С. Резник

РЕФЕРАТ

Текстова частина бакалаврської роботи: 40 с., 19 рис., 2 дод., 19 джерел.

ПРОЦЕДУРНА ГЕНЕРАЦІЯ, МЕТОДИ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ, PCG, RPG, ТРИВИМІРНЕ МОДЕЛЮВАННЯ, ВІРТУАЛЬНІ ПРОСТОРИ

Об''єкт дослідження - методи та моделі процедурної генерації віртуальних просторів.

Предмет дослідження - процес процедурної генерації будівель за допомогою алгоритму Best Fit.

Мета роботи - дослідження методів та моделей процедурної генерації віртуальних просторів.

Методи дослідження - методи комп'ютерного моделювання, статистичні та емпіричні методи.

Визначено найуживаніші методи та моделі процедурної генерації віртуальних просторів.

Здійснено аналіз методу процедурної генерації Best Fit для побудови віртуальних будинків.

На основі результатів виконаних досліджень розроблено алгоритми тривимірної генерації віртуальних будинків.

Упровадження розробленої методики дозволяє рівень реалістичності графічного моделювання порівняно з існуючими способами генерації будівель.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

LOD – рівень деталізації

PCG – процедурна генерація

RPG – рольова гра

ААА – клас високобюджетних комп’ютерних ігор

ГІС – геоінформаційна система

КА – клітинний автомат

ШІ – штучний інтелект

ВСТУП

Актуальність теми. Процедурна генерація - це метод алгоритмічного створення даних за допомогою комбінацій алгоритмів, поєднаних із випадковістю. Створення віртуальних просторів використовується в багатьох сферах людської діяльності від науковою до розважальної. Особливо поширене використання процедурної генерації набуло популярності в розважальній сфері, а саме: ігровій, 3D моделюванні та у кіновиробництві.

Ріст популярності процедурної генерації головним чином зумовлений економічними чинниками, так як можна пришвидшити розробку майже будь-якого продукту комерційного, де потрібні унікальні великі світи, які можуть навіть мати також процедурну генеровані біоми, флору та фауну, або є необхідність генерувати велику кількість однотипних об’єктів з заданими умовами. Створення таких даних надає набагато швидший результат аніж ручна робота одного або навіть десятка спеціалістів, також у випадку ручної роботи в залежності від розміру віртуального світу виникає прямо пропорційна залежність від кількості людей залучених до створення таких просторів до часу витраченого на їх створення. Ще одною дуже привабливою стороною даного методу є суттєва економія ресурсів ПЗП, тому як всі об’єкти генеруються процедурно в реальному часі на ОЕМ, з теперішнім розвитком технологій ОЕМ дозволяє в реальному часі будувати великі віртуальні простори в розумних рамках очікування користувачем для отримуваного результату.

Віртуальні світи стають все більш важливими в контексті ігор і симуляцій. Подивившись, наприклад, на Skyrim від Bethesda, ми можемо побачити, що складність і зусилля, пов’язані зі створенням достовірного та реалістичного 3D-світу, є завданням кількох місяців для великої команди професійних дизайнерів. Але не тільки загальна візуальна якість відіграє важливу роль у процесі проектування; також величезна кількість різних об’єктів, необхідних для уникнення повторюваного вигляду, коли такі об’єкти, як меблі, рослини чи будівлі часто з’являються знову.

Однак якою б чудовою не здавалась процедурна генерація, свої мінуси ця технологія також має, головним недоліком є час на розробку алгоритмів. І тут вже виникає інша прямо пропорційна залежність - складність створення віртуального простору залежить від часу який необхідний на розробку алгоритму для процедурної генерації. Все це зумовлює актуальність дослідження методів та моделей процедурної генерації віртуальних просторів.

Об'єктом дослідження є методи та моделі процедурної генерації віртуальних просторів.

Предметом дослідження є процес процедурної генерації будівель за допомогою алгоритму Best Fit.

Метою бакалаврської роботи є дослідження методів та моделей процедурної генерації віртуальних просторів. Для досягнення поставленої мети, було виділено ряд **завдань:**

1. Здійснити теоретико-інформаційний огляд методів та моделей процедурної генерації;

2. Проаналізувати використання методів та моделей процедурної генерації для побудови віртуальних будівель;

3. Запропонувати методи та моделі генерації тривимірної сітки.

Методи дослідження. В роботі використовувались методи комп'ютерного моделювання, статистичні та емпіричні методи.

Наукова новизна роботи полягає в наступному:

1. Запропоновано вдосконалений спосіб процедурної генерації будівель, який на відміну від існуючих способів використовує ГІС як джерело інформації про будівлю, що дозволяє отримувати візуально кращі результати графічного моделювання.

2. Запропоновано процедуру додавання текстур, що дозволяє підвищити рівень реалістичності графічного моделювання порівняно з існуючими способами генерації будівель.

Практична значущість. Враховуючи сучасні безпекові виклики ззовні кордонів нашої держави та ведення активних бойових дій на території нашої

держави, які викликані прямою збройною агресією зі сторони рф за участю рб, а Україна як суверенна держава котра має виключне право на самозахист своєї територіальної цілісності від зовнішніх загроз, що відповідно створює велику необхідність у навченості особового складу ЗСУ та інших військових формувань по застосуванню сучасних типів озброєння та військової техніки, це може стосуватися не тільки персональних видів озброєння таких як NLAW, Stinger та Javelin, а і цілих військових комплексів, як до прикладу Bayraktar TB2, РК-360МЦ «Нептун» які вже вимагають злагодженої роботи цілого підрозділу.

1 ТЕОРЕТИКО-ІНФОРМАЦІЙНИЙ ОГЛЯД МЕТОДІВ ТА МОДЕЛЕЙ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ

1.1 Процедурна генерація: поняття, особливості та недоліки

Р. Смелік [16] визначає генерацію процедурного вмісту, як «будь-який тип автоматично згенерованого активу на основі обмеженого набору вхідних параметрів, визначених користувачем». Згаданий дослідник, у своїх роботах також посилаються на Т. Родена та І. Парберрі [17], які називають ці типи алгоритмів алгоритмами посилення, які приймають невеликий набір вхідних параметрів для перетворення їх на більший набір вихідних даних.

Д. Тогеліус [7], натомість, формулює визначення за допомогою антитези, зазначаючи, що процедурно згенерований контент не відповідає вмісту, згенерованому користувачами, навіть якщо вони використовують процедурні алгоритми, оскільки вони повинні бути параметризовані вручну.

М. Гендрікс та ін. [9] розглядають процедурну генерацію як альтернативу ручному проектуванню, але підкреслюють необхідність можливої параметризації, щоб дизайнери могли впливати на згенерований об'єкт.

Н. Шейкер та ін. [12] є більш конкретними в своїх міркуваннях та визначають PCG на прикладах того, що таке PCG (наприклад, програмний інструмент для створення випадкових підземель без участі користувача).

Тобто, можна сказати, що процедурна генерація – це автоматичне створення цифрових активів для ігор, симуляцій або фільмів на основі попередньо визначених алгоритмів і шаблонів, які вимагають мінімальної участі користувача.

PCG є предметом дослідження не тільки в області комп’ютерних наук. П. Прусінкевич і А. Ліндемаєр [14] наголошують на зростаючому інтересі до інших спільнот, який виникає внаслідок міждисциплінарності та впливає як на природничі науки, так і на біологію. Такий великий інтерес до інших напрямів дослідження є показником актуальності цієї теми. Гармонізація всіх цих дисциплін, таких як біологія, архітектура, урбаністичні студії, психологія тощо, а також пошук

правильних формалізмів і структур даних вимагає, однак, надзвичайних зусиль. Фінкенцеллер звужує зачеплені області комп'ютерних наук до:

- граматики;
- L-системи;
- граматика форми;
- мови програмування.

Він також зазначає, що програмування є найбільш гнучким, але схильним до помилок методом автоматичної генерації процедурного вмісту.

Оскільки створюваний контент повинен задовільнити певні критерії, а також вирішувати поставлене перед ним завдання, тому найчастіше виділяють наступні особливості [18]:

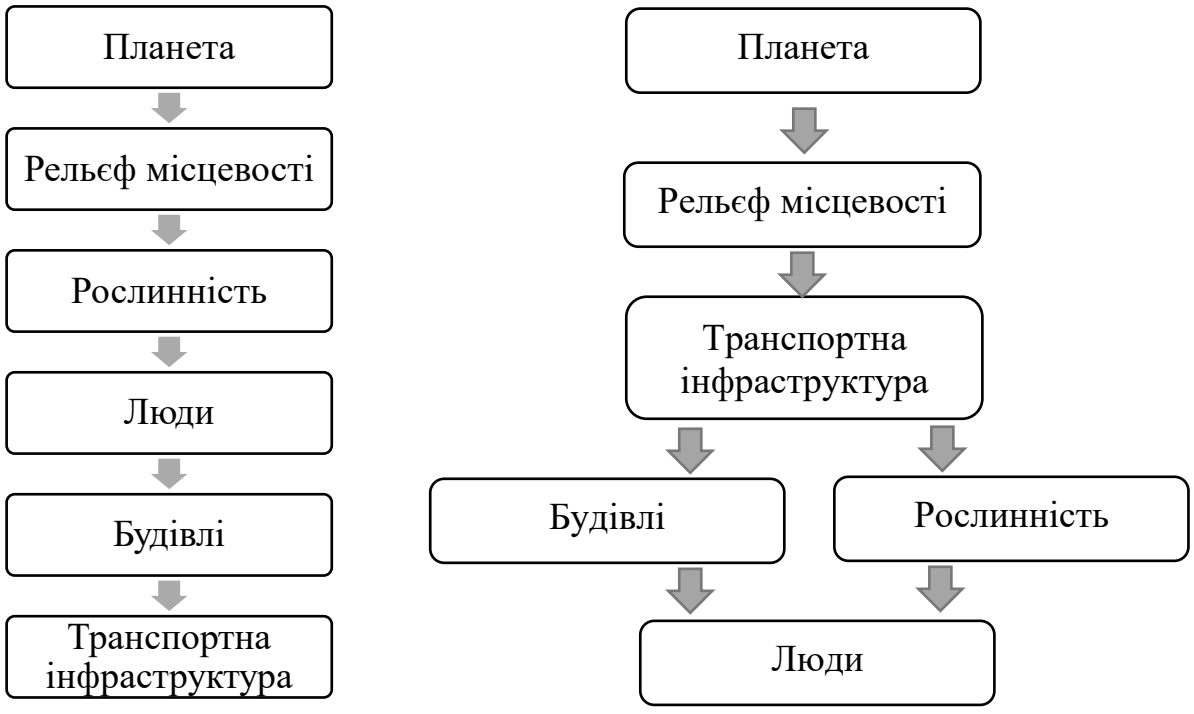
- швидкість: контент повинен генеруватися в задовільних часових рамках по мірі необхідності під час процесу розробки;
- надійність: генератор повинен задовольнити задані критерії та забезпечувати виконання поставлених умов для кінцевого результату;
- контрольованість: можливість контролювати процес генерації контенту в доступній можливості свободи;
- різноманітність: створювати такий контент, щоб не було відчуття одноманітності;
- правдоподібність: згенерований контент повинен виглядати так, ніби він створений людиною.

Комп'ютерні науки часто вимірюють ефективність і зрілість програмного забезпечення або алгоритмів, щоб гарантувати їх якість і застосовність. Широко використовуваний показник - це проста суб'єктивна оцінка того, наскільки створений контент виглядає реалістичним; це не так, якщо людина-спостерігач може легко ідентифікувати його як автоматично згенерований. Ми пропонуємо балансувати між продуктивністю та точністю PCG: якщо алгоритм повертає точний результат (напр. реалістичний ліс), алгоритм потребує більшої обчислювальної потужності, більше пам'яті або більше місткості для створення більшої кількості варіацій дерев, текстур із вищою роздільною здатністю, більш детальної сітки або

більш густого насадження. Залежно від бажаного результату користувач повинен вибрати між продуктивністю і реалістичністю, щоб досягти оптимуму для даної системи та вимог віртуального простору.

Часто згенеровані об’єкти можна класифікувати як ручну роботу або створені природньо. Головне питання полягає в тому, чи можна визначити, чи можна створити ту чи іншу категорію з меншими зусиллями, ніж іншу. Як приклад, складність завдань по створенню 3D дерев можна порівняти зі складністю завдань по створенню 3D будівель. Якщо велика частина видимого контенту створюється автоматично, порядок його створення стає актуальним.

Для прикладу, порівняємо дуже спрощену проекцію природного утворення Землі з процедурним створенням віртуального світу (рис. 1.1).



(а) (б)

Рисунок 1.1 – Схеми побудови віртуального простору в спрощеному природному порядку (а) та в оптимізованому процедурному (б)

У концепції багатьох розробників PCG природного створення віртуального світу ландшафт (включно з водою) на планеті служить основою світу, а потім рослинність, така як дерева та рослини. Пізніше люди будують будівлі на цьому ландшафті та з’єднують їх мережею доріг. Потім поселення з роками то росте, то

гине. Коли рослинність або гори заважають, людство намагається прибрати їх, щоб побудувати дороги чи будівлі.

Зіставлення PCG віртуального світу зі спрощеним природним створенням землі призводить до деяких додаткових ітерацій, де ліси вирубуються для будівництва доріг, місцевість вирівнюється, щоб розмістити на ній міста, або річки перенаправляються, щоб виростити місто в бажаний напрямок. Як альтернативу пропонується використовувати порядок, показаний на малюнку 2.1 (б), для створення рельєфу, за яким слідує транспортна інфраструктура, а потім будівлі та рослинність. Ця процедура, швидше за все, не буде відповідати ідеї природного зростання, але вона може сприяти створенню віртуальних світів за допомогою комп'ютера.

Ознайомившись із засобами процедурної генерації можна виділити основні недоліки при використанні даної системи. Досить ймовірно що ці проблеми можна оминути при правильному плануванні розробки, наявності альтернативних алгоритмів. Розглянемо основні недоліки які можуть виникнути:

1. Надмірна складність:

1.1. Неможливість забезпечити необхідний контроль якості;

1.2. Виникає необхідність покриття великою кількістю тестів.

2. Великі витрати часу:

2.1. Алгоритми процедурної генерації можуть бути дуже складними при розробці;

2.2. Процес процедурної генерації може негативно впливати на продуктивність виконання гри;

2.3. Досить складно керувати результатом, потрібно постійно вносити зміни щоб задовольнити потреби.

3. Випадковість. З цією помилкою зазвичай зіштовхуються початківці. Необхідно мати досвід та навички, щоб процедурно згенерований контент вписався в ігрову механіку. Звичайна випадковість створює відчуття повторюваності, що призводить до не зацікавлення гравця.

1.2 Теоретичний аналіз методів процедурної генерації

1.2.1 Алгоритм Форчуна та мозаїка Вороного

Мозаїка Вороного - це спосіб поділу простору на набір областей (які називають комірками) із заданим набором вхідних точок (які називають ділянками), таким чином, що кожна комірка містить рівно 1 ділянку, а точки всередині комірки це саме ті, чиє найближче місце знаходиться всередині цієї комірки [19].

Мозаїки Вороного мають надзвичайно широкий спектр застосування. Їх можна використовувати для створення текстур води, генерування геометрії кам’янистого ґрунту, ефективного обчислення найближчого сусіда точки або для керування рухом агентів ШІ.

Якщо кількість ділянок є відносно не великою, то можна просто перерахувати кожну точку у заданому просторі та обчислити, яка з них найближча. Це працює, якщо необхідним є визначення того, до якої ділянки належить кожна точка простору, і є кінцева кількість дискретних точок, наприклад, якщо просто потрібно визначити колір для кожного пікселя на зображенні.

З іншого боку, якщо кількість вхідних точок дуже велика або важливі межі між клітинками, тоді потрібен інший підхід. Існує кілька способів обчислення діаграми Вороного для набору вхідних точок, але одним з найзручніших, на нашу думку є алгоритм Форчуна.

Алгоритм Форчуна використовує підхід «розгортки». В ньому послідовно розглядається кожна ділянка та «вирощуються» комірки навколо кожної ділянки під час «розгортання». Як тільки клітина була повністю оточена іншими клітинами, вона, очевидно, не зможе рости далі. Це означає, що необхідно відстежувати лише ті клітини поблизу лінії розгортки, які все ще ростуть. Цей набір зростаючих клітин прийнято називати «береговою лінією» (рис. 1.2).

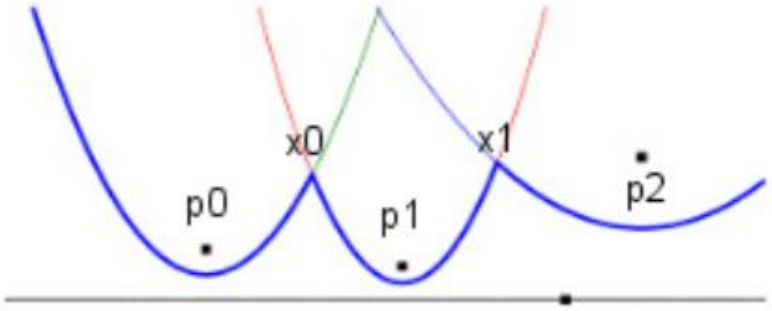


Рисунок 1.2 – Приклад «берегової» лінії

Таким чином, алгоритм виглядає так: якщо існує певна лінія *l*, яка рухається зверху вниз і вона проходить через кожну задану точку, при цьому дані точки вже належать мозаїці Вороного, то маючи точку *x*, що знаходиться на однаковій відстані від точок *a* та *b* (тобто всі вони лежать під лінією розгортки) точка що розглядається не обов'язково має бути на діаграмі, адже над лінією розгортки може знаходитись наступна ділянка, котра ближче. А оскільки визначення діаграми Вороного можливе лише для точок, розташованих ближче до будь-яких інших під *l*, то всі ці точки лежать під параболою, а точка *a* – це фокусна точка, лінія розгортки в такому випадку – директриса згідно з визначенням параболи (рис. 1.3).

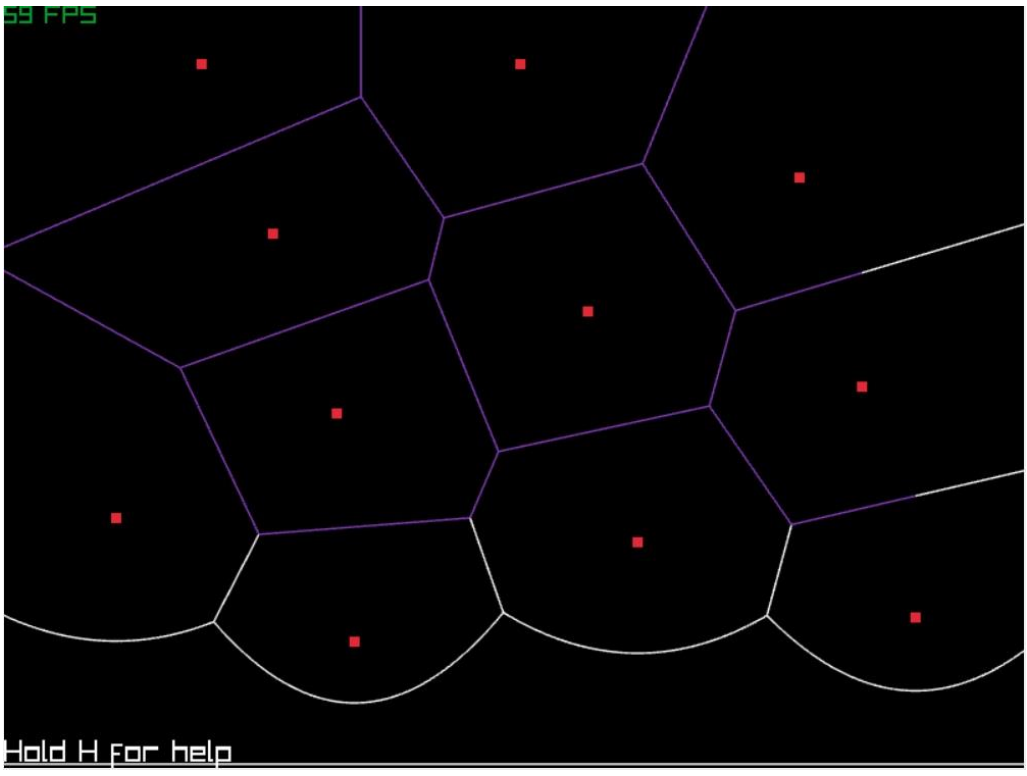


Рисунок 1.3 – Приклад результату алгоритму Форчуна

Деривацію формули для побудови параболи згідно з алгоритмом Форчуна наведено в додатку А.

1.2.2 Метод двійкового розбиття простору

Двійкове розбиття простору (BSP – binary space partition) – це метод рекурсивного поділу простору на два опуклих набори. Вперше був сформульований в 1969 році в контексті тривимірної комп'ютерної графіки [2].

Розглянемо послідовність виконання двійкового розбиття простору. Спочатку до першого вузла(кореня) дерева привласнюється прямокутний рівень заповнений стінами, чи просто пустим простором. Надалі область рівня вузла поділяється на дві підобласті котрі діляться на два дочірніх вузла. Напрямок ділення, а саме вертикальне чи горизонтальне, обирається випадковим чином, після чого надається перевага відповідній координаті ділення. В результаті маємо дві області меншого розміру. Після чого ця послідовність розподілу, як було вказано, рекурсивно повторюється, включаючи отримані підобласті. Процес триває й надалі, поки не задовольнить обрані нами вимоги. Для прикладу, поки розмір кожної розглянутої області не стане занадто маленьким хоча б по одній з осей. Розглянемо детальніше: задається, що ділянка може ділитися тільки тоді, коли і її ширина, і її висота не менше 20 клітин, і при цьому координата ділення не може стояти ближче, ніж на 10 клітин до області кордону. Це гарантує, що всі кінцеві ділянки будуть мати розмір не менше ніж 6 клітин по горизонталі та 6 клітин по вертикалі [2].

Також можливо вказати додаткову умову виходу залежну від кількості ділянок (скажімо, завершити процес розбиття, якщо областей стало більше 10). Якщо початкові розміри рівня значно більше, ніж мінімальна площа однієї ділянки, помножена на максимальну кількість цих ділянок, то вірогідно, що саме обмеження на кількість кімнат буде слушною та остаточною вимогою для завершення алгоритму, а не досягнення мінімальних розмірів кожної з областей. В такому

випадку більш доречно використовувати саме ітеративний варіант алгоритму, а не рекурсивний, і вже після кожного ділення ділянки сортувати поточний список областей зі зменшенням їх площі. Тоді розміри кімнат будуть розподілені відносно рівномірно між собою, що являє собою бажаний результат.

Цей процес поділу областей на підобласті породжує та вибудовує деревоподібну структуру даних, завдяки якому алгоритм і носить свою назву. У кожній вершині(вузлі) дерева зберігається інформація про дані розподілу об'єкта, як приклад, у вузлі зберігаються координати лівого верхнього кута та ширина і висота діленої ділянки, а також, в будь-якому випадку, посилання на двох нащадків - якщо підобласті є, або ж порожні посилання - якщо даний вузол кінцевий. Перша вершина(корінь) дерева зберігає відомості про ділянку всього рівня чи об'єкта цілком.

1.2.3 Метод «Drunkard walk»

Розглянуті алгоритми вирішують проблему пошуку оптимального шляху, однак у сфері обчислення комп’ютерної графіки є ще багато інших завдань, котрі необхідно розглянути. Наприклад процедурна генерація мапи. Існує багато методів генерації мапи, однак жоден з них неможливо вважати універсальним рішенням, тому більшість розробників підлаштовують обрані ними алгоритми під свої потреби.

Одним із таких методів є алгоритм випадкове блукання або ще «Drunkard walk». Випадкове блукання відноситься до так званого математичного формалізму. Цей алгоритм має високу ступінь випадковості, що допомагає створювати дуже різноманітні відкриті та закриті простори [15].

Його основне призначення - опис траєкторії, що утворюється в результаті виконання послідовних випадкових кроків. Час від часу виникають цікаві випадкові блукання різного роду. Найчастіше розглядаються випадкові блукання, котрі мають в основі ланцюг Маркова, проте становлять інтерес також і інші складніші типи блукань.

Ланцюгом Маркова прийнято вважати це випадковий процес, що задовольняє марківським властивостям і приймає кінцеве чи лічильне число значень (станів). Ланцюг може існувати як із дискретним, і з безперервним часом. Найпоширенішим способом візуалізації ланцюгів Маркова є графи переходів. Станами ланцюга називають вершини графа, і напрямне ребро проходить із вершини і у вершину j тільки у тому випадку, якщо ймовірність переходу між відповідними станами не дорівнює нулю. Ця можливість переходу також відображається поряд з відповідним рубом.

1.2.4 Синтез текстур методом k-найближчих сусідів та клітинних автоматів

Текстура – це повсякденний візуальний досвід. Вона може описати широкі та різноманітні характеристики поверхні, такі як рельєф, рослини, мінерали, хутро і шкіра. Оскільки відтворення візуального реалізму фізичного світу є основною метою комп'ютерної графіки, текстури зазвичай використовуються при відтворенні синтетичних зображень. Ці текстури можна отримати з різних джерел, наприклад, намальовані від руки або відскановані фотографії. Намальовані від руки малюнки можуть бути естетичними, але їх важко зробити фотореалістичними. Більшість відсканованих зображення, однак, мають недостатній розмір і можуть призвести до видимих швів або повторення, якщо вони безпосередньо використовуються для відображення текстури.

Синтез текстур – це альтернативний спосіб створення текстур. Тому що синтетичні текстури можна зробити будь-якого розміру, уникнути візуального повторення. Синтез текстур також може створювати розкладні зображення та правильно обробляти граничні умови. Потенційні застосування синтезу текстур також широкі, наприклад: зменшення шуму зображення, заповнення оклюзії та стиснення. Мету синтезу текстур можна сформулювати так: дано зразок текстури, синтезувати нову текстуру, яка, для спостерігача – людини буде схожа на початкову текстуру [17].

Основними проблемами є 1) моделювання – як оцінити стохастичний процес із заданого кінцевого зразка текстури; 2) відбір проб – як розробити ефективну процедуру відбору проб та створювати нові текстури з заданої моделі. І моделювання, і відбір необхідні для успіху синтезу текстури. Візуальна точність створених текстур залежатиме насамперед через точність моделювання, при цьому ефективність вибірки процедури безпосередньо визначить обчислювальну вартість згенерованої текстури.

Основними перевагами цього алгоритму є якість і швидкість: якість синтезованих текстур або краща за ту, яка генерується попередніми методами, а швидкість обчислень на два порядки швидше. Це дозволяє використати алгоритм в області, де традиційно генерація текстури була занадто дорогою.

Клітинний автомат (КА) — дискретна математична модель, яка визначає сукупність та описується набором клітинок, що утворюють періодичну решітку, та заданими правилами переходу, що визначають стан клітини за теперішнім станом самої клітинки та тих її сусідів, що знаходяться від неї на певній відстані, яка не перевищує максимальну [17].

Основний напрям дослідження клітинних автоматів — алгоритмічна розв'язність окремих задач. Також розглядаються питання побудови початкових станів, при яких клітинний автомат вирішуватиме задану задачу (рис. 1.4).

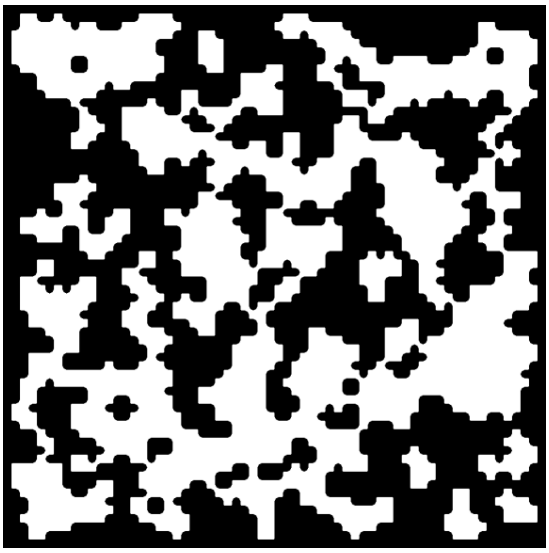


Рисунок 1.4 – Згенерована текстура методом клітинних автоматів

Поняття клітинних автоматів доволі обширне, тому можна знайти доволі багато різних визначень. Найпоширенішими є:

- математичний об'єкт із дискретним простором та часом;

- регулярна структура двійкових скінченних автоматів з однаковими правилами переходів, що виражені у вигляді булевих функцій від станів сусідніх автоматів;

- стилізовані, синтетичні світи, що визначені простими правилами, подібно правилам настільної гри;

- математична ідеалізація фізичної системи, в якій час та простір дискретні, а фізичні величини приймають скінченну множину значень;

- візуальна модель динамічної системи з дискретним часом та простором.

У кожний момент часу кожен елемент КА приймає один стан зі скінченного набору станів. У залежності від цих станів в наступний момент часу набір елементів може прийняти новий стан. Якщо для елементів КА множини можливих станів відрізняються, такий клітинний автомат називається полігенним. Але на практиці використовуються комірки з еквівалентною множинам можливих станів алгебраїчною структурою - лінійні КА.

Якщо з будь-якого початкового стану можна привести клітинного автомат в будь-яку задану конфігурацію шляхом варіювання значення загального вхідного параметра, такий КА називають повним.

Елементи можуть бути геометрично розташовані різноманітним чином. Розмірність простору може бути довільною, а число елементів - як безкінечним, так і скінченним. В останньому випадку виникає додаткова міра свободи в граничних умовах. Вони можуть бути різними, але на практиці використовуються постійні у часі або періодичні граничних умовах. У динамічних КА геометрія може змінюватися з часом, а якщо геометрія різна на різних ділянках простору, такі клітинні називають неоднорідними (рис. 1.5).

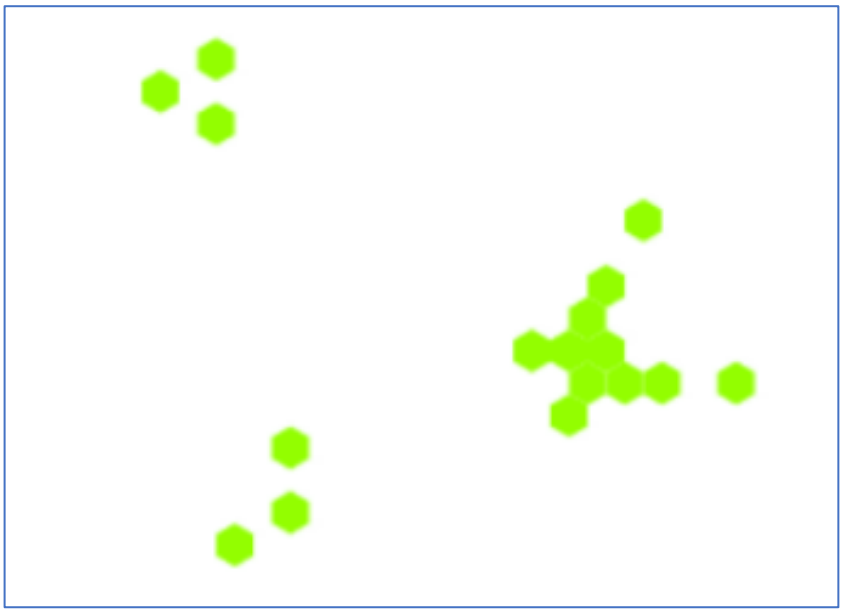


Рисунок 1.5 – Клітинний автомат із комірками, що зображені у вигляді шестикутника

Сусіди - це елементи, від яких залежить елемент КА. Можна назвати поняття сусідства ключовим для КА. При тому сусідство розуміється не в геометричному сенсі, а в інформаційному. Хоча зазвичай інформаційний сенс накладається на геометричний. Сусідство одиничних автоматів встановлюється постійним для кожного одиничного автомата решітки і визначається спеціальним вектором — індексом сусідства. Як правило, розглядаються d-мірні регулярні решітки, в цілочислові точки яких поміщені копії деякого автомата Мура. Стан елемента в наступний момент часу обчислюється зі стану самого елементу і його сусідів. Сусідство більшою мірою визначається геометрією КА. Для різних цілей можлива зміна числа вхідних станів елемента. Якщо для кожного елемента КА число входів і виходів однакове, такий КА називається збалансованим.

2 ВИКОРИСТАННЯ МЕТОДІВ ТА МОДЕЛЕЙ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ДЛЯ ПОБУДОВИ ВІРТУАЛЬНИХ БУДІВЕЛЬ

2.1 Модель даних і топологія

Дія багатьох сучасних ігор AAA відбувається у відкритому світі, який гравець може вільно досліджувати лише з деякими обмеженнями. У таких серіях ігор, як Grand Theft Auto, The Witcher, Fallout, Assassin's Creed або Watch Dogs, одне чи кілька величезних міст є центром сюжету. Свобода, пов'язана з іграми з відкритим світом, сприймається інтенсивніше, коли гравець стикається з меншою кількістю бар'єрів, які заважають йому досліджувати оточення.

Одним із найпоширеніших типів бар'єрів у міському середовищі є будівлі, куди гравець не може потрапити і які представлені лише фасадами. Обмежувальним фактором, щоб не моделювати інтер'єр кожної будівлі, часто є час і гроші. Назвемо деякі цифри: Ubisoft наймає 400-600 людей для розробки гри з відкритим світом AAA, тоді як розробка Grand Theft Auto IV залучила понад 1000 людей і зайняла три роки [11].

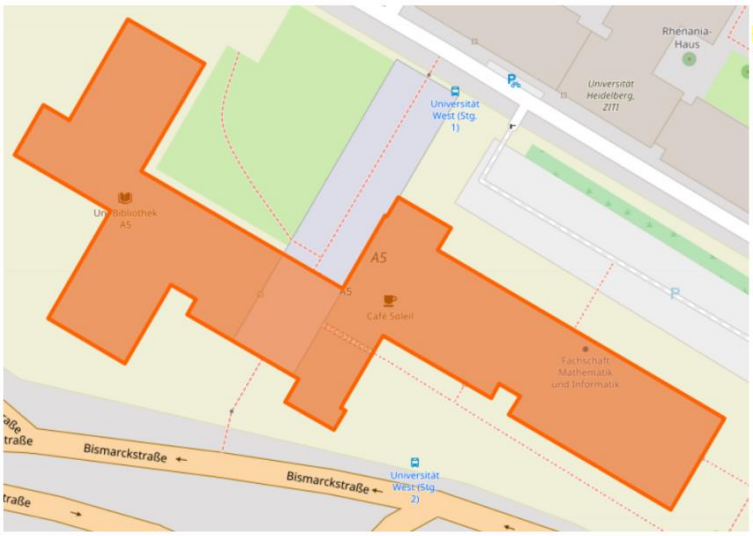
Оскільки багато спільнот не задоволені відсутністю доступності, необхідність більш доступного інтер'єру обговорюється суперечливо, а деякі фанати навіть створюють модифікації, щоб відкривати замкнені двері, створювати відсутні інтер'єри або навіть додавати цілі нові райони до існуючих ігрових середовищ.

Отже, виникає питання, чого могли б досягти розробники ігор, якби будівництво будівель з достовірним інтер'єром було простіше, швидше і дешевше. Двома можливими ефектами можуть бути більше занурення завдяки більшій реалістичності та меншій кількості бар'єрів, або збільшення відтворюваності, оскільки нові області можуть генеруватися знову і знову – навіть під час гри.

Отже, розглянемо процес вдосконалення ігрового досвіду завдяки усуванню бар'єрів доступності до будівель. Процедурний компонент цієї роботи виражається

алгоритмом, який створює будівлю із заданого плану, який може бути багатокутником з отворами або без них.

Основна перевага можливості створювати будівлі для заданого багатокутника полягає в тому, що алгоритм можна застосувати до даних ГІС, що є поширеним способом зберігання планів поверхів будівлі (рис. 2.1). Це дозволяє розробникам ігор миттєво створювати віртуальні карти з реальних географічних місць, що останнім часом стає все більш популярним в іграх.




```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" copyright="OpenStreetMap and contributors"
   attribution="http://www.openstreetmap.org/copyright"
   license="http://opendatacommons.org/licenses/odbl/1-0/">
 <way id="35154744" visible="true" version="14"
   changeset="27641696" timestamp="2014-12-23T00:29:06Z"
   user="whb" uid="202726">
  <nd ref="412350337"/>
  <nd ref="412350338"/>
  <!-- ... -->
  <nd ref="3241380146"/>
  <nd ref="412350349"/>
  <nd ref="412350350"/>
  <nd ref="1019984303"/>
  <nd ref="1019984284"/>
  <nd ref="412350337"/>
  <tag k="addr:city" v="Mannheim"/>
  <tag k="addr:country" v="DE"/>
  <tag k="addr:housenumber" v="6"/>
  <tag k="addr:place" v="A5"/>
  <tag k="addr:postcode" v="68159"/>
  <tag k="addr:street" v="A5"/>
  <tag k="building" v="university"/>
  <tag k="name" v="A5"/>
  <tag k="source" v="HiRes aerial imagery"/>
  <tag k="wheelchair" v="limited"/>
 </way>
</osm>
```

Рисунок 2.1 – Університет Мангейма у відкритій карті вулиць і відповідному представленні XML OSM

Тепер визначимо модель даних для наших алгоритмів (рис. 4.2). Будівля містить кілька кімнат, кілька сходів і принаймні один дах. Текстури визначаються набором текстур кімнати. Щоб сформувати кімнату, потрібні не менше трьох стін. Кімната може містити кілька сходових зон і зон для людей. Обидва визначають простір, який блокує розміщення інших об’єктів. Стіна кімнати має від нуля до n дверей і вікон.

Модель даних на рисунку 4.2 не представляє вичерпну геопросторову топологію; наприклад, він не містить інформації про те, чи є дві кімнати суміжними одна з одною, чи вони перекривають одна одну двома накладеними поверхами. Ця топологія виводиться неявно під час процесу генерації для перевірки просторових відносин, коли це необхідно.

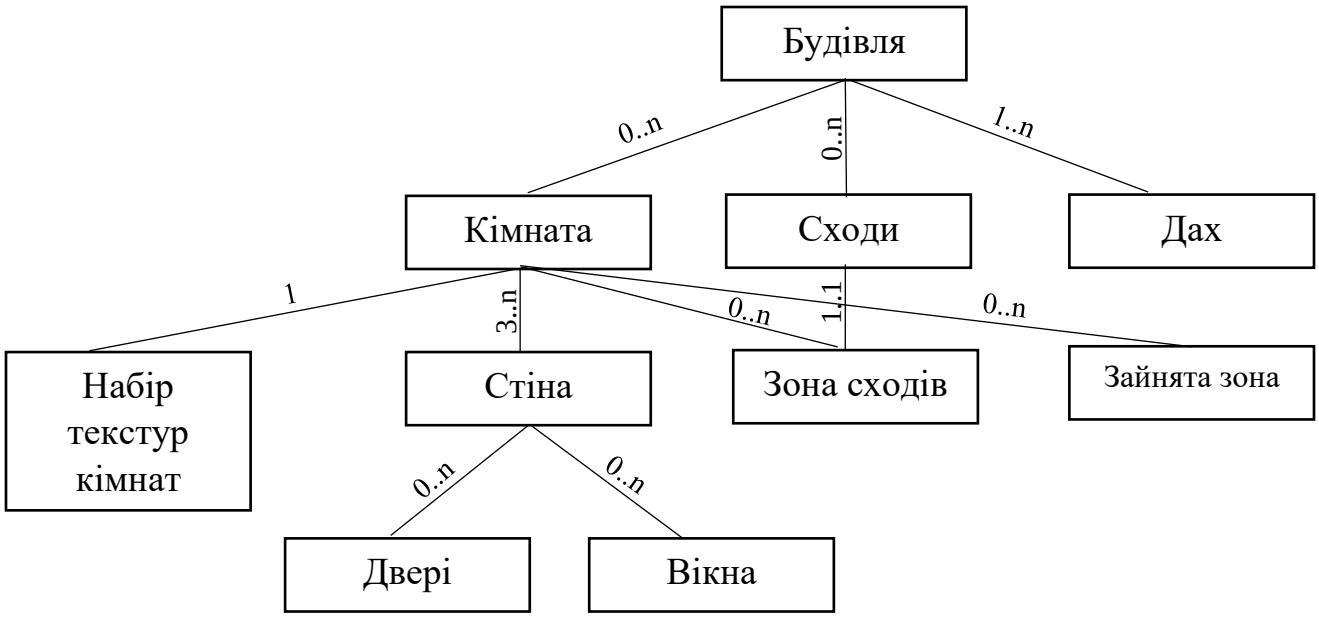


Рисунок 2.2 – Модель даних будівлі

Рішення не відстежувати топологію будівлі в моделі даних базується на простоті визначення залежностей від моделі даних (наприклад, якщо дві кімнати суміжні або якщо вони з’єднані дверима). Це зменшує складність і схильність до помилок моніторингу дійсності, стану та обмежень топології під час кожної зміни в архітектурі будівлі.

2.2 Формування процедурної будівлі

Процес створення будівлі можна розділити на створення екземпляра об’єктної моделі та створення 3d-моделі. Намір поділу обох полягає в тому, щоб виключити всі випадкові операції з генерації 3d-моделі, а отже, гарантувати відтворюваність для кожної процедурно згенерованої будівлі. Крім того, процедурну частину можна замінити на іншу, а генератор сітки все ще можна використовувати без будь-яких змін.

Створення екземпляра об’єктної моделі: модель даних розраховується з використанням усіх розмірів будівлі, макетів і обмежень. Користувач може впливати на модель за допомогою параметризації. Оскільки детермінований екземпляр містить випадкові фактори, він не є відтворюваним, і може бути

успішним після того, як він був невдалим раніше з ідентичними параметрами, визначеними користувачем

Генерація 3d-моделі: модель об’єкта перетворюється на 3d-модель за припущенням, що вона перевірена. Цей процес повністю автоматичний, і на нього можна вплинути, лише змінивши базову об’єктну модель.

2.3 Екземпляр об’єктної моделі

2.3.1 Вибір форми будівлі

Коли йдеться про житлові будинки, існує три найпоширеніші форми будівлі: прямокутна, Т-подібна та Г-подібна. Використання параметризованої ширини та довжини дозволяє нам створити варіацію цих трьох форм для невеликої різноманітності. Тим не менш, не передбачаючи оцінки, ми виявили, що навіть незважаючи на те, що така параметризація основних форм була можливою, частково випадкова форма будівлі призведе до більш цікавої, привабливої та доступної для дослідження будівлі. На додаток до цих найпоширеніших форм плану поверху, представлений тут метод підтримує будь-які можливі закриті форми.

2.3.2 Вибір і розміщення сходової клітки

З огляду на те, що будівля багатоповерхова, вибирається сходовий тип. Для нашого прототипу реалізовано три типи сходів: L-подібні, U-подібні та прямі сходи.

Розташування сходових кліток у багатоповерховому будинку є складним процесом. Він передбачає оптимальне розміщення одночасно на всіх поверхах, а також створення індивідуальної сітки сходової клітки. Сортування поверхів за висотою допомагає підтвердити, що вони справді піднімаються, і що між ними

немає пропущеного рівня. Перетин багатокутника верхнього поверху з багатокутником нижче визначає спільні зони, які піддаються сумніву як можлива зона розміщення сходів. Перетин послідовно обчислюється від верхнього поверху до найнижчого поверху, поки не залишиться n ділянок, які існують на кожному поверсі (рис. 2.3).

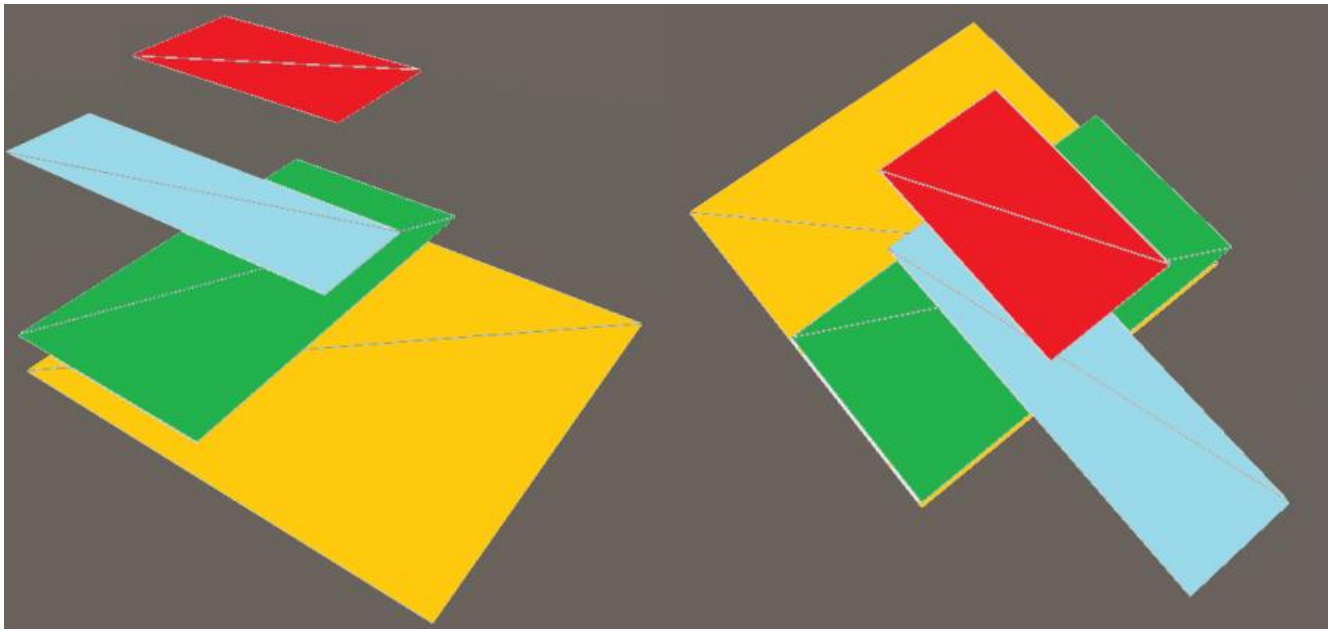


Рисунок 2.3 – Розрахунок зони перетину (червоний полігон) трьох полігонів

На наступному етапі розраховується необхідна площа для сходової клітки. Її розмір залежить від типу сходів, висоти приміщення і кількості ступенів. Щоб уникнути блокуючої стіни на вході та виході зі сходів, додається верхній і нижній майданчики.

Розміщення багатокутника сходової клітки в полігоні поверху досягається накладанням контуру сходової клітки та поверху. Багатокутник сходової клітки повертається, доки весь багатокутник не опиняється в межах багатокутника поверхів (або на його межах, якщо бути точним). Для вимірювання якості знайденого положення підсумовуються довжини країв сходової клітки та багатокутника кімнати. Вдале розміщення забезпечує високу вірогідність накладання. Цей метод називається Best Fit (додаток Б).

На рисунку 2.4 наведено приклад розміщення прямокутних сходів у багатокутному приміщенні. Алгоритм оцінює багато інших поворотів і позицій цих сходових кліток, але для простоти ми включаємо лише сім показаних тут.

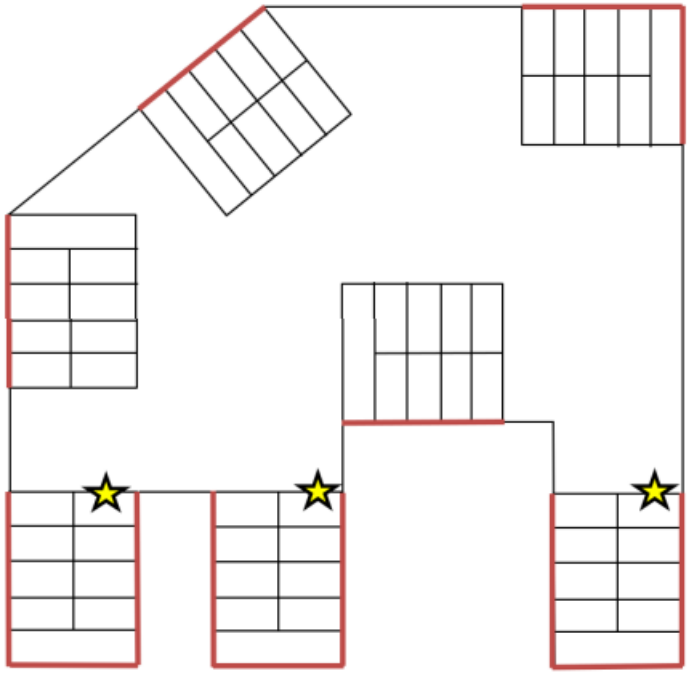


Рисунок 2.4 - Оптимальне розміщення U-подібних сходів в кімнаті

Три позиції сходів у нижній частині кімнати будуть оцінені високо, оскільки значення накладання як кімнати, так і країв сходової клітки є найвищим (див. червоні лінії).

Після розрахунку форми сходів, країв поручнів, сегментів входу та виходу та оптимального положення сходи реєструються для кожної накладеної кімнати.

2.3.3 Початкове розташування кімнат та їх розширення

Після розміщення сходових кліток ми використовуємо ковшовий підхід, щоб розподілити всі кімнати по всіх поверхах, щоб отримати рівномірну кількість кімнат на поверх. Наш алгоритм заснований на зростанні регіону. Створення зразкової будівлі з трьома поверхами та чотирнадцятьма кімнатами призводить до розподілу відповідно по 5, 5 та 4 кімнат. Ці кімнати масштабуються до невеликого

початкового розміру 1x1 одиниць і рівномірно розподіляються на кожному поверсі за допомогою програми Best Fit. Тут ми стикаємося з двома проблемами відповідного алгоритму:

1) Як можна помістити n багатокутників форми A в інший багатокутник B, якщо B має менше вершин, ніж n?

2) Як ми можемо переконатися, що не всі n багатокутників форми A розміщені в одній щільній області багатокутника B?

До першої проблеми звертаються шляхом розбиття країв зовнішнього багатокутника на кілька підребер, щоб завжди було більше або стільки ж вершин, скільки кімнат. У прикладі на рисунку 2.5 (a) у нас є вісім кімнат, які потрібно розмістити в багатокутнику з чотирма вершинами. Отже, ми поділяємо багатокутник, додаючи одне додаткове ребро до наявного ребра, так що загальна кількість вершин для цього багатокутника дорівнює восьми (див. червоні точки на малюнку 2.5 (b)). Підрозділ - це проста поетапна інтерполяція між початковою та кінцевою точками краю.

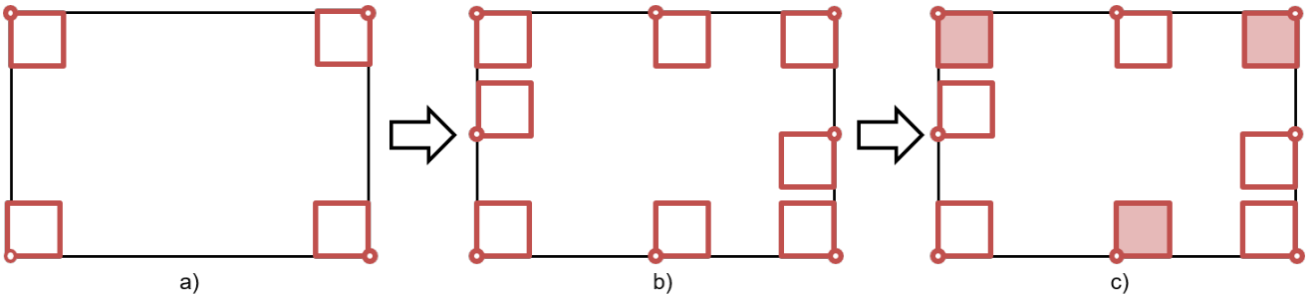


Рисунок 2.5 – Розташування кімнат за допомогою алгоритму Best Fit

Друга проблема розглядається в одному єдиному рішенні, яке має назву Best Fit by Highest Distance. Як відомо з основного методу Best Fit, усі можливі розміщення кімнат зберігаються у списку. Це дозволяє побачити всі можливі зони розміщення кімнат у формі будівлі. На наступному кроці ми беремо середину кімнати та визначаємо n позицій кімнати з усіх можливих позицій з найбільшою відстанню одна до одної.

Пошук найбільшої відстані розглядається в дослідженнях як проблема «найдальших сусідів». Через його складність було обрано алгоритм достатньої апроксимації від Le Bourdais, який є швидким і добре працює.

Коли кожна кімната займає початкову позицію плану поверху, ми розгортаємо їх за циклічним підходом уздовж послідовності A→B→C→A→... Алгоритм зростання розширює вершини багатокутника в кожному напрямку, поки вони не перетнуться з зовнішньою межею (формою будівлі) або з іншим багатокутником (іншою кімнатою). Коли вершина потрапляє на зовнішню межу, багатокутник все ще може бути розширений у всіх інших напрямках. На відміну від цього, перетин з іншим багатокутником, який одночасно розгортається в межах тієї самої фігури, припиняє роботу алгоритму, щоб уникнути поганих пропорцій і примусово отримати більше неправильних форм шляхом об'єднання решти артефактів (рис. 2.6).

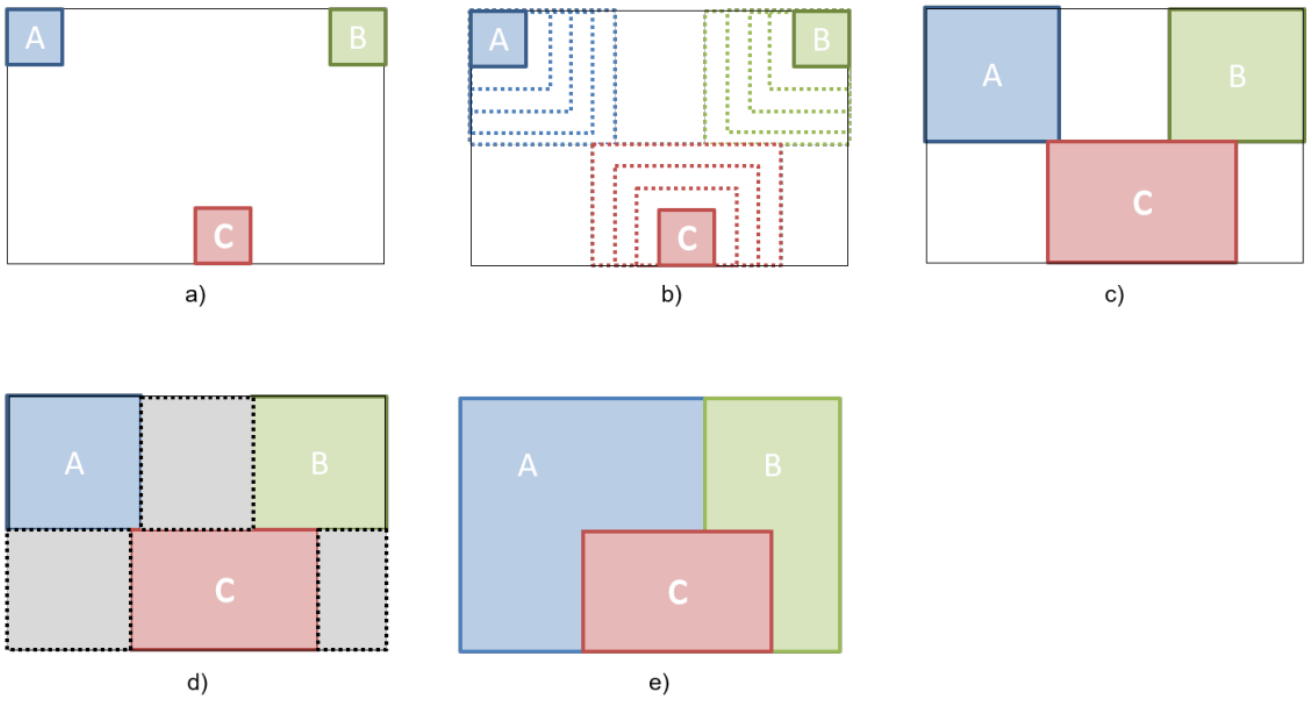


Рисунок 2.6 – Покрокове розширення кімнат

Якщо жодну з кімнат більше не можна розширити, алгоритм зростання досяг свого кінцевого стану. Оптимально, щоб ширина та висота будівлі були кратними розміру кроку, щоб алгоритм міг використовувати весь простір у зовнішньому полігоні. В іншому випадку у зовнішньому багатокутнику можуть бути невеликі

невикористані ділянки, які призведуть до небажаних артефактів у створеній сітці. Після розширення невикористані ділянки залишаються на поверсі, і їх можна побачити у вигляді сірих прямокутників на рисунку 2.6 (d). Ці області можна виділити, віднявши всі форми кімнат із плану поверху. Потім отримані форми об’єднуються з першою сусідньою кімнатою в списку кімнат на сюжеті. Цей підхід подібний до того, що раніше був використаний [8], він використовує сітку та розширює кімнати клітинка за клітинкою. Використовуючи вершини як відправні точки для зростання кожної кімнати, ми урізноманітнюємо процедуру і результат. Крім того, оскільки наш підхід базується на багатокутниках і логічних операціях, ми можемо працювати з непрямокутними формами будівель.

2.3.4 Текстуризація

Щоб застосувати текстури до поверхонь кімнати, ми вводимо набір текстур кімнати (рис. 2.7). Ця структура даних містить вісім різних текстур, по одній для кожної поверхні кімнати, як показано на рисунку 2.8.

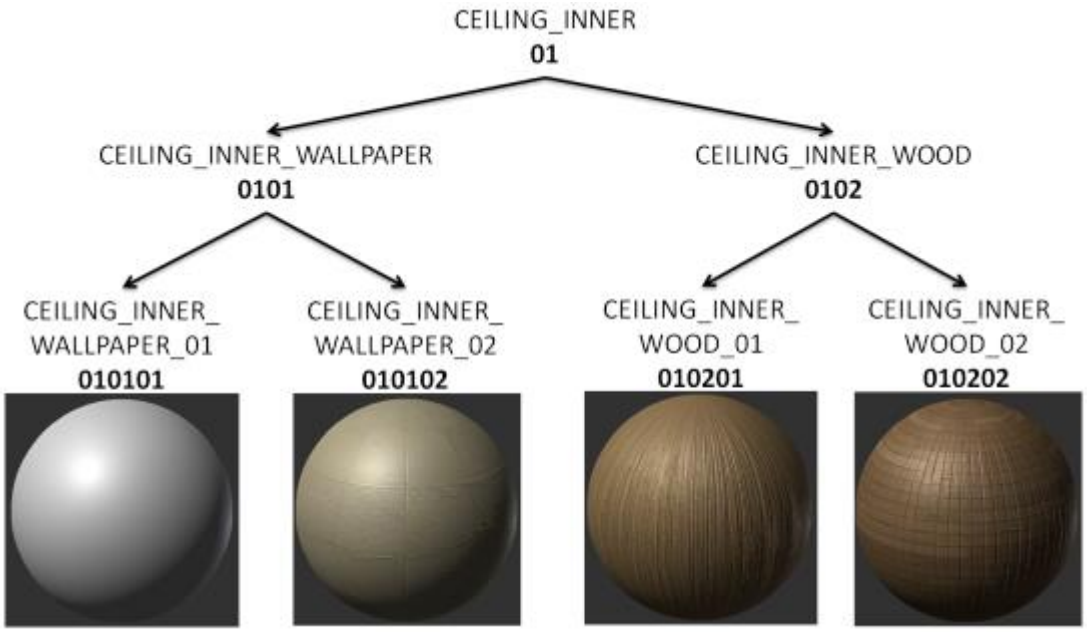


Рисунок 2.7 - Зернистість типу поверхні

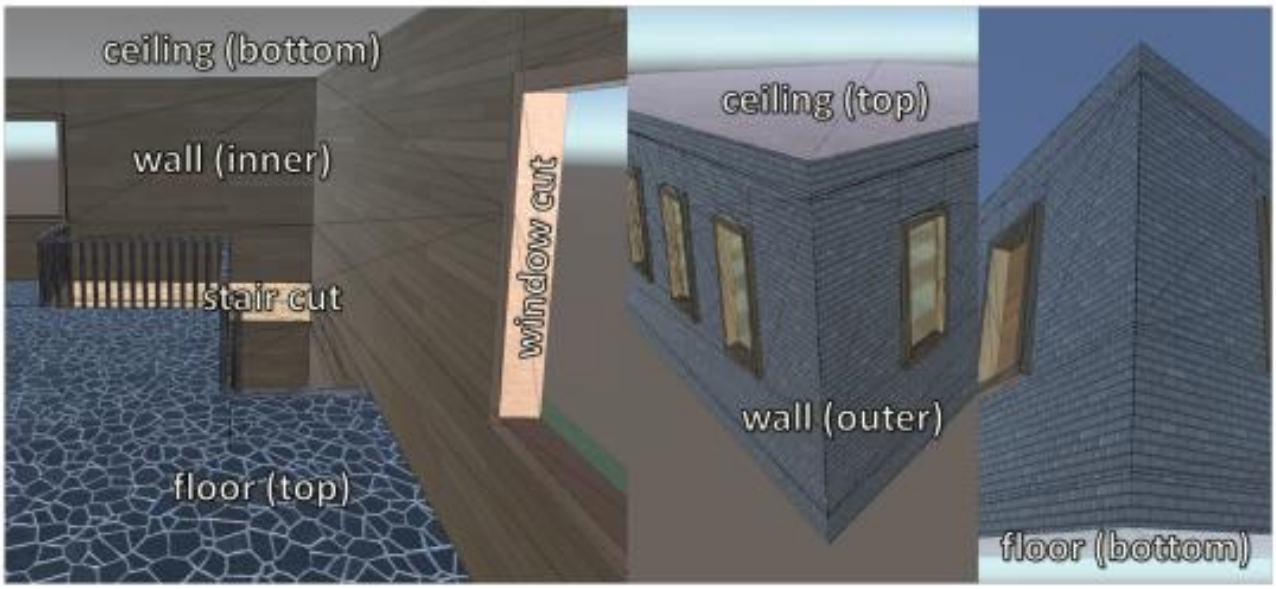


Рисунок 2.8 - Поверхні набору текстур кімнати відображені на гранях сітки

Область розрізу сходів і вікна посилаються на текстуру для тих областей, які створюються, коли ріжучий блок віднімає частини сітки кімнати для дверей, вікна чи сходів. Кожну текстуру в наборі текстур кімнати можна чітко визначити, але для досягнення різноманітного вигляду згенерованої будівлі ми запроваджуємо інший метод текстурування. Цей метод підтримує три рівні деталізації, що дозволяє додавати фактор рандомізації до вибору текстур. Кожна текстура індексується шестизначним номером (рис. 2.7) і назвою, що стосується фактичного файлу текстури. Перші дві цифри його номера відносяться до типу поверхні, наприклад стелі. Другі дві цифри ідентифікують другий рівень деталізації, наприклад внутрішню та зовнішню стелю. Останні дві цифри вказують точну текстуру. Якщо цифри не вказано, алгоритм вибирає випадкову текстуру в заданому діапазоні (наприклад, стеля або внутрішня стеля). Таким чином, ініціалізація набору текстур кімнати з чотирма цифрами ідентифікаторів призведе до кімнат з різними текстурами на кожну ітерацію.

2.3.5 Побудова коридорів та злиття з сходовою кліткою

Перед створенням коридорів до списку наявних приміщень додаються сходи. Ми застосовуємо алгоритм підключених компонентів до всіх країв внутрішніх стін поточного поверху, щоб знайти всі доступні шляхи, які з'єднують кімнати (рис. 2.9). Перебираючи всі зв'язані графи, ми визначаємо найдовший шлях, який має пряме з'єднання зі сходовою кліткою. Для цього ми застосовуємо алгоритм найкоротшого шляху (напр., Dijkstra) [2], використовуючи об'єднані довжини ребер як ваги. Ми використовуємо кожну координату країв стіни як відправну точку, щоб переконатися, що ми насправді знайшли найдовший шлях.

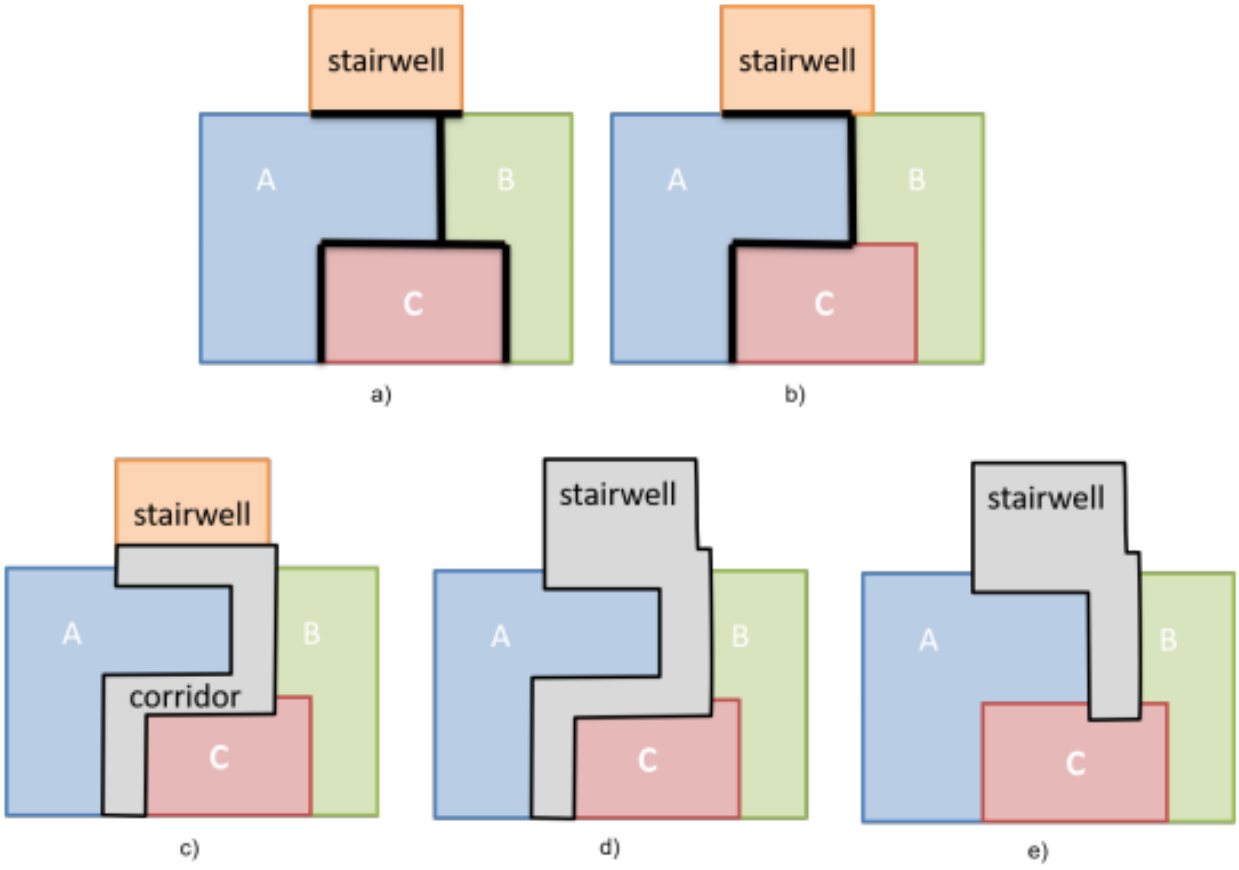


Рисунок 2.9 – Алгоритм підключених компонентів та Dijkstra для побудови коридорів

На всіх поверхах, де коридор не вимагає прямого з'єднання із зовнішньою стіною (наприклад, для вхідних дверей), ми можемо послідовно усунути останній

край, щоб максимізувати ефективну площу будівлі. Кінцевий стан досягається, коли наступне видалене ребро зменшить кількість прилеглих до коридору кімнат.

Найдовший шлях показаний на рисунку 2.9 (b) чорною лінією. Далі він розширюється в ширину, а отриманий багатокутник віднімається від усіх існуючих кімнат (рис. 2.9 (c)); сходова клітка та коридор об'єднані (крок d) шляхом об'єднання багатокутників. Щоб максимізувати розмір кімнати, коридор скорочується край за краєм, доки можна досягти всіх кімнат на поверсі (крок e).

2.3.6 Додавання дверей та вікон

Додавання вікон і дверей відбувається в строгому порядку. Пріоритет мають зовнішні двері, потім внутрішні двері та вікна.

Орієнтуючись на європейський стандарт архітектури, алгоритм розташовує головні вхідні двері в коридорі. Додавши просте правило дозволити головний вхід у вітальню, ми можемо легко змінити планування на американський стиль будівлі.

У реальних будівлях розташування зовнішніх дверей залежить від багатьох факторів, наприклад, місцевості навколо будівлі, сусідства або типу кімнат. Оскільки це не входить у нашу сферу, ми створюємо лише одні двері на першому поверсі, де коридор торкається зовнішньої стіни. Якщо такої стіни немає, вхід створюють на сходовій клітці або, як другий варіант, у найбільшій кімнаті будівлі, якою, як правило, є вітальня.

Розміщення внутрішніх дверей досягається шляхом рекурсивного відстеження того, чи кімната прямо чи опосередковано (через n кімнат) з'єднана з коридором або кімнатою за допомогою зовнішніх дверей. Якщо це не так, алгоритм з'єднує цю кімнату з сусідньою кімнатою, яка ще не має зв'язку з цією кімнатою. Потім алгоритм продовжує роботу з наступними кімнатами, доки всі кімнати не будуть доступні.

Кількість вікон розраховується виходячи з розміру кімнати, використовуючи фіксоване число на квадратний метр. Розташування дверей і вікон у стінах залежить від топології будівлі, відповідно до положення, розмірів і суміжності її

приміщень. Загалом, слід розрізняти зовнішні двері та зовнішні вікна, відповідно внутрішні двері та внутрішні вікна (наприклад, службовий люк між кухнею та їдальнею). Кожна з останніх з’єднує дві кімнати, і, отже, необхідно підтвердити, що обидві кімнати мають спільну стіну та що стіна не заблокована перешкодою (наприклад, сходами).

Зовнішні приміщення та вікна вимагають перевірки лише за наявності вікна чи дверей на цільовій стіні.

Щоб запропонувати динамічне розташування дверей і вікон, до алгоритму додаються правила розміщення First, Last, Best і Random. Ці правила застосовуються, коли алгоритм розміщення вибирає вільні спільні ділянки країв двох суміжних кімнат, які отримують спільні двері або вікно.

3 МЕТОДИ ТА МОДЕЛІ ГЕНЕРАЦІЇ ТРИВИМІРНОЇ СІТКИ ТА ОЦІНКА РЕЗУЛЬТАТУ

3.1 Генерація тривимірної сітки

3.1.1 Кімнати

Кімната керується як замкнутий багатокутник, який не перетинає себе. Цей багатокутник ділиться на трикутник у плоску сітку з одною текстурою, грані якої спрямовані вниз. Зсув висоти поверху розраховується з використанням висоти поверху (FH), тобто товщини фундаменту, і висоти приміщення (RH) (3.1):

$$z\_of\ f\ set = FH + floor \times (2 \times FH + RH) \quad (3.1)$$

Потім отримана сітка видавлюється вгору, щоб блоки FH отримали базову пластину (рис. 3.1).

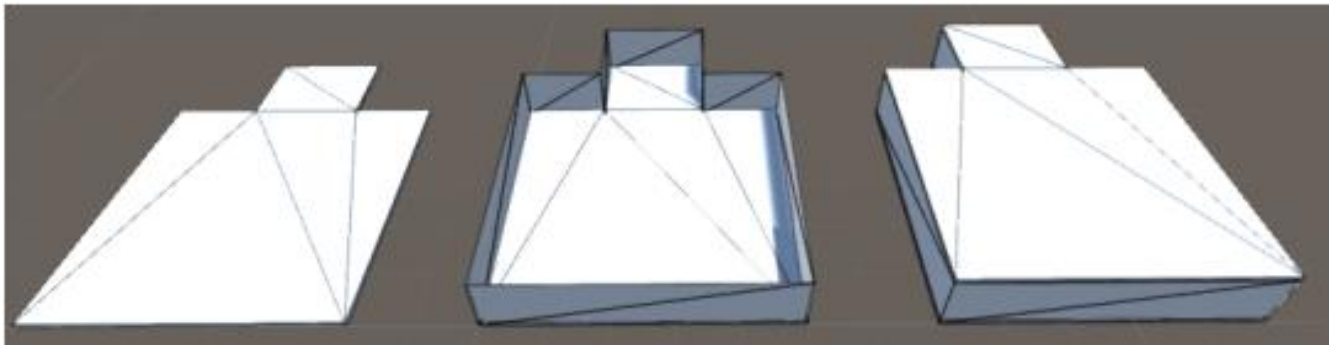


Рисунок 3.1 – Створення підлоги, стін та стелі на сітці

Стіни створюються вздовж зовнішніх країв приміщення заданої товщини. Стеля є копією опорної плити.

Алгоритм видавлювання виявляє граничні ребра багатокутної сітки, створює ідентичні копії проіндексованих вершин у їх точному положенні та копіює їх на певну висоту над оригіналом. Створення цих двох копій є єдиним способом призначити окремі групи індексів, як представлено вище, бічним граням екструзій.

Нарешті сітка закривається ідентичною копією сітки з інвертованим порядком індексів для кожної грані, так що нова поверхня вказує в протилежному напрямку.

3.1.2 Сходи

Сходова сітка складається з перил і ступенів. Кількість кроків можна розрахувати шляхом віднімання вертикального зсуву нижнього поверху з вертикального зміщення верхнього поверху і ділення результату на висоту сходинки. Оскільки краї поручнів ліворуч і праворуч від під'їзду обчислюються та зберігаються в об'єктній моделі, алгоритм просто інтерполює між лівим і правим поруччям, щоб отримати межі сходинок.

На рисунку 3.2 показано, що результат дещо відрізняється від більшості реальних сходів, оскільки сходинки прямокутних сегментів сходів, як правило, прямокутні, але візуальна різниця, на наш погляд, малопомітна та виправдана надійним алгоритмом.

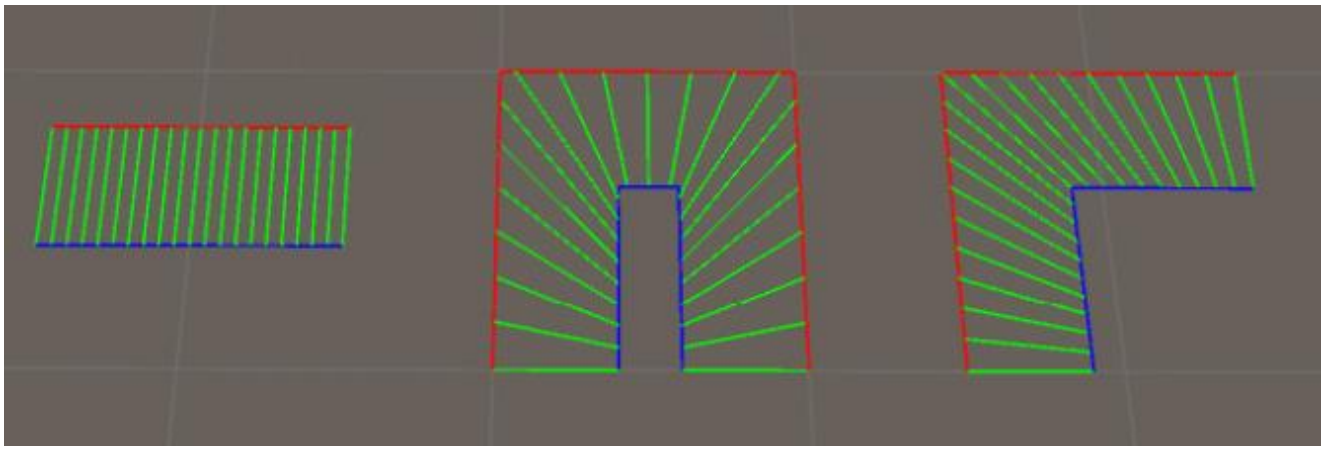


Рисунок 3.2 – Інтерполяція між краями перил сходів

Коли справа доходить до генерації перил (рис. 3.3), ліві та праві векторні пари кожної сходинки зберігаються в масиві, так що розміри кожної сходинки можуть бути доступні після створення сходинок. Ітерація по цих краях дозволяє розміщувати стопки в середніх положеннях країв або створювати суцільну межу шляхом зсуву двовимірних країв до тривимірної сітки.

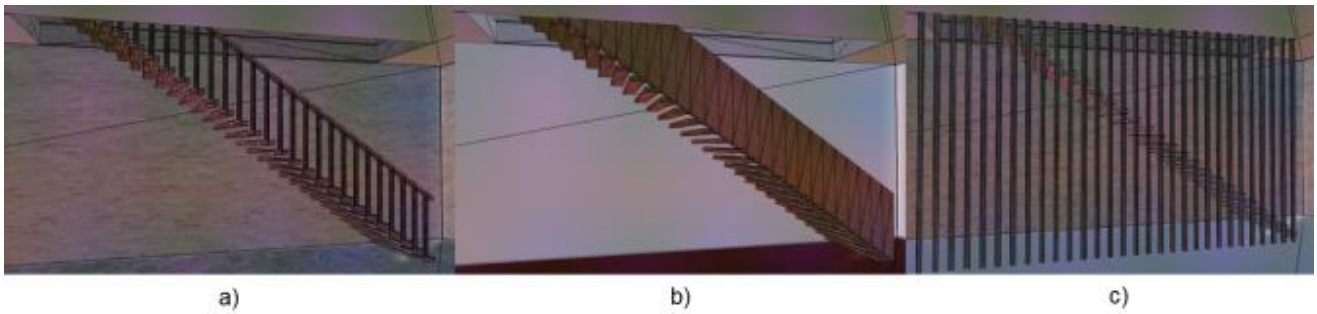


Рисунок 3.3 – Три типи перил: a) палі; b) суцільні; c) палі у висоту стіни

Перила не генеруються, якщо край поруччя перекривається будь-яким краєм стіни навколишньої кімнати, тому є лише поручні на відкритих краях, де хтось може впасти. Краї перил позначаються відповідним чином під час виконання алгоритму Best Fit.

3.1.3 Двері та вікна

Після сходів віконні та дверні вирізи йдуть уздовж призначеної стіни та початкової та кінцевої точок відповідного отвору. Двовимірні лінії екструдуються в тривимірну сітку віднімання, де глибина екструзії визначається товщиною стіни, а ширина – шириною вікна. Отже, ці двовимірні сегменти мають бути екструдовані до тривимірної форми, надаючи двовимірну лінію. Це досягається шляхом видавлювання сегмента спочатку під кутом 90° ліворуч і під кутом −90° праворуч (рис. 3.4).

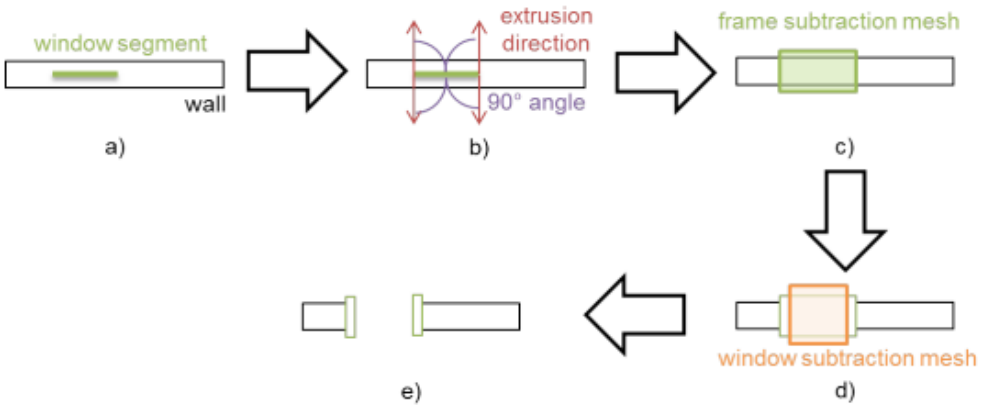


Рисунок 3.4 – Процес двовимірного створення вікна та віконної рами

Потім ця конструкція видавлюється до верху, щоб отримати прямокутну сітку, яка використовується як віконна рама; її можна відняти від сітки кімнати. Друга прямокутна сітка з більшим вертикальним зсувом і меншою висотою та шириною потім віднімається з сітки віконної рами.

Алгоритм для дверей ідентичний, за винятком того, що з дверей знімається поріг. Як можна побачити, CSG значно полегшує процедурне моделювання: замість того, щоб моделювати кожен сегмент стіни окремо, відповідні маніпуляції можна застосувати точно так само до існуючої стіни та легко повторити.

3.1.4 Рівень деталізації

Концепція LOD передбачає кілька етапів якості тривимірної моделі з метою оптимізації використання пам’яті під час процесу візуалізації. Є кілька факторів, які впливають на LOD, наприклад відстань об’єкта до глядача, швидкість його руху та його важливість у сцені. Найбільш впливовим фактором, що впливає на різницю в якості, є кількість вершин сітки, але є й інші фактори, такі як роздільна здатність текстур і складність шейдера. Дослідники підкреслюють корисність LOD у тривимірному моделюванні міста та надають детальну категоризацію LOD у мові розмітки CityGML [5]. Базуючись на цьому визначенні, ми вводимо три рівні LOD для контролю якості створених будівель, які приблизно відповідають LOD 2–4 CityGML (рис. 3.5):

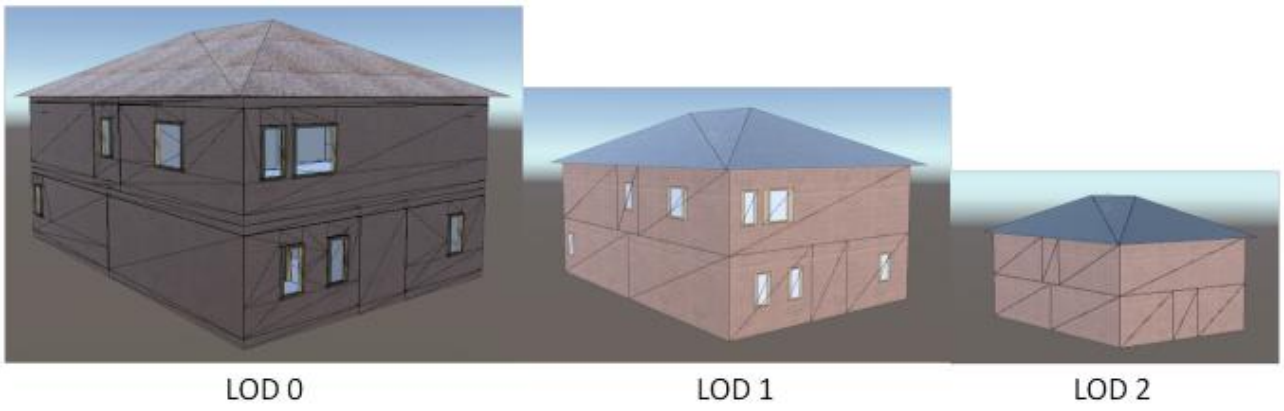


Рисунок 3.5 – Ідентична будівля на трьох рівнях LOD

1) LOD 0 - будівля з інтер'єром, поверхами та сходами,

2) LOD 1 - корпус будівлі з зовнішніми деталями (двері та вікна),

3) LOD 2 - корпус будівлі з даховими конструкціями і простими текстурами. Стінові та віконні сітки та текстури опущені.

Процедура створення сітки будівлі відповідає за створення кожного окремого рівня та включає реалізацію для обробки кожного LOD окремо.

ВИСНОВКИ

Проаналізувавши методи та моделі процедурної генерації віртуальних просторів, можемо зробити наступні висновки:

1. Теоретико-інформаційний огляд методів та моделей процедурної генерації дозволив нам вивести визначення процедурної генерації, як процесу автоматичного створення цифрових активів для ігор, симуляцій або фільмів на основі попередньо визначених алгоритмів і шаблонів, які вимагають мінімальної участі користувача.

Цей підхід є досить привабливим та економічно вигідним зважаючи на економію часу. Проте, в ході теоретичного огляду, нам також вдалося виявити недоліки цього підходу, а саме: надмірна складність; неможливість забезпечити необхідний контроль якості; виникнення необхідності покриття великою кількістю тестів; великі витрати часу; алгоритми процедурної генерації можуть бути дуже складними при розробці; процес процедурної генерації може негативно впливати на продуктивність виконання гри; складність керування результатом, потрібно постійно вносити зміни щоб задовольнити потреби; випадковість.

2. Другий розділ демонструє підхід до процедурної генерації віртуальних будівель. Представлений метод забезпечує доступ до кожної кімнати через коридор, а до кожного поверху можна піднятися унікальними сходами. Це розширює логіку простого генерування сітки за допомогою різноманітних обмежень, які забезпечують валідність створеної будівлі. Ці обмеження випливають із області планування підлоги та розміщення об’єктів у певному просторі. Логічні оператори для сіток, а також для багатокутників виявилися особливо корисними, оскільки правила створення будівлі можна формалізувати подібно до математичного рівняння. Таким чином, секція дверей відповідає простому відніманню двох сіток замість ручного розрахунку набору координат.

3. У третьому розділі було запропоновано методи та моделі створення тривимірних текстурованих моделей будівель. Створення UV-карт є серйозною проблемою, яку ми вирішили, ввівши кортежі груп індексу/сітки. Ці значення допомагали відстежувати текстури в процедурному процесі генерації.

Здійснене дослідження дозволило відкрити нові можливості у процедурній генерації віртуальних просторів на прикладі комп’ютерних ігор. Проте, отримані результати можуть бути використані для створення віртуальних симуляцій, що можуть бути корисними у професійній підготовці та тренуванні військових, що знизить витрати на підготовку та стане вдалим інструментом для економічного збалансування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Aitor Santamar´ıa-Ibirika, Xabier Cantero, Mikel Salazar, Jaime Devesa, Igor Santos, Sergio Huerta, and Pablo G. Bringas. Procedural approach to volumetric terrain generation. The Visual Computer, 30(9):997–1007, Sep 2014. ISSN 1432- 2315. doi: 10.1007/s00371-013-0909-y. URL https://doi.org/10.1007/s00371- 013-0909-y.

2. Brian Johnson and Ben Shneiderman. Tree-maps: A space-filling approach to the visualization of hierarchical information structures. In Proceedings of the 2nd conference on Visualization'91, pages 284–291. IEEE Computer Society Press, 1991.

3. Esben Bach and Andreas Madsen. Procedural Character Generation: Implementing Reference Fitting and Principal Components Analysis. Aalborg University. Department of Computer Science, 2007

4. Filip Biljecki, Hugo Ledoux, and Jantien Stoter. An improved lod specification for 3d building models. Computers, Environment and Urban Systems, 59:25–37, 2016.

5. Handbook of Research on Developing Smart Cities Based on Digital Twins. IGI Global, 2021.

6. Jinmo Kim. Modeling and optimization of a tree based on virtual reality for immersive virtual landscape generation. Symmetry, 8(9):93, 2016.

7. Julian Togelius, Emil Kastbjerg, David Schedl, and Georgios N Yannakakis. What is procedural content generation?: Mario on the borderline. In Proceedings of the 2nd International Workshop on Procedural Content Generation in Games, page 3. ACM, 2011.

8. Julian Togelius, Noor Shaker, and Mark J Nelson. Procedural content generation in games: A textbook and an overview of current research/j. Togelius, N. Shaker, M. Nelson—Berlin: Springer, 2014.

9. Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. ACM Trans. Multimedia Comput. Commun. Appl., 9(1):1:1–1:22, February 2013. ISSN 1551-6857. doi: 10.1145/ 2422956.2422957. URL http://doi.acm.org/10.1145/2422956.2422957.

10. N.V. Barreto and Licinio Roque. A survey of procedural content generation tools in video game creature design. In Second Conference on Computation Communication Aesthetics and X, n/a, 2014.

11. NewTheft. Open all interiors, 2015. URL https://www.gta5-mods.com/scripts/ open-all-interiors.

12. Noor Shaker, Julian Togelius, and Mark J Nelson. Procedural Content Generation in Games. Springer, 2016.

13. Pradyumna Tambwekar, Murtaza Dhuliawala, Lara J Martin, Animesh Mehta, Brent Harrison, and Mark O Riedl. Controllable neural story plot generation via reinforcement learning. arXiv preprint arXiv:1809.10736, 2018.

14. Przemyslaw Prusinkiewicz and Aristid Lindenmayer. The algorithmic beauty of plants. Springer Science & Business Media, 2012.

15. Reinhard Koenig and Katja Knecht. Comparing two evolutionary algorithm based methods for layout generation: Dense packing versus subdivision. AI EDAM, 28 (3):285–299, 2014.

16. Ruben Micha¨el Smelik, Tim Tutenel, Klaas Jan de Kraker, and Rafael Bidarra. A declarative approach to procedural modeling of virtual worlds. Computers & Graphics, 35(2):352–363, 2011.

17. Timothy Roden and Ian Parberry. From artistry to automation: A structured methodology for procedural content creation. Entertainment Computing–ICEC 2004, pages 301–304, 2004.

18. Yun-Gyung Cheong, Mark O Riedl, Byung-Chull Bae, and Mark J Nelson. Planning with applications to quests and story. In Procedural Content Generation in Games, pages 123–141. Springer, 2016.

19. Фоменко В. О. Відеогра з процедурною генерацією рівнів на Unity //Матеріали та програма міжнародної науково-технічної конференції студентів та молодих вчених «Інформатика, математика, автоматика» (IMA – 2021) (Суми, 2021 р.), м. Суми, Україна. - Суми, 2021. - С. 58-59 - https://elitconference.sumdu.edu.ua/proceedings/.

Додаток А

Деривація формули побудови параболи за алгоритмом Форчуна

Необхідна формула наступного вигляду (А.1)

$$f(x) = fx^2+bx+c \quad \text{(А.1)}$$

Згідно з визначенням кривої, відомо що відстань $((x, y_d), (x, f(x)))$ = відстані $((x_f, y_f), (x, f(x)))$, тому, можемо виконати наступні розрахунки

$$\sqrt{(x-x)^2 + (f(x) - y_d)^2} = \sqrt{(x-x_f)^2 + (f(x) - y_f)^2}$$

$$(x-x)^2 + (f(x) - y_d)^2 = (x-x_f)^2 + (f(x) - y_f)^2$$

$$(f(x) - y_d)^2 - \left(f(x) - y_f\right)^2 = (x-x_f)^2$$

$$(f(x)^2 - 2f(x)y_d + y_d^2) - \left(f(x)^2 - 2f(x)y_f + y_f^2\right) = (x-x_f)^2$$

$$2f(x)y_f - 2f(x)y_d + \left(y_d^2 - y_f^2\right) = \left(x-x_f\right)^2$$

$$2f(x)\left(y_f - y_d\right) = \left(x-x_f\right)^2 - \left(y_d^2 - y_f^2\right)$$

$$2f(x)\left(y_f - y_d\right) = \left(x-x_f\right)^2 + \left(y_f^2 - y_d^2\right)$$

$$f(x) = \frac{\left(x-x_f\right)^2}{2\left(y_f - y_d\right)} + \frac{y_f^2 - y_d^2}{2\left(y_f - y_d\right)}$$

$$f(x) = \frac{(x-x_f)^2}{2(y_f - y_d)} + \frac{(y_f - y_d)(y_f + y_d)}{2(y_f - y_d)}$$

$$f(x) = \frac{(x-x_f)^2}{2(y_f - y_d)} + \frac{(y_f + y_d)}{2}$$

Додаток Б

Алгоритм Best Fit для знаходження оптимального простору розміщення полігонів

```
Input: Polygon p2 to be fit in polygon p1
Output: Polygon p such that p1 contains p2 and od is maximal, overlay
        distance od, edges that overlay eo
if p1.isClockwise ≠ p2.isClockwise then
    p1.invertEdges;
end
if p1.area < p2.area then
    return null;
end
res ← empty polygon;
foreach edge e1 in p1 do
    outerEdgeDir ← (e1.p1 - e1.p2).normalize;
    foreach edge e2 in p2 do
        innerEdgeDir ← (e2.p1 - e2.p2).normalize;
        tmp ← copy of p2;
        tmp.move(e1.p1 - e2.p1);
        rotationAngle ← angle between outerEdgeDir and innerEdgeDir;
        rotationPoint ← e1.p1;
        tmp.rotate(rotationAngle, rotationPoint);
        if not p1.contains(tmp) then
            continue;
        end
        tmpOverlayDistance ← calcOverlayDistance(p1, tmp);
        tmpOverlayEdges ← cntOverlayLines(p1, tmp);
        if tmpOverlayDistance > od then
            od ← tmpOverlayDistance;
            eo ← tmpOverlayEdges;
            res ← tmp;
        end
    end
end
return res;
```