НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “ Комп’ютерні науки ”

на тему Класифікація основних тривимірних будівельних елементів BIM моделі

Виконав: студент 4 курсу, групи ТР-52

Кунатова Олександра Анатоліївна _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доцент, к.т.н. Шаповалова Світлана Ігорівна _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ

на дипломну роботу студенту

Кунатовій Олександрі Анатоліївні
(прізвище, ім’я, по батькові)

1. Тема роботи “Класифікація основних тривимірних будівельних елементів BIM моделі”

керівник роботи доцент, к.т.н. Шаповалова Світлана Ігорівна
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201__р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи BIM модель для CAD-системи Allplan, класи розпізнавання: стіна, колона, балка, перекриття

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуючі програмні рішення та можливі засоби реалізації класифікатору будівельних елементів BIM моделі, обґрунтувати обрані програмні застосунки та шляхи розробки програмних додатків, розробити програмне забезпечення, розробити користувацький інтерфейс, зробити висновки за результатами роботи

5. Перелік ілюстративного матеріалу

1. Визначення BIM моделі. 2. Етапи проектування будівлі. 3. Інструменти побудови будівельних конструкцій. 4. Специфікація тривимірного тіла через атрибути. 5. Базові підходи до класифікації. 6. Визначення габаритних характеристик. 7. Дерева рішень. 8. Архітектура системи. 9. Інтерфейс користувача. 10. Результати обчислювальних експериментів.

6. Дата видачі завдання ”___”_____201___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі		
3.	Розробка архітектури та загальної структури системи		
4.	Розробка структур окремих підсистем		
5.	Програмна реалізація системи		
6.	Оформлення пояснювальної записки		
7.	1 Захист програмного продукту		
8.	2 Передзахист		
9.	Захист		

Студент _____ Кунатова О.А.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Шаповалова С.І.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

BIM модель будівлі може містити як стандартні описи будівельних конструкцій, так і описи у вигляді 3D тіл у випадку необхідності побудови елементів складної геометрії. При передачі в розрахункову систему CAE кожен з таких елементів має бути описаний як стандартний будівельний компонент.

Мета роботи — створити програмне забезпечення автоматичного формування специфікації (semantic labeling) тривимірних геометричних об’єктів за BIM моделлю. Метод класифікації — градієнтний бустинг над деревами рішень, реалізований в бібліотеці LightGBM. Програмна система апробована на 50 BIM моделях. Результати випробовування становлять 98,5% коректного розпізнавання.

Записка містить 42 сторінки, 15 рисунків, 2 таблиці, 3 додатки і 25 посилань.

Ключові слова: КЛАСИФІКАЦІЯ 3D ТІЛ, BIM МОДЕЛЬ, ГРАДІЄНТНИЙ БУСТИНГ, ПЛАСТИНЧАТО-СТРИЖНЕВА МОДЕЛЬ, МАШИННЕ НАВЧАННЯ, БУДІВЕЛЬНІ ЕЛЕМЕНТИ.

ABSTRACT

BIM model can contain both standard descriptions of building constructions and descriptions in the form of 3D shapes in the case of the need to construct elements of complex geometry. When transferred to the CAE-system, each of these elements should be described as a standard building component.

The purpose of the work is to create a software for the automatic semantic labeling of three-dimensional geometric objects according to the BIM model. Method of classification — gradient boosting on decision trees, implemented in the LightGBM library. The software is tested on 50 BIM models. The results of the tests are 98.5% correct recognition.

The note contains 42 pages, 15 figures, 2 tables, 3 attachments and 25 links.

Keywords: CLASSIFICATION OF 3D SHAPES, BIM MODEL, GRADIENT BOOSTING, PLATE-BEAM MODEL, MACHINE LEARNING, BUILDING ELEMENTS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BIM — Building Information Model.

CAE — Computer-aided engineering.

CAD — Computer-aided design.

IFC — Industry Foundation Classes.

VCNN — Volumetric convolutional neural network.

B-rep — Boundary representation.

IDE — Integrated Development Environment.

API — Application Programming Interface.

GUI — Graphical User Interface.

ВСТУП

Одним з етапів проектування будівлі є створення архітектурної моделі. Існує багато програмних засобів, які дозволяють створювати BIM модель будівлі (Building Information Model). Такі системи автоматизованого проектування будівлі пропонують архітектору набір інструментів для побудови основних конструктивних елементів будівлі (стіна, перекриття, колона, балка). Проте іноді можливості цих інструментів не дозволяють створити будівельний елемент з необхідною геометрією. У таких випадках архітектор вдається до використання інструментів 3D моделювання геометричними примітивами (паралелепіпед, циліндр, шар та інші). Проблема полягає в тому, що для подальшої роботи, наприклад побудови пластинчато-стрижневої моделі для передачі у розрахункову систему (CAE), кожен елемент моделі має бути описаний як один зі стандартних конструктивних елементів будівлі. Задачу опису кожного 3D об'єкту має розв'язувати експерт, що веде за собою додаткові витрати ресурсів висококваліфікованих спеціалістів. Тому автоматичне розпізнавання тривимірних конструктивних будівельних елементів BIM моделі є актуальним і має практичну значущість.

Значна частина робіт [1 — 3], присвячених класифікації будівельних компонентів BIM моделі, розв'язують задачу класифікації компонентів вже існуючої будівлі на основі хмар точок.

Конструктивні будівельні компоненти BIM моделі є тривимірними об'єктами. До базових підходів розв'язання задачі класифікації 3D об'єктів можна віднести: 1) застосування об'ємних згорткових нейронних мереж VCNN (Volumetric convolutional neural network); 2) градієнтний бустинг (gradient boosting).

VCNN — це нейронні мережі, призначені для аналізу тривимірних даних. Розпізнавання різноманітних об'єктів у реальному часі представлено у роботі [4].

Згідно першому підходу, розпізнається безпосередньо 3D об'єкт. Однак для задачі класифікації будівельних компонентів доцільно використовувати

класифікацію на основі градієнтного бустингу, тому, що для поточної прикладної задачі габаритні характеристики є визначальними.

Градієнтний бустинг — це алгоритм машинного навчання, призначений для вирішення задач регресії та класифікації, вперше запропонований у роботі [5].

В даному дослідженні було обрано метод градієнтного бустингу над деревами рішень тому, що він відноситься до базового підходу. Інші методи пропонують оптимізації для розв’язання специфічних задач.

Мета роботи — створити програмне забезпечення автоматичного формування специфікації (semantic labeling) тривимірних геометричних об’єктів за BIM моделлю.

У першому розділі висвітлюється постановка задачі класифікації основних тривимірних будівельних елементів BIM моделі, визначено поняття BIM моделі, наводиться опис специфікації об’єктів BIM моделі та розглядаються підходи до рішення проблеми класифікації будівельних елементів.

Другий розділ присвячено розгляду існуючих програмних систем класифікації тривимірних будівельних компонентів, методу градієнтного бустингу, розглядається проблема визначення габаритних характеристик тривимірних об’єктів.

У третьому розділі наведено програмні засоби та середовища розробки, які були використані під час розробки програмної системи.

Четвертий розділ містить опис параметрів градієнтного бустингу, архітектури програмної системи та результати її апробації.

У п’ятому розділі наведено системні вимоги, опис інсталяції системи та сценарій роботи користувача із системою.

1 ЗАДАЧА КЛАСИФІКАЦІЇ ОСНОВНИХ ТРИВИМІРНИХ БУДІВЕЛЬНИХ ЕЛЕМЕНТІВ BIM МОДЕЛІ

На даний момент існує багато будівельних CAD-систем, які створюють BIM моделі (Allplan, Revit, ArchiCAD та інші). В цих системах проектується фізична модель будівлі.

Для розрахунків навантажень використовуються такі CAE-системи: Dlubal, STARK ES, САПФИР. Ці системи проводять розрахунки на основі аналітичної, наприклад, пластинчато-стрижневої, моделі будівлі.

Задача полягає в тому, що на основі опису BIM необхідно створити специфікацію будівельних компонентів. Вхідною інформацією є безпосередньо BIM модель, яка представляє вигляд будівлі зовні та її приміщень. Вихідною інформацією є специфікація компонентів моделі.

1.1 Представлення BIM моделі

За визначенням The National BIM Standard-United States, BIM — “це загальний ресурс знань для інформації про об’єкт, що формує надійну основу для прийняття рішень протягом усього його життєвого циклу: від самої ранньої концепції до знесення” [6].

Програмне забезпечення BIM використовується особами, підприємствами та урядовими установами, які планують, проектують, будують, експлуатують та підтримують різноманітні фізичні інфраструктури. Тому BIM охоплює більше, ніж просто геометрію. Вона також охоплює просторові відносини, аналіз світла,

географічну інформацію, кількість та властивості будівельних компонентів (наприклад, деталі виробників).

На рисунку 1.1 представлені приклади спроектованих у CAD-системах BIM моделей.

Рисунок 1.1 — Приклади BIM моделей

Над створенням BIM моделі на різних етапах працюють такі спеціалісти: архітектори, конструктори по залізобетону, конструктори інженерних систем, кошторисники, дизайнери та інші.

Програмне забезпечення BIM визначає об’єкти параметрично, тобто об’єкти визначаються як параметри та відношення до інших об’єктів, так що, якщо змінюється споріднений об’єкт, залежні від нього автоматично змінюються. Кожен елемент моделі може містити атрибути для автоматичного відбору та упорядкування, надання оцінок витрат, а також відстеження та замовлення матеріалів.

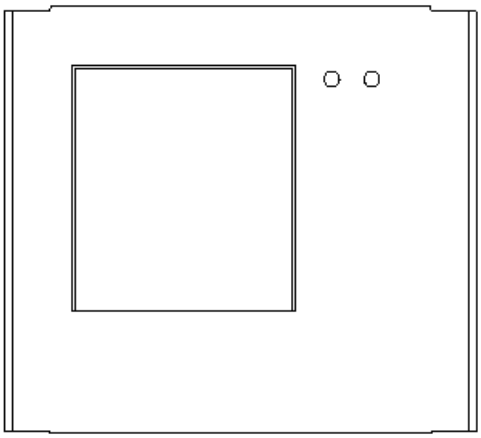
Використання BIM спрощує управління будівельним об’єктом протягом усього життєвого циклу — з передпроектної підготовки до заморожування/реконструкції. Дана технологія заснована на застосуванні точної проектної інформації на кожному етапі будівництва. У великих і малих проектних системах неминуче залучено кілька груп архітекторів, проектувальників,

конструкторів, інженерів, виробників, підрядників, будівельників — і питання про управління інформацією і організації спільної роботи постає вкрай гостро.

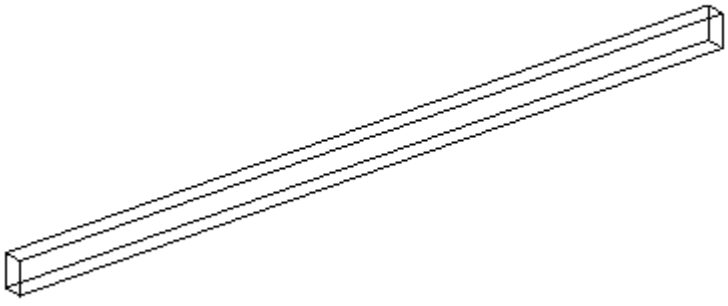
1.2 Представлення елемента BIM моделі

Кожна будівельна конструкція в BIM моделі представляється як окремий тривимірний об’єкт. Приклади представлення основних будівельних конструкцій зображені на рисунку 1.2.

а) перекриття



б) стіна



в) балка

г)колона

Рисунок 1.2 — Приклади представлення основних будівельних конструкцій

Складна BIM модель може містити сотні таких тривимірних об’єктів. Кожна CAD-система зберігає модель будівлі у власному форматі. Існує також незалежний від певної системи формат інформаційної моделі будівлі IFC (Industry Foundation Classes) [7].

Кожна CAD-система надає інструментарій для побудови стандартних будівельних компонентів BIM моделі. Такими інструментами можуть бути інструмент “Стіна”, інструмент “Перекриття” та інші.

На рисунку 1.3 зображено інструментарій блоку “Архітектура” CAD-системи Allplan.

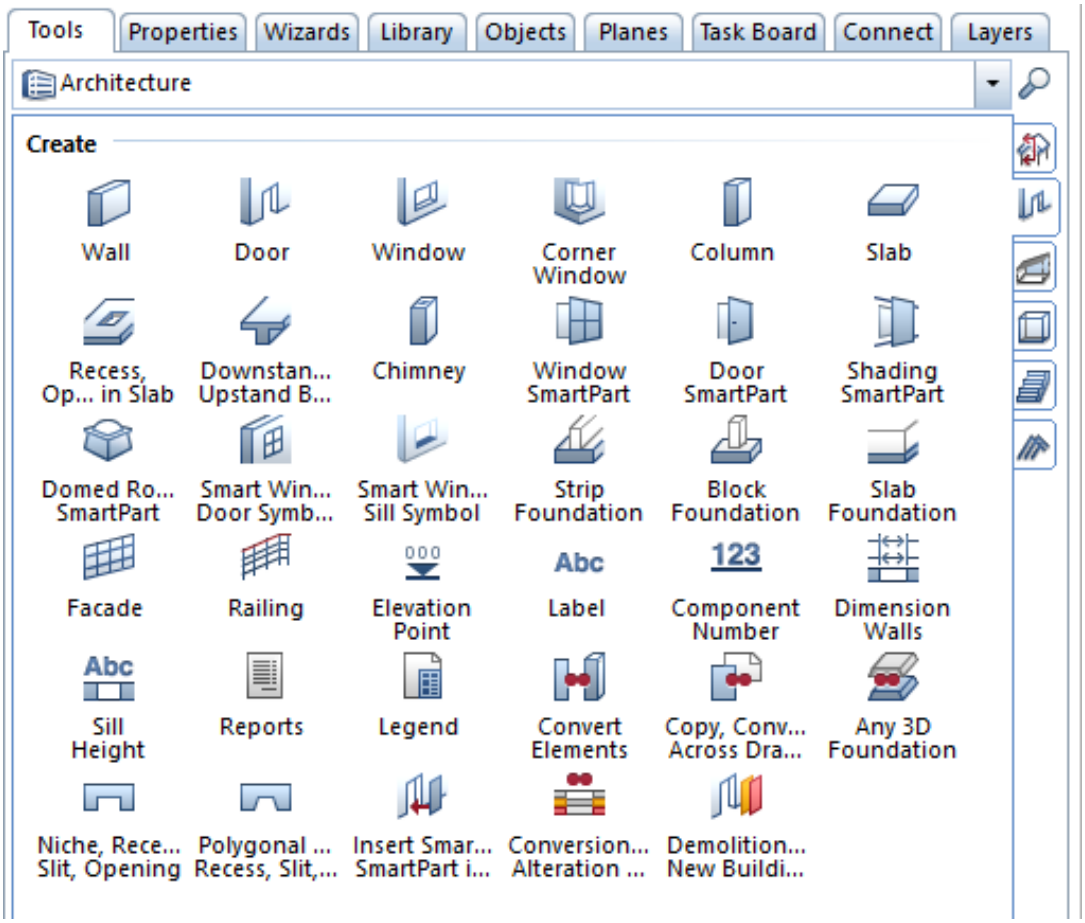


Рисунок 1.3 — Інструменти блоку “Архітектура” будівельної CAD-системи Allplan

У такому випадку кожен елемент BIM модель буде мати специфікацію як стандартний будівельний компонент. Використання стандартних інструментів накладає певні обмеження на геометрію елемента. Тому, у деяких випадках, коли

можливостей цих інструментів недостатньо, проектувальник вдається до використання інструментів побудови геометричних примітивів (паралелепіпед, циліндр, сфера тощо). У такому випадку створений елемент не буде мати опису як стандартного будівельного компоненту.

Існує декілька основних способів визначення геометрії 3D об’єкту: полігональна сітка, граничне представлення та комбінація декількох тривимірних тіл за допомогою булевих операцій. Такими операціями є перетин, об’єднання та віднімання декількох 3D тіл.

Полігональна сітка — це спосіб представлення тривимірного об’єкту, задається сукупністю граней, які є сукупністю направлених ребер, а ребра у свою чергу складаються кожен з двох вершин. Грані полігональної сітки можуть бути трикутниками, чотирикутниками, іншими багатокутниками та мати отвори.

Граничне представлення (B-rep) — це спосіб представлення об’ємного тіла за допомогою опису його границь [8]. Суть граничного представлення полягає в тому, що тверде тіло описується як замкнена просторова область, обмежена набором елементарних поверхонь (граней), зі спільними утворюючими контурами (ребрами) на границі поверхонь та ознакою зовнішньої або внутрішньої сторони поверхні.

Для отримання складних форм до тривимірних елементарних тіл застосовують булеві операції:

— перетин;

— об’єднання;

— віднімання.

1.3 Специфікація будівельних компонентів

Специфікація містить інформацію про належність кожного компонента моделі будівлі певному класу будівельної конструкції.

Наприклад, візьмемо двоповерхову житлову будівлю (рисунок 1.4).

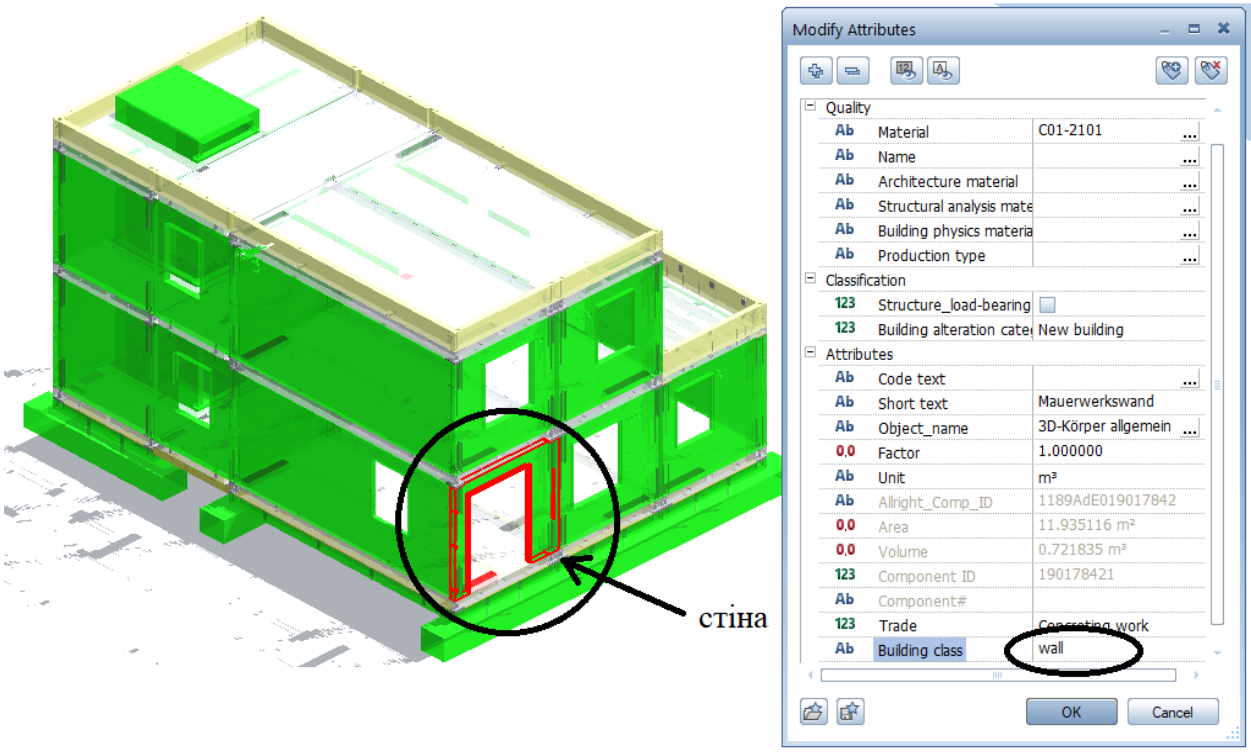


Рисунок 1.4 — Розпізнавання компоненту BIM моделі двоповерхової житлової будівлі

На відображенні у вигляді анімації BIM моделі виокремлено об’єкт, який в специфікації має бути описаний як стіна. У BIM моделі він визначається як 3D тіло, його геометрія визначена за допомогою полігональної сітки та складається з 271 грані.

В цьому прикладі задача полягає у визначенні даного 3D тіла як стіни. Це має відобразитись у відповідному атрибуті “Building class”. В специфікації тривимірного тіла виокремлено відповідне поле, заповнене значенням класу стіна (wall), що відповідає результату розпізнавання.

Результатом розв’язання поточної задачі є заповнений атрибут “Building class” кожного 3D об’єкту BIM моделі відповідним значенням класу. Після заповнення, специфікація тривимірних об’єктів забезпечує повну інформацію для побудови пластинчато-стрижневої моделі будівлі.

1.4 Підходи до рішення задачі класифікації будівельних компонентів

Будівельні компоненти BIM моделі є тривимірними об’єктами. До базових підходів розв’язання задачі класифікації 3D об’єктів можна віднести:

1) застосування об’ємних згорткових нейронних мереж VCNN (Volumetric convolutional neural network);

2) градієнтний бустинг (gradient boosting).

VCNN — це нейронні мережі, призначені для аналізу тривимірних даних. Об’ємні згорткові нейромережі використовуються найчастіше при реалізації об’ємного комп’ютерного зору. Вхідною інформацією для класифікатора на основі VCNN є воксельне представлення тривимірного об’єкту (рисунок 1.5).

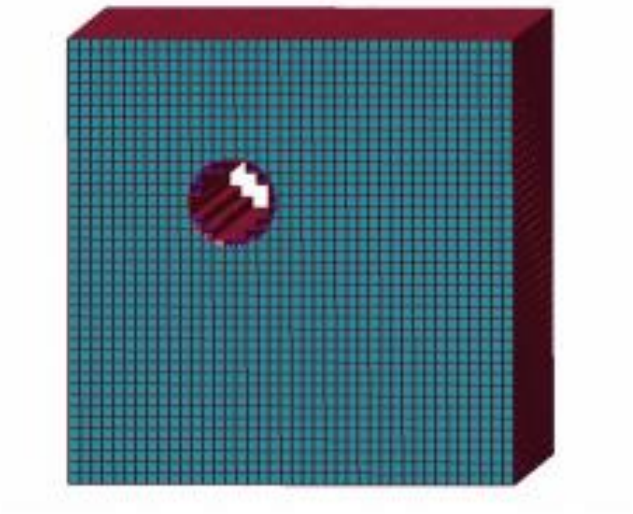


Рисунок 1.5 — Воксельне представлення тривимірного тіла

Воксельне представлення є об’ємним аналогом растрового представлення фігури. Воно являє собою тривимірну матрицю вокселей, кожен елемент якої є булевим значенням [9].

Значним недоліком VCNN є те, що при великій вибірці навчальних прикладів навчання є довготривалим процесом. А також, при подальшому використанні натренованої мережі процес класифікації 3D об’єкта є тривалим через те, що

генерація воксельного представлення з граничного представлення або з полігональної сітки та подальша обробка воксельного представлення згортковою нейромережею є ресурсоемними задачами.

Для задачі класифікації будівельних компонентів доцільно використовувати класифікацію на основі градієнтного бустингу, тому, що для поточної прикладної задачі габаритні характеристики є визначальними.

Градієнтний бустинг — це алгоритм машинного навчання, призначений для вирішення задач регресії та класифікації.

На відміну від об’ємних згорткових нейронних мереж, класифікатор, побудований за допомогою алгоритмів градієнтного бустингу, може вирішити цю задачу на декілька порядків швидше. Проте застосування градієнтного бустингу можна використовувати тільки для задач в яких всі класи дозволяють використовувати єдиний кортеж ознак.

2 АНАЛІЗ ПРОБЛЕМИ КЛАСИФІКАЦІЇ БУДІВЕЛЬНИХ ЕЛЕМЕНТІВ BIM МОДЕЛІ

Розв’язання проблеми автоматичної класифікації тривимірних будівельних елементів BIM моделі наразі є актуальною задачею. Існують програмні системи, які вирішують цю проблему для BIM моделей вже існуючих будівель на основі хмар точок.

2.1 Аналіз існуючих програмних систем класифікації будівельних елементів

Значна частина розробок по класифікації будівельних компонентів BIM моделі присвячена класифікації компонентів вже існуючої будівлі на основі хмар точок. Розпізнавання будівельних елементів має вирішальне значення у процесі оцифрування архітектурної спадщини.

Розв’язання задачі класифікації будівельних конструкцій запропоновано у публікаціях [1, 2]. У роботі [1] вирішено задачу класифікації об’єктів BIM моделі існуючої будівлі на основі алгоритму машинного навчання SVM (support-vector machines) на такі класи: підлога, стеля, дах, стіна і балка. Такі BIM моделі будуються на основі даних хмар точок. Результатом роботи є семантична інформаційна модель будівлі.

У роботі [2] наводиться методика класифікації об’єктів вже існуючої будівлі на класи: підлога, стеля, дах, стіна, вікно та двері, виходячи з геометрії компонентів та контекстної інформації.

Робота [3] присвячена вирішенню на основі VCNN проблеми семантичної сегментації інтер'єру приміщень, представлених у вигляді хмар точок на такі класи: стіна, стіл, стілець, людина, полиця, об'єкт, монітор. У роботі будується об'ємне представлення даних для того, щоб ефективно генерувати велику кількість навчальних прикладів, необхідних для навчання згорткової нейронної мережі. Вхідною інформацією є хмара неопрацьованих точок. Вихідною інформацією є помічена хмара точок, де кожній точці співставлено клас.

Для того, щоб мати можливість використовувати швидку і просту архітектуру VCNN, хмара точок вокселізована, і в першу чергу формується геометричне представлення. Для маркування навчальних даних ручну класифікацію хмари точок переносять у воксельну область за допомогою голосування більшості відповідних точок у вокселі. Як тільки воксельне представлення згенеровано, воно подається на вхід до VCNN, яка призначена для обробки 3D воксельних даних. Результат класифікатора VCNN присвоюється центральному вокселю, і класифікація здійснюється для забезпечення безперервної щільної класифікації всіх вокселів.

Таким чином, існуючі програмні системи вирішують проблему класифікації вже існуючих будівель за хмарами точок, проте не вирішують задачі класифікації основних будівельних компонентів (стіни, колони, балки та перекриття) за BIM моделлю, яка спроектована у CAD-системі.

2.2 Метод градієнтного бустингу

Одним із підходів до вирішення завдань навчання полягає в комбінуванні моделей. Дві основні конкуруючі ідеї даного підходу — беггінг (bagging) [10] і бустінг (boosting) [11]. Перший з них полягає в побудові множини незалежних між собою моделей з подальшим прийняттям рішення шляхом голосування в разі завдання класифікації і усереднення в разі регресії. Даний підхід реалізований в алгоритмі випадкових дерев (random trees або random forest). Бустінг, на противагу

беггінгу, навчає кожну наступну модель з використанням даних про помилки попередніх моделей.

Виділяють наступні типи алгоритмів на основі ансамблів методів: градієнтний бустинг над деревами рішень [5], випадковий ліс (random forest) [12], DART (Dropouts meet Multiple Additive Regression Trees) [13], GOSS (Gradient-based One-Side Sampling) [14].

Градієнтний бустинг — це алгоритм машинного навчання, призначений для вирішення задач регресії та класифікації, вперше запропонований у роботі [5]. Цей алгоритм полягає у комбінуванні (ансамблі) слабких прогнозуючих моделей (наприклад, дерев рішень) у більш сильну модель. Такі ансамблі методів застосовують у машинному навчанні для отримання більшої ефективності у вирішенні задач, ніж це можуть дати окремі моделі. У градієнтному бустингу моделі будуються послідовно.

Навчальна вибірка представляє собою набір пар $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ вектору вхідних змінних x та вихідної змінної y. Задача навчання полягає в тому, щоб використовуючи навчальну вибірку знайти апроксимуючу функцію $\hat{F}(x)$ до функції $F(x)$, яка мінімізує очікуване значення деякої заданої функції втрат за формулою (2.1):

$$\hat{F} = \arg\min_{F} \mathbb{E}_{x,y} [L(y, F(x))] \tag{2.1}$$

де $L(y, F(x))$ — функція втрат.

Метод градієнтного бустингу шукає апроксимуючу функцію $\hat{F}(x)$ у вигляді зваженої суми функцій $h_i(x)$ деякого класу H, які вважаються слабкими моделями. Таким чином, формула (2.2) відображає визначення функції $\hat{F}(x)$:

$$\hat{F}(x) = \sum_{i=1}^{M} \gamma_i h_i(x) + const \tag{2.2}$$

де M — кількість слабких моделей,

γ_i — ваговий коефіцієнт,

$h_i(x)$ — функція слабкої моделі.

Процес знаходження $\hat{F}(x)$ називається навчанням (тренуванням, налаштуванням) моделі, процес визначення виходу *y* з деякого входу *x* за допомогою вже побудованої моделі — передбаченням.

Таким чином, у випадку градієнтного бустингу над деревами рішень, на виході алгоритму навчання ми отримуємо набір з M дерев рішень, і для здійснення передбачення, тобто визначення виходу *y* для нового об'єкта *x*, слід обчислити суму за формулою (2.3):

$$y = T_0(x) + \nu \cdot \sum_{m=1}^{M} T_m(x) \tag{2.3}$$

де T_0 — перше дерево рішень,

ν — коефіцієнт масштабування,

M — загальна кількість побудованих дерев рішень,

T_m — *m*-е дерево рішень.

Фінальний класифікатор представляється у вигляді лінійної комбінації класифікаторів. Пошук оптимальних значень коефіцієнтів цієї лінійної комбінації достатньо трудомістке завдання, тому в градієнтному бустингу використовується жадібний алгоритм поступового додавання класифікаторів [15].

Перевагами градієнтного бустингу є:

— хороша узагальнююча здатність, за допомогою градієнтного бустингу вдається будувати класифікатори, що перевершують за якістю базові алгоритми;

— власні накладні витрати бустингу невеликі, час побудови моделі практично повністю визначається часом навчання базових алгоритмів;

— можливість ідентифікувати об'єкти які є шумовими викидами;

— стійкість до перенавчання.

Проте, алгоритм градієнтного бустингу має і свої недоліки:

— жадібна стратегія послідовного додавання призводить до побудови неоптимального набору базових алгоритмів;

— бустинг може призводити до побудови громіздких композицій, що складаються з сотень алгоритмів. Такі композиції виключають можливість змістовної інтерпретації, вимагають великих обсягів пам’яті для зберігання базових алгоритмів і істотних витрат часу на обчислення класифікацій.

2.3 Застосування градієнтного бустингу до класифікації будівельних елементів BIM моделі

Задача полягає в тому, щоб побудувати класифікатор на основі градієнтного бустингу над деревами рішень для класифікації тривимірних будівельних елементів BIM моделі за габаритними характеристиками.

Визначальними характеристиками тривимірних будівельних елементів є: довжина (length), ширина (width), висота (height).

Вхідною інформацією є вектор *x* габаритних характеристик об’єкту — довжина, ширина, висота. Вихідною інформацією є передбачення *y* — клас об’єкту. Такими класами є стіна, колона, балка, перекриття та допоміжний клас “інше”. Передбачення *y* представляється як вектор ймовірностей приналежності об’єкту до кожного з п’яти класів.

Результатом навчання є послідовність дерев рішень.

2.4 Проблема визначення габаритних характеристик об’єктів

Додаткова проблема полягає в тому, що габаритні характеристики необхідно знайти у випадку, коли грані будівельного компоненту розташовані не паралельно вісям координат.

Приблизним розв’язком проблеми знаходження габаритних характеристик є побудова орієнтованого обмежуючого прямокутнику (axis-aligned bounding box), приклад якого для криволінійної стіни зображено пунктирною лінією на рисунку 1.4.

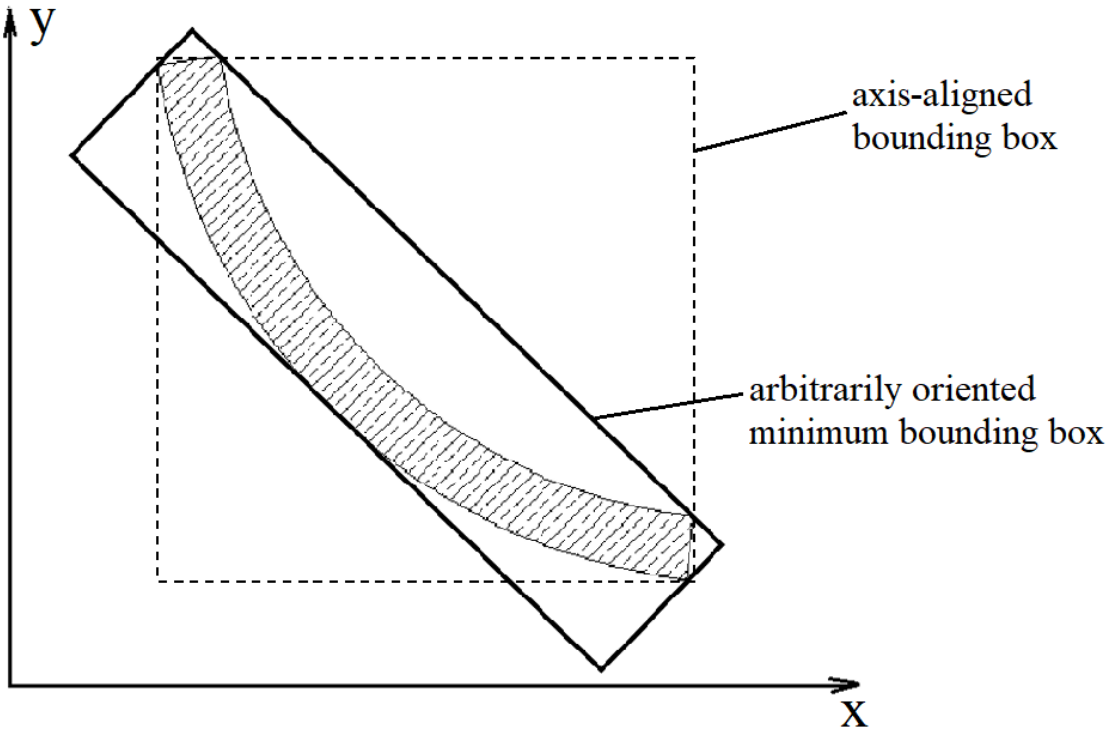


Рисунок 2.1 — Обмежуючі прямокутники криволінійної стіни (вид зверху)

Більш точним рішенням є побудова довільно орієнтованого мінімального обмежуючого прямокутнику (arbitrarily oriented minimum bounding box) — жирна лінія на рисунку 1.4. Метод побудови такого обмежуючого паралелепіпеду описано у роботі [16].

2.5 Засоби програмної реалізації градієнтного бустингу

Метод градієнтного бустингу реалізовано в бібліотеках: LightGBM [17], XGBoost [18], scikit-learn [19] та інших. Найбільш ефективними є перші дві, інші мають нижчу ефективність.

Бібліотека LightGBM — це швидка, розподілена, високопродуктивна бібліотека градієнтного бустингу, що базується на алгоритмі над деревами рішень, використовується для ранжирування, класифікації та багатьох інших завдань машинного навчання.

Бібліотека LightGBM дозволяє обрати одну або декілька з наступних метрик: площа під кривою помилок (Area Under Curve), функції втрат: absolute loss, square loss, logarithmic loss, Huber loss та інші.

Бібліотека XGBoost — оптимізована розподілена бібліотека градієнтного бустингу, розроблена для високоефективної, гнучкої та портативної роботи. Вона реалізує алгоритми машинного навчання градієнтного бустингу. XGBoost забезпечує паралельний бустинг дерев, дозволяє швидко і точно вирішувати багато наукових проблем.

Висновки до розділу 2

У другому розділі було обґрунтовано використання градієнтного бустингу над деревами рішень для розв’язання задачі класифікації будівельних компонентів BIM моделі. Проаналізовано існуючі рішення проблеми розпізнавання будівельних елементів та розглянуто засоби програмної реалізації градієнтного бустингу.

3 ЗАСОБИ РОЗРОБКИ

Програмна система була розроблена за допомогою програмних середовищ розробки (IDE) Visual Studio Community 2017 [20] та JetBrains PyCharm Community 2019 [21]. Графічний інтерфейс користувача створено за допомогою технології WPF [22] мовою програмування C#. Взаємодія розробленої системи з CAD-системою Allplan реалізовано з використанням Allplan NOI API.

Основна частина системи реалізована мовою C++, для створення класифікатору на основі градієнтного бустингу використовувалась мова програмування Python. Взаємодія між частиною системи на C++ з частиною на Python реалізована з використанням Python/C API [23].

3.1 Середовище розробки Visual Studio Community 2017

Програмне середовище розробки Visual Studio Community 2017 містить редактор коду, який підтримує IntelliSense, а також механізми рефакторингу коду. Інтегрований відлагоджувач працює як на рівні вихідного коду, так і на рівні машини. Іншими засобами Visual Studio є: конструктор для створення форм Windows, WPF і веб-додатків, інструмент для створення класів, проектування баз даних і т.д.

Visual Studio містить набір інструментів програмування (IDE), який включає можливість створення додатків на таких мовах програмування, як C#, C++, Visual Basic, J#, F# та іншні. Microsoft Visual Studio дозволяє створювати окремі програми, а також мережеві програми, веб-служби та веб-сайти.

3.2 Середовище розробки JetBrains PyCharm Community 2019

Програмне середовище розробки PyCharm — це інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний відлагоджувач та інструмент для запуску юніт-тестів. PyCharm розроблена компанією JetBrains на основі IntelliJ IDEA.

Програмне середовище PyCharm містить наступні функціональні можливості:

— допомога у написанні та аналізі коду, з завершенням коду, підсвічуванням синтаксису та помилок, інтеграцією linter та швидкими виправленнями;

— проектування та кодова навігація: спеціалізовані види проектів, перегляд структури файлів та швидке перемикання між файлами, класами, методами та застосуваннями;

— рефакторинг Python, включаючи перейменування, виділення методу, введення змінної, введення константи, підняття і опускання методу тощо;

— інтегрований відлагоджувач Python;

— інтеграція з системами управління версіями.

3.3 Технологія WPF

Windows Presentation Foundation (WPF) — це графічна підсистема Microsoft для створення користувальницьких інтерфейсів у Windows-додатках.

WPF використовує XAML, мову на основі XML, для визначення та зв’язування різних елементів інтерфейсу. Програми WPF можна розгортати як автономні програми або розміщувати їх як вбудований об’єкт на веб-сайті. WPF має на меті уніфікувати ряд загальних елементів інтерфейсу користувача, таких як 2D/3D рендеринг, фіксовані та адаптивні документи, типографіка, векторна графіка

та анімація. Ці елементи потім можуть бути зв’язані та маніпулювати на основі різних подій, взаємодій користувача та прив’язки даних.

Прив’язка даних — це гнучкий механізм, який дозволяє через розширення розмітки XAML пов’язувати різні дані (від значень властивостей елементів управління до загальнодоступних властивостей, що реалізують поля бази даних через Entity Framework). Прив’язка даних представлена класом Binding, що дозволяє використовувати прив’язки не тільки в коді, але і в розмітці.

3.4 Allplan NOI API

Allplan NOI API — опис способів (набір класів, процедур, функцій, структур та констант), якими стороння програма може взаємодіяти з CAD-системою Allplan. Частиною Allplan NOI API є інтерфейс NOI (Nemetschek Open Interface), який пропонує бізнес-об’єкти для системи Allplan, такі як 2D-лінії, 3D-лінії, люки, пломби, сплайни і так далі. Він також надає деякий проміжний рівень доступу до бази даних і графічного інтерфейсу користувача.

Allplan NOI API дозволяє керувати різними аспектами візуалізації BIM моделі в Allplan, наприклад, активувати об’єкти, керувати їх видимістю, програмно заповнювати атрибути об’єктів тощо.

3.5 Python/C API

Бібліотеки Python/C API дають C та C++ програмістам доступ до інтерпретатора Python на різних рівнях. API однаково можна використовувати з C++, але для стислості його зазвичай називають Python/C API. Існують два принципово різних причини використання Python/C API. Перша причина полягає в

написанні модулів розширення для конкретних цілей; це модулі C, які розширюють інтерпретатор Python. Другою причиною є використання Python як компонента у більшому додатку, цей метод зазвичай називають вбудовуванням Python у додаток.

Більшість функцій Python/C API мають один або більше аргументів, а також повертається значення типу PyObject*. Цей тип є покажчиком на тип даних, що представляє довільний об’єкт Python. Оскільки всі типи об’єктів Python розглядаються однаково мовою Python у більшості ситуацій, то вони повинні бути представлені одним типом C. Майже всі об’єкти Python зберігаються у купі: неможливо оголосити змінну типу PyObject, тільки змінні покажчика типу PyObject* можуть бути оголошені.

Всі об’єкти Python мають тип і кількість посилань. Кількість посилань важлива, оскільки комп’ютери мають обмежений обсяг пам’яті. Лічильник посилань підраховує, скільки існує різних місць, які мають посилання на об’єкт. Таке місце може бути іншим об’єктом, або глобальною (або статичною) змінною C, або локальною змінною в деякій функції C. Коли лічильник посилань об’єкта стає нульовим, об’єкт звільняється. Якщо він містить посилання на інші об’єкти, їх кількість посилань зменшується.

Висновки до розділу 3

Третій розділ присвячено опису використаних при програмній реалізації технологій, середовищ розробки, бібліотек та API. Наведено функціональні можливості використаних засобів.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У програмній системі для реалізації градієнтного бустингу над деревами рішень використано бібліотеку LightGBM, для взаємодії розробленої системи з CAD-системою Allplan використано Allplan NOI API, Python/C API для виклику Python скриптів з коду на C++, використано мови C#, C++, Python.

4.1 Класифікатор на основі градієнтного бустингу

Як предмет бустингу обрано дерева рішень. Результатом навчання є послідовність дерев рішень. Виділено п’ять класів: стіна, колона, балка, перекриття, інше. Виділено три ознаки для класифікації об’єктів: висота, ширина, довжина.

Для створення навчальної вибірки було проаналізовано 10 BIM моделей будівель різного призначення із заздалегідь визначеними класами будівельних компонентів: 6 житлових багатоповерхівок, 3 офісні будівлі, 1 торгівельний центр.

Навчальна вибірка (таблиця 4.1.) формувалася з BIM моделей системи Allplan [24], у яких всі об’єкти будівлі були визначені специфікацією як стандартні будівельні конструкції.

Таблиця 4.1. Характеристики навчальної вибірки

Клас	Кількість прикладів	
	Навчальна вибірка	Тестова вибірка
Стіна	1413	353
Колона	562	140
Перекриття	182	45
Балка	517	129
Інше	2554	638

Коректного розпізнавання вдалося досягти в результаті моделювання з такими значеннями параметрів:

— кількість ітерацій бустингу — 5000;

— кількість ітерацій для раннього завершення — 100;

— максимальна кількість замикаючих вузлів кожного дерева — 31;

— метрика якості — логістична функція помилки.

Результатом навчання є модель, яка складається з 1448 дерев рішень, фрагмент одного з яких зображено на рисунку 4.1.

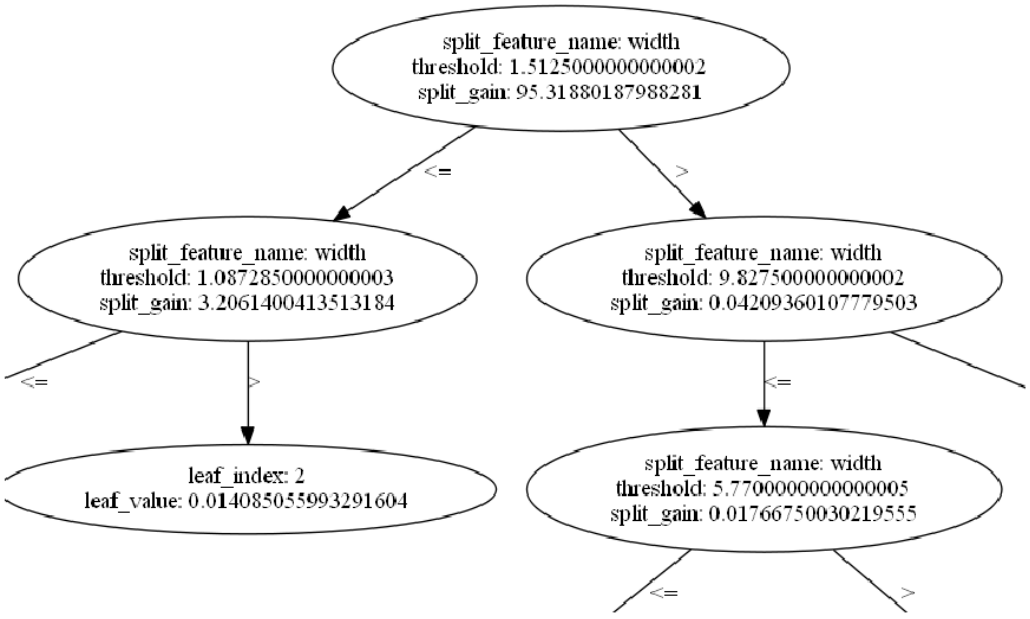


Рисунок 4.1 — Фрагмент одного з дерев рішень щодо оцінювання ширини 3D об’єкта

Навчання було зупинено при досягненні значення логістичної функції у 0,0177651 на 362 ітерації.

4.2 Опис архітектури програмної системи

Програмна система має працювати у двох режимах: навчання та безпосередньо розпізнавання компонентів BIM моделі з визначенням її

специфікації. Відповідно кожному призначенню реалізовано окремий плагін для системи Allplan.

Діаграму компонентів розробленої системи представлено на рисунку 4.2.

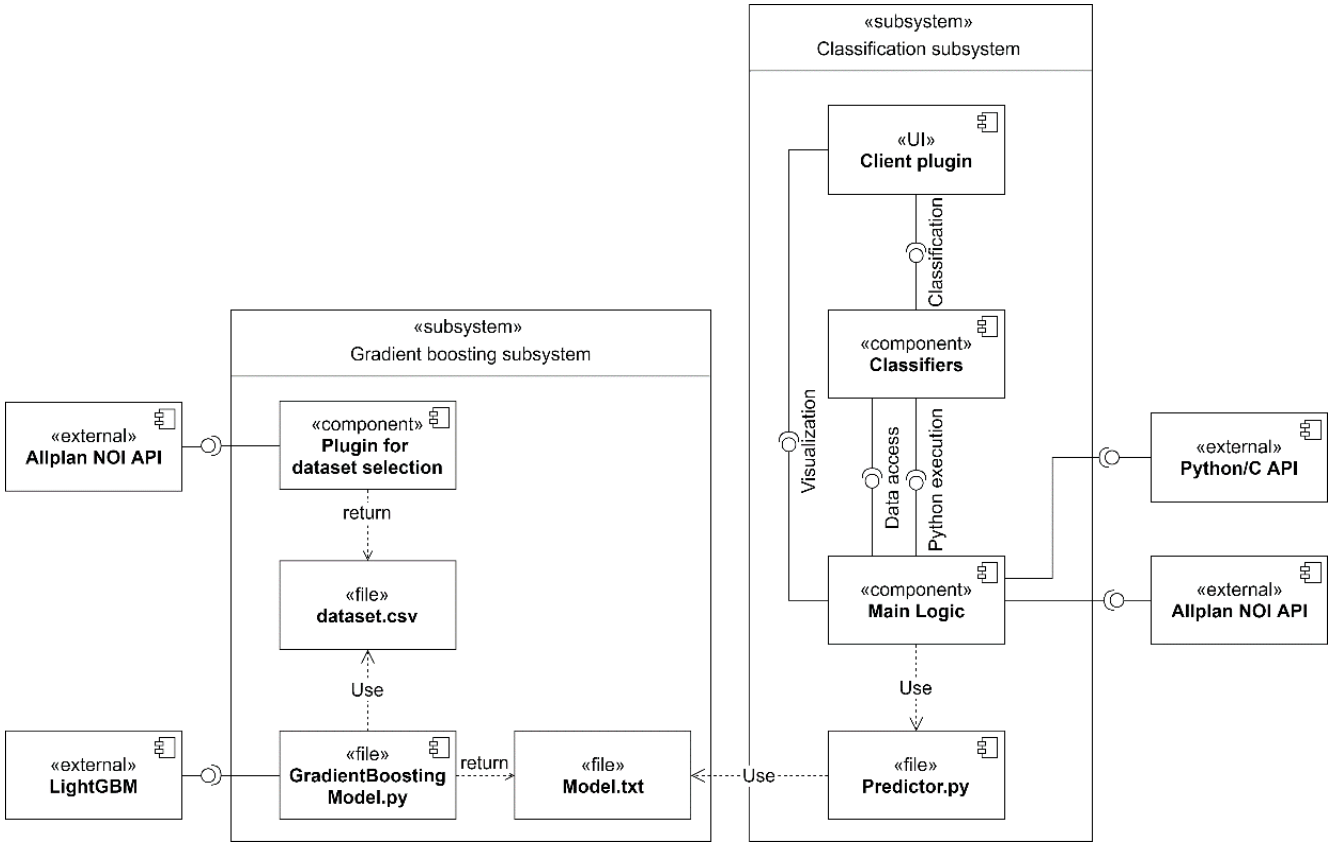


Рисунок 4.2 — Діаграма компонентів програмної системи

Перша підсистема (Gradient boosting subsystem) використовується для формування навчальних прикладів та побудови моделі на основі градієнтного бустингу. У склад цієї підсистеми входить плагін для формування файлу навчальних прикладів (Plugin for dataset selection). Цей плагін реалізовано мовою C++. За допомогою засобів Allplan NOI API він отримує всі будівельні компоненти активної в Allplan BIM моделі та формує файл-таблицю з габаритними характеристиками кожного об’єкту та його класу. Другою задачею цієї підсистеми є безпосередньо навчання. В роботі використовується алгоритм градієнтного бустингу з бібліотеки LightGBM для Python [17]. Результуюча модель записується у файл, який використовується другою підсистемою.

Друга підсистема (Classification subsystem) є оболонкою класифікатору, містить плагін для CAD-системи Allplan із графічним інтерфейсом користувача для

класифікації 3D об’єктів та формування для них специфікації. Графічний інтерфейс користувача (GUI) створено за допомогою технології WPF із застосуванням MVVM патерну. Допоміжна бібліотека з головною логікою системи (Main Logic) реалізована мовою C++ з використанням Allplan NOI API для реалізації взаємодії з Allplan. Вона дозволяє отримувати 3D об’єкти поточної моделі Allplan, візуально активувати об’єкти та керувати їх видимістю. Взаємодія оболонки зі скриптами на Python (Predictor.py) реалізована за допомогою Python/C API [23]. Бібліотека Classifiers другої підсистеми надає зручний інтерфейс (Classification) для використання з боку плагіна.

4.3 Апробація програмної системи

Результати апробації для трьох BIM моделей зведені у таблицю 4.2.

Таблиця 4.2. Апробація системи

№	Назва будівлі	Кількість будівельних конструкцій / Частка коректного розпізнавання, %			
		Стіни	Колони	Балки	Перекриття
1	Промислова будівля	15 / 100	51 / 96	35 / 97	4 / 100
2	Житлова двоповерхова будівля	16 / 100	- / -	2 / 100	15 / 100
3	Житлова триповерхова будівля	40 / 98	10 / 100	34 / 97	24 / 96

Всього в апробації було задіяно 50 BIM моделей, для яких визначено частку коректного розпізнавання компонентів 98,5%.

Висновки до розділу 4

У четвертому розділі описано деталі реалізації розробленої програмної системи класифікації будівельних компонентів BIM моделі. Наведено архітектуру запропонованої системи, параметри моделі градієнтного бустингу. Програмну систему апробовано на 50 BIM моделях, доведено її ефективність.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Для роботи розробленої програмної системи персональний комп’ютер користувача повинен відповідати системним вимогам. Користувачу необхідно встановити програмну систему за допомогою інсталятору.

5.1 Системні вимоги до програмної системи

До апаратного та програмного забезпечення користувача системи висуваються наступні мінімальні вимоги:

— процесор Intel Core i3 з тактовою частотою не менше 2.0 GHz;

— 2 GB оперативної пам’яті;

— 50 MB вільного місця;

— відеокарта з 1 GB RAM;

— операційна система Windows 10;

— встановлений Allplan 2019.

5.2 Інсталяція програмної системи

Розроблена програмна система може бути встановлена на персональний комп’ютер користувача за допомогою програми інсталятора. Інсталятор розроблено засобами InnoSetup [25].

Інсталятор автоматично скопіює всі необхідні файли для роботи системи у відповідні місця.

Розроблений інсталятор має локалізацію на трьох мовах: англійська, німецька та українська. Вибір мови інсталятору зображено на рисунку 5.1.

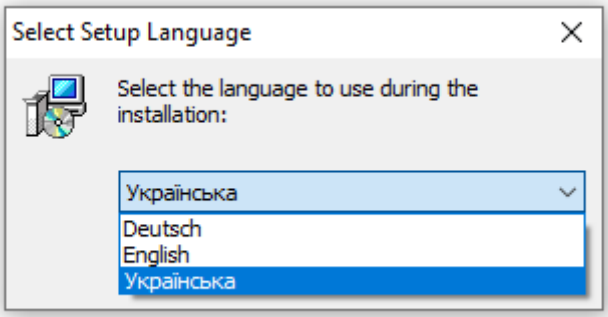


Рисунок 5.1 — Вибір мови інсталяції користувачем

Після вибору мови достатньо натиснути кнопки “Далі” та дочекатися завершення інсталяції. Автоматично буде встановлено всі необхідні файли та завантажено необхідні бібліотеки Python.

5.3 Робота користувача з системою

Розроблена програмна система є плагіном для CAD-системи Allplan. Для роботи із системою необхідно запустити Allplan, відкрити BIM модель, натиснути кнопку меню Plug-in та обрати плагін “Класифікатор 3D тіл” (рисунок 5.2).

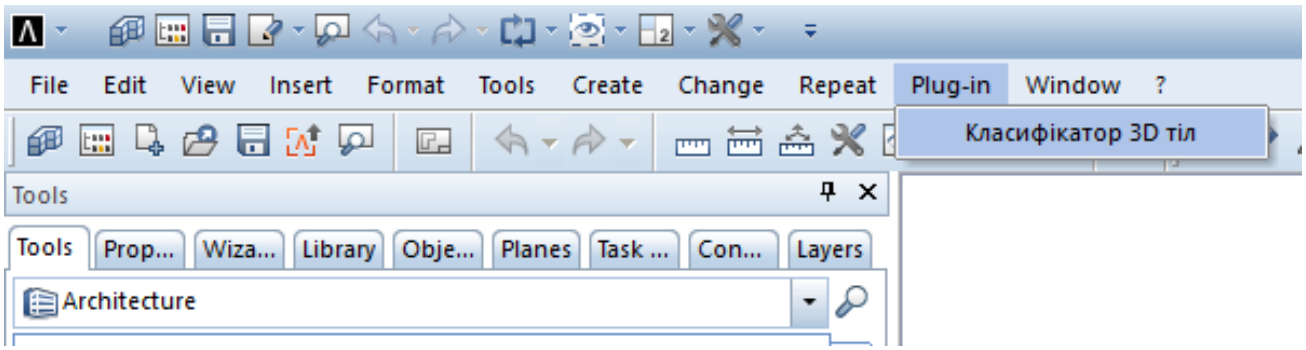


Рисунок 5.2 — Розроблена програмна система у меню Allplan

Після цього відкриється вікно розробленої програмної системи. На рисунку 5.3 зображено графічний інтерфейс користувача плагіну.

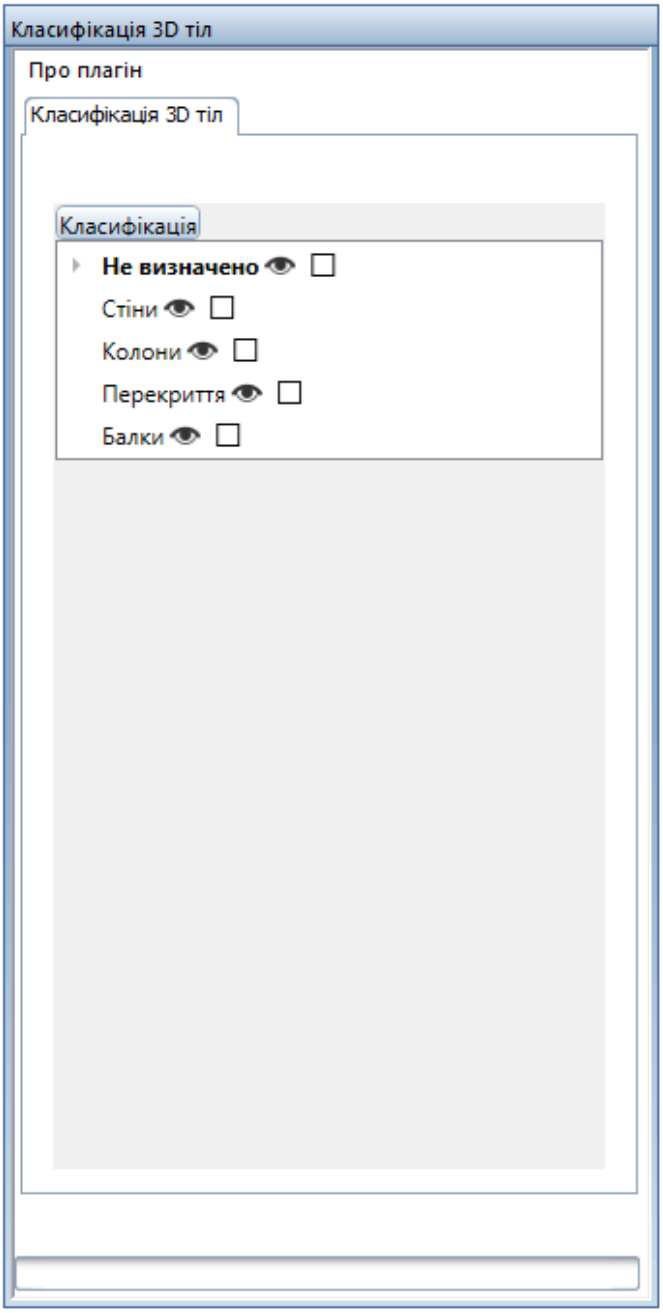


Рисунок 5.3 — Графічний інтерфейс користувача

Всі 3D тіла відкритої BIM моделі спочатку потрапляють у групу “Не визначено”. Для автоматичної класифікації 3D тіл необхідно натиснути кнопку “Класифікація”. Після цього об’єкти потраплять у різні групи дерева (рисунок 5.4).

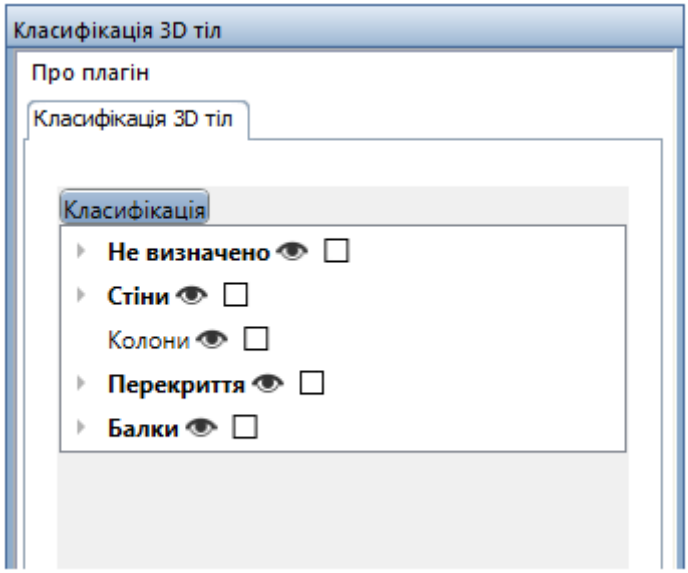


Рисунок 5.4 — Вигляд дерево після класифікації

Програмна система має зручний функціонал для керування візуалізацією 3D тіл в Allplan. Користувач може робити об’єкти видимими/невидимими, активувати та деактивувати їх в Allplan за допомогою відповідних кнопок інтерфейсу (рисунок 5.5).

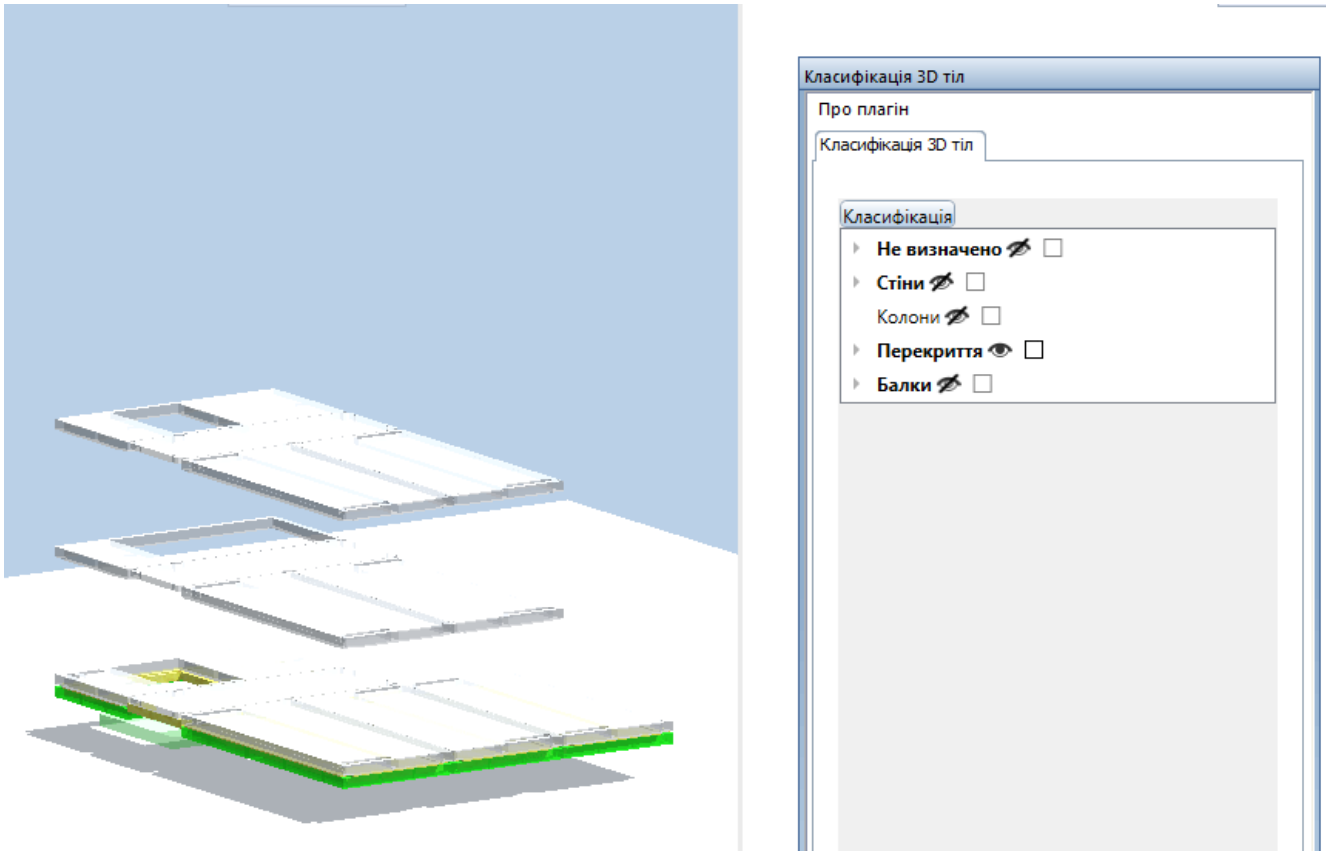


Рисунок 5.5 — Використання функції приховання об’єктів

Список об’єктів певної групи можна переглянути натиснувши на кнопку розкриття списку групи (рисунок 5.6).

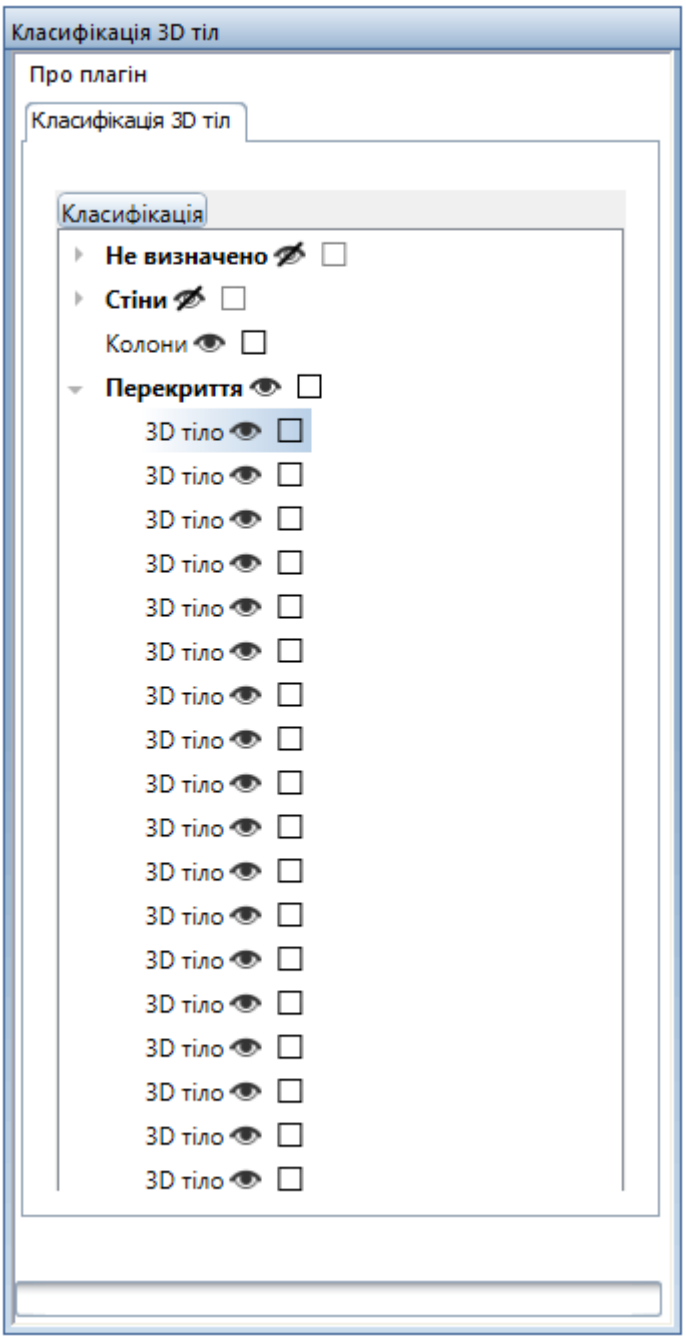


Рисунок 5.6 — Розгорнутий список перекриттів

Таким чином, розроблена система на основі градієнтного бустингу над деревами рішень дозволяє користувачу проводити класифікацію 3D тіл BIM моделі з ціллю створення специфікації цих об’єктів як стандартних будівельних компонентів.

На рисунку 5.7 зображено заповнений програмною системою атрибут “Building class” для тривимірного тіла, яке представляє собою перекриття, значенням “slab”.

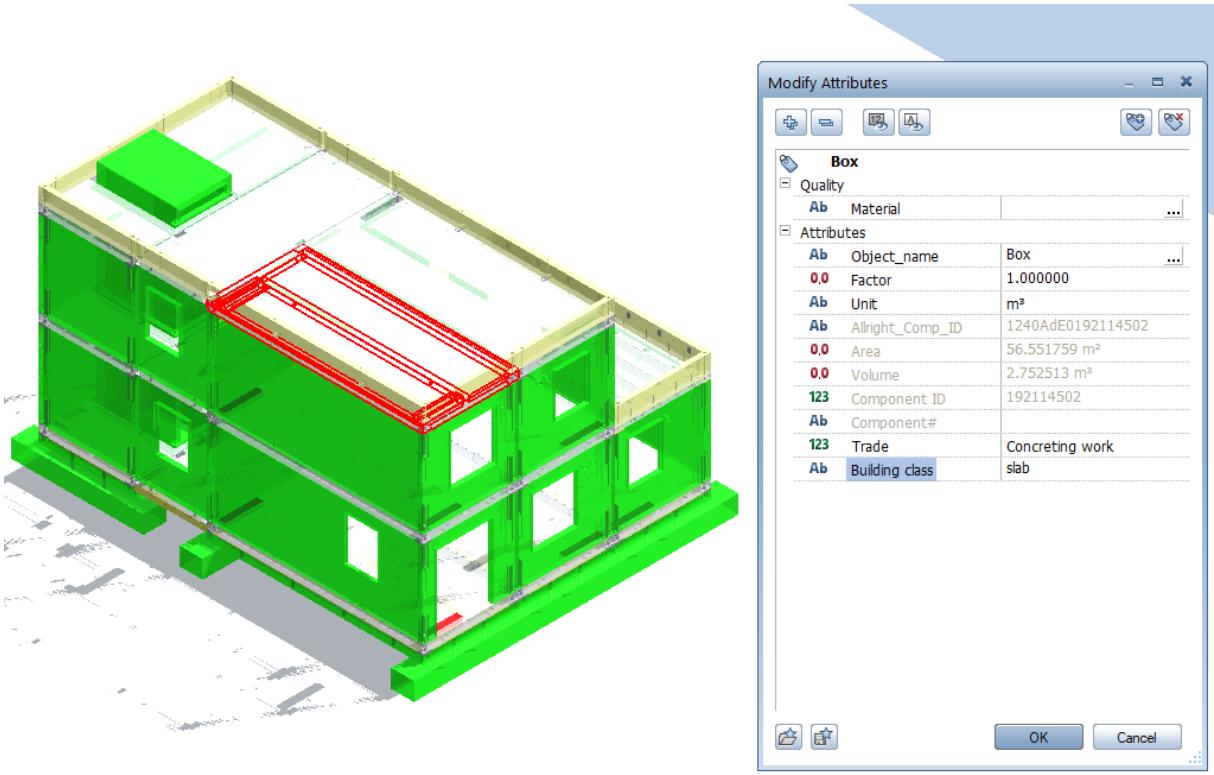


Рисунок 5.7 — Результат роботи системи

Результатом роботи програмної системи є специфікація будівельних компонентів у вигляді заповнених атрибутів “Building class” для кожного тривимірного об’єкту BIM моделі.

Висновки до розділу 5

У п’ятому розділі наведено системні вимоги до розробленої програмної системи. Описано процес інсталяції системи та сценарій роботи користувача. Наведено результат роботи розробленої програмної системи.

ВИСНОВКИ

1. На основі проведеного аналізу існуючих підходів і методів класифікації тривимірних об’єктів обґрунтовано використання градієнтного бустингу для класифікації основних тривимірних будівельних елементів BIM моделі.

2. Як предмет бустингу обрано дерева рішень. Результатом навчання є послідовність дерев рішень. Виділено п’ять класів: стіна, колона, балка, перекриття, інше. Виділено три ознаки для класифікації об’єктів: висота, ширина, довжина. Габаритні характеристики визначаються за орієнтованим обмежуючим прямокутником.

3. Проведено аналіз програмних засобів реалізації поставленої задачі. Використано бібліотеку LightGBM для реалізації градієнтного бустингу над деревами рішень, Allplan NOI API для взаємодії розробленої системи з CAD-системою Allplan, Python/C API для виклику Python скриптів з коду на C++, використано мови C#, C++, Python.

4. Підготовлено навчальну вибірку за десятьма BIM моделями різного призначення: 6 житлових багатоповерхівок, 3 офісні будівлі, 1 торгівельний центр. В роботі наведено значення параметрів навчання, з якими вдалося досягти коректного розпізнавання: кількість ітерації бустингу, кількість ітерацій для раннього завершення, максимальна кількість замикаючих вузлів кожного дерева, метрика якості моделі.

5. Запропоновано програмне рішення автоматичного формування специфікації тривимірних геометричних об’єктів за BIM моделлю у вигляді двох плагінів для CAD-системи Allplan. Плагіни призначені відповідно: для навчання та безпосередньо формування специфікації будівельних компонентів BIM моделі.

6. Програмна система апробована для 50 BIM моделей. Результати випробовування становлять 98,5% коректного розпізнавання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bassier M., Vergauwen M., Van Genechten B. Automated classification of heritage buildings for as-built BIM using machine learning techniques // ISPRS Annals of the PRSSIS. — 2017. — IV-2/W2. P. 25—30.

2. Bassier M., Vergauwen M., Van Genechten B. Automated Semantic Labelling of 3D Vector Models for Scan-to-BIM // Proceedings of the 4th Annual International Conference on Architecture and Civil Engineering. — 2016. — P. 93—100.

3. Babacan K., Chen L., Sohn G. Semantic Segmentation of Indoor Point Clouds Using Convolutional Neural Network // ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, 4(4W4). — 2017. — P. 101—108.

4. Maturana D., Scherer S. Voxnet: A 3d convolutional neural network for real-time object recognition // IEEE/RSJ International Conference on Intelligent Robots and Systems. — 2015.

5. Breinman L. Arcing the edge // University of California, Berkeley. (Technical Report 486). — 1997.

6. The National BIM Standard-United States [Електронний ресурс]. — Режим доступу: https://www.nationalbimstandard.org .

7. IFC Overview summary [Електронний ресурс]. — Режим доступу: http://www.buildingsmart-tech.org/specifications/ifc-overview .

8. ISO 10303-514 Advanced boundary representation [Електронний ресурс]. — Режим доступу: https://www.sis.se/api/document/preview/616081/ .

9. Greene N. Voxel space automata: modeling with stochastic growth processes in voxel space // ACM SIGGRAPH Computer Graphics. — 1989. — P.175—184.

10. Breiman, L. Bagging predictors / L. Breiman // Machine Learning. — 1996. — V. 26, №2. — P. 123—140.

11. Freund Y., Schapire R. Experiments with a New Boosting Algorithm. Machine Learning // Proceedings of the Thirteenth International Conference. — 1996. — P. 148—156.

12. Breiman L. Random forests // Machine learning. Volume 45(1). — 2001. — P. 5—32

13. Rashmi K. V., Gilad-Bachrach R. DART: Dropouts meet Multiple Additive Regression Trees // Artificial Intelligence and Statistic. — 2015. — P. 489—497.

14. Ke G., Meng Q., Finley T., Wang T., Chen W., Ma W., Ye Q., Liu T.Y. Lightgbm: a highly efficient gradient boosting decision tree // Advances in Neural Information Processing Systems. — 2017. — P. 3149—3157.

15. Mason L., Baxter J., Bartlett P., Frean M. Boosting Algorithms as Gradient Descent // Neural Information Processing Systems, vol. 12. — 2000. — P. 512—518 .

16. O'Rourke J. Finding minimal enclosing boxes // International Journal of Computer and Information Sciences. — 1985. — P. 183—199.

17. Microsoft/LightGBM [Електронний ресурс]. — Режим доступу: https://github.com/Microsoft/LightGBM .

18. XGBoosting eXtreme Gradient Boosting [Електронний ресурс]. — Режим доступу: https://github.com/dmlc/xgboost .

19. scikit-learn Machine Learning in Python [Електронний ресурс]. — Режим доступу: https://scikit-learn.org/ .

20. Visual Studio 2017 [Електронний ресурс]. — Режим доступу: https://visualstudio.microsoft.com/ru/vs/older-downloads/ .

21. JetBrains PyCharm Community 2019 [Електронний ресурс]. — Режим доступу: https://www.jetbrains.com/ pycharm/download/#section=windows .

22. Windows Presentation Foundation [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/en-us/dotnet/framework/wpf/ .

23. Python/C API Reference Manual [Електронний ресурс]. Режим доступу: https://docs.python.org/3.6/c-api/ .

24. Allplan. BIM solutions for the AEC [Електронний ресурс]. — Режим доступу: https://www.allplan.com/en/ .

25. InnoSetup [Електронний ресурс]. — Режим доступу: http://www.jrsoftware.org/isinfo.php .

ДОДАТОК А

Класифікація основних тривимірних будівельних елементів BIM моделі

Специфікація

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5297_19Б

Аркушів 2

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТР5297_19Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТР5297_19Б 12-1	ClientPlugin.dll	Плагін з графічним інтерфейсом користувача
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТР5297_19Б 12-2	Classifiers.dll	Бібліотека класифікатор
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТР5297_19Б 12-3	MainLogic.dll	Бібліотека для керування базою даних Allplan та візуалізацією
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТР5297_19Б 12-4	DatasetSelector.dll	Плагін для генерування навчальних прикладів

ДОДАТОК Б

Класифікація основних тривимірних будівельних елементів BIM моделі

Текст програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5297_19Б

Аркушів 7

Київ 2019

Текст бібліотеки MainLogic.dll (взаємодія з інтерпретатором Python)

```
#include "stdafx.h"
#include "PythonHelper.h"
using namespace AllplanLogic;
using namespace System::Runtime::InteropServices;

PythonHelper::PythonHelper()
{
	PyObject *pModule, *pDict, *pClass, *pValue;

	Py_Initialize();

	AddPathToSyspath("D:\\");

	pModule = PyImport_ImportModule("lightgbm_predictor");
	if (pModule != NULL)
	{
		pDict = PyModule_GetDict(pModule);

		pClass = PyDict_GetItemString(pDict, "LightGBMPredictor");

		if (PyCallable_Check(pClass))
		{
			pLightGBMPredictor = PyObject_CallObject(pClass, NULL);
		}
	}
}

System::String^ PythonHelper::Predict(double height, double width, double length)
{
	std::string result;
	if (pLightGBMPredictor == NULL)
		throw std::logic_error("Python object LightGBMPredictor is not initialized.");


	PyObject* pValue = PyObject_CallMethod(pLightGBMPredictor, "predict", "(ddd)", height, width, length);
	if (pValue != NULL)
	{
		result = GetStringValue(pValue);
		Py_DECREF(pValue);
	}
	else
		throw std::logic_error("Error occured while calling predict method.");


	return gcnew System::String(result.c_str());;
}

IEnumerable<KeyValuePair<System::String^, System::String^>>^ PythonHelper::Predict(IEnumerable<AllplanLogic::ObjectFeatures^>^ objectsFeatures)
{
	Dictionary<System::String^, System::String^>^ result = gcnew Dictionary<System::String^, System::String^>(0);
	if (pLightGBMPredictor == NULL)
		throw std::logic_error("Python object LightGBMPredictor is not initialized.");


	PyListObject *pRows = (PyListObject*)PyList_New(0);
	for each(auto features in objectsFeatures)
	{
		PyListObject *pFeatures = (PyListObject*)PyList_New(0);
		PyList_Append((PyObject*)pFeatures, PyUnicode_FromString(SysStringToChar(features->UUID)));
		PyList_Append((PyObject*)pFeatures, PyFloat_FromDouble(features->Height));
		PyList_Append((PyObject*)pFeatures, PyFloat_FromDouble(features->Width));
		PyList_Append((PyObject*)pFeatures, PyFloat_FromDouble(features->Length));

		PyList_Append((PyObject*)pRows, (PyObject*)pFeatures);
	}

	PyObject* pList = PyObject_CallMethodObjArgs(pLightGBMPredictor, PyUnicode_FromString("predict"), (PyObject*)pRows, NULL);
	if (pList != NULL)
	{
		CheckIsList(pList);
		for (size_t i = 0; i < PyList_Size(pList); i++)
		{
			PyObject* pItemList = PyList_GetItem(pList, i);
			CheckIsList(pItemList);
			System::String^ uuid = gcnew System::String(GetStringValue(PyList_GetItem(pItemList, 0)));
			System::String^ predictedClass = gcnew System::String(GetStringValue(PyList_GetItem(pItemList, 1)));
```

```
                        result->Add(uuid, predictedClass);
                        Py_DECREF(pItemList);
                }
                Py_DECREF(pList);
        }
        else
                throw std::logic_error("Error occured while calling predict method.");


        return result;
}

char* PythonHelper::GetStringValue(PyObject* pValue)
{
        char* value;
        if (PyUnicode_Check(pValue)) {
                PyObject * temp_bytes = PyUnicode_AsEncodedString(pValue, "UTF-8", "strict");
                if (temp_bytes != NULL) {
                        value = PyBytes_AS_STRING(temp_bytes);
                        value = _strdup(value);
                        Py_DECREF(temp_bytes);
                }
                else {
                        throw std::invalid_argument("Argument is not a string.");
                }
        }
        else
                throw std::invalid_argument("Argument is not a string.");
        return value;
}

void PythonHelper::CheckIsList(PyObject* pList)
{
        if (!PyList_Check(pList))
                throw std::invalid_argument("Argument is not a list.");
}

void PythonHelper::AddPathToSyspath(char* path)
{
        PyObject* pName = PyUnicode_FromString(path);
        PyObject* syspath = PySys_GetObject("path");
        if (PyList_Insert(syspath, 0, pName))
                throw std::logic_error("Error inserting extra path into sys.path list\n");
        if (PySys_SetObject("path", syspath))
                throw std::logic_error("Error setting sys.path object\n");
        Py_DECREF(pName);
}

char* PythonHelper::SysStringToChar(System::String^ string)
{
        return (char*)(void*)Marshal::StringToHGlobalAnsi(string);
}
```

Текст бібліотеки MainLogic.dll (візуалізація в Allplan)

```
#include "stdafx.h"
#include "DatabaseHelper.h"
#include "IVisualizationService.h"
#include <vcclr.h>
#include <memory>
#include <msclr\marshal.h>
#include <msclr\marshal_cppstd.h>
#include <NemAll_NOI_Pool/noi_DatabaseQuery.h>
#include <NemAll_NOI_Basis_Objects/noi_BasisPolyhedron3D.h>
#include <NemAll_NOI_Basis_Objects/noi_BasisBRep3D.h>
#include <NemAll_NOI_Pool\noi_DatabaseQuery.h>
#include <NemAll_NOI_Pool\noi_Transaction.h>
#include <NemAll_NOI_Pool\NOI_Database.h>
#include <NemAll_NOI_Basis_Wrapper/NOI_BaseWrapperExceptionDefines.h>
#include <NemAll_NOI_Common_Objects/noi_DBObject.h>

using namespace System;
using namespace System::Collections::Generic;
using namespace System::Runtime::InteropServices;
```

```
namespace AllplanLogic
{
	public ref class VisualizationHelper : IVisualizationService
	{
	public:
		VisualizationHelper(DatabaseHelper^ databaseService)
		{
			_databaseService = databaseService;
		}

		void ZoomToObjects(System::String^ uuid) override
		{
			List<System::String^>^ uuids = gcnew List<System::String^>(0);
			uuids->Add(uuid);
			ZoomToObjects(uuids);
		}

		void ZoomToObjects(IEnumerable<System::String^>^ uuids) override
		{
			CNOI_DBObjectPtrVector objects = _databaseService->GetObjectsFromUUIDs(uuids);
			ZoomToObjects(objects);
		}

		void ActivateObjects(IEnumerable<System::String^>^ uuids) override
		{
			CNOI_DBObjectPtrVector objects = _databaseService->GetObjectsFromUUIDs(uuids);
			ActivateObjects(objects);
		}

		void ActivateObject(System::String^ uuid) override
		{
			List<System::String^>^ uuids = gcnew List<System::String^>(0);
			uuids->Add(uuid);
			ActivateObjects(uuids);
		}

		void PassivateObjects(IEnumerable<System::String^>^ uuids) override
		{
			CNOI_DBObjectPtrVector objects = _databaseService->GetObjectsFromUUIDs(uuids);
			PassivateObjects(objects);
		}

		void PassivateObject(System::String^ uuid) override
		{
			List<System::String^>^ uuids = gcnew List<System::String^>(0);
			uuids->Add(uuid);
			PassivateObjects(uuids);
		}

		void DrawObjectVisible(System::String^ uuid) override
		{
			List<System::String^>^ uuids = gcnew List<System::String^>(0);
			uuids->Add(uuid);
			DrawObjectsVisible(uuids);
		}

		void DrawObjectsVisible(IEnumerable<System::String^>^ uuids) override
		{
			CNOI_DBObjectPtrVector objects = _databaseService->GetObjectsFromUUIDs(uuids);
			DrawObjectsVisible(objects);
		}

		void DrawObjectInvisible(System::String^ uuid) override
		{
			List<System::String^>^ uuids = gcnew List<System::String^>(0);
			uuids->Add(uuid);
			DrawObjectsInvisible(uuids);
		}

		void DrawObjectsInvisible(IEnumerable<System::String^>^ uuids) override
		{
			CNOI_DBObjectPtrVector objects = _databaseService->GetObjectsFromUUIDs(uuids);
			DrawObjectsInvisible(objects);
		}

	private:
		DatabaseHelper^ _databaseService;

		void ZoomToObjects(CNOI_DBObjectPtrVector objects)
```

```
{
	CNOI_GeoBoundingBox3DPtr maxBBox = GetMaxBoundingBox(objects);
	if (maxBBox->IsValid())
	{
		NOI_TRY_WITH_DLG()
		{
			NOI_TRY_DB_START();
			CNOI_Service_Drawing::MDIChangeActiveViewClipping(pDB,
				maxBBox->Min()->X(), maxBBox->Min()->Y(), maxBBox->Min()->Z(),
				maxBBox->Max()->X(), maxBBox->Max()->Y(), maxBBox->Max()->Z());
			CNOI_Service_Drawing::UpdateGraphicsEngine(pDB);
			NOI_CATCH_DB_END();
		}
		NOI_CATCH_WITH_DLG(::AfxGetAppName(), _T("An error occured during drawing objects"));
	}
}

CNOI_GeoBoundingBox3DPtr GetMaxBoundingBox(CNOI_DBObjectPtrVector objects)
{
	CNOI_GeoBoundingBox3DPtr maxBBox = CNOI_GeoBoundingBox3D::CreateObj();
	for (size_t i = 0; i < objects.size(); ++i)
	{
		if (objects[i].IsNULL())
			continue;
		CNOI_GeoBoundingBox3DPtr bBox = objects[i]->GetBoundingBox();
		if (bBox->Min()->X() < maxBBox->Min()->X())
			maxBBox->Min()->X(bBox->Min()->X());
		if (bBox->Min()->Y() < maxBBox->Min()->Y())
			maxBBox->Min()->Y(bBox->Min()->Y());
		if (bBox->Min()->Z() < maxBBox->Min()->Z())
			maxBBox->Min()->Z(bBox->Min()->Z());

		if (bBox->Max()->X() > maxBBox->Max()->X())
			maxBBox->Max()->X(bBox->Max()->X());
		if (bBox->Max()->Y() > maxBBox->Max()->Y())
			maxBBox->Max()->Y(bBox->Max()->Y());
		if (bBox->Max()->Z() > maxBBox->Max()->Z())
			maxBBox->Max()->Z(bBox->Max()->Z());
	}
	return maxBBox;
}

void ActivateObject(CNOI_DBObjectPtr object)
{
	CNOI_DBObjectPtrVector objects;
	objects.push_back(object);
	ActivateObjects(objects);
}

void PassivateObject(CNOI_DBObjectPtr object)
{
	CNOI_DBObjectPtrVector objects;
	objects.push_back(object);
	PassivateObjects(objects);
}

void PassivateObjects(CNOI_DBObjectPtrVector objects)
{
	NOI_TRY_WITH_DLG()
	{
		NOI_TRY_DB_START();
		CNOI_Service_ObjectActivation::PassivateObjects(pDB, objects, true);
		NOI_CATCH_DB_END();
	}
	NOI_CATCH_WITH_DLG(::AfxGetAppName(), _T("An error occured during drawing objects"));
}

void ActivateObjects(CNOI_DBObjectPtrVector objects)
{
	NOI_TRY_WITH_DLG()
	{
		NOI_TRY_DB_START();
		CNOI_Service_ObjectActivation::ActivateObjects(pDB, objects, true);
		CNOI_Service_ObjectActivation::VisibleActivationOfActiveObjects(pDB);
		NOI_CATCH_DB_END();
	}
	NOI_CATCH_WITH_DLG(::AfxGetAppName(), _T("An error occured during drawing objects"));
}
```

```
		void DrawObjectsVisible(CNOI_DBObjectPtrVector objects)
		{
			NOI_TRY_WITH_DLG()
			{
				NOI_TRY_DB_START();
				CNOI_Service_Drawing::MDIDrawObjects(pDB, objects);
				NOI_CATCH_DB_END();
			}
			NOI_CATCH_WITH_DLG(::AfxGetAppName(), _T("An error occured during drawing objects"));
		}

		void DrawObjectsInvisible(CNOI_DBObjectPtrVector objects)
		{
			NOI_TRY_WITH_DLG()
			{
				NOI_TRY_DB_START();
				CNOI_Service_Drawing::MDIDrawObjectsInvisible(pDB, objects);
				NOI_CATCH_DB_END();
			}
			NOI_CATCH_WITH_DLG(::AfxGetAppName(), _T("An error occured during drawing objects"));
		}
	};
}
```

Текст бібліотеки ClientPlugin.dll (графічний інтерфейс користувача)

```
using ClassificationObjects.Models;
using Classifiers;
using System;
using System.ComponentModel;
using System.Windows.Input;

namespace ClassificationObjects.ViewModels
{
   public class ItemViewModel : IItemViewModel
   {
      public event PropertyChangedEventHandler PropertyChanged;
      protected virtual void OnPropertyChanged(string propertyName)
      {
         PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
      }

      public string Name
      {
         get => ObjectEntity.Name;
         set { throw new Exception(); }
      }

      public IObjectEntity ObjectEntity { get; set; }

      private IGroupViewModel _parent;
      public IGroupViewModel Parent
      {
         get => _parent;
         set
         {
            if (_parent == value)
               return;
            _parent = value;
            OnPropertyChanged("Parent");
         }
      }

      private bool? _isActive;
      public bool? IsActive
      {
         get => _isActive;
         set
         {
            _isActive = value;
            OnPropertyChanged("IsActive");
         }
      }

      private bool? _isVisible;
      public bool? IsVisible
      {
         get => _isVisible;
```

```
            set
            {
                _isVisible = value;
                OnPropertyChanged("IsVisible");
            }
        }

        private IVisualizationService _visualizationService;
        private IAttributeService _attributeService;

        public ItemViewModel(IVisualizationService visualizationService,
                             IAttributeService attributeService)
        {
            _visualizationService = visualizationService;
            _attributeService = attributeService;
            _isActive = false;
            _isVisible = true;
        }

        ICommand _applyActivationAndUpdateRelatedObjects;
        public ICommand ApplyActivationAndUpdateRelatedObjects
        {
            get
            {
                if (_applyActivationAndUpdateRelatedObjects == null)
                {
                    _applyActivationAndUpdateRelatedObjects = new WpfGuiHelpers.RelayCommand<INodeViewModel>(
                        x =>
                        {
                            ApplyActivation();
                            Parent?.UpdateIsActive();
                        });
                }
                return _applyActivationAndUpdateRelatedObjects;
            }
        }

        private void ApplyActivation()
        {
            if (_isActive.Value)
                _visualizationService.ActivateObject(ObjectEntity.UUID);
            else
                _visualizationService.PassivateObject(ObjectEntity.UUID);
        }

        ICommand _applyVisibilityAndUpdateRelatedObjects;
        public ICommand ApplyVisibilityAndUpdateRelatedObjects
        {
            get
            {
                if (_applyVisibilityAndUpdateRelatedObjects == null)
                {
                    _applyVisibilityAndUpdateRelatedObjects = new WpfGuiHelpers.RelayCommand<INodeViewModel>(
                        x =>
                        {
                            ApplyVisibility();
                            Parent?.UpdateIsVisible();
                        });
                }
                return _applyVisibilityAndUpdateRelatedObjects;
            }
        }

        private void ApplyVisibility()
        {
            if (_isVisible.Value)
                _visualizationService.DrawObjectVisible(ObjectEntity.UUID);
            else
                _visualizationService.DrawObjectInvisible(ObjectEntity.UUID);
        }

        public void WriteBuildingClassAttribute(ObjectClass value)
        {
            _attributeService.WriteBuildingClassAttribute(ObjectEntity.UUID, value.ToString());
        }
    }
}
```

ДОДАТОК В

Класифікація основних тривимірних будівельних елементів BIM моделі

Опис програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5297_19Б

Аркушів 9

Київ 2019

АНОТАЦІЯ

Розроблена система складається з двох підсистем. Перша підсистема використовується для формування навчальних прикладів та побудови моделі на основі градієнтного бустингу. У склад цієї підсистеми входить плагін для формування файлу навчальних прикладів. Цей плагін реалізовано мовою C++. За допомогою засобів Allplan NOI API він отримує всі будівельні компоненти активної в Allplan BIM моделі та формує файл-таблицю з габаритними характеристиками кожного об’єкту та його класу. Другою задачею цієї підсистеми є безпосередньо навчання. В роботі використовується алгоритм градієнтного бустингу з бібліотеки LightGBM для Python. Результуюча модель записується у файл, який використовується другою підсистемою.

Друга підсистема є оболонкою класифікатору, містить плагін для CAD-системи Allplan із графічним інтерфейсом користувача для класифікації 3D об’єктів та формування для них специфікації. Графічний інтерфейс користувача створено за допомогою технології WPF із застосуванням MVVM патерну. Допоміжна бібліотека з головною логікою системи реалізована мовою C++ з використанням Allplan NOI API для реалізації взаємодії з Allplan.

Програмна система була розроблена за допомогою програмних середовищ розробки Visual Studio Community 2017 та JetBrains PyCharm Community 2019. Графічний інтерфейс користувача створено за допомогою технології WPF мовою програмування C#. Взаємодія розробленої системи з CAD-системою Allplan реалізовано з використанням Allplan NOI API.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Розроблена програмна система працює на базі операційної системи Windows 7, Windows 8 або Windows 10. Для функціонування програмної системи необхідно щоб було встановлено Allplan 2019.

Програмна система розроблена з використанням мов програмування C#, C++ та Python та середовищ розробки Microsoft Visual Studio 2017 та JetBrains PyCharm Community 2019.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблена програмна система виконує задачі генерування навчальної вибірки, навчання та безпосередньо класифікації будівельних компонентів BIM моделі.

Програмна система призначена для формування специфікації основних будівельних компонентів BIM моделі у CAD-системі Allplan.

Функціональні обмеження на використання компонентів полягають у необхідності використання CAD-системи Allplan.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Програмна система отримує із бази даних Allplan список всіх тривимірних тіл, потім проводиться класифікація, результат записується у вигляді специфікації будівельних компонентів.

Як метод класифікації використовується градієнтний бустинг над деревами рішень.

Програмна система зв’язана з CAD-системою Allplan за допомогою використання бібліотек Allplan NOI API.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Розроблена програмна система працює на базі операційної системи Windows 7, Windows 8 або Windows 10. Для функціонування програмної системи необхідно щоб було встановлено Allplan 2019.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблена програмна система може бути встановлена на персональний комп’ютер користувача за допомогою програми інсталятора. Інсталятор розроблено засобами InnoSetup.

Інсталятор автоматично скопіює всі необхідні файли для роботи системи у відповідні місця. Розроблений інсталятор має локалізацію на трьох мовах: англійська, німецька та українська.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для розробленої системи є активна BIM модель у CAD-системі Allplan, яка містить тривимірні тіла без визначеної специфікації як будівельних елементів.

Вихідними даними є специфікація тривимірних елементів у вигляді заповнених атрибутів “Building class” відповідних значенням класу будівельного елементу.