ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для інтерактивної генерації аудіо-візуального контенту»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр ПОГОРІЛИЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41
Олександр ПОГОРІЛИЙ

Керівник: Людмила СОЛЯНИК
д.х.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Погорілому Олександру Анатолійовичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для інтерактивної генерації аудіо-візуального контенту»
керівник кваліфікаційної роботи д.х.н., доцент Людмила СОЛЯНИК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про принципи генеративного аудіо-візуального контенту, методи взаємодії людини з цифровими системами, особливості мультимедійних інсталяцій, документація середовища TouchDesigner та довідкові матеріали з Python.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1.Огляд застосунку для вивчення японської мови та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
2.Огляд та аналіз засобів реалізації програмного забезпечення для інтерактивної генерації аудіо-візуального контенту.
3.Проєктування архітектури програмного забезпечення для інтерактивної генерації аудіо-візуального контенту.
4.Програмна реалізація та тестування програмного забезпечення для інтерактивної генерації аудіо-візуального контенту.

5. Перелік графічного матеріалу: *презентація*

1. Мета об’єкт та предмет дослідження.
2. Задачі кваліфікаційної роботи.
3. Вимоги до програмного забезпечення.
4. Програмні засоби реалізації.
5. Діаграма алгоритму роботи.
6. Діаграма аналізу аудіо та відео жестів.
7. Діаграма генерації частинок.
8. Діаграма взаємозв’язку параметрів генерації.
9. Екранні форми.
10. Демонстрація роботи застосунку.
11. Апробація результатів дослідження
12. Висновки

6. Дата видачі завдання«25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд існуючих алгоритмів обробки тексту та методів і технологій побудови тематичного словника	15.03.2025-20.03.2025	
4	Проектування застосунку для автоматизованої побудови тематичного словника навчальної дисципліни	22.03.2025-01.04.2025	
5	Програмна реалізація застосунку	02.04.2025-25.04.2025	
6	Тестування застосунку	25.04.2025-30.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	01.05.2025-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач(ка) вищої освіти _____ Олександр ПОГОРІЛИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Людмила СОЛЯНИК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 59 стор., 8 табл., 17 рис., 20 джерел.

Мета роботи – оптимізація процесу створення інтерактивного аудіо-візуального контенту з використанням методів генеративної графіки.

Об'єкт дослідження – генерація візуального контенту на основі звукових параметрів та сенсорної взаємодії.

Предмет дослідження – програмне забезпечення для генерації інтерактивного контенту у реальному часі.

Короткий зміст роботи: У роботі проаналізовано сучасні підходи до створення інтерактивного аудіо-візуального контенту в реальному часі на основі реакції системи на аудіо- та відеосигнали. Досліджено особливості нодової архітектури середовища TouchDesigner, а також можливості використання Python для обробки сигналів і оптимізації параметрів генерації. Розроблено структурну модель та реалізовано прототип системи, який формує візуальні ефекти з частинок на основі силуету користувача, аудіоаналізу ритму та частот, а також реакції на жести. Реалізовано ключові функції: захоплення відеопотоку, перетворення зображення у силует, аудіоаналіз, генерація частинок на GPU, керування візуальними параметрами через скрипти Python. У ході роботи використано вбудовані компоненти TouchDesigner (TOP, CHOP, SOP), а також розроблено низку користувацьких скриптів для реакції системи на зміну вхідних даних. Проведено тестування роботи системи в режимі реального часу.

Сферою використання застосунку є інтерактивні інсталяції для мистецьких заходів, виставок, DJ/VJ-сетів або освітніх демонстрацій.

КЛЮЧОВІ СЛОВА: ІНТЕРАКТИВНА ВІЗУАЛІЗАЦІЯ, TOUCHDESIGNER, PYTHON, АУДІОАНАЛІЗ, ГЕНЕРАЦІЯ ЧАСТИНОК, ВІДЕОЖЕСТИ, НОДОВА АРХІТЕКТУРА, GPU-ВІЗУАЛІЗАЦІЯ, РЕАЛЬНИЙ ЧАС.

ЗМІСТ

ВСТУП

Актуальність: З розвитком технологій генеративного мистецтва, мультимедійних перформансів та інтерактивних інсталяцій, зростає попит на інструменти, здатні поєднувати візуальну динаміку з живою взаємодією та звуковим супроводом у реальному часі. Особливої актуальності набувають системи, що дозволяють створювати інтерактивні візуальні середовища для використання у сферах культури, медіа, музики та освіти. У цьому контексті важливу роль відіграють інструменти, які забезпечують взаємозв'язок між аудіо-, відео- та поведінковими даними користувача.

Одним із таких сучасних інструментів є TouchDesigner — візуальне середовище та мова програмування для побудови інтерактивних медіасистем, яке дозволяє у режимі реального часу обробляти зовнішні сигнали, виконувати графічний рендеринг на GPU та інтегрувати Python-логіку [1, 2]. Завдяки цьому TouchDesigner стає платформою для створення адаптивних систем генерації візуального контенту, що реагує на звук, рух або інші сенсорні сигнали.

Об'єктом дослідження є генерація візуального контенту на основі звукових параметрів та сенсорних взаємодій.

Предметом дослідження є програмне забезпечення для генерації інтерактивного контенту у реальному часі.

Метою роботи є оптимізація процесу створення інтерактивного аудіо-візуального контенту з використанням методів генеративної графіки.

Методи дослідження: На початковому етапі було проаналізовано сучасні підходи до генеративної візуалізації, систем мультимедійної обробки та інструменти реального часу [3–5]. Далі досліджено можливості TouchDesigner як мови і середовища, а також його взаємодію з Python. Обґрунтовано архітектуру системи, розроблено скрипти обробки сигналів, побудовано топологію нодової структури для генерації частинок. Також у процесі реалізації системи проводилось модульне тестування, оптимізація та аналіз затримки й продуктивності.

Наукова новизна роботи полягає у поєднанні механізмів аналізу аудіо- та відеосигналів у реальному часі з динамічним керуванням параметрами GPU-візуалізації за допомогою скриптів на Python, що забезпечує адаптивну реакцію візуального середовища на зовнішні стимули.

Практична значущість результатів полягає у можливості застосування створеної системи для побудови інтерактивних інсталяцій, цифрового сценічного оформлення, DJ/VJ сетів, освітніх демонстрацій та медіаперформансів. Розроблене рішення може бути використане на більшості сучасних мультимедійних платформ із підтримкою відеозахоплення та аудіовходу.

1. АНАЛІЗ ІНТЕРАКТИВНИХ СИСТЕМ ГЕНЕРАЦІЇ АУДІО-ВІЗУАЛЬНОГО КОНТЕНТУ

1.1 Сутність та значення інтерактивних аудіо-візуальних систем

Інтерактивні аудіо-візуальні системи є важливою складовою сучасного цифрового середовища. Вони поєднують у собі компоненти візуалізації, звуку та взаємодії з користувачем у реальному часі, створюючи унікальний мультимедійний досвід. Такі системи дедалі частіше використовуються в галузях цифрового мистецтва, дизайну середовища, сценографії, електронної музики, виставкових інсталяцій та освіти [1].

Основна відмінність інтерактивних систем від традиційних мультимедійних творів полягає у їхній динамічній природі: візуальний результат формується не лише на основі закладених сценаріїв, а й у відповідь на зовнішні фактори — звук, рух, час або взаємодію з глядачем. Таким чином створюється ефект живого твору, що змінюється в реальному часі.

Сучасні технології дозволяють будувати інтерактивні системи з високим ступенем складності. Вони можуть реагувати на аудіосигнали, мову, рухи тіла, температуру, координати GPS тощо. Однак найбільш поширеними є саме аудіо- та відео-реактивні системи, які аналізують вхідний звук або зображення з камери та перетворюють ці дані на візуальний образ.

Важливими характеристиками таких систем є:

— адаптивність — здатність реагувати на зміну зовнішніх умов;

— неповторність — контент формується унікальною комбінацією чинників;

— візуальна експресивність — потужний емоційний вплив через образи;

— суб'єктивність — глядач стає учасником і навіть співтворцем візуального процесу [2].

Завдяки своїй універсальності, інтерактивні системи використовуються як інструмент комунікації, впливу, навчання та мистецтва. Вони розмивають межу між технологією та творчістю, між інженерією та естетикою, між об’єктом і суб’єктом.

Переваги інтерактивних аудіо-візуальних систем:

— залучення користувача — взаємодія стимулює інтерес та сприяє глибшому зануренню у контент;

— унікальність результату — кожна взаємодія породжує індивідуальний досвід;

— гнучкість застосування — можуть використовуватись у мистецтві, освіті, на заходах тощо;

— реалізація творчих ідей — спрощується створення складних візуальних сценаріїв навіть без глибоких знань у програмуванні;

— підвищення емоційного впливу — поєднання звуку та зображення забезпечує сильний сенсорний ефект.

Недоліки інтерактивних аудіо-візуальних систем:

— високі технічні вимоги — потребують потужного обладнання та графічних процесорів;

— складність розробки — необхідні знання візуального програмування, обробки медіаданих, дизайну;

— нестабільність у реальному часі — можливі затримки, збої при роботі з потоковими даними;

— обмежена доступність — складність створення таких проєктів обмежує їх використання непідготовленими користувачами. використовуються як інструмент комунікації, впливу, навчання та мистецтва. Вони розмивають межу між технологією та творчістю, між інженерією та естетикою, між об’єктом і суб’єктом.

1.2 Специфіка застосування інтерактивних аудіо-візуальних систем

Інтерактивні аудіо-візуальні системи посідають особливе місце в цифровому мистецтві завдяки своїй здатності створювати унікальні мультимедійні досвіди, які змінюються в режимі реального часу залежно від дій глядача чи впливу навколишнього середовища. Вони поєднують візуальні ефекти, звук та взаємодію, відкриваючи нові можливості для митців і дизайнерів у створенні нестандартного мистецького контенту [4].

Основною особливістю таких систем є мультимодальність взаємодії — користувач може впливати на контент через звук, рух, голос або інші сенсорні дані. У результаті створюється унікальний, не повторюваний досвід, який не лише відображає задум автора, але й частково формується самим глядачем. На рисунку 1.1 продемонстровано один із прикладів взаємодії користувача з такими системами.

Рис. 1.1 Приклад взаємодії користувача з інтерактивною інсталяцією

Сучасне цифрове мистецтво активно використовує такі системи в наступних форматах:

— інсталяції – реагують на присутність або звук глядача, змінюючи форму чи колір зображень;

— сценографія для перформансів – створення візуальних композицій, які залежать від музичного супроводу або рухів виконавця;

— медіаінтерфейси для виставок – сенсорні стіни або інтерактивні панелі, що дозволяють зануритись у тему експозиції;

— вуличні інсталяції та мепінг – проєкції на архітектурні об’єкти, які реагують на звуки або активність у просторі;

Приклади застосування інтерактивних аудіо-візуальних систем показано в таблиці 1.1.

Таблиця 1.1

Приклади застосування інтерактивних аудіо-візуальних систем

Сфера застосування	**Тип взаємодії**	**Приклад реалізації**
Цифрове мистецтво		Інсталяція, що змінює колір і форму при наближенні глядача
Виступи музикантів	Реакція на частоти музики	Візуалізація частинок у реальному часі під час DJ-сету
Музеї та виставки	Реакція на звук або голос	Інтерактивний експонат, що «відповідає» на звук голосом або графікою

Продовження таблиці 1.1

Приклади застосування інтерактивних аудіо-візуальних систем

Сфера застосування	**Тип взаємодії**	**Приклад реалізації**
Освітні платформи	Реакція на дотик/рух	Навчальні панелі, що змінюють зображення при взаємодії
Реклама	Аудіо- або відео-активація	Інтерактивні білборди, що реагують на фоновий шум або рух автомобілів

Такі системи відрізняються високим рівнем емоційної залученості глядача, адже кожна взаємодія породжує унікальну графічну реакцію. Це особливо актуально в мистецькому середовищі, де інтерпретація твору часто залежить від контексту сприйняття.

З технічної точки зору, подібні проєкти базуються на спеціалізованих платформах, які дозволяють митцям працювати з медіа у реальному часі. Наприклад, TouchDesigner, Max/MSP, VVVV та Processing — це середовища візуального програмування, які дозволяють будувати складні логіки генерації зображень, синхронізованих із зовнішніми даними

Ще однією важливою специфікою є легкість інтеграції з апаратними засобами: камери, мікрофони, сенсори руху, MIDI-контролери. Це дає змогу створювати складні, але адаптивні мультимедійні рішення, що працюють у просторі та часі.

Таким чином, інтерактивні аудіо-візуальні системи у цифровому мистецтві характеризуються гнучкістю, естетичною виразністю та здатністю до персоналізованого досвіду. Вони суттєво трансформують підходи до створення мистецького контенту, дозволяючи поєднати інженерію, музику та візуальне мистецтво в одному цілісному продукті.

1.3 Цільова аудиторія

Інтерактивні аудіо-візуальні системи орієнтовані на широку, проте спеціалізовану аудиторію, яка цінує взаємодію, експериментальність та нові форми естетичного досвіду. Залежно від контексту використання, такі системи можуть бути спрямовані як на професійних митців, так і на кінцевих користувачів без технічної підготовки.

Основні категорії цільової аудиторії:

1. Митці та медіа-дизайнери.

Ця категорія охоплює художників, сценографів, музикантів, VJ-їв та композиторів, які використовують інтерактивні системи для створення мистецьких інсталяцій, вистав, музичних шоу тощо. Для них важлива гнучкість у роботі з візуальними та звуковими елементами, підтримка реального часу та адаптація під потреби конкретного перформансу.

2. Креативні технологи та розробники мультимедійних рішень.

Цільова аудиторія, яка займається технічною реалізацією проєктів: створенням програмного забезпечення для інтерактивних експозицій, виставкових павільйонів, рекламних дисплеїв. Для неї критичним є інтегрування з апаратним забезпеченням (сенсори, камери, мікрофони) та можливість керування параметрами візуалізації через скрипти.

3. Установи культури та освіти.

Музеї, галереї, інтерактивні центри науки активно впроваджують інтерактивні рішення у свої експозиції. У цьому контексті головне — простота користування, інтуїтивність взаємодії та наочність подання складної інформації через графіку та звук.

4. Студенти та дослідники в галузі медіа-мистецтва.

Навчальні заклади використовують подібні системи для вивчення тем генеративного мистецтва, візуального програмування, інтерактивного дизайну. Для цієї аудиторії важливою є документація, простота вивчення інструментів і доступність ресурсів для творчого експерименту [6].

5. Глядачі (кінцеві користувачі).

Особи, які безпосередньо взаємодіють з інсталяцією або програмою. Їм важливі інтуїтивний інтерфейс, миттєва візуальна реакція, а також естетичний і емоційний ефект взаємодії.

Очікування цільової аудиторії:

— низький поріг входу для митців і новачків (мінімальне програмування);

— висока гнучкість при створенні унікального контенту;

— можливість інтеграції з різними джерелами даних;

— робота в реальному часі без помітних затримок;

— висока візуальна привабливість результату;

— можливість персоналізації візуального досвіду.

Таким чином, інтерактивні аудіо-візуальні системи об’єднують інтереси митців, інженерів, освітян і глядачів. Універсальність та багатофункціональність таких систем дозволяє створювати рішення, які виходять за межі окремої галузі — від перформативного мистецтва до інтерактивної освіти.

1.4 Підходи до реалізації

На сучасному етапі розвитку цифрових технологій реалізація інтерактивних аудіо-візуальних систем базується на комбінації алгоритмічного дизайну, обробки даних у реальному часі та адаптивної візуалізації. Основною тенденцією є перехід від жорстко закодованих медіа до відкритих генеративних моделей, здатних змінюватися у відповідь на зовнішні стимули — звук, рух, світло або взаємодію з користувачем.

Серед основних сучасних підходів можна виокремити такі:

1. Візуальне програмування. Сутність підходу полягає у використанні графічного середовища для створення логіки взаємодії між даними. Замість написання коду, користувач оперує блоками (нодами), які обробляють аудіо- або відеопотік та генерують візуальний результат. Це значно спрощує процес розробки для митців без технічної освіти.

Найбільш поширені платформи:

— TouchDesigner — гнучке середовище з підтримкою інтерактивної графіки, аудіоаналізу, виводу на екрани та інші пристрої у реальному часі [1];

— Max/MSP Jitter — орієнтована на медіаобробку, звук і відео в інтерактивному середовищі;

— VVVV — система з глибокою інтеграцією даних з апаратних сенсорів і великою гнучкістю у побудові візуальних моделей.

2. Генеративна графіка та алгоритмічне мистецтво. Цей підхід передбачає створення візуального контенту на основі формальних правил та алгоритмів. Система заздалегідь не знає, який буде результат, — зображення формується у реальному часі залежно від змінних параметрів: аудіоамплітуди, спектру, ритму, рухів тощо [7].

Для реалізації таких рішень активно використовують:

— Processing — мова програмування для художників, орієнтована на креативне кодування;

— OpenFrameworks, p5.js — альтернативні засоби для створення генеративного мистецтва;

— Python (у поєднанні з бібліотеками: pyaudio, opencv, numpy) — часто використовується як логічний шар у більш складних проєктах.

3. Аудіоаналіз у реальному часі. Більшість сучасних систем інтерактивної візуалізації використовують розкладання аудіосигналу на частоти та амплітуду з подальшою реакцією візуальних параметрів на ці дані. Алгоритми можуть включати:

— перетворення Фур’є (FFT);

— виявлення ритмічних імпульсів;

— аналіз спектру та візуальна відповідність частотним діапазонам.

4. Використання камер та трекінгу руху. Окрім аудіоаналізу, все більше інсталяцій використовують вхід з камери, аналізуючи рух користувача, його позицію чи жести. Це дозволяє створювати повноцінні мультимодальні системи, де візуальний контент реагує на кілька факторів одночасно.

Для цього використовують:

— камери з підтримкою глибини (наприклад, Kinect, RealSense);

— інструменти комп’ютерного зору (OpenCV, MediaPipe);

— нейронні мережі для виявлення тіла, обличчя, жестів.

5. Комбінування програмних і апаратних засобів. Сучасні рішення активно інтегрують програмні середовища з фізичними інтерфейсами: MIDI-контролерами, датчиками руху, мікрофонами, лазерами, екранами з високою роздільною здатністю. Це дає змогу створювати комплексні інсталяції, які функціонують у просторі, а не тільки на екрані.

Таким чином, реалізація інтерактивних аудіо-візуальних систем вимагає глибокого розуміння сучасних технік генеративного дизайну, обробки даних у реальному часі, візуального програмування та мультимодальної взаємодії. Застосування згаданих підходів дозволяє створити гнучкі, естетично привабливі рішення, що змінюються відповідно до аудіовізуального середовища.

1.5 Вимоги до програмного забезпечення

Аналіз сучасних підходів до створення інтерактивних аудіо-візуальних систем, а також дослідження інструментів, які найчастіше застосовуються в цифровому мистецтві, дозволяє сформулювати вимоги до майбутнього програмного забезпечення.

У проєкті буде реалізовано інтерактивну інсталяцію, яка в режимі реального часу генерує візуальні ефекти у відповідь на звуковий сигнал, жести користувача або інші зовнішні впливи. Система має поєднувати гнучкість, розширюваність та стабільність роботи під навантаженням. Її розробка базується на візуальному середовищі TouchDesigner з використанням скриптів на Python для реалізації логіки поведінки ефектів і обробки даних.

На основі поставленої мети сформульовано функціональні та нефункціональні вимоги до системи:

Функціональні вимоги:

— програма повинна здійснювати аналіз аудіосигналу в реальному часі;

— на основі аудіоаналізу має відбуватись генерація візуального контенту у вигляді та графічних частинок, що змінюються відповідно до звукових характеристик;

— система повинна реагувати на жести або рухи користувача, отримані через сенсори, змінюючи параметри візуалізації;

— програма повинна забезпечувати синхронну роботу аудіоаналізу та візуальної генерації без ручного втручання під час виконання.

Нефункціональні вимоги.

— висока стабільність роботи в режимі реального часу(30хв+);

— мінімальна затримка між обробкою аудіо та візуальним відгуком(до 100 мс), та між обробкою жестів/рухів(до 150мс);

— оптимізована продуктивність для плавної графіки (в ідеалі - 60 fps, допустимо - 30 fps).

2. ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

У межах реалізації кваліфікаційної роботи було обрано середовище TouchDesigner як основну платформу розробки інтерактивного програмного забезпечення. TouchDesigner — це потужне візуальне середовище розробки, що поєднує в собі функціонал медіа-серверу, інструментів графічного дизайну, системи візуального програмування та рушія реального часу. Воно створене канадською компанією Derivative і орієнтоване на митців, дизайнерів, дослідників медіа та розробників генеративних систем [1].

TouchDesigner орієнтований на роботу в режимі реального часу, що є критично важливим для задач генеративної візуалізації, оскільки дозволяє миттєво відображати результат змін у коді або структурі сцени. Система використовує GPU-прискорення для обробки зображення, що забезпечує високу продуктивність навіть у складних сценах[3].

Однією з ключових переваг TouchDesigner є його гнучкість у роботі з зовнішніми пристроями. Він підтримує численні протоколи для зв'язку з іншими системами, такі як:

— OSC (Open Sound Control);

— MIDI;

— DMX;

— WebSocket / TCP / UDP;

— Kinect, Leap Motion, RealSense — для взаємодії з користувачем;

— Spout / NDI — для передавання відео в реальному часі між додатками.[1]

Для користувачів, які мають досвід програмування, TouchDesigner також відкриває широкі можливості завдяки вбудованій підтримці мови Python. Це дозволяє реалізовувати складну логіку взаємодії, автоматизацію, генерацію даних

та обробку подій, які важко або неможливо реалізувати лише засобами візуального програмування.[2]

TouchDesigner активно використовується в:

— цифровому мистецтві;

— сценічних інсталяціях;

— генеративному дизайні;

— рекламних проєктах;

— архітектурному мапінгу;

— VJ-перформансах.[4]

Завдяки поєднанню гнучкого інтерфейсу, підтримки реального часу, потужного інструментарію для обробки графіки та відкритості до розширення за допомогою коду, TouchDesigner є ідеальним середовищем для реалізації системи, що генерує інтерактивний аудіо-візуальний контент.

2.2 Мови програмування

Мови програмування є основним інструментом для створення програмного забезпечення. Вони надають розробникам можливість формалізувати логіку взаємодії з даними, обробки інформації та керування апаратними ресурсами комп’ютера. Вибір мови програмування безпосередньо впливає на архітектуру системи, швидкість розробки, продуктивність та масштабованість готового рішення.

Сучасні мови програмування поділяються на кілька типів за рівнем абстракції (низькорівневі та високорівневі), парадигмами (процедурні, об’єктно-орієнтовані, функціональні) та способом реалізації (компільовані чи інтерпретовані). У сфері розробки інтерактивних візуальних систем найбільше поширення отримали високорівневі мови, які підтримують швидку розробку, інтеграцію з мультимедійними бібліотеками та інструменти візуального програмування.

TouchDesigner як мова візуального програмування

TouchDesigner є не лише середовищем розробки, але й інструментом візуального програмування, у якому логіка роботи проєкту створюється шляхом з'єднання операторів у графічному інтерфейсі. Кожен оператор (нод) виконує певну функцію — обробку відео, аудіо, геометрії або взаємодію з даними. Таким чином, TouchDesigner дозволяє будувати складні інтерактивні системи без написання коду, що особливо зручно для митців, дизайнерів та дослідників, які не мають глибокої технічної підготовки.

Програмування в TouchDesigner реалізується через логічні блоки, які відповідають за різні типи операцій:

— TOP (Texture Operators) — для обробки зображень;

— CHOP (Channel Operators) — для роботи з часом і сигналами;

— SOP (Surface Operators) — для роботи з геометрією;

— DAT (Data Operators) — для обробки текстових даних;

— COMP (Component Operators) — для побудови складних інтерфейсів.

Це дає змогу створювати графічні проєкти на основі логічних зв’язків без необхідності писати традиційний програмний код, що відкриває широкі можливості для реалізації інтерактивних візуальних систем [1].

Мова програмування Python

Python — це високорівнева мова програмування загального призначення, що вирізняється простим синтаксисом, розвиненою екосистемою бібліотек і великою спільнотою розробників. Вона підтримує кілька парадигм програмування: процедурну, об’єктно-орієнтовану та функціональну. Python широко використовується в наукових дослідженнях, розробці вебзастосунків, автоматизації процесів, обробці даних і створенні інтерактивного контенту.

Однією з головних переваг Python є велика кількість готових бібліотек для роботи з мультимедійним контентом, графікою, машинним навчанням, а також інтеграція з різноманітними платформами. Python має відкритий код, активно розвивається та підтримується спільнотою розробників у всьому світі [2].

Застосування Python у TouchDesigner

У TouchDesigner Python використовується як внутрішня скриптова мова для реалізації додаткової логіки, що виходить за межі можливостей візуального програмування. Python дозволяє:

— створювати динамічні взаємодії між елементами системи;

— обробляти зовнішні події та сигнали;

— керувати параметрами нодів у реальному часі;

— імпортувати сторонні бібліотеки;

— зберігати та обробляти дані.

Використання Python розширює межі стандартних можливостей TouchDesigner. Наприклад, можна реалізувати адаптивну поведінку об’єктів у відповідь на аудіоаналіз, будувати складні алгоритми генерації контенту або взаємодіяти з API зовнішніх сервісів.

Python у TouchDesigner використовує інтерпретатор CPython 3.x, що дозволяє інтегрувати більшість стандартних бібліотек. Код пишеться безпосередньо в Python Script DAT або як вирази в параметрах нодів. Таким чином, в одному проєкті можна органічно поєднувати нодову логіку з імперативним програмуванням [3].

2.3 Бібліотеки

Під час розробки інтерактивної системи генерації аудіо-візуального контенту важливу роль відіграють бібліотеки Python, які надають додаткові можливості для обробки даних, аналітики та взаємодії з зовнішніми пристроями. У цьому проєкті було використано три основні бібліотеки: OpenCV, NumPy та Pandas.

OpenCV (Open Source Computer Vision Library) — це одна з найпопулярніших бібліотек для комп’ютерного зору. Вона широко застосовується для обробки відео та зображень у реальному часі, виявлення об’єктів, відстеження рухів, фільтрації шумів, накладання ефектів тощо. У контексті даного проєкту OpenCV використовується для аналізу відеопотоку з камери, зокрема:

— виявлення присутності користувача;

— фільтрації кадрів;

— передобробки відео для інтеграції з TouchDesigner.

Бібліотека підтримує безліч алгоритмів обробки зображень, таких як згладжування (blur), детекція контурів, робота з колірними просторами, та працює з камерою в режимі реального часу [12].

NumPy (Numerical Python) — це базова бібліотека для наукових обчислень у Python. Вона надає інструменти для роботи з масивами, матрицями та широким набором математичних функцій. NumPy дозволяє ефективно обробляти великі обсяги числових даних, що важливо при роботі з аудіосигналами, а також при трансформації масивів відеофреймів перед візуалізацією. У системі генерації контенту NumPy використовується для:

— нормалізації даних;

— обчислення середніх значень амплітуди;

— підготовки вхідних масивів для подальшої обробки в TouchDesigner.

Ця бібліотека є особливо ефективною для швидкої обробки вхідних даних із низькою затримкою, що є критичним для інтерактивних систем [13].

Pandas — потужна бібліотека для обробки та аналізу табличних даних. Хоча у традиційному розумінні вона частіше застосовується в аналізі даних, у даному проєкті Pandas використовується для:

— логування подій;

— структурування зібраних даних під час тестування;

— аналізу результатів взаємодії користувача з системою;

— генерації статистичних звітів для відлагодження системи.

Завдяки зручному API та можливості легко фільтрувати, сортувати й групувати дані, Pandas дає змогу відстежувати ефективність системи, а також робити висновки на основі поведінкових шаблонів користувача [14].

Ці три бібліотеки працюють у синергії з TouchDesigner, дозволяючи розширити функціональні можливості середовища і покращити обробку вхідних даних. Вони забезпечують інструментарій, необхідний для складної логіки, аналітики та рефлексії поведінки користувача, без перевантаження основного візуального інтерфейсу. Функціональність цих бібліотек показано в таблиці 2.1.

Таблиця 2.1

Функціональність використаних бібліотек

Бібліотека	Основне призначення	Приклади використання
OpenCV	Обробка відео	Виявлення руху, фільтрація, робота з камерою
NumPy	Числові обчислення	Обробка аудіоамплітуд, масиви відеоданих
Pandas	Аналіз даних	Статистика взаємодії, логування, звіти

2.4 Система контролю версій

У процесі розробки програмного забезпечення важливу роль відіграє система контролю версій — інструмент, що дозволяє ефективно відстежувати зміни у проєкті, зберігати історію редагувань, працювати над кодом у команді, а також повертатися до попередніх станів проєкту у разі виникнення помилок або потреби в аналізі [16].

Контроль версій критично важливий для збереження послідовності розробки, документування змін та забезпечення стабільності. Він дає змогу:

— працювати паралельно над різними модулями системи;

— уникати конфліктів при редагуванні файлів;

— вести журнал змін;

— створювати ізольовані гілки для тестування нових функцій;

— автоматизувати процес розгортання.

Найбільш поширеною системою контролю версій сьогодні є Git — децентралізована система, яка забезпечує високу гнучкість і швидкість у роботі з репозиторіями. Вона підтримує локальне збереження змін, злиття гілок, тегування релізів і спільну роботу з іншими розробниками [16].

У межах цієї кваліфікаційної роботи використовувалась платформа GitHub — хмарний сервіс для зберігання Git-репозиторіїв, який надає зручний вебінтерфейс, функції керування проєктом, документацію, візуалізацію історії комітів і можливість спільного редагування. Окрім цього, GitHub дозволяє створювати приватні або публічні репозиторії, відкривати pull-запити, коментувати зміни, використовувати систему задач (issues) та вбудовану wiki-документацію [16].

Серед переваг використання GitHub для даного проєкту:

— централізований доступ до всієї історії змін;

— безпечне зберігання коду;

— простота резервного копіювання;

— можливість контролю версій як Python-скриптів, так і конфігураційних файлів TouchDesigner;

3. ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

3.1 Проектування архітектури системи

Проєктування архітектури інтерактивної системи генерації аудіо-візуального контенту є ключовим етапом у процесі розробки, оскільки визначає загальну структуру компонентів, їхню взаємодію та відповідальність. Основна мета архітектурного проєктування полягає в тому, щоб створити гнучку, масштабовану й ефективну систему, здатну працювати в реальному часі з високим ступенем стабільності та інтерактивності.

Обрана система має реактивний характер — вона змінює візуальний вихід залежно від дій користувача та/або параметрів аудіо. Архітектура побудована за принципами модульності, що дозволяє легко змінювати, додавати або видаляти окремі компоненти.

Загалом архітектура системи складається з таких основних модулів:

— модуль захоплення вхідних даних (аудіо, відео або сенсорна інформація);

— модуль попередньої обробки даних;

— модуль аналізу параметрів;

— модуль генерації візуального контенту;

— модуль логіки (на Python);

— модуль виведення зображення.

Кожен із цих модулів реалізований за допомогою відповідної нодової структури в TouchDesigner з інтегрованими Python-скриптами для логічного контролю та динамічного управління параметрами.

Загальну архітектуру системи продемонстровано на рисунку 3.1.

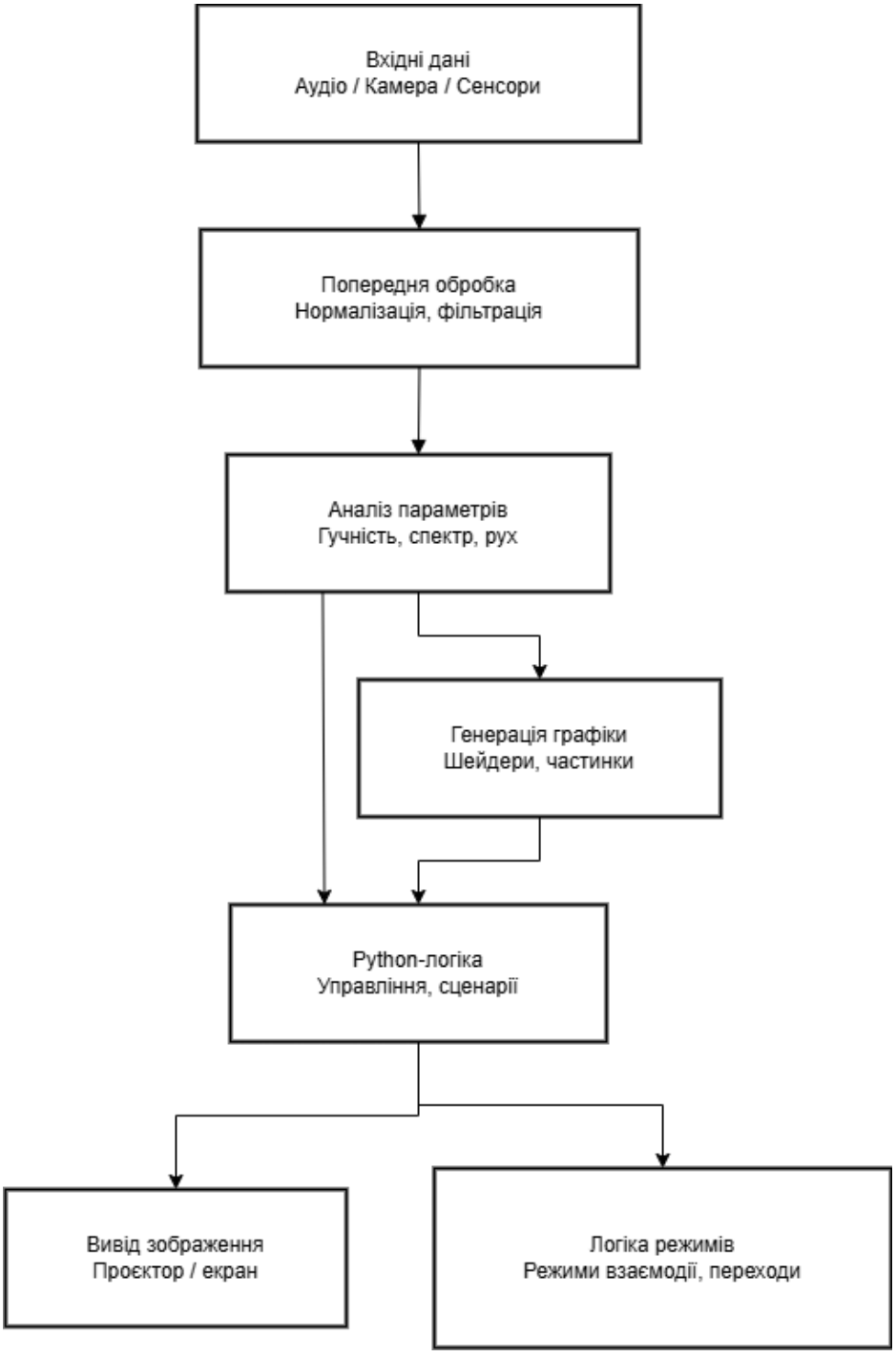


Рис. 3.1 Архітектура системи

Особливості архітектурного підходу:

1. Модульність – система проєктувалась із розподілом на незалежні компоненти, що дозволяє змінювати або оновлювати будь-який модуль без потреби перебудовувати інші.

2. Реактивність – всі обчислення відбуваються у реальному часі, що потребує оптимізації процесів та врахування затримок при обробці даних [6].

3. Паралельність обробки – завдяки мультипоточності TouchDesigner та можливості оптимізації через GPU-процеси, система здатна обробляти велику кількість вхідних параметрів паралельно.

4. Візуальне програмування – основна структура базується на візуальному зв’язуванні нодів, що робить процес проєктування прозорим і контрольованим, а також зручним для подальшої адаптації.

Завдяки інтеграції Python можливе додавання алгоритмічних елементів, які складно або неможливо реалізувати виключно за допомогою візуальних нодів. Наприклад, логіка генерації візуального шуму, поведінкових моделей частинок або аналіз руху з камери – усе це реалізується через Python-скрипти, вбудовані у відповідні модулі [1, 7].

Кожен компонент має свій конкретний вхід і вихід, що дозволяє будувати потік даних максимально прозоро. Основні модулі системи та їх функції наведено в таблиці 3.1.

Таблиця 3.1

Основні модулі системи та їх функції

Модуль	Опис функції
Захоплення даних	Отримання сигналів з мікрофона, камери або інших сенсорів
Попередня обробка	Нормалізація сигналу, згладжування шуму, виявлення ключових точок
Аналіз параметрів	Визначення рівня гучності, частотного спектру, інтенсивності руху
Генерація візуального	Побудова графіки відповідно до параметрів аудіо або відео
Логіка управління	Сценарна поведінка, перемикання режимів, обробка подій

Продовження таблиці 3.1

Основні модулі системи та їх функції

Модуль	Опис функції
Вивід	Відображення фінального зображення на екрані

3.2 Архітектура обробки даних користувача

На етапі проектування клієнтської частини системи основну увагу було приділено розробці механізмів збору, аналізу та інтерпретації даних від користувача в реальному часі. Основна мета — забезпечити інтуїтивну взаємодію людини із системою, де будь-який звук або рух може впливати на зміну візуального образу.

Основні джерела даних клієнтської частини:

— аудіовхід (мікрофон або музичний сигнал);

— відеовхід (камера користувача).

Модуль аудіоаналізу

Аудіосигнал є одним із ключових джерел взаємодії. У проекті він розділяється на чотири частотні діапазони, кожен з яких відповідає за певний візуальний параметр частинок:

— kick (низькі частоти) — керує швидкістю частинок;

— high (високі частоти) — впливає на турбулентність;

— rhythm (ритмічні патерни) — змінює розмір частинок;

— low (нижні частоти) — модулює колірну палітру частинок.

Для обробки сигналу використовуються ноди audioDeviceIn, audioAnalyze, audioSpectrum, які потім підключаються до системи частинок через CHOP to SOP та математичні обробники math, lag, limit, select [1, 6]. На рисунку 3.2 показано діаграмо взаємодії аудоаналізу з параметрами.

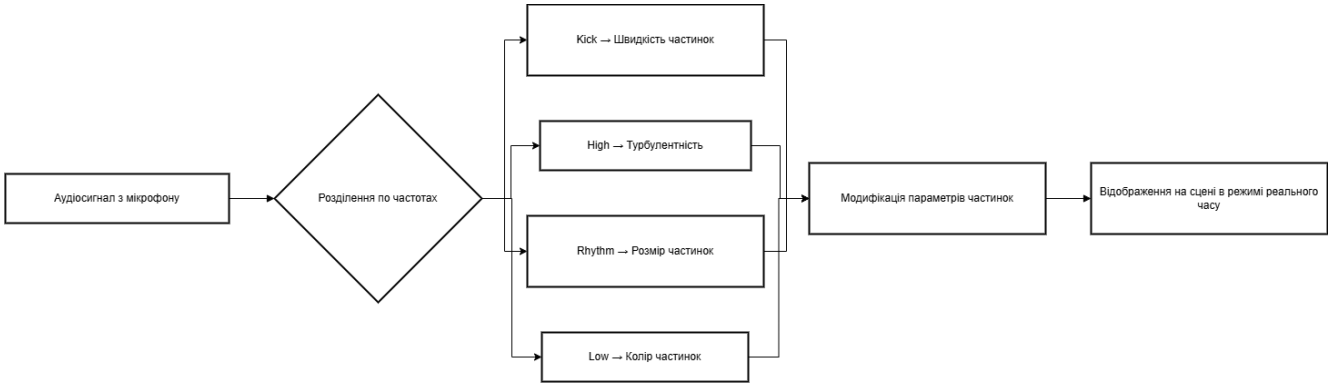


Рис. 3.2 Діаграма взаємодії аудіоаналізу з параметрами частинок

Модуль аналізу руху

Рух користувача відстежується за допомогою відеопотоку з камери та обробки оптичного потоку (opticalFlow, analyze, threshold). Система обробки даних визначає напрямок руху користувача та відповідно змінює напрямок руху частинок у візуалізації. Логіка реалізована так, що:

— вертикальні рухи (вгору/вниз) напряму відповідають на напрямок частинок;

— горизонтальні (вліво/вправо) — викликають зворотній напрямок руху частинок [5].

Це створює ефект динамічної реакції системи на дії користувача, при якому навіть незначне переміщення в полі зору камери візуалізується як зміна у сцені.

На рисунку 3.3 показано схему обробки руху користувача.

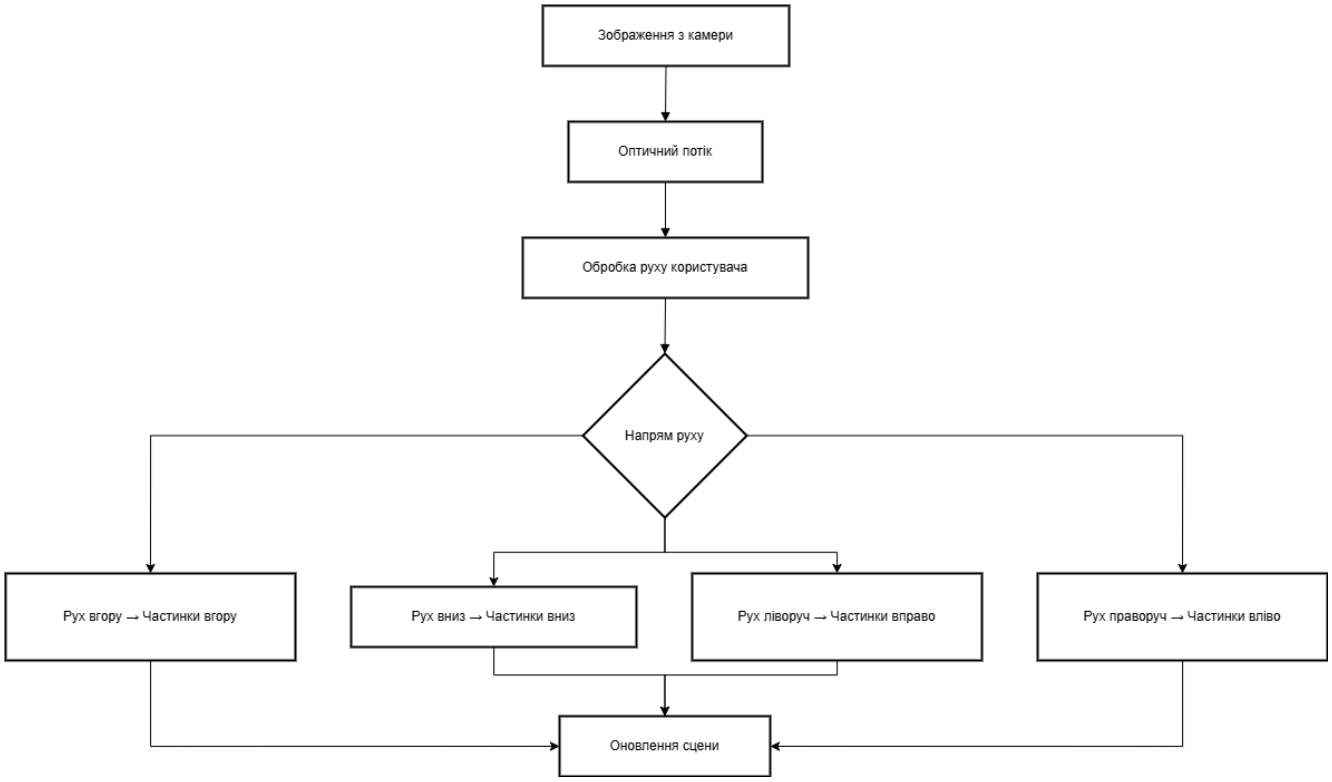


Рис. 3.3 Блок-схема аналізу руху користувача та зміни напряму частинок

Загальна логіка клієнтської частини

Всі сигнали (аудіо і відео) обробляються паралельно. Система створює інтерактивну реактивну візуалізацію, де параметри частинок (розмір, напрямок, швидкість, колір, турбулентність) змінюються синхронно з діями користувача. Це дозволяє досягти високого рівня залучення та персоналізації досвіду [4].

Усі сигнали агрегуються через центральний вузол управління, який логічно розподіляє дані до відповідних об'єктів у сцені. В TouchDesigner ця логіка реалізована за допомогою CHOP-ноди merge та розгалуження сигналів на групи параметрів, що керують SOP-геометрією частинок і шейдерами для їх рендеру [1, 6].

3.3 Архітектура генерації візуального контенту

Процес генерації візуального контенту в інтерактивній системі є ключовим елементом, що забезпечує естетичну цінність та емоційний вплив. Архітектура цієї частини проєкту була спроєктована з урахуванням вимог до високої візуальної

динаміки, відповідності до зовнішніх тригерів (звуку та руху) і стабільності роботи в реальному часі.

Основні компоненти генерації зображення

Візуальний контент генерується за допомогою мережі нод у TouchDesigner, яка обробляє вхідні сигнали та перетворює їх у графічні об'єкти, що анімовано змінюються в залежності від поведінки користувача. Основою візуалізації виступає система частинок (particles system), в якій геометричні об’єкти (точки або полігони) поводяться згідно з фізичними та логічними параметрами, що змінюються динамічно.

Кожен параметр частинок керується окремим каналом даних, які генеруються внаслідок аудіо- або відеоаналізу. Це дозволяє створити ефект живої сцени, яка ніколи не повторюється повністю.

На рисунку 3.4 показано загальну рендерингову структуру.

TouchDesigner дає змогу реалізувати складні графічні трансформації через поєднання SOP (Surface Operators), CHOP (Channel Operators), TOP (Texture Operators) та MAT (Materials). В основі генерації зображення були застосовані наступні принципи:

— турбулентність — формується на базі перлінівських шумів або даних з аудіоаналізу; створює ефект хаотичного руху частинок;

— колір — змінюється відповідно до низькочастотних аудіо-компонентів (Low); колірна палітра адаптується в реальному часі [1] ;

— швидкість — визначається рівнем амплітуди Kick-сигналу; використовується для створення вибухових або імпульсних візуальних ефектів;

— розмір — змінюється залежно від ритмічних аудіо-елементів; дозволяє підкреслити музичну структуру [4] ;

— напрямок руху — синхронізується з вектором оптичного потоку з камери; реалізується шляхом зміни нормалей або векторів переміщення частинок.

Рис. 3.4 Рендерингова структура

Процес формування сцени

У фінальній стадії, всі параметри об'єднуються в єдину геометричну структуру (SOP), після чого передаються в рендеринговий вузол, який формує остаточне зображення на екрані. Візуальний стиль підтримується матеріалами (Phong MAT, PBR MAT), анімацією камери (Camera, Render TOP), та пост-ефектами (Bloom, Feedback, Blur, Level тощо) [1, 6].

Система також містить модуль буферизації кадрів (feedback loop), що дозволяє зберігати попередні кадри та накладати їх на поточне зображення, створюючи ефект “шлейфу” або залишкового руху — популярного прийому у візуальних перформансах [5].

Гнучкість налаштувань

Система була спроєктована так, щоб дозволити швидке перепідключення каналів керування до різних візуальних параметрів. Це забезпечує гнучкість у налаштуванні сцен без необхідності повного перекомпонування проєкту.

Крім того, можлива подальша інтеграція з іншими пристроями — наприклад, MIDI-контролерами, сенсорами, або мережевими потоками даних через протоколи OSC та WebSocket [1].

3.4 Архітектура скриптової логіки

Для забезпечення адаптивності й динамічної реакції системи на вхідні сигнали, під час проєктування було закладено концепцію модульної скриптової архітектури, де кожен Python-скрипт виконує окрему вузькоспеціалізовану функцію. Такий підхід дозволяє підвищити масштабованість, гнучкість, а також значно спрощує налагодження та підтримку.

Концептуальна побудова логіки

Уся скриптова логіка системи поділяється на три основні категорії:

— обробка аудіосигналу — скрипти для збору, аналізу, класифікації частотних діапазонів;

— обробка відео/руху — аналіз рухів користувача з вебкамери, визначення напрямку руху;

— передача та трансляція параметрів — керування змінами параметрів візуалізації на основі вхідних даних, координація між нодами, модифікація станів частинок;

Такий поділ на функціональні модулі дозволяє реалізувати чітке розмежування відповідальностей між скриптами, що відповідає принципам структурного програмування та знижує когнітивне навантаження при розробці складних систем [6].

Планування структури скриптів

На етапі проєктування було складено карту системних залежностей: кожен Python-скрипт має чітко визначене місце в структурі TouchDesigner-проєкту. Основні пункти розміщення:

— у DAT-ноди типу Text DAT — кожен скрипт вбудовується у свою ноду;

— зв’язки через Callbacks / Custom Parameters / Run() — реалізують виклики між логічними модулями;

— використання Expression DAT — для безперервного моніторингу параметрів.

Умовну взаємодію скриптів наведено на рисунку 3.5.

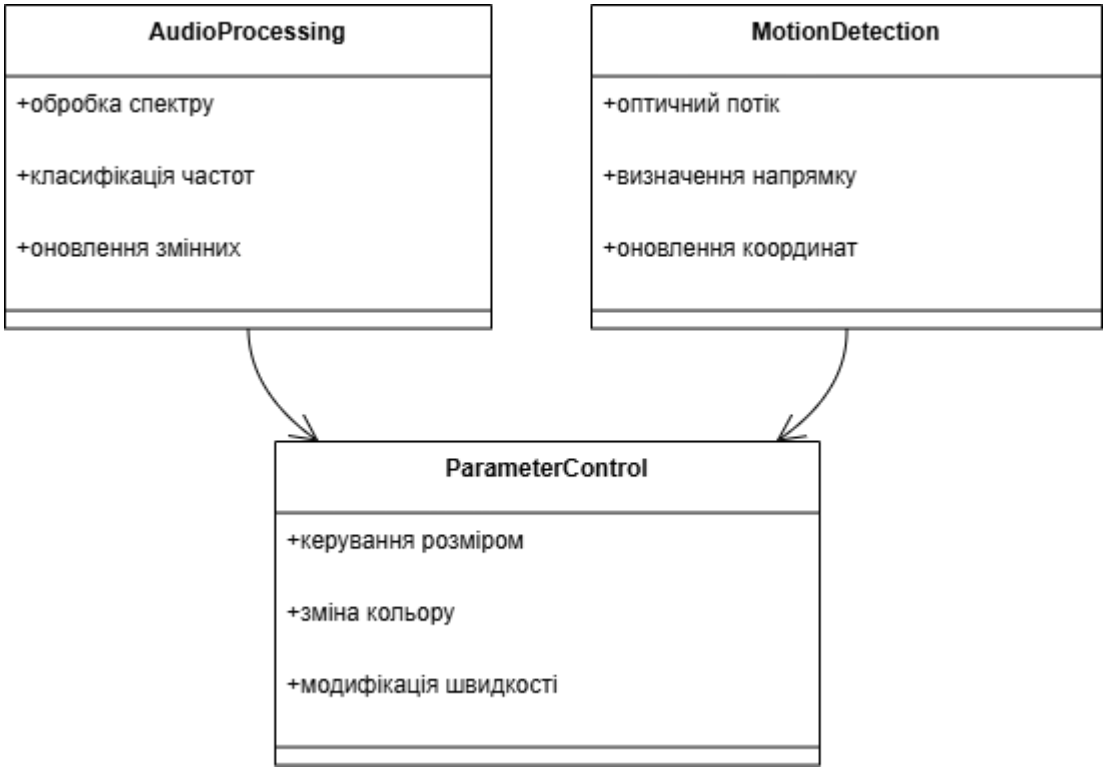


Рис. 3.5 Взаємозв’язок скриптів

Принципи проєктування скриптів

Кожен скрипт було запроєктовано відповідно до наступних принципів:

— читабельність і лаконічність — кожна функція виконує одну дію;

— мінімальна кількість залежностей — уникнення складних імпортів та сторонніх бібліотек;

— реактивність — логіка базується на подіях (наприклад, зміна значення параметра);

— перевірка вхідних даних — перед обробкою здійснюється валідація сигналів.

Визначення обов’язкових скриптів

На рівні проектування було визначено потребу у щонайменше 9 окремих скриптів, кожен з яких відповідає за:

— розбиття аудіо на діапазони, оновлення CHOP;

— аналіз потоку з камери, розрахунок вектору;

— присвоєння швидкості частинкам;

— обчислення кольору на основі low freq;

— розмір частинок залежно від ритму;

— турбулентність через high freq;

— визначення напрямку руху частинок;

— передача параметрів у відповідні SOP;

— журналювання подій для налагодження.

Абстрактна логіка взаємодії

Всі скрипти працюють асинхронно та не блокують один одного, що забезпечує стабільність роботи системи в режимі реального часу. Скрипти активуються залежно від оновлення сигналів (наприклад, значення CHOP змінюється — запускається відповідний script DAT). На рисунку 3.6 показано схему активації скриптів за подією.

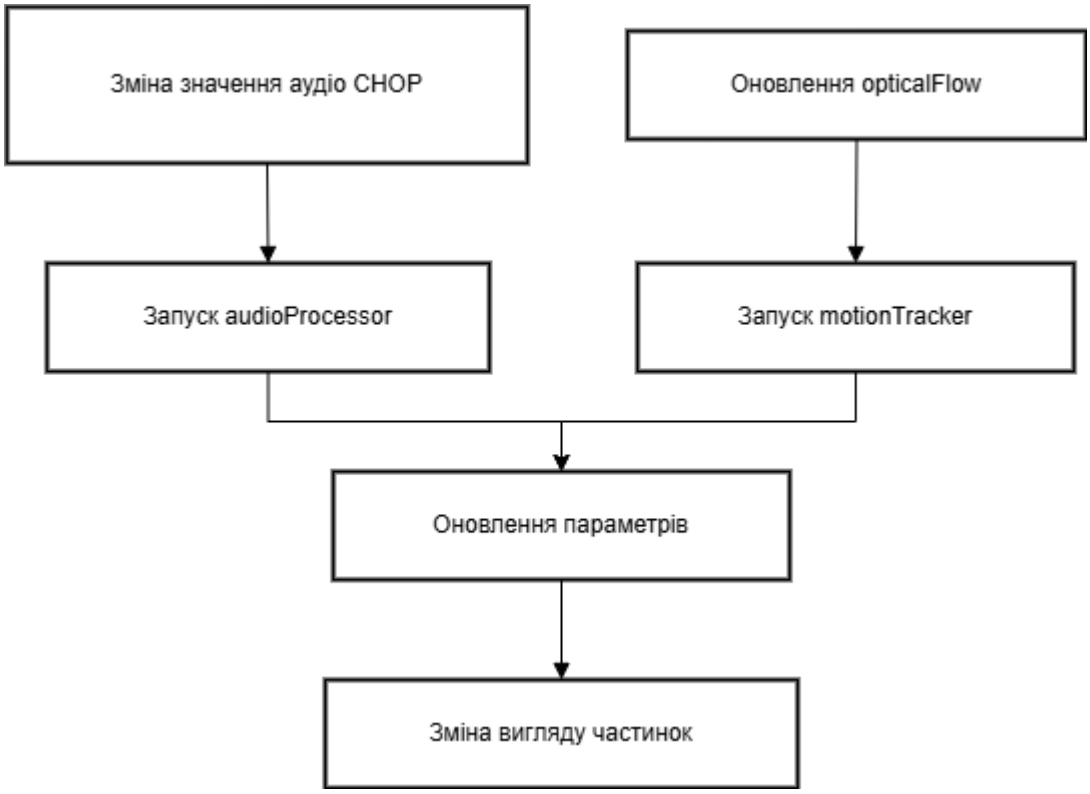


Рис. 3.6 Принципова схема активації скриптів за подією

4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Реалізація аудіоінтерактивного модуля

Аудіоінтерактивний модуль є ключовим компонентом системи, що забезпечує генерацію візуального контенту відповідно до вхідного аудіосигналу. У проєкті використано мікрофонний вхід або музичний сигнал, який обробляється у реальному часі за допомогою відповідних нодів у TouchDesigner.

Основні етапи обробки аудіосигналу:

1. Отримання аудіо. Початкове захоплення сигналу відбувається за допомогою ноди audioDeviceIn, яка підключається до внутрішнього мікрофону або аудіоінтерфейсу системи.

2. Аналіз спектру. Обробка сигналу здійснюється через audioAnalyze — потужну ноду, що дозволяє отримувати значення для певних частотних діапазонів, включно з low, mid, high, kick, rhythm та іншими. У проєкті були обрані ті, що мають найбільший візуальний потенціал: Kick, High, Rhythm і Low.

3. Виділення частотних зон. За допомогою select CHOP користувач виокремлює потрібні канали з audioAnalyze, наприклад: "kick", "rhythm", "high", "low", які відповідають окремим реакціям системи.

4. Згладжування та обмеження сигналів. Щоб уникнути стрибків у візуалізації, використовуються ноди lag — для пом'якшення переходів, та limit — для встановлення верхніх і нижніх меж сигналу.

5. Математична трансформація сигналів. Потім дані передаються до math CHOP, де масштабуються, нормалізуються, обробляються через combine channels, та готуються до передачі у візуальний блок. Наприклад, значення каналу kick нормалізується від 0 до 1 і далі множиться на множник швидкості частинок.

6. Подача сигналу в геометрію. Отримані значення передаються у відповідні параметри SOP-об’єктів (через CHOP to SOP, Null, або безпосередньо в Geo) для зміни їх властивостей — швидкості, розміру, колірного каналу та ін.

В таблиці 4.1 наведено зіставлення частот до параметрів.

Таблиця 4.1

Реалізоване зіставлення частот до параметрів

Частотна зона	Область звуку	Контролює параметр	Опис впливу
Kick	низькі частоти (20–80 Гц)	швидкість	частинки рухаються швидше або повільніше
High	високі частоти (2–8 кГц)	турбулентність	змінюється динамічність траєкторій
Rhythm	середні частоти (200–600 Гц)	розмір	частинки пульсують або змінюють масштаб
Low	нижній бас (80–200 Гц)	колір	змінюється відтінок/яскравість рендеру

Опис ключових нодів:

— audioDeviceIn — вибір джерела звуку та його захоплення;

— audioAnalyze — виділення характеристик сигналу (середнє, пікове значення, ентропія);

— select — вибір каналів для подальшої обробки;

— lag — задає інерцію для плавності змін;

— limit — встановлює допустимі межі значення;

— math — дозволяє масштабувати, інвертувати, сумувати та ін. дані перед передачею у візуальні модулі.

Вся реалізована структура аудіо аналізу продемонстрована на рисунку 4.1, а на рисунку 4.2 наведено спрощену схему обробки аудіосигналу.

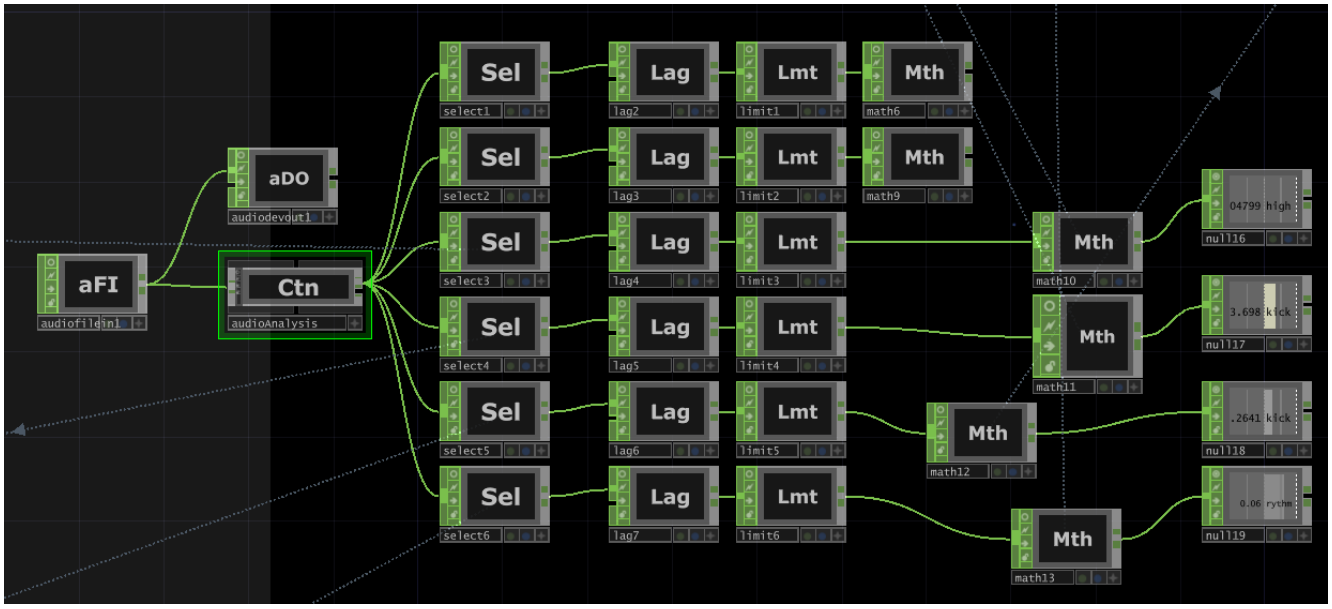


Рис. 0.1 Реалізований аудіо аналіз

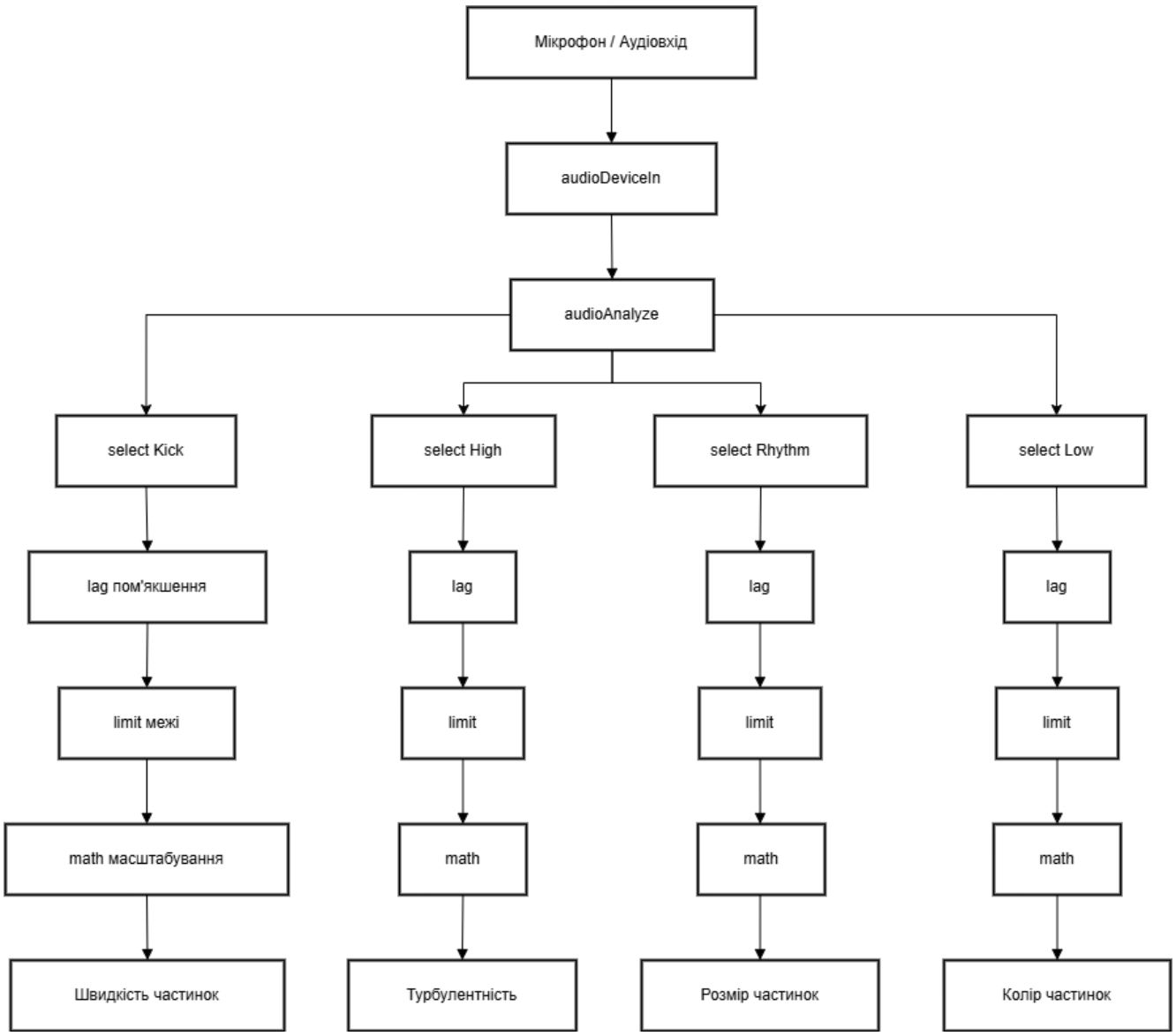


Рис. 0.2 Обробка аудіосигналу

4.2 Реалізація модуля аналізу руху користувача

Інтерактивність візуальної системи значною мірою залежить від реакції на рухи користувача. У цьому підрозділі детально розглянуто реалізацію модуля аналізу руху, який дає змогу змінювати напрямок руху частинок залежно від переміщення об’єкта в кадрі.

Загальна логіка модуля

Модуль аналізу руху побудовано на основі аналізу відеопотоку з камери. Вхідне відео опрацьовується за допомогою нод Video Device In, Reorder, Optical Flow, Threshold, Analyze та Math. Визначений напрямок руху передається до відповідних Constant-нод, що керують поведінкою частинок через систему Particle SOP.

Обробка відеосигналу

1. Отримання відео з камери — за допомогою ноди VideoDeviceIn, яка приймає вхідний потік з камери користувача.

2. Передобробка:

— Null використовується як буфер;

— Reorder перебудовує колірні канали (наприклад, RGBA → RGB).

3. Обчислення напрямку руху:

— OpticalFlow визначає зміну пікселів між кадрами. Цей вузол створює векторне поле напрямків руху [12] ;

— Threshold фільтрує незначні рухи;

— Analyze обчислює середнє значення векторів руху по напрямках (вверх/вниз/вліво/вправо).

4. Подальша логіка:

— значення з Analyze обробляються через Math, Logic, Function та Merge, щоб визначити напрямок з найвищим вектором;

— результати записуються до Constant1 (вплив по осі X) та Constant2 (вплив по осі Y), які змінюються залежно від обчислених напрямків.

В таблиці 4.2 описані характеристики та параметри використаних нод. На рис. 4.3 наведена спрощена діаграма обробки руху користувача, а на рисунку 4.4 продемонстрована загальна реалізована структура яка стосується обробки відео.

Переваги реалізованого рішення:

2. мінімальне споживання ресурсів у реальному часі;
3. висока точність виявлення руху;
4. можливість легко змінювати логіку реакції;
5. легка масштабованість (додавання жестів чи складніших сценаріїв).

Таблиця 4.2

Характеристики та параметри використаних нод

Назва ноди	**Призначення**	**Ключові параметри**
VideoDeviceIn	Відеовхід з камери	Resolution, Device, Active
OpticalFlow	Аналіз руху в кадрі	Window Size, Motion Vectors
Threshold	Виділення змін	Threshold Value, Output Range
Analyze	Отримання числових значень з потоку	Statistic (Average), Scope
Math	Нормалізація та масштабування значень	From Range → To Range
Logic	Перетворення значень у булеві	Logic Type, Compare To
Function	Оцінка функцій користувача	Expression
CHOP Execute	Обробка сигналу за подією в Python	valueChange(channel, sampleIndex, val, prev)

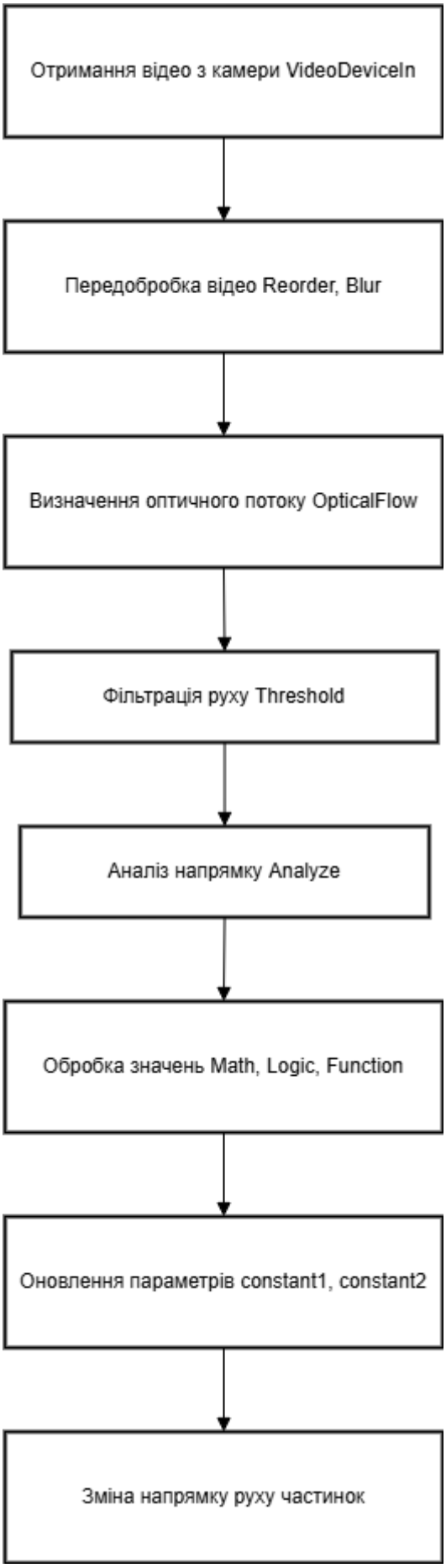


Рис 0.3 Обробка руху користувача

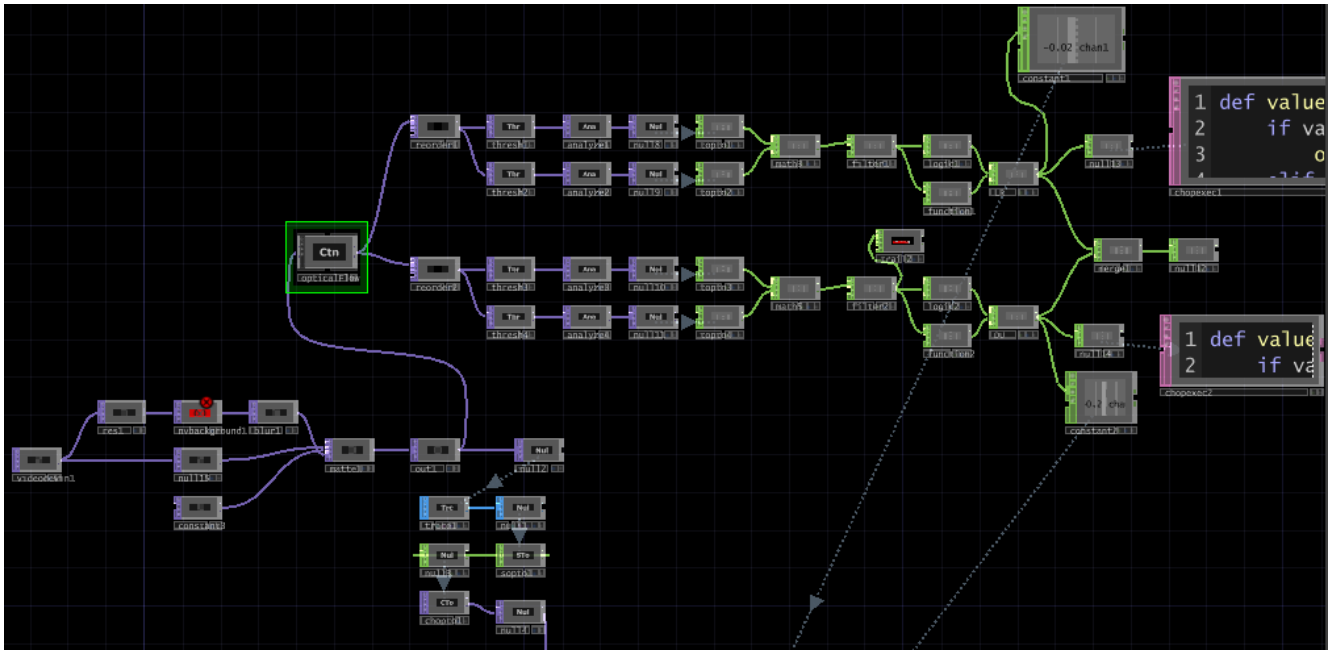


Рис. 0.4 Реалізована структура

4.3 Реалізація модуля генерації частинок

Модуль генерації частинок є центральним елементом аудіо-візуальної системи. Він відповідає за створення графічного контенту, який реагує на зміну параметрів аудіо- та відеоаналізу в реальному часі. В TouchDesigner для цього було використано ноду particlesGpu, яка забезпечує високу продуктивність завдяки обчисленням на графічному процесорі (GPU).

Базова структура генерації

В основі системи частинок лежить геометричний примітив — сфера (sphere1), яка має параметри радіусу, прив’язані до значень з аудіоаналізу. Для додання варіативності поверхні сфери використовувалась нода facet, а положення в просторі регулюється через transform. Таким чином, ми маємо початкову форму, яка динамічно змінюється відповідно до сигналів з модуля аналізу ритму. На рисунку 4.5 показана ця базова структура генерації.

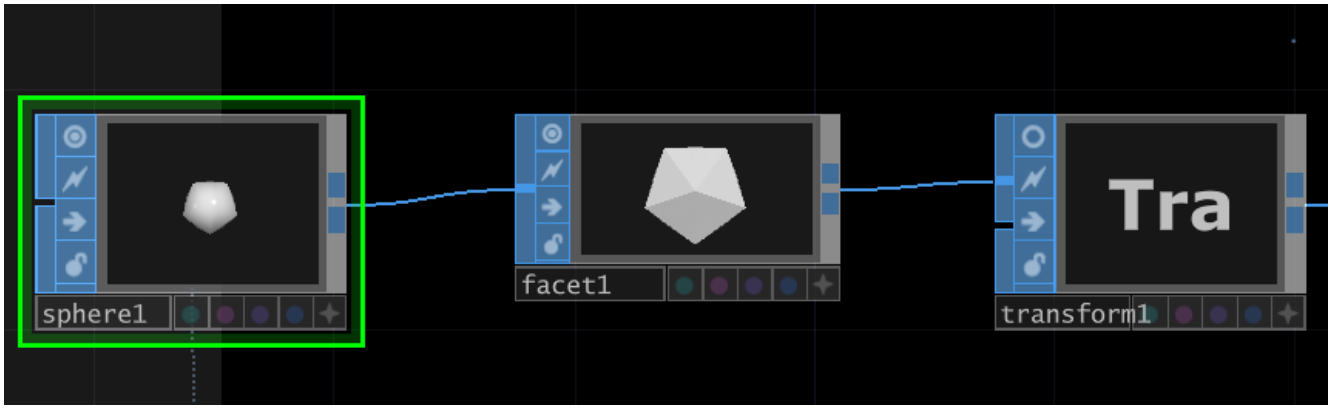


Рис. 0.5 Базова структура генерації

На етапі генерації частинок налаштовуються основні параметри в particlesGpu, зокрема:

— тип генерації: By Particles per Frame;

— кількість народжених частинок: 500 за кадр;

— тривалість життя: випадкове значення в діапазоні 5–10 кадрів;

— розмір: випадковий від 1 до 3 одиниць;

На рисунку 4.6 показано також і додаткові параметри Particles GPU.

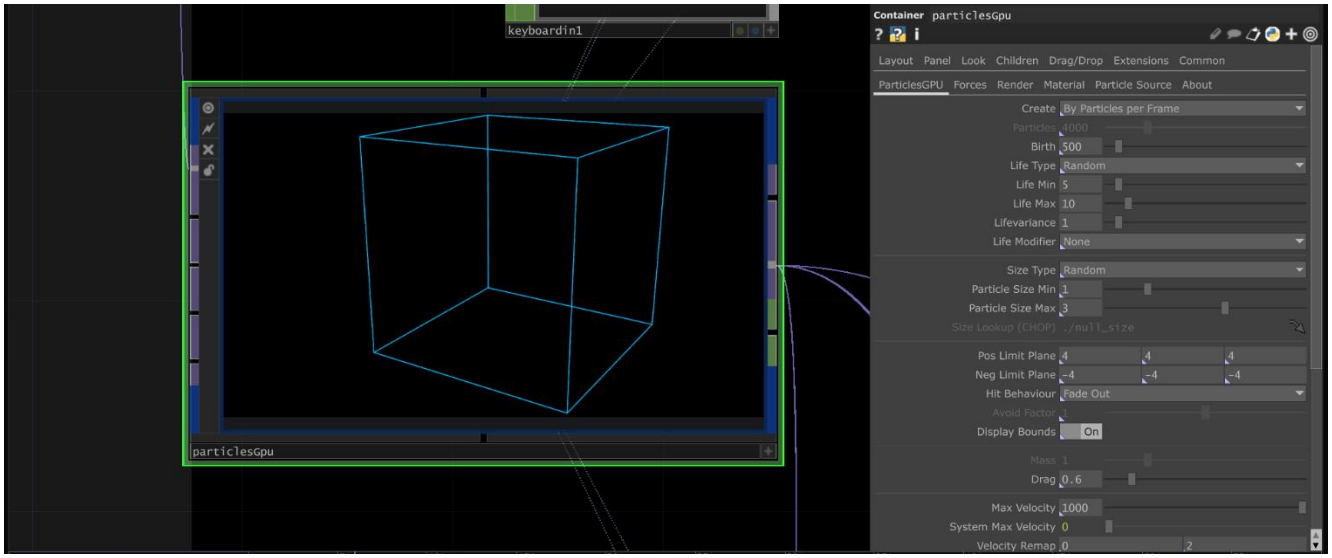


Рис. 0.6 Particles GPU

Ці параметри задають динамічність і природність поведінки частинок. Випадкова генерація розміру та життя створює ефект "живого" середовища, в якому кожна частинка унікальна.

Контроль сил та турбулентності

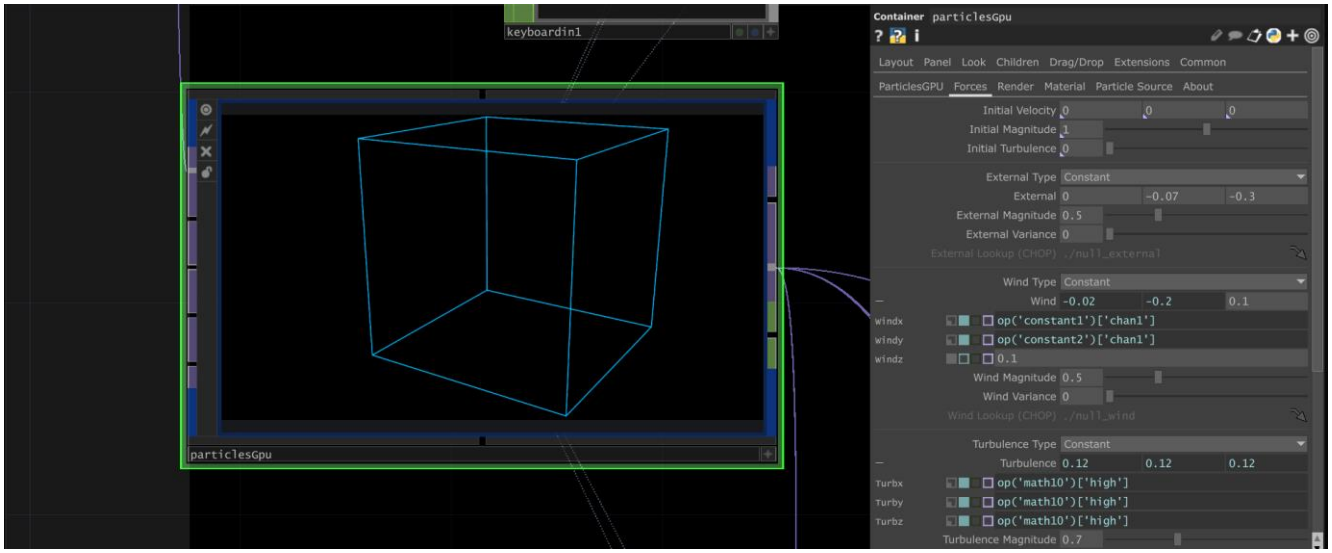
На вкладці Forces встановлено такі параметри:

— Initial Velocity: 0, 0, 0 — частинки не мають початкової швидкості, вона визначається зовнішніми силами;

— External Force: значення X і Y компонент беруться з constant1 та constant2, які контролюються рухом користувача;

— Turbulence: значення беруться з аудіоаналізу (високі частоти), прив’язані через ноду math10.

На рисунку 4.7 показано також додаткові параметри модуля Forces в ParticleGPU.



Рисунку 0.7 Particles GPU (Forces)

Таким чином, система реагує не тільки на музику, але й на рух користувача, створюючи справжню взаємодію між людиною та візуальним середовищем.

Вплив параметрів аудіо на систему частинок

Аналіз аудіо дозволяє керувати окремими характеристиками частинок:

— kick (низькі частоти) → швидкість частинок (через Noise → Null → Velocity) ;

— high (високі частоти) → турбулентність (через Math → Turbulence) ;

— rhythm (середні частоти) → розмір частинок (через Math → Sphere radius) ;

— low (бас) → колір (через Math → Noise → Color).

Ці параметри підключаються до відповідних входів у модулі particlesGpu, утворюючи комплексну систему, де навіть невеликі зміни у музиці візуально відображаються на сцені.

Генерація геометрії та її властивості

Основна фігура генерації — це сфера, створена нодою sphere1. Тип: Polygon, радіус задається через три канали op('math13')['rythm'], які відповідають за X, Y, Z компоненти.

Для варіації геометрії використовуються:

— facet — для зміни нормалей і варіацій світла;

— transform — для розміщення фігури в просторі;

— null — для зручної точки підключення до наступних операторів.

Результатом є система частинок, яка виглядає як хмара, що постійно змінює форму, масштаб, колір і напрямок руху відповідно до звуку та дій користувача. Це створює глибокий естетичний ефект, характерний для сучасних медіаінсталяцій [4, 5]. На рисунку 4.8 показана загальна схема впливу аудіо- та відеоданих на параметри системи частинок.

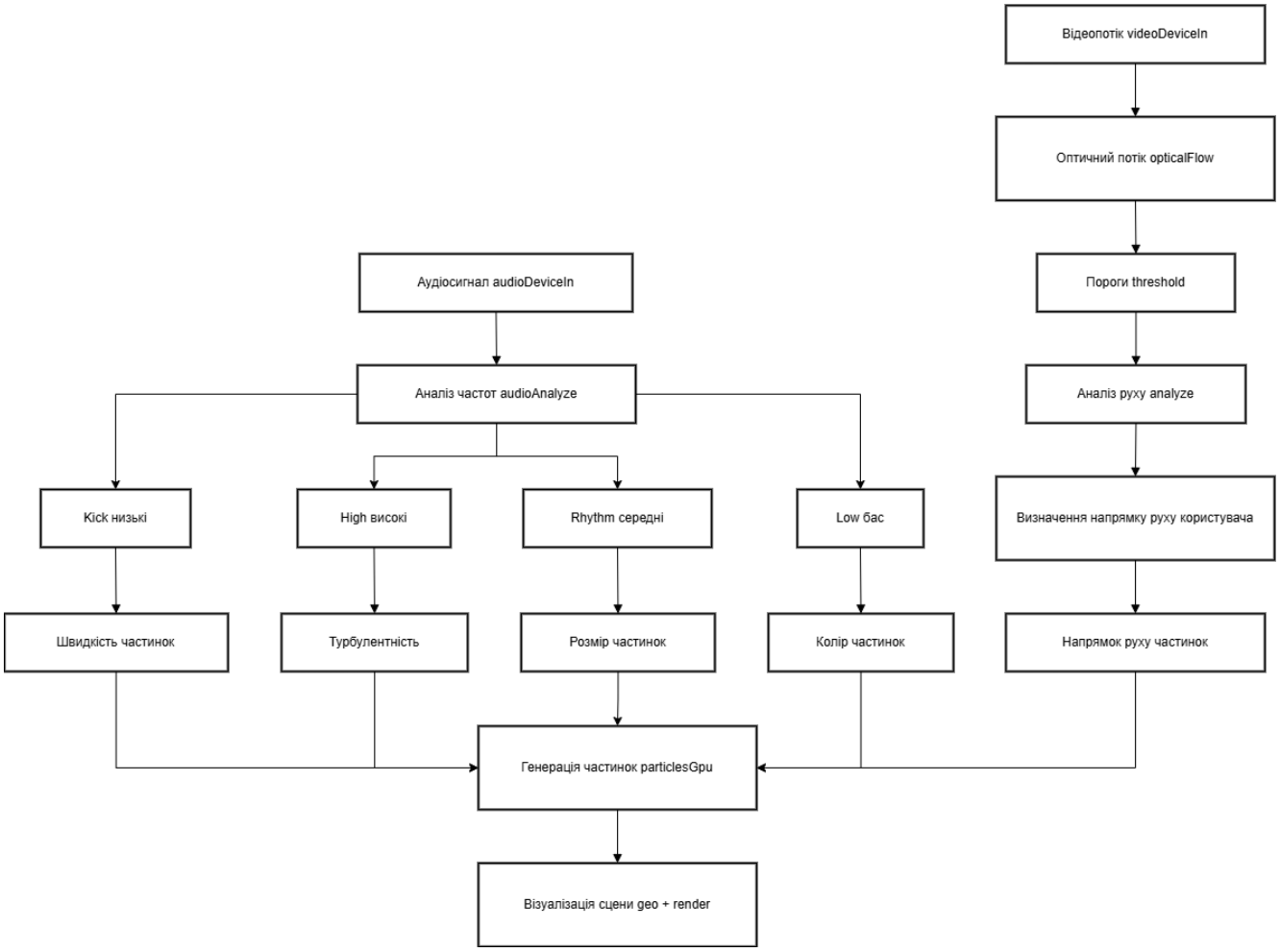


Рис. 0.8 Схема впливу аудіо- та відеоданих на параметри системи частинок

4.4 Реалізація скриптової логіки

У рамках реалізації системи було створено набір кастомних Python-скриптів, які керують логікою взаємодії, реакцією на аудіо- та відеодані, параметрами частинок та зміною сцен. Ці скрипти реалізовані за допомогою DAT Execute, CHOP Execute та Text DAT компонентів TouchDesigner. Вони забезпечують адаптивність і розширюваність проєкту, дозволяючи формувати гнучку поведінкову модель.

Скрипти умовно поділяються на функціональні блоки:

— обробка аудіо/відео сигналів;

— модифікація параметрів частинок;

— логіка перемикання сцен;

— параметризація анімації.

Основні скрипти проєкту:

1) CustSel — обробка вибору користувача.

Цей скрипт відповідає за початкове зчитування та фільтрацію вхідних даних, визначаючи, які параметри наразі активні (наприклад, звук чи рух). На його основі система визначає сценарій подальшої обробки.

Призначення: ініціалізація станів, обробка тригерів аудіо/відео.

2) DatVal — перевірка та нормалізація даних.

Скрипт DatVal виконує функції валідації, обробки меж значень (наприклад, порогових значень) та приведення даних до одного масштабного діапазону. Це критично важливо для коректної передачі значень візуальним модулям.

Призначення: фільтрація шумів, обмеження діапазону, згладжування.

3) GeomGen — генерація геометрії частинок.

Цей скрипт керує параметрами SOP-геометрії, зокрема положенням, розміром та швидкістю частинок. Отримуючи нормалізовані дані з аудіоаналізу та руху, він модифікує властивості геометричних об'єктів у сцені.

Призначення: динамічна геометрія, оновлення позицій, розміру.

4) NoiseGen — генерація шумових патернів.

NoiseGen забезпечує змінні фрактальні або випадкові текстури, що використовуються для створення ефектів турбулентності та хаотичності. Він оновлюється залежно від значень високих частот аудіо.

Призначення: вплив на турбулентність, деструкцію частинок.

5) ImgProc — обробка вхідного відеопотоку.

ImgProc відповідає за всі операції над відеопотоком з камери: обробку оптичного потоку, обчислення напрямку руху та передавання цих значень далі для візуальної реакції.

Призначення: визначення напряму руху користувача.

6) Composite — комбінування графічних шарів.

Composite збирає всі візуальні компоненти, включаючи геометрію, шумові патерни та ефекти від відео, в єдину сцену. Він також виконує останній етап рендерингу перед виводом.

Призначення: об'єднання шарів, фінальна композиція.

7) CMerge — керування вихідним потоком.

Останній у ланцюгу, CMerge забезпечує логіку перемикання сцен, зміну режимів виводу та обробку глобальних подій, як-от ініціалізація чи завершення взаємодії.

Призначення: логіка виходу, управління фінальним виглядом сцени.

В таблиці 4.3 описані вхідні та вихідні дані основних скриптів та їх основні функції.

Таблиця 4.3

Скрипти та їх функції

Назва скрипта	**Основна функція**	**Вхідні дані**	**Вихід/Вплив**
CustSel	Вибір режиму взаємодії	Аудіо/відео	Параметри подій
DatVal	Нормалізація/обмеження значень	Аудіодані	Скориговані параметри
GeomGen	Генерація частинок	Нормалізовані дані	Параметри геометрії
NoiseGen	Шумові патерни	High-фреквенції	TOP-шум/деформація
ImgProc	Обробка руху користувача	Відео	Напрямок руху
Composite	Збірка візуальних шарів	SOP, TOP	Візуальний рендер

Продовження таблиці 4.3

Скрипти та їх функції

Назва скрипта	Основна функція	Вхідні дані	Вихід/Вплив
CMerge	Керування фінальною сценою	Усі попередні	Остаточний вигляд, режими рендеру

Ці скрипти дозволяють створити динамічну, гнучку логіку реакції системи на користувацькі сигнали. Їх реалізація значно підвищує інтерактивність і надає системі "поведінку", яка адаптується до різних ситуацій.

4.5 Візуалізація сцени

Фінальний етап в системі інтерактивної генерації аудіо-візуального контенту — це візуалізація сцени, в якій результат обробки аудіо, відео і логіки реалізується у вигляді графічного зображення. У середовищі TouchDesigner рендеринг здійснюється за допомогою відповідних SOP, TOP та MAT нод, а також шляхом налаштування світла, камери та фінального компонування.

На рисунку 4.9 продемонстрована реалізована структура фінального рендеру сцени.

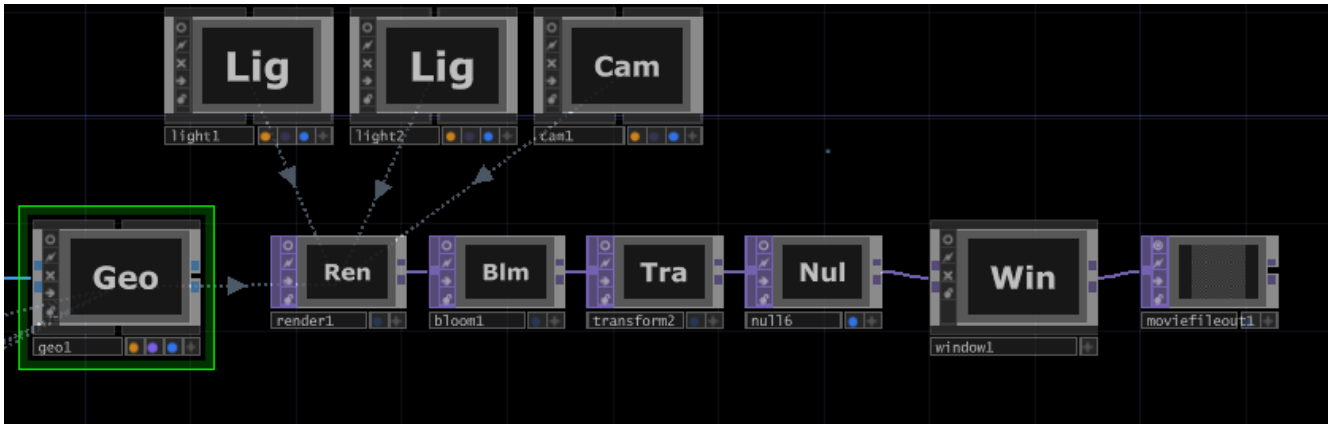


Рис. 0.9 Загальна структура фінального виводу сцени

Основними візуальними компонентами сцени є:

— Geometry COMP — містить 3D об'єкти, які генеруються динамічно залежно від аудіо- та відеоаналізу. Ці об'єкти — частинки, сформовані за допомогою SOP геометрії та оновлювані скриптами;

— Camera COMP — визначає точку зору. У проекті використано перспективну камеру, яка статично спрямована на центр сцени для фокусування на основних змінах;

— Light COMP — відповідає за освітлення сцени. Задано один джерело світла типу Point, розміщене над геометрією для акцентування форми та об’єму частинок;

— Render TOP — здійснює рендеринг всієї сцени в текстуру;

— Composite TOP — фінально об’єднує декілька візуальних шарів, включаючи фон, постобробку та кольорову корекцію.

Використані матеріали

Матеріали (MAT) додають сцені стилістичних властивостей. Для частинок застосовано Phong MAT, з увімкненим shading, reflection та transparency, що забезпечує глибший тривимірний вигляд.

Параметри Phong MAT:

— Diffuse: 0.9;

— Specular: 0.5;

— Transparency: 0.3;

— Reflectivity: 0.1.

Ці значення забезпечують ефект легкості та глибини частинок, дозволяючи бачити накладення та динамічні відтінки при русі.

Додаткові ефекти

Щоб посилити естетику сцени, використано декілька TOP-нод для постобробки:

— Level TOP — для контрасту, яскравості та насиченост;

— Blur TOP — для легкого розмиття заднього плану;

— Edge TOP — підкреслення контурів частинок.

Сам потік рендеру схематично показано на рисунку 4.10.

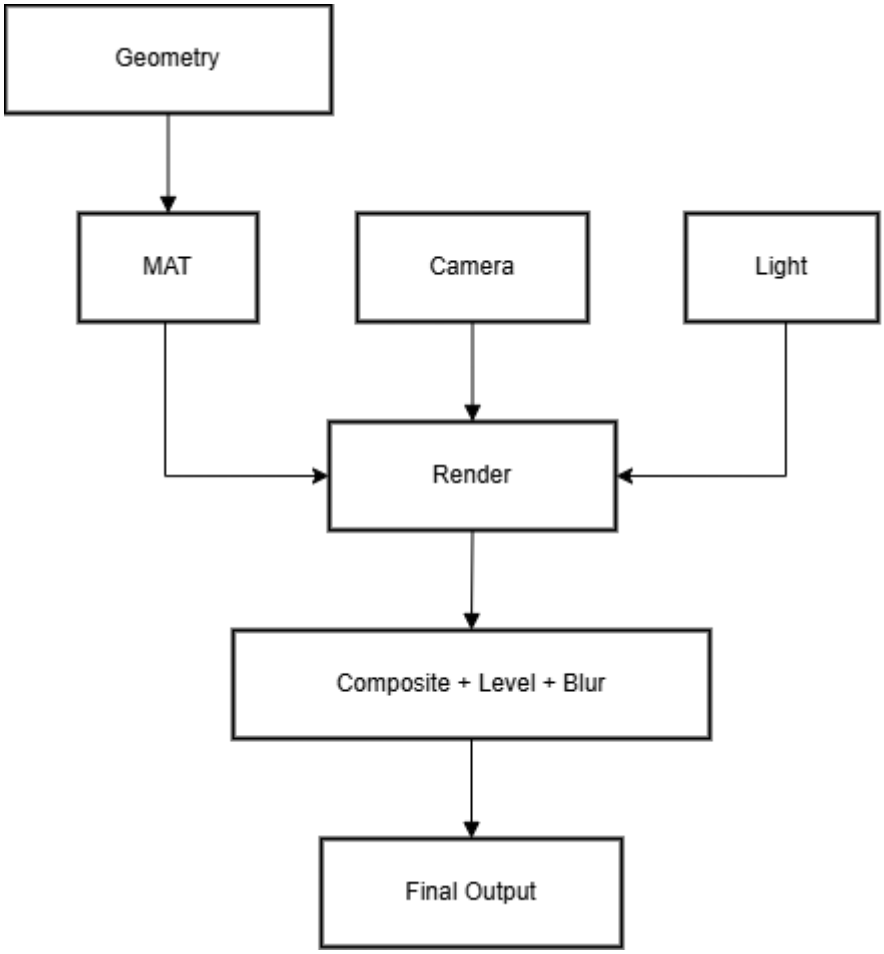


Рис. 0.10 Потік рендерингу

Параметри виводу

Фінальний результат виводиться на екран через Out TOP, з наступними параметрами:

— роздільна здатність: 1920x1080;

— FPS: 60;

— Aspect Ratio: 16:9;

— Pixel Format: RGBA 8-bit;

Ці параметри дозволяють досягти плавної та якісної візуалізації навіть при інтенсивній взаємодії з користувачем.

Візуалізація — це не лише результат графічної обробки, а й ключовий інтерфейс для зворотного зв’язку між системою та користувачем. Кожна зміна у звуковому або руховому сигналі миттєво відображається на візуальному шарі, створюючи ефект “живої” сцени, що реагує в реальному часі. Це підтверджує

важливість злагодженої роботи всіх модулів системи — від аудіоаналізу до рендерингу [5, 6, 13].

В таблиці 4.4 описано призначення та типи компонентів фінального рендеру.

Таблиця 4.4

Компоненти фінального рендеру

Компонент	**Тип**	**Призначення**
Geometry COMP	SOP	Містить об’єкти-частинки
Camera COMP	CAM	Фіксує сцену з однієї перспективи
Light COMP	LIGHT	Освітлення сцени
Phong MAT	MAT	Додає матеріал та ефекти до частинок
Render TOP	TOP	Візуалізація сцени
Composite TOP	TOP	Постобробка: контраст, кольори, blur
Out TOP	TOP	Вивід сцени на екран

Таким чином, реалізація рендерингу в TouchDesigner забезпечує не лише технічну обробку графіки, а й цілісну художню презентацію, що робить сцену гнучкою, інтерактивною та візуально виразною.

4.6 Тестування

Тестування є ключовим етапом розробки інтерактивного програмного забезпечення, оскільки дозволяє перевірити стабільність, коректність і взаємодію всіх модулів системи в умовах реального часу. У межах даного проєкту тестування здійснювалось як модульно, так і інтегровано, з фокусом на аудіо-, відеоаналіз, скриптову логіку та рендеринг візуального контенту.

Цілі тестування

Основними цілями тестування були:

— перевірка коректності обробки аудіосигналу;

— перевірка реакції на рух користувача в кадрі;

— перевірка відповідності змін параметрів частинок до вхідних даних;

— оцінка продуктивності системи при тривалому використанні;

— виявлення можливих затримок чи збоїв у візуалізації.

Тестування проводилося на наступному апаратному забезпеченні:

— ОС: Windows 11 x64;

— CPU: AMD Ryzen 7 5800H;

— GPU: NVIDIA GeForce RTX 3060 (6 GB VRAM);

— RAM: 16 GB DDR4;

— TouchDesigner: 2023.11310 (Commercial License).

Система стабільно працювала в реальному часі з середньою частотою кадрів ~40 FPS.

Методика тестування

Тестування проводилося вручну у трьох сценаріях:

1. Аудіоактивність без відеовходу — для перевірки модуля аудіоаналізу.

2. Рух користувача без аудіо — тестування реакції частинок на рух.

3. Сумісна активність — одночасне відстеження звуку і руху для повного інтерактивного досвіду.

Сценарії тестування разом з очікуваннями та результатами описано в таблиці 4.5.

Таблиця 4.5

Сценарії тестування

№	**Сценарій**	**Очікувана поведінка**	**Результат**
1	Тест аудіо без відео	Зміна параметрів відповідно до звуку	успішно

Продовження таблиці 4.5

Сценарії тестування

№	**Сценарій**	**Очікувана поведінка**	**Результат**
2	Тест відео без аудіо	Частинки рухаються відповідно до руху користувача	успішно
3	Комбінована взаємодія	Всі параметри частинок змінюються одночасно	успішно
4	Тест із фоновим шумом	Система не має реагувати на незначний шум	успішно
5	Надмірна активність (стрес)	Частота кадрів не має падати нижче 30 FPS	успішно

Виявлені проблеми та їх вирішення

У процесі тестування були виявлені незначні затримки при одночасному обробленні великих обсягів даних. Це було усунено шляхом оптимізації CHOP-мереж, зменшення кількості SOP-полігонів, а також увімкнення режиму "Cook Every Frame Off" для неключових нод.

Також було виявлено, що при низькому освітленні камера могла неадекватно інтерпретувати рухи користувача. Для вирішення застосовано додаткову обробку зображення через threshold TOP перед надсиланням на opticalFlow [2].

ВИСНОВКИ

Результатом виконаної роботи стала оптимізація процесу створення інтерактивного аудіо-візуального контенту з використанням методів генеративної графіки з використанням.

У процесі виконання роботи були реалізовані такі задачі:

Проаналізовано предметну область: визначено сутність інтерактивних аудіо-візуальних систем, охарактеризовано їхні можливості, напрями застосування в цифровому мистецтві та ключові технології. Були досліджені аналогічні реалізації та визначено переваги використання TouchDesigner у поєднанні з Python для творчих задач генеративної візуалізації.

Обрано засоби реалізації: TouchDesigner як середовище для графічного програмування візуального контенту, Python як основна мова скриптової логіки, OpenCV, NumPy та Pandas як допоміжні бібліотеки для аналізу даних, GitHub — для контролю версій, а також візуальні діаграми та схеми для планування архітектури проєкту.

Спроєктовано архітектуру системи: створено структурну схему клієнтської частини з модулями аудіоаналізу, відстеження руху, генерації частинок, скриптової логіки та фінального рендерингу. Було спроєктовано логіку взаємодії частотних діапазонів та руху користувача з параметрами графічної сцени.

Реалізовано систему у середовищі TouchDesigner: реалізовано обробку аудіо через audioDeviceIn, audioAnalyze та серію CHOP-нод для керування параметрами частинок. Впроваджено відеоаналіз через opticalFlow та логіку руху за допомогою скриптів. Побудовано модуль генерації частинок з параметрами, що змінюються у реальному часі, а також реалізовано 7 скриптів на Python, що забезпечують взаємодію та логічну обробку вхідних даних.

Проведено тестування системи: виконано серію тестів у різних режимах взаємодії. Система показала стабільну роботу з високим FPS, правильну реакцію на аудіо- та відеовхід, а також ефективну обробку даних з боку скриптової логіки.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Погорілий О.А., Соляник Л.О. Методи оптимізації використання нодової системи TouchDesigner для створення інтерактивного аудіо-візуального контенту. Інтелектуальний аналіз та обробка інформації: Матеріали VI Всеукраїнської науково-технічної конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”. 24.04.2025, ДУІКТ, Київ, Україна, с. 592-595.

2. Погорілий О.А., Соляник Л.О. Основні шляхи гарантування безпеки для системи інтерактивної генерації аудіо-візуального контенту. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених “Інформаційні технології - 2025”, 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 312-313.

ПЕРЕЛІК ПОСИЛАНЬ

1. TouchDesigner Documentation – Derivative [Електронний ресурс]. – Режим доступу: https://docs.derivative.ca (дата звернення: 25.05.2025).
2. TouchDesigner Python API Reference [Електронний ресурс]. – Режим доступу: https://docs.derivative.ca/Python (дата звернення: 25.05.2025).
3. Бертол В. Динамічна візуалізація у генеративному мистецтві // Комп'ютерна графіка та візуалізація. – 2020. – № 3(28). – С. 15–22.
4. Шиффман Д. Природа коду: візуалізація, симуляція та генеративні системи. – Нью-Йорк: Nature of Code Press, 2012. – 324 с. URL: https://natureofcode.com/book/ (дата звернення: 25.05.2025).
5. Рінальдо К. Інтерактивні системи та залучення глядача // Journal of New Media Art. – 2018. – № 1. – С. 42–50.
6. NumPy Documentation [Електронний ресурс]. – Режим доступу: https://numpy.org/doc/ (дата звернення: 26.05.2025).
7. OpenCV Python Tutorials [Електронний ресурс]. – Режим доступу: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html (дата звернення: 26.05.2025).
8. GitHub Docs. Introduction to GitHub [Електронний ресурс]. – Режим доступу: https://docs.github.com/en/get-started (дата звернення: 26.05.2025).
9. Айзенштейн С. М. Мистецтво рухомого зображення: збірка лекцій. – Харків: Мистецтво, 2017. – 416 с.
10. Мельник С. І., Яковенко П. В. Основи комп’ютерної графіки. – Київ: Ліра-К, 2021. – 296 с.
11. Гладкий В. І. Інтерактивні системи візуалізації даних. – Львів: ЛНУ ім. І. Франка, 2020. – 212 с.
12. Haeberli P. Paint by numbers: abstract image representations // SIGGRAPH Comput. Graph. – 1990. – Vol. 24, No. 4. – P. 207–214.

13. McCormack J. Open Problems in Evolutionary Music and Art // EvoMUSART. – Springer, 2012. – P. 428–439.

14. Rankin L. Designing for Live Visuals: A Guide to VJing and Real-Time Projection. – London: Routledge, 2020. – 238 p.

15. Ковальчук І. О. Мультимедійні технології: навч. посіб. – Київ: Кондор, 2021. – 320 с.

16. Сущенко А. А. Основи цифрової обробки зображень. – Харків: ХНУРЕ, 2019. – 180 с.

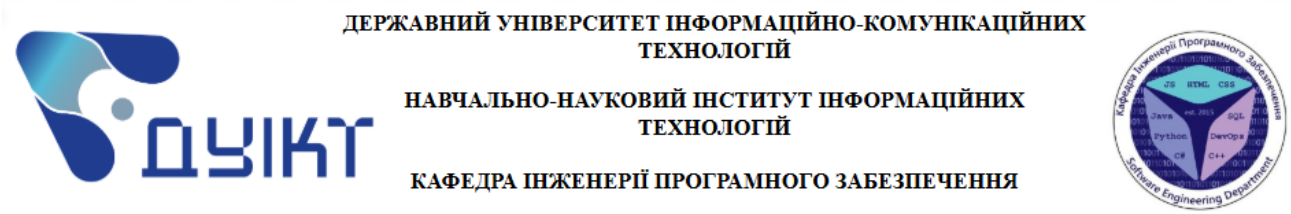
17. Salen K., Zimmerman E. Rules of Play: Game Design Fundamentals. – MIT Press, 2003. – 672 p.

18. Матвійчук А. В. Основи візуального програмування. – Чернівці: Технодрук, 2021. – 144 с.

19. Генеративна естетика: теорія і практика цифрового мистецтва / За ред. О. Б. Кириленка. – Київ: Академія, 2022. – 250 с.

20. Хейман Б. Інтерактивні інсталяції та алгоритмічні композиції // Сучасне мистецтво. – 2023. – № 14. – С. 88–96.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



Розробка програмного забезпечення для інтерактивної генерації аудіо-візуального контенту

Виконав студент 4 курсу
групи ПД-41
Погорілий Олександр Анатолійович
Керівник роботи
Д.х.н, доцент, доцент кафедри ІПЗ Соляник Людмила Олексіївна

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи**: оптимізувати процес створення інтерактивного контенту з використанням методів генеративної графіки.
- **Об'єкт дослідження:** генерація візуального контенту на основі звукових параметрів та сенсорної взаємодії.
- **Предмет дослідження**: програмне забезпечення для генерації інтерактивного контенту у реальному часі.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати існуючі підходи до генерації аудіо-візуального контенту в реальному часі
2. Дослідити можливості TouchDesigner для обробки аудіо та відеоданих і взаємодії з Python
3. Розробити концептуальну архітектуру програмного рішення
4. Реалізувати систему інтерактивної генерації візуального контенту
5. Інтегрувати Python для оптимізації поведінки системи та логіки реакцій
6. Забезпечити стабільну роботу системи в режимі реального часу

3

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

- Програма повинна здійснювати аналіз аудіосигналу в реальному часі.
- На основі аудіоаналізу має відбуватись генерація візуального контенту у вигляді та графічних частинок, що змінюються відповідно до звукових характеристик.
- Система повинна реагувати на жести або рухи користувача, отримані через сенсори, змінюючи параметри візуалізації.
- Програма повинна забезпечувати синхронну роботу аудіоаналізу та візуальної генерації без ручного втручання під час виконання.

Нефункціональні вимоги:

- Висока стабільність роботи в режимі реального часу(30хв+).
- Мінімальна затримка між обробкою аудіо та візуальним відгуком(до 100 мс), та між обробкою жестів/рухів(до 150мс).
- Оптимізована продуктивність для плавної графіки (в ідеалі - 60 fps, допустимо - 30 fps)

4

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

TouchDesigner

pandas

NumPy

OpenCV

5

Алгоритм роботи

6

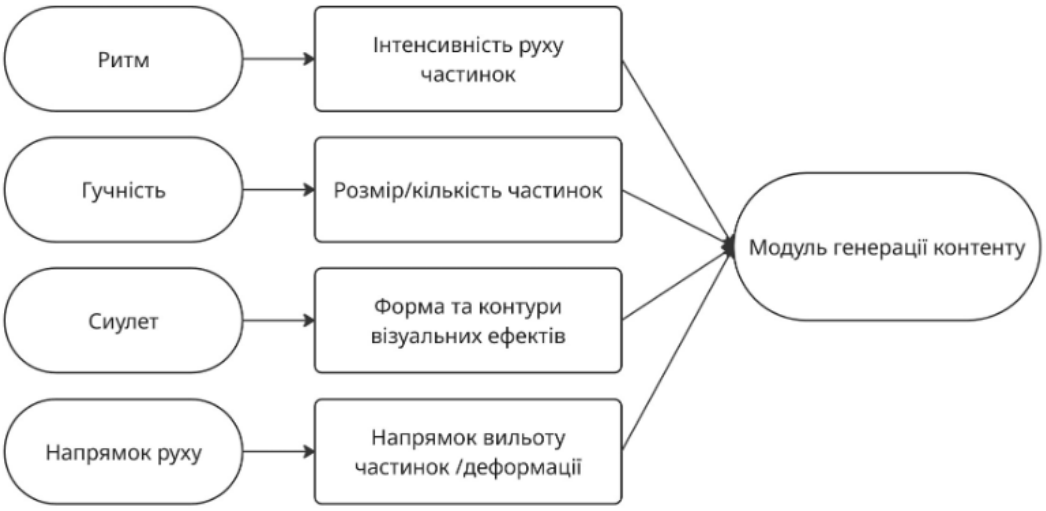
Робота аналізу аудіо та відео жестів

7

Процес генерації частинок

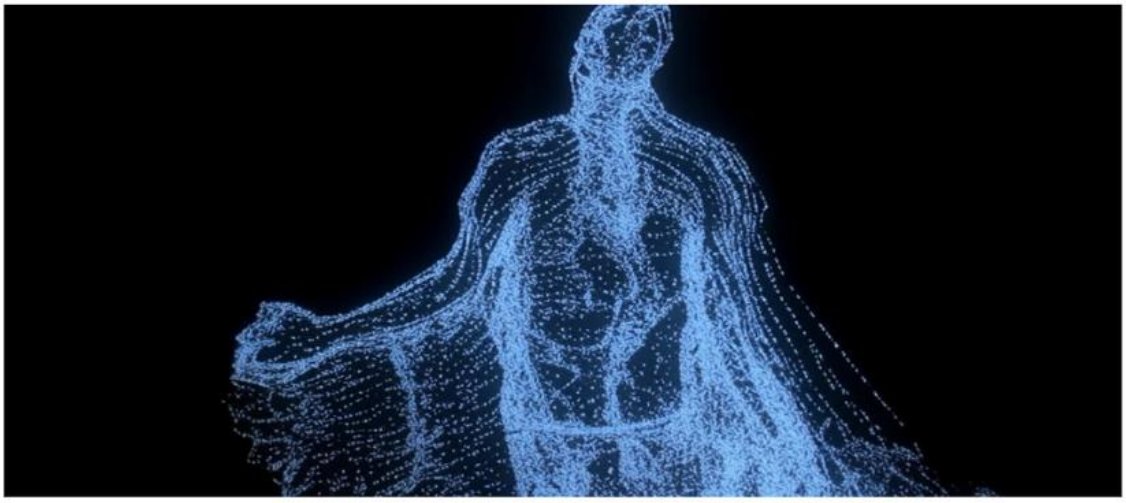
8

Взаємозв'язок параметрів генерації



9

ЕКРАННІ ФОРМИ



Головний екран

10

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ

11

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Погорілий О.А., Соляник Л.О. Методи оптимізації використання нодової системи TouchDesigner для створення інтерактивного аудіо-візуального контенту. Інтелектуальний аналіз та обробка інформації: Матеріали VI Всеукраїнської науково-технічної конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”. 24.04.2025, ДУІКТ, Київ, Україна, с. 592-595

2. Погорілий О.А., Соляник Л.О. Основні шляхи гарантування безпеки для системи інтерактивної генерації аудіо-візуального контенту. Матеріали XII Всеукраїнської науково-практичної конференції молодих учених “Інформаційні технології - 2025”, 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с. 312-313.

12

ВИСНОВКИ

1. Проаналізовано існуючі підходи до генерації аудіо-візуального контенту.
2. Досліджено середовище TouchDesigner, зосереджено увагу на обробці відео, обробці аудіо і можливостях Python для керування логікою.
3. Сформовано концептуальну архітектуру системи, яка об’єднує три канали: відео, аудіо, і жести.
4. Реалізовано інтерактивну систему генерації візуального контенту
5. Інтегровано Python для створення логіки поведінки візуальних ефектів.
6. Досягнуто стабільної роботи системи в режимі реального часу.

13

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

Cmerge:

```
def onCook(scriptOp):
    # Перевірка наявності вхідних SOP
    if not scriptOp.inputs:
        debug('No input SOPs connected')
        return
    input_sops = scriptOp.inputs

    # Очищення вихідного SOP
    scriptOp.clear()

    # Об'єднання вершин
    all_points = []
    for sop in input_sops:
        all_points.extend(sop.points)
    for point in all_points:
        new_point = scriptOp.appendPoint()
        new_point.x = point.x
        new_point.y = point.y
        new_point.z = point.z

    # Об'єднання примітивів
    all_prims = []
    point_offset = 0
    for sop in input_sops:
        for prim in sop.prims:
            new_prim = scriptOp.appendPrim(prim.type)
            for vert in prim.verts:
                new_vert = new_prim.appendVert()
                new_vert.point = all_points.index(vert.point) +
point_offset
            all_prims.append(new_prim)
        point_offset += len(sop.points)

    # Логування
    debug(f'Merged {len(all_points)} points and
{len(all_prims)} primitives')
    return
```

Composite:

```
import numpy as np

def onSetupParameters(scriptOp):
    page = scriptOp.appendCustomPage('Composite
Parameters')
    p = page.appendMenu('BlendMode', label='Blend
Mode', menuItems=['Over', 'Add', 'Subtract', 'Multiply',
'Screen'])
    return

def onCook(scriptOp):
    # Перевірка наявності двох вхідних TOP
    if len(scriptOp.inputs) < 2:
        debug('Need at least two input TOPs')
        return
    top_a = scriptOp.inputs[0]
    top_b = scriptOp.inputs[1]

    # Отримання піксельних даних
    width = top_a.width
    height = top_a.height
    pixels_a = top_a.pixels
    pixels_b = top_b.pixels

    # Отримання режиму змішування
    blend_mode = scriptOp.par.BlendMode.eval()

    # Композитинг пікселів
    result = np.zeros((height, width, 4), dtype=np.float32)
    if blend_mode == 'Over':
        # Стандартне альфа-змішування
        for y in range(height):
            for x in range(width):
                alpha_a = pixels_a[y, x, 3]
                alpha_b = pixels_b[y, x, 3]
                result[y, x, :3] = (pixels_a[y, x, :3] * alpha_a +
pixels_b[y, x, :3] * alpha_b * (1 - alpha_a)) / (alpha_a +
alpha_b - alpha_a * alpha_b)
                result[y, x, 3] = alpha_a + alpha_b - alpha_a *
alpha_b
    elif blend_mode == 'Add':
        result = np.clip(pixels_a + pixels_b, 0, 1)
    elif blend_mode == 'Subtract':
        result = np.clip(pixels_a - pixels_b, 0, 1)
    elif blend_mode == 'Multiply':
        result = pixels_a * pixels_b
    elif blend_mode == 'Screen':
        result = 1 - (1 - pixels_a) * (1 - pixels_b)
    else:
        debug('Unknown blend mode')
        return

    # Копіювання результату
    scriptOp.copyNumpyArray(result)

    # Логування
    debug(f'Composited with blend mode:
{blend_mode}')
    return
```

CustSel:

```
import tdu

def onSetupParameters(scriptOp):
    page = scriptOp.appendCustomPage('Select
Parameters')
    p = page.appendString('Channels', label='Channels',
numRows=5)
    p = page.appendInt('StartSample', label='Start
Sample')
    p = page.appendInt('EndSample', label='End Sample')
    return

def onCook(scriptOp):
    # Перевірка наявності вхідного CHOP
    if not scriptOp.inputs:
        debug('No input CHOP connected')
        return
    input_chop = scriptOp.inputs[0]

    # Отримання параметрів
    channels_str = scriptOp.par.Channels.eval()
    start_sample = scriptOp.par.StartSample.eval()
    end_sample = scriptOp.par.EndSample.eval()

    # Валідація параметрів
    if start_sample < 0 or end_sample < 0 or start_sample
> end_sample:
        debug('Invalid sample range')
        return

    # Розбір вибраних каналів
    selected_channels = [chan.strip() for chan in
channels_str.split(',') if chan.strip()]
    if not selected_channels:
        debug('No channels selected')
        return

    # Очищення вихідного CHOP
    scriptOp.clear()

    # Створення вихідних каналів
    for chan_name in selected_channels:
        if chan_name in input_chop.chans:
            chan = input_chop.chans[chan_name]
            new_chan = scriptOp.appendChan(chan_name)
            for i in range(max(0, start_sample),
min(len(chan.vals), end_sample + 1)):
                new_chan[i] = chan.vals[i]
        else:
            debug(f'Channel {chan_name} not found in input
CHOP')

    # Логування
    debug(f'Selected channels: {selected_channels},
samples {start_sample} to {end_sample}')
    return
```

DatVal:

```
def validate_numeric_data(data, min_val, max_val):
    issues = []
    for i, val in enumerate(data):
        try:
            num = float(val)
            if not (min_val <= num <= max_val):
                issues.append(f"Value {num} at index {i} out
of range [{min_val}, {max_val}]")
        except ValueError:
            issues.append(f"Invalid number at index {i}:
{val}")
    return issues

def validate_channel_data(chop, min_val, max_val):
    issues = []
    for chan in chop.chans:
        for i, val in enumerate(chan.vals):
            if not (min_val <= val <= max_val):
                issues.append(f"Channel {chan.name} value
{val} at index {i} out of range")
    return issues

def onSetupParameters(scriptOp):
    page = scriptOp.appendCustomPage('Validation
Parameters')
    page.appendFloat('MinValue', label='Min Value')
    page.appendFloat('MaxValue', label='Max Value')
    page.appendStr('LogFile', label='Log File Path')
    return

def onCook(scriptOp):
    min_val = scriptOp.par.MinValue.eval()
    max_val = scriptOp.par.MaxValue.eval()
    log_file = scriptOp.par.LogFile.eval() or
'validation_log.txt'
    issues = []

    # Перевірка вхідних даних
    if scriptOp.inputs and isinstance(scriptOp.inputs[0],
tdu.Chop):

issues.extend(validate_channel_data(scriptOp.inputs[0],
min_val, max_val))
    elif scriptOp.inputs and isinstance(scriptOp.inputs[0],
tdu.Dat):
        data = scriptOp.inputs[0].text.split()
        issues.extend(validate_numeric_data(data, min_val,
max_val))

    # Логування
```

```
    with open(log_file, 'a') as f:
        for issue in issues:
            f.write(f"{issue}\n")
    debug(f"Validation completed with {len(issues)}
issues")
    return
```

DUscript:

```
def valueChange(channel, sampleIndex, val, prev):
    if val == 0:
        op('constant2').par.value0 = -0.2
    elif val == 1:
        op('constant2').par.value0 = 0.4
    elif val == -1:
        op('constant2').par.value0 = -0.4
    return
```

GeomGen:

```
def create_grid_points(scriptOp, rows, cols, spacing):
    points = []
    for r in range(rows):
        for c in range(cols):
            p = scriptOp.appendPoint()
            p.x = c * spacing - (cols - 1) * spacing / 2
            p.y = r * spacing - (rows - 1) * spacing / 2
            p.z = 0
            points.append(p)
    return points

def create_grid_polys(scriptOp, points, rows, cols):
    for r in range(rows - 1):
        for c in range(cols - 1):
            poly = scriptOp.appendPoly(4, closed=True)
            poly[0].point = points[r * cols + c]
            poly[1].point = points[r * cols + c + 1]
            poly[2].point = points[(r + 1) * cols + c + 1]
            poly[3].point = points[(r + 1) * cols + c]

def create_sphere_points(scriptOp, radius, segments):
    points = []
    for i in range(segments + 1):
        theta = i * math.pi / segments
        for j in range(segments):
            phi = j * 2 * math.pi / segments
            p = scriptOp.appendPoint()
            p.x = radius * math.sin(theta) * math.cos(phi)
            p.y = radius * math.sin(theta) * math.sin(phi)
            p.z = radius * math.cos(theta)
            points.append(p)
    return points

def create_sphere_polys(scriptOp, points, segments):
    for i in range(segments):
        for j in range(segments):
            poly = scriptOp.appendPoly(4, closed=True)
            idx = i * segments + j
            poly[0].point = points[idx]
            poly[1].point = points[idx + 1 if j < segments - 1
else idx - segments + 1]
            poly[2].point = points[idx + segments + 1 if j <
segments - 1 else idx + 1]
            poly[3].point = points[idx + segments]

def onSetupParameters(scriptOp):
    page = scriptOp.appendCustomPage('Geometry
Parameters')
    page.appendInt('Rows', label='Rows')
    page.appendInt('Cols', label='Columns')
    page.appendFloat('Spacing', label='Spacing')
    page.appendFloat('Radius', label='Radius')
    page.appendInt('Segments', label='Segments')
    page.appendMenu('ShapeType', label='Shape Type',
menuItems=['Grid', 'Sphere'])
    return

def onCook(scriptOp):
    scriptOp.clear()
    shape_type = scriptOp.par.ShapeType.eval()
    if shape_type == 'Grid':
        rows = max(2, scriptOp.par.Rows.eval())
        cols = max(2, scriptOp.par.Cols.eval())
        spacing = scriptOp.par.Spacing.eval()
        points = create_grid_points(scriptOp, rows, cols,
spacing)
        create_grid_polys(scriptOp, points, rows, cols)
    else:
        radius = scriptOp.par.Radius.eval()
        segments = max(4, scriptOp.par.Segments.eval())
        points = create_sphere_points(scriptOp, radius,
segments)
        create_sphere_polys(scriptOp, points, segments)
    debug(f"Generated {shape_type} geometry with
{len(scriptOp.points)} points")
    return
```

HIGH:

```
def valueChange(channel, sampleIndex, val, prev):
    # Згладжування (lag)
    prev_val = op('const_lag4').par.value0.eval()
    smoothed = prev_val + (val - prev_val) * 0.3

    # Обмеження (limit)
    clamped = max(-0.12, min(0.12, smoothed))

    op('HIGH').par.value0 = round(clamped, 4)
```

ImgProc:

```
import numpy as np

def gaussian_kernel(size, sigma):
    kernel = np.zeros((size, size), dtype=np.float32)
    center = size // 2
```

```
    for x in range(size):
        for y in range(size):
            dx = x - center
            dy = y - center
            kernel[x, y] = np.exp(-(dx**2 + dy**2) / (2 *
sigma**2))
    return kernel / np.sum(kernel)

def apply_blur(pixels, kernel):
    height, width, channels = pixels.shape
    result = np.zeros_like(pixels)
    ksize = kernel.shape[0]
    pad = ksize // 2
    padded = np.pad(pixels, ((pad, pad), (pad, pad), (0,
0)), mode='edge')
    for y in range(height):
        for x in range(width):
            for c in range(channels):
                result[y, x, c] = np.sum(padded[y:y+ksize,
x:x+ksize, c] * kernel)
    return result

def onSetupParameters(scriptOp):
    page = scriptOp.appendCustomPage('Image
Processing')
    page.appendFloat('BlurSigma', label='Blur Sigma')
    page.appendInt('KernelSize', label='Kernel Size')
    page.appendMenu('FilterType', label='Filter Type',
menuItems=['Blur', 'Edge'])
    return

def onCook(scriptOp):
    width = scriptOp.width
    height = scriptOp.height
    pixels = np.zeros((height, width, 4), dtype=np.float32)
    filter_type = scriptOp.par.FilterType.eval()

    if filter_type == 'Blur':
        sigma = scriptOp.par.BlurSigma.eval()
        ksize = max(3, scriptOp.par.KernelSize.eval() | 1)  #
Непарне
        kernel = gaussian_kernel(ksize, sigma)
        pixels = apply_blur(pixels, kernel)
    else:
        pixels[:, :, :3] = 1.0  # Простий ефект для
демонстрації

    scriptOp.copyNumpyArray(pixels)
    debug(f"Applied {filter_type} filter with
sigma={sigma}")
    return
```

KICK:

```
def valueChange(channel, sampleIndex, val, prev):
    prev_val = op('const_lag5').par.value0.eval()
    smoothed = prev_val + (val - prev_val) * 0.1
    clamped = max(0, min(1, smoothed))
    op('KICK').par.value0 = round(clamped, 4)
```

KICK1:

```
def valueChange(channel, sampleIndex, val, prev):
    prev_val = op('const_lag6').par.value0.eval()
    smoothed = prev_val + (val - prev_val) * 0.2
    clamped = max(0, min(0.5, smoothed))
    op('KICK1').par.value0 = round(clamped, 4)
```

LRscript:

```
def valueChange(channel, sampleIndex, val, prev):
    if val == 0:
        op('constant1').par.value0 = -0.02
    elif val == 1:
        op('constant1').par.value0 = 0.1
    elif val == -1:
        op('constant1').par.value0 = -0.1
    return
```

NoiseGen:

```
import numpy as np
import random
import math

# Клас для генерації Perlin-шуму
class PerlinNoise:
    def __init__(self, seed=0):
        random.seed(seed)
        self.permutation = [i for i in range(256)]
        random.shuffle(self.permutation)
        self.permutation += self.permutation

    def fade(self, t):
        return t * t * t * (t * (t * 6 - 15) + 10)

    def lerp(self, t, a, b):
        return a + t * (b - a)

    def grad(self, hash, x, y):
        h = hash & 15
        u = x if h < 8 else y
        v = y if h < 4 else x
        return (u if (h & 1) == 0 else -u) + (v if (h & 2) == 0
else -v)

    def noise(self, x, y):
        X = int(x) & 255
        Y = int(y) & 255
        x -= int(x)
        y -= int(y)
        u = self.fade(x)
        v = self.fade(y)
        A = self.permutation[X] + Y
        B = self.permutation[X + 1] + Y
```

```
        return self.lerp(v, self.lerp(u,
self.grad(self.permutation[A], x, y),
                                 self.grad(self.permutation[B], x -
1, y)),
                         self.lerp(u, self.grad(self.permutation[A
+ 1], x, y - 1),
                                 self.grad(self.permutation[B + 1],
x - 1, y - 1)))

def onSetupParameters(scriptOp):
    page = scriptOp.appendCustomPage('Noise
Parameters')
    page.appendFloat('Amplitude', label='Amplitude')
    page.appendFloat('Frequency', label='Frequency')
    page.appendInt('Octaves', label='Octaves')
    page.appendInt('Seed', label='Seed')
    page.appendMenu('NoiseType', label='Noise Type',
menuItems=['Perlin', 'Simplex', 'Random'])
    return

def onCook(scriptOp):
    scriptOp.clear()
    num_samples = scriptOp.numSamples
    amplitude = scriptOp.par.Amplitude.eval()
    frequency = scriptOp.par.Frequency.eval()
    octaves = max(1, scriptOp.par.Octaves.eval())
    seed = scriptOp.par.Seed.eval()
    noise_type = scriptOp.par.NoiseType.eval()

    chan = scriptOp.appendChan('noise')
    perlin = PerlinNoise(seed)

    for i in range(num_samples):
        value = 0.0
        if noise_type == 'Perlin':
            for o in range(octaves):
                scale = 2 ** o
                value += perlin.noise(i * frequency * scale /
num_samples, 0) / scale
        elif noise_type == 'Simplex':
            value = random.random()  # Замість Simplex для
спрощення
        else:
            value = random.random()
        chan[i] = value * amplitude

    # Логування для дебагінгу
    debug(f"Generated noise: type={noise_type},
amplitude={amplitude}, samples={num_samples}")
    return
```

PerfMon:

```
import time
import psutil

def get_fps():
    return 1.0 / max(0.001,
op.TDPerformance.frameTime)

def get_memory_usage():
    process = psutil.Process()
    return process.memory_info().rss / (1024 * 1024)  #
MB

def get_cpu_usage():
    return psutil.cpu_percent(interval=0.1)

def onSetupParameters(scriptOp):
    page = scriptOp.appendCustomPage('Monitoring
Parameters')
    page.appendFloat('UpdateInterval', label='Update
Interval (s)')
    page.appendStr('LogFile', label='Log File Path')
    page.appendInt('FpsThreshold', label='FPS Threshold')
    return

def onCook(scriptOp):
    interval = scriptOp.par.UpdateInterval.eval()
    log_file = scriptOp.par.LogFile.eval() or
'performance_log.txt'
    fps_threshold = scriptOp.par.FpsThreshold.eval()

    fps = get_fps()
    memory = get_memory_usage()
    cpu = get_cpu_usage()

    log_entry = f"Time: {time.ctime()}, FPS: {fps:.2f},
Memory: {memory:.2f} MB, CPU: {cpu:.2f}%"
    if fps < fps_threshold:
        log_entry += " [WARNING: Low FPS]"

    with open(log_file, 'a') as f:
        f.write(f"{log_entry}\n")

    scriptOp.text = log_entry
    debug(f"Performance metrics: {log_entry}")
    return
```

RHYTHM:

```
def valueChange(channel, sampleIndex, val, prev):
    prev_val = op('const_lag7').par.value0.eval()
    smoothed = prev_val + (val - prev_val) * 0.3
    clamped = max(0.05, min(0.06, smoothed))
    op('RYTHM').par.value0 = round(clamped, 5)
```