МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра комп'ютерної інженерії

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри,
д-р техн. наук, проф.

_____ І. М. Журавська

« __ » _____ 2024 р.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

Графічний фреймворк для рекламних дисплеїв

Спеціальність 123 «Комп'ютерна інженерія»

123 – КПР.ПЗ.01 – 605.21810510

Студент _____А. Г. Качанов
підпис
«__» _____ 202__ р.

Керівник доцент кафедри КІ, канд. техн. наук, доцент _____ В. Ю. Савінов
підпис
«__» _____ 202__ р.

Миколаїв – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ASCII– American Standard Code for Information Interchange

BPI – Bytes Per Inch

CAD – Computer Aided Design

CAE – Computer Aided Engineering

DDL – Data Definition Language

DEC – Digital Equipment Corporation

EBCDIC– Extended Binary Coded Decimal Interchange Code

EPROM – Erasable Programmable Read-Only Memory

GSM – Global System for Mobile communication

MIPS– Millions of Instructions Per Second

RAM – Random Access Memory

UV – відповідність між координатами на поверхні тривимірного об'єкту (X, Y, Z) і координатами на текстурі (U, V).

VFX – Visual effects

Гб – Гігабайт

ОС – Операційна система

ПЗ – Програмне забезпечення

Вступ

У сучасному світі реклама стала неодмінною складовою нашого повсякденного життя. Вона проникає в усі сфери нашого існування, використовуючи різноманітні канали масової комунікації для трансляції свого повідомлення. Зокрема, реклама поширюється через телебачення, пресу (газети, журнали), радіо, Інтернет, соціальні медіа, прямі продажі, акції, розсилки, події, спонсорство, зображення, звуки та інші елементи.

Рекламна індустрія складається з компаній, агентств, медіа-компаній та фахівців, які спільно працюють над створенням та розповсюдженням рекламних матеріалів. Це включає в себе копірайтерів, дизайнерів, маркетологів, аналітиків та інших спеціалістів, які забезпечують ефективність та розкриття потенційної аудиторії.

Розробка графічного фреймворку може стати новим етапом у розвитку рекламної індустрії. Інтеграція технологій штучного інтелекту, інтерактивних рішень та високоякісного візуального вмісту може принести нові можливості для ефективного спілкування з аудиторією та залучення уваги до брендів і продуктів. Таким чином, розробка програмно-апаратних комплексів може внести істотний внесок у сучасну рекламну індустрію, відкривши нові шляхи для взаємодії між брендами та споживачами.

Мета: розроблення графічного фреймворку для рекламних дисплеїв з використанням Arduino та бібліотеки stdio.h.

Об’єкт: відображення якісних рекламних компаній засобами мікроконтролерних модулів на базі Arduino.

Предмет: алгоритми рендерингу за допомогою графічного фреймворку для рекламних дисплеїв.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

- у аналітичному огляді проаналізувати існуючі види реклами та виявити їх переваги та недоліки;

- обґрунтувати вибір складових, комплектуючих та технологій для розроблення;
- розробити схеми алгоритму роботи пристрою зовнішньої реклами;
- розробити апаратне та програмне забезпечення приладу для показу рекламного повідомлення ;
- проаналізувати ефективність роботи готового графічного фреймворку;
- оцінити перспективи подальших розроблень та удосконалень;
- розробити питання з охорони праці та безпеки життєдіяльності.

Практичне значення отриманих результатів роботи апаратно-програмного комплексу мають значуще практичне значення в контексті розвитку зовнішньої реклами та маркетингу. Створений апаратний комплекс відкриває нові можливості для впровадження інноваційних підходів у сфері реклами, допомагаючи підвищити ефективність маркетингових заходів.

Завдяки характеристикам Arduino, комплекс забезпечує високий рівень взаємодії, що робить рекламу більш привабливою для аудиторії. Використання апаратно-програмного комплексу дозволяє ефективно виконувати різноманітні завдання, що стає найбільш помітною перевагою. Такий комплекс здатен показувати рекламу без зайвих затримок, що сприяє збільшенню зацікавленості аудиторії та покращенню взаємодії з нею.

Соціальний аспект реклами значно змінився за останні роки, і розроблений комплекс дозволяє ефективно відповідати на ці зміни. Він пропонує зручні та сучасні способи показу реклами, що відображається у браузерах та на моніторах, що сприяє підвищенню уваги до реклами та полегшує її сприйняття аудиторією.

У цілому, розроблений програмно-апаратний комплекс відкриває нові можливості для ефективної реклами та маркетингу, сприяючи покращенню комунікації між брендами та споживачами, а також забезпечуючи більшу доступність та привабливість рекламних матеріалів.

РОЗДІЛ 1

АНАЛІТИЧНИЙ ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ТА СКЛАДОВИХ СИСТЕМИ

1.1 Реклама

Рекламна діяльність має на меті популяризацію продукту або послуги серед цільової аудиторії. Це одна з найдавніших стратегій маркетингу, що спрямована на вплив на поведінку цільової аудиторії з метою здійснення покупок, продажів або виконання певних дій. Використовуючи чітко адаптоване повідомлення, реклама може бути спрямована на велику або нішову аудиторію.

З поширенням Інтернету виникло розрізнення на традиційну та цифрову рекламу. Традиційна реклама охоплює друковану, телевізійну та радіорекламу, що існує понад 150 років. Друкована реклама залишається найефективнішою для бізнесу, оскільки звертається до цільової аудиторії через газети, журнали та листівки.

Цифрова реклама ґрунтується на будь-якій рекламній діяльності в Інтернеті, такій як медійна реклама, контекстна реклама, реклама в соціальних мережах та інше. Ця форма реклами є більш доступною та легше відстежується, тому вона стала широко використовуваною формою маркетингу.

Рекламу (рис. 1.1) можна класифікувати за:

принципом роботи;

цільовою аудиторією ;

універсальністю ;

креативом;

тривалістю показу.

Рисунок 1.1 − Різновиди реклами

Різновиди реклами, такі як телевізійна , інтернет-маркетингова та соціальна реклама, взаємодіють для створення комплексної стратегії просування товарів і послуг. Кожен тип реклами має свої особливості та переваги, що дозволяє брендам ефективно взаємодіяти з різними сегментами аудиторії та досягати поставлених маркетингових цілей.

1.2 Аналіз конкурентів

1.2.1 Сітілайт

Ця форма зовнішньої реклами є закритою конструкцією, яка розташовується вздовж тротуарів, біля торгових центрів, зупинок транспорту та інших місць громадського скупчення. Сітілайти не бояться негативного

впливу погодних умов, оскільки вони обладнані підсвічуванням і дозволяють розміщувати інформацію з обох боків.

Цей формат реклами призначений переважно для пішоходів, оскільки розташований на рівні очей, що забезпечує більш довготривалий візуальний контакт з клієнтами. Це дозволяє розмістити на ньому більше інформації, ніж на звичайних білбордах.

Рисунок 1.2 – Сітілайт

Характеристики:

- підтримує формат TIF;
- роздільна здатність для друку – 75 dpi;
- розмір макета – 1200×1800 мм, робоче поле – 1100×1700 мм;
- колірна модель – CMYK;
- шари зведені один;
- відсутність альфа-каналів;
- стиснення LZW (зменшує розмір файлу без втрати якості);
- превью у форматі JPEG чи PNG до 1 Мбайт.

У таблиці 1.1 наведені плюси і мінуси даного виду.

Таблиця 1.1 – Плюси та мінуси Сітілайт

Плюси	**Мінуси**
Простота застосування	Постійне місце дислокації
Зовнішня привабливість	Вартість розміщення
Можливість змінювати текст	Відсутність автономного живлення
Розповсюдженність	
Великий відсоток креативу	

Сітілайт - це ефективний інструмент освітлення, який поєднує в собі економію енергії та високу світлову ефективність. Застосування сітілайтів сприяє створенню комфортних інтер'єрів та сприяє сталому розвитку, зменшуючи енергоспоживання та викиди в атмосферу.

1.2.2 **Транзитна реклама**

Ця форма реклами розміщується на об'єктах транспортної інфраструктури, а також ззовні та всередині автобусів, тролейбусів, автомобілів та інших транспортних засобів. Її величезною перевагою є широке охоплення аудиторії. Наприклад, реклама, розміщена на бортах автобусів, щодня курсує вулицями міста та привертає увагу навіть тих, хто не користується громадським транспортом. Найчастіше такий формат використовують великі компанії з великим рекламним бюджетом.

Рисунок 1.3 – Транзитна реклама

Характеристики:

- застосування – на різних видах транспорту;
- живлення – не потребує;
- постійний показ одного повідомлення.

Таблиця 1.2 – Плюси та мінуси Транзитної реклами

Плюси	**Мінуси**
Одна з найдешевших	Відсутня інтерактивність
Привертає багато уваги за рахунок великих розмірів	Не можна змінювати зміст реклами
Різні носії	
Постійна зміна дислокацій	

Транзитна реклама виявляється потужним інструментом просування, оскільки ефективно сприяє досягненню цільової аудиторії в русі, забезпечуючи високу видимість та максимальний охоплення. Її стратегічне розташування на транспортних маршрутах робить її не лише засобом реклами, але і важливою частиною міського пейзажу, що підкреслює її значення для сталого розвитку медійного простору.

1.2.3 **Білборд**

Білборди - це великі щити, які встановлюють на вулицях та вздовж трас. Вони чудово працюють для привернення уваги як пішоходів, так і автомобілістів. Білборди легко помітити завдяки їх великому розміру, тому за їх допомогою можна швидко охопити велику кількість цільової аудиторії.

Однак головне - це продумати оформлення, оскільки потенційні клієнти мають лише декілька секунд, щоб сприйняти інформацію. Тому краще використовувати слогани та короткі фрази з приблизно шести словами замість довгих текстів.

Рисунок 1.4 – Білборд

Характеристики:

- висота 3 метри та ширина 6 метрів;
- застосування – на вулиці;
- фіксація – на металевому закріплені.

Таблиця 1.3 – Плюси та мінуси Білбордів

Плюси	**Мінуси**
Висока передача інформації за рахунок розміру	Вигорає на сонці

Зазвичай встановлені в людних місцях	Висока вартість
Можливість повторного показу	Не інтерактивний

Білборди визначаються як важливий елемент рекламної інфраструктури, забезпечуючи широкий охоплення цільової аудиторії та неперевершену візуальну привабливість. Їх впливна роль у створенні образу брендів та формуванні публічної свідомості робить білборди не тільки ефективними засобами маркетингу, але й невід'ємною частиною сучасної міської естетики.

1.2.4 **Штендер**

Це маленька переносна конструкція, яку часто розміщують поряд з торговими точками, салонами краси, кафе, ресторанами та барами для того, щоб привернути увагу перехожих. У порівнянні з іншими видами зовнішньої реклами, штендери є менш витратними. Вони прості у встановленні і легко переміщуються з одного місця в інше. Найчастіше на вулицях можна побачити прямокутні або арочні штендери, але також є і фігурні.

Рисунок 1.5 – Штендер

Характеристики:

- вага – 3 кг;
- рекламне поле – широкоформатний друк;

Таблиця 1.4 – Плюси та мінуси Штендеру

Плюси	**Мінуси**
Мала вага конструкції	При неякісному нанесені рекламний матеріал може тріснути
Міцність і стійкість до вітру	Потрібно постійно виносити та заносити
Доступність по ціні	
Тривалий термін служби	
Компактні габарити	

Штендер виявляється важливим інструментом для привертання уваги та розповсюдження інформації в об'ємних міських просторах. Компактність та можливість стратегічного розташування роблять штендери ефективними засобами комунікації, сприяючи інформаційній доступності та взаємодії з аудиторією.

1.2.5 **Жива реклама**

. Це зовнішня реклама, яка взаємодіє з людьми. Такий спосіб передачі інформації до цільової аудиторії включає в себе використання костюмів, плакатів-покажчиків, табличок. Найчастіше "живу" рекламу можна побачити поруч з торговими точками для залучення покупців. При правильному підході вона не лише підвищує інтерес потенційних покупців, але й формує емоційний зв'язок, що сприяє укладанню угоди.

Рисунок 1.6 – Жива реклама

Таблиця 1.5 – Плюси та мінуси живої реклами

Плюси	Мінуси
Гнучкість та мобільность	Вимагає зацікавленої аудиторії
Невелика вартість	
Можливість творчого підходу	

Жива реклама виявляється надзвичайно ефективною стратегією, оскільки вона пропонує не лише візуальні враження, а й активну взаємодію з людьми. Здатність створювати неповторний та емоційний досвід дозволяє живій рекламі виходити за рамки звичайних кампаній, взаємодіючи з глядачами на більш особистому рівні

1.3 Порівняння моделей-конкурентів

При виборі будь-якого виду реклами, важливо спиратися на характеристики реклами, розуміючи, що саме для вас є найбільш важливим. Основними критеріями для оцінки можуть бути мобільність, креативність, можливість взаємодії з аудиторією, розміри рекламного оголошення, ступінь інтерактивності та вартість. Кожен з цих критеріїв може мати вагому роль у

визначенні ефективності рекламного заходу, тому важливо ретельно зважити їх у контексті конкретної ситуації та цілей рекламної кампанії.

При виборі реклами потрібно звертати увагу на наступні моменти:

1. актуальність – вона повинна бути затребуваною;
2. креативність – адже нудна реклама не принесе ніякої користі;
3. зрозумілість – людина при перегляді повинна одразу розуміти про що йде мова в оголошені.

Порівняльну характеристику розглянутих видів реклами наведено в таблиці 1.6.

Таблиця 1.6 – Порівняльна таблиця моделей-конкурентів

Вид реклами / **Характеристика**	**Сітілайт**	**Транзитна реклама**	**Білборд**	**Штендер**	**Жива реклама**
Креативність	Середня	Висока	Висока	Низька	Висока
Мобільність	Відсутня	Висока	Відсутня	Середня	Висока
Можливість комунікувати з іншими	Відсутня	Відсутня	Відсутня	Відсутня	Можливо
Розміри рекламного оголошення, м	1,8×1,2	В залежності від розміру транспортного засобу	6,0×3,0	0,6×1,0	В залежності від розміру людини
Інтерактивність	Присутня	Відсутня	Відсутня	Відсутня	Відсутня
Вартість, грн	Від 2900	Від 4000	Від 6600	Від 1800	Від 1000

Порівнявши дані види реклами, можемо зробити висновки, що сітілайт найбільш оптимальний варіант, при великій кількості необхідних характеристик має не найбільшу ціну. Але якщо необхідна максимальна зацікавленість споживачів, то потрібно вибирати білборд хоча вартість є високою.

Тому прийнято рішення розробити прилад для показу реклами, який буде вміщати тільки необхідні складові, та буде відповідати бажаній вартості.

1.4 Формування вимог до апаратно-програмного забезпечення

Перед тим, як розпочати збирати пристрій, важливо визначитися з тим, яким характеристикам та якій ціні мають відповідати комплектуючі. Після порівняння аналогів потрібно обрати найбільш підходящий варіант, що відповідає усім необхідним критеріям, саме для вашого пристрою.

Для даного пристрою необхідно обрати такі комплектуючі: мікроконтролер, матричний дисплей LCD, модуль годин реального часу, датчик диму та датчик вологості. Підбір правильних комплектуючих гарантує оптимальну роботу пристрою та відповідність його функціональності вимогам проекту.

1.4.1 **Мікроконтролер**

В якості програмованої платформи вирішено обрати один з мікроконтролерів родини *Arduino*. *Arduino* підходить, як початківцям так і професіоналам та являє собою електронний конструктор з зручною платформою швидкої розробки електронних пристроїв.

Саме Arduino обрано через: доступність, зручність, відкриту архітектуру та програмний код, простоту мови програмування. Arduino дає змогу комп'ютеру взаємодіяти з фізичним світом, виходячи за рамки віртуального. Пристрої на базі Arduino можуть управляти іншими виконавчими пристроями, або використовуючи різні датчики, отримувати інформацію про навколишнє середовище. Програмування пристрою відбувається через USB без використання програматорів.

Незважаючи на те, що на ринку існує безліч різновидів Arduino, найчастіше використовуються як інженерами, так і любителями:

- **Arduino UNO;**

- **Arduino Nano.**

Різниця між Arduino UNO та Arduino Nano:

Плата Arduino Nano схожа на плату Arduino UNO, включаючи подібний мікроконтролер, такий як Atmega328p. Таким чином, вони можуть використовувати подібні програми. Основна різниця між ними – це розмір.

Оскільки розмір Arduino Uno вдвічі більший за наноплату. Тож дошки Uno використовують більше місця в системі. Програмування UNO можна виконати за допомогою кабелю USB, тоді як Nano використовує кабель mini USB. Основні відмінності між Uno та Nano перелічені у таблиці 1.7.

Таблиця 1.7 – Порівняльна таблиця Arduino UNO та Arduino Nano

Технічні характеристики	**Arduino Uno**	**Arduino Nano**
Процесор	ATmega 328P	ATmega 328P
Вхідна напруга	5V/7-12V	5V/7-12V
Швидкість ЦП	16 MHz	16MHz
Аналогові вхід/вихід	6/0	8/0
Цифрові вхід/вихід	14/6	14/6
EEPROM/SRAM [kB]	1/2	1/2
Флеш-память	32	32
USB	Звичайний	Міні
USART	1	1

Було віддано перевагу Arduino Nano (рис. 1.7).

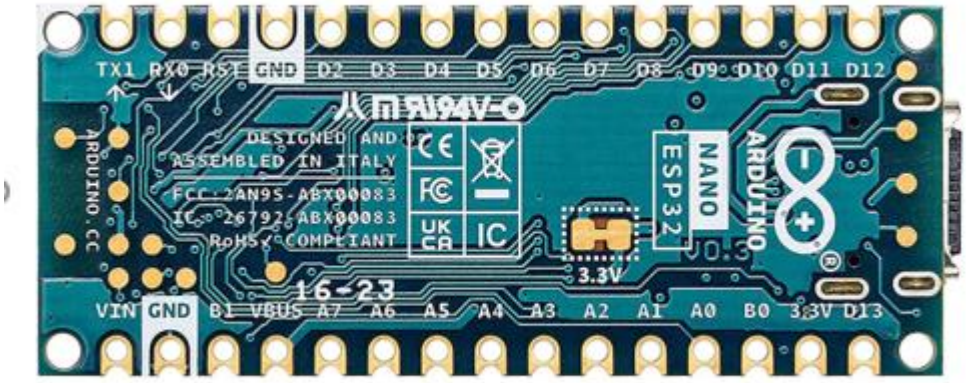


Рисунок 1.7 – Arduino Nano

Зв'язок плати Arduino UNO може здійснюватися з використанням різних джерел, таких як використання додаткової плати Arduino або комп'ютера. Мікроконтролер, що використовується на платі Nano (ATmega328), забезпечує послідовний зв'язок (UART TTL), за допомогою цифрових пінів TX та RX. Програмне забезпечення Arduino складається з послідовного монітора, що дозволяє легко передавати та приймати текстову інформацію від плати [4].

1.4.2 Про потужність

Максимальна потужність, яку може безпечно забезпечити Arduino Uno при живленні від USB, становить близько 400 мА при напрузі 5 В. Світлодіодні матричні дисплеї мають досить високе споживання струму і можуть використовувати до 1 А, якщо яскравість встановлена на максимум.

Завжди використовується зовнішнє джерело живлення для дисплея. Ніколи не підключаються більше чотирьох світлодіодних матриць безпосередньо до мікроконтролера і завжди тримається яскравість менше 50%, щоб уникнути руйнування регулятора напруги мікроконтролера.

1.4.3 Інтерфейс SPI та Специфікація MAX7219

MAX7219 має інтерфейс SPI - годинник, дані, вибір

1. Дані - **MOSI** - головний вихід, послідовний вхід. 7219 є підпорядкованим пристроєм.
2. Вибір мікросхеми - **Завантаження (CSn)** - активний низький вибір мікросхеми .
3. Годинник – **SCK.**
4. **Земля.**

1.5 Повне з’єднання світлодіодної матриці

Кожен 3-контактний роз’єм на схемі вище символізує один модуль, описаний у попередньому розділі (LED Matrix + MAX72xx), тепер з’єднали всі ці модулі разом.

Усі мікросхеми MAX722xx мають спільні лінії MOSI та SCK, MISO не використовується, кожна мікросхема займає окрему лінію Slave Select.

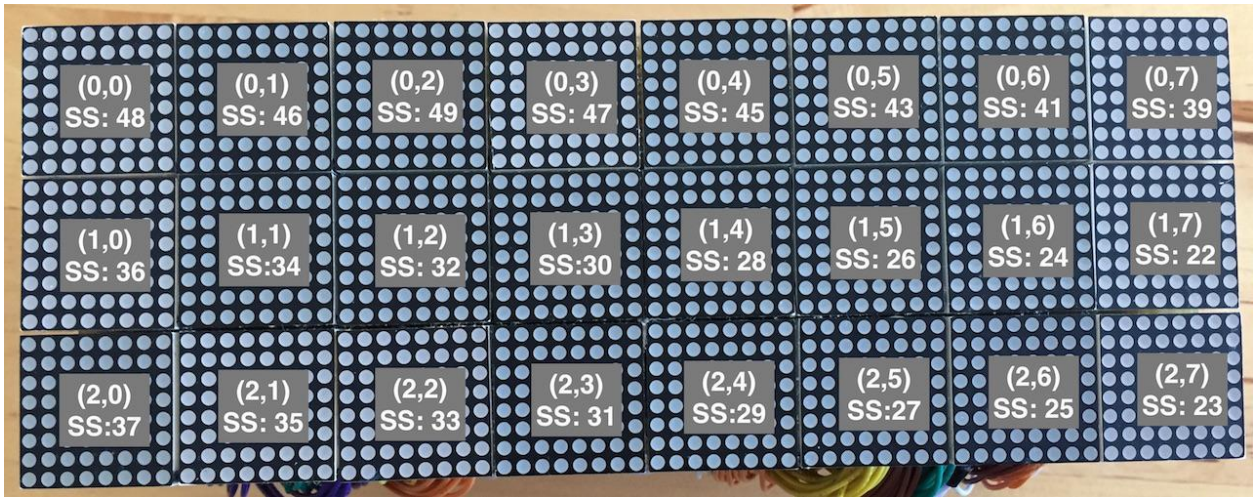


Рисунок 1.18–Підключення світлодіодних матриць

Усі мікросхеми MAX722xx мають спільні лінії MOSI та SCK, MISO не використовується, кожна мікросхема займає окрему лінію Slave Select.

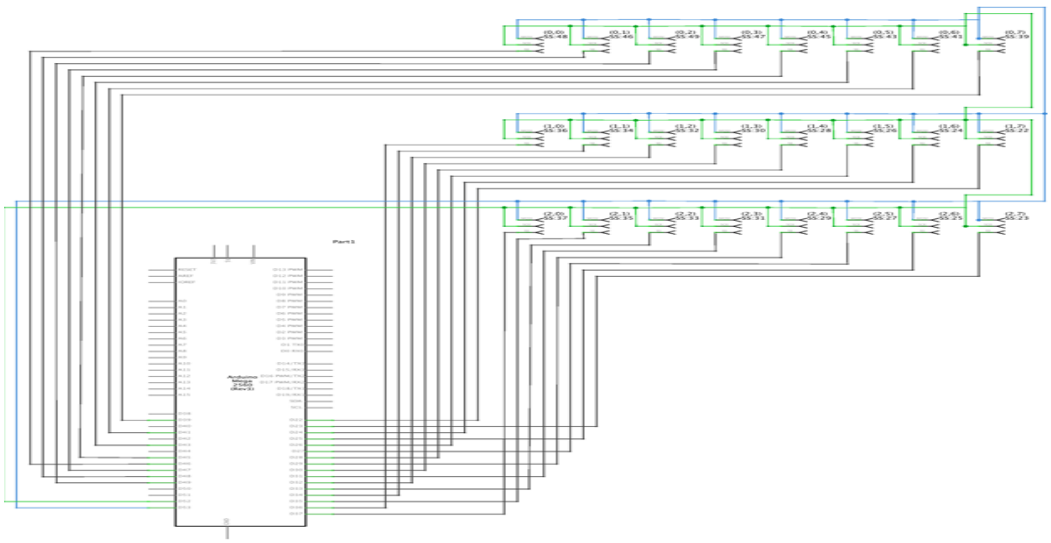


Рисунок 1.19–Схематичне підключення елементів світлодіодних матриць

Положення світлодіодної матриці на схемі вище безпосередньо відповідає їх розташуванню на фізичному дисплеї, який використовувався для тестування. Крім того, кожен модуль має опис, що вказує його позицію та рядок Select Slave, наприклад: (2,1) SS: 35 дає нам другий модуль у третьому рядку (відлік від нуля) і PIN:35 на Arduino для лінії Select Slave.

1.6 Детальний огляд Arduino Uno

Плата Arduino UNO має максимальний номінальний струм 40 мА, тому навантаження не повинно перевищувати цей номінальний струм, інакше можна пошкодити плату. Вона постачається з кварцевим генератором 16 МГц, що є його робочою частотою.

Розпінування Arduino Uno складається з 14 цифрових контактів, починаючи з D0 до D13, і має 6 аналогових контактів, починаючи від A0 до A5. Крім того, є 1 контакт для скидання, який використовується для програмного скидання плати, і 6 контактів живлення, які забезпечують різні рівні напруги.

Arduino UNO підтримує 3 типи протоколів зв'язку: Послідовний протокол, Протокол I2C та Протокол SPI. Крім USB, для живлення плати

також можна використовувати акумулятор або адаптер змінного та постійного струму.

Плата Arduino Uno має вбудований інтерфейс USB для розвитку послідовного зв'язку з комп'ютером. Мікроконтролер Atmega328, що розміщений на платі, має ряд функцій, таких як таймери, лічильники, переривання, ШІМ, центральний процесор, контакти вводу-виводу і базується на тактовій частоті 16 МГц.

Arduino Uno є платформою з відкритим вихідним кодом, де кожен може змінювати та оптимізувати плату на основі кількості інструкцій та завдань, які він хоче виконати. Ця плата має вбудовану функцію регулювання, яка тримає напругу під контролем, коли пристрій підключено до зовнішнього пристрою.

На платі Arduino Uno розміщено кілька цифрових і аналогових контактів вводу-виводу, які працюють при напрузі 5 В. Ці контакти мають стандартні робочі характеристики від 20 мА до 40 мА. У платі використовуються внутрішні підтягуючі резистори, які обмежують струм, що перевищує задані умови роботи. Однак занадто велике збільшення струму робить ці резистори марними і може пошкодити пристрій.

Arduino Uno постачається з можливістю взаємодії з іншими платами, мікроконтролерами та комп'ютерами Arduino. Atmega328, розміщений на платі, забезпечує послідовний зв'язок за допомогою контактів, таких як Rx і Tx. Atmega16U2, вбудований на платі, забезпечує шлях для послідовного зв'язку за допомогою драйверів USB com. Послідовний монітор надається в програмному забезпеченні IDE, яке використовується для надсилання або отримання текстових даних з плати. Нижче наведено кілька основних застосувань плати:

– Вбудована система.

– Система безпеки та оборони.

– Цифрова електроніка та робототехніка.

– Прилавок для паркування.

– Вагові машини.

– Таймер зворотного відліку світлофора.

– Медичний інструмент.

– Аварійне світло для залізниць.

– Домашня автоматизація.

– Про слова автоматизація.

Arduino має велику спільноту, яка розвиває та ділиться знання з широким колом аудиторії. Доступна швидка підтримка щодо технічних аспектів будь-якого електронного проекту. Коли ви вибираєте плату Arduino перед інши контролера, вам не потрібно влаштовувати додаткові периферійні пристрої та пристрої, оскільки більшість функцій легко доступні на платі, що робить ваш проект економічним за своєю природою та вільним від великої кількості технічних знань.

1.7 Порівняння Arduino та Raspberry Pi

Raspberry Pi має багато різних моделей, які працюють від процесора ARM. Від оригінальної одноядерної моделі з тактовою частотою 700 МГц у 2012 році до сучасної чотириядерної моделі 1,5 ГГц. Моделі Arduino зазвичай живляться від мікроконтролерів Atmel і часто мають частоту менше 100 МГц. Наприклад, Arduino Uno працює на частоті 16 МГц.

Ці мікросхеми значно повільніші, ніж ті, що є в Raspberry Pi, але Arduino не має стільки накладних витрат, як запуск операційної системи Linux. Але є що сказати про процесор з фіксованою швидкістю. Він надійний і не має масштабування, яке може спричинити проблеми з часом для проектів, які потребують абсолютної точності.

Аналізуючи потужність процесора Raspberry Pi завжди був явним переможцем. Arduino Portenta H7 — потужна плата, але вона не може зрівнятися з Pi за потужністю. Найнижча специфікація Raspberry Pi, яку можемо придбати, - це Raspberry Pi Zero W, який має один процесор з тактовою частотою 1 ГГц і все ще забезпечує більшу потужність, ніж двоядерний 480

МГц STM32H747. Але Raspberry Pi потребує більшої потужності, оскільки він також працює під керуванням операційної системи .

Коли справа доходить до споживання електроенергії, Raspberry Pi 4 досить голодна плата. У Raspberry Pi 4 був новий роз’єм живлення USB C і офіційний блок живлення з більш високим рейтингом, який забезпечував до 3 А для Pi та будь-яких підключених до нього пристроїв. Таким чином, Raspberry Pi 4 теоретично може працювати з потужністю до 15 Вт. Arduino Uno може витягнути максимум 500 мА через USB. Він може споживати більше струму, якщо використовується

Гнучкість тут є перевагою. Arduino Uno може працювати з діапазоном напруг, які регулюються до 5 В, необхідних для плати. Живлення може подаватись через порт USB, роз’єм постійного струму (від 6 до 20 В, який направляється через стабілізатор 5 В) або через контакт VIN, який підключається безпосередньо до мікроконтролера, тому перед підключенням завжди перевіряйте правильну напругу. Споживання струму для Arduino Uno, на якому працює «блимання», становить близько 40 мА, але додавання компонентів збільшить кількість використовуваного струму.

Виводи загального призначення (GPIO) є з’єднання з платою, і за допомогою них код можна використовувати для взаємодії з навколишнім світом. Виводи GPIO є двостороннім зв’язком, вони можуть бути входа або вихода , і вони можуть використовувати спеціальні протоколи зв’язку.

GPIO Raspberry Pi складається з 40 контактів (120 при використанні обчислювального модуля), а контакти є сумішшю цифрових широтно-імпульсної модуляції (PWM) і спеціальних протоколів, таких як I2C, SPI і UART. Багато контактів GPIO використовуються для більш ніж однієї функції/протоколу.

Arduino Uno має менше контактів GPIO (Arduino Mega має набагато більше контактів GPIO), але він має основи, цифрові контакти, ШІМ, I2C, SPI. У Arduino є те, чого немає у Raspberry Pi, аналогові входи, які використовують постійний сигнал, як правило, напругу, як засіб передачі даних.

Можливо, йому не вистачає великої кількості контактів GPIO, але в Arduino є все, що нам потрібно, щоб почати з електронних проектів, включаючи аналогові входи. Це може здатися невеликим, але вони відкривають перелік додаткових датчиків і входів. Для точного керування в проекті можна використовувати потенціометри та аналогові джойстики. Мікросхеми, такі як датчик температури TMP36, можна використовувати для збору точних даних для проєкту. Якщо вашому проекту Arduino потрібно більше контактів GPIO, тоді Arduino Mega має 70 контактів, а плати клонів можна купити відносно дешево. Вартість: Raspberry Pi проти Arduino

Найдешевшим Raspberry Pi є Zero W, який продається за 10 доларів і забезпечує повноцінний комп’ютер Linux, з Wi-Fi і Bluetooth і доступом до важливого GPIO (хоча вам доведеться припаяти контакти самостійно). Використання Pi Zero W як вбудованого пристрою є недорогим способом створення проекту IoT. Найдорожчим Raspberry Pi є Pi 4 8 ГБ, який продається за 75 доларів, але щоб отримати максимальну віддачу від цієї плати, вам потрібно буде придбати додаткові аксесуари та плати HAT.

Вартість Arduino може варіюватися від кількох доларів за плату-клон до майже 100 доларів за офіційну Portenta H7. ATtiny85, недорогий мікроконтролер лише з шістьма контактами GPIO, можна купити менш ніж за 2 долари, і він пропонує достатньо потужності для проектів робототехніки. Клони Arduino UNO можна купити відносно дешево і забезпечують досить хорошу сумісність у порівнянні з офіційними платами .

Різниця в ціні між клонами та офіційними платами Arduino відображає підтримку, яку компанія надає своїм спільнотам. Плати клонів не підтримують безпосередньо спільноту, але вони дешеві і в основному добре працюють. Офіційні плати працюють надзвичайно добре, і, купуючи плату, ви підтримуєте спільноту та Arduino у створенні нових продуктів та допоміжних матеріалів.

Arduino є явним переможцем за найнижчу вартість плати. Але це супроводжується парою обмежень. Вам знадобиться комп’ютер для

програмування Arduino, а також компоненти. Швидше за все, у вас вже є комп’ютер, тож це не пов’язано безпосередньо з витратами , вартість компонентів може сильно варіюватися, залежно від ваших вимог. Raspberry Pi — це власний комп’ютер, але йому все одно потрібні компоненти та доповнення, щоб максимально використовувати його. Хоча плати Arduino мають вбудовану пам’ять, для кожного Raspberry Pi потрібна карта microSD (перегляньте наш список найкращих карт microSD Raspberry Pi , або ви можете завантажитися з USB).

Raspberry Pi — це повноцінний настільний комп’ютер Linux, який просто має доступ до GPIO завдяки SoC Broadcom. Підключення Raspberry Pi до монітора, клавіатури та ші надає нам користувацький досвід, не дуже віддалений від звичайного комп’ютера.

Оскільки Raspberry Pi працює під керуванням Linux, він має доступ до багатьох різних мов програмування, деякі з яких також можна використовувати з GPIO. Python і Scratch — два очевидні приклади мов, які можуть працювати з GPIO, але є багато інших, включаючи Node-RED, Ruby і C.

З Arduino наш вибір дещо обмеженіший. Arduino IDE — це просто набір функцій C/C++, які компілюються та розміщуються на плату. Плати Arduino призначені для підключення та програмування іншим комп’ютером, на якому працює IDE. Існують альтернативи Arduino IDE та мові C/C++. За допомогою перепрошивки спеціального файлу проекту в Arduino його можна також використовувати з Python або з мова на основі блоків, таки як ArduBlockly і mBlock. Але це не дуже часто, і, враховуючи, що у вас є лише 16 КБ пам’яті для вашої програми на Arduino Uno, програ не можуть бути такими складними .

Враховуючи безліч варіантів використання, Raspberry Pi — це універсальна платформа для всіх можливих проєктів. Існує багато мов, що охоплюють рівні кваліфікації та парадигми для проєктів розробників, що включають GPIO, та мови для розробки програмного забезпечення,

системного адміністрування та веб-розробки. Основні мови, особливо Python, є дуже популярними мовами , які мають велику кількість бібліотек розширення та велику підтримку.

У Arduino є певний вибір, коли справа доходить до мов програмування, але це не повноцінний комп’ютер, і це обмежує кількість і тип коду, який ви можете кинути на нього.

Raspberry Pi — це повнофункціональний настільний комп’ютер Linux, який можна використовувати для повсякденної діяльності або як сервер, але він також забезпечує GPIO, який дозволяє використовувати комп’ютер у великих і малих проєктах. Від простого миготіння світлодіода до комп’ютерного зору, машинного навчання та робототехніки Raspberry Pi містить багато функцій у платі розміром з кредитну картку.

Насправді Raspberry Pi може робити все, що може зробити Arduino, але йому потрібна невелика допомога у вигляді HAT і додавання плат, оскільки деякі функції, такі як аналого-цифрове перетворення, не вбудовані.

Це правда, що Raspberry Pi не має такої швидкості, як Arduino. Зрештою, Raspberry Pi — це комп’ютер, якому необхідно завантажити операційну систему, перш ніж можна буде виконати будь-яку роботу, і, коли ви хочете його вимкнути, вам дійсно слід дати команду вимкнення та терпляче чекати, поки система вимкнеться.

Arduino — це єдина плата завдань, яка запускає одну програму за раз і відразу запускається, як тільки ви її ввімкнете, а коли ви захочете її вимкнути, ви можете просто витягнути вилку. Безпосередність Arduino є великою перевагою для проектів, які збирають дані або простої роботизації.

Arduino - це дійсно універсальна плата, але Raspberry Pi - це повноцінний комп'ютер і апаратна платформа для злому. Якщо вам потрібен бездротовий зв’язок, необроблена потужність обробки та доступ до GPIO, Raspberry Pi надає все це в невеликому пакеті.

Хоча професіонали використовують їх для дуже серйозних кінцевих продуктів, Raspberry Pi і Arduino також розроблені для навчальних цілей, і це

зрозуміло завдяки тисячам проектів і ресурсів, доступних в Інтернеті. Raspberry Pi є безсумнівним фаворитом для освіти, оскільки він працює з багатьма різними мовами програмування, може використовуватися для навчання основних обчислювальних концепцій і використовуватися як інструмент дослідження. І ви можете буквально підключити його та запустити без додаткового комп'ютера.

Екосистема Arduino орієнтована на написання коду на бажаній мові програмування для спілкування з платою. Він робить одну справу за раз, але робить це дуже добре. У середовищі класу Raspberry Pi буде домінувати завдяки своїй універсальності.

Крім того, те, що ви дізнаєтеся, працюючи з Raspberry Pi, нескінченно краще можна перенести на інші платформи. Python, найпопулярніша мова на Pi, працює на веб-серверах Windows, Mac і навіть Linux. Якщо ви не використовуєте плату Arduino або щось сумісне, знання коду Arduino не так корисно, як знання Python.

Raspberry Pi відповідає потребам сучасності. Він економний, простий у використанні, простий у зберіганні та може бути використаний для багатьох тем і проектів. Arduino — чудова плата для навчання, але в класі універсальність і простота використання є ключовими перевагами.

Висновки до розділу 1

У розділі проаналізовано основні характеристики конкурентів на ринку та виділено їх недоліки. Встановлено, що сітілайт найбільш оптимальний варіант, при великій кількості необхідних характеристик має не найбільшу ціну. Але якщо необхідна максимальна креативність та увага то білборд буде кращим, хоча й має великий недолік в вартості та відсутність інтерактивності. Оптимальний набір для подальшого розроблення складається з мікроконтролера Arduino.

Розглянуто різновиди давачів забруднення повітря, за класифікацією. За допомогою порівняльного аналізу, відповідно до необхідних характеристик та прийнятній вартості, обрано комплектуючі: мікроконтролер Arduino Nano, дисплей LCD2004 з інтерфейсним модулем I2C,модуль реального часу DS1302, світлові модулі WS2812B та інші додаткові комплектуючі.

РОЗДІЛ 2
ОСНОВНІ АЛГОРИТМИ КОМП'ЮТЕРНОЇ ГРАФІКИ

2.1Растеризація

Растеризація – це процес перетворення геометричних об'єктів, заданих у векторній формі, в растрове зображення, що складається з пікселів. У комп'ютерній графіці растеризація одна із основних етапів візуалізації 3D сцен.

У процесі растеризації кожен геометричний об'єкт розбивається на безліч маленьких фрагментів, званих пікселями. Кожен піксель отримує свої колірні та глибинні значення, які визначають його видимість та зовнішній вигляд.

Растеризація виконується за допомогою графічного апарату комп'ютера, який перетворює векторні дані на растрове зображення. Цей процес включає кілька кроків:

Трансформація

Спочатку геометричні об'єкти перетворюються за допомогою матриць перетворення, таких як масштабування, поворот та зміщення. Це дозволяє змінити розмір та положення об'єктів у сцені.

Відсікання

Потім відбувається відсікання об'єктів, які знаходяться за межами області відображення або не є видимими для спостерігача. Це дозволяє прискорити процес растеризації, крім непотрібних об'єктів.

Розбиття на трикутники

До кожного об'єкта відбувається його розбиття на трикутники, оскільки трикутники є базовими примітивами для растеризации. Це робиться за допомогою алгоритму тесселяції, який розбиває об'єкт на множину трикутників.

Растеризація трикутників

Після розбиття об'єкта на трикутники кожен трикутник розтеризується окремо. Для кожного пікселя усередині трикутника обчислюються його колірні та глибинні значення, використовуючи інтерполяцію між вершинами трикутника.

Запис у буфер кадру

Після растеризації всіх трикутників отримані значення пікселів записуються в буфер кадру. Буфер кадру є растровим зображенням, яке відображатиметься на екрані.

Таким чином, растеризація є важливим етапом візуалізації 3D сцен, який дозволяє перетворити векторні дані на растрове зображення. Цей процес включає трансформацію об'єктів, відсікання невидимих об'єктів, розбиття на трикутники, растеризацію трикутників і запис в буфер кадру.

2.2Трасування променів

Трасування променів – це один із основних алгоритмів комп'ютерної графіки, який використовується для створення реалістичних зображень. Він заснований на моделюванні шляху світла, яке поширюється від джерела світла і відбивається від поверхонь об'єктів.

Основна ідея трасування променів полягає в тому, що для кожного пікселя зображення ми відправляємо промені в сцену та визначаємо, який об'єкт чи поверхню перетинає цей промінь. Потім ми обчислюємо освітлення цієї поверхні, враховуючи світлові джерела та властивості матеріалу.

Існують основні кроки трасування променів. Генерація первинних променів: для кожного пікселя зображення ми генеруємо промінь, який проходить через цей піксель і направлений у сцену.

Перетин променів з об'єктами: для кожного променя ми перевіряємо, чи він перетинає якийсь об'єкт у сцені. Для цього ми обчислюємо перетин променя з кожним об'єктом і вибираємо найближчий перетин.

Обчислення освітлення: після визначення найближчого перетину ми обчислюємо освітлення цієї точки з огляду на світлові джерела та властивості матеріалу об'єкта. Це включає розрахунок тіней, відображень і заломлень світла.

Рекурсивне трасування променів: для об'єктів, що мають відбивні або заломлюючі властивості, ми генеруємо додаткові промені, що відбиваються або заломлюються від поверхні і продовжують трасування.

Остаточне обчислення кольору пікселя: після трасування всіх променів ми об'єднуємо отримані значення освітлення та обчислюємо остаточний колір пікселя.

Трасування променів є обчислювально інтенсивним процесом, так як потрібно безліч обчислень для кожного пікселя зображення. Однак, завдяки своїй здатності створювати реалістичні зображення з ефектами відбиття та

заломлення світла, трасування променів широко використовується у комп'ютерній графіці та візуалізації.

2.3 Глобальне освітлення

Глобальне освітлення – це метод моделювання освітлення у комп'ютерній графіці, який враховує взаємодію світла з довкіллям та об'єктами у сцені. Він дозволяє створювати більш реалістичні та природні зображення, враховуючи різні фізичні явища, такі як відображення, заломлення та розсіювання світла.

Основні компоненти глобального висвітлення:

Довкілля

Навколишнє середовище включає всі об'єкти і матеріали, які знаходяться в сцені. Вона визначає, як світло взаємодіє з об'єктами і як воно поширюється у просторі. Різні властивості довкілля, такі як колір, відбивна здатність і прозорість, впливають візуальне сприйняття сцени.

Джерела світла

Джерела світла відіграють важливу роль у глобальному висвітленні. Вони визначають напрям, інтенсивність та колір світла, що падає на об'єкти в сцені. Різні типи джерел світла, такі як точкові джерела, спрямовані джерела та майданні джерела, створюють різні ефекти освітлення та тіні.

Модель відображення світла

Модель відображення світла визначає, як світло джерел відбивається від поверхонь об'єктів. Вона враховує різні фактори, такі як відбивна здатність матеріалу, кут падіння світла та нормаль до поверхні. Модель відображення може бути простою, наприклад, модель Фонга, або складнішою, що враховує відблиски, затінення та інші ефекти.

Глобальна ілюмінація

Глобальна ілюмінація відповідає за поширення світла у сцені та створення ефектів, таких як віддзеркалення світла від одного об'єкта на інший.

Вона враховує як пряме світло джерел, а й непряме освітлення, що відбивається від навколишніх об'єктів. Глобальна ілюмінація може бути досягнута за допомогою різних методів, таких як трасування променів, метод Монте-Карло та методи освітлення з використанням карти сферичної гармоніки.

Глобальне освітлення є важливим інструментом у комп'ютерній графіці, що дозволяє створювати реалістичні та ефектні зображення. Воно вимагає обчислювальних ресурсів, але сучасні алгоритми та техніки дозволяють досягти високого ступеня реалізму та деталізації візуалізації.

2.4 Тіньові алгоритми

Тіньові алгоритми - це методи, які використовуються в комп'ютерній графіці для створення ефекту тіней. Тіні відіграють важливу роль у створенні реалістичних та об'ємних зображень, оскільки вони допомагають передати інформацію про форму та розташування об'єктів у сцені.

Алгоритм тіньових карт

Один із найпоширеніших тіньових алгоритмів – це алгоритм тіньових карт (shadow mapping). Він ґрунтується на ідеї створення карти глибини з точки огляду та використання цієї карти для визначення того, які пікселі знаходяться в тіні.

Алгоритм тіньових карток складається з наступних кроків:

Рендеринг сцени з точки зору та збереження глибини кожного пікселя в текстуру – тіньову карту.

Рендеринг сцени з точки освітлення та перевірка глибини кожного пікселя за допомогою тіньової карти. Якщо піксель перебуває у тіні, він затемняється або змінюється відповідним чином.

Алгоритм тіньових карт простий у реалізації та забезпечує хороші результати для більшості сцен. Однак він має деякі обмеження, такі як

артефакти, пов'язані з роздільною здатністю текстури тіньової карти та обмеженнями точності глибини.

Алгоритми об'ємних тіней

Алгоритми об'ємних тіней (volumetric shadow algorithms) використовуються для створення тіней, які мають об'єм і можуть проникати через прозорі об'єкти. Ці алгоритми ґрунтуються на моделюванні взаємодії світла з об'ємними об'єктами, такими як туман, дим або хмари.

Одним із прикладів алгоритмів об'ємних тіней є алгоритм розсіяного освітлення (ambient occlusion). Він заснований на ідеї моделювання того, як світло розсіюється всередині об'ємних об'єктів та створює тіні на навколишніх поверхнях.

Алгоритми об'ємних тіней забезпечують більш реалістичні та складні ефекти тіней, але вони також вимагають більше обчислювальних ресурсів та складніше у реалізації.

Інші тіньові алгоритми

У комп'ютерній графіці існує багато інших тіньових алгоритмів, які можуть бути використані для створення різних ефектів тіней. Деякі з них включають:

Алгоритми м'яких тіней (soft shadow algorithms), які створюють більш плавні та розмиті тіні.

Алгоритми тіней від прозорих об'єктів (shadow algorithms for transparent objects), які враховують взаємодію світла з прозорими матеріалами.

Алгоритми тіней від джерел світла з розподіленою формою (shadow algorithms for area light sources), які враховують розмір і форму джерел світла.

Вибір конкретного тіньового алгоритму залежить від необхідного ефекту та обмежень обчислювальних ресурсів.

2.5Алгоритми згладжування

Алгоритми згладжування у комп'ютерній графіці використовуються для створення більш плавних та реалістичних зображень. Вони дозволяють усунути ступінчастість (аліасинг) на межах об'єктів і зробити переходи між кольорами та яскравостями більш плавними.

Існує кілька різних алгоритмів згладжування, кожен з яких має свої особливості та застосовується у різних ситуаціях. Розглянемо деякі з них:

Алгоритм суперсемплювання (Supersampling)

Алгоритм суперсемплювання заснований на ідеї збільшення роздільної здатності зображення для більш точного розрахунку кольору кожного пікселя. Для цього кожен піксель розбивається на кілька підпікселів, і для кожного обчислюється колір. Потім отримані кольори підсумовуються та усереднюються, щоб отримати остаточний колір пікселя.

Алгоритм суперсемплювання дозволяє згладити ступінчастість на межах об'єктів та покращити якість зображення, але потребує більше обчислювальних ресурсів і може призвести до збільшення часу рендерингу.

Алгоритми фільтрації (Filtering)

Алгоритми фільтрації використовуються для згладжування зображення шляхом застосування різних фільтрів до пікселів. Вони ґрунтуються на математичних операціях, таких як пакунок або усереднення значень пікселів.

Існує кілька типів алгоритмів фільтрації, включаючи:

Білінійна фільтрація (Bilinear filtering) – використовує лінійну інтерполяцію для обчислення кольору пікселя з урахуванням його сусідів.

Трилінійна фільтрація (Trilinear filtering) – комбінує білінійну фільтрацію з інтерполяцією між різними рівнями деталізації текстури.

Анізотропна фільтрація (Anisotropic filtering) – враховує напрям текстури та застосовує різні рівні фільтрації у різних напрямках.

Алгоритми фільтрації дозволяють згладити ступінчастість та покращити якість текстур, але можуть призвести до деякої втрати деталей.

Алгоритми згладжування кривих (Curve smoothing)

Алгоритми згладжування кривих використовуються для усунення ступінчастості на межі кривих і зробити їх більш плавними. Вони ґрунтуються на математичних методах інтерполяції та апроксимації кривих.

Існує кілька алгоритмів згладжування кривих, включаючи:

Алгоритм Чебишева (Chebyshev algorithm) – використовує поліноми Чебишева для апроксимації кривих.

Алгоритм Безьє (Bezier algorithm) – заснований на використанні кривих Без'є для згладжування кривих.

Алгоритм Б-сплайн (B-spline algorithm) - використовує B-сплайн для апроксимації кривих.

Алгоритми згладжування кривих дозволяють зробити межі об'єктів більш плавними та природними, але потребують додаткових обчислень і можуть призвести до певної втрати деталей.

2.6 Алгоритми текстурування

Алгоритми текстурування – це методи, які у комп'ютерної графіці нанесення текстури на поверхню об'єкта. Текстура є зображенням, яке може бути застосовано до об'єкта, щоб надати йому додаткові деталі та реалістичність.

Афінне текстурування

Афінне текстурування – це простий та швидкий алгоритм, який застосовує текстуру до об'єкта, використовуючи афінні перетворення. Він заснований на припущенні, що текстура та об'єкт мають однакову форму та розміри. Афінне текстурування може бути використане для простих об'єктів, але не завжди дає реалістичні результати.

Проекційне текстурування

Проекційне текстурування – це складніший алгоритм, який враховує перспективу та спотворення при нанесенні текстури на об'єкт. Він

використовує матриці проекції та текстурні координати для правильного відображення текстури на об'єкті. Проекційне текстурування дозволяє створювати більш реалістичні та деталізовані зображення, особливо для об'єктів із складною геометрією.

Мапінг текстурних координат

Мапінг текстурних координат – це процес прив'язки текстури до геометричних координат об'єкта. Кожна точка на поверхні об'єкта має відповідні текстурні координати, які визначають, яка частина текстури відображатиметься на цій точці. Мапінг текстурних координат може бути виконаний різними способами, включаючи сферичний мапінг, циліндричний мапінг та планарний мапінг.

Фільтрування текстур

Фільтрування текстур – це процес згладжування текстури для усунення пікселізації та створення більш плавного зображення. Різні методи фільтрації можуть бути використані, включаючи білінійну фільтрацію, трилінійну фільтрацію та анізотропну фільтрацію. Фільтрування текстур дозволяє отримати більш якісні та реалістичні зображення.

Упаковка текстур

Упаковка текстур – це процес об'єднання кількох текстур в одну текстуру для оптимізації використання пам'яті та прискорення процесу рендерингу. Упаковка текстур дозволяє зменшити кількість звернень до пам'яті та покращити продуктивність під час роботи з великою кількістю текстур.

Алгоритми текстурування відіграють важливу роль у створенні реалістичних та деталізованих зображень у комп'ютерній графіці. Вони дозволяють додати додаткові деталі та ефекти до об'єктів, роблячи їх більш привабливими та реалістичними для глядача.

2.7 Алгоритми анти-аліасингу

Анти-аліасинг – це техніка, яка використовується у комп'ютерній графіці для згладжування країв та усунення ступінчастості (ефекту “стрибків”) на зображеннях. Він дозволяє створювати більш плавні та реалістичні зображення, покращуючи якість візуалізації.

Суперсемплінг

Один із основних методів анти-аліасингу – це суперсемплінг. Він у тому, що кожного пікселя зображення обчислюється його колір, враховуючи як його центральну точку, а й сусідні точки. Для цього піксель ділиться на дрібніші підпікселі, і для кожного з них обчислюється колір. Потім кольори пікселів комбінуються, щоб отримати остаточний колір пікселя. Це дозволяє згладити краї та зменшити ступінчастість.

Мультисемплінг

Мультисемплінг - це метод, який використовує кілька семплів для кожного пікселя зображення. Замість того, щоб ділити піксель на підпікселі, мультисемплінг використовує кілька точок усередині пікселя для обчислення кольору. Це дозволяє отримати більш точні та плавні зображення, особливо при роботі з об'єктами, що мають складні форми та криві.

Фільтрування

Фільтрування – це метод, який використовує різні фільтри для згладжування зображень. Один із найпоширеніших фільтрів – це фільтр Гауса. Він застосовує розмиття до зображення, видаляючи високочастотні компоненти та згладжуючи краї. Це дозволяє зменшити ступінчастість та створити більш плавні переходи між кольорами.

Субпіксельне зміщення

Субпіксельне зміщення – це метод, який використовує зміщення підпікселів для покращення якості зображення. Він ґрунтується на припущенні, що піксель складається з трьох підпікселів – червоного, зеленого

та синього. Шляхом усунення підпікселів на невелику відстань та комбінування їх кольорів можна створити більш точне та плавне зображення.

Алгоритми анти-аліасингу відіграють важливу роль у створенні якісних та реалістичних зображень у комп'ютерній графіці. Вони дозволяють усунути ступінчастість і зробити зображення більш плавними та приємними для сприйняття.

2.8 Алгоритми об'ємного рендерингу

Алгоритми об'ємного рендерингу – це методи у комп'ютерній графіці, які дозволяють створювати реалістичні зображення тривимірних об'єктів з урахуванням їхньої внутрішньої структури та властивостей. Вони використовуються для візуалізації об'єктів, які мають об'єм та можуть бути прозорими або мати складну внутрішню структуру, таку як туман, дим, рідина тощо.

Основні алгоритми об'ємного рендерингу:

1. Рейкастінг (Ray casting)

Рейкастінг – це метод, при якому для кожного пікселя зображення створюється промінь, який проходить через сцену та взаємодіє з об'єктами. Промінь перевіряє, чи він перетинає об'єкти на своєму шляху, і розраховує освітлення та колір пікселя в залежності від результатів перетину. Цей метод дозволяє враховувати прозорість та відображення об'єктів.

2. Воксельний рендеринг (Voxel rendering)

Воксельний рендеринг – це метод, у якому тривимірні об'єкти представляються як тривимірної сітки, що з вокселів (тривимірних пікселів). Кожен воксель має свій колір та прозорість, і вони об'єднуються разом для створення реалістичного зображення. Цей метод дозволяє враховувати складну внутрішню структуру об'єктів.

3. Маршинг кубів (Marching cubes)

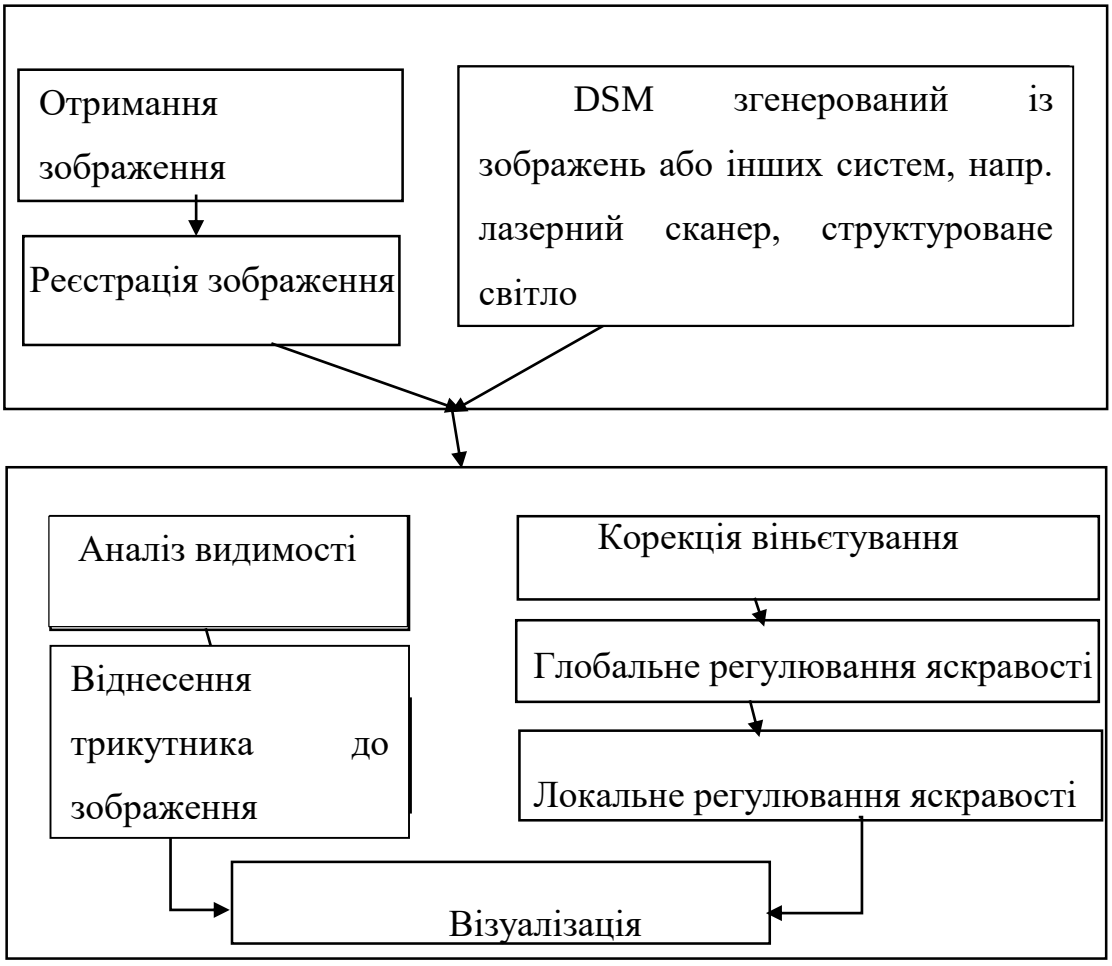
Маршинг кубів – це метод, у якому тривимірні об'єкти представляються як сітки кубів. Кожен куб має свій колір та прозорість, і вони об'єднуються разом для створення реалістичного зображення. Цей метод особливо корисний для візуалізації складних об'єктів, таких як рідина або туман.

4. Текстурний рендеринг (Texture mapping)

Текстурний рендеринг – це метод, коли на поверхню об'єкта накладається текстура, що визначає його колір, освітлення та інші властивості. Цей метод дозволяє створювати більш реалістичні та деталізовані зображення об'єктів.

Алгоритми об'ємного рендерингу відіграють у створенні реалістичних і деталізованих зображень тривимірних об'єктів. Вони дозволяють враховувати складну внутрішню структуру об'єктів та створювати ефекти прозорості, відображення та тіні, що робить зображення більш реалістичними та привабливими для сприйняття.

Фотореалістична реконструкція 3D об'єктів



2.9 Алгоритми візуалізації частинок

Алгоритми візуалізації частинок застосовуються до створення ефектів, що з рухом і взаємодією безлічі дрібних об'єктів, званих частками. Ці алгоритми широко застосовуються в комп'ютерній графіці, в іграх, анімації, симуляції та візуалізації фізичних явищ.

Генерація частинок

Перший крок у алгоритмах візуалізації частинок – це генерація самих частинок. Частинки можуть бути створені випадково або з використанням певних правил і параметрів. Наприклад, в алгоритмах візуалізації вогню частинки можуть бути створені в певній області і мати початкову швидкість і напрямок руху.

Оновлення стану частинок

Після генерації частинок їх стан оновлюється на кожному кроці алгоритму. Це включає зміну їхньої позиції, швидкості, прискорення та інших параметрів, які визначають їх рух та взаємодію з навколишнім середовищем або іншими частинками.

Візуалізація частинок

Найважливіший етап алгоритму – це візуалізація частинок. Частинки можуть бути представлені у вигляді точок, спрайтів, текстур чи інших графічних елементів. Їх зовнішній вигляд може змінюватись в залежності від їх стану, наприклад, колір, розмір або прозорість можуть змінюватися під час руху або взаємодії з іншими частинками.

Взаємодія частинок

Частинки можуть взаємодіяти один з одним або з довкіллям. Наприклад, вони можуть стикатися, притягуватися або відштовхуватися один від одного, створюючи ефекти вибухів, диму чи рідин. Алгоритми візуалізації частинок повинні враховувати ці взаємодії та оновлювати стан частинок відповідно до них.

Видалення частинок

Частинки можуть бути видалені зі сцени, коли вони виходять за межі певної області або їхній час життя закінчується. Це дозволяє оптимізувати продуктивність алгоритму та уникнути накопичення великої кількості частинок, які вже не видно чи не впливають на візуалізацію.

Алгоритми візуалізації частинок дозволяють створювати різноманітні ефекти, такі як вогонь, дим, вибухи, пил, водні струмені та багато іншого. Вони відіграють важливу роль у створенні реалістичних та захоплюючих візуальних ефектів у комп'ютерній графіці та анімації.

2.10 Алгоритми візуалізації рідин

Алгоритми візуалізації рідин використовуються для створення реалістичних та динамічних ефектів рідин, таких як вода, олія, кров та інші. Вони відіграють важливу роль у комп'ютерній графіці, анімації, ігровій індустрії та візуальних ефектах.

Симуляція рідини

Для візуалізації рідин необхідно спочатку створити симуляцію рідини. Симуляція рідини ґрунтується на фізичних принципах, таких як закони збереження маси, імпульсу та енергії. Існує кілька підходів до симуляції рідини, включаючи методи на основі частинок, сіткові методи та методи на основі рівнянь Нав'є-Стокса.

Методи на основі частинок представляють рідину як набір частинок, кожна з яких має свої фізичні властивості, такі як маса, швидкість та щільність. Частинки взаємодіють один з одним і з навколишнім середовищем, створюючи ефекти потоку, вихорів та турбулентності.

Сіткові методи представляють рідину як сітку осередків, у яких зберігаються значення фізичних властивостей рідини, такі як щільність та швидкість. Значення в осередках оновлюються з урахуванням рівнянь Навье-Стокса, які описують рух рідини.

Методи з урахуванням рівнянь Навье-Стокса є математичними моделями, які описують рух рідини з урахуванням законів збереження маси, імпульсу та енергії. Розв'язання цих рівнянь дозволяє визначити швидкість та тиск рідини у кожній точці простору.

Візуалізація рідини

Після створення симуляції рідини можна приступити до візуалізації. Візуалізація рідини включає відображення її форми, руху, поверхневих ефектів та інших характеристик.

Для відображення форми рідини часто використовуються методи растеризації чи трасування променів. Методи растеризації являють собою рідину у вигляді набору пікселів на екрані, в той час як методи трасування променів моделюють шлях променів світла через рідину, щоб визначити їх колір та інтенсивність.

Для відображення руху рідини використовуються анімаційні техніки, такі як інтерполяція між кадрами симуляції або використання швидкісних полів для визначення переміщення частинок або сіток.

Поверхневі ефекти, такі як бульбашки, піни та краплі, можуть бути додані за допомогою спеціальних алгоритмів, які моделюють їхню поведінку та взаємодію з рідиною.

Додаткові характеристики рідини, такі як заломлення світла, відбиття та заломлення можуть бути додані за допомогою алгоритмів трасування променів або з використанням спеціальних шейдерів.

В цілому алгоритми візуалізації рідин дозволяють створювати реалістичні та захоплюючі ефекти рідин, які можуть бути використані в різних галузях комп'ютерної графіки та анімації.

○ Висновки до розділу 2

У розділі 2 проаналізовано, що основні алгоритми комп'ютерної графіки є важливим елементом сучасного інформаційного середовища,

віддзеркалюючи високий рівень технологічного прогресу та творчого потенціалу. Використання цих алгоритмів у програмах, від візуалізації даних до ігор та анімації, стає ключовим фактором для створення реалістичних та захоплюючих візуальних ефектів.

Алгоритми обробки зображень, зокрема фільтрація, сегментація та розпізнавання образів, дозволяють покращити якість та зручність роботи з графічним вмістом. У той же час, алгоритми рендерингу, такі як алгоритми трасування променів та методи закріплення тіней, забезпечують реалістичний вигляд сцени.

Завдяки алгоритмам комп'ютерної графіки, користувачі отримують можливість взаємодії з інформацією та віртуальними світами на більш високому рівні. З розвитком технологій, важливим стає оптимізація алгоритмів для забезпечення високої продуктивності та ефективності графічних систем.

Усе це підкреслює важливість постійного дослідження та вдосконалення алгоритмів комп'ютерної графіки, щоб відповідати високим стандартам візуальної якості та функціональності, які сучасні користувачі очікують у цифровому середовищі. Розроблено блок-схему texture mapping алгоритму.

3 РОЗДІЛ
АПАРАТНО-ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

В якості приладу для реклами обрано апаратно-програмний комплекс з пробігаючими рекламними анімаціями. Даний комплекс буде корисним через свою інтерактивність, адже можливо виводити різні анімаційні зображення. Також через велику вартість існуючих аналогів на ринку, великий вибір необхідних деталей для збору та широкі можливості для подальшого удосконалення приладу.

3.2Вибір технології та мови програмування

Так як у даному проекті використовується плата Arduino , для даного проекту обрано онлайн середовище розробки Wokwi(рис. 3.1).

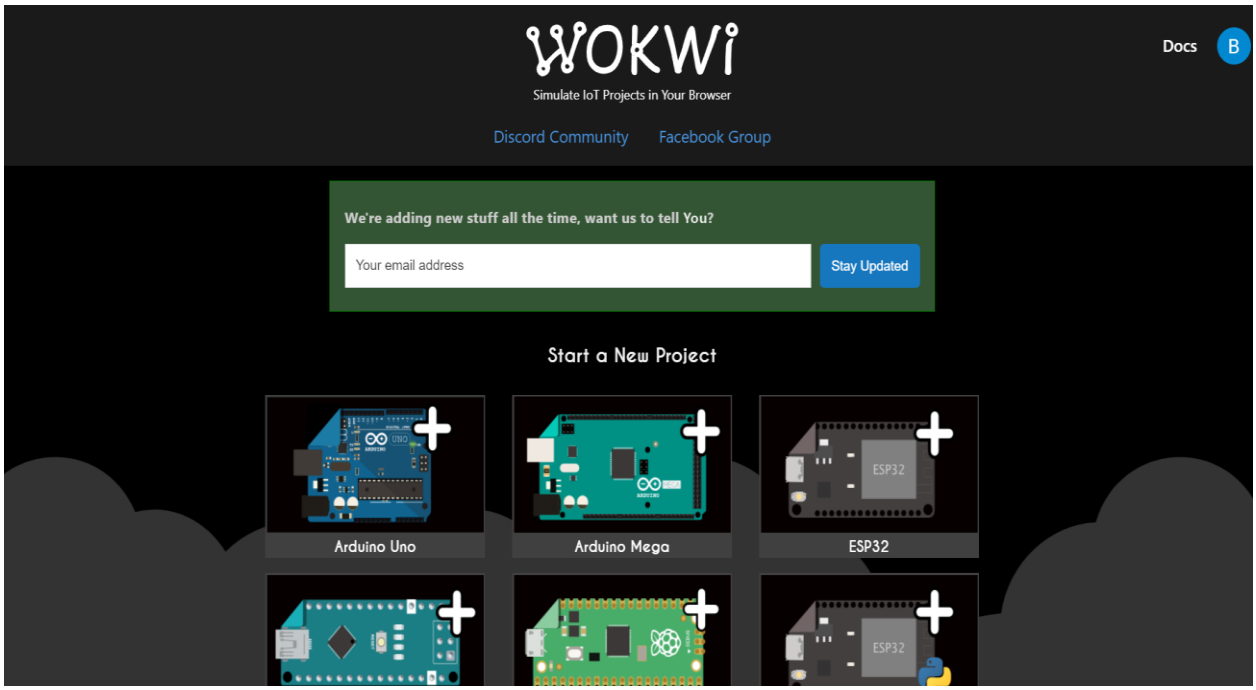


Рисунок 3.1 –Wokwi

Через простоту програм, які написані за допомогою Wokwi, їх називаються скетчами. По суті, це текстові файли, написані мовою Arduino.

Мова програмування Arduino складається з трьох ключових складових. По-перше, це набір функцій, які дозволяють керувати платою. Ці функції дозволяють виконувати аналіз символів, проводити математичні операції та виконувати різноманітні завдання - наприклад, функції digitalRead() та digitalWrite() дозволяють зчитувати та записувати значення.

Кожен скетч, написаний на мові Arduino, має дві основні функції: setUp() та loop(). Виконання скетча завжди розпочинається з setUp(), яка виконується один раз після увімкнення або скидання плати. Після цього виконується loop(), яка дозволяє повторювати програму до тих пір, поки не буде вимкнене живлення або не відбудеться скидання плати.

Також у мові Arduino присутні спеціальні значення, які використовуються для представлення констант і змінних. Більшість типів даних (масиви, булеві значення, символьні рядки, числа з плаваючою комою тощо) аналогічні тим, що використовуються в мові C++. Також можливе виконання операцій з перетворенням типів. Остання частина мови Arduino відома як структура. Вона містить невеликі фрагменти коду, такі як оператори, які дозволяють організувати та керувати логікою програми.

Основні переваги використання Arduino IDE виявляються у його здатності працювати як локальна програма або онлайн-редактор, у швидкості створення скетчів, налаштуванні параметрів плати та використанні інтегрованих бібліотек. Наприклад, ось деякі переваги, які відзначають користувачі цієї системи:

- Характеристики плати – це інструмент, оснащений набором опцій керування платою, де користувачі можуть обирати ту плату, яку вони планують використовувати. При необхідності іншої плати, вони можуть легко змінити варіант, використовуючи розкривне меню. Дані про порти оновлюються автоматично при внесенні змін у плату або при виборі нової моделі.
- Документація – цей засіб надає можливість користувачам створювати детальну документацію своїх проектів. Вона

допомагає відстежувати прогрес та виявляти будь-які зміни, внесені в процесі розробки. Крім того, наявність документації полегшує іншим розробникам використання скетчів на їхніх власних платформах розробки, що сприяє спільному обміну досвідом та ідеями.

- Загальний доступ до ескізів — Arduino IDE надає можливість користувачам обмінюватися своїми скетчами з іншими програмістами. Кожен скетч має унікальне онлайн-посилання, яке можна поділитися з колегами або друзями. Ця корисна функція доступна лише у версії, яка працює у хмарному середовищі.
- Вбудовані бібліотеки - це ключовий елемент програми, яка містить сотні бібліотек, що були розроблені та підтримуються спільнотою Arduino. Вони доступні для використання користувачами без необхідності встановлення будь-яких сторонніх додаткових компонентів.
- Незважаючи на те, що Arduino IDE спочатку розроблений для використання з платами Arduino, він також включає в себе можливість легко підключатися до стороннього обладнання. Це розширює область застосування Arduino IDE і дозволяє використовувати його з різноманітними пристроями, незалежно від їх виробника.

3.2.1 Стандарти синтаксису

Щодо синтаксису, Arduino IDE має свої відмінності від C++. Перша відмінність, яку можна помітити, це використання фігурних дужок для визначення блоків коду. Якщо пропустити закриваючу фігурну дужку після відкриваючої, система видасть помилку. Arduino IDE підсвічує закриваючу дужку після натискання на відкриваючу, що полегшує перевірку. Крім того,

схоже на C++, Arduino також вимагає кінця кожного твердження крапкою з комою, інакше виникне помилка.

Ще одна відмінність полягає у способі коментування. У мові Arduino є два способи коментування залежно від того, чи є це однорядковий або блочний коментар. Щоб закоментувати лише один рядок, потрібно почати його з двох похилих рисок:

```
// a comment here
#define LED_PIN
void setup() {
pinMode(LED_PIN, OUTPUT);
```

Якщо потрібно додати багаторядковий коментар для приміток, можна використати такий формат: почати коментар з поєднання косої риски та зірочки, а завершити його - зірочкою та косою рискою:

```
/* a comment here
a cemment there
there are comments everywhere */
#define LED_PIN
void setup() {
pinMode(LED_PIN, OUTPUT);
```

При додаванні коментарів важливо мати на увазі, що компілятор Wokwi абсолютно ігноруватиме їх. Це означає, що вони не будуть включені у програму для мікроконтролера та не займатимуть пам'ять пристрою.

3.2.2 Опис коду.

1. Прописуємо основні оператори виводу.

```
#define MAX_MILLIWATTS 0
#define BRIGHTNESS     255
#define LED_PIN        2
#define COLOR_ORDER    GRB
#define CHIPSET        WS2812B
#define kMatrixWidth   16
#define kMatrixHeight  16
#define XY_MATRIX      (ROWMAJOR)
#define NUM_LEDS       ((kMatrixWidth) * (kMatrixHeight))
```

2. Реалізація з запитів та фрагментів коду.

```
CRGB fadeTowardColour(CRGB& cur, const CRGB& target, uint8_t amount) {
  nblendU8TowardU8(cur.red,   target.red,   amount);
  nblendU8TowardU8(cur.green, target.green, amount);
  nblendU8TowardU8(cur.blue,  target.blue,  amount);
  return cur;
}
CRGB fadeTowardColour_video(CRGB& cur, const CRGB& target, uint8_t amount) {
  nblendU8TowardU8_video(cur.red,   target.red,   amount);
  nblendU8TowardU8_video(cur.green, target.green, amount);
  nblendU8TowardU8_video(cur.blue,  target.blue,  amount);
  return cur;
}
```

3. Функція зміщення

```
void nblendU8TowardU8(uint8_t& cur, const uint8_t target, uint8_t amount) {
  if (cur == target) return;
  if (cur < target ) {
    uint8_t delta = target - cur;
    delta = scale8(delta, amount);
    cur += delta;
  } else {
    uint8_t delta = cur - target;
    delta = scale8(delta, amount);
    cur -= delta;
  }
}
void nblendU8TowardU8_video(uint8_t& cur, const uint8_t target, uint8_t amount) {
  if (cur == target) return;

  if (cur < target ) {
    uint8_t delta = target - cur;
    delta = scale8_video(delta, amount);
    cur += delta;
  } else {
    uint8_t delta = cur - target;
    delta = scale8_video(delta, amount);
    cur -= delta;
  }
}
}
```

4.Розрахунок масштабного коефіцієнта та масштабованих значень для меншого значення палітри.

```
uint8_t f1 = 255 - offset;
    red1   = scale8_LEAVING_R1_DIRTY(red1,   f1);
    green1 = scale8_LEAVING_R1_DIRTY(green1, f1);
    blue1  = scale8_LEAVING_R1_DIRTY(blue1,  f1);
```

5. Розрахунок для сусіднього значення палітри.

```
uint8_t red2   = entry->red;
    uint8_t green2 = entry->green;
    uint8_t blue2  = entry->blue;
    red2   = scale8_LEAVING_R1_DIRTY(red2,   offset);
    green2 = scale8_LEAVING_R1_DIRTY(green2, offset);
    blue2  = scale8_LEAVING_R1_DIRTY(blue2,  offset);
    cleanup_R1();
```

6. Обмеження частоти кадрів та уточнення часу простою за допомогою вбудованого світлодіода.

```
cfg.frame++;
  uint32_t frame_time = millis() - cfg.millis;  // wraparound safe
  int32_t pause = MS_GOAL - frame_time;
  // if (pause < 0 && SERIAL_UI == 1) { Serial.print(-pause);
Serial.println("ms late"); }
  digitalWrite(LED_BUILTIN, HIGH);
  if (pause > 0) delay(pause);
  digitalWrite(LED_BUILTIN, LOW);
  cfg.millis = millis();
  FastLED.show();
}
```

3.3Вибір компонентів АПЗ

Так само, як і в багатьох інших мовах програмування, у мові Arduino є можливість імпортувати зовнішні бібліотеки. Якщо вбудованих бібліотек недостатньо, ви можете завантажити їх з Інтернету або навіть створити свої власні.

Коротко кажучи, бібліотека - це заздалегідь написаний набір коду, який надає додаткові функціональні можливості.

Структура бібліотеки:

- бібліотека - це папка, що складається з файлів з файлами коду C ++ (.cpp) і файлів заголовків C ++ (.h);
- файл .h описує структуру бібліотеки і оголошує все її змінні і функції;
- файл .cpp містить реалізацію функції.

Якщо потрібно, можна використовувати як загальні бібліотеки C, так і ті, що специфічні для Arduino. Після вибору бібліотеки потрібно її встановити. Щоб додати певну бібліотеку до свого скетчу, слід скористатися директивою #include та вказати назву бібліотеки, яку необхідно використовувати. Важливо пам’ятати, що не потрібно додавати крапку з комою після цього твердження, оскільки це не є твердженням, яке потрібно закінчувати.

Бібліотеки використані в графічному фреймворку

```
#include <Arduino.h>
#include <FastLED.h>
#include <stdint.h>
#include <avr/pgmspace.h>
```

3.3.1 Бібліотека Arduino.h

Arduino.h — це директива препроцесора, яка включає стандартну бібліотеку Arduino у ваш програмний код. Ця бібліотека містить в собі декларації та визначення функцій, типів даних та констант, необхідних для розробки програмного забезпечення для плат Arduino.

Основні функції Arduino.h включають в себе налаштування вводу/виводу, управління пінами, роботу з аналоговими та цифровими сигналами, взаємодію з таймерами та затримками, а також багато іншого.Ця бібліотека є обов'язковою для використання в більшості проектів Arduino і автоматично додається до кожного нового скетчу Arduino IDE. Бібліотека SPI

Бібліотека SPI дозволяє спілкуватися з одним або кількома пристроями SPI (Serial Peripheral Interface). Підтримується тільки головний режим SPI для керування периферійними мікросхемами SPI.

Часто SPI використовується іншими бібліотеками (наприклад, Ethernet), які забезпечують легкий доступ до певного пристрою SPI. Хоча ви можете

використовувати SPI безпосередньо, інші бібліотеки, які додають особливості чіпа, частіше використовуються.

3.3.2 Бібліотека FastLED.h

Бібліотека FastLED.h - це потужний інструмент для керування адресованними світлодіодними стрічками та матрицями з допомогою мікроконтролерів Arduino та інших платформ, що підтримують мову програмування C++. Основними особливостями FastLED.h є швидкість, простота використання та багатофункціональність.

Основні функції та можливості FastLED.h включають в себе:

Підтримка різноманітних типів адресованих світлодіодів: WS2811, WS2812, APA102, SK6812, та інші.

Можливість керувати яскравістю та кольором кожного світлодіода окремо.

Реалізація різних ефектів освітлення, таких як плавна зміна кольору, переливання, мерехтіння, імітація вогню та багато іншого.

Простий інтерфейс програмування, що дозволяє швидко створювати складні ефекти та анімації.

Підтримка роботи з різними типами світлодіодних стрічок та матриць, включаючи одновимірні, двовимірні та тривимірні конфігурації.

Можливість синхронізації керування декількома стрічками світлодіодів для створення складних сцен та візуальних ефектів.

Загалом, бібліотека FastLED.h є незамінним інструментом для розробки світлодіодних проектів на платформі Arduino, який надає широкі можливості для керування світлодіодами та створення захоплюючих ефектів освітлення.

3.3.3 Бібліотека stdint.h

Бібліотека stdint.h - це стандартна бібліотека мови програмування C, яка містить набір визначень для цілочисельних типів з точно визначеною довжиною бітів. Ця бібліотека дозволяє програмістам написати переносимий

код, який не залежить від конкретних архітектур або компіляторів, оскільки розмір цілочисельних типів може варіюватися в залежності від обраної платформи.

Основні типи, визначені в бібліотеці stdint.h, включають:

int8_t: 8-бітне ціле число зі знаком.

uint8_t: 8-бітне ціле число без знаку.

int16_t: 16-бітне ціле число зі знаком.

uint16_t: 16-бітне ціле число без знаку.

int32_t: 32-бітне ціле число зі знаком.

uint32_t: 32-бітне ціле число без знаку.

int64_t: 64-бітне ціле число зі знаком.

uint64_t: 64-бітне ціле число без знаку.

Ці типи дозволяють програмістам явно вказати розмір цілочисельних типів, що сприяє покращенню переносимості коду між різними платформами. Крім того, використання цих типів забезпечує однозначність у розмірі цілих чисел, що дозволяє уникнути проблем з розміром даних при перенесенні коду на різні платформи і компілятори.

Бібліотека stdint.h є важливою складовою стандарту мови програмування C і забезпечує надійну та переносиму роботу з цілочисельними типами даних у програмах на C.

3.3.4 Бібліотека avr/pgmspace.h

Бібліотека avr/pgmspace.h - це частина стандартної бібліотеки Arduino, яка надає функції для роботи з програмною пам'яттю мікроконтролерів AVR. Основна функціональність цієї бібліотеки полягає у забезпеченні доступу до константних даних, які зберігаються у програмній пам'яті (Flash пам'ять) мікроконтролера.

Основні можливості avr/pgmspace.h включають:

Оголошення та використання макросів для зберігання константних даних у програмній пам'яті мікроконтролера. Це дозволяє економити оперативну пам'ять, особливо корисно для пристроїв з обмеженими ресурсами.

Функції для читання даних з програмної пам'яті, такі як pgm_read_byte, pgm_read_word, pgm_read_dword, які дозволяють читати одиночні байти, слова та двійкові дані з програмної пам'яті.

Функції для читання рядків з програмної пам'яті, які дозволяють зручно працювати з текстовими рядками, збереженими у програмній пам'яті.

Використання avr/pgmspace.h дозволяє ефективно використовувати програмну пам'ять мікроконтролера AVR, зменшуючи обсяг оперативної пам'яті, яка може бути важливою для пристроїв з обмеженими ресурсами, такими як мікроконтролери Arduino Uno або Nano.

Ця Опис інтерфейсів АПЗ

3.3.5 Схеми підключення датчиків до Arduino

Спочатку обираємо Arduino nano та натискаємо на нього.

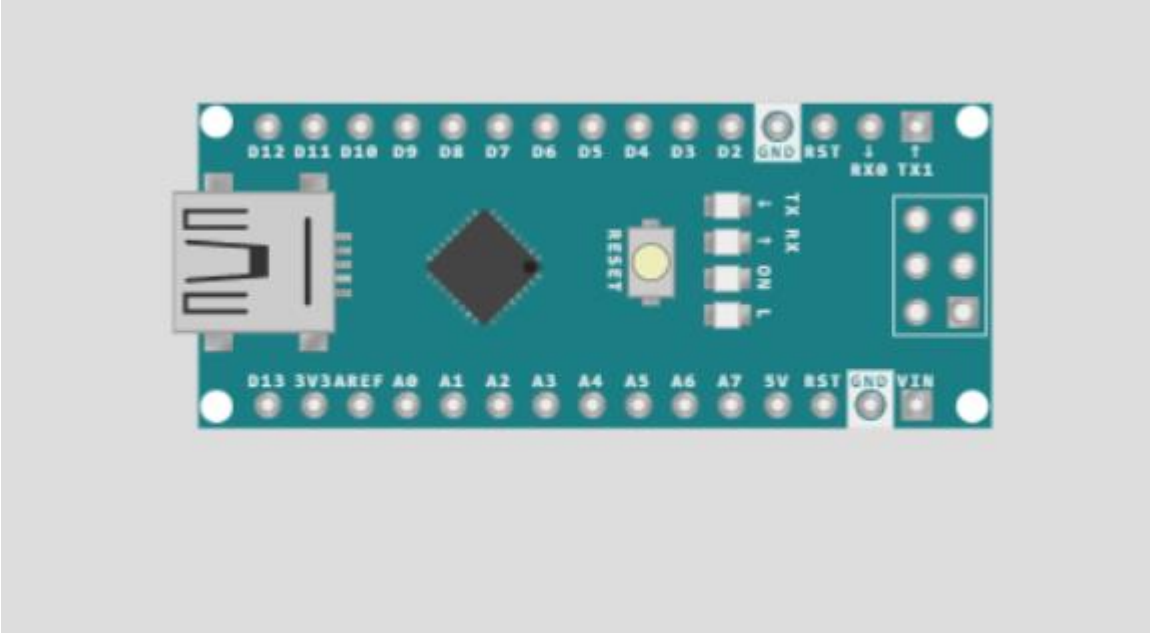


Рисунок 3.2 – Вигляд Arduino nano

Потім обираємо світлодіодну матрицю та з'єднуємо їх

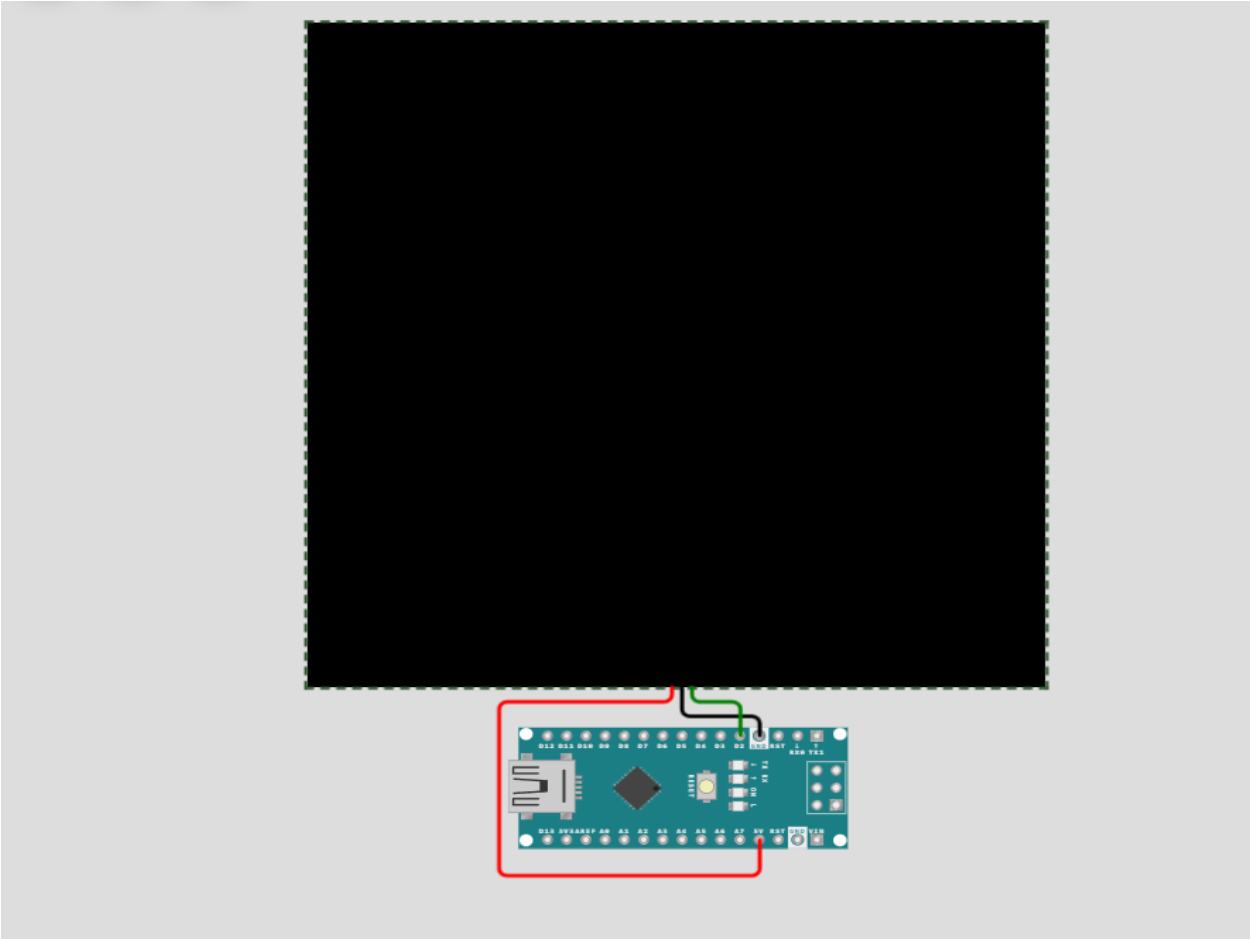


Рисунок 3.3 – Готова структура світлодіодного рекламного дисплею

○ Висновки до розділу 3

В цій частині роботи була обрана мова програмування, апаратні компоненти та бібліотеки для реалізації графічного фреймворку для рекламних дисплеїв. Через те що фреймворк виконано онлайн , то замість звичної Arduino IDE обрано онлайн середовище розробки WOKWI.

Модель оперує інформацією заданою в комплексі. Показано результати збору графічного фреймворку, загальний вигляд . Також описано використані в фреймовку бібліотеки та здійснений певний аналіз коду прописаного при налаштуванні приладу.

Проект може бути використаний як важлива складова для подальших досліджень у сфері зовнішньої реклами, де важливо мати зручні та ефективні

засоби рекламування. Також, з огляду на використання Arduino Nano, він показує можливості високопродуктивного мікроконтролера .

Завдяки легкій розширюваності та гнучкості коду, цей проект може слугувати основою для подальших вдосконалень та розширень, таких як додавання нових функцій, інтеграція з іншими пристроями або розвиток більш складних сценаріїв автоматизації.

РОЗДІЛ4
АНАЛІЗ РЕЗУЛЬТАТІВ ФРЕЙМВОРКУ

4.1 Тест приладу

Створений комплекс передбачає розміщення у окремому приміщенні для моніторингу, що дозволить вести контроль за ефективністю рекламних кампаній. На основі зібраних даних співробітниками установи буде розроблено набір стратегій та правил, спрямованих на збільшення числа клієнтів та залучених учасників.

Основний результат впровадження розробленого графічного фреймворку - підвищення ефективності інтерактивної реклами при зменшенні витрат на неї.

1.1.1 Експерементальна оцінка

За допомогою розробленого приладу були проведені експеримент при якій анімації найбільше затрат на електроенергію.

Анімація	**Енергоспоживання на цикл анімації**	**Час роботи на батареї(год)**	**Поточне споживання(мА)**
Прапор	38	2	27
Привид	46	1	36
Бетмен	44	1	34
Мураха	40	2	30
Крадій	38	2	28
Мавпа	43	1	33
Пташка	38	2	27

Таблиця 4.1 – Результати вимірювань

Отже можливо зробити висновок ,що найкращі показники електрозбереження фреймворк показав на анімації прапора та пташки.

1.1.2 Огляд споживання електроенергії графічним дисплеєм

Аналіз використання електроенергії для Arduino Nano та світлодіодної матриці може бути проведений наступним чином:

Arduino Nano:

Споживання електроенергії: Arduino Nano має дуже низьке споживання енергії у сплячому режимі. У режимі бездіяльності (коли мікроконтролер не виконує жодних операцій), споживання енергії може бути дуже маленьким, в декілька міліампер. У режимі активності, коли мікроконтролер виконує код, споживання енергії може зростати в залежності від обсягу операцій.

Світлодіодна матриця:

Споживання електроенергії: Споживання енергії світлодіодної матриці залежить від кількості та типу світлодіодів, які вона має. Зазвичай споживання електроенергії світлодіодної матриці може бути відносно великим, особливо при використанні більшого розміру матриці або яскравих світлодіодів.

Спільне використання:

Управління живленням: Для зменшення споживання електроенергії можна використовувати методи керування живленням. Наприклад, вимикачі або реле можуть вимикати живлення світлодіодної матриці у разі неактивності, що дозволить зберігати енергію.

Оптимізація програмного забезпечення: Оптимізація програмного забезпечення для Arduino Nano та світлодіодної матриці може допомогти зменшити споживання електроенергії. Наприклад, використання режимів сплячості та оптимізація коду можуть зменшити навантаження на мікроконтролер та споживання енергії.

1.1.3 Основні анімації комплексу

Написане ПЗ дозволяє графічному фреймворку працювати .

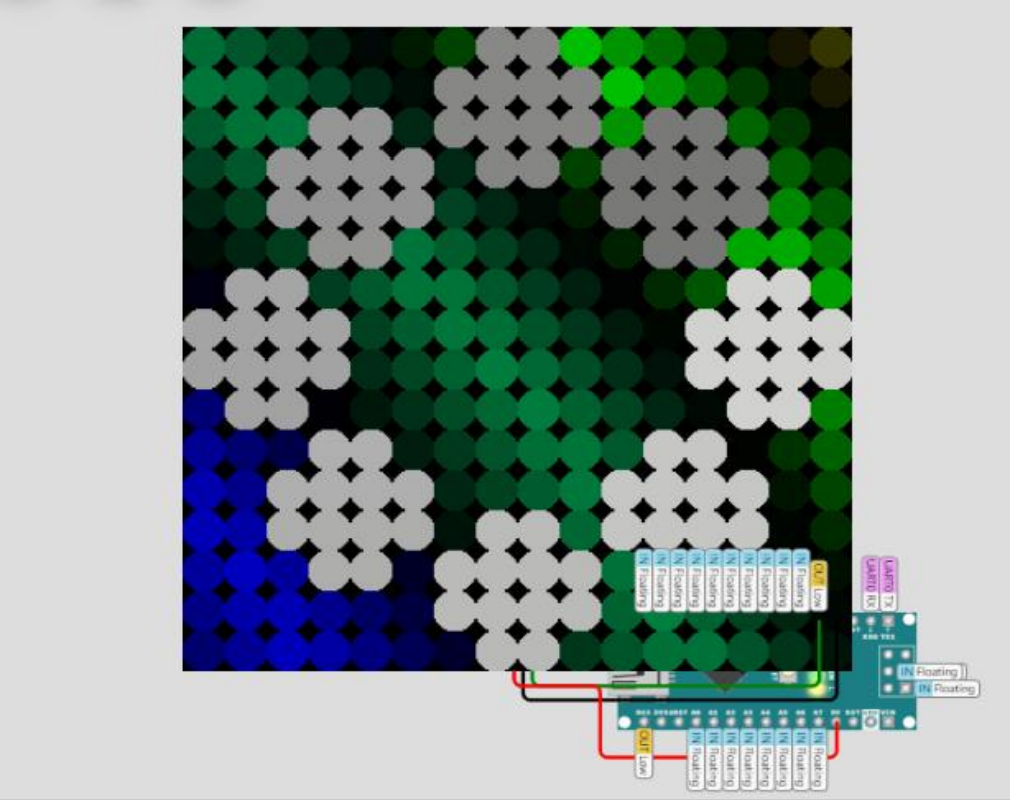


Рисунок 4.1– Старт роботи комплексу

Саме ця частина апарату є найважливішою , адже анімація створена для того,щоб зацікавлювати людей .

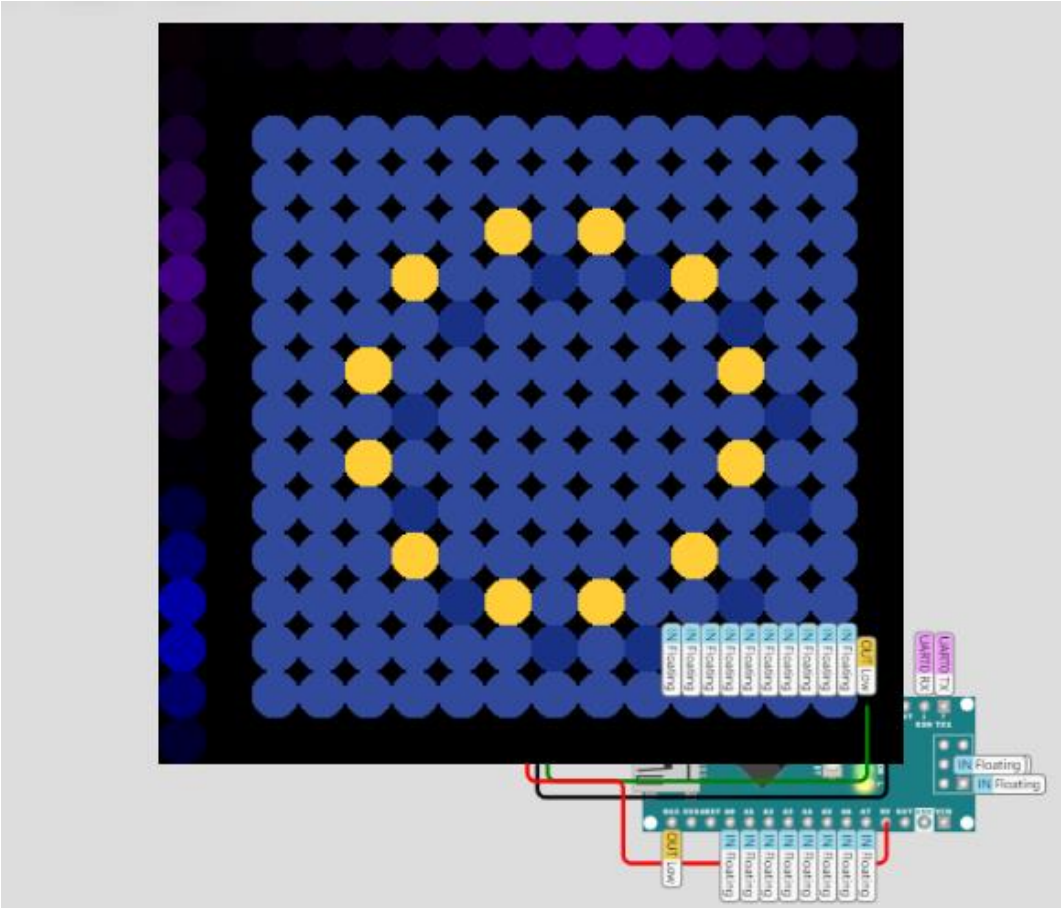


Рисунок 4.2– Друга анімація

Цю анімацію можливо використовувати для привітання різних країн Європейського союзу.Адже при зустрічі делегатів країн їм буде показана анімація ,яка показуватиме ,що ми хочемо хороших відносин з країнами ЄС.

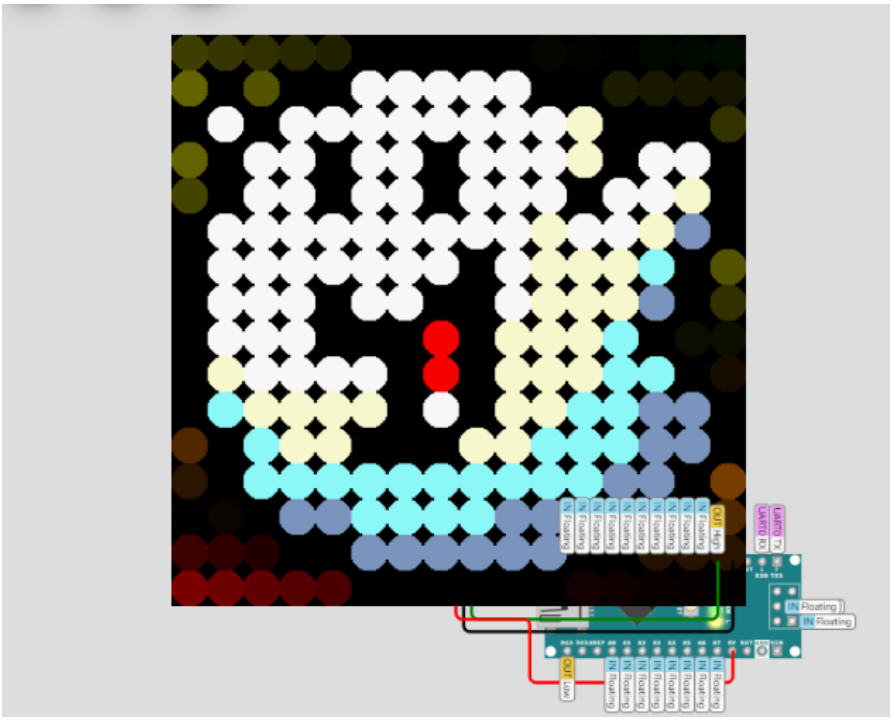


Рисунок 4.3– Анімація для кінотеатру

Цю анімацію можливо використовувати для реклами нових мультфільмів.Адже використана яскрава картинка, яка одразу ж приверне увагу дітей , які є основною цільовою аудиторією цього жанру кінематографу.Тому є сенс запропонувати співпрацю кінотеатрам будь-якого міста України.

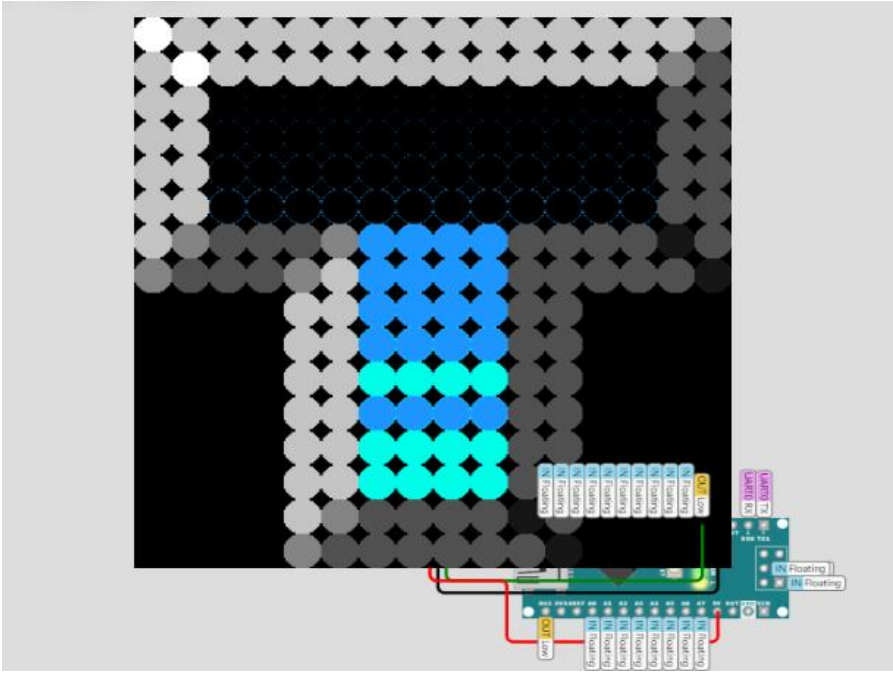


Рисунок 4.4– Анімація для трамвайних та тролейбусних зупинок

Цю анімацію можливо використовувати для реклами трамвайних та тролейбусних зупинок.Адже не завжди люди мають змогу чітко бачити зупинку міського транспорту.Тому така анімація привернула б увагу людей й дала б чітко людям зрозуміти де знаходиться потрібна їм станція.

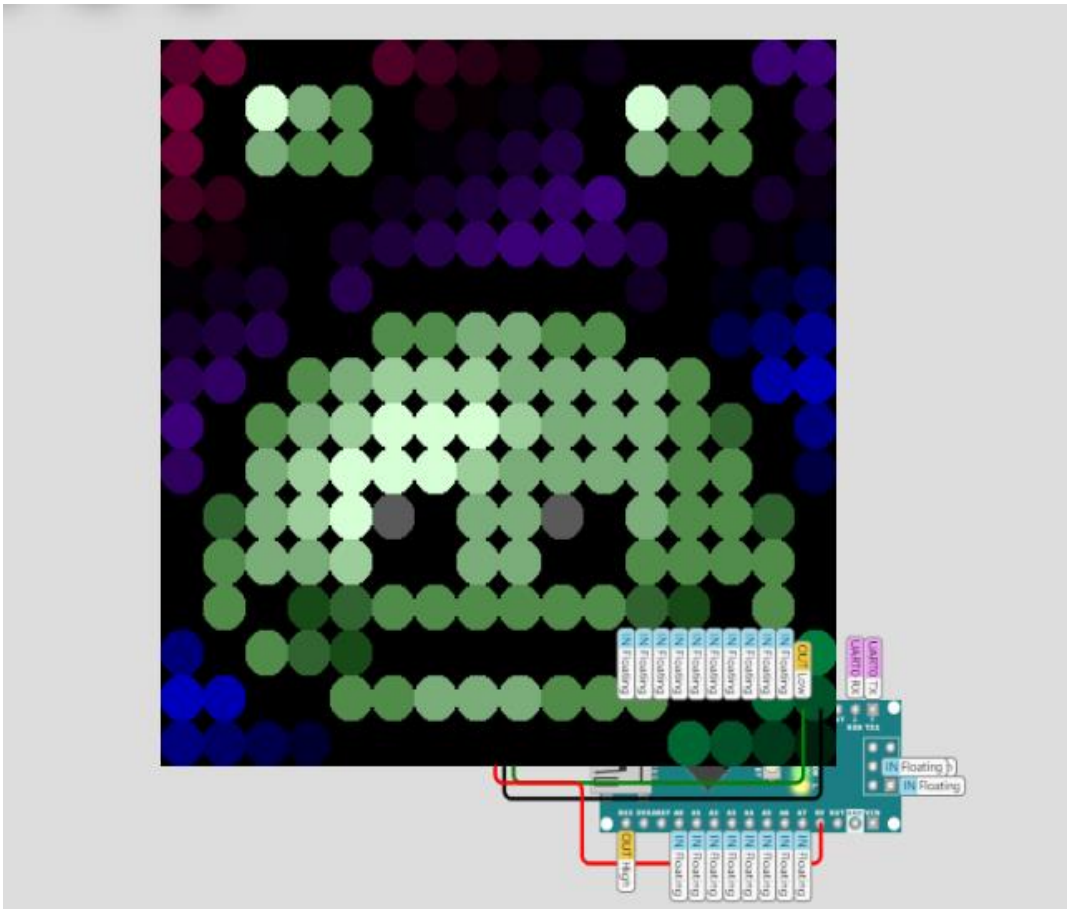


Рисунок 4.4– Анімація для реклами засобів боротьби з комахами

Таку анімацію можна використовувати для реклами засобів боротьби з комахами. Адже чітка картинка мурахи дасть поняття людям, що засіб допоможе в боротьбі з ними.Тому можна запропонувати свої послуги всім основним виробникам засобів проти комах, а також магазинам які займаються продажем цих засобів.

Висновки до розділу 4

У цьому розділі була розглянута робота графічного дисплею, який забезпечує можливість відображення різноманітної інформації з використанням графічних елементів. Оглянувши вживання електроенергії, детально проаналізували параметри та споживання електроенергії пристроїв, що використовують графічний дисплей, що є важливим аспектом в їхньому ефективному функціонуванні та інтеграції з існуючими системами.

З огляду на отримані результати аналізу можна зробити висновок, що послуги анімації дисплею можуть бути цікавими для широкого кола користувачів, зокрема:

Рекламних агентств, що зацікавлені у приверненні уваги клієнтів за допомогою яскравих та привабливих рекламних повідомлень на графічних дисплеях.

Торгових точок та бізнесів, які прагнуть покращити візуальне представлення своїх товарів та послуг шляхом використання анімаційного вмісту на дисплеях.

Організаторів заходів, які хочуть створити вражаючу атмосферу та забезпечити інформаційну підтримку заходу за допомогою графічних дисплеїв з анімаційним вмістом.

Загалом, аналіз використання графічних дисплеїв, їхнього електроживлення та потенційних користувачів анімаційного вмісту надає можливість розуміти переваги та перспективи використання цих технологій у різних сферах діяльності.

Технічні характеристики споживання напруги при різних видах анімацій показали на яких анімаціях найбільша витрата електроенергії ,а на яких найбільші.

Висновки

Розроблено програмно-апаратний комплекс для інтерактивної зовнішньої реклами на базі мікроконтролерної плати Arduino відповідно до поставлених мети та цілей.

Проведений аналіз сучасних видів реклами виявив деякі проблеми, такі як висока вартість, недостатня мобільність та неактуальність. Оцінка переваг існуючих видів показала, що вартість, актуальність та мобільність є ключовими критеріями. Проте для креативної, інтерактивної та максимально ефективної реклами ціна може значно зростати, що є головним недоліком існуючих систем.

Підбір компонентів для комплексу був здійснений на основі їх характеристик та переваг перед аналогами. Обрано мікроконтролер Arduino Uno та світлодіодну матрицю MAX7219, оскільки це найкращий вибір комплектуючих для створення комплексу, що відповідає очікуванням.

Для розроблення програмної частини було використано технологію Arduino та онлайн середовище розробки WOKWI, оскільки в проекті використовується плата Arduino Nano.

Тестування пристрою показало, що комплекс привертає увагу та має певні результати. Отримані результати свідчать про потенціал у подальшому розвитку та удосконаленні технічних можливостей дисплею.

Основний результат впровадження розробленого програмно-апаратного комплексу - підвищення якості реклами, зниження вартості розміщення та впровадження креативних ідей для оголошень.

Щодо можливих поліпшень, рекомендується додати можливість введення реклами через мобільний застосунок, а також зберігання історії рекламних кампаній та їхніх результатів. Додаткові покращення можливі згідно з бажанням та потребами замовника.

Вирішене питання охорони праці та безпеки життєдіяльності після детального ознайомлення та вивчення відповідних вимог і стандартів.

Перелік джерел посилання

1. SparkFun. "Arduino Tutorials". URL: https://learn.sparkfun.com/tutorials/tags/arduino (дата доступу: 06.01.2024).
2. Arduino Forum. URL: https://forum.arduino.cc/ (дата доступу: 05.01.2024).
3. Arduino Project Hub. URL: https://create.arduino.cc/projecthub (дата доступу: 09.01.2024).
4. Adafruit Industries. "Arduino Playground". URL: https://learn.adafruit.com/category/arduino (дата доступу: 02.01.2024).
5. GitHub. "Arduino Repository". URL: https://github.com/arduino/Arduino (дата доступу: 08.01.2024).
6. YouTube. "Arduino Tutorials". URL: https://www.youtube.com/results?search_query=arduino+tutorial (дата доступу: 10.01.2024).
7. Instructables. "Arduino Projects". URL: https://www.instructables.com/howto/arduino/ (дата доступу: 19.01.2024).
8. Arduino Education. URL: https://www.arduino.cc/education (дата доступу: 17.01.2024).
9. Arduino Playground Wiki. URL: https://playground.arduino.cc/Main/ArduinoStarterKit/ (дата доступу: 02.01.2024).
10. Arduino Blog. URL: https://blog.arduino.cc/ (дата доступу: 01.01.2024).
11. Arduino Stack Exchange. URL: https://arduino.stackexchange.com/ (дата доступу: 5.01.2024).

12. Arduino Documentation. URL: https://www.arduino.cc/en/Reference/HomePage (дата доступу: 03.01.2024).
13. Arduino Core Libraries. URL: https://github.com/arduino/ArduinoCore-API (дата доступу: 02.02.2024).
14. Arduino Reddit Community. URL: https://www.reddit.com/r/arduino/ (дата доступу: 07.01.2024).
15. Arduino Project Ideas. URL: https://create.arduino.cc/projecthub/projects (дата доступу: 09.01.2024).
16. Arduino Programming Courses on Udemy. URL: https://www.udemy.com/courses/search/?q=arduino (дата доступу: 12.01.2024).
17. Arduino Projects Book. URL: https://www.arduino.cc/en/Main/ArduinoStarterKit (дата доступу: 13.01.2024).
18. Arduino Magazine. URL: [поставте тут посилання, якщо є] (дата доступу: 16.01.2024).
19. Official Arduino Twitter Account. URL: https://twitter.com/arduino (дата доступу: 12.01.2024).
20. Arduino. "What is an Arduino?". URL: https://www.arduino.cc/en/Guide/Introduction (дата доступу: 04.01.2024).

Додаток А

Лістінг програми

```
      #include <Arduino.h>
      #include <FastLED.h>
      #include <stdint.h>
      #include <avr/pgmspace.h>
      #include "heatshrink_config.h"
      #include "heatshrink_decoder.h"

      enum XY_matrix_config { SERPENTINE = 1, ROWMAJOR = 2, FLIPMAJOR = 4,
FLIPMINOR = 8 };

      // MAX_MILLIWATTS can only be changed at compile-time. Use 0 to disable
limit.
      // Brightness can be changed at runtime via serial with 'b' and 'B'
      #define MAX_MILLIWATTS 0
      #define BRIGHTNESS     255
      #define LED_PIN        2
      #define COLOR_ORDER    GRB
      #define CHIPSET        WS2812B
      #define kMatrixWidth   16
      #define kMatrixHeight  16
      #define XY_MATRIX      (ROWMAJOR)
      #define NUM_LEDS       ((kMatrixWidth) * (kMatrixHeight))

      #define SERIAL_UI      1    // if 1, can be controlled via keypresses in
PuTTY
      #define FADEIN_FRAMES   32    // how long to reach full brightness when
powered on
      #define MS_GOAL        20   // to try maintain 1000 / 20ms == 50 frames
per second

      #define  GIF2H_MAX_PALETTE_ENTRIES  140  //  RAM  is  always  tight  on
ATmega328...
      #define GIF2H_NUM_PIXELS NUM_LEDS    // all images must be the same size
as the matrix

      CRGB leds[NUM_LEDS];
      CRGBPalette16 currentPalette;

      uint16_t XY(uint8_t x, uint8_t y) {
        uint8_t major, minor, sz_major, sz_minor;
        if (x >= kMatrixWidth || y >= kMatrixHeight)
          return NUM_LEDS;
        if (XY_MATRIX & ROWMAJOR)
          major  =  x,  minor  =  y,  sz_major  =  kMatrixWidth,    sz_minor  =
kMatrixHeight;
        else
          major  =  y,  minor  =  x,  sz_major  =  kMatrixHeight,  sz_minor  =
kMatrixWidth;
        if (XY_MATRIX & FLIPMINOR)
          minor = sz_minor - 1 - minor;
```

```
        if ((XY_MATRIX & FLIPMAJOR) ^ (minor & 1 && (XY_MATRIX & SERPENTINE)))
          major = sz_major - 1 - major;
        return (uint16_t) minor * sz_major + major;
      }


      #if FASTLED_USE_PROGMEM == 1
      #define FL_PGM_READ_PTR_NEAR(x) (pgm_read_ptr_near(x))
      #else
      #if !defined(FL_PGM_READ_PTR_NEAR)
      #define FL_PGM_READ_PTR_NEAR(addr) ({typeof(addr) _addr = (addr); *(void
* const *)(_addr); })
      #endif
      #if !defined(memcpy_P)
      #define memcpy_P(dest, src, n) memcpy(dest, src, n)
      #endif
      #endif


      // Stuff from FastLED pull requests and gists that will hopefully be
merged one day

      // Blend one CRGB color toward another CRGB color by a given amount.
      // Blending is linear, and done in the RGB color space.
      // These functions modify 'cur' in place.
      // https://gist.github.com/kriegsman/d0a5ed3c8f38c64adcb4837dafb6e690
      CRGB fadeTowardColour(CRGB& cur, const CRGB& target, uint8_t amount) {
        nblendU8TowardU8(cur.red,   target.red,   amount);
        nblendU8TowardU8(cur.green, target.green, amount);
        nblendU8TowardU8(cur.blue,  target.blue,  amount);
        return cur;
      }
      CRGB  fadeTowardColour_video(CRGB&  cur,  const  CRGB&  target,  uint8_t
amount) {
        nblendU8TowardU8_video(cur.red,   target.red,   amount);
        nblendU8TowardU8_video(cur.green, target.green, amount);
        nblendU8TowardU8_video(cur.blue,  target.blue,  amount);
        return cur;
      }

      // Helper function that blends one uint8_t toward another by a given
amount
      void nblendU8TowardU8(uint8_t& cur, const uint8_t target, uint8_t amount)
{
        if (cur == target) return;

        if (cur < target ) {
          uint8_t delta = target - cur;
          delta = scale8(delta, amount);
          cur += delta;
        } else {
          uint8_t delta = cur - target;
          delta = scale8(delta, amount);
          cur -= delta;
        }
```

```
      }
      void nblendU8TowardU8_video(uint8_t& cur, const uint8_t target, uint8_t
amount) {
        if (cur == target) return;

        if (cur < target ) {
          uint8_t delta = target - cur;
          delta = scale8_video(delta, amount);
          cur += delta;
        } else {
          uint8_t delta = cur - target;
          delta = scale8_video(delta, amount);
          cur -= delta;
        }
      }

      // FastLED uses 4-bit interpolation.  8-bit looks far less janky.
      // https://github.com/FastLED/FastLED/pull/202
      CRGB ColorFromPaletteExtended(const CRGBPalette16& pal, uint16_t index,
uint8_t brightness, TBlendType blendType) {
        // Extract the four most significant bits of the index as a palette
index.
        uint8_t index_4bit = (index >> 12);
        // Calculate the 8-bit offset from the palette index.
        uint8_t offset = (uint8_t)(index >> 4);
        // Get the palette entry from the 4-bit index
        const CRGB* entry = &(pal[0]) + index_4bit;
        uint8_t red1   = entry->red;
        uint8_t green1 = entry->green;
        uint8_t blue1  = entry->blue;

        uint8_t blend = offset && (blendType != NOBLEND);
        if (blend) {
          if (index_4bit == 15) {
            entry = &(pal[0]);
          } else {
            entry++;
          }

          // Calculate the scaling factor and scaled values for the lower
palette value.
          uint8_t f1 = 255 - offset;
          red1   = scale8_LEAVING_R1_DIRTY(red1,   f1);
          green1 = scale8_LEAVING_R1_DIRTY(green1, f1);
          blue1  = scale8_LEAVING_R1_DIRTY(blue1,  f1);

          // Calculate the scaled values for the neighbouring palette value.
          uint8_t red2   = entry->red;
          uint8_t green2 = entry->green;
          uint8_t blue2  = entry->blue;
          red2   = scale8_LEAVING_R1_DIRTY(red2,   offset);
          green2 = scale8_LEAVING_R1_DIRTY(green2, offset);
          blue2  = scale8_LEAVING_R1_DIRTY(blue2,  offset);
          cleanup_R1();
```

```
          // These sums can't overflow, so no qadd8 needed.
          red1   += red2;
          green1 += green2;
          blue1  += blue2;
        }
        if (brightness != 255) {
          // nscale8x3_video(red1, green1, blue1, brightness);
          nscale8x3(red1, green1, blue1, brightness);
        }
        return CRGB(red1, green1, blue1);
      }
      CRGB ColorFromPaletteExtended(const CRGBPalette32& pal, uint16_t index,
uint8_t brightness, TBlendType blendType) {
        // Extract the five most significant bits of the index as a palette
index.
        uint8_t index_5bit = (index >> 11);
        // Calculate the 8-bit offset from the palette index.
        uint8_t offset = (uint8_t)(index >> 3);
        // Get the palette entry from the 5-bit index
        const CRGB* entry = &(pal[0]) + index_5bit;
        uint8_t red1   = entry->red;
        uint8_t green1 = entry->green;
        uint8_t blue1  = entry->blue;

        uint8_t blend = offset && (blendType != NOBLEND);
        if (blend) {
          if (index_5bit == 31) {
            entry = &(pal[0]);
          } else {
            entry++;
          }

          // Calculate the scaling factor and scaled values for the lower
palette value.
          uint8_t f1 = 255 - offset;
          red1   = scale8_LEAVING_R1_DIRTY(red1,   f1);
          green1 = scale8_LEAVING_R1_DIRTY(green1, f1);
          blue1  = scale8_LEAVING_R1_DIRTY(blue1,  f1);

          // Calculate the scaled values for the neighbouring palette value.
          uint8_t red2   = entry->red;
          uint8_t green2 = entry->green;
          uint8_t blue2  = entry->blue;
          red2   = scale8_LEAVING_R1_DIRTY(red2,   offset);
          green2 = scale8_LEAVING_R1_DIRTY(green2, offset);
          blue2  = scale8_LEAVING_R1_DIRTY(blue2,  offset);
          cleanup_R1();

          // These sums can't overflow, so no qadd8 needed.
          red1   += red2;
          green1 += green2;
          blue1  += blue2;
        }
        if (brightness != 255) {
          // nscale8x3_video(red1, green1, blue1, brightness);
```

```
    nscale8x3(red1, green1, blue1, brightness);
  }
  return CRGB(red1, green1, blue1);
}

typedef struct HSprite {
  uint16_t datasize;
  uint16_t frames;
  uint16_t duration;
  uint8_t flags;
  uint8_t palette_entries;
  CRGB palette[];
  uint8_t hs_data[];
} HSprite;

#include "gifs.h"

FL_PROGMEM extern const HSprite *const hsprite_list[] = {
  (HSprite*) &HSpr_35disk,
  (HSprite*) &HSpr_load,
  (HSprite*) &HSpr_vn,
  (HSprite*) &HSpr_unionflag16,
  (HSprite*) &HSpr_eu,
  (HSprite*) &HSpr_ghost,
  (HSprite*) &HSpr_scheisse,
  (HSprite*) &HSpr_devil_nod,
  (HSprite*) &HSpr_kputrummis,
  (HSprite*) &HSpr_rjb,
  (HSprite*) &HSpr_owl,
  (HSprite*) &HSpr_twylogo,
  (HSprite*) &HSpr_gear,
  (HSprite*) &HSpr_16x16_oric,
  (HSprite*) &HSpr_zx_k_ref,
  (HSprite*) &HSpr_invader,
  (HSprite*) &HSpr_c64,
  (HSprite*) &HSpr_amigaroll_trs,
  (HSprite*) &HSpr_joystick,
  (HSprite*) &HSpr_slime,
  (HSprite*) &HSpr_DigDug16x16,
  (HSprite*) &HSpr_octorokblue,
  (HSprite*) &HSpr_ptititi,
  (HSprite*) &HSpr_burgertime,
  (HSprite*) &HSpr_pouet_avatar_poi_charly_walk2,
  (HSprite*) &HSpr_rockhell,
  (HSprite*) &HSpr_pouet_avatar_mario,
  (HSprite*) &HSpr_kirby_run,
  (HSprite*) &HSpr_bub,
  (HSprite*) &HSpr_sirlord,
  (HSprite*) &HSpr_plane,
  (HSprite*) &HSpr_fire,
  (HSprite*) &HSpr_bird16,
  (HSprite*) &HSpr_mspacman,
  (HSprite*) &HSpr_ghost3,
  (HSprite*) &HSpr_batman2,
  (HSprite*) &HSpr_lemming,
```

```
        (HSprite*) &HSpr_mwk,
        (HSprite*) &HSpr_gwain2,
        (HSprite*) &HSpr_scoopexrulez,
        (HSprite*) &HSpr_8bit_avatar,
      };
      #define HSPRITES_N (sizeof(hsprite_list) / sizeof(hsprite_list[0]))


      typedef struct {
        uint8_t xw, yw, xd, yd;
        int16_t xdd, ydd, bcd;
      } RainbowSmoothiePreset;
      typedef struct {
        uint32_t frame = 0;
        uint32_t millis = 0;
        uint8_t effect = 2;
        uint8_t palette_n = 1;
        uint8_t alpha_fade = 64;
        uint8_t brightness = BRIGHTNESS;
        uint8_t rs_preset_n = 0;
        TBlendType currentBlending = LINEARBLEND;
        uint8_t extendedmixing = 1;
        RainbowSmoothiePreset rs;
      } Config;
      Config cfg;

      // /*__attribute__ ((section(".noinit")))*/ uint8_t * nv_preset;
      void setup() {
        FastLED.addLeds<CHIPSET, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS);
        FastLED.setCorrection(UncorrectedColor);
        FastLED.setTemperature(UncorrectedTemperature);
        FastLED.setDither(DISABLE_DITHER);
        // FastLED.setMaxRefreshRate(400);
        if                (MAX_MILLIWATTS                >                0)
FastLED.setMaxPowerInMilliWatts(MAX_MILLIWATTS);

        if (SERIAL_UI == 1) {
          Serial.begin(250000);
          halp();
        }

        pinMode(LED_BUILTIN, OUTPUT);
        inc_palette(0);
        inc_rs_preset(0);
      }

      #define MAX_EFFECTS 3
      void loop() {
        uint8_t effect = cfg.effect + 1;
        if (effect & 1) rainbow_smoothie();
        if (effect & 2) scintillating_heatshrink();
        if (SERIAL_UI == 1) poll_serial();
        if (cfg.frame <= FADEIN_FRAMES)
          FastLED.setBrightness(scale8(cfg.brightness,  (cfg.frame  *  255)  /
FADEIN_FRAMES));
```

```
        // cap the frame rate and indicate idle time via the built-in LED
        cfg.frame++;
        uint32_t frame_time = millis() - cfg.millis;  // wraparound safe
        int32_t pause = MS_GOAL - frame_time;
        //  if  (pause  <  0  &&  SERIAL_UI  ==  1)  {  Serial.print(-pause);
Serial.println("ms late"); }
        digitalWrite(LED_BUILTIN, HIGH);
        if (pause > 0) delay(pause);
        digitalWrite(LED_BUILTIN, LOW);
        cfg.millis = millis();
        FastLED.show();
      }

      ////////////////////////////////////////////////////////////////////////
      ////////////////////////////////////////////////////////////////////////

      void poll_serial() {
        if (Serial.available() <= 0)
          return;
        int input = Serial.read();
        switch (input) {
          case 'e':
            cfg.effect = addmod8(cfg.effect, 1, MAX_EFFECTS);
            break;
          case 'f':
            cfg.alpha_fade = qsub8(cfg.alpha_fade, 1);
            break;
          case 'F':
            cfg.alpha_fade = qadd8(cfg.alpha_fade, 1);
            break;
          case '{':
            cfg.currentBlending = LINEARBLEND;
            break;
          case '}':
            cfg.currentBlending = NOBLEND;
            break;
          case '[':
            cfg.extendedmixing = 1;
            break;
          case ']':
            cfg.extendedmixing = 0;
            break;
          case 'p':
            inc_palette(1);
            break;
          case '~':
            inc_rs_preset(1);
            break;
          case 'B':
            cfg.brightness = qadd8(cfg.brightness, 1);
            FastLED.setBrightness(cfg.brightness);
            break;
          case 'b':
            cfg.brightness = qsub8(cfg.brightness, 1);
```

```
            FastLED.setBrightness(cfg.brightness);
            break;
          case 'd':
            FastLED.setDither(DISABLE_DITHER);
            break;
          case 'D':
            FastLED.setDither(BINARY_DITHER);
            break;
          case ',':
            cfg.rs.bcd -= 64;
            break;
          case '<':
            cfg.rs.bcd -= 256;
            break;
          case '.':
            cfg.rs.bcd += 64;
            break;
          case '>':
            cfg.rs.bcd += 256;
            break;
          case '#':
            randomise_rainbowsmoothie();
            break;
          case 13:
            print_params();
            break;
          case 10:
            break;
          default:
            Serial.println(F("?"));
        }
      }
      void halp() {
        Serial.println(F(" #\trandomise rainbow smoothie\r\n"
                         " ~\tchoose next preset for rainbow smoothie\r\n"
                         " <enter>\tprint parameters\r\n"
                         " e\tswitch to next Effect\r\n"
                         " p\tswitch to next palette\r\n"
                         " {}\tturn extended palette blending on/off\r\n"
                         " []\tturn linear palette blending on/off\r\n"
                         " ,.<>\tchange colour cycling rate\r\n"
                         " bB\tbrightness down/up\r\n"
                         " dD\tbinary dither off/on\r\n"
                         " fF\tdecrease/increase fade out rate\r\n"));
      }
      void print_params() {

Serial.println(F("bright\tfade\txw\tyw\txd\tyd\txdd\tydd\tbcd\tFPS"));
        Serial.print(cfg.brightness); Serial.print("\t");
        Serial.print(cfg.alpha_fade); Serial.print("\t");
        Serial.print(cfg.rs.xw); Serial.print("\t");
        Serial.print(cfg.rs.yw); Serial.print("\t");
        Serial.print(cfg.rs.xd); Serial.print("\t");
        Serial.print(cfg.rs.yd); Serial.print("\t");
        Serial.print(cfg.rs.xdd); Serial.print("\t");
```

```
  Serial.print(cfg.rs.ydd); Serial.print("\t");
  Serial.print(cfg.rs.bcd); Serial.print("\t");
  Serial.println(FastLED.getFPS());
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

// FL_PROGMEM extern const TProgmemPalette16 myRedWhiteBluePalette_p = {
//     CRGB::Red,    CRGB::Gray,    CRGB::Blue,    CRGB::Black,
//     CRGB::Red,    CRGB::Gray,    CRGB::Blue,    CRGB::Black,
//     CRGB::Red,    CRGB::Red,     CRGB::Gray,    CRGB::Gray,
//     CRGB::Blue,   CRGB::Blue,    CRGB::Black,   CRGB::Black
// };
// FL_PROGMEM extern const TProgmemPalette16 myBlackWhitePalette_p = {
//     CRGB::Black,    CRGB::Black,    CRGB::White,    CRGB::Grey,
//     CRGB::Black,    CRGB::Black,    CRGB::White,    CRGB::Grey,
//     CRGB::Black,    CRGB::Black,    CRGB::White,    CRGB::Grey,
//     CRGB::Black,    CRGB::Black,    CRGB::White,    CRGB::Grey,
// };
FL_PROGMEM extern const TProgmemRGBPalette16 RainbowStripeColors2_p = {
  0xFF0000, 0x7F0000, 0xAB5500, 0x552A00,
  0xABAB00, 0x555500, 0x00FF00, 0x007F00,
  0x00AB55, 0x00552A, 0x0000FF, 0x00007F,
  0x5500AB, 0x2A0055, 0xAB0055, 0x55002A
};

// DEFINE_GRADIENT_PALETTE(froth616_gp) {
//             //             http://soliton.vm.bytemark.co.uk/pub/cpt-
city/fractint/base/tn/froth616.png.index.html
// // Size: 116 bytes of program space.
//     0, 247,  0,247,
//    17,  75,  0, 79,
//    33, 247,  0,  0,
//    51,  75,  0,  0,
//    68, 247,248,  0,
//    84,  75, 91,  0,
//   102,   0,248,  0,
//   119,   0, 91,  0,
//   135,   0,  0,247,
//   153,   0,  0, 79,
//   170,   0,248,247,
//   186,   0, 91, 79,
//   204,   0,  0,  0,
//   237,   0,  0,  0,
//   255, 247,248,247};

FL_PROGMEM extern const TProgmemRGBPalette16 *const palettes16[] = {
  &RainbowColors_p, &RainbowStripeColors_p, &RainbowStripeColors2_p,
  // &HeatColors_p, &CloudColors_p, &LavaColors_p, &OceanColors_p,
  // &ForestColors_p, &PartyColors_p,
  // &myRedWhiteBluePalette_p, &myBlackWhitePalette_p
};
#define RGB16PALETTES (sizeof(palettes16) / sizeof(palettes16[0]))
FL_PROGMEM extern const TProgmemRGBGradientPalettePtr palettesGrad[] = {
```

```
        // froth616_gp,
      };
      #define         RGBGRADPALETTES         (sizeof(palettesGrad)         /
sizeof(TProgmemRGBGradientPalettePtr))

      void inc_palette(uint8_t n) {
        uint8_t total_palettes = RGB16PALETTES + RGBGRADPALETTES;
        cfg.palette_n = n = addmod8(cfg.palette_n, n, total_palettes);
        if (n < RGBGRADPALETTES) {
          currentPalette           =           (TProgmemRGBGradientPalettePtr)
FL_PGM_READ_PTR_NEAR(&palettesGrad[n]);
          return;
        }
        n -= RGBGRADPALETTES;
        if (n < RGB16PALETTES) {
          currentPalette          =          *(TProgmemRGBPalette16          *)
FL_PGM_READ_PTR_NEAR(&palettes16[n]);
          return;
        }
      }

      ////////////////////////////////////////////////////////////////////////
      ////////////////////////////////////////////////////////////////////////

      FL_PROGMEM const RainbowSmoothiePreset rs_presets[] = {
        //xw yw   xd   yd     xdd     ydd    bcd
        { 4,  3,  80, 208,  15327,  -3010,  27000},
        // { 4,  4,  14,  14,   6514, -16076,  16384},
        { 1,  1,  34,  86,  -7661,  30835,  9600},
        // { 7, 12, 105,  46,   5563,   5313,-10240},
        // {19, 19, 200, 237,  20142,  14656, -2857},
        // {20,  6, 116,  33,  -8683, -10095, -4633},
        // { 4,  3,  16,  48,  15347,   6636,  3331},
        // { 5, 10,   2,  44,  -7661,  30835, -2414},
        // { 3,  3,  64, 231,   8691,   7618,  3795},
        // { 7,  5,  36, 111,  25249,   4738, -1283},
        // { 4,  8, 255, 255, -32767, -32690, -6234},
        // { 2,  5, 173, 103, -18586,  14361,  3788},
      };
      #define          MAX_RS_PRESETS          (sizeof(rs_presets)          /
sizeof(RainbowSmoothiePreset))

      void inc_rs_preset(uint8_t n) {
        cfg.rs_preset_n = addmod8(cfg.rs_preset_n, n, MAX_RS_PRESETS);
        uint8_t* src = (uint8_t *) &rs_presets[cfg.rs_preset_n];
        uint8_t* dst = (uint8_t *) &cfg.rs;
        memcpy_P(dst, src, sizeof(cfg.rs));
      }

      void rainbow_smoothie() {
        uint32_t startHue = cfg.frame * cfg.rs.bcd;
        uint32_t  yHueDelta  =  ((uint32_t)sin16(cfg.frame  *  cfg.rs.xd)  *
cfg.rs.xw);
        uint32_t  xHueDelta  =  ((uint32_t)cos16(cfg.frame  *  cfg.rs.yd)  *
cfg.rs.yw);
```

```
        uint32_t lineStartHue = startHue - (kMatrixHeight / 2) * yHueDelta;
        uint32_t yd2 = cfg.rs.ydd * 2048;
        uint32_t xd2 = cfg.rs.xdd * 2048;
        for (byte y = 0; y < kMatrixHeight; y++) {
          lineStartHue += yHueDelta;
          yHueDelta += yd2;
          uint32_t pixelHue = lineStartHue - (kMatrixWidth / 2) * xHueDelta;
          uint32_t xhd = xHueDelta;
          for (byte x = 0; x < kMatrixWidth; x++) {
            pixelHue += xhd;
            xhd += xd2;
            if (cfg.extendedmixing == 1) {
              leds[XY(x,   y)]   =   ColorFromPaletteExtended(currentPalette,
pixelHue >> 7, 255, cfg.currentBlending);
            } else {
              leds[XY(x,  y)]  =  ColorFromPalette(currentPalette,  pixelHue  >>
15, 255, cfg.currentBlending);
            }
          }
        }
      }

      void randomise_rainbowsmoothie() {
        random16_add_entropy(random());
        cfg.rs.bcd = random16(16383) - 8192;
        cfg.rs.xd = random8();
        cfg.rs.yd = random8();
        cfg.rs.xdd = random16() - 32768;
        cfg.rs.ydd = random16() - 32768;
        cfg.rs.xw = random8(24);
        cfg.rs.yw = random8(24);
      }

      struct hstatus {
        uint32_t last_millis;
        heatshrink_decoder hsd;
        size_t heatsunk;
        uint16_t frame;
        uint8_t hs_spr_n = -1;
        uint8_t loops;
        CRGB palette[GIF2H_MAX_PALETTE_ENTRIES];
      } hstatus;

      // HStatus hstatus;

      void heatshrunk_sprite_reset(struct hstatus * status, uint8_t hs_spr_n)
{
        // decompress the palette and leave the decompressor ready to emit
pixels
        status->hs_spr_n = hs_spr_n;
        status->heatsunk = 0;
        status->frame = 0;
        HSprite           *spr_ptr           =           (HSprite           *)
FL_PGM_READ_PTR_NEAR(&hsprite_list[hs_spr_n]);
        // uint16_t datasize = FL_PGM_READ_WORD_NEAR(&spr_ptr->datasize);
```

```
        uint8_t     palette_entries     =     FL_PGM_READ_BYTE_NEAR(&spr_ptr-
>palette_entries);

        // set heatshrink's window buffer position, so the pixels end up aligned
with the start of the buffer
        heatshrink_decoder_reset(&status->hsd);
        status->hsd.head_index -= palette_entries * sizeof(CRGB);

        uint8_t *dst = (uint8_t *) &status->palette;
        uint16_t remaining = palette_entries * sizeof(CRGB);
        size_t count = 0;
        while (remaining) {
          heatshrink_decoder_sinkP(&status->hsd,    &spr_ptr->hs_data[status-
>heatsunk],
                                   HEATSHRINK_STATIC_INPUT_BUFFER_SIZE,
&count);
          status->heatsunk += count;
          HSD_poll_res pres;
          do {
            pres  =  heatshrink_decoder_poll(&status->hsd,  dst,  remaining,
&count);
            dst += count;
            remaining -= count;
          } while ((pres == HSDR_POLL_MORE) && remaining);
        }
      }

      void  heatshrunk_sprite_prepare(struct  hstatus  *  status,  uint8_t
hs_spr_n) {
        HSprite          *spr_ptr          =          (HSprite          *)
FL_PGM_READ_PTR_NEAR(&hsprite_list[hs_spr_n]);
        uint16_t datasize = FL_PGM_READ_WORD_NEAR(&spr_ptr->datasize);
        uint16_t frames = FL_PGM_READ_WORD_NEAR(&spr_ptr->frames);
        uint16_t duration = FL_PGM_READ_WORD_NEAR(&spr_ptr->duration);
        // uint8_t flags = FL_PGM_READ_BYTE_NEAR(&spr_ptr->flags);

        if (hs_spr_n != status->hs_spr_n) {
          // a different sprite has been selected;  force a reset
          status->frame = frames;
          status->loops = 0;
          duration = 0;
        }

        // time to decompress the next frame?
        if ((cfg.millis - status->last_millis) < duration) return;
        if (frames == 1 && status->loops != 0) return;

        // need to loop/reset first?
        if (status->frame >= frames) {
          heatshrunk_sprite_reset(status, hs_spr_n);
          status->loops += 1;
        }

        status->frame++;
        status->last_millis = cfg.millis;
```

```
        size_t count = 0;
        uint16_t remaining = GIF2H_NUM_PIXELS;
        while (remaining) {
          if (status->heatsunk < datasize) {
            heatshrink_decoder_sinkP(&status->hsd,  &spr_ptr->hs_data[status-
>heatsunk],
                                     datasize - status->heatsunk, &count);
            status->heatsunk += count;
          }
          HSD_poll_res pres;
          do {
            pres   =   heatshrink_decoder_poll(&status->hsd,   0,   remaining,
&count);
            remaining -= count;
          } while ((pres == HSDR_POLL_MORE) && remaining);
          if (0 && (status->heatsunk >= datasize)) {
            heatshrink_decoder_finish(&status->hsd);   // not strictly needed
when using static buffers
          }
        }
      }

      void heatshrunk_sprite_plot(int8_t xstart, int8_t ystart, struct hstatus
* status, const uint8_t fade_speed) {
        CRGB *pal = (CRGB *) status->palette;
        uint8_t         *pixels         =         status->hsd.buffers         +
HEATSHRINK_STATIC_INPUT_BUFFER_SIZE;
        // plot each sprite pixel modulo the size of the matrix, i.e. wrap the
image
        for (uint8_t y = 0; y < kMatrixHeight; y++) {
          for (uint8_t x = 0; x < kMatrixWidth; x++) {
            uint16_t led_index = XY(addmod8(x, xstart, kMatrixWidth),
                                    addmod8(y, ystart, kMatrixHeight));
            uint8_t palette_index = *pixels++;
            if (palette_index > 0 && cfg.effect != 1) {
              // non-0 palette entry;  copy the palette entry to the LED
              leds[led_index] = pal[palette_index];
            } else {
              // palette entry 0 is the mask;  fade this LED to the mask colour
(black usually)
              fadeTowardColour(leds[led_index],           pal[palette_index],
fade_speed);
            }
          }
        }
      }

      void scintillating_heatshrink() {
        static uint8_t spr_n = 0;
        static uint32_t last_spr_change = 0;
        if ((hstatus.loops > 0) && (cfg.millis - last_spr_change > 2500)) {
          last_spr_change = cfg.millis;
          spr_n = addmod8(spr_n, 1, HSPRITES_N);
        }
        heatshrunk_sprite_prepare(&hstatus, spr_n);
```

```
        heatshrunk_sprite_plot(0 /*- (cos8(cfg.frame & 0x7f) >> 3)*/, 0,
&hstatus, cfg.alpha_fade);
      }
```