UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki

Adam Makiewicz
nr albumu: 235281

Generowanie płytek PCB jako dodatek do programu graficznego Blender

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
dr Piotr Arłukowicz

Gdańsk 2020

Streszczenie

Istnieje wiele programów do projektowania obwodów elektronicznych, jednak żadne z nich, z uwagi na swoje ścisłe zastosowania, nie posiada odpowiednich narzędzi do zaawansowanego renderowania i produkcji materiałów o charakterze edukacyjnym i prezentacyjnym. Popularny program do tworzenia grafiki 3D — Blender, dzięki możliwości rozbudowania go o dodatki jest narzędziem mogącym wspomagać ten proces.

Niniejsza praca przedstawia proces tworzenia i implementacji dodatku do programu Blender. Narzędzie ma na celu tworzenie wizualizacji i wspomaganie projektowania płytek drukowanych, używanych w procesie tworzenia obwodów elektronicznych. Pierwszy rozdział przedstawia wymagane funkcje programu oraz opis środowiska i technologii wykorzystanych w pracy. Przedstawione są formaty i omówiona budowa plików projektowych, wchodzących w skład schematów elektronicznych, które są danymi wejściowymi dla tworzonego systemu.

Kolejny rozdział konkretyzuje wymagania projektowe, wynikające bezpośrednio z opisanych wcześniej, niezbędnych funkcjonalności. Sprecyzowane wymagania są konieczne i pozwalają podzielić rozwiązanie w procesie implementacji na niezależne moduły. Rozdział drugi opisuje także strukturę interfejsu Blendera, jak i całego stworzonego projektu w kontekście implementacji wzorca projektowego i wzajemnych zależności modułów.

Rozdział trzeci przedstawia implementację modułów tworzonego narzędzia, równocześnie z przedstawieniem kolejności przepływu danych do poszczególnych komponentów i funkcji. Kolejność podrozdziałów przedstawia proces wykonywany w aplikacji.

Ostatni rozdział podsumowuje całokształt wykonanej pracy oraz przedstawia użycie dodatku na przykładowych plikach projektowych. Przedstawione zostają także możliwości dalszego rozwoju systemu.

Słowa kluczowe

Elektronika, 3D, Blender, Addon, Python, PCB, RS-274X, Gerber

Spis treści

Wprowadzenie

Obwody drukowane lub płytki drukowane (zwane dalej "PCB", ang. *Printed Circuit Board*) to podstawa dla każdego modułu elektronicznego. Są to płytki izolacyjne z połączeniami elektrycznymi i punktami lutowniczymi, służącymi do montażu podzespołów elektronicznych. Dzięki swojej budowie oraz dobranym częściom składowym, pozwalają one inżynierom z roku na rok konstruować coraz to nowocześniejsze i bardziej funkcjonalne urządzenia. PCB służy przede wszystkim do montowania wszelkich podzespołów elektronicznych oraz zapewnienia im wspólnego stabilnego połączenia.

Tworzenie PCB składa się z trzech głównych etapów [1]:

- Logic Design - Stworzenie schematu logiki i reguł projektowych oraz spisu użytych komponentów i ich wzajemnych połączeń.
- Layout - Zaprojektowanie układu, który decyduje o fizycznym położeniu i połączeniach (tzw. trasowanie – *routing*) podzespołów.
- Produkcja przemysłowa.

Najważniejszym punktem projektowania układu jest rozmieszczenie komponentów. Ten proces jest skomplikowanym, twórczym przedsięwzięciem i prawdopodobnie jednym z najtrudniejszych aspektów procesu projektowania PCB. Wielu inżynierów uważa go za formę sztuki (rysunek 1.) gdyż w przeciwieństwie do schematu, który opiera się tylko na matematyce, jest nieco bardziej płynny i elastyczny oraz pozwala na kreatywne wdrożenie swojego projektu w życie.

Nie oznacza to jednak pełnej dowolności w projekcie, gdyż należy wziąć pod uwagę techniczną wiedzę, pomiary i zależności takie jak: optymalizacja długości ścieżek oraz ich szerokość, ochrona miejsc narażonych na dużą temperaturę, ograniczenia mechaniczne i montażowe itd. Nawet zastosowanie automatycznego wyznaczania ścieżek do optymalizacji nie zawsze da poprawny

rezultat.[1] Z uwagi na ilość i różnorodność ograniczeń nie jest możliwa całkowita automatyzacja sprawdzania poprawności wykonanego projektu, zatem przydatna dla projektanta okazuje się wizualizacja efektu końcowego. Jest ona także niezbędnym elementem procesu marketingowego, logistycznego czy edukacyjnego.

Rysunek 1. "Katsushika Hokusai Electronic Circuit Board" - Joel Betancourt znany jako Garabating

Źródło: https://garabating.com/post/44549621917/katsushika-hokusai-electronic-circuit-board

[1] Autodesk.com - Top 10 pcb component placement tips

ROZDZIAŁ 1

Cel i zakres pracy magisterskiej

Celem niniejszej pracy jest stworzenie łatwego do rozbudowania i narzędzia, umożliwiającego import plików używanych bezpośrednio w przemyśle projektowania PCB do programu Blender. Następnie dodatek będzie interpretował i wyświetlał pełnowymiarowy model 3D płytki drukowanej, która powstałaby w procesie produkcji przemysłowej. Aby stworzyć wizualizację wymagane będą projektowe pliki warstw i układu elementów. Ponadto, w ramach pracy stworzona zostanie podstawowa baza modeli 3D, konieczna w przypadku wyświetlania elementów na płytce. Dodatek będzie posiadał prosty i przejrzysty interfejs który zapewni dostęp do wszystkich funkcjonalności, ale nie przytłoczy odbiorcy nadmiarem funkcji. Pozwoli to nie tylko osobom z poza branży grafiki komputerowej na łatwy dostęp do wizualizacji i edycji swoich projektów ale także na łatwiejszą integrację projektów przemysłowych z marketingową i graficzną częścią przemysłu.

Rozdział ten na wstępie definiuje wymagania funkcjonalne jakie spełniać musi tworzone rozwiązanie. Analiza tych wymagań umożliwia zidentyfikowanie i opisanie pożądanego zachowania systemu. Następny fragment opisuje technologie, jakimi posłużono się w trakcie tworzenia pracy. Zostają także opisane typy i rozszerzenia plików na jakich omawiany system będzie operował.

1.1. Wymagania funkcjonalne

Zrealizowany system jest dodatkiem (ang. *add-on*) do programu Blender, kompatybilnym z wersją 2.8 wzwyż. Wybór konkretnie tej wersji programu był podyktowany jego nową odsłoną oferującą między innymi duże zmiany w interfejsie programistycznym aplikacji (zwanym dalej API — ang. *application programming interface*). Dodatek udostępnia użytkownikowi dodatkowe funkcjonalności z poziomu graficznego interfejsu programu w postaci panelu. Następujące wymagania zostały sformułowane z punktu widzenia użytkownika.

- Wybranie folderu zawierającego wszystkie pliki projektu PCB lub wybranie pojedynczych plików warstw.
- Wskazanie plików pozycji elementów (ang. *"Pick And Place"*)[1].
- Użycie wbudowanej lub własnej biblioteki modeli 3D podzespołów.
- Wybór końcowej rozdzielczości i miejsca zapisu plików wytworzonych w procesie renderowania.
- Przycisk tworzący model 3D płytki na podstawie wybranych plików.
- Użytkownik powinien być informowany o błędach powstałych w trakcie procesu generowania płytki.

[1] Więcej o strukturze plików projektowych PCB w punkcie 1.2.3

1.2. Opis technologii wykorzystanych w pracy

1.2.1. Python

Python jest językiem programowania wysokiego poziomu, posiadającym nieograniczone możliwości poprzez rozbudowę go o zewnętrzne pakiety.[2] API Blendera jest w większości przygotowane do użycia właśnie Pythona i chociaż istnieją ograniczenia tego, do jakich funkcjonalności programu mamy dostęp, jest to jedyne oficjalnie wspierane rozwiązanie dzięki któremu wiele można osiągnąć bez konieczności zagłębiania się w kod C / C++ Blendera.

1.2.2. Blender 2.8

Darmowy program z otwartym kodzie źródłowym (ang. *Open-source*[3]) cechujący się wszechstronnością i możliwością rozbudowania go o dodatkowe biblioteki lub skrypty napisane w języku Python, które rozszerzają podstawowe funkcjonalności.

Dodatek do programu Blender różni się od dodatkowej biblioteki Pythona jedynie pewnymi dodatkowymi wymaganiami. Musi posiadać obiekt zawierający metadane, takie jak: nazwa, wersja, kategoria, autor, etc. Określa też minimalną wersję Blendera wymaganą do uruchomienia skryptu. Dodatek jest więc sposobem na enkapsulację modułu Pythona w sposób, który użytkownik może z łatwością wykorzystać.

Blender ma wbudowany interpreter Pythona, który jest ładowany po uruchomieniu programu. Utrudnia to znacząco wykorzystanie automatycznego pobierania i instalowania zależnych od siebie pakietów, co za tym idzie, tworzony w ramach pracy system, aby ułatwić korzystanie z niego, musi być niezależny od zewnętrznych, dynamicznie pobieranych bibliotek.

[2] https://www.python.org/about/
[3] https://www.blender.org/about/license/

1.2.3. Środowisko Visual Studio Code

Visual Studio Code jest to darmowy edytor kodu który według badań przeprowadzanych co roku przez serwis StackOverflow[4,5] cieszy się coraz większym uznaniem. Wybór tego środowiska nie był jednak dyktowany jego popularnością lecz nowymi możliwościami otwierającymi się dzięki instalowaniu w nim rozszerzeń.[6] Dodatek stworzony przez Jacques Lucke na potrzeby rozwoju addonów do programu Blender pozwala między innymi na automatyzację procesu aktualizowania tworzonego addonu przez tworzenie skrótu w wewnętrznych folderach Blendera, co znacznie przyśpiesza pracę nad systemem. Ponadto posiada ułatwienia takie jak debugowanie w konsoli, tworzenie odpowiednich struktur i przydatne komendy. Sposób działania dodatku ma na celu także upewnienie się, że rozszerzenie nie koliduje z innym menedżerem pakietów Pythona.[7]

1.2.4. Pliki projektowe PCB

Gerber

Nowoczesne płytki drukowane są projektowane przy pomocy dedykowanego oprogramowania a ostatnim etapem produkcji dla projektanta jest wygenerowanie między innymi plików Gerber [2]. Jest to otwarty, wektorowy, powszechnie stosowany format o standardzie ASCII, służący do przesyłania danych projektowych obwodów drukowanych do przemysłu produkcji elektroniki. Wszystkie systemy do projektowania obwodów drukowanych pozwalają na eksport projektu jako pliki Gerber i każde przemysłowe oprogramowanie do ich obróbki potrafi je interpretować, umożliwiając profesjonalistom w dziedzinie PCB bezpieczną i wydajną wymianę danych projektowych [3].

[4] https://insights.stackoverflow.com/survey/2018/#development-environments-and-tools

[5] https://insights.stackoverflow.com/survey/2019#development-environments-and-tools

[6] https://code.visualstudio.com/docs/editor/extension-gallery

[7] https://marketplace.visualstudio.com/items?itemName=JacquesLucke.blender-development

Płytki drukowane mogą być jednostronne (jedna warstwa miedzi), dwustronne (dwie warstwy miedzi po obu stronach warstwy podłoża) lub wielowarstwowe (zewnętrzna i wewnętrzna warstwa miedzi, naprzemiennie z warstwami podłoża) [4].

Pliki Gerber reprezentują między innymi warstwy miedzi, maskę lutowniczą, legendę oraz dane wiercenia i trasy. Dodatkowe atrybuty dostarczają informacji o sposobie montowania, połączeń i nazw poszczególnych elementów. Format pliku Gerber jest prosty, kompaktowy i jednoznaczny. Jest bazowany na języku *G-code*, oznaczanym RS-274. Dzięki zastosowaniu 7-bitowych znaków ASCII jest czytelny dla człowieka i łatwy do debugowania. Obecnie używany od 2014 roku format to tzw. Rozszerzony Gerber (ang. *Extended Gerber*) lub RS-274X.[8]

Drill

Kolejnym, generowanym przez projektanta płytki formatem używanym w produkcji są pliki wierceń — NC lub CNC (ang. *Numerical control, computer numerical control*), pierwotnie zaprojektowane przez twórców wiercących i trasujących maszyn CNC jako zastrzeżone, dedykowane formaty wejściowe dla ich urządeń. Znane są pod nazwami takimi jak: Excellon, Hitachi, Sieb & Meyer, Posalux itd.[5]. Wszystkie z pośród tych formatów są podobne, ponieważ opierają się na wspomnianym wcześniej G-code. Rodzaje wierceń w PCB dzielą się na otwory zwykłe - NPTH (ang. *Not Plated Through Hole*) i pokryte miedzią - PTH (ang. *Plated Through Hole*)[6]. Stosowanie innego standardu nie jest jednak konieczne, jako że z czasem formaty te zmieniły swoje zastosowanie i obecnie powszechnie stosowaną praktyką jest generowanie tych plików w formacie Gerber.

Placement

Ostatnim i dosyć kluczowym elementem są pliki wskazujące elementy rozmieszczane na płytce. Jest to prosty format nazywany *Pick-And-Place, Placement list, X-Y file, Mount SMD* i składa się z kilku wartości:

[8] Dokumentacja formatu Gerber - https://www.ucamco.com/en/gerber

- *Ref / Designator* - Indeks elementu na płytce i w projekcie
- *Value* - Wartość elementu (np. pojemność, rezystancja, napięcie)
- Pozycja podana we współrzędnych kartezjańskich
- *Footprint / Package* - Nazwa elementu, zazwyczaj zawiera także informację o jego wymiarach
- Rotacja elementu
- Informacja czy objekt znajduje się na wierzchniej czy też dolnej stronie płytki
- Ewentualne komentarze i dodatkowe informacje

Plik zawiera opis elementów montowanych za pomocą technologii montażu przewlekanego - THT (ang. *Through-hole technology*) oraz powierzchniowego - SMT (ang. *Surface Mount Technology*), powszechnie stosowanego przemysłowo [7]. Nie jest to jednak format ściśle ustandaryzowany i przy masowej produkcji każdy producent musi manualnie zweryfikować opisy elementów. Na szczęście każdy program do tworzenia PCB posiada eksporter do generowania tychże plików. Dodatkowo są one proste w zapisie i z łatwością edytowalne przez człowieka.

1.2.5. Pliki VRML i X3D

Format *.wrl* zwany VRML (ang. *Virtual Reality Modeling Language*) powstały w 1994 roku, stał się pierwszym internetowym formatem 3D, został później zastąpiony przez format X3D [8]. Jego ówczesna powszechność sprawiła, że wiele programów przemysłowych tworzonych w latach dziewięćdziesiątych posiada bazy modeli w tym właśnie formacie. Więcej o praktycznym wykorzystaniu tego standardu w rozdziale 3.4.

ROZDZIAŁ 2

Architektura zrealizowanego systemu

Aby zobrazować podstawowe założenia projektu, w niniejszym rozdziale zostanie omówiona koncepcyjna struktura zrealizowanego systemu. Przedstawi ona bardziej szczegółowo mechanizmy działania najważniejszych komponentów aplikacji, a finalnie pomoże w samym procesie tworzenia oprogramowania.

2.1. Założenia i wymagania projektowe

W przypadku tak rozległych możliwości rozwoju systemu, kluczowe jest właściwe zdefiniowanie wymagań projektu i dążenie do ich realizacji. Postawienie ograniczeń w kwestii wspieranych formatów na jakich działać będzie dodatek jest konieczne z uwagi na zróżnicowanie technologii używanych w procesie tworzenia PCB. Z drugiej strony nacisk na modułowość rozwiązania pozwoli później na łatwiejsze dodanie dowolnego rozszerzenia lub funkcji. System powinien implementować wszystkie następujące funkcjonalności:

- Implementacja interfejsu przy pomocy API Blendera 2.8
- Czytanie i interpretacja plików Gerber (RS-274X)
- Tworzenie modelu płytki na podstawie otrzymanych plików, zgodnego z rzeczywistymi wymiarami
- Czytanie i interpretacja pliku placement w formacie *.csv*
- Udostępnienie użytkownikowi możliwości użycia dostarczonej lub własnej bazy modeli podzespołów

2.2. Struktura projektu

Podstawowy wybór strukturalny następującego rozwiązania jest podyktowany wymaganiami i sposobem komunikacji z API Blendera. Struktura folderów dodatku do programu Blender jest dowolna z założeniem, że w folderze głównym znajduje się plik *___init___.py* odpowiedzialny za rejestrowanie dodatku. Jest on automatycznie wykonywany po wybraniu folderu i włączeniu addonu w sekcji "Dodatki" w programie Blender. Pozostałe elementy są pogrupowane według konwencji modułów w języku Python, zatem każdy moduł posiada swój folder, jest to jednak podyktowane tylko enkapsulacją modułów i wygodą w używaniu referencji między skryptami.

2.2.1. Główna struktura interfejsu API Blendera

Z poziomu kodu, API Blendera udostępnia główne moduły pod słowem kluczowym **bpy**[1] a jego podstawowe i główne elementy to:

- **bpy.data** – Zapewnia dostęp do danych bieżącego pliku *.blend* (Jest to rozszerzenie używane przez program Blender). Każde z jego pól jest zbiorem obiektów danego typu (sceny, obiekty, zbiory wierzchołków, materiały, kolekcje itp.).
- **bpy.context** – Zawiera wszelkie dane środowiskowe obrazujące bieżący stan programu (bieżący wybór, tryb i region edytora) oraz wiele globalnych właściwości takich jak: preferencje użytkownika, bieżące ustawienia sceny.
- **bpy.ops** – Zawiera wszystkie *Operatory* Blendera. (W API każda komenda Blendera jest zaimplementowana jako metoda klasy *Operator*).
- **bpy.types** – Posiada definicje wszystkich klas używanych w strukturach.

[1] https://docs.blender.org/api/current/index.html

Istnieje także wiele mniejszych, pobocznych modułów i podmodułów pomocniczych. Niektóre z nich nie należą nawet do głównego modułu **bpy**. Mniejsze moduły, między innymi: bpy.props, bpy_extras, bpy.utils, mathutils, bmesh, będą omówione lub użyte w dalszej części tej pracy.

API Blendera wymaga od klas implementacji ściśle zdefiniowanych metod. Jest to rodzaj „umowy" pomiędzy skryptem a systemem rdzenia Blendera. Zgadzamy się na wdrożenie wymaganych funkcji w swojej klasie a Blender zgadza się wywoływać je w ściśle określonych okolicznościach. W programowaniu obiektowym taka lista zakontraktowanych funkcji i właściwości nazywa się „interfejsem". Aby pomóc w jego implementacji, API Blendera dostarcza odpowiednie klasy bazowe, które w żargonie programowania obiektowego są tak zwanymi „klasami abstrakcyjnymi". Zapewniają domyślne, puste implementacje wszystkich metod wymaganych przez interfejs. Nasze klasy dziedziczą tę domyślną treść z klasy bazowej.

2.2.2. Implementacja wzorca projektowego

MVP (ang. *Model–view–presenter*) jest wzorcem architektury oprogramowania opierającym się na trzech głównych założeniach[9]:

- Model reprezentuje dane które są przetwarzane i wysyłane do prezentera
- Widok (*View*) wyświetla dane uzyskane z prezentera i przekazuje dane wejściowe wprowadzane przez użytkownika do prezentera
- Prezenter jest wywoływany z Widoku aby wyświetlać dane pobrane z Modelu i przetwarzać dane wejściowe

System został zaimplementowany w oparciu o ten właśnie wzorzec z uwagi na to, że jest to struktura logicznie wynikająca ze sposobu używania API Blendera. W tym przypadku, w dużym uproszczeniu Modelem jest logika modułu, Widokiem – klasa odpowiedzialna za renderowanie i odbieranie danych wejściowych a Prezenter to API Blendera.

Skrypty w języku Python można zintegrować z Blenderem na następujące sposoby:

- Definiując silnik renderujący
- Poprzez zdefiniowanie operatorów
- Poprzez zdefiniowanie menu, nagłówków i paneli
- Wstawiając nowe przyciski do istniejących menu, nagłówków i paneli

Odbywa się to poprzez zdefiniowanie klasy, która jest podklasą istniejącego typu. Tak więc, przechodząc do rzeczywistego stanu rzeczy, omawiany addon o nazwie *PCB-Blender* jest zdefiniowany jako skompresowane archiwum składające się z następujących elementów:

- *PCB_LayoutPanel* – Klasa odpowiedzialna za wyświetlanie interfejsu i przekazywanie danych wybranych przez użytkownika, dziedzicząca z klas abstrakcyjnych *Panel* i *ImportHelper*. Wraz z klasami pomocniczymi znajduje się w pliku PCB_Blender_panel.py.
- *PCB_Generate* – Klasa dziedzicząca z klasy *Operator*, znajdująca się razem z klasami pośrednimi w pliku PCB_Blender.py.
- *___init___.py* – Plik zawierający metadane i rejestrujący powyższe klasy,
- Foldery zawierające pozostałe moduły Pythona do których odnosi się główny Operator – *PCB_Generate*.

2.2.3. Baza modeli

Jednym z pierwszych z wyzwań podczas tworzenia dodatku był dobór zasobów w postaci modeli 3D. Aby zrealizować założenia pracy, wymagana była baza modeli podzespołów elektronicznych, montowanych na PCB. Z uwagi na mnogość producentów, rodzajów i typów podzespołów oraz fakt, że każdy program służący do projektowania może oznaczać je inaczej, optymalnym rozwiązaniem wydaje się udostępnienie użytkownikowi podstawowej

bazy modeli. Ponadto zastosowanie szukania modeli częściowo dopasowanych nazwą. Z uwagi na fakt, że niemożliwym jest obsłużenie wszystkich wyjątków, koniecznością staje się umożliwienie użytkownikowi kożystania z własnych, wybranych modeli. Istnieje wiele stron udostępniających modele 3D na zasadach komercyjnych i darmowych licencji. Są one często wykorzystywane przez twórców, jak również programy projektowe. Oto kilka z nich zaprezentowanych w tabeli: 2.1.

Adres URL
https://grabcad.com/
https://www.3dcontentcentral.com/
https://www.digikey.com/en/resources/3d-models
https://www.te.com/
https://www.traceparts.com/en

Tabela 2.1. Publicznie dostępne bazy modeli podzespołów

Źródło: Opracowanie własne

Nie posiadają one jednak możliwości masowego pobierania modeli. Pomijając tą niedogodność, która musiałaby być rozwiązana skryptem automatyzującym pobieranie oraz pracę, również ilość pobranych materiałów znacznie przekroczyłaby racjonalny poziom. Z pomocą przychodzi tu darmowy zbiór modeli używanych w programie KiCad[2]. Modele są dostępne między innymi w formacie *.wrl* więc można zaimportować je do Blendera. Więcej o przetwarzaniu plików *.wrl* oraz tworzeniu bazy modeli w rozdziale 3.4.

[2] https://kicad.github.io/packages3d/

ROZDZIAŁ 3

Szczegóły implementacyjne systemu

W tym rozdziale zostanie omówiona implementacja kluczowych i pobocznych funkcjonalności systemu. Sposób implementacji narzucony przez wybrany wzorzec projektowy wpasowuje się w przyjęte i zalecane praktyki programowania obiektowego jak również tworzenia dodatków w Blenderze. Dzięki przyjętym wcześniej założeniom, funkcjonalności programu zostały podzielone na niezależne części których implementacja zostanie teraz przedstawiona. Niniejszy rozdział zastał podzielony na fragmenty przedstawiające: rejestrowanie dodatku, implementację interfejsu, renderowanie PCB, i tworzenie bazy modeli.

3.1. Rejestrowanie addonu

Zgodnie z dokumentacją, każdy dodatek do Blendera musi posiadać obiekt *bl_info*, zawierający metadane addonu i implementować metody *register* oraz *unregister*. Instalacja dodatku następuje poprzez wybranie folderu dodatku, wówczas dodatek zostaje dodany do listy wyświetlającej między innymi metadane dostarczone w obiekcie *bl_info*. Po aktywowaniu dodatku w programie zostaje wykonana metoda *register* i analogicznie *unregister* w przypadku usuwania addonu. Wprowadzone w wersji 2.8 API Blendera usprawnienia znacznie ułatwiają ten proces. Jeżeli nie są wymagane dodatkowe funkcjonalności podczas rejestrowania i usuwania dodatku lub jego komponentów, istnieje możliwość użycia funkcji *register_classes_factory* z pakietu **bpy.utils**, która automatycznie je zaimplementuje. Poniżej przedstawiony jest w całości plik odpowiedzialny za instalowanie, włączanie i wyłączanie dodatku.

Listing 3.1. *__init__.py* – Plik rejestrujący dodatek.

```
bl_info = {
        "name" : "PCB-Blender",
        "author" : "Adam Makiewicz",
        "description" : "Addon generates PCB model from Gerber files",
        "blender" : (2, 80, 0),
        "version" : (0, 1, 4),
        "category" : "Generic",
        "location" : "Scene Properties > PCB Renderer"
}
        import bpy
        from . PCB_Blender_panel import PCB_LayoutPanel
        from . PCB_Blender import PCB_Generate

        classes = (PCB_LayoutPanel, PCB_Generate)
        register, unregister = bpy.utils.register_classes_factory(classes)
```

Źródło: Opracowanie własne

3.2. Implementacja interfejsu

Klasa *PCB_LayoutPanel* dostarcza całą funkcjonalność renderowania interfejsu dzięki dziedziczeniu z wewnętrznej klasy bazowej *Panel*. Poprzez implementację tej klasy, w programie zostaje utworzony panel w wybranym regionie. Właściwości takie jak nazwa, miejsce wyświetlania się panelu, wielkość, kontekst, typ regionu itp. muszą być zdefiniowane w odpowiednim formacie. Dzięki temu są one automatycznie interpretowane przez API i poprawnie wyświetlane. Właściwości klasy zaczynają się od przedrostka *bl_*, jest to konwencja używana w celu odróżnienia wbudowanych właściwości wewnętrznych klas Blendera od tych, które zostają dodane przez programistę. Przykład implementacji wbudowanych właściwości klasy dziedziczącej znajduje się poniżej.

Listing 3.2. Właściwości panelu z pliku *PCB_Blender_panel.py*

```
class PCB_LayoutPanel(Panel):
        bl_label = "PCB Renderer"
        bl_idname = "SCENE_PT_layout"
        bl_space_type = 'PROPERTIES'
        bl_region_type = 'WINDOW'
        bl_context = "scene"
```

Źródło: Opracowanie własne

Oprócz posiadania statycznych właściwości, klasa powinna przechowywać i przekazywać dynamiczne zmienne. Aby dodać zmienną do zarejestrowanej klasy w Blenderze należy użyć jednej z Definicji Właściwości (ang. *Property Definition*) z modułu **bpy.props**. Moduł ten definiuje właściwości rozszerzające wewnętrzne dane Blendera. Wynik tych funkcji służy do przypisywania właściwości do klas zarejestrowanych w Blenderze i nie można ich używać bezpośrednio. Są to między innymi *StringProperty, FloatProperty, EnumProperty, BoolProperty* lub dowolne ich zestawienie jako jedna strukturalna zmienna. Funkcje wymagają każdorazowego zdefiniowania nazwy, opisu, wartości domyślnej, podtypu itp. więc aby uniknąć redundancji kodu zostały stworzone funkcje pomocnicze, przedstawione we fragmencie kodu 3.3. Zdefiniowanie typu *StringProperty* jako ścieżka do pliku lub folderu sprawia, że do elementu wyświetlającego tą zmienną automatycznie zostanie przypisany przycisk otwierający eksplorator plików w celu wybrania pliku.

Listing 3.3. Funkcje pomocnicze tworzące zmienną o tym samym typie lecz innym zastosowaniu – ścieżka pliku i ścieżka folderu

```
def FilePath(_name, _description="", _default=""):
        return StringProperty(name=_name, default = _default,
        description=_description, subtype = 'FILE_PATH')

def DirPath(_name, _description="", _default=""):
        return StringProperty(name=_name, default = _default,
        description=_description, subtype = 'DIR_PATH')
```

Źródło: Opracowanie własne

Oprócz zmiennych, aby poprawnie zaimplementować dany interfejs, klasa musi zawierać też funkcję *draw*, odpowiedzialną za renderowanie elementów w panelu. Wygląd i rozmieszczenie obiektów są zdefioniowane przy pomocy obiektu klasy *UILayout* i jego elementów, takich jak kolumny i rzędy w których umieszcza się referencje do wspomnianych wyżej, dynamicznych zmiennych. W elementach układu można umieszczać opisy, ikony a nawet definiować ich zachowanie przy pomocy instrukcji warunkowych. Poniżej znajduje się fragment kodu, używający zmiennej *boolowskiej* oraz instrukcji warunkowej w celu zdefiniowania zachowania listy rozwijanej.

Listing 3.4. Fragment kodu z klasy *PCB_LayoutPanel* definiujący zachowanie się rozwijalnej listy elementów

```
row = layout.row()
        row.prop(context.scene, "expand", icon="TRIA_DOWN"
        if context.scene.expand
        else "TRIA_RIGHT", icon_only=True, emboss=False)
        row.label(text="Or select individually specific files:")

        if bpy.context.scene.expand:
                col = layout.column()
                col.label(
                text="To use gerber folder, collapse this section!",
                icon='ERROR')
```

Źródło: Opracowanie własne

Następnie w jednym z elementów zostaje umieszczone wywołanie operatora z klasy *PCB_Generate*, który przyjmuje kontekst w którym zostal wywołany — a zatem również wszystkie zmienne.

3.3. Tworzenie PCB

Jak wspomniano wcześniej, funkcja wykonania operatora (*execute*) pobiera w argumencie informacje o środowisku i bieżący kontekst w jakim jest wywoływana. Następnie wykonuje w programie odpowiednie obliczenia i przekształcenia, które zostały podzielone na następujące kategorie:

3.3.1. Funkcje użytkowe

Tak samo jak w przypadku klasy odpowiedzialnej za renderowanie interfejsu, tutaj też zostały stworzone funkcje pomocnicze w celu uniknięcia powtarzalności kodu oraz zwiększenia czytelności przebiegu całego programu. Są one używane w obrębie całego projektu.

Funkcje informacyjne

Wszystkie z tych funkcji odnoszą się do obiektu klasy **bpy.context.window_manager**.

- Funkcje *RegisterProgress, UpdateProgress* oraz *EndProgress* służą do wyświetlania kursora postępu podczas wykonywania obliczeń i są używanie na przestrzeni wszystkich innych funkcji o znaczącej złożoności.
- Funkcja *ShowMessageBox* jest używana we wstępnej fazie walidacji otrzymanych zmiennych zanim zostaną wywołane pozostałe funkcje obliczające. Służy do wyświetlenia komunikatu informacyjnego lub błędu. Zazwyczaj zestawiona z anulowaniem lub przerwaniem wykonywania obliczeń, objaśniająca użytkownikowi powód zaprzestania działania.

Funkcje edytujące

Funkcje odnoszące się do bieżącego kontekstu i modyfikujące go wraz ze stanem aplikacji.

- *DeselectAll* — Odznacza wszelkie obiekty aktywne na scenie, przydatna funkcjonalność gwarantująca powtarzalność wykonywanych operacji bez znaczenia w jakim stanie początkowym użytkownik uruchomi polecenia generujące obiekty.

- *ChangeArea* — Zmienia tryb okna wyszukując obecny aktywny region w kontekście.

- *ChangeClipping* — Modyfikuje minimalną odległość od renderowanego obiektu w oknie widoku do jakiej może zbliżyć się kamera. Podstawową wartością w Blenderze jest 10 centymetrów co jest zazwyczaj zbyt dużą odległością na oglądanie płytki drukowanej rzeczywistych wymiarów.

- *PurgeOrphanData* — Funkcja czyszcząca pamięć podręczną programu z nieużywanych na scenie elementów (wytworzonych obiektów, siatek modeli, zaimportowanych zdjęć i dołączonych plików *.blend*). Nie jest ona obecnie wywoływana podczas przebiegu programu, jednak jest używana w procesie rozwoju i testowania aplikacji, ułatwiając resetowanie stanu całego programu w celu uzyskania powtarzalności testów.

3.3.2. Interpretacja plików

Placement

O ile pliki placement i ścieżka do bazy modeli 3D zostały zapewnione, program uruchomi metodę *ReadPlacement_csv*, przeczyta plik *.csv*, zarejestruje nazwy, numery porządkowe, pozycje i rotacje wymaganych komponentów. Nazwy plików muszą zostać przycięte do dlugości 63 znaków, ponieważ jest to limit długości nazwy obiektu w wewnętrznej strukturze Blendera. Następnie, przeszukując rekursywnie folder wskazany jako baza modeli, dołączy wszystkie znalezione pliki *.blend*, jednocześnie sprawdzając czy plik posiada szukany obiekt. Nie są to operacje rzutujące na ilość alokowanej pamięci ponieważ tworzą tylko połączenia z plikiem źródłowym, jednak wraz ze wzrostem ilości przeszukiwanych plików, znacząco wzrasta czas wykonywanych operacji.

W przypadku gdy któryś z modeli nie został znaleziony jest uruchamiana funkcja wyszukiwania modelu o podobnej nazwie. Jest to algorytm, który porównuje początek nazwy obiektu, jednak dzięki uwzględnieniu stosowanej konwencji w nazewnictwie modeli podzespołów, jest w stanie osiągnąć o

wiele mniejszą złożoność czasową niż podstawowy algorytm naiwny. Nazwy komponentów są to zazwyczaj ciągi nazw i oznaczeń oddzielone znakami "_".

Listing 3.5. Algorytm wyszukiwania komponentu po nazwie z uwzględnieniem całych słów oddzielanych separatorem "_"

```
for missing in required:
    separatedList = missing.split('_')
    for compfile in compfiles:
        UpdateProgress(i/len(compfiles))
        found = False
        with bpy.data.libraries.load(compfile, link=True)
        as (data_from, data_to):
            i = 0
            # Search models with names starting with most keywords possible
            while i < len(separatedList)-1:
                newSearch = separator.join
                (separatedList[:len(separatedList)-i])
                newFound = [value for value in
                data_from.meshes if value.startswith(newSearch)]
                if len(newFound) > 0:
                    elementFound = min(newFound, key=len)
                    requested  = separator.join(separatedList)
                    print("Found: ", elementFound,
                    " similar to requested: ", requested)
                    data_to.meshes.append(elementFound)
                    for col in layout_table:
                        if col[2] == requested:
                            col[2] = elementFound
                    found = True
                    break
                else:
                    i+=1
        if found:
            break
```

Po uzyskaniu listy modeli, program ustawia je w odpowiednich miejscach zgodnych z wcześniej przeczytanymi koordynatami, uwzględniając ich rotację oraz skalę. Skala komponentów różni się w zależności od tego jaki typ

jednostki jest używany w projekcie (milimetr lub mil — 1/1000 cala). W przypadku braku modelu, w odpowiednie miejsce zostaje ustawiony pusty obiekt, posiadający nazwę i numer porządkowy. Wczytane modele oraz informacje o dołączonych plikach *.blend* są zapisywane w tak zwanej pamięci podręcznej (ang. *cache*) programu. Sprawia to, że kolejna operacja wczytująca modele nie musi dodawać więcej plików z bazy modeli oraz ładować siatki już użytych modeli komponentów. Dzięki temu kolejne użycie programu (bez wyłączania go) jest znacznie szybsze.

Gerber

Do wczytania plików Gerber i Excellon został wykorzystany moduł Pythona *pcb-tools 0.1.6*, stworzony przez Paulo Henrique Silva i udostępniony na zasadach licencji Apache[1]. Narzędzia udostępniane przez moduł pozwalają na interpretację instrukcji G-code zawartych w plikach i zamianę ich na obiekty. Moduł, aby umożliwić mu funkcjonowanie w ramach projektu, został zmodyfikowany. Modyfikacje obejmują:

- Statyczne dołączenie bibliotek używanych przez moduł
- Zmiany nagłówków sktyptów, aby wskazywały na dołączone moduły
- Dodanie słów kluczowych do słownika wyszukującego i klasyfikującego typy warstw po nazwie pliku, w skrypcie *layers.py*
- Lustrzane odbicie dolnej warstwy renderowaniej płytki[2]
- Przechwytywanie danych wyjściowych w celu zapisu do pliku w formacie *.PNG*
- Dodana obsługa wykrywania obramowania płytki na podstawie wczytanych warstw, na wypadek gdyby warstwa odpowiedzialna za kształt nie została znaleziona (przedstawiona we fragmencie kodu 3.6)

[1] https://pypi.org/project/pcb-tools/

[2] Dolne warstwy płytek w programach projektujących są tworzone w odbiciu lustrzanym aby wprowadzić przejrzystość. Projektowanie odbywa się zazwyczaj w widoku rzutu z góry.

Listing 3.6. Obliczanie obramowania płytki na podstawie warstw

```
x_range = [10000, -10000]
y_range = [10000, -10000]
for layer in layers:
        bounds = layer.bounds
        if(self.first_bounds is None):
                self.first_bounds = bounds

        if self.first_bounds is not None:
                layer_x, layer_y = self.first_bounds
                x_range[0] = min(x_range[0], layer_x[0])
                x_range[1] = max(x_range[1], layer_x[1])
                y_range[0] = min(y_range[0], layer_y[0])
                y_range[1] = max(y_range[1], layer_y[1])
width = x_range[1] - x_range[0]
height = y_range[1] - y_range[0]

scale = math.floor(min(float(max_width)/width, float(max_height)/height))
self.scale = (scale, scale)
```

W procesie interpretacji pliku zostaje utworzona nowa instancja klasy *PCB*, która będzie zawierała między innymi obiekty reprezentujące poszczególne warstwy i funkcje wskazujące na odpowiednie grupy warstw. Typy warstw są rozpoznawane przez program przy pomocy wyszukiwania wzorca w jego nazwie. W przyszłości możliwe jest dodanie wyszukiwania wzorca także w treści pliku. Byłaby to niezawodna i uniwersalna, lecz skomplikowana i czasochłonna operacja. Wszystkie fragmenty tekstu są porównywane za pomocą wyrażeń regularnych zdefiniowanych w słowniku, znajdującym się w pliku *layers.py*. Fragment kodu 3.7 pokazuje główną funkcję, odpowiedzialną za definiowanie typu warstwy.

Każdej warstwie zostaje przydzielony typ (określający sposób w jaki będzie musiała zostać później wyrenderowana) oraz grupa obiektów 2D opisujących kształty — tzw. prymitywy (ang. *primitives*), składające się z wielokątów i krzywych o różnych grubościach. Ostatecznie warstwy są sortowane w odpowiedniej kolejności i dzielone na grupy: górne, dolne, wiercenia, i kontur.

Listing 3.7. Funkcja klasyfikująca warstwę do odpowiedniej kategorii.

```
def guess_layer_class(filename):
    try:
        layer = guess_layer_class_by_content(filename)
        if layer:
            return layer
    except:
        pass
    try:
        directory, filename = os.path.split(filename)
        name, ext = os.path.splitext(filename.lower())
        for hint in hints:
            if hint.regex:
                if re.findall(hint.regex, filename, re.IGNORECASE):
                    return hint.layer
            patterns = [r'^(\w*[.-])*{}([.-]\w*)?$'
                        .format(x) for x in hint.name]
            if ext[1:] in hint.ext
            or any(re.findall(p, name, re.IGNORECASE) for p in patterns):
                return hint.layer
    except:
        pass
    return 'unknown'
```

3.3.3. Renderowanie

Biblioteka *Cairocffi*, używana przez moduł *pcb-tools*, jest oparta o CFFI[3,4]. Jest to zamiennik modułu *Pycairo* — zestawu powiązań Pythona i obiektowego API dla *Cairo*. *Cairo* jest to biblioteka grafiki wektorowej 2D z obsługą wielu formatów wyjściowych, w tym buforów obrazów, plików *PNG, PostScript, PDF* i *SVG*. Klasa *GerberCairoContext* w pliku cairo_backend.py odpowiada za powiązania funkcji renderowania obiektów bezpośrednio do biblioteki *Cairocffi*. Jak wspomniano wcześniej, moduł ten został statycznie dołączony do projektu.

[3] Interfejs do języka Python, umożliwiający wywoływanie metod w języku C
[4] https://cffi.readthedocs.io/en/latest/

Jeżeli użytkownik wskazał pliki warstw i wierceń lub folder je zawierający, program po wykonaniu analizy stworzy instancję obiektu PCB. Następnie jest uruchamiana metoda *CreateImage* która przekazuje obiekt PCB do klasy *GerberCairoContext*. Wszystkie składowe warstw i obiekty płytki zostały wcześniej pogrupowane w celu stworzenia dwóch obrazów płytki — górnej i dolnej warstwy. Obrazy są tworzone poprzez nakładanie na siebie w odpowiedniej kolejności wyników z renderów poszczególnych składowych warstw: miedzi, laminatu, oznaczeń i wierceń. Właściwości i klasyfikacja warstwy odpowiadają za kolor tła i elementów (prymitywów). Fragment kodu 3.8 przedstawia główną funkcję renderującą warstwę. Efektem wyjściowym jest stworzenie dwóch obrazów w formacie *.PNG*. Pliki posiadają wcześniej zdefiniowaną rozdzielczość, a zapisane są w folderze wskazanym przez użytkownika. Nazwy plików posiadają prefiks "Top_layer" lub "Bottom_layer" oraz datę i czas stworzenia jako sufiks.

Listing 3.8. Główna funkcja renderująca warstwę.

```
def render_layer(self, layer, filename=None, settings=None, bgsettings=None,
                 verbose=False, bounds=None):
    if settings is None:
        settings = THEMES['default'].get(layer.layer_class, RenderSettings())
    if bgsettings is None:
        bgsettings = THEMES['default'].get('background', RenderSettings())
    if self._render_count == 0:
        if verbose:
            print('[Render]: Rendering Background.')
        self.clear()
        if bounds is not None:
            self.set_bounds(bounds)
        else:
            self.set_bounds(layer.bounds)
        self.paint_background(bgsettings)
    if verbose:
        print('[Render]: Rendering {} Layer.'.format(layer.layer_class))
    self._render_count += 1
    self._render_layer(layer, settings)
    if filename is not None: self.dump(filename, verbose)
```

Tworzenie modelu 3D

Kolejnym krokiem jest stworzenie modelu płytki poprzez uruchomienie metody *CreateModel*. W zależności od tego czy podczas interpretacji pliku gerber, została wykryta warstwa konturowa lub nie, zostaje stworzona siatka będąca obramowaniem płytki. W przypadku braku warstwy konturowej, wymiary siatki zostają wybrane jako punkty graniczne (ang. *bounds*) obiektu PCB, to znaczy prostokątu obliczonego na podstawie maksymalnych i minimalnych koordynatów użytych we wszystkich warstwach. Model jest odpowiednio skalowany aby zachować rzeczywiste wymiary. Dwuwymiarowy kształt płytki zostaje następnie pogrubiony metodą *Extrude* o 1.6 milimetra — jest to standardowa przyjęta grubość płytki po złożeniu wszystkich warstw [2]. Funkcjonalność opisana powyżej jest zaprezentowana w wycinku 3.9

Listing 3.9. Fragment funkcji tworzącej model płytki.

```
mesh = None
    scaler = mathutils.Vector((1, 1, 1))
    if units == "metric":
        scaler = mathutils.Vector((.001, .001, .001))
    if units == "inch":
        scaler = mathutils.Vector((.0254, .0254, .0254))
    if(pcb_instance is not None):
        if(pcb_instance.outline_layer is not None):
            outline = pcb_instance.outline_layer
            mesh = RenderOutline(name, outline, mat, None, scaler)
    else:
        bounds = ctx.first_bounds
        mesh = RenderBounds(name, bounds, scaler, mat)
    if mesh is None:
        bounds = pcb_instance.layers[0].bounds
        mesh = RenderBounds(name, bounds, scaler, mat)
    if extrude and mesh:
        Extrude(mesh, 0.0016, extrudeMat)
    return mesh
```

Ostatecznie zostają stworzone i nadane obu stronom płytki dwa materiały używające wcześniej stworzonych plików *.PNG* jako tekstur. Dodatkowo jest stworzony lub załadowany (jeżeli już istnieje w pamięci podręcznej) materiał boczny imitujący kolor krzemowej płytki i przypisane są mu odpowiednie wartości. Fragment 3.10 przedstawia opisany proces.

Listing 3.10. Fragment funkcji tworzącej model, nadający modelom płytki odpowiednie materiały.

```
mat = bpy.data.materials.new(name = name)
mat.use_nodes = True
bsdf = mat.node_tree.nodes["Principled␣BSDF"]
texImage = mat.node_tree.nodes.new('ShaderNodeTexImage')
texImage.image = bpy.data.images.load(
os.path.join(source_folder, name + '.png'))
mat.node_tree.links.new(bsdf.inputs['Base␣Color'],
texImage.outputs['Color'])

extrudeMat = None
if extrude:
    extrudeMat = bpy.data.materials.get("ExtrudeMat")
    if extrudeMat is None:
        extrudeMat = bpy.data.materials.new(name = "ExtrudeMat")

    extrudeMat.use_nodes = True
    bsdf = extrudeMat.node_tree.nodes["Principled␣BSDF"]

    # Base substrate color
    bsdf.inputs[0].default_value = (0.350555, 0.266215, 0.0896758, 1)

    # Subsurface factor
    bsdf.inputs[1].default_value = 0.05

    extrudeMat.node_tree.links.new(bsdf.inputs['Base␣Color'],
    texImage.outputs['Color'])
```

3.4. Baza modeli

Aby stworzyć modele komponentów i zapisać je w postaci pliku *.blend* wymagane było użycie samego programu Blender. Niestety importer plików *.wrl* nie działa prawidłowo (nie wczytuje materiałów, ucina modele, tworzy bardzo dużą ilość niezoptymalizowanej geometrii). Ponadto nie posiada możliwości importu wielu plików na raz.

Z uwagi na powyższe, w ramach pracy powstała zmodyfikowana wersja wbudowanego w Blender importera plików *.wrl* w celu stworzenia bazy modeli. Było to możliwe dzięki udostępnionemu publicznie kodowi Blendera. Niektóre z usprawnień zostały zaproponowane jako oficjalna zmiana w przyszłych wersjach programu Blender. Modyfikacje obejmują:

- Dodanie obsługi materiałów wraz z przypisywaniem domyślnego koloru w przypadku braku informacji, przedstawione poniżej.

Listing 3.11. Wczytywanie danych materiału z pliku *.wrl*

```
bpymat = bpy.data.materials.new(vrmlname)
diff_color = mat.getFieldAsFloatTuple(
        'diffuseColor', [0.5, 0.5, 0.5, 1], ancestry)

if(len(diff_color) == 3):
    # .wrl does not have alpha, we need to add 4th element
    diff_color = diff_color + [1]
bpymat.diffuse_color = diff_color

bpymat.specular_color = mat.getFieldAsFloatTuple(
        'specularColor', [0.0, 0.0, 0.0], ancestry)
```

- Optymalizacja procesu tworzenia geometrii (usuwanie powtarzających się wierzchołków).

- Obsługa importu wielu plików z wybranego folderu.

- Z uwagi na wprowadzoną obsługę wczytywania wielu plików oraz stosowania optymalizacji geometrii, wymagana była implementacja czyszczenia pamięci (ang. *garbage collection*). Zmniejszenie alokowanej pamięci podręcznej programu znacznie przyspieszyło działanie algorytmu.
- Wczytywanie nazwy obiektu.

Listing 3.12. Wczytywanie nazwy obiektu z pliku *.wrl*

```
# .wrl Appearance node structure:
# ('material', 'DEF', 'Name_of_the_material', 'Material')
# Material name is also used for caching

if mat.reference is not None:
    if mat.reference.id is not None and len(mat.reference.id) >= 3:
        vrmlname = mat.reference.id[2]
```

- Zapisywanie materiałów tworzonych przy imporcie w pamięci podręcznej w celu ponownego użycia.
- Obsługa wyjątków i błędów powstałych przy wczytywaniu nietypowych lub wielokrotnie zagnieżdżonych struktur w plikach *.wrl*.

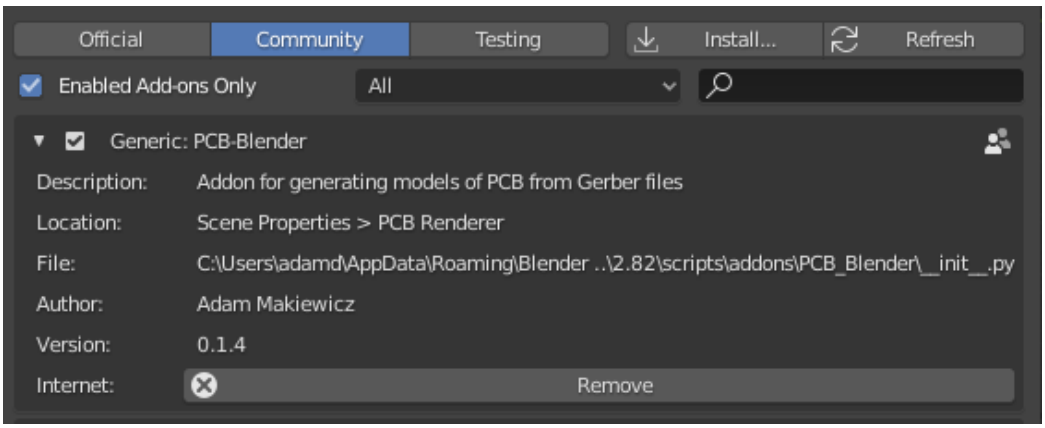
ROZDZIAŁ 4

Efekty pracy

4.1. Praktyczne zastosowanie

Niniejszy podrozdział przedstawia wygląd i przykładowy sposób użycia dodatku w praktyce.

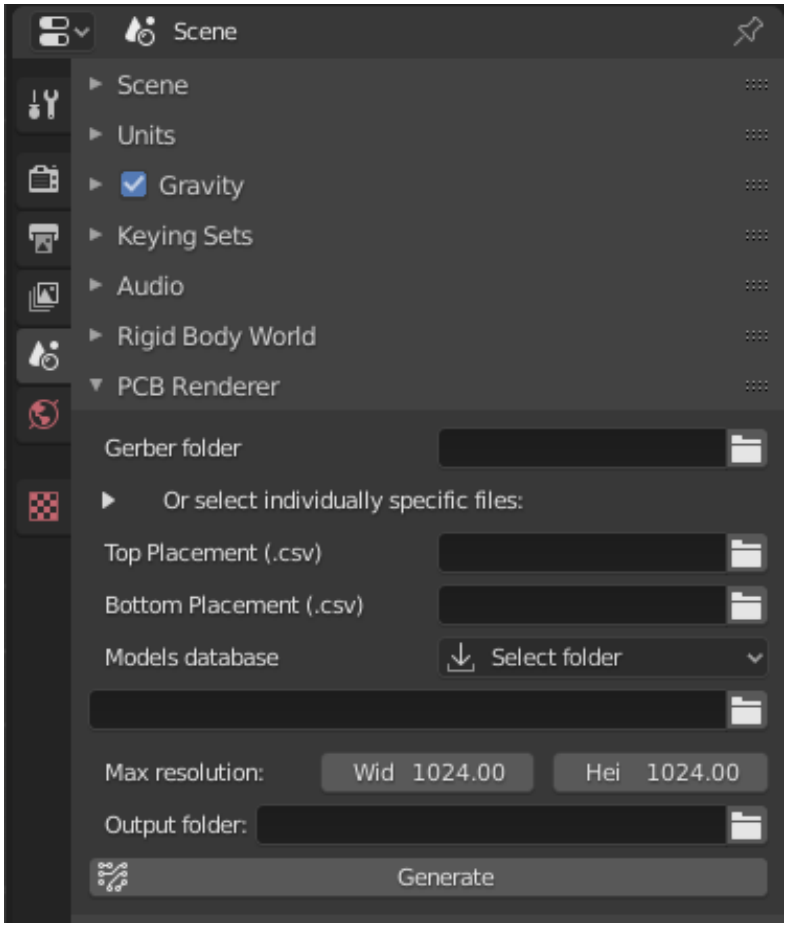
Poniżej przedstawiono opis dodatku, wyświetlany w panelu preferencji. Po wybraniu archiwum dodatku należy go włączyć (zainstalować) zaznaczając pole wyboru po lewej stronie nazwy dodatku.



Rysunek 4.1. Opis dodatku wyświetlany po instalacji w oknie preferencji

Źródło: Opracowanie własne

Jak przedstawia zdjęcie 4.2, po instalacji, panel opcji dodatku znajduje się w zakładce właściwości sceny.



Rysunek 4.2. Zainstalowany dodatek w programie Blender 2.82a

Źródło: Opracowanie własne

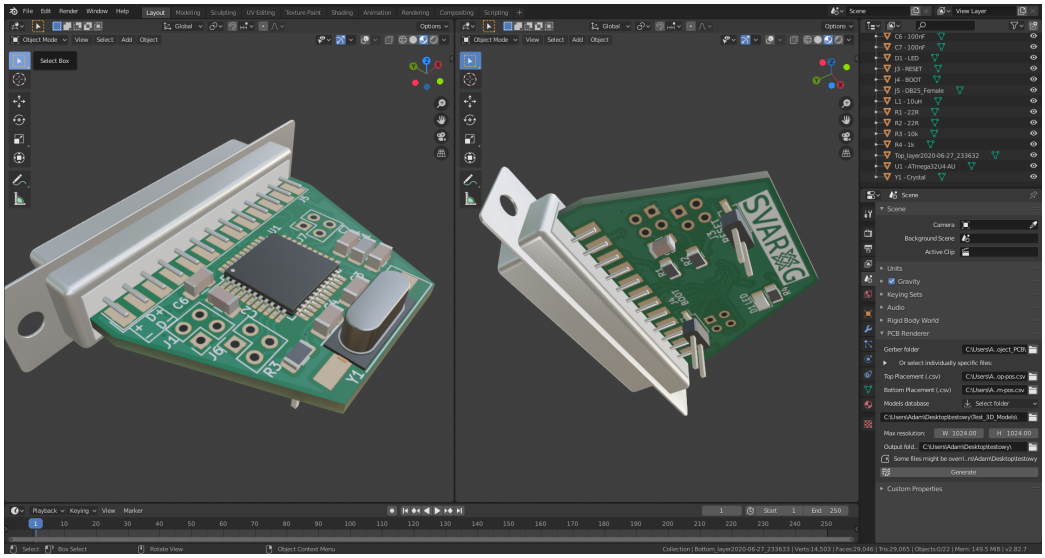
Listing 4.1. Przykładowy początek pliku warstwy (Gerber)

```
G04 #@! TF.FileFunction,Copper,L2,Bot*
G04 #@! TF.FilePolarity,Positive*
%FSLAX46Y46*%
G04 Gerber Fmt 4.6, Leading zero omitted, Abs format (unit mm)*
G04 Created by KiCad (PCBNEW (5.1.4)-1) date 2020-02-27 17:49:24*
%MOMM*%
%LPD*%
G04 APERTURE LIST*
%ADD10C,0.300000*%
%ADD11C,0.100000*%
%ADD12C,1.250000*%
%ADD13R,1.700000X1.700000*%
%ADD14O,1.700000X1.700000*%
G04 APERTURE END LIST*
D10*
X91356714Y-83101571D02*
X91856714Y-82387285D01*
X92213857Y-83101571D02*
```

Listing 4.2. Przykładowy plik placement (*.csv*)

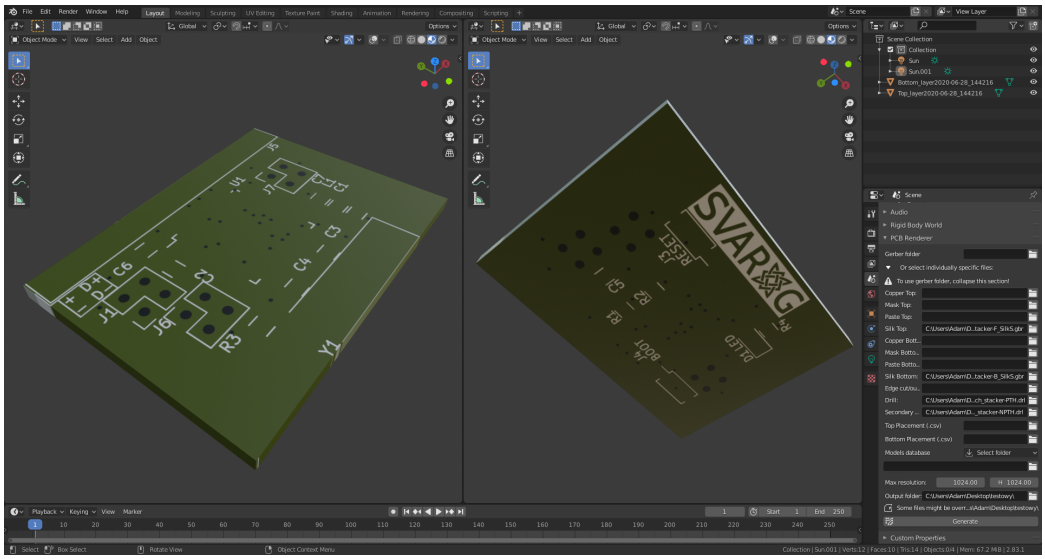
```
Ref,Val,Package,PosX,PosY,Rot,Side
"C5","1uF","C_1206_3216Metric",66.675000,-86.611000,90.000000,bottom
"D1","LED","LED_1206_3216Metric",84.579000,-94.488000,180.000000,bottom
"R1","22R","R_1206_3216Metric",70.863000,-86.614000,180.000000,bottom
"R2","22R","R_1206_3216Metric",70.609000,-90.297000,180.000000,bottom
"R4","1k","R_1206_3216Metric",84.204000,-96.774000,0.000000,bottom
```

Po wypełnieniu pól w palenu dodatku, tzn. wybraniu plików warstw (Gerber) oraz placement, wskazaniu folderu z modelami 3D i folderu wyjściowego, program jest gotowy do wygenerowania płytki. Modele 3D muszą znajdować się w plikach *.blend* oraz posiadać nazwy zgodne z wartością *"Package"* w pliku placement. Jeżeli dane zostały wprowadzone prawidłowo, w przypadku naciśnięcia przycisku *"Generate"* zostanie wygenerowany model, w przeciwnym wypadku użytkownik zostanie powiadomiony o ewentualnym błędzie w postaci wyskakującego okna.



Rysunek 4.3. Płytka stworzona za pomocą dodatku

Źródło: Opracowanie własne



Rysunek 4.4. Wizualizacja wybranych warstw płytki — nadruku oraz wierceń

Źródło: Opracowanie własne

4.2. Możliwości dalszego rozwoju systemu

Ten podrozdział opisuje potencjalne dodatkowe funkcjonalności które mogą być wprowadzone do projektu w celu polepszenia doświadczeń użytkownika i zwiększenia możliwości programu.

PCB-Blender

- Stworzenie kolejnych baz modeli na podstawie komponentów z innych programów projektowych.
- Stworzenie słownika nazw modeli aby wiele nazw wskazywało na jeden model.
- Renderowanie dowolnej ilości warstw (możliwość wybrania dowolnie wielu plików i kolejności ich renderowania).
- Dopisywanie wyrażeń regularnych (wykrywanie i poprawna klasyfikacja większej ilości rozszerzeń i nazw plików).
- Definiowane innych palet i schematów kolorystycznych dla renderowanych obrazów płytek.
- Wsparcie dla innych formatów plików projektowych (moduł pcb-tools zapowiada w swoim opisie planowane wsparcie dla formatów IPC-2581, ODB++ i innych[1]).

Importer formatu *.WRL*

- Optymalizacja — zmniejszenie zapotrzebowania na pamięć przy przetwarzaniu dużych plików.
- Zmniejszenie rozmiaru plików przy pomocy instancjonowania geometrii (wiele elementów posiada powtarzające się elementy).

[1] https://pcb-tools.readthedocs.io/en/latest/about.html

Zakończenie

Praca ta omawia proces tworzenia dodatku do programu Blender. Na podstawie plików projektowych, dodatek tworzy realistyczną i wierną rzeczywistym wymiarom wizualizację modelu 3D płytki drukowanej, która powstałaby w procesie produkcji przemysłowej. W pierwszym rozdziale zostały opisane wszystkie technologie użyte podczas tworzenia oprogramowania. Dalej, została omówiona architektura realizowanego systemu, implementacja interfejsu i logiki, a także proces tworzenia zasobów w postaci bazy modeli 3D. Aby dodatek był łatwy w użytkowaniu, całość stworzonego i opisanego w tej pracy oprogramowania, wraz z bazami modeli koniecznymi do prawidłowego wczytywania plików *placement*, jest dostępna pod adresem dołączonym do pracy w załączniku A.

W efekcie końcowym zostały utworzone dwa dodatki. Jeden oferujący zakładaną funkcjonalność oraz drugi, powstały jako narzędzie do tworzenia bazy modeli. W wyniku prac implementacyjnych udało się zrealizować projekt, który jest zgodny z przyjętymi wcześniej wymaganiami i założeniami przedstawionymi na etapie opracowywania koncepcji rozwiązania. Powstałe oprogramowanie nie wyczerpuje jednak wszystkich możliwości rozwiązania problemu, jako, że instnieje i ciągle powstaje wiele programów i formatów używanych w produkcji PCB. Dzięki zastosowaniu modułowego podejścia do systemu, jest on uniwersalny i łatwy do rozbudowania o dodatkowe funkcjonalności.

Bibliografia

[1] Nadine Abboud, Martin Grötschel, and Thorsten Koch. Mathematical methods for physical layout of printed circuit boards: An overview. *OR Spectrum*, 30:453–468, 06 2008.

[2] R.S. Khandpur. *Printed Circuit Boards: Design, Fabrication, Assembly and Testing*. Tata McGraw-Hill, 2005.

[3] A. Williams. *Build Your Own Printed Circuit Board*. McGraw-Hill Education, 2004.

[4] C. Schroeder. *Printed Circuit Board Design Using AutoCAD*. EDN series for design engineers. Newnes, 1998.

[5] Jean-Pierre Charras. Xnc format: Gerber takes data into the future. *Design007*, 4:24–28, 04 2019.

[6] S.H. Voldman. *ESD: Analog Circuits and Design*. ESD series. Wiley, 2014.

[7] R. Prasad. *Surface Mount Technology: Principles and Practice*. Springer US, 2013.

[8] David Raggett. *Extending WWW to support Platform Independent Virtual Reality*. Hewlett Packard Laboratories, 1994.

[9] S. Millett. *Professional ASP.NET Design Patterns*. EBL-Schweitzer. Wiley, 2010.

Spis rysunków

DODATEK A

Źródła

- Repozytorium projektu: https://github.com/adammak23/PCB-Blender
- Autodesk.com - Top 10 pcb component placement tips: https://www.autodesk.com/products/eagle/blog/top-10-pcb-component-placement-tips-pcb-beginner/
- Oficjalna strona Python Software Foundation: https://www.python.org/about/
- Oficjalna strona Blender Foundation: https://www.blender.org/
- Dokumentacja formatu Gerber: https://www.ucamco.com/en/gerber
- Specyfikacja formatu VRML: http://www.graphics.stanford.edu/courses/cs248-98-fall/Assignments/Assignment3/VRML2_Specification/
- Dodatki do edytora Visual Studio https://code.visualstudio.com/docs/editor/extension-gallery
- Moduł pcb-tools: https://pypi.org/project/pcb-tools/
- Dokumentacja CFFI: https://cffi.readthedocs.io/en/latest/

Pliki użyte w celu stworzenia bazy modeli pochodzą z oficjalnej strony pakietów KiCad[1], która udostępnionia je na zasadach licencji **CC-BY 3.0**[2].

[1] https://kicad.github.io/packages3d/
[2] http://creativecommons.org/licenses/by/3.0/