UNIWERSYTET GDAŃSKI
Wydział Matematyki, Fizyki i Informatyki

Adam Makiewicz
nr albumu: 235281

Generowanie płytek obwodu drukowanego w środowisku Python jako dodatek do programu graficznego Blender

Praca magisterska na kierunku:
INFORMATYKA
Promotor:
dr Piotr Arłukowicz

Gdańsk 2020

Streszczenie

Istnieje wiele programów do projektowania płytek drukowanych, jednak żadne z nich, z uwagi na swoje ścisłe zastosowania, nie posiadają odpowiednich narzędzi do zaawansowanego renderowania, animacji i tworzenia szeroko pojętej "sztuki". Popularny program do tworzenia grafiki 3D - Blender, z uwagi na możliwość rozbudowania go o dodatki jest znakomitym narzędziem mogącym wspomagać ten proces.

Słowa kluczowe

wizualizacja, grafika, 3D, Blender, Python, PCB, elektronika

Spis treści

Wprowadzenie

Obwody drukowane czy też inaczej płytki drukowane (zwane dalej "PCB", ang. Printed Circuit Board) to podstawa dla każdego modułu elektronicznego. Dzięki swojej budowie oraz dobranym częściom składowym pozwalają 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, spis użytych komponentów i ich wzajemnych połączeń
- Layout - Zaprojektowanie układu, który decyduje o fizycznym położeniu i połączeniach (tzw. trasowanie – *routing*) komponentów
- Produkcja przemysłowa

Najważniejszym punktem projektowania układu jest rozmieszczenie komponentów. Ten proces jest satysfakcjonującym, twórczym przedsięwzięciem i prawdopodobnie jednym z najtrudniejszych aspektów procesu projektowania PCB. Wielu inżynierów uważa go za formę sztuki 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 mnogość technicznej wiedzy, pomiarów i zależności takich jak: optymalizacja długości ścieżek oraz ich szerokość, ochrona miejsc narażonych na dużą temperaturę, ograniczenia mechaniczne i montażowe, itd. Nawet zastosowanie automatucznego wyznaczania ścieżek do optymalizacji nie zawsze da poprawny rezultat.[1] Z uwagi na ilość i różnorodność ograniczeń nie

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

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

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

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. Istnieje wiele programów do projektowania PCB jednak żadne z nich, z uwagi na swoje ścisłe zastosowania, nie posiadają odpowiednich narzędzi do zaawansowanego renderowania, animacji i tworzenia szeroko pojętej “sztuki”. Popularny program do tworzenia grafiki 3D - Blender, z uwagi na możliwość rozbudowania go o dodatki jest znakomitym narzędziem mogącym wspomagać ten proces.

ROZDZIAŁ 1

Cel i zakres pracy magisterskiej

Celem niniejszej pracy jest stworzenie łatwego do rozbudowania i spójnego systemu umożliwiającego import plików projektowych używanych bezpośrednio w przemyśle PCB do programu Blender, następnie interpretację i wyświetlenie pełnowymiarowego modelu 3D płytki drukowanej która powstałaby w procesie produkcji przemysłowej. Dodatek powinien posiadać prosty i przejrzysty interfejs który zapewnia dostęp do wszystkich funkcjonalności, ale nie przytłacza odbiorcy nadmiarem funkcji. Pozwoli to nie tylko osobom technicznym 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.

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 nowy wygląd programu i duże zmiany w interfejsie programistycznym aplikacji (ang. *API*). Wtyczka udostępnia użytkownikowi dodatkowe funkcjonalności z poziomu graficznego interfejsu programu. Następujące wymagania zostały sformułowane z punktu widzenia użytkownika.

- Wybranie folderu zawierającego wszystkie pliki projektu PCB lub wybranie pojedynczych plików warstw i plików pozycji elementów (ang. *"Pick And Place"*)[1]

- Wybranie wbudowanej lub własnej biblioteki modeli 3D

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

- Wybór końcowej rozdzielczości i miejsca zapisu plików wytworzonych w procesie renderowania

- Przycisk tworzący model 3D płytki na podstawie wybranych plików

1.2. Opis technologii wykorzystanych w pracy

1.2.1. Python

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

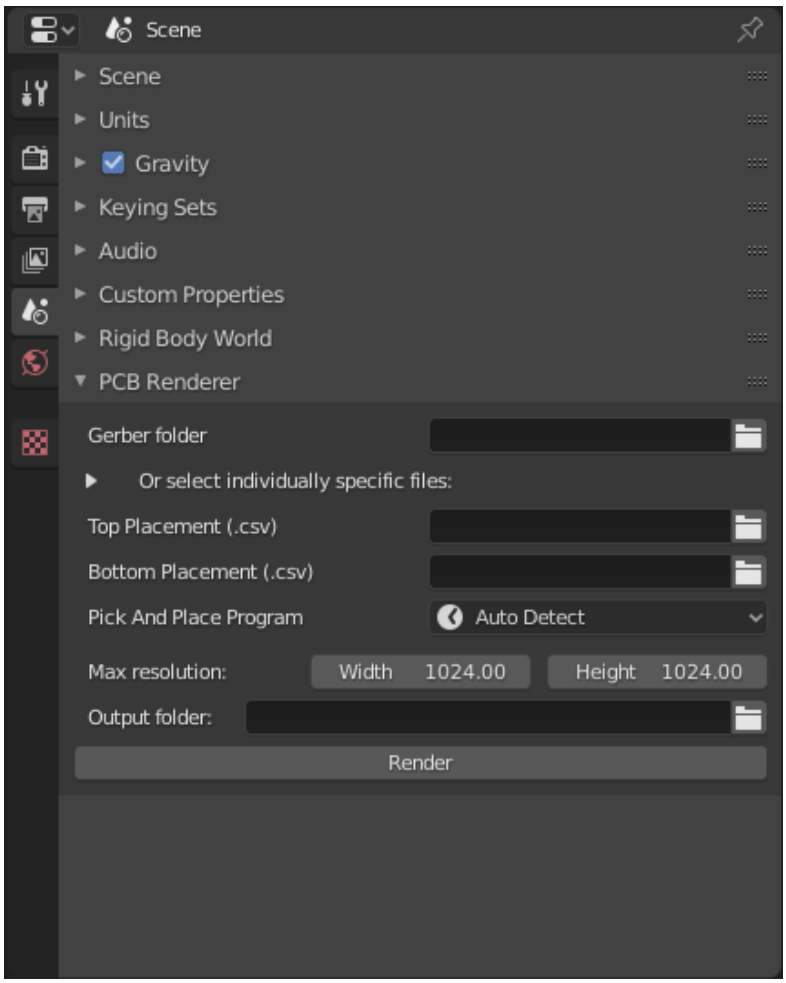
1.2.2. Blender 2.8

Darmowy program *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 jak 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

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



Rysunek 1.1. Zainstalowany dodatek w programie Blender 2.82a

Źródło: Opracowanie własne

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.

1.2.3. 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]

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.[4]

[4] https://www.ucamco.com/en/gerber

Drill

Kolejnym, generowanym przez projektanta płytki formatem używanym w produkcji są pliki wierceń (ang. *NC / CNC drill*), 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:

- *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.

1.2.4. 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 2.2.3.

1.2.5. Środowisko Visual Studio Code

Visual Studio Code jest to darmowy edytor kodu który według badań przeprowadzanych co roku przez serwis StackOverflow[5,6] cieszy się coraz większym uznaniem. Wybór tego środowiska nie był jednak dyktowany jego popularnością lecz nowymi możliwościami otwierającymi się dzięki instalowaniu w nim rozszerzeń.[7] 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

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

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

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

odpowiednich struktur i przydatne komendy. Sposób działania dodatku ma na celu także upewnienie się, że rozszerzenie nie koliduje z innym menedżerem pakietów Pythona.[8]

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

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 działał 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. 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.
- **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.

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,

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

będą omówione 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 obsługa wszystkich wyjątków, koniecznością staje się umożliwienie użytkownikowi kożystanie z własnych,

wybranych modeli. Istnieje wiele stron udostępniających modele 3D na zasadach komercyjnych jak i darmowych licencji. Są one często wykorzystywane przez twórców jak i 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 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 omówiony sposób implementacji kluczowych i pobocznych funkcjonalności systemu.

3.1. Rejestrowanie addonu

Każdy dodatek do Blendera musi posiadać obiekt *bl_info*, zawierający metadane addonu oraz implementować metody *register* oraz *unregister*. Wprowadzone w wersji 2.8 API usprawnienia znacznie ułatwiają ten proces. Jeżeli nie potrzebujemy dodatkowych funkcjonalności podczas rejestrowania i usuwania dodatku lub jego komponentów, możemy użyć funkcji *register_classes_factory* z pakietu **bpy.utils**, która automatycznie je zaimplementuje.

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

```
bl_info = {
        "name" : "PCB-Blender",
        "author" : "Adam Makiewicz",
        "description" : "Addon for generating models of PCB 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)
```

3.2. Implementacja interfejsu

Klasa *PCB_LayoutPanel* dostarcza całą funkcjonalność renderowania interfejsu dzięki dziedziczeniu z wewnętrznej klasy bazowej *Panel*. 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 do odróżnienia wbudowanych właściwości klas Blendera od tych, które zostają dodane przez programistę.

Listing 3.2. Przykład 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"
```

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 – ścieżki 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')
```

Oprócz zmiennych, aby poprawnie zaimplementować dany interfejs, klasa musi zawierać też funkcję *draw* odpowiedzialną za renderowanie elementów w panelu. Wygląd i rozmieszczenie elementów jest zdefioniowane przy pomocy obiektu klasy *UILayout* i jego elementów takich jak kolumny i rzędy w których umieszcza się referencje do wspomnianych wyżej, dynamicznych zmiennych. 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 metod.

Funkcje informacyjne

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

- Funkcje *RegisterProgress, UpdateProgress* oraz *EndProgress* służą do wyświetlania postępu w wykonywaniu 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* — Jak wskazuje nazwa, 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.

3.3.2. Interpretacja plików

Placement

O ile pliki placement i ścieżka do bazy modeli 3D zostały zapewnione, program przeczyta plik *.csv* i 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 pliku Blendera. Następnie, przeszukując rekursywnie folder wskazany jako baza modeli, dołączy wszystkie znalezione pliki *.blend*, jednocześnie sprawdzając czy plik posiada szukany model. Nie są to operacje rzutujące na ilość alokowanej pamięci ponieważ tworzą tylko połączenia z plikiem źródłowym.

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 3.4. Algorytm wyszukiwania komponentu po nazwie z uwzględnieniem całych słów (czy ten kod w ogóle jest potrzebny?)

```
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, która różni się w zależności 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.

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 maszynowych zawartych w plikach i zamianę ich na obiekty — tzw. prymitywy (ang. *primitives*), składające się z podstawowych wielokątów i krzywych — a także ekstrakcję informacji o wymiarach. Moduł został zmodyfikowany aby umożliwić mu funkcjonowanie bez dynamicznego pobierania zewnętrznych bibliotek.

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

Renderowanie

Cairocffi jest to oparty o CFFI[2,3] zamiennik modułu *Pycairo* — zestawu powiązań Pythona i obiektowego API dla *Cairo*. *Cairo* jest to biblioteka grafiki wektorowej 2D z obsługą wielu urządzeń 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*. Efektem wyjściowym jest stworzenie obrazu w formacie .PNG i zapisanie go w folderze wskazanym wcześniej przez użytkownika.

3.4. zanim będziemy mogli wczytać plik placement musmy mieć modele - stworzenie bazy modeli, importer *.wrl

3.5. czytanie placement .csv i stawianie modelu

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

ROZDZIAŁ 4

Podsumowanie

https://github.com/adammak23/PCB-Blender

W efekcie końcowym zostały utworzone *de facto* 2 addony.

4.1. Realizacja założonych celów pracy magisterskiej

4.2. Problemy napotkane podczas realizacji systemu

4.2.1. importer *.wrl

4.3. Możliwości rozwoju systemu

wspieranie innych rozszerzeń: https://pcb-tools.readthedocs.io/en/latest/about.html (with planned support for IPC-2581, ODB++ and more.)

4.4. Wnioski

Zakończenie

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 tabel

Spis rysunków

Oświadczenie

Ja, niżej podpisany(a) oświadczam, iż przedłożona praca dyplomowa została wykonana przeze mnie samodzielnie, nie narusza praw autorskich, interesów prawnych i materialnych innych osób.

.....	.....
data	podpis