BLENDER ADD-ON ZA PROCEDURALNO GENERIRANJE ZGRADA

Šimić, Luka

Undergraduate thesis / Završni rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:603523

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

BLENDER ADD-ON ZA PROCEDURALNO GENERIRANJE ZGRADA

Završni rad

Luka Šimić

Osijek, 2019.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 07.07.2019.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada

Ime i prezime studenta:	Luka Šimić
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3988, 19.09.2018.
OIB studenta:	52147607177
Mentor:	Doc.dr.sc. Časlav Livada
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Blender Add-on za proceduralno generiranje zgrada
Znanstvena grana rada:	**Obradba informacija (zn. polje računarstvo)**
Predložena ocjena završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 2 razina
Datum prijedloga ocjene mentora:	07.07.2019.
Datum potvrde ocjene Odbora:	25.07.2019.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.08.2019.

Ime i prezime studenta:	Luka Šimić
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3988, 19.09.2018.
Ephorus podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Blender Add-on za proceduralno generiranje zgrada**

izrađen pod vodstvom mentora Doc.dr.sc. Časlav Livada

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD 1

1.1. Zadatak završnog rada 2

2. KORIŠTENE TEHNOLOGIJE 3

2.1. Blender 3

2.2. Python 3

2.3. BMesh modul i Blender aplikacijski moduli 4

3. BLENDER PROŠIRENJA 5

3.1. Struktura Blender proširenja 5

3.2. Operatori 7

3.3. Korisničko sučelje 7

4. GENERIRANJE SLOŽENIH MODELA NA PRIMJERU ZGRADE 8

4.1. Podjela složenih modela na manje cjeline 9

4.2. Generiranje rasporeda (tlocrta) zgrade 10

4.3. Povezivanje objekata i materijala iz drugih datoteka 12

4.4. Generiranje UV Mapa 13

4.5. Cycles *render engine*, kreiranje i apliciranje materijala 15

4.6. Analiza rada proširenja 17

5. ZAKLJUČAK 20

LITERATURA 23

SAŽETAK 24

ABSTRACT 25

ŽIVOTOPIS 26

1. UVOD

3D modeliranje je proces stvaranja 3D modela koristeći jedan od brojnih dostupnih alata. S obzirom na to da je kreiranje složenih 3D modela kreativan proces, on iziskuje puno vremena i ljudskog rada. Cilj ovog rada je istražiti koncept proceduralnog generiranja 3D modela, odnosno izraditi program koji bi za korisnički definirane parametre dao gotov 3D model spreman za daljnje korištenje. Ovaj rad realizira taj koncept na primjeru generiranja zgrada.

Cilj je istražiti isplativost proceduralnog generiranja 3D modela, bez korištenja modela koji su ranije napravljeni ručno, već generirati cijeli 3D model programski.

Za realizaciju se koristi programski paket Blender, a rad je izrađen u obliku proširenja koristeći Python. Rad uvelike koristi BMesh modul i BMesh operatore koje pruža programski paket Blender.

Dan je opis programskog paketa Blender i programskog jezika Python, zatim uvod u pisanje proširenja u Pythonu, instalacija i korištenje istih, pregled korištenih funkcionalnosti i analiza rada rješenja.

1.1. Zadatak završnog rada

U radu je potrebno opisati mehanizam proceduralnog generiranja objekata u Blenderu. Potrebno je opisati način stvaranja modela iz primitiva te kako napraviti kompoziciju iz više elemenata. Potrebno je napraviti model koristeći parametre (npr. dimenzije, broj katova, broj prozora i sl.) uz prikladne materijale i UV mape.

2. KORIŠTENE TEHNOLOGIJE

U ovom su poglavlju ukratko objašnjene korištene tehnologije i alati potrebni za realizaciju ovog završnog rada. Ti alati uključuju Blender, te Blender Python module, i programski jezik Python.

Uz njih, korišteni su JetBrains PyCharm IDE i Git, te GitHub za kontrolu verzija izvornog koda.

2.1. Blender

Blender je besplatan, open-source alat za 3D modeliranje. Razvija ga zajednica pod okriljem Blender Foundation – Nizozemske javne korporacije. Blender pruža brojne alate za 3D modeliranje, animaciju, simulaciju, rendering i sl. [1]

Uz alate za 3D modeliranje, Blender omogućava korisnicima izradu proširenja (*engl. Addon*) koristeći programski jezik Python. Izradom proširenja moguće je dodati nove specijalizirane funkcionalnosti i alate ili provesti automatizaciju dijela posla.

Prva verzija Blendera (1.0) objavljena je 1994. godine.

U ovom radu korištena je verzija 2.79b.

Tehnički zahtjevi Blendera značajno ovise o kompleksnosti i razini detalja modela s kojim se radi, dok vrijeme potrebno za rendering značajno ovisi o broju uzoraka (*engl. Samples*) i rezoluciji slike. Okvirne tehničke specifikacije potrebne za rad dane su tablicom 2.1.

Tablica 2.1. Tehnički zahtjevi Blendera [2]

Minimum	Recommended	Optimal
32-bit dual core 2Ghz CPU with SSE2 support	64-bit quad core CPU	64-bit eight core CPU
2 GB RAM	8 GB RAM	16 GB RAM
1280x768 Display	Full HD display	Full HD displays
Mouse or trackpad	Three button mouse	Three button mouse
OpenGL 2.1 compatible graphics with 512 MB RAM	OpenGL 3.2 compatible graphics with 2 GB RAM	Dual OpenGL 3.2 compatible grapchi cards with 4 GB RAM

2.2. Python

Python je interpretirani programski jezik visoke razine. Koristi koncepte dinamičkog pisanja i *duck-typinga*, te podržava i objektno orijentirani i funkcionalni pristup. Razlika između Pythona i sličnih programskih jezika (poput Jave, C++ ili C#) je ta da se u Pythonu blokovi koda odvajaju *indentacijom*. [3]

Instalacija Blendera dolazi s Python interpreterom, koji se aktivira pri pokretanju Blendera, i ostaje aktivan. Dio same funkcionalnosti Blendera napisan je u Pythonu, a Python se koristi za pisanje proširenja za Blender.

Blender dolazi s ugrađenom Python konzolom, koja se koristi pri testiranju skripti i proširenja, te s ugrađenim alatom za oblikovanje teksta, koji se može koristiti za pisanje skripti i proširenja. Pomoću Python konzole možemo u stvarnom vremenu pristupati informacijama i objektima sadržanima u trenutno otvorenoj Blender datoteci. Iako je u Blender ugrađen alat za oblikovanje teksta, dobro je koristiti vanjske alate (npr. Visual Studio, PyCharm) koji pružaju dodatne mogućnosti i više funkcionalnosti. [4]

2.3. BMesh modul i Blender aplikacijski moduli

Blender aplikacijski moduli pružaju sučelje i funkcije za interakciju s trenutno otvorenom Blender datotekom putem programskog koda. Sučelje, koje je podijeljeno u više modula, pruža mogućnosti pristupa objektima i ostalim podacima u datoteci, odnosno trenutno aktivnoj sceni, manipulaciju tim istim objektima putem operatora, dodavanje modifiera i sl. Gotovo svaka mogućnost Blendera ima odgovarajuću funkciju dostupnu putem modula te ju možemo pozvati programski. [5]

Operatori i funkcije dostupne u Blender aplikacijskim modulima ovise o trenutnom *bpy.context* objektu. U tom se objektu nalaze informacije o trenutno dostupnim, označenim i aktivnim objektima. Pri radu s operatorima potrebno je obratiti pozornost na *bpy.context* objekt i pratiti koje smo objekte odabrali programski, što se u nekim slučajevima može pokazati zahtjevno i značajno otežava razvoj programskih proširenja.

Na isječku koda 2.1. prikazan je primjer poziva operatora za dupliciranje objekta. Potrebno je prvo sve označene objekte odznačiti, kako bismo izbjegli slučajno dupliciranje neželjenog objekta, zatim programski označiti objekte koje želimo duplicirati te na kraju pozvati operator.

```
for ob in bpy.context.selected_objects:
    ob.select = False

ob = bpy.context.scene.objects["Cube"]
ob.select = True
bpy.context.scene.objects.active = ob
bpy.ops.object.duplicate()
```

Isječak koda 2.1. Primjer korištenja Blender operatora

BMesh je *standalone* modul koji dolazi uz Blender, koji se koristi za uređivanje objekata. Uz osnovne strukture podataka *(vertex, edge, face*) pruža alate, operatore i funkcije za manipulaciju i uređivanje geometrije. [6]

BMesh operatori se razlikuju od standardnih operatora po tome što ne ovise o trenutnom kontekstu, nego vrše operacije nad predanim BMesh objektom. Zbog toga svaki BMesh operator zahtjeva da mu kao parametar bude predan željeni objekt, ali to omogućava rad na više različitih objekata istovremeno, te značajno olakšava razvoj proširenja.

Na isječku koda 2.2 prikazan je primjer poziva BMesh operatora. Operatoru je potrebno proslijediti kao parametar BMesh objekt i geometriju (*verts/edges/faces*) koju želimo duplicirati. Tu geometriju možemo dobiti kao rezultat nekog drugog BMesh operatora. Zbog toga je moguće BMesh operatore ulančavati, odnosno rezultat jednog operatora je moguće proslijediti kao parametar koji je potreban za sljedeći operator. Tako se programski mogu izvoditi složene operacije s geometrijom, koje u sebi sadrže niz jednostavnijih operatora.

```
bm = bmesh.new()
bm.from_mesh(foo_mesh)
geom = bm.verts[:] + bm.edges[:] + bm.faces[:]
ret_dup = bmesh.ops.duplicate(bm, geom=geom)
bmesh.ops.delete(bm, geom=ret_dup["geom"], context=1)
```

Isječak koda 2.2. promjer korištenja BMesh operatora

3. BLENDER PROŠIRENJA

Blender proširenja su manji programi, odnosno skripte, napisane u Pythonu koje proširuju osnovnu funkcionalnost programa, koristeći djelom Blender aplikacijske module. Mogu definirati nove operatore, funkcije, tipove i strukture podataka, elemente korisničkog sučelja i sl.

Blender proširenja se mogu koristiti kao kratke skripte, koje pokrećemo unutar samog Blendera koristeći ugrađeni editor, ili mogu biti korištene kao Python moduli koji se kopiraju u posebno određene mape na računalu. Lokacija mape u kojoj se nalaze proširenja ovisi o samoj instalaciji. Na računalima koje koriste operacijski sustav Windows, ta se mapa najčešće nalazi u „*C:\Program Files\Blender Foundation\Blender\2.79\scripts\addons*“, dok se na Linux operacijskim sustavima proširenja najčešće nalaze u *„usr\share\blender\scripts\addons*“.

3.1. Struktura Blender proširenja

Blender proširenja su strukturirana u obliku Python modula. Iako se cijeli kod može staviti u istu datoteku, dobro je kod raščlaniti u logičke cjeline koje se pišu u više datoteka. Tako je kod lakši za pisanje i održavanje.

Proširenje se može podijeliti u nekoliko glavnih cjelina.

- *__init__.py* datoteka – *entry point* Python modula
- Korisničko sučelje
- Pomoćne (*engl. Utility*) klase i funkcije
- Glavni dio – glavne klase i funkcije, operatori i sl.

__init__.py datoteka je *entry point* Python modula i samog proširenja. U njoj se nalaze funkcije za inicijalizaciju, odnosno registraciju pisanih klasa koje se koriste unutar samog proširenja te *bl_info* struktura, koja sadrži informacije o samom proširenju, poput autora, potrebne verzije Blendera i sl. Primjer *__init__*.py datoteke i *bl_info* strukture dan je u isječku koda 3.1.

```
bl_info = {
    "name": "Procedural building generator",
    "description": "Proceduraly generate and edit buildings",
    "author": "Luka Šimić",
    "version": (0, 8, 0),
    "blender": (2, 79, 0),
    "location": "View3D > Toolbox > PBG",
    "warning": "Under development. Might cause stability issues.",
    "wiki_url": "https://github.com/lsimic/ProceduralBuildingGenerator/wiki",
    "tracker_url":"https://github.com/lsimic/ProceduralBuildingGenerator/issues”,
    "support": "COMMUNITY",
    "category": "Add Mesh"
}

def register():
def unregister():
```

Isječak koda 3.1. primjer __init__.py datoteke i bl_info strukture

Korisničko sučelje sadrži definicije svojstava koje su potrebne za funkcionalnost proširenja, postavljanje njihovih početnih vrijednosti, definiranje panela i izgleda sučelja, te njegovu lokaciju unutar Blendera, te povezivanje operatora s elementima korisničkog sučelja.

Glavni dio sadrži osnovne klase i funkcije koje sadržavaju glavninu funkcionalnosti proširenja, u njima se nalaze operatori, funkcije za generiranje, proračune i sl.

Pomoćne klase i funkcije su one koje se koriste na više mjesta u kodu. One ne sadrže osobitu funkcionalnost ali su potrebne za sam rad proširenja. Moguće je iste pomoćne klase i funkcije koristiti u više različitih projekata, pa ih je zbog toga dobro razdvojiti u zaseban dio.

3.2. Operatori

Operatori su funkcije koje možemo pozivati unutar Blendera kako bi izvršili određenu funkciju. Kako bi operatori bili dostupni, potrebno ih je registrirati. Nakon registracije operator se može pozvati putem dinamičkog menija (*dynamic spacebar menu*) ili putem definiranog elementa korisničkog sučelja. Uz to, moguće je i definirati operatore koji se vrte u pozadini, ili se izvršavaju na određene događaje (*engl. Event)* poput *timera.* Takvi operatori se u Blenderu nazivaju modalni operatori (*engl. Modal operator*). Primjer jednostavnog operatora i njegove registracije dan je isječkom koda 3.1. Operator u primjeru nakon pozivanja ispisuje poruku na konzolu.

```
class MyOperator(bpy.types.Operator):
    bl_idname = "bar.my_operator"
    bl_label = "My Operator"

    def invoke(self, context, event):
        print("operator called")
        return {"FINISHED"}

def register():
    bpy.utils.register_class(MyOperator)
```

Isječak koda 3.2. Primjer jednostavnog Blender operatora

3.3. Korisničko sučelje

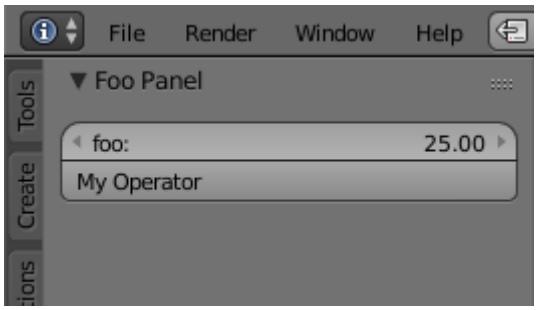
Korisničko sučelje proširenja definira se u nekoliko koraka. Prvo je potrebno odrediti parametre koji su potrebni za rad operatora i funkcija. Te parametre je potrebno staviti unutar *PropertyGroup* klase koju je potrebno registrirati. Zatim treba definirati panele koji sadrže dane parametre, te njihovu lokaciju. Primjer definicije parametra (*engl. Property*) i panela korisničkog sučelja dan je u isječku koda 3.3. Na slici 3.1. se nalazi panel, koji je nastaje kao rezultat danog koda.

```
class FooPropertyGroup(PropertyGroup):
    foo_property = FloatProperty(
        name="foo",
        default=25.0
    )

class FooPanel(Panel):
    bl_label = "Foo Panel"
    bl_category = "FOO"
    bl_space_type = "VIEW_3D"
    bl_region_type = "TOOLS"
    bl_context = "objectmode"

    def draw(self, context):
        col = self.layout.column(align=True)
        col.prop(context.scene.FooPropertyGroup, "foo_property")
        col.operator("bar.my_operator", text="My Operator")
```

Isječak koda 3.3. Primjer definiranja korisničkog sučelja



Slika 3.1. Primjer izgleda panela korisničkog sučelja

Kako bismo mogli dohvatiti podatke koji su uneseni putem definiranog korisničkog sučelja, prvo je potrebno klasu koja sadržava parametre (*PropertyGroup*) i klasu koja sadržava panel registrirati unutar *register()* funkcije, koja se nalazi u *__init__.py* datoteci. Zatim podacima možemo pristupiti na način prikazan u isječku koda 3.3.

```
properties = bpy.context.scene.FooPropertyGroup
val = properties.foo_property
print(val)
```

Isječak koda 3.4. primjer dohvaćanja vrijednosti korisnički definiranog parametra

4. GENERIRANJE SLOŽENIH MODELA NA PRIMJERU ZGRADE

U ovom je poglavlju objašnjen pristup modeliranju složenih 3D modela koristeći Blender i BMesh API. Generiranje složenih modela može se podijeliti na nekoliko osnovnih koraka. To su:

- Analiza željenog modela
- Podjela modela na manje logičke cjeline, odnosno elemente
- Generiranje manjih pojedinih elemenata
- Generiranje UV mapa, postavljanje materijala
- Dupliciranje i postavljanje manjih pojedinih elemenata

U ovom je radu generiranje složenih modela dano na primjeru generiranja zgrada, koje prate stil secesije. Secesijski stil je odabran zbog toga što su njegovi oblici dovoljno kompleksni kako bi se na njima temeljito istražio koncept proceduralnog generiranja. Također, zgrade u secesijskom stilu su česte u Osijeku, pa samim tim postoji pristup velikoj količini primjera. Na slici 4.1. je prikazan primjer secesijske zgrade.

Slika 4.1. Zgrada županijskog suda u Osijeku

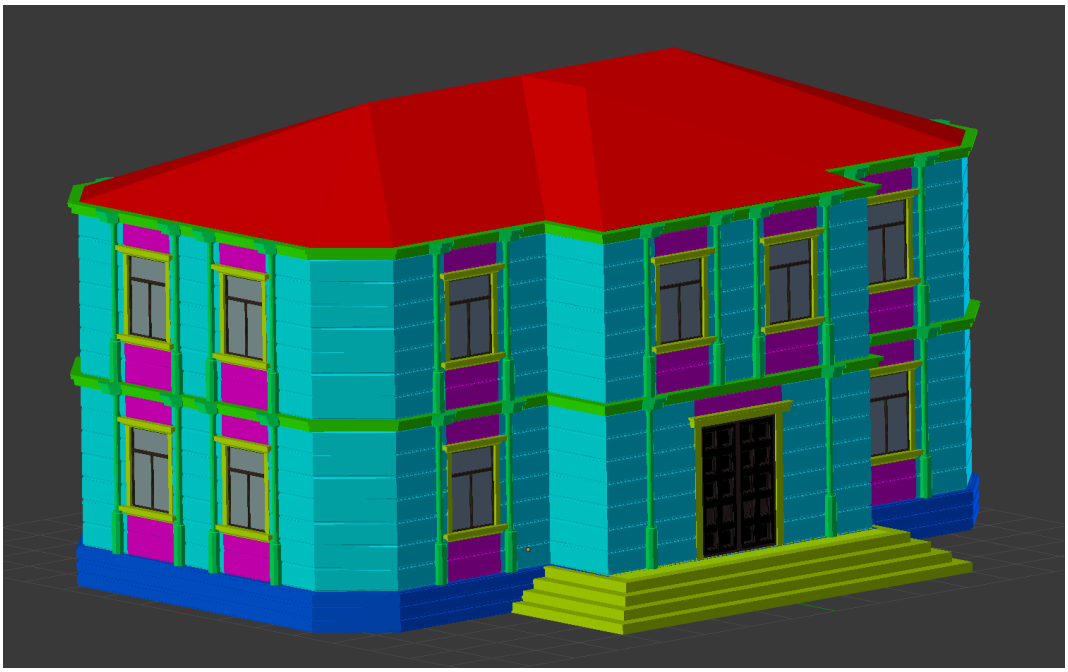
4.1. Podjela složenih modela na manje cjeline

Složeni model je lakše generirati ako se podjeli na manje cjeline. Na taj će način svaka cjelina imati manje geometrije, pa je zbog toga lakše pratiti položaj točaka, te se olakšava njihov odabir i rad s operatorima. Model se može podijeliti prema određenim kriterijima. Neki od kriterija koje treba uzeti u obzir su:

- Je li ta cjelina i na stvarnom modelu fizički odvojena od ostatka (npr. vrata su fizički odvojena od zidova)
- Pojavljuje li se određena cjelina više puta (npr. prozori)
- Koristi li određena cjelina drugačije materijale ili *modifiere*
- Treba li određena cjelina biti lako zamjenjiva bez utjecaja na ostatak modela

Nakon podjele, svaka se cjelina može generirati neovisno jedna od druge. Ako se određena cjelina pojavljuje više puta, potrebno ju je generirati samo jednom, te ju duplicirati i pomaknuti na potrebno mjesto.

Primjer podjele složenog modela na cjeline na primjeru zgrade dan je na slici 4.2. Različite cjeline naznačene su različitim bojama



Slika 4.2. Primjer podjele složenog modela na cjeline

Na primjeru zgrade, možemo vidjeti cjeline na koje je ona podijeljena. Neke od cjelina su krov, vrata, zidovi, prozori i sl. Za svaku od tih cjelina postoji jedna funkcija koja služi za njihovo generiranje. Svaka funkcija za generiranje ima parametre koji su za nju specifični, te parametre koji su zajednički za sve cjeline. Na primjer, parametar *visina_kata* potreban je više funkcija, dok je parametar *broj_stepenica* potreban samo funkciji za generiranje stepenica. Svaka funkcija ima svoje parametre grupirane u određenu klasu, te odgovarajući panel na korisničkom sučelju. Svaka klasa s parametrima ima metodu koja dohvaća parametre podešene putem korisničkog sučelja.

4.2. Generiranje rasporeda (tlocrta) zgrade

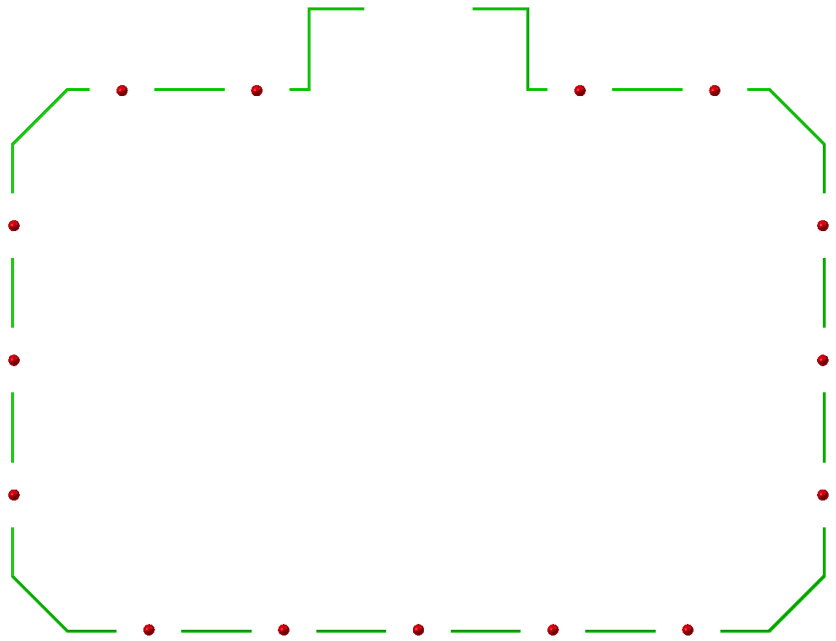
Raspored zgrade podrazumijeva položaje svih prozora, stupova i zidova koji se generiraju. Za generiranje rasporeda zgrade potrebni su parametri poput osnovnih dimenzija zgrade, veličine prozora i slično. Uz to, funkcija kao parametar prima vanjski tlocrt zgrade, u obliku poredane liste točaka.

Moguće je funkciju za generiranje rasporeda izmijeniti tako da ona prima *BMesh* ili *Mesh* objekt, ali bi u tom slučaju bilo potrebno provjeriti i poredati sve točke predanog objekta.

Funkcija za svaki vanjski zid zgrade prvo proračunava položaje prozora, uz poznate dimenzije i ostale parametre. Zatim uzimajući u obzir položaje prozora, računa točke koje definiraju zidove. Kao rezultat vraća *dictionary* koji sadržava liste svih pozicija prozora i stupova koji su definirani kao *tuple* oblika *((x_lokacija, y_lokacija, z_lokacija), rotacija_z)*. Pozicije zidova vraća u obliku liste, koja sadržava liste točaka prema kojima se *ekstrudiraju* zidovi.

Funkcija uzima u obzir položaj vrata, te je zbog toga raspored prvog kada, odnosno prizemlja drugačiji od rasporeda za ostale katove.

Pojednostavljeni prikaz rasporeda nalazi se na slici 4.3. Na slici su vidljivi položaj zidova i položaji prozora.



Slika 4.3. Pojednostavljeni prikaz rasporeda zgrade

Nakon generiranja pojedinih dijelova/cjelina zgrade, oni se predaju funkciji *apply_positions()* koja kao parametar prima raspored elemenata. Stvara povezane kopije predanih objekata (*linked duplicate*), te ih translatira na odgovarajuću poziciju i postavlja im odgovarajuću rotaciju na Z osi. Funkcija je dana u isječku koda 4.1.

```
def apply_positions(obj: bpy.types.Object, positions: list, group):
    for position in positions:
        dup = obj.copy()
        group.objects.link(dup)
        dup.location.x = position[0][0]
        dup.location.y = position[0][1]
        dup.location.z = position[0][2]
        dup.rotation_euler.z = position[1]
        bpy.context.scene.objects.link(dup)
```

Isječak koda 4.1. funkciaj apply_positions()

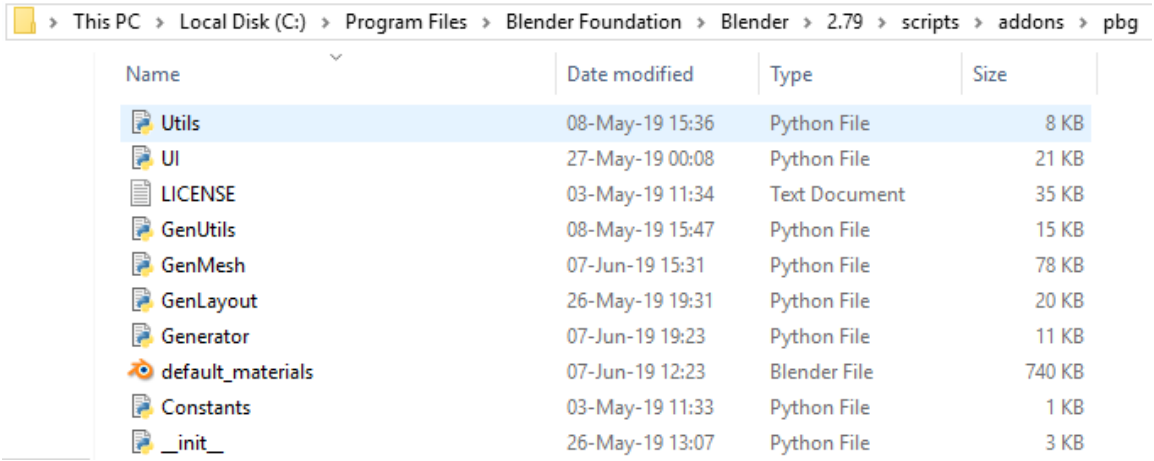
Funkcija *apply_positons()* također prima i parametar *group*. U tom se parametru nalazi grupa u koju grupiramo sve objekte koji se generiraju. Ta se grupa koristi kako bi se pri ponovnom generiranju moglo lakše doći do objekata generiranih prethodnim pozivom za njihovo brisanje i kako bi poboljšala organizaciju unutar same Blender scene.

4.3. Povezivanje objekata i materijala iz drugih datoteka

Moguće je programskim putem dodati, odnosno povezati dijelove postojećih datoteka, odnosno objekte, materijale i sl. iz drugih datoteka s trenutnom.Tako jedna ili više datoteka se mogu koristiti kao baza, u kojima se nalaze objekti potrebni za rad našeg proširenja, ali koje nema smisla svaki puta generirati.

Za povezivanje s drugim datotekama potrebno je prvo znati putanju do njih. Pomoću Blender API-ja ili ugrađenih Python funkcija moguće je doći do putanje trenutno otvorene Blender datoteke, ili trenutno aktivne Python datoteke. Nakon toga je jednostavno doći do tražene Blender datoteke ako je poznata relativna putanja.

Na primjeru proširenja za generiranje zgrada, Blender datoteka koja služi kao baza nalazi se u istoj mapi kao i Python skripta. Struktura datoteka proširenja prikazana je na slici.



Slika 4.4. Struktura proširenja

Gledajući putanju, datoteka *default_materials.blend* se nalazi u istoj mapi u kojoj se nalazi i datoteka *Generator.py*. Zbog toga, ako je poznata lokacija datoteke *Generator.py* lako je doći do potrebne putanje. To se postiže korištenjem ugrađene Python biblioteke *os* odnosno pozivom funkcije *os.path* kako je prikazano u isječku koda 4.2.

```
def load_materials() -> dict:
    directory = os.path.dirname(os.path.realpath(__file__))
    path = directory + "\\" + "default_materials.blend"
    materials = dict()
    with bpy.data.libraries.load(path, link=False) as (data_from, data_to):
        for m_name in data_from.materials:
            bpy.ops.wm.append(filename=l_name, directory=path + "\\Material\\")
            materials[m_name] = bpy.data.materials[m_name]
    return materials
```

Isječak koda 4.2. Učitavnje materijala iz vanjske datoteke

Pozivom operatora *bpy.ops.wm.append()* uključujemo željeni objekt, u ovom slučaju materijal, iz datoteke određene putanjom u trenutno otvorenu Blender datoteku.

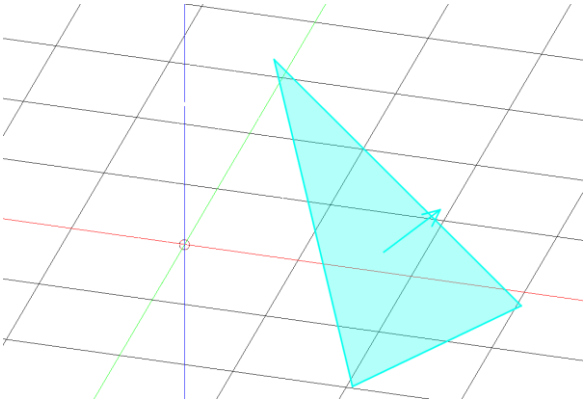
U funkciji se zbog jednostavnosti pristupa svi materijali spremaju u *dictionary* koji se vraća kao rezultat funkcije, ali materijali su bez obzira na to dostupni, i mogu se dodjeljivati objektima.

4.4. Generiranje UV Mapa

UV mape se koriste kada je potrebno 2D teksturu, odnosno sliku, primijeniti na 3D model. Proces UV mapiranja sastoji se da svakoj točki 3D modela pridružimo lokaciju na 2D teksturi, odnosno UV koordinate.

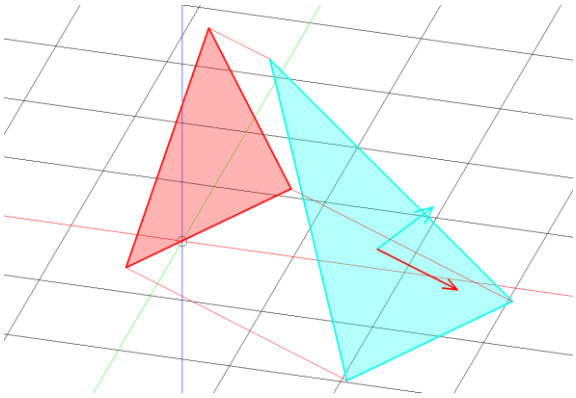
Blender pruža nekoliko gotovih načina, odnosno funkcija, za generiranje UV mapa, ali funkcije koje taj dio posla odrađuju automatski često ne daju zadovoljavajuće rezultate pa je zbog toga potrebno dodatno ručno uređivati UV mape. Zbog toga je potrebno implementirati funkciju koja će odraditi UV mapiranje na točno određen način koji će davati konzistentne i jednake rezultate za svaki *face* objekta.

Funkcija radi posebnu projekciju za svaki poligon objekta, Jedan poligon objekta s pripadajućom normalom prikazan je na slici 4.6.



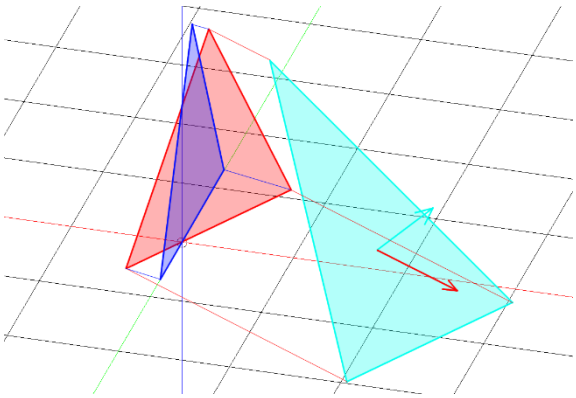
Slika 4.5. Generiranje UV mapa – početni poligon

Pomoću BMesh operatora se dohvati normala poligona. Za daljnje računanje potrebna je samo x-y komponenta normale. Pomoću x-y normale stvara se matrica ortogonalne projekcije, te se svaka točka poligona množi s tom matricom. Kao rezultat dobiva se projekcija poligona na ravninu definirana normalom u x-y osi i u ishodištu koordinatnog sustava, prikazano na slici 4.6.



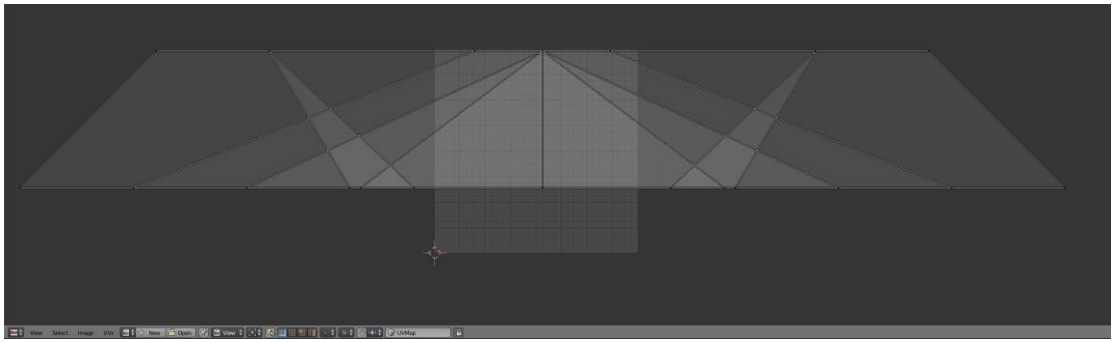
Slika 4.6. Generiranje UV mapa - projekcija poligona u x-y smjeru normale

Nakon toga, potrebno je rotirati projekciju. Kut rotacije računa se kao kut između x osi koordinatnog sustava i x-y smjera normale. Pomoću izračunatog kuta stvara se matrica rotacije. Množenjem svake točke s matricom rotacije dobivamo kao rezultat poligon koji se nalazi u y-z ravnini, prikazano na slici 4.7. YZ koordinate svake točke tada odgovaraju njenim UV koordinatama.



Slika 4.7. Generiranje UV mapa - krajnji položaj poligona

Funkcija za generiranje UV mapa ne generira novu geometriju, već samo množenjem matrica projekcije i matrica rotacije računa nove koordinate za svaku točku poligona. Funkcija je prikazana u isječku koda 4.3. UV mapa krova koja je rezultat ove funkcije prikazana je na slici 4.8.



Slika 4.8. UV mapa krova (ručno pomaknuta i smanjena radi bolje prezentacije)

Glavna prednost ovog pristupa je ta što su točke koje se nalaze na istoj visini na modelu, nalaze na istoj visini na UV mapi. To je bitno jer se na taj način smanjuje vidljivost prekida u UV mapi (*engl. Seams*).

```
def uv_unwrap(bm):
    uv_layer = bm.loops.layers.uv.verify()
    bm.faces.layers.tex.verify()
    bm.verts.ensure_lookup_table()
    for face in bm.faces:
        no = face.normal
        vec_no = mathutils.Vector((no[0], no[1], 0.0))
        vec_no.normalize()
        mat_proj = mathutils.Matrix.OrthoProjection(vec_no, 3)
        vec_x = mathutils.Vector((1.0, 0.0, 0.0))
        angle = vec_x.xy.angle_signed(vec_no.xy)
        mat_rot = mathutils.Matrix.Rotation(angle, 3, "Z")
        for loop in face.loops:
            loopuv = loop[uv_layer]
            vert_proj = mat_proj * loop.vert.co
            vert_uv_co = mat_rot * vert_proj
            uv = (vert_uv_co[1], vert_uv_co[2])
            loopuv.uv = uv
```

Isječak koda 4.3. Funkcija za generiranje UV mape

Za prolazak kroz poligone i njihove točke koristi se BMesh API, dok se za stvaranje matrica koristi *mathutils* biblioteka (koja dolazi ugrađena u Blender).

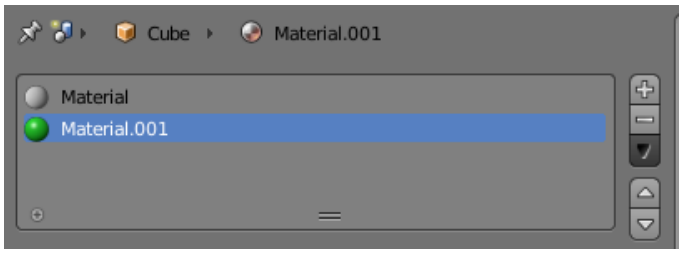
4.5. Cycles *render engine*, kreiranje i apliciranje materijala

Materijali se u Blenderu apliciraju na objekte koristeći *material slotove*. Tako, ako objekt ima više *slotova* za materijale, moguće je na isti objekt aplicirati više njih. Svakom poligonu objekta može biti pridružen jedan *material slot*. Pridruživanje se može izvesti koristeći BMesh API, na način koji je prikazan isječkom koda 4.4. Na istom isječku koda je prikazano dodavanje dva različita materijala na objekt. Nakon izvođenja danog koda, na objektu se nalaze dva materijala, kao što je prikazano na slici 4.9.

```
me = cube.data
bm.faces.ensure_lookup_table()
for i in range(0, 3):
    bm.faces[i].material_index = 1

bm.to_mesh(me)
cube.materials.append(py.data.materials["Material"])
cube.materials.append(py.data.materials["Material.001"])
```

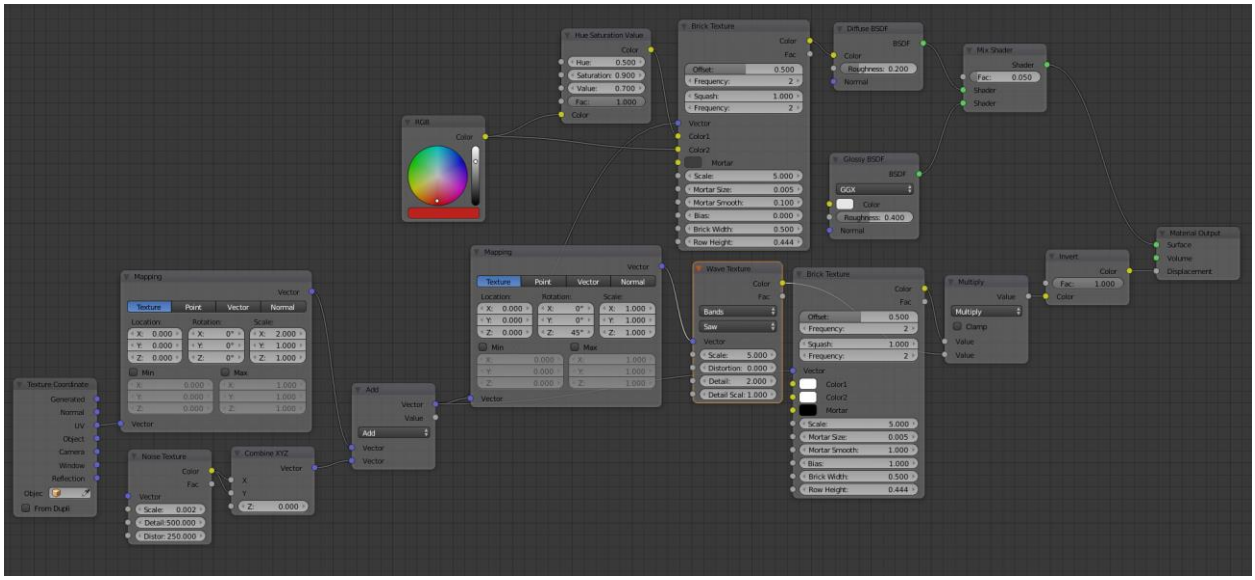
Isječak koda 4.4. Apliciranje materijala na objekt i poligone



Slika 4.9. Materijali na objektu

Blender u verziji 2.79b sadrži dva *render engine-a – Cycles* i *Internal*. *Internal render engine* će u verziji 2.80 biti zamijenjen sa EEVEE *engineom*.

Materijali se za *Cycles render engine* kreiraju koristeći *node editor*. Za potrebe projekta kreirano je pet različitih materijala koji se apliciraju na generirane zgrade. Od tih materijala, samo materijal koji se aplicira na krov ima složenu strukturu *nodeova* koja je prikazana na slici 4.10.

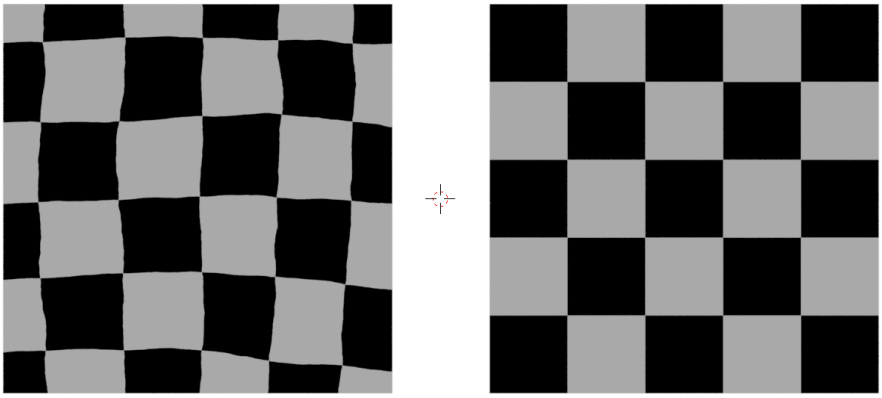


Slika 4.10. Cycles materijal krova zgrade

Materijal se može podijeliti na 3 glavne cjeline

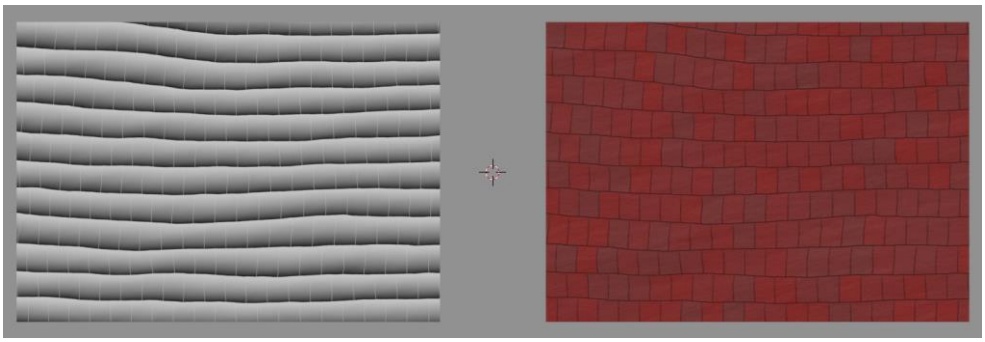
- Dodavanje varijacije u generiranu UV mapu
- Generiranje *displacement* mape
- Generiranje *shadera*

Varijacija se u generiranu UV mapu dodaje korištenjem *Noise Texture nodea*, koji se množi s UV mapom objekta. Na taj način se u UV mapu unosi izobličenje, te se linije koje bi bez UV mape bile prikazane ravno, prikazuju blago izobličene, kao što je prikazano na slici 4.11. Razinu izobličenja i *scale* UV mape moguće je promijeniti izmjenom parametara *noise texture nodea*, odnosno *mapping* nodea.



Slika 4.11. Materijal s izobličenjem (lijevo) i bez izobličenja (desno)

Kao osnova za materijal koristi se *brick texture*, jedan se *brick texture* koristi za određivanje boje crjepova, dok se drugi koristi za generiranje *displacement* mape. *Displacement* mapa se generira kao kombinacija gore navedene *brick texture* i *Wave texture*, pomoću koje se postiže dojam nagiba crjepova. *Displacement* mapa i materijal prikazani su na slici 4.12.



Slika 4.12. Displacement mapa (lijevo) i materijal (desno)

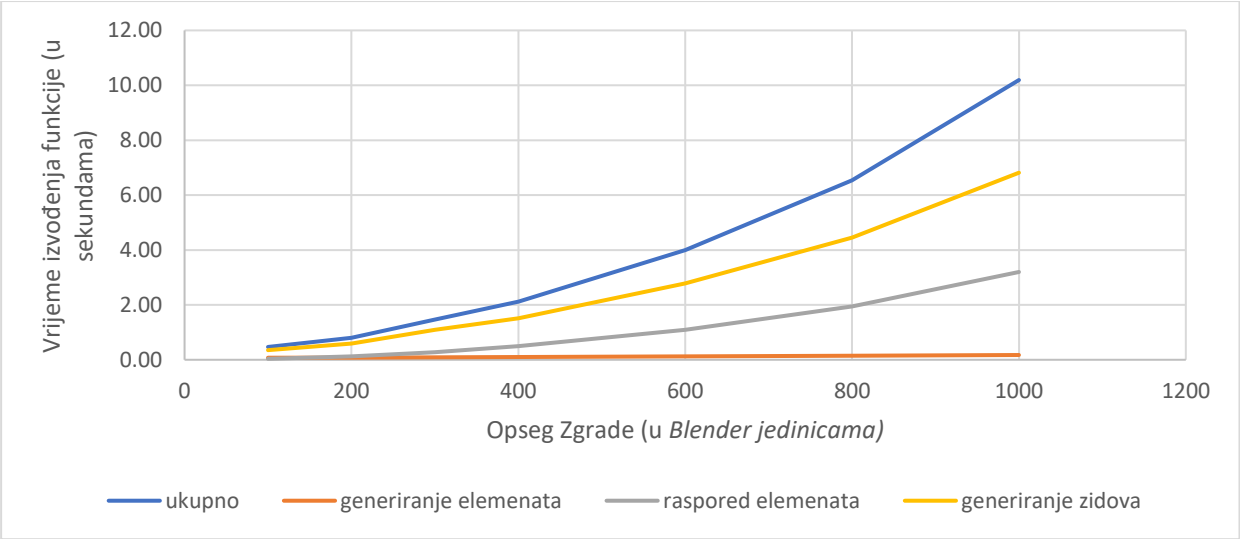
4.6. Analiza rada proširenja

Analiza rada proširenja provedena je mjerenjem vremena koje je potrebno za izvođenje pojedinih funkcija, u ovisnosti o promijenjenoj širini i visini zgrade, dok svi ostali parametri ostaju nepromijenjeni. U vrijeme izvođenja nije uračunato vrijeme potrebno za brisanje objekata koji su ostali kao rezultat prethodnog poziva funkcije. Primjer mjerenja vremena izvođenja dan je isječkom koda 4.5.

```
time_start = time.time()
obj_window = GenMesh.gen_mesh_windows(context, params_general, params_windows)
group.objects.link(obj_window)
time_end = time.time()
msg = "obj_window generated in: " + str(time_end - time_start) + " seconds"
print(msg)
```

Isječak koda 4.5. Primjer mjerenja vremena izvođenja funkcije

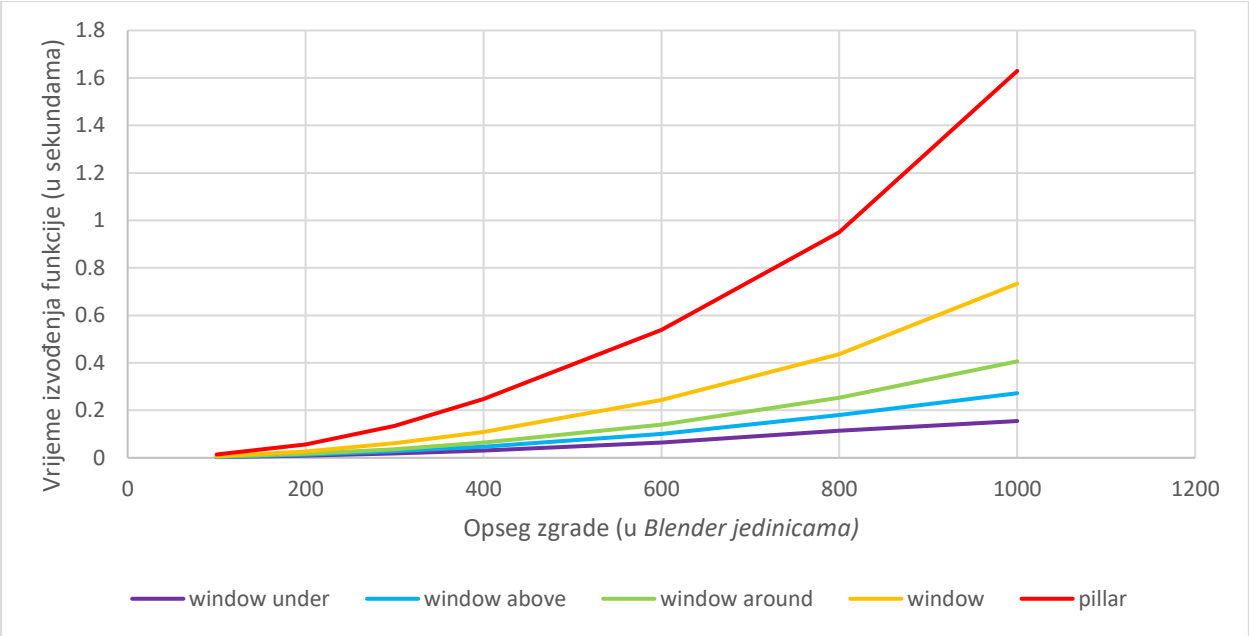
Na grafu 4.1. prikazana su vremena izvođenja pojedinih funkcija u ovisnosti o dimenzijama, odnosno opsegu zgrade. Iz grafa je vidljivo da na vrijeme generiranja najviše utječe generiranje zidova.



Graf 4.1. Ovisnost vremena izvođenja funkcije o dimenzijama

Na grafu 4.1. su ostale funkcije grupirane u dvije kategorije – funkcije koje služe za generiranje pojedinih elemenata i funkcije koje te iste elemente raspoređuju prema generiranom rasporedu. Vidljivo je da funkcije koje generiraju pojedine elemente ne ovise o veličini zgrade jer se pozivaju samo jednom, dok funkcije koje raspoređuju dane elemente imaju veće vrijeme izvođenja ovisno o dimenzijama zgrade.

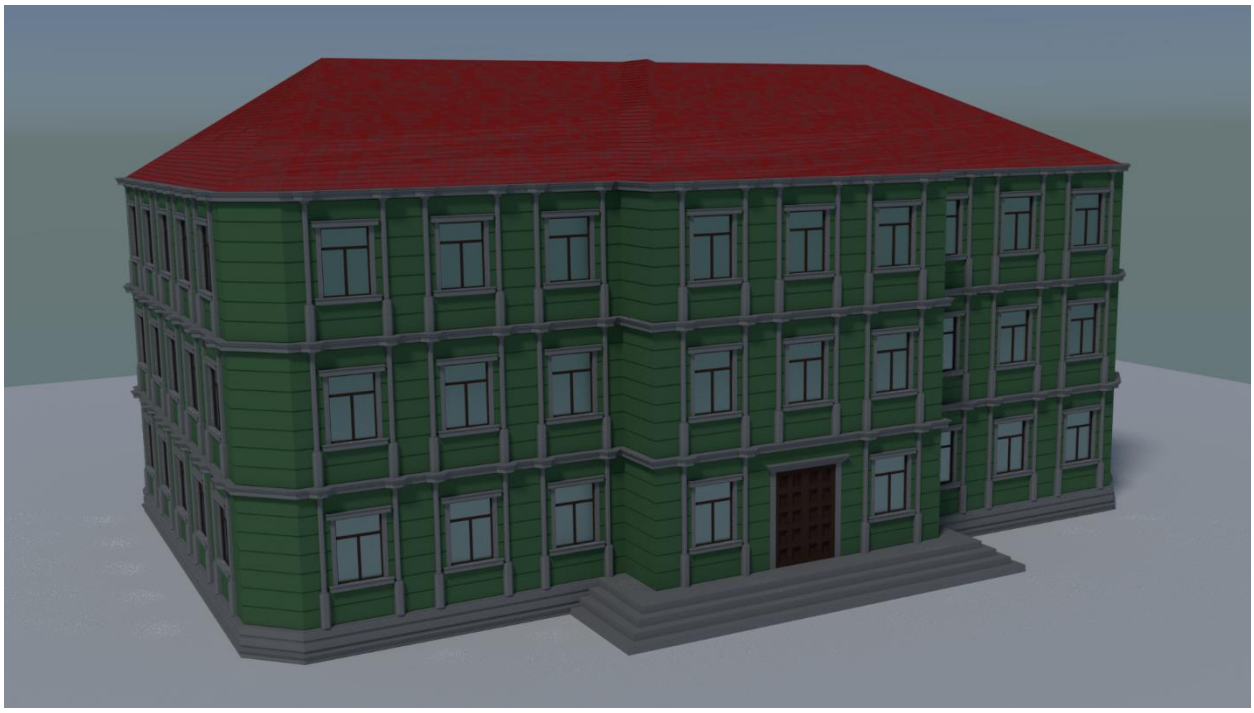
Na isti se način može promatrati vrijeme rasporeda pojedinih komponenti u odnosu na dimenzije zgrade. Vidljivo je da je najviše vremena potrebno za generiranje „stupića“ jer je njihov broj dvostruko veći u odnosu na broj ostalih elemenata. Vrijeme rasporeda elemenata ovisi o kompleksnosti geometrije samih elemenata. Prozori, koji imaju najkompleksniju geometriju i najviše poligona, imaju i najveće vrijeme raspoređivanja, dok ostali elementi, koji imaju jednostavniju geometriju i manji broj poligona imaju manje vrijeme raspoređivanja.



Graf 4.2. Vrijeme raspoređivanja pojedinih elemenata u odnosu na dimenzije

5. ZAKLJUČAK

Proširenje generira modele koji su spremni za *render* koristeći *cycles engine*. Primjer rendera dan je na slici 5.1. Za osvjetljenje se uz *sun lamp* sa zadanim postavkama koristi i *sky texture*, također sa zadanim postavkama.



Slika 5.1. Cycles render generirane zgrade

U poglavlju 4.6. dani su rezultati mjerenja vremena izvođenja pojedinih funkcija, iz kojih je vidljivo da najveći dio vremena, u prosjeku 70% ukupnog vremena izvođenja odlazi na generiranje zidova. Razlog tomu je taj što generiranje zidova za sve dijelove koristi BMesh API. Zbog toga što BMesh koristi kompleksnije strukture podataka i što se koriste kompleksniji operatori, vrijeme izvođenja funkcije za generiranje zidova je relativno veliko.

Generiranje zidova može se ubrzati izmjenom funkcije za generiranje rasporeda/tlocrta zgrade kako bi ona vraćala dimenzije i lokacije dijelova zidova koji se ponavljaju (npr. dijelovi zidova koji se nalaze između prozora uvijek su jednakih dimenzija). Zatim bi se također trebala izmijeniti sama funkcija za generiranje zidova, kako bi generirala dijelove zidova koji se ponavljaju i omogućila njihovo kopiranje i odvojeno generirala dijelove zidova koji se ne ponavljaju u modelu.

Ovaj je pristup korišten za ostale dijelove zgrade. Tijekom razvoja proširenja, za kopiranje pojedinih dijelova koristio se BMesh API. Promjenom kopiranja pojedinih dijelova iz korištenja BMesh operatora u stvaranje povezanih kopija (*engl. Linked duplicates*) ostvareno je ubrzanje od gotovo 30 puta.

Još jedna prednost korištenja povezanih kopija je ta što je omogućeno lakše dodavanje izmjena na dijelove, jer je potrebno izmijeniti bilo koji od kopiranih objekata kako bi se promjene primijenile na sve njih.

Negativna strana korištenja vezanih kopija je ta što Blender, u slučaju velikog broja objekata u jednoj sceni/datoteci može raditi sa smanjenim performansama. Iako to nije problem u slučaju ovog proširenja, taj se problem može pojaviti kod proširenja koja rade s još većim brojem individualnih objekata.

Još jedan način na koji bi se moglo ostvariti značajno poboljšanje performansi je uvođenje višenitnosti. Određene funkcije za generiranje i raspored objekata mogle bi se izvoditi u više niti paralelno. Tako bi se ostvarilo ubrzanje na višejezgrenim procesorima, ali bi to ubrzanje bilo osjetno tek pri generiranju velikih modela.

Ovaj koncept je moguće nadograditi na više načina, pomoću kojih bi on imao još veću korist i mogao bi se primijeniti u industriji.

Dodavanjem mogućnosti za izvoz/uvoz parametara, npr. u *json* formatu omogućila bi se razmjena samih parametara, a ne gotovog modela. Razmjenom samo parametara bi bila značajno bolja, jer bi samo parametri, odnosno *json* datoteka koja ih sadržava, bili nekoliko redi veličine manji nego gotov, generiran model.

Uz to, proširenje trenutno, zbog jednostavnosti, svaki puta pri ponovnom pozivu briše sve objekte koji su prethodno generirani. Zbog toga se pri ponovnom pozivu funkcije brišu objekti koji će biti generirani na isti način i imati isti rezultat. To se može poboljšati praćenjem parametara koji su promijenjeni od zadnjeg poziva funkcije, i generirati samo one dijelove koje je zaista potrebno.

Također bi bilo poželjno napraviti validaciju unesenih podataka i parametara, jer parametri koji su uneseni u nekim slučajevima mogu uzrokovati neočekivane rezultate, nestabilnost u radu programa, a u najgorem slučaju mogu uzrokovati rušenje programa. Validacija unesenih parametara bila bi kompleksna, zbog velikog broja parametara koje je moguće mijenjati i zbog velikog broja parametara čije vrijednosti ovise o drugima.

Ovakav pristup stvaranju 3D modela može se, uz dodatni razvoj, primijeniti u industriji 3D računalnih igara. U tom bi slučaju omogućio brzo generiranje velikog broja modela (*asseta*) koji bi se mogli jednostavno prilagoditi potrebama. Uz to, može se primijeniti i pri kreiranju arhitekturnih vizualizacija, filmova i generalno u *CGI* industriji.

Primjenom sličnih koncepta, uz značajne izmjene, može se i ostvariti funkcija koja bi prema zadanima parametrima generirala velik broj verzija prototipa proizvoda, koji bi se zatim mogli testirati i analizirati kako bi se odredila optimalna verzija.

LITERATURA

[1] Blender, »About - blender.org,« [Mrežno]. Dostupno: https://www.blender.org/about/. [Pokušaj pristupa srpanj 2019].

[2] Blender, »Requirements,« [Mrežno]. Dostupno: https://www.blender.org/download/requirements/. [Pokušaj pristupa lipanj 2019].

[3] Python Software Foundation, »Python.org,« [Mrežno]. Dostupno: https://www.python.org/. [Pokušaj pristupa lipanj 2019].

[4] Blender, »API overview,« [Mrežno]. Dostupno: https://docs.blender.org/api/current/info_overview.html. [Pokušaj pristupa lipanj 2019].

[5] Blender, »Blender - Quickstart Introduction,« [Mrežno]. Dostupno: https://docs.blender.org/api/current/info_quickstart.html. [Pokušaj pristupa srpanj 2019].

[6] Blender, »BMesh Module,« [Mrežno]. Dostupno: https://docs.blender.org/api/current/bmesh.html. [Pokušaj pristupa lipanj 2019].

SAŽETAK

Ovaj rad daje uvod u pisanje proširenja za programski paket Blender koristeći programski jezik Python. Opisuje osnovne operatore za manipulaciju podataka unutar Blendera, kreiranje, kopiranje i brisanje objekata, te daje pregled i primjere korištenja BMesh modula i njegovih operatora. Uz to, daje primjer programskog generiranja UV mapa.

Proceduralno generiranje zgrada izvedeno je podjelom većeg modela na manje cjeline, koje se generiraju odvojeno koristeći BMesh modul, a zatim se kopiraju na mjesta koja su prethodno određena programski.

Ključne riječi: Blender, Proceduralno generiranje, Proširenje, Python, 3D modeliranje

ABSTRACT

This thesis gives an introduction into writing add-ons for Blender using Python programming language. The thesis describes basic operators for manipulating data inside Blender, such as creating, copying and deleting objects, and gives an overview and examples of using the BMesh module and its operators. It also gives an example of generating custom UV maps using Python.

Procedural generation of buildings was accomplished by dividing the larger model into smaller sections, which are generated separately using the BMesh module. They are then copied and placed to coordinates which were programmatically determined before.

Keywords: Add-On, Blender, Procedural Generation, Python, 3D modelling,

ŽIVOTOPIS

Luka Šimić rođen je 4. rujna 1997. godine u Osijeku. Nakon završenog osnovnog i srednjoškolskog obrazovanja, 2016. godine upisuje Fakultet Elektrotehnike, Računarstva i Informacijskih Tehnologija u Osijeku, preddiplomski smjer Računarstvo. Radi kao freelance 3D artist. Aktivan je u lokalnoj IT zajednici i sudjeluje u njenim brojnim projektima.