Středoškolská technika 2024

Setkání a prezentace prací středoškolských studentů na ČVUT

VYTVÁŘENÍ MODELŮ DOMŮ V PYTHONU

Vojtěch Fila

Gymnázium Aloise Jiráska
T. G. Masaryka 590, Litomyšl

Prohlášení

Prohlašuji tímto, že jsem soutěžní práci vypracoval samostatně pod vedením paní Mgr. Markéty Bartošové a uvedl v seznamu literatury veškerou použitou literaturu a další zdroje včetně internetu.

Prohlašuji rovněž, že tištěná a elektronická verze této práce jsou shodné.

Nemám závažný důvod, který by bránil zpřístupnění této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů.

V Litomyšli dne 15. 5. 2024 Vojtěch Fila

Anotace

Tato práce se zabývá problematikou generování střech z půdorysu domu a následný tisk 3D modelu výsledného domu. Tato práce má za úkol zjednodušeně objasnit, jak funguje celý kód, které programy byly užity pro tvorbu výsledného kódu, důvody pro jejich užití a také vysvětlit do hloubky jak fungují nejsložitější části kódu. Tato práce je určená pro laiky i pro experty v oboru programování, ale je potřeba základní znalost teoretických řešení střech pro plné pochopení práce. Největší část této práce pojednává o způsobu fungování vypracovaného kódu, poté komentuji modely použité při vývoji a jejich přínos k vývoji.

Klíčová slova

Python; teoretické řešení střech; modely domů; automatizace střech.

Poděkování

Chtěl bych poděkovat mým rodičům za podporu ve všemožných soutěžích a vzdělávacích seminářích, která neztrácí na síle a která mě motivovala pracovat na této práci. Dále bych chtěl poděkovat paní Bartošové, která vedla celou moji práci a dokázala tolerovat mé pozdní odevzdání pracovních verzí. Nakonec bych chtěl poděkovat paní Flaškové, která mě poprvé uvedla do problematiky řešení střech, která mi přišla tak nudná, že jsem se ji rozhodl automatizovat.

Obsah

Úvod

Tuto práci jsem původně začal dělat jako menší projekt pro pobavení. Jednoho dne jsem si uvědomil, jak nudné bylo řešit střechy v rámci deskriptivní geometrie a rozhodl jsem se, že bude zábavnější zkusit tento proces automatizovat. Můj kód se postupně stával složitější a složitější a až po dosažení prvních výsledků mě napadlo, že by bylo možné z toho udělat celou práci.

Tato práce se bude zabývat postupným vysvětlením fungování mého kódu, různých rozhodnutí provedených při vývoji, přiblížení modelů použitých při vývoji a programy, které byly použity kromě mého kódu. Hlavní záměr mého kódu je vytvořit model domu pouze z jeho půdorysu, výšky okrajových zdí a sklonu střechy. Celý kód je psaný v programovacím jazyku Python, který jsem zvolil pro jeho jednoduchost a protože jsem s ním už měl zkušenosti. Kód jsem vyvíjel postupně. Když kód vytvořil správně model jednoho domu, tak jsem přešel na další model domu, který byl složitější a měl testovat jiné vlastnosti kódu.

Použité programy

Během mé práce jsem používal 3 programy, které mají každý svoji velmi specifickou funkci. Nejdříve vytvořím půdorys či model domu v programu OpenSCAD, ve kterém používám velmi limitovanou část jeho funkcí pro simplifikaci mého programu. Další program je Pycharm, který není nutný k zapnutí Pythonu, ale velmi usnadňuje programování. Poté, co je vytvořen finální model domu, je nutné ho převést z formátu .scad do .stl formátu, aby bylo možné otevřít finální model ve sliceru. K tomu je znovu použit OpenSCAD, který dokáže exportovat ve formátu .stl. Nakonec otevřu soubor ve formátu .stl v PrusaSliceru, rozhodnu se, jaká nastavení chci aplikovat a vyexportuji soubor ve formátu gcode. Tento postup vyžaduje relativně hodně kroků. Přesto si myslím, že je rychlejší než manuální modelování domů pro valnou většinu modelů.

OpenSCAD

První program, který je potřeba pochopit, je OpenSCAD. Tento program je určený přímo k 3D tisku, ale přesto je rozdílný oproti všem ostatním programům na modelování. OpenSCAD totiž místo toho, aby jako u ostatních programů měla velkou část moci myš, dává veškeré pravomoci měnit model klávesnici. To ve výsledku zanmená, že jediné, co myš ovládá, je pozice a orientace kamery. Vše ostatní je ovládáno klávesnicí a 3D objekty se nemodelují, ale píšou. Tím dává OpenSCAD najevo svoje zaměření na CAD (computer aided design, design s pomocí počítače) aspekty 3D modelování místo vizuálních aspektů, čili je vhodný pro design například mechanických součástek a celkově pro design modelů určených k 3D tisku, ale nehodí se například na animaci filmu.

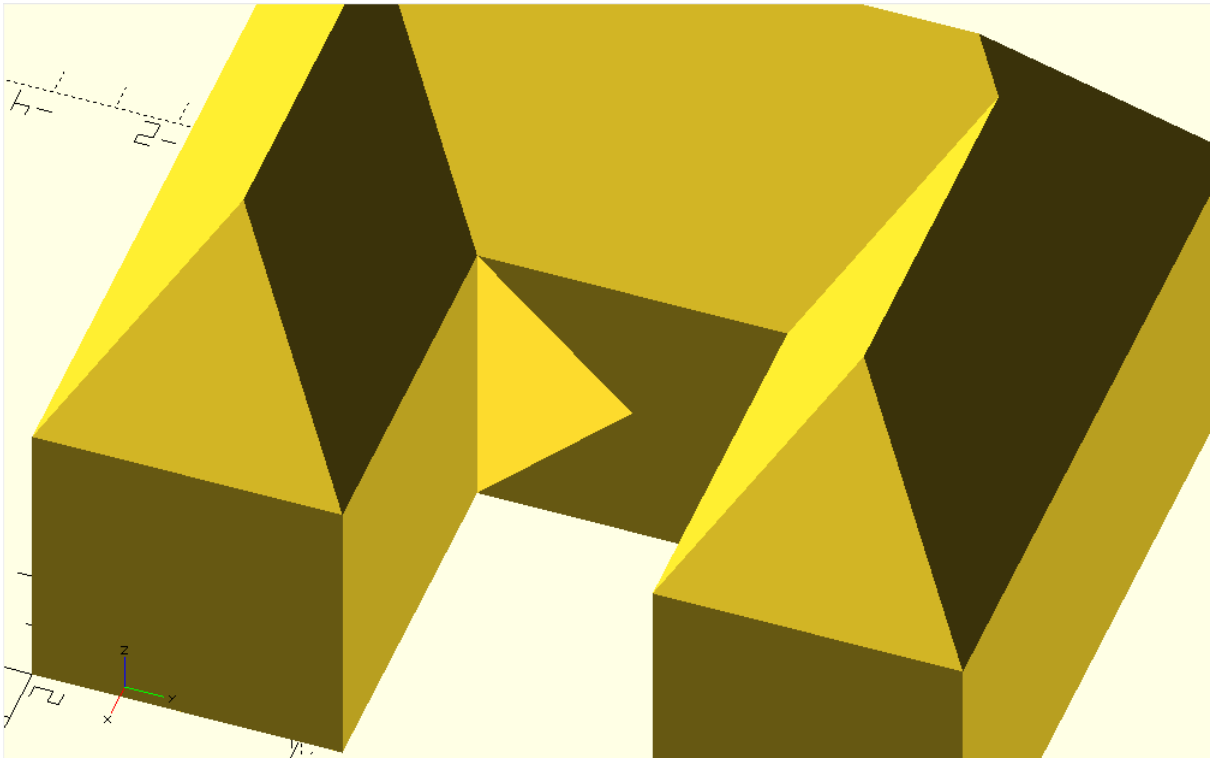
Tento přístup mě zaujal, už když jsem s OpenSCADem pracoval poprvé, ale rozhodující faktor byla funkce **polyhedron**.

Teoretický zápis této funkce vypadá takto:

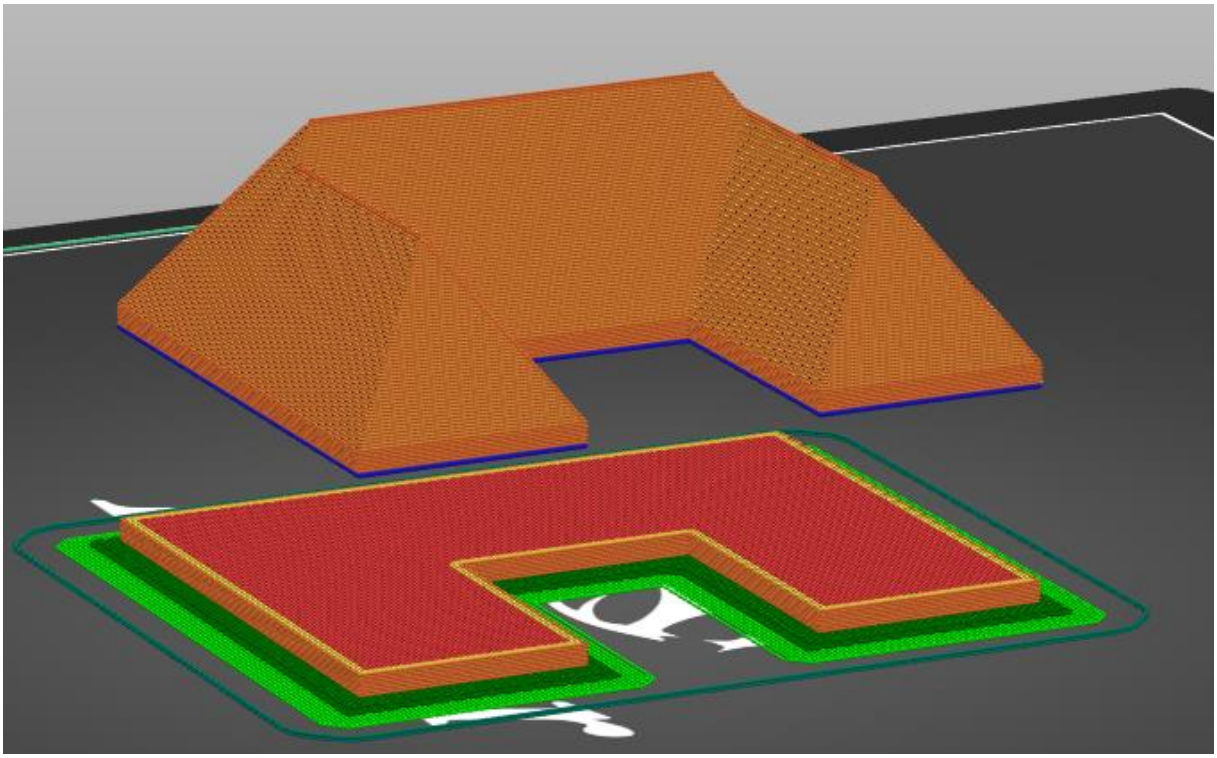
$$Polyedron(points=[\ (X_0, Y_0, Z_0), (X_1, Y_1, Z_1), \ldots], faces=[\ (P_0, P_1, P_2, P_3, \ldots), \ldots\ ]$$

Kde **points** je seznam bodů, kde každý bod je seznamem svých souřadnic X_n, Y_n, Z_n. **Faces** je seznam stěn, kde každá stěna je seznam indexů bodů o minimální délce 3 v **points** ve formátu, P_1, P_2, P_3, …, P_n. **Faces** začíná s číslováním bodů standartně od 0 a vytváří trojúhelníky spojující body P_a, P_{a+1}, P_{a+2}, přičemž pokud nejsou všechny body stěny v jedné rovině, tak se stěna dle nutnosti zlomí v trojúhelníky. Tato funkce nevytváří veškeré trojúhelníky, ale je za ní velmi komplexní systém, který zajišťuje správnost stěn. Tento systém není příliš intuitivní, ale právě to, že tento program používá přesné souřadnice, které

člověk (nebo program) do této funkce zadá, dělá tuto funkci ideální pro můj program. Tato funkce má samozřejmě i své nevýhody, které leží právě v té části, která generuje stěny. Ta je velmi háklivá na pořadí, ve kterém jsou body napsány a pokud jsou body zadány špatně, tak to bude mít efekt na celkový model a výsledný efekt je naprosto nepředvídatelný, jak je vidět na obrázku.

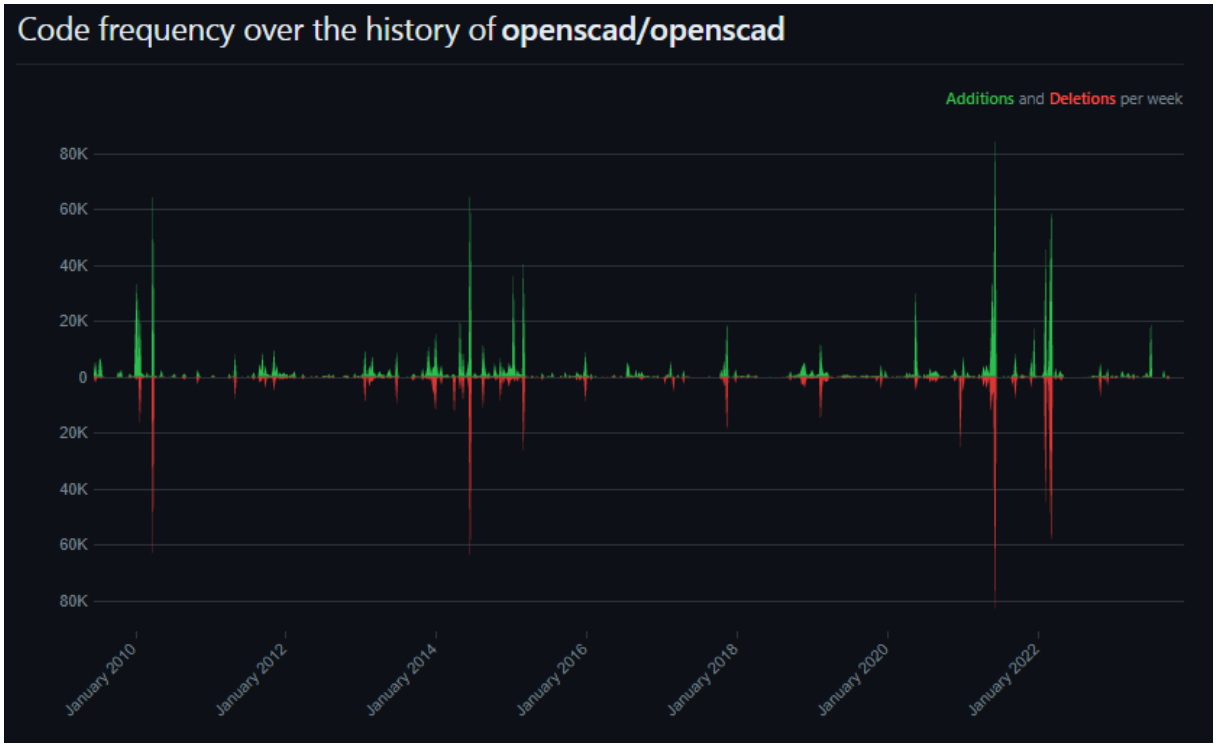


Obr. 1: Malá chyba způsobená prohozením pořadí dvou bodů



Obr. 2: Zde je model, který by byl vytisknutý s chybou

Druhá přednost, na kterou jsem přišel až při práci, je že soubory ve formátu .scad jsou jednoduché na otevření pomocí pythonu. Bez jakýchkoliv knihoven je možné v pythonu tento typ souboru otevřít. Náročnost je stejná jako otevření jakéhokoliv textového souboru v Pythonu. Na druhou stranu jsem nenašel jedinou knihovnu, která by dokázala extrahovat body přímo z souboru formátu .scad do pythonu, takže je nutné celý soubor načíst jako string a najít v něm veškeré hodnoty, které se poté musí přeložit do čísel.
OpenSCAD samozřejmě není definován pouze touto jedinou funkcí, je to velmi komplexní program a má mnoho různých funkcí. Není příliš populární, nejspíš v důsledku jeho velmi spartánského vzhledu a nutnost psát vše klávesnicí namísto klasických myší ovládaných funkcí, které jsou normou u většiny programů na modelování vyúsťuje v jeho vzhled nepřipomínající program na modelování [2].



Obr. 3: Zde můžete vidět počet přidaných/odebraných řádků kódu z OpenSCADu

Další distinktivní vlastností OpenSCADu je, že je plně open-source, což je plus ve světě dominovaném komerečními softwary. Je možné stáhnou si libovolnou minulou verzi, díky tomu, že repozitář OpenSCADu je volně k nalezení na internetové stránce www.github.com/openscad/openscad. Zde je také možné si zobrazit, kolik řádků kódu bylo přidáno a odebráno a z toho přibližně odhadnout popularitu OpenSCADu v různých časových obdobích, jak je vidět na obrázku 3. Můžete si všimnout mezer mezi jakoukoliv změnou, které se často pohybují v řádech měsíců. Také jsou velmi silně vidět 3 období, kdy probíhaly největší změny v kódu. Tyto změny jsou ale také doprovázeny mazáním o téměř stejné nebo stejné velikosti, takže si myslím, že se jednalo o celkové přepsání kódu nebo podstatnou optimalizaci. Ale žádné změny kódu z té doby nenaznačují celkovou optimalizaci, například jsem našel uživatele, který dva dny po sobě nahrál dvě změny, kde každá přidala přibližně deset tisíc řádků kódu a odebrala přibližně dva a půl tisíce řádků kódu. Když jsem si ale dostatečně prohlédl tyto změny, tak jsem zjistil, že to byly pouze změny ikon v programu, kde se každá ikona počítala za více než sto řádků kódu. Většinou se hodnota ikony pohybovala okolo dvou set řádků, ale některé se dostaly i přes pět set řádků. Jediná změna v kódu bylo přepsání umístění ikon[3]. Myslím si, že tímto se dají vysvětlit všechny tyto obrovské vlny,

jediná vlna u které bych udělal výjimku je ta, která se stala kolem konce roku 2009. Můj důvod pro toto tvrzení je zaprvé, se všemi ostatními existuje i stejně velká vlna mazání, s touto je přibližně poloviční a zadruhé, v tu doby byl tento program čerstvě vytvořen, takže bylo možné do něj rapidně přidávat nové funkce, které stále ještě neměl. Celkově je o tomto programu mnohem více informací než o komerčních programech, navíc člověk má možnost ten program libovolně modifikovat, pokud na to má dostatečné znalosti.

Pycharm

Pro psaní kódu jsem používal program Pycharm, který byl užitečným nástrojem ve veškerých částech vývoje kódu. Navíc je vyvíjen firmou původem z České Republiky, kterou je dle mého názoru lepší podporovat než megakorporaci typu Microsoft a Visual studio. Velmi subjektivní výhoda Pycharmu je, že je intuitivnější než jiná vývojová prostředí se kterými jsem pracoval. Dříve jsem již pracoval ve Visual Studiu, ale Pycharm lépe využívá barvy pro orientaci a vše, co není důležité, má pro oko velmi nezajímavou barvu. Velmi promyšlený je způsob implementace Gitu do prostředí. Pycharm ve spolupráci s Gitem zvýrazňuje změněné, vymazané a přidané řádky a podporuje vrácení zpět pouze jedné sekce kódu z poslední verze.

Prusa Slicer

Poslední program, který byl použit při vývoji je Prusa slicer, který je používán až při úplném konci. Otevře se v něm soubor ve formátu .stl. Poté jsou zvolena nastavení pro tisk, většina modelů byla ve sliceru zvětšena na 400%, pro zlepšení vizuální stránky výstupu. Je ale možné vytvářet větší půdorysy, které v závěru vytvoří model ve stejné velikosti, například pokud je nutné zachovat určité měřítko. Prusa slicer je stejně jako Pycharm český software, který byl v této práci použit jelikož hledá efektivněji

Kód

Jak již bylo řečeno, všechen kód je napsán v Pythonu 3.9. Tento programovací jazyk byl použit jelikož jsem s ním již měl zkušenosti a tato práce neměla za úkol, abych se naučil nový programovací jazyk. Kromě toho také nebyl cílem naučit se vytvářet grafy v Pythonu, které jsem používal k vizuální kontrole, který je uvedený v seznamu použité literatury. [4]. K procesu vytváření grafů jsem použil cizí kód, ale tato část je zaprvé použitá pouze pro zjednodušení kontroly, protože je jednodušší poznat, že je úsečka na špatném místě pomocí obrázku než pomocí číselných souřadnic a zadruhé nikdy nevykonává jakoukoliv logickou operaci, například nalezení průniky dvou přímek. Nevidím tedy použití cizího kódu jako znehodnocení své práce.

Při psaní kódu jsem dodržoval zásady PEP8, aby bylo co nejjednodušší pro vnějšího pozorovatele porozumět libovolné funkci. Nemohl jsem naneštěstí tyto zásady dodržovat vždy, hlavně při vytváření jmen proměnných. Ve složitějších funkcích je s relativně vysokou frekvencí pracováno s několika body a nebo úsečkami, které sdílí velkou řadu parametrů, včetně definičních. Ve výsledku jsem musel často používat pouze čísla pro jejich rozlišení, takže při případném náhledu na kód je potřeba věnovat zvláštní pozornost koncům proměnných.

V celém kódu je využito minimum knihovem, kvůli velmi specifickým úkonům, které jsou prováděny. Celkem byly použité pouze 3 knihovny, numpy, matplotlib a math. z toho jsou numpy a matplotlib použity pouze pro vizuální kontrulu, takže počet knihoven, které doopravdy figurují v kódu, se sníží na jednu. Knihovnu math jsem použil pro klasické matematické operace, například funkci tangens, která je použita pro nalezení výšky bodů střechy.

Celkově by se kód dal rozdělit na 3 zásadní části, kde první a poslední jsou relativně jednoduché a slouží pouze k převodu z openSCAD souboru do Pythonu a naopak. Prostřední

část je nejzásadnější délkou i složitostí, protože tato část zpracovává všechny body a vytváří nové body a spoje mezi nimi.

Globální části kódu

Některé části kódu prolínají mezi více než jednou částí, většina z nich tím způsobem, že jsou vytvořeny v první části a poté jsou používány v druhé části. Ty a zároveň různé globální konstanty bych chtěl představit v této části a zdůvodnit můj postup při jejich vytváření.
Celkem jsem použil jen jednu třídu, a to pro úsečky. Rozhodl jsem se nepoužít třídu pro body, protože bod sám o sobě nese relativně málo informací vyjma souřadnic X, Y a Z. Zvažoval jsem vytvoření speciální třídy pro body, jenže jsem nenašel více než 3 aplikace pro vlastní třídu. Na druhou stranu třída úsečky v sobě nese velké množství informací. Nejdříve se do ní zadají 2 body, které jsou koncovými body přímky. Všechny body jsou zadávány jako seznam souřadnic, protože většina zadaných bodů se nikdy znovu nevyskytne, takže nelze použít pointer na index seznamu všech bodů. Poté je nutné zadat "počáteční" bod. Tento bod je některý z bodů, které již byly pevně určeny a (po případném upravení z souřadnice) budou v konečném výstupu. Díky tomuto bodu je jednodušší zkracovat úsečku, aby spojovala dva body, kterými prochází. Také slouží pro hledání úseček, které prochází určitým bodem. Dalším argumentem je zda-li je úsečka okrajová zeď nebo ne. Předposlední argument určuje, zda-li je úsečka finalizovaná. Pokud úsečka není okrajová, tak je finalizovaná pokud oba dva její okrajové body jsou průsečíky přímek. Pokud je úsečka okrajová, pak je finalizovaná, pokud se dá přejít z jednoho jejího krajního bodu na druhý jen pomocí neokrajových finalizovaných úseček. Poslední argumet je seznam walls. Pokud je úsečka okrajovou zdí, tak jsou zde uložena dvě identická čísla, která odpovídají pořadí zdi. v opačném případě jsou v seznamu uložena dvě čísla, která odpovídají dvěma okrajovým zdem. Pokud z těchto zdí vytvoříme přímky a najdeme jejich průnik, poté pomocí průniku najdeme osu úhlu dvou okrajových zdí, tak získáme přímku, na které leží úsečka.
Dále jsou použity tři globální konstanty v pravém slova smyslu. Nejdůležitější z nich je konstanta ROUNDING_ERROR, jejíž hodnota je mnou zvolená. Její funkce je chránit čísla před možnou změnou způsobenou převody mezi různými číselnými soustavami. Její důležitost ale není tolik v její hodnotě, jako v její prosté existenci. Důvodem tohoto jevu je, že pokud dojde ke změně při převodech, tak se řádově pohybuje přibližně kolem 10^{-7}, takže jakékoliv číslo, které je řádově pod 10^{-2} může nahradit tuto konstantu. Zbylé dvě konstanty jsou drawing a printing, které určují, zda se má po určení všech bodů zobrazit vizuální reprezentace s pohledem zvrchu a zda se má konečný soubor také vypsat do konzole programu.
V kódu jsou použity dvě "konstanty", které ale mohou být změněny, pokud uživatel zapíše správné příkazy do zdrojového souboru. První z nich je roof_height. Tu je možné změnit, pokud je zadán pouze půdorys domu. v tomto případě je možné změnit její hodnotu, tím, že se v zdrojovém souboru změní její hodnota. Například změna na hodnotu 10 by vypadala takto: roof_height = 10;. Jednotka této konstanty je milimetr a určuje vertikální vzdálenost mezi patou budovy a začátkem střechy. Záporné hodnoty mají stejný efekt jako kladné, jelikož kód najde vršek budovy. Druhá "konstanta" je angle, která udává úhel sklonu střechy. Tu je možné změnit podobným příkazem jako výšku, tedy roof_angle = 30;. Zadávanou jednotkou jsou stupně bez jakékoliv limitace hodnoty. Pokud je ovšem zadaná hodnota mimo interval (0°, 90°) tak je možné získat nečekané výsledky. Pokud úhel náleží do intervalu (0°, -90°), tak kód vytvoří model, ze kterého byla střecha vyříznuta, nebo pokud je výška budovy dostatečně nízká, tak střecha skončí pod celou budovou a spodní podstava budovy je částečně odhalená. Některé modely, které podstoupí tuto proceduru, mají překvapivě mnoho podobností s konstrukcí stok. v neposlední řadě, pokud se tangens úhlu rovná nekonečnu, tak kód vytvoří mnohoúhelník s podstavou tvaru půdorysu domu, jehož výška se pohybuje řádově v 10^{13} až

10^{14}, což je efektivně nekonečně vysoké. v případě, že je úhel -90° nastane v podstatě stejná situace, s jediným rozdílem v tom, že podstava domu je nyní nekonečně nízko. Poslední příklad je, pokud je tangens úhlu roven nule. v tomto případu se namísto střechy vytvoří pouhá plocha. Zabýval jsem se pouze případy. které patří do intervalu [-90°, 90°]. s úhlem ale pracuje jedině funkce tangens, jejíž všechny hodnoty jsou obsáhlé v tomto intervalu, hodnoty mimo tento interval vyústí ve stejné výsledky.
V celém kódu se používá několik seznamů pro body. Nejdůležitější z nich je seznam Points, ve kterém jsou po většinu trvání kódu uloženy body, které nepatří do nejspodnější vrstvy domu. Na konec tohoto seznamu jsou postupně přidávány dokončené body střechy. Nové body střechy jsou uložené do seznamu roof_points. Tento seznam ukládá nově nalezené body střechy, které ještě nejsou dokončené. V seznamu jsou potom body použity v dalším zavolání funkce connect_roof_points. Po úspěšném nalezení dalších bodů nebo po nalezení všech bodů jsou již zaznamenané body přesunuty do seznamu Points, kde zůstanou netknuty do vypočítání Z souřadnice a uložení do .scad souboru.
Pro úsečky existuje pouze jeden seznam, který ovšem neobsahuje všechny úsečky použité při průběhu kódu. Do tohoto seznamu jsou ukládány pouze okrajové zdi a úsečky nalezené pomocí os úhlů. Na první pohled se zdá, že to jsou všechny úsečky, které jsou potřeba, ale já jsem při vývoji narazil na problémy, které se dají nejlépe vyřešit vytvářením dalších úseček. Například funkce, která rozhoduje, zda je bod v půdorysu nebo mimo. Ta funguje na principu vytvoření libovolné polopřímky, která končí v bodu, který zkoumáme. Pokud polopřímka protne lichý počet okrajových zdí, je bod v půdorysu a funkce vrátí hodnotu True. V opačném případě funkce vrátí False. Celá funkce je složitější, kvůli ošetření možných chyb nastávajících když polopřímka splývá s okrajovou zdí.
Poslední globální seznam je Faces. Téměř celou funkční část kódu obsahuje pouze stranu podstavy a okrajové zdi. Až jako jeden z posledních kroků před vytvořením konečného souboru jsou do seznamu přidány všechny strany střechy najednou. Rozhodl jsem se, že je jednodušší přidat všechny stěny najednou než přidávat každou stěnu během iterací funkce connect_roof_points. Hlavním důvodem bylo předejití možných duplicitních stěn. Ty by teoreticky neměly mít efekt v OpenSCADu, ale pokud je možné se vyvarovat takového případu, tak je lepší jiné řešení i za cenu lehce zvýšeného výpočetního času.

Čtení souboru

Tato část má za úkol přečíst soubor typu .openSCAD a pracovat s ním jako s proměnou typu string. Použití tohoto přístupu ale s sebou nese obrovské množství nevýhod. Například je nutné nejdříve definovat body budovy nebo půdorysu a až po nich definovat stěny. Hlavně jsou ale nevýhody spojené právě s funkcí polyhedron. Tato funkce dokáže mistrovským způsobem vytvářet stěny, které mohou být nespojité, například dům s uzavřeným dvorem. Bohužel způsob, jakým tato funkce funguje, není obecně znám. Kvůli tomu je můj program limitován pouze pro domy, jejichž půdorys je tvořen jedním spojitým mnohoúhelníkem.
Další limitace kódu, která je ale přidaná pro zjednodušení kódu, je pořadí bodů ve zdrojovém souboru. Pro správné fungování celého kódu je nutné body seřadit tak, aby při pohybu po okraji půdorysu po nebo proti směru hodinových ručiček člověk narazil na body ve stejném pořadí jako v jakém jsou zadány. Tuto limitaci jsem zvolil, aby bylo možné vytvářet úsečky jen jako spojnice dvou po sobě následujících bodů.
Podstatná část této části kódu spočívá pouze v hledání hodnot bodů a stran ve zdrojovém souboru. Všechny souřadnice bodů jsou konvertovány do typu hodnoty float, pro zajištění, aby nedošlo k případné ztrátě desetinných míst. Celá hledání je ale pouhá operace s proměnou typu string, pro porozumění kódu málo důležitá.
V této části jsou ale vytvořené seznamy všech bodů, stran a úseček, které bych nyní trochu rozvedl. Body jsou uloženy v seznamu Points, ze kterého jsou ale odebrány všechny body

kromě těch, které mají nejvyšší souřadnici Z. Zbytek bodů je uložen v jiném seznamu a tyto dva seznamy jsou sloučeny až po určení všech bodů střechy. Do seznamu Points jsou v průběhu programu přidávány nové body, které určují tvar střechy. Strany jsou také uloženy v prostém seznamu, který je používám mnohokrát během průběhu programu.
Strany jsou přečteny ze zdrojového souboru. Pokud je ve zdrojovém souboru uložen model domu beze střechy, je ze stran odebrána ta strana, která obsahuje všechny nejvyšší body. Pokud je zadán pouze půdorys a jsou vytvořeny nové body, tak jsou vytvořeny i nové vertikální stěny. Ty jsou vytvořeny nalezením indexů spodních dvou bodů, poté je ke každému indexu přičten celkový počet bodů v podstavě a nakonec jsou indexy a přičtené indexy seřazeny, aby vytvořily fungující stěnu.

Tvorba domu

Tato část je nejobsáhlejší jak do složitosti tak i délky kódu. Zároveň to je zdaleka nejdůležitější část celého kódu. z toho důvodu nebude vysvětlen naprosto všechen kód, hlavně některé části, které jsou velmi intuitivní budou přeskočeny. Také už nebudou vysvětlovány konstanty a proměnné, které pro funkci kódu nejsou příliš důležité či jsou relativně intuitivně pojmenovány.
Celá tato část začíná funkcí setup_angles. Tato funkce získá jako argument seznam Points. Její výstup je vytvoření os úhlů dvou krajních zdí, které mají bod ze seznamu jako jeden ze svých krajních bodů. Osu úhlu jsem se nejdříve snažil vytvořit pomocí trigonometrických funkcí, jenže tento proces měl řadu chyb. Hlavní z nich byla možnost tupého úhlu vytvářejícího stejnou hodnotu jako ostrý úhel. Bylo zde samozřejmě očividné řešení pomocí mnoha logických funkcí, zvolil jsem ale řešení, které je dle mého názoru mnohem elegantnější. Bod, kterým má procházet osa, je zvolen jako nový počátek. Hodnoty souřadnic druhých bodů úseček jsou poté převedeny relativně k novému počátku. Pokud by délky dvou úseček nebyly stejné, jsou relativní souřadnice vynásobeny poměrem obou úseček. První krajní bod je vytvořen sečtením obou relativních X souřadnic a obou relativních Y souřadnic, u druhého jsou oba součty vynásobeny mínus jedničkou. Poté jsou k oběma přičteny souřadnice nového počátku, který je zároveň určen jako začátek úsečky. Tím vznikne osa úhlu, která protíná vnitřní i vnější úhel.
S těmito osami dále pracuje funkce find_lines. Tato funkce najde všechny osy úhlů a použije je k nalezení všech ostatních os, které ji protínají. Vyloučí při tom všechny osy, které se protínají a zároveň mezi svým začátkem a průsečíkem porovnávané osy protínají krajní zeď nebo jejichž průsečík není v půdorysu. z možných průsečíků najde ty osy, které vytvoří průsečík nejblíže k počátku. v případě, že je průsečík nejbližší k počátku obou os, tak jsou osy brané jako dokončené, jejich první okrajový bod je určen jako začátek každé z os a druhým bodem je jejich průsečík. Tento průsečík je přidán do seznamů Points a roof_points. Seznam roof_points je určený pro již nalezené body střechy, které ovšem nemají všechny úsečky vycházející z nich. Tento proces samozřejmě také funguje pro případy, kde mají tři a více os stejný průsečík, pouze se zvýší počet změněných os.
S body seznamu roof_points pracuje funkce connect_roof_points, která je v celém programu s velkým náskokem nejkomplexnější. Je to rekurzivní funkce, která vytváří nové hrany střechy. Zjednodušené vysvětlení způsobu fungování je, že při každém svém opakování tato funkce posune body střechy blíže k finalizaci. Toto většinou probíhá ze dvou stran zároveň, takže výsledný efekt lehce připomíná Shaker sort.
První část této funkce je dedikovaná pro její případné ukončení. v případě, že zbývá spojit pouze dva body, které by byly spojeny hranou střechy. Pokud dojde k této situaci, jsou oba body okamžitě spojeny a celá funkce je ukončena.
V normálním případě je ovšem chod celé funkce mnohem složitější. v podstatě existují dvě části funkce, jedna pracuje s body a druhá s polopřímkami budoucích hran, které začínají

v bodech ze seznamu roof_points. Tyto polopřímky jsou vytvořené v první části této funkce. k jejich vytvoření je nejdříve zapotřebí získat indexy okrajových zdí, které budou definovat výslednou polopřímku. k nalezení indexů je prohledán seznam Lines, ze kterého jsou vybrány úsečky, které mají jako krajní bod právě hledaný bod střechy. z těchto úseček jsou zjištěny indexy krajních zdí těchto úseček. Nehledě na počet úseček, které se protínají k vytvoření bodů, budou v indexech zdí právě dva indexy, které se neopakují. Poté je nalezen průnik obou okrajových zdí s těmito indexy.
Zde bych vložil vysvětlení funkce, která hledá průnik dvou úseček. Tato funkce nejdříve zkontroluje, zda obě funkce nejsou rovnoběžky. Pokud jsou, vrátí hodnotu False. v opačném případě vypočítá rychlost změny X a Y souřadnic, z nich vypočte průnik na ose Y a poté se k oběma úsečkám chová, jako by to byly lineární funkce. Pro ty poté jednoduše najde jejich průsečík. Tento přístup má ale jednu zvláštnost, která je občas výhodou občas nevýhodou. Jelikož se k úsečkám chová jako k soustavě dvou rovnic, tak je možné, že nalezený průnik nebude na jedné nebo na obou z úseček. To jest, místo protínání úseček ve skutečnosti tento program protíná přímky. Pro zjednodušení jsem to ovšem nazval protínání úseček.
Po nalezení průniku obou okrajových stěn je vytvořena osa úhlu mezi těmito dvěma stěnami. Ta prochází nedokončeným střešním bodem a je vybrán okrajový bod této osy. Vždy vybírán ten bod, který je od střešního bodu ve směru budoucí nové hrany střechy. To je ovšem nezbytně nutné kvůli mému způsobu pracování s protnutí přímek, tento systém je v programu pouze pro zjednodušení. Všechny polopřímky, které jsou vytvořeny v rámci jednoho volání funkce jsou uloženy do seznamu roof_walls, se kterým se pracuje v druhé části funkce po projití všech bodů v seznamu roof_points.
V druhé části funkce se nejdřív prohledají zdi všech polopřímek v seznamu roof_walls. Pokud mají libovolné dvě úsečky společné indexy okrajových zdí, jsou jejich průsečíky nalezeny jako první. Pokud mají společnou krajní stěnu, tak se musí protínat a pouze je prohledán seznam všech nedokončených stěn, které případně také sdílejí tuto stěnu.
Po vyřešení prioritních úseček jsou vyřešeny všechny ostatní v seznamu. Odfiltrovány jsou všechny, u kterých by se mezi průsečíkem sebe a právě aktivní úsečky nacházela okrajová zeď a ty, jejichž průsečík by se nenacházel v podstavě budovy. z výsledných úseček jsou vybrány ty, jejichž průsečík je nejblíže k počátku střešní úsečky. Poté je jako jeden z krajních bodů u všech těchto úseček nastaven průsečík, tyto úsečky jsou označeny jako dokončené a jsou spolu s průsečíkem přidány na seznam namalovaných úseček a bodů pro případnou vizuální kontrolu.
Po nalezení všech úseček a střešních bodů je nutné vrátit všechny spodní body podstavy do seznamu bodů. Zároveň je nutné přidat strany do seznamu *Faces*, který v tuto chvíli obsahuje podstavu a okrajové stěny.
Pro úspěšné vytvoření stěny je nutné zapsat indexy všech jejích bodů ve stejném pořadí, v jakém budou spojeny pro vytvoření stěny. Nezáleží ovšem na směru, tedy je možné body zapsat po i proti směru hodinových ručiček. K hledání správného pořadí bodů jsem použil funkci *recursion_maze*. Tato funkce, jak jméno napovídá, používá rekurzi pro hledání bodů tvořících stěnu. Tato funkce je volaná pro každou okrajovou stěnu. Hlavní argumenty této funkce jsou konečný bod, počáteční bod a seznam všech bodů, kterými už funkce prošla. Funkčností je podobná algoritmu A star. Pořadí budoucích bodů se totiž odvíjí od jejich vzdálenosti ke konečnému bodu. Takto je časová složitost velmi silně snížena. Hlavní důvod pro to je způsob konstrukce střech, jelikož při hledání dalšího bodu stěny bude další bod téměř vždy blíž ke konečnému bodu okrajové zdi.

Vytvoření nového souboru

Po vytvoření nových stran a uložení do seznamu zbývá poslední krok před vytvořením nového souboru. Pro všechny body střechy je vypočítaná jejich Z souřadnice. K tomu je

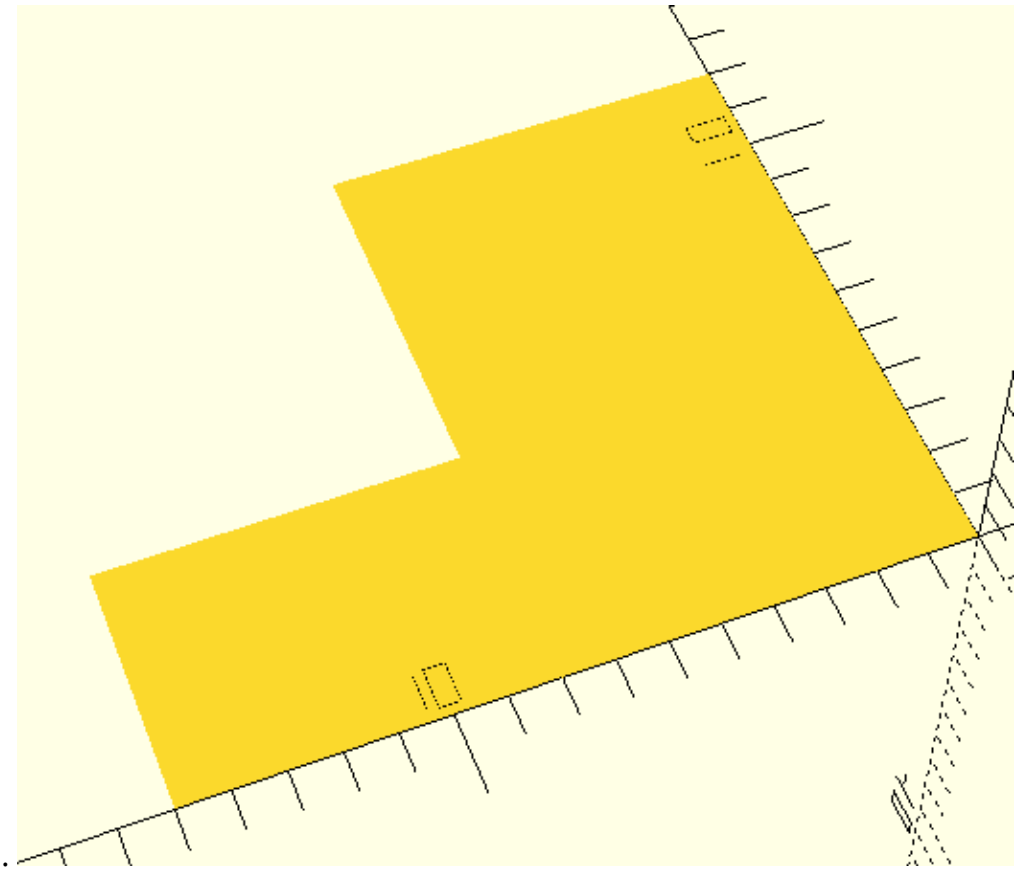
nalezena úsečka spojující tento bod a okrajovou zeď. Pomocí trigonometrických funkcí je nalezena správná výška bodu střechy a k té je přičtena výška okrajových zdí.
Když jsou veškeré body a stěny ve své finální podobě, tak z nich kód vytvoří string, který poté uloží do souboru formátu .scad. Jméno tohoto souboru je house.scad a stačí již jen otevřít soubor v OpenSCADu, převést ho na soubor formátu .stl a poté ve sliceru na soubor formátu .gcode, ze kterého dokáže 3D tiskárna vytisknout model domu.

Testovací modely

V této sekci bych okomentoval modely domů, které byly použité při vývoji kódu a jak mi pomohly vylepšit kód.
Modely jsem zadával jako půdorysy i jako modely bez střechy, aby byly vyzkoušeny obě metody vstupních souborů. Celkově byly modely vytvářeny tak, aby výsledné domy byly v rámci možností realistické. Hlavně jsem se soustředil, aby domy byly tvořeny z pravých úhlů a složitost modelu jsem zvyšoval přidáváním úhlů. Domy s půdorysem který obsahuje jiné než pravé úhly jsem taky otestoval, ale upustil jsem od používání těchto půdorysů. Můj způsob hledání jejich osy fungoval, ale hlavní důvod pro nepoužívání těchto půdorysů je vizuální kontrola. U pravého úhlu je jednoduché vizuálně potvrdit správnost osy, ale u některých velmi ostrých a velmi tupých úhlů je tato kontrola těžší.

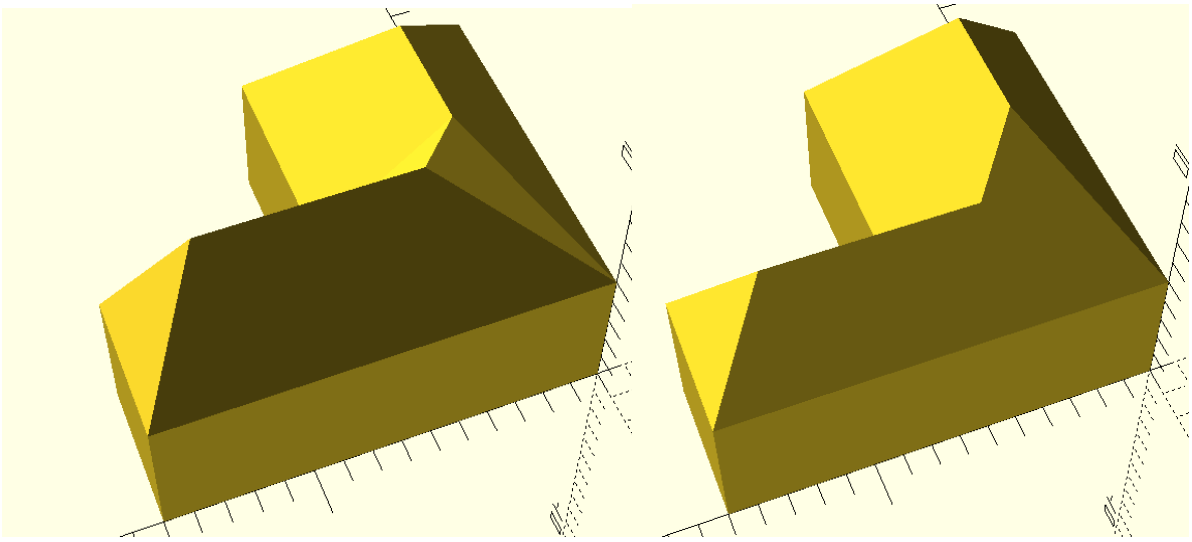
První model - Jednoduchý dům

Tento model jsem používal pro prvotní testování. Vytvořil jsem dům s takovýmto půdorysem, jelikož pouhý čtyřúhelník by nemusel spustit všechny chyby v kódu. Na začátku jsem ovšem chybně předpokládal, že všechny body střechy mají být v jedné rovině, jelikož jsem před touto prací řešil střechy pouze teoreticky při pohledu z vrchu. To způsobilo že střecha tohoto domu byla tvořena z mnohem více trojúhelníků než bylo nezbytně nutné



Obr. 4: Půdorys Jednoduchého domu, jak je vidět v programu OpenSCAD

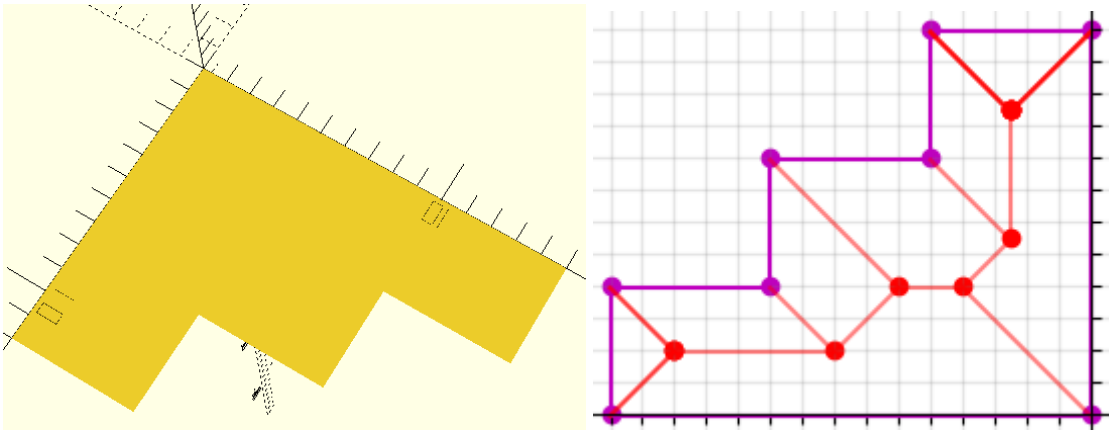
Půdorys Jednoduchého domu, jak je vidět v programu OpenSCAD



Obr. 5: Původní model s hřebenem střechy v jedné rovině a opravený model

Druhý model - Dům ve tvaru schodů

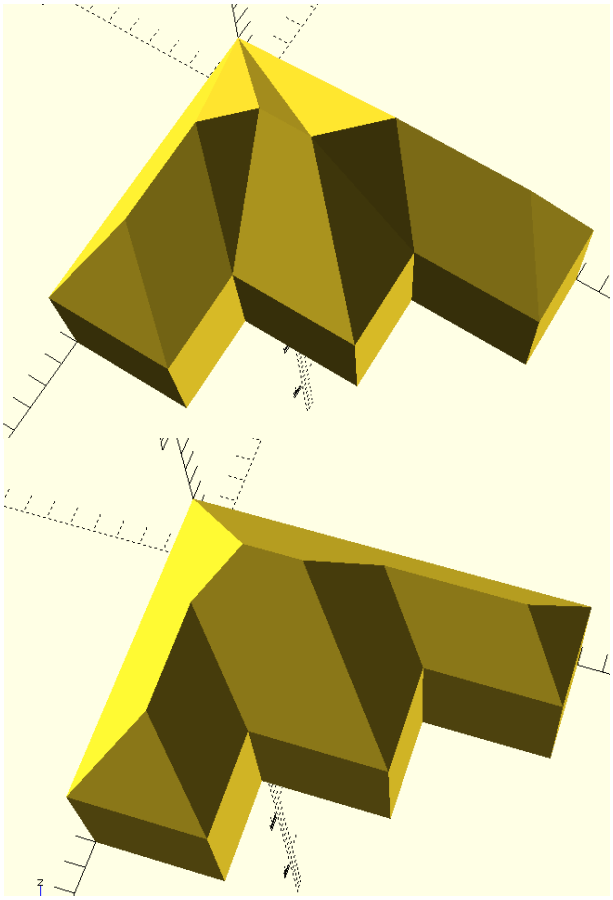
Druhý model se v mnohém podobá prvnímu, ale stále byl cenným přínosem do vývoje. Hlavní charakteristikou modelu, která pomohla s testováním je velký počet os úhlů které se navzájem protínají. To umožnilo vývoj a testování funkcí, které kontrolují pozici průsečíků a zdali jsou obě osy od počátku do průsečíku v půdorysu. Dále složitější půdorys poukázal na některé méně optimální řešení v kódu. Například před použitím tohoto půdorysu jsem nacházel okrajové stěny, které se mají protnout tím, že jsem z bodu na střeše hledal okrajové zdi pomocí spojnic. Při práci s tímto modelem jsem si uvědomil, jak neefektivní toto je. proto jsem si všechny okrajové zdi označil čísly, každou osu úhlu jako čísla dvou okrajových zdí, kterými je vytvořena. Kód při hledání okrajových zdí pro bod nalezne okrajové zdi, jejichž osy jsou všechny spojnice končící v bodu. Z podstaty střechy se budou přesně dvě čísla označující okrajové zdi vyskytovat přesně jednou. Tyto okrajové zdi jsou poté použity pro hledání další osy úhlu.



Obr. 6: Půdorys domu ve tvaru schodů v programu OpenSCAD a při vizuální kontrole

Tato úprava kódu byla v tu chvíli relativně malá optimalizace. Při práci s většími modely je ale tato optimalizace znatelná. Časová složitost optimalizace je $O(n)$, časová složitost před optimalizací je silně závislá na modelu, ale řádově se pohybuje v $O(n^3)$ nebo vyšším. Moje optimalizace má samozřejmě i nevýhody. Největší nevýhoda je zvýšení náročnosti na paměť,

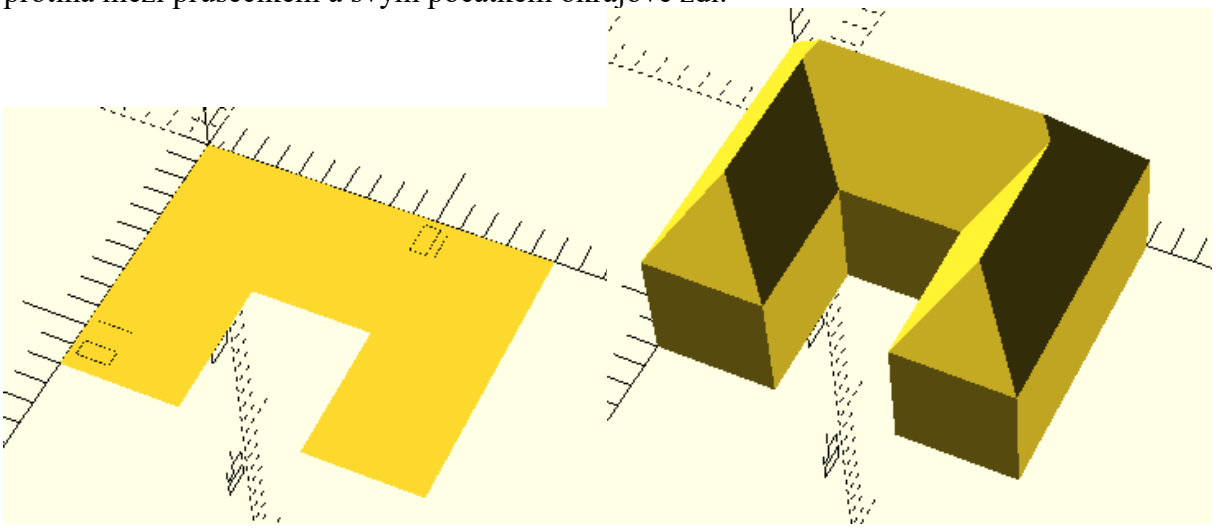
jelikož všechny úsečky musí mít připojený seznam s okrajovými zdmi, ze kterých byly vytvořeny.



Obr. 7: Původní model s hřebenem střechy v jedné rovině a opravený model

Třetí model - Dům ve tvaru U

Další model byl zaměřený na testování na velmi nekonvexním půdorysu domu. Tento model poukázal na přehlédnutí v kódu právě při situacích, které vyvstanou při silně nekonvexním modelu. Například vyvstávaly možnosti, kde se protnuly dvě osy úhlů mimo půdorys. V předchozích modelech tento případ nebyl možný, takže mě nenapadlo ho ošetřit. Další chyba bylo například protnutí přímek, kde průsečík vycházel v půdorysu, ale alespoň jedna z nich protíná mezi průsečíkem a svým počátkem okrajové zdi.

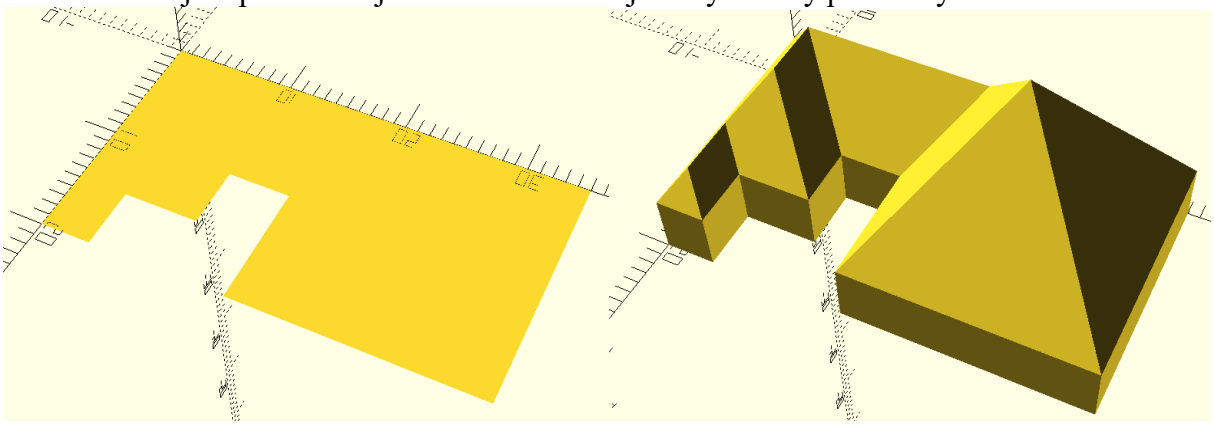


Obr. 8: Podstava a vyřešený model domu ve tvaru U

Postupně se složitějšími modely jsem také vylepšoval starší funkce, pokud se vynořily některé okrajové varianty. Také jsem stále lehce optimalizoval kód, například jsem několikrát přepsal funkci hledající první průsečíky os úhlů. Jednak jsem vylepšoval způsoby, kterými generuje funkce body. Ale také jsem udržoval tuto a ostatní funkce v obrazu s novými vývoji v kódu.

Čtvrtý model - Dům ve tvaru zrcadlového čísla 9

Tento model je ze všech použitých modelů nejsložitější. Kromě velkého množství různých os úhlů má dva důležité testovací prvky. Prvním z nich je testování, zda kód dokáže protnout 3 osy úhlů najednou a poté pokračovat se správnými okrajovými zdmi. Kvůli tomuto parametru bylo nutné přepsat některé části kódu, například funkcí hledající okrajové zdi pro bod a hlavně funkce, která hledá první body z prvotních os stěn. Pro podporu protnutí více os najednou jsem přidal seznam pro každou osu. Do toho jsou přidány všechny osy, které jsou protnuty předtím, než osa protne okrajovou stěnu. Poté jsou prohledány pro nalezení těch, které se navzájem protnou nejdříve a u těchto os jsou vytvořeny průsečíky.



Obr. 9:Podstava a vyřešený model domu ve tvaru zrcadlového čísla 9

Druhý testovací parametr byl průsečík, kde se budou stýkat dvě splývající osy úhlů. Tento případ mě inspiroval k lepšímu řešení hledání nových bodů. Uvědomil jsem si, že přímky, které mají společnou jednu okrajovou zeď, ze které byly vytvořené, se musí protnout. To mi umožnilo optimalizovat kód, jelikož jsem nemusel kontrolovat osy, které neměly žádnou společnou okrajovou zeď s právě hledanou osou. Zároveň jsem ale musel přidat možnost pro protnutí více os najednou. To jsem vyřešil tím, že všechny možné osy uložím do seznamu, ze kterého jsou použity osy s nejbližším průsečíkem.

Používané modely

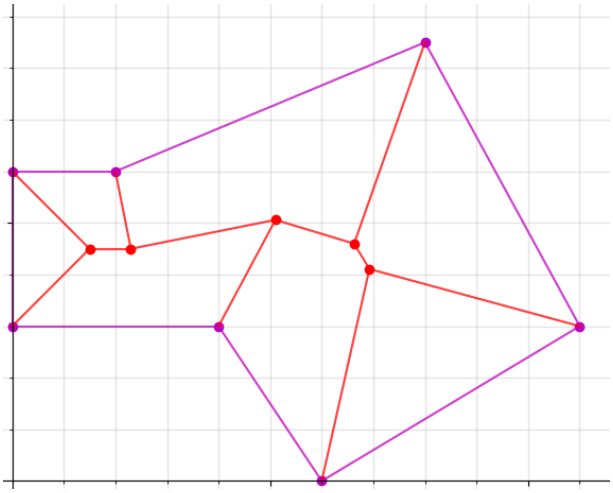
V této sekci budu dále rozebírat modely domů, které použil k dalšímu testování a vývoji kódu. Hlavní záměr použití těchto modelů bylo celkové otestování univerzálnosti kódu. Tyto modely tedy mají své unikátní vlastnosti, ovšem slouží pouze ke sledování robustnosti kódu namísto aktivního vývoje nových funkcí.

Samozřejmě můj kód není perfektní, takže i tyto modely odhalily chyby, které mě motivovaly k dalšímu vylepšení kódu. Jedna obzvláště zajímavá chyba nastávala ve funkci, která hledala průsečík dvou přímek. V případě, že první zadaná přímka byla rovnoběžná s osou y a druhá s ní byla různoběžná, nalezený průsečík byl na zdánlivě náhodném místě. Tato chyba byla způsobena tím, že funkce přiřadila průsečíku x souřadnici jednoho z bodů různoběžné přímky, místo přiřazení x souřadnice jednoho z bodů přímky rovnoběžné s osou y. Důvod vzniku této chyby bylo pojmenování vstupních čar stejně, pouze na konci jména byly obě přímky rozlišeny čísly 1 a 2.

Dále jsem testoval funkce, které by neměly mít vliv na hlavní část řešení střechy, tedy možnost zvolení výšky střechy nad podstavou a zvolení úhlu střechy. Ty jsem hlavně testoval pro různé okrajové případy, které by mohly narušit jejich funkci.

Dům bez pravých rohů

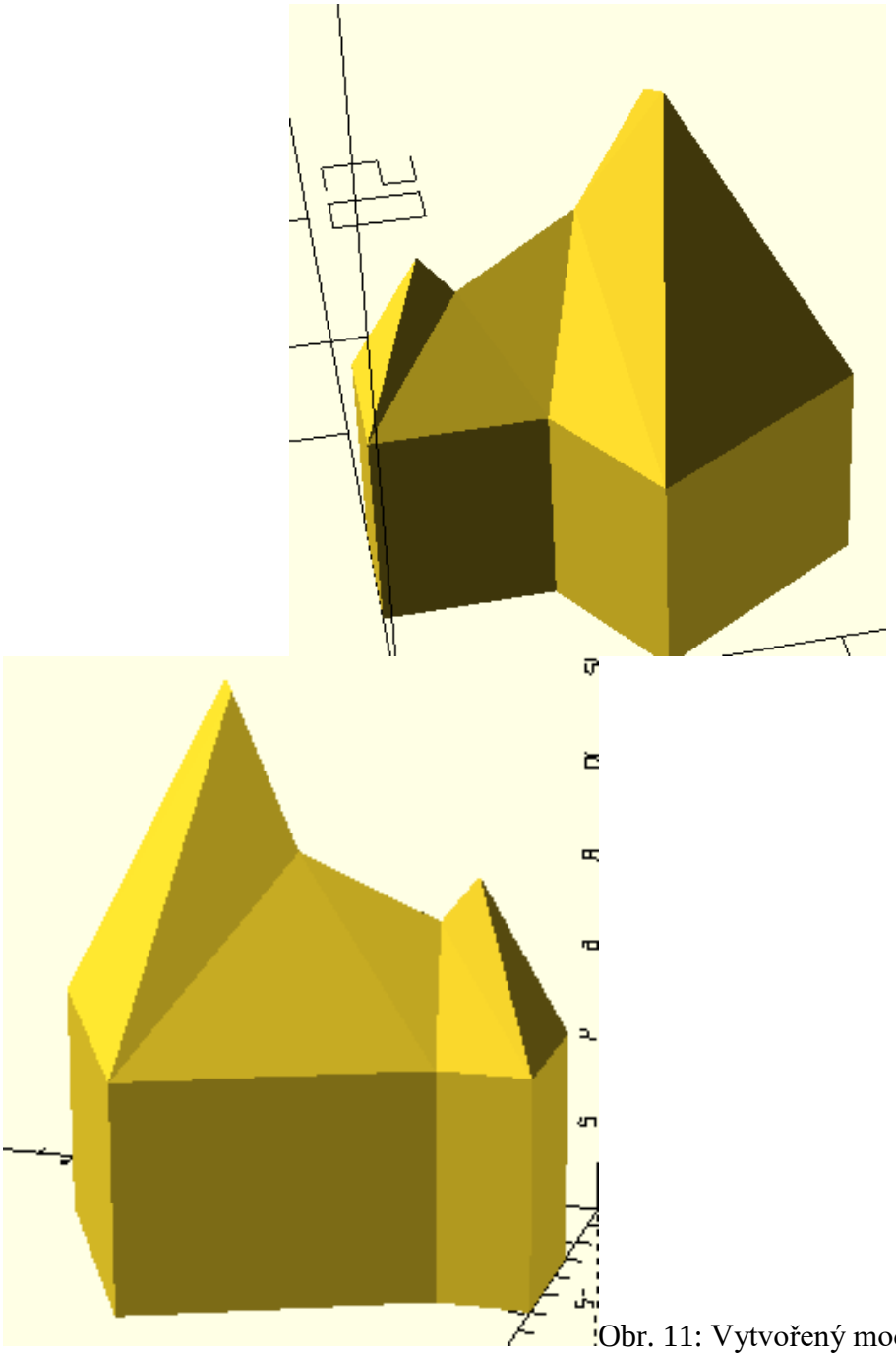
Tento dům jsem přidal do testování z očividných důvodů, všechny předchozí modely byly tvořené jen z pravých úhlů. Musel jsem tedy otestovat, zda můj kód dokáže vyřešit i dům bez pravých úhlů. Hlavní bod úrazu při řešení tohoto modelu byl jeho zvláštní povrch. Kód vyřešil půdorys správně, ale 3D model domu měl lomené části střechy, které se jinak vždy vyskytovaly v jedné rovině. Po přečtení jiných prací, které se také zabývají teoretickým řešením střech jsem ale zjistil, že střechy pro domy s lichoběžníkovým půdorysem musejí mít alespoň jeden zlom [1].



Obr. 10: Vyřešený půdorys domu bez pravých úhlů

Na tomto modelu jsem dále testoval jak funkci pro změnu úhlu střechy, tak i funkci pro změnu výšky střechy. Obě tyto funkce neměly problém s nestandardním půdorysem. Změna úhlu sklonu střechy jenom zvýraznila lomené plochy střechy.

Tento půdorys byl převzat z diplomové práce, která obsahovala půdorysy používané při výuce teoretického řešení střech [1]. Snažil jsem se používat půdorysy, které byly svým způsobem unikátní a dále posouvaly můj kód a tato práce byla skvělým zdrojem modelů. Původní model ale měl některé z úhlů pravé, ty byly mnou upraveny aby celý půdorys neobsahoval ani jeden pravý úhel.

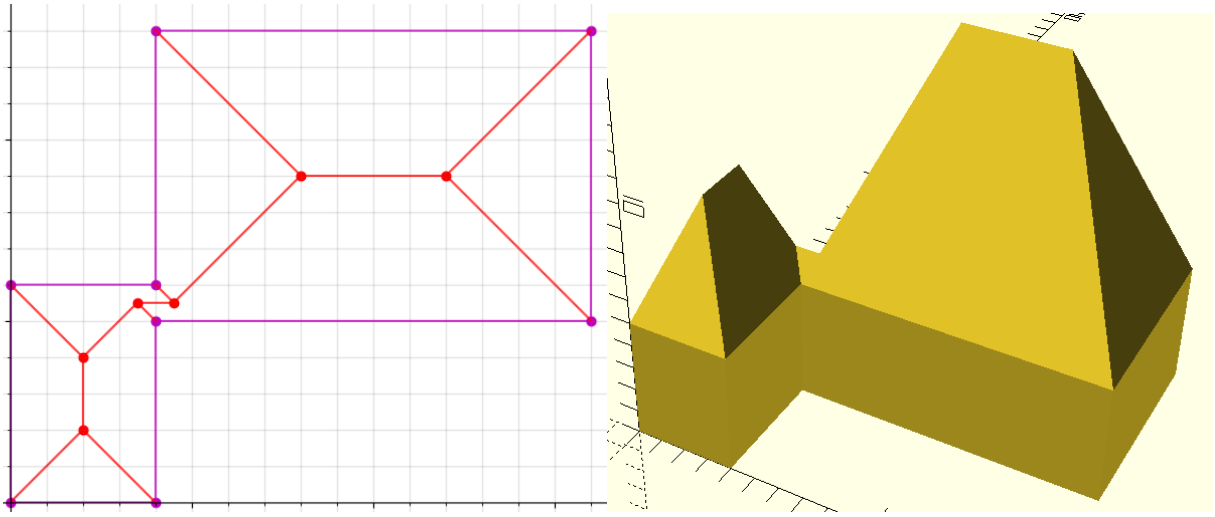


Obr. 11: Vytvořený model domu bez pravých rohů

Dům s úzkým spojem

Tento model měl za úkol otestovat, zda je možné řešení velmi úzkého spoje. Zároveň jsem přidal další testování vysokého sklonu střechy. To jsem sice již testoval, ale tento model měl sloužit jako ujištění bezchybnosti této funkce. Teoreticky by spoj neměl dělat řešení střechy žádný problém. To se také potvrdilo, celý model byl vyřešený bez nutnosti opravovat chyby v kódu.

Tento model se dále snaží klást kódu překážky, které často působí potíže studentům. V tomto trendu jsem dále pokračoval. Zdálo se mi, že právě v těchto složitějších modelech kód vyniká a poráží člověka jak v rychlosti tak i správnosti řešení, nemluvě o tvorbě modelu.



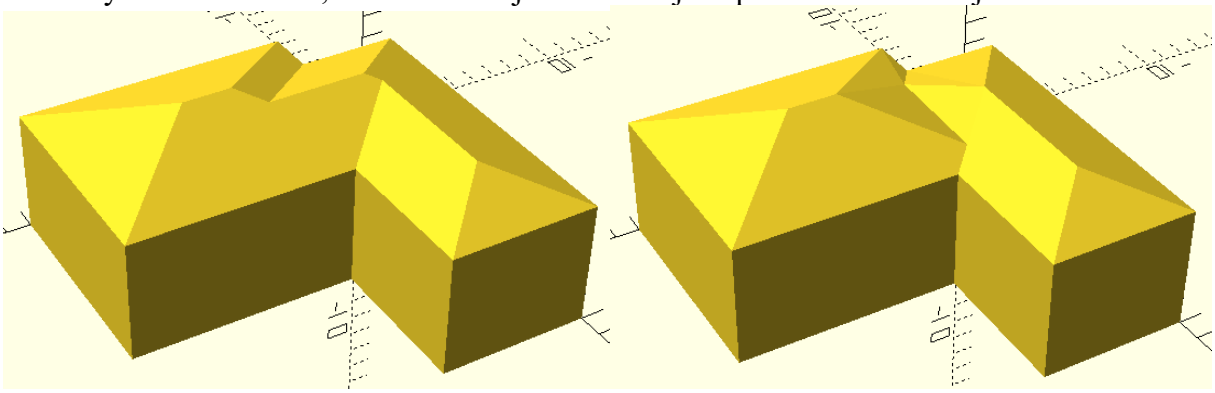
Obr. 12: Vyřešený půdorys a model domu s úzkým spojem

Dům s častým chybným řešením

Tento půdorys byl také převzat z [1], kde byl uveden jako typ půdorysu, kde studenti často chybují. Jejich řešení je chybné kvůli vytvoření žlabu, kde střecha nemá žádný sklon. Z tohoto důvodu se tam hromadí voda a může dojít k poškození střechy. Chtěl jsem tedy vidět, jak si můj kód poradí s půdorysem, se kterým bojuje mnoho studentů. Můj kód tento půdorys vyřešil bez větších problémů.

Po bližším zkoumání půdorysu jsem našel jeden z možných důvodů náročnosti půdorysu. Studenti si při řešení nepředstaví střechu ve třetím rozměru a řeší ji pouze jako čáry na papíru. Při takovém přístupu je velmi jednoduché protnout dvě osy úhlů ve špatném pořadí, což způsobí vytvoření žlabu. Na papíře to ovšem vypadá jen jako další úsečka, i když ve třetím rozměru je to očividně žlab.

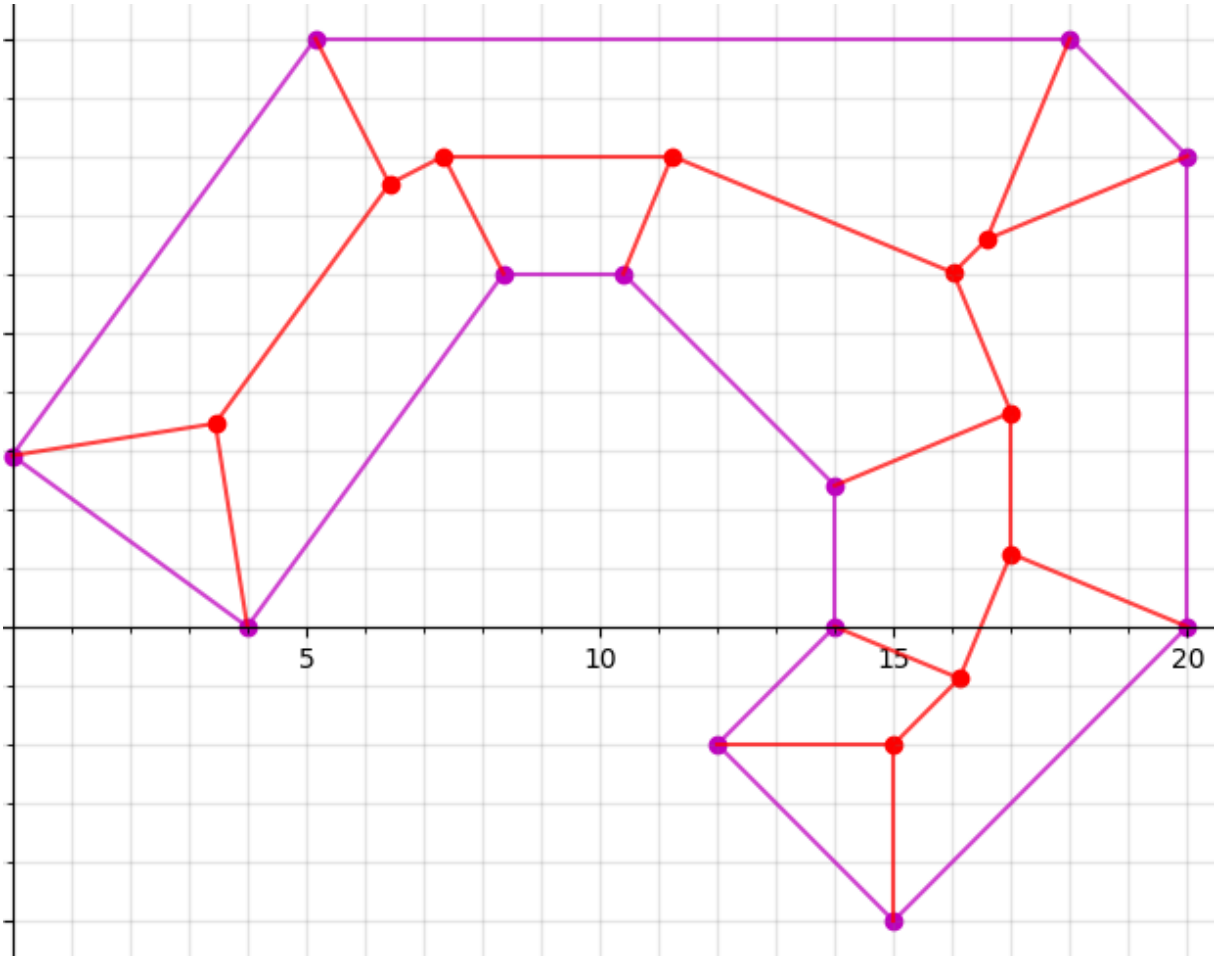
Pro porovnání se správným řešením jsem vytvořil i 3D model se žlabem. Ten jsem ovšem musel vytvořit manuálně, takže některé jeho části nejsou plně v rovině či nejsou rovnoběžné.



Obr. 13: Správně vyřešený model a model se žlabem

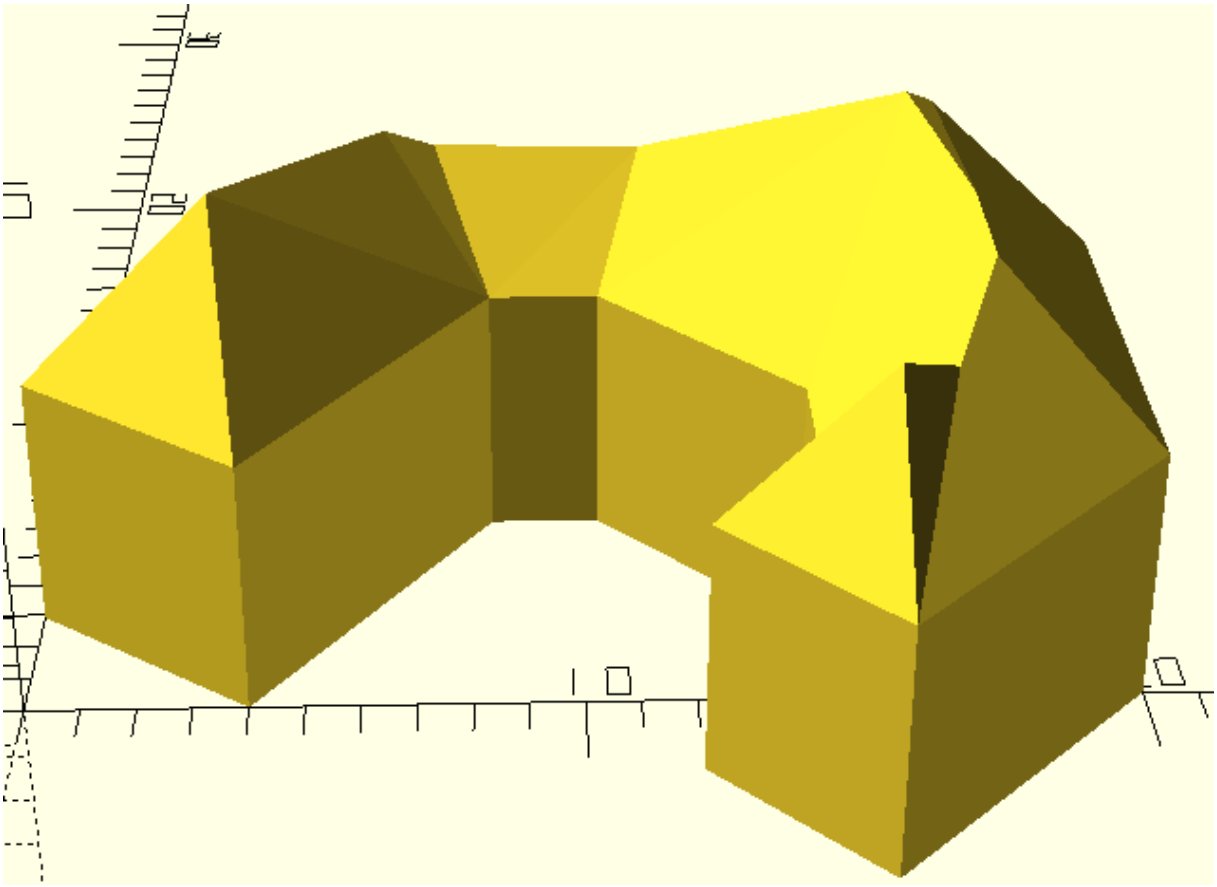
Dům ve tvaru C

Tento model je také převzatý z [1]. Zaujal mě pro svůj počet úhlů a kvůli novince v ohledu vytvoření prvních os úhlů. Po prvním vytvoření jsou zde přítomné 3 průsečíky místo obvyklých dvou. Tento půdorys byl ovšem velmi náročný na plné dokončení, jelikož na konci se můj kód dostane do slepé uličky, kde musí protnout 3 osy úhlů najednou. Přitom žádná z nich nepatří do os úhlů půdorysu, což jen přidává na náročnosti řešení půdorysu. Řešení tohoto problému mě ovšem vedlo k zvýšení univerzálnosti kódu, takže bych označil zahrnutí tohoto půdorysu jako dobrou volbu.



Obr. 14: Vyřešený půdorys domu ve tvaru C

Další problém, který se vyskytl při testování tohoto půdorysu, bylo chybné nalezení průsečíků dvou os. Kód sice byl ošetřen, aby kontroloval, zda obě osy neprotínají okrajovou stěnu při spojování průniků. Neošetřil jsem ale případ, kde by mohla osa protínat druhou osu ale předtím protínat osu, která již byla dokončena.
Při testování jsem také vyvinul funkce, která kontroluje, zda se bod nachází na přímce. Původně byla myšlená pro zakončení půdorysu, ale nakonec jsem se rozhodl pro odolnější řešení v možnosti pro přímku protnout několik jiných os najednou, nehledě na to, zda jsou právě v pracovním seznamu nebo ne.
Tento půdorys ovšem nemá všechny úhly pravé, takže výsledný model nemá ploché stěny. Spojnice mezi některými body jsou taky velmi nevypočitatelné, takže z estetického pohledu teoretické řešení nepůsobí příliš lákavě. Různé "skoky" mezi body střechy jsou způsobeny různými délkami spojnic s okrajovými stěnami, které poté určují výšku bodů.

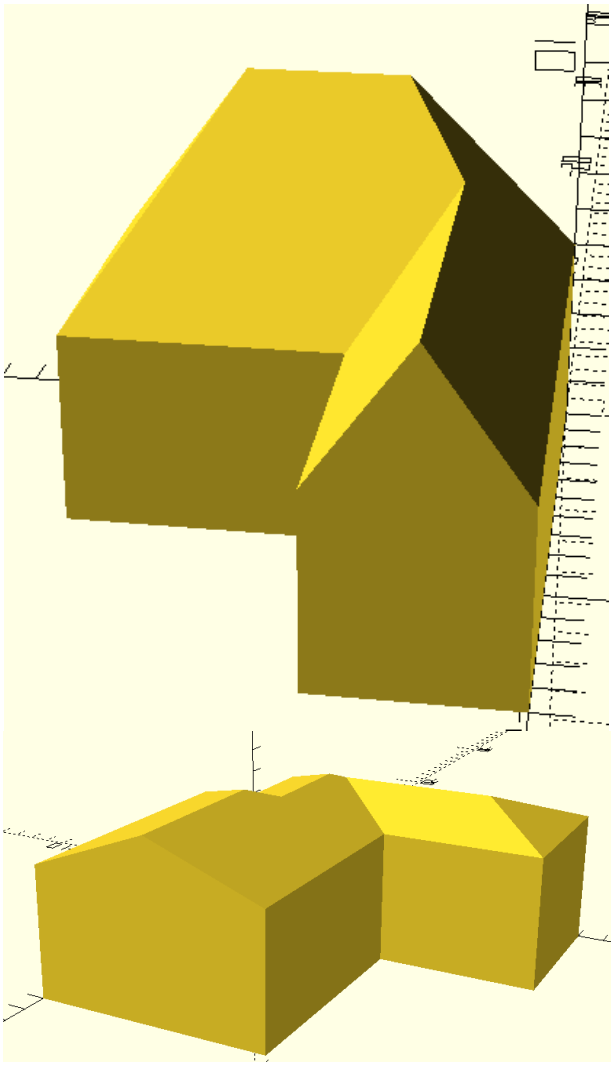


Obr. 15: Vyřešený dům ve tvaru C

Budoucnost programu

Nyní, když mám již poměrně robustní kód, tak začínám podporovat zakázané okraje. Vývoj této funkce je stále na počátku, jelikož střechy se zakázanými okapy jsou mnohonásobně složitější než střechy bez nich. Přesto už ale mám 2 vyřešené modely, které vypadají velmi slibně. Kód je ale silně limitován, zatím zvládá pouze relativně jednoduché půdorysy, kde nejsou dva zakázané okapy na sousedících zdech. Zároveň musí v půdorysu existovat alespoň jeden bod, kde se mohou dvě osy úhlů okrajových zdí protnout a začít tak tvořit střechu.

Dále plánuji přidání podpory pro půdorysu, jejichž obrys je tvořen více než jednou spojitou křivkou. Aktuálně je možné, že by kód správně vytvořil půdorys, ale hlavní problém je ve správném přečtení a zpracování půdorysu. Takové půdorysy také někdy potřebují složitější postup na jejich vyřešení, při kterém je vytvořit několik pomocných os úhlů.



Obr. 16: Dva modely se zakázaným okapem

Závěr

Celkově jsem vytvořil kód, který správně vytvořil modely domů. Musím ale říct, že stále je v celém kódu schováno obrovské množství různých chyb, které čekají na příležitost se vynořit na povrch. Proto stále vyvíjím a testuji kód. Práce na tom kódu mě baví a již přemýšlím o dalších funkcích, například přidání podpory pro zakázané okapy.

Můj cíl byl vytvořit kód, který by za mě řešil střechy, což se mi povedlo. Zároveň jsem si při tom prohloubil znalosti problematiky střech a procvičil jsem si kódování. Také jsem měl možnost pracovat s programem, který má okolo jednoho tisíce řádků, což pro mě byla vítaná změna a hodnotná zkušenost. Předpokládám, že se k problematice střech někdy vrátím a vytvořím lepší kód. Do té doby si myslím, že výsledky mojí práce jsou dostatečné.

Seznam použité literatury

[1] BISOVÁ, Kristína. *Teoretické řešení střech* [online]. Brno, 2020 [Citováno 16. 3. 2023]. Dostupné z: https://is.muni.cz/th/rrzho/Diplomova_prace.pdf. Diplomová práce. Masarykova univerzita, Přírodovědecká fakulta. RNDr. Jan Vondra, PhD.

[2] KINTEL, Marius. Webové sídlo. Dostupné z: https://openscad.org/about.html. [Citováno 19. 12. 2023]

[3] FITZGERALD, Patrick. Online. In: Stack Overflow. 31. 1. 2021. Dostupné z: https://stackoverflow.com/questions/13430231/how-i-can-get-cartesian-coordinate-system-in-matplotlib. [Citováno 8. 2. 2024]

[4] GITHUB. Online. Dostupné z: https://github.com/openscad/openscad/commits/master. [Citováno 19. 12. 2023]