VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

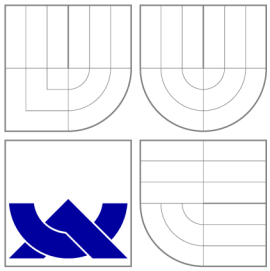
SYSTÉM PRO TVORBU 3D MODELŮ BUDOV Z PŮDORYSŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

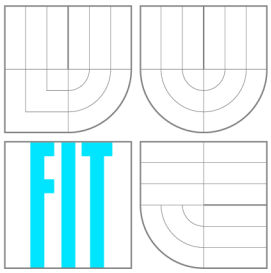
AUTOR PRÁCE
AUTHOR

Bc. ZDENĚK JURKA

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SYSTÉM PRO TVORBU 3D MODELŮ BUDOV Z PŮDORYSŮ

SYSTEM FOR MODELLING OF 3D BUILDINGS FROM FLOOR PLANS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. ZDENĚK JURKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV PŘIBYL
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se zabývá popisem a návrhem aplikace pro tvorbu trojrozměrných modelů budov z jejich půdorysů a jejich publikací na internetu. V úvodní části je popsán postup pro tvorbu půdorysu budovy. Následující část popisuje postup převodu tohoto půdorysu na trojrozměrný model budovy. V dalších částech je popsána implementace aplikace pro tvorbu půdorysů budovy a generování jejího trojrozměrného modelu. Práce se také zabývá implementací aplikace pro prohlížení vytvořeného modelu ve webovém prohlížeči. V závěrečných částech jsou shrnuty dosažené výsledky a omezení zvoleného řešení. Dále jsou zde také diskutována možná rozšíření vytvořené aplikace.

Abstract

This work deals with description and design of application for generating 3D building models from floor plans and their publication on the Internet. First chapter describes methods used for floor plans creation. Next part describes how are foor plans transformed into three-dimensional model. Following chapters describes implementation details concerning application used for creation of building floor plan and three-dimensional model generation. Work also covers implementation of web based application for viewing models in web browser. Final chapters sums up results and limitations of described solution. There are also discussed possible extensions of current project.

Klíčová slova

Editor, .NET, WPF, XNA, Půdorysy budov, 3D Modely budov

Keywords

Editor, .NET, WPF, XNA, Building floor plans, 3D Building models

Citace

Zdeněk Jurka: Systém pro tvorbu 3D modelů budov
z půdorysů, diplomová práce, Brno, FIT VUT v Brně, 2012

Systém pro tvorbu 3D modelů budov z půdorysů

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Jaroslava Přibyla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Zdeněk Jurka
21. května 2012

Poděkování

Rád bych poděkoval panu Ing. Jaroslavu Přibylovi za pomoc a odborné vedení při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Pokud hovoříme o plánech budov, máme většinou na mysli dvourozměrné plány půdorysů. Tento druh plánu můžeme najít na mnoha místech a v různých formách od výkresů projektové dokumentace stavebních projektů po orientační plánky v nákupních centrech. V některých případech je tento druh vizualizace dostatečný, ale pokud prezentujeme plán budovy na počítači, je vhodné mít k dispozici její trojrozměrný model.

V případě trojrozměrných modelů budov se většinou jedná o architektonické modely tvořené ve speciálních nástrojích architekty nebo například o budovy v počítačových hrách, které jsou tvořeny profesionálními 3D grafiky.

Trojrozměrné modely pomáhají architektům objevovat chyby a nedostatky v jejich návrzích. Také umožňují předvést klientům výslednou podobu projektu dlouho před začátkem samotné realizace. Dále je možné pomocí virtuálního průchodu budovou realizovat intuitivnější navigaci po budově než pomocí dvourozměrných plánů. V počítačových hrách jsou namodelované budovy nedílnou součástí virtuálních světů.

Pro tvorbu designových a architektonických návrhů existuje řada nástrojů. Tyto nástroje jsou však ve spoustě případů poměrně složité na používání, nebo se jedná o velmi drahé profesionální nástroje. Jejich další nevýhodou je, že mnoho z nich poskytuje pouze neinteraktivní vizualizaci výsledného návrhu. Další možností je tvorba modelu v některém z obecných editorů pro tvorbu 3D grafiky. Tyto nástroje jsou popsány v následující kapitole.

Tato práce se zabývá popisem a návrhem aplikace pro tvorbu trojrozměrných modelů budov z jejich půdorysů a jejich publikací na internetu. Hlavním cílem je jednoduchost a rychlost použití pro uživatele aplikace. Aplikace by také měla podporovat snadnou rozšiřitelnost svojí funkcionality.

První část tvoří jednoduchý editor pro tvorbu dvourozměrného plánu budovy, na základě kterého bude vytvořen trojrozměrný model. Další část tvoří export takto vytvořeného modelu do aplikace, kterou bude možné publikovat přímo prostřednictvím internetových stránek v internetovém prohlížeči. Tato aplikace by měla umožnit průchod vytvořeným modelem budovy.

Tato kapitola pokrývá motivaci a specifikaci tvořeného projektu. V následující kapitole je popsán způsob, jakým se vytváří půdorys budovy. Další kapitola se poté zabývá popisem postupů použitých pro generování trojrozměrného modelu budovy na základě těchto půdorysů. V kapitole 4 jsou shrnuty a popsány technologie a knihovny použité při tvorbě aplikace. Další kapitola se poté zabývá implementací vlastního editoru. V kapitole 6 je popsána tvorba uživatelského rozhraní editoru. Následující kapitola popisuje tvorbu webové aplikace pro prohlížení vytvořených modelů budov. V kapitole 7 jsou popsány dosažené výsledky a omezení vytvořené aplikace. Závěrečná kapitola shrnuje dosažené výsledky a dis-

kutuje možné pokračování projektu.

1.1 Dostupná řešení

Mezi aplikace, které dovolují vytvořit dvourozměrný plán budovy, patří například *Microsoft Office Visio* [13] nebo *SmartDraw* [16]. Aplikace určené k pokročilým návrhům budov reprezentuje *Archicad* [3]. Pro vizuální návrh 3D modelu budov je možné použít 3D modelovací programy jako je *Autodesk 3Ds Max* [1]. V případě návrhu interiéru budovy jsou k dispozici specializované nástroje jako je *Sweet Home 3D* [17] nebo *Chief Architect* [8].

U jednodušších aplikací jako je SmartDraw a Visio se jedná o multifunkční aplikace, které dovolují vytvářet velké množství různých diagramů, grafů a plánků z různých oblastí. V případě tvorby plánku budov umožňují vytvořit jednoduchý plánek podlaží budovy a umístit do plánku další objekty jako jsou stoly, skříně apod. V tomto případě se jedná spíše o plánky orientační s možností rozvržení jednotlivých objektů v místnosti.

Pokročilé aplikace jako je Archicad dovolují pokročilý návrh celé budovy včetně možnosti definování materiálů jednotlivých částí budovy apod. Zde se již jedná o kompletní návrh budovy včetně všech technických podrobností. Aplikace dále dovolují zobrazení 3D modelu celého objektu včetně volby vizuálního vzhledu jednotlivých částí budovy. Tyto aplikace jsou určeny pro profesionální použití v oblasti architektury a umožňují vytvářet kompletní stavební plány.

V případě pouhého vizuálního návrhu bez vazby na tvorbu plánů je možné využít 3D modelovací software jako již zmíněné 3Ds Max. Tento druh softwaru umožňuje pokročilý vizuální návrh objektu s velice realistickým výstupem v podobě 3D scény.

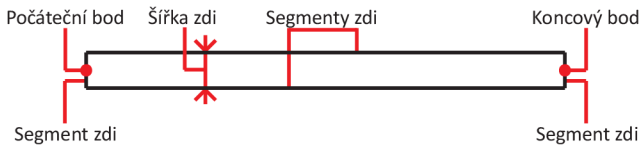
Kapitola 2

Tvorba půdorysu

Prvním krokem při generování modelu budovy je získání dat od uživatele. Z těchto dat je následně vytvořen půdorys vytvářené budovy. V této etapě se řeší problémy jako tvorba rohů mezi zdmi, umístění portálů ve zdech apod. Při tvorbě modelu je zapotřebí půdorys každého podlaží budovy. Půdorysy jednotlivých podlaží sdílí obvodové zdi, které tvoří obrysový tvar budovy. Při návrhu těchto postupů bylo čerpáno z [25] a [26].

2.1 Vytváření zdí

Poloha zdi je určena dvěma body zadanými uživatelem. Tyto body zároveň určují délku zdi. Samotná zeď je tvořena čtyřmi segmenty, jak je vidět na obrázku 2.1. Dalším parametrem je šířka zdi, která určuje polohu jednotlivých segmentů.



Obrázek 2.1: Vytváření zdi

Zdi jsou rozděleny do dvou kategorií a to obvodové a vnitřní. Toto rozdělení usnadňuje další operace potřebné pro tvorbu půdorysu budovy a budou popsány dále v této kapitole.

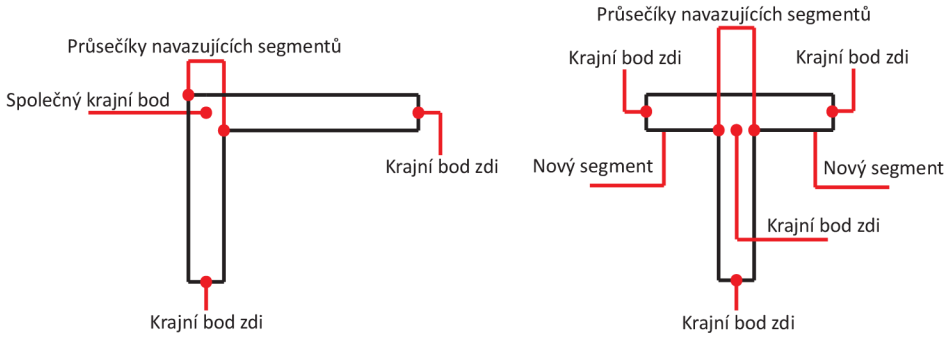
- *Obvodové zdi* – představují obvodovou část budovy a jsou sdíleny mezi všemi patry.
- *Vnitřní zdi* – představují zdi uvnitř budovy a jsou unikátní pro každé patro budovy.

2.2 Napojování zdí

Při vytváření půdorysu je třeba jednotlivé zdi napojovat na sebe tak, aby vznikl požadovaný tvar budovy. Při napojování jednotlivých zdí na sebe mohou nastat dva případy propojení.

Prvním případem je stav, kdy zdi sdílí jeden krajní bod. V tomto případě je roh vytvořen pomocí průsečíků navazujících segmentů. Dále jsou zde odebrány krajní segmenty, které náleží ke společnému krajnímu bodu. Tato situace je znázorněna na obrázku 2.2 vlevo. Napojení pomocí krajních bodů je možné pouze u zdí stejného druhu. Tedy jen mezi obvodovými, nebo jen mezi vnitřními zdmi.

Druhý případ nastává, když je zeď připojena k segmentu jiné zdi. V tomto případě na druhé zdi vzniknou dva nové segmenty, jejichž krajní body jsou určeny průsečíkem se segmenty připojované zdi. Na připojované zdi se také odebere příslušný krajní segment. Tato situace je patrná z obrázku 2.2 vpravo.

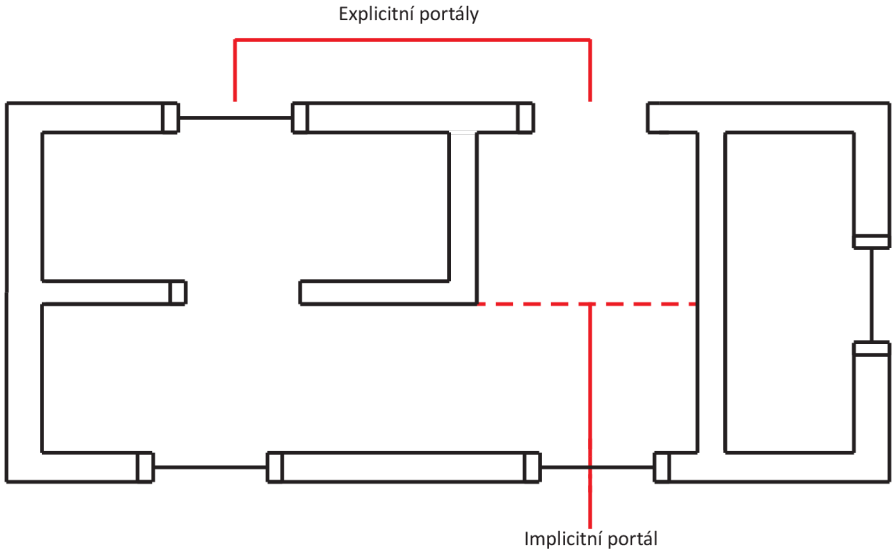


Obrázek 2.2: Napojování zdí

2.3 Portály

Každé podlaží může být rozděleno na jednotlivé oblasti (místnosti). Tyto oblasti jsou spolu propojeny pomocí portálů.

Portál spojuje dvě nezávislé oblasti. Všechny portály mají zdrojový prostor a cílový prostor, které spojují. Každý portál může být *explicitní* nebo *implicitní*. Portály můžeme dále rozdělit podle jejich orientace na *horizontální* a *vertikální*. [33]



Obrázek 2.3: Portály v plánu budovy

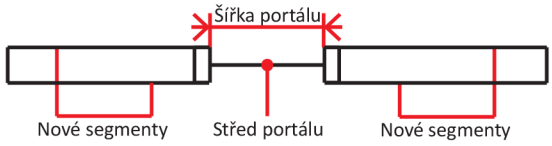
- *Explicitní* – představuje spojení prostorů, které jsou odděleny fyzickou bariérou (zdí), jsou to typicky dveře nebo okna. Obvykle jsou tvořeny otvory ve zdech.

- *Implicitní* – představuje spojení prostorů, které nejsou fyzicky oddělené. Jedná se o přímé propojení dvou prostorů tak, že vypadají jako jeden velký prostor. Tento rozdíl je patrný z obrázku 2.3.
- *Horizontální* – spojují prostory na stejném podlaží.
- *Vertikální* – tvoří propojení mezi jednotlivými patry, jako jsou schodiště nebo výtahové šachty. Vertikální portály jsou typicky implicitní.

[33]

2.4 Vytváření portálů

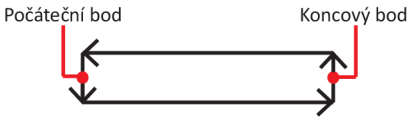
Portály v našem případě reprezentují okna a dveře. Portál je určen zdí, ke které patří, svým středem a šířkou. Při tvorbě portálu je třeba rozdělit segmenty cílové zdi. Tím vzniknou 4 nové segmenty, jejichž krajní body jsou určeny středem portálu a jeho šířkou. Tvorba portálu je znázorněna na obrázku 2.4.



Obrázek 2.4: Vytváření portálů

2.5 Podlahy a stropy

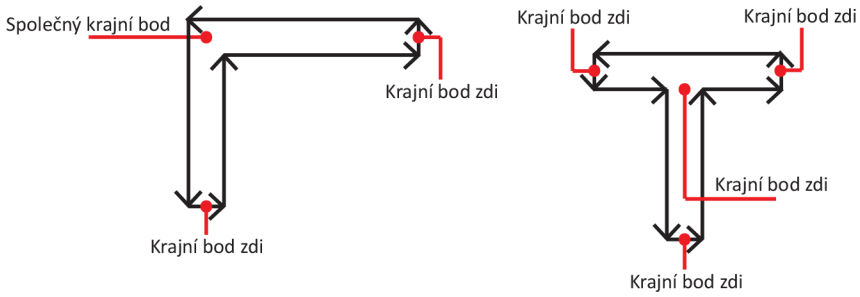
Podlahy a stropy můžeme reprezentovat polygony, jejichž obrys je tvořen krajními body jednotlivých segmentů ve zdech, které tvoří danou místnost. Pro tento účel je vhodné reprezentovat tyto segmenty jako orientovaný graf, v němž krajní body segmentů představují uzly grafu a samotné segmenty potom orientované hrany mezi těmito uzly. Orientace segmentů zdi je patrná z obrázku 2.5.



Obrázek 2.5: Orientace segmentů zdi

Orientace těchto hran musí být zachovány i při navazování jednotlivých zdí, kdy se odstraňují jednotlivé krajní segmenty. Orientace segmentů při napojování zdí jsou znázorněny na obrázku 2.6. Tento způsob tvorby grafu umožňuje zachování správné orientace ve všech případech napojení zdí, jako je například napojení koncového bodu na počáteční, počátečního bodu na počáteční apod.

Polygony jsou zde pak určeny orientovaným cyklem v takto vytvořeném grafu. Je nutné rozlišit tři druhy cyklů podle jejich orientace a typu zdi, ke které příslušejí. Jedná se o vnitřní, vnější a obrysové cykly. Rozlišení jednotlivých cyklů je patrné z obrázku 2.7.

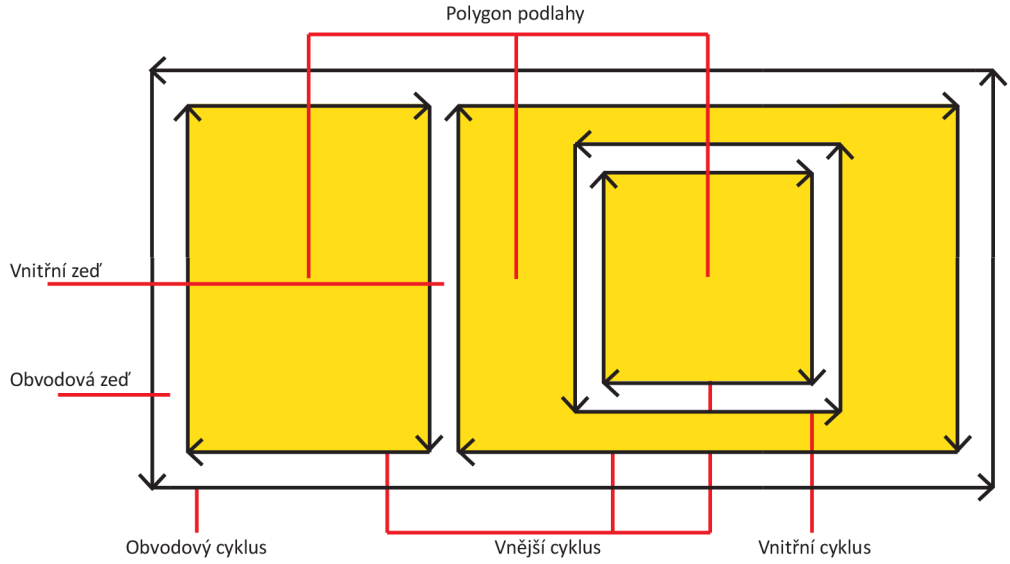


Obrázek 2.6: Orientace segmentů při napojování zdí

- *Vnější cyklus* – cyklus ve směru hodinových ručiček, může být tvořen segmenty libovolného typu.
- *Vnitřní cyklus* – cyklus proti směru hodinových ručiček, může být tvořen segmenty libovolného typu.
- *Obvodový cyklus* – cyklus proti směru hodinových ručiček, tvořený pouze segmenty obvodových zdí.

Vnější cykly představují vlastní vnější obrys dané místnosti, v tomto polygonu je nutné vytvořit díry, které jsou dány vnitřními cykly uvnitř tohoto obrysu.

Obvodový cyklus představuje celkový vnější obrys budovy, který je použit pro generování střechy a vytváření propojení mezi patry.



Obrázek 2.7: Druhy cyklů

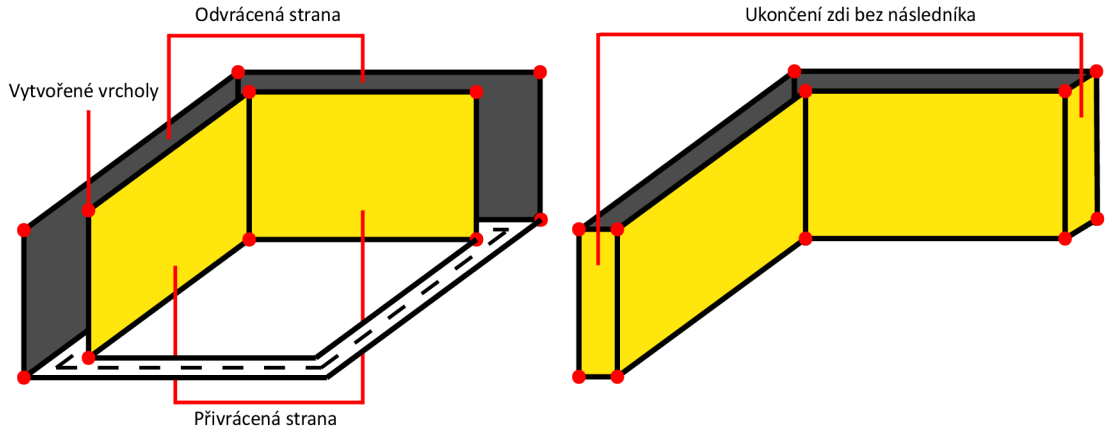
Kapitola 3

Tvorba 3D modelu

Pro tvorbu trojrozměrného modelu jsou použity půdorysy jednotlivých pater. Postup jejich tvorby byl popsán v předchozí kapitole. Tato kapitola se zabývá popisem způsobu, jak z těchto půdorysů vytvořit kompletní trojrozměrný model budovy. Při návrhu těchto postupů bylo čerpáno z [25] a [26].

3.1 Model zdi

Pro každý segment zdi je vygenerován jeden polygon, který je dán krajními body segmentu a výškou podlaží, do kterého zeď patří. Z těchto informací jsou vygenerovány čtyři body v prostoru, které tvoří obrys polygonu zdi. Při generování polygonu je třeba dodržet jeho správnou orientaci. Generování modelu zdi je znázorněno na obrázku 3.1.



Obrázek 3.1: Vytváření polygonu zdi

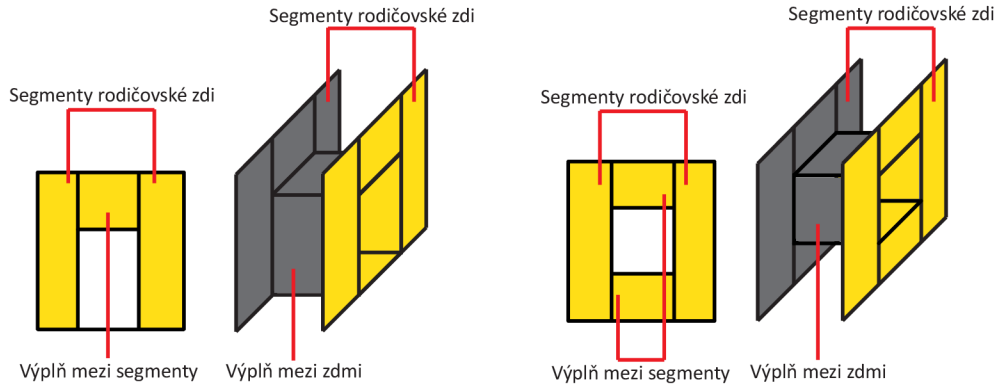
3.2 Model okna a dveří

Pro portály je třeba vytvořit výplň mezi jednotlivými segmenty zdi, které portál obklopují. Dále je nutné vyplnit prostor mezi zdmi, jež portál spojuje.

Výplň dveří je dána jejich šířkou a výškou. Spolu s pozicí dveří v rodičovské zdi je možno určit body, mezi kterými se vygenerují polygony výplně. Výplň mezi zdmi představují čtyři polygony vytvořené mezi body, které se získají z výše popsaných informací. Další polygon

je umístěn nad dveřmi a vyplňuje prostor mezi dveřmi a stropem. Tvorba modelu pro dveře je vidět na obrázku 3.2 vlevo.

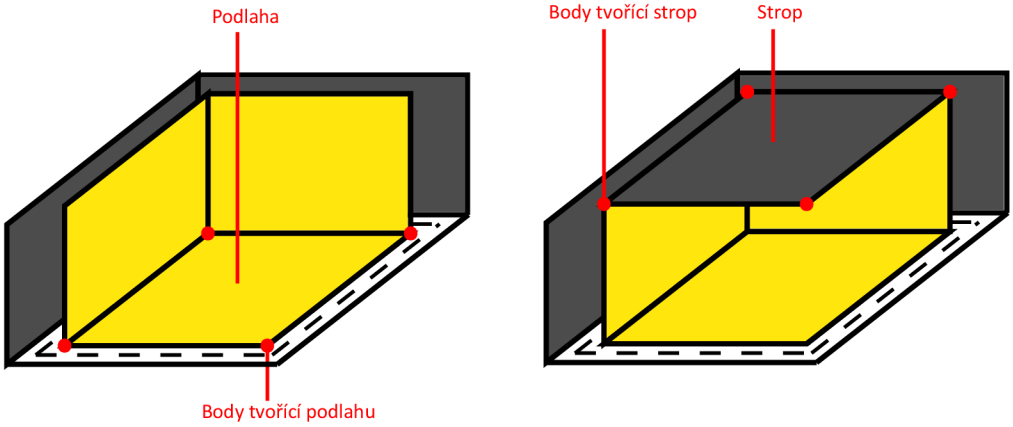
V případě okna je navíc zapotřebí informace o výšce okna nad úrovní podlahy, kde je umístěn další polygon. Ten vyplňuje prostor mezi podlahou a oknem, jak je patrné z obrázku 3.2 vpravo.



Obrázek 3.2: Tvorba výplně portálů

3.3 Stropy a podlahy

Tvary stropů a podlah jsou určeny z půdorysu patra, jak bylo popsáno v předchozí kapitole. Polygony získaně z půdorysů je možné přímo použít jako část modelu budovy. Tyto polygony představují podlahy, jak je vidět na obrázku 3.3 vlevo. Strop pouze posuneme do patřičné výšky a otočíme orientaci polygonu oproti podlaze, což je patrné z obrázku 3.3 vpravo.



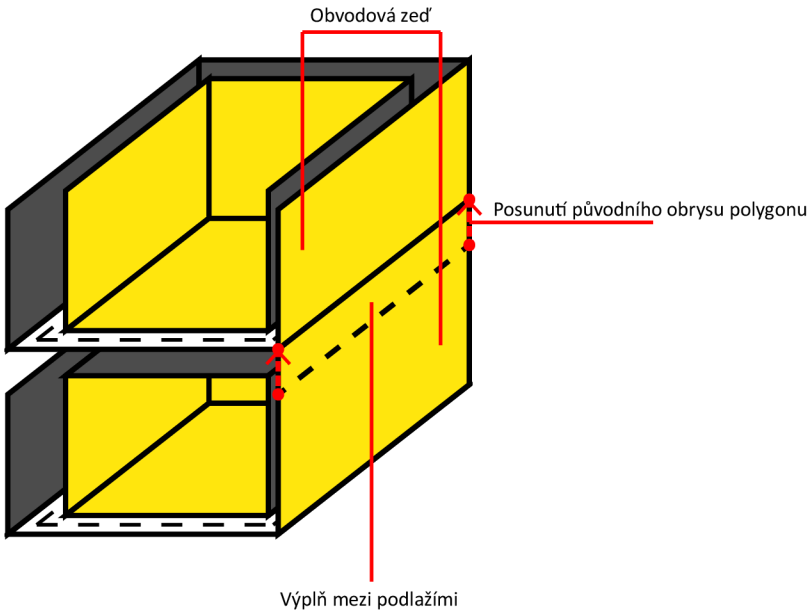
Obrázek 3.3: Tvorba podlahy a stropu

3.4 Skládání podlaží

Popsané postupy se zabývají tvorbou topologie budovy v rámci jednoho podlaží. Při skládání jednotlivých podlaží na sebe vytvořenou topologii pouze posuneme o patřičnou výšku,

která je dána číslem podlaží, výškou jednoho podlaží a výškou stropu.

Následně je třeba vyplnit viditelné mezery mezi jednotlivými podlažími. Pro tento účel je možné využít obvodového cyklu získaného z půdorysu budovy, jak bylo popsáno v minulé kapitole. U všech segmentů, které tvoří tento cyklus, můžeme dva horní body příslušného polygonu posunout směrem nahoru o výšku stropu. Tím dosáhneme propojení jednotlivých podlaží mezi sebou a dále také napojení nejvyššího patra na střechu budovy, jak je znázorněno na obrázku 3.4.



Obrázek 3.4: Tvorba výplně mezi podlažími

3.5 Střecha

Střecha je vytvořena na základě obrysu budovy, jehož získání bylo popsáno v předchozí kapitole. Body, které tvoří obrys budovy, představují přímo polygon ploché střechy. Tento polygon je nutné pouze umístit nad nejvyšší patro budovy.

Kapitola 4

Použité technologie

Tato kapitola popisuje technologie a knihovny použité při tvorbě aplikace. Jsou zde popsány hlavní vývojové platformy, dále pak knihovny použité pro tvorbu grafického uživatelského rozhraní, grafické knihovny, pomocné knihovny a nástroje.

4.1 Vývojové platformy

Jako hlavní vývojová platforma bylo zvoleno prostředí .NET ve verzi 4.0. Hlavním programovacím jazykem byl jazyk C#. Pro tvorbu webové aplikace bylo použito prostředí Silverlight.

4.1.1 C# .NET Framework

Jazyk C# je objektově orientovaný a typově bezpečný programovací jazyk, který je odvozen od jazyků C, C++ a Java. Tento jazyk je speciálně vytvořen pro technologii .NET, jedná se však o samostatný jazyk, který není sám o sobě součástí platformy .NET.

Základem platformy .NET je její běhové prostředí označované jako modul CLR (Common Language Runtime). Kód spuštěný pod kontrolou tohoto modulu bývá označován jako řízený kód (managed code). Libovolný kód spouštěný v tomto běhovém prostředí podléhá dvoufázovému procesu překladu. Nejdříve se zdrojový kód přeloží do jazyka IL (nízkoúrovňový jazyk s jednoduchou syntaxí). Překlad do nativního kódu pro cílovou platformu probíhá až v okamžiku volání dané části kódu. Tento způsob překladu se označuje jako JIT (just in time compilation). Hlavní výhodou kódu spuštěného v rámci modulu CLR je možnost využití knihovny základních tříd platformy .NET.

Knihovna tříd .NET Frameworku (BCL – base class library) obsahuje třídy grafického uživatelského rozhraní systému Windows, třídy pro grafický výstup, třídy pro práci se sítí a souborovým systémem, třídy pro přístup k databázím a pro možnost dotazování nad daty a mnoho dalších tříd. [29]

4.1.2 Silverlight

Silverlight je platforma pro tvorbu Rich Internet Applications [21]. Jejím základem je zásuvný modul, který běží uvnitř internetového prohlížeče. Tento zásuvný modul funguje ve všech hlavních prohlížečích (Internet Explorer, Mozilla Firefox, Safari, Opera a Google Chrome) stejně jako na platformě Macintosh.

K tvorbě těchto aplikací slouží XAML, HTML, JavaScript a C# (případně jiný jazyk s podporou CLR). K dispozici je podmnožina funkcí poskytovaných knihovnou základních tříd .NET Frameworku. Jazyk XAML pro Silverlight je podmnožinou WPF. Silverlight mimo jiné podporuje vektorovou grafiku, animace, zpracování videa a zvuku. V poslední verzi byla přidána podpora pro akcelerovanou 3D grafiku v prohlížeči pomocí XNA frameworku. Nicméně dle dostupných informací bude aktuální verze poslední a na místo tohoto externího zásuvného modulu nastoupí HTML5.[22]

Důvody pro výběr této technologie jsou uvedeny v kapitole 7.

4.2 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní bylo tvořeno pomocí technologie Windows Presentation Foundation s rozšířením o Ribbon aplikační menu.

4.2.1 WPF

Windows Presentation Foundation je knihovna sloužící k tvorbě grafických uživatelských rozhraní. Na rozdíl od knihovny WinForms nepoužívá grafickou knihovnu GDI a popisovače oken k vytváření grafických ovládacích prvků, ale využívá služeb rozhraní DirectX a vektorovou grafiku. Dále zavádí podporu pro práci s videem, zvukem a animacemi, navázání vlastností všech prvků uživatelského rozhraní na datové zdroje apod. Vývoj aplikace je rozdělen do dvou částí a to na popis vzhledu pomocí značkovacího jazyka XAML a na tvorbu logiky pomocí kódu na pozadí.

XAML (XML for Applications Markup Language – značkovací jazyk XML pro aplikace) je způsob zápisu XML používaný při definování hierarchické struktury uživatelského rozhraní. Umožňuje přímo vytvářet prvky uživatelského rozhraní, definovat jejich vzhled, animace a chování. Dále poskytuje prostředky k tvorbě stylů pro takto definovaná rozhraní.

Kód na pozadí je napsán v některém jazyce platformy .NET. Oficiálně podporovaným jazykem je jazyk C#. V tomto místě je možné vytvořit programovou logiku pro uživatelské rozhraní. Postup jeho tvorby je velmi podobný tvorbě webových aplikací pomocí ASP.NET. [30]

4.2.2 Ribbon for WPF

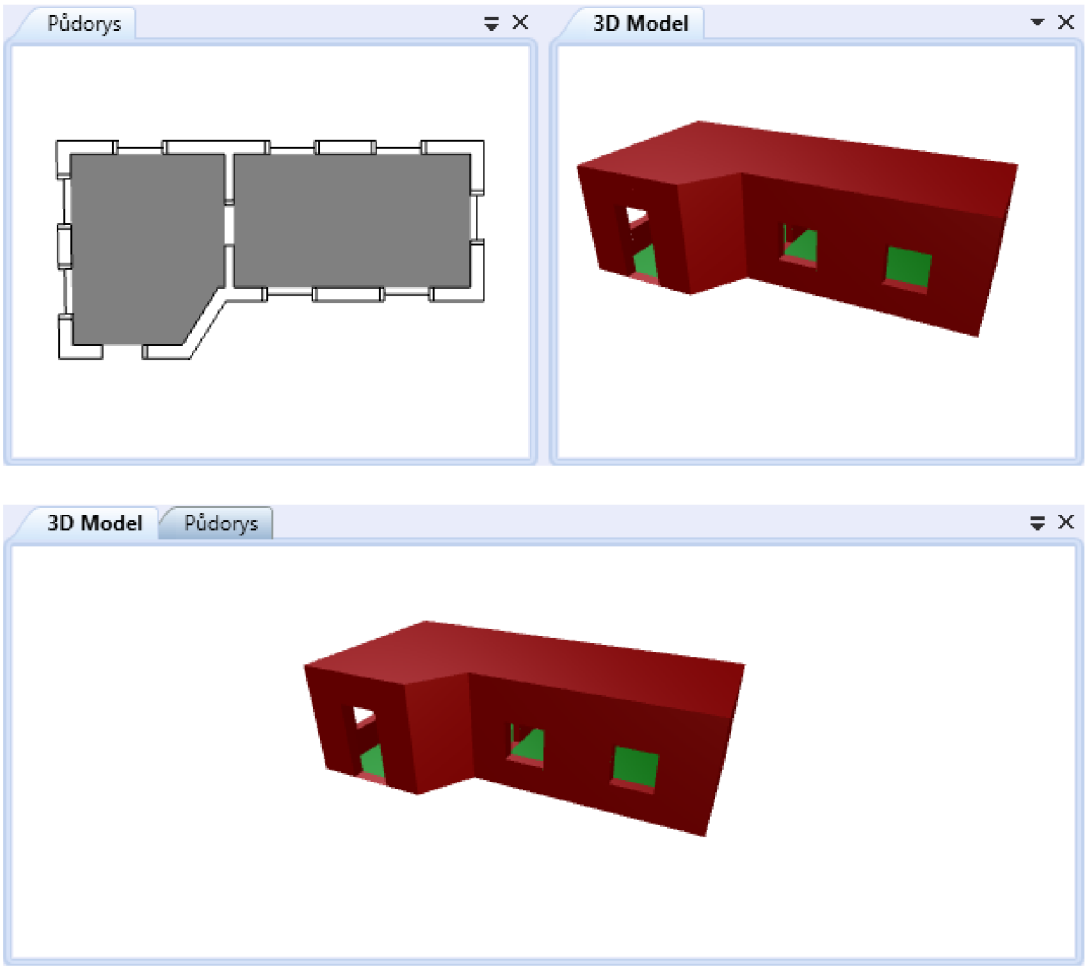
Ribbon je panel nástrojů, který organizuje přístup k jednotlivým funkcím aplikace do jednotlivých záložek, které jsou umístěny v horní části okna aplikace. Ribbon nahrazuje tradiční panely nástrojů a aplikační menu.

Knihovna Ribbon for WPF je implementace tohoto uživatelského prvku pro Windows Presentation Foundation. Obsahuje všechny základní prvky uživatelského rozhraní Ribbon jako jsou skupiny, kontrolky (tlačítka, zaškrtávací tlačítka, apod.), aplikační menu, panel rychlého přístupu, integrace s oknem aplikace apod. [15]

4.2.3 AvalonDock

Knihovna AvalonDock implementuje plovoucí layoutovací systém pro WPF. Tato implementace je podobná dokovacímu systému, který je použit v aplikaci Visual Studio. Tento je považován za standard, dle něhož se uvedený druh dokovacích systémů implementuje. Jedná se zřejmě o jednu z nejlepších knihoven tohoto druhu, které jsou dostupné zdarma.

Použití tohoto systému je vidět na obrázku 4.1, kde je vidět stejný obsah umístěný na různých pozicích uvnitř okna aplikace.[5]



Obrázek 4.1: Použití knihovny AvalonDock

4.2.4 Avalon Wizard

Avalon Wizard implementuje prvek grafického uživatelského rozhraní zvaný průvodce (Wizzard) pro knihovnu WPF. Obsahuje jednoduché prostředí pro tvorbu vlastních průvodců, umožňuje změnu vzhledu pomocí skinů, případně umí převzít vzhled z aktuálního systémového nastavení. [4]

4.3 Grafické knihovny

Jako knihovna pro vizualizaci byl použit XNA framework. Pro prácí s polygony byla použita knihovna General Polygon Clipper library.

4.3.1 XNA

XNA je oficiální Microsoft knihovna pro práci s DirectX na platformě .NET primárně zaměřená na vývoj her. Obsahuje nástroje pro grafiku, zvuk a podporu vstupních zařízení. Dále zahrnuje rozšiřitelnou knihovnu pro správu obsahu (Content Pipeline).Pracuje na několika platformách jako jsou Windows, Xbox, Zune a Windows Phone 7.

XNA neposkytuje přístup ke všem možnostem DirectX stejně jako nedovoluje přístup k interním popisovačům, což znemožňuje přímou spolupráci s jinými DirectX knihovnami.

XNA může pracovat ve dvou profilech Reach a HiDef. Profil definuje množinu funkcí podporovaných hardwarem. Reach profil implementuje high-level shader language (HLSL) Shader Model 2.0 a HiDef profil implementuje HLSL Shader Model 3.0.

HiDef profil je určen pro výkonný hardware kompatibilní s DirectX 10 nebo vyšší a podporuje nejnovější grafické funkce. Reach profil je navržen pro podporu širokého spektra zařízení. Podporuje pouze omezenou sadu grafických funkcí, ty jsou ale dostupné na všech zařízeních kompatibilních s XNA Frameworkem. [23]

4.3.2 Collada

Collada představuje formát pro ukládání 3D objektů a animací. Je založena na veřejně dostupném XML schématu. Soubory ve formátu Collada tedy představují klasický XML dokument, nejpoužívanější koncovkou těchto souborů bývá *.dae*.

Collada formát podporuje uložení geometrie, animací, světel, kamer materiálů apod. Dále umožňuje v souboru uložit kompletní graf scény.

Tento formát standardizovalo The Khronos Group Inc. [11] Collada je podporována velkým množstvím editorů 3D obsahu jako je například dříve popisovaný Blender a 3D Studio Max. [9]

4.3.3 General Polygon Clipper library

General Polygon Clipper library (GPC) je knihovna napsaná v jazyce C++, která poskytuje funkcionalitu pro operace nad polygony. Mezi hlavní vlastnosti patří podpora polygonů s dírami, implementace všech booleovských operací nad polygony a triangulace polygonů. Jejími dalšími výhodami jsou velká rychlost implementovaných algoritmů a dostupnost C# wrapperu pro práci s touto knihovnou.

Činnost této knihovny je představena na obrázku 4.2. Na něm jsou vidět dva polygony představující mapu Velké Británie a Irska a polygon reprezentující spirálu. Na tyto polygony je aplikována operace xor. Jak je patrné z obrázku, společná část obou polygonů je vykreslena světlou barvou, samostatné části potom tmavou. Z tohoto výstupu je patrná jak velká přesnost této knihovny a vzhledem k tomu, že tento výpočet probíhal v reálném čase, tak i její rychlost. Obrázek byl pořízen z testovací aplikace dostupné na webových stránkách knihovny. [28]

4.4 Pomocné knihovny

V této části jsou popsány další knihovny a nástroje, které byly použity při implementaci aplikace.

Obrázek 4.2: Ukázka činnosti GPC

4.4.1 Castle Windsor

Castle Windsor představuje implementaci IoC kontejneru pro platformu .NET. IoC kontejner slouží pro správu, registraci a kontrolu doby života abstraktních datových typů. Jeho použití zjednodušuje tvorbu tříd založených na dependency injection (více viz [27] a [20]) a dále také umožňuje snížení vazeb mezi třídami, kdy není třeba znát konkrétní datový typ, ale pouze rozhraní, které datový typ publikuje. Tento systém je určen hlavně pro správu jednotlivých služeb v aplikaci. [7]

4.4.2 PostSharp

PostSharp je knihovna, která posytuje Aspect Oriented Programming (AOP) přístup pro platformu .NET. Základní myšlenkou AOP je přesunout opakující se části kódu jako je logování, kontrola oprávnění, ošetření výjimek apod., které je jinými přístupy značně obtížné izolovat a zabránit jejich opakování, do tzv. aspektů, které obsahují potřebný kód. Následně se pomocí deklarativních zápisů určují místa v kódu, kde se má aspekt vykonat.

Součástí této knihovny je rozšíření pro překladový systém .NET, které umožňuje formou postprocessingu injektovat kód vytvořený v aspektech na patřičná místa v kódu. [14]

4.4.3 Log4Net

Log4Net je .NET implementace logovací knihovny log4j a patří mezi jednu z nejpoužívanějších logovacích knihoven pro platfotmu .NET. Mezi její velké výhody patří široká nastavitelnost, konfigurace pomocí XML, a podpora vícevláknového a meziprocesového přístupu. [2]

4.4.4 DotNetZip

DotNetZip je knihovna pro platformu .NET určená pro práci se soubory ve formátu zip. Výhodou této knihovny je její rychlost a jednoduché API, které umožňuje jednoduchý přístup k obsahu archivu, jeho dekompresi, případně vytváření nového archivu, nebo přidávání obsahu do již existujícího. [10]

4.4.5 Visual Studio Async CTP

Visual Studio Async CTP je rozšíření pro platformu .NET, které přináší podporu asynchronního programováni pomocí tzv. async await přístupu (více viz [32]), který byl pro platformu .NET představen v .NET Frameworku 4.5, nicméně tato část byla uvolněna jako rozšíření i pro starší verze .NET Frameworku. Součástí knihovny je rozšíření překladového systému .NET. [18]

Kapitola 5

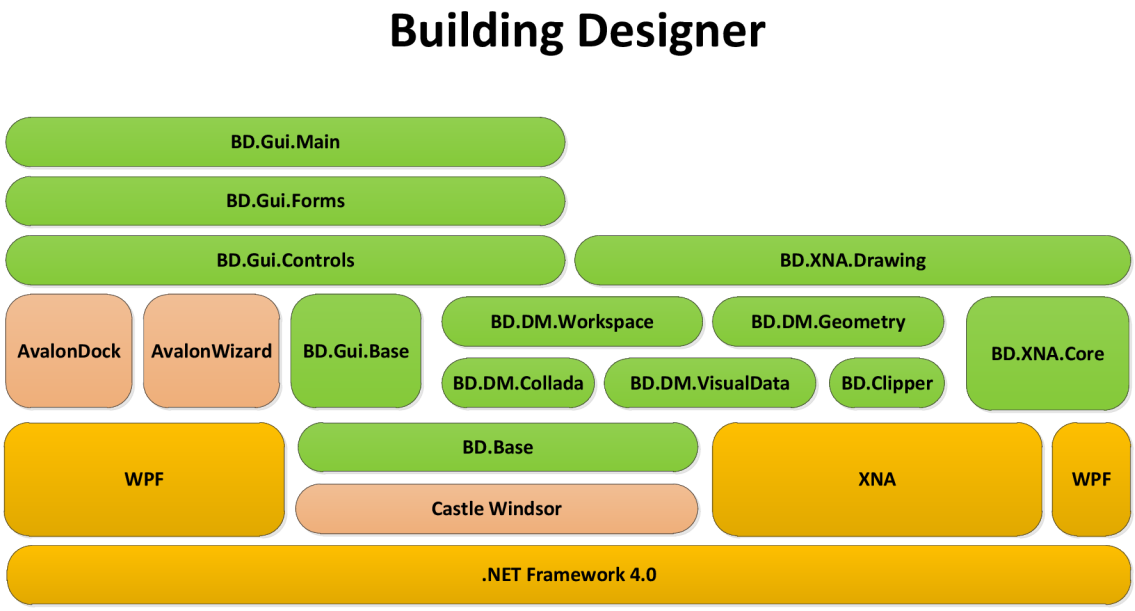
Implementace editoru

Tato část se zabývá postupem implementace editoru v prostředí .NET. Základní popis použitých knihoven a vývojové platformy je v kapitole 4.

5.1 Architektura aplikace

Celá aplikace je rozdělena do několika vrstev, jak je naznačeno na obrázku 5.1. Jednotlivé vrstvy jsou logicky rozděleny do čtyř částí a to na část, která pracuje s XNA, část pro grafické uživatelské rozhraní, část pro práci s daty a část se základní funkcionalitou.

Základním jmenným prostorem je *BuildingDesigner*, který je v obrázku 5.1 zkrácen na *BD*. Každý zelený blok představuje jednu assembly (sestavení – stavební modul v .NET, jedná se o dll knihovnu nebo exe soubor) výsledné aplikace (některé menší části jsou spojeny do jedné knihovny s oddělenými jmennými prostory).



Obrázek 5.1: Architektura aplikace

5.1.1 Základní funkcionalita

Tuto část představují následující jmenné prostory:

- BuildingDesigner.Base
- BuildingDesigner.Helpers
- BuildingDesigner.Clipper

V části *BuildingDesigner.Base* jsou umístěna základní rozhraní a třídy pro celou aplikaci jako je rozhraní *ICommand* a *IParametrizedCommand* pro návrhový vzor *Command*, základní typy výjimek apod. Dále se zde nachází static proxy *IoC* pro přístup k IoC kontejneru CastleWindsor.

Sekce *BuildingDesigner.Helpers* obsahuje pomocné třídy, které rozšiřují funkcionalitu základních tříd .NET a XNA. Většina této funkcionality je implementována pomocí extension metod stávajících tříd. Nachází se tu například implementace cyklu foreach pomocí lambda výrazů implementovaná nad rozhraním *IEnumerable*, nebo metoda pro spouštění událostí implementována nad třídou *Object*.

Blok *BuildingDesigner.Clipper* obsahuje získaný wrapper knihovny GPC jak bylo popsáno v části 4.3.3. Dále se tu nachází třídy, které propojují datový model knihovny GPC a XNA.

5.1.2 Datový model a přístup k datům

Tato část je tvořena jmenným prostorem *BuildingDesigner.DataModel*, který je v obrázku 5.1 zkrácen na *BD.DM*. Součástí tohoto jmenného prostoru jsou následující bloky:

- BuildingDesigner.DataModel.VisualData
- BuildingDesigner.DataModel.Geometry
- BuildingDesigner.DataModel.Collada
- BuildingDesigner.DataModel.Workspace

Jmenný prostor *BuildingDesigner.DataModel.VisualData* představuje datové třídy, které popisují grafické objekty jako jsou zdi, okna, dveře apod. Tyto třídy popisují celý grafický model vytvářené budovy.

V bloku *BuildingDesigner.DataModel.Geometry* jsou umístěny třídy, které zajišťují generování geometrie pro jednotlivé části modelu. Tyto třídy jsou dále zodpovědné za přegenerování modelu, v případě změny některé jeho části.

Část *BuildingDesigner.DataModel.Collada* zahrnuje třídy, které popisují datový model formátu Collada, který byl blíže popsán v sekci 4.3.2.

Jmenný prostor *BuildingDesigner.DataModel.Workspace* zprostředkovává přístup k datovému modelu a operacím nad ním. Nachází se zde operace jako je export do formátu Collada, ukládání modelu apod.

5.1.3 XNA

V tomto bloku se nachází následující podbloky:

- BuildingDesigner.XNA.Core
- BuildingDesigner.XNA.Drawing

Část *BuildingDesigner.XNA.Core* implementuje adaptér XNA pro použití s WPF. Dále jsou zde třídy, které implementují herní komponenty z XNA frameworku pro obecné použití.

Jmenný prostor *BuildingDesigner.XNA.Drawing* obsahuje implementaci vlastního vykreslování celé scény a to ve 2D pohledu pro vytváření plánu a 3D pohledu pro náhled vygenerovaného modelu.

5.1.4 Grafické uživatelské rozhraní

Tato oblast zahrnuje čtyři podbloky:

- BuildingDesigner.Gui.Base
- BuildingDesigner.Gui.Controls
- BuildingDesigner.Gui.Forms
- BuildingDesigner.Gui.Main

Blok *BuildingDesigner.Gui.Base* obsahuje upravené kolekce pro prvky uživatelského rozhraní a třídy pro přidávání chování uživatelských prvků k již existujícím pomocí kompozice.

Část *BuildingDesigner.Gui.Controls* představuje vlastní vytvořené prvky uživatelského rozhraní jako je například plátno pro XNA, panely nástrojů apod.

Jmenný prostor *BuildingDesigner.Gui.Forms* obsahuje hlavní formulář aplikace a průvodce pro export modelu do webové aplikace a formátu Collada.

Blok *BuildingDesigner.Gui.Main* zahrnuje třídy pro práci s jednotlivými částmi GUI a jejich dynamické sestavování za běhu. Mezi tyto třídy patří například třída *DockablePaneManager*, která umožňuje dynamickou správu dokovacích panelů.

5.2 Propojení WPF a XNA

Knihovna XNA je primárně určena pro tvorbu her na systémech s operačním systémem Windows a Windows Phone, nebo pro zařízení Xbox360 a Zune. Tomu odpovídá i základní aplikační model této knihovny. Ten je tvořen hlavně třídou *Game*, která implementuje hlavní vykreslovací smyčku, smyčku pro aktualizaci scény, dále pak obsahuje přístup k *Content Pipeline*, stromu herních komponent apod. Na systémech Windows tato třída interně vytváří okno aplikace, v jehož kontextu hra běží. Handle tohoto okna slouží pro vykreslování výsledné scény a jeho smyčka zpráv je použita pro řízení smyčky aktualizace herní scény (reakce na idle zprávu). Tento aplikační model je však nevhodný pro vizualizaci scény v rámci jiné aplikace. K tomuto účelu je třeba vytvořit nový model ze základních tříd, které poskytuje XNA Framework.

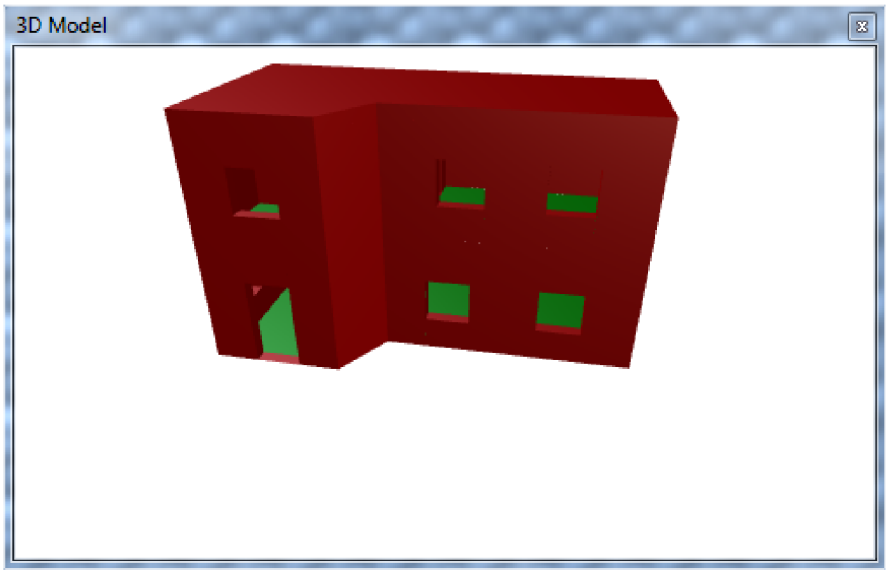
Prvním krokem pro navázání XNA s technologií WPF je vytvoření nového grafického kontextu pro vykreslování do daného prvku uživatelského rozhraní. Při vytváření nového

grafického kontextu je však třeba znát gdi handle cílového objektu, do kterého se bude vykreslovat. Nicméně v aplikacích založených na WPF máme k dispozici pouze handle celého okna, protože vytváření a správu dalších prvků uživatelského rozhraní včetně směrování zpráv mezi nimi provádí WPF interně a transparentně vůči zbytku systému. Tuto situaci lze obejít několika způsoby.

Často používaný je způsob, kdy se nad hlavním formulářem aplikace WPF vytvoří další okno, do kterého se vykresluje. Takto vytvořené okno se poté umístí nad cílovou kontrolku. Toto řešení nicméně přináší spoustu problémů, jako například nutnost synchronizace mezi těmito okny (každé běží ve vlastním vlákně s vlastní smyčkou zpráv), problém s řešením modálních dialogových oken a spoustu dalších. Mnohem vhodnějším přístupem je vytvoření vykreslovacího kontextu pro celé okno a následné omezení vykreslování na oblast, kde se nachází cílový prvek.

Dalším krokem je vytvoření vykreslovací a aktualizační smyčky. Vykreslovací smyčka je navázána na událost *CompositionTarget.Rendering*, kterou spouští renderovací vlákno WPF při vykreslování. V případě nenáročných aktualizací scény je možné na tuto událost navázat i smyčku aktualizace scény, jak je tomu i tomto případě.

V rámci implementace tohoto řešení byl vytvořen následující model. Prvek uživatelského rozhraní *XnaCanvas*, který reprezentuje plátno, do nějž se vykresluje. Tento prvek je možné umístit na jakékoliv místo grafického uživatelského rozhraní postaveného na WPF. Další částí je třída *AbstractDrawingBehaviour*, která implementuje podobné rozhraní jako třída *Game*, tedy metody jako *Initialize, LoadContent, Update, Draw* apod. Další částí jsou třídy *GraphicsDeviceService* a *WpfGraphicsProvider*, které se starají o vytváření a udržování vykreslovacího kontextu při změně velikosti ovládacího prvku, jeho zrušení, případně přemístění do jiného okna. Poslední částí je třída *DrawingController*, která propojuje tyto tři popsané části dohromady. Výsledný prvek uživatelského rozhraní, který je schopný vykreslit XNA scénu je zobrazen na obrázku 5.2, kde je vložen do jednoduchého okna.



Obrázek 5.2: XNA plátno

5.3 General Polygon Clipper library a XNA

Knihovna GPC je napsána v jazyce C++ nicméně je pro ni k dispozici wrapper napsaný v jazyce C#. Tento wrapper pracuje s vlastní datovou reprezentací vrcholů jednotlivých polygonů. Tyto datové typy však nejsou kompatibilní s datovými typy knihovny XNA, které využívá implementovaná datová vrstva popisující model budovy. Z tohoto důvodu bylo nutné vytvořit adaptéry, které umožňují převod mezi jednotlivými reprezentacemi.

Interní střída *VertexList*, která v knihovně GPC představuje vrchol, byla upravena tak, aby jako parametry jednotlivých metod brala struktury které představují vrcholy v knihovně XNA. Tímto je zajištěno transparentní předávání vrcholů z datového modelu do této knihovny.

Při triangulaci vrací GPC jako výsledek třídu *Tristrip*, která představuje triangulovaný polygon. Pro tuto třídu byla vytvořena obalová třída *TriangleStrips*, která převádí interní reprezentaci geometrie typu triangle strip do reprezentace, kterou umí zobrazit *XNA*. Pro tento převod byla dále implementována třída *TriangleStrip*, která představuje jeden triangle strip, jež je možno zobrazit knihovnou XNA.

5.4 Datová reprezentace modelu

Celý model budovy je reprezentován stromovou strukturou, jejíž uzly představují jednotlivé části modelu. Bázovou třídou pro všechny uzly stromu je třída *Node*. Uzly stromu nesou informaci o jednotlivých částech 2D a 3D geometrie modelu, některé uzly však slouží pouze pro zjednodušení operací nad modelem.

Jednotlivé vlastnosti těchto uzlů vyvolávají při změně událost *PropertyChanged*, která je součástí základního rozhraní *INotifyPropertyChanged*. Tato událost nese jméno vlastnosti, která se změnila. Na základě těchto událostí jsou spouštěny akce pro regenerování modelu. Implementace tohoto rozhraní je poměrně zdlouhavá, protože není možné použít automatické vlastnosti (auto property). Tuto problematickou část je možné vyřešit pomocí aspektů z knihovny *PostSharp*. Ve verzi, která je dispozici zdarma, je sice zapotřebí samotnou událost naimplementovat přímo na požadované třídě, nicméně její vyvolání je možné realizovat pomocí aspektu. V našem případě je událost implementována v třídě *NotifyPropertyChangedNode*. Následně je možné vlastnosti naimplementovat pomocí automatických vlastností a pouze odekorovat příslušným atributem *NotifyPropertyChanged*.

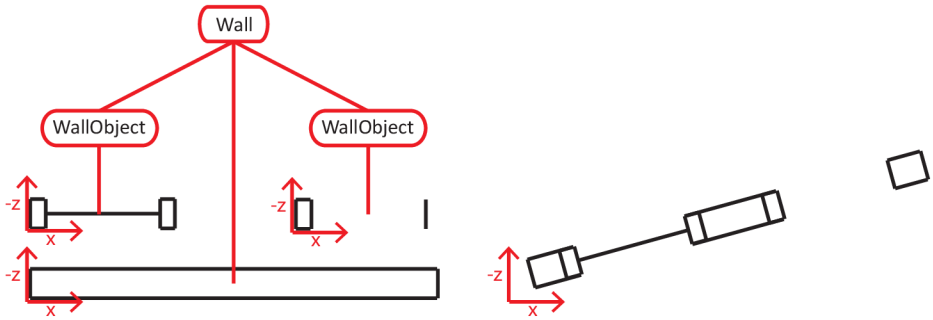
Nad celým stromem je naimplementován iterátor pro průchod stromem. Pomocí tohoto iterátoru je možné stromem procházet pomocí jediného příkazu foreach. Tato implementace také umožňuje nad stromem používat dotazy pomocí technologie LINQ (Language-Integrated Query - obecný dotazovací jazyk nad daty integrovaný v platfomě .NET, více viz [12]), včetně využití paralelních dotazů.

Pro přístup k datovému modelu slouží třída *Workspace*. Tato třída exportuje rozhraní pro přístup ke kořenovému uzlu grafu modelu a právě editovanému patru budovy. Dále pak obsahuje metody pro vyhledávání uzlů ve stromu podle zadaných pravidel. Nejčastěji využívaným přístupem je vyhledání objektu na zadaných souřadnicích v prostoru. K tomuto účelu všechny uzly, které toto vyhledání dovolují, implementují rozhraní *ISelectable*.

5.5 Tvorba modelu

Jednotlivé uzly s geometrií pracují v lokálním souřadném systému. Takto vytvořená geometrie je pak pomocí transformace uložené v uzlu a transformací v uzlech po cestě ke kořenu stromu transformována na výslednou pozici v modelu. Tento přístup značně zjednodušuje generování geometrie modelu. Nicméně při výpočtech, které pracují s několika uzly, je potřeba převádět mezi jednotlivými lokálními souřadnými systémy. Uzly, které obsahují transformace jednotlivých částí modelu, jsou odvozeny od bázově třídy *VisualNode*. Tato třída poskytuje metody pro převod bodů do lokálního souřadného systému a z lokálního souřadného systému do souřadného systému modelu. Dále také publikuje svoji transformační matici, která je získána z lokální transformace a transformací rodičovských uzlů.

Popsané transformace jednotlivých uzlů jsou patrné z obrázku 5.3, kde je ilustrován model zdi se dvěma portály. Model zdi i jednotlivých portálů je vytvořen v lokálním souřadném systému jak je vidět na tomto obrázku vlevo. Transformace v uzlu zdi obsahuje posunutí zdi a její natočení vůči ose x. Transformace uložené v uzlech portálu nesou pouze posunutí od začátku zdi. Složením jednotlivých transformací dostaneme výsledný model, který je vidět na obrázku 5.3 vpravo.



Obrázek 5.3: Transformace uzlů stromu modelu

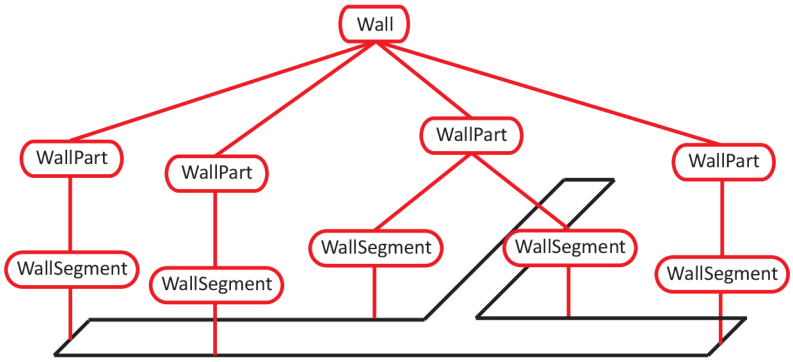
Výše uvedený systém respektují i kolizní testy používané k výběru objektů ve stromu pomocí rozhraní *ISelectable*. V těchto testech se vyhledává na zadaných souřadnicích pomocí obalové koule kolem bodu prohledávání. Střed tohoto tělesa je při kolizním testu transformován do lokálního souřadného systému daného uzlu pomocí inverzní transformace uzlu. Po této transformaci se vypočítá samotná kolize s obalovým tělesem daného uzlu. Tento přístup umožňuje použití osově zarovnaných obalových těles pro jednotlivé objekty scény.

Problémem tohoto přístupu je možnost vzniku artefaktů v modelu způsobených zaokrouhlováním během převodů mezi jednotlivými souřadnými systémy.

5.5.1 Zdi

Uživatel při tvorbě půdorysu zadá zeď jako dva body. V modelu je zeď reprezentována počátečním bodem, délkou a úhlem natočení oproti ose x. Zeď jako taková nenese žádnou geometrii, tu nesou pouze synovské uzly, které reprezentují jednotlivé segmenty zdi. Tyto uzly jsou navíc sdruženy v uzlech, které představují jednotlivé části zdi. Tato reprezentace umožňuje jednoduché operace nad všemi segmenty dané části zdi. Strom modelu zdi je znázorněn na obrázku 5.4.

Zeď je reprezentována třídou *Wall*, jednotlivé části zdi pak představuje třída *WallPart* a jednotlivé segmenty třída *WallSegment*.



Obrázek 5.4: Strom modelu zdi

Jednotlivé segmenty jsou vytvářeny pomocí délky a šířky zdi ve středu souřadného systému. Odtud jsou poté při vlastním vykreslování umístěny na správnou pozici pomocí transformace získané z natočení zdi vůči ose x a posunutí, které udává počáteční bod zadaný uživatelem. Změny délky nebo šířky zdi, jsou propagované pomocí třídy *WallPart* ke všem příslušným segmentům. O samotné generování geometrie se starají třídy *WallGeometry* a *WallSegmentGeometry*.

Při změně pozic krajních bodů jsou tyto změny zachytávány ve třídách, které se starají o navazování jednotlivých zdí na sebe. V těchto třídách se vyhledávají zdi v daném patře, které se nacházejí na patřičné pozici. V případě, že se taková zeď najde, se provede změna geometrie tak, aby odpovídala patřičnému druhu napojení. Druhy napojování jednotlivých zdí na sebe byly popsány v kapitole 2.2. O navazování pomocí koncových bodů se stará třída *WallCornerGeometry*, o připojování k jednotlivým segmentům zdí se stará třída *WallSegmnetConnectionGeometry*.

Samotný trojrozměrný model zdi se generuje zvlášť pro každý segment. Tuto funkcionalitu zajišťuje třída *WallSegmentGeometry3D*. V této třídě se pomocí bodů, které tvoří polygon zdi, vytvoří dva trojúhelníky výsledného modelu a na základě orientace segmentu se vygenerují patřičné normály. Normála je získána z rodičovského uzlu *WallPart*. Výška zdi je pak získána z rodičovského uzlu *Floor*. Vykreslená zeď je vidět na obrázku 5.5, kde je vlevo ukázán půdorys a vpravo vygenerovaný model.

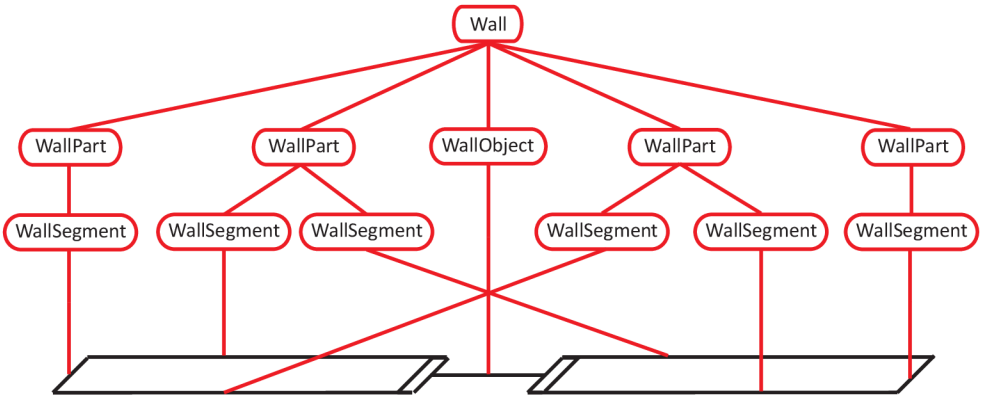
Obrázek 5.5: Vykreslený model zdi

5.5.2 Portály

Uživatel zadá pozici portálu pomocí jednoho bodu, který určuje střed portálu. Na základě této pozice se určí rodičovská zeď tohoto portálu. Pozice portálu v modelu je dána zdí, ke

které přísluší, a vzdáleností od počátku této zdi. Strom modelu zdi s portálem je předveden na obrázku 5.6.

Portály ve zdech představuje bázová třída *WallObject*, ze které jsou odvozeny třídy pro jednotlivé portály *Window* a *Doors*. O navazování portálu na rodičovskou zeď při změně pozice středu portálu se stará třída *WallObjectGeometry*.

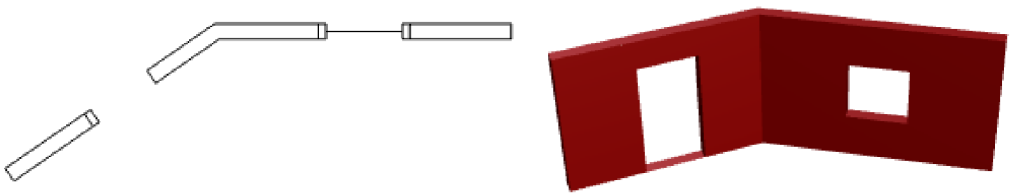


Obrázek 5.6: Strom modelu zdi s portálem

Bázovou třídou pro generování trojrozměrného modelu portálu je třída WallObject-Geometry3D, která zajišťuje výpočet významných bodů, jež jsou společné pro oba druhy portálů. Jedná se o krajní body základny portálu, které je možné určit ze šířky portálu a tloušťky zdi, a krajní body horní části portálu, jež se určí z předchozích bodů doplněných o výšku rodičovského patra.

Z této třídy jsou pak odvozeny třídy, které generují geometrii specifickou pro daný druh portálu, jsou to třídy WallObjectGeometry3D a DoorsGeometry3D. Zbylé body modelu dveří jsou určeny z jejich výšky. U okna je třeba vzít v úvahu ještě informaci o výšce okna nad podlahou.

Popis generovaných polygonů je uveden v kapitole 3.2. Každý zde popisovaný polygon je reprezentován dvěma trojúhelníky, v cílovém modelu, ke kterým jsou vytvořeny příslušné normály. Modely portálů jsou zobrazeny na obrázku 5.7, kde je vlevo vidět půdorys a vpravo vygenerovaný model.



Obrázek 5.7: Modely portálů

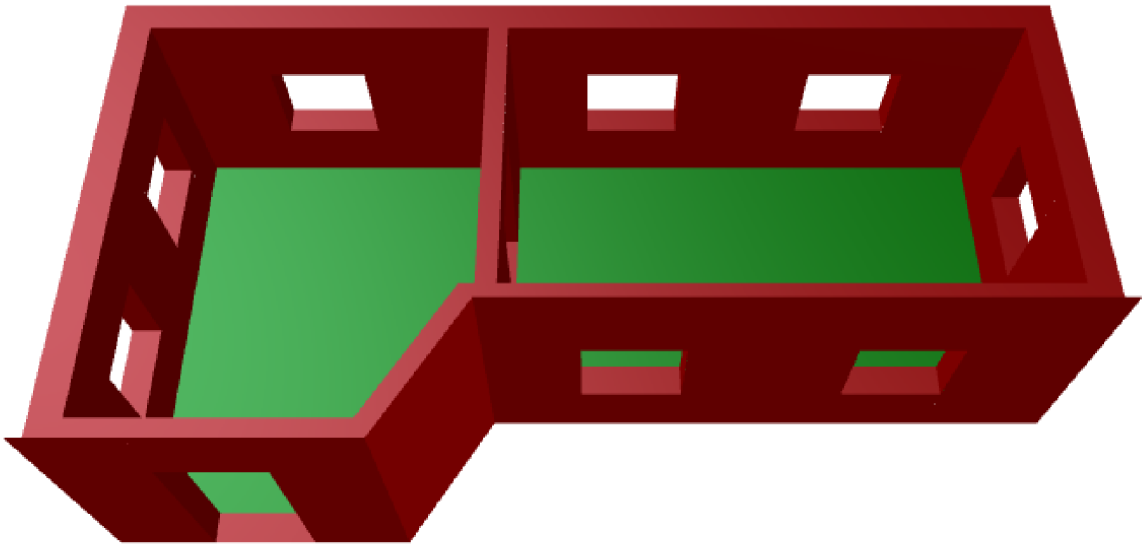
5.5.3 Podlahy a stropy

Tvorba podlah a stropů je založena na vyhledávání cyklů v grafu segmentů zdí, jak bylo popsáno v kapitole 2.5. Pro tento účel si každý segment udržuje referenci na předcháze-

jící a následující segment v tomto grafu. Cykly jsou pak vyhledávány v těchto seznamech segmentů.

Tyto seznamy jsou aktualizovány třídami, které se starají o napojování jednotlivých zdí na sebe. Změny referencí v tomto seznamu jsou zachytávány třídou *CycleGeometry*. Tato třída také zachytává změny na vlastnostech, které nesou informace o pozici zdi. Tyto události jsou důležité pro zdi, které mají na ostatní navázány pouze jeden konec.

Všechny tyto změny jsou spravovány třídou *CycleManager*. Tato třída se stará o vytváření, rušení a aktualizaci existujících cyklů. O vytváření cyklů se stará třída *CycleFactory*. Tato třída pro daný segment prozkoumá, zda je součástí cyklu, a v případě že ano, pak vytvoří odpovídající druh cyklu. Třída *CycleManager* poté provede na základě typu cyklu aktualizaci již existujících cyklů, vytvoří novou podlahu a strop, případně vytvoří střechu budovy.



Obrázek 5.8: Model budovy s podlahou

Bázovou třídou pro cykly je třída *Cycle*, z ní jsou poté odvozeny třídy pro specifické druhy cyklů a to *OuterCycle*, *InnerCycle* a *BoundaryCycle*. V cyklu je udržován seznam segmentů, které ho tvoří. Tato informace je důležitá pro aktualizaci cyklů na základě změny některého segmentu. Dále je zde pak udržován polygon, který tento cyklus tvoří, ten je využíván pro řezání vnějších polygonů vnitřními a pro vytváření podlah a stropů.

Třída představující vnější cyklus vytváří podlahu a strop, reprezentované třídami *Bottom* a *Ceiling*, které vkládá jako synovské uzly do patřičného patra. V případě zániku cyklu je pak ze stromu modelu odstraní. Rodičovskou třídou třídy *Bottom* je třída *PolygonShape*, třída *Celing* je odvozena od třídy *Bottom*.

Na třídě Bottom jsou naimplementovány operace pro vytvoření díry v polygonu a její zacelení. Pro tyto operace je udržován vnitřní seznam všech polygonů, které představují vnitřní díry. V případě přidání nové díry se polygon díry přidá do seznamu a nad existujícím polygonem se provede operace rozdílu s novým polygonem díry. Pro zacelení díry se polygon odebere ze seznamu děr a následně se v původním polygonu bez děr vytvoří znovu všechny díry. V původní implementaci byla pro zacelení použita operace sjednocení nad polygony.

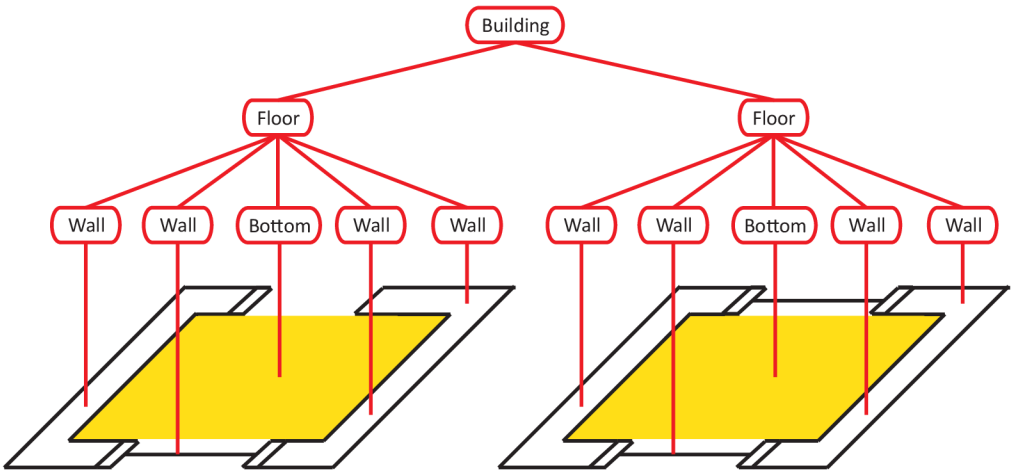
Tato operace však v některých případech zanechávala v polygonu neexistující díry. Z tohoto důvodu byl zvolen výše popsaný postup, i když je výpočetně náročnější. Pro operace nad polygony a jejich triangulaci je použita knihovna General Polygon Clipping Library. Model obsahující vytvořené podlahy je vidět na obrázku 5.8.

Třída *Ceiling* rozšiřuje třídu *Bottom* pouze o otočení orientace vytvořeného polygonu a jeho posun do patřičné výšky. Třída *PolygonShape* poskytuje rozhraní pro vykreslení vygenerovaného polygonu. Pro určení orientace cyklu byla použita část knihovny *Clipper* získaná z [24].

Popsané řešení umožňuje při editaci modelu v externím editoru jednoduchým způsobem aplikovat různé materiály na jednotlivé podlahy a stropy.

5.5.4 Podlaží

Všechny předcházející objekty se vkládají jako synovské uzly do právě editovaného patra, které je reprezentováno třídou *Floor*. Podlaží si nese informace o své výšce nad základnou budovy a výšku stropu. Dále obsahuje instanci třídy *CycleManager*, která byla popsána dříve v této kapitole. Jednotlivá patra jsou vkládána jako synovské uzly do uzlu *Building*, který reprezentuje kořen modelu budovy. Zjednodušený strom modelu budovy je zobrazen na obrázku 5.9, ve kterém jsou vynechány podstromy uzlu *Wall*, jež byly představeny dříve v této kapitole.



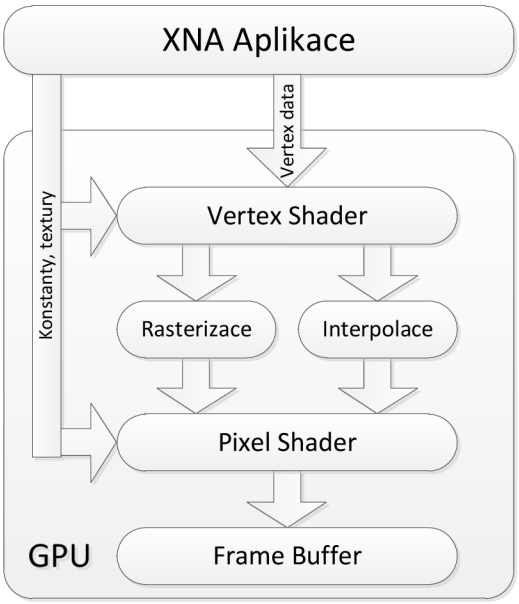
Obrázek 5.9: Strom modelu budovy

Výška nad základnou podlahy je použita pro vytvoření transformace pro synovské uzly, které jsou vytvářeny na úrovni základny a do potřebné výšky transformovány právě touto transformací. Pomocí výšky stropu a obvodového tvaru získaného pomocí obvodového cyklu jsou určeny segmenty zdi, pro které je potřeba upravit vygenerovaný polygon tak, aby navazoval na další patro.

5.6 Vizualizace

Pro vizualizaci grafických dat je použita knihovna XNA a dříve popsané propojení této knihovny s technologií WPF. Přenos grafických dat z aplikace na grafickou kartu je realizován pomocí rozhraní, které publikuje třída *GraphicsDevice*. Další data jako jsou matice

transformací, parametry osvětlovacího modelu apod. se přenáší pomocí rozhraní bázové třídy *Effect*. Tato data jsou předána na vstup příslušného shaderu. Zjednodušené schéma přenosu dat mezi aplikací a grafickou kartou je znázorněno na obrázku 5.10.



Obrázek 5.10: Přenos dat v XNA

Hlavní vykreslovací smyčky jsou implementovány v třídách odvozených od *AbstractDrawingBehaviour* a to *DrawingBehaviour2D* pro zobrazení půdorysu budovy a *DrawingBehaviour3D* pro zobrazení trojrozměrného modelu. Kamery pro zobrazování scény implementují rozhraní *ICamera*, které publikuje metody pro pohyb kamery a pro přibližování respektive oddalování kamery od modelu. Jednotlivé druhy kamer jsou popsány dále v této kapitole.

Při zobrazování modelu se vychází z grafu scény, který je tvořen jednotlivými uzly stromu modelu budovy. Z tohoto stromu jsou odfiltrovány uzly, které nesou daný typ geometrie.

5.6.1 2D Geometrie

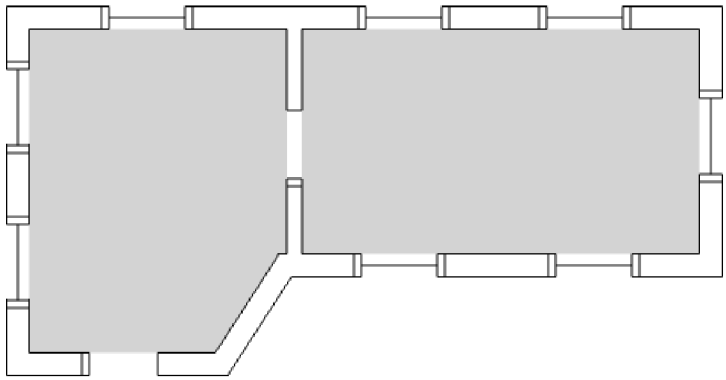
Všechny uzly, které představují část dvojrozměrného modelu půdorysu implementují rozhraní *IDrawableObject*. V tomto rozhraní publikují svoji transformační matici, získanou ze stromu modelu, dále pak druh objektu, který daný uzel představuje, a pořadí, v jakém má být objekt vykreslen. Druh objektu je reprezentován výčtem *Model2D*.

V samotném vykreslovacím cyklu se poté ze stromu daného patra odfiltrují pomocí LINQ dotazu jednotlivé uzly, které implementují toto rozhraní. Před samotným vykreslením je ještě zapotřebí nastavit patřičnou transformační matici ve vykreslovacím řetězci. Pro každý objekt se poté na základě druhu objektu určeného výčtem *Model2D* pomocí továrny *PresentationFactory* získá třída popisující jeho vizuální reprezentaci. Tyto třídy implementují rozhraní *IPresentation*.

V těchto třídách se převede datová reprezentace uzlu na patřičný vizuální objekt. Model zdi je vykreslen pomocí třídy WallSegmentPresentation, tento model představuje pouze jednoduchá čára patřičné délky. Pro okna a dveře jsou tu třídy WindowPresentation a Do-

orsPresentation. Pro podlahu je připravena třída TriangleStripPresentation, která vykreslí triangle strip představující polygon podlahy.

Pro zobrazení půdorysu je použita ortogonální projekce. Tuto projekci představuje kamera popsaná třídou *OrthogonalCamera*. V této třídě jsou implementovány metody pro pohyb kamerou a pro funkci zoom. Výstup vizualizace půdorysu je představen na obrázku 5.11.



Obrázek 5.11: Zobrazení půdorysu budovy

5.6.2 3D Geometrie

Uzly představující trojrozměrné části modelu implementují rozhraní *IDrawableObject3D*. V tomto rozhraní objekty publikují transformační matici a informaci o viditelnosti objektu. Toto rozhraní rozšiřuje rozhraní *ITriangleStrip* a *ITriangleList*. Každé z těchto rozhraní reprezentuje jeden typ geometrie pro vykreslení.

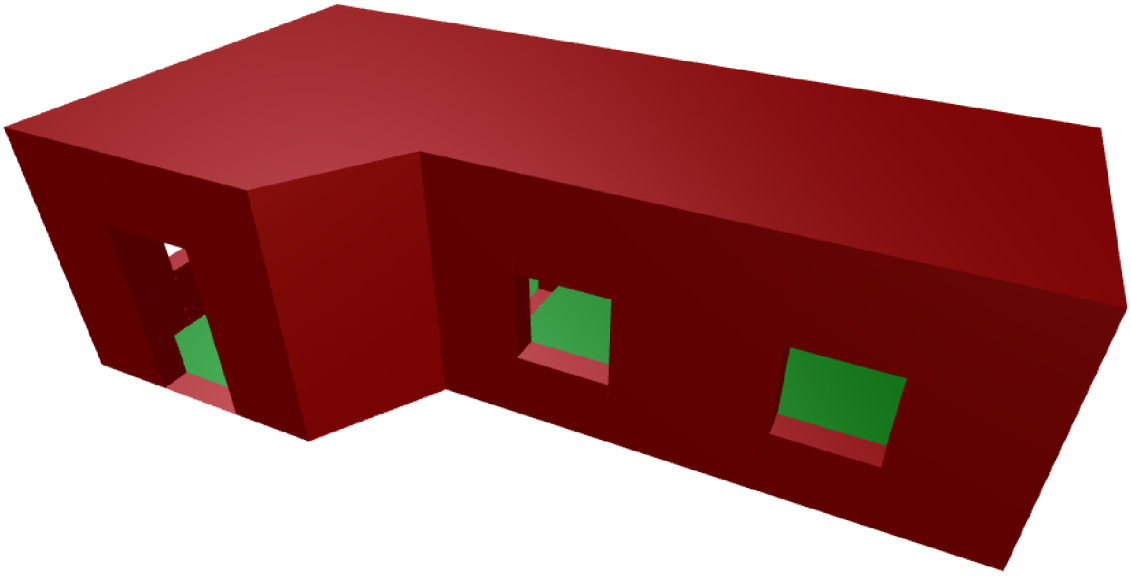
Ve vykreslovacím cyklu se ze stromu modelu odfiltrují pomocí LINQ dotazu objekty, které implementují rozhraní *IDrawableObject3D*. Tyto objekty se poté předají továrně *PresentationFactory3D*, která na základě jejich typu vytvoří objekt, který umí tento druh geometrie zobrazit. Jedná se o objekty typu *TriangleStripPresentation* a *TriangleListPresentation* implementující již zmiňované rozhraní *IPresentation*.

Pro zobrazení modelu budovy je použita perspektivní projekce. Tuto projekci představuje kamera popsaná třídou *OrbitCamera*. V této třídě jsou implementovány metody pro pohyb kamerou okolo modelu a pro funkci zoom. Výstup vizualizace modelu budovy je ukázán na obrázku 5.12.

5.7 Export do Collada

Základem exportéru do formátu Collada jsou třídy popisující datový model tohoto formátu. Třídy datového modelu byly získány z [6]. Tento balíček byl jeho autorem rozšířen o několik pomocných tříd, jež usnadňují práci s datovou reprezentací Collady. Všechny jeho třídy mají prefix *Grendgine_Collada*.

Třídy popisující formát Collada byly vygenerovány pomocí nástroje XML Schema Definition Tool (více informací viz [19]), který je součástí vývojového balíku .NET. Jednou z jeho funkcí je právě vygenerování datového modelu popsaného XML schématem. Pro samotný export je však použito pouze zlomku tříd tohoto komplexního datového modelu.



Obrázek 5.12: Zobrazení modelu budovy

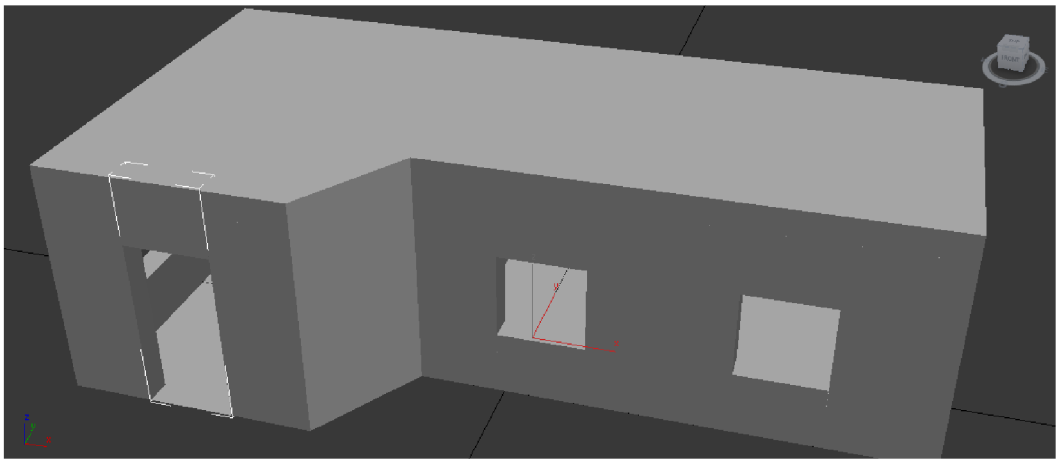
Získaný model bylo potřeba upravit, protože při testování měla aplikace Blender problémy s jeho importem. Problém byl způsoben tím, že Blender zpracovává vstupní Collada soubor sekvenčním přístupem a v tomto modelu bylo v některých případech prohozeno pořadí elementů oproti pořadí předpokládaném parserem Blenderu. Problém vyřešila jednoduchá záměna pořadí jednotlivých vlastností, které představovaly problematické elementy.

Základ exportu je implementován v třídě *ExportToColladaCommand*. V této třídě se převádí jednotlivé druhy 3D uzlů ze stromu na pole prvků typu float, se kterými pracuje Collada. Pro uzly typu *ITriangleList* se přímo vytvoří dvě pole, kde jedno představuje vrcholy a druhé normály exportovaných vrcholů. U uzlů typu *ITriangleStrip* se geometrie převádí také na geometrii typu triangle list, protože při přímém exportování geometrie triangle strip docházelo k problémům s importem v jiných aplikacích. Z tohoto seznamu se poté vytvoří opět dvě pole představující vrcholy a normály. Během převodu vrcholů a normál do výstupního pole je provedena jejich patřičná transformace podle transformační matice uzlu.

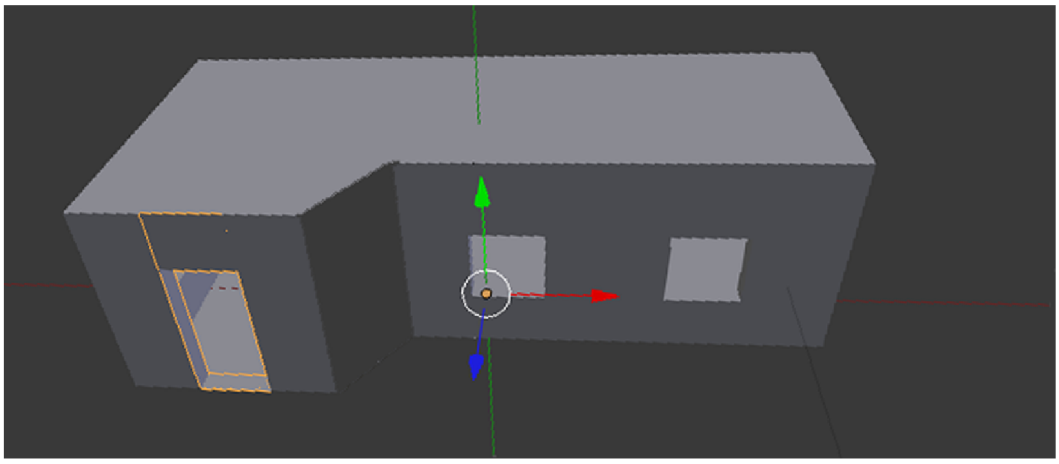
Vytvořená pole vrcholů a normál se dále předají ke zpracování třídě *ColladaBuilder*. Tato třída zajišťuje mapování takto získaných dat na jednotlivé třídy datového modelu Collady. V tomto místě se také provádí otočení orientace jednotlivých trojúhelníků, které je v Collada opačné než v XNA. Toto otočení je řešeno převrácením indexů do pole vrcholů a normál. Pro každý uzel grafu modelu je vytvořen jeden uzel geometrie a jeden uzel ve scéně Collada souboru. Tento postup umožňuje jednoduché editování jednotlivých objektů modelu po importu jakýmkoliv editorem, který podporuje tento formát.

Výsledné uložení výstupního souboru je realizováno obyčejnou XML serializací kořenového elementu *Grendgine_Collada*.

Na obrázku 5.13 můžeme vidět model importovaný do aplikace 3D Studio max. Obrázek 5.14 pak ukazuje model importovaný do aplikace Blender.



Obrázek 5.13: Model importovaný do aplikace 3D Studio Max



Obrázek 5.14: Model importovaný do aplikace Blender

5.8 Ukládání dat

Pro ukládání editovaných dat byla zvolena binární serializace, která představuje nejsnazší přístup k ukládání dat v prostředí .NET. Z tohoto důvodu jsou všechny třídy datového modelu a třídy, které se starají o generování geometrie, označeny atributem *Serializable*. Nevýhodou tohoto přístupu je to, že takto uložená data je prakticky nemožné načíst mimo prostředí .NET, nicméně díky implementaci exportu do formátu Collada nebyla tato nevýhoda považována za problém. Další nevýhodou tohoto přístupu je, že v případě změny datového modelu nemusí být možné deserializovat dříve uložená data.

5.9 Kompatibilita

Aplikace byla vyvíjena a testována převážně na systému Windows 7. Nicméně byla testována i na systémech Windows Vista a Windows XP. V případě Windows XP se mohou vyskytnout komplikace, které jsou blíže popsány v kapitole zabývající se uživatelským rozhraním. Na ostatních testovaných systémech se problémy s kompatibilitou neobjevily.

Na cílovém systému musí být nainstalován .NET Framework ve verzi 4.0 (nestačí verze client profile). Dále musí být přítomny knihovny XNA ve verzi 4.0, které je možné získat z balíku XNA Framework Redistributable 4.0. Všechny tyto prerekvizity by měl automaticky nabídnout k instalaci vytvořený instalační program.

Zdrojové kódy byly vytvářeny v prostředí Visual Studio 2010. Překladové prostředí tohoto vývojového nástroje bylo rozšířeno o knihovnu PostSharp a Async CTP. Instalace těchto knihoven je nezbytná pro úspěšné přeložení zdrojových kódů. Dále je zapotřebí, aby překladové prostředí obsahovalo XNA sdk ve verzi 4.0 a Silverlight sdk ve verzi 5.0. Další knihovny třetích stran jsou získány prostřednictvím NuGet balíčků, z tohoto důvodu je tedy vhodné, aby překladové prostředí obsahovalo i toto rozšíření, případně je nutné stáhnout tyto knihovny ručně a dodat jejich reference do nastavení jednotlivých projektů.

Kapitola 6

Grafické uživatelské rozhraní

Tato kapitola se zabývá návrhem a tvorbou uživatelského rozhraní editoru. Grafické uživatelské rozhraní bylo vytvořeno pomocí technologie WPF. Aplikace je tvořena jako okenní aplikace pro systémy Windows. Při tvorbě byla použita knihovna Ribbon pro vytvoření jednotlivých panelů nástrojů a menu, která jsou v aplikaci použita.

Základní rozhraní aplikace tvoří panel nástrojů a dvě plátna pro vykreslování grafiky. Jedno plátno umožňuje kreslení vlastního půdorysu budovy a druhé slouží k zobrazení vygenerovaného trojrozměrného modelu. Jednotlivé prvky hlavního okna jsou popsány na obrázku 6.1.

Celé rozhraní je implementováno pomocí návrhového vzoru *Command*. Použití tohoto přístupu umožňuje zcela oddělit implementaci uživatelského rozhraní od implementace jednotlivých akcí. Prvky uživatelského rozhraní pouze informují nižší vrstvu o tom, že nastala nějaká událost, ale veškerá implementace reakcí na tyto události je pouze v této nižší vrstvě, která je nazávislá na uživatelském rozhraní. Jednotlivé implementace těchto akcí jsou uloženy v *IoC* kontejneru *Castle Windsor* popsaném v kapitole 4.4.1.

Časově náročné akce jako je ukládání, otevírání souborů nebo export jsou zpracovávány asynchronně s hlavním gui vláknem. Díky tomu nedochází k zamrzání uživatelského rozhraní během těchto akcí. Asynchronní operace jsou implementovány pomocí knihovny Async CTP popsané v části 4.4.5.

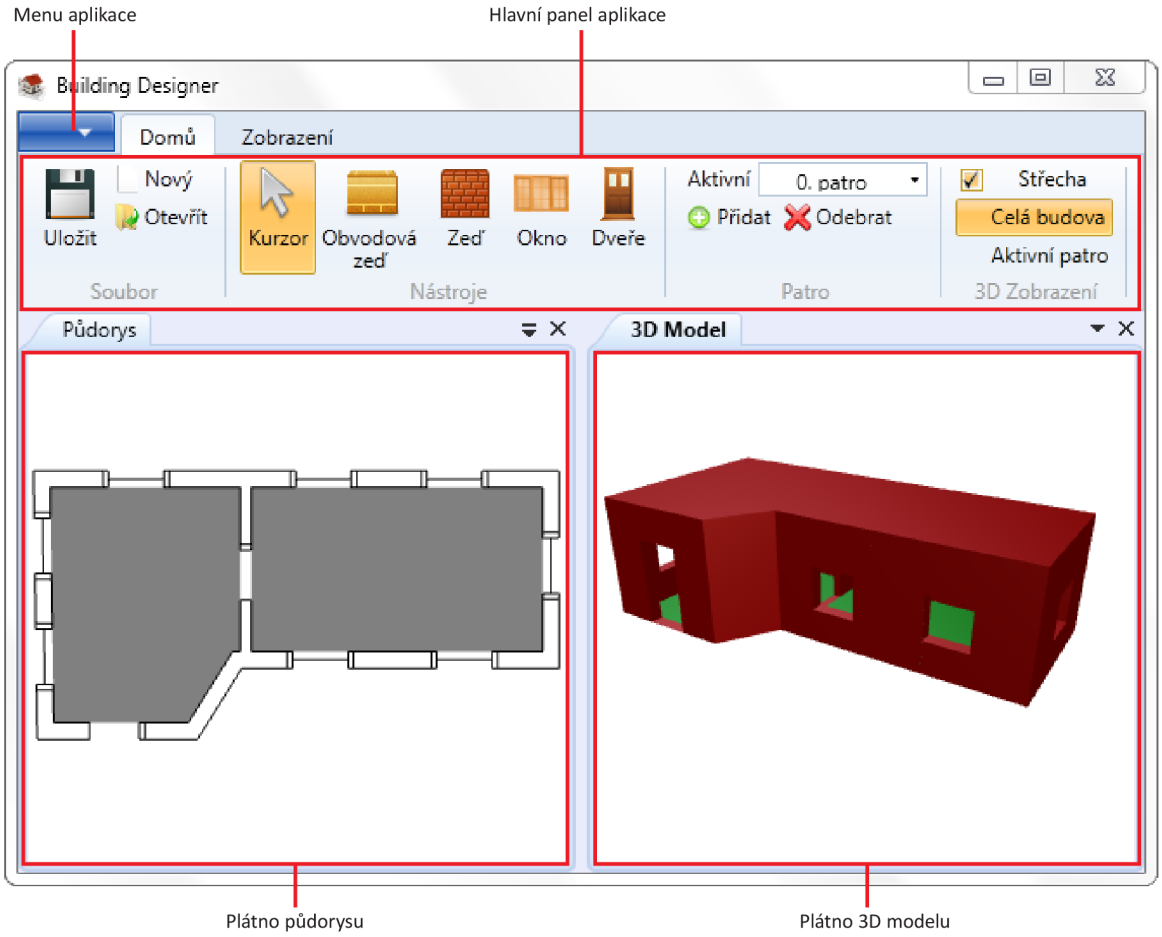
Přestože podle oficiální dokumentace má být prostředí WPF plně kompatibilní se systémy Windows XP, tak se mohou na některých těchto systémech projevit problémy s kompatibilitou. Pozorovány byly nedostatky při vykreslování pomocí XNA a při práci s dokovacím systémem.

6.1 Hlavní okno aplikace

Základ aplikace tvoří hlavní okno reprezentované třídou *MainWindow*, které je odvozené od bázové třídy *RibbonWindow*. Tento druh okna rozšiřuje klasické WPF okno právě o podporu panelů Ribbon.

Hlavní panel, který obsahuje přístup k hlavním funkcím aplikace, je realizován pomocí knihovny *Ribbon for WPF* popsané v kapitole 4.2.2. Panely typu Ribbon patří dnes k často používanému stylu grafického uživatelského rozhraní na platformě Windows.

Jednotlivé panely jsou implementovány jako samostatné prvky uživatelského rozhraní. Tyto panely reprezentují třídy *HomeTab* a *ViewTab*. Hlavní panel reprezentovaný třídou *HomeTab* poskytuje přístup k nejdůležitějším funkcím aplikace jako je ukládání a otevírání



Obrázek 6.1: Uživatelské rozhraní aplikace

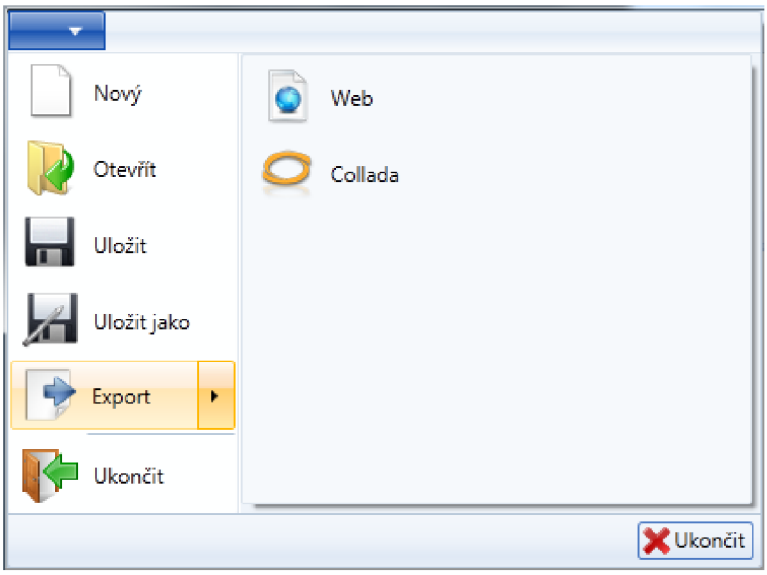
souborů, jednotlivé nástroje pro kreslení nebo nastavení vykreslování modelu budovy. Panel zobrazení, který je implementován v třídě *ViewTab*, poskytuje přístup k funkcím jako je nahrání obrázku na pozadí, podle kterého je možné kreslit půdorys, nebo znovuotevření zavřených pláten.

Další část tvoří hlavní menu aplikace, které poskytuje přístup k práci se soubory a také k dialogu pro export modelu do webové aplikace, nebo do formátu Collada. Toto menu je zobrazeno na obrázku 6.2.

6.2 Plátna pro kreslení

Plátna pro kreslení jsou vytvořena pomocí prvku *XnaCanvas*, který byl popsán v předchozí kapitole. Obě plátna jsou umístěna v hlavním okně pomocí dokovací knihovny *Avalon-Dock* popsané v kapitole 4.2.3. Tato knihovna umožňuje měnit způsob, jakým jsou plátna umístěna v hlavním okně, případně plátno vytáhnout do samostatného okna. Tato knihovna dovoluje lépe využít systémy s více monitory.

Hlavním problémem ovšem zůstává fakt, že použitá knihovna obsahuje chyby, které občas způsobí nemožnost dokování obsahu na potřebné místo. Dalším problémem je, že v případě vytažení plátna do samostatného okna se mohou projevit problémy s výkonem.



Obrázek 6.2: Hlavní menu aplikace

Tento problém je způsoben tím, že knihovna vyžaduje značné množství komunikace mezi takto nově vytvořeným oknem a dokovacím prvkem v hlavním okně. Protože každé okno běží ve vlastním vlákně, je zapotřebí synchronizace při této komunikaci a tím může dojít i k možným výkonostním problémům.

Tyto nedostatky by měly být opraveny v příští verzi knihovny, nicméně v době tvorby aplikace nebyla tato verze ve stavu, kdy by ji bylo možné použít. I přes popsané problémy však tato knihovna patří k nejlepším volně dostupným implementacím dokovacího systému pro WPF.

6.3 Uživatelský vstup

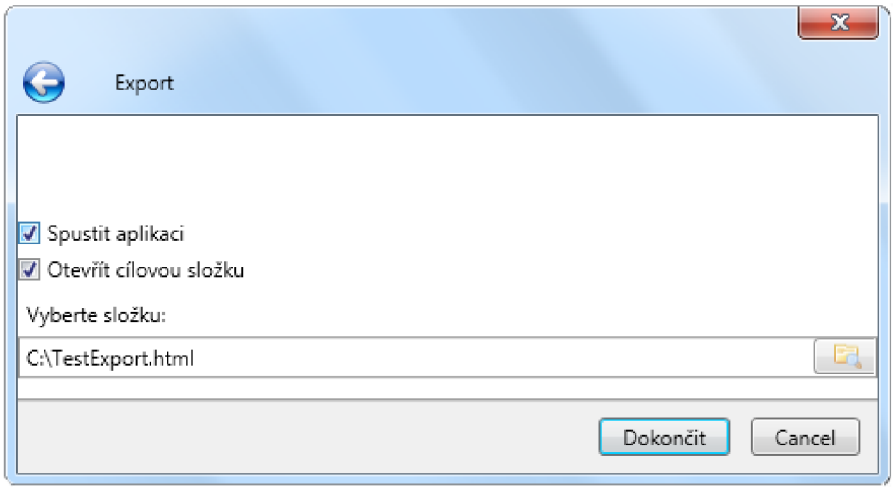
Pro zpracování vstupů od uživatele jsou nad plátny odchytávány události myši a klávesnice, na základě kterých jsou pak spuštěny příslušné akce, které představují přidání prvků do stromu modelu apod. Odchytávání těchto událostí se děje ve třídě *Input*. Přes rozhraní této třídy je možné nastavit obsluhu pro jednotlivé události, jako je stisk tlačítka myši, pohyb myší, případně stisknutí klávesy na klávesnici. Pro každé plátno existuje jedna instance této třídy. Přístup k nim je zprostředkován pomocí třídy *InputManager*.

Jednotlivé akce jako je kreslení zdi, přidání okna, dveří apod. si poté při své inicializaci nastaví patřičné obsluhy jednotlivých událostí, vyvolaných vstupem uživatele. V těchto akcích se poté souřadnice, které jsou předávány v souřadném systému okna, v případě potřeby převedou do souřadného systému scény. Tento převod zajišťuje třída *CoordinateConverter*.

6.4 Dialog pro export

Dialog pro export byl vytvořen pomocí knihovny *AvalonWizard*, která byla popsána v kapitole 4.2.4. V tomto dialogu je možno nejdříve vybrat export do webové aplikace nebo do formátu Collada. Dále už je jen zapotřebí vybrat cílový soubor. Samotný export je prováděn v samostatném vlákně, aby nedocházelo k blokování hlavního hlavního vlákna aplikace.

Implementace dialogu se nachází ve třídě *ExportWizard*. Tato třída představuje okno, v němž se nachází jediný prvek, kterým je průvodce z knihovny *AvalonWizard*. V tomto prvku jsou vytvořeny dvě vnitřní stránky. Na první je možné vybrat, zda se bude exportovat do webové aplikace nebo do formátu Collada. Na druhé se nachází prvky pro výběr cílového souboru a zaškrtávácí tlačítka, která umožní otevření cílové složky pro export, nebo přímé spuštění webové aplikace. Dialog exportu do webové aplikace je ukázán na obrázku 6.3. Kontrolka pro výběr cílového souboru byla získána z [31].



Obrázek 6.3: Dialog pro export webové aplikace

Kapitola 7

Tvorba webové aplikace

Webová aplikace je implementována pomocí technologie Silverlight, která byla popsána v kapitole 4.1.2. Při volbě technologie vhodné pro webovou aplikaci byla analyzována i technologie WebGL. Nicméně tato technologie není dostupná v prohlížečích Internet Explorer, který stále představuje poměrně často používaný prohlížeč. Další nevýhodou je nemožnost sdílení částí zdrojových kódů se zbytkem aplikace. Z těchto důvodů byla zvolena technologie Silverlight, která je podporována ve všech hlavních prohlížečích na platformě Windows. Hlavní nevýhodou zvolené technologie je fakt, že není dostupná na jiných systémech než je Windows a Mac OS.

7.1 Vizualizace

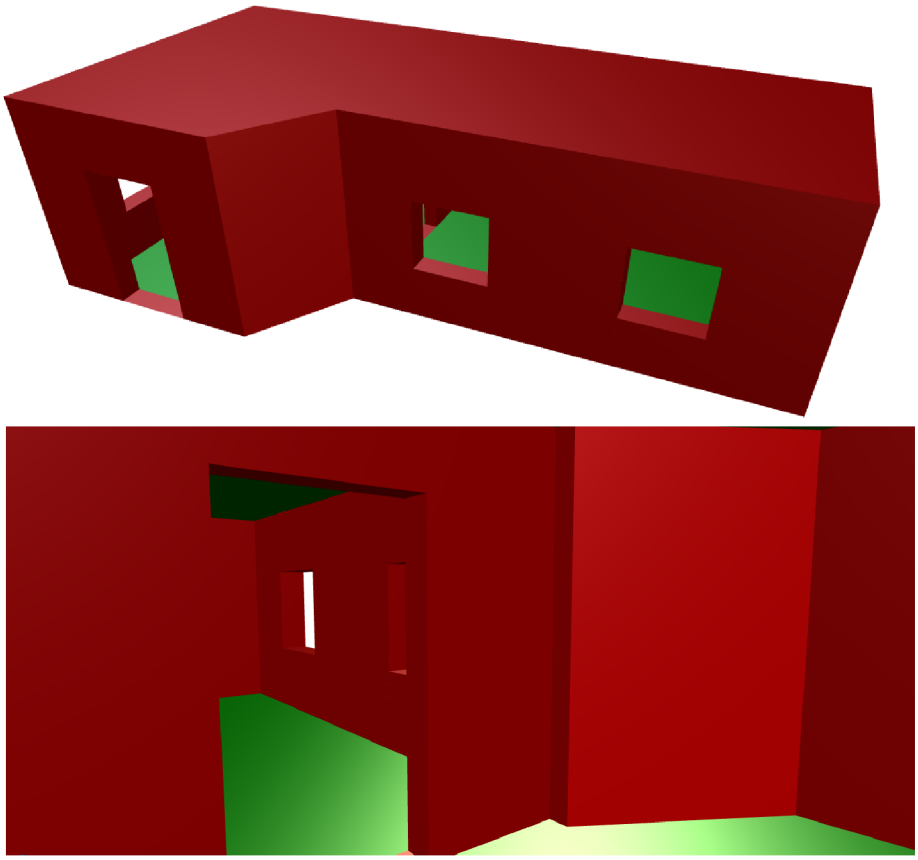
Zobrazení modelu budovy je implementováno pomocí technologie XNA. Technologie XNA přítomná v technologii Silverlight obsahuje podmnožinu funkcionality dostupné v plném XNA, které bylo použito pro implementaci vizualizace v hlavní aplikaci.

Vykreslování je realizováno pomocí prvku *DrawingSurface*, který umožňuje vykreslování pomocí XNA. Vykreslovací smyčka je realizována zachytáváním události *Draw* na tomto prvku. V obsluze této události se provede vykreslení scény a invalidace plátna, čímž se zajistí nekonečná vykreslovací smyčka. Smyčka pro aktualizaci scény je realizována pomocí časovače, který má nastavenou periodu na 25 milisekund. Tato hodnota se ukázala jako dostatečná pro plynulou aktualizaci scény.

Samotné vykreslování a aktualizaci scény zajišťuje třída *DrawingController*, která je obdobou dříve popsané třídy *DrawingBehaviour* v hlavní aplikaci. Při vykreslování se projde seznam všech objektů, které reprezentují části modelu budovy. Pro každý z nich se pomocí třídy *PresentationFactory3D* získá implementace rozhraní *IPresentation*, která umí daný objekt vykreslit. Tento princip je stejný jako vykreslování modelu v hlavní aplikaci. Kód jednotlivých tříd byl převzat z hlavní aplikace a případně drobně poupraven pro použití v prostředí Silverlight.

Pro zobrazení scény jsou implementovány dvě kamery, mezi kterými může uživatel libovolně přepínat. První kamera umožňuje pohyb okolo modelu a je reprezentována třídou *OrbitCamera*, jejíž kód byl částečně převzat z implementace v hlavní aplikaci. Budova zobrazená pomocí této kamery je vidět na obrázku 7.1 nahoře. Druhá kamera umožňuje plynulý průchod modelem budovy. Tato kamera je reprezentována třídou *FreeCamera*. Výstup zobrazený pomocí této kamery je ukázán na obrázku 7.1 dole.

Pohyb kamerou pomocí myši je realizován na základě událostí myši na vykreslovacím



Obrázek 7.1: Zobrazení modelu budovy

plátně. Pohyb pomocí klávesnice je implementován v obsluze aktualizace scény. V této metodě se při každé aktualizaci scény kontroluje stav klávesnice a případně se provede patřičný posun pozice kamery.

7.2 Importovaná data

Grafická data jsou do webové aplikace importována z editoru. Jedná se pouze o data samotného modelu budovy, která jsou reprezentovaná pomocí geometrie typu triangle list nebo triangle strip. I ve webové aplikaci je model budovy reprezentován jako skupina samostatných objektů, které mají transformaci, jež určuje pozici objektu v modelu.

Grafická data jsou reprezentována bázovou třídou *DrawableObject3D*, která poskytuje přístup k transformační matici objektu. Z ní jsou poté odvozeny třídy *TriangleList* a *TriangleStripImpl*, které představují jednotlivé druhy geometrie. Třída *TriangleStripImpl* je pomocná třída, která zapouzdřuje kolekci tříd typu *TriangleStrip*. Takto reprezentovaná grafická data jsou vykreslena postupem, který byl popsán v předchozí části.

Další importovanou částí je seznam pater budovy, který umožňuje uživateli přepínat mezi jednotlivými patry při průchodu budovou. Z editoru se exportuje pouze výška patra nad zemí a jeho název. Takto popsané patro reprezentuje třída *FloorSimple*. Postup přenosu informací o patrech z hlavní aplikace do webové je stejný jako níže popsaný postup pro přenos grafických dat.

7.3 Export z hlavní aplikace

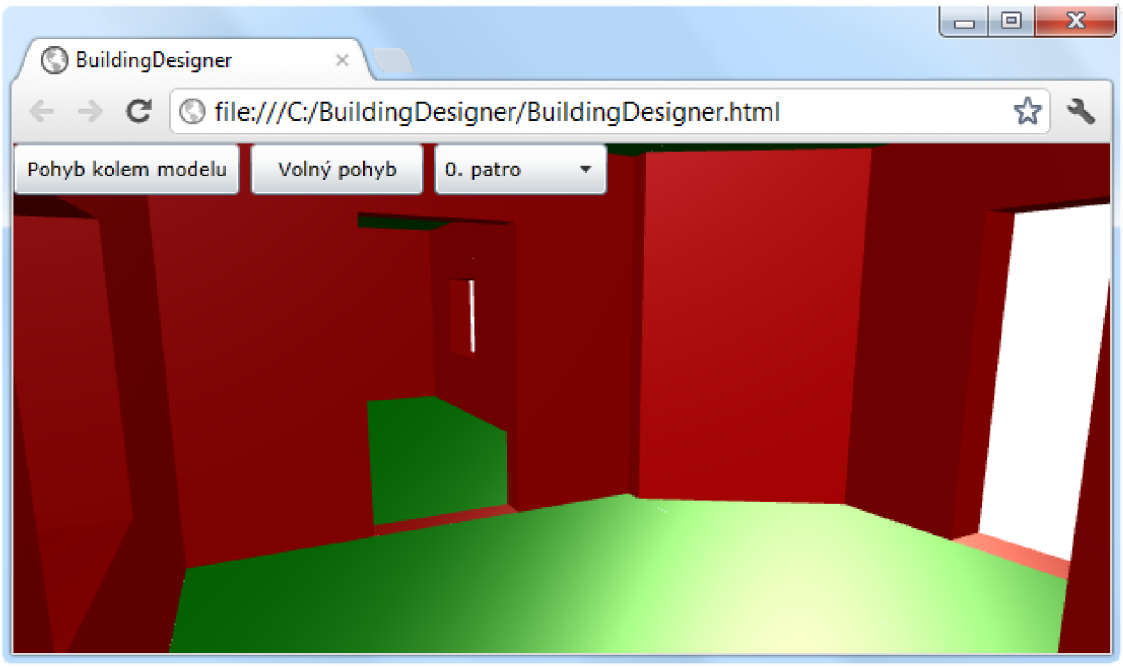
Při exportu grafických dat se projde strom modelu a pro jednotlivé uzly, které nesou 3D geometrii, se vytvoří obalové třídy, které obsahují pouze tato grafická data. Tento datový model je totožný s datovým modelem použitým ve webové aplikaci, který byl popsán v předchozí části. Výsledné datové objekty se pomocí serializace uloží do paměťového streamu.

Pro importování těchto dat do webové aplikace je možné se z běžící webové aplikace připojit na server a data si stáhnout. Nicméně toto řešení má jednu nevýhodu, protože na cílové doméně musí být dostupný soubor, který popisuje cross domain pravidla pro přístup (tzv. cross domain policies). Jedná se o podobný přístup, který je použit u technologie Flash. Z tohoto důvodu byl zvolen postup, který je popsán níže.

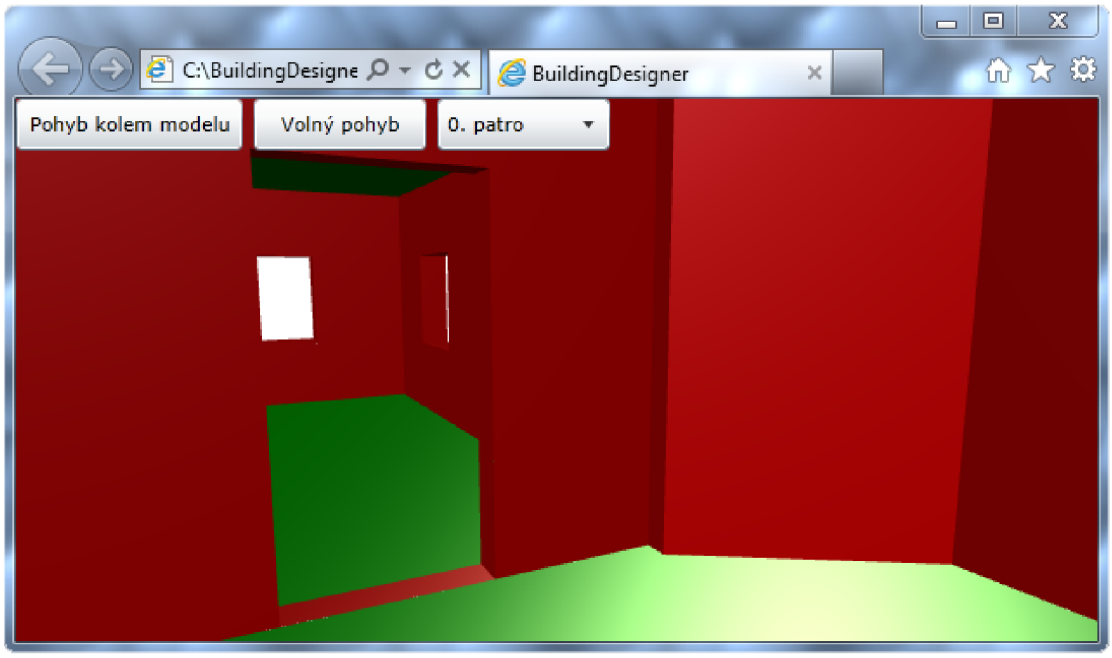
Aplikace vytvořená pomocí technologie Silverlight je reprezentována pomocí xap balíčku. Tento balíček představuje obyčejný archiv ve formátu zip, v němž jsou uloženy všechny potřebné dll knihovny a další soubory potřebné pro běh aplikace. Z tohoto balíčku lze načítat data a přistupovat k souborům, aniž by došlo k porušení bezpečnostních pravidel, která v technologii Silverlight zabraňují v přístupu na pevný disk hostitelského počítače, nebo pravidla pro přístup na web popsaná výše.

Tohoto je využito při exportu dat do webové aplikace, kdy hlavní aplikace zapíše serializovaná data uložená v memory streamu do xap balíčku připravené webové aplikace. Ta si pak při spuštění tato data ze svého balíčku přečte a deserializuje do datových objektů, které může použít pro vykreslení modelu. Pro přístup ke xap balíčku je využita knihovna DotNetZip popsaná v kapitole 4.4.4.

Nevýhodou tohoto řešení je situace, kdy se změní kód webové aplikace. V tomto případě není možné snadno převést již vygenerované aplikace na novou verzi. Řešením by bylo nové vyexportování z hlavní aplikace, případně by bylo možné napsat program, který by převedl stávající aplikace na nové verze programově. Možným řešením je i ruční rozbalení xap balíčku, nahrazení potřebných souborů a následné zabalení zpět do archivu.



Obrázek 7.2: Zobrazení modelu budovy ve webovém prohlížeči Google Chrome



Obrázek 7.3: Zobrazení modelu budovy ve webovém prohlížeči Internet Explorer

7.4 Zobrazení ve webovém prohlížeči

Výsledná aplikace se do webové stránky vkládá pomocí tagu object podobně jako například Flash aplikace. Dále je nutné přidat parametr, který zapne podporu hardwarové akcelerace. Při exportu z hlavní aplikace je vytvořena ukázková html stránka, která ukazuje, jak je možné aplikaci do html stránky vložit.

Při načtení stránky v prohlížeči je nutné v nastavení pluginu Silverlight přidat zobrazovanou stránku mezi důvěryhodné, aby bylo možné zapnout hardwarovou akceleraci. Toto nastavení je součástí bezpečnostních opatření platformy Silverlight.

Uživatelské rozhraní webové aplikace obsahuje pouze dvě tlačítka pro přepnutí kamery a rozbalovací menu pro přepnutí patra při průchodu budovou. Po kliknutí do okna aplikace dojde k aktivování vykreslovacího plátna a je možné se pohybovat po budově pomocí kurzorových šipek. Při držení levého tlačítka myši je poté možno otáčet pohledem kamery. Výsledná aplikace je ukázána na obrázku 7.2 a 7.3.

Kapitola 8

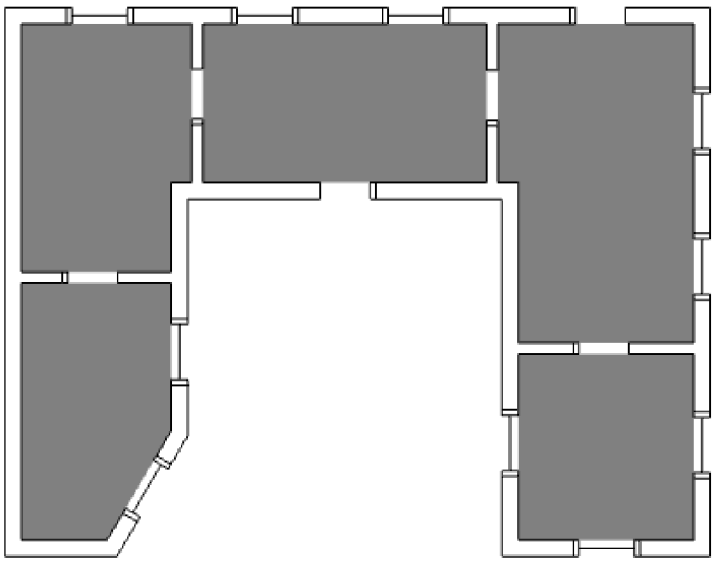
Dosažené výsledky

V této kapitole jsou předvedeny dosažené výsledky, omezení zvoleného řešení a známé chyby. Výsledky jsou prezentovány pomocí obrázků získaných z vytvořené aplikace.

8.1 Vygenerovaný model

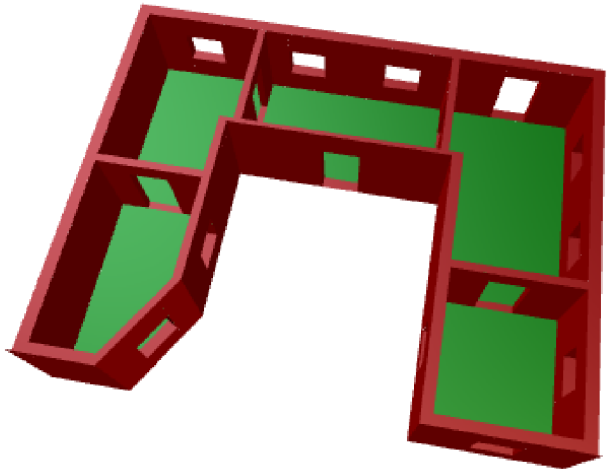
Výstupem aplikace je trojrozměrný model budovy bez aplikovaných textur případně dalších materiálů. Webová aplikace umožňuje zobrazení a průchod tímto vygenerovaným modelem.

Na obrázku 8.1 je vidět vytvořený půdorys patra budovy. Z obrázku je patrné použití obvodových zdí na vytvoření obrysu budovy a vnitřních zdí na vytvoření jednotlivých pokojů. Do půdorysu je umístěno několik portálů. Tmavé části v půdorysu označují vygenerované podlahy budovy.



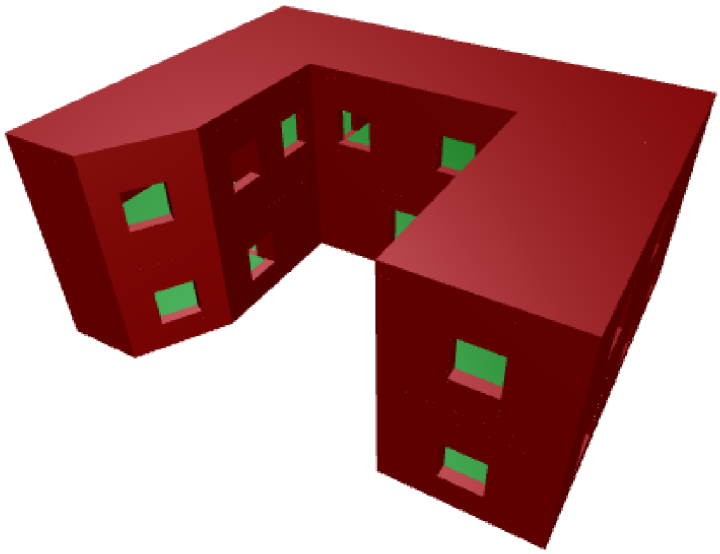
Obrázek 8.1: Půdorys patra budovy

Obrázek 8.2 zobrazuje trojrozměrný model budovy vytvořený z tohoto půdorysu. Na tomto obrázku jsou vidět vytvořené modely pro portály a jednotlivé druhy zdí. Zelenou barvou jsou potom vykresleny podlahy v jednotlivých místnostech.



Obrázek 8.2: Vygenerovaný model patra budovy

Na obrázku 8.3 je zobrazen model celé budovy tvořený dvěma patry. Z tohoto obrázku je patrné použití společného obrysu pro všechny patra budovy.

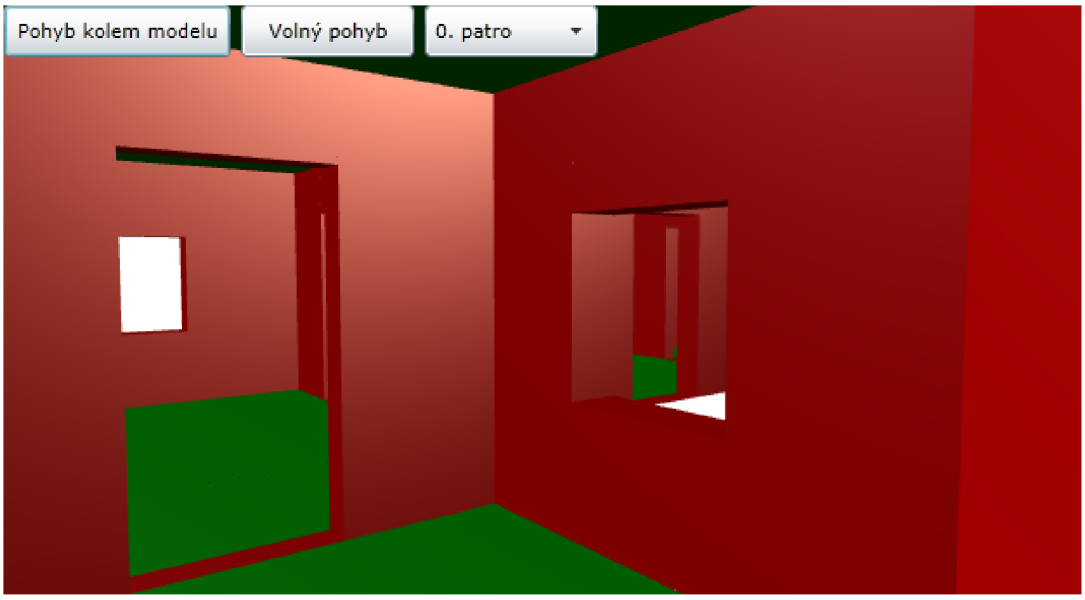


Obrázek 8.3: Vygenerovaný model budovy

Obrázek zobrazuje 8.4 průchod budovou ve vytvořené webové aplikaci a obrázek 8.5 zobrazení celého modelu pomocí této aplikace. Tyto obrázky byly vytvořeny v prohlížeči Google Chrome.

8.2 Spolupráce s dalšími aplikacemi

Díky exportu do formátu Collada je možné model importovat do jiných aplikací, které tento formát podporují.



Obrázek 8.4: Průchod modelem v internetovém prohlížeči

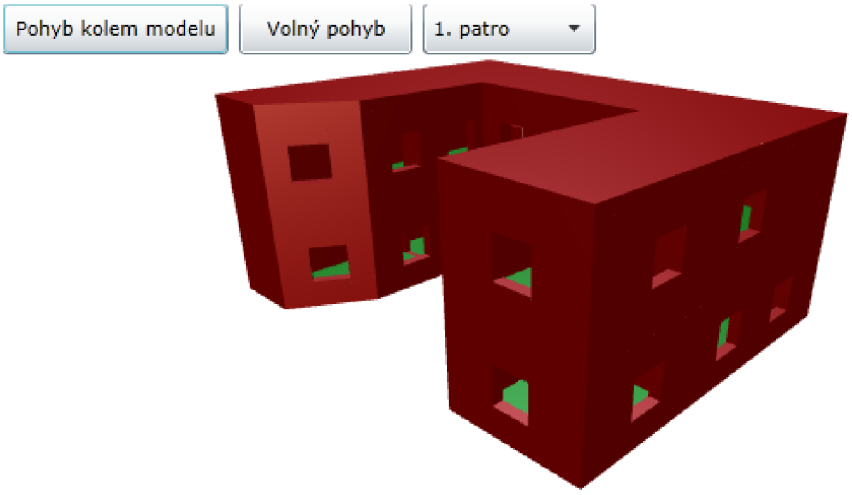
Model budovy upravený v externím editoru je zobrazen na obrázku 8.6. V tomto případě byl model upravován v aplikaci 3D Studio Max, kde na něj byly aplikovány materiály pro podlahu a zdi, dále pak byly doplněny modely dveří a oken.

Na obrázku 8.7 je vidět model vytvořený v aplikaci Lexcoad vizualizovaný pomocí prohlížeče modelů Yeti. V této aplikaci byly do modelu přidány materiály a další objekty jako nábytek a schodiště.

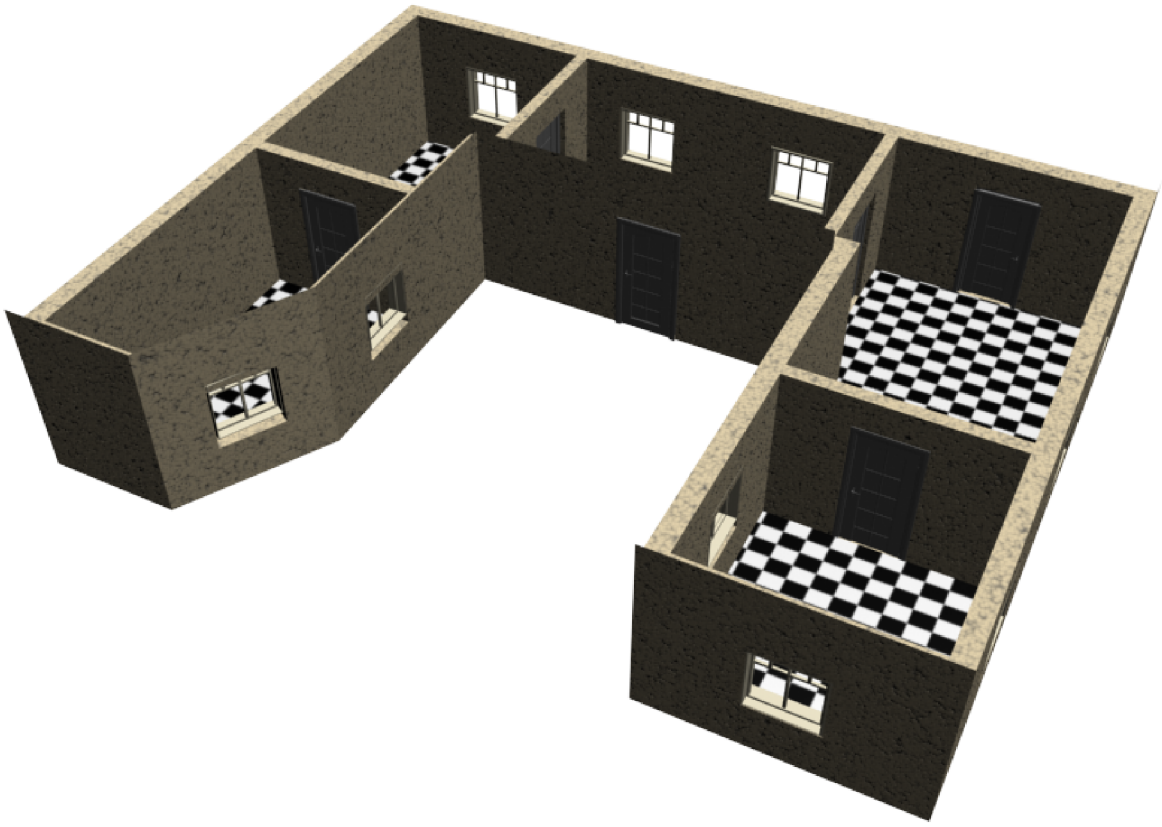
8.3 Omezení zvoleného řešení

Prvním omezením zvoleného postupu pro generování trojrozměrného modelu je nutnost vytvářet všechna patra se stejným obrysem. V případě kdy mají patra různý tvar je třeba řešit složitější případy propojení mezi patry než jak bylo ukázáno v kapitole 3.4. Rozdíl mezi těmito dvěma řešeními je znázorněn na obrázku 8.8, kde je vlevo vidět implementované řešení a vpravo řešení potřebné pro patra s různými tvary. Díky tomuto omezení se značně zjednodušila implementace návazností mezi jednotlivými patry, avšak znemožňuje vytvářet budovy členitějších tvarů.

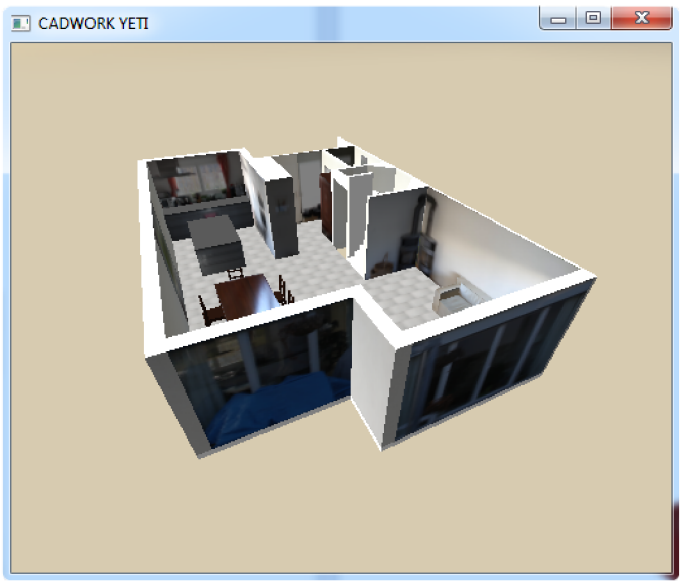
Dalším omezením je nemožnost vytvoření budovy s vnitřním nádvořím. Tento případ je vidět na obrázku 8.9, kde je půdorys budovy, která má pomocí obvodové zdi vytvořené vnitřní nádvoří. Z tohoto půdorysu je patrné, že uprostřed půdorysu v místě nádvoří je vygenerována podlaha. Na obrázku 8.10 je zobrazen model vygenerovaný pro tento půdorys, v němž je na místě nádvoří běžná místnost. Toto omezení je způsobeno postupem, pomocí kterého se vyhledávají jednotlivé polygony podlah v modelu. Navržený a implementovaný postup počítá pouze s jednou souvislou obvodovou zdí, která tvoří vnější obrys budovy.



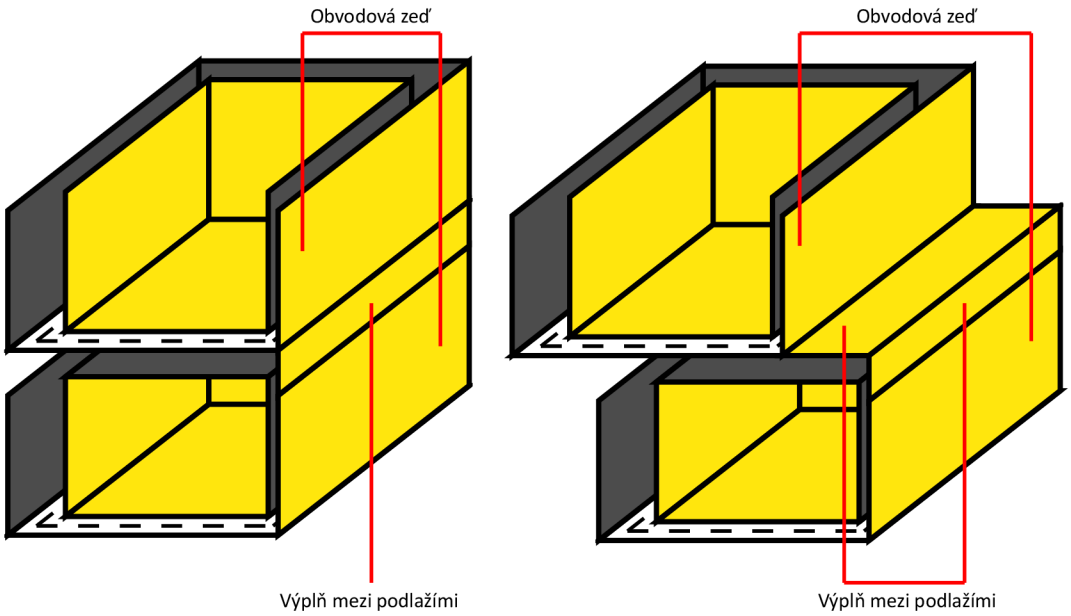
Obrázek 8.5: Zobrazení modelu v internetovém prohlížeči



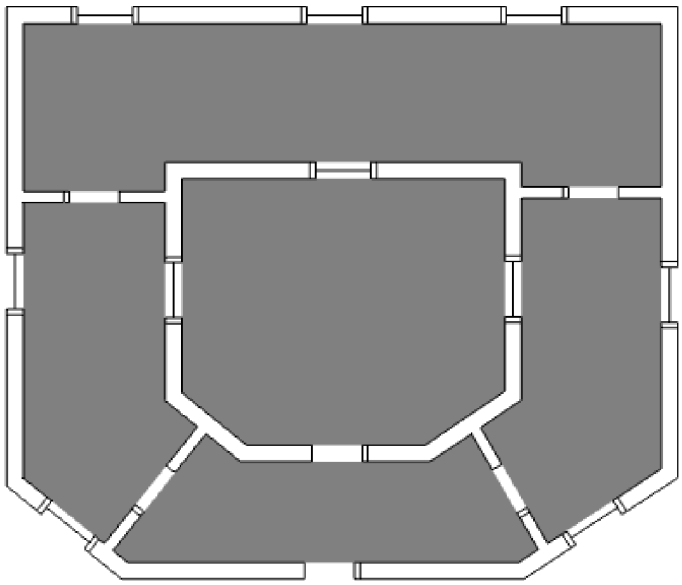
Obrázek 8.6: Model upravený v externím editoru



Obrázek 8.7: Model upravený v externím editoru



Obrázek 8.8: Tvorba výplně mezi podlažími

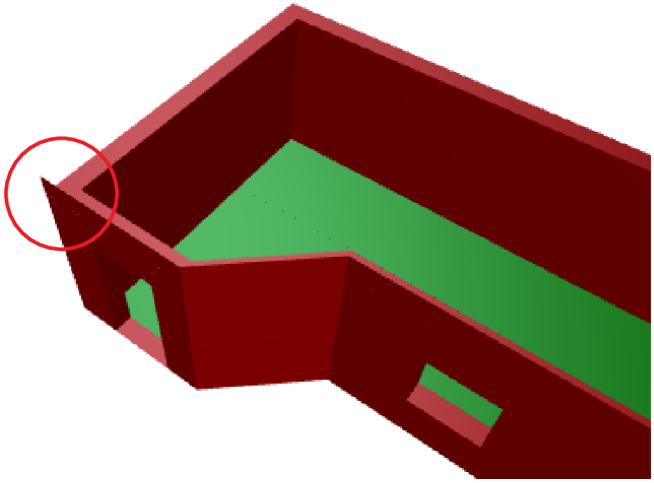


Obrázek 8.9: Půdorys budovy s vnitřním nádvořím

Obrázek 8.10: Model budovy s vnitřním nádvořím

8.4 Známé chyby a nedostatky

Při prohlížení jednotlivých pater je při pohledu na obvodovou zeď vidět převýšení vnější strany obvodové zdi. Tento jev je způsoben prodloužením tohoto polygonu o výšku stropu patra, které slouží k propojení jednotlivých pater. Převýšení je patrné pouze u polygonů, které jsou orientované směrem k pozorovateli, protože odvrácené polygony nejsou vůbec vykreslovány. Tento jev je patrný z obrázku 8.11.

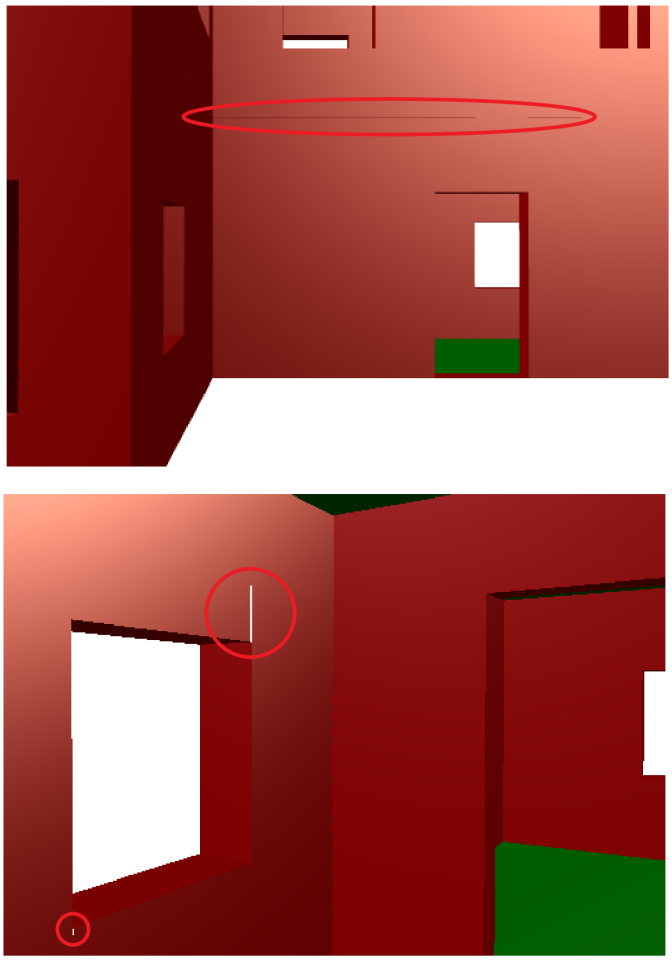


Obrázek 8.11: Propojení mezi patry budovy

V některých případech vznikají v modelu artefakty, které je možno vidět na obrázku 8.12. Tento jev by mohl být způsoben zvolenou datovou reprezentací modelu. Při výpočtech jednotlivých částí modelu je třeba převádět mezi souřadnými systémy jednotlivých částí modelu, jak bylo popsáno v kapitole 5.5. Při těchto transformacích dochází k zaokrouhlování výsledků, které mohou způsobit pozorované nepřesnosti v modelu.

Při vytváření půdorysu v jistých případech dojde k odstranění existujících podlah v modelu. V této situaci je zapotřebí odstranit zdi, při jejichž propojování tato chyba vznikla, případně zdi na ně navazující. Tato chyba je zřejmě způsobena nekorektním upravováním seznamu navazujících segmentů zdí, které bylo popsáno v kapitole 5.5.1. Podobná chyba může nastat v případě, kdy se kolem vytvořeného modelu vnitřních zdí vytvoří obvodová zeď. V implementaci se počítalo s postupem, kdy se vytvoří nejdříve obvodová zeď a poté teprve vnitřní struktura budovy. V době dokončování projektu nebyla ani jedna z těchto chyb uspokojivě vyřešena. Obecně je část kódu starající se o vytváření podlah a navazování jednotlivých segmentů na sebe patrně nejrizikovější částí z hlediska implementačních chyb. Důvodem je fakt, že výpočty jsou závislé na uživatelském vstupu pomocí myši a případné chyby jsou obtížně laditelné a zachytitelné pomocí testů.

Dále jsou známy dva výkonnostní problémy aplikace. První z nich nastává při otevírání uložených souborů a je způsoben neúměrnou dobou deserializace uložených objektů. Tento problém může být způsoben nevhodně zvoleným datovým modelem, který se ukládá, nebo známým problémem .NET deserializace v případě vnořených kolekcí. Další problém může nastat ve chvíli, kdy se v modelu nachází velký počet segmentů zdí. Tento problém je zapříčiněn způsobem, kterým se vyhledávají jednotlivé segmenty v modelu.



Obrázek 8.12: Artefakty ve vytvořeném modelu

Kapitola 9

Závěr

Tato práce se v úvodních částech zabývá návrhem postupů pro generování trojrozměrných modelů budov z jejich půdorysů. V dalších částech je poté popsán postup implementace editoru pro tvorbu půdorysů budov a generování jejich modelů. Dále je pak uveden postup implementace internetové aplikace pro prohlížení takto vytvořených modelů. V závěrečné části jsou shrnuty dosažené výsledky a popsány nedostatky zvoleného řešení. Z těchto částí vyplývá, že bylo splněno celé zadání práce.

Mezi hlavní přednosti a přínosy vytvořené aplikace patří schopnost exportu do formátu Collada a přímá publikace na web. Export do formátu Collada lze použít jako vstup dalších aplikací, jež slouží pro úpravu trojrozměrných modelů. Publikace na web umožňuje interaktivní průchod vytvořeným modelem ve webovém prohlížeči. Hlavní nevýhodou jsou omezení na tvar vytvářené budovy, které vyplynuly ze zvoleného postupu generování modelu, a chyby, které se nepodařilo odladit před odevzdáním projektu. Po rozšíření funkcionality editoru se předpokládá využití aplikace ve spolupráci se softwarem Lexocad.

Vhodným rozšířením implementované aplikace by bylo rozšíření funkcionality editoru pro vytváření půdorysů, jako je přichytávání k mřížce, zobrazení vodítek k jednotlivým částem modelu apod. Dále by bylo vhodné rozšířit uživatelské rozhraní o prostředí, v němž by bylo možné upravovat parametry vytvořených trojrozměrných objektů, jako je jejich výška, šířka apod. Dále by bylo vhodné vylepšit vizuální podobu zobrazovaného modelu o aplikaci materiálů na jednotlivé části budovy. Mezi další vhodná rozšíření patří možnost vkládat modely (okna, dveře, nábytek apod.) vytvořené v jiném editoru do vytvářené scény. Internetovou aplikaci by bylo vhodné rozšířit o možnost navigace vytvořenou budovou.

Během implementace se ukázalo, že volba technologie WPF pro implementaci uživatelského rozhraní byla velmi vhodná, protože tato technologie umožňuje jednoduchou a efektivní tvorbu těchto rozhraní. Jediným problémem této technologie bylo propojení s XNA, nicméně i tento problém se podařilo uspokojivě vyřešit. Jako velká výhoda se ukázal fakt, že je dostupné velké množství knihoven, které implementují klíčové části uživatelského rozhraní.

Technologie XNA poskytuje velice jednoduché a příjemné rozhraní pro vizualizaci trojrozměrných scén. Avšak pro vizualizaci dvojrozměrných scén již tak vhodná není, protože jí chybí některé důležité funkce jako vykreslení základních geometrických primitiv apod. Z tohoto důvodu by pro implementaci části pracující s půdorysem zřejmě bylo vhodnější zvolit knihovnu více orientovanou na aplikace typu CAD. Na druhou stranu díky použití knihovny XNA bylo možné velice jednoduše přenést vizualizaci modelu do internetového prohlížeče.

Jako velice přínosné se ukázaly knihovny CastleWindsow a PostSharp, které značně zre-

dukovaly množství potřebného kódu a pomohly i k čistšímu návrhu aplikace. Jako velice užitečná se ukázala také technologie LINQ dostupná v .NET Framework, která umožnila jednoduše realizovat operace nad stromem modelu. Obecně se zvolené technologie a knihovny ukázaly jako vhodné pro řešení tohoto problému.

Literatura

[1] 3ds Max - 3D Modeling, Animation, and Rendering Software - Autodesk. [online], [cit. 2012-05-09].
URL http://usa.autodesk.com/3ds-max/

[2] Apache log4net. [online], poslední změna: 24. listopadu 2011. [cit. 2012-05-09].
URL http://logging.apache.org/log4net/

[3] ArchiCAD 15 - Overview. [online], [cit. 2012-05-09].
URL http://www.graphisoft.com/products/archicad/

[4] Avalon Wizard. [online], poslední změna: 30. ledna 2011. [cit. 2012-05-09].
URL http://avalonwizard.codeplex.com/

[5] AvalonDock. [online], poslední změna: 20. února 2012. [cit. 2012-05-09].
URL http://avalondock.codeplex.com/

[6] C# Collada 1.5 Classes. [online], poslední změna: 13. ledna 2011. [cit. 2012-05-09].
URL http://sourceforge.net/projects/csharpcollada/

[7] Castle Windsor - Castle Project. [online], [cit. 2012-05-09].
URL http://docs.castleproject.org/Windsor.MainPage.ashx

[8] Chief Architect Interior Design Software for Professional Interior Designers. [online], [cit. 2012-05-09].
URL http://www.chiefarchitect.com/

[9] COLLADA - COLLADA. [online], poslední změna: 18. července 2007. [cit. 2012-05-09].
URL https://collada.org/mediawiki/index.php/COLLADA

[10] DotNetZip Library. [online], poslední změna: 31. července 2011. [cit. 2012-05-09].
URL http://dotnetzip.codeplex.com/

[11] The Khronos Group Inc. [online], [cit. 2012-05-09].
URL http://www.khronos.org/

[12] LINQ (Language-Integrated Query). [online], [cit. 2012-05-09].
URL http://msdn.microsoft.com/en-us/library/bb397926.aspx

[13] Microsoft Office 2010 — Microsoft Visio 2010. [online], [cit. 2012-05-09].
URL http://www.microsoft.com/cze/office2010/produkty/visio.aspx

[14] PostSharp. [online], [cit. 2012-05-09].
URL http://www.sharpcrafters.com/postsharp/features

[15] Ribbon (WPF). [online], [cit. 2012-05-09].
URL http://msdn.microsoft.com/en-us/library/ff799534.aspx

[16] SmartDraw - Communicate Visually. [online], [cit. 2012-05-09].
URL http://www.smartdraw.com/

[17] Sweet Home 3D. [online], poslední změna: 28. dubna 2012. [cit. 2012-05-09].
URL http://www.sweethome3d.com

[18] Visual Studio Asynchronous Programming. [online], [cit. 2012-05-09].
URL http://msdn.microsoft.com/en-us/vstudio/gg316360

[19] XML Schema Definition Tool (Xsd.exe). [online], [cit. 2012-05-09].
URL http://msdn.microsoft.com/en-us/library/x6c1kb0s(v=vs.100).aspx

[20] Caprio, G.: Design Patterns: Dependency Injection. [online], [cit. 2012-05-09].
URL http://msdn.microsoft.com/en-us/magazine/cc163739.aspx

[21] Duhl, J.: *Rich Internet Applications*. IDC, 2003.

[22] Hanselman, S.; Evjen, B.: *ASP.NET 3.5 v jazycích C# a Visual Basic*. Computer Press, první vydání, 2009, ISBN 978-8-025-12069-9.

[23] James, S.: *3D Graphics with XNA Game Studio 4.0*. Packt Publishing, první vydání, 2010, ISBN 978-1-849690-04-1.

[24] Johnson, A.: Clipper - an open source freeware polygon clipping library. [online], poslední změna: 3. května 2012. [cit. 2012-05-09].
URL http://www.angusj.com/delphi/clipper.php

[25] Kashlev, D.; of Technology. Dept. of Electrical Engineering, M. I.; Science, C.: *Efficient 3D building model generation from 2D floor plans*. Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2008.
URL http://books.google.cz/books?id=6Bo8QwAACAAJ

[26] Lewis, R.: Generating Three-Dimensional Building Models from Two-Dimensional Architectural Plans. Technická zpráva, Two-Dimensional Architectural Plans, Computer-Aided Design, 1996.

[27] Martin, R. C.: The Dependency Inversion Principle. [online], [cit. 2012-05-09].
URL http://www.objectmentor.com/resources/articles/dip.pdf

[28] Murta, A.: GPC General Polygon Clipper library from The University of Manchester. [online], poslední změna: 11. prosince 2011. [cit. 2012-05-09].
URL http://www.cs.man.ac.uk/~toby/gpc/

[29] NAGEL, C.; et al.: *C# 2008 Programujeme profesionálně*. Computer Press, první vydání, 2009, ISBN 978-80-251-2401-7.

[30] Stephens, R.: *WPF Programmer's Reference: Windows Presentation Foundation with C# 2010 and .NET 4*. Wiley Publishing, Inc., 2010, ISBN 978-0-470-47722-9.

[31] Sumi, P.: A WPF File Selection control. [online], poslední změna: 14. března 2008. [cit. 2012-05-09].
URL http://www.hardcodet.net/?s=FileSelector

[32] Torgersen, M.: Asynchronous Programming in C# and Visual Basic. [online], 2010, [cit. 2012-05-09].
URL http://www.microsoft.com/en-us/download/details.aspx?id=14058

[33] Whiting, E.; Battat, J.; Teller, S.: S (2007) Topology of Urban Environments: Graph construction from multi-building floor plan data. In *In: Dong A, Moere VA, Gero JS (eds) Computer-Aided Architectural Design Futures 2007. vol XII*, 2007, s. 115–128.

Příloha A

Obsah CD

`bp` adresář obsahuje zprávu ve formátu pdf
`edit` adresář obsahuje zdrojové soubory práce pro LaTeX
`poster` adresář obsahuje plakátek prezentující práci
`source` adresář obsahuje zdrojové kódy celé aplikace
`binary` adresář obsahuje spustitelnou verzi aplikace
`install` adresář obsahuje instalační program aplikace

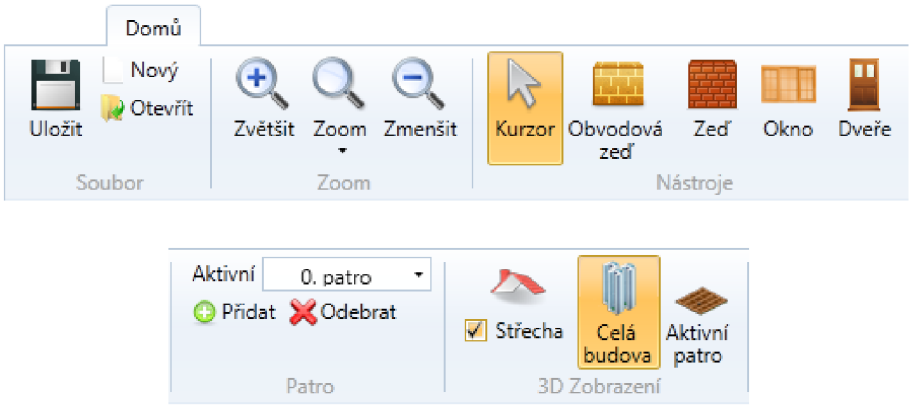
Příloha B

Manual

B.1 Panely nástrojů

V této části jsou popsány panely nástrojů, které zpřístupňují hlavní funkce aplikace.

B.1.1 Hlavní panel



Obrázek B.1: Hlavní panel nástrojů

První skupina tlačítek hlavního panelu (viz. obrázek B.1) představuje akce pro práci se soubory:

- *Uložit* – uložení stávajícího projektu do souboru.
- *Nový* – vytvoření prázdného projektu.
- *Otevřít* – načtení projektu ze souboru.

Následující skupina ovládacích prvků umožňuje zoom nad 2D plátnem, tato funkcionalita je dostupná i pomocí kolečka myši. Jedná se o prvky:

- *Zvětšit* – přiblížení pohledu.
- *Zoom* – zoom na 100%, případně po rozkliknutí na jinou zvolenou hodnotu.

- *Zmenšit* – oddálení pohledu.

Další část umožňuje výběr jednotlivých nástrojů pro vytváření půdorysu. Tyto nástroje fungují pouze pro 2D plátno. Jedná se o nástroje:

- *Kurzor* – umožňuje pohybovat 2D kamerou a vybírat vytvořené objekty, které je možno mazat pomocí klávesy *delete*.
- *Obvodová zeď* – tvorba obvodové zdi, jednotlivé body se zadávají kliknutím levého tlačítka myši. Jednotlivé zdi na sebe automaticky navazují v rozích.
- *Zeď* – tvorba vnitřní zdi, body se zadávají kliknutím levého tlačítka myši.
- *Okno* – vložení okna, vkládá se kliknutím na příslušnou pozici v půdorysu.
- *Dveře* – vložení dveří, vkládají se kliknutím na příslušnou pozici v půdorysu.

Následující sekce umožňuje pracovat s patry budovy.

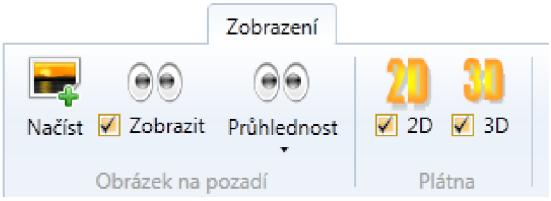
- *Aktivní patro* – umožňuje výběr právě editovaného patra, pro které je zobrazen půdorys.
- *Přidat* – přidá další patro do budovy, zkopíruje do něj obvodové zdi a nastaví jej jako aktivní.
- *Odebrat* – odebere patro z budovy.

V poslední sekci je možné ovlivnit, jakým způsobem se zobrazí model v 3D plátně. Jedná se funkce:

- *Střecha* – povolí nebo zakáže zobrazení střechy budovy.
- *Celá budova* – zobrazí celou budovu.
- *Aktivní patro* – zobrazí pouze aktivní patro.

B.1.2 Panel zobrazení

Tento panel je ukázán na obrázku B.2 a umožňuje přistup k funkcím, které ovlivňují zobrazení aplikace.



Obrázek B.2: Panel zobrazení

První skupina ovládacích prvků umožňuje nastavit obrázek, který je zobrazen jako pozadí 2D plátna. Tento obrázek může sloužit jako předloha vytvářeného plánu. Jedná se o prvky:

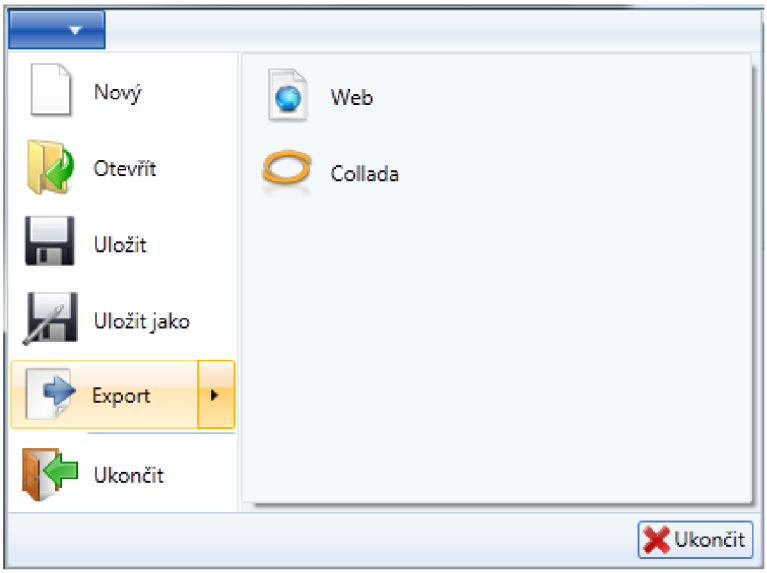
- *Načíst* – umožňuje načíst obrázek, který se zobrazí jako pozadí 2D plátna. Obrázek je možno načíst i za pomoci drag drop souboru s obrázkem na plátno.
- *Zobrazit* – zobrazí nebo skryje načtený obrázek na pozadí.
- *Průhlednost* – nastavení průhlednosti načteného obrázku.

Následující skupina tlačítek umožňuje měnit viditelnost jednotlivých pláten.

- *2D* – zobrazí nebo skryje 2D plátno.
- *3D* – zobrazí nebo skryje 3D plátno.

B.2 Hlavní menu

Hlavní menu poskytuje přístup k operacím se soubory, které byly popsány v předcházející části. Dále umožňuje export modelu do webové aplikace a do formátu Collada. Hlavní menu je zobrazeno na obrázku B.3.

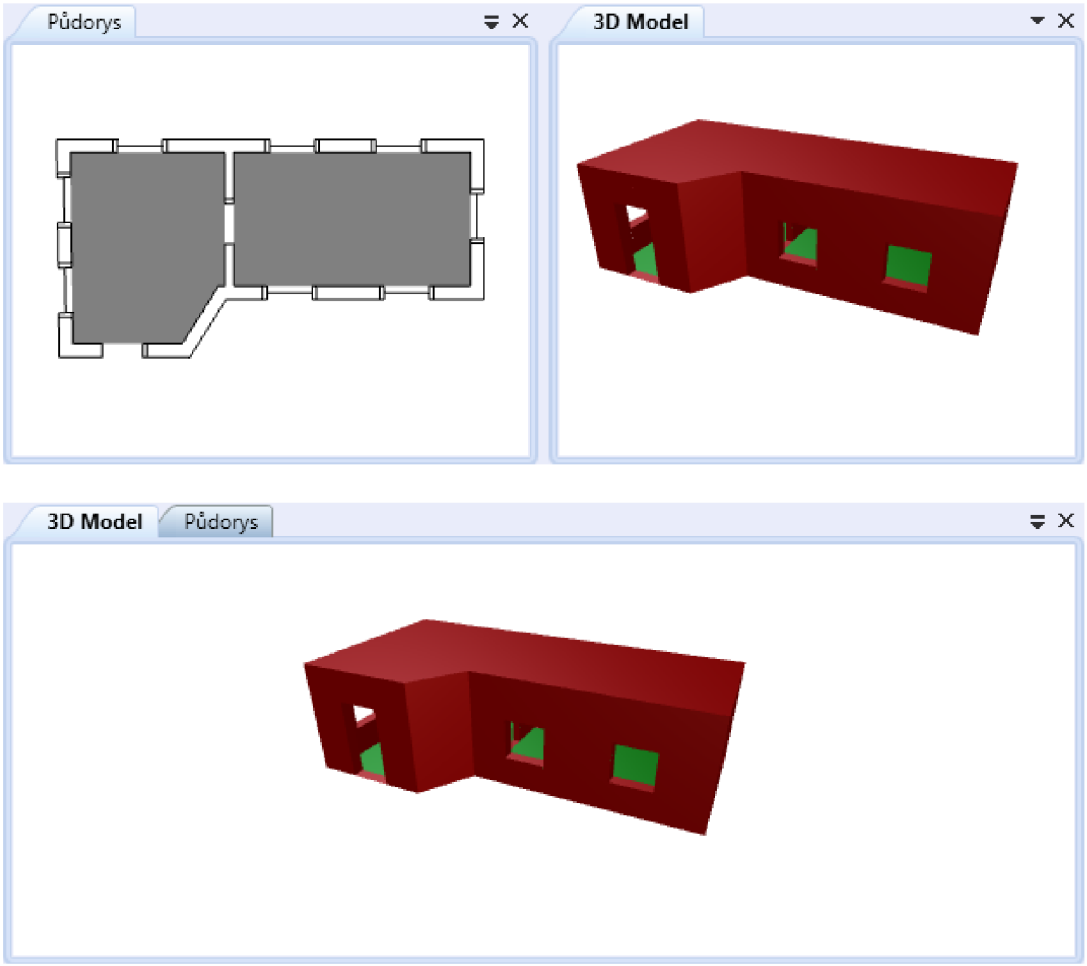


Obrázek B.3: Aplikační menu

B.3 Plátna pro kreslení

V aplikaci jsou přítomna dvě plátna, jedno pro kreslení 2D půdorysu a jedno pro zobrazení vygenerovaného modelu budovy. Obě plátna lze přemisťovat v rámci aplikace pomocí dokovacího systému, který funguje podobně jako například systém v aplikaci Visual Studio. Plátna jsou ukázána na obrázku B.4.

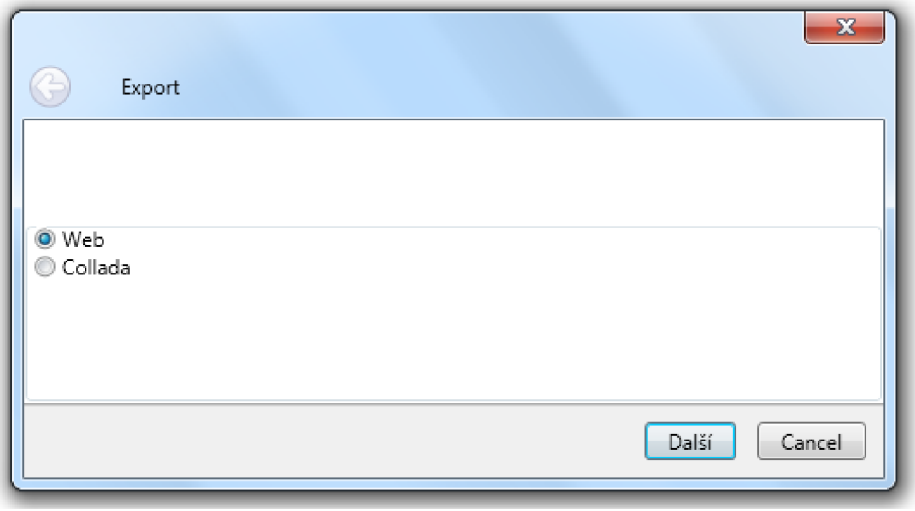
Plátno půdorysu umožňuje na základě vybraného nástroje vkládat jednotlivé prvky do půdorysu. Pokud je zvolen kurzor, je možné držením levého tlačítka myši a pohybem myši pohybovat kamerou. V plátnu pro zobrazení modelu je možné tímto způsobem otáčet kamerou, kolečkem myši se pak přibližovat případně oddalovat od modelu. Pri držení pravého tlačítka myši a pohybem myši je pak možné pohybovat samotným modelem.



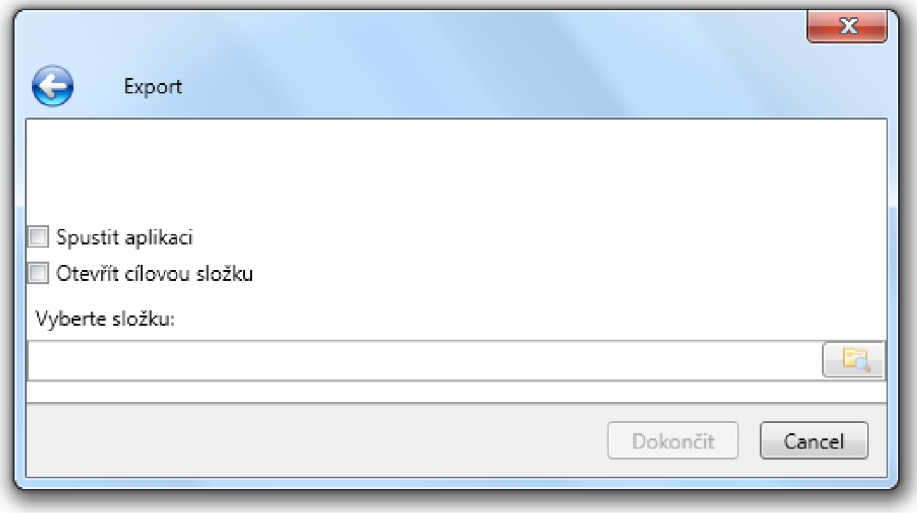
Obrázek B.4: Plátna pro kreslení

B.4 Dialog pro export

V tomto dialogu je možné nejdříve zvolit cílový formát, pokud nebyl vybrán přímo z menu, jak je vidět na obrázku reffig:ManualExport1. Následující krok umožňuje vybrat cílový soubor, do kterého se export provede. Tato část je zobrazena na obrázku B.6. Dále je zde možno zvolit, zda se má po exportu otevřít cílová složka, případně rovnou spustit webová aplikace. V případě webové aplikace se exportuje ukázková HTML stránka a xap balíček s vlastní aplikací.



Obrázek B.5: Výběr formátu pro export



Obrázek B.6: Export do webové aplikace