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

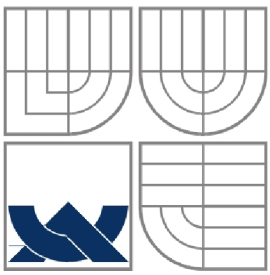
APLIKACE PRO NÁVRH A VIZUALIZACI LEZECKÝCH STĚN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

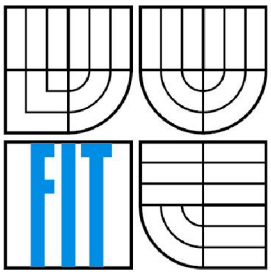
AUTOR PRÁCE
AUTHOR

TOMÁŠ NÁPLAVA

BRNO 2010



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

APLIKACE PRO NÁVRH A VIZUALIZACI LEZECKÝCH STĚN

CLIMBING WALL DESIGN AND VISUALIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ NÁPLAVA

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PAVEL ŽÁK

BRNO 2010

Abstrakt

Práce se zabývá návrhem a realizací programu pro modelování umělých lezeckých stěn. Aplikace, psaná v programovacím jazyku Python, je nadstavbou profesionálního grafického programu Blender. Umožňuje uživateli vytvořit model stěny v krátkém časovém intervalu, aniž by požadovala rozsáhlé znalosti ovládání Blenderu., tento model otexturovat, rozmístit na něj úchyty a doplnit scénu o další příslušenství. Jádrem programu je space handler skript, vyhodnocující události, přicházející od uživatele.

Abstract

This bachelor's thesis deals with design and implementation of a program for climbing wall modeling. Application is written in programming language Python and it is a plug-in for professional graphics program Blender. It allows user to create a model of a wall in rapidly small amount of time, not demanding any deep knowledge of using Blender. Model can be textured, its surface covered with hand holds and the whole scene can be enriched with other accessories. The core of the program consists in space handler script which evaluates events coming from the user.

Klíčová slova

3D modelování, Blender, editace Meshe, lezecká stěna, space handler skript

Keywords

3D modeling, Blender, Mesh editing, climbing wall, space handler script

Citace

Tomáš Náplava: Aplikace pro návrh a vizualizaci lezeckých stěn, bakalářská práce, Brno, FIT VUT v Brně, 2010

Aplikace pro návrh a vizualizaci lezeckých stěn

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Pavla Žáka
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Náplava

Brno, 19. 5. 2010

Poděkování

Tímto bych chtěl poděkovat Ing. Pavlu Žákovi za jeho pomoc při řešení úlohy a rovněž svým rodičům, kteří mi umožnili plně se na tuto práci soustředit.

Obsah

1 Úvod

S oblastí počítačové grafiky se dnes můžeme potkat v různých odvětvích lidské činnosti, kde s úspěchem našla uplatnění. Ať už se jedná o její nasazení ve trojrozměrných hrách či simulátorech, nebo kompresi obrazových dat z digitálního fotoaparátu, přes rendering snímků a tvorbu animovaných filmů až po nástroje pro architektonický návrh, počítačová grafika nás doprovází na každém kroku. Vizualizace výsledků a vytvoření grafického uživatelského rozhraní jsou v oblasti informačních technologií, podobně jako vizualizace v jakékoli jiné oblasti, klíčové prvky, jelikož za 90 % všech získaných informací jsme dlužni právě lidskému zraku.

S ohledem na tuto skutečnost se pokusím vytvořit nástroj, jenž bude sloužit k návrhu a náhledu umělé lezecké stěny.

Lezení na stěnu je samo o sobě výzvou, stěna láká případné zájemce, aby ji pokořili. Ti pak při cestě nahoru svádějí zarputilý boj a jsou omezeni pouze svou fyzickou silou, tělesnou váhou a začátečníci pochopitelně nedostatkem zkušeností či vytrvalostí. Přitom není až tak důležité, aby byl lezec vyložený kulturista, jako to, aby měl tu správnou techniku a také vyvážený poměr mezi svojí hmotností a množstvím síly, kterou je schopen při výstupu vynaložit.

Není proto divu, že se tato činnost těší v posledních letech stále větší oblibě, lezecké stěny mohou být spatřeny v nafukovacím provedení na hudebních festivalech, ve městech vznikají nová sportovní centra specializovaná na tuto sportovní aktivitu. Dokonce Centrum sportovních aktivit VUT podporuje lezení a zve zájemce do kurzu boulderingu, který je s klasickým lezením příbuzný, liší se však v tom, že při něm není použito žádného jištění a lezci stoupají pouze do relativně bezpečných výšek, odkud by pád nebyl zas tak drastický.

Se vznikem těchto nových center vyvstává potřeba stěny nějakým způsobem vizualizovat, aby si mohl jak stavitel, tak i ten, kdo si chce stěnu nechat postavit, mohl udělat představu o tom, jak bude budoucí dílo vypadat a konkretizovat požadavky na jeho provedení.

V následujících kapitolách se zaměřím na specifikaci problému, jeho analýzu a způsoby možného řešení a implementaci.

2 Teorie

2.1 Lezecké stěny

Umělé lezecké stěny jsou překážky postavené pro nácvik horolezců za nepříznivého počasí nebo pro horolezce začátečníky, na lezení na tyto stěny lze ale rovněž pohlížet jako na samostatný sport, oblíbený zejména ve městech. Nejjednodušší stěny jsou zhotoveny z překližky s dírami, využívají se konstrukce z hliníku nebo železa, na které je nanesena vrstva betonu, která dá stěně její finální podobu a poměrně autentický povrch. Co se týče bezpečnosti, sportovci bývají většinou jištěni lany, ale i tak je dole k dispozici žíněnka pro měkčí přistání. K výstupu nahoru pomáhají horolezcům různé úchyty a výčnělky přišroubované ke stěně. Je to právě rozmístění a počet těchto madel spolu s tvarem stěny, které určuje její obtížnost. Pokud je na jedné stěně více drah, rozdílných vůči sobě náročností výstupu, bývají úchyty jedné dráhy barevně odlišeny od ostatních.

Ilustrace 1: lezecká stěna na palubě zaoceánské lodi [obrázek převzat z http://www.cruiseweb.com/rci-imagelibrary/ship-Freedom/400w-RCI_Freedom_RockClimbers.jpg]

2.2 Blender

Při volbě návrhové aplikace jsem měl na výběr si buď naprogramovat vlastní aplikaci, anebo použít existující program a upravit jeho vlastnosti podle potřeby. Jak už název kapitoly napovídá, rozhodl jsem se pro druhou variantu, pro program Blender. Jednak proto, že se jedná o profesionální grafický nástroj pro návrh 3D modelů, jenž umí zacházet s polygonálními modely, implicitními plochami, NURBS plochami, Bezierovými a NURBS křivkami, ale i proto, že je volně šiřitelný a je dostupný na platformách Windows, Linux a Mac OS X.

Vzhledem k tomu, že hodlám použít hotový program, ušetřím si práci, kterou bych jinak vynaložil k naprogramování manipulace se scénou, načítání objektů do scény, vyhodnocení základních událostí od uživatele apod., a můžu se tak plně soustředit na můj vlastní přínos, čímž je obohacení Blenderu o takové prvky, které usnadní tvorbu modelu lezecké stěny.

Rozhraní v Blenderu je rozděleno do několika oken, tím nejdůležitějším z nich je 3D View, které poskytuje paralelní nebo perspektivní projekci scény a objektů do ní zasazených, tyto objekty je možné zde ručně vytvářet a editovat pomocí rozličných nástrojů, dále textový editor pro tvorbu a spuštění uživatelských skriptů, scripts window, kde se zobrazují prvky uživatelského rozhraní vytvořené pomocí skriptu, okno s uživatelským nastavením a jiná, jako třeba okno pro práci se zvuky nebo animacemi, která však pro tvorbu lezeckých stěn nejsou až tak důležitá.

2.3 Python

Blender je implementován v jazyce Python, což je obecný, vyšší programovací jazyk, který spatřil světlo světa v roce 1991. Je požadován za velice čitelný a snadný na pochopení, přesto umožňuje několik stylů programování, jako objektové, imperativní a funkcionální [8].

Python je svou syntaxí podobný C/C++, některé operátory jsou převzaté, zdaleka ale ne všechny. Zdrojový kód psaný v Pythonu je oproti programu v C/C++ na první pohled odlišný v tom, že v něm chybí jakékoli středníky či složené závorky, znaky, které vymezují jednotlivé příkazy či bloky příkazů. Alespoň trochu si ušetříme psaní, protože další příkaz je jednoduše oddělen novým řádkem a žádný středník není potřeba. Bloky příkazů se od sebe odlišují odsazením, tedy příkazy stejného bloku musí na začátku řádku předcházet stejná skupina mezer a tabulátorů. Na jednu stranu je to výhodné, protože jsme tím nuceni odsazení dodržovat a zachovávat tak čitelnost kódu, na druhou stranu se můžou „bílé“ znaky snadno poplést nebo může být dost pracné, když se například rozhodneme z dříve napsaného kódu odsadit 50 řádků do zvláštního bloku.

Při deklaraci proměnných není nutné uvádět jejich typ, ten se určí dynamicky za běhu programu podle typu výrazu na pravé straně přiřazení při inicializaci proměnné. I přes typovou kontrolu je možné do jedné proměnné uložit po sobě celé číslo, číslo v plovoucí řádové čárce či řetězec. Mezi vestavěnými typy v Pythonu jsou i některé dynamické datové struktury, jako seznamy (nemusí být vůbec homogenní), slovníky, tuple a set, které usnadňují programátorům práci.

Python se rovněž používá jako skriptovací jazyk, stejně tak je to i v případě programu Blender, který je vybaven interpretem, s nímž lze pracovat jako s kalkulačkou přijímající pythonovské příkazy, nebo pomocí něho provádět skripty s tradiční koncovkou .py.

2.4 Mesh

Mesh představuje nejzákladnější druh modelu v Blenderu. Anglický termín *mesh* lze přeložit jako síť, model tělesa je totiž složen ze sítě bodů, které jsou navzájem spojeny hranami. Viditelné jsou pouze plochy tělesa a nezáleží přitom, zda se jedná o manifold, či non-manifold objekt, mesh se dá tedy zařadit jako hraniční polygonální model. V Blenderu jsou všechny objekty reprezentovány uspořádanou dvojicí, kde první člen určuje jméno objektu a druhý je jméno data bloku, jehož typ může být právě zmíněný Mesh, kde jsou uložena vlastní data.

Uložení informací o Meshi je hierarchické a dal by se popsat jako tzv. okřídlená hrana [5]. To v jednoduchosti znamená to, že nedochází k redundanci dat. Objekt totiž obsahuje ukazatel na seznam jeho stěn, každá stěna pak disponuje ukazatelem na seznam hran, které stěnu tvoří, a konečně hrana musí obsahovat ukazatele na dva vrcholy, jež spojuje. Takže ve výsledku jsou trojrozměrné souřadnice vrcholů, určující jejich polohu, ukládány pouze jednou. Z toho také plyne, že pokud chceme přemístit některé z vrcholů, upravíme přímo jejich souřadnice a o ostatní části (hrany, plochy) se nemusíme příliš zajímat, jelikož se po překreslení samy dostanou do nové, správné pozice díky zmíněným odkazům.

Při generování nových objektů pomocí skriptu je fundamentální částí definování souřadnic vrcholů, které budou objekt tvořit. Tyto souřadnice můžeme získat tak, že je budeme mít v skriptu uvedeny jako konstanty, nebo je vhodným algoritmickým postupem vypočítat za běhu programu, anebo kombinací obou přístupů. Samotné vrcholy nejsou ale na scéně vůbec vidět, a proto je nutné opatřit objekt stěnami.

Stěna se vytvoří tak, že se zavolá konstruktor dané třídy a do proměnné k tomu určené se přiřadí ukazatel na seznam vrcholů. Přitom je potřeba dbát na to, aby tento předem definovaný seznam měl délku 3, nebo 4, všechny plochy jsou tudíž buďto trojúhelníky, nebo čtyřúhelníky. Pokus o vložení stěny s jiným počtem vrcholů skončí chybou. Jak jsem již nastínil v minulém odstavci, seznam vrcholů, který se přiřadí některé ploše, neobsahuje konkrétní hodnoty, ale je vyžadováno, aby seznam zahrnoval pouze odkazy na dané vrcholy.

Pro pořadí vrcholů v seznamu platí dvě pravidla. První pravidlo říká, že vrcholy nesmí být zadávány na přeskáčku (to je možno splést pouze u čtyřúhelníkových ploch), musí tedy být řazeny v takovém sledu, v jakém jsou postupně procházeny po hranách dané stěny. Druhé pravidlo zajišťuje, že stěny tělesa nebudou naruby, neodpovídala by pak normála, což je vektor, kolmý k ploše. Normálový vektor hraje důležitou roli při výpočtu osvětlení nebo při zjištění viditelnosti dané plochy z pozice pozorovatele, takže jeho správné určení je poměrně zásadní. Můžeme při tom využít pravidla pravé ruky, kdy plochu jakoby obejmeme vnitřní stranou pravé dlaně a prsty jdou po směru vkládaných vrcholů, tak jak jsou v seznamu. Tehdy ukazuje palec, který je kolmý k ostatním prstům, směr normálového vektoru, směřuje tedy do prostoru mimo těleso. Pokud tomu tak není, je zřejmé, že bude nutné pořadí vrcholů v seznamu otočit. Dodržení správného pořadí vrcholů je důležité i z toho důvodu, že při vytváření nových ploch se automaticky generují i hrany a inicializují se ukazatele na příslušné dva vrcholy, které hraně náleží.

Kromě třídy Mesh existuje ještě jedna podobná, která se nazývá NMesh. Jedná se o třídu, která je předchůdcem třídy Mesh, a nyní se považuje za zastaralou. Vývojáři sice dbají na to, aby tato starší verze byla ze strany Blenderu podporována, nedoporučuje se však ji používat, už jen proto, že je ochuzena o některé proměnné a nedosahuje takové funkčnosti a rychlosti výpočtu, jako je to v případě třídy nové. Ve snaze vymýtit používání staré třídy jsou veškeré instance třídy NMesh po vložení do scény přetypovány na instance třídy Mesh. Jediná situace, kdy se můžeme vůbec setkat s instancí zastaralé třídy a čelit tak jisté heterogennosti objektů, z čehož může pramenit spousta chyb, je, když pomocí skriptu získáme objekt pomocí metody staré třídy, to provést lze.

Další vyčerpávající informace o modelování Meshe lze najít v literatuře [1].

2.5 Space handler

Podkapitola vychází z dokumentace Blenderova API [7].

Space handler vypadá na první pohled jako obyčejný pythonovský skript. Jeho spuštění se ale od ostatních skriptů poměrně liší. Zatímco jiné skripty se spustí, následně se provedou a po nějakém čase skončí, space handler skript je volán automaticky v nekonečné smyčce, jestliže nastane nějaká událost. Událostí se rozumí například uživatelský vstup skrze klávesnici nebo myš. Space handler skripty tedy byly navrženy pro práci s těmito událostmi a reakci na ně.

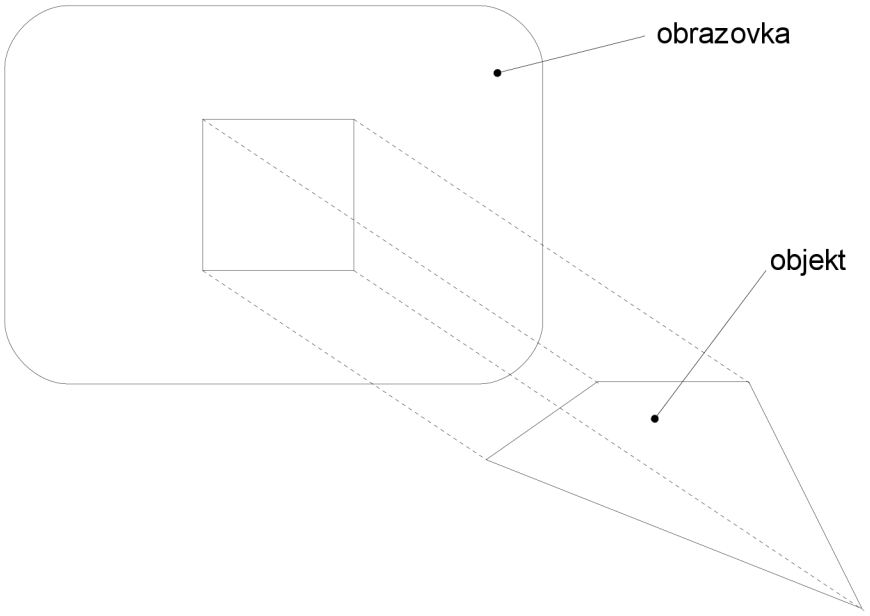
Space handler je od ostatních skriptů rozpoznán interpretem hned na prvním řádku souboru. Tam se musí nacházet komentář a za ním informace, o jaký druh space handleru se jedná a které z oken Blenderu je k němu přidruženo. Na výběr jsou dva typy skriptů, jeden se nazývá „Event“ a druhý typ je „Draw“. Skript typu Event má na starost zpracování přicházejících událostí, tyto události identifikuje porovnáním s položkami slovníku, kde jsou všechny možné druhy uloženy. Dále se může zachovat jedním ze dvou způsobů, buďto událost zpracuje a nadefinuje její nové chování, přičemž Blender už nebude tuto událost více zpracovávat, anebo ji jednoduše ignoruje. Tato možnost, že můžeme na danou událost v Event handleru jistým způsobem reagovat nebo ji nechat být, je dána tím, že událost je prvně poskytnuta ke zpracování skriptu a až poté samotnému Blenderu.

Druhý typ Draw je určený ke kreslení do okna a jeho vyvolání je podmíněno zneplatněním kreslící plochy, překreslením apod. Bylo by chybou zpracovávat události ve skriptu typu Draw (všechny by navíc měly hodnotu 0), stejně tak jako používat kreslící funkce OpenGL uvnitř těla Event skriptu, oba typy mají svou konkrétní roli a tyto přístupy není radno kombinovat. Při použití Draw space handleru je nutné dát si pozor, abychom zanechali prostor ve stejném stavu, v jakém jsme do něj vstoupili. Toto je navíc pojištěno uložením atributů OpenGL na zásobník před spuštěním skriptu a jejich následného vyjmutí ze zásobníku po skončení.

Další podstatný údaj, spočívající na prvním řádku space handleru je název prostoru, ke kterému se skript váže. V současnosti je možné plně využít jen prostor 3D View, vývojáři Blenderu mají ale v plánu doimplementovat potřebnou podporu i pro další prostory – textový editor, skriptové okno, uživatelské nastavení, atd.

2.6 Projekční matice

Projekční matice slouží k tomu, aby mohly být 3D objekty transformovány pro jejich dvourozměrné zobrazení. Je dána tím, jakým směrem je scéna natočena k pozorovateli a zda se jedná o paralelní nebo perspektivní projekci. Nejjednodušším případem je paralelní projekce, kdy je průmětna rovnoběžná s rovinou os x a y. Pak stačí zanedbat všechny souřadnice v ose z, tedy je vynulovat a projekce je hotova. Při obecné paralelní projekci je situace o to složitější, že prvně musíme přenést průmětnu do roviny rovnoběžné s osami x a y, projekční matice je ve výsledku poskládána z více transformací dohromady, viz. [4].



Kresba 1: paralelní projekce

3 Současný stav

Co se týče podobných programů pro návrh a vizualizaci lezeckých stěn, žádný takový se mi nepodařilo najít, takže je možné, že vůbec neexistuje. Na internetu se dají najít pokyny k návrhu a konstrukci stěny, rovněž zásady, které by měl stavitel při technické realizaci dodržet. Všechny takové postupy ale začínají pouze náčrtem na papír a o žádném programu, který by umožňoval poněkud přesnější náhled na budoucí dílo, zde není zmínka. Samozřejmě jsou k dispozici modelovací programy jako form-Z, Maya, Lightwave nebo solidThinking, ty ale nejsou specializované primárně pro tvorbu lezeckých stěn. Pomineme-li možnost, že podobný nástroj už skutečně někde slouží soukromé firmě pro kompletní servis v oblasti lezeckých stěn, bude program v tomto směru poměrně originální.

4 Návrh

Jak vyžaduje zadání, výsledná aplikace by měla být snadno ovladatelným nástrojem pro tvorbu lezeckých stěn. Přitom je důležité si uvědomit, že ne každý je expertem na vytváření 3D modelů v programu Blender. Proto by měl být uživatel schopen pracovat s programem, i když zatím neměl s Blenderem moc co do činění nebo zná jen jeho základní ovládání, popsaném ve stručném tutoriálu [6], bohatší zkušenosti nejsou samozřejmě na škodu.

K ovládání pluginu bude sloužit pouze jednoduché a srozumitelné rozhraní a manipulace s myší. Navíc by bylo výhodné, kdyby skript zachoval původní funkci středního a pravého tlačítka myši a pracoval pouze s levým, poněvadž zbylé dvě slouží v Blenderu jako nástroje pro rotování scény a selekci objektu, což se bude určitě hodit i pro moji aplikaci. Kromě práce s myší by měl být uživatel obeznámen s tím, jak se do Blenderu vloží základní útvary (mezerník) a jak se vyvolají jejich základní geometrické transformace – otáčení kolem obecné osy, změna měřítka a přesun (klávesy R, S a G, odvozeno od slov rotate, scale a grab) a také jak se mění pohledy na scénu (z pohledu kamery, zepředu, z boku a seshora; klávesy 0, 1, 3 a 7 na numerické klávesnici). Žádné další ovládací prvky Blenderu nebude muset uživatel znát, protože bude mít k dispozici rozhraní, které se o všechno ostatní postará.

K vytvoření hrubého tvaru lezecké stěny bude podle návrhu stačit vložení jednoduchého objektu, např. objekt Plane nebo Cube, a jeho následné deformace, posun a přidávání nových hran, čímž se stěna rozdělí na více menších a dochází tak k stále většímu zjemňování povrchu objektu. Kromě manipulace s hranami bude aplikace schopná vytvářet a přemisťovat vrcholy. Rozhraní bude obsahovat tlačítka pro přepínání mezi editací vrcholů a hran. Přemístění vrcholů a hran se bude dít tak, že se jednoduše přetáhnou myší na novou pozici, zatímco nová hrana bude vložena nakreslením myší na obrazovku a nový vrchol se objeví po stisku levého tlačítka myši. Manipulace s objekty bude řešena pomocí space handler skriptu, který bude zaznamenávat příchozí události a předefinovávat chování programu, coby odpovědí na tyto události.

Až bude tvar stěny vymodelován, bylo by vhodné mít k dispozici prostředky pro nanesení textury na objekt, pokud možno tak, aby se výsledek jevil uživateli relativně estetický a odpovídající realitě. Nedílnou součástí každé lezecké stěny jsou úchyty a výstupky, bez nichž by nebylo možné se dostat nahoru. Ty mám v plánu namodelovat pomocí textury, anebo jako samostatné objekty, čímž by se jistě dosáhlo větší plastičnosti. Konečně bude mít uživatel možnost vložit do scény některý z předem namodelovaných doplňků a neřešit tak jejich zhotovení.

4.1 Modelování stěny

O tom, co a jak se namodeluje, bude rozhodovat algoritmus (text 1), který jsem navrhl. Tok programu je zejména ovlivněn pěti stavy, a těmi jsou *mys_stlacena*, což je příznak značící, že levé tlačítko myši je stlačeno, *editace_hran* a *editace_vrcholu*, což jsou vzájemně komplementární stavy, z nichž právě jeden je v jednom okamžiku aktivní (to by mělo být zajištěno prvkem uživatelského rozhraní typu radio button), stejně jako stavy *kresleni* a *presun*, které určují jednu z akcí, která se bude provádět s vybranou hranou.

```
for(;;){
    získej událost;
    if (událost == stisk myši){
        mys_stlacena = true;
        ulož souřadnice;
        if(editace_hran && presun){
            vyber nejbližší hranu a ulož ji;
            vypočítej vektor pro posunutí a ulož;
        }
        else if(editace_vrcholu){
            vyber nejbližší vrchol;
            if(žádný vrchol na blízku){
                vyber nejbližší plochu;
                rozděl plochu na menší přidáním vrcholu;
            }
            else
                vypočítej vektor pro posunutí a ulož;
        }
    }
    else if(událost == uvolnění myši){
        mys_stlacena = false
        if(editace_hran && kresleni)
            nakresli novou hranu;
    }
    else if(událost == pohyb myši){
        if(mys_stlacena)
            přesuň vrchol nebo hranu po vektoru;
        else
            zvýrazni vrchol nebo hranu;
    }
}
```

Text 1: modelovací algoritmus

4.1.1 Přemístění vrcholu

I tak zdánlivě jednoduchá operace, jakou je přemístění vrcholu v trojrozměrném prostoru, se při detailnějším pohledu na věc nemusí jevit až tak jednoduše. Při řešení tohoto problému se musíme potýkat s několika úskalími:

- výběr správného vrcholu pro přesun
- přístup k datům udávajících pozici vrcholu
- metoda výpočtu nových souřadnic
- uložení nových souřadnic

Bylo by jistě záhodno, kdybychom pouze klikli myší na obrazovku a program sám poznal, že se na daném místě nachází vrchol a že s ním chceme pracovat. Ve skutečnosti musíme ale vše naprogramovat sami. Základní nesrovnalost tkví v tom, že chceme modelovat trojrozměrný objekt a máme k dispozici pouze myš, jež se pohybuje v prostoru o jednu dimenzi chudší. Lépe jsou na tom někteří architekti využívající ve specializovaných návrhových programech 3D myš, jež umožňuje i přemístění kurzoru do hloubky, nelze však počítat s tím, že běžný uživatel takovouto vymožeností oplývá, je to spíše raritou a jak se ukáže, nebude koneckonců toto zařízení ani potřeba.

Využijeme totiž toho, že si všechny body promítneme na obrazovku. K tomu je zapotřebí získat projekční matici, až ji získáme, nic nám nebrání v tom, abychom spočítali souřadnice promítnutých bodů. To provedeme tak, že vektor obsahující souřadnice jednotlivých bodů doplněný o váhu bodu (vektor má tedy celkem 4 složky) vynásobíme projekční maticí 4 x 4. Přitom nesmíme

zapomenout, že násobení matic (vektor je rovněž matice, s jedním řádkem) není komutativní, záleží tedy na pořadí, v jakém matice mezi sebou vynásobíme. I kdybychom omylem pořadí zaměnili, asi bychom na to brzo přišli, poněvadž bychom neměli co s čím násobit, první matice totiž musí obsahovat stejný počet sloupců, co druhá matice řádků. Násobení vektoru projekční maticí předchází ještě vynásobení maticí, spjaté s objektem, jehož je daný bod součástí. Do této matice se ukládají všechny geometrické transformace objektu. Kdybychom na tento krok zapomněli, výsledky projekce by sice byly správné, ale pouze do chvíle, kdy bychom začali s objektem manipulovat.

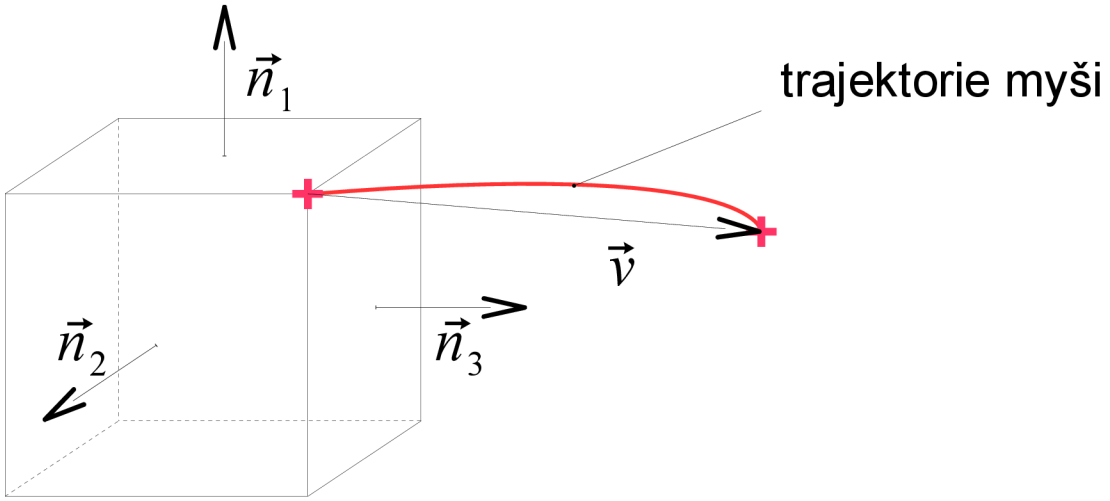
Až jsou všechny body promítnuty a redukovány o svou souřadnici v ose z, zbývá tyto hodnoty interpolovat pro velikost okna, kde jsou zobrazeny. Dále je nutné provést jisté posunutí, jelikož se v Blenderu nachází počátek souřadnic vždy uprostřed okna 3D View. Teprve tehdy, až podrobíme souřadnice bodů všem těmto úpravám, můžeme s čistým svědomím porovnávat vypočtené hodnoty s pozicí myši. Pak už není problém určit, který vrchol a z jakého objektu se nachází nejblíže kurzoru myši a v jaké vzdálenosti.

Jak jsem nastínil ve výše uvedeném algoritmu, vrcholy se budou přesouvat, pokud je uživatelské rozhraní nastaveno pro práci s vrcholy a zároveň pokud bylo myš stlačena dostatečně blízko některého vrcholu. Díky projekci bodů jsme schopni určit, jestli takováto situace nastala. Aby si byl uživatel vědom toho, že se myš nachází blízko vrcholu a je tedy možné s ním hýbat, program mu to dá najevo tím, že vrchol zvýrazní. To je na jednu stranu pro uživatele jistě přívětivá vlastnost, ale na druhou stranu si to bere svou daň v té podobě, že při každé změně pozice kurzoru se spustí mechanismus, určující, zda je nějaký vrchol na blízku. To může být ale pro model s velkým počtem vrcholů časově náročná operace a aplikace nestačí odpovědět v reálném čase a může se jevit uživateli jako pomalá.

Když už víme, který vrchol se má přesunout, je nasnadě položit si otázku, kde vůbec bude jeho nová pozice. V Blenderu funguje přemisťování objektu tím způsobem, že každý z jeho vrcholů se pohybuje po rovině, do které daný bod náleží a navíc je tato rovina kolmá k pohledovému vektoru. Pohledový vektor je vektor, jehož počátek se nachází v počátku souřadnic a jeho koncový bod leží v pozici pozorovatele. Pokud bychom zvolili tuto cestu a snažili se napodobit chování Blenderu, docela bychom si tím zkomplikovali život. Není sice velký problém spočítat rovnici roviny, ve které se budou body pohybovat, ale získat dvě souřadnice, abychom mohli třetí z rovnice dopočítat, už není tak triviální záležitostí. Místo toho jsem zvolil jednodušší variantu, která sice nedovoluje pohyb v úplně libovolném směru, pro potřeby programu by ale měla dostačovat. Vrcholy budou přemístěny ve směru normálového vektoru plochy a souřadnice myši budou sloužit pouze pro určení míry, jak daleko mají být body po normále posunuty, případně v jejím opačném směru.

Kromě plošných objektů je běžné, že je vrchol sdílen několika stěnami. Máme tak k dispozici více normálových vektorů, z nichž vybíráme kandidáty pro určení směru pohybu vrcholu. První nápad byl ten, že se vybere normála stěny, která je natočena nejvíce přímo směrem k pozorovateli, neboli se vybere normála, jejíž odchylka od pohledového vektoru je nejmenší. To se ukázalo jako ne příliš vhodné řešení, protože zkrátka nebylo moc dobře vidět, kam se vrchol posouvá, a navíc ovládání nebylo příliš přirozené.

Pak se naskytla jiná alternativa, která spočívala v tom, že se vybere jeden vektor z množiny normálových vektorů stěn, jejichž je vrchol součástí. A to ten, který nejvíce odpovídá směru tahu myši, tedy ten, jehož projekce na obrazovku má nejmenší odchylku od vektoru zadaného dvěma body, místem, kde bylo tlačítko myši stlačeno, a místem, kde dojde k jeho opětovnému uvolnění. Celou situaci zachycuje obrázek.

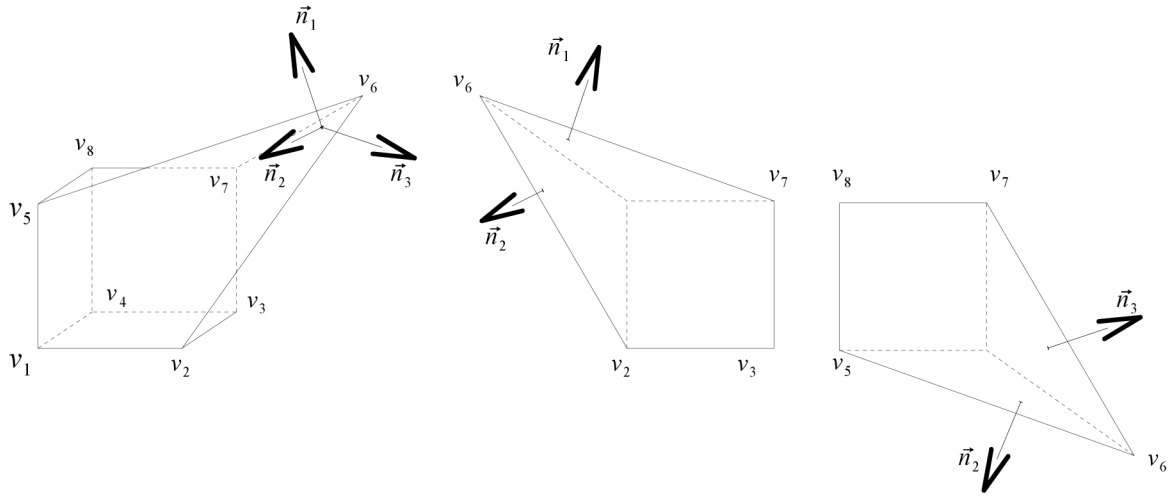


Kresba 2: přesunutí vrcholu

V případě, že pracujeme se stěnou tvaru krychle, vrchol sousedí s celkem třemi stěnami a vybíráme tedy jednu ze tří normál. Z obrázku by mělo být patrné, že bude zvolen normálový vektor č.3 a vrchol se bude pohybovat směrem „do boku“. Tento způsob je výhodný v tom, že uživatel smí pohybovat s vrcholem ve více směrech a přitom není nucen jakkoliv upravovat pohled na scénu, jak je tomu v případě standardního chování Blenderu. Ovládání je více intuitivní, poněvadž vrchol je přemístěn v takovém směru, který nejvíce odpovídá trajektorii myši.

Může se zde objevit jistý zádrhel, protože pohyb myši je použit jak pro zjištění vektoru posunutí, tak i pro určení míry, do jaké vzdálenosti se vrchol po vektoru posune. To může mít za následek, že se vrchol sice bude pohybovat ve směru správně zvolené normály, tažením myši se však změní její trajektorie a může dojít k tomu, že jiný normálový vektor bude vyhodnocen jako nejvíce rovnoběžný, což se projeví ostrým přeblikáváním vrcholu, pokud se myš nachází v hraniční oblasti. Tomu se dá předejít tím, že vektor posunutí se zjistí během prvních např. 10 událostí, kdy dojde ke změně pozice kurzoru myši a pak přejde program do stavu, kdy už vektor posunutí zůstává nezměněn a pohyb myši slouží už jen pro výpočet vzdálenosti.

I tak má tento přístup jeden zásadní nedostatek, kvůli kterému bylo nutné zvolit jinou cestu. Problém, který diskvalifikuje tuto metodu, nastává tehdy, když vymodelujeme stěnu s příliš špičatými vrcholy. Ať už zvolíme kterýkoli z dostupných normálových vektorů, nikdy se nám už nepodaří tuto „špici“ odstranit. Pro snazší pochopení přikládám obrázek (kresba 3). Na něm je znázorněna deformovaná kostka, jejíž vrchol č.6 už není možné vrátit do původní pozice.



perspektivní pohled pohled z pravého boku pohled seshora

Kresba 3: nevratné posunutí vrcholu

U posledního řešení je stejně jako u první metody zvolený pouze jeden směr, ve kterém je dovoleno vrcholu posouvat se. Nelze na to však pohlížet jako na krok zpět, jelikož je tento způsob oproštěn od nedostatků druhé metody. Směr posunutí bude dán skládáním normálových vektorů jednotlivých stěn , sousedících s vrcholem, dohromady. To by mělo zajistit, že při modelování se neobjeví žádný nevhodně umístěný vrchol, kterého se nejde zbavit, ale bude možné vrátit se o krok zpět obnovením jeho původní pozice. Místo skládání vektorů jsem v programu použil aritmetický průměr jednotlivých složek vektorů, tím se výpočet značně zjednoduší a přitom podává dobré výsledky. Jen je třeba dbát na to, aby byly normálové vektory před vypočítáním průměru normalizovány a byly tak vzájemně porovnatelné.

Až se ocitneme ve fázi, kdy je nám znám vrchol, se kterým se má posouvat, a také vektor, určující směr posunutí, můžeme přistoupit k výpočtu nových souřadnic. K tomu nám poslouží tyto vzorce:

$$coord_2.x = coord_1.x + v.x \cdot q \quad (1)$$

$$coord_2.y = coord_1.y + v.y \cdot q \quad (2)$$

$$coord_2.z = coord_1.z + v.z \cdot q \quad (3)$$

$Coord_1$ v nich značí původní souřadnice vrcholu, $coord_2$ jsou souřadnice nové, v reprezentuje vektor posunutí a konečně q je koeficient, jež říká, jak velký násobek vektoru se pro výpočet použije, jinak řečeno, jak daleko se vrchol bude po vektoru posouvat. Není vůbec na škodu si uvědomit, že původní pozice vrcholu a vypočtený vektor udávající směr posunutí musí zůstat během posouvání vrcholu stále stejné, jinak bychom nedosáhli kýženého výsledku, jediné, co se bude v rovnicích měnit, je koeficient q. Výpočet tohoto koeficientu jsem navrhl tak, že bude růst při pohybu myši směrem ke kladným hodnotám os x a y. Může nabývat i záporných hodnot, což značí, že vrchol se bude pohybovat v opačném směru vektoru posunutí.

Staré souřadnice vrcholu se nahradí novými a objekt je poté překreslen. Celá operace přemisťování vrcholu končí ve chvíli, kdy je puštěno levé tlačítko myši.

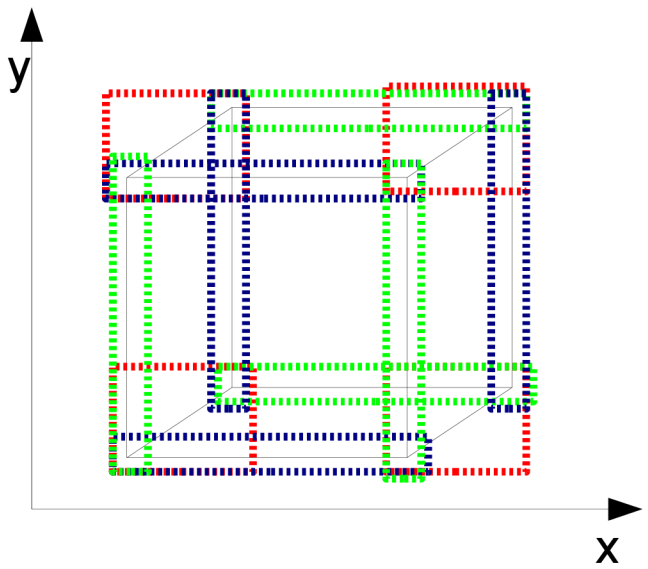
4.1.2 Přemístění hrany

Dalším požadavkem na modelování stěny je, aby bylo možné hýbat s jednotlivými hranami. Metoda posunu hrany se nebude téměř lišit od metody použité pro manipulaci s vrcholy až na výjimku, že výpočet nových souřadnic se nebude vztahovat pouze na jediný vrchol, ale rovnou na dva vrcholy, které hranu tvoří. I v tomto případě platí, že se budou body hrany pohybovat po vektoru, jež vznikne aritmetickým průměrem normálových vektorů ploch, kterých je hrana součástí.

V případě přemisťování vrcholů byla situace poměrně jednoduchá v tom, že jsme všechny body ze scény promítli na obrazovku a pomocí Pythagorovy věty jednoduše zjistili jejich vzdálenost od kurzoru, popř. který z nich se nachází nejblíž. Jak ale zjistit, se kterou hranou má uživatel v plánu pohybovat? Pomocí minulého postupu bychom se sice dopátrali toho, který je jeden vrchol hrany, ale už nezjistíme, který je ten druhý, protože jich může být neomezený počet. A to ještě za předpokladu, že uživatel uchopí hranu za její vrchol, což je podle mého názoru dost nepřirozené, spíše je pravděpodobnější, že myš stlačí někde v prostoru mezi oběma vrcholy.

Jednou z možností je použít hrubou sílu a rasterizovat všechny hrany jako přímky a takto získané hodnoty porovnávat se souřadnicemi kurzoru. I když bychom takto rozpoznali asi nejpřesněji, se kterou hranou máme pracovat, je výpočet zbytečně složitý, navíc pixely nemusí přesně odpovídat těm, jak je rasterizuje Blender.

Místo toho jsem navrhl jiný způsob nalezení požadované hrany. Postup se bude skládat ze dvou fází, v té první by se měl okruh kandidátních hran rapidně zmenšit a ve druhé už určit konkrétní hranu. V první části se sestrojí pomyslné obdélníkové oblasti, které budou odpovídat hranám, promítnutým na obrazovku. Souřadnice levého spodního rohu obdélníka, jenž je rovnoběžný s osami, budou tvořeny menší *x*-ovou a *y*-ovou souřadnicí z obou vrcholů hrany a pravému vrchnímu rohu budou odpovídat naopak větší hodnoty. Po lepší názornost přikládám obrázek 4.



Kresba 4: obdélníkové oblasti krychle

Pro všechny hrany ze scény pak sestavím takové obdélníky a zapíšu podmínku, že pokud se nalézá kurzor myši uvnitř obdélníka, přidám hranu do seznamu hran, u kterých je možnost, že se s nimi bude hýbat. Z obrázku si lze povšimnout, že obdélníky přesně nekopírují souřadnice vrcholů hran, ale jsou o něco málo větší. Tím je nastavená jistá tolerance, aby byl uživatel schopen hranu pohodlně označit, týká se to obzvlášť hran rovnoběžných s osami *x* nebo *y*. Pro krychli z obrázku bude v závislosti na souřadnicích myši seznam hran mít délku 0 – 3. K dalšímu zpracování v tomto případě tedy dojde maximálně jen u jedné čtvrtiny všech hran.

V dalším kroku se vypočítají souřadnice všech středů hran a vybere se ta hrana, jejíž střed je nejblíže kurzoru myši. Asi by bylo možné vynechat celý proces s obdélníkovými oblastmi, a přímo hledat nejbližší střed hrany, mělo by to ale negativní dopad na chování programu. Představme si situaci, kdy se v bezprostřední blízkosti nachází hrany podstatně různých velikostí. Kurzor je sice situován nad dlouhou hranou, ale je od jejího středu vzdálenější než ke středu druhé hrany. Ve výsledku by byla vybrána kratší hrana, byť se nad ní kurzor vůbec nenacházel, což může uživatele nepříjemně překvapit, ne-li rozčílit. Další výhodu práce s obdélníky lze spatřit v tom, že není nutné nastavovat prahovou hodnotu, která by určovala, zda už jsme od středu hrany příliš daleko a žádná hrana nebude vybrána.

Ve snaze podpořit interakci s uživatelem je vybraná hrana zvýrazněna, aby tak bylo naprosto jasné, která bude přesouvána.

4.1.3 Přidání nového vrcholu

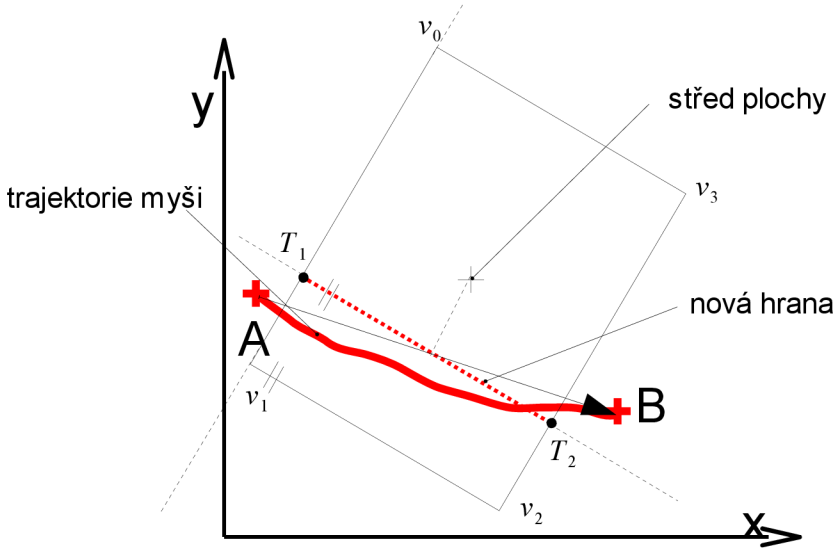
Modelovací algoritmus je nastaven tak, že k přidání nového vrcholu dochází tehdy, když uživatel stiskne levé tlačítko myši a poblíž není žádný vrchol, se kterým by se dalo pohybovat. V opačném případě to uživatel rozpozná tím, že se daný vrchol zvýrazní. Nové vrcholy není možné umístit kamkoliv do scény, ale vždy jsou pouze vloženy do stěny již existujícího objektu, kterou rozdělí na více menších stěn. Opakováním tohoto procesu se bude počet vrcholů zvyšovat a docházet tak k stále intenzivnějšímu zjemňování povrchu objektu.

Rozdělí se taková plocha, jejíž střed je nejblíže k místu kliknutí, což spočítáme již známým způsobem. Dále je možno přidat jednoduchý test, který bude mít za úkol zjistit, zda je plocha vůbec viditelná či zda není náhodou od pozorovatele odvrácena. Nemůžeme počítat s tím, že uživatel má zájem na tom, aby byl vložen vrchol do stěny, kterou vlastně ani nevidí. To se dá realizovat výpočtem odchylky pohledového vektoru od normálového vektoru dané plochy. Jestliže je odchylka těchto dvou vektorů větší než 90°, můžeme plochu prohlásit za odvrácenou a požadavek na vytvoření nového vrcholu směle ignorovat.

Až je daná plocha identifikována, spočítá se průměr ze všech jejích vrcholů, čímž se získají výchozí souřadnice nového bodu. K těm je navíc připočtena zanedbatelná hodnota, která vychýlí nový bod z přesného středu plochy, takže ten nebude ležet v rovině dané zbylými vrcholy. Je to vhodné proto, aby ten, kdo program požívá, vůbec poznal, že je nějaký nový vrchol vložen.

4.1.4 Přidání nové hrany

Asi nejsložitější operací z dosud zmíněných je přidání nové hrany. To se bude dít v podobném duchu jako vkládání vrcholu, nová hrana rozdělí jednu z ploch na dvě menší ve vhodném poměru. Celá záležitost by měla probíhat tak, že myš bude tažena od jedné hrany plochy k hraně protější, kde dojde k jejímu uvolnění. Nová hrana bude spojnicí mezi těmito dvěma hranami a navíc bude rovnoběžná s jednou ze hran plochy. Tažení myši přes více než jednu plochu se nebude předpokládat.



Kresba 5: přidání nové hrany

Celý postup se bude vztahovat na vrcholy promítnuté na obrazovku. Prvním krokem je nalezení odpovídající plochy, do které se nová hrana vloží. Taková plocha se začne hledat až po uvolnění myši, kdy jsou známy souřadnice, kde tažení myši bylo iniciováno a kde bylo tlačítko myši opět uvolněno. Pak je vyhledána plocha, jejíž střed je nejblíže bodu, který je aritmetickým průměrem zmíněných souřadnic, jedná se tedy o bod, který půlí úsečku, jež je na obrázku značena jako *AB*.

V této chvíli už jsme obeznámení s tím, která z ploch bude figurovat při vložení nové hrany. Neznáme však souřadnice bodů T_1 a T_2, ale víme o nich to, že budou jistě ležet na hranách, tvořenými dvojicemi vrcholů v_0, v_1 a v_3, v_2.

Nová hrana bude rovnoběžná s některou z hran plochy, vyberu z nich tedy takovou, která má k úsečce *AB* nejmenší odchylku. Tato hrana náleží přímce (v obrázku označena jako y_1), která se dá popsat rovnicí přímky ve směrnicovém tvaru (4).

$$y = k \cdot x + q \tag{4}$$

$$k = \tan(\alpha) \tag{5}$$

Nová hrana náleží přímce (y_2), jejíž rovnice se s rovnicí její rovnoběžky bude shodovat v parametru k, určující její sklon, lišit se od ní bude v koeficientu q, jenž způsobuje posunutí v ose y. Takto se vypočítá směrnice z první rovnice a dosazením do druhé získáme parametr q a kompletní rovnici přímky y_2(8).

$$k_1 = k_2 = \tan(\alpha) = \frac{v_2.y - v_1.y}{v_2.x - v_1.x} \tag{6}$$

$$q_2 = y - k_2 \cdot x = v_1.y - \frac{v_2.y - v_1.y}{v_2.x - v_1.x} \cdot v_1.x \tag{7}$$

$$y_2 = k_2 \cdot x + q_2 = \frac{v_2.y - v_1.y}{v_2.x - v_1.x} \cdot (x - v_1.x) + v_1.y \tag{8}$$

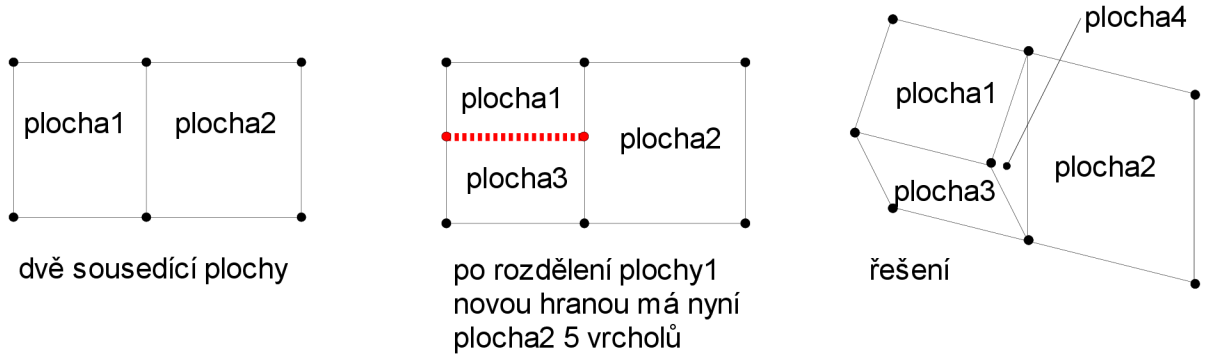
Z toho, že jsou nám známy souřadnice vrcholů v_0 a v_1, jsme schopni sestrojit rovnici přímky y_3.

$$y_3 = \frac{v_1.y - v_0.y}{v_1.x - v_0.x} \cdot (x - v_1.x) + v_1.y \tag{9}$$

Když položíme rovnice y_2 a y_3 sobě rovny, dostaneme jednu rovnici o jedné neznámé, jejímž vyřešením získáme souřadnici v ose x jejich vzájemného průsečíku T_1. Výpočet tím ještě není u svého konce, nesmíme zapomenout, že souřadnice bodu T_1 jsou pouhou projekcí reálného trojrozměrného bodu, jehož se ve skutečnosti snažíme dopátrat.

Cílem tohoto postupu je totiž zjistit poměr velikostí dvou vektorů, z nichž oba dva mají počátek ve vrcholu v_1, koncový bod prvního vektoru leží v bodě T_1 a druhý končí ve vrcholu v_0. Stejný poměr je totiž zachován i mezi původními, nepromítnutými vektory. Číslo, které vyjadřuje velikostní poměr mezi oběma vektory, by se mělo pohybovat v rozmezí 0 – 1, jinak je ve výpočtu patrně chyba. Abychom dostali skutečné, trojrozměrné souřadnice bodu T_1, vynásobíme nepromítnutý trojsložkový vektor s počátkem ve vrcholu v_1 a koncovým bodem v_0 vypočteným poměrem. Obdobným způsobem se určí i souřadnice druhého bodu T_2.

Plocha, kterou rozdělujeme novou hranou, nemusí být ve scéně nutně osamocena, ale je dokonce pravděpodobné, že je obklopena jinými plochami, které s naší plochou sousedí přes hranu. Hrana je tedy sdílena více plochami a při jejím rozdělení dojde k vložení nového vrcholu i do ostatních ploch. V tom se skrývá potíž, jelikož plochy se v Blenderu mohou skládat pouze ze tří, anebo čtyř vrcholů, jiný počet je nepřípustný. Pokud by se tato situace neohlídala, snadno bychom se mohli dopustit toho, že se budeme snažit rozšířit plochu o její pátý vrchol, což vyvolá chybu.



Kresba 6: vkládání nové hrany

Rozdělení ploch od jednoho kraje objektu až ke druhému by tento problém vyřešilo, ale bylo by to dosti náročné řešení, navíc lze debatovat o tom, zda je takové chování vůbec žádané. Místo toho je do objektu vložena pomocná boční plocha, která zabrání rozšíření plochy o nepatřičný vrchol. Navíc lze realizovat podmínku, která zajistí, že „bočnice“ bude vložena jen tehdy, kdy je skutečně potřeba.

4.2 Texturování

Na nanášení textur v Blenderu není nic zas tak složitého, ale pokud s tím člověk nemá moc zkušeností, může to chvíli trvat. Prvně je potřeba vytvořit nějaký materiál, který je s objektem spjat, jeden materiál je ale možné použít pro více objektů najednou a šetřit tak místo v paměti. V menu pro nastavení materiálu je spousta ovládacích prvků a najít je zde možné vše od nejzákladnějších

vlastností, jako je nastavení barvy pomocí barevných modelů RGB nebo HSV i s čtvrtým kanálem alfa pro dosažení průhlednosti, až po aktivaci ray-tracingu. Pro nanesení textury se musíme ale přepnout jinam pomocí tlačítka připomínající levhartí kůži.

Na výběr je zde několik typů procedurálních textur, tedy takových textur, které nemají charakter obrázku, který obsahuje informace o jednotlivých pixelech a na objekt se třeba i opakovaně nanese, ale jsou formulovány matematickým předpisem a konkrétní podoba textury se vypočte až pro daný objekt, viz. [2]. My ale musíme zvolit jako typ textury obrázek a cílový obrázek pak nahrát z disku. Pro správné zobrazení textury na lezeckou stěnu je dále zapotřebí změnit mapování textury z typu *Flat* na typ *Cube* a také změnit nastavení souřadnic z *Orco* na *Glob*.

Kroků k nanesení textury je tedy poměrně dost a bylo by užitečné, aby tento postup byl skriptem nějakým způsobem zautomatizován. Uživatele totiž hlavně zajímá, co a čím má být otexturováno. Rozhraní tedy bude vybaveno dvěma prvky pro výběr objektu, jenž bude „otapetován“ a dále menu s několika předdefinovanými texturami materiálů, jejichž užití je při konstrukci lezecké stěny běžné. Opakování textury by mělo být nastaveno tak, aby byl zachován realistický poměr mezi velikostí jejího vzorku a celé stěny.

4.3 Úchyty

Nebýt úchytů na lezecké stěně, sportovec by se asi jen těžko mohl vůbec dostat nahoru. Rozdělují se do několika kategorií, některé je možné uchopit a zároveň na nich i spočinout nohou, jiné to třeba nedovolují, poněvadž jsou tvořeny škvírou, kam se vleze pouze ruka, nejjistější je úchyt připomínající madlo, jehož se dá pevně držet. Lezci si navíc před výstupem opráší ruce magnesiem, aby tak dosáhli co nejvyššího tření mezi rukou a úchytem.

Těchto výčnělku je na umělé stěně zkrátka spoustu a i model stěny se bez nich neobejde, protože bez nich to prostě není ono. První návrh počítal s tím, že úchyty budou součástí textury stěny. To má jistě výhodu v tom, že jejich rozmístění více méně odpovídá skutečnosti a navíc se dá taková metoda poměrně lehce implementovat. Na druhou stranu, úchyty jako součást textury naprosto vyřazují ze hry jakoukoliv jejich editaci, přidání, či smazání úchytu, posunutí, atd.

Z těchto důvodů jsem se rozhodl pro náročnější variantu, a to takovou, že úchyty a stupy budou modelovány jako samostatné objekty. Od toho si slibuji, že bude možné určit individuální rysy každého chytu, jejich celkový počet na stěně a v neposlední řadě by měly působit více estetičtějším dojmem než úchyty realizované pouhou texturou. Díky tomuto návrhu by tak uživateli mělo být umožněno například rozlišit barvy jednotlivých tras úchytů na stěnách, kde se trasy mezi sebou vzájemně proplétají a každá trasa má svoji náročnost.

Každý úchyt bude kopií již předdefinovaného objektu, který se posléze vloží do scény. Bylo by však úmorné, úchyty vytvářet po jednom a pak je ručně umisťovat na stěnu vzhledem k tomu, že jich bývá podstatně velké množství. Návrh je takový, že uživatel se sám rozhodne, jaký počet výstupků bude ve scéně přítomno a vybere objekt, tedy stěnu, která má jimi být pokryta. Program by pak měl sám vložit požadovaný počet úchytů na povrch stěny v rovnoměrném rozložení na všechny její plochy. Je jasné, že takovéto náhodné rozmístění nebude naprosto dokonalé, jako kdyby ho dělal profesionální návrhář umělé stěny, je ale pravděpodobné, že pro řadu úchytů už nebude třeba hledat jinou vhodnou pozici a zůstanou na svém místě a my tak ušetříme mnoho času, než kdybychom umisťovali všechny úchyty na stěnu ručně.

Otázka tedy zní, jak rovnoměrně rozložit určitý počet úchytů na povrch objektu, který se může skládat z libovolného množství ploch libovolných rozměrů.

Pro tento problém jsem navrhl následující řešení. Nejprve spočítám součet obsahů všech ploch objektu a získám tak celkový povrch objektu. Tato hodnota se pak vydělí vhodným koeficientem a zjistí se tak plocha jedné jednotky. Čím vyšší je tento koeficient, tím by rozložení mělo být přesnější. Poté sestavím pole, naplněné indexy jednotlivých ploch objektu, přičemž indexy jedné plochy se budou v poli tolikrát opakovat, kolika jednotkám povrch dané plochy odpovídá. Platí tedy, že čím větší daná plocha bude, tím více indexů bude v poli obsahovat a tím větší je

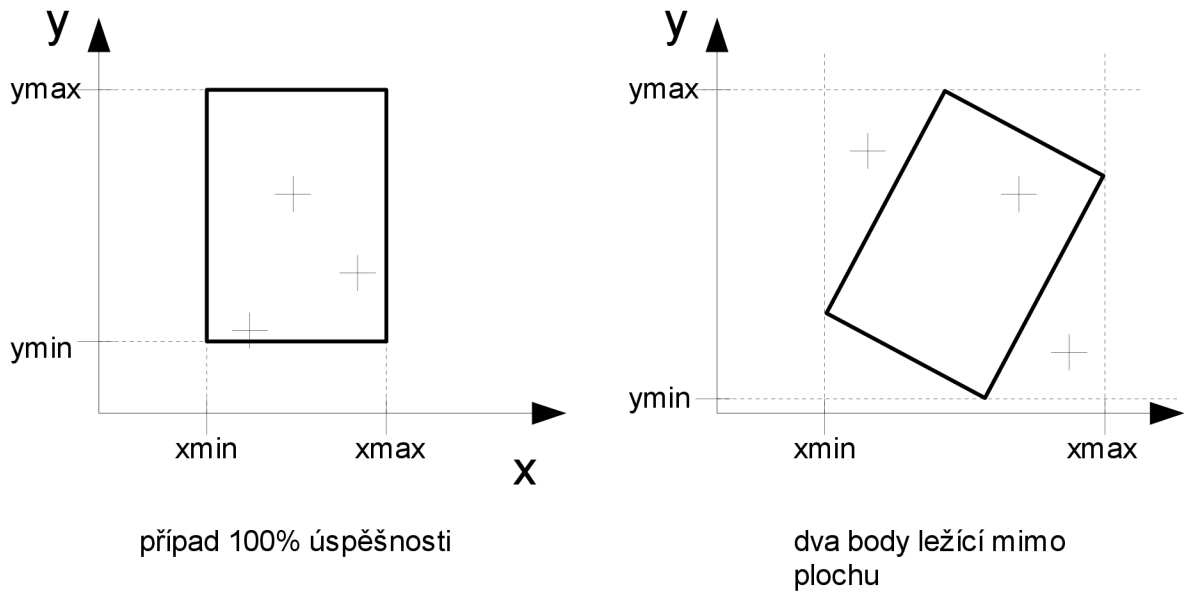
pravděpodobnost, že bude plocha vybrána. Pak už stačí pouze vygenerovat náhodné číslo v rozmezí 0 – (délka pole – 1), které bude sloužit jako index do pole, ukrývajícího indexy ploch. Nutno poznamenat, že délka pole nemusí být rovna výše zmíněnému koeficientu. Je to způsobeno tím, že se nastřádají malé plochy, jejichž obsah nedosahuje obsahu ani jediné jednotky a nebudou mít v poli žádný index, i když součet jejich obsahů větší jak obsah jednotky být může.

Tím se vyřeší spravedlivé vybírání ploch, do kterých se umístí úchyt. Nyní je na řadě generování náhodných souřadnic v rámci vybrané plochy. Mezi všemi vrcholy se spočítají minimální hodnoty v osách x a y. Vygenerují se souřadnice v tomto rozsahu, v úvahu přicházejí tedy pouze body z obdélníkové výseče roviny, ještě však zbývá třetí souřadnice v ose z. Ta se získá vypočtením z rovnice roviny, do níž plocha náleží, koeficienty a, b, c jsou složky normálového vektoru této plochy, koeficient d se získá dosazením jednoho z vrcholů.

$$a \cdot x + b \cdot y + c \cdot z + d = 0 \tag{10}$$

$$z = -\frac{a \cdot x + b \cdot y + d}{c} \tag{11}$$

Výsledkem je sice bod, který náleží do výseče stejné roviny jako plocha, to ale neznamená, že náleží i této ploše. Je to způsobeno tím, že plocha může být natočena vůči osám x a y a přestože vybíráme souřadnice x a y ve správném rozsahu hodnot, nemusíme se vždy trefit dovnitř plochy.

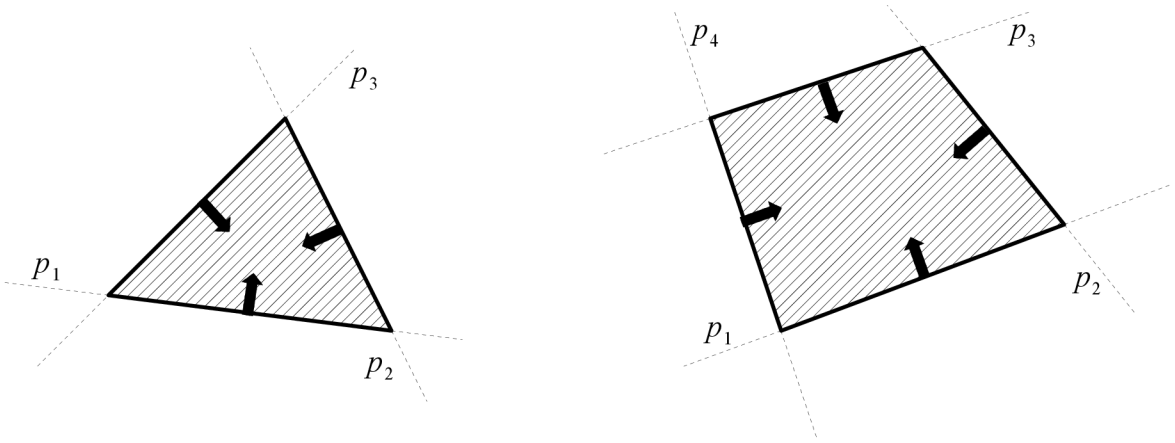


Kresba 7: natočení vůči osám

Úchyt umístěný na souřadnice vně plochy se na stěně projeví tím, že budou jakoby viset ve vzduchu a nikoliv připoutané k jejímu povrchu, jak se požaduje. Proto je nutné sestavit přídavný test, který zkontroluje, zda souřadnice leží opravdu i ve vybrané ploše, která má tvar trojúhelníku nebo čtyřúhelníku. Pokud vypočítané souřadnice testem neprojdou, bude zapotřebí generovat nové tak dlouho, dokud některý z bodu testem neprojde. Pak je možné přistoupit k výpočtu souřadnic dalšího úchytu.

Test je založený na tom, že jsou nám známy souřadnice vrcholů plochy a lze tedy spočítat rovnice přímek, do nichž náleží jednotlivé hrany plochy. Tyto přímky rozdělují rovinu na dvě

poloroviny a pokud má nový bod ležet uvnitř plochy, musí zároveň ležet uvnitř všech žádoucích polorovin.



Kresba 8: test příslušnosti bodu k ploše

Výpočet proběhne pouze v rovině, přičemž jsou složky souřadnic vrcholů v ose z jsou zanedbány. Taková plocha neodpovídá projekci a její tvar se změní, to si ale můžeme dovolit, protože nás nezajímá konkrétní tvar promítnuté plochy, ale to, zda se vypočtená pozice úchytu nachází uvnitř této plochy, na což nemá zanedbání z složky vliv.

Jeden krok testu spočívá v sestavení rovnice přímky ze dvojice vrcholů, které jí náleží. Samozřejmě se musí ošetřit stavy, kdy daná hrana nelze popsat rovnicí přímky (směrnice $k = \infty$). Do této rovnice se dosadí souřadnice v ose x libovolného sousedního vrcholu. Porovnáním y souřadnice sousedního vrcholu s výsledkem rovnice zjistíme, ve které polorovině musí ležet náhodně vygenerovaný bod. Pokud jsou souřadnice sousedního vrcholu v ose y větší, leží vrchol „nad přímkou“ a hledaná polorovina je ta, nacházející se „nad přímkou“, v druhém případě je tomu přesně naopak. Stejným způsobem zjistíme, do které ze dvou polorovin patří generovaný bod. Pokud si poloroviny, ve kterých leží sousední bod a bod generovaný neodpovídají, je možné test ihned ukončit, s tím závěrem, že generovaný bod nenáleží dané ploše. Tento krok se opakuje pro všechny dvojice vzájemně sousedících vrcholů plochy. Generovaný bod lze prohlásit za náležející ploše tehdy, když bude ležet ve všech určených polorovinách.

Metoda je použitelná pro všechny trojúhelníkové plochy a pro čtyřúhelníkové plochy vyjma těch, které jsou nekonvexní (konkávní). Tímto pojmem se nazývá rovinný nebo prostorový útvar, v němž je možné najít spojnici dvou bodů, náležících tomuto útvaru, pro kterou platí, že celá, anebo alespoň její část leží vně útvaru. Vzhledem ke způsobu modelování stěny by ale k vytváření nekonvexních čtyřúhelníkových ploch nemělo docházet, ledaže by tak uživatel učinil záměrně. Navíc lze v každé pořádné učebnici Blenderu najít poznámku, že se máme tvorby takových ploch vyvarovat a jejich použití se rozhodně nedoporučuje.

Velikost jednoho úchytu bude automaticky vypočítána tak, aby byla ve vhodném poměru s velikostí celé stěny. To znamená, že bude potřeba najít funkci určující délku hrany úchytu v závislosti na celkové ploše stěny.

4.4 Doplňky a scéna

Kromě samotné stěny a jejích úchytů bude mít uživatel možnost vložit do scény další doplňkové objekty, jimiž se okolí stěny obohatí a podtrhne tak cílový dojem z modelu. Doplňky, jako lana,

žíněnky nebo lavičky, se namodelují předem a uloží se do souboru s koncovkou .blend. Z takového souboru je možné načíst různorodé prvky, od materiálu a osvětlení až po kompletní objekty.

Co se týče scény, ta může být realizována vložením objektu typu Plane a jejím dostatečným roztažením a výběrem vhodné barvy pozadí. Okolí se dá také otexturovat, poslední možností je, že se stěna se vším všudy umístí do již existujícího vymodelovaného prostředí.

5 Implementace

Implementace proběhla dle návrhu s použitím programovacího jazyka Python. Zdrojový kód je rozdělen pro lepší čitelnost celkem do čtyř souborů, navíc jsou tak odděleny logické celky programu. Program by měl být až na nepatrné úpravy přenositelný, už z toho důvodu, že se jedná o klasický skript v Pythonu. Měl by tedy být schopen pracovat všude tam, kde je k dispozici program Blender (ve verzi 2.49b) a interpret Pythonu.

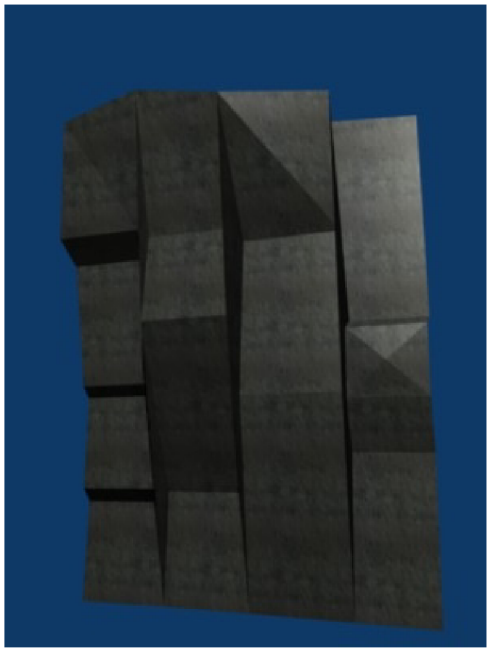
Pro start programu stačí otevřít hlavní skript ve skriptovacím okně Blenderu a spustit ho. Skript už sám vyhledá potřebné moduly, ty se však musí nacházet ve stejné složce. Nejdříve ze všeho se aktivují space handler skripty, jeden slouží pro modelaci stěny a druhý ovládá kreslení jednoduchých obrazců do scény, potřebné pro zvýraznění vrcholů a hran. Výhoda, které se dá využít, je, že space handler skripty se volají automaticky ve smyčce a kdykoliv je to potřeba a nemusíme se tím o ně už dále starat. V těle hlavního skriptu se rovněž registrují funkce pro vykreslení grafického uživatelského rozhraní a funkce zpracovávající události, vznikající při práci s tímto rozhraním. Jednotlivé skripty spolu komunikují a mohou si vyměňovat informace pomocí systému globálních proměnných, jež jsou uloženy ve slovníku, který je sdílen všemi skripty.

Uživatelské rozhraní je velice jednoduché, obsahuje prvky pro přepínání mezi editací vrcholů a hran, přepínaní kreslení nové hrany nebo jejího posunu, dále tlačítka pro nanášení textury a rozmístění úchytů na stěnu.

6 Experimentování a výsledky

V této fázi jsem provedl několik experimentů s výsledným programem a vyzkoušel tak, co všechno je v jeho silách. Pro začátek jsem ověřil, zda budu schopen s jeho využitím namodelovat hrubý model stěny podle vzoru na obrázku, viz ilustrace 2.

Ilustrace 2: umělá stěna v Ostravě, převzato ze stránek http://www.ben-kurzy.cz



Ilustrace 3: model ostravské stěny

I když tvar stěny není příliš složitý a najdou se dozajista i členitější povrchy, párkrát jsem byl nucen přepnout se do editačního módu a ručně upravit souřadnice neposlušných vrcholů. Vytvořený model není sice naprosto identický se svou skutečnou předlohou, dokazuje ale, že alespoň přibližný tvar stěny vymodelovat za pomoci skriptu lze.

Pokud jde všechno podle plánu a uživatel netápá v tom, jak se program ovládá, je vymodelování podobné stěny otázkou čtvrt hodiny. Odvážím se tvrdit, že je to doba stejná, ne-li kratší, než ta, potřebná ke zhotovení stěny klasickým způsobem. Výhoda je ale ta, že si nemusíme pamatovat klávesové zkratky potřebných operací nebo je hledat v menu, ale pohodlně vše obsloužíme myší a klikáním do jednoduchého GUI.

Jistou nevýhodou je, že svislé pláty stěny jsou od sebe odděleny „bočnicí“, jejíž přítomnost je odůvodněna na konci kapitoly 4.1.4. Ta se objevuje v místech, kde se prohýbání ploch ležících vedle sebe liší. To nemusí vždy působit estetickým dojmem.

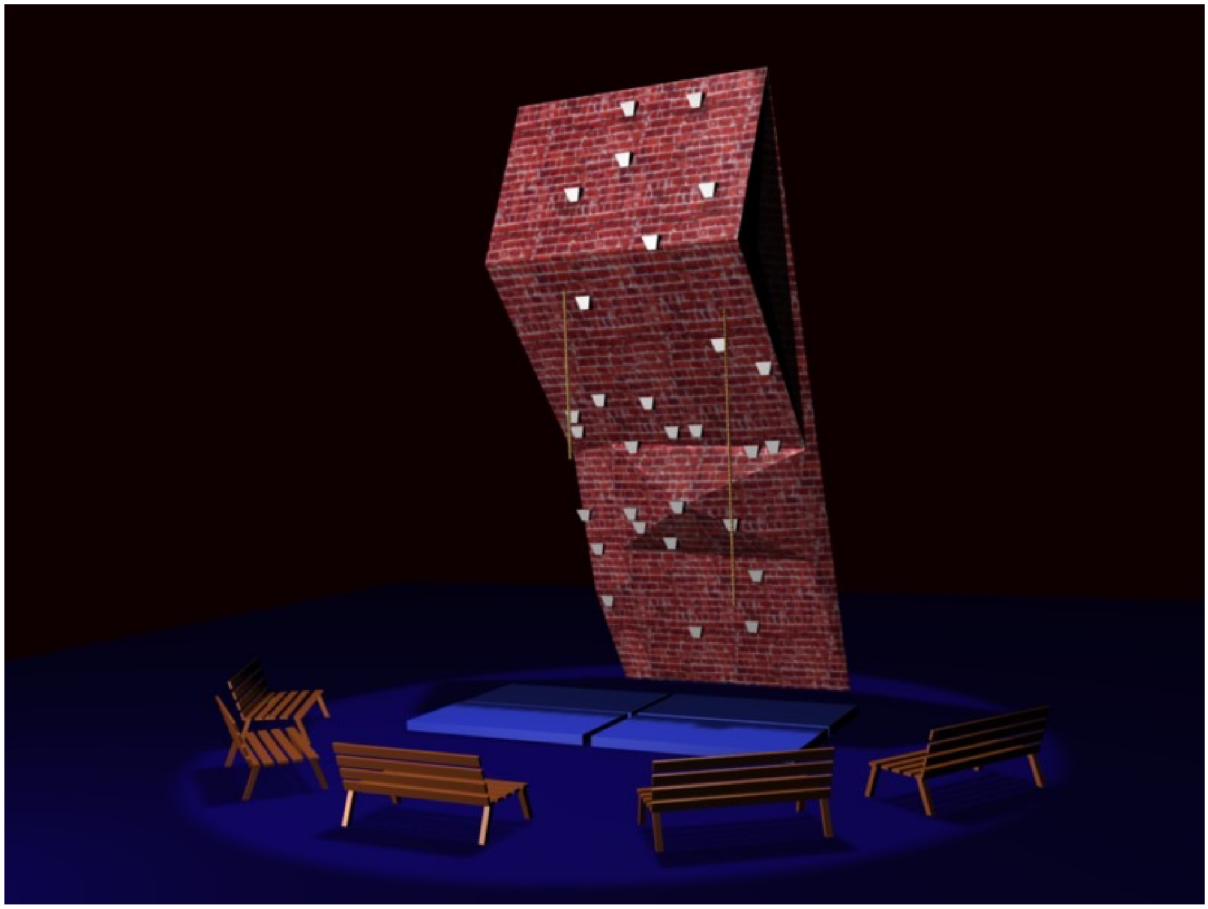
V dalším experimentu jsem navrhl stěnu podle své fantazie, doplnil ji o úchyty, osvětlení, lana, žíněnky a lavičky. Výsledek je na ilustraci 4.

Pokud by byl tvůrce modelu nespokojen s výsledkem, vždy je tu možnost použít pokročilé modelovací techniky Blenderu, které stěně dodají na její věrohodnosti. Ku příkladu rozdělit všechny plochy objektu na menší nástrojem Subdivision, pro toho, komu vzhled stěny realizovaný jako obrázková textura nestačí, bych zase doporučil texturu procedurální v kombinaci s displace

mappingem. Taková metoda dodá stěně členitý povrch podobající se kameni, návod je možné nalézt zde [3].

Program z podstaty způsobu modelování rozdělováním plochy na menší nedovoluje generování lezeckých stěn, jejichž povrch by byl nespojitý a obsahoval díry či výklenky. Jedna právě taková stěna se nachází na palubě lodi Freedom of the Seas a je zachycena v ilustraci 1 na straně 3, je to ale spíše rarita a klasické stěny žádnými otvory nedisponují.

Kvůli již zmíněnému problému s „bočnicemi“ jsem dospěl k názoru, že program není vhodný pro modelování složitých stěn v celku, ale je lepší pospojovat několik svislých, jednodušších plátů dohromady.



Ilustrace 4: experiment 2

7 Závěr

Cílem této práce bylo seznámit se s návrhem 3D aplikací a vlastnoručně si vyzkoušet navrženou aplikaci pro vizualizaci lezeckých stěn implementovat. Díky tomu, že výsledná aplikace byla nadstavbou programu Blender, osvojil jsem si základní ovládání tohoto, co se zdrojů týče, nenáročného, leč výkonného programu a dozvěděl se, jak lze přistupovat k vlastnostem objektů ve scéně a žádoucím způsobem je upravovat.

Během řešení byly aplikovány znalosti z oblasti matematiky, zejména analytická geometrie v rovině a v prostoru, matice a jejich násobení, coby fundamentálního prostředku geometrických transformací, a operace s vektory.

Při implementaci bylo nutné se neustále vypořádávat se základním problémem návrhu 3D aplikací, a tom tím, že vzory objektů, jež máme v úmyslu upravovat, se nacházejí v trojrozměrném systému, zatímco nám jsou k dispozici pouze zdánlivé obrazy těchto objektů, promítnuté do roviny obrazovky.

Přínosem je aplikace, která je přenositelná mezi systémy a bude fungovat všude tam, kde je k dispozici program Blender (verze 2.49b) a interpret jazyka Python. Po krátkém zorientování se v ovládání rozhraní umožňuje uživateli vytvářet realistické modely umělých lezeckých stěn, aniž by musel být detailně obeznámen i s ovládáním samotného Blenderu nebo mít představu o vnitřní reprezentaci modelu.

Bylo otestováno, že navržená aplikace se dá pro daný účel použít, i když je stále co zdokonalovat, jak už to u většiny rozsáhlejších děl bývá. Program tak může být předmětem dalšího vývoje a rozšiřování funkčnosti, už teď ale máme v rukou jistý základ, na který je možno úspěšně navázat.

8 Literatura a zdroje

1) Gumster, van J.: *Blender For Dummies*. Vydání I.. Wiley Publishing, Indianopolis, 2009. s. 83 – 108. ISBN 978-0-470-40018-0.
2) Hess, R.: *The Essential Blender*. Vydání II.. No Starch Press, San Fransisco, 2007. s. 234. ISBN 978-1-59327-166-4.
3) Kolingerová, I.: Blender 2.45. 2010, [online], [cit. 17. 5. 2010]. URL <http://iason.zcu.cz/~kolinger/Pokr/Blender.pdf>.
4) Kršek, P., Španěl, M.: Geometrické transformace ve 2D a 3D. 2008, [online], [cit. 17. 5. 2010]. URL <https://www.fit.vutbr.cz/study/courses/IZG/private/lecture/izg_slide_transformace_print.pdf>.
5) Kršek, P., Španěl, M.: Reprezentace 3D objektů. 2008, [online], [cit. 17. 5. 2010]. URL <https://www.fit.vutbr.cz/study/courses/IZG/private/lecture/izg_slide_3d_objekty_print.pdf>.
6) Střelák, D.: Modelujeme 3D v Blenderu. 13. 7. 2006, [online], [cit. 17. 5. 2010]. URL <http://www.grafika.cz/art/3d/blender-tutorial-1-prostredi.html>.
7) Blender Python related features. 24. 10. 2008, [online], [cit. 17. 5. 2010]. URL <http://www.blender.org/documentation/248PythonDoc/>.
8) Python (programming language). 2010, [online], [cit. 17. 5. 2010]. URL <http://en.wikipedia.org/wiki/Python_%28programming_language%29>.