TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 Elektrotechnika a informatika

Studijní obor: Informatika a logistika

Aplikace pro generování 3D modelu krajiny

Application for Generating 3D Model of Landscape

Bakalářská práce

Autor: **Petr Holec**

Vedoucí BP práce: Ing. Petr Kretschmer

Konzultant: Ing. Jiří Hnídek

V Liberci 28.5. 2009

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Abstrakt

S postupným a stále se zrychlujícím rozvojem výpočetní techniky se zejména v posledních letech rozmohl fenomén počítačem upravované a počítačem vytvářené grafiky. Ať už se jedná o kinematografii, herní průmysl, reklamní průmysl nebo jen úpravu digitálních fotografií pro vlastní použití, dopomohl k tomu právě rozmach v tomto odvětvích. Je pak jasné, že zkušenosti nabyté při práci s grafikou jsou dnes velmi cenné. A zkušenosti nabyté při práci s trojrozměrnou (3D) grafikou jsou ještě cennější. Už i v prostředí internetových prohlížečů se trojrozměrná grafika objevuje pomocí jazyku Java a Flash animací.

Zaměříme-li se pak na vytváření obrazu v reálném čase, jsou to grafické karty, na jejichž „bedrech“ tento úkol leží. Největším hnacím strojem pro rozvoj grafických karet je práce s video záznamy a počítačové hry. Největšími dodavateli grafických čipů jsou NVIDIA a ATI. I v aplikačním rozhraní jsou dva „rivalové“, obdobně jako na trhu s čipy, a to DirectX a OpenGL. DirectX je produktem společnosti Microsoft, stejně jako řada operačních systému Windows. Jdou proto ruku v ruce a na jiných operačních systémech je implementace nesnadná. U OpenGL, jak už napovídá název, se jedná o open-source (s otevřeným zdrojem, přístupným) grafickou knihovnu. Je použitelná na různých operačních systémech, ať už jádra Unix nebo systémy společnosti Apple. To je její největší výhoda, vývojáři grafických aplikací mají usnadněnou práci při vývoji verze na jiný operační systém, než je Windows. To snad bude i důvodem, proč jeden z nejznámějších herních vývojářů John Carmack (Doom 3 engine, Quake 1,2,3 engine) zůstává věrný tomuto API.

To by snad byly pohnutky, které vedly k výběru zadání z oboru počítačové grafiky. Samotné téma je pak vytvoření aplikace, která bude generovat trojrozměrný terén a následně zobrazovat v reálném čase. Z programovacích jazyků byl zvolen jazyk C++, protože je stále velmi populární a velmi rozšířený.

Další text se zabývá dokumentací funkcionality výsledného programu a zahrnuje popis prostředků pro jeho uskutečnění (teorie grafů, matematický aparát aj.).

Obsah

Úvod

Na začátek by bylo vhodné shrnout jednotlivé technologie a prostředky použité při tvorbě projektu. Po této části bude následovat představení aplikace, rozbor jednotlivých vizuálních prvků a popis práce s aplikací. Poslední část bude mít formu analýzy kódu a vysvětlení složitějších algoritmů, které byly při vývoji použity. Motivace už byla nastíněna dříve a bude i částečně obsažena v dalším textu, jelikož právě výhody použitých prostředků jsou důvody, proč byly zvoleny. Cílem projektu bylo vytvořit aplikaci pro generování a následně vykreslování trojrozměrného terénu. Za vývojové prostředí bylo zvoleno Microsoft Visual Studio. Co se týče konkrétních verzí, jedná se o verzi 2005 a 2008. Aplikace byla vyvíjena na operačních systémech Windows XP a Windows Vista. Výsledný program je primárně určený pro systémy Windows, ale díky zvoleným knihovnám a postupům se do budoucna předpokládá jeho rozšíření i na jiné platformy.

1. O vývojovém prostředí

1.1. O jazyku C++

C++ je kompilovaný programovací jazyk, který se prvně začal používat v roce 1980. V tu dobu se ovšem jmenoval ještě C with Classes a jak název napovídá, je pokračovatelem známého jazyku C. Svůj dnešní název získal jazyk v roce 1983. V syntaxi C nebo i C++ znamenají dvě plus zvětšení proměnné o jedničku, takže pokud bylo C první verze, C++ je analogicky verzí druhou. Autorem C++ je Dán Bjarne Stroustrup. Ten v roce 1979 započal vyvíjet nový programovací jazyk inspirován jiným jazykem, Simulou, s kterým přišel do styku. U Simuly shledal některé její aspekty velmi užitečné pro programování velkých aplikací, ale přišla mu velmi pomalá na to, aby se v praxi uplatnila. Rozhodl se proto, že rozšíří jazyk C o prvky Simuly. „Céčko“ zvolil, protože bylo rychlé, univerzální, přenositelné a široce používané. V průběhu dalších let přidal mimo jiného virtuální funkce, přetěžování operátorů a funkcí, konstanty, uživatelem ovládaný datový sklad a další. V roce 1985 byla vydána kniha The C++ Programming Language, která poskytovala důležitou oporu pro programování v nově vznikajícím jazyku. O čtyři

roky později byla vypuštěna do světa verze 2.0, která sebou přinesla například abstraktní třídy, statické metody třídy a mnoho dalšího. Tu dobu ještě nebyla zavedena žádná norma, která by se týkala C++. K tomu došlo až v roce 1998, další pak v roce 2003. Velkou výhodu oproti jazyku C má C++ v podpoře objektově orientovaného programování (OOP). A jako každý jazyk, který je objektově orientován (i když C++ není plně) má jeho výhody. OOP disponuje objekty a třídami, jenž mají vlastnost zapouzdření dat. Vlastnostem (parametrům) tříd se dají nadefinovat různá přístupová práva, přístupnost pak k těmto vlastnostem nebo i metodám závisí na místě, odkud konkrétní vlastnost nebo metodu požadujeme. Další a nejspíše největší výhodou OOP oproti procedurálnímu programování je dědičnost tříd. Díky ní se zvyšuje znovupoužitelnost kódu, který byl vytvořen. A to odvozováním podtříd od jiné obecnější třídy. Snadnější modifikace kódu, které se projevuje i pak na podtřídách je jistě také přínosem.

1.2. O API OpenGL

OpenGL je aplikační rozhraní, nebo-li API (Application Programming Interface) pro práci s grafickou kartou (akcelerátorem). Je multiplatformní a jde ho tudíž implementovat na skoro všechny známé počítačové platformy. První verze OpenGL byla do světa vypuštěna v roce 1992 společností Silicon Graphics, Inc. (dále jen SGI). Společnost SGI byla založena v roce 1982 dvojicí Jimem Clarkem a Abbey Silverstonovou v Kalifornii. V dnešní době má pobočky na všech obydlených kontinentech. Produktem této společnosti je výpočetní technika určená k náročným, převážně grafickým operacím. Jsou to tedy servery a clustery (seskupení více počítačů pro výpočet), úložiště dat a vizualizační aplikace (např. VUE). V dnešní době je standard OpenGL spravován skupinou Khronos Group do níž patří např. Intel, AMD, Apple, Fujitsu, NVIDIA a spousta dalších. Od verze 2.0 podporuje OpenGL shadery. Shader je program pro zpracovávání dat přímo na grafické kartě. Pro OpenGL se píše v jazyku GLSL (OpenGL Shading Language) pro DirectX pak HLSL. Prozatím jsou známé tři typy shaderů. Prvním je vertex shader. Ten provádí operace na jednotlivých vrcholech, nevýhodou oproti geometry shaderu (zmíněný níže) je v tom, že vstupuje vždy jeden vertex (vrchol) a vystupuje jeden. Pixel shader se stará o manipulaci s pixelem, který dostane od vertex shaderu. V OpenGl se pixel shader označuje za fragment

shader. Hlavní funkcí tohoto shaderu je nanášení hodnoty osvětlení, textury a jiných složitějších efektů (např. bump mapping). Posledním typem shaderu je geometry shader. Stojí mezi vertex shaderem a pixel shaderem. Jeho schopností je upravit geometrii přidáním nebo ubráním vertexů. Využitím může být třeba simulace zatravnění, aplikace displacement mapy nebo efekt tessellation. Ačkoliv je síla tohoto nástroje veliká, podpora u API je až od verze 10 u DirectX a nativně od verze 3.0 u OpenGL (dříve skrze rozšíření - extensions). S příchodem shaderů se na grafických kartách začaly používat paralelní shaderovací jednotky (pipeline), podle příslušných shaderů (vertex, pixel). Tento přístup byl dost neflexibilní a v případě, že aplikace byla náročnější na práci s texturami a osvětlením, tudíž pro pixel shader jednotku, jednotka vertex shaderu zůstala zatím nevyužita. S příchodem nové generace grafických karet přišlo i řešení tohoto problému. Vznikla unifikovaná shader jednotka (unified shader pipeline), která může fungovat jako kterákoliv z uvedených. Vytížení procesoru se tak rovnoměrně rozloží.

Pro další porozumění některých pasáží textu je nezbytné vysvětlit několik další pojmů. Jedním z nich je pojem „textura“ (angl. texture). Textura je pojem z počítačové grafiky pro grafické médium aplikované na geometrii. Existují jednorozměrné, dvourozměrné, dokonce i třírozměrné, dále pak tu jsou i animované nebo parametrické textury. Největší zastoupení mají dnes pravděpodobně dvourozměrné. Základním stavebním prvkem textury je texel. Název pochází z anglické spřežky texture a element, podobně tomu je i u pixelu (picture element). Samotná aplikace textury se nazývá mapování textury. Jedná se o přiřazování souřadnic na textuře jednotlivým vrcholům geometrie. Při vykreslování se pro jednotlivé pixely výsledného obrazu interpolují souřadnice a dále se pak pracuje s barevnou hodnotou texelu na takto získaných souřadnicích. Výhoda použití textury je kvalitní visuální dojem s úsporou výpočetního času. Pro grafickou kartu je často jednodušší úkon nanést texturu na daný objekt, než provádět operace se složitou geometrií. Extrémem jsou takzvané billboardy. Jedná se o textury nanesené na čtvercovou nebo obdélníkovou plochu, která je vždy kolmá na pohledový vektor. Vlastně by se dalo říci, že se za pozorovatelem otáčí. Výpočetní čas v tomto případě byl ušetřen radikálně, ale vizuální efekt nemusí být vždy ideální. Závisí na každém programátorovi, jak této metody využije.

Textura je prvkem skupiny map. Jednou z map je i „bump“ neboli „normal“ mapa. Ta slouží k přímému nastavení normálových vektorů jednotlivým částem geometrie, obdobě jako se aplikuje texturová mapa. Rozdíl je v tom, že při nastavení textury se použije texel jako barevná složka. U normálových map se použije hodnota texelu jako normálový vektor, respektive se použijí jednotlivé barevné složky jako jednotlivé složky prostorového vektoru. Normálový vektor je potřebný pro výpočet korektního osvětlení. Ve výsledku jednotlivým pixelům připadá i normálový vektor. Způsob osvětlení pak není tak závislý na geometrii. Dojde k efektu složitější geometrie při pohybu světla. Tato technika je velmi vhodná pro aplikování na plochy, které jsou těžce viditelné pod ostrými úhly. Pokročilejší metodou vytvoření tohoto efektu je „parallax mapping“, kde dochází ještě navíc k posunutí mapovaných texelů textury vlivem normal mapping pro vytvoření dokonalejšího efektu.

2. Způsob práce ve vybraném prostředí

2.1. Práce s OpenGL

Z programátorského hlediska se OpenGL chová jako stavový automat. Pokud změníme nějaký parametr pro vykreslení scény, zůstane zachován do další změny. To znamená pokud například změníme barvu příkazem *glColor3f(...)* v nějaké funkci, barva zůstane zachovaná, i když funkci dávno opustíme. Programování v OpenGL je ryze procedurálního rázu a názvy jednotlivých procedur si drží určitou konvenci. Bude vysvětleno třeba na funkci *glColor3f*. Předpona „gl“ značí, ke které knihovně funkce náleží. Tato předpona „gl“ značí, že funkce náleží zrovna k OpenGL a tudíž všechny funkce OpenGL začínají „gl“. Dále jsou tu i předpony „glu“, „glut“ a další. Po předponě navazuje tělo názvu funkce, které symbolizuje funkčnost. Funkce *glColor3f* pracuje tedy s barvou. A koncovka funkce má význam formátu vstupních parametrů, pokud jsou všechny stejného typu. Pokud tomu tak není, koncovka se neuvádí. „3f“ má za význam tři vstupní parametry typu „float“. Pokud se jako parametr má předat pole hodnot, tvoří se koncovka tak, že za písmenem značící datový typ parametru, následuje písmeno „v“ jako vektor. V následující tabulce (Tabulka 1) je přehled jednotlivých parametrů s odpovídající koncovkou.

koncovka	Datový typ (velikost)	syntaxe jazyku C	syntaxe OpenGL
b	integer (8 bitů)	signed char	GLbyte
s	integer (16 bitů)	short int nebo int	GLshort
i	integer (32 bitů)	int nebo long int	GLint
f	float (32 bitů)	float	GLfloat
d	float (64 bitů)	double	GLdouble
ub	unsigned integer (8 bitů)	unsigned char	GLubyte
us	unsigned integer (16 bitů)	unsigned short / int	GLushort
ui	unsigned integer (32 bitů)	unsigned int / long	GLuint

Tabulka 1 – Přehled koncovek funkcí OpenGL

Pro vykreslení geometrie slouží dvojice funkcí, které ohraničují v kódu prostor pro volání funkcí k tomu určených. První, *glBegin,* začne vykreslení a *glEnd()* ukončí vykreslení. Tyto dva příkazy fungují podobně jako v C fungují „{“ a „}“ a nebo v Pascalu „begin“ a „end“. Vytyčený prostor pak může obsahovat (mimo jiné) funkce pro nastavení jednotlivých vrcholů. Vstupní parametr funkce *glBegin(parametr)* určuje typ geometrického útvaru, jenž se má začít vykreslovat. Těmto útvarům se také někdy říká geometrická primitiva. Parametr je typu enum (výčet) a může nabývat následujících hodnot.

a) Geometrie bez plochy

GL_POINTS – bod; nebo skupina bodů v závislosti na počtu zadaných vrcholů

GL_LINES – úsečky; n-tá dvojice vrcholů určuje n-tou úsečku

GL_LINE_STRIP – řetěz úseček; vrcholy n a n+1 určují n-tou úsečku

GL_LINE_LOOP – obdobně jako GL_LINE_STRIP, s tím rozdílem, že se řetěz uzavře úsečkou vedenou od posledního vrcholu k prvnímu

b) Geometrie s plochou

GL_TRIANGLES – trojúhelníky; n-tá trojice vrcholů určuje n-tý trojúhelník

GL_TRIANGLE_STRIP – pruh trojúhelníků; tři po sobě jdoucí vrcholy určují trojúhelník

GL_TRIANGLE_FAN – trs trojúhelníků; dvojice po sobě jdoucích vrcholů určuje společně s prvním zadaným vrcholem trojúhelník

GL_QUADS, GL_QUAD_STRIP – čtyřúhelník a pruh čtyřúhelníků; obdobně jako ekvivalentních parametrů pro trojúhelník s rozdílem, že je třeba zaručit konvexnost úhlů čtyřúhelníku a to, že jednotlivé vrcholy leží v jedné rovině. Pokud tomu tak není, geometrie se patrně nevykreslí správně.

GL_POLYGON – n-úhelník(polygon); zadané vrcholy reprezentují vrcholy polygonu a jako takové musí ležet v jedné rovině a výsledný polygon musí být konvexní

Jednotlivé vrcholy a jejich vlastnosti zadáváme příkazy glVertex, glColor, glNormal, glIndex, glTexCoord do prostoru vytyčeného glBegin a glEnd. Funkce ve výčtu nejsou všechny, které mohou být v tomto prostoru, ale rozhodně jsou zásadní pro zobrazení. Dále bude popsán jejich význam.

glVertex – souřadnice vrcholu a váha (x,y,z,w); vstupní parametry mohou být typy GLfloat, GLdouble, GLint, GLshort (nebo příslušná pole) jejich počet může být 2 až 4, výchozí hodnoty z souřadnice je 0 a váhy w je 1

glColor – podkladová barva vrcholu RGBA formátu (r,g,b,a); vstupní parametry mohou být typy GLfloat, GLdouble, GLudouble, GLint, GLuint, GLshort, GLushort (nebo příslušná pole), jejich počet je buď 3 nebo 4, výchozí hodnota pro alpha je 1, nabývá výsledné hodnoty v rozmezí <0; 1>, proto jsou hodnoty celočíselných typů namapovány na toto rozmezí

glNormal – složky normálového vektoru (x,y,z); vstupní parametry mohou být typy GLbyte, GLdouble, GLint, GLshort (nebo příslušná pole), jejich počet je vždy 3, jednotlivé složky nabývají hodnoty v rozmezí <-1; 1>, proto jsou hodnoty celočíselných typů namapovány na toto rozmezí

Normálový vektor slouží pro výpočet nasvícení objektu, je to vektor kolmý k osvětlované rovině. Pro správný výpočet je třeba, aby byl vektorem jednotkovým. Toho lze docílit voláním funkce glEnable(GL_NORMALIZE).

glIndex – barva vrcholu v barevné mapě; v dnešní době se prakticky nepoužívá, protože všechny soudobé grafické karty pracují v režimu true color

glTexCoord – souřadnice vrcholu na textuře a v čase (s,t,r,q); pro nejrozšířenější typ textur, a to 2D, nám budou stačit první dva parametry, vstupní parametry mohou být typu GLfloat, GLdouble, GLint, GLshort a jejich počet 1 až 4

```
glBegin(GL_LINES);
            glColor3f(0.5f, 0.5f, 0.5f);
      for(int i=-10;i<=10;++i) {
            glVertex3f(i,0,-10);
            glVertex3f(i,0,10);
            glVertex3f(10,0,i);
            glVertex3f(-10,0,i);
      }
glEnd();
```

Ukázka kódu 1 – Vykreslování primitiv v OpenGL

V ukázce (Ukázka kódu 1) jsou shrnuty poznatky z předešlého textu. Jedná se o úryvek kódu z vykreslovací funkce (*drawScene*), který vykresluje základní síť. Barva je nastavená na šedou, poté se cyklem vykreslí dvacet dvojic úseček. Jedna z dvojice je rovnoběžná s osou x, druhá je rovnoběžná s osou z.

2.2. Práce s GLUT

OpenGL Utility Toolkit neboli GLUT je rozšiřující knihovna pro OpenGL. Hlavní funkcí této knihovny je rozšířit API o přehledné funkce pro práci s vykreslovacím oknem, s událostmi klávesnice a myši. Tyto funkce v samotném OpenGL chybí z důvodu zachování kompatibility s různými operačními systémy. Mimo to poskytuje funkce pro vykreslení jednoduchých základních geometrických těles (koule, kužel, válec i „slavná“ konvice) a jednoduché pop-up menu. GLUT knihovna však skýtá určitá omezení. Knihovna požaduje po uživateli spuštění funkce *glutMainLoop(),* což je funkce, která skončí v nekonečné smyčce. Kód, který bude napsán „po ní“, nebude nikdy vykonán. Funkce se ukončí, až když uživatel zavře vytvořené okno. Což je třeba mít na paměti. Existují různé metody, jak tento problém obejít. Například lze

vykreslovací smyčku nechat spustit v jiném vlákně, pak ale je třeba synchronizovat jednotlivá vlákna. Dalším způsobem je odchytit událost zavření okna. K inicializaci GLUT knihovny slouží funkce začínající *glutInit**. Hlavní z nich je *gluInit*, ta by měla být v aplikaci volána pouze jednou a žádná z ostatních funkcí GLUT a nebo OpenGL by neměla být volána před ní.

glutInit - přejímá inicializační parametry funkce main, iniciuje GLUT

glutInitWindowSize – nastavuje výchozí velikost okna

glutInitWindowPosition - nastavuje výchozí pozici okna

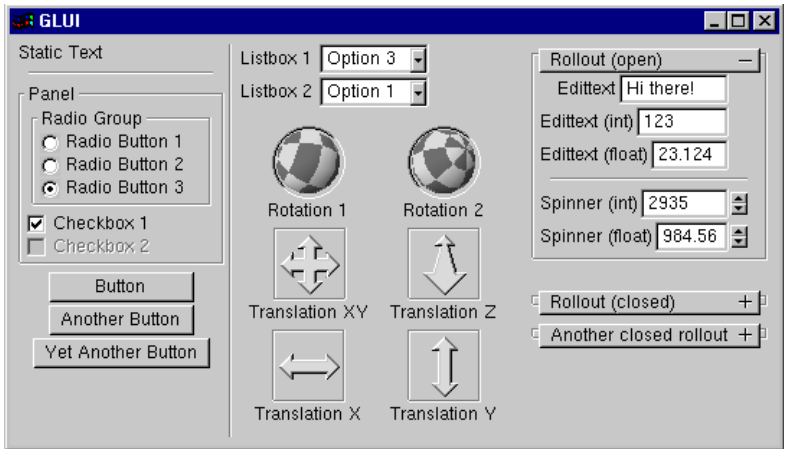
glutInitDisplayMode – nastavení zobrazovacího módu

glutCreateWindow – vytvoří okno pro vykreslení

Dále je potřeba zaregistrovat callback funkce pro události myši, klávesnice, změnu velikosti okna, překreslení okna a jiné. To lze pomocí GLUT funkcí *glut*Func(parametr)*, kde parametr je název volané funkce. A v poslední řadě zavoláme funkci *glutMainLoop()*.

2.3. Práce s GLUI

Další rozšiřující knihovnou, jenž byla použita a je nutné si jí představit, je A GLUT-Based User Interface Library (zkráceně GLUI). GLUI je silný nástroj vytvořený pomocí knihovny GLUT a jazyku C++. Tato knihovna s sebou přináší jednoduchý přístup k vytváření dalších GLUT oken obsahující formulářové prvky tak, jak je známe z operačních systémů. Vytváření editovatelných polí, tlačítek, panelů nebo seznamů je otázkou několika řádků kódu. Od verze 2.0 pak podporuje (mimo jiného) rozbalitelné panely, vytváření podoken nebo více dimenzionální polohovací prvky.



Obrázek 1 – Ukázka ovládacích prvků knihovny GLUI

Velkou výhodou GLUI je to, že je celá napsaná pomocí příkazů GLUT a je tudíž platformově nezávislá. Dalším zajímavým prvkem této grafické knihovny je používání „živých proměnných“ (live variables). Při vytváření nového ovládacího prvku se v parametru předá odkaz na proměnnou v programu. Při změně hodnoty ovládacího prvku se pak tato změna přímo promítne na proměnné. Navíc pokud dojde ke změně hodnoty nějakého ovládacího prvku, GLUI vynutí překreslení hlavního vykreslovacího okna. V případě, že se změní přímo hodnota živé proměnné, slouží k tomu synchronizační funkce *sync_live* a *sync_live_all,* aby se změny projevily zpětně na ovládacích prvcích. Posledním způsobem, jak spravovat hodnoty ovládacích prvků, jsou callback funkce a konstantní hodnoty přiřazené jednotlivým prvkům.

```
subW1->add_button_to_panel(
      rollData,"Save heights",
      GLUI_SAVE_HEIGHTS, glui_callback
);
```

Ukázka kódu 2 – Vytvoření ovládacího prvku s callback funkcí

V ukázce kódu (Ukázka kódu 2) je volána metoda pro vytvoření tlačítka v panelu. Hodnota rollData je odkaz na objekt panelu, do kterého se má nové tlačítko vložit. „Save heights“ je popisek tlačítka, GLUI_SAVE_HEIGHTS je konstatní hodnota předaná callback funkci, jejíž jméno je předáno v posledním parametru (glui_callback). Tlačítko má za účel uložit hodnoty výšek terénu do souboru.

```
subW2->add_checkbox("Toggle cross",&visibleCross);
```

Ukázka kódu 3 – Vytoření ovládacího prvku s „živou“ proměnnou

V této ukázce (Ukázka kódu 3) dojde k vytvoření zaškrtávacího pole s popiskem „Toggle cross“ a o jehož správu se bude starat živá proměnná visibleCross, jejíž adresa je předána jako poslední parametr. Tento ovládací prvek přepíná viditelnost středového kříže kamery. Pro správnou práci knihovny GLUI je třeba zaregistrovat callback funkce pro jednotlivé události myši, klávesnice a jiné. Obdobně tomu bylo u knihovny GLUT. V tomto případě přejímá ošetření událostí

knihovna GLUI a registrování callback funkcí u GLUT není nadále potřeba. K zaregistrování se použije hlavní objekt knihovny GLUI_Master a příslušné metody. V další ukázce (Ukázka kódu 4) jsou postupně zaregistrovány callback funkce pro vykreslení scény, události klávesnice, události myši a změnu velikosti okna.

```
GLUI_Master.set_glutDisplayFunc(drawScene);
GLUI_Master.set_glutKeyboardFunc(handleKeyPress);
GLUI_Master.set_glutMouseFunc(handleMouse);
GLUI_Master.set_glutReshapeFunc(handleResize);
```

Ukázka kódu 4 – Registrace callback funkcí

To by měly být snad všechny informace potřebné pro pochopení práce s GUI.

2.4. Práce s „stbi a Vec3f“

Nyní budou představeny zdrojové kódy, jenž byly použity pro ulehčení nebo zpřehlednění práce. V zásadě jsou dva. První je kód pro práci s obrázky v různých formátech. Původně byl součástí jiné knihovny pro práci s texturami (SOIL). Jedná se o soubory *stb_image_aug.** a na ně vázané soubory (*stbi_DDS_aug_c.h* a *stbi_DDS_aug.h*), jejichž funkcionalita nebyla vlastně využita. Z tohoto zdrojového kódu (knihovny) byly použity vlastně jenom tři funkce. Význam těchto funkcí bude popsán níže.

```
unsigned char *img = stbi_load(filename, &x, &y,&n, 1);
……
stbi_image_free(img);
```

Ukázka kódu 5 – Využití funkcí stbi pro načtení

Na prvním řádku v ukázce (Ukázka kódu 5) je první ze zmíněných funkcí. Jak její název napovídá, slouží k načítání obrázku. Podporované formáty jsou JPEG, BMP, TGA, PNG, PSD a HDR (ovšem s nějakými omezeními). Jako parametry přejímá cestu k souboru (filename), adresy proměnných, do kterých budou uloženy informace o rozměrech (&x, &y) a počet složek na pixel (&n). Obvykle bývají 1, 3 nebo 4. Pokud je jedna složka na pixel, jedná se o monochromatický obrázek, pokud jsou 3, má obrázek všechny tři barevné složky. Čtvrtá složka je „průhlednostní barva“ neboli alfa kanál. Poslední parametr určuje počet

požadovaných složek. Tato ukázka je vyňata z funkce pro načítání výškové mapy, proto je poslední parametr roven 1 (pouze 1 výška). Funkce na posledním řádku ukázky uvolní data z paměti.

```
if (stbi_write_bmp(filename,tw,th,3, pixely)){
      ……
      ……
};
```

Ukázka kódu 6 – Využití funkcí stbi pro uložení

Poslední nezmíněná funkce má za úkol uložit obrázek ve formátu BMP s rozměry tw a th, s počtem složek na pixel 3 (Ukázka kódu 6). Ukázka je součástí funkce pro uložení výřezu vykreslovacího okna. Poslední parametr jsou samotná data. Ještě existuje jedna ukládací funkce v tomto kódu, která ukládá obrázek ve formátu TGA. BMP bylo zvoleno pro velikou podporu a obecnou jednoduchost formátu.

Jak již bylo uvedeno výše, jsou v práci použity dva externí kódy. Tím druhým je *vec3f.**, který zjednodušuje práci s prostorovými vektory. Kód deklaruje datový typ třírozměrného vektoru. Obsahuje funkce pro získání velikost vektoru, normalizaci vektoru, skalární a vektorový součin. Navíc jsou dodefinovány operátory součtu, součinu, rozdílu a podílu. Hlavní přínos měl tento datový typ pro výpočet normálových vektorů terénu.

3. O programu

3.1. Představení projektu a práce s ním

Původní vidina byla vytvořit aplikaci, jenž by kombinovala užitkovou část aplikace s čistě estetickou částí. Užitkovou částí je myšleno to, že by ze vstupních dat, hodnot, přidáním uživatelova snažení se vytvořila nová data, hodnoty. Co se týče výsledné podoby aplikace, tak vstupní hodnoty jsou obrazová data, soubory. Snažení uživatele, práce, je aplikování jednotlivých výškových map, ať už ze vstupních hodnot (souborů), nebo generovaných náhodně. Dále pak to je pozicování textury, změna zdroje pro textur, změna barevného povrchu, pozicování světla a jeho editace jeho barevné složky, manipulace s kamerou. Výstupními hodnotami je výšková mapa odpovídající terénu, normálové vektory

jednotlivých vrcholů a statický obraz vizualizace. Vezmeme-li pak estetickou část, je to právě uživatelské rozhraní.

Nyní bude představen samotný projekt a manipulace s ním, případně budou nastíněny výsledky, kterých lze prací s aplikací dosáhnout. Projekt má dvě okna. První okno je konzolové a slouží pouze pro výpis nejrůznějších informací, ať už o úspěchu nebo neúspěchu té které činnosti. Druhé okno je okno OpenGL a tvoří jádro aplikace a rozhraní pro přístup uživatele.

Klávesové zkratky jsou zavedeny celkem čtyři. Klávesa „escape“ ukončí aplikaci. Klávesa „u“ skryje uživatelské rozhraní, tzv. UI (User Interface). Jsou to všechny viditelné prvky vytvořené knihovnou GLUI již několikrát zmíněnou. Klávesa „c“ přepíná režim kamery mezi „volnou kamera“ a „kamerou s cílem“. A klávesa „p“ uloží výřez pro vizualizaci do souboru. To lze i tlačítkem. Důvod této klávesové zkratky je zřejmý a to ten, že pokud bude skryto UI, nelze stisknout tlačítko. Pokud některá klávesová zkratka nespustí tu kterou úlohu, je možné, že klávesový vstup je přejímán od některého formulářového pole. Kliknutím myši na část okna s vizualizací by mělo tento drobný problém vyřešit.

Vstupy myši by se daly rozdělit na dvě skupiny, vstupy na výřezu vizualizace a vstupy na UI. V první skupině se jedná o ovládání kamery. Se stisknutým levým tlačítkem ovládá pohyb myši otáčení kolem cíle kamery (ohniska). V režimu volné kamery otáčí kamerou okolo její pozice. Prostřední tlačítko zajišťuje pohyb kamery po osách. Pravé tlačítko s následným pohybem myši nahoru nebo dolů přibližuje nebo oddaluje kameru od ohniska. V případě, že se jedná o volnou kamera, posouvá kamerou vpřed a vzad. Vstup myši na UI nemá smysl popisovat snad až na polohovací ovládací prvek symbolizovaný šipkami. Ovládání je na nich intuitivní.

Význam jednotlivých ovládacích prvků bude nadále popsán. Jsou rozděleny do dvou majoritních skupin. Prvky na dolní liště jsou v první skupině a jsou tvořeny čistě zaškrtávacími poli.

Autorotate – zapíná / vypíná automatickou rotaci kamery okolo ohniska

Toggle grid – zapíná / vypíná zobrazení orientační sítě

Toggle cross - zapíná / vypíná zobrazení kříže symbolizujího ohniska

Toggle fullscreen - zapíná / vypíná celoobrazovkový režim

Druhá skupina je znatelně početnější a proto jsou prvky v ní rozděleny do tématických podskupin. Jednotlivé podskupiny jsou utříděny v rozbalovacích panelech.

Panel s popiskem „Terrain generation“ obsahuje prvky pro práci s fyzickou podstatou terénu. Skupina „Method“ obsahuje přepínací prvky, které slouží k nastavení způsobu aplikace další modifikace terénu.

Set - nastaví nové hodnoty, všechny staré se přepíší

Add - nové hodnoty přičtou k původním

Substract – nové hodnoty se odečtou od původních

If lower – nové hodnoty se nastaví pouze tam, kde jsou nižší než původní

If higher – nové hodnoty se nastaví pouze tam, kde jsou vyšší než původní

Vstupní pole s popiskem „Z limit“ určuje, jako sílu bude mít nová modifikace, respektive určuje její výškový rozsah. Rolovací seznam s popiskem „Map“ slouží k výběru výškové mapy ze souboru. Výpis obsahuje pouze soubory s koncovkou BMP nebo JPG, které jsou v adresáři „heightmaps“. Pokud je seznam prázdný, v adresáři nejsou soubory splňující tato kritéria. Není nutno však vypínat aplikace pro aktualizace seznamu, k tomu slouží tlačítko s popiskem „Refresh lists“ (o tom níže). Dvě tlačítka v tomto bloku slouží k aplikaci mapy. První, s popiskem „Load map“, aplikuje mapu ze souboru vybraného v seznamu výše. Druhé, s popiskem „Random map“, vygeneruje náhodou mapu a následovně aplikuje. Poslední ovládací prvek, vstupní pole s popiskem „Max. height“, nastavuje maximální výšku celého terénu. Vnitřní funkce posune celý terén tak, aby byla nejnižší hodnota rovna nule. Proto je maximální výška současně i celkovým rozsahem terénu.

Skupina s popiskem „Texture and colors“ obsahuje prvky pro nastavení barevného nebo texturového pokrytí (nebo obou dvou). V levé části panelu jsou nastavení dvou barev pro barevné pokrytí. Zaškrtávací pole zapíná popřípadě vypíná barevné pokrytí. Dvě skupiny vstupních polí pak nastavují barvu. Horní skupina nastavuje barvu v dolních výškách terénu. Barva je zadávaná ve formátu RGB tudíž červená, zelená, modrá. Hodnoty jsou vždy v rozsahu od 0 do 255 včetně. Dolní skupina polí nastavuje barvu v horních výškách terénu. Obě barvy jsou na terénu lineárně interpolovány. Pravá část panelu „Texture and colors“ slouží k nastavení textury a jejího mapování. Zaškrtávací pole, podobně jako u barevného pokrytí, zapíná respektive vypíná texturové pokrytí. Výběr s popisem

„File“ slouží k výběru souboru pro načtení textury. Při změně výběru je nová textura okamžitě načtena. Seznam obsahuje soubory s koncovkou BMP nebo JPG v adresáři „textures“. Polohovací prvek s popisem „Offset XY“ ovlivňuje posunutí textury v osách X a Y (vzhledem k textuře v osách U a V). Vstupní pole „Tiling X“ a „Tiling Y“ nastavují opakování textury v jednotlivých osách. Zadané hodnoty jsou v mezích 0 až 100. Nepředpokládá se, že by uživatel chtěl aplikovat texturu mimo tyto meze. Pokud ano, je vždy možnost upravit texturu v nějakém grafickém programu.

Skupina „Light“ slouží k jednoduchému nastavení světla, zejména pak jeho pozice a barevné složky. Blok s popiskem „Color“ nastavuje právě barevnou složku. Význam a použití jednotlivých podčástí je shodný s s bloky „Low color“ a „High color“. Zaškrtávací pole s popisem „Toggle light model“ zapíná respektive vypíná zobrazení fyzické interpretace světla. Ta slouží pouze pro přehled a byla pro ní zvolena bílá „koule“ (jedná se o kvadrik a ten, jako takový, není ideální koulí). Další dvě polohovací zařízení slouží pro nastavení pozice světla. Není třeba snad nadále popisovat. Vhodné je akorát uvést, že výška světla je předpokládaná kladná. Při manipulaci s pozicí světla se zobrazuje fyzická interpretace pro lepší přehlednost a zobrazuje se i červený „kužel“, jehož špička se dotýká roviny definované osou X a Y. Důvodem je rovněž lepší přehlednost.

Posledním skupinou jsou prvky pro ukládání dat a správu „Data output“. Tlačítko „Save heights“ uloží výšky současného terénu do souboru, jehož jméno se přebírá ze vstupního pole s popiskem „Filename“. Není třeba vyplňovat koncovku, výstupní formát je bitmapa a proto aplikace sama doplní koncovku na BMP. Soubor bude uložen v adresáři „heightmaps“. V případě, že je zaškrtnuto pole „save normals with heights“, uloží se společně s výškami normálové vektory. Rovněž ve formátu bitmap (do stejného adresáře) s rozdílem, že název bez koncovky bude končit „-normals“. Tlačítko „Save screen“ má stejnou funkci jako klávesová skratka „p“. Uloží výřez pro vizualizaci do souboru ve formátu bitmap do adresáře „screenshots“. Pro název souboru se použije opět hodnota vyplněná v poli „Filename“.

Následující text popíše příklad použití aplikace. Pokud po spuštění aplikace zjistíme, že nejsou žádné výškové mapy v příslušném adresáři, tlačítkem „Random map“ vygenerujeme nové. Postupně je ukládáme pod různými názvy.

Stiskem tlačítka „Refresh lists“ obnovíme seznamy souborů textur a výškových map. Dále aplikujeme uložené mapy na současný terén s různou sílou efektu, jenž lze měnit ovládacím prvkem „Z limit“. Pro uložení maximálního výřezu nejprve uvedeme jméno souboru pro uložení („Filename“), zapneme případně vypneme jednotlivé prvky zobrazení (texturu, barevné pokrytí, orientační síť aj.). Přepneme do celoobrazovkového režimu („Toggle fullscreen“), klávesou „u“ skryjeme UI. Nastavíme vhodně kameru. K tomuto úkonu můžeme použít buď kameru s „cílem“, která je vhodnější pro nastavení oddálenější kamery, nebo „volnou“ kameru, která je vhodnější pro nastavení scény blíže k terénu. Nakonec klávesou „p“ uložíme obraz do souboru. Je třeba mít na paměti, že formát výstupního souborů je bitmap, což je nekomprimovaný formát a z toho důvodů náročný na místo na disku.

3.2. Struktura kódu a důležité algoritmy

Do této chvíle byla samotná aplikace popisována z pohledu uživatele, nyní však bude rozebrána z programátorské perspektivy. Nejprve představíme jednotlivé zdrojové soubory, jejich obsah a význam pro chod celé aplikace.

3.2.1. Soubor „terrainGen.cpp“

Vstupním bodem aplikace je právě tento soubor a proto k tomuto souboru není hlavičkový soubor (*.h). Jelikož je vstupním bodem, obsahuje funkci main, která je vstupním bodem všech konzolových aplikací vytvořených v jazyce C++. Struktura kódu v tomto souboru je následující. Nejprve se vnoří hlavičkové soubory, dále jsou deklarovány konstanty použité pro práci s GLUI knihovnou. Poté jsou deklarovány nebo incializovány proměnné pro práci s GLUI, případně se jedná o pointery na jednotlivé ovládací prvky, jenž jsou používány globálně v celém dokumentu. Navazuje inicializace samotného objektu terénu („teren“), kamery („kamera“) a objektu světla („svetlo“) s proměnnými potřebnými pro ovládání vstupu myši. Zbytek kódu tvoří funkce, které si blíže popíšeme.

Funkce **takeScreenshot** slouží k ukládání výřezu vizualizace do souboru. Ve vstupním parametru přebírá název souboru, do kterého má být obrázek uložen. Návratová hodnota funkce je typu boolean a hodnota true je vrácena v případě úspěšného uložení souboru. V opačném případě je vrácena hodnota false. Co se týče těla funkce, tak nejprve jsou načteny rozměry výřezu. V případě, že je

viditelné UI, zavolá se funkce GLUI pro navrácení rozměrů bez podoken vytvořených GLUI. Poté získány jednotlivé pixely skrze funkci OpenGL. Jelikož jsou pixely vertikálně převráceny, jsou použity dva vnořené cykly pro otočení. Nakonec je výsledek uložen do souboru pomocí příkazu z externího kódu.

Funkce **handleResize** je callback funkcí, jenž je použita pro změnu velikosti okna. Přejímá dva vstupní parametry a to novou šířku a výšku okna. Pokud je viditelné UI, hodnoty pro nastavení viewportu jsou nastaveny funkcí knihovny GLUI. Zbytek těla funkce nastavuje (resp. resetuje) OpenGL parametry pro vykreslení okna výřezu okna.

Funkce **handleKeyPress** je rovněž callback funkcí. Slouží pro správu událostí klávesnice. Zde jsou ošetřeny klávesové skratky („p“, „u“ a „esc“). Za rozebrání stojí snad jen ošetření skratky pro skrývání UI, kde je větvení podle živé proměnné „hiddenUI“. Podle té se skryje nebo objeví UI. Poté je zavolána funkce handleResize s aktuálními rozměry okna. Ty jsou uchovávány v globálních proměnných. „handleResize“ je zavoláno kvůli resetování okna a správnému vykreslení vizualizačního výřezu.

Další funkce, **initRendering,** slouží k inicializaci funkcí OpenGL. Většinou jsou to funkce, resp. stavové proměnné OpenGL, které se za běhu aplikace dále nemění. Součástí funkce je i inicializace pole vertexů (vertex array), což je prostředek OpenGL pro ukládání dat do paměti grafikcé karty pro rychlejší práci s nimi.

Funkce **drawScene** obstarává samotné vykreslování. Jedná se opět o callback funkci. Nejprve se vyčistí barevný a hloubkový buffer, nastaví se transformační matice na jednotkovou a typ matice na modelovou. Dále se pak volají vykreslovací metody jednotlivých objektů (teren, svetlo, kamera). Následuje podmíněné vykreslení poloprůhledného kuželu ke světlu o němž již bylo napsáno, nebo také orientační sítě. Funkce *glutPostRedisplay* zavolá znovu funkci pro překreslení. Proto je v podmínce s automatickou rotací.

Funkce **handleMotion** je callback funkcí pro ošetření pohybu, konkrétně pohybu myši. Do globálního vektoru (pole) *Buttons* se ukládá, které tlačítko je stisknuto. Poté jsou upravovány vlastnosti kamery. Nakonci je volána GLUT funkce pro překreslení v případě vypnuté autorotace (pokud je zapnutá, volá se vždy na konci vykreslující smyčky).

Callback funkce **handleMouse** se stará o vstupy myši. Do globálního pole „buttons“ zaznamenává, které tlačítko myši bylo stisknuto a ukládá aktuální pozici pro pozici myši. O vlastní pohyb kamery se pak stará předešlá funkce.

Funkce **loadTexture** je funkcí pro načtení textury ze souboru. Nejprve se načtou data ze souboru pomocí funkce stbi_load. Při úspěchu se pomocí příkazů OpenGL vytvoří textura a nastaví se její parametry. Nakonec se předají obrazová data pro texturu. Funkce vrací neznaménkový integer (GLuint), jenž identifikuje texturu v paměti grafické karty.

Funkce **loadTargetFiles** pracuje s objektem *filenameArray_* , jenž je datového typu vector. Jedná se o datový typ pro snadnější a bezpečnější práci s dynamickým polem. Kromě tohoto objektu je tento typ ještě použit pro ukládání dat třídy terénu. V objektu *filenameArray_* jsou uloženy jména souborů textur a výškových map a o který typ z nich se jedná. Funkce nejprve nastaví výchozí adresář na adresář s texturami (textures), prohledá ho a nalezené soubory s koncovkou „jpg“ přidá do objektu *filenameArray_*. To samé opakuje s koncovkou „bmp“. Po skončení těchto dvou cyklů opakuje celý proces ještě pro adresář s výškovými mapami. Funkce v podstatě získává jména použitelných souborů z obou adresářů a ukládá je do pole.

Funkce **GLUI_refreshLists** má za úkol aktualizovat oba seznamy, jak seznam textur tak seznam výškových map. Na začátku oba seznamy vyčistí. Poté vyčistí i pole jmen souborů (*filenameArray_*). Dále zavolá výše zmíněnou funkci loadTargetFiles, čímž naplní pole jmény souborů. Toto pole cyklem projde a přidá nové prvky do seznamů podle příznaku (type). V případě, že se jedná o první prvek seznamu, použije ho jako prvek, jenž byl vybrán. Pokud se jedná o texturu, pak jí rovou načte.

Funkce **glui_callback** je jediná funkce starající se o události ovládacích prvků UI. V zásadě jde o jedno větvení (switch) v závislosti na vstupním parametru. Tím je vždy jedna z konstant UI, deklarovaných na začátku kódu. Konstanta signalizuje, který prvek nebo skupina prvků (jak tomu je u nastavení barvy) tuto funkci zavolal. Jednotlivé podvětve tohoto větvení nemá velký význam popisovat. Zmínění si zaslouží snad jen dvě z nich. První je případ, že funkci zavolalo tlačítko pro uložení výšek. V této části kódu se (mimo jiného) vytvoří pole pro zápis, zjistí se celkový výškový rozsah terénu. Poté se všechny výšky přepíší do tohoto pole ze starého rozsahu <0; celkový výškový rozsah>

do nového rozsahu <0 ; 255>. Následně dojde k uložení souboru s nastavením jedné složky na pixel (monochromatický obrázek). Podobně je nakládáno s normálovými vektory, pokud je uživatel zvolil zapsat. Jednotlivé složky jsou ze starého rozsahu <-1; 1> převedeny do nového rozsahu <0; 255>. Nová data se pak rovněž uloží do souboru, tentokrát tři složky na pixel. Výsledný obrázek je nazelenalý, to je z důvodu, že složka „Y“ normálového vektoru je vždy kladná. Té odpovídá zelená barva (xyz - rgb). Proto je zelená barva v obrázku dominantní. Co se týče os, v aplikaci byly zvoleny osy X a Y za horizontální a osa Z za vertikální. Ovšem v OpenGL je osa Z braná jako osa hloubková vzhledem k pohledu. To je i důvod, proč se někdy označuje hloubkový buffer za z-buffer. Druhým slíbeným případem je přepínání celoobrazovkového režimu. V tomto úseku se stranou ukládá aktuální velikost okna při přechodu do celoobrazovkového režimu, aby se při návratu nastavila původní velikost. To je nutné z důvodu kolize při skrývání UI.

Poslední nezmíněnou funkcí zůstává už jen funkce **main**. Význam této funkce by už měl být zřejmý, proto se podívejme rovnou na její obsah. Nejprve se nastaví výchozí hodnota pro generování náhodných čísel. Dále se nastaví pozice kamery a přepočítají se hodnoty vertexového pole a normálových vektorů. Načte se pole souborů pro seznamy. Iniciuje se okno OpenGL a zaregistrují se callback funkce přes objekt knihovny GLUI. Poté už se generují ovládací prvky knihovny GLUI a zajišťuje se jejich správné umístění pomocí panelů. Nakonec se spustí nekonečná smyčka funkcí *glutMainLoop*.

3.2.2. Soubor „light.cpp a light.h“

Tyto dva soubory obsahují deklarace a definice třídy „Light“. Tato samotná třída není příliš komplexní, obsahuje několik vlastností a metod. Vlastnosti jsou vlastně jen dvě, obě dvě jsou typu Vec3f (vektor o třech složkách). Jedná se o pozici a barvu světla. Obě jsou se signaturou private a jsou tudíž přístupné pouze v rámci této třídy. Metod je víc než dvě, ale dají se dělit na dvě skupiny. Ty které pracují s vlastnostmi a ty zbylé. Ty, které pracují s vlastnostmi, začínají „get“ nebo „set“ a jejich cílem je vracet hodnotu příslušné vlastnosti nebo jí měnit. Ty zbylé jsou dvě. „Glinit“ slouží k inicializaci světla. Obsahuje vlastně jen jeden OpenGL příkaz na zapnutí světla LIGHT0. Tato metoda je ve třídě Light spíš kvůli pozdější práci se světlem. Poslední metodou a jedinou metodou, která bude

probrána blíže je *GlOutput*. Metody s názvem GlOutput jsou ještě obsaženy ve třídách *Terrain* a *Camera*. Jsou použity uvnitř těla funkce *drawScene*. Mají za účel zjednodušit tělo této funkce a využít vlastnosti. Tělo metody GlOutput náležící ke třídě Light obsahuje nastavení OpenGL modelu světla, jeho ambientní složky, difúzní složky a pozice světla. Přebírá jeden vstupní parametr typu boolean, který rozlišuje, zda se vykreslí fyzická interpretace světla či nikoliv. Pokud se jedná o konstruktor třídy *Light*, nastavuje pozici světla dle vstupního parametru a barvu na čistě bílou.

3.2.3. Soubor „camera.cpp a camera.h“

U této dvojice souborů se jedná o třídu *Camera*. Obecně jsou dva typy kamer. Kamera prvního typu sleduje nějaký objekt, v tomto textu označována jako „kamera s cílem“. Kamera druhého typu žádný objekt nesleduje a je proto označována za „volnou kameru“. Oproti předchozí třídě je tato o trochu složitější. Obsahuje „private“ vlastnosti pro uložení pozice a otočení, ty jsou typu *Vec3f*. Dále je zde ukazatel na proměnnou autorotace. Booleanovská proměnná *targetCam*, která nese informaci o tom, o kterou kameru ze dvou typů se jedná, který mód kamery je aktivní. A poslední vlastností je *zOffset*, který má smysl pouze pro kameru s cílem. Určuje vzdálenost od cíle. Konstruktor nastavuje jako výchozí kameru s cílem se vzdáleností 30 jednotek od cíle. Ve vstupním parametru přebírá ukazatel na proměnnou, jenž řídí automatickou rotaci. Tím se dostáváme k metodám. Metody pro přímé nastavování parametrů nebo získávání jejich hodnot (set* a get*) nebudou pro jejich triviálnost rozebírány. Metoda Rotate a Translate patří taktéž k těm triviálnějším. Jejich účelem je inkrementovat vektor rotace a pozice o zadanou hodnotu. Jsou zmíněné jako příklad využití deklarace operátorů (viz Ukázka kódu 7).

```
void Camera::Translate(Vec3f point){
      position += point;
}
```

Ukázka kódu 7 – Metoda Translate třídy Camera

Přeposlední zmíněnou metodou bude metoda pro přepínání typu kamery (*toggleCamType*), která jednoduše neguje proměnnou *targetCam*. Jak se dá již tušit, tou poslední je GlOutput. Vstupní parametr přejímá informaci, zda vykreslit

kříž pro znázornění cíle kamery. Tělo metody obsahuje transformační OpenGL funkce. Proto je možná na místě objasnit, jak se pracuje s kamerou v OpenGL. Kamera v OpenGL je abstraktní pojem. Pokud chceme posunout „kameru“ v nějakém směru, posuneme vše ostatní ve směru opačném. Pokud tedy chceme pootočit kameru kolem jejího ohniska, nejprve posuneme vše směrem od kamery a potom vším pootočíme (viz Ukázka kódu 8).

```
glTranslatef(0,0,-zOffset);
……
glRotatef(rotation[0], 1.0f, 0.0f, 0.0f);
glRotatef(rotation[1], 0.0f, 1.0f, 0.0f);
glRotatef(rotation[2], 0.0f, 0.0f, 1.0f);
```

Ukázka kódu 8 – Funkce OpenGL pro práci s maticí

To je i důvodem, proč všechny posuny jsou se znaménkem mínus.

3.2.4. Soubor „terrain.cpp a terrain.h“

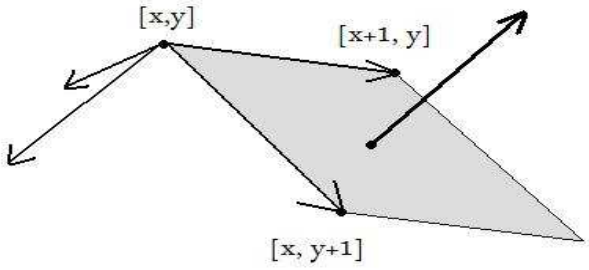
Oproti předešlým souborům obsahují tyto více jak jednu třídu. Existuje určitá konvence, že každá třída by měla mít vlastní hlavičkový soubor (případně cpp). Zde není dodržena. Nově definované třídy a datové struktury jsou velmi jednoduché, proto na přehlednosti kódu neuberou. Většina z nich se navíc týká výhradně třídy Terrain, což byl další důvod, proč je ponechat na tomto místě. Popsány budou nejdříve tyto datové struktury. „FileTarget“ slouží pro uchovávání informací o souborech. Ukládá jméno souboru, typ a pořadové číslo. Typ rozlišuje, zda se jedná o texturu nebo výškovou mapu. Pořadové číslo slouží pro správnou funkci GLUI seznamů. „Vertex_VNTC“ je datový typ, struktura, která je použita pro vytvoření pole terrainArray2. To slouží pro rychlejší vykreslení pomocí prostředku OpenGL zvaného „vertex array“. Tento typ obsahuje složky pozice, normálového vektoru, souřadnic textury a barvy. Dalším datovým typem je třída Texture (pro práci s texturou). Vlastnostmi jsou posunutí textury, opakování textury a id. Konstruktor, zároveň jediná metoda, přebírá v parametru id textury vytvořené OpenGL. Nastavuje hodnoty vlastností tak, že textura je rovnoměrně roztažena po celém terénu. To znamená bez posunutí a s jedním opakováním. Další datovým typem je *TerrainPoint*. Jedná se o stavební kámen terénu. Obsahuje vlastnosti pro uložení výšky, pozice v poli a příznaku editace (*changing*). Poslední jmenovaná vlastnost slouží pro modifikační algoritmy.

Konstruktor přebírá hodnoty pro všechny vlastnosti a následně je nastavuje. Nyní bude představena poslední a také nejrozsáhlejší třída celého programu.

Třída *Terrain* slouží pro práci s terénem a je tudíž stěžejní třídou celé aplikace. Je od ní sice odvozen pouze jediný objekt (stejně tak od třídy *Camera* a *Light*), ale do budoucna se předpokládá, že jich bude potřeba odvodit více. Nejdříve představíme vlastnosti s přístupovými právy „private“. Jedná se o posunutí celého terénu, rozchod mezi jednotlivými vrcholy v osách X a Y, rozlišení terénu, dvě barvy pro barevné pokrytí, minimální výška a maximální výška. Dále to je proměnná, která uchovává informaci o tom, zda je pole normálových vektorů aktuální, a konečně dvě pole (dva vektory) typu *TerrainPoint*. Jedno z nich uchovává aktuální informace terénu a ve druhém je uložen editační terén. Editační terén je využíván ve funkci pro gerenování náhodného terénu (o té později). Pokud se jedná o „veřejné“ vlastnosti, jsou jen dvě. Jedna z nich je typu Texture, pro uchování parametrů textury. Druhá je ukazatel na pole vertexů. Dále zbývá představit metody, nejprve ty jednodušší. Pro správu barev jsou metody *GetColor* a *SetColor*. U obou je první vstupní parametr typu *enum* (výčet), který je definován v souboru (stdafx.h). Tento konkrétní výčet nabývá hodnot „COLOR1“, „COLOR2“ a „ALL“. Jmenovaný vstupní parametr ovlivňuje, se kterou hodnotou bude funkce pracovat. V případě SetColor, která barva bude nastavena (buď první, druhá nebo obě). V případě GetColor, hodnoty které barvy budou vráceny. Pokud je parametr roven „ALL“, vrátí funkce „NULL“ a do konzole vypíše chybovou hlášku.

Metody začínající „RefreshGl“ mají za úkol obnovit data v poli vertexů. Je nutno podotknout, že pole vertexů je dvojnásobné oproti celkovému počtu vrcholů. Důvodem k tomu je fakt, že pro vykreslení terénu je použit „GL_TRIANGLE_STRIP“ a metoda použití „vertex array“ vyžaduje, aby vrcholy byly řazeny za sebou. Proto je každý vrchol uložen dvakrát, protože je použit pro pruh, který ho míjí zleva, a pruh, který ho míjí zprava. Metody pro obnovení dat v poli vertexů jsou celkem tři. Jedna se stará o vrcholy a normály, volá poté další dvě. První z těch dvou obnovuje barvy jednotlivých vrcholů, druhá obnovuje souřadnice textury. Ty dvě jsou na té první nezávislé kvůli ovládacím prvkům pro změnu barvy a nastavení textury. Metoda *GlOutput* zajišťuje vykreslení celého terénu. Nejprve je nastaven materiál a základní barva. Poté se podmíněně aktivuje vykreslení textury a pomocí OpenGL funkce pro vertex array i barevné pokrytí. Nakonec je vykreslena samotná geometrie a vypnuto texturování.

Funkce **ComputeNormals** je velice důležitá pro správné vykreslení celé scény. Vypočítává normálové vektory jednotlivých vrcholů a tudíž je zodpovědná za správné osvětlení terénu. V první fázi se spočítají „hrubé“ normálové vektory vždy použitím výšek sousedních vrcholů a aktuálního vrcholu pro jednotlivé dílčí vektory.



Obrázek 2 – Výpočet normálového vektoru

Rozdílem obou bodů v prostoru se získá vektor. Vektorovým součinem vždy vedlejší vektorů získáme vektor kolmý na rovinu jimi tvořenou – normálový vektor roviny (viz Obrázek 2). Výsledný normálový vektor získáme sumou jednotlivých normálových vektorů přilehlých ploch. Jelikož mají všechny stejnou váhu, je třeba zajistit, aby měli všechny stejnou velikost. Vyřešeno převodem na jednotkové vektory. Jelikož mají všechny stejnou váhu, je třeba zajistit, aby měli všechny stejnou velikost. Vyřešeno převodem na jednotkové vektory. V druhé fázi výpočtu normálového vektoru se k „hrubému“ normálovému vektoru přičtou okolní normálové vektory násobené určitým koeficientem. Tím dojde k vyhlazení. Na začátku celé funkce je testována podmínka, zda je třeba normálové vektory přepočítat.

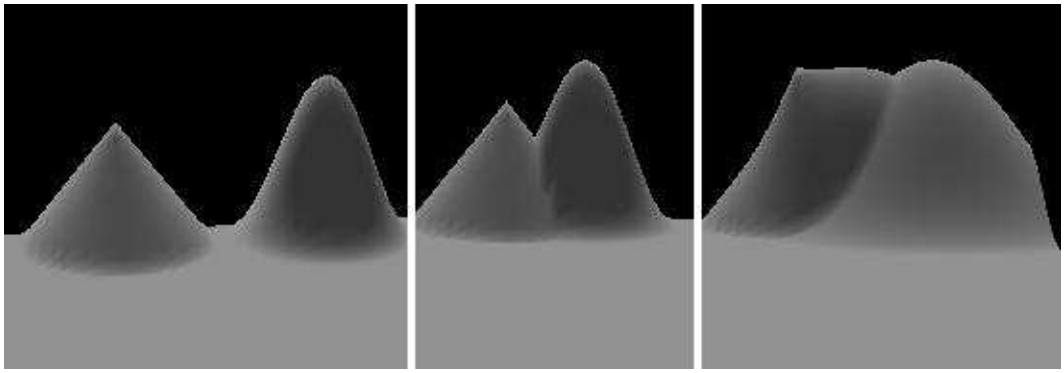
Stěžejní metodou pro vygenerování náhodného terénu je metoda **ModifyPoint**. Účelem této funkce je vzít zadaný bod, přiřadit mu novou hodnotu výšky a okolní body body upravit taktéž s určitým zeslabujícím efektem. Parametry pro funkci tedy jsou souřadnice bodu, cílová výška, vzdálenost zeslabení, typ zeslabení (lineární nebo sinus), způsob aplikace efektu (shodný se způsobem aplikace mapy). Metoda využívá algoritmu prohledávání do šířky neboli BFS (breadth-first search).

Algoritmus prohledávání do šířky

1. Vybere se počáteční bod, nastaví se mu ohodnocení na 0, bod se vloží do fronty.

2. Z fronty se odebere první prvek. Pro všechny jeho sousedy, kteří nejsou ohodnoceni, se nastaví ohodnocení na ohodnocení odebraného prvku plus jedna. Poté se přidají do fronty.
3. Bod 2. se opakuje do doby, dokud není fronta prázdná.

Ohodnocení je vzdálenost (počet hran) od počátku. Tento algoritmus je ovšem upraven s tím, že odebraný prvek se vloží do druhé fronty. Ta slouží pro vyčištění příznaku editace, o kterém bylo psáno dříve ve spojitosti s vlastnostmi TerrainPoint. Příznak slouží k ukládání ohodnocení. Po skončení je třeba, aby byl vyčištěn pro další použití. Dalším rozdílem je, že algoritmus prohledávání do šířky prohledává celý graf. V metodě je použito omezení, které nedovoluje přidat další prvky do fronty v závislosti na vstupním parametru „vzdálenost zeslabení“.



Obrázek 3 – Různá použití metody ModifyPoint

Na obrázku (Obrázek 3) je vidět vybraná funkčnost metody *ModifyPoint*. V levé části jsou ukázány jednotlivé typy zeslabení (lineární vlevo, sinus vpravo). Uprostřed a vpravo jsou ukázky způsobu aplikace. Prostřední příklad byl vytvořen nejprve editací typu sinus metodou přímého nastavení. Poté byl změněn druhý vrchol lineárním typem metodou „pokud vyšší“ (HIGHER). Příklad vpravo se liší akorát metodou přidání druhé editace. Zde se jedná o metodu „přidat“ (ADD).

Metody *ModifyPoint* přímo využívá metoda **RandomTerrain**. Ta cyklicky projde náhodné body terénu a aplikuje metodu ModifyPoint typu sinus metodou „přidat“. Výška je taktéž náhodná a může být i záporná. Náhodná je i vzdálenost zeslabení, ta záporná být nemůže. Metoda *ModifyPoint* je však vždy aplikovaná na editační terén. Po průchodu celého terénu a vygenerování náhodného editačního terénu se nejprve upraví rozsah výšek editačního terénu a poté se aplikuje na samotný terén jednou z vybraných metod. Metody

odpovídají uživatelově volbě v bloku „Terrain generation“ v panelu „Method“. Po aplikaci se použije ještě „škálovací“ metoda, pro upravení maximální výšky. Nastaví se v proměnné, že nejsou přepočítány normály a zavolá se metoda na jejich přepočet (*ComputeNormals*).

Ve třídě Terrain existuje jedna škálovací metoda. Nazývá se **ComputeBoundingBox**. Jak název napovídá, byla z počátku určena k jinému účelu. Ten jí zůstal a rozšířil se o další funkcionalitu. Cíle této metody jsou trojí. Prvním cílem je zjistit maximální a minimální výšku celého terénu. Druhým cílem je zajistit, aby minimální výška byla rovna nule (uzemnění). Celý terén tedy posunout, pokud je třeba. Třetím cílem je upravit rozsah terénu, tudíž maximální (eventuelně minimální) výšku. Metoda má navíc dva režimy práce. V prvním režimu je metoda volána na samotný terén, v druhém pak na editační terén. Metoda ovlivňuje vlastnosti minimální a maximální výšky (*boundingLow* a *boundingHigh*). Její vstupní parametry jsou „uzemnění“ (*grounding*), maximální výška a boolean proměnná, která rozlišuje, který z terénů se má použít. Návratová hodnota metody je výškový rozsah, tedy rozdíl maximální a minimální výšky. Ačkoliv je tato metoda v zásadě triviální, její využití se políná celou aplikací. Tato metoda je například použita i při ukládání výšek díky její návratové hodnotě.

Poslední metoda z třídy Terrain, která bude představena, a vlastně poslední metoda vůbec, je metoda **LoadHeightmap**. Cílem metody je načíst ze souboru (zadaného vstupním parametrem) data a ta aplikovat na terén. Na první pohled jednoduchý úkol. Ovšem mohou nastat problémy dvojího typu. Za prvé, jeden z rozměrů je větší než rozlišení terénu v dané ose. Za druhé, jeden z rozměrů je menší než rozlišení v dané ose. Pokud je problém prvního rázu (cílový rozměr je menší než zdrojový) a je tedy nutno vtěsnat větší počet pixelů do menšího, zjistíme nejdřív kolik pixelů připadá průměrně na jeden pixel cíle (označíme jako mocnost skupiny). Poté po skupinách přičítáme do nového pixelu hodnoty původních pixelu skupiny a na konci vydělíme mocností. Je nutno upozornit, že díky zaokrouhlování počtu použitých pixelů ze skupiny dochází k odchylce, a proto průměrná výška takto vzniklého terénu se bude lišit od terénu, který by měl shodné rozlišení. Proto je vhodné tento postup označit pouze za aproximační. Případ kdy je cílové rozlišení větší než zdrojové je poněkud svízelnější. Je totiž nutno si „vymyslet“ nové hodnoty. Řešením problému by mohl být algoritmus zde použitý ovšem i ten je třeba označit za aproximační. Nejprve určíme tzv. středy. Jsou to pozice původních indexů v novém rozlišení. Pokud je staré

rozlišení 3 a nové rozlišení 5, jsou pozice středů 1, 3, 5. Výsledná pozice středů bude popsána výpočtem níže (Vzorec 1). „n“ je vzálenost středů, „i“ je původní index, „k“ je pozice středu.

$$n = cílovéR / původníR \qquad k = (i*n)+0.5*(n-1)$$

Vzorec 1 – Vzorce pro výpočet středu

Interval <k-n ; k+n> označuje oblast nových indexů, do nichž se rozpočítá hodnota původního indexu. Váha hodnoty je úměrná vzdálenosti nového indexu „j“ od středu (Vzorec 2).

$$váha = 0.5+cos((j-k)*pi/n)/2$$

Vzorec 2 – Výpočet váhy

Toto je velmi hrubá aproximace, obzvláště pak výpočet váhy. Přesnější varianta by byla proložit středy polynomem a pak odečíst přímo funkční hodnotu v daném bodě nebo použít metodu bézierových křivek.

Tím by byla ukončena část představování vnitřní funkcionality projektu. V dobré víře, že bylo vše patřičně objasněno, bude opuštěna. Následuje závěrečná část, ve které bude shrnuto využití celé aplikace, možnosti rozšíření a celkové shrnutí.

4. Závěr a shrnutí

Cílem snažení bylo vytvořit soběstačnou aplikaci, která by splňovala nějaké funkční předpoklady. Aplikace by měla splňovat načítání vstupních dat, pro ty byl zvolen obecně používaný formát bitmap. Výstup aplikace by se měl dát uložit. Ukládání je dvojího typu. Buď se ukládá vizualizace do souboru nebo se ukládají data terénu. Nad touto částí je vhodé se zamyslet. Pro další práci na vývoji této aplikace je potřeba obě tyto části zlepšovat. Co se týče práce s daty, vzorem můžou být moderní modelovací aplikace, jako jsou 3D Studio MAX, Maya, Lightwave, Blender nebo Cinema 4D. Z těch orientovaných na modelování terénu je to zejména Bryce. Pokud se jedná o vizualizační část, je potřeba se rozmyslet, zda aplikace bude směřovat dále směrem k renderování (vykreslování) scény v reálném čase (real-time) nebo nikoliv. Pokud se jedná o renderování real-time, právě počítačové hry jsou ty aplikace, které ukazují, čeho lze dosáhnout.

Konkrétně v renderování terénu nebo popřípadě krajiny je to například Crysis nebo Farcry 2, kde se na to vývojáři zaměřili. Příkladem reálnějších vidin nových efektů v této aplikaci by bylo simulování efektů počasí pomocí částicových systémů a mlhy. Bylo by dobré se pokusit implementovat podporu multi-texturingu. Textury a materiály terénu by se mohly generovat v závislosti na normálových vektorech. Pomocí shader programů vytvořit dokonalejší materiály (zmíněný normal mapping). Po přidání možnosti importovat trojrozměrné modely by se terén dal rozšířit o stromový, keřový porost a zatravnění. Pokrytí by bylo v závislosti na výšce aktuálního bodu nebo normálového vektoru. Při úspěšné implementaci by se aplikace dala ještě rozšířit o animaci flóry a přidáním fyzikálního efektu větru, jenž by ovlivňoval jak animaci částicových systému nebo právě animaci zeleně. Jelikož by se nejednalo nadále už o ryze statickou scénu, obohatil by se výstup o video sekvenci. Co se týče modelování samotného terénu, mohlo by se jednat o simulaci erozních jevů, modelování pomocí křivek NURBS a jiné metodiky. Nakonec se ukázalo, že použití GLUI nebyla ta nejlepší volba, neboť její použití způsobilo několik problému při kompilaci výsledné aplikace. Čas strávený nad jejich odstraněním mohl být využit pro rošíření funkcionality aplikace. Pozitiva, které práce přinesla autorovi, jsou lepší orientace v jazyku C++ a API OpenGL. Dalším pozitivem je nabytí představy o tom, co programování 3D grafiky obnáší a jaké jsou její možnosti.

5. Použité zdroje

- Pavel Tišnovský : Seriál Grafická knihovna OpenGL

  URL: www.root.cz/serialy/graficka-knihovna-opengl/
- Wikipedia, the free encyclopedia

  URL: http://cs.wikipedia.org/wiki/
- Wikipedia, veřejná encyklopedie

  URL: http://cs.wikipedia.org/wiki/
- Paul Rademacher, GLUI, A GLUT-Based User Interface Library
  URL: http://www.nigels.com/glt/glui/
- OpenGL documentation

  URL: http://www.opengl.org/documentation/

- SGI

  URL: http://www.sgi.com/

- Videotutorials Rock

  URL: http://www.videotutorialsrock.com/