WebGL Earth

Virtuální glóbus v internetovém prohlížeči

Petr Sloup
325196@mail.muni.cz

Fakulta informatiky,
Masarykova univerzita,
Botanická 68a, 602 00, Brno

Abstrakt Tato práce popisuje základní prvky analýzy, návrhu a implementace projektu WebGL Earth – interaktivního trojrozměrného virtuálního glóbu dostupného přímo v moderních internetových prohlížečích (Mozilla Firefox 4, Google Chrome 10, ...) bez nutnosti instalovat jakýkoli dodatečný software. Stěžejním bodem práce je návrh vhodných datových struktur a kódu v jazyce GLSL a JavaScript pro načítání mapových dlaždic, jejich správu v omezeném množství grafické paměti a následné mapování obrazových dat z Mercator projekce na kouli. Výsledná *open-source* (GNU GPL) aplikace je schopná v reálném čase zobrazovat jak detailní uliční síť, tak satelitní snímky planety z existujících mapových podkladů dostupných pomocí OpenStreetMap, MapQuest, Bing SDK nebo standardu TMS. Do dalšího vývoje se již zapojili studenti a akademičtí pracovníci ze Španělska (Universitat Politècnica de València, Spain) i komerční sféra (CampToCamp S.A., France/Switzerland).

Klíčová slova: HTML5, WebGL, GIS, virtuální glóbus, texturování, Mercator dlaždice

Abstract. This work describes most important parts of analysis, design, and implementation of the WebGL Earth project – interactive three-dimensional virtual globe running directly in modern web browsers (Mozilla Firefox 4, Google Chrome 10, ...) without the need to install any additional software. The key part of the thesis is the design of appropriate data structures and code in the GLSL and JavaScript languages for the loading of map tiles, their management in a limited amount of graphical memory and subsequent mapping of image data from the Mercator projection onto the sphere. The resulting *open-source* (GNU GPL) application is able to display both detailed street network and satellite imagery from existing map sources, available via OpenStreetMaps, MapQuest, Bing SDK or the TMS standard. The project already has contributors from both academical (Universitat Politècnica de València, Spain) and commercial areas (CampToCamp S.A., France/Switzerland).

Keywords: HTML5, WebGL, GIS, virtual globe, texturing, Mercator tiles

1 Úvod

S příchodem HTML5 a WebGL se otevírají zcela nové možnosti webových prezentací a to ve všech oblastech včetně geografických informačních systémů. Trojrozměrné vizualizace Země jsou velmi moderní a umožňují znázornit některé atributy (elevační data, 3D budovy, ...), které klasický pohled shora-dolů zachytit nedokáže.

Mnoho internetových stránek potřebuje svým návštěvníkům nějaká data přiblížit na mapě (ať už se jedná o vektorové vrstvy nebo rastrové překryvy) a často i na 3D modelu Země. V současnosti je jedinou možností, jak takové 3D vizualizace dosáhnout, použití *closed-source* Google Earth API. To ovšem vyžaduje instalaci doplňku do prohlížeče [1] a k tomu uživatel ve svém operačním systému zpravidla potřebuje odpovídající oprávnění.

Cílem projektu *WebGL Earth* je poskytnout alternativní, *open-source* řešení implementované v jazyce JavaScript a dostupné přímo v internetových prohlížečích s podporou WebGL. Výsledkem je tedy plně multi-platformní 3D aplikace fungující i na moderních mobilních zařízeních (chytré telefony, tablety, ...).

Technicky nejnáročnější částí vizualizačního procesu je správa a práce s obrovskými texturovými daty (až 2 miliardy × 2 miliardy pixelů[1]). Cílem této práce je porovnat existující metody zobrazování obrovských textur na vesmírných tělesech (převážně Země) a v případě potřeby navrhnout metody nové. Je také nutné zohlednit charakter dostupných dat – projektu bude využívat volně dostupné dlaždice v Mercatorově zobrazení (OpenStreetMaps, Bing Maps, ...).

Vzhledem ke stanovenému jazyku (JavaScript, který je jakožto skriptovací jazyk poměrně pomalý) je nutné, aby nalezené řešení co nejméně zatěžovalo CPU. WebGL nám poskytuje přístup k programovatelným jednotkám grafické karty pomocí speciálních programů (tzv. *shaders*) psaných v jazyce GLSL ES 1.0 (*OpenGL Shading Language for Embedded Systems*), které jsou k tomuto účelu naopak velmi vhodné.

1.1 Mercatorovo zobrazení

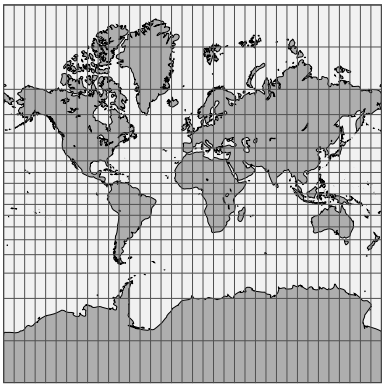
Mercatorovo zobrazení je jeden z možných způsobů projekce povrchu koule do roviny a je definováno následujícími rovnicemi [7], kde λ a φ jsou zeměpisná šířka a délka, $[x, y]$ jsou souřadnice bodu na zobrazené mapě:

$$x = \lambda \tag{1}$$

$$y = \sinh^{-1}(\tan(\varphi)) \tag{2}$$

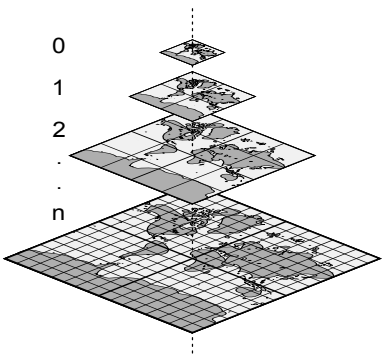
Toto si je také možné představit jako zobrazení povrchu koule na plášť válce, jehož osa je totožná s osou Země, a následné rozvinutí pláště válce do roviny. Poledníky v takto vzniklé pravoúhlé síti mají pravidelné rozestupy, zatímco vzdálenosti mezi rovnoběžkami se směrem k pólům zvyšují do nekonečna.

[1] Při 23 úrovních přiblížení a dlaždicích $256 \times 256\,\mathrm{px} - 2^{23^2} \times 256$



Obrázek 1. Mapa světa v Mercatorově zobrazení.

Jak je vidět na obrázku 1, takto vytvořená mapa zachovává úhly (jedná se tedy o konformní zobrazení) a tvary malých objektů. Velikosti vzdálených objektů ale již porovnatelné nejsou – například Grónsko se jeví stejně velké jako Afrika. Kvůli nemožnosti zachytit rozumně polární oblasti (samotné póly by se zobrazily v nekonečnu) končí zpravidla mapy v tomto zobrazení na $\varphi = \pm 85.05113°$ tak, aby měl výsledný grafický výstup čtvercový tvar. Toho se dá s úspěchem využít při výstavbě pyramidy dlaždic (obrázek 2), kde se n-tá úroveň skládá z $2^n \times 2^n$ dlaždic.



Obrázek 2. Pyramida Mercator dlaždic.

Takto připravené dlaždice (sady obrázků o rozměrech obvykle 256×256 px) jsou volně dostupné například z projektu OpenStreetMaps (až do úrovně 18), po

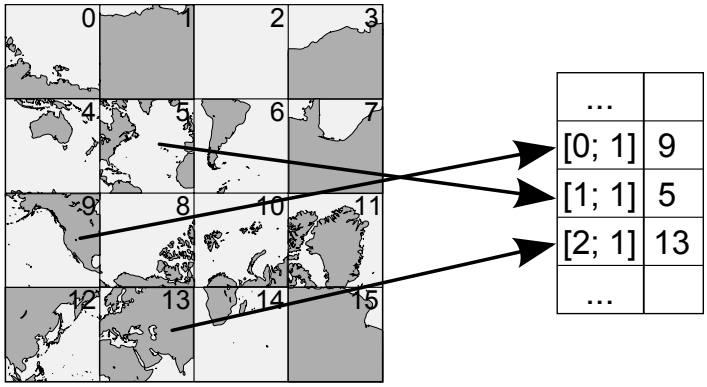
registraci několik verzí z Bing Maps (včetně leteckých snímků) a stejný systém dlaždic využívají i další projekty včetně oblíbených Google Maps.

2 Existující metody

2.1 Virtual Textures

Jako *Virtual Textures* (někdy také *MegaTextures*, ale jedná se principiálně o to stejné [4]) se označuje metoda správy velkých textur, která (jak název naznačuje) přejímá základní myšlenky virtualizace paměti z operačních systémů.

Celá textura je rozdělena na stránky (které v našem případě odpovídají jednotlivým dlaždicím) a v grafické paměti se vytvoří velký stránkovací buffer, do kterého jsou tyto stránky nahrávány. Souběžně s bufferem je třeba udržovat tzv. *tabulku stránek*, ze které lze při vykreslování v konstantním čase zjistit, zda-li a kde je daná stránka dostupná.



Obrázek 3. Princip tabulky stránek.

Aktualizace probíhá jednoduchým zaplněním volného nebo přepsáním obsazeného místa v bufferu a odpovídající změnou v tabulce stránek. K výběru vhodného místa se většinou využívá LRU (*Least Recently Used*) struktura.

Pro naše potřeby je zásadní nevýhodou tohoto řešení špatná škálovatelnost – paměťová složitost tabulky je lineární vzhledem k počtu stránek. Potřebujeme zobrazovat dlaždice úrovně 18 a více – až 2^{18^2} dlaždic (stránek) na nejdetailnější úrovni. Takto obrovskou tabulku stránek by nebylo možné uchovávat v grafické ani operační paměti běžných počítačů.

S využitím skutečnosti, že potřebné dlaždice jsou většinou z malé, relativně ohraničené oblasti by bylo možné udržovat „tabulku tabulek“, ve které by byly pouze odkazy na detailnější tabulky u těch dlaždic, pro které jsou k dispozici podrobnější data. Při vykreslovaní by se postupovalo tímto stromem tabulek až

k odkazu na paměťový prostor v bufferu. Takto koncipovaná hierarchie by ovšem byla poměrně náročná na aktualizace a správu paměti.

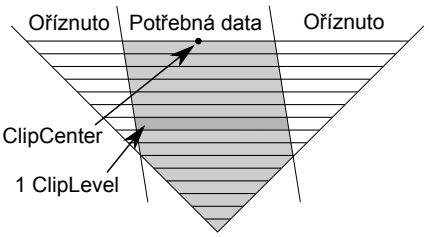
Tato metoda je pro nás ale nepoužitelná hlavně proto, že jediná efektivní možnost umístění velké tabulky stránek je v grafické paměti (jako textury), ale naše cílová platforma (WebGL – GLSL ES 1.0) ve specifikaci nedefinuje podporu *Vertex Texture Fetch* (přistupování k texturovým bufferům z vertex shaderu) jako povinnou [3], bylo by tedy potřeba realizovat vyhledávání až ve fragment shaderu pro každý pixel zvlášť.

2.2 ClipMapping

Tato technika, kterou popsal C. Tanner [6], vychází z pozorování ohledně mipmappingu. Mipmapa (jak ji definoval L. Williams [8]) je série obrázků s postupně snižovaným (polovičním) rozlišením až do 1×1 px. Současné grafické karty jsou schopné generovat *mipmap pyramidu* přímo při nahrávání textur do grafické paměti. Během vykreslování se potom v závislosti na vzdálenosti a úhlu kamery od povrchu vybírá vhodný mipmap level a z něj (případně ze dvou nejvhodnějších v případě použití trilineárního filtrování) dochází ke čtení potřebného texelu (pixelu textury). Tímto lze výrazně snížit počet čtených texelů a zároveň (za použití kvalitního filtrování při tvorbě mipmapy) bojovat proti aliasingu.

Výpočty za výběrem správného mipmap levelu jsou nastaveny tak, že dochází vždy k výběru texelu (nebo častěji více texelů) takovým způsobem, aby mapování pixel:texel bylo co nejblíže 1:1. Z tohoto lze odvodit, že z žádného mipmap levelu není během renderování jednoho snímku potřeba více texelů, než kolik pixelů má viewport, do kterého se scéna renderuje.

Pokud je tedy textura mapována na geometrii spojitě, lze určit jakýsi „střed zájmu“ (*ClipCenter*) a zjistit, která data budou v aktuálním snímku potřeba.



Obrázek 4. Oblast clipmapy jako podmnožina mipmapy.

Jelikož ClipMapa ze své podstaty obsahuje data na vyrenderování pouze jednoho konkrétního snímku, je potřeba ji aktualizovat při každé změně ClipCenter. S využitím poznatku, že texturu lze (při nastavení zalamovacího režimu na `gl.REPEAT`) adresovat „dokola“ (tzv. toroidální adresování), nám stačí nahradit již nepotřebná data těmi novými a nedochází tedy ke zbytečným přesunům bloků paměti.

Každý *ClipLevel* (viz obrázek 4) je v grafické paměti uložen jako samostatná textura včetně celé mipmap pyramidy. Při vykreslování je tedy potřeba rozhodnout, se kterou texturou se má každý segment geometrie vykreslit. Návrh hardwarově nezávislého clipmappingu [5] popisuje dva různé způsoby:

1. Pro každý segment geometrie (každou dlaždici) použijeme nejdetailnější dostupný ClipLevel a vykreslíme.
2. Vykreslíme všechny segmenty, pro které jsou k dispozici nejdetailnější data, poté segmenty s méně podrobnými daty a takto pokračujeme až k poslednímu ClipLevelu.

Oba tyto způsoby předpokládají rozdělení geometrie do většího množství samostatných celků – to by mělo za následek větší zátěž CPU (objemnější datové struktury, časté přepínání vertex bufferů), což je v rozporu s naším cílem odlehčit co nejvíce pomalému JavaScriptu přesunem náročných výpočtů na GPU. Tento způsob je pro nás tedy také (v klasické variantě) nevhodný.

3 Naše metody

3.1 Dlaždicový buffer s metadaty

Modifikací Virtual Textures (sekce 2.1) lze získat metodu použitelnou v námi stanovených podmínkách. Podobně jako u původního řešení máme jeden velký buffer (texturu) na ukládání dlaždic. Místo tabulky stránek ovšem zavádíme k tomuto bufferu *metadata* (tabulka 1), která popisují dlaždice uložené na jednotlivých pozicích v bufferu (tzv. „slotech“).

Tabulka 1. Příklad metadat k 8×8 bufferu

ID „slotu“	Úroveň přiblížení	Souřadnice dlaždice
0	8	[79; 230]
1	-1	prázdno
2	14	[6396; 2436]
⋮		
63	7	[8; 45]

Metadata vstupují do vertex shaderu (jako pole `uniform` proměnných), kde pro každý vrchol dochází k výpočtu, která dlaždice je potřeba. Prohledáním metadat se poté zjistí její dostupnost a případné umístění v bufferu. Výrazné optimalizace lze dosáhnout, pokud na metadatech definujeme uspořádání podle úrovně přiblížení a souřadnic dlaždice – ve vertex shaderu pak lze provádět hledání metodou půlení intervalu.

Aktualizace probíhá podobně jako u Virtual Textures – zaplněním nebo nahrazením prostoru v bufferu a aktualizací metadat. Vždy před odesláním metadat do vertex shaderu se ovšem musíme ujistit, že jsou správně seřazená.

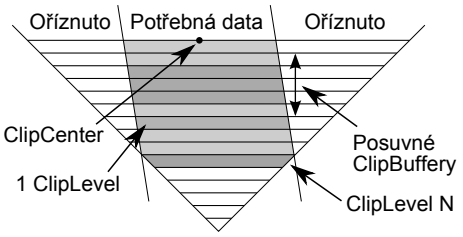
Obrázek 5. Diagram stručně znázorňující proces renderování pomocí metadat.

Na první pohled viditelným nedostatkem jsou chyby na okrajích dlaždic, kde kvůli filtrování textur dochází k částečným „přesahům“ do vedlejších (nesouvisejících) texelů v bufferu. Poměrně jednoduchou opravu lze provést striktním „clampingem“ texturových koordinátů ve fragment shaderu. Při použití této metody na 3D terén (založený na výškové mapě) ovšem není tato oprava možná a na rozhraní terénních dlaždic jsou viditelné „skoky“ v geometrii.

Tímto způsobem je možné zadaný problém řešit. Počet instrukcí vykonávaných ve vertexu shaderu ovšem roste s velikostí bufferu a již při 8×8 dlaždicích trvá vykreslování na méně výkonných grafických kartách příliš dlouho a na některých kartách se program kvůli nucenému rozbalování smyček („loop unrolling“) vůbec nezkompiluje.

3.2 ClipStack

ClipMapping (sekce 2.2) se ukázal jako velmi zajímavý způsob správy velkých textur. Zavedením několika úprav a zjednodušení vzniká modifikace vhodná k implementaci v JavaScriptu a WebGL, kterou jsme pojmenovali *ClipStack*.



Obrázek 6. Znázornění ClipStacku

ClipStack je kolekce ClipLevelů. Každý ClipLevel představuje podmnožinu jedné úrovně pyramidy dlaždic a udržuje si informace o umístění oblasti, kterou pokrývá (tzv. *offset*).

Z důvodů úspory paměti nemá každý ClipLevel svůj vyhrazený buffer. Místo toho má ClipStack vytvořený fixní počet *ClipBufferů* (zapouzdřený buffer), které

jsou „posouvány“ mezi vrstvami tak, aby pokrývaly aktuálně potřebný ClipLevel (v závislosti na vzdálenosti kamery) a určitý počet bezprostředně nadřazených (tzv. *fallback levels*).

Ke každému ClipLevelu se také udržují *metadata* popisující, které části bufferu jsou validní a které je teprve potřeba naplnit daty ze serveru.

Takto vzniklá struktura se udržuje aktuální přepočítáváním ClipCenter a offsetů jednotlivých vrstev (s tím souvisí také odpovídající úprava metadat) a v případě potřeby již zmiňovaným posouváním ClipBufferů. V takovém případě dochází u deaktivovaného ClipLevelu (ten, kterému byl buffer odebrán) k celkovému resetu metadat.

Při vykreslování se ve vertex shaderu vypočítají souřadnice potřebné dlaždice a z metadat aktuálně nejvhodnějšího ClipLevelu se zjistí, zda-li je dlaždice dostupná. V případě potřeby dojde k nahlédnutí do nadřazených vrstev. Vzhledem k omezenému počtu aktivních vrstev (v naší implementaci 3) může nastat situace, kdy potřebná data nejsou připravena ani v jednom aktivním ClipLevelu (buďto ještě nedošlo k jejich načtení, nebo jsou mimo pokrytou oblast). Pro takové případy zavádíme *ClipLevelN*, což je speciální, zjednodušená vrstva, která pokrývá celou planetu.

4 Implementace

4.1 Segmented Plane

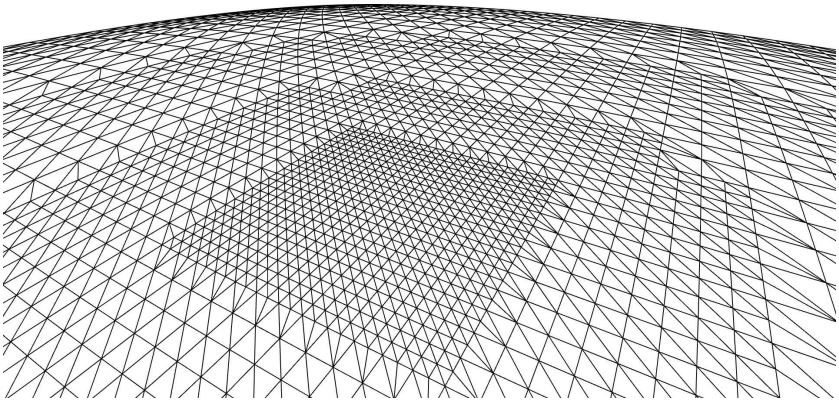
Aby bylo možné výslednou planetu zobrazit, potřebujeme také odpovídajícím způsobem optimalizovanou geometrii, na kterou bychom mohli texturová data aplikovat. Jak jsme již výše zmínili, je třeba, aby hrany polygonů byly zarovnány s hranami dlaždic. Není možné vytvořit geometrii tak velkou, aby současně pokrývala celý povrch Země.

Pozorovali jsme, že při pohledu shora-dolů je počet viditelných dlaždic prakticky neměnný a ani s volnou kamerou neroste tak rychle jako celkový počet dlaždic v dané úrovni přiblížení.

Na základě tohoto pozorování jsme vyvinuli strukturu *Segmented Plane*. Tato dvourozměrná čtvercová síť je navržena tak, aby reprezentovala konstantní množství dlaždic a zároveň.

Během renderování jsou tato statická dvourozměrná data transformována (na základě pozice a orientace kamery) tak, aby tvořila pokud možno potřebnou podmnožinu povrchu planety. Veškeré výpočty jsou optimálně realizovány ve vertex shaderu. Tím, že je každý vrchol Segmented Plane transformován na odpovídající vrchol výsledné geometrie, se zároveň řeší problém transformace Mercator souřadnic na kouli. Ačkoli mezi vrcholy je prováděna pouze lineární interpolace, na výsledném modelu nejsou žádné viditelné odchylky.

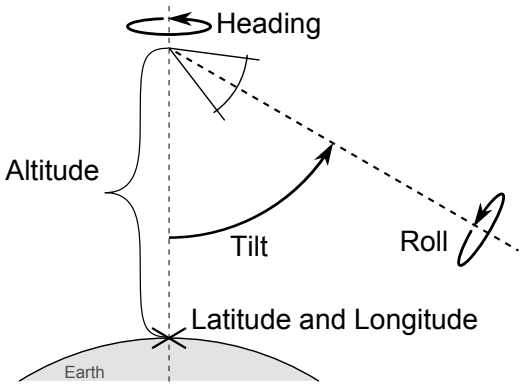
Kdybychom použili pouze 4 vrcholy pro každou dlaždici, Země by vypadala velmi „hranatě“ při nižších úrovních přiblížení. Tento problém řešíme zavedením dodatečného rozdělování geometrie. Později, jak ukazuje obrázek 7, jsme také zavedli postupné změny detailu na úrovni geometrie, tak aby byla pokryta co největší oblast planety a zároveň zůstal dostatek detailů pro 3D terén.



Obrázek 7. Segmented Plane s postupným úbytkem geometrických detailů

4.2 Volná kamera

Abychom mohli plně využít možností trojrozměrného zobrazení, bylo potřeba vyvinout *„volnou“ kameru*, která umožňuje uživateli měnit pozici a směr, ze kterého je na zobrazovanou planetu nahlíženo.

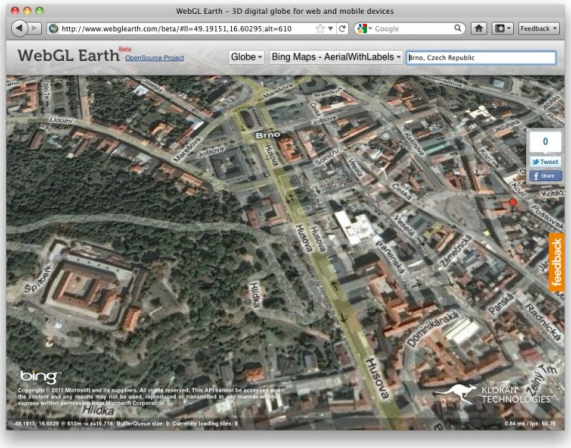


Obrázek 8. Znázornění atributů „volné“ kamery

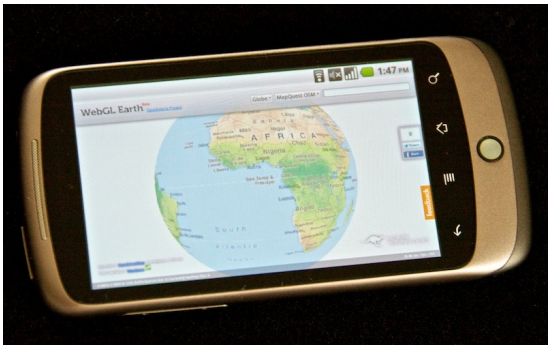
Tento objekt popisující virtuální kameru, je definován několika atributy (obrázek 8), které jednoznačně určují pozici a orientaci v prostoru.

5 Závěr

Poslední dvě nastíněné metody jsme úspěšně implementovali v projektu WebGL Earth (`http://www.webglearth.org/`).



Obrázek 9. WebGL Earth – mírně sklopený pohled na centrum Brna



Obrázek 10. WebGL Earth běžící na Google Nexus One

Dříve popsaná metoda s jediným dlaždicovým bufferem (sekce 3.1) se ukázala méně paměťově náročná, ale za cenu vyšších nároků na výpočetní výkon a horší vizuální výsledky (které se projevily hlavně po implementaci 3D terénu). V současné době tedy projekt využívá jeden ClipStack (viz. sekce 3.2) se třemi

aktivními vrstvami na povrchová data a druhý se dvěma aktivními vrstvami na elevační data.

Reference

1. Google Inc. *Google Earth API Developer's Guide*. URL: `http://code.google.com/apis/earth/documentation/` [cit. 2011-04-13].
2. Khronos WebGL Working Group. *WebGL Specification*. 10. února 2011. URL: `http://www.khronos.org/registry/webgl/specs/1.0/` [cit. 2011-04-03].
3. The Khronos Group Inc. *The OpenGL ES Shading Language*. 12. května 2009. URL: `http://www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf` [cit. 2011-04-03].
4. Mayer A. J. *Virtual Texturing*. Institute of Computer Graphics and Algorithms, Vienna University of Technology, 14. října 2010. URL: `http://www.cg.tuwien.ac.at/research/publications/2010/Mayer-2010-VT/` [cit. 2011-04-05].
5. Seoane A., Taibo J., Hernández L., López R., Jaspe A. Hardware-Independent Clipmapping. *WSCG '2007: The 15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2007 - Full Papers Proceedings II*. s 177–183. Eurographics Association, 2007.
6. Tanne C. C., Migdal C. J., Jones M. T. The Clipmap: A Virtual Mipmap. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. s 151–158. SIGGRAPH '98. New York, 1998. ISBN: 0-89791-999-8.
7. Weisstein, E. W. *Mercator Projection*. MathWorld. URL: `http://mathworld.wolfram.com/MercatorProjection.html` [cit. 2011-04-08].
8. Williams L.: Pyramidal parametrics. *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*. s 1–11. SIGGRAPH '83. New York, 1983. ISBN: 0-89791-109-1.