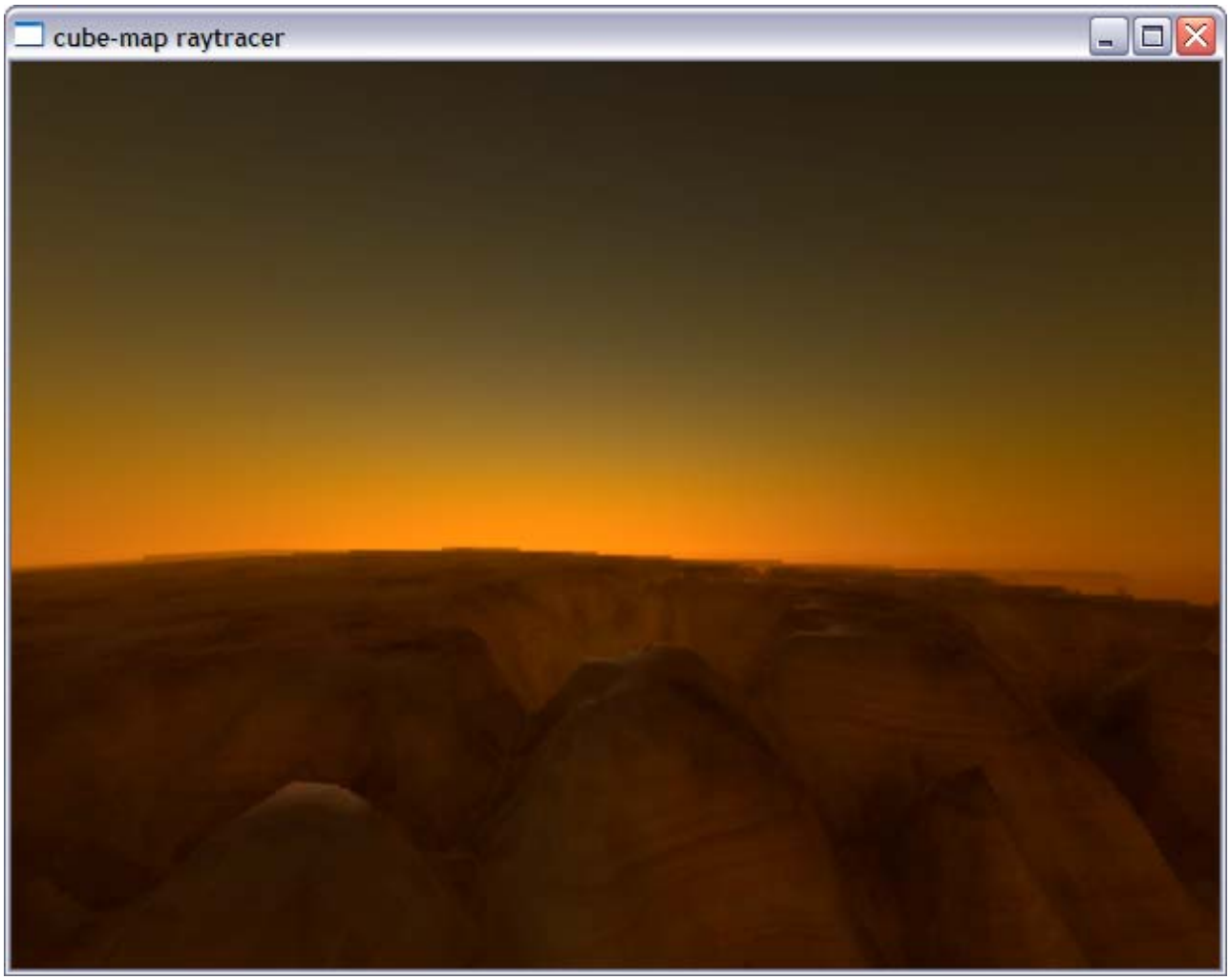


tx01 - Generování 2D textur pro SkyBox

Hanuš Jiří, Bc. xhanus04@stud.fit.vutbr.cz
Kolář Vít, Bc. xkolar40@stud.fit.vutbr.cz
Polok Lukáš, Bc. xpolok00@stud.fit.vutbr.cz

1. Zadání

- Vytvořit jednoduchý RayTracer, který vygeneruje 4 nebo 5 (=4+horní) textur pro zobrazování SkyBoxu.
- Vytvořit procedurální texturu, kterou raytracer promítne do výsledných textur.
- Textury budou uloženy do BMP souborů.
- Proces generování bude možné parametrizovat pro vytvoření různých textur (hodně/málo mraků, den/ráno/šero, ...)
- Pro demonstraci funkčnosti je možné vytvořit vlastní program

2. Popis programu

Program generuje kompletní skybox (tzn. 6 textur), jež se dají použít buďto pro manuální namapování na geometrii skyboxu, nebo jako cubemap textura (GL_ARB_texture_cubemap). Je umožněno uložení do formátu BMP nebo do formátu TGA, po vygenerování je zároveň možné cubemapu zobrazit. Generované textury obsahují fyzikálně simulovanou oblohu a procedurální terén s volumetrickou mlhou a ambient occlusion stínovacím modelem. V plánu bylo přidat ještě oblačnost, to však nebylo v době odevzdání splněno.

Raytracer podporuje multisampling, pro vzorkování používá vzorky s Poissonovým rozložením, generované silně optimalizovaným algoritmem dart-throwing.

Protože obloha obsahuje mnoho hladkých přechodů jež způsobují viditelné artefakty, je použit dithering pro umělé navýšení počtu zobrazitelných barev. Není použit ‚klasický' Floyd-Stenbergův algoritmus, ale zajímavý algoritmus s rozložením s charakteristikou modrého šumu jehož vynálezcem je Viktor Ostroumhkov [7].

Program definuje jednoduchý formát skriptu pro popis generovaného prostředí. Skript je programem přeložen a následně proveden. Generování je pseudonáhodné, tzn. pro jeden skript vyjde vždy stejná textura.

3. Ovládání programu

Program se spouští z příkazové řádky, přijímá tyto parametry:

```
skygen -i <taskfile> [-o <image%s.tga>] [-v|--verbose] [-s|--see-results] [-h|--help]
```

kde:

- *taskfile* je soubor se vstupním skriptem
- *image%s.tga* je výstupní obrázek. Formátovací řetězec ‚%s' bude nahrazen za kombinace ‚(**neg**|**pos**)[**xyz**]'. Je možné použít příponu .tga nebo .bmp.
- *verbose* zapíná průběžný výstup
- *see-results* po dokončení zobrazí interaktivní GLUT prohlížeč hotového skyboxu
- *help* zobrazí nápovědu k ovládání programu

4. Rozdělení práce

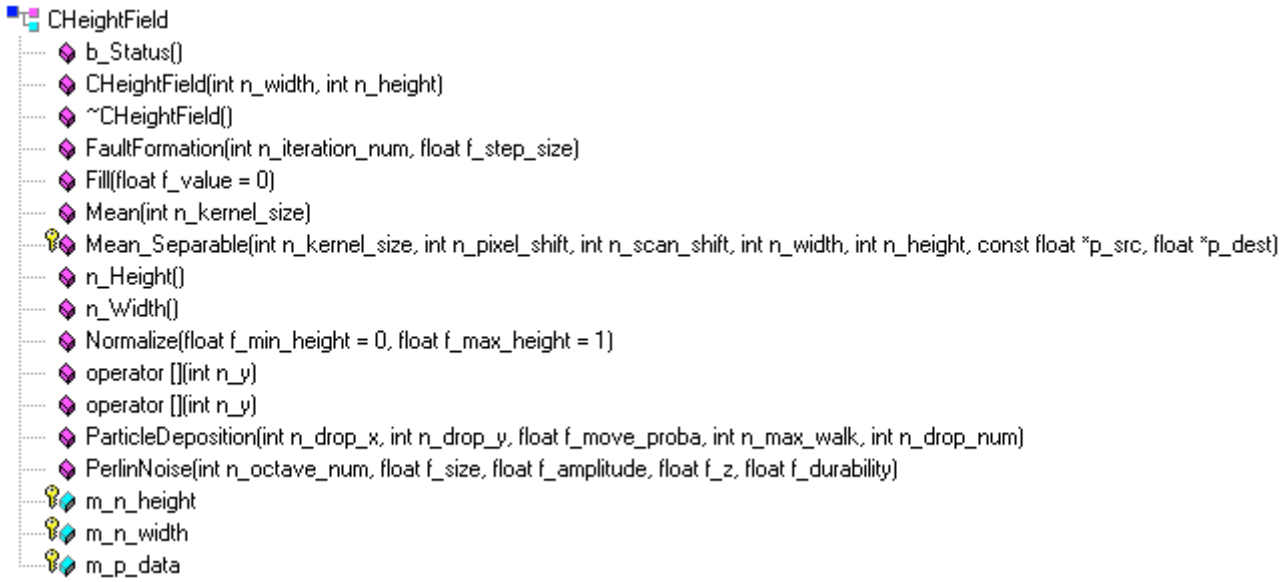
Jiří Hanuš: generování terénu, ukládání bitmap
Vít Kolář: vektorová knihovna, ukládání formátu targa
Lukáš Polok: simulace atmosféry, vlastní raytracer, kostra aplikace

5. Jiří Hanuš

5.1 Generování terénu

Terén je generován jako výšková mapa. Ta je reprezentována třídou CHeightField. Jednotlivé vzorky výškové mapy jsou potom desetinná čísla. Výšková mapa nabízí několik funkcí pomocí nichž lze pseudonáhodně generovat terén. Za poznámku stojí že je použit náhodný generátor ze standardní knihovny, jež může mít různé implementace (linux vs. windows), takže výsledky nemusí být shoné.

Metody a členy třídy CHeightField jsou zde:



Třída má jednoduchý konstruktor, jež přijímá požadovaný rozměr výškové mapy. Protože v konstruktoru probíhá alokace, je zde ještě funkce b_Status, která vrací booleovskou hodnotu v níž se odráží úspěch konstruktoru.

K jednotlivým prvkům heightmapy se lze dostat pomocí operátorů [], jsou zde dvě verze lišící se v konstantnosti návratové hodnoty. Rozměry heightmapy se dají zjistit pomocí funkcí n_Width a n_Height. Další funkce již souvisí s vlastním generováním výškové mapy.

Funkce Fill je velice jadnoduchá a vše co dělá je, že naplní výškovou mapu konstantní hodnotou a tím pádem vytvoří plochou rovinu.

Funkce Normalize najde extrémní výškové hodnoty (tzn. nejvyšší a nejnižší bod), potom spočítá rovnici přímky po které se budou výškové hodnoty posouvat a přepočítá je tak, aby minimální a maximální hodnota odpovídaly zadaným parametrům funkce. Rovnice přímky je v tomto případě dána rovnicí:

$$y = k \cdot x + q$$

$$k = (\text{desired-max} - \text{desired-min}) \;/\; (\text{max} - \text{min}) \qquad \text{pokud max} > \text{min}$$

$$k = 0 \qquad \text{pokud max} = \text{min}$$

$$q = \text{desired-min} - \text{min} \cdot k$$

Kde *x* je vstupní hodnota, *y* je výstopní hodnota (výška). *k* a *q* jsou parametry rovnice přímky, *desired-min* a *desired-max* jsou parametry funkce Normalize a *min* a *max* jsou aktuální hodnoty ve výškové mapě.

Funkce Mean počítá aritmetický průměr výšek pod čtvercovým oknem dané velikosti. Zamýšleným efektem je jednoduchý vyhlazovací filtr. Pro větší efektivitu výpočtu je filtr proveden separovaně a provádí se ve dvou průchodech, jednou pro horizontální vyhlazení a podruhé pro vertikální.

Funkce využívá skutečnosti že vzorky nejsou ničím vážené (mají všechnystejnou váhu) a proto pro vypočítání hodnoty každého dalšího vzorku stačí pouze odečíst výšku, která je v okně úplně vlevo a přičíst výšku která je vpravo, hned vedle okna (při posouvání filtrovacího okna zleva doprava). Pro vypočítání prvního vzorku je potřeba spočítat hodnotu normálním způsobem, ostatní vzorky se jen ‚nasouvají' a ‚vysouvají' z akumulátoru. Tím se, hlavně při velkých filtrech, výrazně sníží počet čtení z pole výškové mapy a celá operace se velmi urychlí. Cena filtru v podstatě závisí jen na rozlišení výškové mapy, závislost na velikosti filtru je odstraněna. Rovnice pro filtr v jednom směru jsou:

$$h(x) = \begin{cases} \sum_{i=-k/2}^{k/2} h(\mathrm{clamp}(i)) \\ h(x-1) - h(\mathrm{clamp}(x-1-k/2)) + h(\mathrm{clamp}(x+k/2)) \qquad \text{pro } x > 0 \end{cases}$$

$$\mathrm{clamp}(x) = \min(\mathrm{size}-1, \max(0, x))$$

Kde h(x) je výšková funkce (vzorek z výškové mapy), size je rozlišení mapy v daném směru a x je souřadnice. Funkce clamp zajišťuje že index nikdy nevyjde mimo hranice pole.

Operátor Fault Formation má simulovat utváření terénu posuvem zemských desek. Jeho činnost spočívá v opakované modifikaci stávajících hodnot výškové mapy, a to přičtením nebo odečtením určité hodnoty. To zda se bude hodnota přičítat nebo odečítat závisí na pozici roviny, jež rozděluje celý terén na dvě oblasti. Pozice roviny je vybírána náhodně, ale tak aby vždy do terénu zasahovala, tedy aby ani jedna z oblastí nebyla prázdná. Výsledky takové operace jsou dosti neuhlazené, je proto vhodné použít vyhlazení filtrem Mean.

Dalším jednoduchým operátorem je Particle Deposition, tedy utváření terénu částicovým systémem. Idea je následující: z oblohy na terén padá kamení a tím pádem terén navyšuje. Pozice ze které kamení padá se mírně posouvá náhodným směrem. Pokud kámen dopadne na svah, valí se dolů tak dlouho, dokud nedosáhne nějaké stabilní polohy.

Implementace je vcelku jednoduchá, na začátku je dána pozice emitoru částic, pravděpodobnost jeho pohybu při každém kroku (pohyb je přitom čistě náhodný), maximální počet kroků při hledání stabilní polohy pro částici a počet částic které se mají umístit. Pro rozhodnutí zda je částice na stabilním terénu je použit sobelův operátor. Pokud je jeho absolutní hodnota větší, než experimentálně stanovený práh, částice se bude posouvat směrem, opět určeným za pomoci sobelova operátoru. Když částice dosáhne stabilní polohy, popřípadě počet pokusů o dosažení stabilní polohy přesáhne daný limit, terén se na pozici částice navýší o nějakou malou hodnotu.

Posledním operátorem je přičtení šumové funkce Perlin Noise. Funkce se jmenuje po jejím vynálezci, Kenu Perlinovi, a je v poli procedurálního generování textur téměř tak standardní jako například funkce sinus, proto nemá smysl se s ní příliš rozepisovat. Za zmínku stojí že využívá předem vygenerovanou tabulku s pseudonáhodnými hodnotami a výsledná hodnota funkce se počítá pomocí interpolace skalárních součinů argumentu s hodnotami v nejbližších pólech. Pro generování terénu je použita fraktální podoba této funkce, kdy se vlastní perlin noise počítá v několika oktávách s různou frekvencí a amplitudou. Frekvence se vždy zdvojnásobuje, amplituda se při každé oktávě násobí daným faktorem a podle toho může buď klesat, být konstantní a nebo stoupat. To potom ovlivňuje poměry vysokých a nízkých frekvencí ve výsledné šumové funkci a její celkový charakter.

5.2 Ukládání obrázků ve formátu BMP

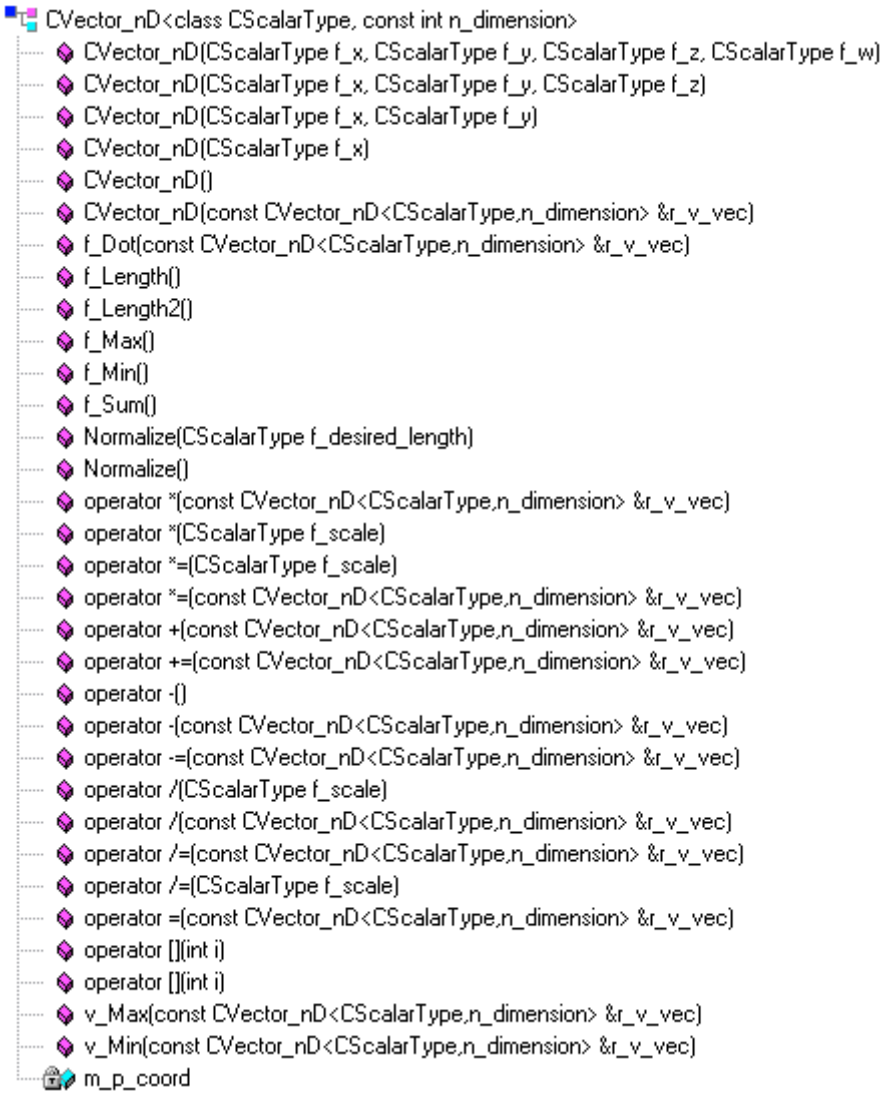
Formát BMP je dobře známý grafický formát, používaný systémy Windows a OS2. Formáty nejsou vzájemně kompatibilní, náš program používá rozšířenější Windows alternativu. Bitmapa má dvě hlavičky, BITMAPINFOHEADER a BITMAPFILEHEADER, které jsou standardně definované v hlavičce windows.h. Program ale musí být přeložitelný i na linuxu a proto byly struktury zkopírovány přímo do zdrojového textu programu. Bitmapa je uložena jako nekomprimované RGB hodnoty v přesnosti 8 bitů na kanál, zarovnané na řádku na hranice 32bitů.

6. Vít Kolář

6.1 Vektorová knihovna

Úkolem bylo naprogramovat jednoduchou šablonu pro obecný n-rozměrný vektor. V aplikaci se doopravdy používají dvojrozměrné, třírozměrné a čtyřrozměrné vektory. Šablona má dva parametry, a to skalární typ ze kterého se vektor skládá a dimenzi vektoru. V šabloně jsou potom všechny užitečné operace, které se dají jednoduše vyjádřit pro libovolnou dimenzi. Dále existují ještě specializované třídy, které z obecné šablony dědí a přidávají operace, nativní jen pro určité dimenze (například vektorový součin může být definován jen pro R3 a R7).

Základní šablona vypadá takto:



Na začátku vidíme množství konstruktorů, jež inicializují jednotlivé prvky pole. Pokud je v konstruktoru méně prvků než dimenzí vektoru, vyšší dimenze jsou inicializovány na nulu. Toto chování je potom změněno u čtyřrozměrného vektoru, který je použit především jako datový typ barvy a čtvrtá souřadnice má implicitní hodnotu 1 (alfa kanál). Prázdný konstruktor nechává vektor neinicializovaný.

K prvkům se přistupuje pomocí operátorů [], dvě verze jsou pro konstantí přístup (vrací skalární hodnotu), nebo pro zápis hodnoty (vrací referenci na skalární hodnotu).

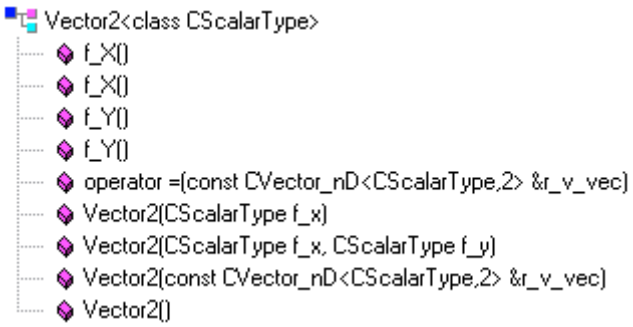
Jsou zde primitivní aritmetické operace (sčítání, odčítání, násobení, dělení) mezi dvěma vektory a mezi vektorem a skalárem, operace mezi dvěma vektory jsou prováděny mezi odpovídajícími složkami. Je zde unární minus.

Dalšími jednoduchými operacemi jsou minimum a maximum, ty existují každá ve dvou verzích, první verze jsou unární a vrací nejmenší (největší) složku vektoru, zatímco druhé verze jsou binární a vracejí vektor složený z meších (větších) odpovídajících složek ze dvou vektorů.

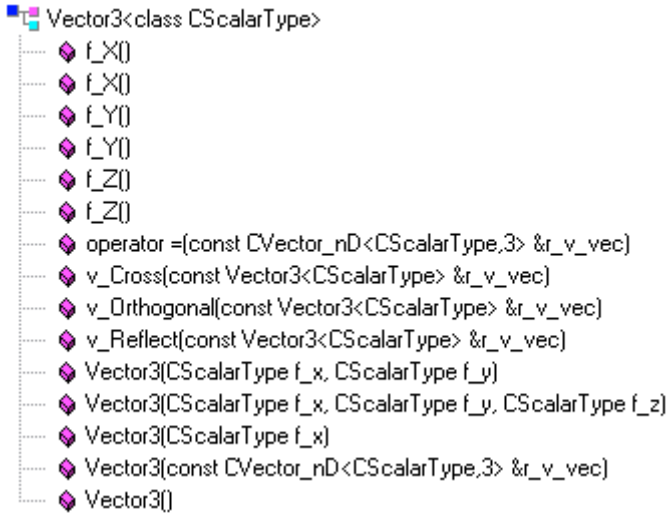
Funkce f_Sum vrací skalární sumu všech složek vektoru.

Mezi základní operace lineární algebry patří délka vektoru, délka vektoru na druhou (šetří odmocninu, formálně je sklárním součinem vektoru sama se sebou), skalární součin mezi dvěma vektory a normalizace vektoru na délku 1, popřípadě na jinou délku.

Specializovaná třída pro dvourozměrný vektor nepřidává nové operace, pouze zpřístupňuje složky vektoru pomocí pojmenovaných funkcí:



Specializovaná třída pro třírozměrný vektor přidává několik nových operací:

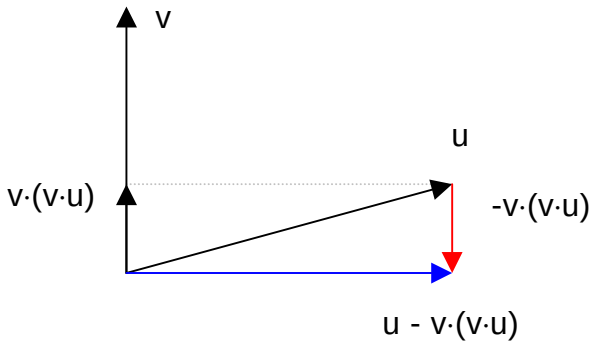


Jedná se o funkce v_Cross, tedy vektorový součin. Vektorový součin je počítán podle jednoduchého cyklického pravidla, známého pod mnemotechnickou pomůckou „xyzzy“, tedy pokud máme vektory *u* a *v*:

$$w.x = u.y \cdot v.z - u.z \cdot v.y$$
$$w.y = u.z \cdot v.x - u.x \cdot v.z$$
$$w.z = u.x \cdot v.y - u.y \cdot v.x$$

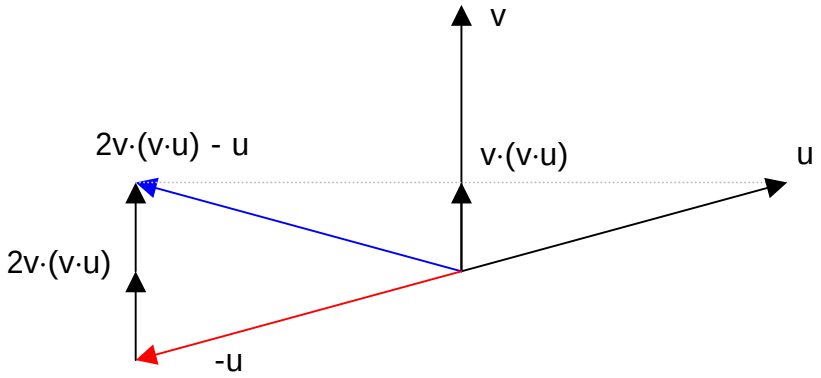
Potom platí $w = u \times v$.

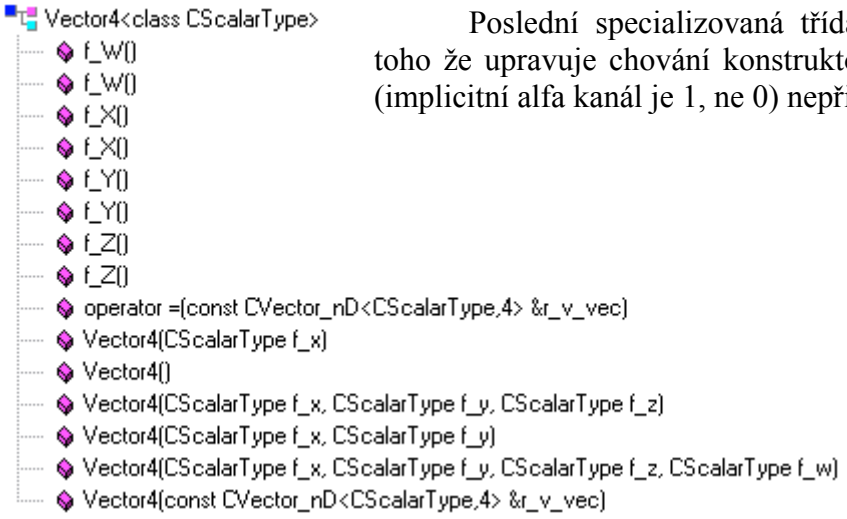
Další funkcí je v_Orthogonal, jež počítá vektor kolmý na daný vektor *u*, v rovině s jiným vektorem *v*. Funkce je všeobecné známá jako Gram-Schmidt ortogonalizace. Její funkce je přitom velmi jednoduchá, využívá se promítnutí vektoru *u* na vektor *v* a následné odečtení výsledku od *u*. Tím se *u* „narovná“ tak, že je kolmý na *v*. Implementace počítá s tím že *v* má délku jedna. Operaci znázorňuje následující obrázek:



Pro rekapitulaci, vektor mířící vzhůru, *v* je rovnoběžný s vektorem na něj promítnutým, *v·(v·u)*, kde násobení v závorce je skalární součin a násobení závorkou je tedy součin vektoru se skalárem. Červený vektor je tentýž, jen obrácený a modrý je původní vektor *u*, ke kterému byl červený vektor přičten. Ten je již výsledným kolmým vektorem na *v*.

Poslední nová funkce v_Reflect funguje na velice podobném principu, počítá vektor *u*, který se odrazil od povrchu s normálou *v*. Opět se počítá projekce *u* na *v*, jen tentokrát se obrátí samotné *u* a projekce se k němu dvakrát přičte:





Poslední specializovaná třída čtyřrozměrného vektoru, krom toho že upravuje chování konstruktorů s méně než čtyřmi parametry (implicitní alfa kanál je 1, ne 0) nepřidává nic nového.

6.2 Zápis obrázku ve formátu TGA

TGA je obrazový formát Truevision (nyní Pinacle) Targa [8], který byl dříve nativní pro zobrazovací zařízení Hercules. Obrázek v tomto formátu může mít barevnou tabulku, tabulku odstínů šedi, nebo RGB barvy, navíc může obsahovat alfa-kanál a může být komprimován pomocí RLE. Podporované bitové hloubky jsou 5 nebo 8 bitů na kanál.

Formát ve kterém je obrázek zapisován v naší aplikaci je jednoduchý nekomprimovaný RGB, 8 bitů na pixel. Za hlavičkou tak následují již přímo hodnoty jednotlivých pixelů.

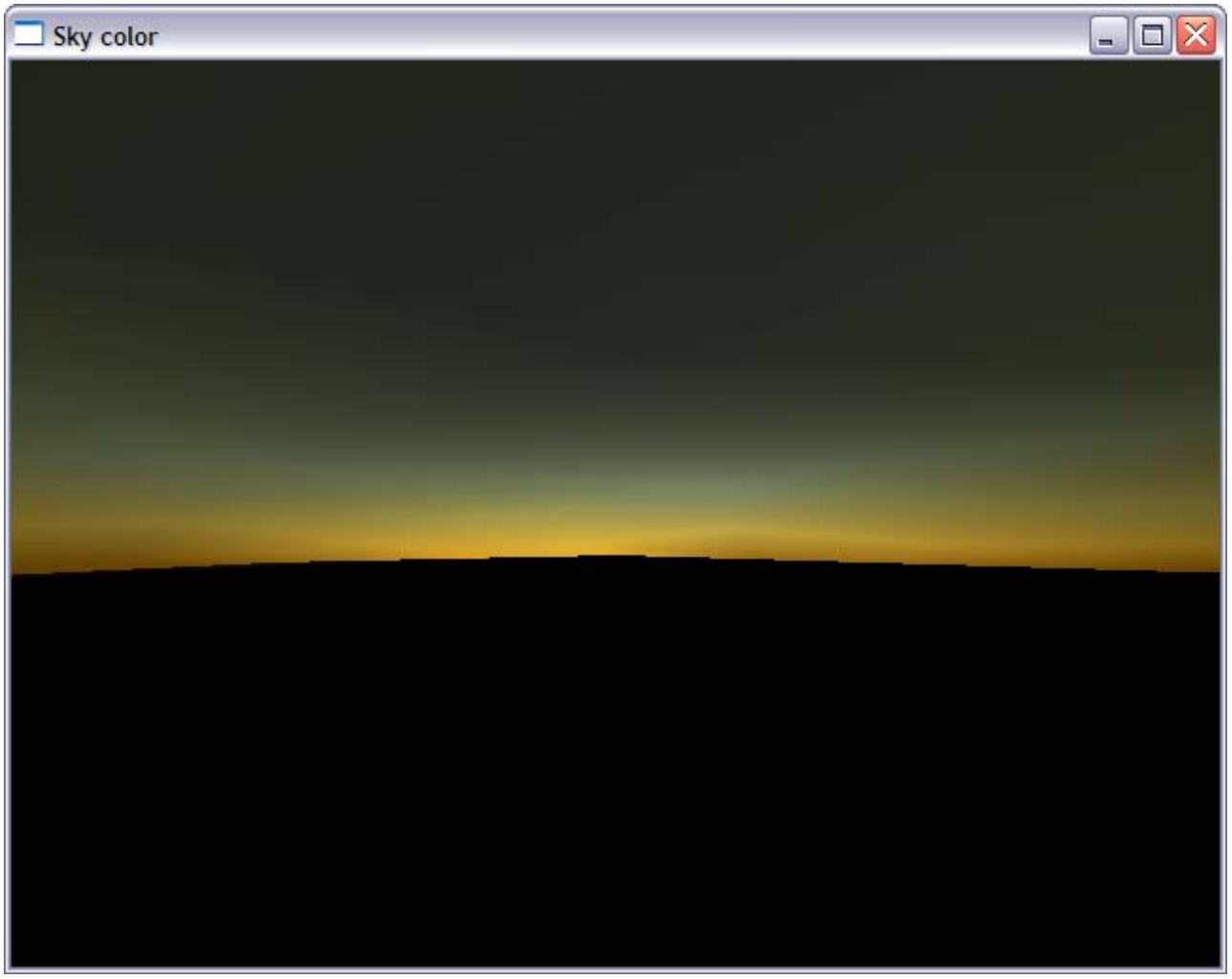
7 Lukáš Polok

7.1 Simulace atmosféry

Byl využit paper Nishita ‘97 [1], jež popisuje metodu simulace šíření světla v atmosféře. Schopnost šíření světla je ovlivněna jednak molekulami vzduchu (raileigh scattering) a také vodními parami obsaženými v atmosféře (mie scattering). Raileigh scattering závisí na vlnové délce, mie scattering nezávisí. Model šíření světla je dosti zjednodušený, avšak výsledky jsou realistické. Oproti klasické simulaci která by buďto využívala pouze tři základní vlnové délky (pro R, G a B), nebo spektrum černého tělesa o teplotě 5200K, jež se slunečnímu spektru blíží, naše simulace využívá ASTM G173-03 Reference Spectra [2], což jsou hodnoty slunečních emisí, naměřené nad atmosférou s krokem 1nm.

Simulace není omezená na pohled zevnitř atmosféry, první simulace probíhaly s kompletní planetou:

(obrázky nejsou nějak upravené, po vyrovnání histogramu
případně gamma korekci by zřejmě vypadaly i lépe)


Sky color

7.2 Raytracing terénu

Pro raytracing terénu byla použita akcelerační struktura typu uniform grid, pro vyplnění se používají jednoduše jen bounding boxy jednotlivých trojúhelníků. Původně byl vyvinut objemový scanline-based rasterizér, avšak pro danou geometrii nebyl o moc efektivnější, avšak o dost pomalejší. Podle všeho to však byl ojedinělý pokus, jedinný volumetrický rasterizer jež se ve spojení s uniform grid používá je 3D DDA (ten používáme též), aspoň podle toho co se podařilo najít na google.

Terén se pro raytracing převádí z heightmapy na trojúhelníkovou síť, jež se vloží do unifrom grid a poté se raytracuje. Pro detekci průsečíku paprsku s trojúhelníkem byl použit T. Moller, B. Trumbore, Fast, minimum storage ray-triangle intersection [6].

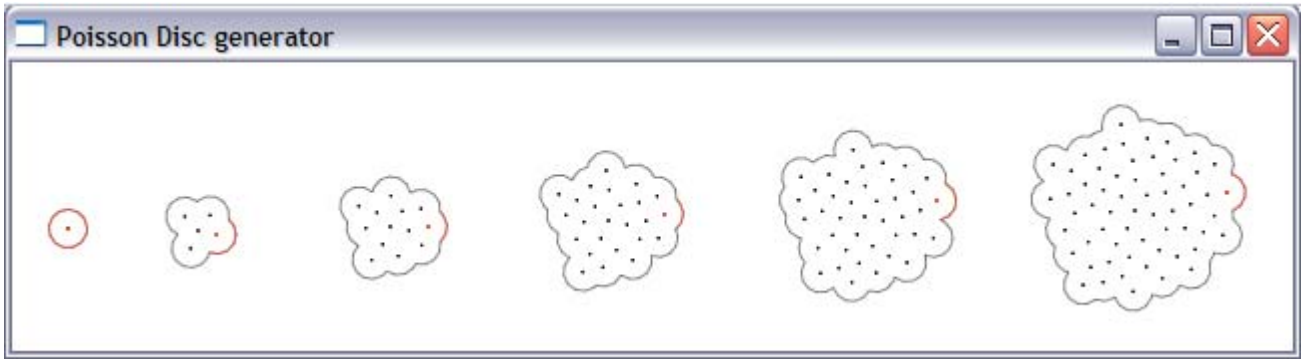
7.3 Multisample antialiassing

Protože po několika testech se ukázalo že generování skyboxu bude rozhodně dlouhodobější a náročný výpočet a proto bylo rozhodnuto zamířit spíše na kvalitu výsledku. Proto byl implementován jednoduchý multisample antialiassing. Jako vzorkovací jádro v aplikaci slouží generátor Poisson disc rozložení.

Původním nápadem bylo implementovat dart throwing, ten však má několik problémů. Jedním je délka výpočtu, neboť nikdy není jasné kdy je výpočet ukončen. Dalším jsou potom možné ‚díry' v distribuci, pokud se výpočet náhodou ukončí příliš brzy.

Jako jednoduché rozšíření algoritmu byla implementována struktura volných výsečí. Body se potom generují tak, že se umisťují na volné okraje jiných bodů. Tím se zaručí lepší hustota bodů a zároveň je lehké definovat podmínku ukončení algoritmu. Algoritmus taktéž běží docela rychle. Podobný nápad byl po implementaci vzevrubným zkoumáním internetu nalezen v [9], avšak tam autoři používají složitější strukturu která obsahuje větší počet výsečí. Naše implementace obsahuje pouze jednu výseč na jeden bod, díky pořadí operací je zaručeno že to bude dostačující (výseč nikdy nebude rozdělena na dvě disjunktní výseče). Náš nezávisle nalezený postup je navíc o něco rychlejší.

Následující obrázek ukazuje několik kroků generátoru v akci:



V aplikaci byl generátor mírně upraven, aby generované množiny bodů byly opakovatelné (navazovaly na sebe).

7.4 Osvětlovací model terénu

Terén je osvětlen přímým světlem ze slunce, je použit jednoduchý dokonale difůzní osvětlovací model. Jako další světlo je potom použito světlo, přicházející z okolí. Proto je z každého bodu vystřelováno několik ambient occlusion paprsků, jež se rozprostírají po polokouli, centrované okolo normály povrchu a je vzorkováno světlo, přicházející z atmosféry. Tím je dosaženo realističtějších výsledků.

Ambient occlusion paprsky mají opět Poissonovské rozložení.

Terén je navíc ještě zahalen do mlhy, jež je směrem s klesající výškou exponenciálně hustší. Barva mlhy závisí na barvě horizontu v daném směru, bylo tak dosaženo velice romantických výsledků.

Všechny parametry simulace lze nastavit v konfiguračním skriptu.

7.5 Texturování terénu

Pro texturování terénu je v konfiguračním skriptu možné nadefinovat několik vrstev textur. Každá vrstva má svou matching funkci, která určuje na které části terénu se má textura aplikovat. Parametry jsou normála a výška texturovaného bodu, ty jsou použity jako argumenty do gaussovských rozložení a výstupem je váha texturovací funkce v daném bodě. Tím bylo například u testovací scény Grand Canyonu dosaženo odlišného texturování pro úbočí svahů a náhorní plošiny. Stejným chématem by neměl být problém umístit například trávu do údolí a sníh na vrcholky hor.

Každá texturovací funkce má parametr generování texturovacích souřadnic (volume, x-z nebo y), funkci perturbace souřadnic (fraktálový noise) a vlastní funkci pro generování obsahu textury. Ta je fraktálním perlinovým šumem, avšak je možné šumovou funkci použít jako argument v dalších funkcích jako například exp, abs, sqrt, ... Konečný výsledek šumové funkce je poté použit pro určení barvy z palety. Paleta je funkce definovaná na diskrétní množině bodů, mezi nimi se barva lineárně interpoluje.

7.6 Formát konfiguračního skriptu

Konfigurační skript obsahuje parametry a příkazy potřebné ke generování terénu, jeho syntaxe je až podezřele podobná syntaxi kaskádových stylů (.css). Skládá se z těchto částí:

Větev output. Obsahuje parametry výstupu jako rozlišení, nastavení multisamplingu a postprocessing obrazu (normalizace histogramu, jas / kontrast a gamma korekci). Větev output vypadá takto:

```
output {
	texture-size: 1024;
	sample-num: 16;
	ambient-occlusion-sample-num: 64;
	shadow-sample-num: 15;
	shadow-sample-offset: 5; /* this ends up being a lot of extra rays per pixel */
	color-system: 'SMPTE';
	/ * one of NTSC / EBU (PAL / SECAM) / SMPTE / HDTV / CIE / Rec709 (CIE's r709) */
	normalize: true;
	gamma: 1;
	bias: 0; scale: 1;
}
```

Další větví je atmosphere. Tady se nastavují vlastnosti atmosféry, jednak poloměr planety a atmosféry, výška ve které se nalézá pozorovatel, vektor směru slunce (tím se určuje denní doba, nemělo by být těžké implementovat rozšíření na zadávání času a zeměpisných / GPS souřadnic) a poté parametry modelu atmosféry. Jejich význam se dá nalézt v [1].

```
atmosphere {
	planet-r: 6378e3f;
	atmosphere-r: 6500.0e3;
	observer-altitude: 500;
	sun-direction: 1 .1 3; /* x points to right, y upwards, z to front */

	u: .7;
	k-mie: .25e-12;
	k-raileigh: .15e-31; /* do not touch if you don‘t have to */
}
```

Protože simulace atmosféry je náročná a ve výpočtu osvětlení terénu se její výsledky několikrát znovupoužijí, celá atmosféra se předpočítá do dočasné textury. Její parametry se také nastavují, je tu rozlišení a vertikální odříznutí (nemá smysl počítat hodnoty které jsou stejně skryty pod terénem):

```
sky-color {
	texture-size: 64; /* sky color is pre-calcuated to texture with lower size */
	y-cutoff: -.2; /* anything below -.2f won't be calculated and is set to black */
}
```

Pro vzorkování horizontu je vyhrazena další větev, která určuje výšku horizontu a parametry vzorkovacího jádra:

```
horizon-color {
	gauss: 0 1 0 .5; /* filter parameters */
	cutoff: .1;
	y-range: -.1 .1; /* sampled from sky-color cubemap; this limits y-coordinate */
}
```

Další, zatím nevyužitá větev je větev s mraky. Nemá cenu ji zatím příliš popisovat neboť stejně není implementována:

```
clouds {
	layer {
		altitude: 10e3;
		thickness: 100;

		perturb: 0 1 100 400 .5;
		noise: 0 1 100 400 .5; /* bias octave scale amplitude durability */
		function: exp;
	}
}
```

Poslední a asi největší větví je nastavení generátoru terénu. Je zde velikost heightmapy, sekvence operátorů generujících terén, parametry mlhy a sekvence generátorů textur:

```
terrain {
	width: 256;
	height: 256;
	tile-size: 100; /* heightmap params */

	ambient: .6; /* ambient light is gathered arround from the sky */
	diffuse: .6; /* diffuse light comes from the sun */
```

```
	generator {
		fill: 0;
		fault-formation: 50 .125 12345;
		mean: 50;
		particle-depo: .5 .5 1 10 50000 12378;
		normalize: 0 1;
		exp; /* pixel-wise exp function */
		mean: 3;
		normalize: -4000 0;
	}
	/* generates grand-canyonish scenery */

	fog: -3500 1000 .0005f; /* base y, height, thickness (multiplier) */

	texture {
		texgen: y-axis;
		perturb: 0 1 .002 40 .5;
		noise: -.5 6 .01f 1 .7; /* bias octave scale amplitude durability */
		function: exp;

		palette: 0 0 0 0;
		palette: .5 .839 .486 .423;
		palette: 1 .988 .875 .690; /* first is palette position, others are RGB */

		height-gauss: 0 0 0 1;
		ref-normal: 0 1 0;
		slope-gauss: 1 -1 1 .2;
		match-distort: 0 0 1 1 .5;
	}

	texture {
		texgen: xz-plane;
		perturb: 0 1 .002 40 .5;
		noise: -.5 6 .01f 1 .7; /* bias octave scale amplitude durability */
		function: exp;

		palette: 0 0 0 0;
		palette: .5 .839 .486 .423;
		palette: 1 .99 .875 .69; /* first comes palette pos, next are RGB coeffs */

		height-gauss: 0 0 0 1;
		ref-normal: 0 1 0;
		slope-gauss: 0 1 1 .7; /* bias multiplier center sigma */
		match-distort: 0 0 1 1 .5;
	}
}
```

7.7 Prezentace výsledků

Výsledky jsou nakonec zkonvertovány do fixed-point hodnot a uloženy jako jednotlivé stěny skyboxu. Nebyl použit navrhovaný formát skyboxu se čtyřmi nebo pěti stěnami ale namísto toho byla použita cubemapa. Ta poslední dobou patří mezi velice populární rozšíření grafického rozhraní OpenGL a pro skyboxy se používá hojně. Navíc je stále možné výsledky použít jako navrhovaný skybox se čtyřmi stěnami.

Program umí po dokončení výpočtu výsledky i zobrazit v jednoduchém glut okně, myší se dá otáčet kamerou a prozkoumat tak vygenerovaný skybox.

Použité reference

[1] – Nishita‘s Siggraph '93 paper - http://nis-lab.is.s.u-tokyo.ac.jp/~nis/cdrom/sig93_nis.pdf
[2] - ASTM G173-03 Reference Spectra - http://rredc.nrel.gov/solar/spectra/am1.5/
[3] - Generování terénu - http://collective.valve-erc.com/index.php?doc=1040644009-23694100
[4] - Particle deposition - http://www.lighthouse3d.com/opengl/terrain/index.php3?particle
[5] - Perlin noise - http://www.noisemachine.com/talk1/
[6] - T. Moller, B. Trumbore, Fast, minimum storage ray-triangle intersection - http://www.cs.virginia.edu/~gfx/Courses/2003/ImageSynthesis/papers/Acceleration/Fast%20MinimumStorage%20RayTriangle%20Intersection.pdf
[7] - A Simple and Efficient Error-Diffusion Algorithm - http://www.iro.umontreal.ca/~ostrom/publications/pdf/SIGGRAPH01_varcoeffED.pdf
[8] - Truevision Targa - http://en.wikipedia.org/wiki/Truevision_TGA
[9] - A Spatial Data Structure for Fast Poisson-Disk Sample Generation - http://www.cs.virginia.edu/~gfx/pubs/antimony/antimony.pdf