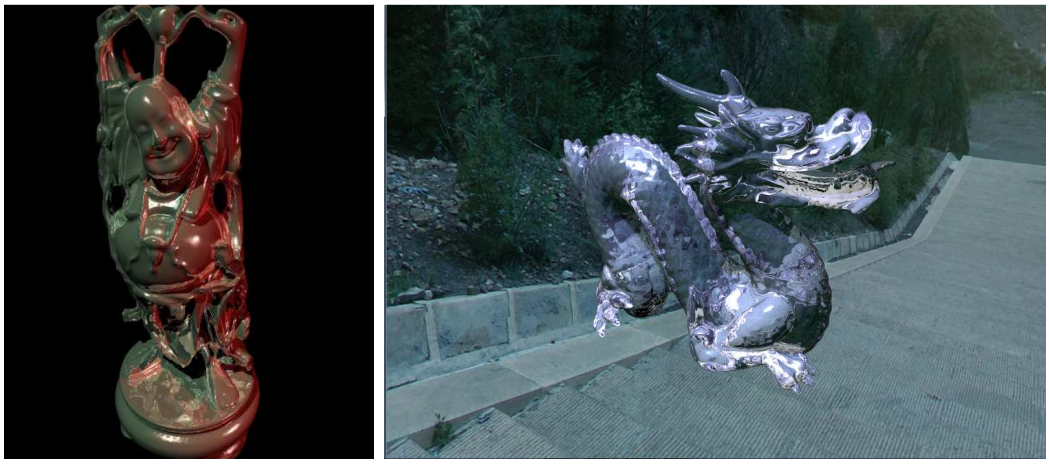
Environmentální mapa

Cílem dnešního cvičení je pokusit se naimplementovat podporu pro zobrazování environmentálních map do stávajícího ray traceru. Výsledek by mohl vypadat například takto:



Obrázek 1: Příklad bez a s využitím environmentální mapy pro lepší znázornění odrazů a lomů trasovaných paprsků

Co k tomu budete potřebovat? Nejprve musíme získat vhodné textury. Takové můžete najít např. na stránce:

```
http://www.humus.name/index.php?page=Textures
```

V archivu je uložena šestice textur, které když promítneme na jednotlivé stěny pomyslné krychle, která obsahuje celou naši scénu a délka její hrany je nepoměrně větší než jsou rozměry této scény, resp. její hrana má nekonečnou délku, pak získáme dojem celistvého prostoru, který obklopuje pozorovatele ve všech směrech pohledu bez ohledu na jeho aktuální polohu.

Postup implementace

Nejprve si vytvoříme novou třídu `CubeMap`, která bude zabalovat celou funkcionalitu. Konstruktoru můžeme předat jména všech šesti souborů s vybranými texturami. Jednotlivé textury lze s výhodou reprezentovat jako instance třídy `Texture`, která má již implementovánu např. bilineární interpolaci v rámci metody `get_texel`.

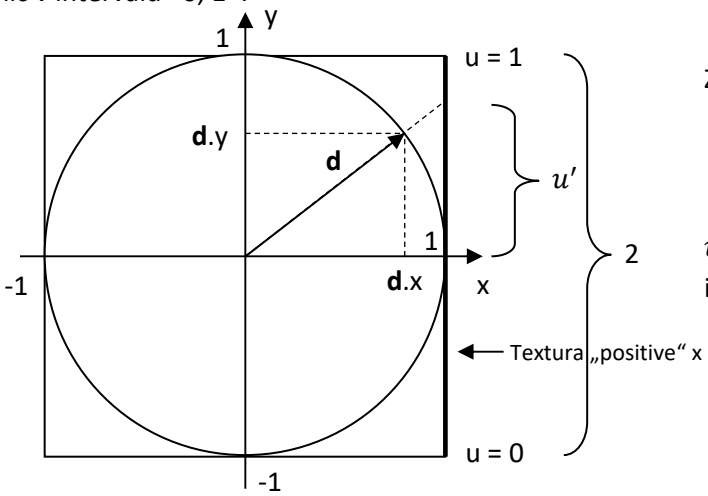
Výše uvedené již částečně naznačuje, jak to celé bude fungovat. U paprseku, který nezaznamenal žádnou kolizi s objekty scény a letí do nekonečna, chceme, aby jako RGB hodnotu vrátil barvu pozadí. V úvodních cvičeních jsme uvažovali např. černé pozadí a paprsek vrátil hodnotu (0, 0, 0). Nyní bychom chtěli tuto konstantu nahradit barvou odpovídající barvě environmentální textury v daném směru letu paprsku. Jak již bylo řečeno, u tohoto paprsku nás tedy nezajímá jeho počátek **o**, ale jen směrový vektor **d**. Pouze na základě vektoru **d** se tedy musíme rozhodnout, ze které z šesti textur budeme načítat příslušný texel o souřadnicích (*u*, *v*).

Správnou osu (x, y nebo z) poznáme podle nejdelší (tj. v absolutní hodnotě) složky vektoru **d**. K tomuto můžete využít metodu `Vector3.LargestComponent(true)`, která vrátí index

nejdelší složky daného vektoru. Nyní víme, že barvu pozadí chceme načítat z textur např. positive x nebo negative x. Mezi „positive“ a „negative“ se rozhodneme podle znaménka dominantní složky vektoru **d**. Máme-li rozhodnuto, ze které textury budeme načít texel pozadí (např. positive x), musíme ze zbylých dvou složek vektoru **d** spočítat jeho texturovací souřadnice (u, v). To provedeme v metodě `get_texel` následovně:

```
Color4 CubeMap::get_texel( Vector3 & direction )
{
      …
      case POS_X:
            const float tmp = 1.0f / abs( direction.x );
            u = ( direction.y * tmp + 1 ) * 0.5f;
            v = ( direction.z * tmp + 1 ) * 0.5f;
      break;
      …
      Color4 texel = maps_[map]->get_texel( u, v );
      return texel;
}
```

Příslušné složky směrového vektoru **d** podělíme velikostí té nejdelší. Tím se dostáváme u jednotlivých souřadnic do rozsahu <-1, 1>, což musíme dále převést do přípustného rozsahu texturovacích souřadnic v intervalu <0, 1>.



Z podobnosti trojúhelníků plyne:

$$\frac{u'}{1} = \frac{d.y}{d.x}$$

u' musíme ještě normalizovat do intervalu <0, 1>

$$u = \frac{u' + 1}{2}$$

Obrázek 2: Znázornění postupu výpočtu texturovacích souřadnic

U ostatních pěti směrů postupujeme obdobně.

Problém může nastat v nesprávné orientaci jednotlivých textur, která bude mít za následek nenavazování jednotlivých textur v rozích pomyslné krychle. Náprava tohoto stavu spočívá v postupném směřování kamery ve směrech (1, 0, 0), (-1, 0, 0), atd., kdy sledujeme, zdali vidíme požadovanou texturu v požadované orientaci. V případě neshody můžeme provést buď inverzi příslušné texturovací souřadnice (např. $u = 1 - u$) nebo překlopení samotného obrázku kolem požadované osy (např. pomocí externího grafického editoru nebo přímo s využitím druhého parametru metody `LoadTexture`).