Vykreslovač 3D objektů v konzoli

Cíl programu:

Rasterizace 3d objektů v formátu .obj pomocí ascii znaků do konzole

Nezáleží mi až tolik na rychlosti, jinak bych to nepsal v pythonu, musí to být ale snesitelné.

Jednoduché osvětlení

Jak s programem pracovat:

Po spuštění v konzoli (je třeba dostatečne zvětšit velikost okna konzole), uvidíte pouze blikající svislé čáry vlevo a v pravo, program se defaultně zapne ve **vykreslovácím režimu**. Aby se dalo cokoliv dělat musí se nejprve přepnout do **režimu příkazového** stisknutím klávesy **ESC.**

V příkazovém režimu se dají používat příkazy.

Argumenty se odděluují mezerou, zde píšu ve tvaru: ”příkaz [argument1] [argument2] ...”

Některé argumenty vyžadují tvar celého čísla (int) nebo desetinného (float)

**Použitelné příkazy jsou tyto:**

**q, quit** – program se vypne

**load** [relativní cesta k .obj souboru (string)] - načtě daný model

!!!Program předpokládá, že každá stěna modelu je trojúhelník, v opačném případě vykresovaný model nemusí vypadat správně a je třeba ho triangulizovat!!!

**setwindow, setw** [šířka (int)] [výška (int)] - nastaví rozlišení vykreslovaného okna

**setlight, setl** [x (float)] [y (float)] [z (float)] - nastaví směr přicházejícího světla na vektor (x,y,z)

**setspeed, sets** [rychlost (float)] - nastaví rychost kamery v vykreslovacím režimu

Po zmáčknutí enter, se příkaz vykoná a automaticky přepne program do vykresovacího režímu. Ve vykreslovacím režímu se kamera dá ovládat pomocí kláves WASD, mezerník, levý shift, a šipek a to takto:

W - dopředu

S – dozadu

A – vlevo

D – vpravo

Mezerník - nahoru

Levý shift - dolů

Šipkama se kamera otáčí do příslušného směru.

Pozor! Program není stavěný na příliš složité modely

Tip: Pokud ve vaši konzoli jde zmensit velikost písma, můžete tím značně zlepšit rozlišení

O kódu programu:

Principy, kterýma se kód řídí, a které dodržovat při dalších úpravách pokud je tak možno:

Třídy (class) jsou použity jen pro zabalení více proměnných dohromady, nemají žádné metody

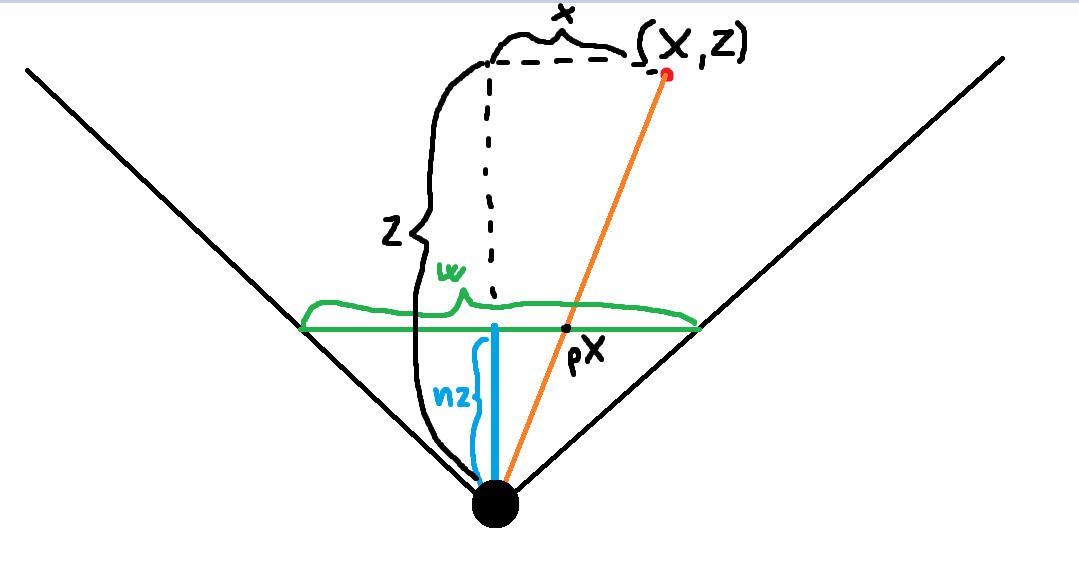
Funkce nepoužívají, ani neupravují žádné globální proměnné.

Princip fungování:

Každou funkci popisovat nebudu, protože si můžete kód přečíst sami, jména funkcí jsou o sobě dost vypovídající. Popíšu ale základní princip a některé zajímavější části.

Základ je umět promítnout bod z 3d prostoru do 2d prostoru obrazovky (nějak vypočítat kde na obrazovce bude).

Uvažujme tedy následující pohled na scénu z vrchu kde červený bod (x,z) chceme zobrazit na bod pX na obrazovce (zeleně), černé kolečko znamená pozici kamery která se dívá ve směru z+, nz je vzdálenost mezi myšlenou kamerou a obrazovkou:



Z obrázku vidíme: X / Z = pX / nZ -> pX = X \* nZ / Z

To samé platí i pro souřadnici Y.

Když se na rovnici podíváme tak to dává smysl, čím je něco dál (větší Z), tak to na obrazovce vypadá menší. Co je, ale to nZ? NZ ovlivnuje úhel který kamera vidi, pokud nz = w/2 tak kamera vídí úhel 90 stupnu, což je celkem vhodné.

Funkce “project(origin, cam)” dělá v podstatě tohle, ale bere navíc v potaz otočení a pozici kamery, to jsou, ale věci které si dokážete sami domyslet jak by se vyřešily.

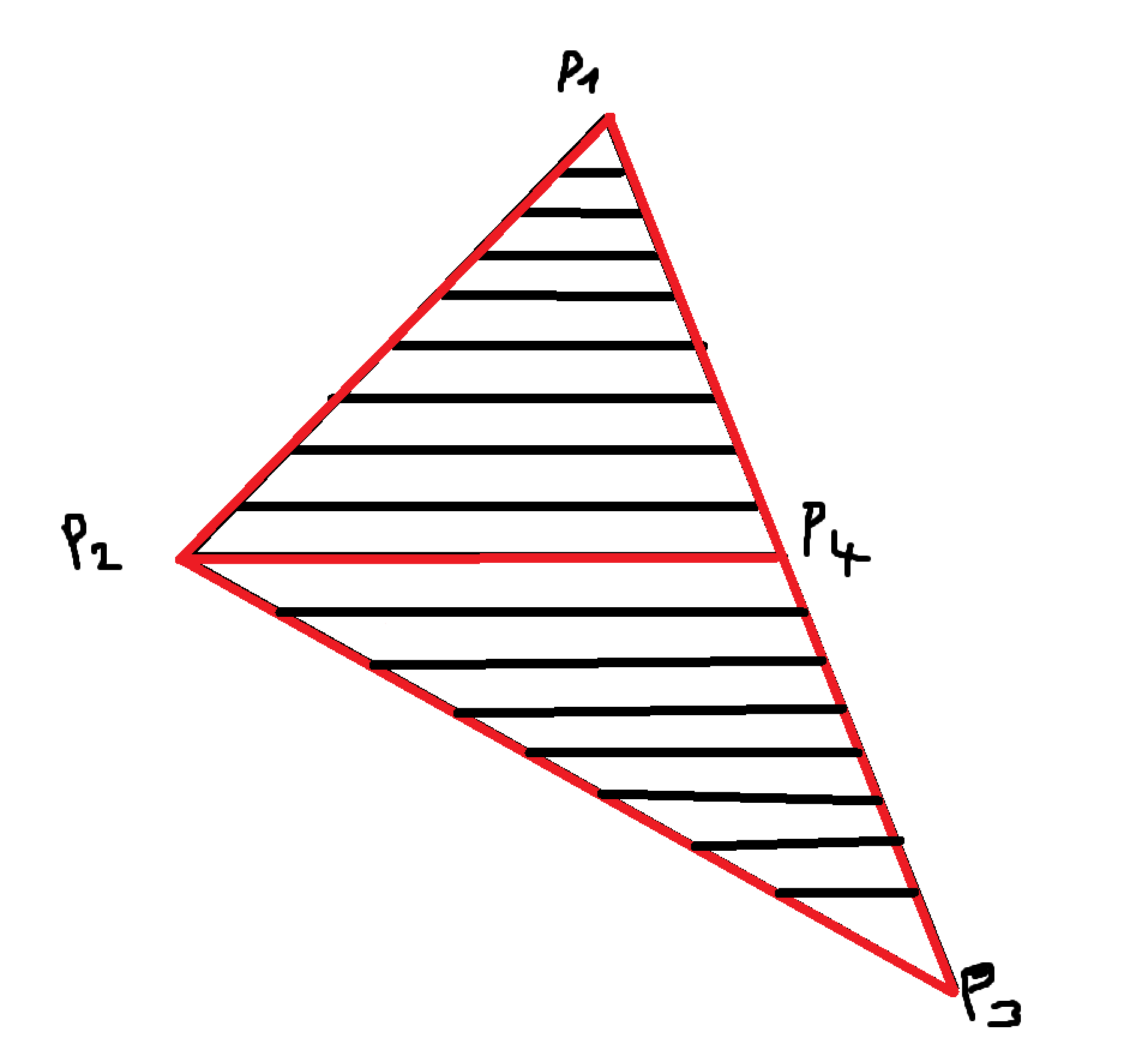
Máme pozici bodů na obrazovce, co dál? Mohli bychom je jednoduše pospojovat příslušné body pomocí algoritmu na kreslení čar a měli bychom nakreslenou takovou kostru. Cíl je, ale kreslit stěny a na to potřebujeme kreslit trojuhelníky. Stěny jsou reprezentovány jako trojice indexů bodů které danou stěnu tvoří. Na vyplnění trojúhelníku mezi nimi existuje řada algoritmů.

Já jsem zvolil tento:

Trojúhelník p1,p2,p3 rozdělím horizontální čarou na dva trojúhleníky s plochým spodkem a vrchem.

Bod p4 každý snadno dopočítá.

Potom postupně řádek po řádku vyplním vrchní a spodní trojúhelník (jak přesně na to si každý může rozmyslet sám).



Čím ale takové trojúhelníky vyplnovat?

To závísí na množství dopadajícího světla.

To můžeme zjistit pomocí skalárního součinu normálového vektoru a vektoru globálního světla.

Oba vektory mají délku 1, pokud mají stejný směr tak míra světla je maximální neboli 1, pokud jsou kolmé tak nulová, což nějak tak odpovídá tomu co chceme vidět.

Potom je třeba nějak převést rozsah 0 – 1 na ascii znaky, já jsem vybral: " .:-=+\*#%@"

Poslední zádrhel je je možné že překreslíme trojúhelník v popředí trojúhelníkem více vzadu.

Je opět více způsobů řešení, buď třídit trojúhelníky podle vzdálenosti, nebo jak jsem to udělal já je mít separátní pole které pro ukládání vzdálenosti každeho bodu na obrazovce a přepisovat ho jenom pokud je nový bod blíže než byl ten předchozí.

Popis je poměrně skromný, ale zbytek věcí co program dělá jsou jednoduché a jakýkoliv aspoň trochu schopny programátor by dokázal rychle přijít jak na to.

Program není perfektní a na mnohých místech by se dal snadno zlepšit, jak do rychlosti tak do výsledného obrazu!