Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

KIV/NET

Ray-tracing

Jméno a příjmení studenta: František Koleňák
Vypracováno: 28.05.2020

1. Zadání úlohy

Vytvořte programové jádro pro software renderující scénu metodou sledování paprsku (viz rychlý nástin). Navrhněte vhodné datové struktury pro následující prvky programu:

- paprsek
- obecný objekt scény
- nejméně dva konkrétní objekty scény (koule, plocha)
- zdroj světla
- scéna (sdružuje objekty, světla a kameru)
- kamera (generuje paprsky a výsledný obraz)
- Umožněte uložení a načtení souboru se scénou (navrhněte formát takového souboru) a výpočet obrazu scény do souboru. Implementujte všechny potřebné metody. Jako výstup postačí textový soubor s čísly reprezentujícími hodnoty jasu jednotlivých pixelů vypočteného obrazu (stačí intenzitní úrovně, rozlišení barvy není nutné).

Uživatelské rozhraní raytraceru:

Vytvořte nad první úlohou grafické uživatelské rozhraní, které umožní:

uložit scénu do XML souboru
načíst scénu z XML souboru
zobrazit scénu jako 2D náhled v jednotlivých nárysech
přidávat objekty a světla do scény
pohybovat objekty, světly a kamerou ve scéně (stačí posuvný pohyb, není nutno uvažovat rotace)
nastavovat vlastnosti objektů
spustit rendering scény ve zvoleném rozlišení a zobrazit jeho výstup na obrazovku
kdykoliv přerušit rendering
uložit výstup renderingu do bitmapového souboru

2. Úvod

„Sledování paprsků (z anglického ray tracing) je metoda renderování (výpočtu a zobrazení) 3D počítačové grafiky; přesněji řečeno metoda globálního osvětlení. Na rozdíl od běžného života, kdy se paprsky pohybují od zdroje, odráží se a lámou, až se nakonec střetnou s okem pozorovatele, zde paprsky vycházejí z kamery. To protože ze zdrojů světla vychází nekonečné množství paprsků a nedalo by se v rozumném čase spočítat, které dopadnou na pixely plátna, přes které se oko dívá. Tuto techniku používají například počítačové programy POV-Ray nebo Blender.“[1]

Naším úkolem bude:

- Definovat kameru pomocí vhodných parametrů tak, aby ji bylo možné libovolně umístit a natočit v prostoru
- V prostoru umístit zdroj světla
- Implementovat podporu základních těles (koule a plocha)
- Pro každý pixel vystřelte pouze jeden paprsek pro zjištění vystínované barvy zasaženého tělesa a druhý paprsek pro výpočet stínu
- Umožnit uživateli uložení vyrenderovaného obrázku
- Uložit scénu do souboru
- Volitelně: přidat více těles, umožnit vlastnosti objektů (např. průhlednost)

3. Známé metody

Pseudokód Raytracingu[1]:

```
Pro každý pixel v obraze
{
  Vytvoř paprsek mířící z oka skrze pixel roviny, kterou renderuješ
  Inicializuj NejbližšíT na NEKONEČNO a NejbližšíObjekt na ŽÁDNÝ

  Pro každý objekt ve scéně
  {
     Jestliže paprsek protíná tento objekt
     {
        Jestliže je vzdálenost protnutí t menší než NejbližšíT
        {
           Nastav NejbližšíT na vzdálenost protnutí t
           Nastav NejbližšíObjekt na tento objekt
        }
     }
  }

  Jestliže je NejbližšíObjekt roven ŽÁDNÝ
  {
     Vyplň tento pixel barvou pozadí
  }
  Jinak
  {
     Vyšli stínový paprsek ke každému zdroji světla (zjištění zdali je ve stínu pro difuzní a spekulární složku)
     Jestliže je povrch objektu odrazivý, vytvoř odražený paprsek (rekurze)
     Jestliže je povrch objektu průhledný, vytvoř refrakční paprsek (lom světla, rekurze)
     Použij NejbližšíObjekt a NejbližšíT k výpočtu stínovací funkce (barva objektu)
     Vyplň tento pixel výslednou barvou stínovací funkce
  }
}
```

Phongovo stínování(anglicky Phong shading)[2]: je soubor technik používaných v počítačové grafice. Tyto techniky zahrnují především model odrazu světla z povrchu materiálu (Phongovo osvětlení) a odhad barvy pixelu založený na interpolaci normály povrchu (Phongova interpolace, obvykle označována per-pixel).

4. Navrhovaná modifikace, metoda

Raytracing samotný má strukturu velice podobnou uvedenému pseudokódu.

Struktura - Kamera

Nutné definice pro kameru jsou pozice kamery, směrový vektor, a vektor směrem nahoru. V konstruktoru jsem si dopočítal vektor doprava, tak že vektor doprava je rovný $-1*(nahoru \times smer)$.

Struktura - Světlo

Bodové světlo je nadefinováno svojí polohou, barvou(vždy bílá) a intenzitou.

Struktura - Koule

Koule je dána středem (S) a průměrem (R). Dále má parametry barva, vlastnosti a index lomu.

Výpočet průsečíku(viz také [3]):Nejdříve musíme mít splněnou podmínku, že směrový vektor je normalizovaný, potom koeficient a =1. Poté si vypočítáme koeficient $b=2\cdot\vec{s}^{T}(X_A-X_S)$, kde (X_A-X_S) je poloha počátku paprsku mínus poloha středu koule. Koeficient $c=(X_A-X_S)^T\cdot(X_A-X_S)-r^2$. Pokud diskriminant $d=b^2-4\cdot a\cdot c$ je větší než nula, pak průsečík s koulí existuje. Parametr t_1 je $t_1=(-b-\sqrt{d})/2$ a parametr t_2 má předpis $t_2=(-b+\sqrt{d})/2$. Ten parametr, který je menší, ale větší než 0, je parametr který použijeme. Ke zjištění bodu průsečíku použijeme rovnici: $Prusecik=X_A+t\cdot smer\ paprsku$. Normála ve zjištěném průsečíku je rovna $normala=2\cdot(Prusecik-X_S)$.

Struktura - Obdélník v rovině

Je dán minimem a maximem, ze kterého si dopočítáme zbylé body, kterými je definován. Minimum a maximum se zadávají v rovině xy (z = 0). Nejdříve nalezneme bod $Q=[x_{min}, y_{max}]$. Poté vektory p a q. Vektor p je rovný $p=min-Q$, q je $q=max-Q$. Normála obelníku je rovna $\vec{n}=\vec{q}\times\vec{p}$.

Jelikož při počítání průsečíku počítáme vlastně průsečík s rovinou (s předpisem $a\cdot x+b\cdot y+c\cdot z+d=0$) potřebujeme znát parametr **d**, ten se vypočte podle vzorce $d=-\vec{n}\cdot Q$. Poté dosadíme hodnoty a získáme parametr **t,** $t=\frac{-\vec{n}\cdot X_A+d}{\vec{n}\cdot\vec{s}}$. Pokud je $\vec{n}\cdot\vec{s}$ menší nebo rovno nule pak průsečík s rovinou neexistuje, pokud je větší musíme ještě zkontrolovat zda paprsek protnul obdélník. Vektor **b** je roven $\vec{b}=Prusecik-Q$, poté vypočteme **s** a **r**. $s=\frac{\vec{p}\cdot\vec{b}}{\vec{p}^{T}\cdot\vec{p}}$, $r=\frac{\vec{q}\cdot\vec{b}}{\vec{q}^{T}\cdot\vec{q}}$. Pokud jsou s i r v rozmezí od nuly do jedné, pak paprsek protl obdélník.

Translace obdélníku: K pozici podlahy přičteme o kolik chceme posunout. Po posunutí musíme změnit i vektory p a q a normálu.

Rotace obdélníku: Buď souřadnice pozice přenásobíme rotační maticí, nebo vypočítáme podle vzorečku. Po rotaci musíme změnit vektory p a q a normálu.

Pro rozlišení podlahy je aplikován **šachovnicový vzor**: Musíme určit v jaké vzdálenosti jsme. Tu určíme následovně: vypočítáme $\vec{pr}=B-Q$, kde $B=[max.x, min.y]$. Poté vypočítáme $x=\frac{p\cdot pr}{p\cdot p}$, $y=\frac{q\cdot pr}{q\cdot q}$. Poté určíme hodnotu $xx=(Math.Floor(x*100/(100/M))+Math.Floor(y*100/(100/N)))\%2$. M a N udává počet kostiček v obdélníku. Funkce Math.Floor zaokrouhluje na nejbližší celé číslo menší než zadaná hodnota.

Struktura - Kvádr

Zadáno je minimum a maximum kvádru. Z toho vypočítám zbylé body(př. c = new Bod(max.X,max.Y,min.Z)). Z těchto

bodů pak určím jednotlivé stěny kvádru(př. obdelnik1 = new Obdelnik(min, c)) .

Průsečík s kvádrem: Výpočet průsečíku se počítá pro všechny obdélníky. Tam kde vyjde nejmenší t, tak pro to t je průsečík. Výpočet průsečíku s obdélníkem, normála atd. viz Obdélník v rovině.

Struktura - Elipsoid

Elispoid je zadán koeficienty a,b,c v bodě 0,0,0.

Translace elipsoidu: Pouze přičtu o kolik se má posunout ke stávající pozici středu.

Je dána matice C. C[1,1] = b^2*c^2, C[2,2] = a^2*c^2, C[3,3] = a^2*b^2, C[4,4] = $-a^2*b^2*c^2$ všude jinde je nula. Poté vypočítáme koeficienty $a=\vec{u}^T\cdot C\cdot\vec{u}$, $b=2\cdot\vec{u}^T\cdot C\cdot(A-S)$, $c=(A-S)^T\cdot C\cdot(A-S)$, kde u je směrový vektor který má homogení souřadnici rovnou 0, A pozice odkud vycházel paprsek, S střed elipsoidu. Pokud b^2-4ac je větší než 0 pak existuje průsečík. Parametr t se vypočítá takto $t_1=(-b-\sqrt{d})/(2\cdot a)$. Parametr $t_2=(-b+\sqrt{d})/(2\cdot a)$. Ten který je menší, ten použiji.

Výpočet normály: $\vec{n}=[(P_x-X_{Sx})/a^2 \quad (P_y-X_{Sy})/b^2 \quad (P_z-X_{Sz})/c^2]$. X_S je pozice středu, a,b,c jsou osy elipsoidu a P je bod průsečíku.

Struktura - Součet objektů

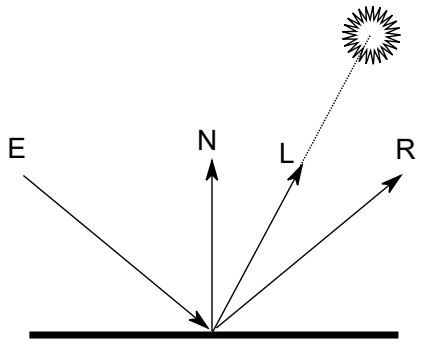
Sečtením dvou těles vznikne jedno těleso. Musíme hlídat parametry t1 a t2. Tělesa můžou být navzájem různě posunutá tedy musíme kontrolovat parametry t1 a t2 a vracet interval odkud dokud jsou.

Struktura - Rozdíl objektů

Zde zase existuje spoustu případů jak mohou tělesa být. Proto zase musíme kontrolovat parametry t1 a t2.

Vytvoření paprsků

Potřebujeme získat levý horní roh. Postup je následující: Získáme bod S = poziceKamery + směr kamery. Vypočteme $h=2\cdot\tan(fovy)$, $w=(h\cdot width)/height$. Fovy (field of view) je zadáno při vytvoření kamery. Width a height jsou rozměry okna. Bod 0,0 na plátně získáme takto



Obrázek 1: Vektory pro počítání Phongovo stínování.

$pocatek=S+(((0.5\cdot h)\cdot Camera.up)+((-0.5\cdot w)\cdot Camera.right))$. Zjištění bodu na jakém jsme, tedy pixelu se vypočítá takto $pixel=pocatek+j\cdot dx\cdot Camera.right-i\cdot(dy\cdot Camera.up)$.
$dx=w/width$, $dy=h/height$ kde *i* a *j* jsou čísla dvou for cyklu ve kterých probíhá raytracing. Směrový vektor aktuálního paprsku se vypočítá odečtením bodů
$smerový\ vektor=pixel-poziceKamery$. Tímto máme dostatek parametrů pro vytvoření paprsku se směrovým vektorem a počátkem v počátku kamery.

Phongovo stínování

Phongovo stínování počítám tak, že vytvořím paprsek do světla(na obrázku *Obrázek 1* vektor L) a zjistím normálu v průsečíku(Záleží na objektu). Difúzní složka se pak rovná s $I_D = N \cdot L$. Pro spekulární složku zjistím paprsek, který se odrazí v bodě průsečíku (R). Tento vektor vypočítám následovně: $R = (2 \cdot (E \times N)) \times N + E$. Pro vypočítání spekulární složky, vektor odrazu skalárně vynásobím se směrovým vektorem paprsku, který směřuje ke světlu $L \cdot R$. Pokud je výsledek menší než nula vynásobím ho -1. Pokud je větší než nula pak mu nastavím hodnotu nula. Poté výsledek umocním (např na 100) a výsledný koeficient je rovný spekulární složce $I_S = (L \cdot R)^n$. Výsledná barva je pak rovna $B = I_d \cdot C + I_s$, kde C je barva objektu.

Zrcadlení

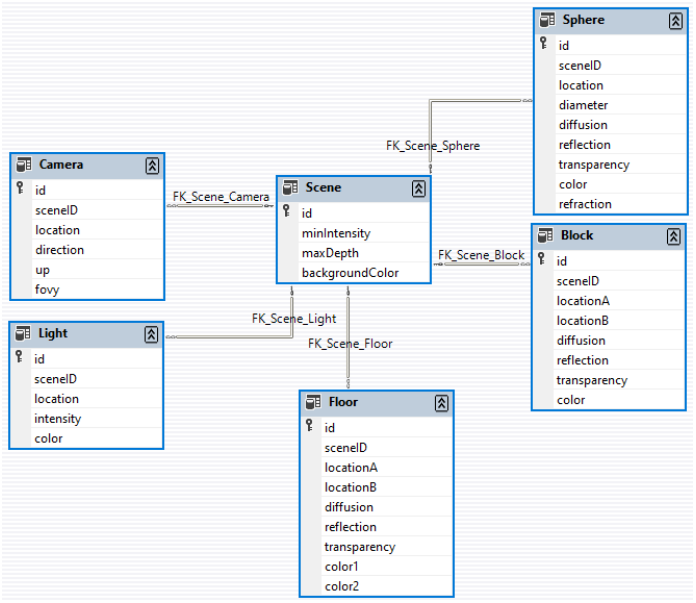
Vektor použitý pro zrcadlení používám ten co se vypočetl při výpočtu spekulární složky u Phonga. Pouze vynásobím -1. Zrcadlení se počítá pouze pokud má těleso ks nenulové a pouze pokud existuje průsešík s paprskem odražení. Z vektoru odrazu a průsečíku vytvořím paprsek. Poté rekurzivně zavolám metodu na výpočet barvy. Výsledná barva pro odražený paprsek je pak $B = kS \cdot R$, kde kS udává jak moc těleso odráží a R je barva na kterou paprsek narazil.

Průhlednost

Vektor směru se vypočítá následovně: Vypočítá se jako $\cos(\theta_1) = -u \cdot N$. U $\cos(\theta_1)$ musíme změnit směr normály podle toho jestli jsme v tělese(-1*normála) nebo ne. $\cos(\theta_2) = \sqrt{1 - (\frac{n_2}{n_2})^2 \cdot (1 - \cos^2(\theta_1))}$. Pak výsledný vektor je $t = \frac{n_1}{n_2} \cdot l + \frac{n_1}{n_2} \cdot \cos(\theta_1) \cdot N - \cos(\theta_2) \cdot N$. Poté zjistím zda existuje průsečík s paprskem se směrovým vektorem *t* s počátkem v průsečíku. Pokud existuje průsečík a těleso je průhledné pak rekurzivně zavolám metodu pro zjištění barvy. Výsledná barva pro paprsek, který prošel objektem je pak $B = kT \cdot Q$, kde *kT* udává jak moc je těleso průhledné a *Q* je barva na kterou paprsek narazil.

5. Databáze programu

Aby byl program schopen ukládat přednastavené scény a poté je případně i nahrát, byl program doplněn o databázi typu *Dataset*. Tato databáze umožňuje procházení dat, kde je možné použít filtry pro jednoduché nalezení požadovaných dat. V programu je definována databáze pomocí designeru, který umožňuje snadné vytvoření požadované struktury. Struktura je navíc generována a vytváří možnost programově procházet danou databázi. Výsledná databáze je vidět na obrázku *Obrázek 2*.



Obrázek 2: Dataset databáze

Prozatím databáze podporuje pouze objekty scény, kamery, světla, podlahy, koulí a kvádrů. Ostatní popisované struktury nejsou v databázi implementovány. Program je schopný načíst soubor pojmenovaný *database.xml* a uložený ve složce s programem. Pro práci s databází slouží soubory *DatabaseHandler* a *Parser*. *DatabaseHandler* slouží přímo pro práci s *Dataset* objektem. Umožňuje načtení a uložení databáze uloženého jako *.xml* soubor a vytvoření testovací scény. Zároveň obsahuje podporující třídu *DataTableHelper*, která k uložení a načtení dále podporuje zobrazení celé tabulky či datasetu.

Parser je třída, která je využita pro vytvoření objektu scény, se kterým dále pracuje renderer. Objekt je vytvořen na základě datasetu, který je mu předán, a prochází jednotlivé tabulky a přidává nalezené objekty do scény.

6. Render scény

Rendering scény je podle uvedeného algoritmu výše. Pro zobrazení je pro každý pixel vypočtena výsledná vystínovaná barva. Po výpočtu všech pixelů je výsledek zobrazen v okně, které podporuje výsledek uložit ve formátu *.PNG* na disk.

Uživatelské rozhraní přidalo podmínku do renderu scény takovou, že je možno rendering kdykoliv zrušit.

7. Uživatelské rozhraní

Uživatelské rozhraní(anglicky User Interface(UI)) umožňuje podle zadání vytvářet objekty scény, posouvat s nimi, ukládat a načítat celou scénu do/ze souboru. Dále umožňuje nastavovat vlastnosti objektů.

Uživatelské rozhraní je vytvořeno grafickým systémem Windows Presentation Foundation. Tento systém umožňuje vytvářet funkční uživatelské rozhraní, které lze snadno skládat v editoru. Tento systém se navíc snadno integruje do stávajícího c# projektu.

Uživatelské rozhraní je rozděleno do hlavního okna a nastavovacího okna. Hlavní okno obsahuje veškerou logiku svých prvků, kromě prvků které byly převzaty z komunity.

DragCanvas

Třída vylepšující stávající třídu *Canvas*. Tato třída, převzata z [4], vylepšuje rodiče o možnost hýbání prvků, které jsou v ní umístěny. Toho jsem využil na pohyb tvarů obdélníku a kružnice.

DropDownButton

Třída převzata z [5]. Byla využita pro lepší vzhled seznamu objektů ve scéně.

8. Závěr

Zadání bylo splněno, avšak někde v kódu je chyba při renderování. Tato chyba, nejspíše způsobená spatným výpočtem vektorů způsobuje špatný odraz v některých úhlech.

9. Programový deník

Programový deník není kompletní, jelikož jsem ho začal dělat až při druhé fázi semestrální práce. Také je v něm jasně vidět podcenění náročnosti vytvoření UI.

Datum	Aktivita	Délka[hod]
24.5	Vytvoření projektu UI, návrh	4
26.5	Základní dělení UI	4
28.5	Přidání ovládacích prvků, funkcionalita	7
29.5	Funkcionalita UI, dokumentace	12
		27

10. Uživatelský manuál

Pro spuštení přeložené verze je potřeba, aby na počítači byl nainstalován .NET. Program je možno spustit poklikáním na ikonu UI.exe. Přiložený soubor *database.xml* slouží jako databáze scény. Program je nastavený tak, aby tento soubor načetl a zobrazil načtenou scénu. Vhodnou úpravou je možno pozměnit scénu. Prozatím program podporuje načtení pouze objektů: *Scene, Camera, Light, Sphere, Block, Floor*. Veškerý předpis je možno získat z dokumentace nebo prostudováním souboru *database.xml,* ve kterém je uveden i popis databáze.

Na obrázcích Figure 1 a Figure 2 je popsané uživatelské rozhraní. Tlačítko *Front* přepíná pohyb v ose XY, tlačítko *Top* přepíná do osy XZ a tlačítko *Right* přepíná do osy ZY. Všechny osy jsou pouze kladné. Vlastnosti barvy (*Color(R,G,B*)) mohou nabývat hodnot od 0 do 255. Vlastnosti objektu *diffusion, reflection a transparency* mohou nabývat hodnoty od 0 do 1(0-100%).

Renderování scény lze spustit tlačítkem Render a kdykoliv zrušit stisknutím tlačítka cancel.

Figure 1: Popis UI 1.

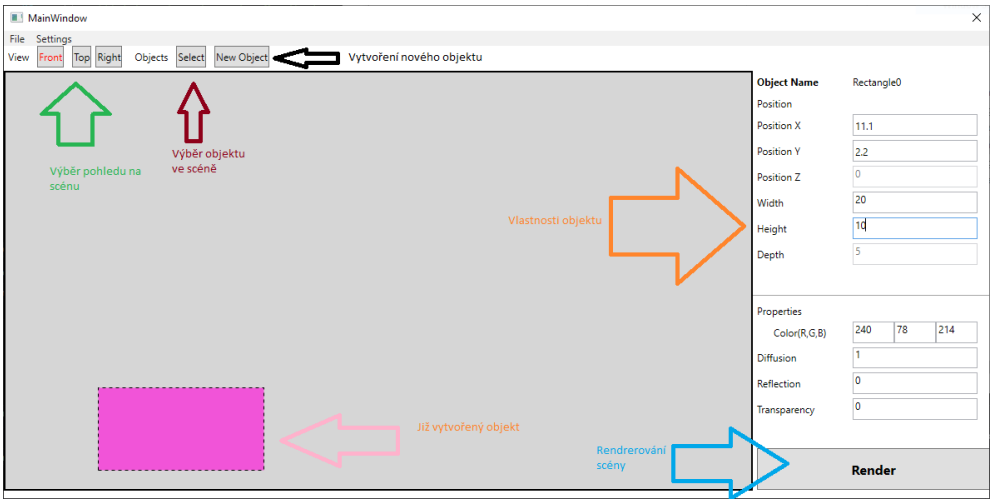


Figure 2: Popis UI2

10. Odkazy

[1] Sledování paprsku.[online]. [cit. 2020-04-13] Dostupné z: **http://cs.wikipedia.org/wiki/Sledov%C3%A1n%C3%AD_paprsku**

[2] Phongovo stínování.[online]. [cit. 2020-04-13] Dostupné z: http://cs.wikipedia.org/wiki/Phongovo_st%C3%ADnov%C3%A1n%C3%AD

[3] Ray tracing (graphics).[online]. [cit. 2020-04-13] Dostupné z: http://en.wikipedia.org/wiki/Ray_tracing_(graphics)#Example

[4] Dragging Elements in a Canvas.[online]. [cit. 2020-05-29] Dostupné z: www.codeproject.com/Articles/387977/Dragging-Elements-in-a-Canvas-2

[5] DropDownButtons in WPF.[online]. [cit. 2020-05-29] Dostupné z: http://andyonwpf.blogspot.com/2006/10/dropdownbuttons-in-wpf.html