nedeljainformatike v8.0

Renderovanje: Od 3D majmuna do slike na ekranu

Pavle Pađin

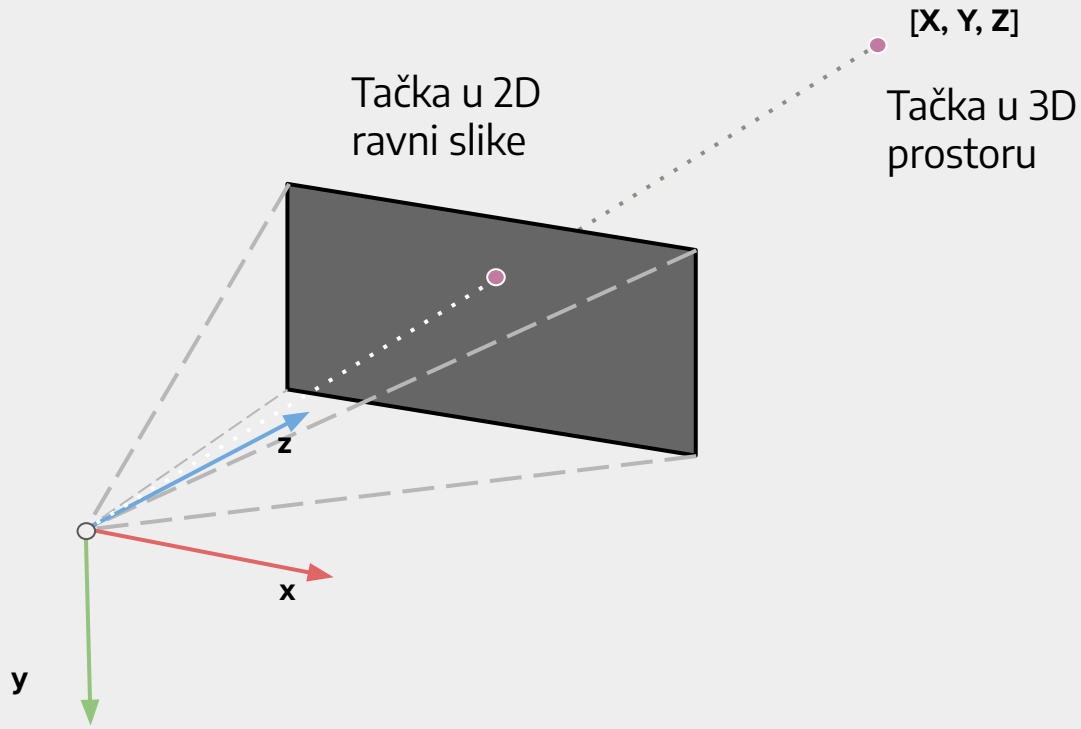
Matematička gimnazija

17. 04. 2022.

Agenda

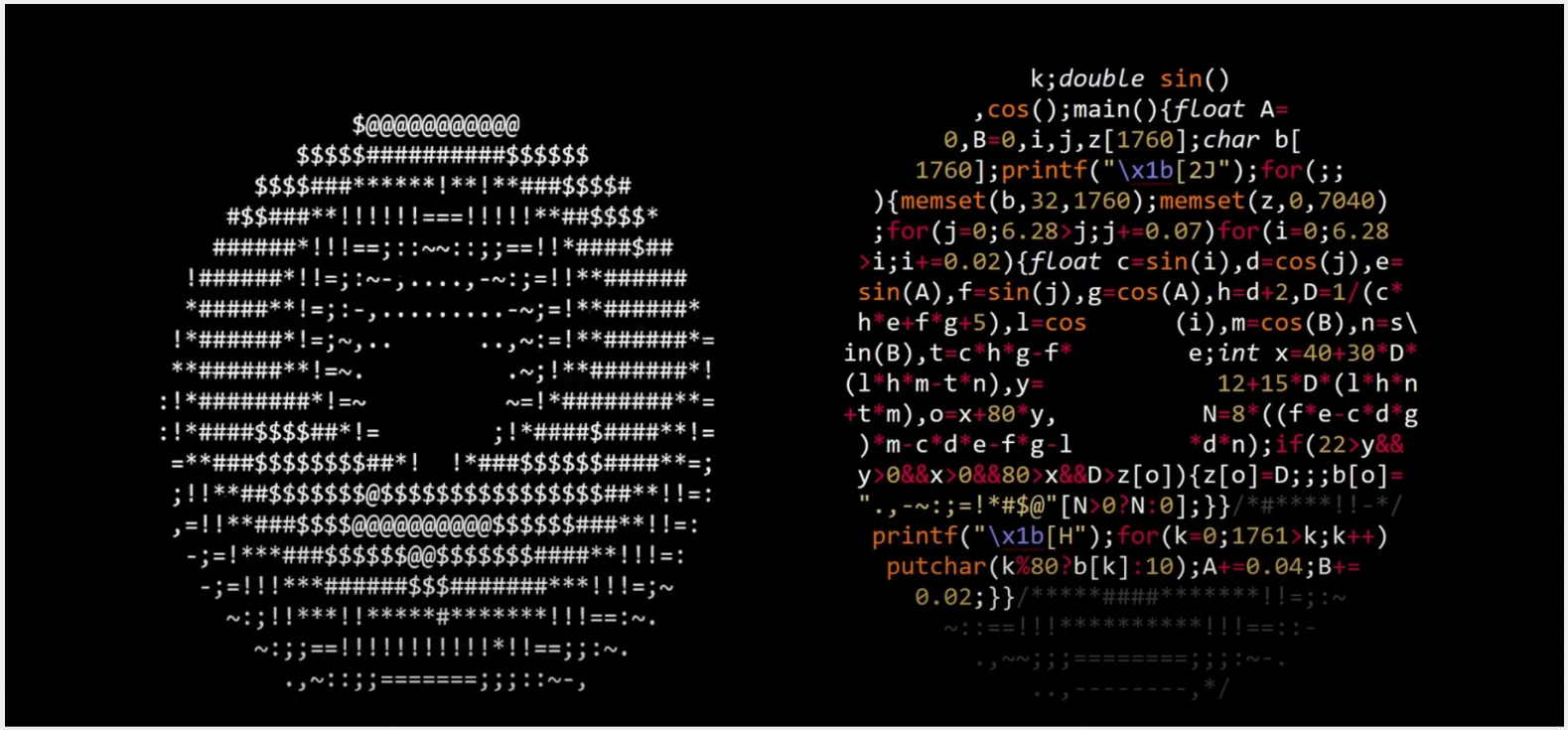
1. Šta je renderovanje
2. Simple pinhole model kamere
3. Rasterizacija

Šta je renderovanje?



Proces kojim dolazimo od 3D objekta do 2D slike na ekranu

Primer: Rotirajuća krofna



Izvor: https://www.youtube.com/watch?v=DEqXNfs_HhY&ab_channel=LexFridman

Kako od 3D koordinata do 2D slike?

Uopšteni postupak renderovanja može se podeliti na dve glavne celine:

1. **Pravila igre:**
   Definišemo model kamere - Matematički model kojim modelujemo vezu između 3D tačaka i piksela na slici

2. **Naš igrač:**
   Konstruišemo algoritam koji će “pametno” da projektuje tačke u 2D ravan slike (*rasterizacija* ili *ray tracing*)

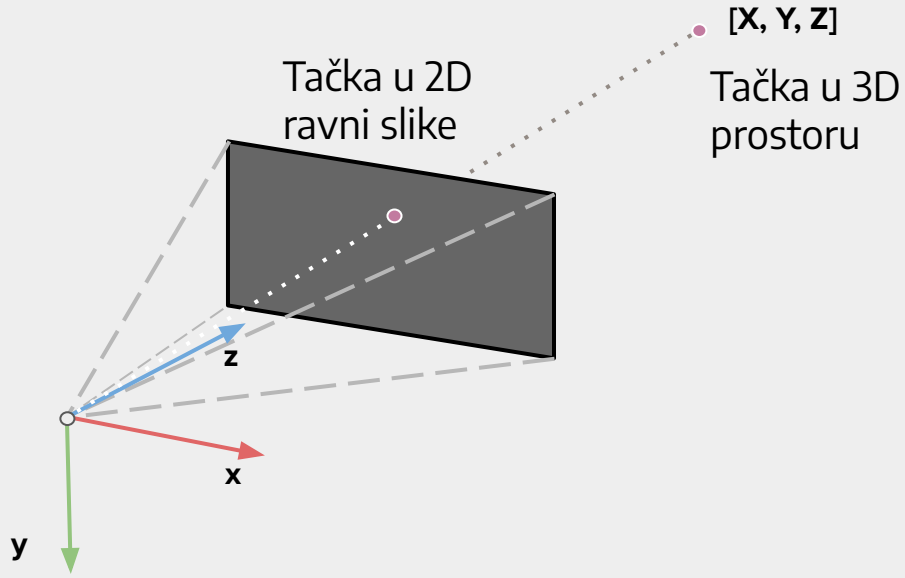
Agenda

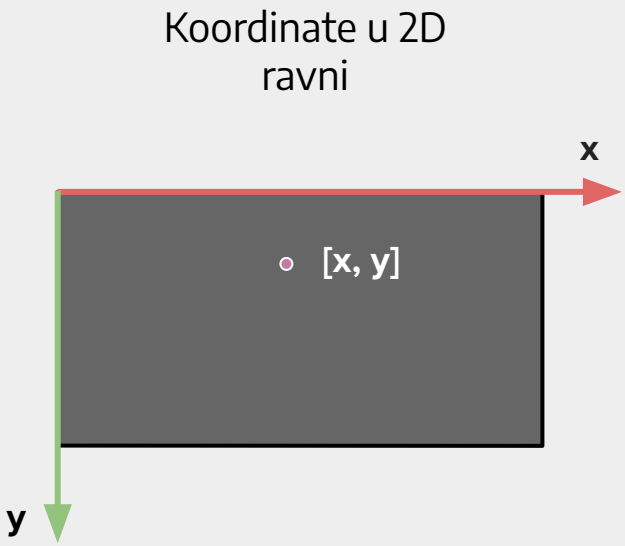
1. Šta je renderovanje
2. *Simple pinhole* model kamere
3. Rasterizacija

Simple pinhole model kamere

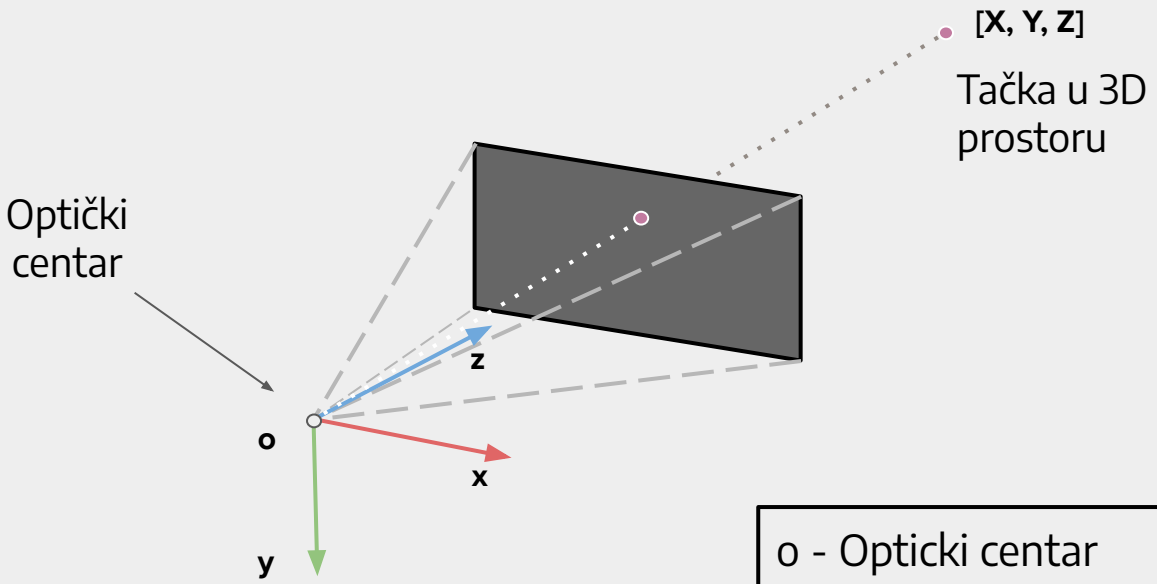
Zašto nam je potreban matematički model?

On je suštinski pojednostavljenje stvarnih relacija između 3D objekata i 2D slika. Omogućava lakše izračunavanje





Delovi *simple pinhole* modela



o - Opticki centar

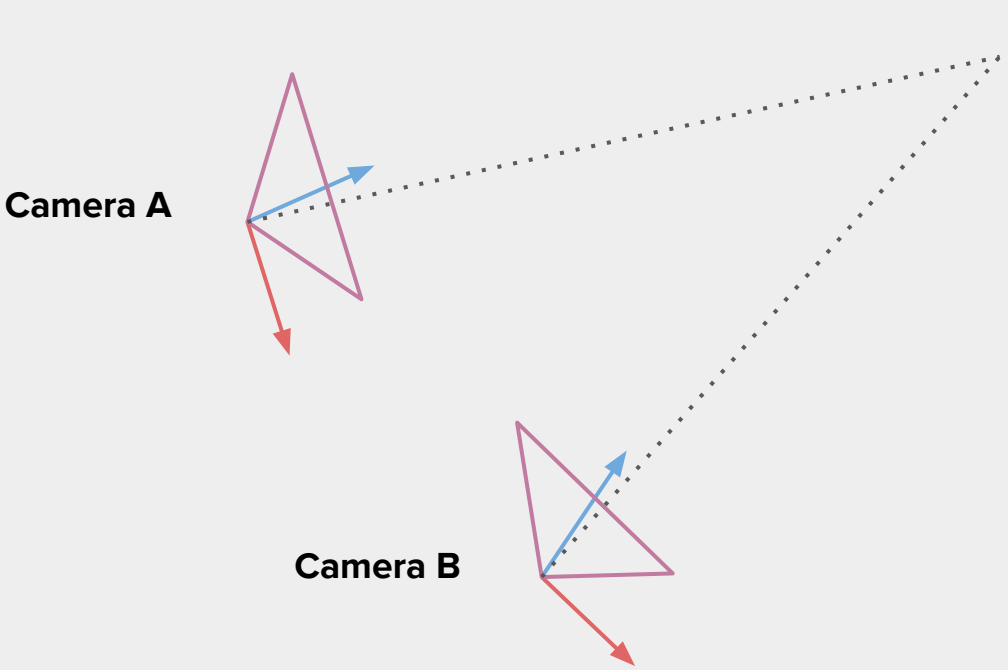
X, Y, Z - koordinate u 3D prostoru

x, y - koordinate tačke u 2D ravni

z - dubina tačke

f - žižna daljina (fokus) - rastojanje optičkog centra od ravni slike

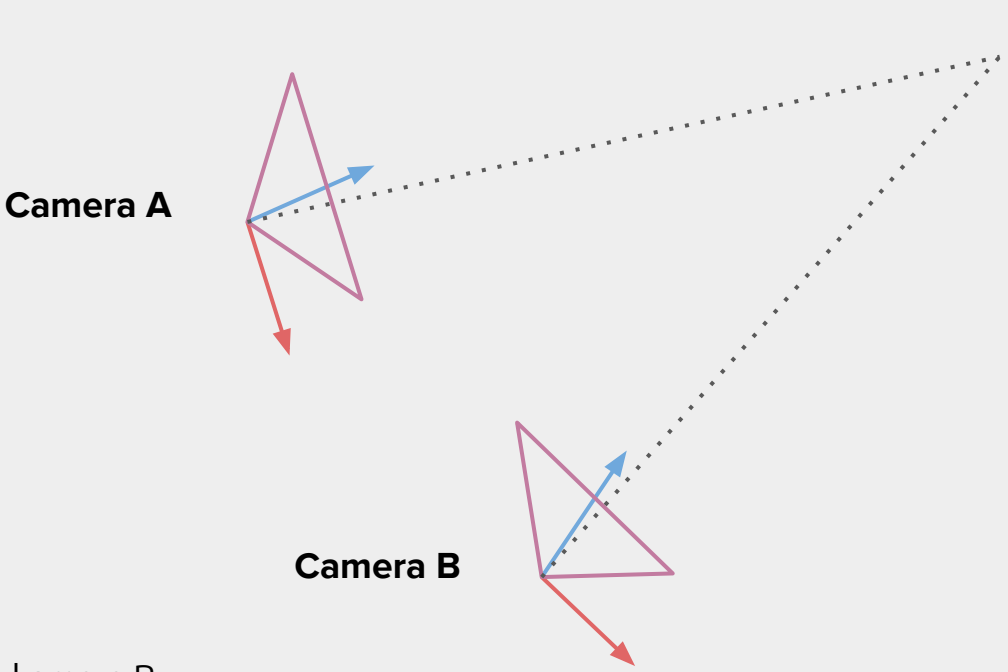
1. Prebacivanje u koordinatni sistem kamere



Transformacija krutog tela
(rotacija + translacija)

$$Pba = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

1. Prebacivanje u koordinatni sistem kamere



Transformacija krutog tela
(rotacija + translacija)

$$Pba = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix}$$

Tačka u k.s. kamere B

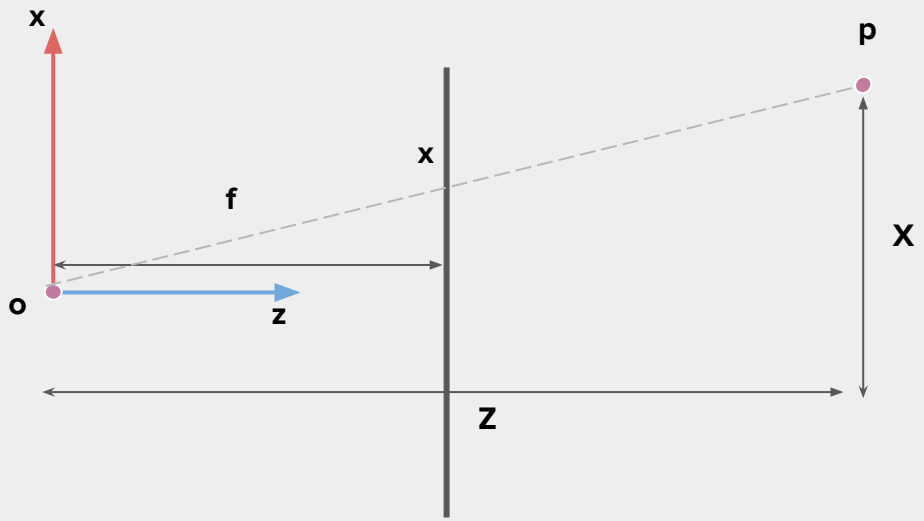
$$\begin{pmatrix} Xb \\ Yb \\ Zb \\ 1 \end{pmatrix} = Pba \begin{pmatrix} Xa \\ Ya \\ Za \\ 1 \end{pmatrix}$$

Tačka u k.s. kamere A

2. Projektovanje tačke u ravan slike

Iz sličnosti trouglova, možemo da dobijemo ove relacije:

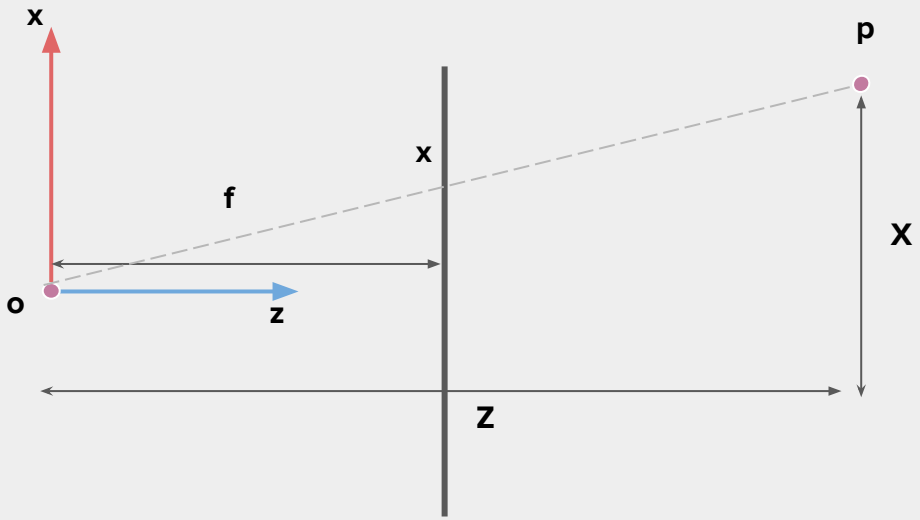
$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X \\ Y \end{pmatrix}, \quad z = Z$$



2. Projektovanje tačke u ravan slike

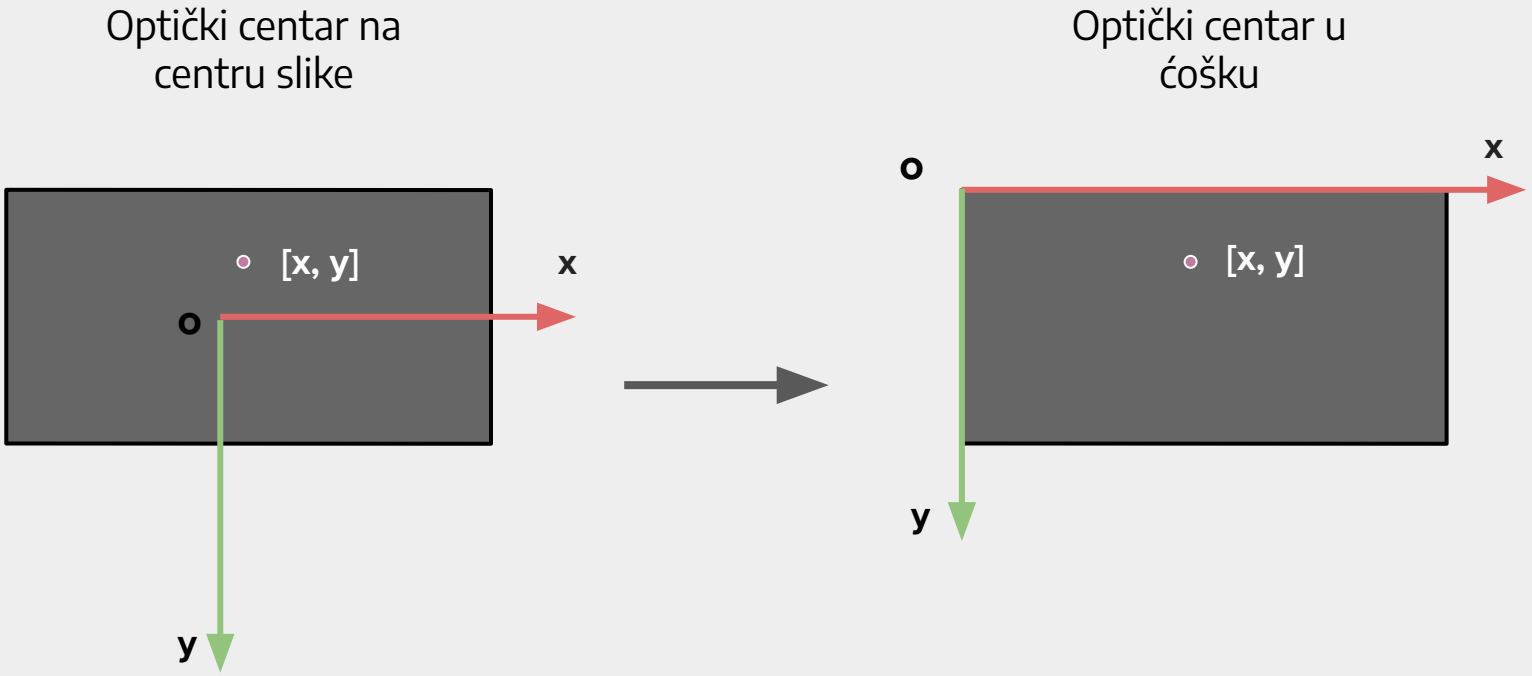
Kada skupimo sve u matričnu formu:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f' & 0 & 0 \\ 0 & f' & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$, gde je $f' = \frac{f}{Z}$



2. Projektovanje tačke u ravan slike

Želimo da pomerimo naš koordinatni početak u levi gornji ćošak (kao što su koordinate piksela na slici)



2. Projektovanje tačke u ravan slike

Želimo da pomerimo naš koordinatni početak u levi gornji ćošak (kao što su koordinate piksela na slici)

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f' & 0 & 0 \\ 0 & f' & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \longrightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f' & 0 & cx \\ 0 & f' & cy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

cx - pola širine slike
cy - pola visine slike

Celokupan model kamere

Kada spojimo sve delove prethodno objašnjene, dobijamo ovakvu relaciju između 3D koordinata tačaka i 2D koordinata piksela

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} f' & 0 & cx \\ 0 & f' & cy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Matrica kalibracije

Matrica projekcije
Od 4 dim. do 3 dim.

Od proizvoljnog k.s. do
k.s. naše kamere

Agenda

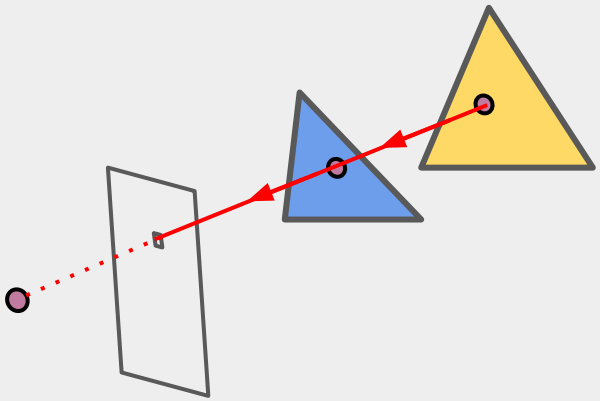
1. Šta je renderovanje
2. Simple pinhole model kamere
3. Rasterizacija

Rasterizacija vs Ray Tracing

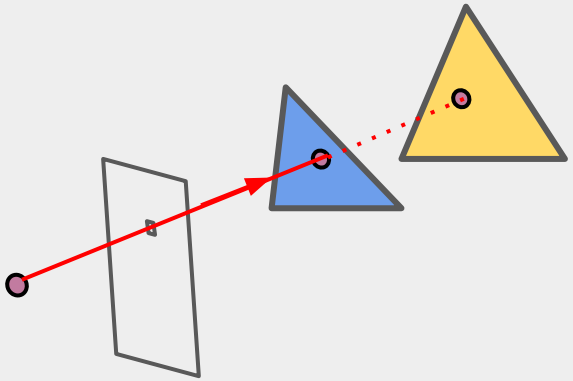
Rešavaju isti problem, ali imaju suprotne pristupe

Rasterizacija - Projektujemo tačke na ravan slike i gledamo gde padnu

Ray Tracing - Puštamo “zrak” kroz svaki piksel slike i gledamo koja 3D tačka je najbliža zraku



Rasterizacija

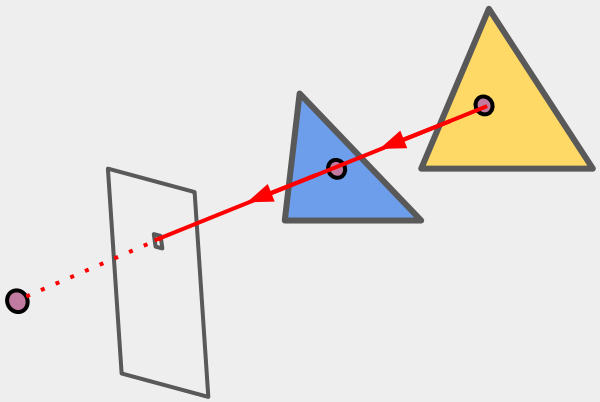


Ray tracing

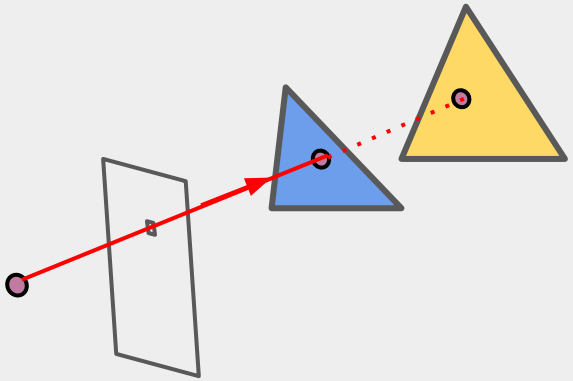
Rasterizacija vs Ray Tracing

Prednosti rasterizacije: Manja kompleksnost, manji hardverski zahtevi, zgodna za real-time aplikacije

Prednosti raytracing-a: Veća preciznost, bolje radi sa kompleksnijom geometrijom, sve češće u video igricama



Rasterizacija



Ray tracing

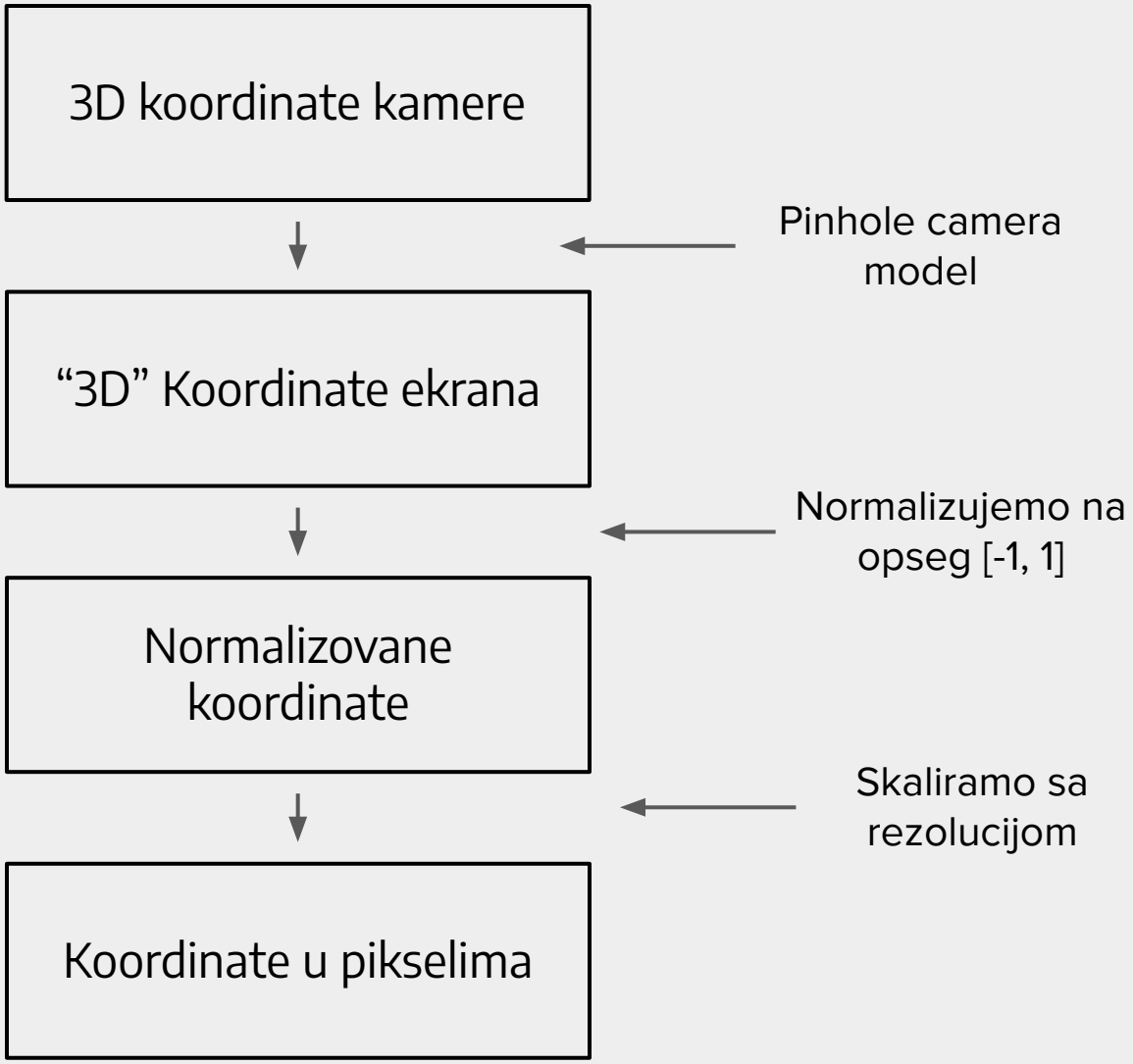
Rasterizacija - Algoritam

Na ulazu dobijamo 3D objekat koji je predstavljen pomoću velikog broja trouglića (mogu i drugi geometrijski oblici)

Dve faze algoritma:

1. **Projekcija:** Svaki trougao u 3D prostoru projektujemo na 2D ravan slike (u prostor piksela)
2. **Rasterizacija** (u užem smislu) - Odredimo koje piksele na slici treba obojiti bojama trougla

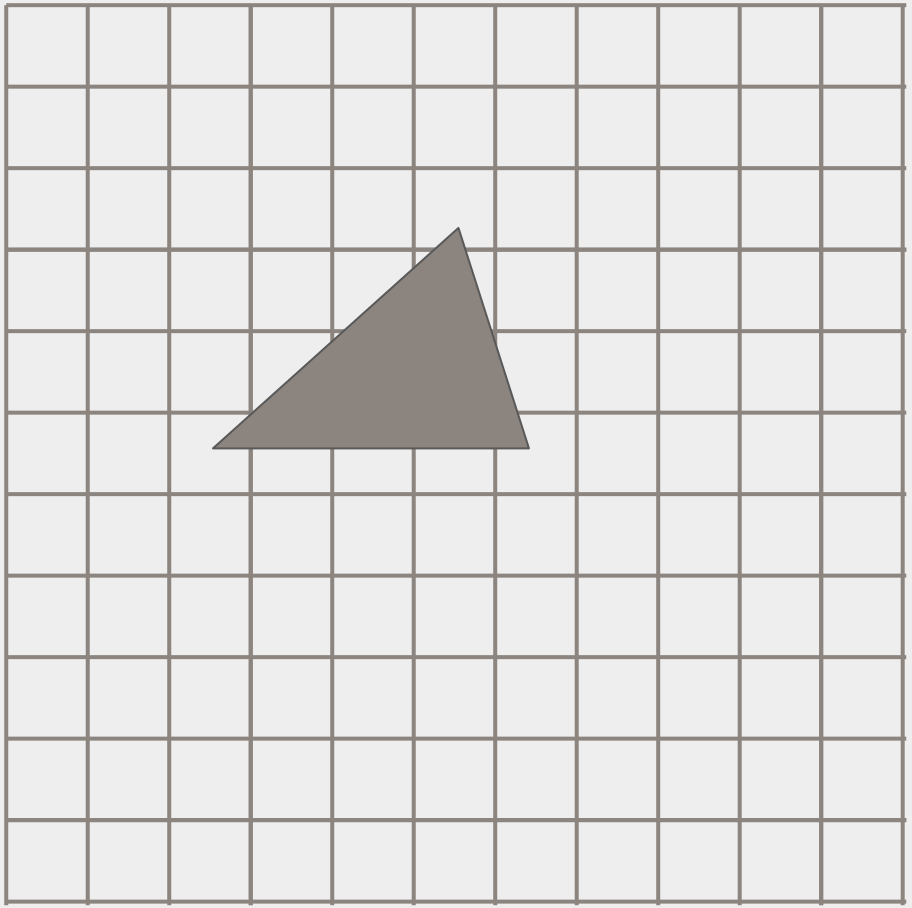
Faza projekcije



Faza rasterizacije

U ovoj fazi odgovaramo na sledeće pitanje:

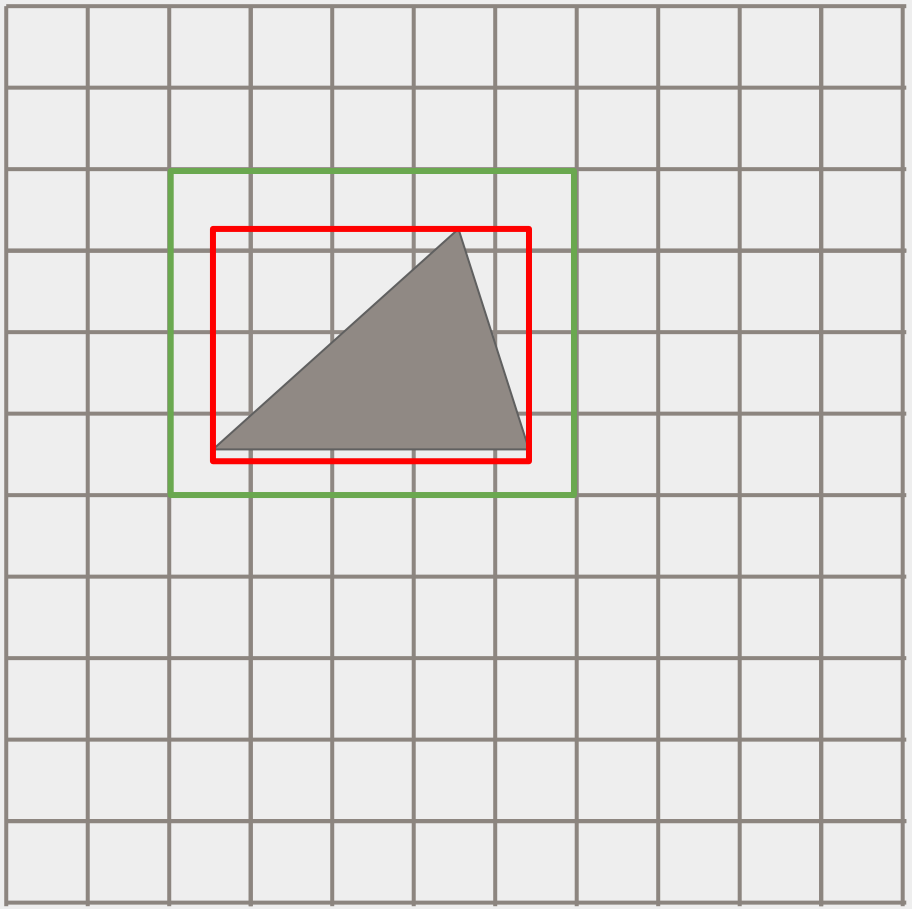
Projektovali smo trougao, ali koje piksele treba da obojimo?



Faza rasterizacije

Hoćemo da obojimo samo piksele čiji se centar nalazi u unutrašnjosti trougla.

Za početak, jedini pikseli koje treba da razmatramo su oni u pravougaonom omotaču trougla



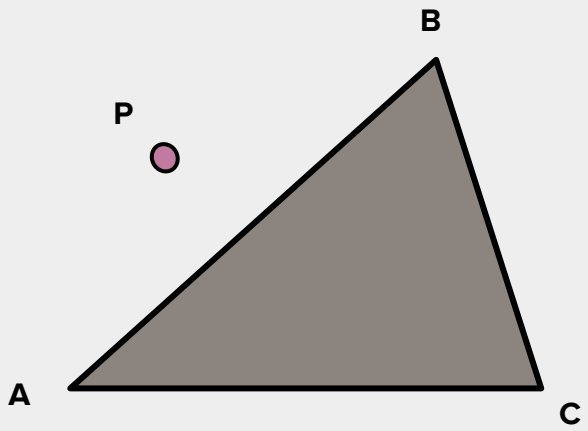
Pravougaoni omotač

Pikseli koji dolaze u obzir

Kako znamo da li se centar piksela nalazi u trouglu?

Koristićemo neku funkciju koja nam govori sa koje strane prave se nalazi tačka.

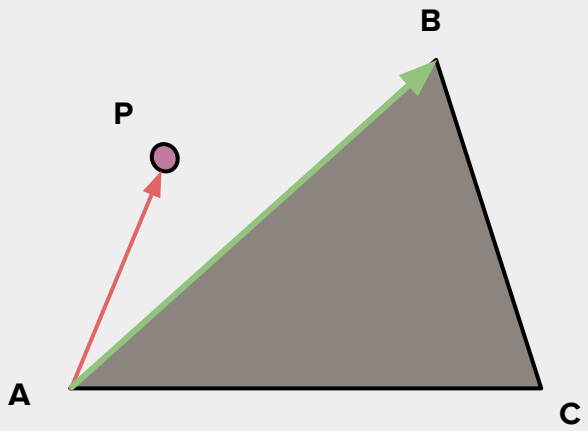
Ako za svaku stranicu trougla dobijemo da se tačka nalazi sa unutrašnje strane, to znači da je u trouglu



Vektorski proizvod

Smer vektorskog proizvoda vektora **AP** i **AB** je pozitivan ako je tačka P sa desne strane duži AB, u suprotnom negativan

Ako je vektorski proizvod pozitivan za sve stranice trougla, onda se tačka P **nalazi u trouglu**



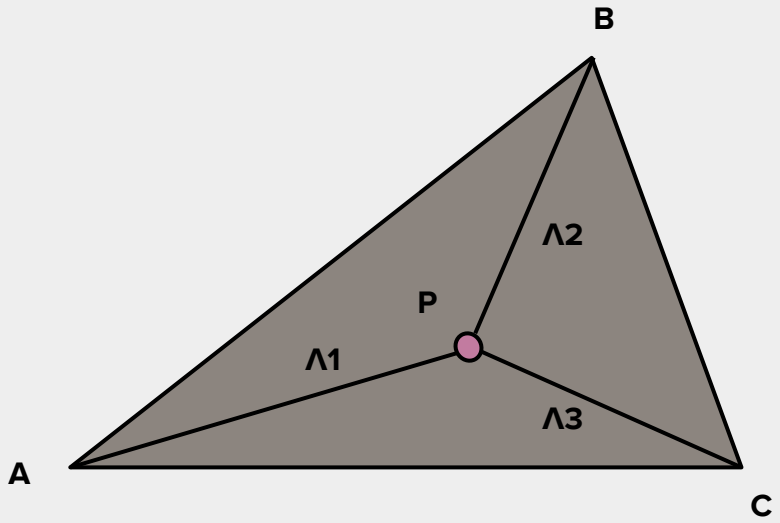
AP x AB > 0 P sa desne strane AB
AP x AB < 0 P sa leve strane AB

Baricentrične koordinate

Govore nam kako da interpoliramo tačku u trouglu pomoću njenih temena

Tačku P dobijamo kao:

$$P = \lambda_1 A + \lambda_2 B + \lambda_3 C$$



Gde su:

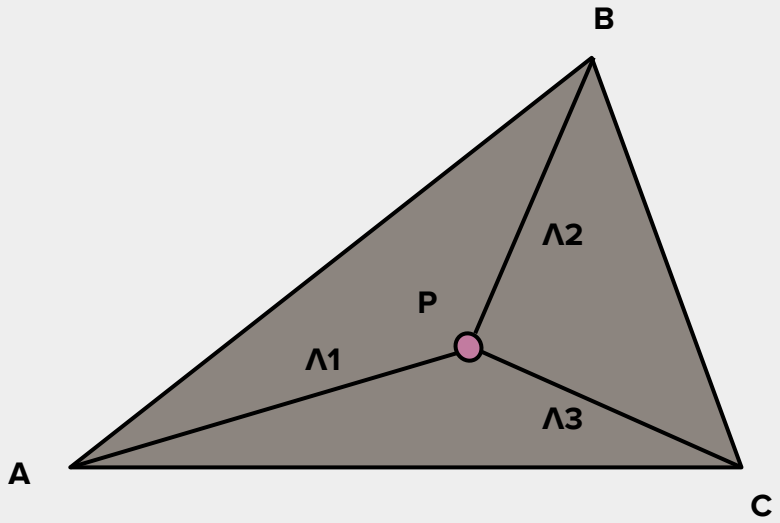
$$\lambda_1 = \frac{P_{PBC}}{P_{ABC}} \quad \lambda_2 = \frac{P_{PAC}}{P_{ABC}} \quad \lambda_3 = \frac{P_{PAB}}{P_{ABC}}$$

Ovim postupkom dobijamo i boju i z koordinatu tačke P!

Baricentrične koordinate i vektorski proizvod

Moguće je iskoristiti prethodno dobijeni rezultat

Intenzitet vektorskog proizvoda dva vektora nam daje površinu paralelograma koji oni opisuju



Dakle, imamo sledeću vezu:

$$P_{PBC} = \frac{1}{2}|BP \times BC|$$

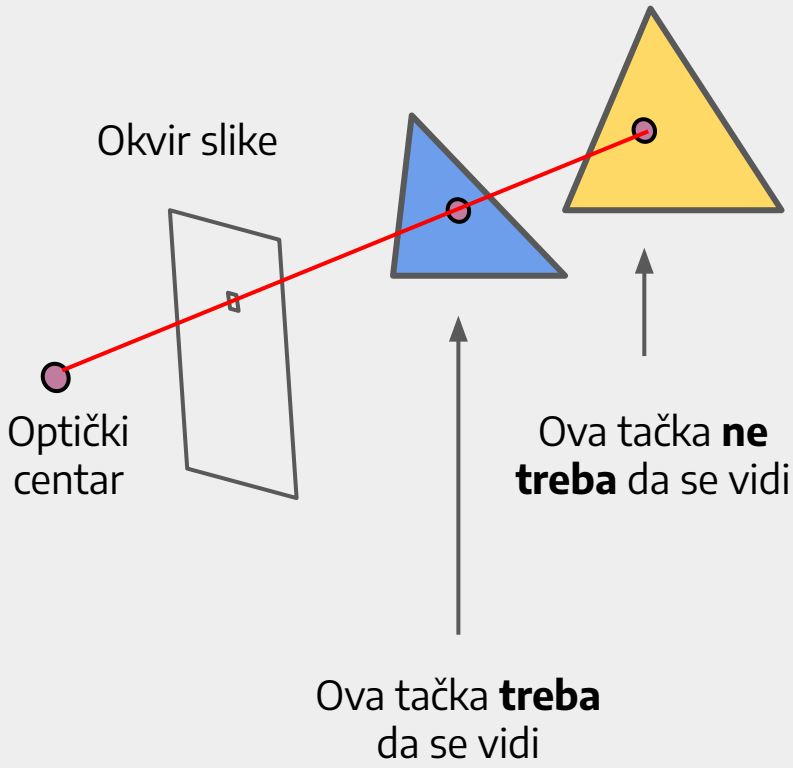
Ovo smo već izračunali

Frame i depth buffer

Pored boja piksela (frame buffer) pamtimo i dubine piksela (depth buffer, z koordinata)

Pikselu dodeljujemo boju samo ako je ta tačka na trouglu bliža od prethodno najbliže tačke (što se čuva u depth bufferu)

Tako rešavamo problem vizibilnosti -Treba da se prikazuju samo objekti koji nisu zaklonjeni nekim bližim objektom (sa manjom z koordinatom)



Rasterizacija - Pseudo kod

```
for (t in triangles){
        a, b, c = project(t.A, t.B, t.C);  // projection stage

        for (pixel in pixels){
                p = triangleAt(a,b,c, pixel);  //point on triangle at pixel coords

                if (pixel.inTriangle(a, b, c) && p.z < depth_buffer[pixel]){
                        frame_buffer[pixel] = p.color;
                        depth_buffer[pixel] = p.z;
                }
        }
}
```

Hvala na pažnji!

Pitanja?