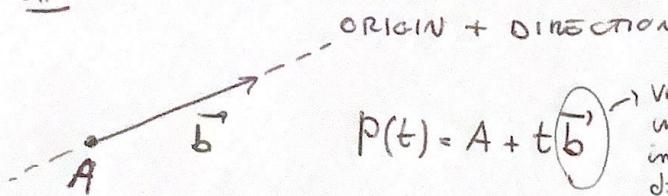


RAY TRACING

1. Ray Class

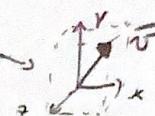


sarebbe struttura pesante solo nelle
shaders dato che usiamo sempre
d'attuale allo shader

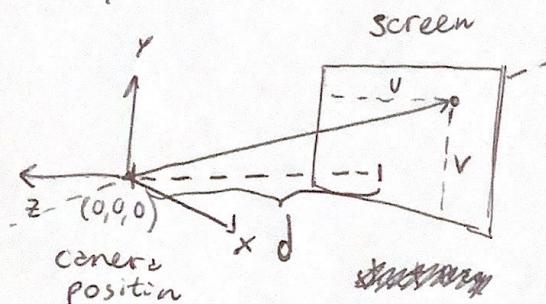
start Ray

Vec3 origin

Vec3 direction



1.2 Ray in Scene



inviamo dei raggi dalla posizione della
camera per tutti i pixel della view per
andare a controllare il colore associato ad
ogni pixel

1. inviare il ray
2. trovare cosa interseca il ray
3. trova il colore associato al ray

per ogni punto dello screen dell'ocio invia un rayo partendo dalla posizione
della camera (eye)

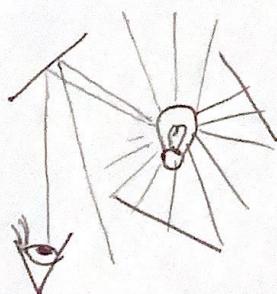
Ci servono

- pos camera
- distanza della view
- UV coordinates
- dimensioni della view

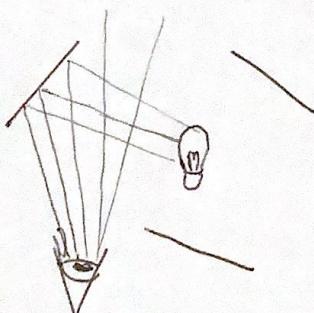
devo avere persino tutte queste
info alle shaders

(QUI PROBABILMENTE DOVREMO FARCI
QUALCHE CALCOLO PER CAPIRE BENE
TUTTE LE INFO ASSOCIADE ALLA PROJECTION
VIEW)

Perché ci servono i raggi che partono dall'occhio?



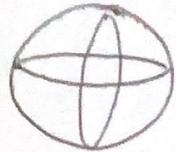
se faccio partire i raggi dalla
fonte, i raggi che raggiungono
nostro occhio vengono "speciati"



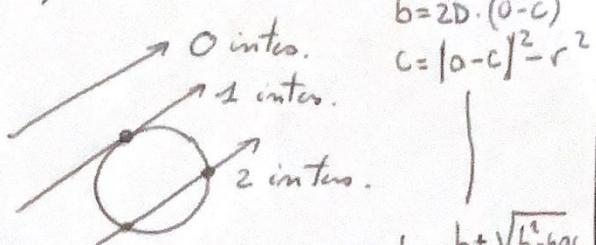
metto però solo dal nostro occhio
dato che è solo quello che possiamo
osservare

2 Sfere

2.1



abbiamo ora delle sfere sulle scree e vogliamo sapere se i nostri raggi vedono queste sfere



$$\begin{cases} P(t) = A + t \vec{b} & \text{eq. ray} \\ x^2 + y^2 + z^2 = r^2 & \text{eq. sfera} \end{cases}$$

$$|P(t) - C|^2 = r^2 \text{ equazione analitica intese due raggi sfera} \Rightarrow t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

2.2 poniamo quindi: aggiungere una funzione allo shader

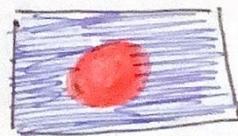
hit-sphere (vec3 center, double radius, Ray ray) { }

Verifica se il raggio interseca la sfera

(ogni ANTE LA SFERA LA CREANO NEL MAIN E PASSANO LE INFO LEGATE A CENTRO E RADICO CON IN/CUT)

PENSO CHE CREANO TANTE SFERE E LE PASSANO CON UN ARRAY)

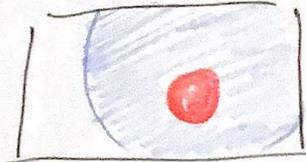
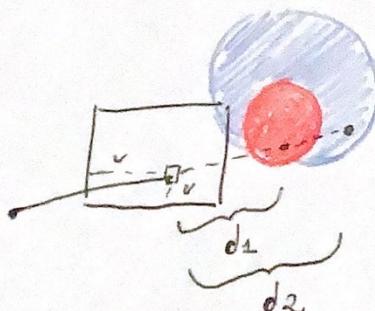
quindi poniamo due cl
hit-sphere ~~true~~ return color. RED
return color. BLUE



2.3 se ora aggiungo più sfere come col mio colore

```
struct Sphere {
    vec3 centre
    float radius
    vec4 color
}
```

le creano tutte nel main e le poniamo nello shader, ma poniamo vedere per ogni raggio quale sfera viene colpita e tiene conto delle distanze più brevi e rende questo il colore della sfera corrispondente

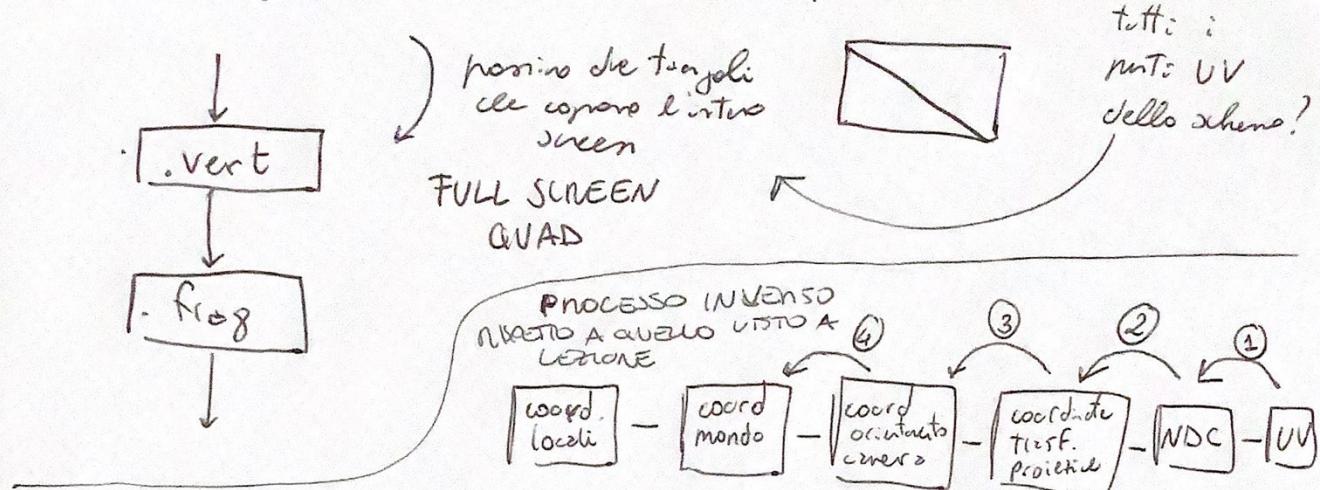


raggio colpisce entro, la $d_1 < d_2$
quindi renderizza rosso

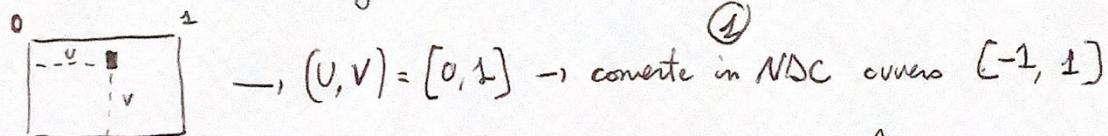
DUE CONCETTI SU VULKAN

Tutto quello visto ad ora si fa solo frag shader, però il punto è che Vulkan necessita di un vert shader quindi che si fa?

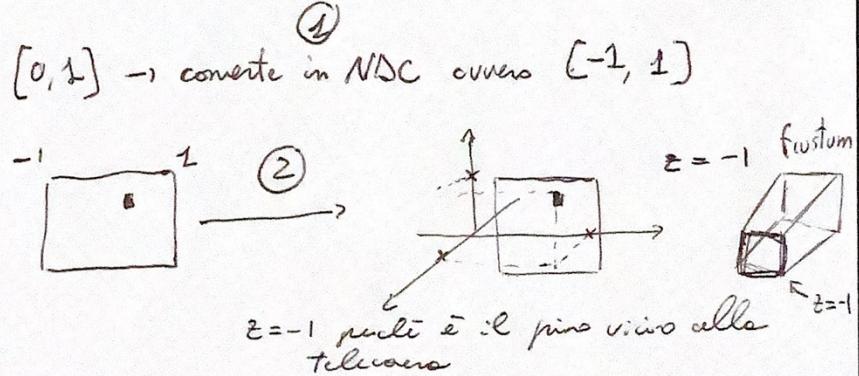
vert ha bisogno di un mesh ma il mesh non so se lo devo ottenere



Cosa fa le shader frag?



crea punto nello spazio

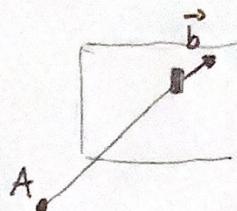


trasformare coordinate rispetto alla vista

* inverso della Proj Matrix e le nonellazioni $\frac{x}{w} \frac{y}{w} \frac{z}{w}$

④ coordinate mondo

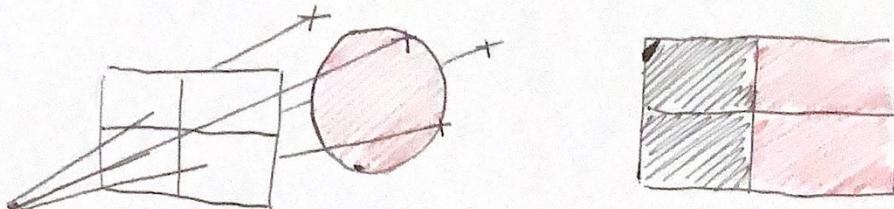
* inverso view Matrix e nonellazioni per ottenere un vettore unitario



FINE V1 PROGETTO

3. Aliasing

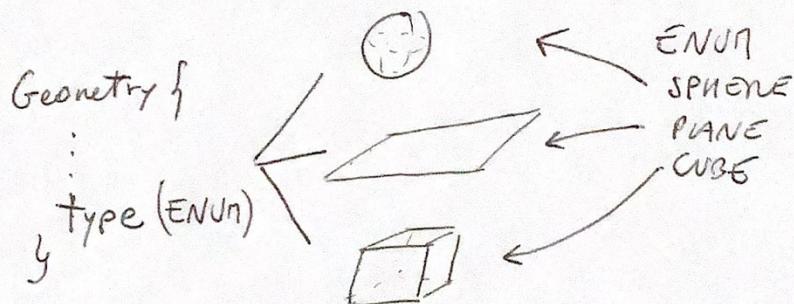
nel documento si introduce anche il problema legato all'point sampling, ovvero stiamo campionando il nostro schermo rispetto al n° di raggi che generiamo e quindi la risoluzione si allarga al n° di raggi.



Mai però alias un raggio per un pixel della screen quindi non abbiamo problemi. Se vogliamo una risoluzione migliore, allora dovranno integrare i risultati: attesa a dare due le luci per ottenere una risoluzione migliore.

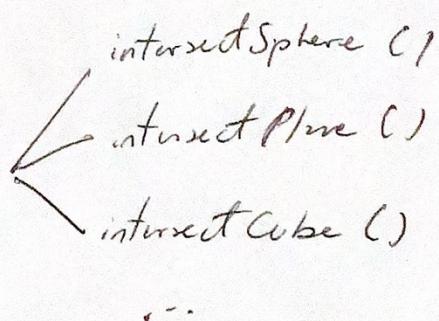
STEP SUCCESSIVO

rendere è tutto genico e non solo con Sphere



uso un array di Geometry[n]

verifico se il raggio interseca le geometrie e
verifico il tipo delle geometrie e se ho
il metodo di calcolo dell'intersezione
(geometria interseca un raggio con
un piano o con un cubo sono due
caso, quindi saranno gestiti in modi
differenti)



intersezione piano

$$\vec{N} \cdot (\vec{P} - \vec{P}_0) = 0 \text{ formula piano} \quad \text{raggio } \vec{R}(t) = \vec{O} + \vec{t} \vec{D}$$

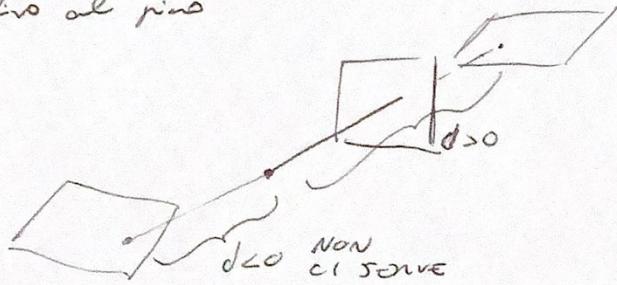
1) verifica che non sia // al piano se \vec{N} e \vec{t} siano //
 $\text{dot}(\vec{N}, \vec{t}) ? = 0$

2) sostituisce all'equazione del piano l'origine del raggio O

$$\vec{N} \cdot (\vec{P} - \vec{O}) = x$$

ottiene un regolo che va da O a P

3) verifica che sia >0 la distanza lungo la direzione del raggio
fra al piano

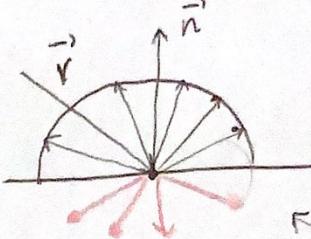


4. Diffuse Materials materiali con superfici non perfettamente lisce

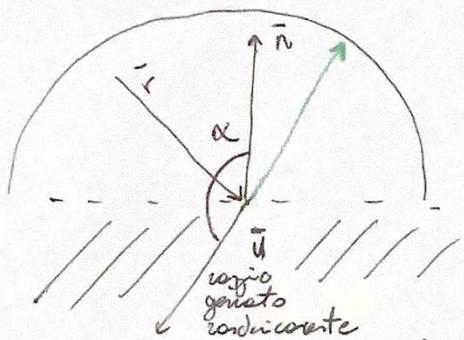
Diffuse objects non emettono luce, ne bensì riflettono e recuperano il colore degli oggetti circostanti.

Quello che fanno è riflettere i raggi da un'area.

Il più semplice esempio di diffuse reticolare è quello di una rete con griglie parallele in più direzioni



Ora cosa succede se raggi luminosi entrano dall'interno dell'oggetto, dato che i raggi non si intersecano?



surface algoritmo:

$$\text{dot}(\vec{n}, \vec{o})$$

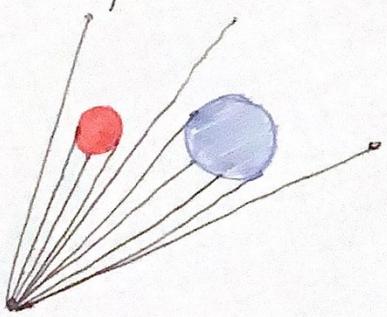
se $\text{dot} < 0$ allora l'angolo α tra \vec{n} e \vec{o} è $> 90^\circ$, e quindi significa che il raggio è uno di quelli rossi. Si pone

→ in questo caso invertiamo il raggio e ottieniamo quello conetto

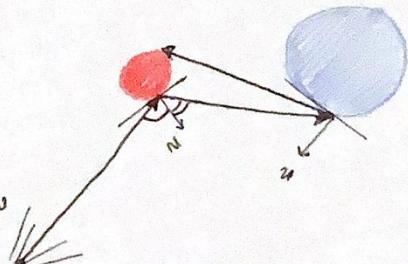
quello che bisogna fare ora è un processo ricorsivo.

Fino ad ora infatti ci sono limitati a incassare i raggi dalla sfera e se battiamo qualcosa perdiamo il colore dell'oggetto colpito.

Ora invece gli volta le calciate in qualcosa vado a rigenerare un nuovo raggio con direzione casuale e rendere ricorsivo



questo
solo se
in raggio,
rispettivamente
tutti i
raggi



iniziamo per ora a fare cle

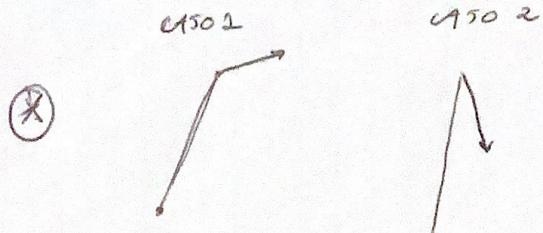
se colpisco qualcosa ritorno $\frac{1}{2}$ colore della sfera che incassa

se non colpisco nulla ritorno il colore del cielo

APPUNTI SULLA SHADER

- per generare un valore random partendo da un seed usare i PC o www.pcg-random.org

- inoltre meglio non lavorare con una distilitria uniforme per estrarre un valore. Altrettanto male di accadere è che



appare spostato la telecamera il nuovo
regione stelle sarà dunque totalmente
diversa, meglio invece usare una
distilitria Normale

con seed gerano in qualche modo dalle coordinate dei pixel che sono
sopra di esse tra loro

• Le SHADER NON SUPPORTANO LA RICORSIONE, QUINDI E' DA FARLE CON
UN CICLO

inoltre il ciclo torna solo quando il raggi non colpisce più qualcosa,
ma è troppo \rightarrow mettiamo in MAX_DEPTH = 10

• la soluzioe poi deve anche la lambertian distribution, ovvero che un
raggi rifleso da un materiale diffuso è proporzionale al $\cos(\theta)$, ovvero il
cos dell'angolo tra \vec{n} e vettore rifleso.

TRADOTTO: "E' più probabile che il raggi sia rifleso vicino alle normale,
piuttosto che lontano"

vec.direction = hit.normal + random.Direction()

1 Light

ora se segui il codice ottieni una cosa come

Ora vogliono introdurre una forte luce, di modo tale

de ottieni effetti di gesto tipo
effetto di ombre solo gli oggetti che non
sono forte di luce.



struct RayTracingMaterial {

color color → se riflette luce != nero altrimenti se fonte di luce → nero

emissionColor → binario se fonte di luce

strength

}

quindi se nel cielo dei raggi avrai 2 variabili

color = 1

incomingLight = 0

for - - - material material

incomingL += (KmissinColor * Kstrength) * color

color += material.color

di base noi moltiplichiamo il colore * il colore del rettangolo da l'Hopital con il Ray.

In più tieni conto delle luci che viene in più

Considera un solo raggio

- se incontra luce incoming += (1.0 * 1.0) * 1.0 → colore binario

- se incontra oggetto incoming += (0.0 * 1.0) + 1.0 → nero ombra

5. Progressive Rendering

Ovviamente la nostra immagine è sfocata perché invia un solo ray per pixel. Se incrementiamo il numero di raggi per pixel rispetto a migliorare la risoluzione, ma paghiamo tanti in termini di FPS, con 100 rays per pixel l'immagine è fruscata sullo schermo.

5a

Se stiamo lavorando con un rendering, per cui lo stiamo rendendo come luce, una soluzione è il Progressive rendering.

Invece di inviare 100 rays per pixel a ogni frame, se invia solo 1, ma ritraso il colore del realistico frame per approssimarsi di volta con il valore max, di modo tale da migliorare non a non deperire il tempo l'immagine.

PROBLEMA VULKAN: non esistono delle variabili globali tra shaders, quindi dovranno passare alle shaders una cosa come

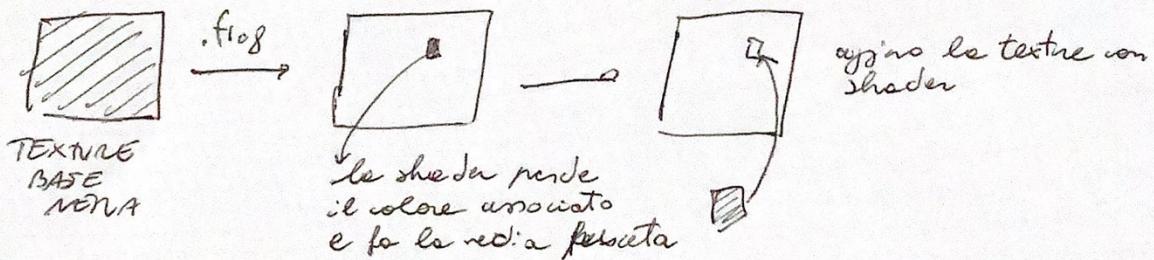
struct f

 vec4 color //align=16)

}

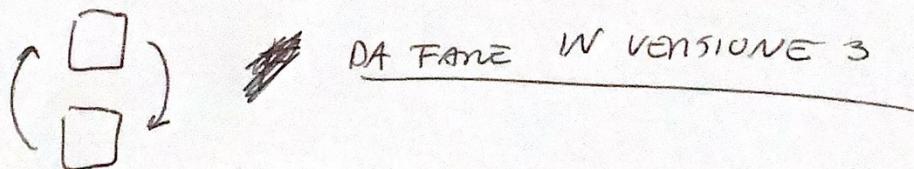
e poi creare un buffer con prendere width * height * sizeof(f) = 7,68 Mbytes troppo grande e troppo costoso

=> PASSARE TUTTO SU UNA TEXTURE



PROBLEMA: Ultimo step non si può fare con una shader, deve in Frame Buffer

dobbiamo trovare in modo per lavorare con due texture da switchare (ring negz)



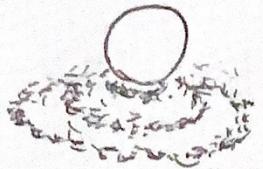
Accorgimenti WC

-1 quando ho aperto la luce mi sono accorto che c'era una sorta di pattern nei punti che venivano rifletti sul pavimento, come delle sfere concentriche

CENTRIFIC ANTIFACTING

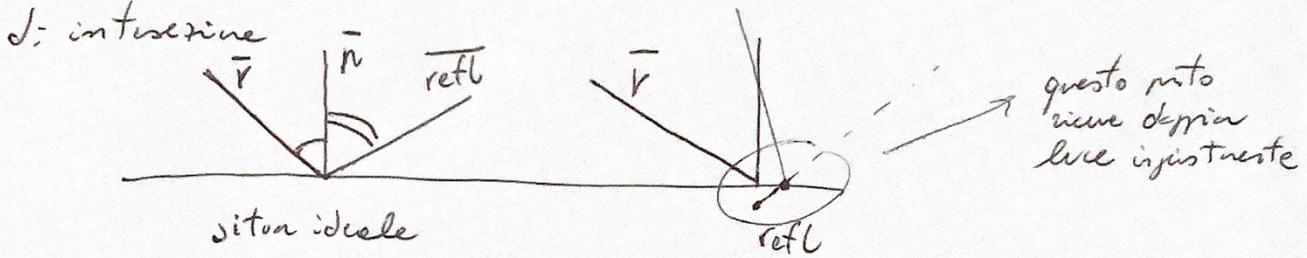
o

SOF INTERSECTION PROBLEM



problema non legato alla gamma dei neri cosulli, dato che anche se il suo raggio è solcato si vede bene che la gamma di neri cosulli sia giusta.

Il problema è legato al 2° raggio smesso. Se questo che il suo raggio attraversa un punto = intersection point, per problemi di antialiasing, la stessa superficie potrebbe essere filtrata una seconda volta, generando quindi un falso punto d'intersezione.

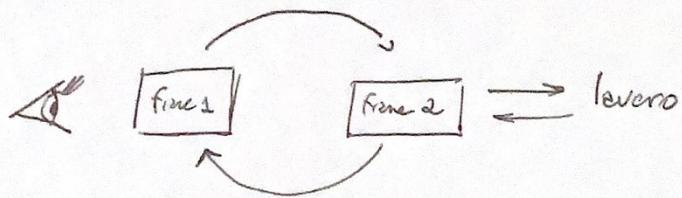


avviene quindi: un offset al punto d'incastello

FINE VERSI ONE 2

5.6 Progressive Rendering con Texture

L'idea è sfruttare le swap chain vista a lezione



Se vogliono ad esempio ordine a fare fade ~~the~~ verso a binario

a. Mostro immagine A (0,0,0)

b. nel frattempo aggiro l'immagine B (1,1,1)

c. succedono e così via

quello che faccio in b è $A + (1,1,1)$

Command

nel creare `VBuffers`, prima di andare a iniziare il renderingPass5 (prima perché se no poi Vulkan vuole me sotto istanze del subpass)

in retode `copyImage` far

VKCmdPipeline
Barrier

- 1) usa le barrier per la sincronizzazione per layout dst e src \checkmark
- 2) fa lo copy dell'immagine con `VKCmdCopyImage()`
- 3) ancora due barrier

PRACTICAMENTE È DA SUDE 65 DI L21

1) le barrier servono perché dall'intero ordine a trasformare le `VkImage` da un layout a un altro specifico

`VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL` = immagine per color attachment

`PRESENT_KHR` = da presentare alla superficie

`TRANSITION_DST_OPTIMAL` = immagine da destinare per una `copy` operation