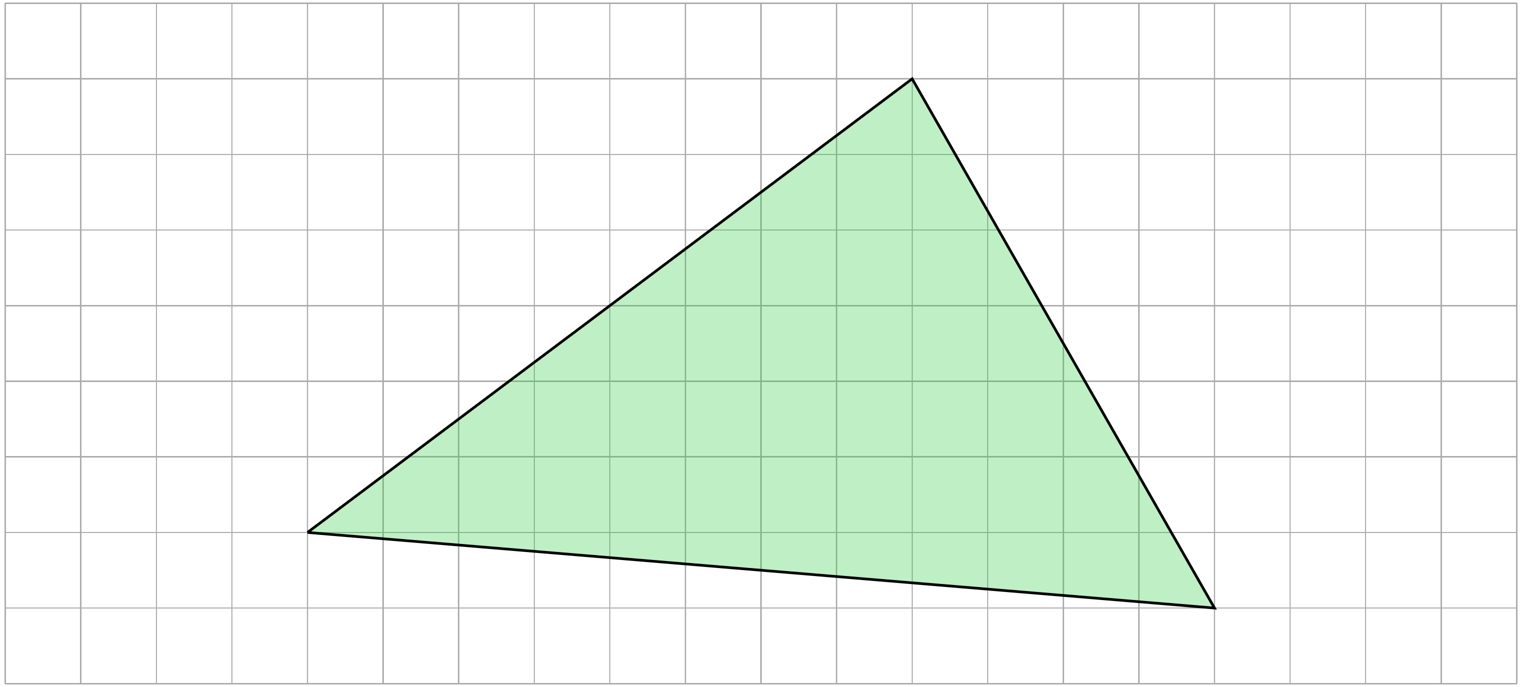
Raytracing

Rasterisering

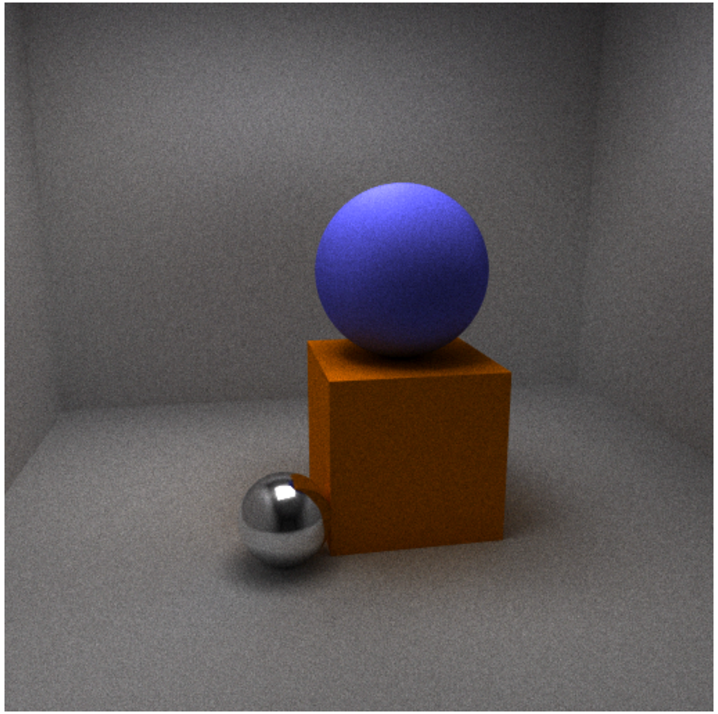
- Traditionell grafik bygger på rasterisering
- Föremål projiceras på bildytan och de täckta pixlarna fylls i
- Hög grad av parallellisering

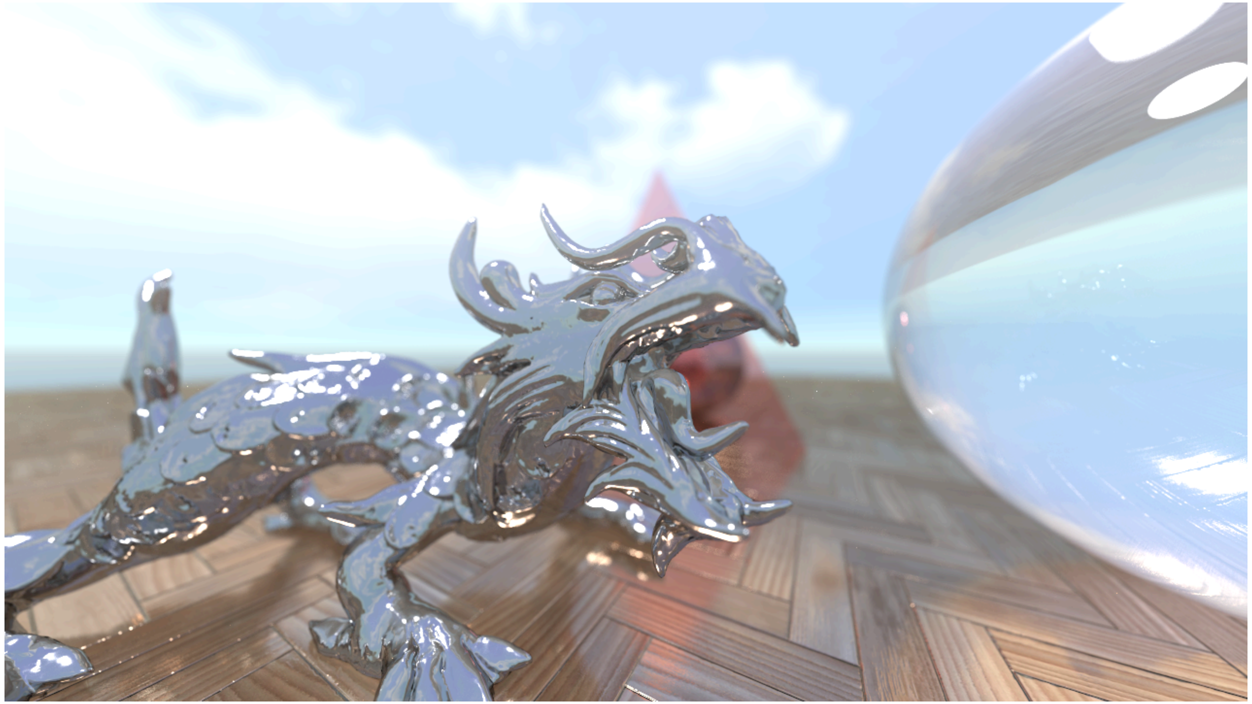


Raytracing

- Baserat på faktiska fysiska modeller.
- Simulerar hur ljusstrålar interagerar med en miljö.
- Mer realistiskt resultat än rasterisering, men dyrare.

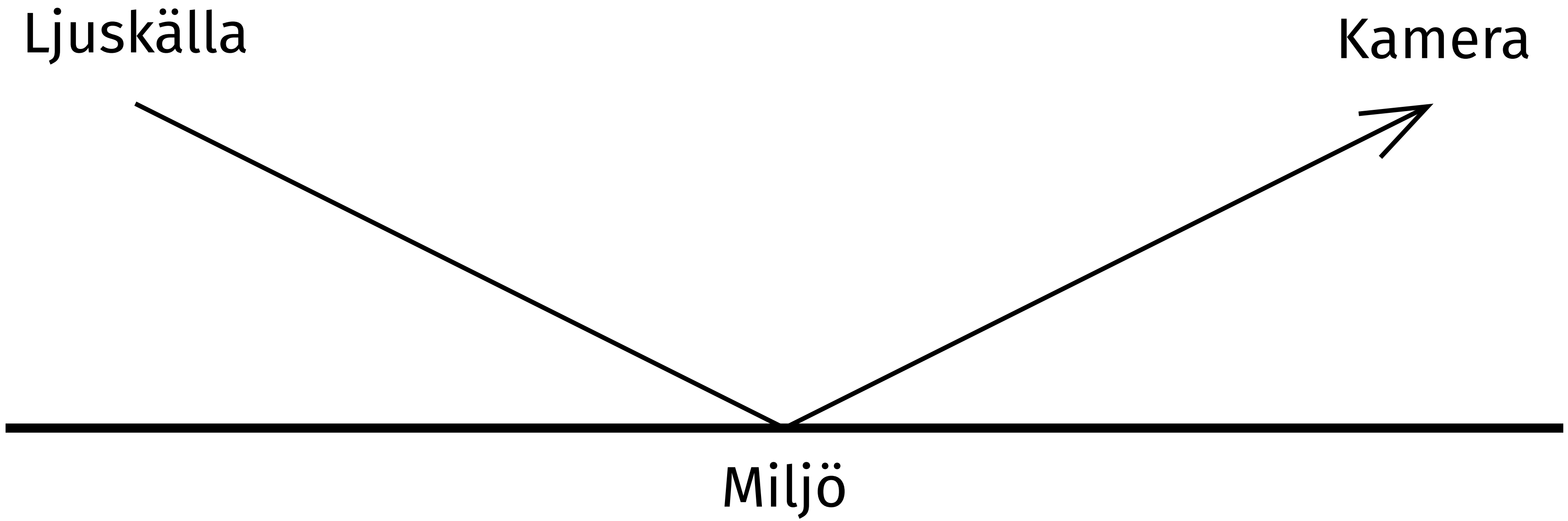
Raytracing



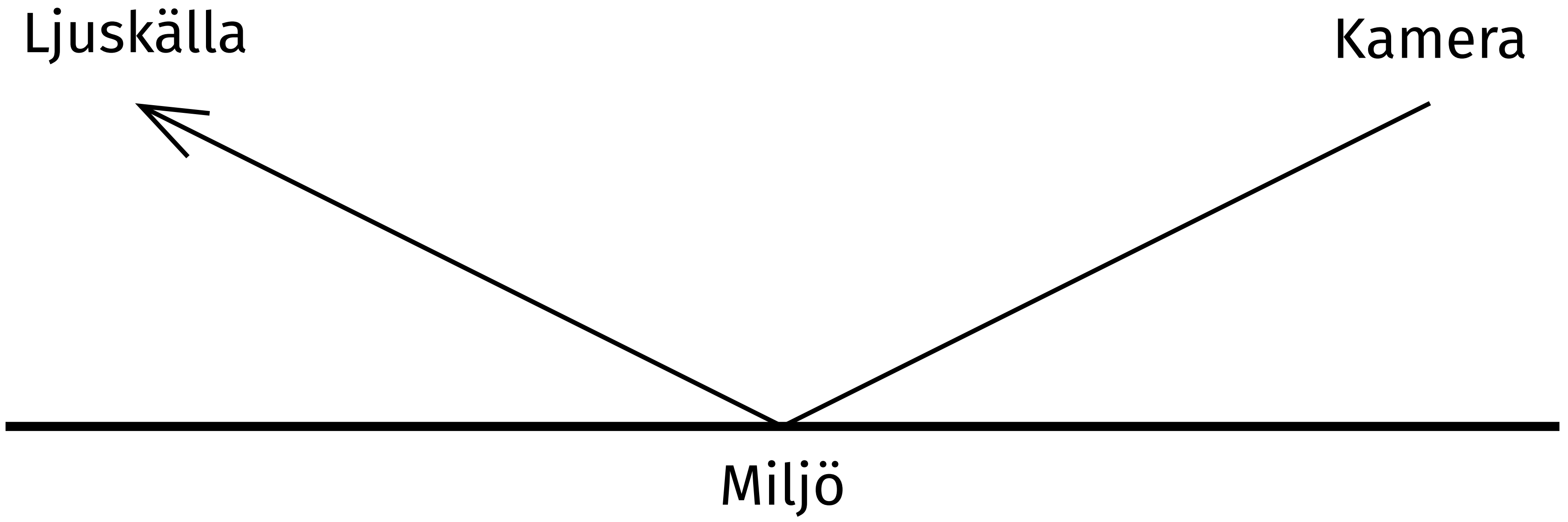


Raytracing

- Vi simulerar ljusstrålar från ljuskällor till kameran.
- Vad kan det det finns för problem med detta?



- Vi simulerar strålar baklänges istället.
- Vi tar bara hänsyn till ljus som träffar kameran.



Raytracing

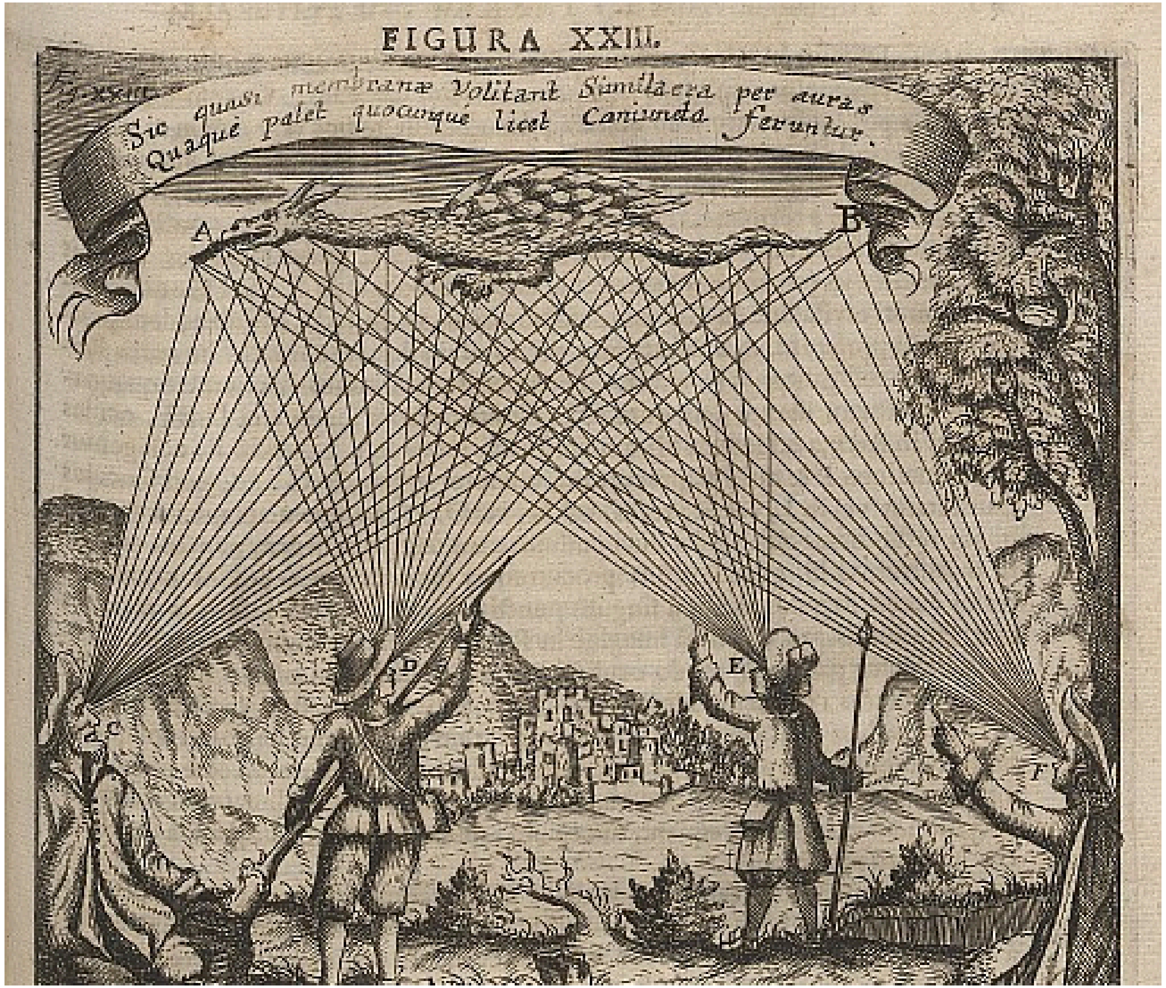


FIGURA XXIII.
Sic quasi membranæ Volitant Simulacra per auras
Quaque patet quocunque licet Conspicienda feruntur.
A
B
C
D
E
F

Matte!

Strålar/rays

- En stråle utgår från en punkt $\vec{O}$ och har en riktningsvektor $\vec{d}$.
- Kan parametriseras på följande sätt:

$$\vec{O} + t\vec{d}, \;\; t \geq 0$$

Skärningspunkter – plan

- Vi har ett plan med normalvektor $\vec{n}$ som innehåller punkten $\vec{x}$.
- Var (för vilket t) skär strålen $\vec{O} + t\vec{d}$ planet?

$$t = \frac{\left(\vec{x} - \vec{O}\right) \cdot \vec{n}}{\vec{d} \cdot \vec{n}}$$

- Visas på tavlan.
- Vad innebär det om vi får $t < 0$?

- Vi har en sfär med mittpunkt $\vec{c}$ och radie r.
- Var skär strålen $\vec{O} + t\vec{d}$ sfären?

$$\vec{v} = \vec{O} - \vec{c}, \;\; a = \vec{d} \cdot \vec{d},$$

$$b = 2\vec{v} \cdot \vec{d}, \;\; c = \vec{v} \cdot \vec{v} - r^2,$$

$$\boxed{at^2 + bt + c = 0.}$$

- Visas på tavlan.
- Vi kan få flera lösningar, vilken ska vi välja?

Skärningspunkter – trianglar

- Vi har en triangel med hörn $\vec{p_1}$, $\vec{p_2}$ och $\vec{p_3}$.
- Var skär strålen $\vec{O} + t\vec{d}$ triangeln?
- *En metod*: hitta först skärningspunkt med planet.
- Planets normal är parallell med $(\vec{p_2} - \vec{p_1}) \times (\vec{p_3} - \vec{p_1})$.
- Avgör sedan om punkten ligger i triangeln (visas på tavlan).
- Bättre metod: **Möller-Trumbore-algoritmen**.

Kamera

- Kan representeras av en brännpunkt som strålarna utgår från och ett bildplan som scenen avbildas på.
- Visas på tavlan.

Reflektioner

- En stråle med riktningsvektor $\vec{d}$ studsar mot en yta med normalvektor $\vec{n}$.
- Om $|\vec{n}| = 1$ blir den nya riktningsvektorn

$$\vec{r} = \vec{d} - 2\left(\vec{d} \cdot \vec{n}\right)\vec{n}$$

- Visas på tavlan.
- För att få matta ytor, kan vi addera en liten slumpmässig vektor (förenklad modell).

Raytracing – pseudokod

for varje pixel p **do**
- $\vec{b} \leftarrow$ punkten i bildplanet som motsvarar p
- $R \leftarrow$ stråle som går från brännpunkt via $\vec{b}$
- $C \leftarrow \vec{1}$ (helt vit färgvektor)
- $s \leftarrow$ max. antal studs
- **while** $s > 0$ **do**
  - Hitta föremål som R skär (närmast, d.v.s. minsta t)
  - **if** ingen skärningspunkt **then**
    - $C \leftarrow C * F_B$ (bakgrundsfärg)
    - **break**
  - $R \leftarrow$ studsa R
  - $C \leftarrow C * F$ (färgen på föremålet)
  - $s \leftarrow s - 1$
- **if** $s = 0$ **then** $C = \vec{0}$ (svart)
- Ge p färgen C

Renderingsekvationen

$$L_\mathrm{o}(\boldsymbol{x}, \omega_\mathrm{o}, \lambda, t) = L_\mathrm{e}(\boldsymbol{x}, \omega_\mathrm{o}, \lambda, t) + \int_\Omega f_\mathrm{r}(\boldsymbol{x}, \omega_\mathrm{i}, \omega_\mathrm{o}, \lambda, t) L_\mathrm{i}(\boldsymbol{x}, \omega_\mathrm{i}, \lambda, t)(\omega_\mathrm{i} \cdot \boldsymbol{n})\,\mathrm{d}\omega_\mathrm{i}$$

L_o totalt ljus ut

L_e strålat ljus ut

L_i totalt ljus in

λ våglängd

t tid

$\boldsymbol{x}$ plats i rymden

$\boldsymbol{n}$ normalvektor

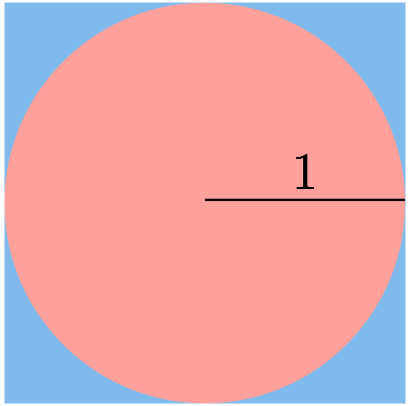
ω_i inåtriktning

ω_o utåtriktning

Ω halvsfären som omringar $\boldsymbol{n}$

f_r reflektivitetsfunktionen (`BRDF`, t.ex. Lambert) för materialet, avgör utseendet

- Täcker många aspekter av rendering med fysiska modeller
- Inte alla, exempelvis transmission
- Varianter finns, både mer och mindre komplicerade
- Svår integral!



$$\frac{\int_{\bigcirc} \mathrm{d}A}{\int_{\square} \mathrm{d}A}$$

```python
def montecarlo():
    n = 0
    for i in range(0, 1_000_000):
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)
        if x**2 + y**2 <= 1:
            n += 1
    return n / 1_000_000
```

Monte Carlo i raytracing

Huvudsakliga egenskapen hos en *path tracer* är att de använder Monte Carlo metoden för att hitta en approximativ lösning av renderingsekvationen

$$L_\mathrm{o}(\boldsymbol{x}, \omega_\mathrm{o}, \lambda, t) = L_\mathrm{e}(\boldsymbol{x}, \omega_\mathrm{o}, \lambda, t) + \int_\Omega f_\mathrm{r}(\boldsymbol{x}, \omega_\mathrm{i}, \omega_\mathrm{o}, \lambda, t) L_\mathrm{i}(\boldsymbol{x}, \omega_\mathrm{i}, \lambda, t)(\omega_\mathrm{i} \cdot \boldsymbol{n})\,\mathrm{d}\omega_\mathrm{i}$$

```
def bounce_ray(x, d, n, mat):
    v = random_hemisphere_vector(n) # Generera en ny slumpmässing stråle
    incoming = trace_ray(x, v)       # Sänd en rekursiv stråle
    reflectance = mat.brdf(x, d, v) # Beräkna reflektiviteten
    probability = 1 / (2 * PI)       # Beräkna sannolikheten för strålen
    return mat.emittance(d, n) +
        reflectance * incoming * dot(v, n) / probability
```

Skicka strålar många gånger för varje pixel och ta det genomsnittliga värdet

Importance sampling

- Likformig fördelning ger många strålar med väldigt lite ljus
- Ge större sannolikhet att välja strålar som bidrar mycket ljus

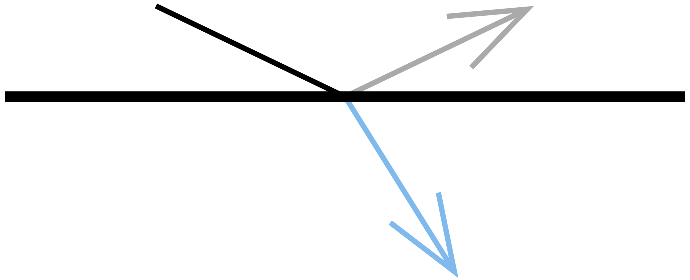
```
def bounce_ray(x, d, n, mat):
    # Ej längre uniform fördelning!
    v = random_hemisphere_vector(x, n) # Generera en ny slumpmässing stråle
    incoming = trace_ray(x, v)         # Sänd en rekursiv stråle
    reflectance = mat.brdf(x, d, v)    # Beräkna reflektiviteten
    # Sannolikheten är nu en funktion istället
    return mat.emittance(d, n) +
        reflectance * incoming * dot(v, n) / probability(x, v, n)
```

Transmittering

- En del av strålningen reflekteras, resten transmitteras
- Snells lag för att hitta brytningsvinkeln mellan brytningsindex n_1 och n_2
- Fresnels ekvation för reflektivitet, eller Schlick's approximation:

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5, R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

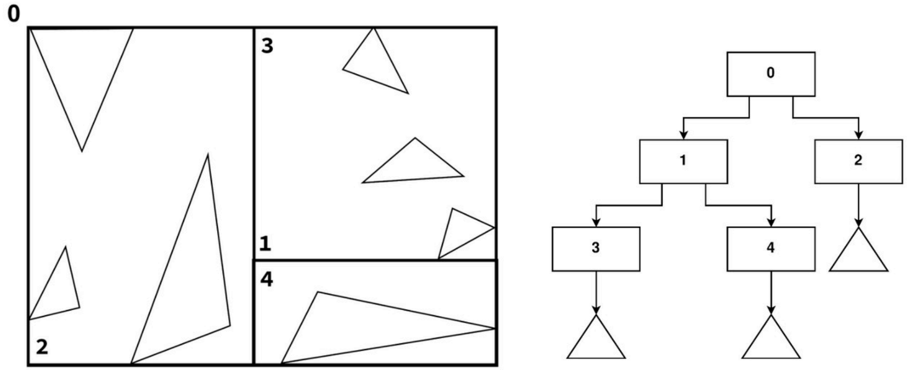
- *Importance sampling*: välj reflektionsstråle med sannolikhet R, transmitterings-stråle med sannolikhet $1 - R$

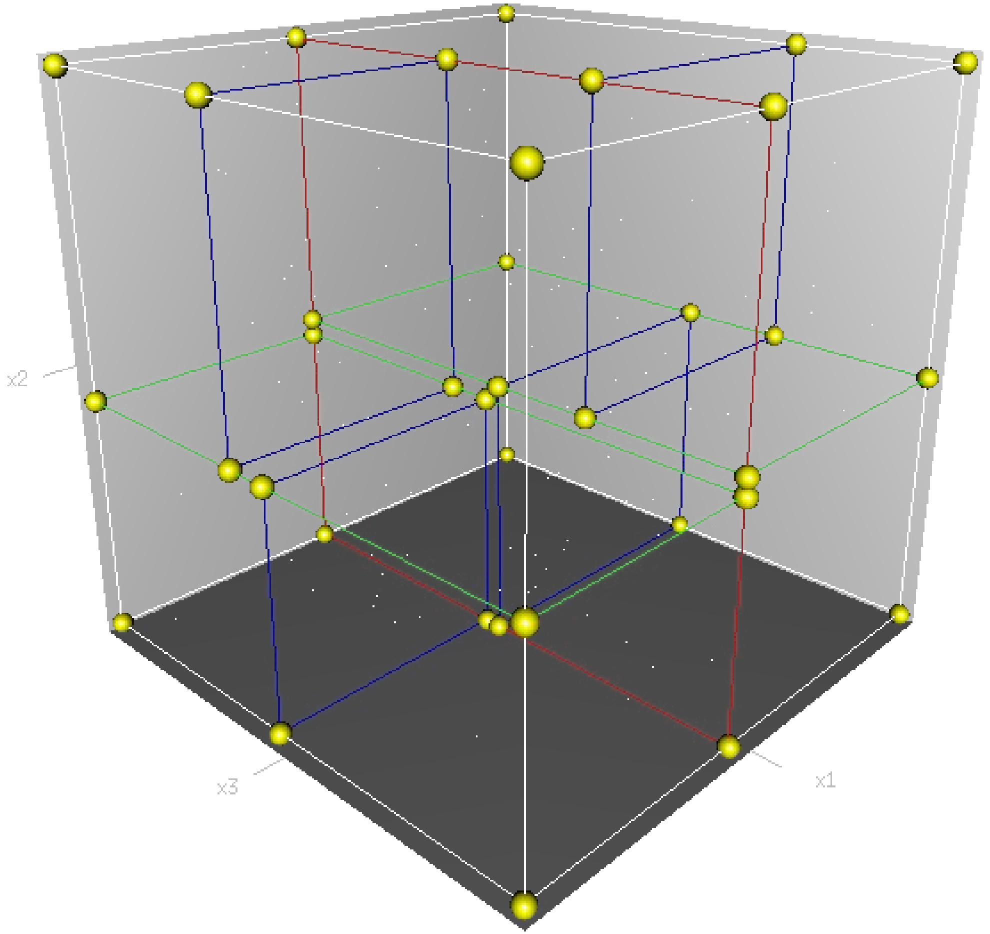


Accelerationsstrukturer

Hur gör man det snabbt? Accelerationsstrukturer!

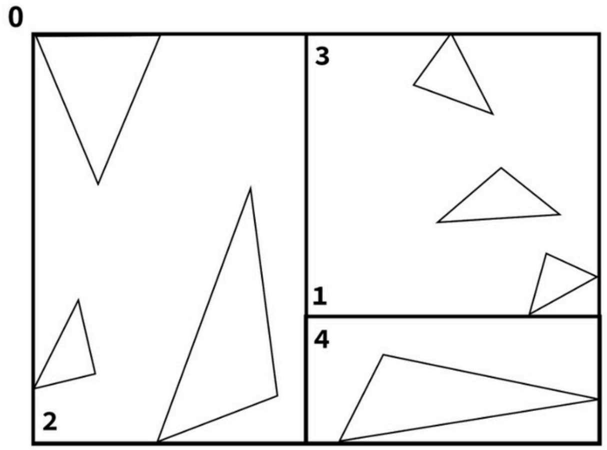
- Bounding volume hierarchy (BVH)
- K-dimensional tree (KD-tree)
- Octree

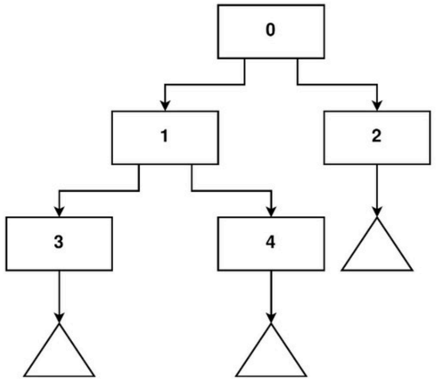




Bounding volume hierarchy

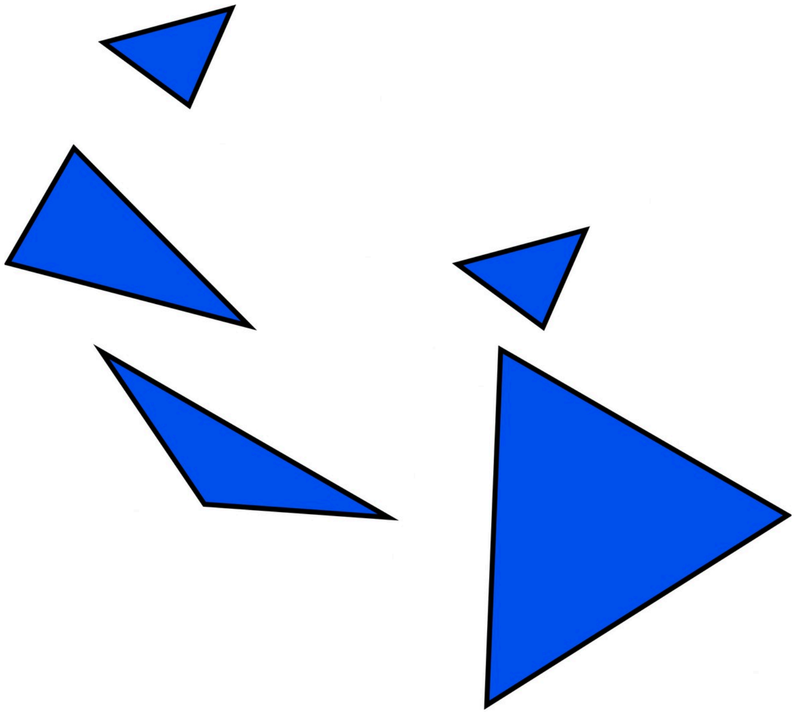
- Scenen sluts in en volym
- Volymer delas in i mindre volymer
- Axis-Aligned Bounding-Boxes (AABB’s)
- Logaritmisk tidskomplexitet för intersection





Hur delar vi upp lådor?

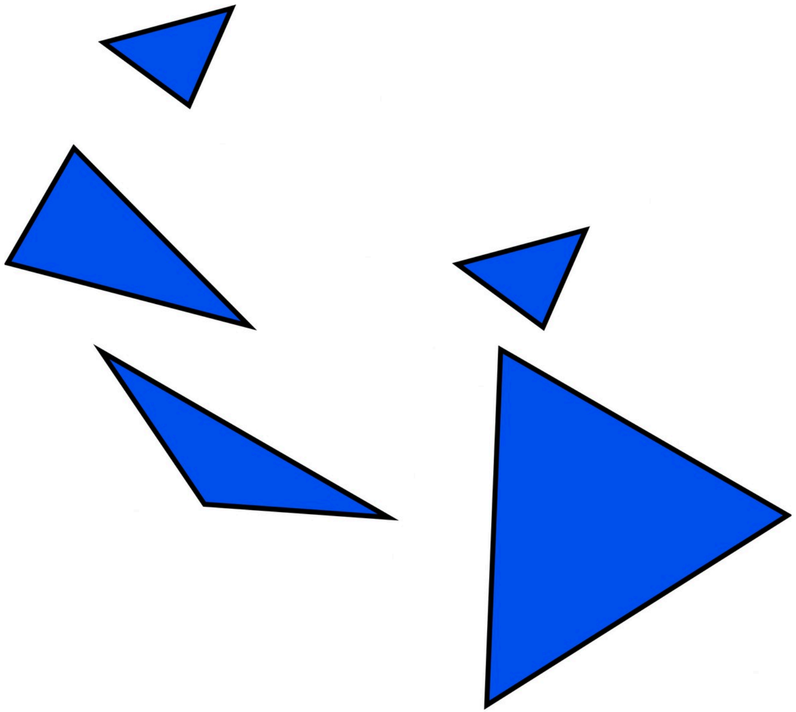
- Sortera primitiver efter centroid
- Dela volymen på mitten



Hur många primitiver i varje låda?

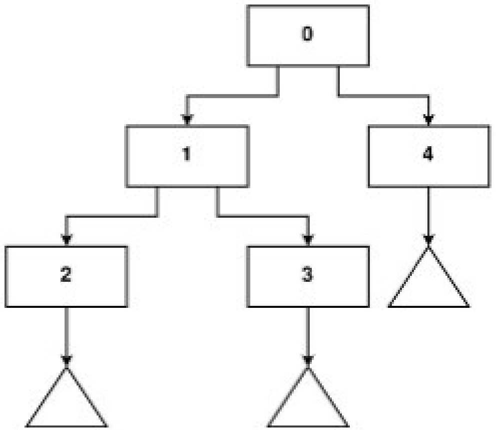
- Surface Area Heuristic
- Testa många olika ställen att dela volymen
- Använd en kostnadsfunktion

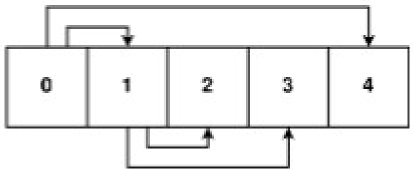
$$c = n_l A_l + n_r A_r$$



Hur använder vi trädet?

- Trädsökning från roten
- Trädet kan representeras effektivt





Läxa:
Skriv en raytracer