Marco Tarini - Computer Graphics 2023/2024
Università degli Studi di Milano

Approcci al rendering 2/2: rasterization-based (intro)

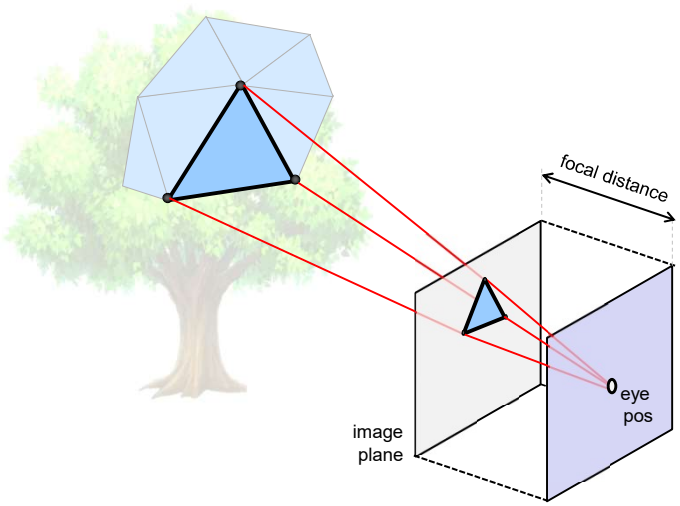
67

Algoritmi di rendering: due categorie

- ✓Bastati su Ray-tracing:
  - ⇒*per ogni pixel*:
    lancio un raggio;
    trovo le intersezioni del raggio con le primitive;
    determino il colore del punto colpito
    (per es, calcolando le luci da cui è raggiunto,
    l'illuminazione…)
- ✓Basati su Rasterization
  - ⇒*per ogni primitiva*:
  - ⇒la proietto sullo schermo, in 2D ("trasformazione")
  - ⇒converto la forma 2D in pixel ("rasterizzazione")
  - ⇒determino il colore di questi pixel ("lighting")

68

Rasterization based rendering



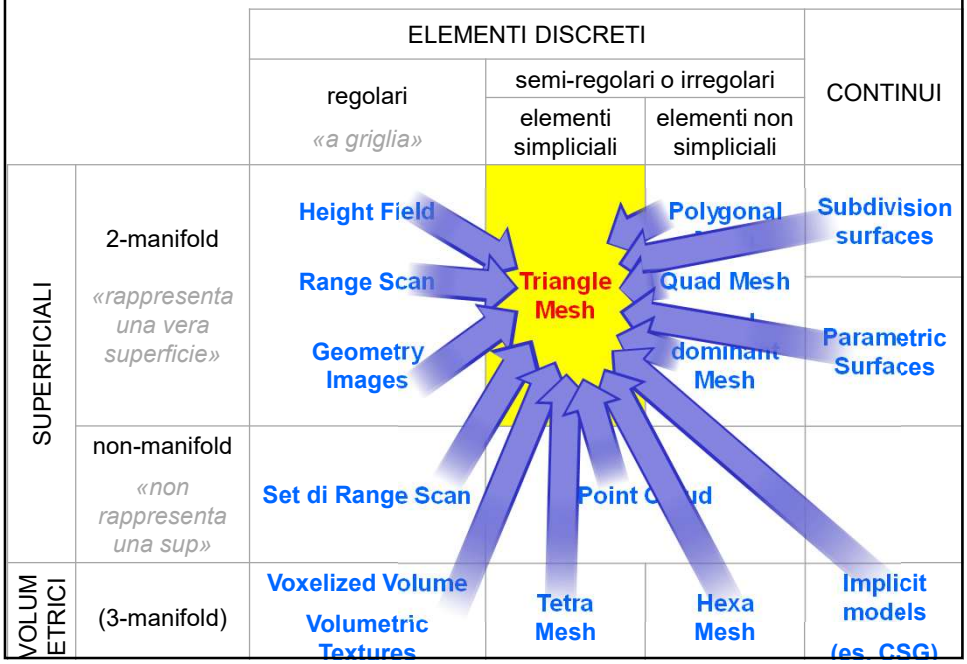
70

Algoritmi di rendering basati su rasterizzazione

- ✓ Sono resi molto popolari dal supporto di Hardware specializzato: GPU
  - ⇒ è proprio l'algoritmo per cui questo Hardware è stato originalmente pensato e progettato!
  - ⇒ Rendering accelerato con GPU: necessario per rendering in tempo reale (per es, videogame)
  - ⇒ (Recentemente: anche algoritmi basati su ray-tracing vengono spesso accelerati su GPU)
- ✓ La principale (e quasi unica) primitiva di rendering supportata è il triangolo
  - ⇒ Cioè, la GPU è in grado rasterizzare triangoli
- ✓ Dunque, l'Hardware-supported rendering in tempo reale è soprattutto rendering di Tri-meshes
  - ⇒ E' questo il motivo per il quale, nella prima metà del corso, ci siamo occupati così spesso di come convertire qualsiasi altro tipo di modello in una mesh triangolare

71

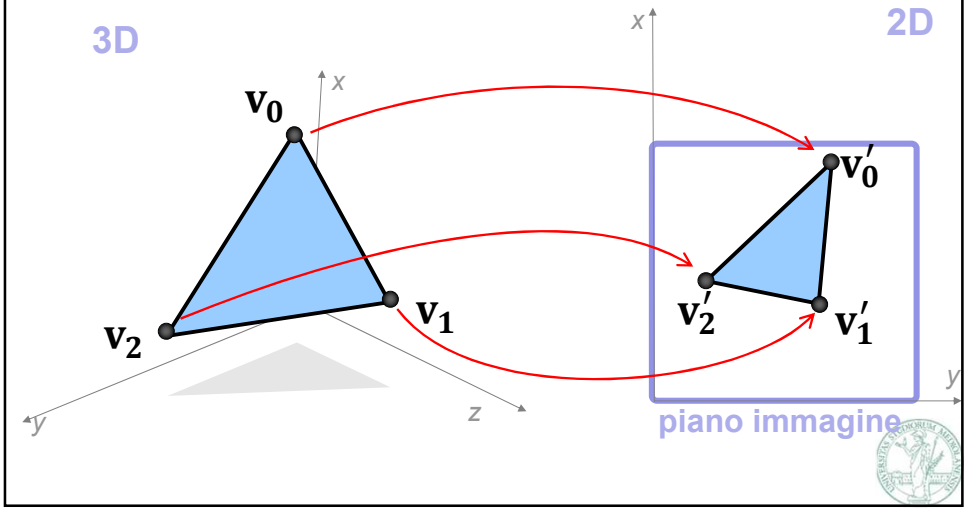
Al fine di eseguire rendering (rasterization based)



72

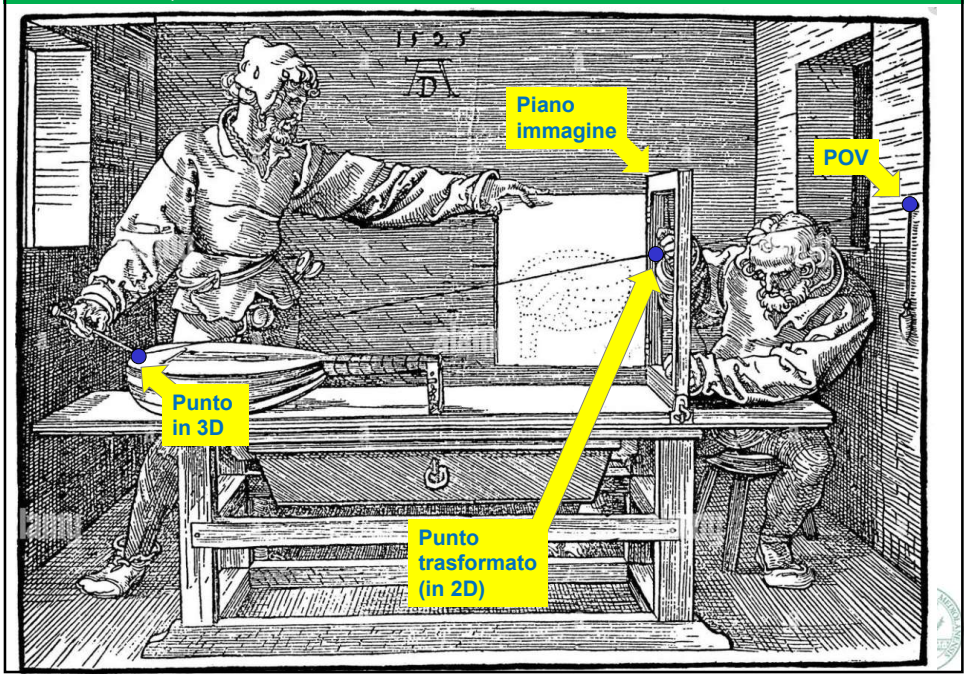
Trasformare un triangolo (per es, da 3D a 2D)

1. trasformare ciascuno dei suoi tre vertici,
2. connetterli formando in un nuovo triangolo (2D)



74

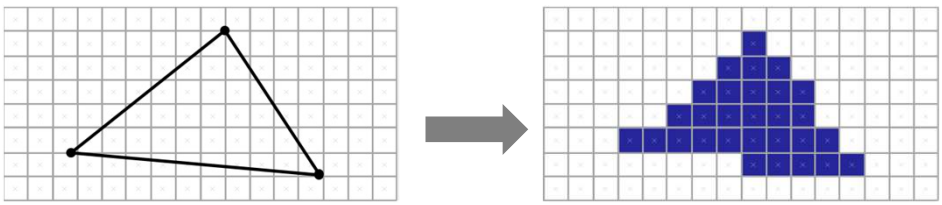
In fondo, l'idea non è nuova



75

Rasterizzare (convertire in pixel) un triangolo

- ✓ Task: trovare quali «frammenti» generare
  - ⇒ uno per ogni pixel dell'immagine output contenuto dal triangolo
  - ⇒ (nota: è un task che svolgiamo in 2D, sull'immagine)
  - ⇒ «Frammento» = un piccolo pacchetto di dati per il quale vogliamo generare un pixel (contiene esempio: normale, colore di base, materiale…)
  - ⇒ Nota: lo distinguiamo da «pixel»: il colore RGB finale generato per un dato frammento



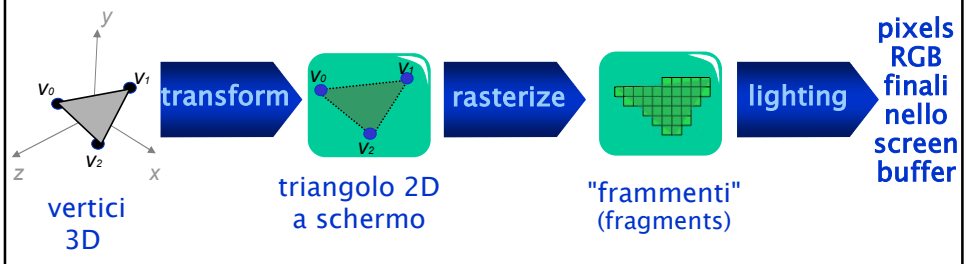
76

Anche detto T&L: Transform & Lighting

- ✓ *Transform* :
  - ⇒ trasformazioni di sistemi di coordinate
  - ⇒ scopo: portare le primitive in spazio schermo
- ✓ *Lighting* :
  - ⇒ computo illuminazione
  - ⇒ scopo: calcolare il colore RGB finale di ogni pixel della immagine finale

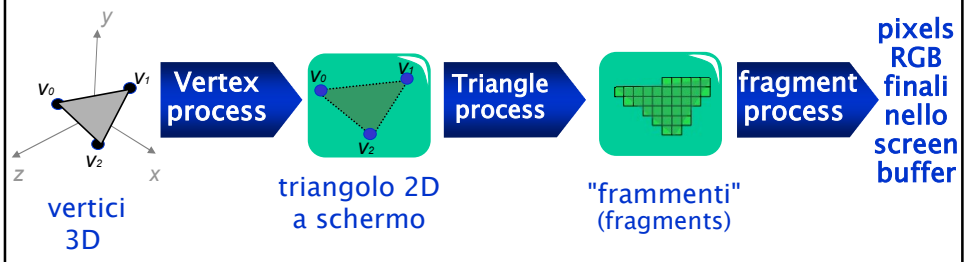
79

Rasterization-Based Rendering: il pipeline



80

Rasterization-Based Rendering: il pipeline



81

Rasterization-based rendering

- ✓ Input: una mesh (array di vertici e triangoli)
- ✓ L'algoritmo a pipeline (catena di montaggio) consiste in diverse fasi (stage della catena):

1. Fase per vertice:
   ogni vertice viene portato in una posizione 2D sullo schermo
   ⇒ Si tratta di una "trasformazione spaziale"
2. Fase per triangolo:
   ogni triangolo 2D viene rasterizzato a schermo
   ⇒ Viene cioè identificato un "frammento" per ogni pixel coperto dal triangolo 2D
3. Fase per frammento
   ⇒ Per ogni frammento, si computa il colore RGB del pixel corrispondente (tipicamente, calcolando l'illuminazione)

83

Considerazioni generali sul Pipeline

- ✓ Le fasi, che sono in cascata, avvengono in parallelo
  - ⇒ Mentre processo (trasformo) i vertici di un triangolo, sto anche processando (rasterizzando) il triangolo precedente, e processando (computando il lighting) dei frammenti del triangolo generato ancora prima
- ✓ Ogni fase del pipeline avviene in parallelo
  - ⇒ Ogni vertice, triangolo, frammento può essere processato in contemporanea con altri
- ✓ Il pipeline procede tanto velocemente quanto la sua fase più lenta
  - ⇒ Che è detta «il collo di bottiglia» (bottleneck)
- ✓ Terminologia per il pipeline di rendering
  - ⇒ Se il bottleneck è nella fase per vertice, l'applicazione si dice «transform limited» o «geometry limited»: non riesce ad effettuare abbastanza trasformazioni geometriche (ad esempio, la mesh ha una risoluzione troppo elevata)
  - ⇒ Se è nella fase per frammento, l'applicazione si dice «fill limited»: non riesce a riempire il buffer di pixel abbastanza velocemente (ad esempio, l'immagine di output ha una risoluzione troppo elevata)

84

Rendering Algorithms Paradigms: in breve

RAY-TRACING

```
For each image pixel p:
  make a ray r
  for each primitive o in scene:
     if intersect(r,o)
     then find color for o
          color p with it
```

LIGHTING

RASTERIZATION BASED:

```
For each primitive o:
  find where o falls on screen
  rasterize 2D shape
  for each produced pixel p :
    find color for o
    color p with it
```

PROJECTION 3D → 2D (aka TRANSFORM)

LIGHTING

85

Rendering Algorithms paradigms: ancora più in preve

RAY-TRACING

```
for each pixel:
    for each primitive
```

RASTERIZATION

```
for each primitive:
    for each pixel
```

86

THE GREAT DEBATE

RASTERIZATION

VS

RAY-TRACING

87

Quali primitive di rendering, nei due approcci?

Q: quali primitive di rendering usando rasterization?

A: qualunque cosa io sappia:

1) trasformare da 3D a 2D (facile), ma soprattutto
2) «rasterizzare» (in 2D)

convertire in pixel

per le GPU

Allo stato attuale, tre cose:

1. PUNTI — Usato, per es, per le point clouds (**point-splatting**)
2. SEGMENTI — Usato, per es, per rendering di mesh in **wireframe**
3. **TRIANGOLI** — Usato per **tri-mesh**. Il caso principale! (di gran lunga). Le GPU sono ottimizzate per questo caso. E' il motive principale della popolarità delle tri-mesh.

Q: quali primitive di rendering usando ray-tracing?

A: qualunque cosa io sappia:

intersecare con un raggio

efficacemente!

Quindi:

⇒ Triangoli? — Per es, per triangle mesh

Anche, ma anche primitive più complesse, come:

⇒ Implicit surfaces — Per es, per CSG
⇒ Height-fields
⇒ Voxelized dataset — Il metodo principale per «direct volume rendering»
⇒ Subdivision surfaces…

88

Ray-tracing : semplicità ed eleganza?

```
typedef struct{double x,y,z}vec;vec U,black,amb={.02,.02,.02};struct sphere{
vec cen,color;double rad,kd,ks,kt,kl,ir}*s,*best,sph[]={0.,6.,.5,1.,1.,1.,.9,
.05,.2,.85,0.,1.7,-1.,8.,-.5,1.,.5,.2,1.,.7,.3,0.,.05,1.2,1.,8.,-.5,.1,.8,.8,
1.,.3,.7,0.,0.,1.2,3.,-6.,15.,1.,.8,1.,.7,.0,.0,0.,.6,1.5,-3.,-3.,12.,.8,1.,
1.,5.,0.,0.,0.,.5,1.5,};yx:double u,b,tmin,sqrt(),tan();double vdot(A,B)vec A
,B;{return A.x*B.x+A.y*B.y+A.z*B.z;}vec vcomb(a,A,B)double a;vec A,B;{B.x+=a*
A.x;B.y+=a*A.y;B.z+=a*A.z;return B;}vec vunit(A)vec A;{return vcomb(1./sqrt(
vdot(A,A)),A,black);}struct sphere*intersect(P,D)vec P,D;{best=0;tmin=1e30;s=
sph+5;while(s-->sph)b=vdot(D,U=vcomb(-1.,P,s->cen)),u=b*b-vdot(U,U)+s->rad*s
->rad,u=u>0?sqrt(u):1e31,u=b-u>1e-7?b-u:b+u, tmin=u>1e-7&&u<tmin?best=s,u:
tmin;return best;}vec trace(level,P,D)vec P,D;{double d,eta,e;vec N,color;
struct sphere*s,*l;if(!level--)return black;if(s=intersect(P,D));else return
amb;color=amb;eta=s->ir;d= -vdot(D,N=vunit(vcomb(-1.,P=vcomb(tmin,D,P),s->cen
)));if(d<0)N=vcomb(-1.,N,black),eta=1/eta,d= -d;l=sph+5;while(l-->sph)if((e=l
->kl*vdot(N,U=vunit(vcomb(-1.,P,l->cen))))>0&&intersect(P,U)==l)color=vcomb(e
,l->color,color);U=s->color;color.x*=U.x;color.y*=U.y;color.z*=U.z;e=1-eta*
eta*(1-d*d);return vcomb(s->kt,e>0?trace(level,P,vcomb(eta,D,vcomb(eta*d-sqrt
(e),N,black))):black,vcomb(s->ks,trace(level,P,vcomb(2*d,N,D)),vcomb(s->kd,
color,vcomb(s->kl,U,black))));}main(){printf("%d %d\n",32,32);while(yx<32*32)
U.x=yx%32-32/2,U.z=32/2-yx++/32,U.y=32/2/tan(25/114.591559026l),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*minray!*/
```

un intero raytracer (in C) su una business card :-P !
(by Paul Heckbert)

90

Render farms

- ✓ Il rendering di entrambi i tipi sono massicciamente paralelizzabili (sono cioè caratterizzati da un elevato livello di *Implicit Parallelism*)
- ✓ Per avvantaggiarsene, abbiamo bisogno di hardware parallelo
- ✓ Per es, per gli algoritmi di ray-tracing…

Pixar/Disney Render Farm

91

Vantaggi degli approcci basati su rasterizzazione

- ✓ Complessità lineare con numero di primitive
- ✓ Per ogni primitiva, si processano solo i pixel coinvolti – migliore scalabilità?
- ✓ Attualmente possono contare sul supporto GPU, su qualsiasi piattaforma

  ⇒E' la classe di algoritmo per la quale le GPU sono state pensate

RASTERIZATION BASED

92

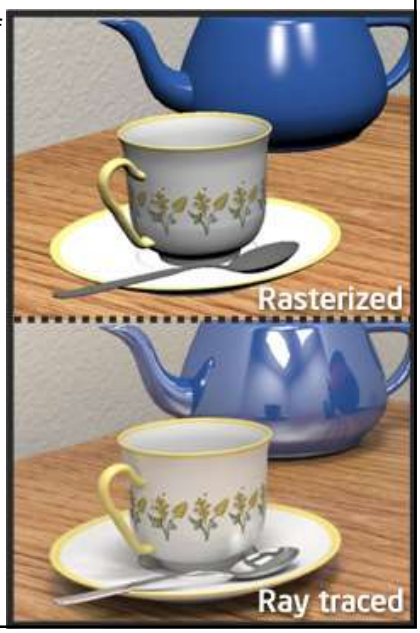
Visione storica / tradizionale (i luoghi comuni)

- ✓Ray-tracing based
  - ⇒Lenti, ma ottima qualità di immagini
  - ⇒Gli algoritmi standard per il rendering offline
  - ⇒Per esempio, usati nella movie industry
  - ⇒Alcuni software ray-tracers / path-tracers noti: POV-ray, renderman (pixar), YafaRay, Mitsuba
- ✓Rasterization based
  - ⇒Veloce, ma approssimato
  - ⇒Gli algoritmi standard per il rendering online
  - ⇒Per esempio, usato nella game industry
  - ⇒Alcuni API basati su rasterization based: OpenGL, WebGL, DirectX, Metal, Vulkan

93

Il dibattito dura da decenni

"Real Time Ray-Tracing: The End of Rasterization?" (Jeff Howard, 2007)



95

I luoghi comuni 2/2

✓ Raytracing :

⇒ effetti visuali complessi → realismo

- ombre, riflessioni speculari, rifrazioni, riflessioni multiple...

⇒ *quindi:* perfetto per rendering off-line / hi-quality!

⇒ Il rendering della movie industry

✓ Rasterization:

⇒ veloce

- per ciascuna primitiva, processa solo i pixel coinvolti (invece di: per ogni pixel, processa tutte le primitive)

⇒ *quindi:* perfetto per il rendering real-time e approssimato

⇒ Il rendering della game industry

97

La realtà: raytracing non è necessariamente lento

✓ Perché…

⇒ È anch'esso intrinsecamente parallelizzabile

- alto livello di **parallelismo implicito**
- quindi, implementabile con HW parallelo (per es, render farms)

⇒ Le primitive possono essere più varie ed espressive

- Ciascuna rappresenta una forma più articolata
- Riduce il numero di primitive necessarie a comporre la scena virtuale

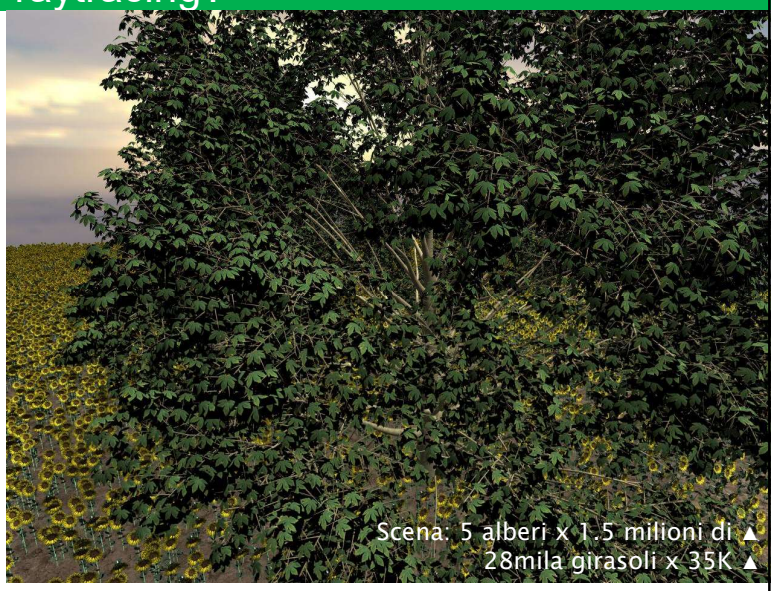
⇒ Ma soprattutto: **Ottimizzazioni**: usando apposite strutture dati, è possibile ridurre fortemente il numero di primitive da testare per ogni raggio

- Strutture di **indicizzazione spaziale**, come **octree** (o altre)
- Nota: è un po' più arduo per scene animate
- In questo modo, le complessità degli algoritmi di Ray-Tracing può essere resa **sublineare** col numero di primitive

98

Real-time raytracing?

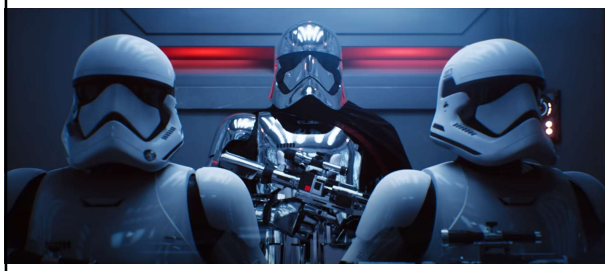
Un precursore → (su HW specializzato per questo caso)



OpenRT Project
inTrace Realtime Ray Tracing Technologies GmbH
MPI Informatik, Saarbrueken - by Ingo Wald et al , 2004

99

Oggi: real-time raytracing sta diventando comune



Unity
Real time Raytracing
(2019)

Unreal + Nvidia
Real-time Raytracing
(2019)

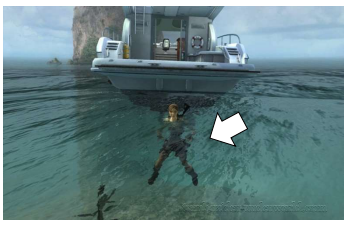
102

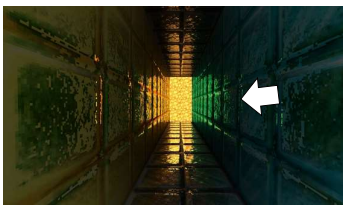
La realtà: rasterization based può riprodurre effetti compessi e ralistici

- ✓ Esistono algoritmi basati sul rasterization per simulare, oppure approssimare:
  - ⇒ Occlusioni («in una foto, gli oggetti più vicini alla camera nascondono quelli dietro»). (ebbene sì: abbiamo bisogno di un'apposita strategia, che vedremo, anche solo per riprodurre questo naturale stato di cose)
  - ⇒ Ombre portate (sia «soft» che «hard»)
  - ⇒ Diffrazioni
  - ⇒ Riflessioni speculari
  - ⇒ Riflessioni multiple (illuminazione «globale»)
  - ⇒ Rifrazioni e semitrasparenze
  - ⇒ …e altri effetti naturali da riprodurre negli algoritmi di raytracing
- ✓ Si tratta di algoritmi specializzati, ciascuno pensato per riprodurre un effetto
  - ⇒ Spesso sono adottati nei videogames
  - ⇒ Nota: non sempre combinarli insieme è semplice e diretto

103

La realtà: rasterization based supporta effetti di luce compessi





104

Lo stato attuale dell'industria di intrattenimento

- ✓ L'industria cinematografica si avvale quasi esclusivamente di algoritmi basati su ray-tracing
  - ⇒ Per il rendering finale (offline)
  - ⇒ Ma usa algoritmi basati su rasterization per: effettuare preview (real time), per costruire asset (ad esempio, modellare le mesh), etc
  - ⇒ Rendering finale: parallelizzato massicciamente su render farms, ma cmq spesso lento (ore o anche decine di ore per fotogramma)
- ✓ L'industria dei videogames si avvale ancora, in maggioranza, di algoritmi basati su rasterization
  - ⇒ Accelerati su GPU consumer-level
  - ⇒ Dunque i modelli 3D usati sono sempre tri-meshes
  - ⇒ Trend recente (ancora difficile valutarne la portata): passaggio ad algoritmi di ray-tracing, oppure misti

106

La soluzione del dibattito

> *Rasterization is fast,*
> *but needs cleverness*
> *to support complex visual effects.*
>
> *Ray tracing supports complex visual effects,*
> *but needs cleverness*
> *to be fast.*

David Luebke (NVIDIA)

108