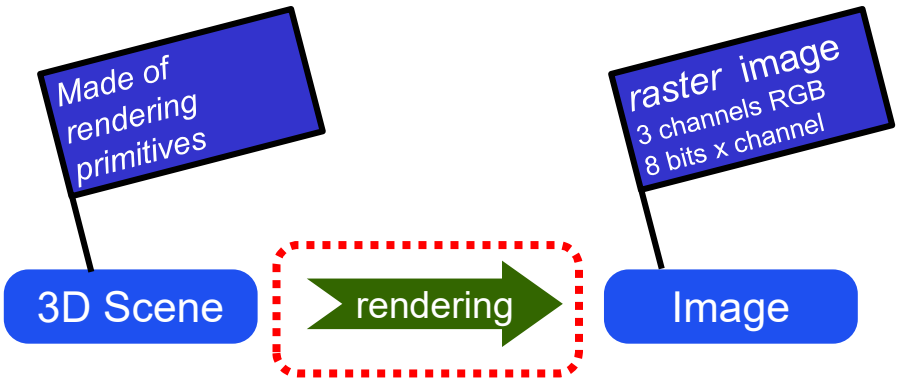
Marco Tarini - Computer Graphics 2019/2020
Università degli Studi di Milano

Rendering Overview: Rasterization

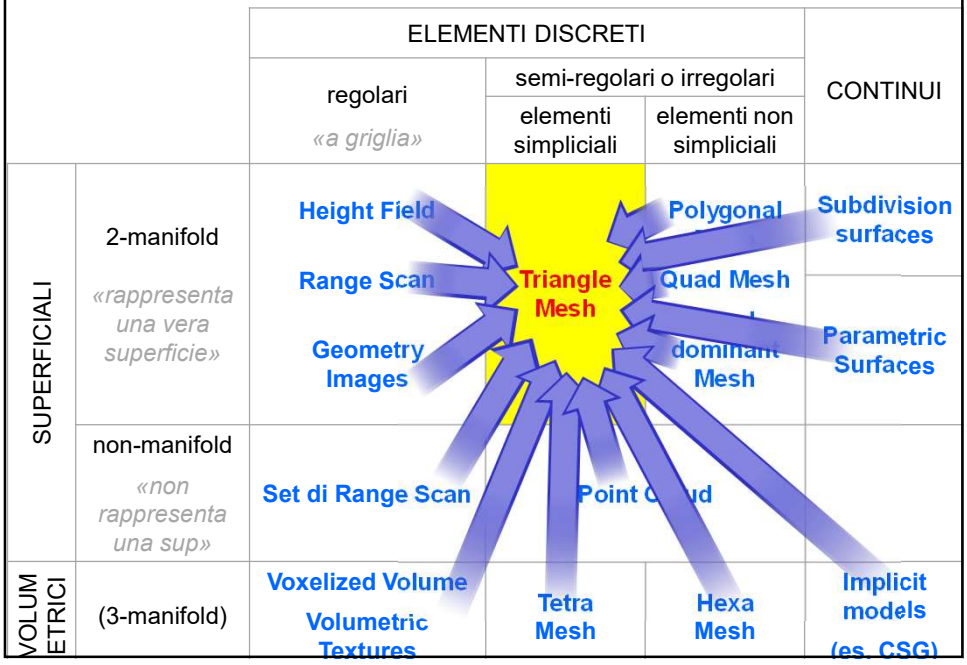
47

PART 2: *the other main approach*



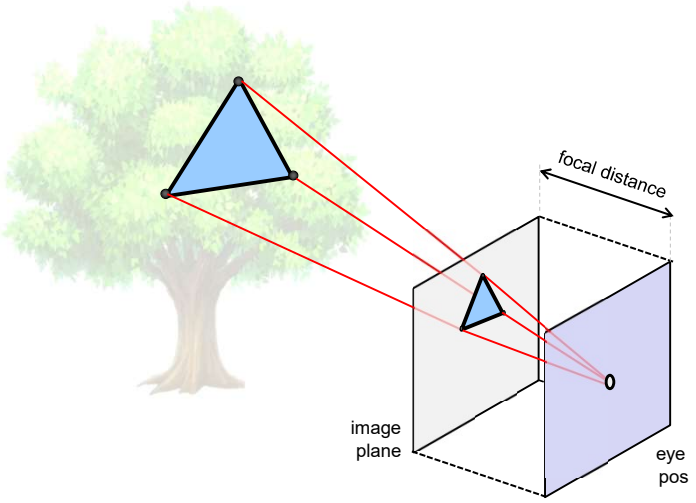
48

Al fine di eseguire rendering (rasterization based)



50

PART 2: *the other main approach*



51

Rendering Algorithms Paradigms

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

52

to “Rasterize”:

- ✓ convert a 2D shape
  - e.g.: text, polygons, curves, …
- ✓ into pixels
  - in a raster image

Jim Blinn, C.G. pioneer.
(OLD photo)
One hobby: colleting algorithms
to rasterize circles

53

Rasterization based : Primitive di rendering

Q: quali primitive di rendering per rasterization based?

A: qualunque cosa io sappia:

1) proiettare da 3D a 2D
2) «rasterizzare» in 2D
   - == convertire in pixel

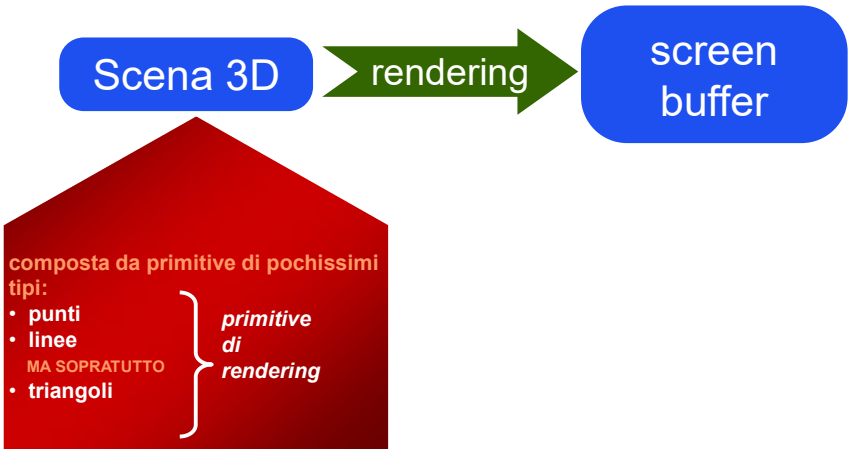
Most commonly:

⇒ *SEGMENTI*
⇒ *PUNTI*
⇒ ma soprattutto *TRIANGOLI*

54

Rasterization-based HW-supported rendering

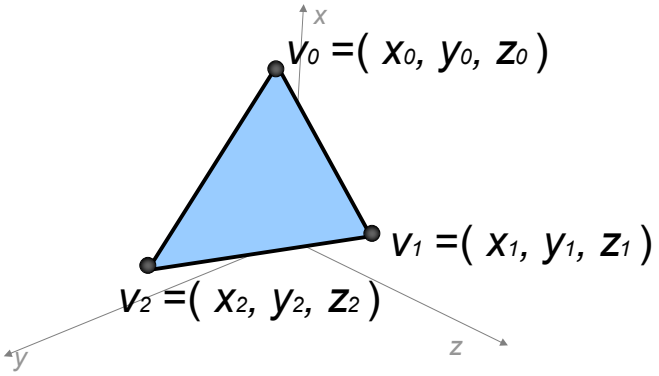
✓ also known as
Transform and Lighting (**T&L**)



55

Rasterization-based rendering (HW supported)

✓ L'intera scena è composta da triangoli 3D



56

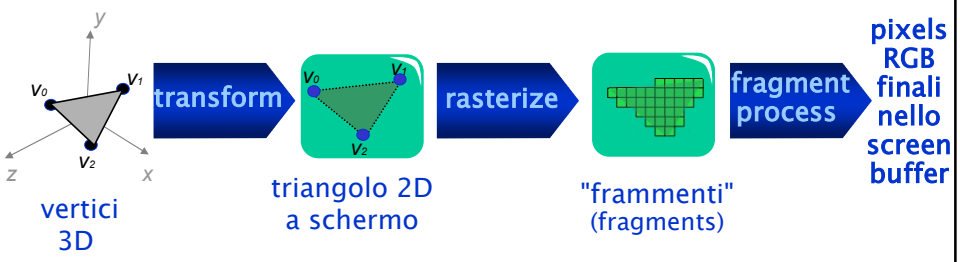
Anche detto T&L: Transform & Lighting

✓ *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

57

Rasterization-Based Rendering: il pipeline



58

Rasterization-based rendering

- ✓ Input: insieme di vertici che formano triangoli
- ✓ L'algoritmo a pipeline (catena di montaggio) consiste in diverse fasi:

1. Fase per vertice: ogni vertice viene portato in una posizione 2D sullo schermo
   - ⇒ Si tratta di una "transformazione 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)

60

Considerazioni generali sul Pipeline

- ✓Ogni fase del pipeline avviene in parallelo
  - ⇒Ogni vertice, triangolo, frammento può essere processato in contemporanea con ciascun'altro
- ✓Le 3 fasi sono in cascata
- ✓Il pipeline va tanto veloce quanto la sua fase più lenta
  - ⇒Detta il collo di bottiglia
  - ⇒Se il collo di bottiglia avviene per vertice, l'applicazione si dice «transform limited»
  - ⇒Se avviene per frammento, l'applicazione si dice «fill limited»

61

THE GREAT DEBATE

RASTERIZATION

VS

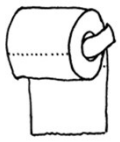
RAY-TRACING

62

Rendering Algorithms Paradigms

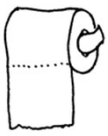
RAY-TRACING

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



RASTERIZATION BASED:

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



63

Vantaggi del ray-tracing

- ✓ E' una classe di algoritmi concettualmente semplice
- ✓ E' facile ottenere effetti visuali complicati
- ✓ Simula in modo abbastanza accurato la sintesi di un'immagine da parte di (ad esempio) una macchina fotografica (pur fra molte semplificazioni)
- ✓ E' possibile renderizzare direttamente qualsiasi «primitiva di rendering»: basta implementare la funzione di computo di intersezione con un raggio
- ✓ Parallelismo implicito:
  ogni pixel può essere implementato in parallelo

64

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 su una buseness card :-P !
(by Paul Heckbert)

65

Vantaggi degli approcci basati su rasterizzazione

- ✓ Complessità lineare con numero di primitive
- ✓ Per ogni primitiva, si processano solo i pixel coinvolti
- ✓ E' l'algoritmo per il quale le GPU sono state pensate, dunque supporto HW garantito

66

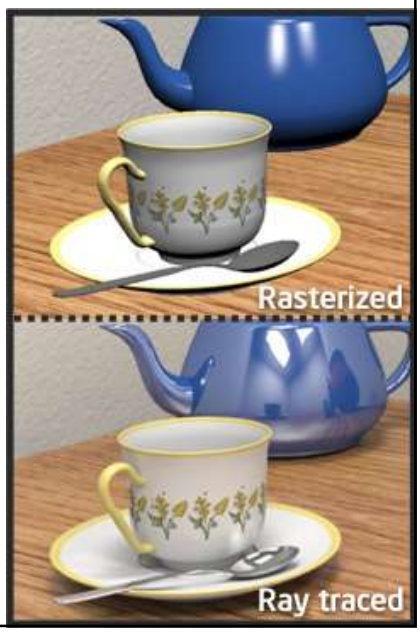
Visione storica / tradizionale

- ✓Ray-tracing
  - ⇒Lento, ma accurato
  - ⇒Gli algoritmi standard per il rendering offline
  - ⇒Per esempio, usato nella movie industry
  - ⇒Alcuni software ray-tracers noti:
    POV-ray, renderman (pixar), YafaRay, Miztuba
- ✓Rasterization based
  - ⇒Veloce, ma approssimato
  - ⇒Gli algoritmi standard per il rendering online
  - ⇒Per esempio, usato nella game industry
  - ⇒Alcuni API noti per rasterization based:
    OpenGL, DirectX, Metal, Vulkan

67

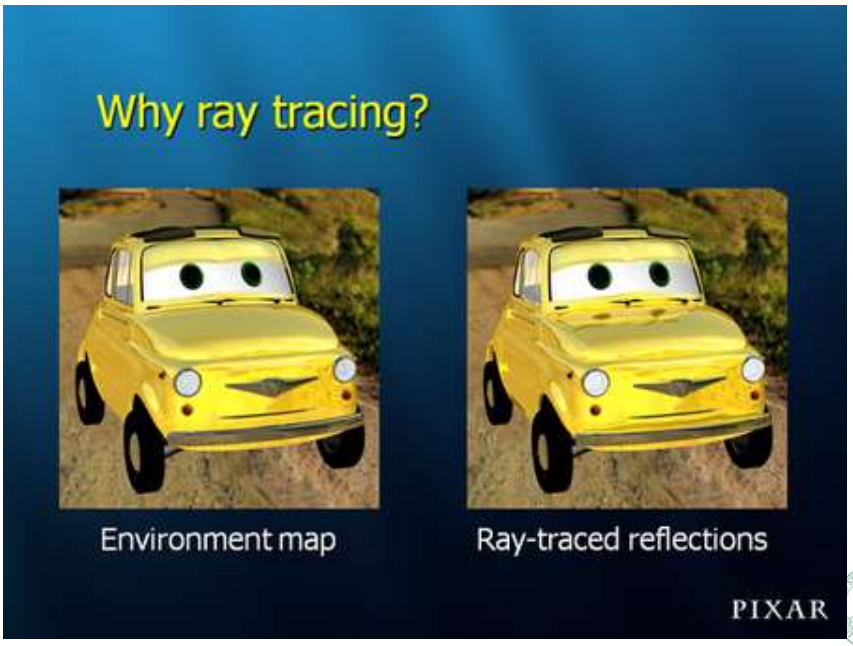
Il dibattito dura da decenni

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



69

Il dibattito dura da decenni



70

La realtà: raytracing non è necessariamente lento

✓perché:

⇒È intrinsecamente parallelizzabile
- alto livello di **parallelismo implicito**
- quindi, implementabile con HW parallelo

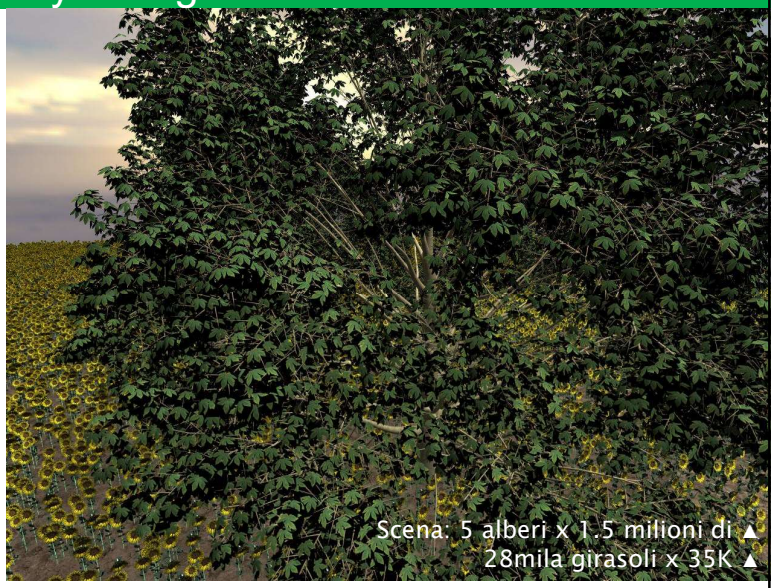
⇒Le primitive possono essere forme complesse
- Questo riduce il numero di primitive necessarie a comporre una data virtuale

⇒Ottimizzazioni: usando apposite strutture dati, è possibile ridurre il numero di primitive da testare per ogni raggio
- Strutture di **indicizzazione spaziale**
- Nota: è più arduo per scene animate
- Le complessità degli algoritmi di Ray-Tracing può essere resa **sublineare** col numero di primitive

72

Real time raytracing: esiste

Un precursore → (su HW specializzato per questo caso)



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

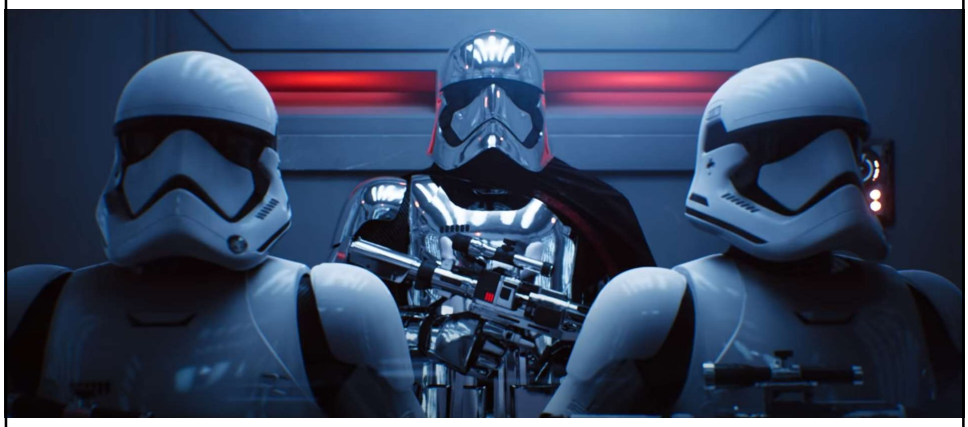
73

Real time raytracing: esiste

Unity Real time Raytracing (2019)

74

Real time raytracing: esiste



Unreal + Nvidia Real-time Raytracing

75

La realtà: rasterization based supporta effetti di luce compessi

- ✓ Esistono molti algoritmi basati sul rasterization per simulare o approssimare:
  - ⇒ Ombre portate
  - ⇒ Diffrazioni
  - ⇒ Rifelssioni speculari
  - ⇒ Riflessioni multiple (illuminazione «globale»)
  - ⇒ Rifrazioni
- ✓ Si tratta di algorimi specializzati per ciascuno di questi effetti

76

La realtà: rasterization based supporta effetti di luce compessi




77

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)

79