Rendering Paradigms

SciViz 2018/2019

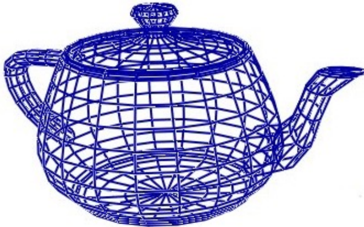
Paolo Cignoni

(thx to Marco Tarini for part of the slides)

3D Rendering

```
Rim:
  { 102, 103, 104, 105,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,  14,  15 }
Body:
  {  12,  13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,  26,  27 }
  {  24,  25,  26,  27,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,  39,  40 }
Lid:
  {  96,  96,  96,  96,  97,  98,  99, 100, 101, 101, 101, 101,   0,   1,   2,   3 }
  {   0,   1,   2,   3, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117 }
Handle:
  {  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,  52,  53,  54,  55,  56 }
  {  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,  28,  65,  66,  67 }
Spout:
  {  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,  78,  79,  80,  81,  82,  83 }
  {  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,  91,  92,  93,  94,  95 }

Vertices:

  {  0.2000,  0.0000, 2.70000 }, {  0.2000, -0.1120, 2.70000 }, {  0.1120, -0.2000, 2.70000 },
  {  0.0000, -0.2000, 2.70000 }, {  1.3375,  0.0000, 2.53125 }, {  1.3375, -0.7490, 2.53125 },
  {  0.7490, -1.3375, 2.53125 }, {  0.0000, -1.3375, 2.53125 }, {  1.4375,  0.0000, 2.53125 },
  {  1.4375, -0.8050, 2.53125 }, {  0.8050, -1.4375, 2.53125 }, {  0.0000, -1.4375, 2.53125 },
  {  1.5000,  0.0000, 2.40000 }, {  1.5000, -0.8400, 2.40000 }, {  0.8400, -1.5000, 2.40000 },
  {  0.0000, -1.5000, 2.40000 }, {  1.7500,  0.0000, 1.87500 }, {  1.7500, -0.9800, 1.87500 },
  {  0.9800, -1.7500, 1.87500 }, {  0.0000, -1.7500, 1.87500 }, {  2.0000,  0.0000, 1.35000 },
  {  2.0000, -1.1200, 1.35000 }, {  1.1200, -2.0000, 1.35000 }, {  0.0000, -2.0000, 1.35000 },
  {  2.0000,  0.0000, 0.90000 }, {  2.0000, -1.1200, 0.90000 }, {  1.1200, -2.0000, 0.90000 },
  {  0.0000, -2.0000, 0.90000 }, { -2.0000,  0.0000, 0.90000 }, {  2.0000,  0.0000, 0.45000 },
  {  2.0000, -1.1200, 0.45000 }, {  1.1200, -2.0000, 0.45000 }, {  0.0000, -2.0000, 0.45000 },
  {  1.5000,  0.0000, 0.22500 }, {  1.5000, -0.8400, 0.22500 }, {  0.8400, -1.5000, 0.22500 },
  {  0.0000, -1.5000, 0.22500 }, {  1.5000,  0.0000, 0.15000 }, {  1.5000, -0.8400, 0.15000 },
  {  0.8400, -1.5000, 0.15000 }, {  0.0000, -1.5000, 0.15000 }, { -1.6000,  0.0000, 2.02500 },
  { -1.6000, -0.3000, 2.02500 }, { -1.5000, -0.3000, 2.25000 }, { -1.5000,  0.0000, 2.25000 },
  { -2.3000,  0.0000, 2.02500 }, { -2.3000, -0.3000, 2.02500 }, { -2.5000, -0.3000, 2.25000 },
  { -2.5000,  0.0000, 2.25000 }, { -2.7000,  0.0000, 2.02500 }, { -2.7000, -0.3000, 2.02500 },
  { -3.0000, -0.3000, 2.25000 }, { -3.0000,  0.0000, 2.25000 }, { -2.7000,  0.0000, 1.80000 },
  { -2.7000, -0.3000, 1.80000 }, { -3.0000, -0.3000, 1.80000 }, { -3.0000,  0.0000, 1.80000 },
  { -2.7000,  0.0000, 1.57500 }, { -2.7000, -0.3000, 1.57500 }, { -3.0000, -0.3000, 1.35000 },
  { ...0000,  0.0000, 1.35000 }, { -2.5000,  0.0000, 1.12500 }, { -2.5000, -0.3000, 1.12500 },
```

3D model
- 3D coordinates
- primitives...



3D Rendering

raster image
3 channels RGB
8 bits x channel

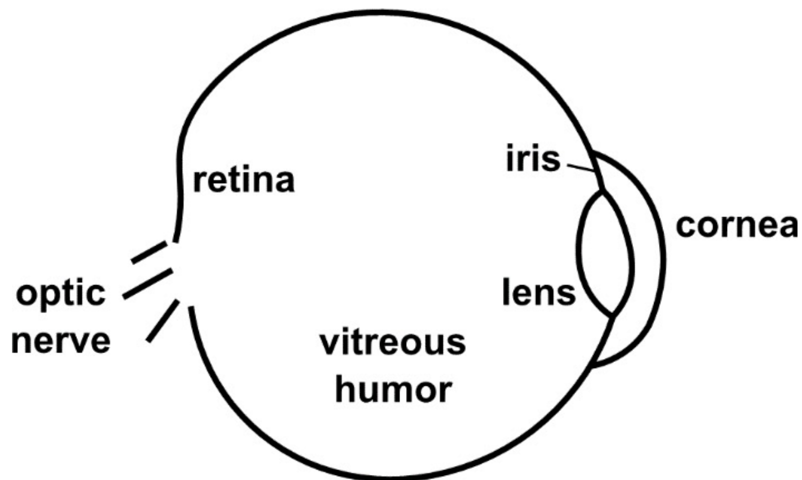
3D Rendering

Rendering algorithms

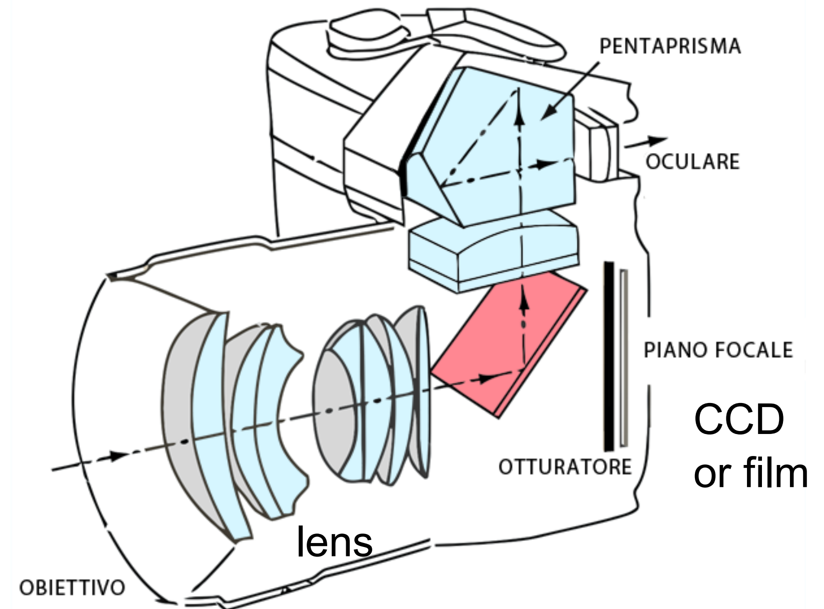
- To visualize 3D scenes we need to transform them into a synthetic image that can be displayed on the screen
- Many different algorithms exist to transform a 3D scene into a raster image, we briefly overview two families:
  - Raytracing algorithms
  - Rasterization-based algorithms

Modelling the picture making apparatus

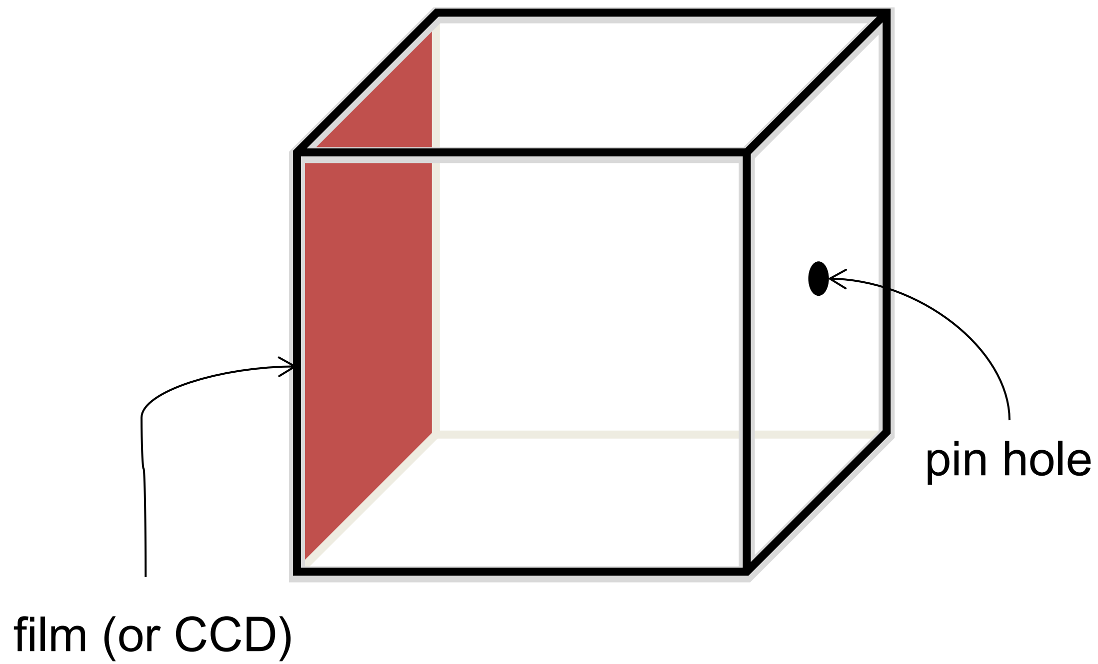
- Human Visual System



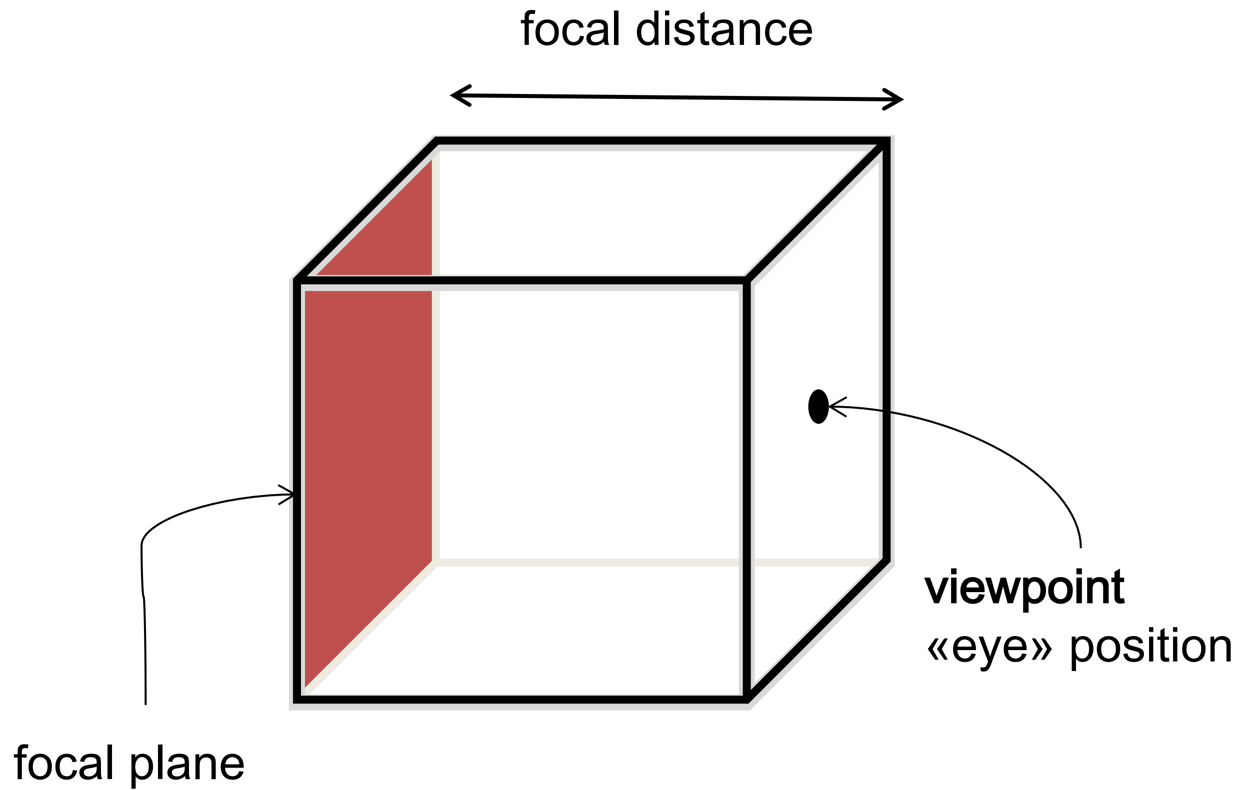
- Camera



Pin-Hole camera

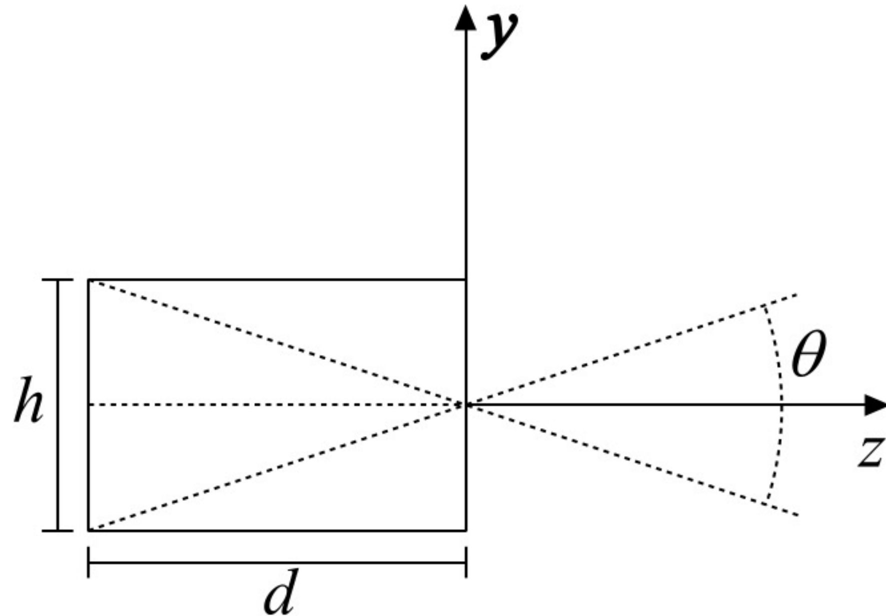


Pin-Hole camera



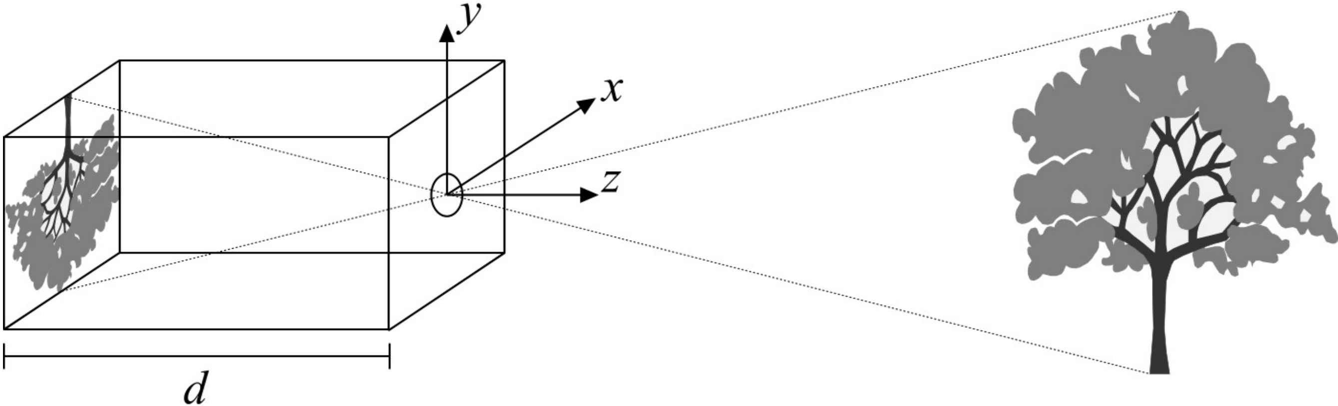
Pin Hole Camera

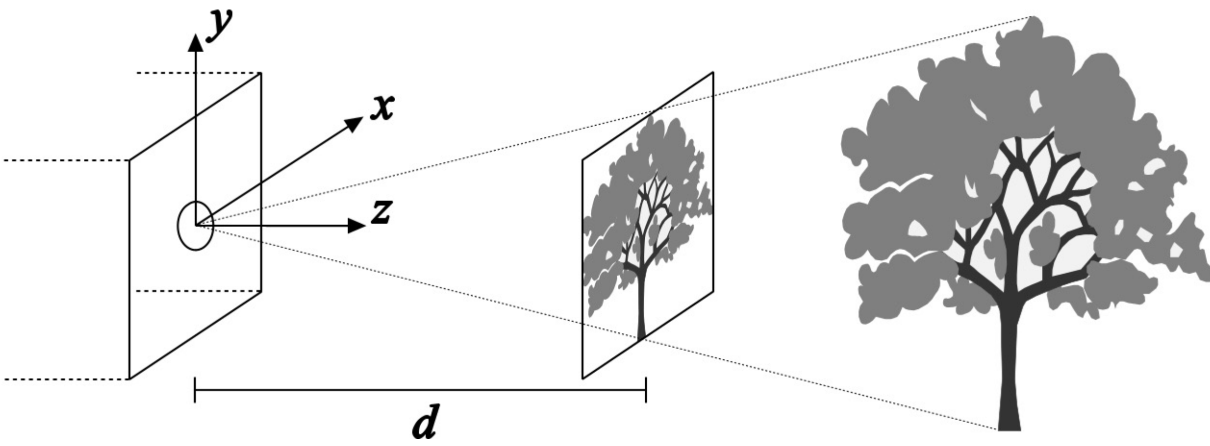
The virtual camera consists of a parallelepiped in which the front face has an infinitesimal small hole (pinhole camera) and on the back side the images are formed;
the angle θ of view can be modified by varying the ratio between the focal distance *d* and the size *h* of the image plane



Pin Hole Camera

By convention we assume the existence of an image plane between the scene and the projection center





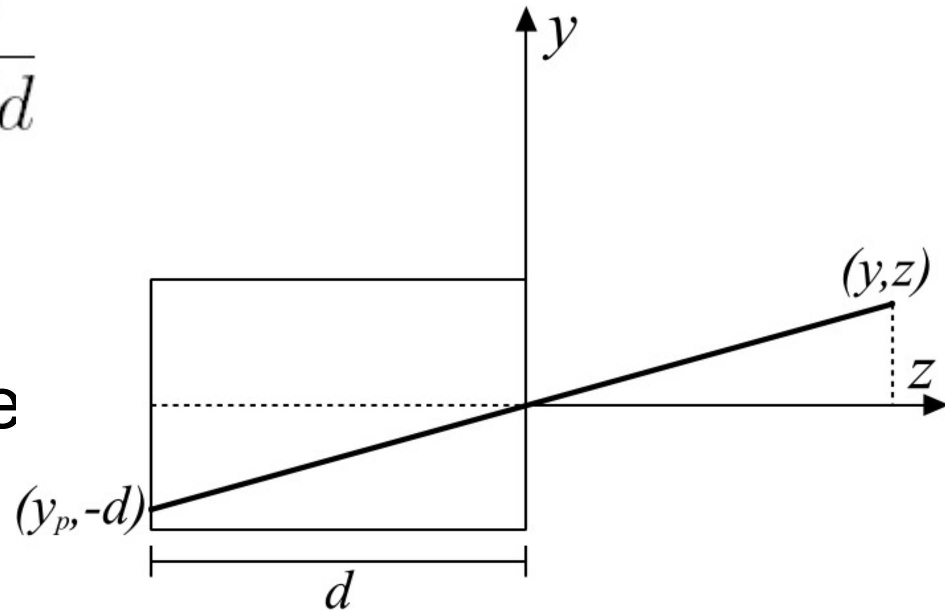
Pin Hole Camera

La metafora utilizzata per descrivere le relazioni scena/osservatore è quella della macchina fotografica virtuale (*synthetic camera*).

Il generico punto $P=(x,y,z)$ della scena ha coordinate $P_p=(x_p,y_p,-d)$ sul piano immagine. Dove:

$$x_p = -\frac{x}{z/d} \quad y_p = -\frac{y}{z/d}$$

La trasformazione non
è lineare, non è affine
non è reversibile



3D Viewing

Il processo di formazione dell'immagine di sintesi in 3D consta di una sequenza di operazioni:

- Definizione della trasformazione di proiezione (il modo di mappare informazioni 3D su un piano immagine 2D);
- Definizione dei parametri di vista (punto di vista, direzione di vista, etc.);
- Clipping in 3D (i parametri di vista individuano un volume di vista; occorre rimuovere le parti della scena esterne a tale volume);
- Trasformazione di proiezione e visualizzazione della scena (con trasformazione "window-to-viewport" finale).

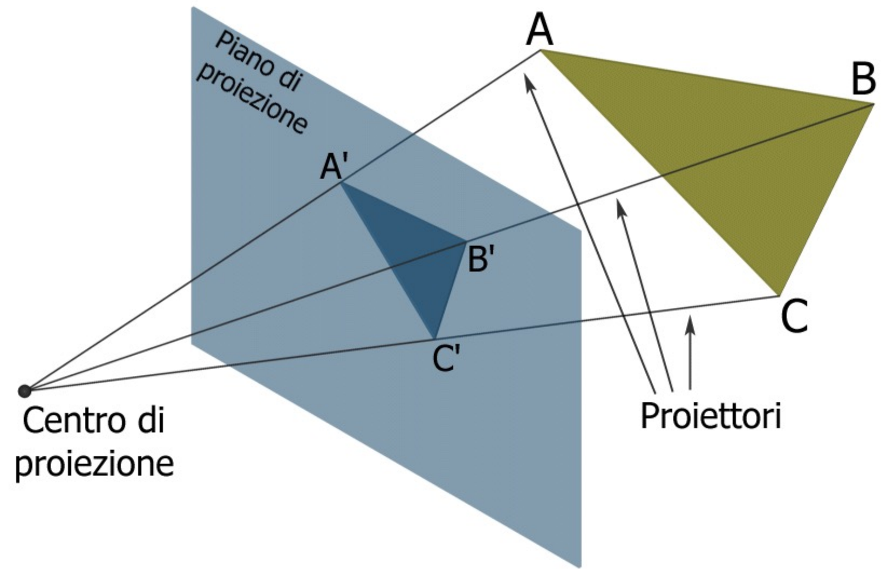
Trasformazioni di proiezione

Si dice *proiezione* una trasformazione geometrica con il dominio in uno spazio di dimensione n ed il codominio in uno spazio di dimensione $n-1$ (o minore):

In computer graphics le trasformazioni di proiezione utilizzate sono quelle dallo spazio 3D (il mondo dell'applicazione) al 2D (la superficie del dispositivo di output)

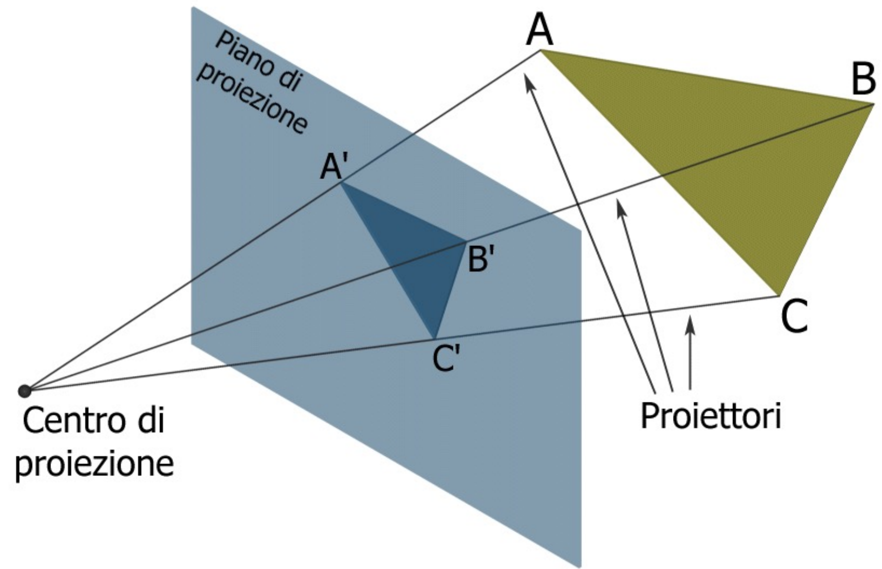
Trasformazioni di proiezione

Da un punto di vista geeometrico, la proiezione è definita per mezzo di un insieme di rette di proiezione (i *proiettori*) aventi origine comune in un centro di proiezione, passanti per tutti i punti dell'oggetto da proiettare ed intersecanti un piano di proiezione.



Trasformazioni di proiezione

La proiezione di un segmento è a sua volta un segmento; non è quindi necessario calcolare i proiettori di tutti i punti di una scena, ma solo quelli relativi ai vertici delle primitive che la descrivono.



Trasformazioni di proiezione

Le proiezioni caratterizzate da:
proiettori rettilinei (anziché curve generiche);
proiezione giacente su un piano (anziché su una superficie generica)
prendono il nome di *proiezioni geometriche piane.*

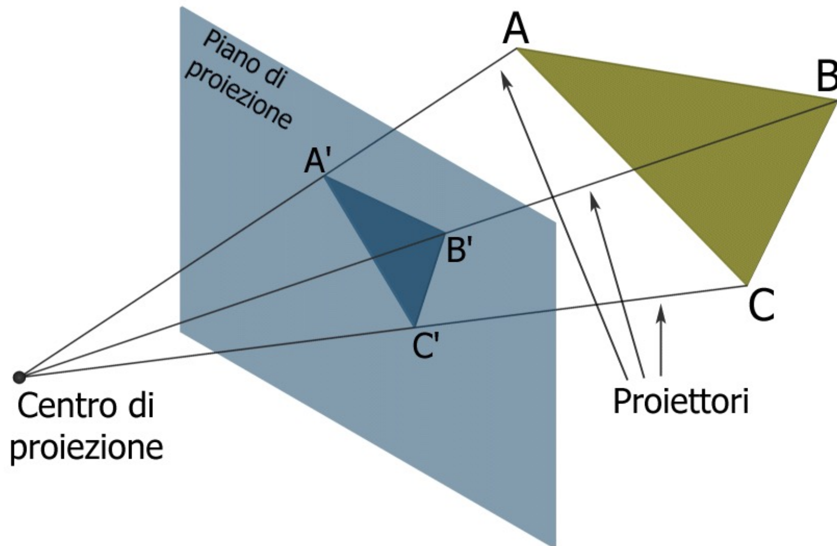
Molte delle proiezioni cartografiche non sono proiezioni geometriche piane

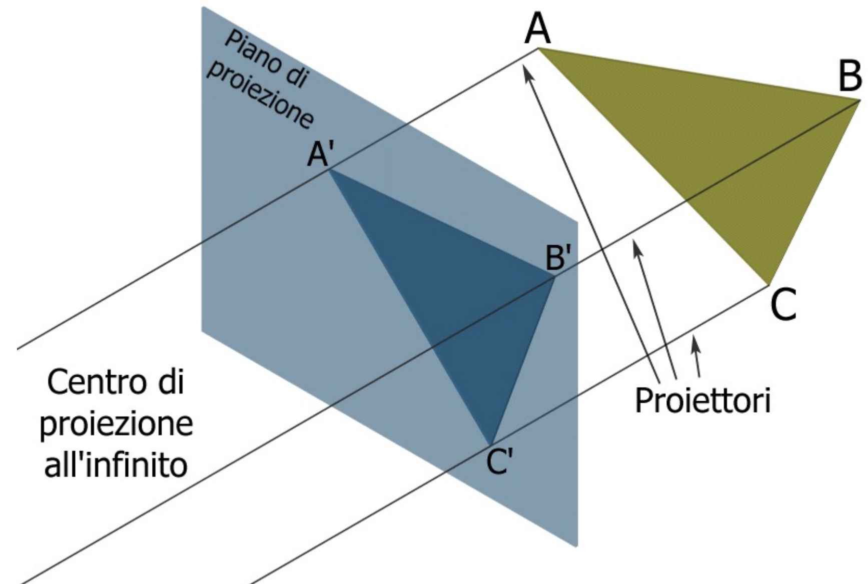
Trasformazioni di proiezione

Le proiezioni geometriche piane si classificano in:

Proiezioni prospettiche (distanza finita tra il centro ed il piano di proiezione);

Proiezioni parallele (distanza infinita tra il centro ed il piano di proiezione).



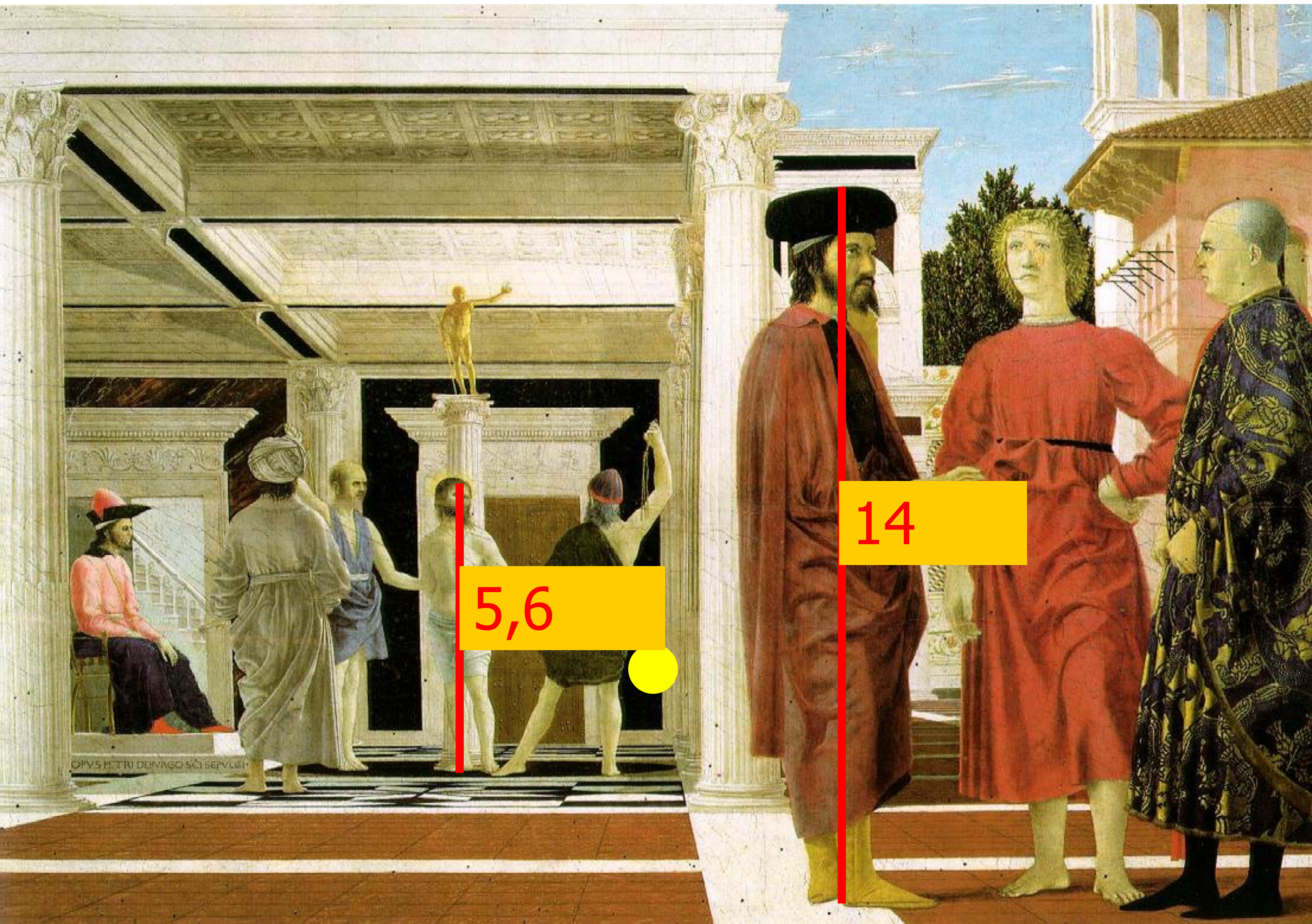


Trasformazioni di proiezione

Le proiezioni parallele prendono il nome dai proiettori che sono, appunto, tra loro tutti paralleli;
Mentre per una proiezione prospettica si specifica un centro di proiezione, nel caso di proiezione parallela si parla di direzione di proiezione;

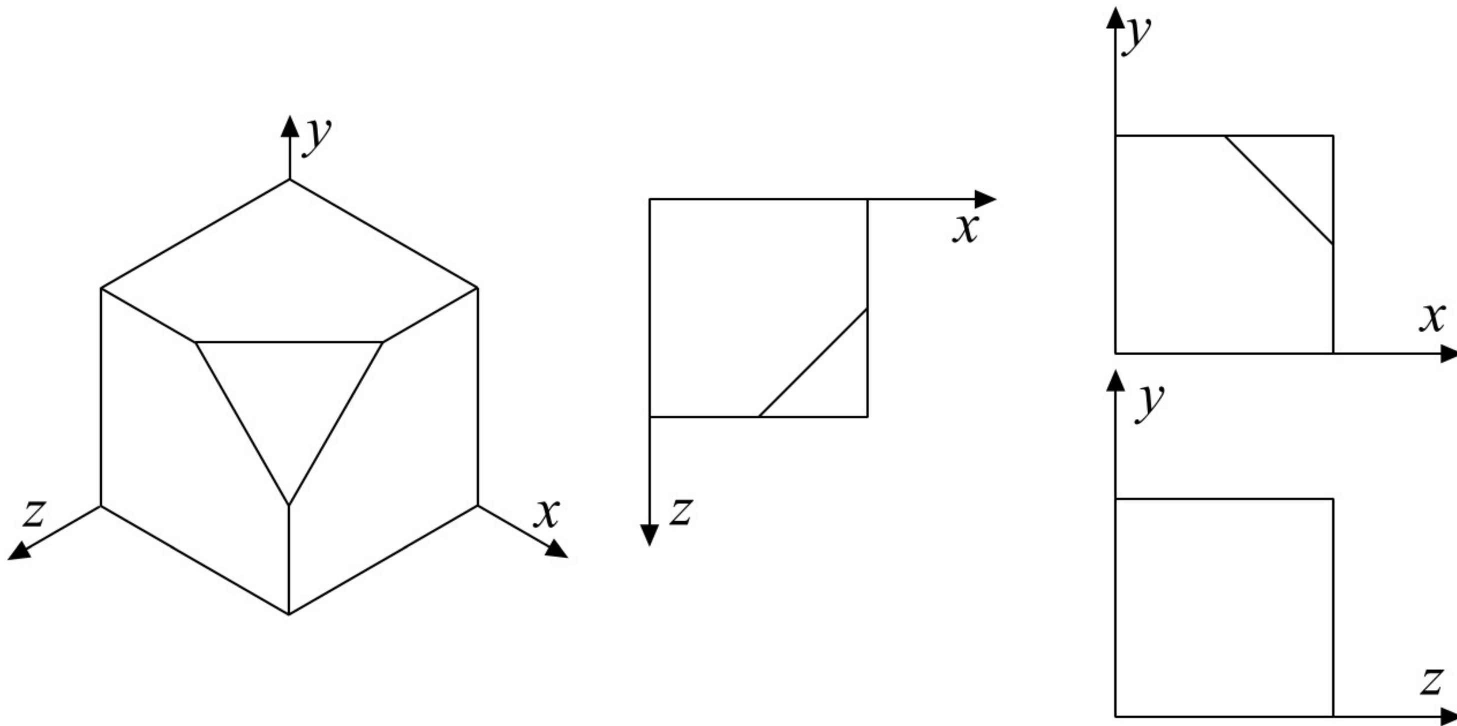
La proiezione prospettica è più realistica della parallela in quanto riproduce la visione reale (gli oggetti appaiono di dimensione decrescenti via via che ci si allontana dall'osservatore);

“La flagellazione” di Piero della Francesca (1469)

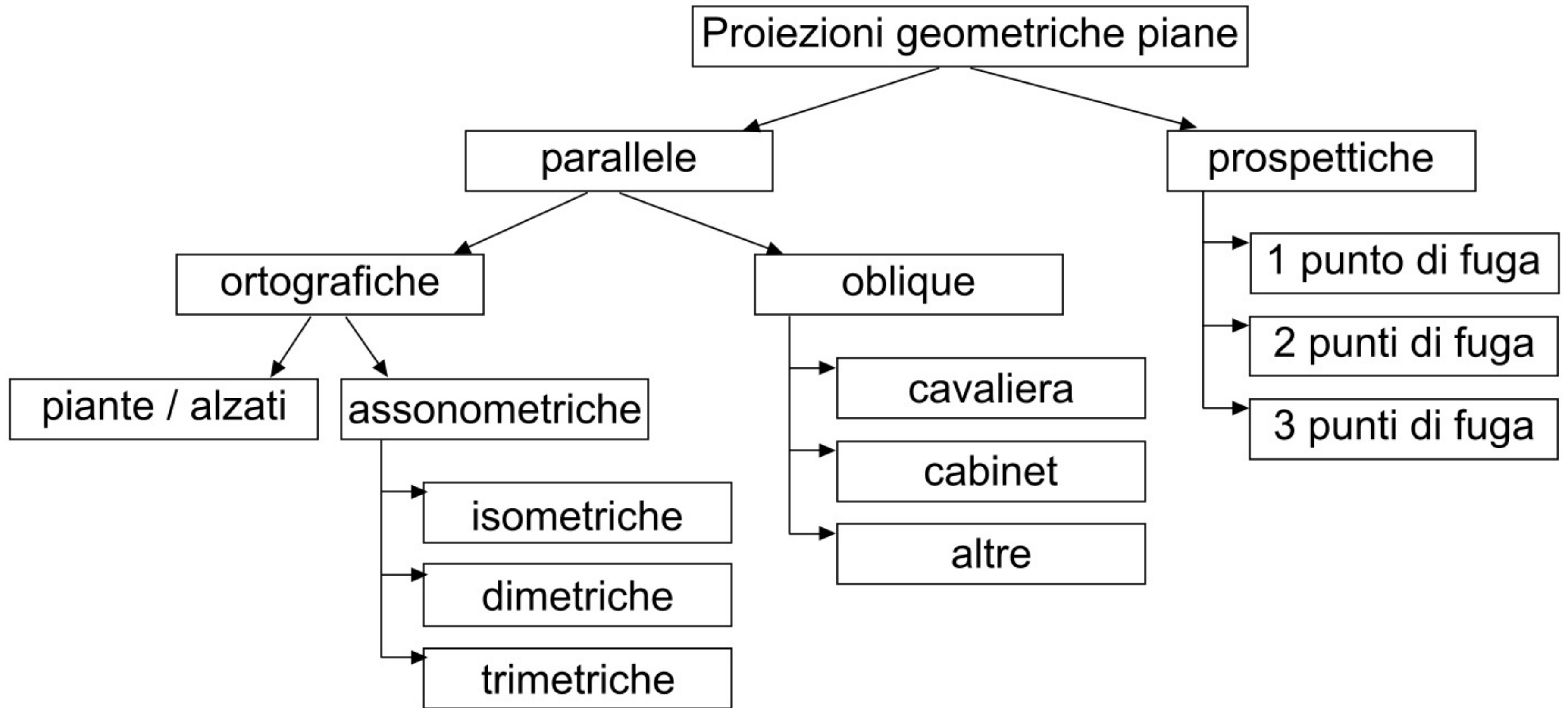


Trasformazioni di proiezione

Le <u>proiezioni parallele</u> sono molto utili quando si voglia far sì che linee parallele nel modello tridimensionale rimangano tali nella proiezione (ad esempio per effettuare misure sulla proiezione);

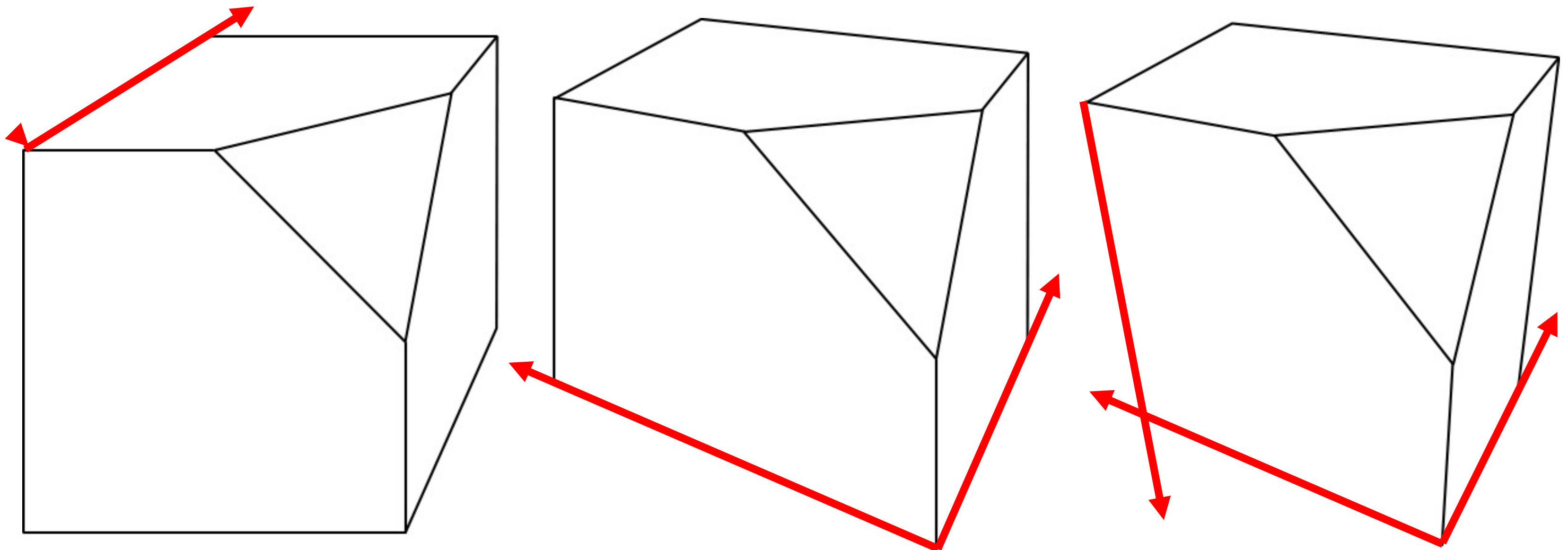


Trasformazioni di proiezione: classificazione



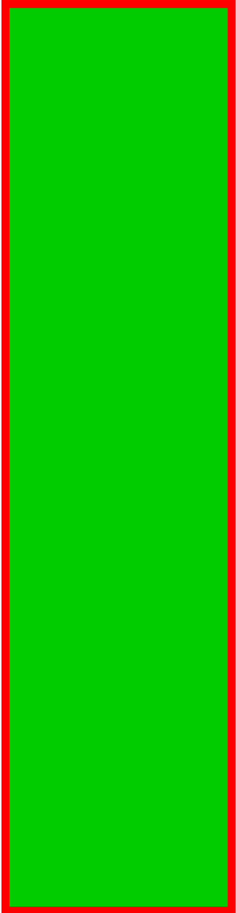
Le proiezioni prospettiche

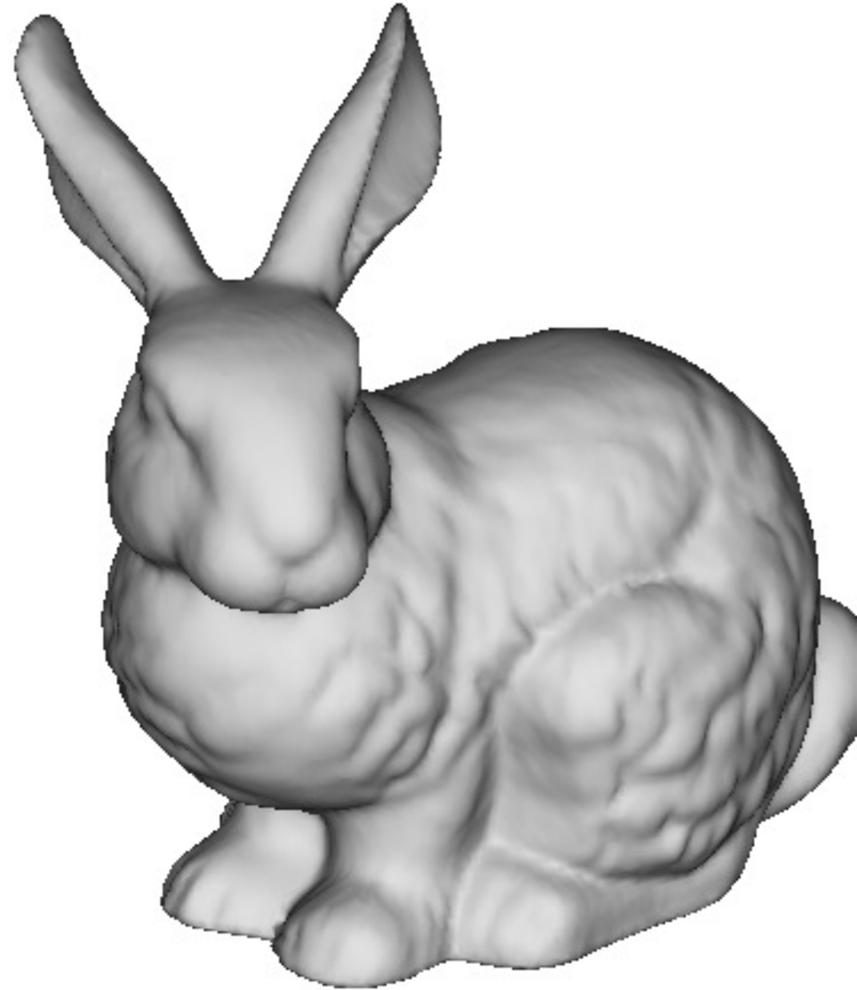
Nella proiezione prospettica, ogni insieme di linee parallele (ma non parallele al piano di proiezione) converge in un punto detto <u>punto di fuga;</u>
Se l'insieme di linee è parallelo ad uno degli assi coordinati, il <u>punto di fuga</u> si chiama <u>principale</u> (massimo 3);



La proiezione prospettica

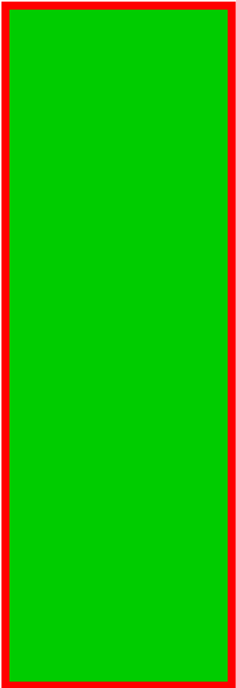
Al variare della distanza focale (d)

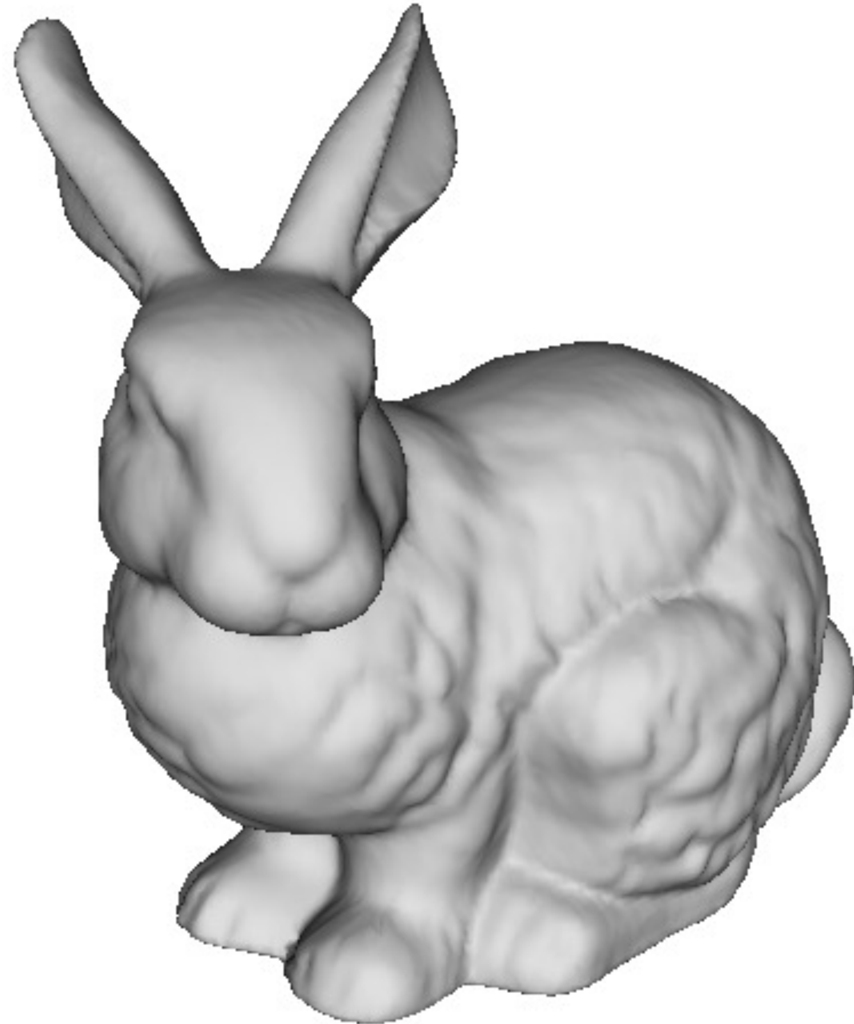




La proiezione prospettica

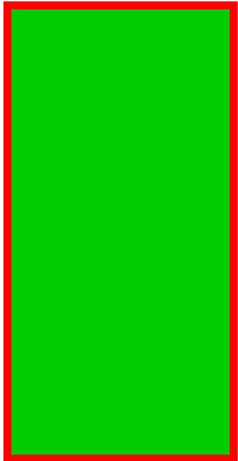
Al variare della distanza focale (d)





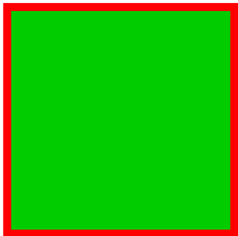
La proiezione prospettica

Al variare della distanza focale (d)

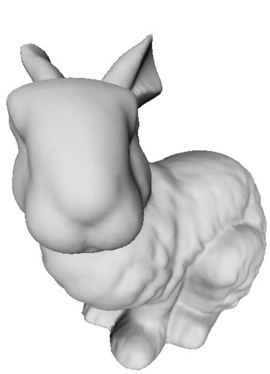
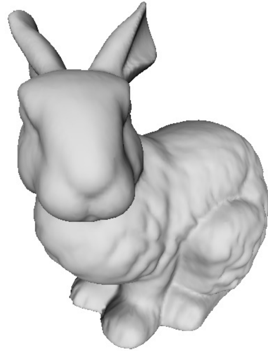
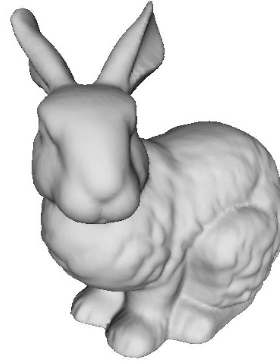
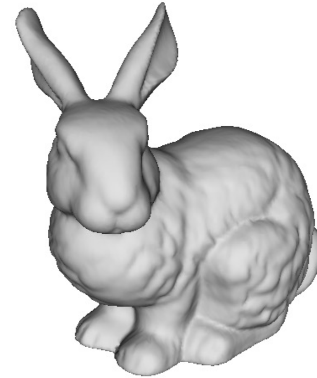
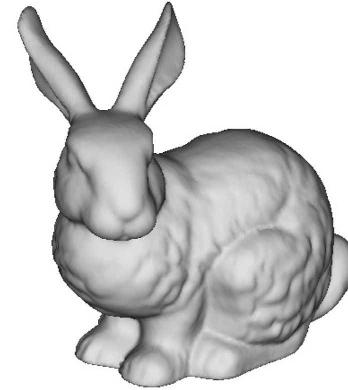


La proiezione prospettica

Al variare della distanza focale (d)



La proiezione prospettica

d piccolo

d grande

d infinito
(p. parallela)

Più distorsione prospettica.
Effetto "fish-eye" (grandangolo)

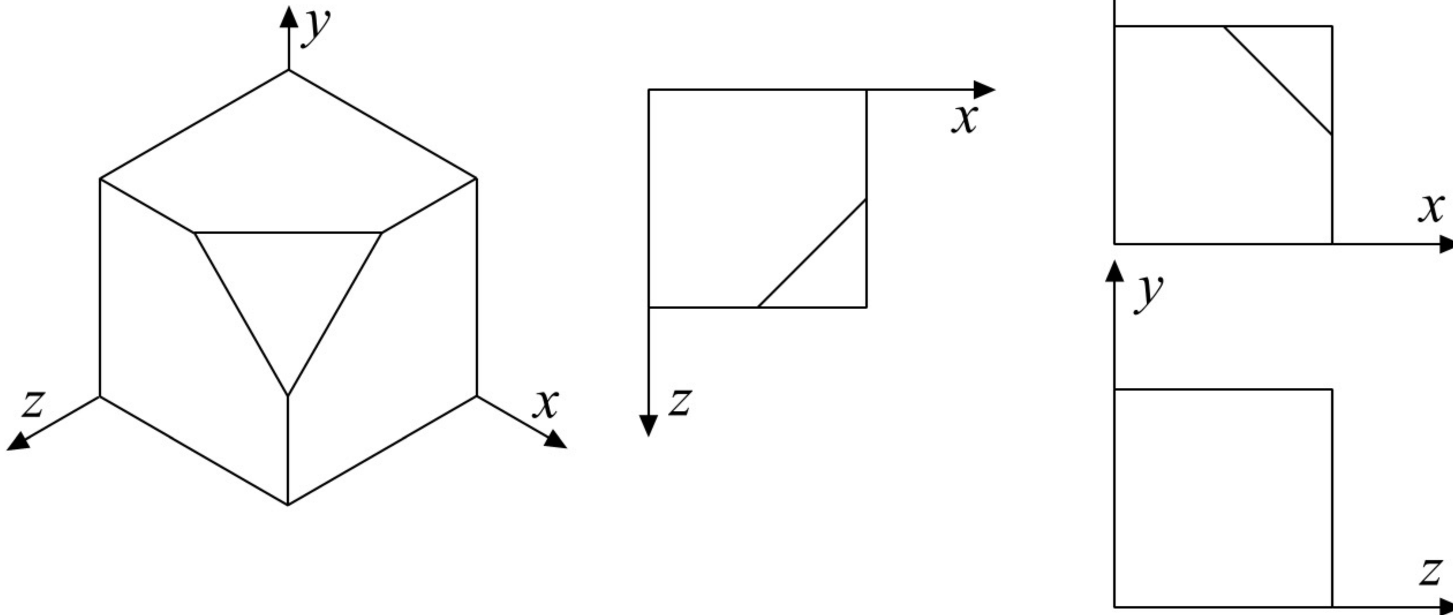
Proporzioni più mantenute
Effetto "zoom"
(eg. vista dal satellite)

Le proiezioni parallele

Le proiezioni parallele si classificano in base alla relazione esistente tra la direzione di proiezione e la normale al piano di proiezione; Se le due direzioni coincidono si parla di proiezioni ortografiche, altrimenti di proiezioni oblique.

Le proiezioni parallele ortografiche

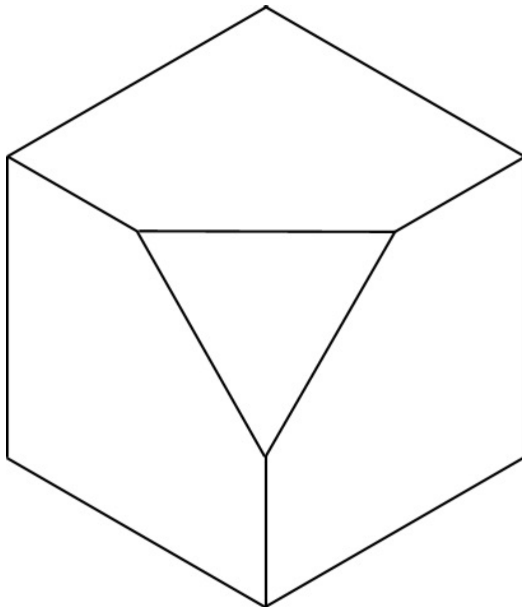
1 – Piante / Alzati: il piano di proiezione è perpendicolare ad uno degli assi cartesiani; le distanze misurate sulla proiezione coincidono con le distanze misurate sul modello 3D.



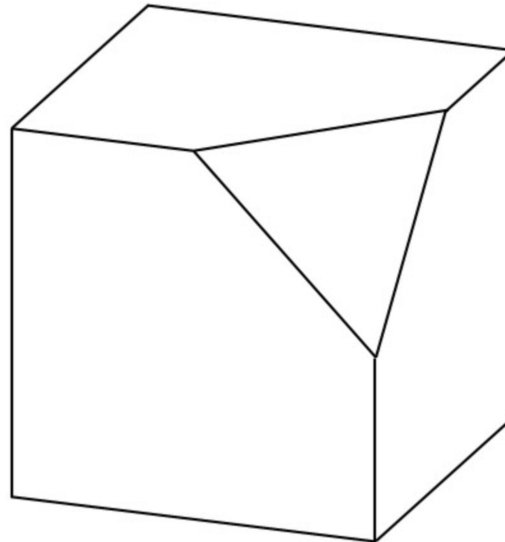
Le proiezioni parallele ortografiche

2 – Assonometriche: il piano di proiezione non è perpendicolare ad alcuno degli assi cartesiani; le distanze misurate sulla proiezione sono diverse dalle distanze reali.

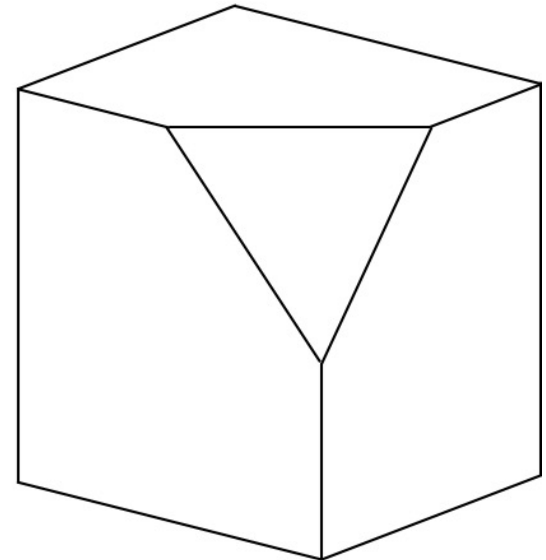
ass. isometrica



ass. dimetrica



ass. trimetrica



Le proiezioni parallele ortografiche

<u>Assonometrica isometrica</u>: la normale al piano di proiezione (direzione di proiezione) forma angoli uguali con i tre assi cartesiani (sono rispettate le proporzioni rispetto al modello 3D);

<u>Assonometrica dimetrica</u>: la normale al piano di proiezione (direzione di proiezione) forma angoli uguali con due degli assi cartesiani (due fattori di scala distinti rispetto al modello 3D);

<u>Assonometrica trimetrica</u>: la normale al piano di proiezione (direzione di proiezione) forma angoli diversi con i tre assi cartesiani (tre fattori di scala distinti rispetto al modello 3D).

Rendering Paradigms

Image Space

vs

Object Space

or

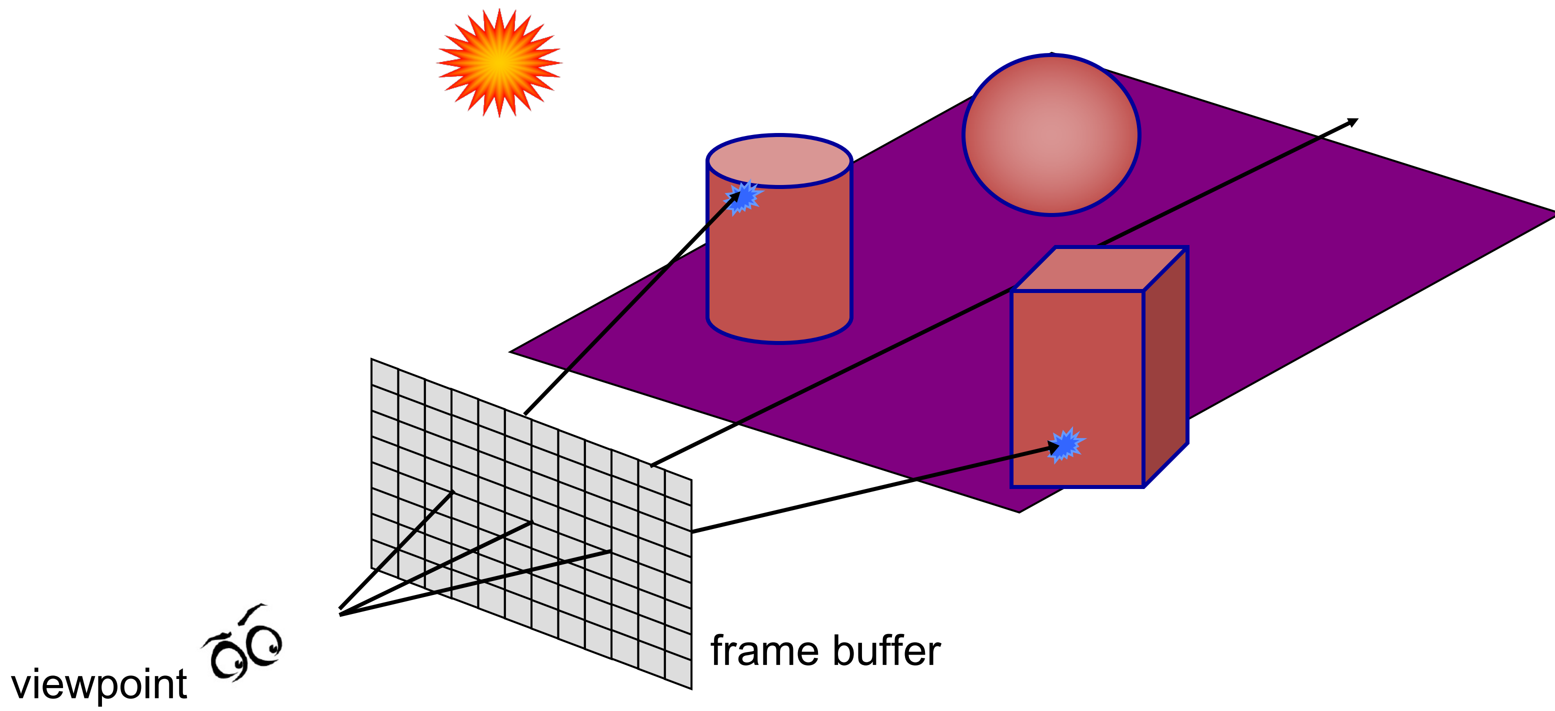
Raytracing

vs

Rasterization

Ray-Tracing

- Main idea: trace light rays backward
  - For each pixel of the screen:
    - Shoot a ray (called primary ray)
    - Find the intersection with the closest primitive

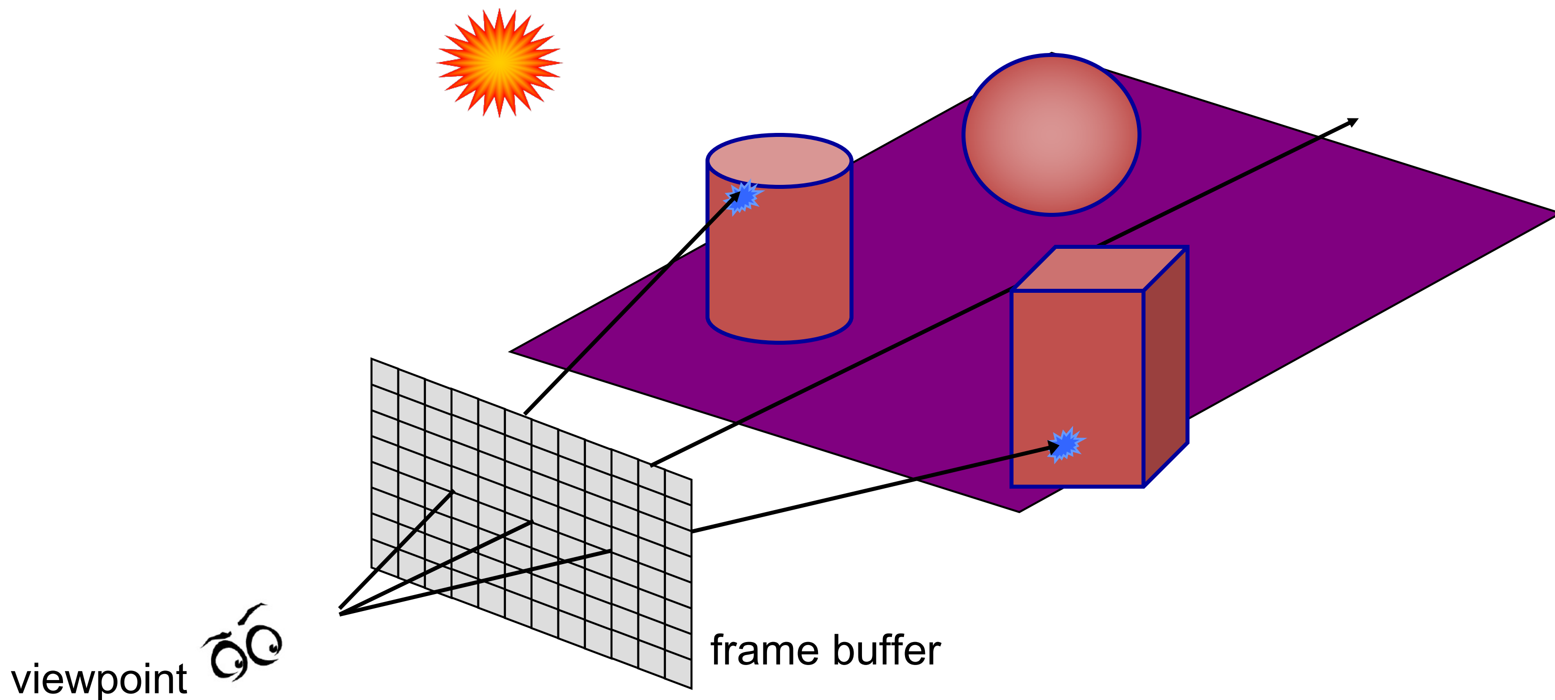


Ray Tracing pseudo-code

```
For each pixel p:
   make a ray r (viewpoint to p)
   for each primitive o in scene:
      find intersect(r,o)
   keep closest intersection oj
   find color of oj at p
```

Ray-Tracing

- Implementing:
  - The core is:
    intersection ray/primitive (it must be extremely optimized)

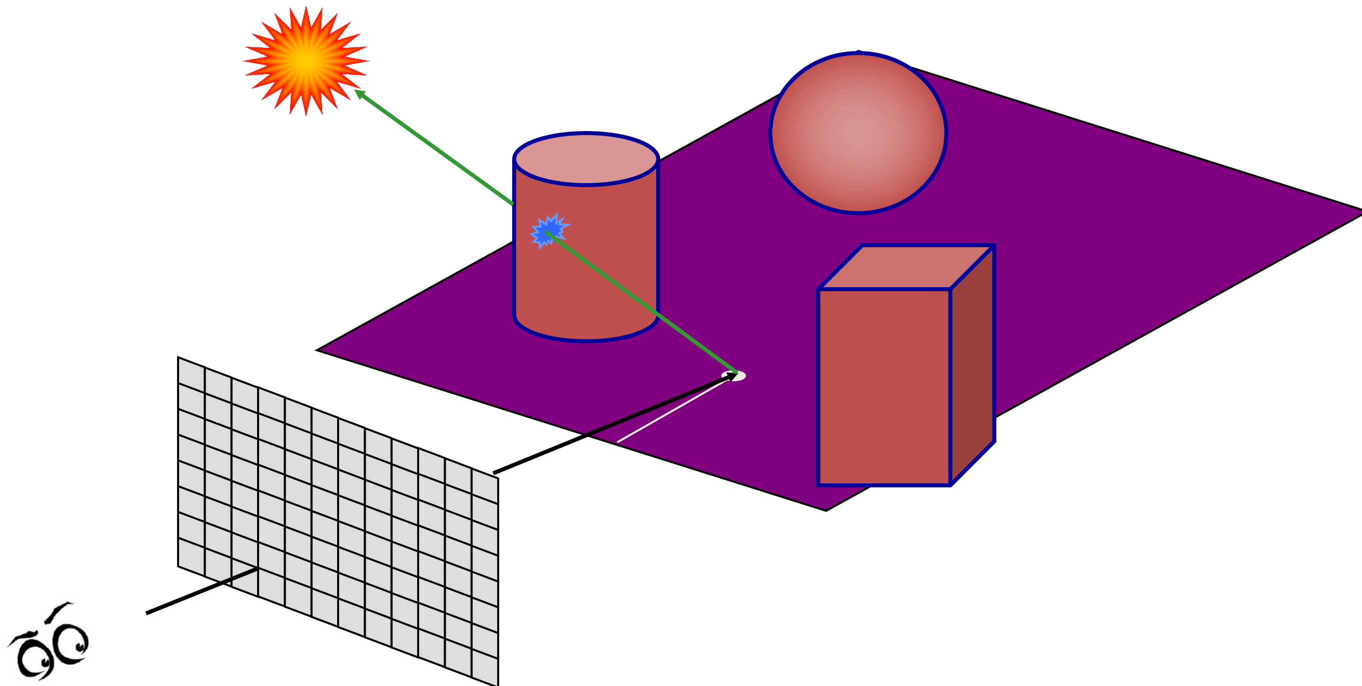


Ray-Tracing

- Easy to handle: shadow (sharp ones)

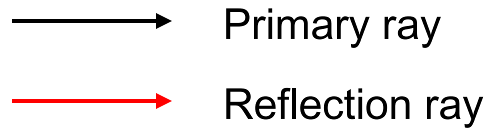
Primary ray

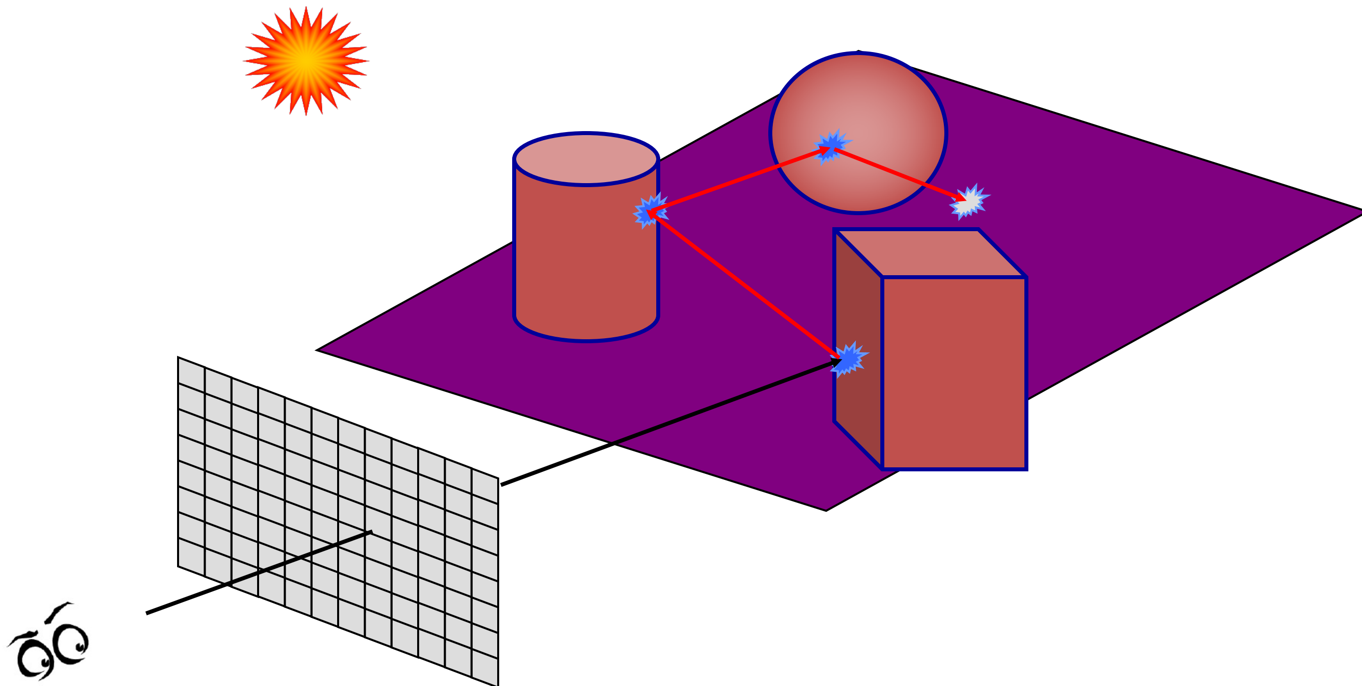
Shadowing ray (if it intersect something it is in shadow)



Ray-Tracing

- Easy to hanlde: multiple specular reflections





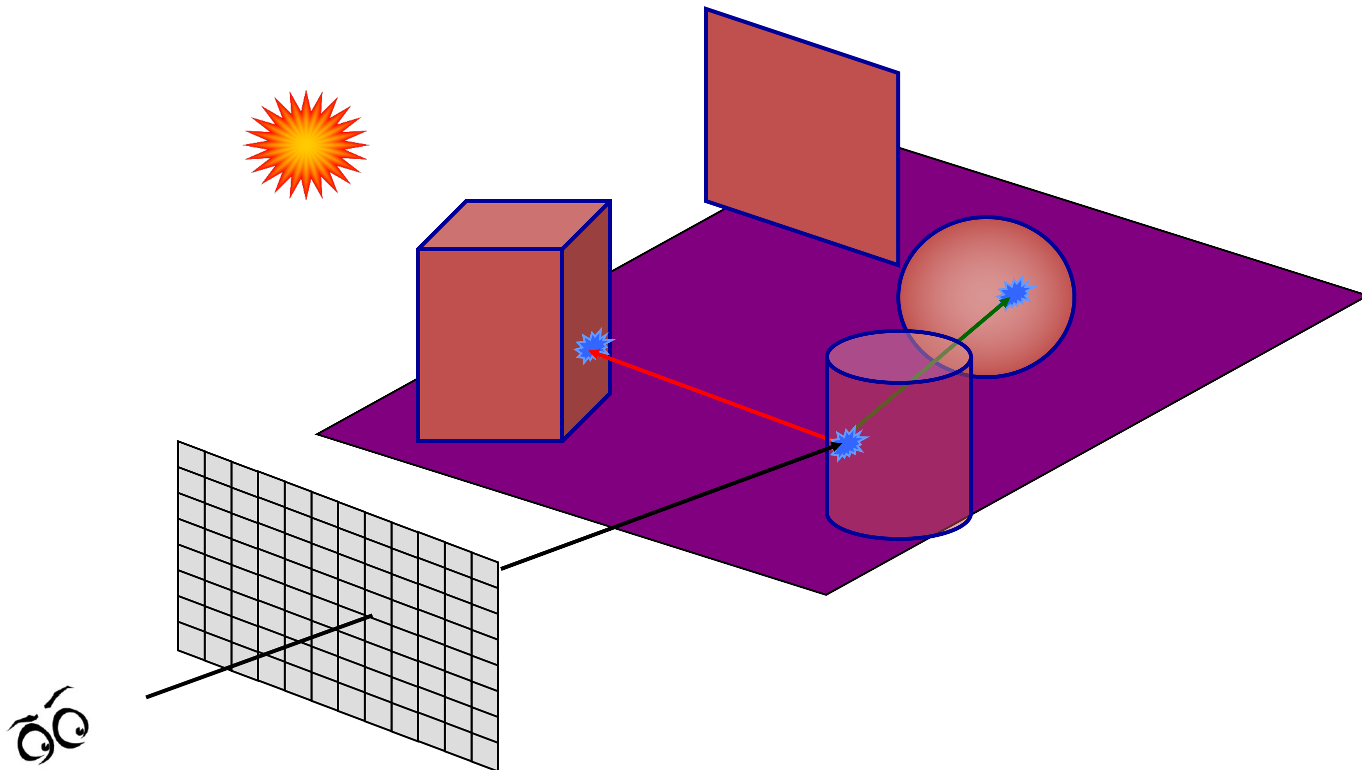
Ray-Tracing

- Facile fare: semitrasparenze con rifrazioni

raggio primario

raggio di riflessione

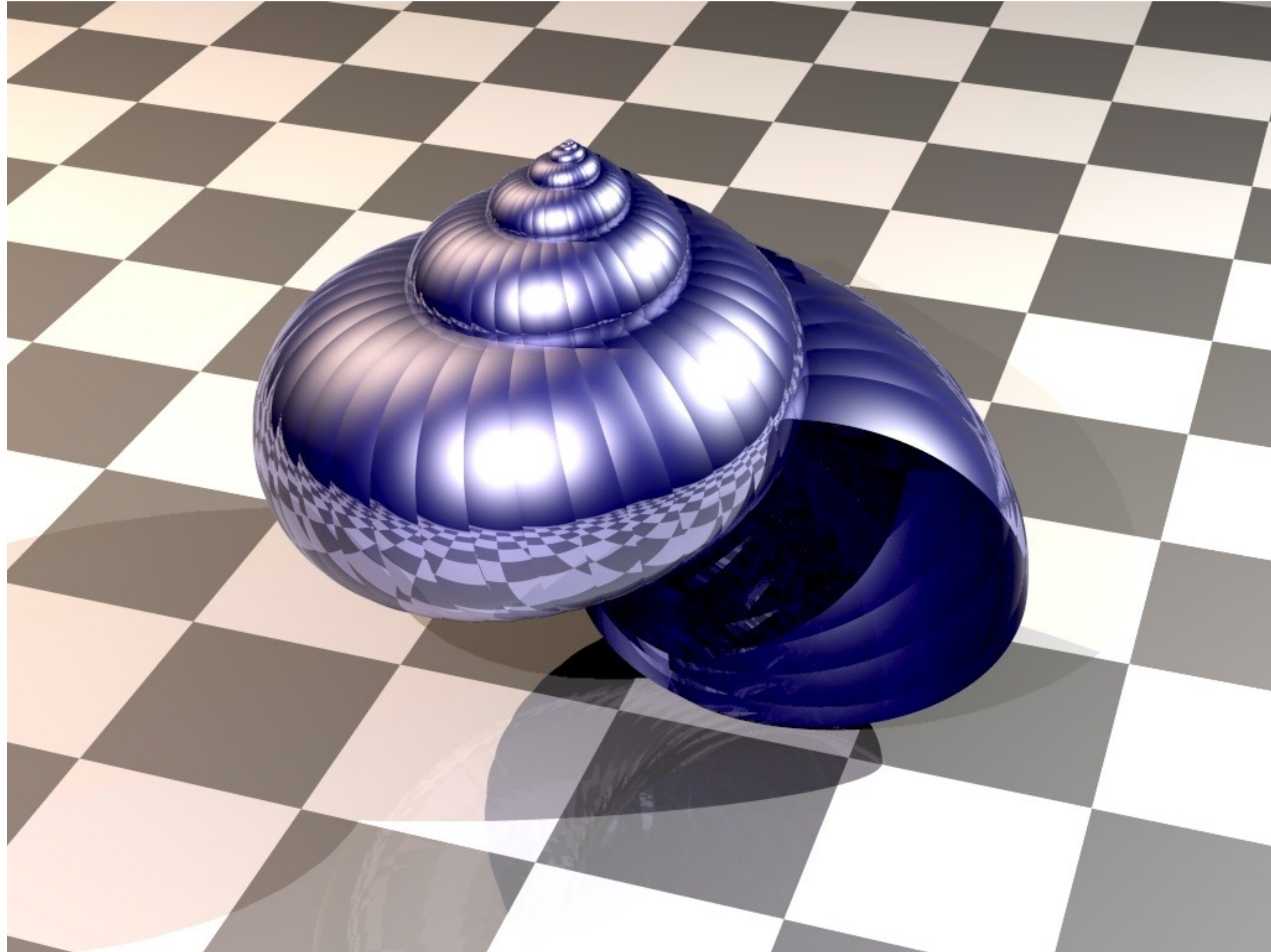
raggio di rifrazione



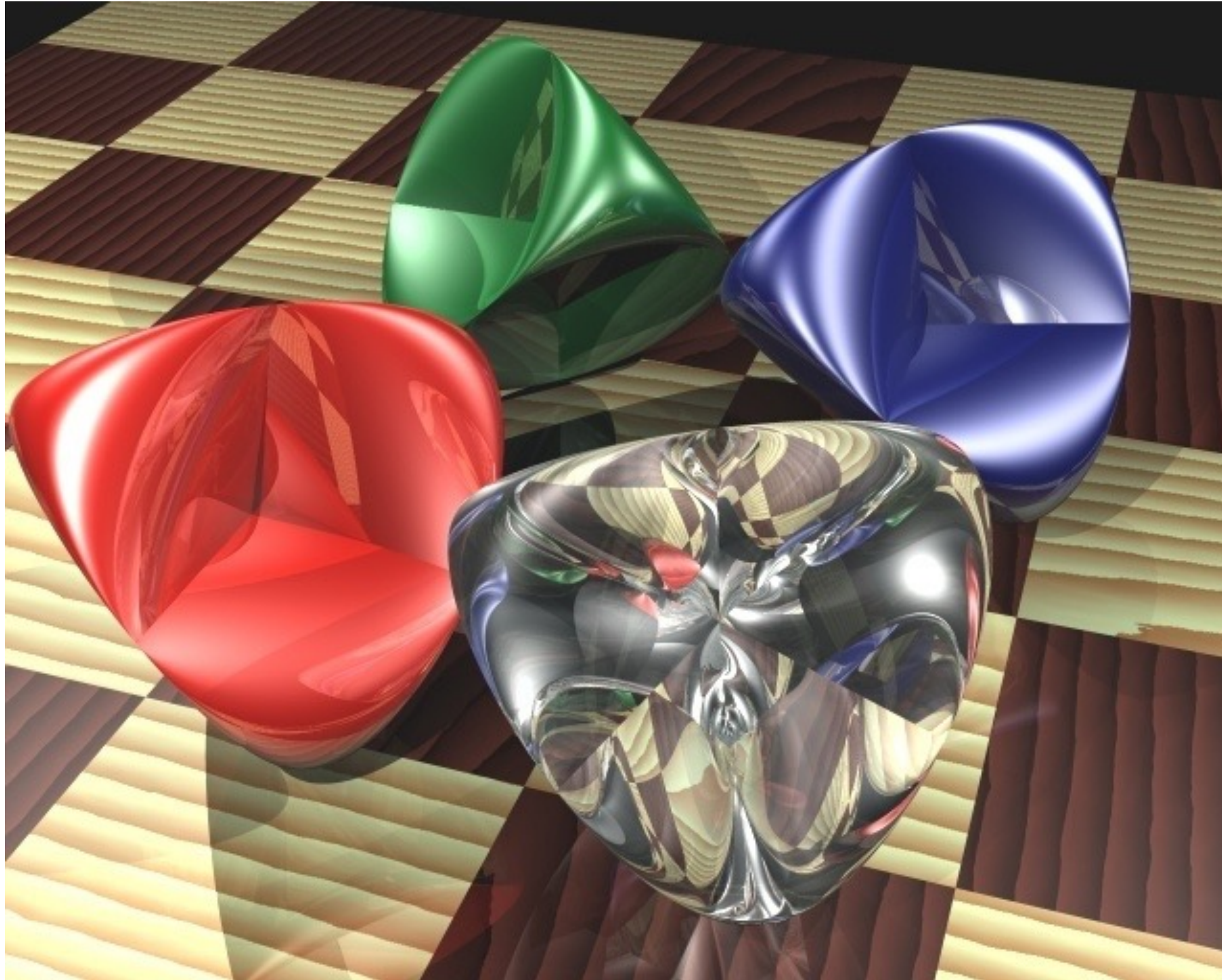
Ray Tracing pseudo-code: secondary rays

```
For each pixel p:

  make a ray r (from viewpoint to p)

  while (…)
    for each primitive o in scene:
       find if intersect(r,o)
    keep closest intersection;
    r = new_bounce( r );

  find color for p (according to all path)
```

Ray-Tracing: examples

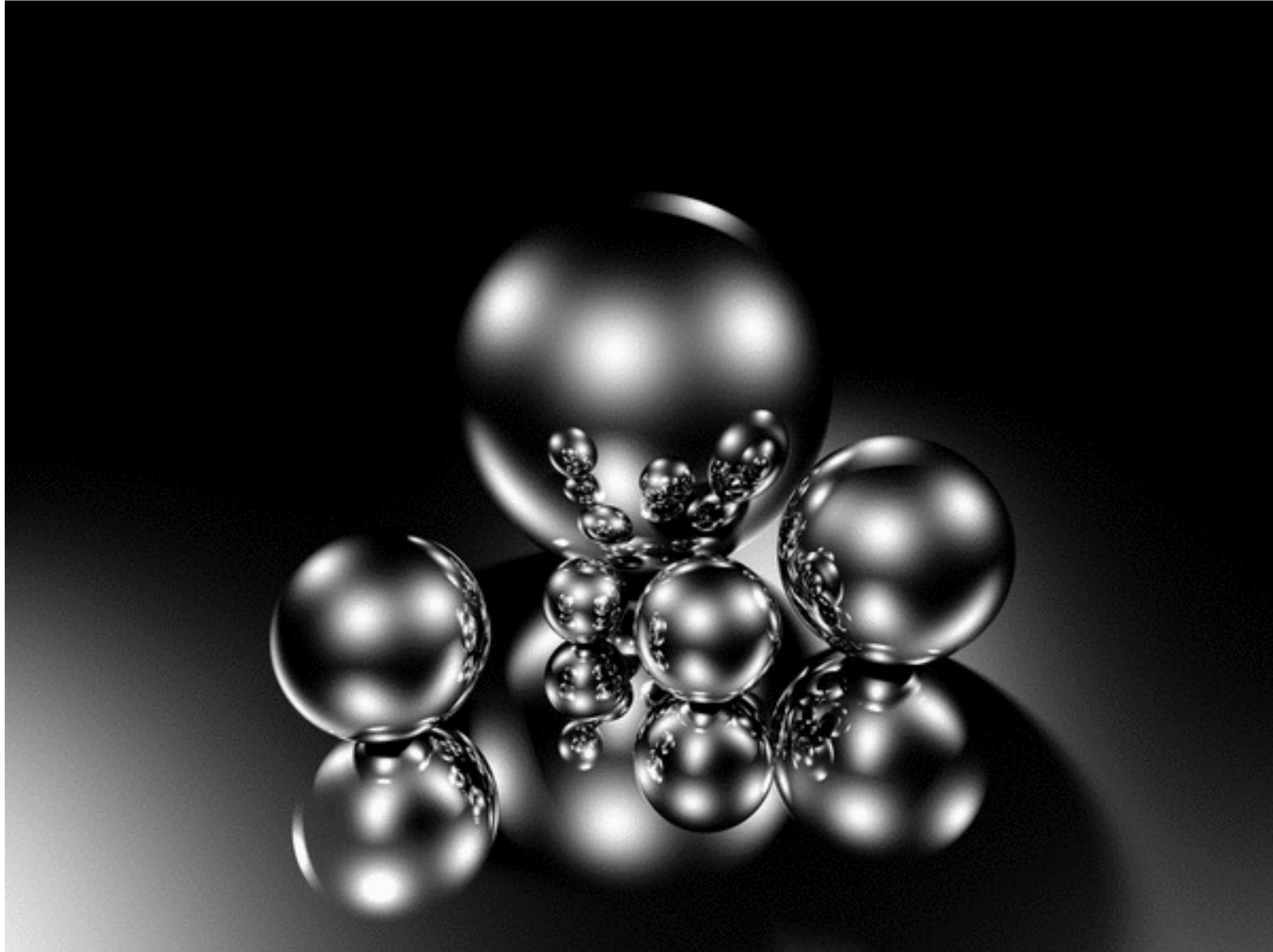


Ray-Tracing: examples



(Advanced Rendering Toolkit - Alexander Wilkie - 1999)

Ray-Tracing: examples



Ray-Tracing : cost

- Main core:
  - Computing the intersection between a 3D RAY e 3D PRIMITIVES
- Computationally complex
  - High cost
  - Yet SUBLINEAR! (wrt to the primitive count)
    - If you are using the right structures: spatial indexing!
- Once used only for off-line
  - Now realtime in simpler case exploiting GPUs

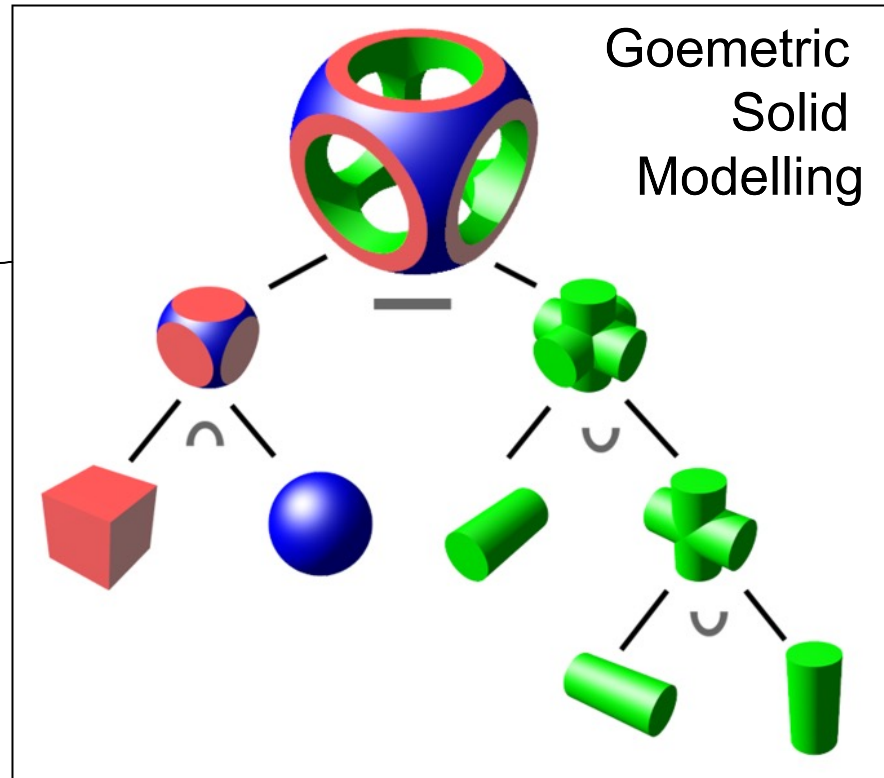
Ray-Tracing : Primitive di rendering

Q: quali primitive di randering per raytracing?

A: qualunque cosa io sappia intersecare con un raggio!

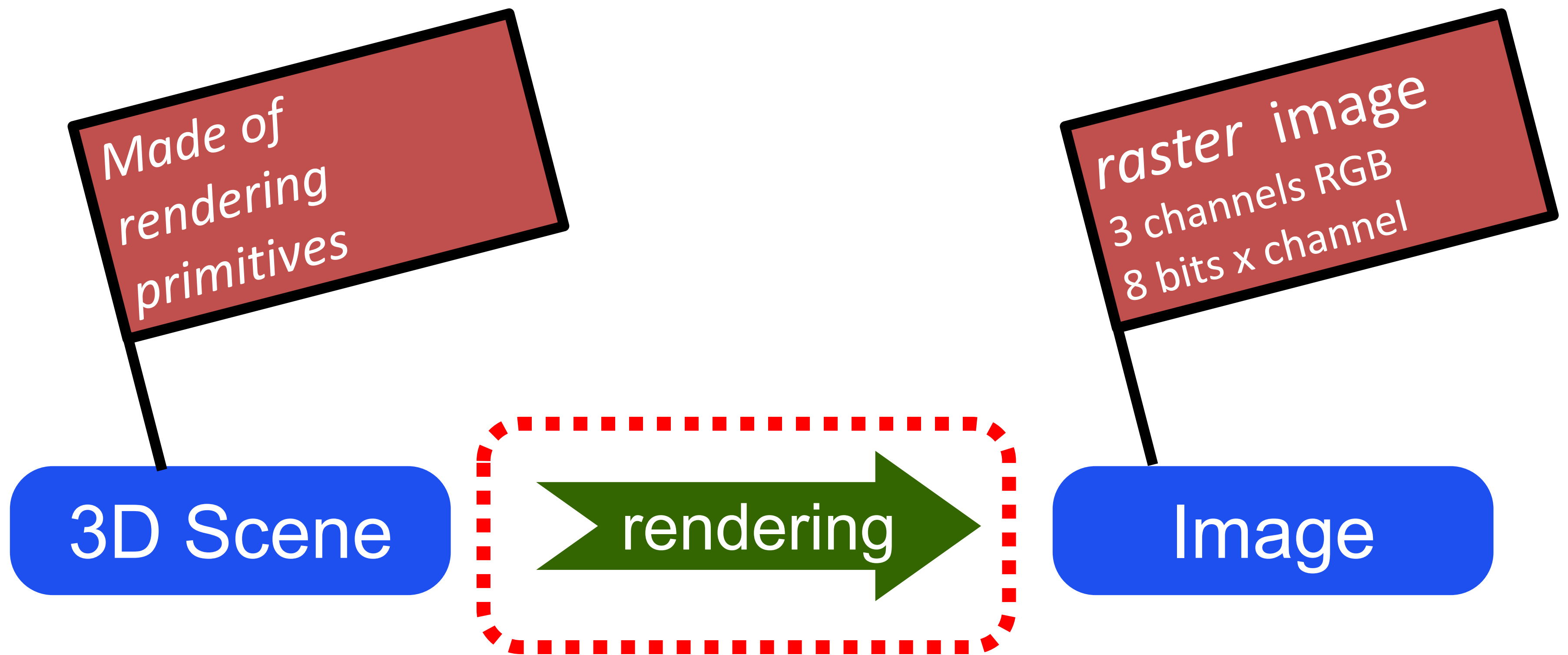
Es:

- triangoli
- superfici implicite
  - sfere
  - GSM
  - …



RayTracing su GPU (es su Nvidia RTX)

PART 2: *the other main approach*



Rendering Algorithms Paradigms

RAY-TRACING

```
For each 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

to “Rasterize”:

- convert a 2D shape
  - e.g.: text, polygons, curves, …

into pixel
  - of a raster image

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

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) «rasterize» in 2D

- == convert into pixel

Most commonly:

- *SEGMENTS*
- *POINTS*
- Usually *TRIANGLES*

aka T&L: Transform & Lighting...

- *Transform* :
  - trasformazioni di sistemi di coordinate
  - scopo: portare le primitive in spazio schermo
  - ... ma anche comporre le primitive per formare la scena
- *Lighting* :
  - computo illuminazione
  - scopo: calcolare il colore finale di ogni parte della scena
    - risultante da
      - le sue caratteristiche ottiche
      - l'ambiente di illuminazione
      - la geometria (forma degli oggetti)

Rasterization-based HW-supported rendering

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

composta da primitive di pochissimi tipi:

- punti
- linee

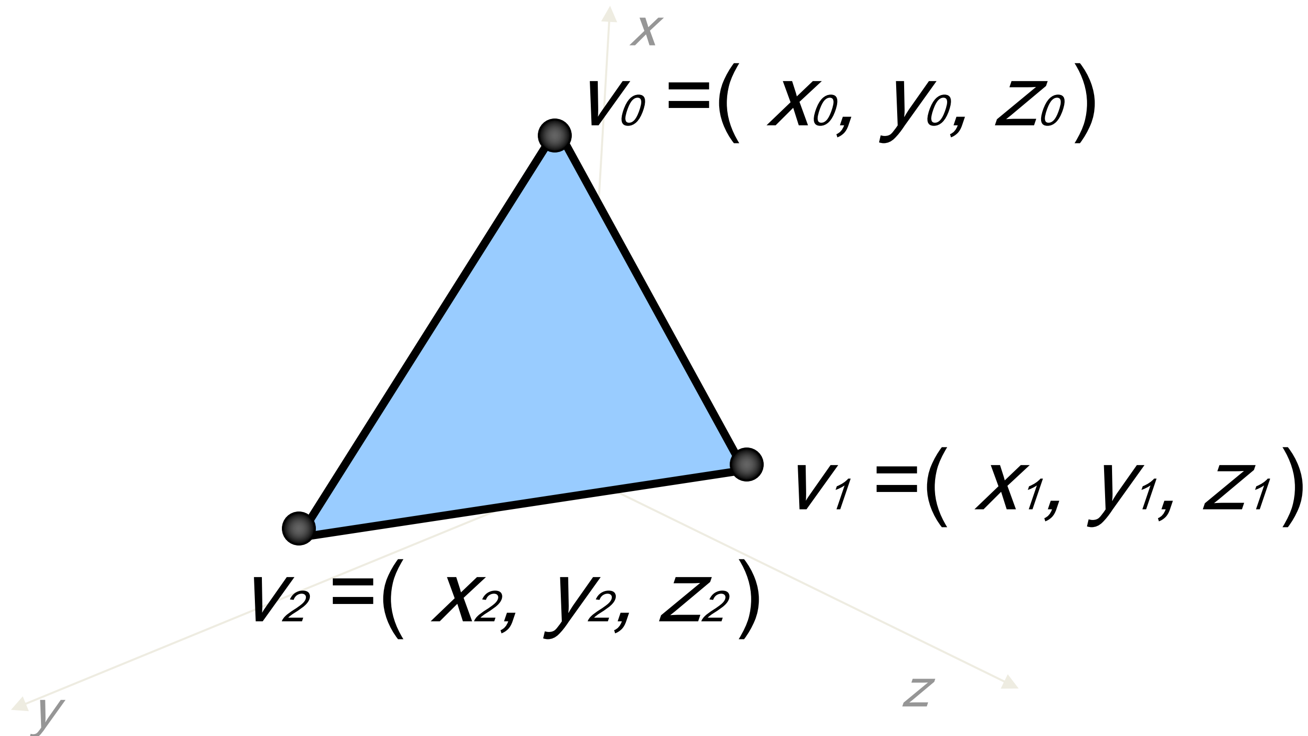
MA SOPRATUTTO

- triangoli

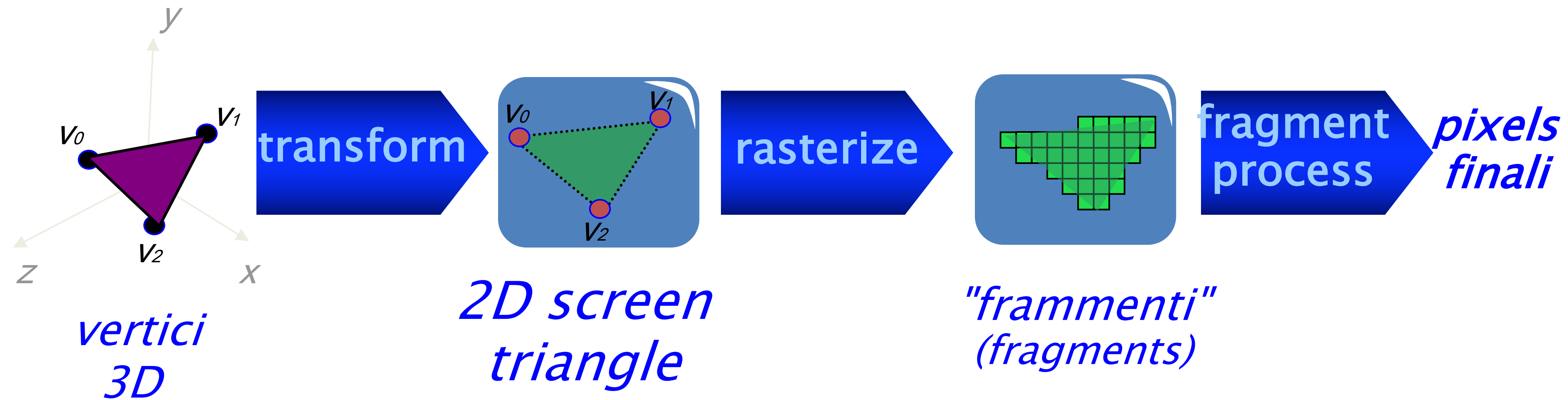
primitive di rendering

Rasterization-based HW-supported rendering

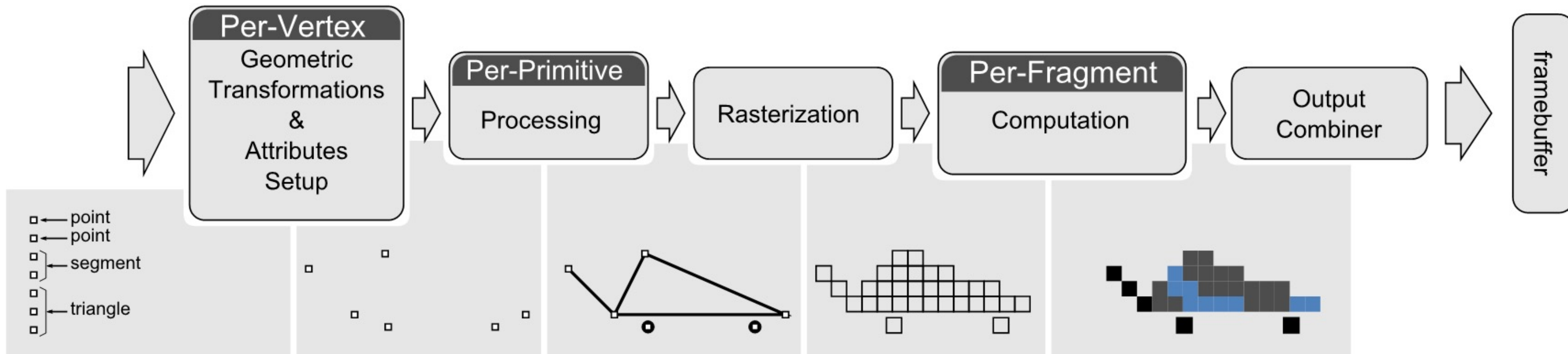
- decompose entire scene with 3D **TRIANGLES**



... Rasterization-Based Rendering



Rasterization-based pipeline



Rasterization-based rendering

- Input: set of vertices and its associated attributes
- Algorithm goes through several phases:

1. Per-vertex transformations and attributes setup
2. Primitive processing
3. Rasterization
4. Per-fragment computation

THE GREAT DEBATE

RASTERIZATION

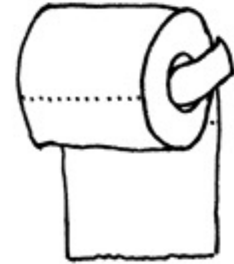
VS

RAY-TRACING

Rendering Algorithms Paradigms

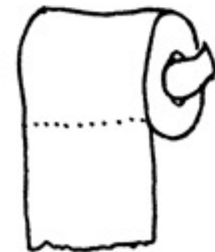
RAY-TRACING

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



RASTERIZATION BASED:

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



Advantages of ray tracing

- Conceptually simple algorithm
- Takes into account GLOBAL effects
- More realistic rendering of lighting effects
- *More freedom of primitives*
  (intersect easier than project+rasterize)
- Potentially great scalability
  with scene complexity

Ray-tracing : simple and elegant?

```
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(25/114.5915590261),U=vcomb(255.,
trace(3,black,vunit(U)),black),printf("%.0f %.0f %.0f\n",U);}/*minray!*/
```

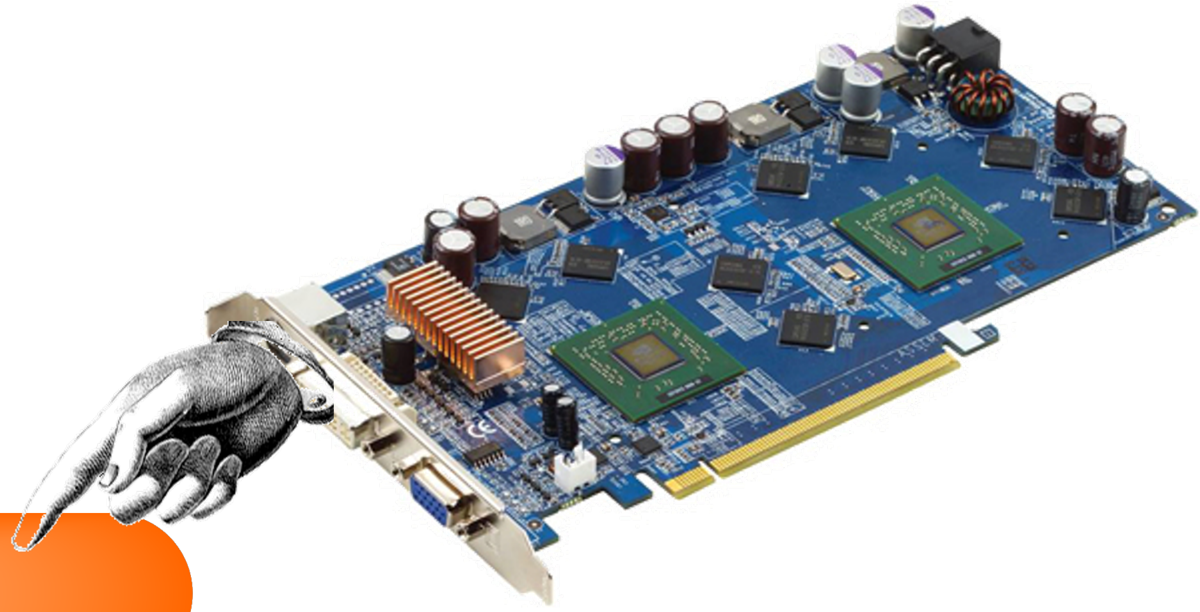
A whole raytracer on a business card :-P !
(by Paul Heckbert)

Advantages of rasterization

- More easily parallelizable?
- More controllable complexity
- Better-suited for graphics cards (for now)
- Easier with dynamic data?
- Better treatment of anti-aliasing (more later on)
- Better scalability with image resolution

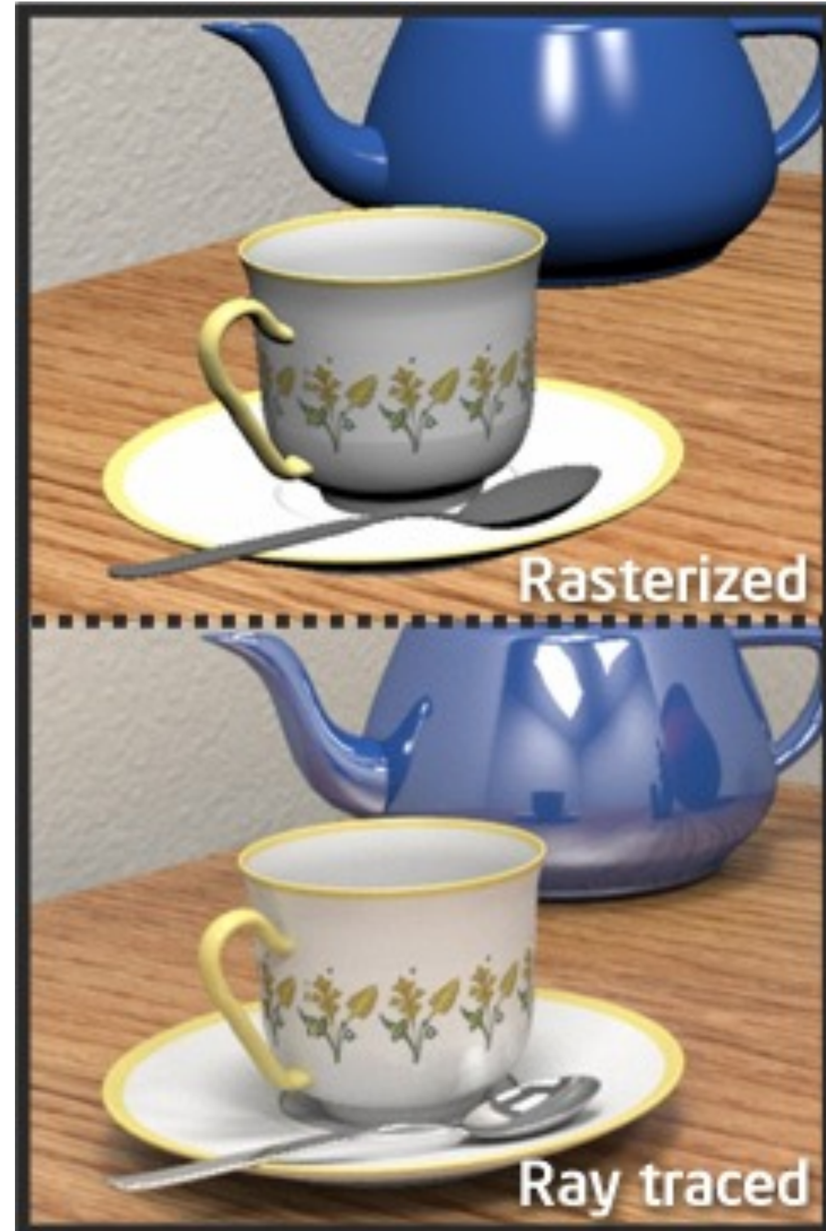
Historic View

- Ray-tracing
  - algorithm of choice for OFF LINE rendering
  - used for MOVIES (and CGI static images)
  - Pixar, Dreamworks, etc. favourite
  - well known ray-tracers: POV-ray, renderman, YafaRay…
- Rasterization
  - algorithm of choice for REAL TIME rendering
  - used for GAMES (and previews, and 3D on web…)
  - Nvidia, ATI, etc. favourite
  - First to get specialized HW



- Ray-Tracing
- Rasterization based
- Image based (per es. light field)
- Radiosity
- Point-splatting
- …

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



A fairer comparison

Why ray tracing?

Environment map

Ray-traced reflections

PIXAR

The commonplaces

- Raytracing:
  - good for complex visual effects
    - shadows, specular reflections, refractions...
  - SW-based (CPU)
  - *therefore:* off-line, hi-quality renderings!

- Rasterization:
  - fast!
    - for each primitive, process only a few pixels
      (instead of: for each pixel, process *all* primitives)
  - HW-based (GPU)
  - *therefore:* real-time, compromise-quality renderings!

Reality : Ray-Tracing

not necessarily SLOW! →

- algorithm and data structures for efficient "spatial queries"
- even sub-linear with number of primitive!

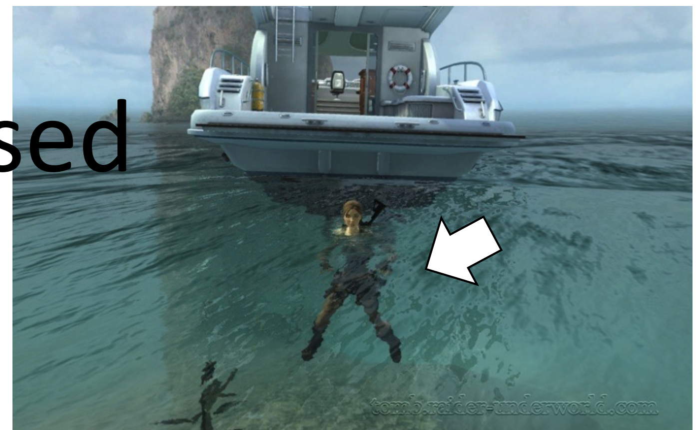
not necessarily SW:

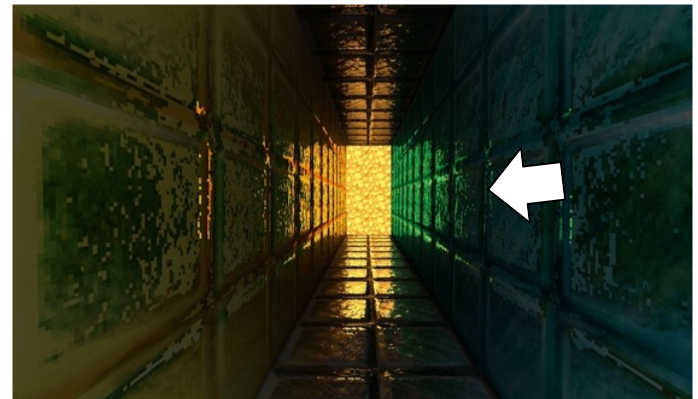
- Specialized HW?
- RTX?

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

Reality: Rasterization based

- not necessarily
  without complex visual effect
  (they just require some
  approximation and
  a few advanced algorithms)





The reality

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)