3D for fun and profit

A simple introduction

Marco Callieri
ISTI-CNR
callieri@isti.cnr.it

Who Am I ?

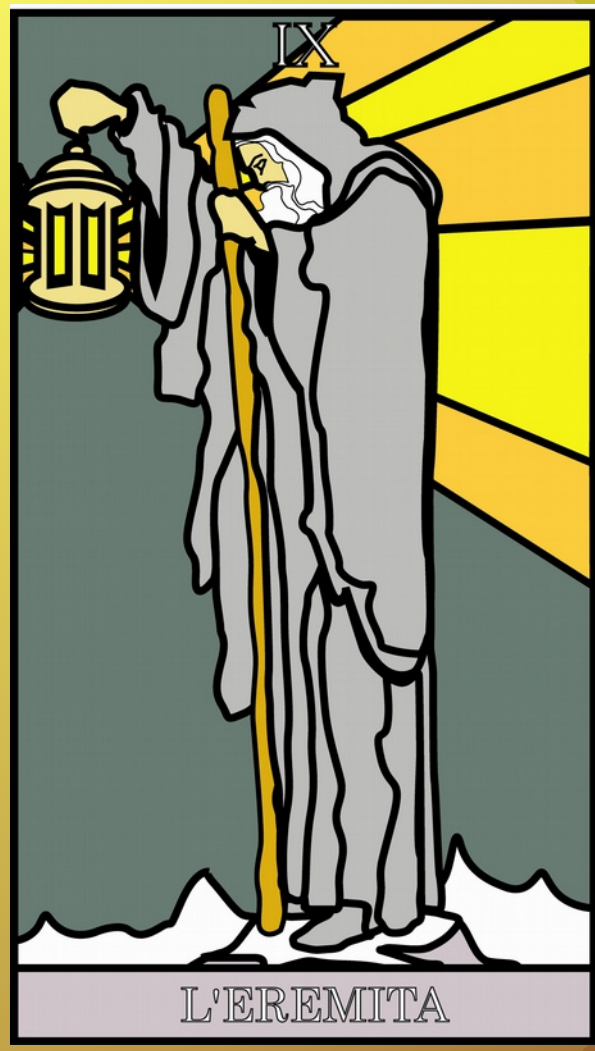
Marco Callieri

Master Degree & PhD in computer science

Researcher at the Visual Computing Lab,

ISTI-CNR, in Pisa

I work on 3D for Cultural Heritage, 3D data manipulation and rendering... lot of experience in 3D scanning and data processing



Beside this:

an eclectic artisan, an avid gamer, a former biker, a good cook, an incorrigible geek... and much more

3D from 10.000 feet

3D has become quite a buzzword in the last few years

Seems that, with 3D, everything is better...

We aim at providing you a focused knowledge of devices, tools and techniques that employ 3D in the field of cultural heritage...

However, we do believe an introduction to basic, general 3D concepts may help... even a simple (and very incomplete) one as this :) :)

Double Nature

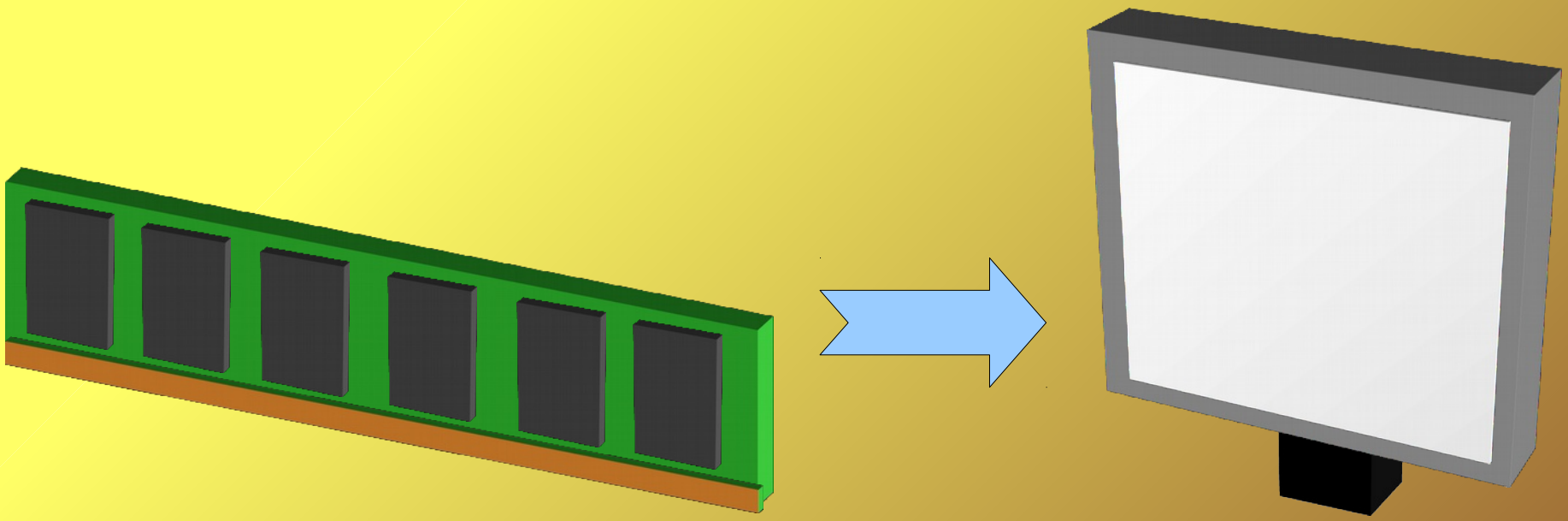
3D is a peculiar kind of data, because it has two sides:

- **PURE DATA:** 3D is made of measures, it is pure geometric information, a numerical representation of something... you may use this number-based representation do calculation, simulation, measurements on it.

- **VISUAL DATA:** can be displayed, presented and perceived visually using the part of the brain we use for everyday perception… you may explore, analyze and understand it in the same way you look at the physical world

From numbers to visual

Starting from this idea, you may speculate that there is a **numerical representation** by which a 3D environment may be stored in the computer memory, AND **some way to produce an image** that represent a visual over this environment...



Wording

A common word in the 3D field, "*rendering*" is used in different ways, but has a general meaning

RENDERING: the process by which some data in memory is converted into something that can be directly perceived by the user.

Pure data

Let us start from the representation....

How the 3D data is (generally) managed.

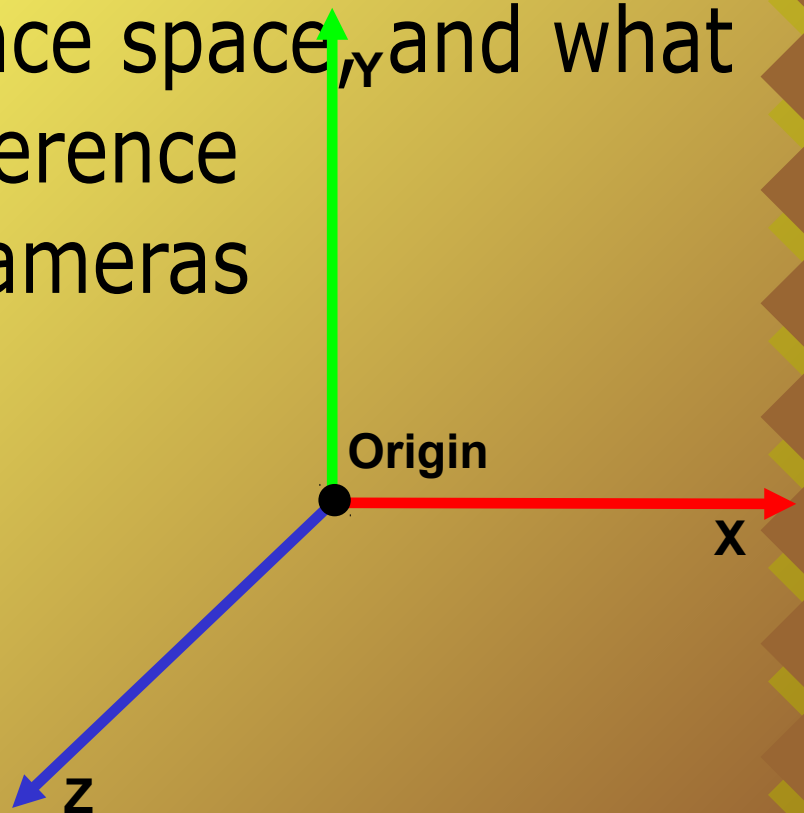
Beware, there are many, many ways to represent, store, manipulate 3D, depending on what you have to work on, the purpose of the work, the amount of data you have to manage.... but we will start simple

3D space

Our 3D world is a space defined by an origin (a reference point) and three orthogonal axis. Points in this space are defined by their three coordinates [X Y Z].

All the 3D data is referenced in one of these spaces.

Each 3D model has a local reference space and what we call "scene" is just another reference space where models, lights and cameras are positioned...



Vertices - Faces

Simplest entity: **VERTEX**. 3 coordinates [XYZ] that define a point in the reference space.

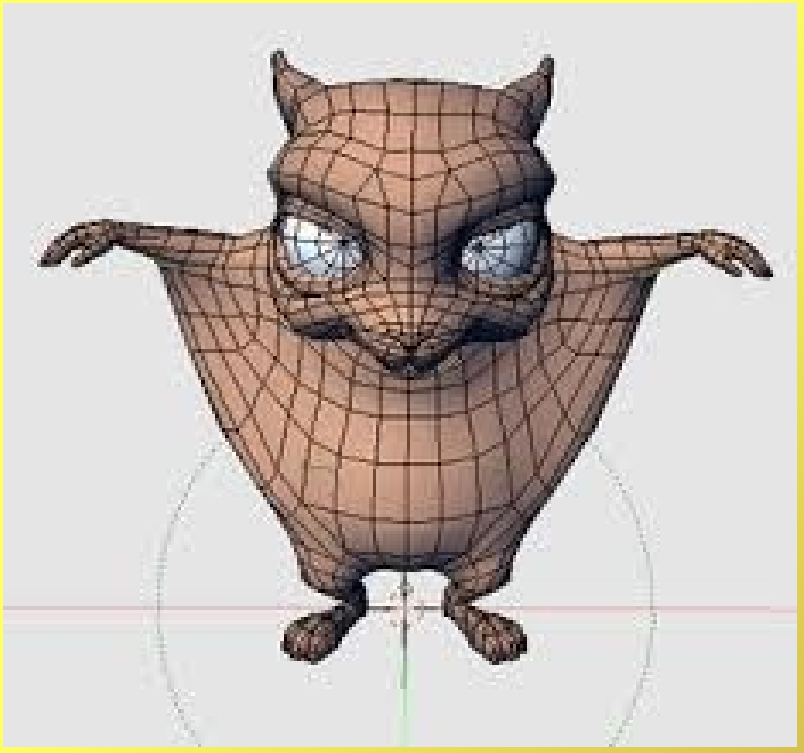
With many vertex, you start seeing a shape, but stil there is not a SURFACE... How do you create a surface?

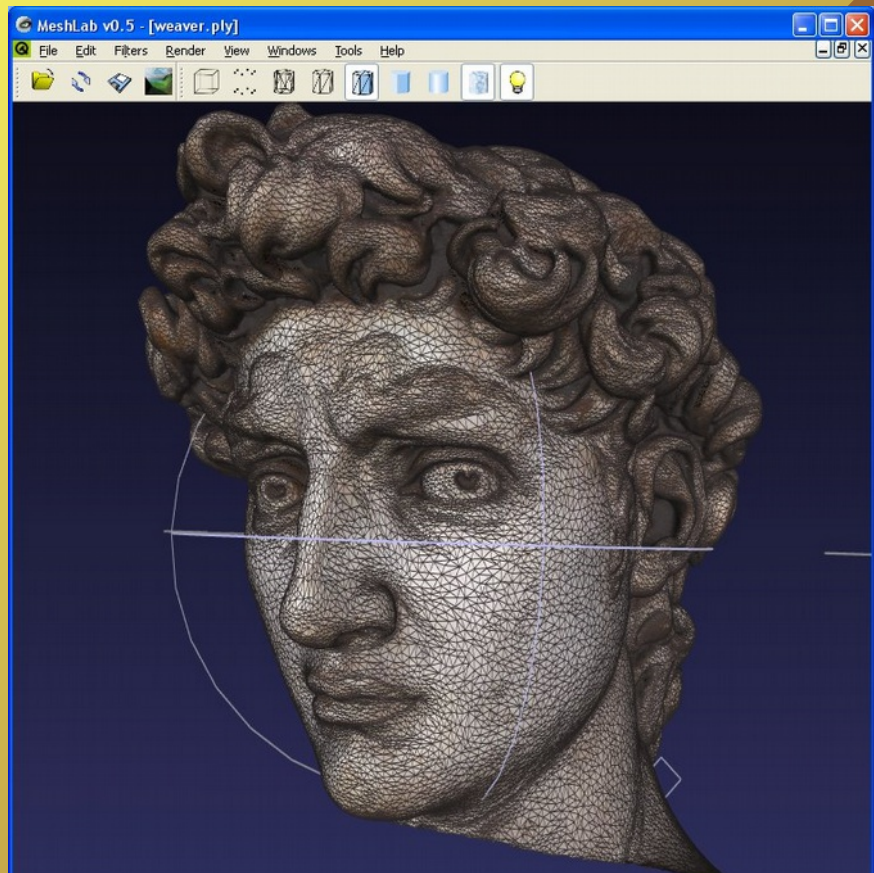
By connecting vertices in a **MESH** of faces

Vertices - Faces

Faces may be TRANGLES, QUADS, or N-gons...

- Triangles are the simplest and most used primitives
- Quads are the most used primitives in 3D modelling
- Ngons are used, but generally reduced to triangles

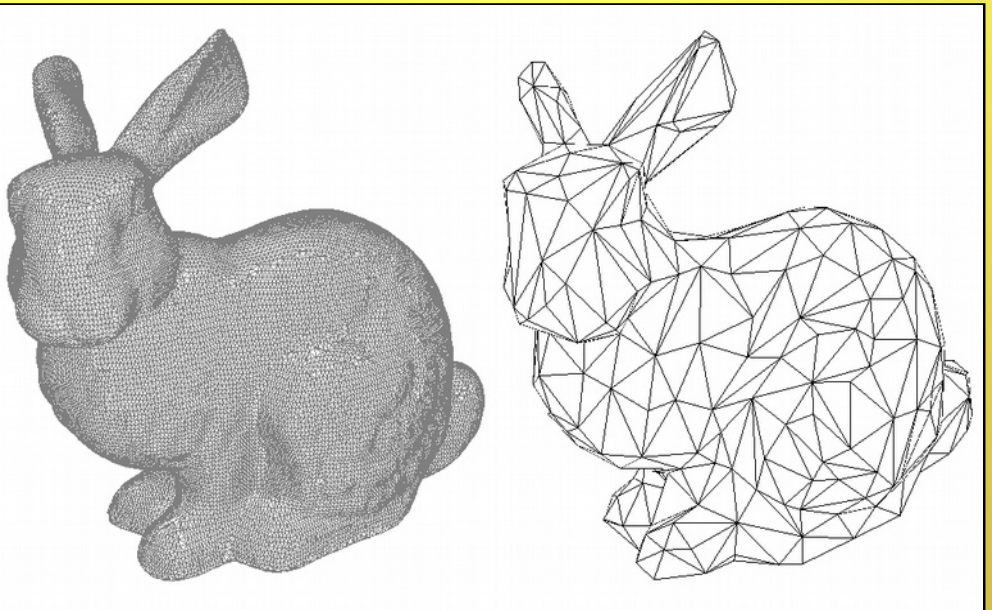


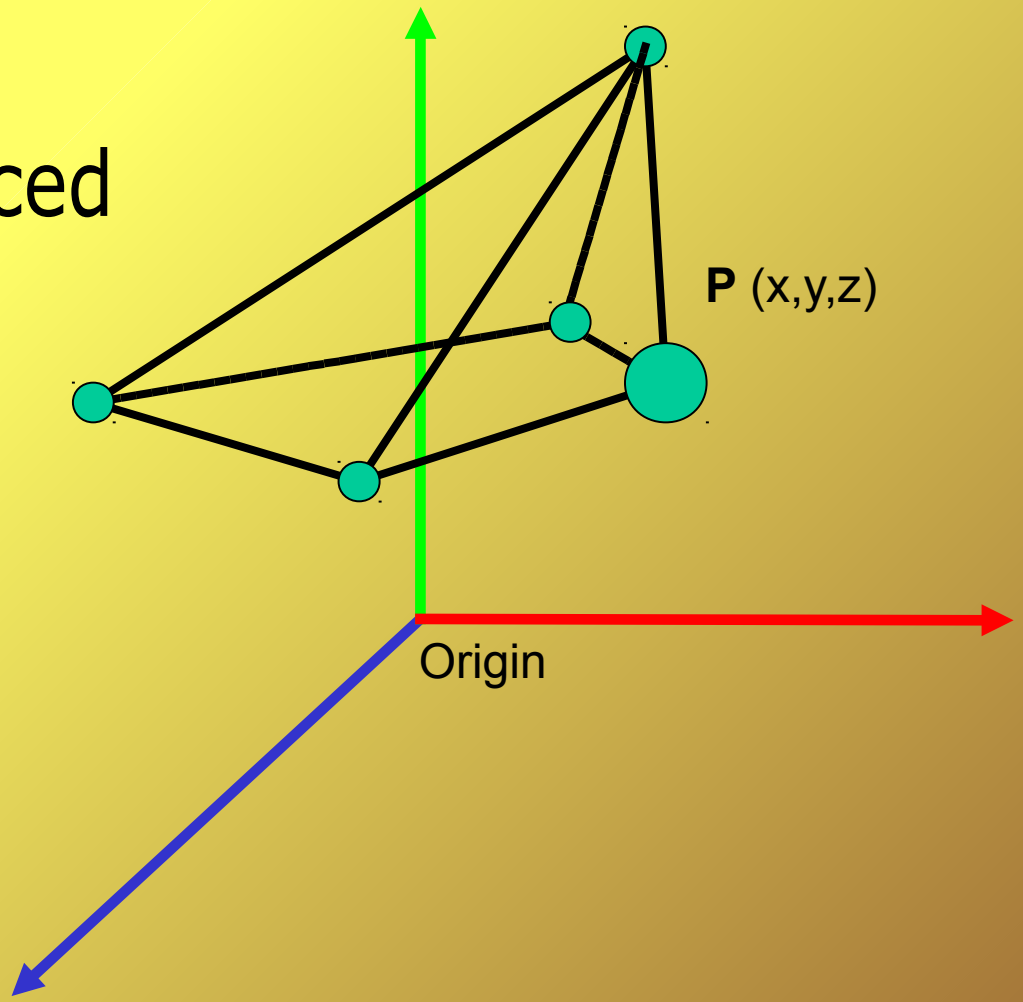


A 3d model

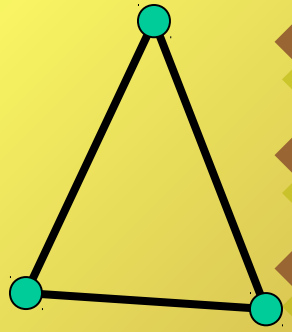
This is a simple 3D model...

A triangular mesh, referenced to its local origin





3D model



Most 3D models (especially the ones you'll see in our course) are made just by VERTICES (points in the 3D space) connected in TRIANGLES.

Triangles are always planar, with enough triangles you may represent any shape, the mathematics and algebra used to manipulate triangles is simple, fast, and easy to be managed by computers...

Surface and other data

Triangulated models does only have geometrical information of a zero-thickness surface (only the external shell).

Other info, like color, specularity, roughness, transparency... are generally mapped on this surface.

3D geometry

Texture mapped rendering

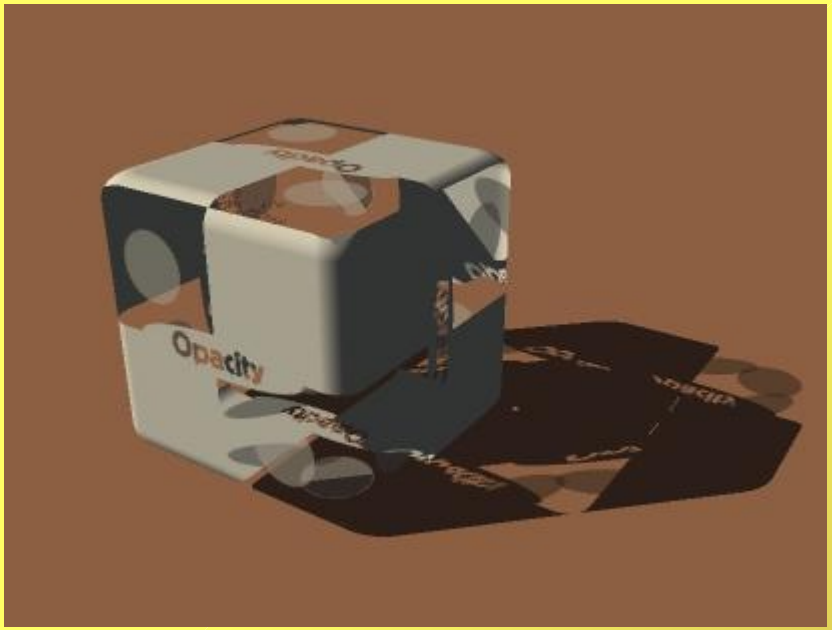
Other values

Normal map

Displacement map

Other values

Transparency map



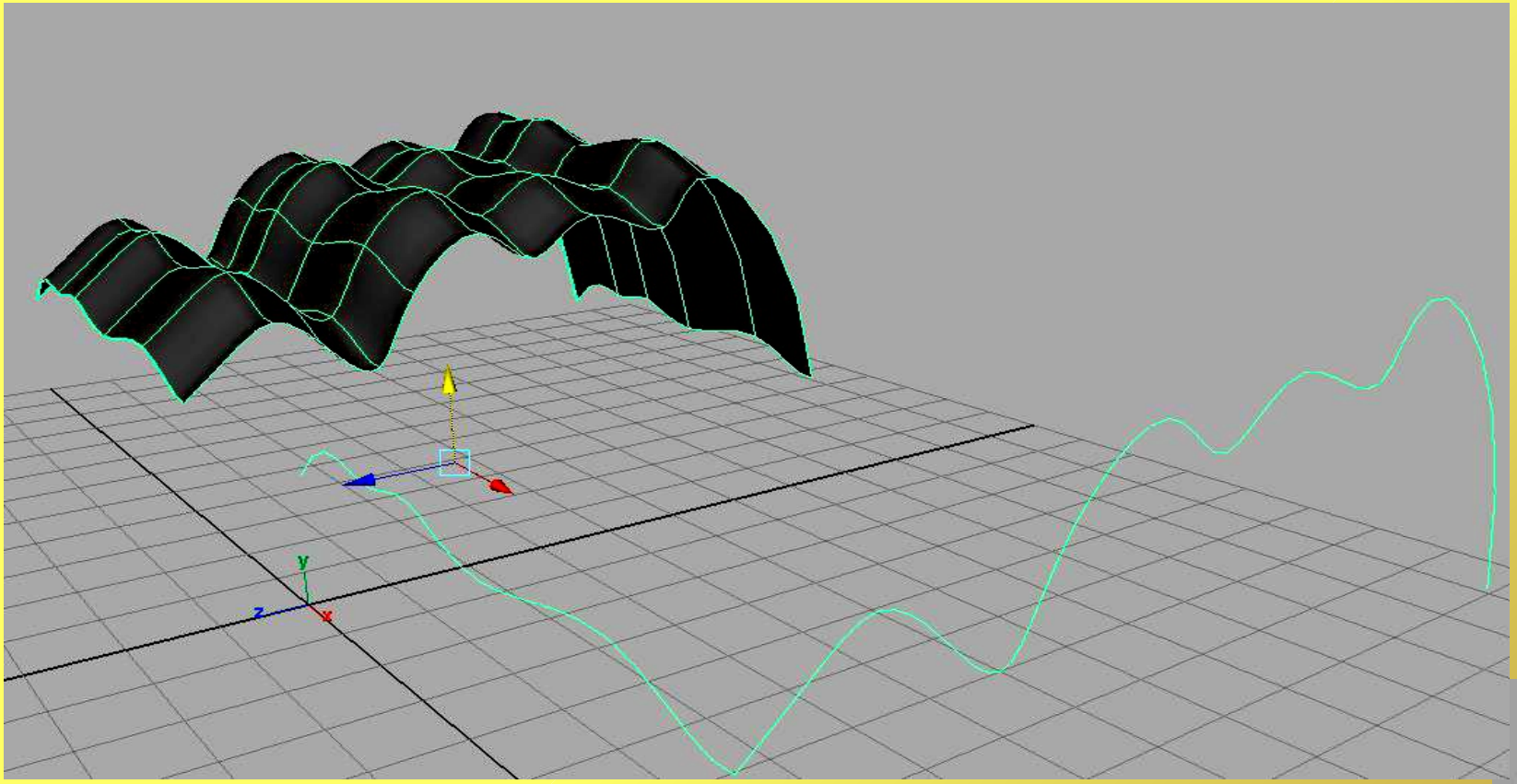
Roughness/reflection map

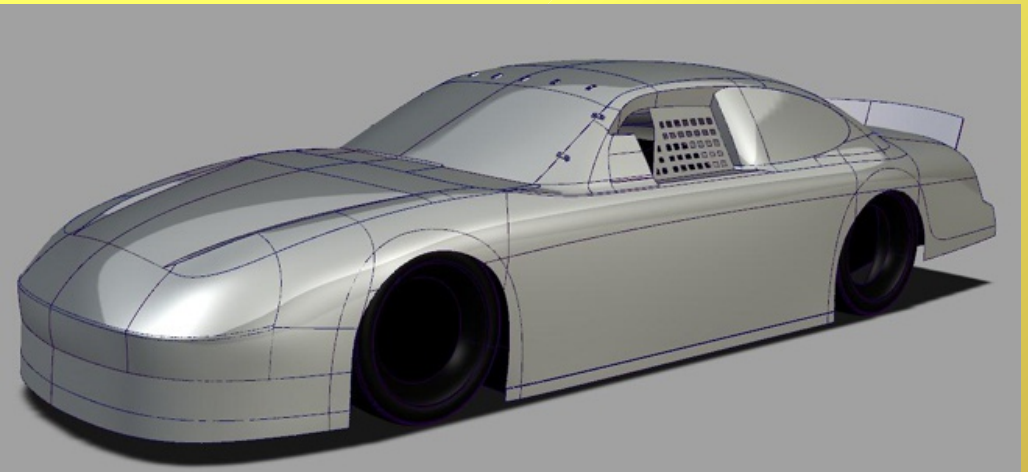
Mathematical curves

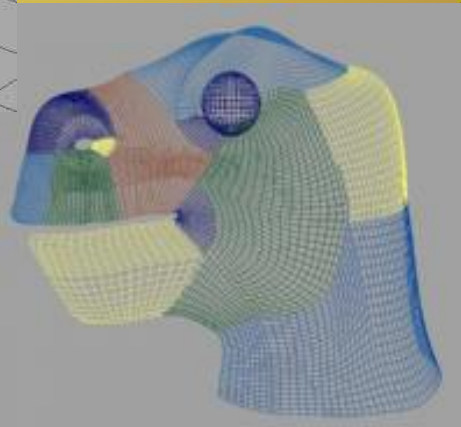
A more *continuous* representation may be obtained by NURBS or other mathematical 3D curves and shapes...

They are used mostly in CAD and modeling tools, are much smaller in memory (the mathematical representation is very compact) and are mainly used to represent smooth surfaces

Mathematical curves





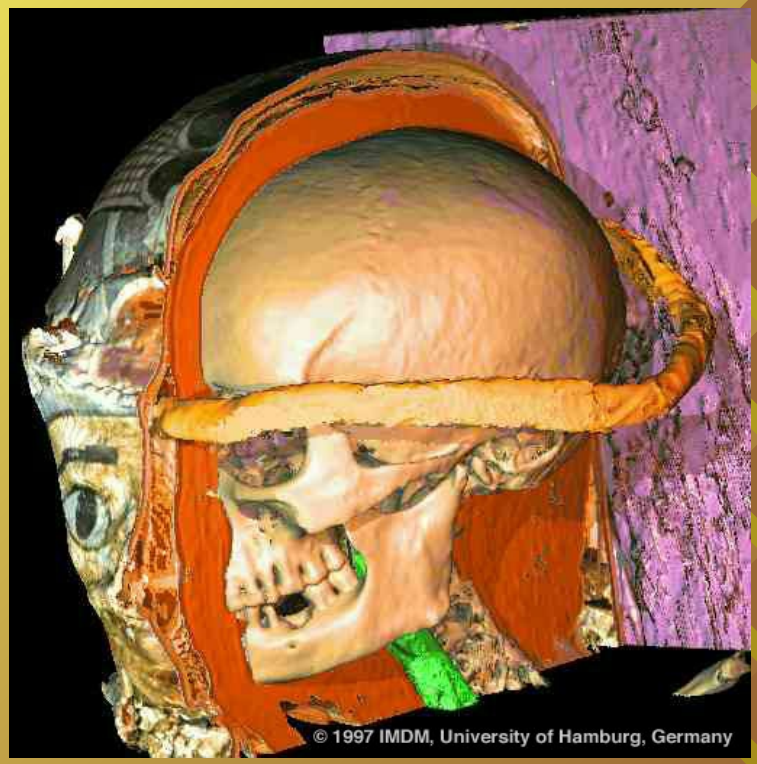


Volumetric data

TAC, PET, X-RAY and other medical/industrial equipment may produce a volumetric representation of objects: for each point in the 3D space, there is a value of density (and other data).

Volume data is used in medicine but also in CH (mummies).

We will not cover this kind of data
I mentioned just for background..



And much more

There are many other ways to define surfaces, volumes and other kind of 3D data... but they fall outside the scope of this course.

Each one has its own use, strenghts and weakness...

The scene

One or more models are arranged inside a space, trough rotations translations and scale operations, defining a **scene**.

The scene

Translation: a translation vector (how much the model has moved along the three axis) is added to all vertices

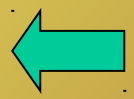
$$[\ X+dx,\ \ Y+dy,\ Z+dz\ ]$$

Scale: all vertices are multiplied by a scale factor

$$[\ a * X,\ a * Y,\ a * Z\ ]$$

Rotation: more complex... each rotation may be expressed as a matrix, to rotate a vertex, you multiply by this matrix

1	0	0
0	cos Θ	-sin Θ
0	sin Θ	cos Θ



Example:
rotation on the X axis

Matrix

Matrices... Matrices everywhere...

All the operations to manipulate 3D models in the scene are matrix-vector, matrix-matrix and vector-vector multiplications...

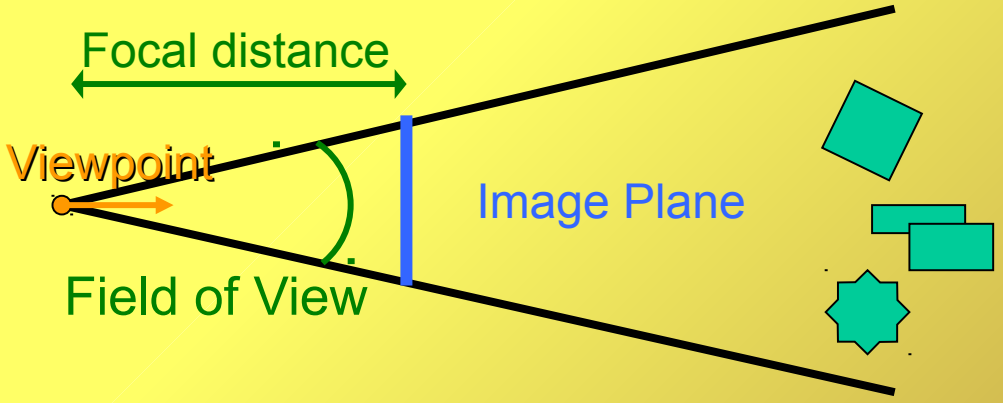
Why this is important?

Because you can go **FAST** since this stuff is damn easy

The scene, again

Beside 3D models, in a scene, there are often many other elements

- One or more ***camera***s: defining view over the scene
- **Lights**: used to simulate illumination
- …



Rendering

Now we have the 3D data in memory... we want to create a visual representation out of it.

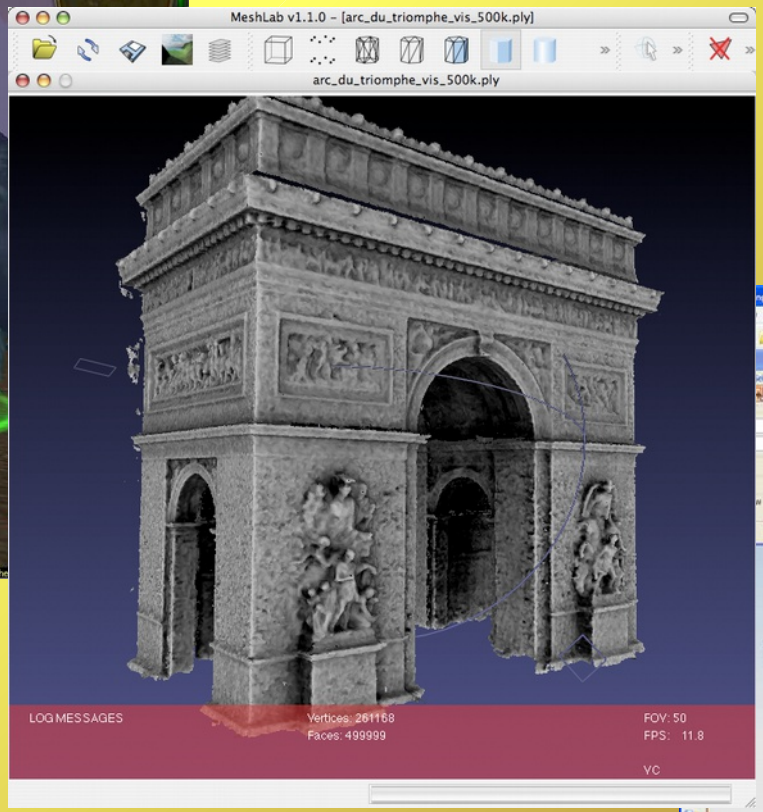
Good idea... but what, exactly, I want to do with this visual representation?

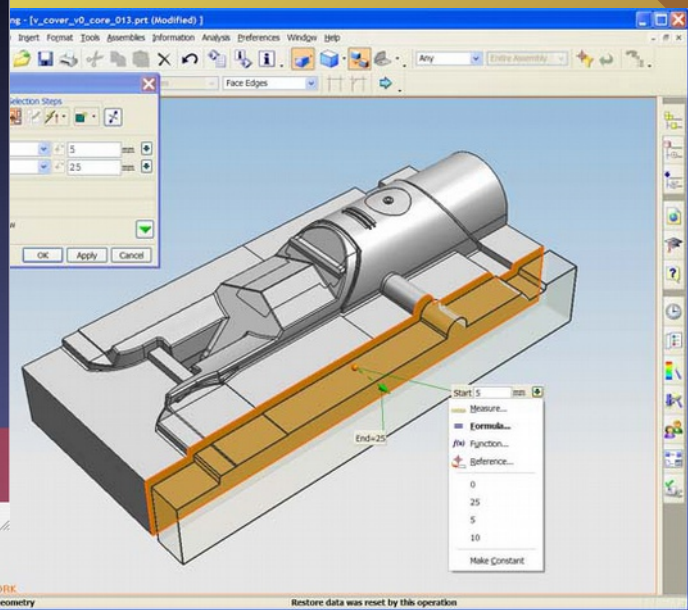
There are two main families of “rendering“...

Interactive 3D

You may have encountered applications where a digital 3D world is *somehow* displayed and, *somehow* the user may change the view over this world or the world

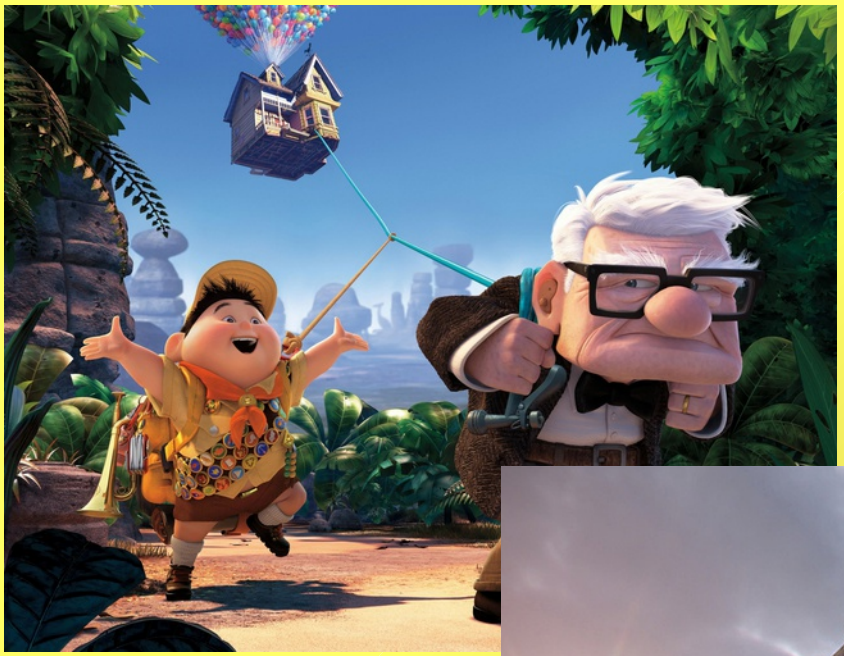
(The description is very very general, but for a purpose)

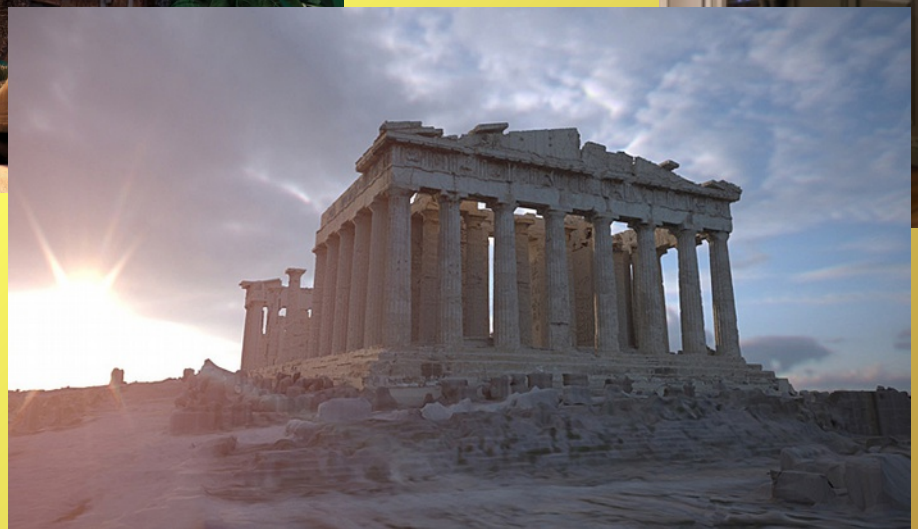




Synthesis images

On the other hand, you are certainly aware that some movies/images are generated from 3D data in such a way that they appear photorealistic





From the same data

For the sake of simplicity, we may say that the SAME 3D data may be used for both kind of rendering...

You may surely imagine this is not "strictly" true, and that is correct... You will need specific data for each of the two.

But in the end, deep down, *everything* boils down to vertices, faces, textures...

Interaction

In a relatime application it is possible to “interact“ with the 3D scene in some way, and instantly see the effect of our interaction.

At minimum, it is possible to change the point of view over the 3D space.

On a different level, it may be possible to interact with scene elements (moving models around the scene).

Editing tools (3D modeling, CAD and similar) let you also modify models.

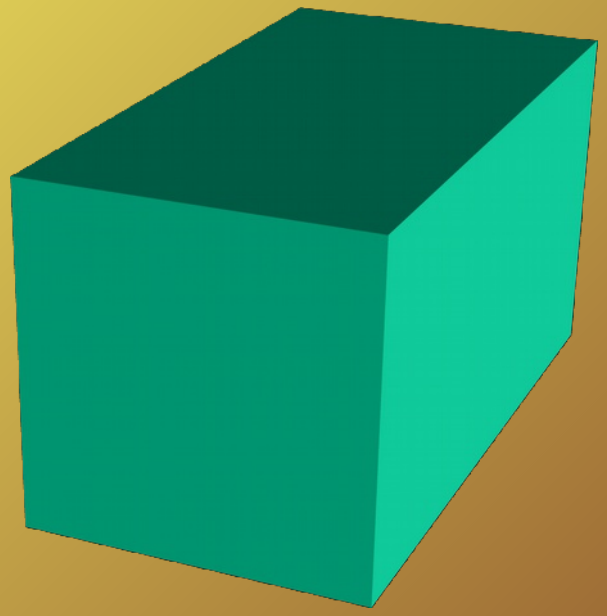
How this interaction is done, does change from one tool to another...

Realtime Rendering

Bringing these kind of data to the screen in an interactive application is called realtime rendering.

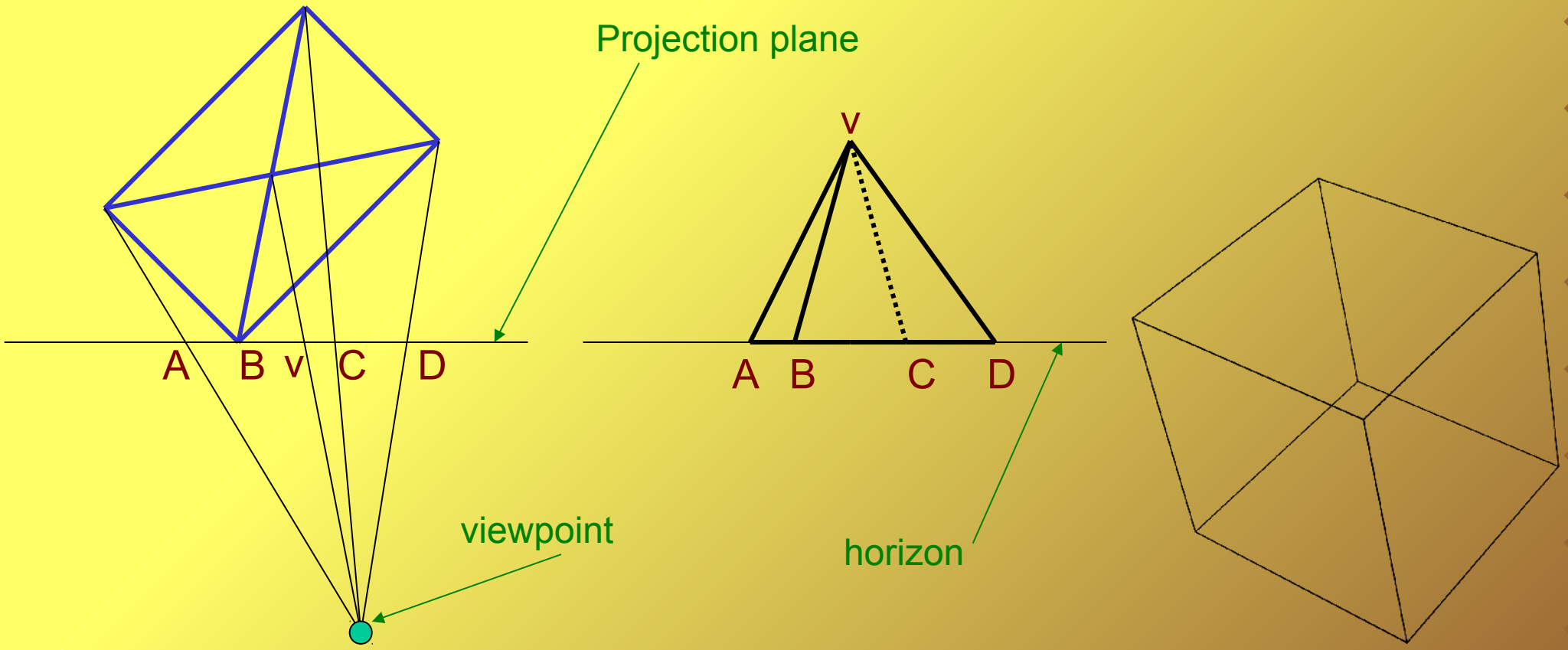
Basically, is a way to simulate how the image is formed in our eyes (perspective projection) and how light interacts with objects (lighting)...

Of course this simulation is simple and crude, to be able to do it many times per second



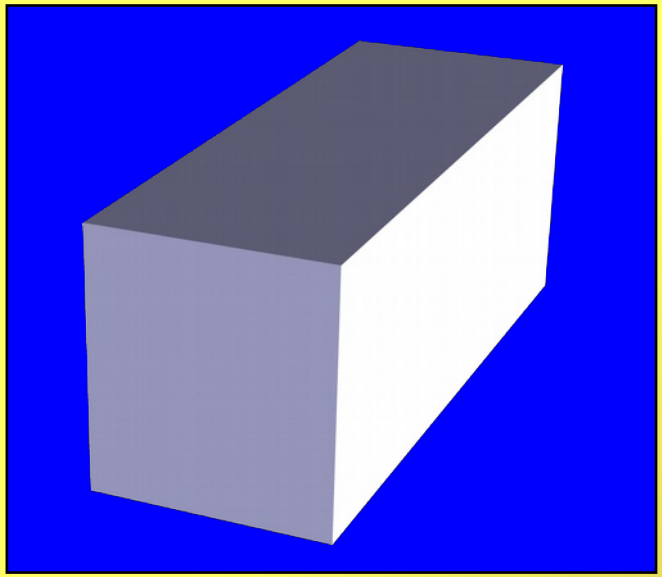
Projection

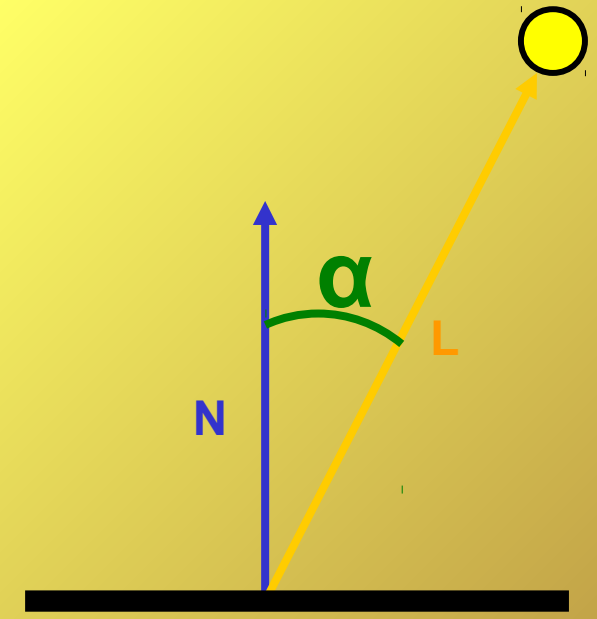
Given a 3D space, with 3D models and a CAMERA, the vertices are perspective-projected like we did at school in technical drawings... Only vertices are projected, triangles are drawn filling the space between points...

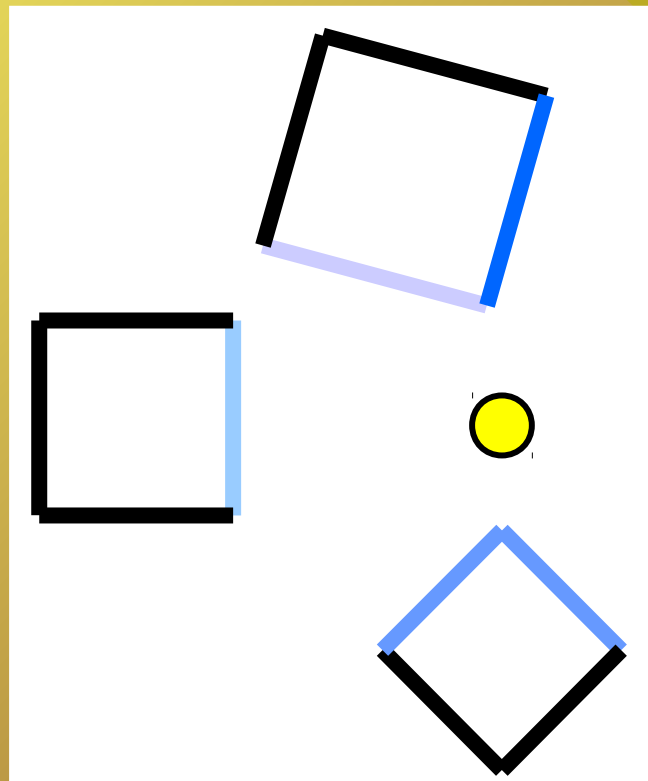


Lighting

The projected model will appear correctly from a geometrical point of view, but completely flat. To simulate lighting, the simplest method is to calculate the angle between light and each triangle. The more orthogonal the light is, the brighter the surface.







Lighting

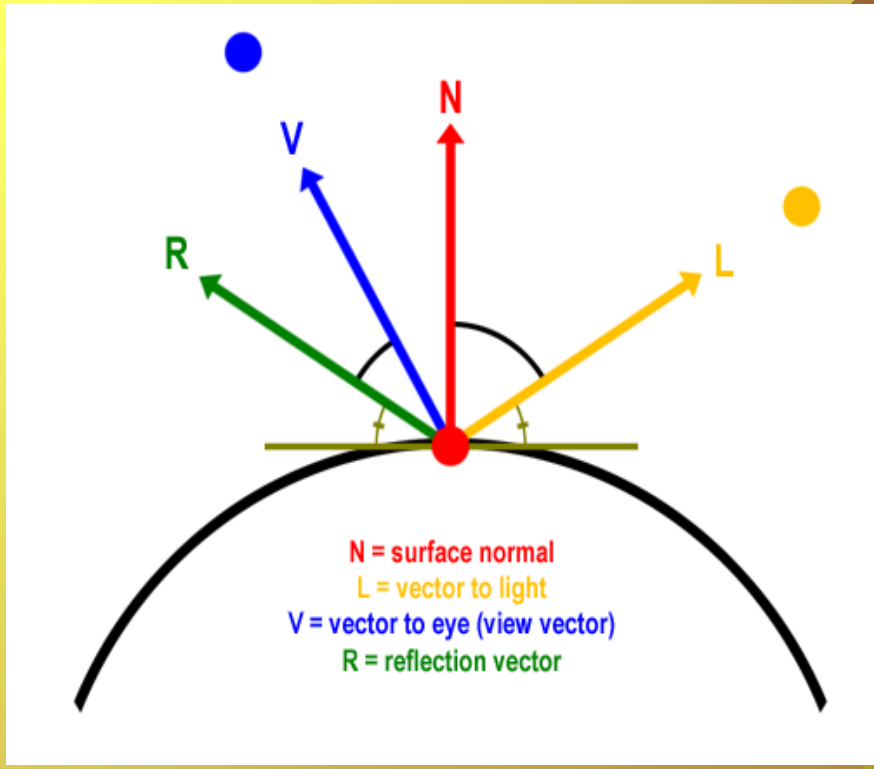
Things may get much more complex than this... Adding more lighting effects, more complex lighting models, and other real-time tricks.

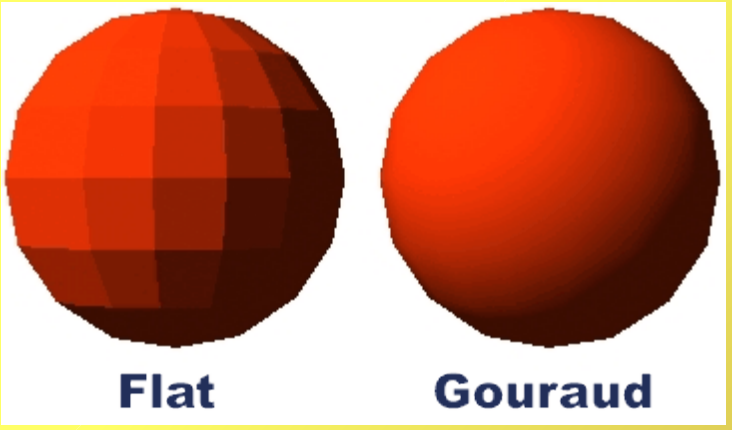
TRICKS, because almost nothing that is done in real-time rendering is really physically correct...

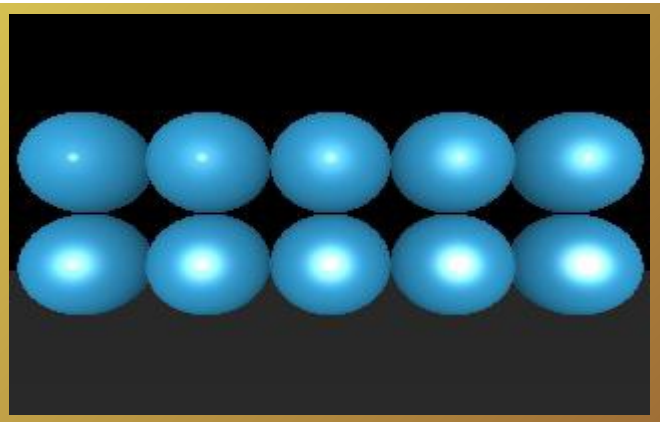
We will return to realtime soon, but first...

Lighting

Calculate lambertian for each pixel, I get "smooth" shading.. with a few more multiplications I add specular reflections...







Matrices, again

How we do we move objects around the scene?

Matrices-vector multiplication

How do you do the perspective projection?

Matrices-vector multiplication

How you do calculate shading?

Vertex-vertex multiplication

All the rendering rely on really simple atomic operations....

Offline Rendering

Again, you may speculate there should be a way to go from a 3D data representation to a photorealistic representation. Yes, this is correct. Yes, this method is much more complex and cannot be achieved in realtime (we will call it OFFLINE rendering)..

Light is what make us see, a better simulation of light behaviour is the key to photorealism...

The three main rendering methods try to simulate how light travels in the scene.

Light is

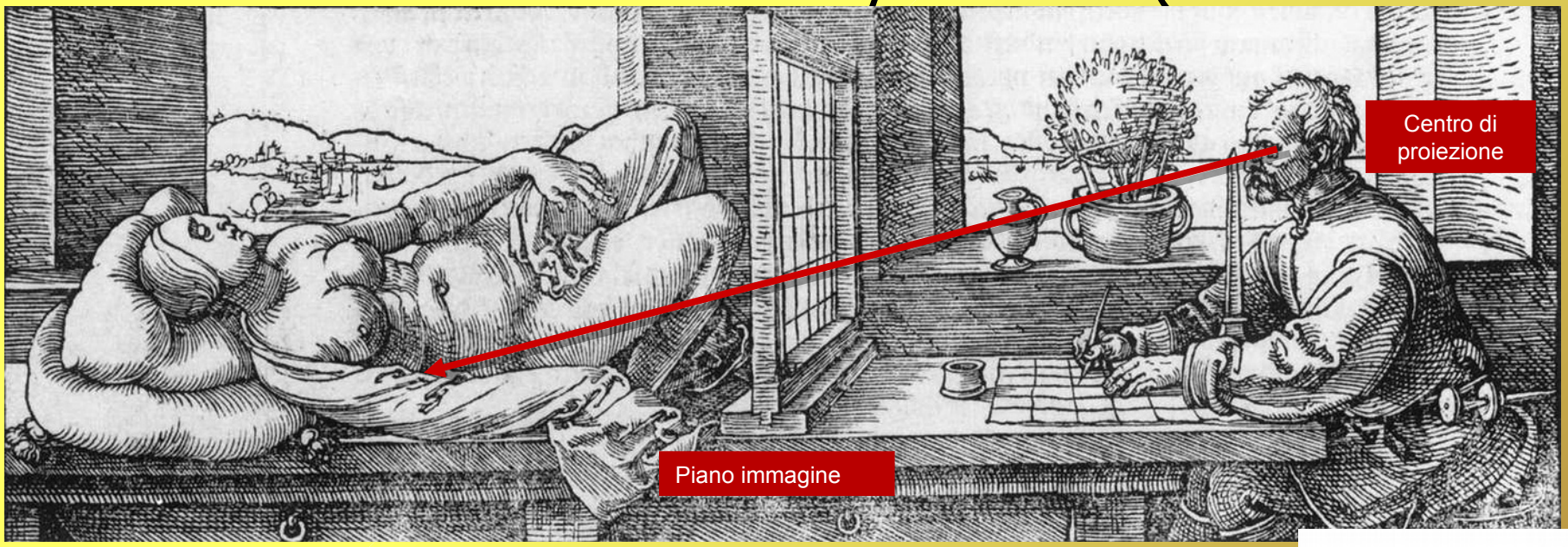
Light is *matter* → geometry of optics → **Ray Tracing**

Light is *energy* → energy equilibrium → **Radiosity**

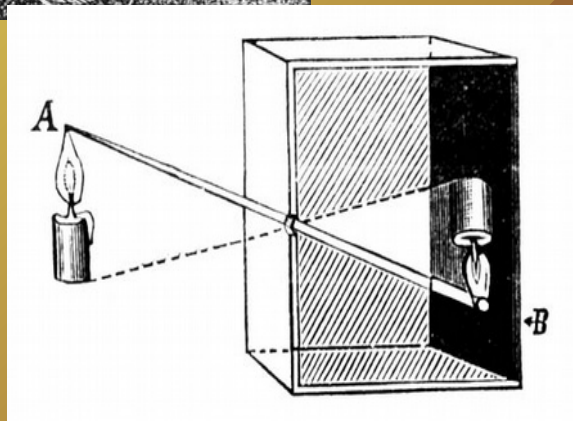
Light is both *matter & energy* → energy equilibrium achieved through photon traveling according to optical laws → **Photon Tracing**

Ray Tracing

Light travel in straight rays, following strict geometrical (optical) rules. Perspective image formation has been used by artists (and cameras)

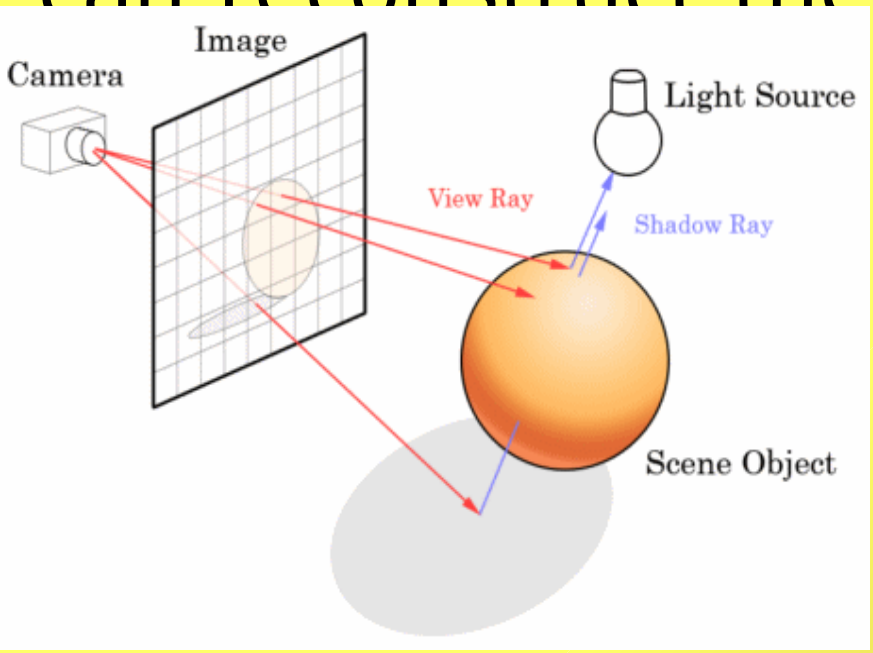


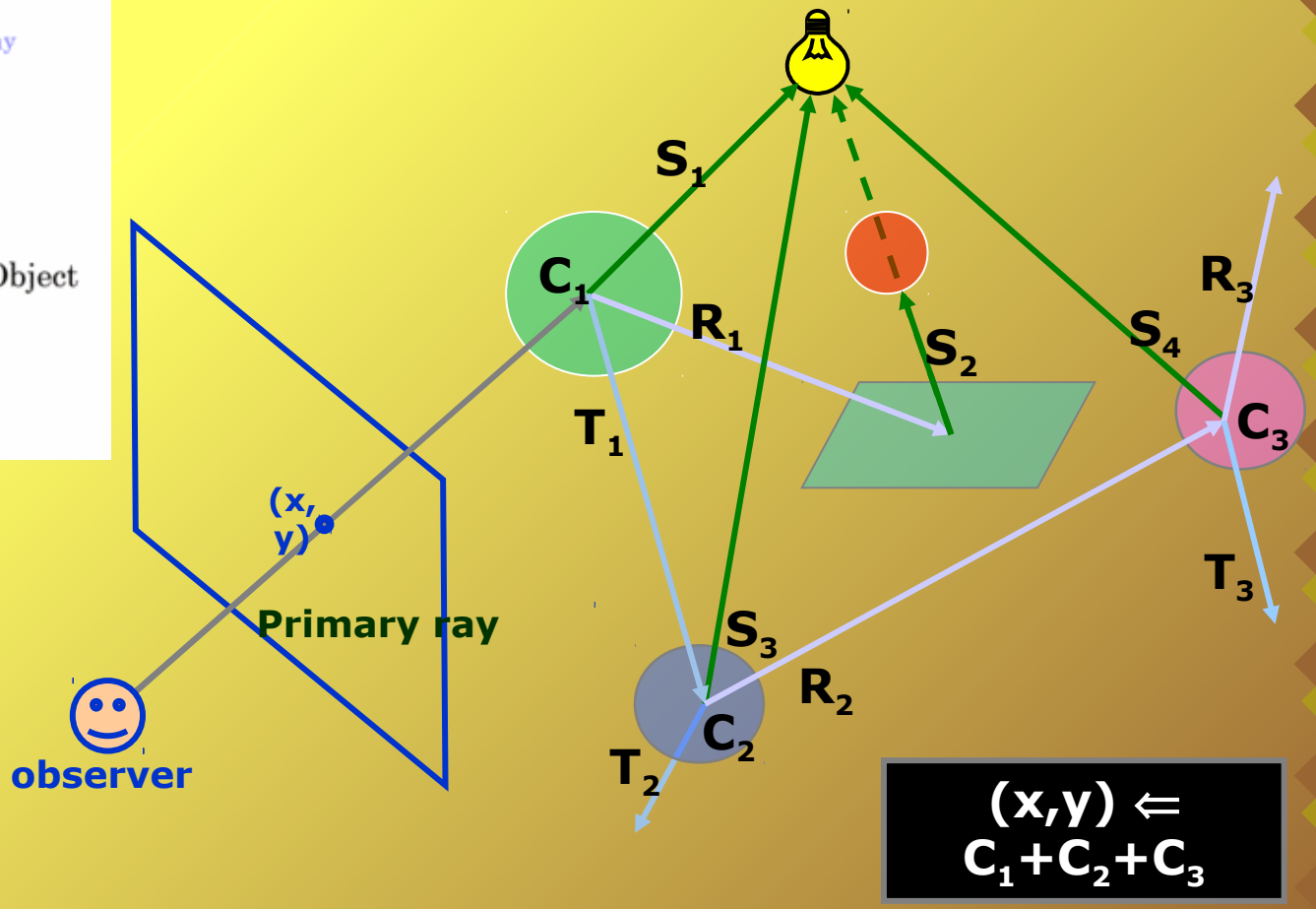
The Dührer grid



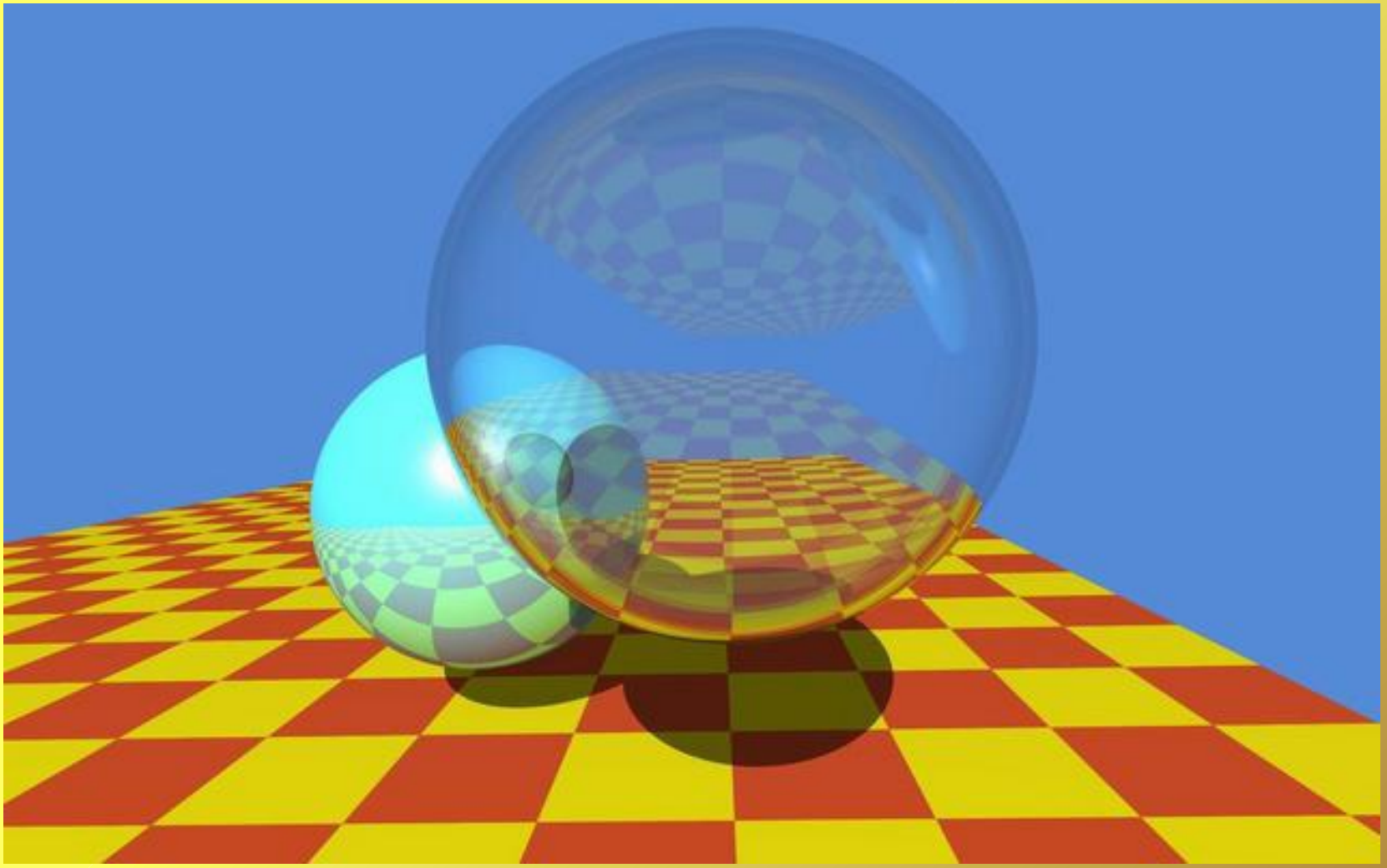
Ray Tracing

If i follow back **each** ray entering my eyes, I can reconstruct the image I would see...





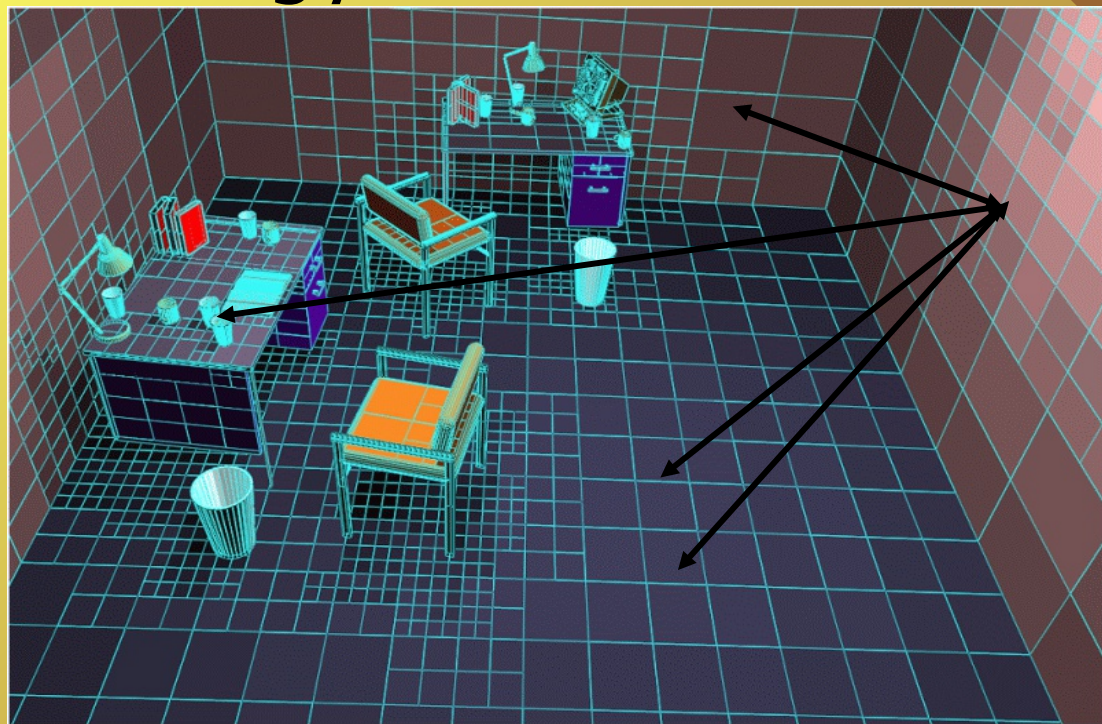
Ray Tracing



Ray Tracing

Radiosity

Light is energy and, when emitted from a light source is just bounced/absorbed/transmitted around the scene. Idea: divide the scene in patches, calculate how energy moves around the scene until no more energy remains...



Radiosity

Photon Mapping

Light is „carried“ by photons, let us just sprinkle some (millions) in the scene and follow each one to see how light goes around

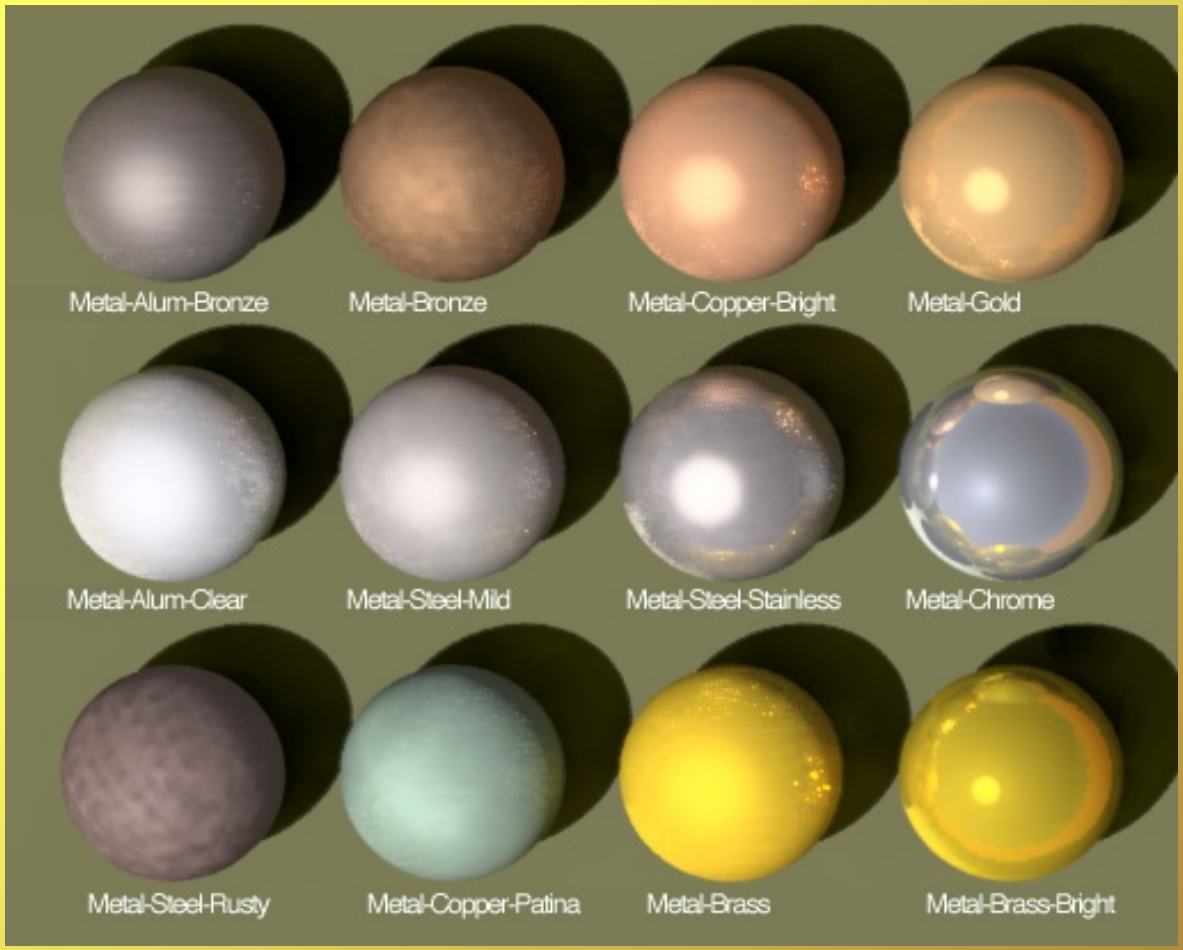
Combining all

The various techniques started separate, but are converging. Most existing tool mix and match different strategies and algorithms, to combine the strengths of the various approaches...

Combining all

Not only light

I lied (again), the added realism is also due to much more complicated calculation on light-matter interaction...



Material(s)

Materials may be (and have been often) described using LIGHTING/SURFACE MODELS, by specifying a series of parameters ...

Diffuse color, roughness

Specular color, “hardness“, shinyness

Specularity, reflection level, glossyness

Transparency, translucency, IOR

Emissive color, intensity, fosforescence

……..

!! all these parames may be defined using textures !!

Material(s)

Materials mdoels are nice, but limited in variability.

The next step is WRITE A PROGRAM that calculates the visible color of a point on the surface, given the illumination...

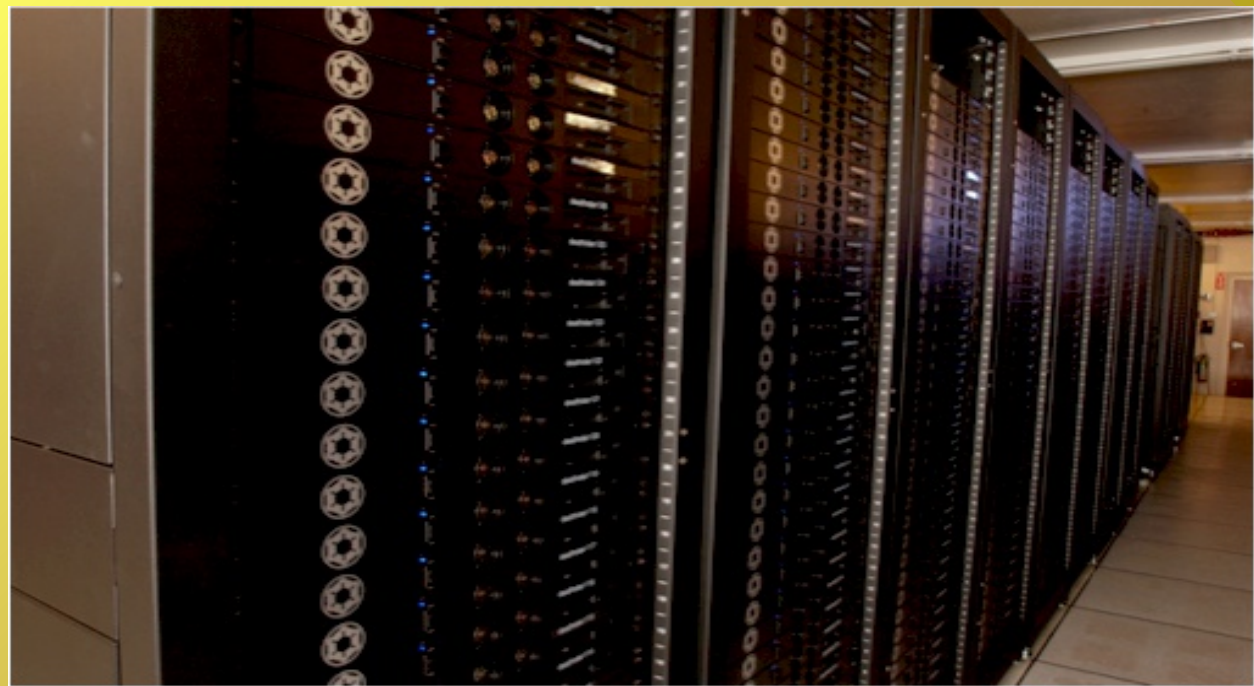
This is called a "**SHADER**"

A shader is a piece of code which is used to specify some aspect of the rendering, from the geometry, to the screen projection, from the light-material interaction to the final image filtering.

```
surface metal(float Ka = 1; float Ks = 1; float rough = 0.1;)
{
normal Nf = faceforward(normalize(N), I);
vector V = - normalize(I);
Oi = Os;
Ci = Os * Cs * (Ka * ambient() + Ks * specular(Nf, V, rough));   }
```

Back to realtime

The computational power required to follow these methods is too high for realtime, but many of the more accurate lighting and material calculations have been transferred from offline to realtime...



Modern 3D Video Cards

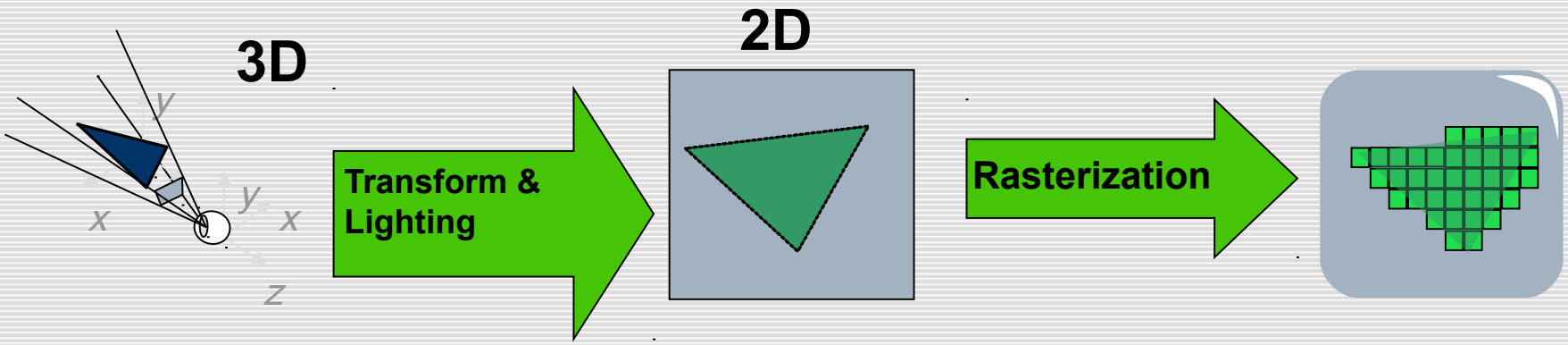
The first cause for the 3D explosion has been the availability of “accelerated“ 3D video cards.

First of all, all the mathematical calculation for scene setup and perspective projection are managed by the video card that, on this specific task, is much faster than the CPU...

But this does not explain why realtime 3D is nowadays so nice to see...

Cenni storici sull'hardware grafico (1)

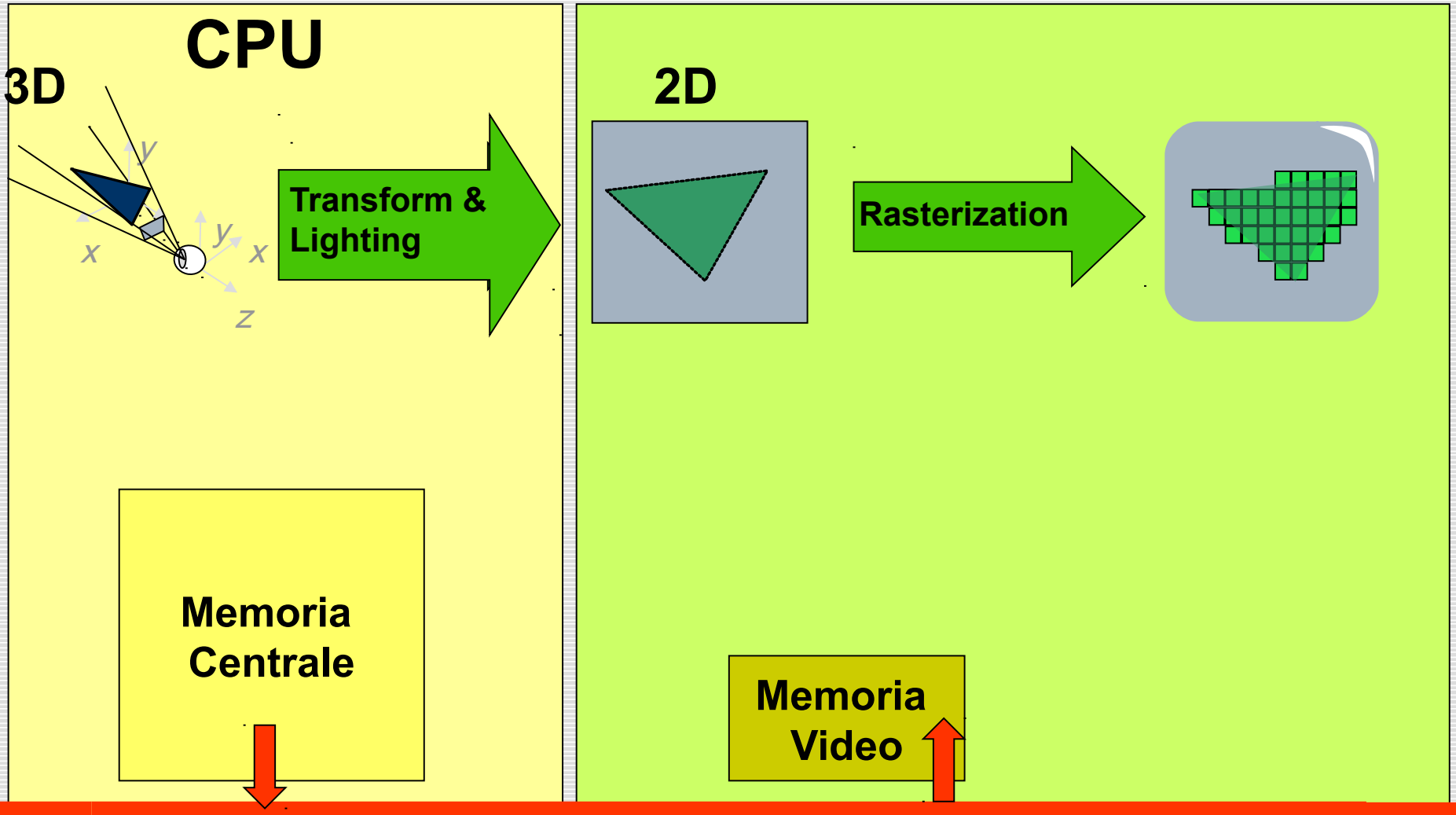
- Lo schema base della pipeline di rendering



- Fino a metà degli anni 90 tutte le parti della pipeline erano eseguite dalla CPU

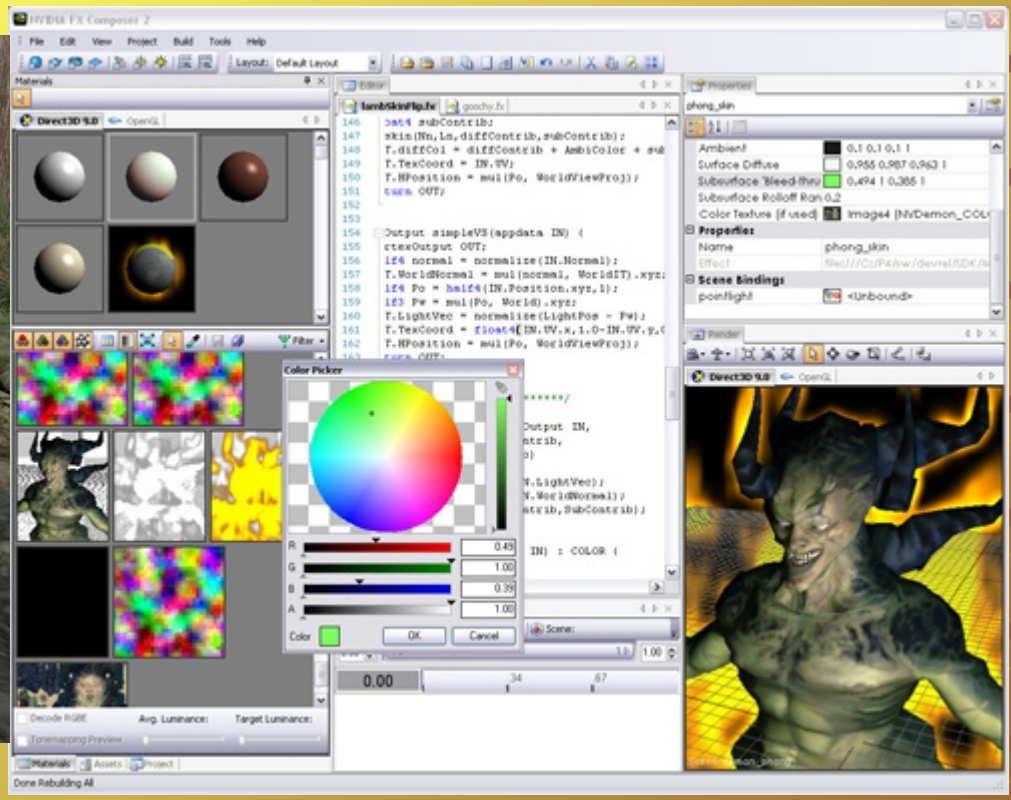
Cenni storici sull'hardware grafico (2)

- 1995-1997: 3DFX Voodoo



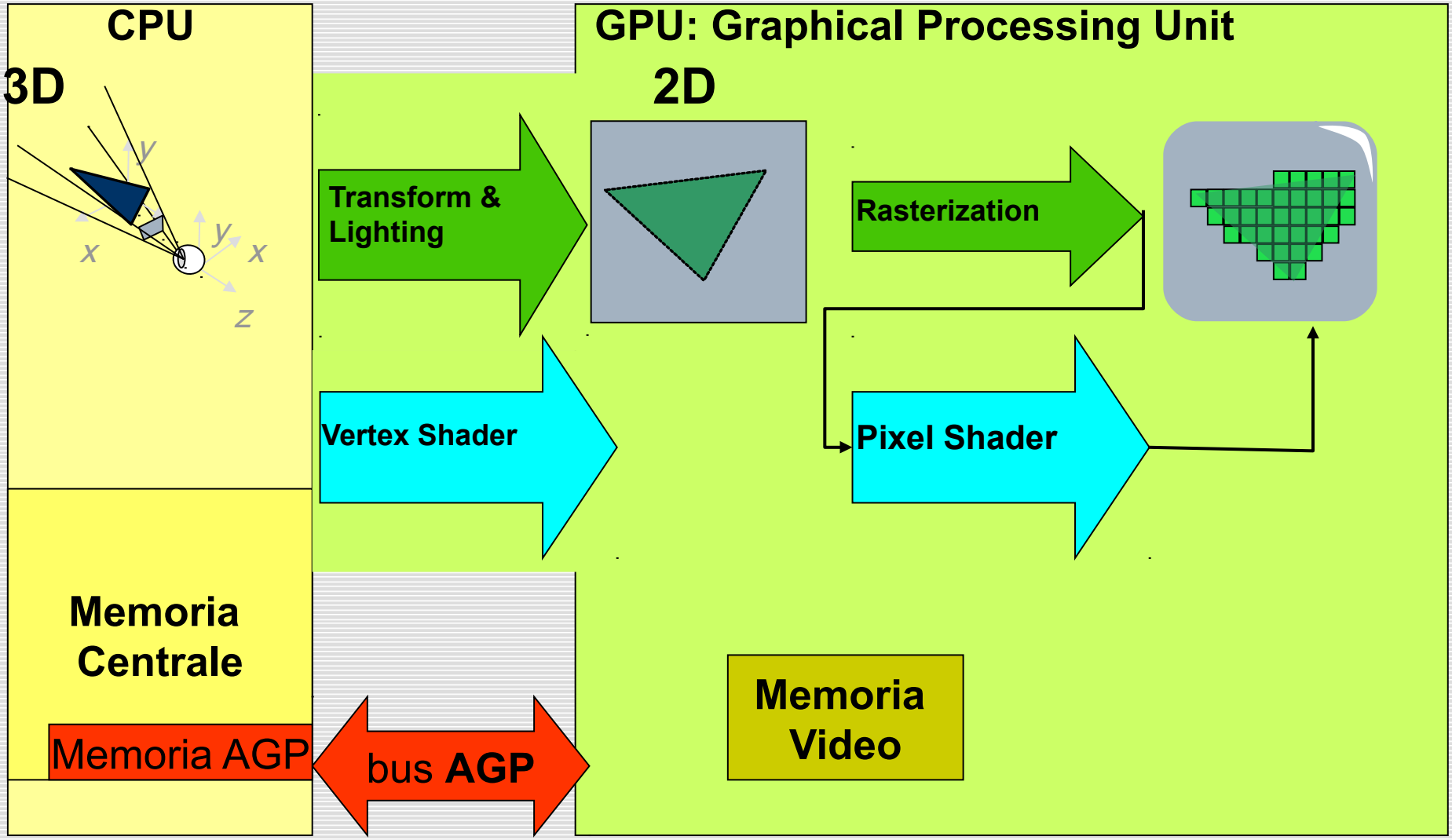
Modern 3D Video Cards

Modern videocards are “programmable“, that means it is possible to employ SHADERS to mimic some of the lighting calculation and material simulation from the offline rendering methods, obtaining stunning results...



Cenni storici sull'hardware grafico (3)

- 2002-3: Pixel (o fragment) Shader (GeForceFX, Radeon 8500)



Cenni storici sull'hardware grafico (4)

- 2007: geometry shader, stesso hardware per tutti gli shaders (NVidia 8800)