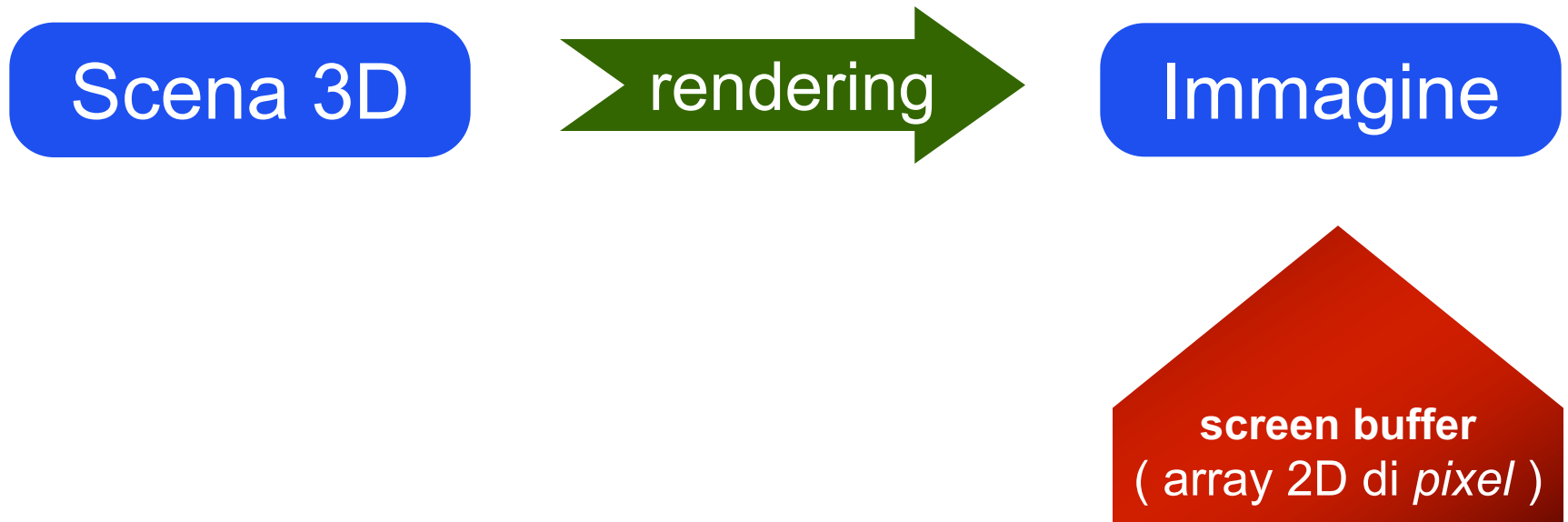


Lighting & Texturing

3D GM&P 23/24

Paolo Cignoni

Rendering



Rendering nei games

- Real time
 - (20 o) 30 o 60 FPS
- Hardware based
 - Pipelined, stream processing
- Complessità:
 - Lineare col numero di primitive
- Classe di algoritmi:
 - rasterization based

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 :

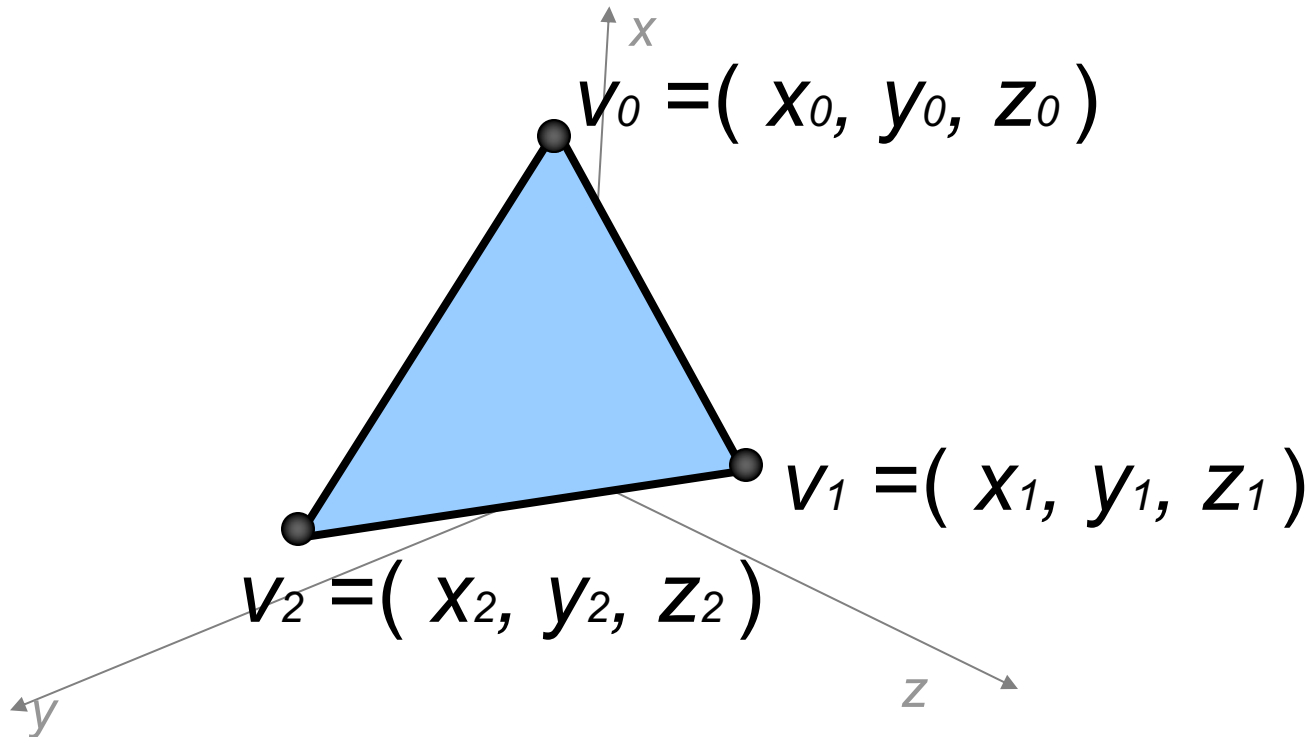
find color for o

color p with it

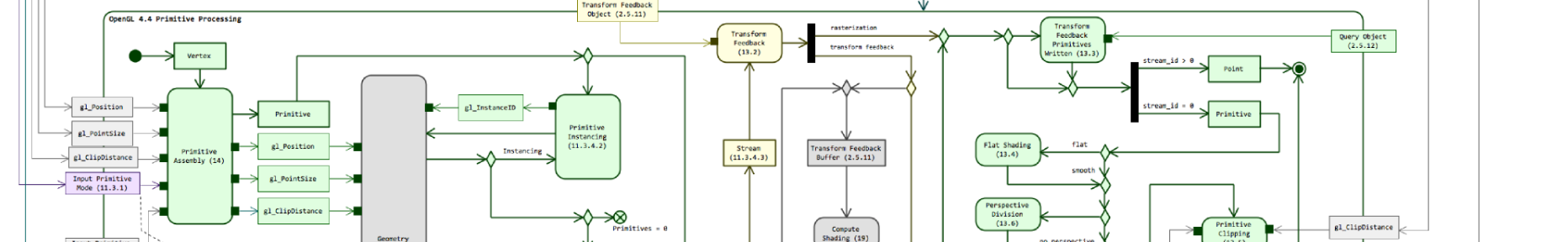
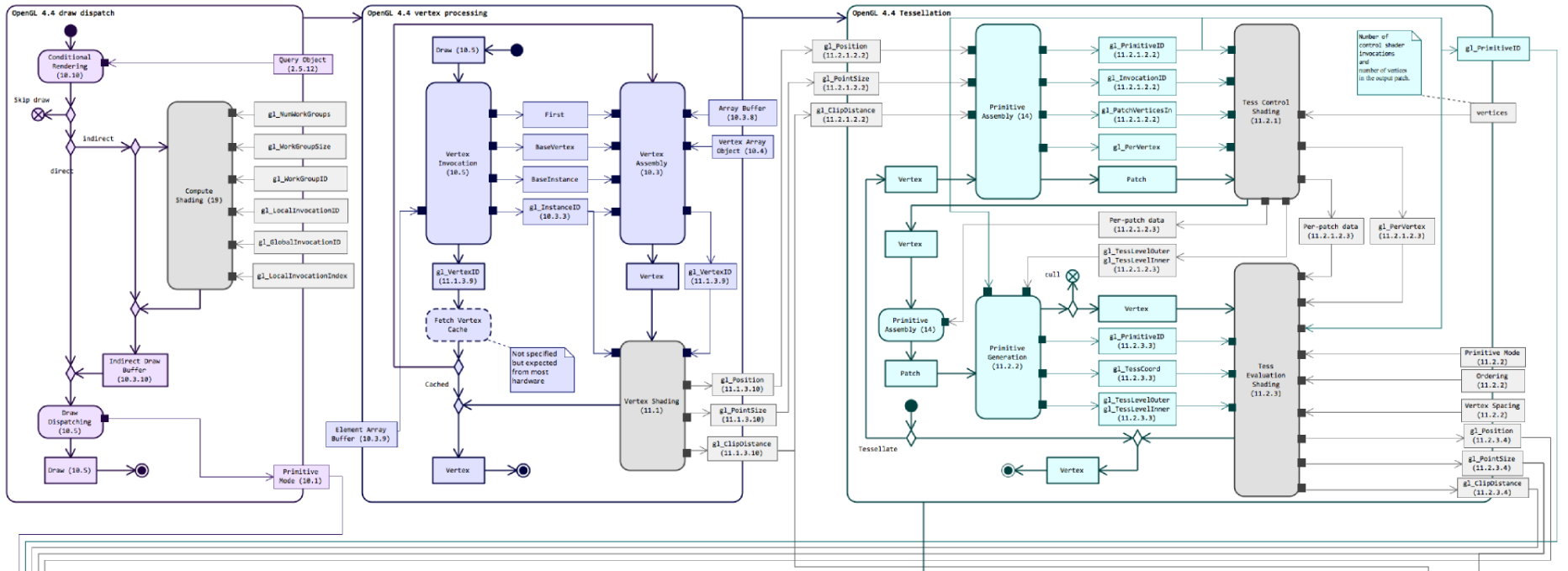
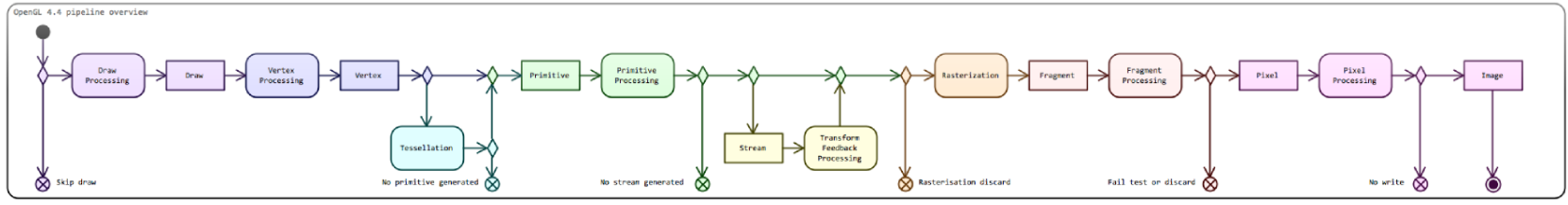
PROJECTION
3D \rightarrow 2D
(aka TRANSFORM)

LIGHTING

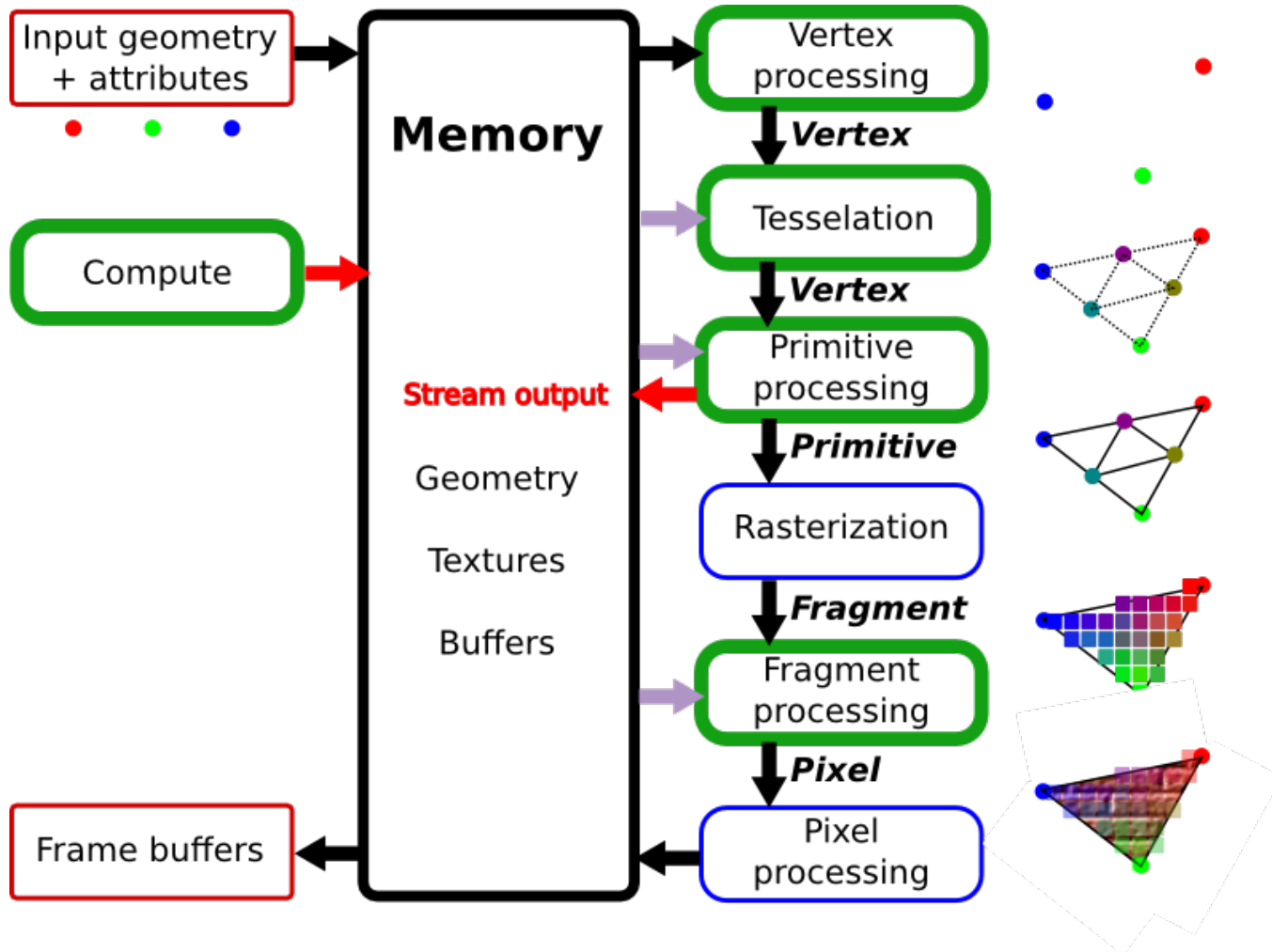
Il triangolo



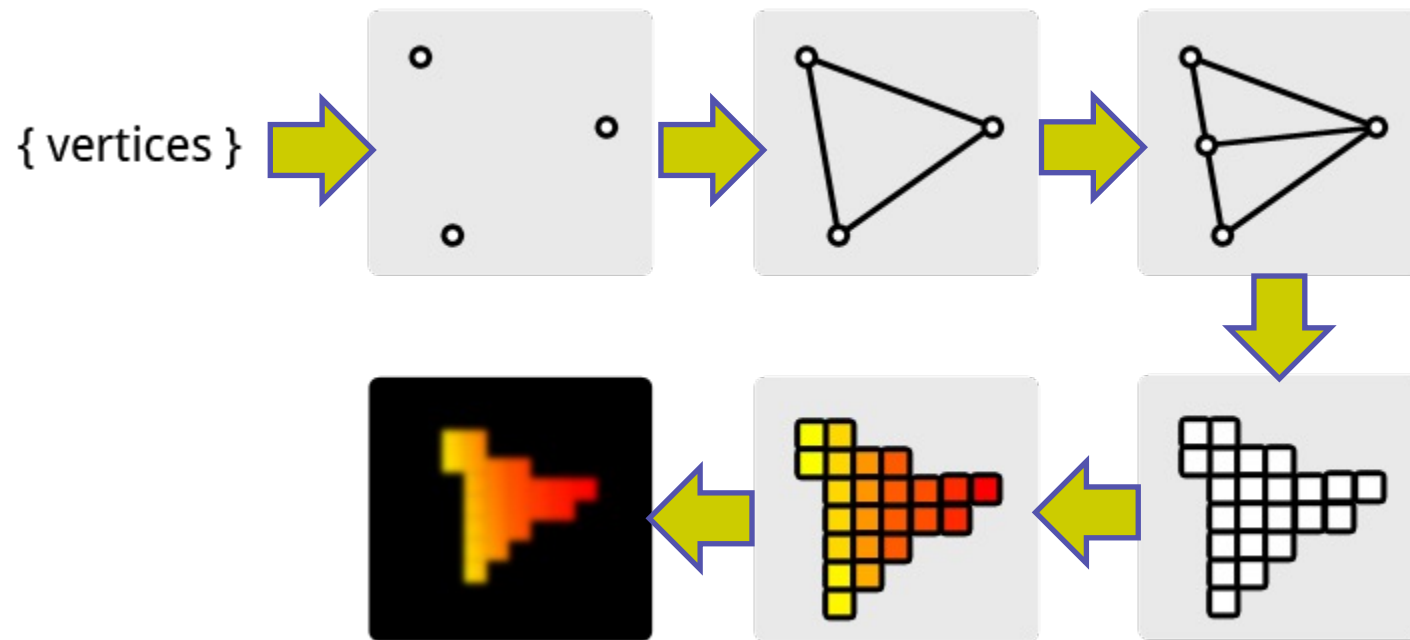
GPU pipeline



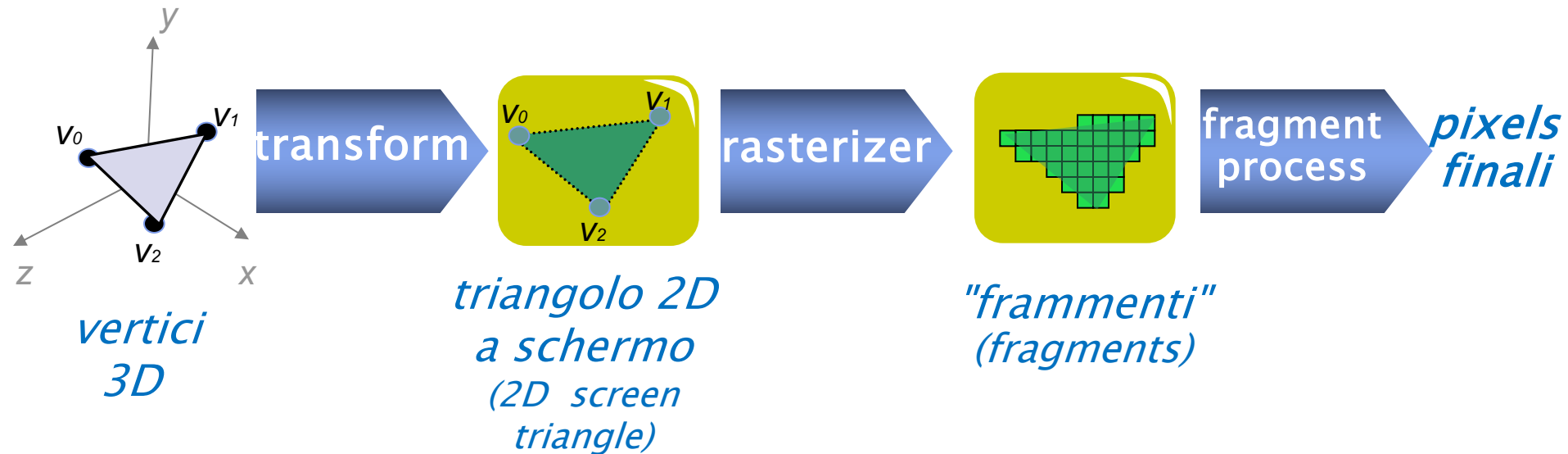
GPU pipeline – simplified



GPU pipeline – simplified *more*



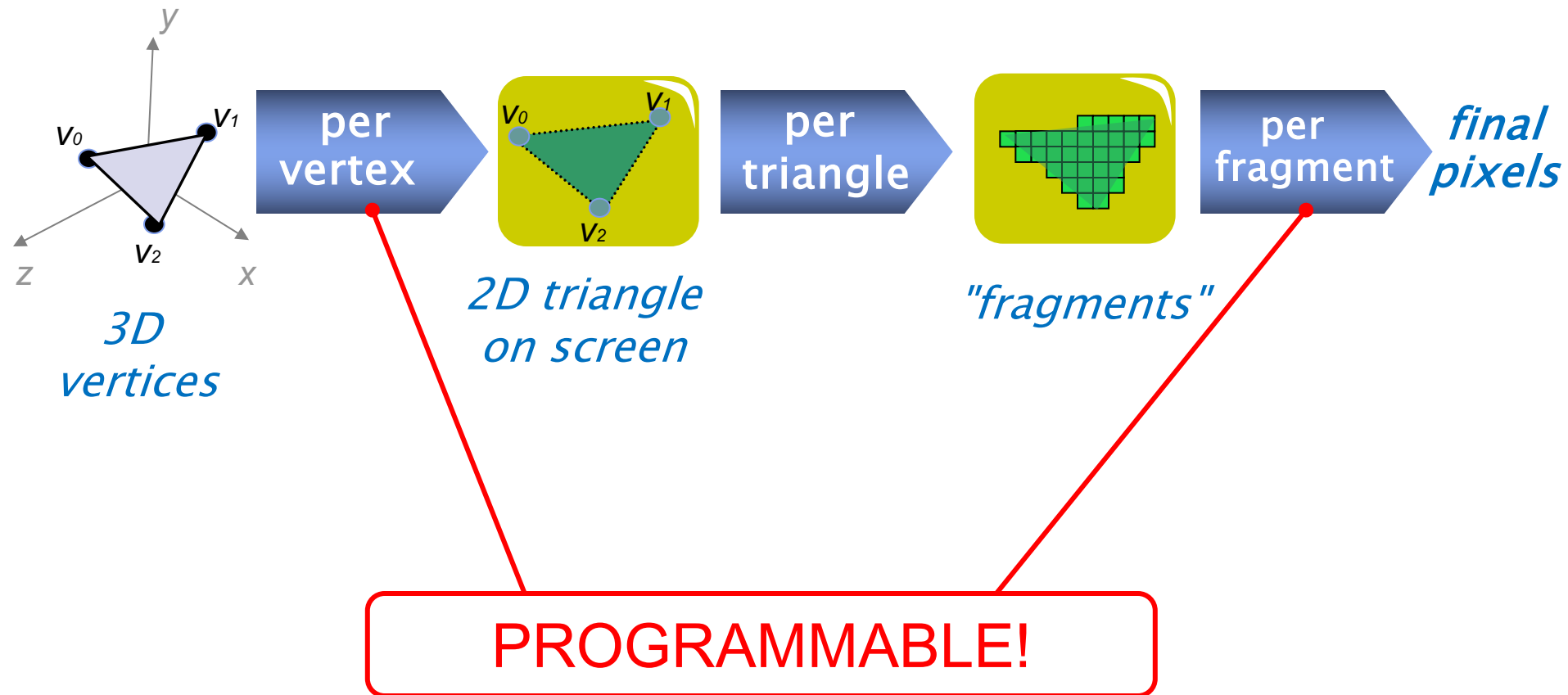
GPU pipeline – simplified even *more*



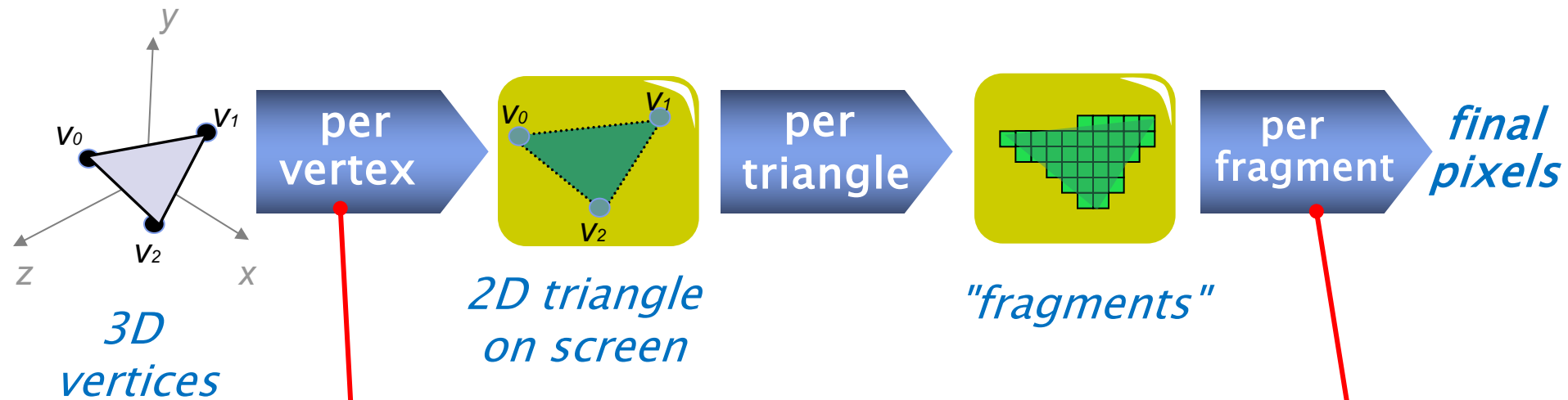
Rasterization based rendering: base schema

- Per vertice: (vertex shader)
 - skinning (from rest pose to current pose)
 - transform (from object space to screen space)
- Per triangle: (rasterizer)
 - rasterization
 - interpolation of per-vertex data
- Per fragment: (fragment shader)
 - lighting (from normal + lights + material to RGB)
 - texturing
 - alpha kill
- Per pixel: (after the fragment shader)
 - depth test
 - alpha blend

Rasterization-Based Rendering



Rasterization-Based Rendering



The user-defined
"Vertex Shader"
(or vertex program)

The user-defined
"Fragment Shader"
(or pixel program)

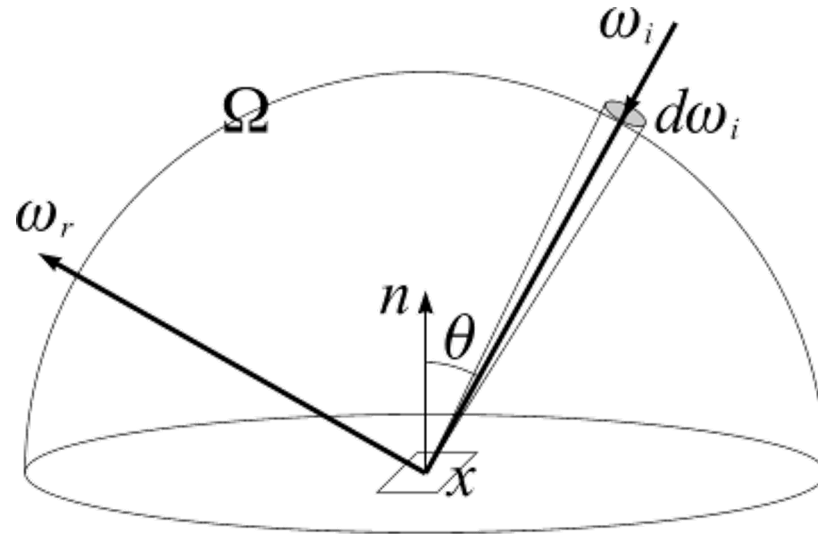
Shading languages

- High level:
 - **HLSL** (High Level Shader Language, Direct3D, Microsoft)
 - **GLSL** (OpenGL Shading Language)
 - **CG** (C for Graphics, Nvidia -- deprecated)
- Low lever:
 - **ARB** Shader Program
(the “assembler” of GPU -- deprecated)

Modelli di illuminazione

- **Modello di illuminazione:** formulazione matematica dell'equazione del trasporto dell'energia luminosa
- L'equazione che risolve questo problema: **equazione di illuminazione**
- **Lighting:** calcolo del bilancio luminoso
- **Shading:** calcolo del colore di ogni pixel dell'immagine

The Rendering Equation



$$L_o(x, \vec{\omega}_r) = L_e(x, \vec{\omega}_r) + L_r(x, \Omega)$$

$$L_o(x, \vec{\omega}_r) = L_e(x, \vec{\omega}_r) + \int_{\Omega} f_r(x, \vec{\omega}_i, \vec{\omega}_r) L_i(x, \vec{\omega}_i) (\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i$$

The Rendering Equation

$$L_o(x, \vec{\omega}_r) = L_e(x, \vec{\omega}_r) + L_r(x, \Omega)$$

- The light (brightness) in a point of the scene is the sum of the reflected and emitted light

$$L_o(x, \vec{\omega}_r) = L_e(x, \vec{\omega}_r) + \int_{\Omega} f_r(x, \vec{\omega}_i, \vec{\omega}_r) L_i(x, \vec{\omega}_i) (\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i$$

- Reflected light is an integral, summing all the light contributions from all the directions taking into account the angle of arrival over the surface

The Rendering Equation: parameters

x Surface point where we compute light;

$\vec{\omega}_r$ Viewpoint direction

$\vec{\omega}_i$ Incident light direction

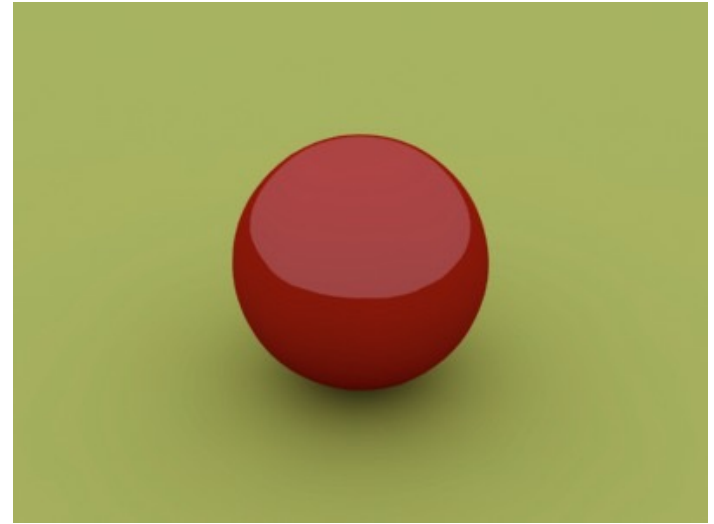
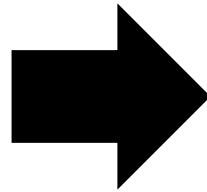
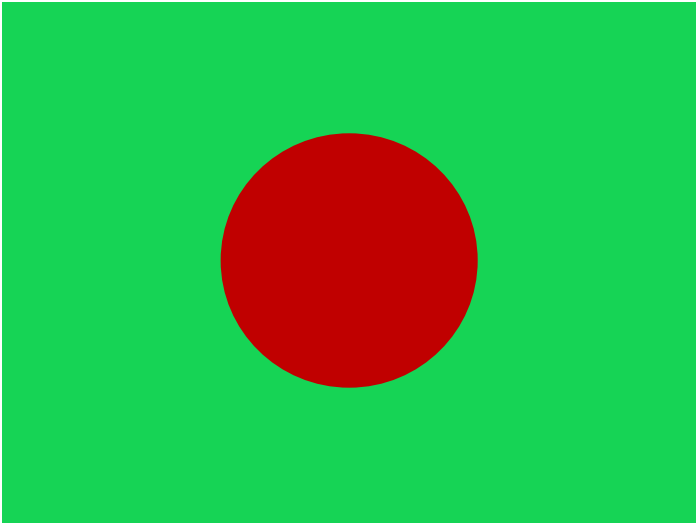
f_r Function estimating the amount of light arriving from that direction

$\vec{\omega}_i \cdot \vec{n}$ Cosine between the normal of the surface and the incidence angle

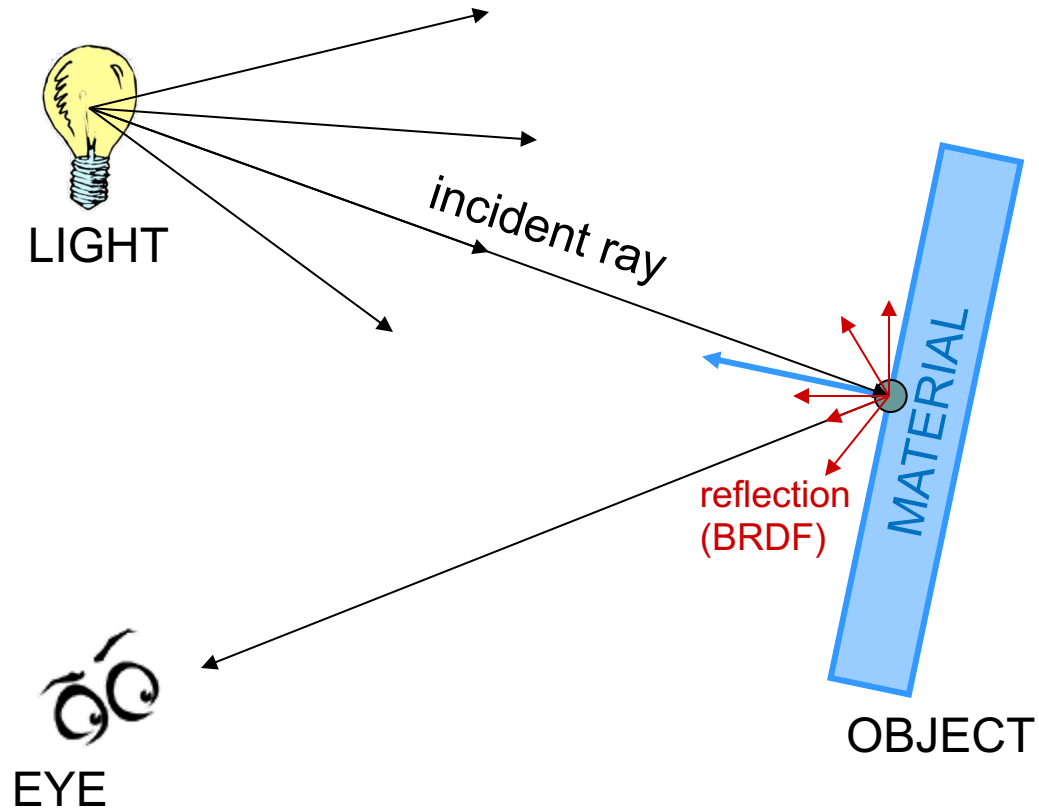
The Rendering Equation

- Calcolo esatto dell'equazione della radianza: operazione complessa e molto costosa
- Sistema di grafica interattiva: formula utilizzabile per tutti i punti della scena più volte al secondo
- Semplificazione dell'equazione

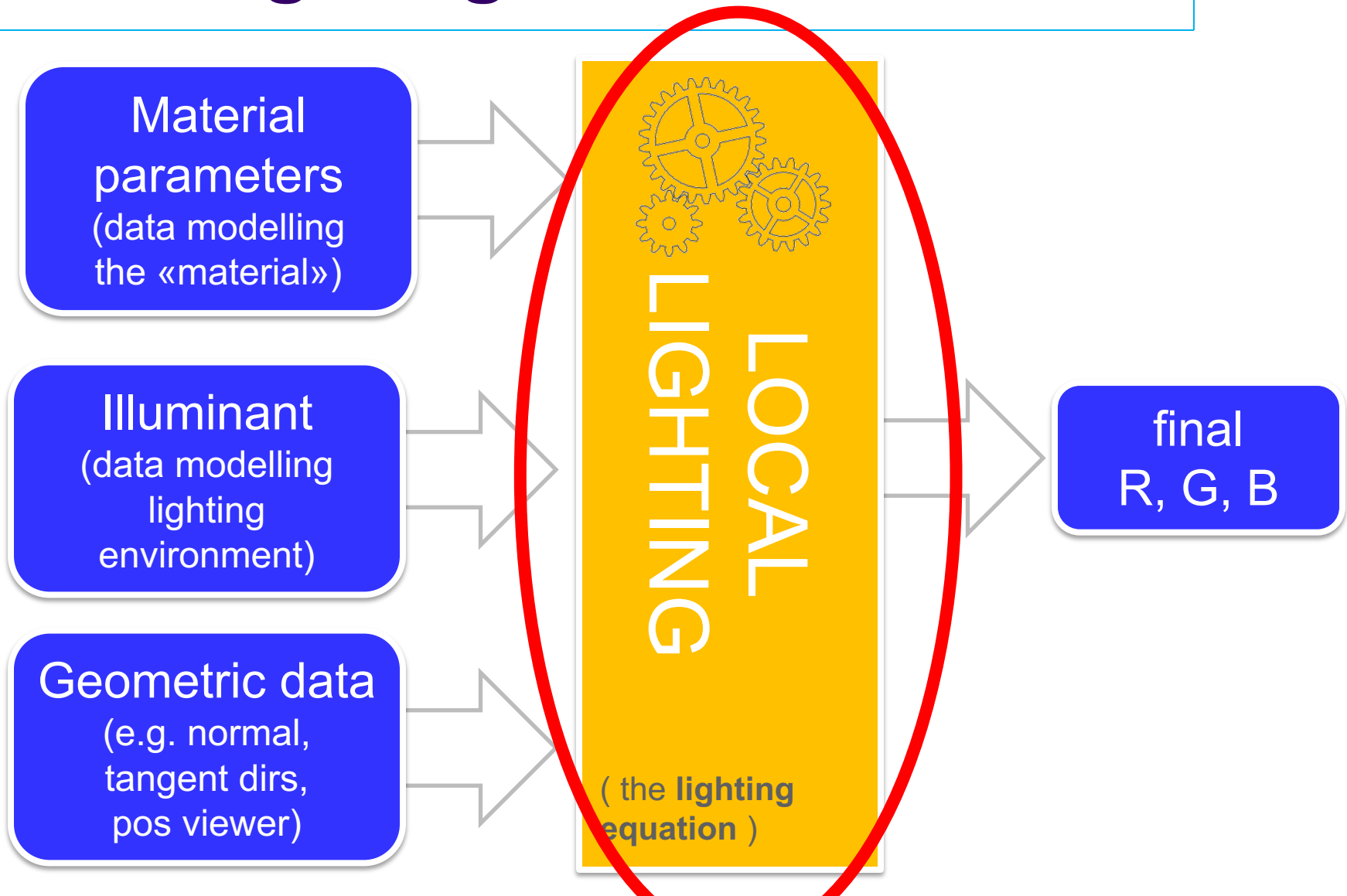
Rendering effects: local lighting



Local lighting



Local lighting



Lighting equations

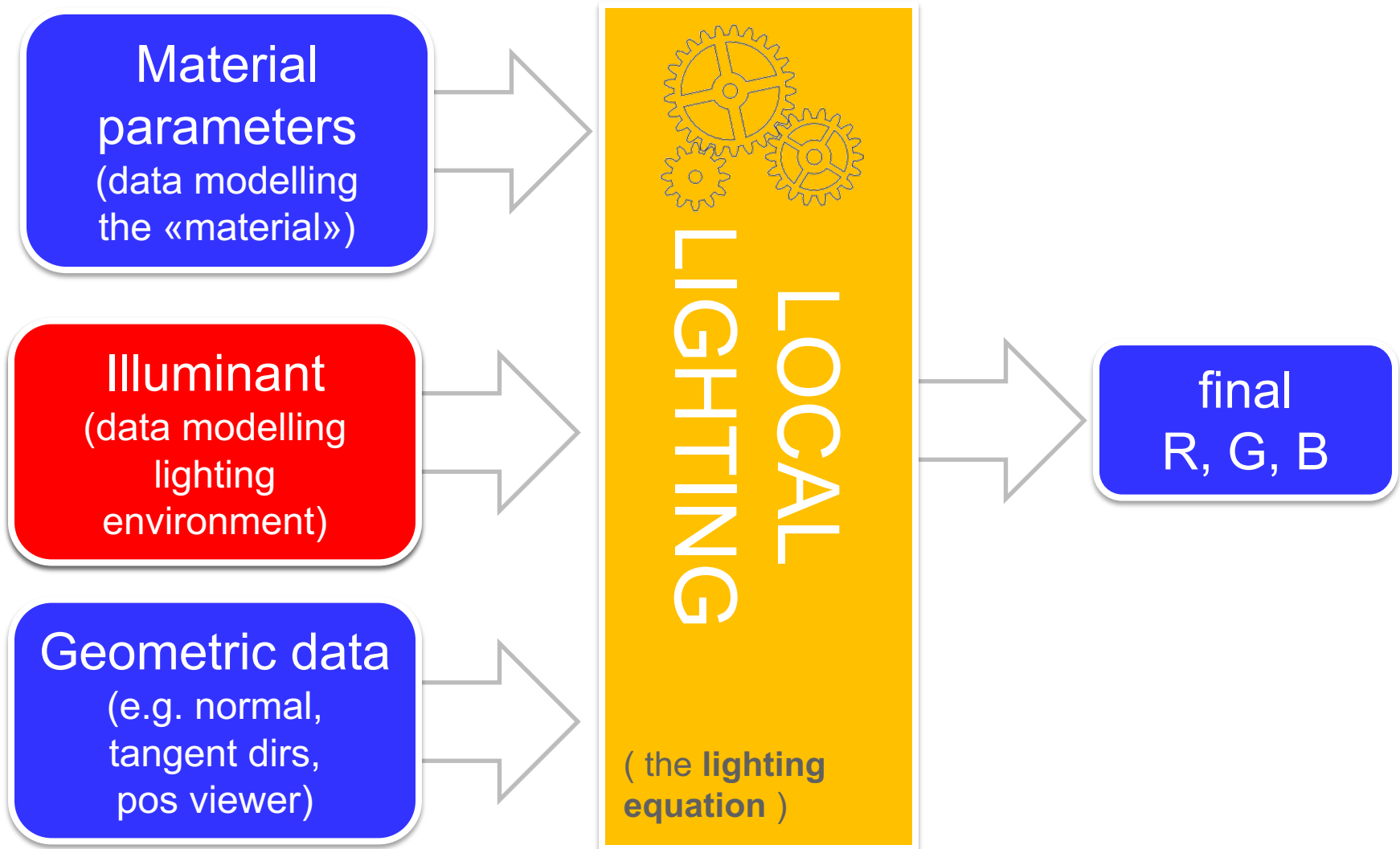
- Many different equations...
 - Lambertian
 - Blinn-Phong } simplest, most commonly used
 - Beckmann
 - Heidrich–Seidel
 - Cook–Torrance
 - Ward (anisotropic)
 - ...
 - add Fresnel effects
- Varying levels of
 - complexity
 - realism
 - (some are *physically based*, some are... just tricks)
 - material parameters allowed
 - richness of effects

Lighting equations: most common cases

- Diffuse (aka Lambertian)
 - physically based
 - only dull materials
 - only material parameter:
 - base color
(aka albedo, aka “diffuse” color)
- Specular (aka Blinn-Phong)
 - just a trick
 - add simulated reflections (highlights)
 - additional material parameters:
 - specular intensity (or, color)
 - specular exponent (aka glossiness)

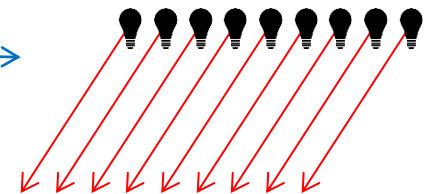
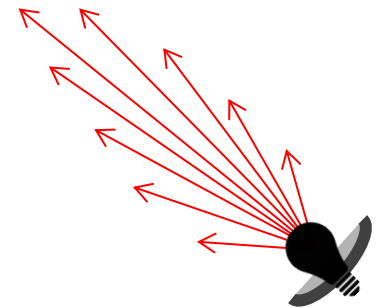
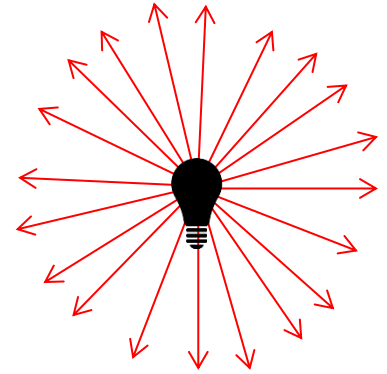


Local lighting



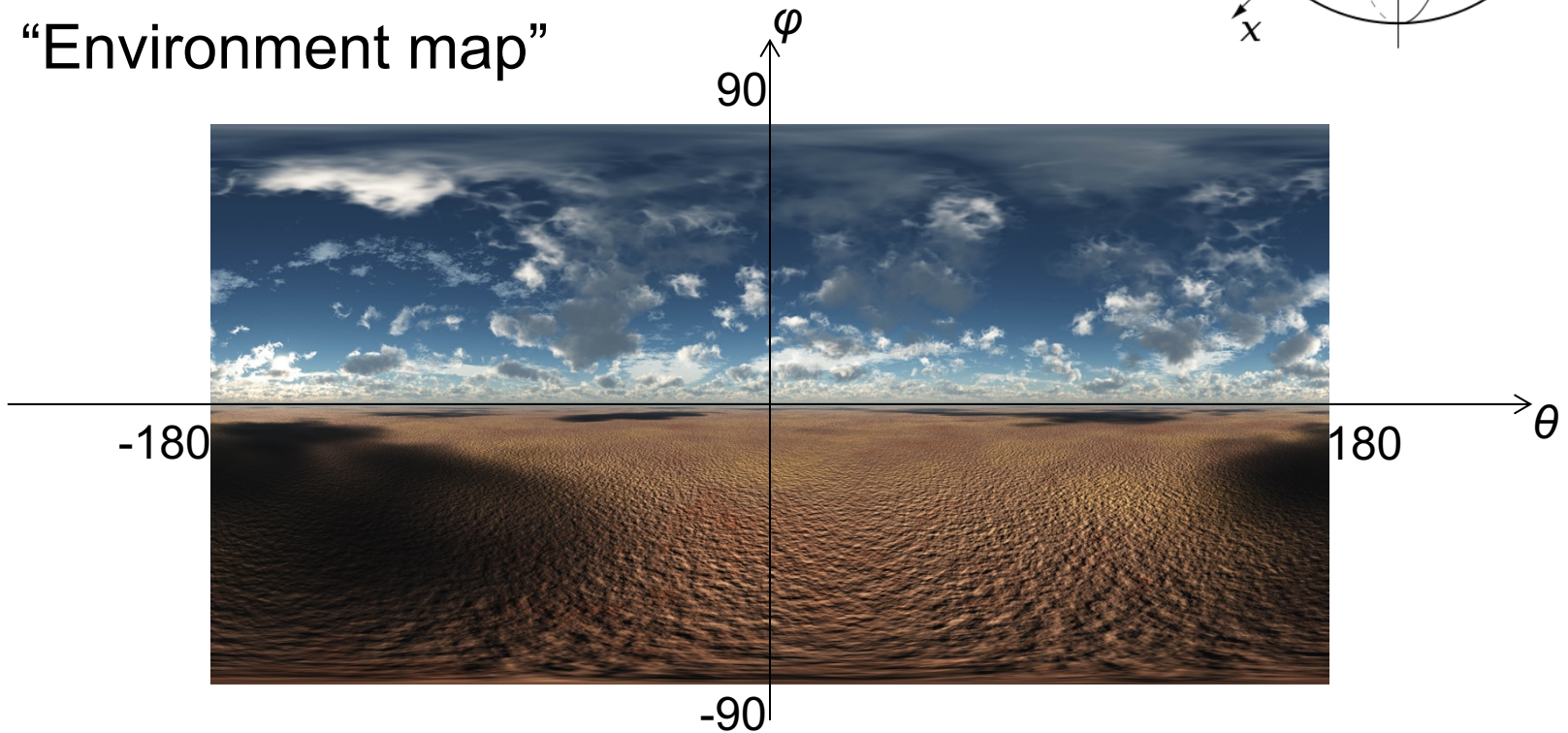
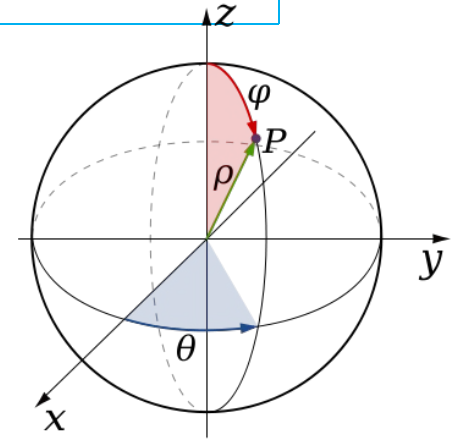
Illumination environments: discrete

- a finite set of “light sources”
 - few of them (usually 1-4)
- each sitting in a node of the scene graphs
- types:
 - point light sources
 - have: position
 - spot-lights
 - have: position, orientation, wideness (angle)
 - directional light sources
 - have: orientation only
- extra attributes:
 - color
 - intensities
 - (other minor attributes)

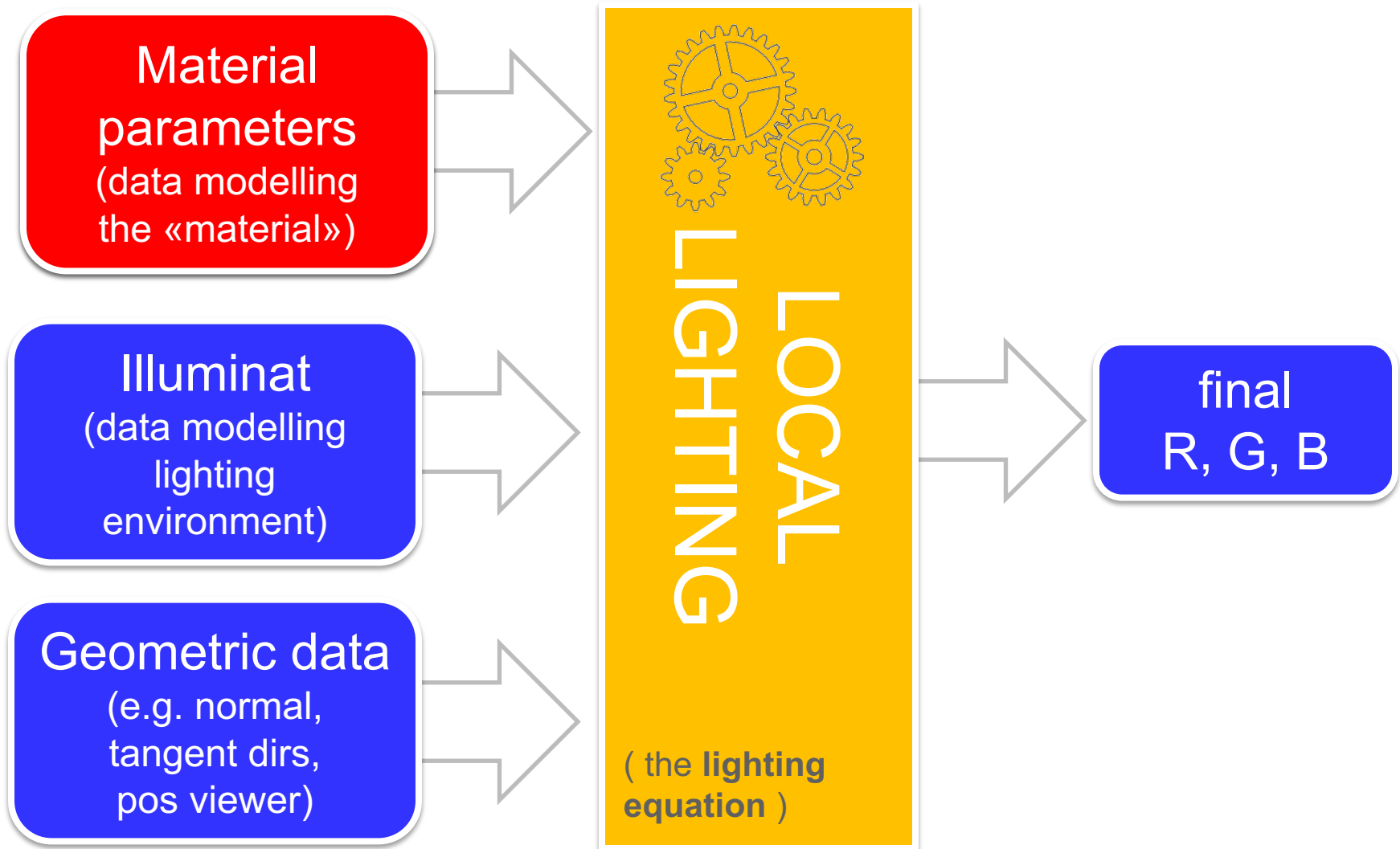


Illumination environments: densely sampled

- From each direction (on the sphere) a light intensity / color
- Asset to store that: “Environment map”



Local lighting



GPU rendering of a Mesh in a nutshell (reminder)

- Load...
 - store all data on **GPU RAM**
 - Geometry + Attributes
 - Connectivity
 - **Textures**
 - **Shaders**
 - **Material Parameters**
 - **Rendering Settings**
- ...and Fire!
 - send the command: “*do it*” !

THE MESH ASSET

THE MATERIAL ASSET

Terminology

- Material **parameters**
 - parameters modelling the optical behavior of physical object
 - part of the input of the lighting equation
- Material **asset**
 - an abstraction used by game engines
 - consisting of
 - a set of textures (e.g. diffuse + specular + normal map)
 - a set of shaders (vertex + fragment)
 - a set of global parameters (e.g. global glossiness)
 - rendering settings (e.g. back face culling?)
 - corresponds to the status of the rendering engines

Material parameters

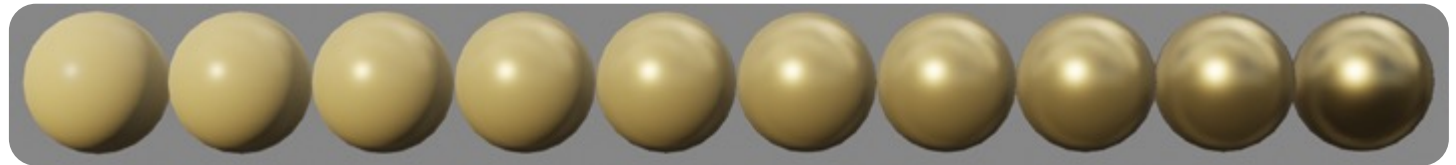


Authoring material parameters

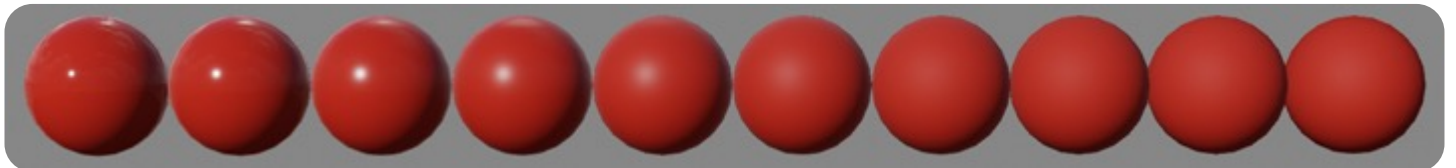
- Q: which materials parameters needs be defined?
A: depends on chosen lighting equation
- Idea:
game engine lets **material artist** choose *intuitively named* material parameters, then pick a **lighting equation** accordingly
 - the one best suiting them
 - “speak material-artist language”

Authoring material parameters


- Popular choice of “intuitive parameters”:
 - Base color (rgb)
 - Specularity (scalar)
 - “Metal-ness” (scalar)



- Roughness (scalar)



Physically based materials (PBR)

- Basically, a buzzword 😊
 - Meanings:
 - 1. use accurate material parameters
 - physically plausible
 - maybe measured
 - instead of: invented and tuned by intuition (by the material artist)
 - 2. keep each lighting element separated (e.g. its own texture)
 - use fewer shortcuts than usual
 - e.g. use:
 - **base color**: *one* texture
 - **baked AO**: *another* texture
 - instead of:
 - **base color** x **baked AO** : one texture
- Ambient Occlusion:
see later
- 

Texture mapping

(aka params how to store them)

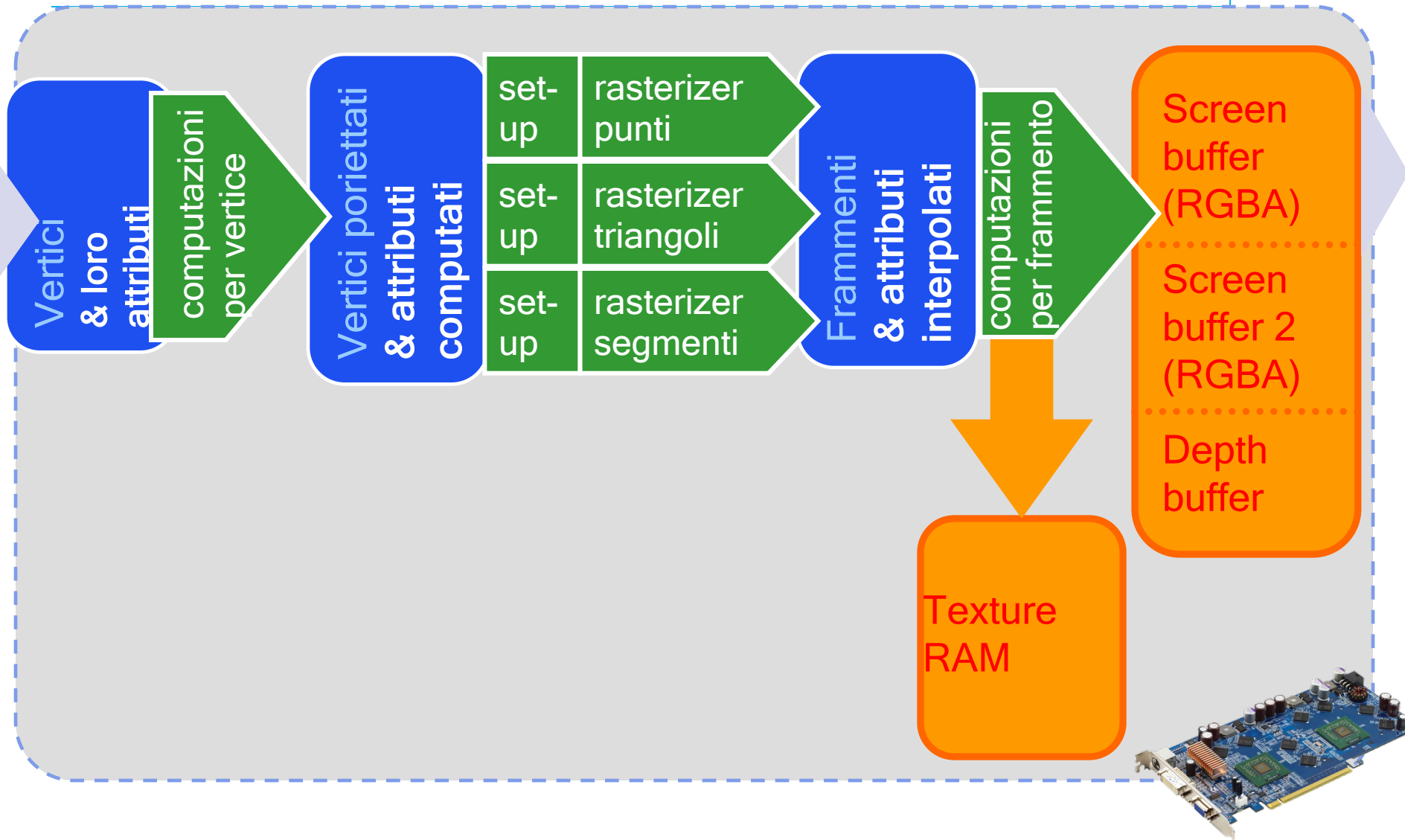
- Shading: funziona finché modello geometrico ha complessità della composizione del materiale
- La descrizione di un materiale non uniforme è corretta a partire da una rappresentazione geometrica con suddivisione (*tassellazione*) in primitive collegata alle discontinuità del materiale da rappresentare
- Altrimenti? Texture mapping!

Texture mapping

(aka params how to store them)

- Shading: funziona finché modello geometrico ha complessità della composizione del materiale
- La descrizione di un materiale non uniforme è corretta a partire da una rappresentazione geometrica con suddivisione (**tassellazione**) in primitive collegata alle discontinuità del materiale da rappresentare
- Altrimenti? Texture mapping!

Memoria RAM nelle schede grafiche



Texture Mapping

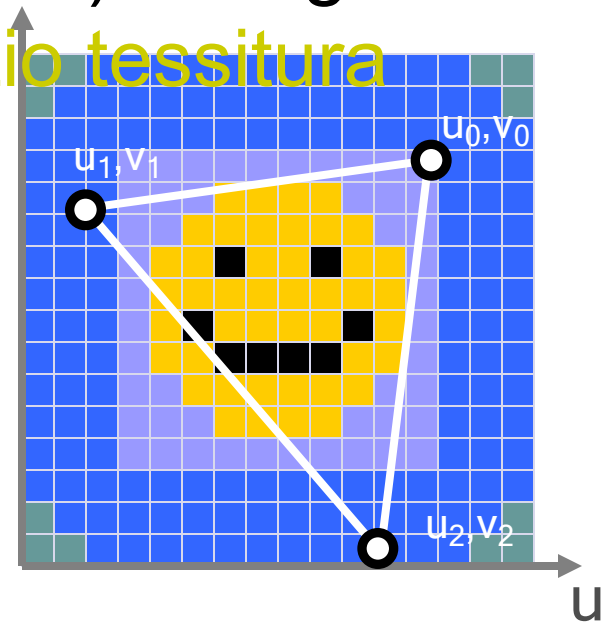
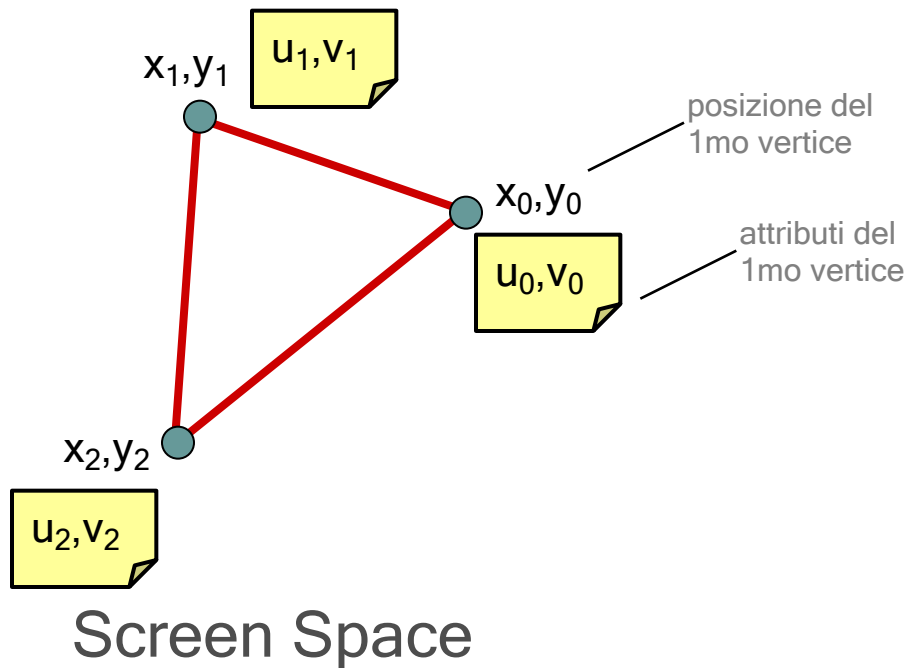
- Nelle operazioni per frammento si può accedere ad una RAM apposita: la **Texture RAM** strutturata in un insieme di Textures (“**tessiture**”)
- Ogni tessitura è un array 1D, 2D o 3D di Texels (campioni di tessitura) dello stesso tipo

Texels

- Sono esempi di texels:
 - Ogni texel un colore (componenti: R-G-B, o R-G-B-A): la tessitura è una “color-map”
 - Ogni texel una componente alpha: la tessitura è una “alpha-map”
 - Ogni texel una normale (componenti: X-Y-Z): la tessitura è una “normal-map” o “bump-map”
 - Ogni texel contiene un valore di specularità: la tessitura è una “shininess-map”

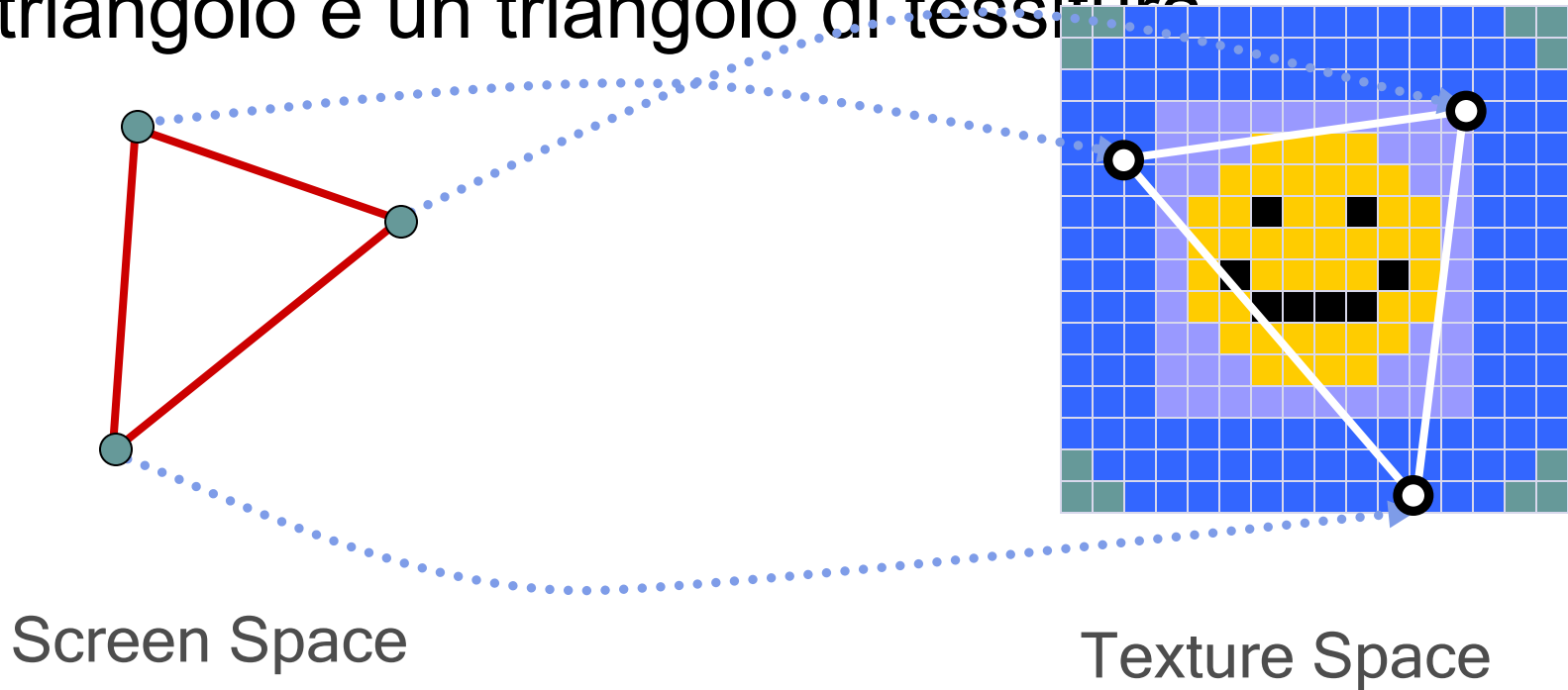
Texture Mapping

- Ad ogni **vertice** (di ogni triangolo) assegno le sue coordinate u, v nello **spazio tessitura**



Texture Mapping

- Così in pratica definisco un **mapping** fra il triangolo e un triangolo di tessitura

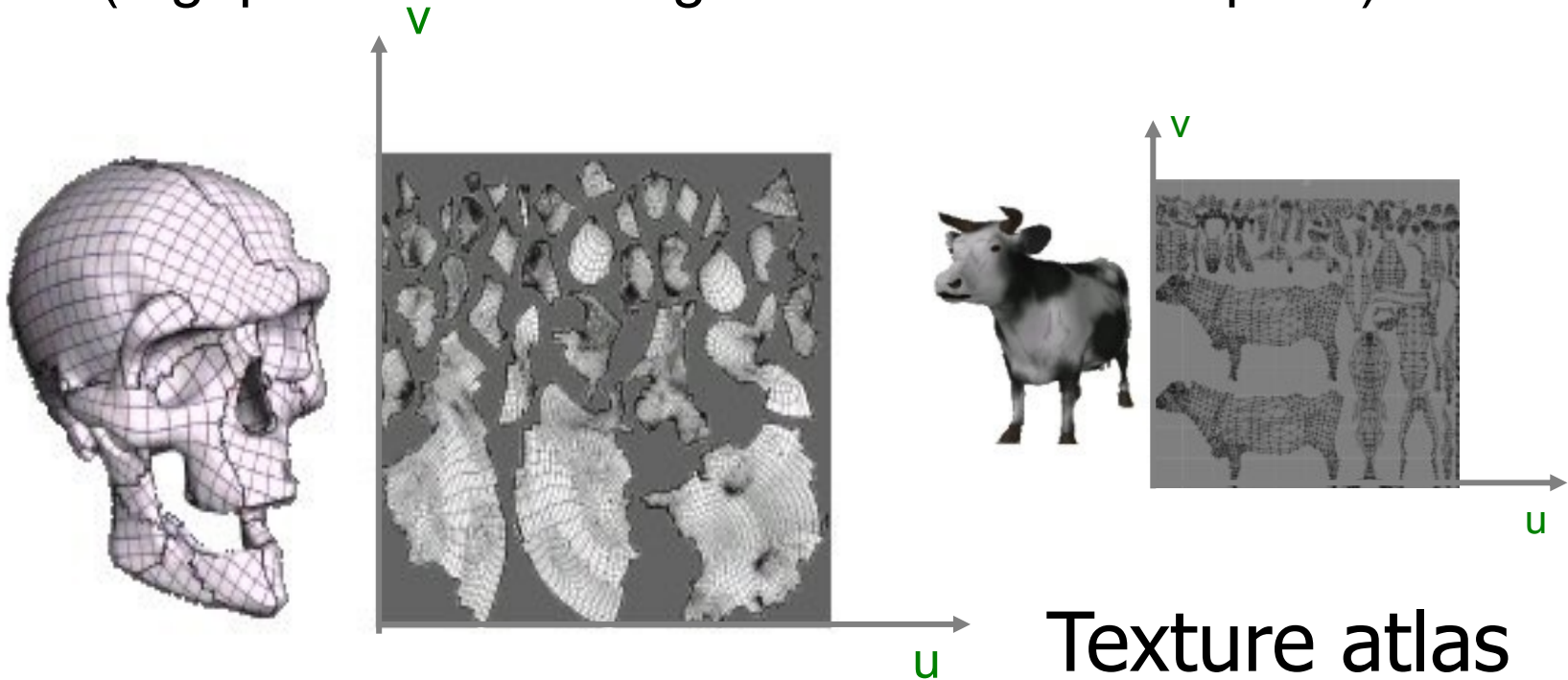


Assegnazione delle coordinate texture

- Due classi di soluzioni:
 - Calcolare le coordinate textures on-the-fly durante il rendering...
 - Precomputarle (e salvarle insieme alla mesh)
- Non esiste una soluzione ideale, dipende dall'applicazione che stiamo progettando
- Modelli con una sola texture l'avranno precomputata, per altri che variano dinamicamente l'assegneremo in rendering

u-v mapping: Not a trivial problem

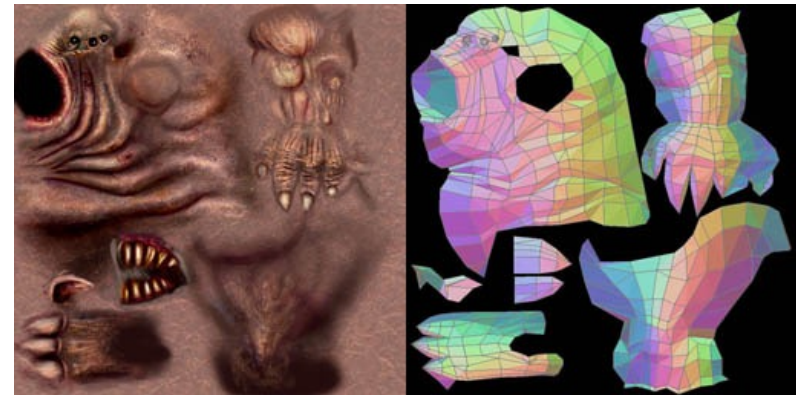
- Preprocessing step: assign to each vertex a proper uv coord
(e.g. place each triangle into the texture space)



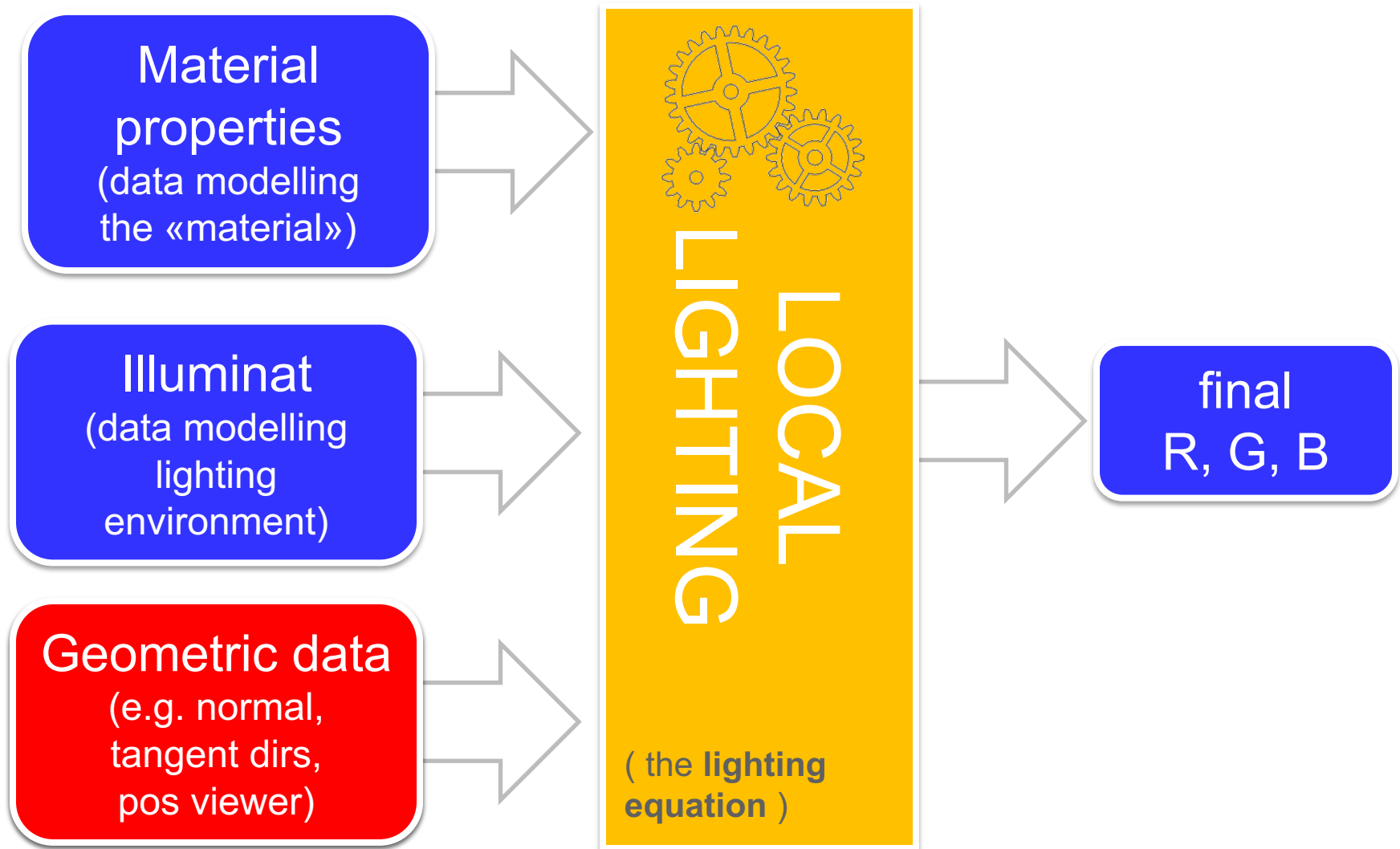
Problema difficile: u-v mapping



fatto a mano,
o automatizzato

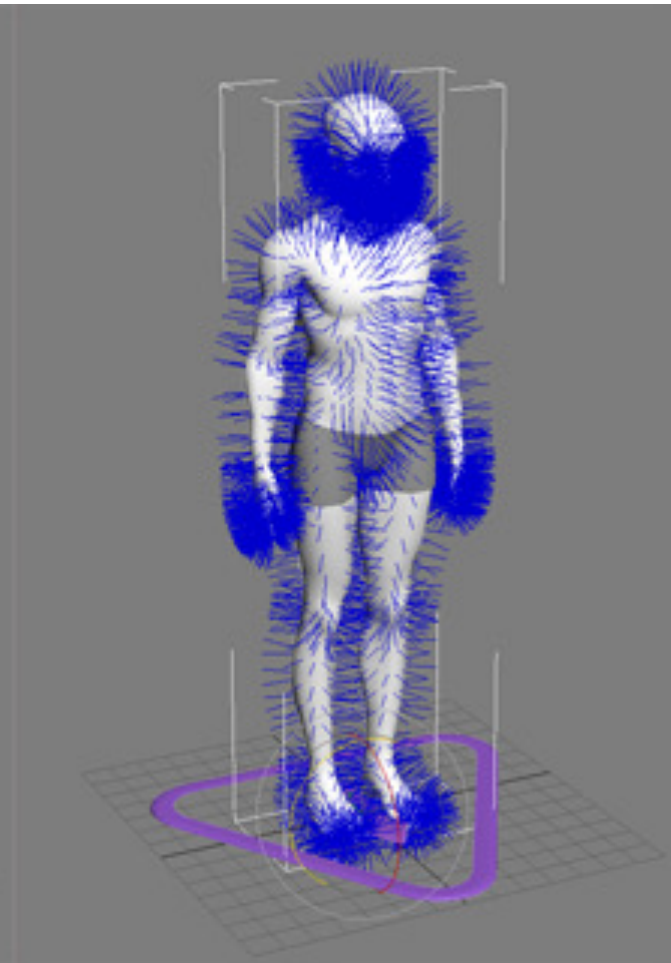
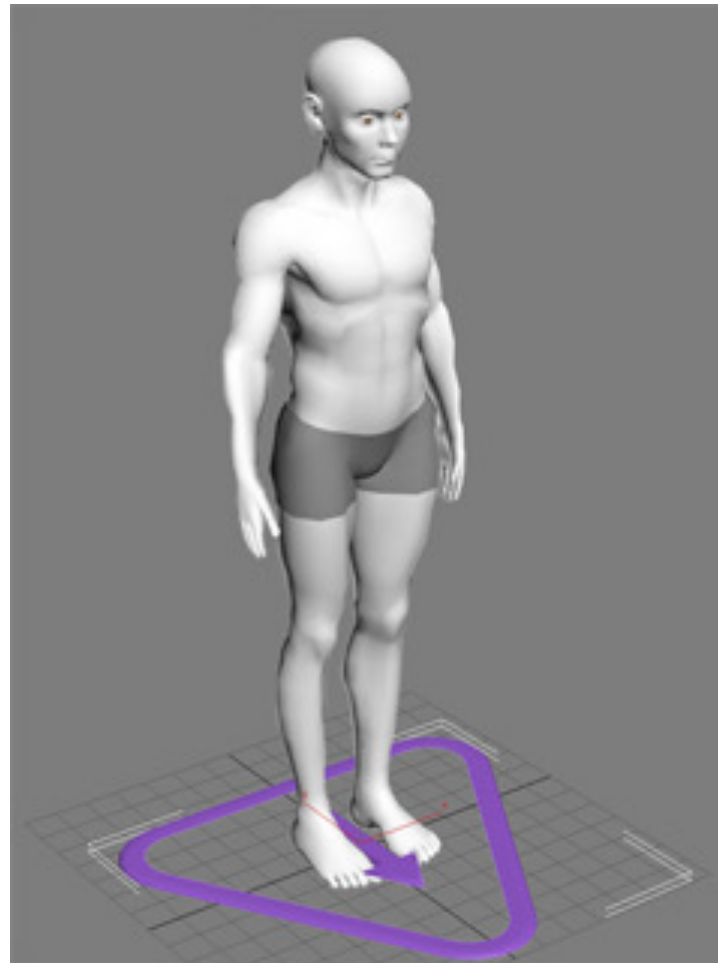


Local lighting in brief



Reminder: normals

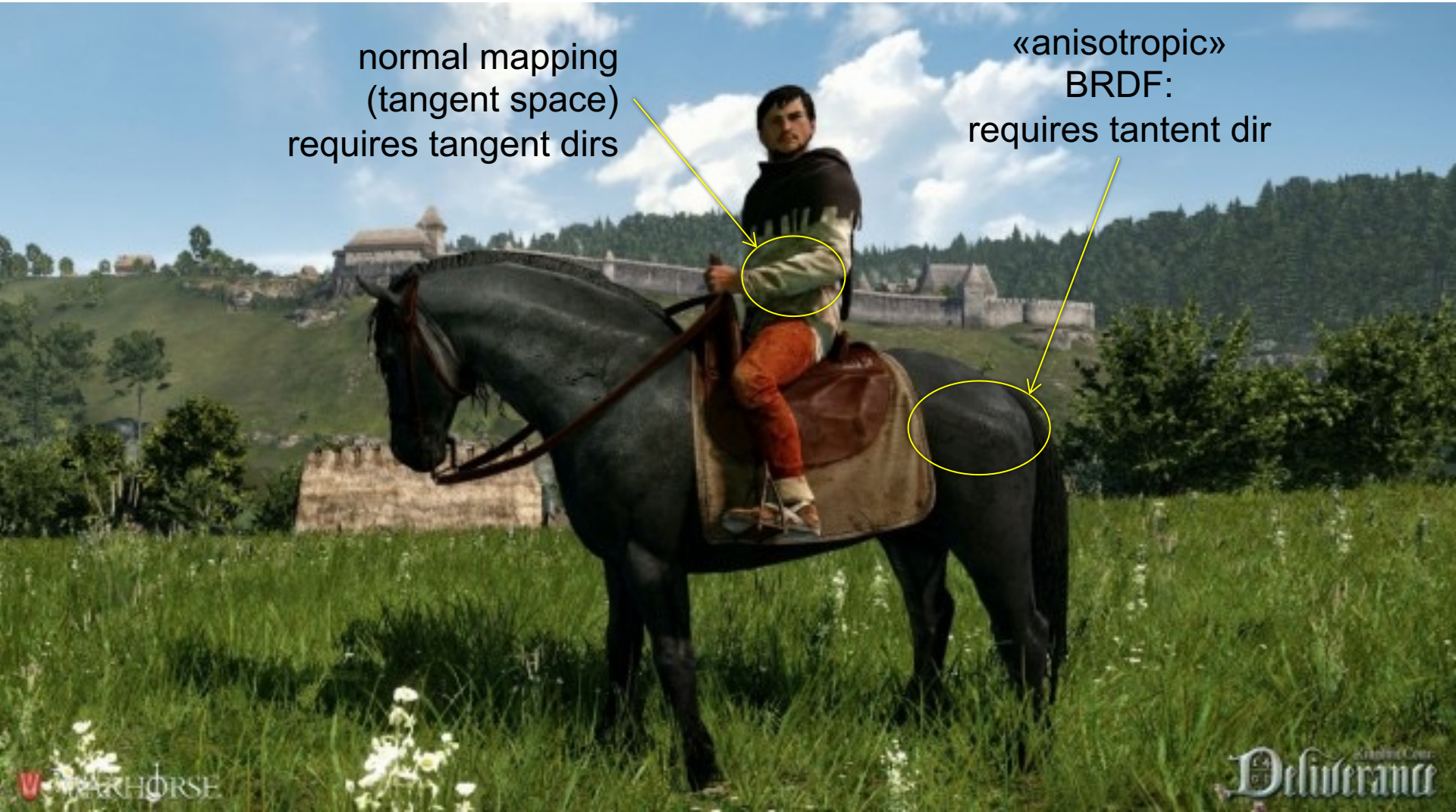
- Per vertex attribute of meshes



Reminder: Tangent dirs

normal mapping
(tangent space)
requires tangent dirs

«anisotropic»
BRDF:
requires tangent dir



Material qualities: improving



indie 2006

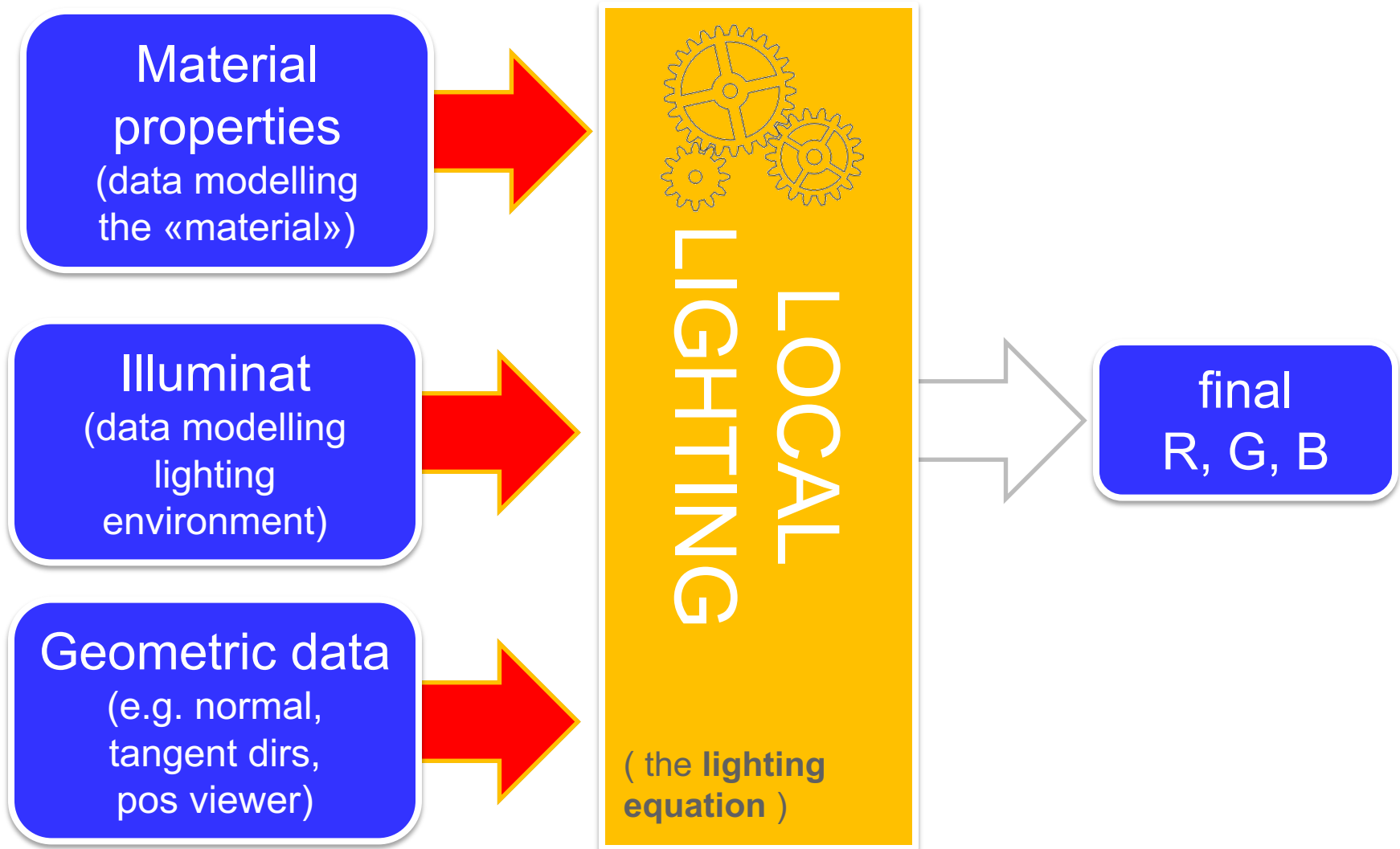


indie 2010

Material qualities: improving



Local lighting in brief



Lighting equation: how

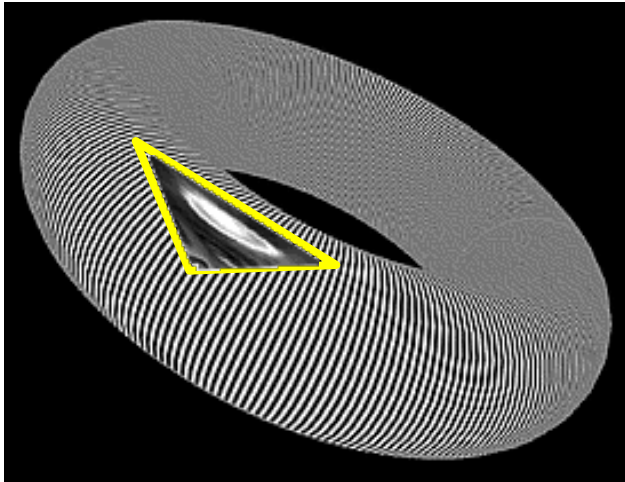
- Computed in the fragment shader
 - most game engine support a subset as default ones
 - any custom one can be programmed in shaders!
 - Material + geometry parameters stored :
 - in textures (highest freq variations)
 - in vertex attributes (smooth variations)
 - as “material assets” parameter (no variation)
 - for example, where are
 - diffuse color
 - specular color
 - normals
 - tangent dirs
- typically stored?

Environment mapping: sferico



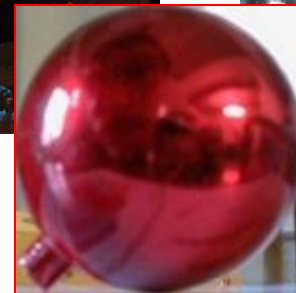
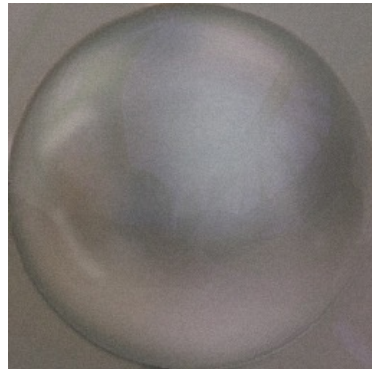
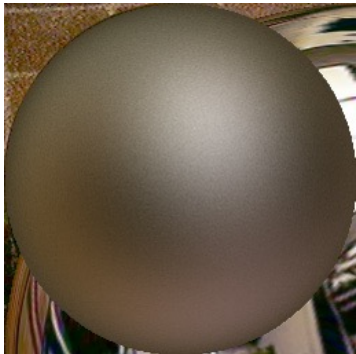
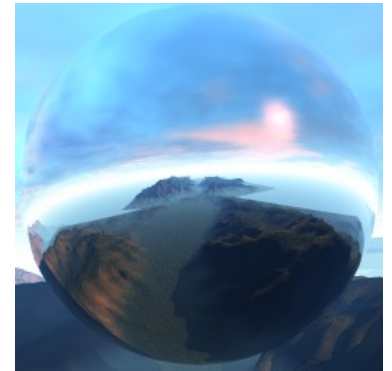
Environment map: una tessitura che memorizza il colore dell'ambiente "riflesso" da ogni normale della semisfera.

Come coordinata tessitura, basta usare la normale trasformata!



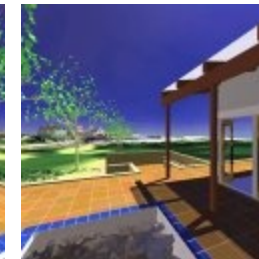
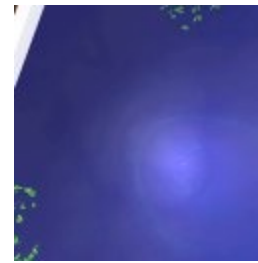
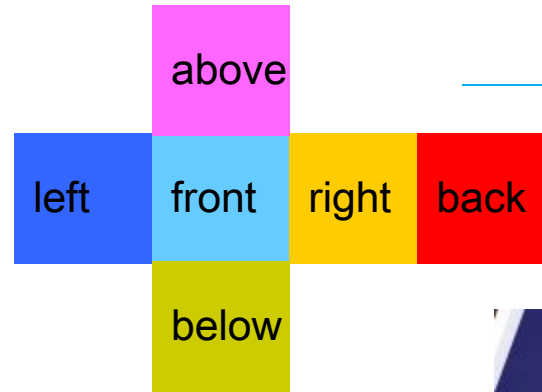
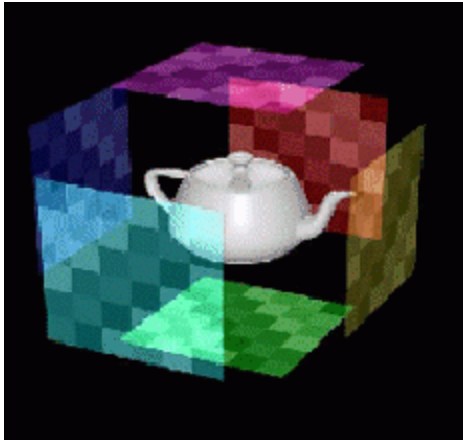
Environment mapping: sferico

simula oggetto a specchio che riflette uno sfondo lontano



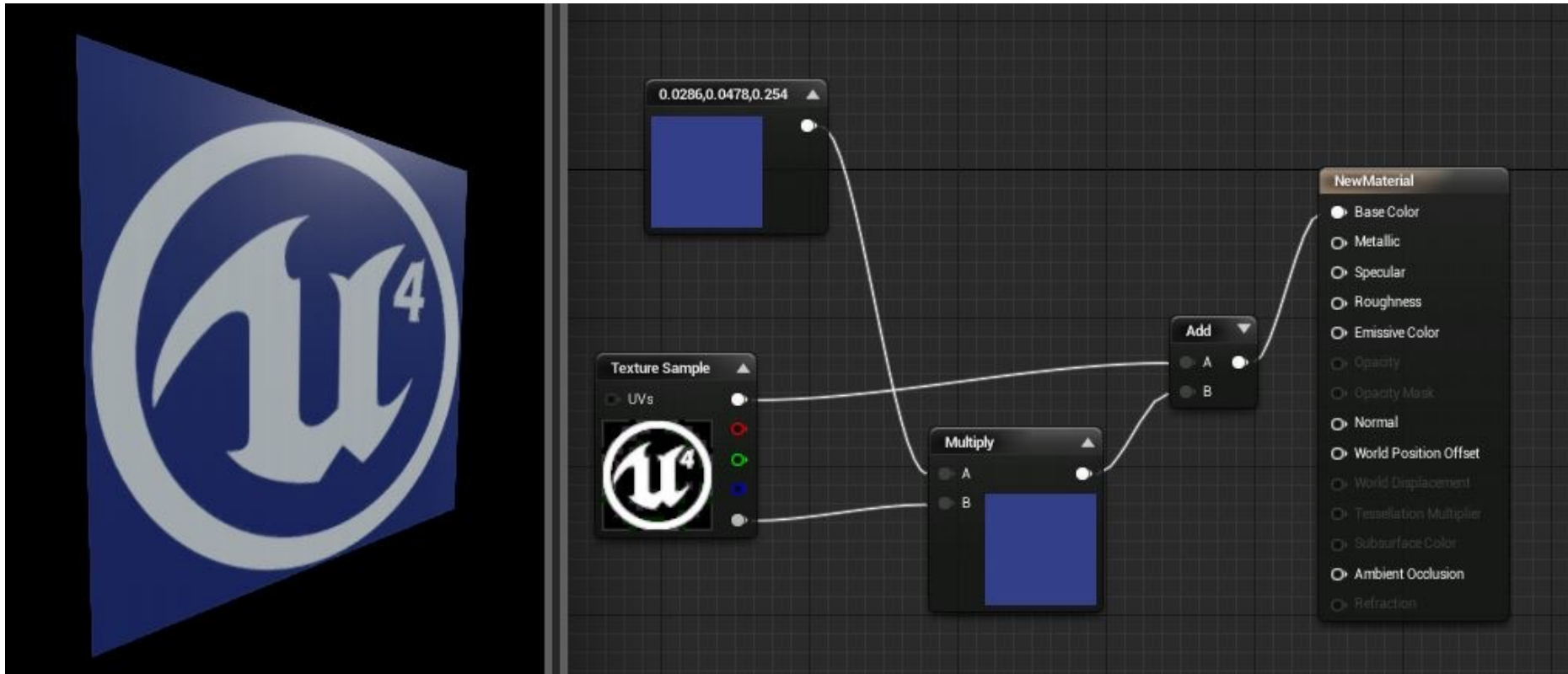
simula un materiale complesso
(condizioni di luce fisse)

Environment mapping: cubico



How to feed parameters to the lighting equation

- Hard wired choice of the game engine
- WYSIWYG game tools
 - E.g. in Unreal Engine 4



Rendering techniques popular in games

- Shadowing
 - shadow mapping ← con **PCF**
 - Screen Space Ambient Occlusion ← **SSAO**
- Camera lens effects
 - Flares
 - limited Depth Of Field ← **DoF**
- Motion blur
- High Dynamic Range ← **HDR**
- Non Photorealistic Rendering ← **NPR**
 - contours
 - toon BRDF
- Texture-for-geometry
 - Bumpmapping
 - Parallax mapping

Shadow mapping



Shadow mapping



Shadow mapping in a nutshell

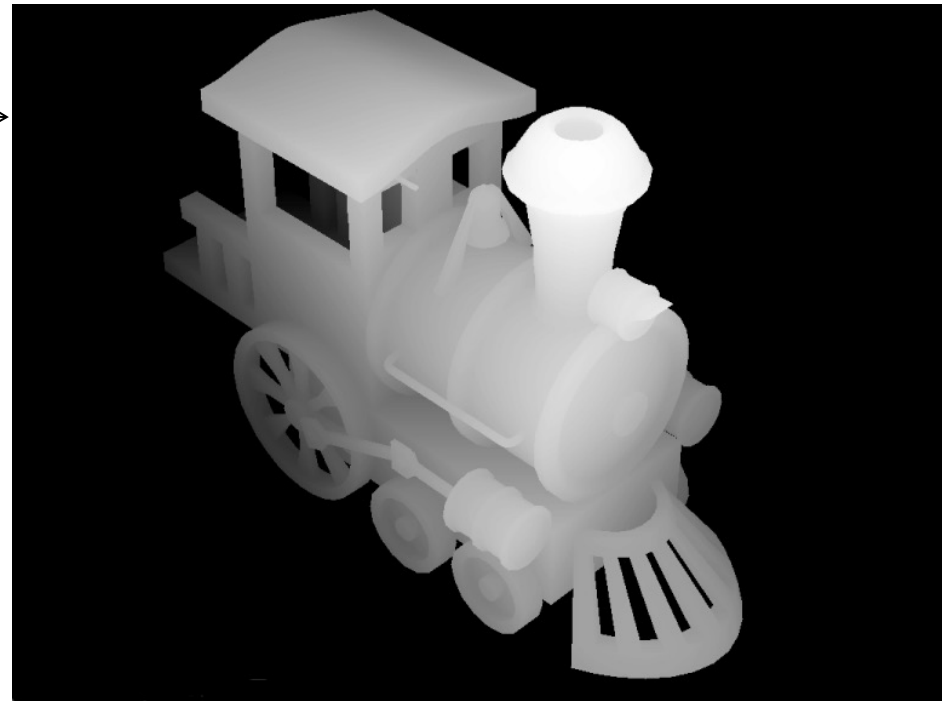
Two passes.

1st rendering: camera in light position

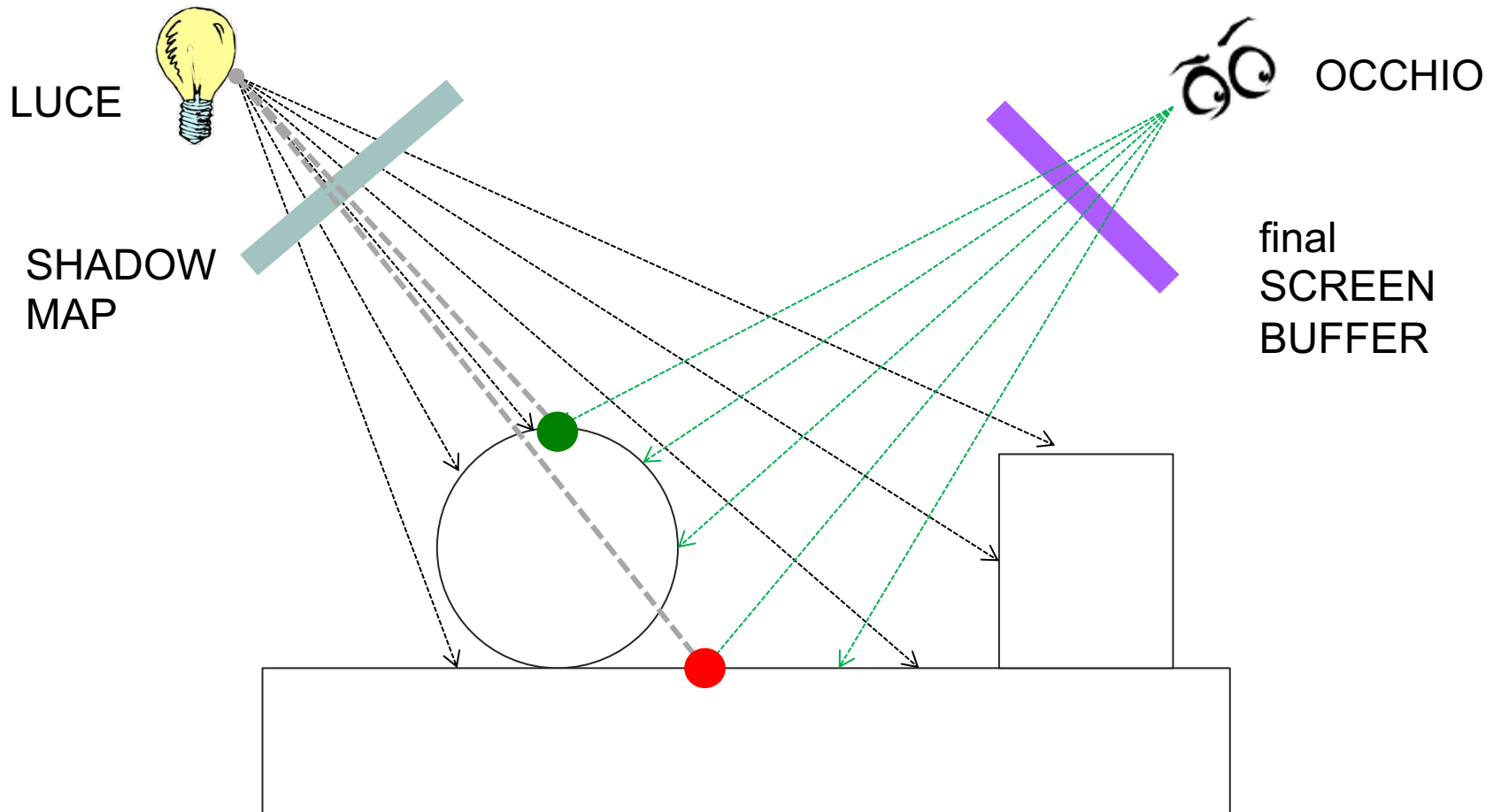
- produces: depth buffer
- called:
the shadowmap

2nd rendering:
camera in final position

- for each fragment
- access
the shadowmap once
to determine
if fragment is reached
by light or not



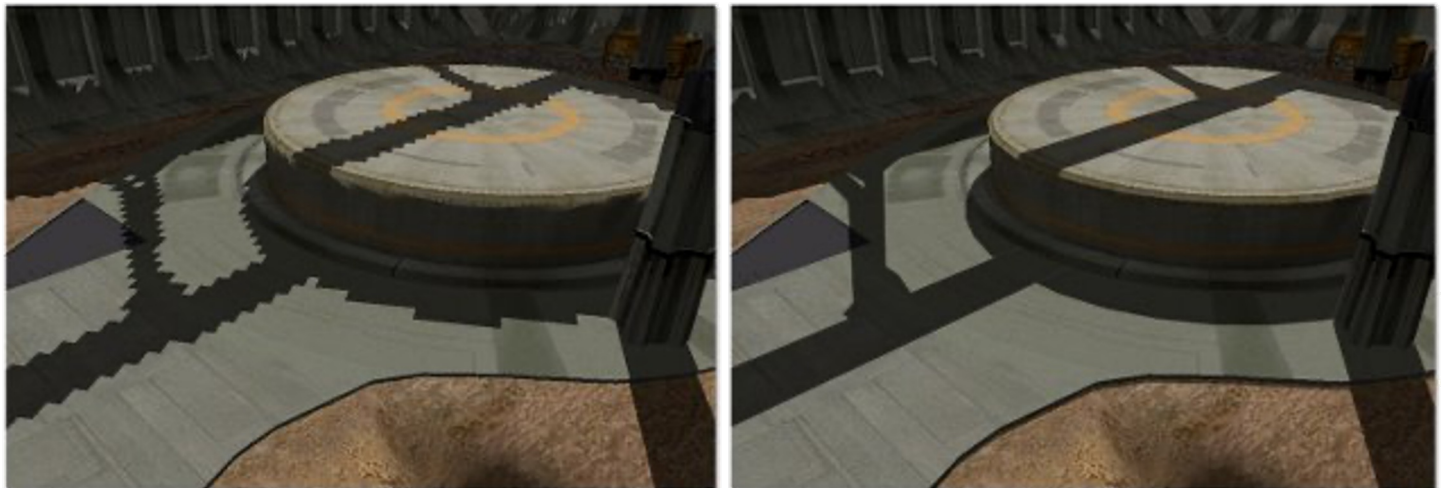
Shadow mapping in a nutshell



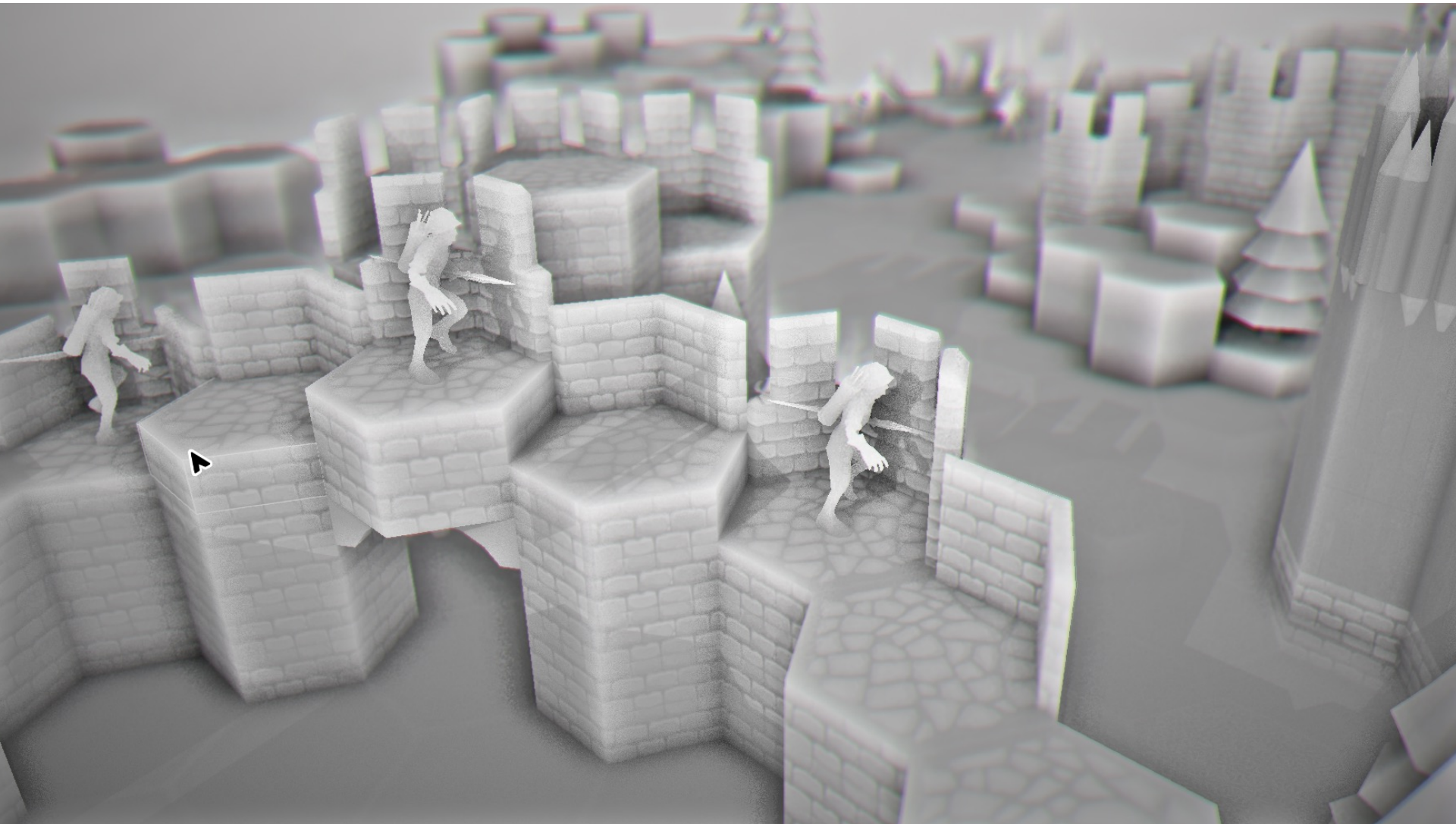
Shadow Mapping: issues

- Rendering shadowmap:
 - redo every time object move
 - can bake once and for all for static objects
 - (jet another reason to tag them)
- Shadowmap resolution:
 - it matters! aliasing effects
 - remedies: **PCF**, **multi-res shadow-map**

optional topics
(no exam)



Screen Space AO



Screen Space AO



OFF

Screen Space AO



ON

Screen Space AO in a nutshell

- First pass: standard rendering
 - produces: rgb image
 - produces: depth image
- Second pass:
screen space technique
 - for each pixel, look at depth VS its neighbors:
 - neighbors in front?
difficult to reach pixel: darken ambient
 - neighbors behind?
pixel exposed to ambient light: keep it lit

Parallax Mapping

Normal map
only



Parallax Mapping

Normal map
+ Parallax map

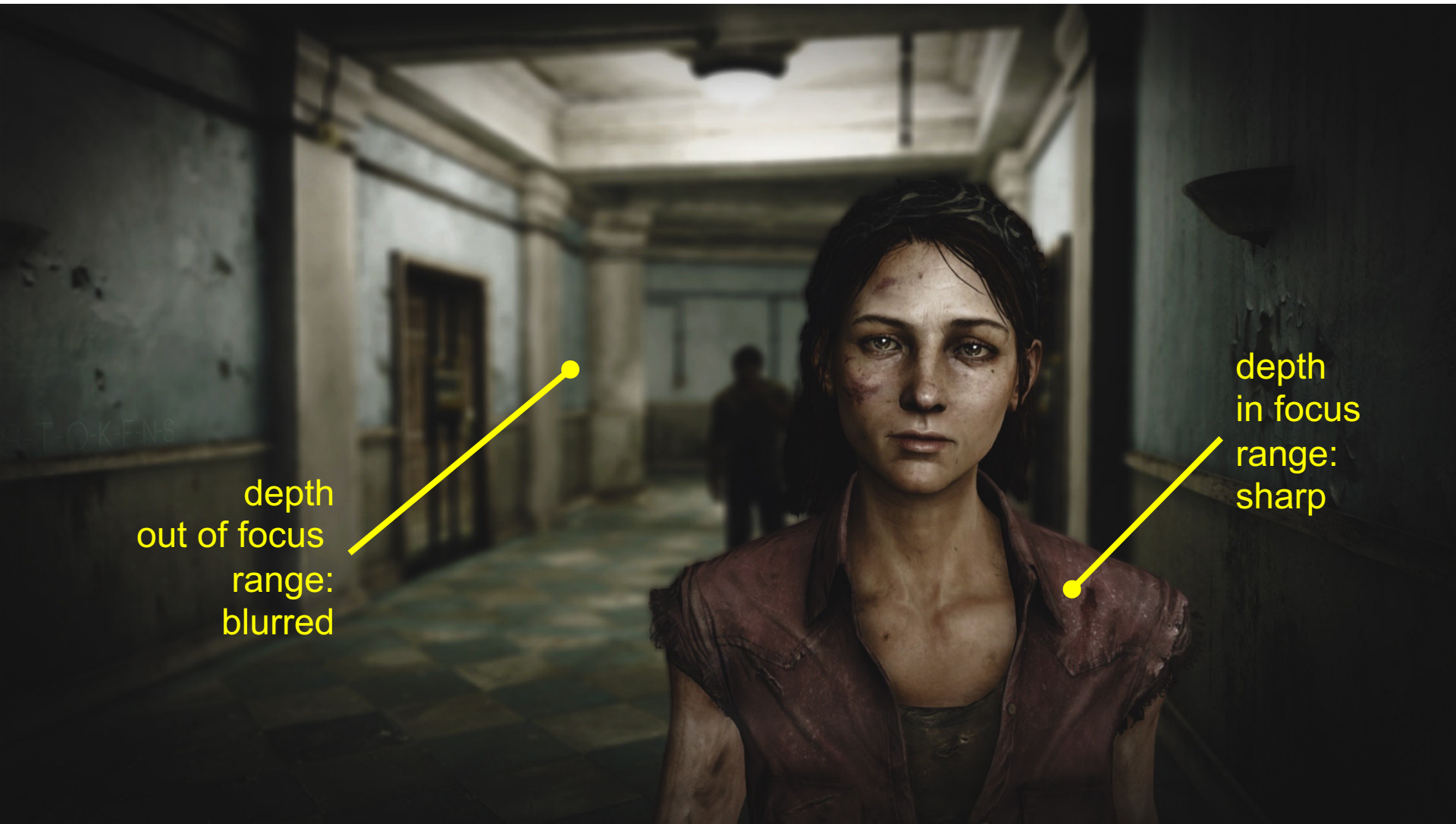


Parallax mapping: in a nutshell

- Texture-for-geometry technique
- Texture used:
 - displacement maps
 - color / rgb map



(limited) Depth of Field



depth
out of focus
range:
blurred

depth
in focus
range:
sharp

(limited) Depth of Field in a nutshell

- First pass: standard rendering
 - rgb image
 - depth image
- Second pass:
screen space technique:
 - pixel inside of focus range? keep
 - pixel outside of focus range? blur
 - (blur = average with neighbors pixels
kernel size \sim amount of blur)

HDR - High Dynamic Range (limited Dynamic Range)



HDR - High Dynamic Range in a nutshell

- First pass: normal rendering, BUT use lighting / materials with HDR
 - pixel values not in $[0..1]$
 - e.g. sun emits light with = RGB $[500,500,500]$:
 - >1 = “overexposed”! “whiter than white”
- Second pass:
screen space technique:
 - blur image, *then* clamp in $[0,1]$
 - i.e.: overexposed pixels lighten neighbors
 - i.e.: they will be max white $(1,1,1)$,
and their light bleeds into neighbors

HDR IN PHOTOGRAPHY

NO HDR



HDR



NO HDR



"HDR"



HDR IN VIDEO GAMES

Part of a general pattern

- We are used to defects in real world images so they ***must*** be emulated for more realism
 - Lens flare/bloom
 - Lens distortion
 - Color aberration
 - Film/CCD Noise
 - Motion blur
 - Atmospheric haze
 - Depth of field

You can google both:

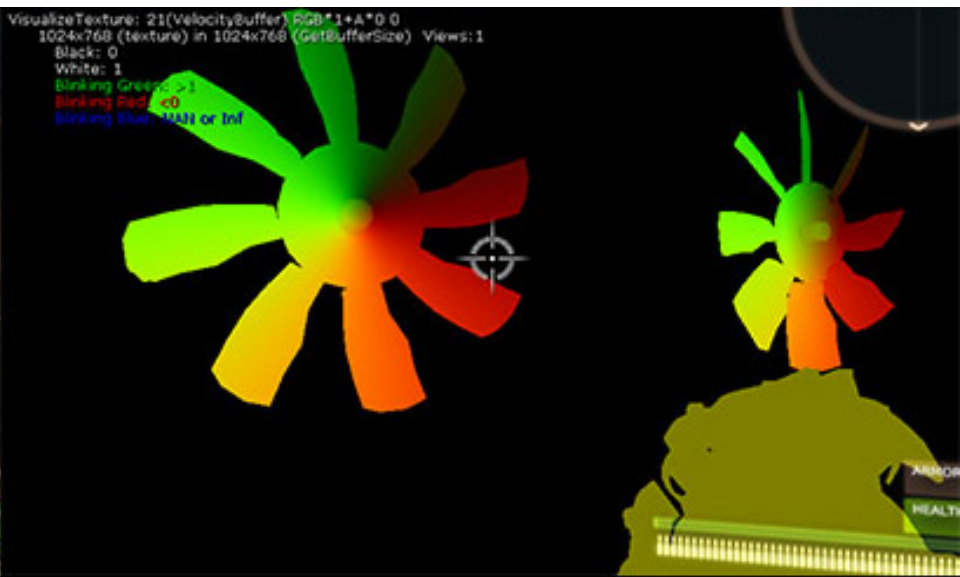
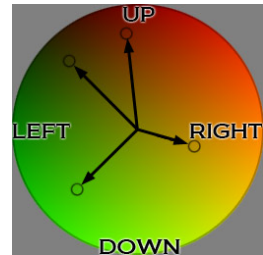
- ***Removing XXX with photoshop/GIMP***
- ***Adding XXX in Unreal/Unity***

Motion Blur



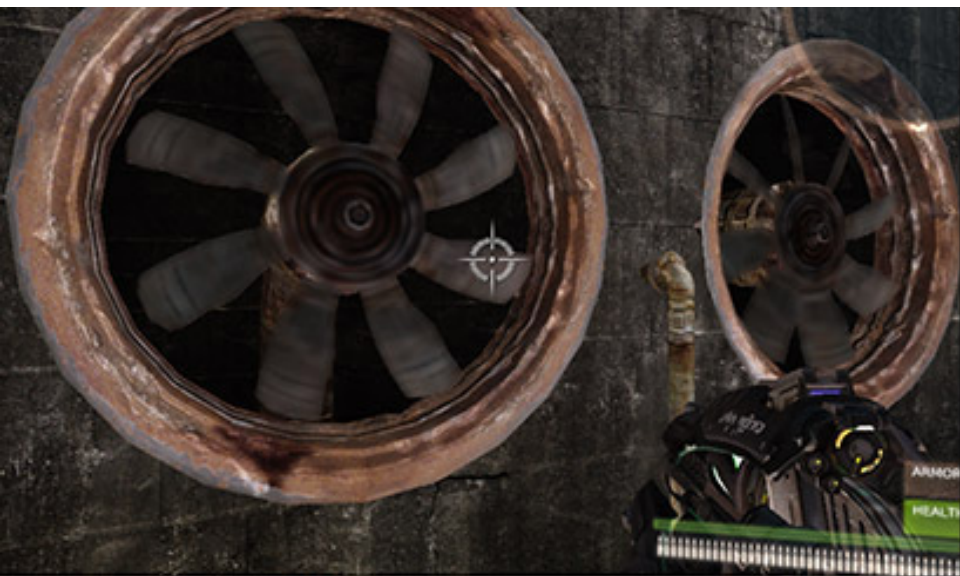
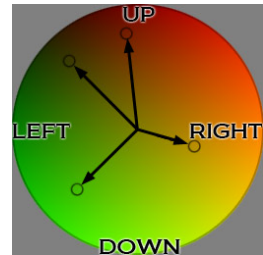
Motion blur

- Two main approach
 - Multi frame integration
 - **Velocity vector blur**



Motion blur

- Two main approach
 - Multi frame integration
 - **Velocity vector blur**



```
VisualizeTexture: 21(VelocityBuffer, RGB*1+A*0.0  
1024x768 (texture) in 1024x768 (GetBufferSize) Views: 1  
Black: 0  
White: 1  
Blinking Green: > 1  
Blinking Red: < 0  
Blinking Blue: Null or Inf
```



NPR rendering / Toon shading / Cel Shading



NPR rendering: Toon shading / Cel Shading



Toon shading / Cel Shading in a nutshell

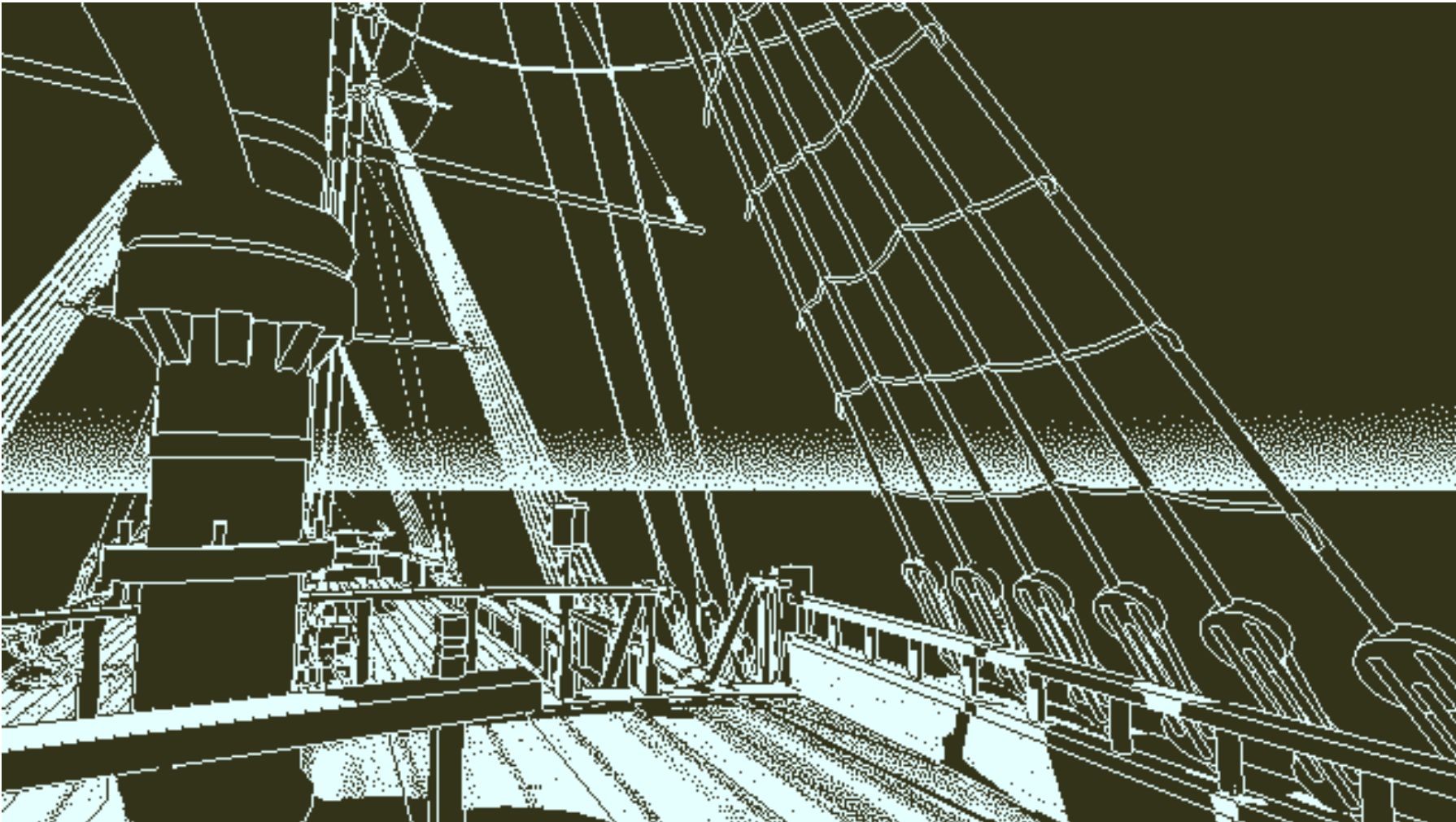
- Simulating “toons”
- Two effects:
 - add lines
 - at discontinuity lines of:
 1. depth,
 2. normals,
 3. materials
 - quantize lighting:
 - e.g. 2 or 3 tones: light, medium, dark instead of continuous
 - simple variation of lighting equation

NPR rendering: simulated pixel art



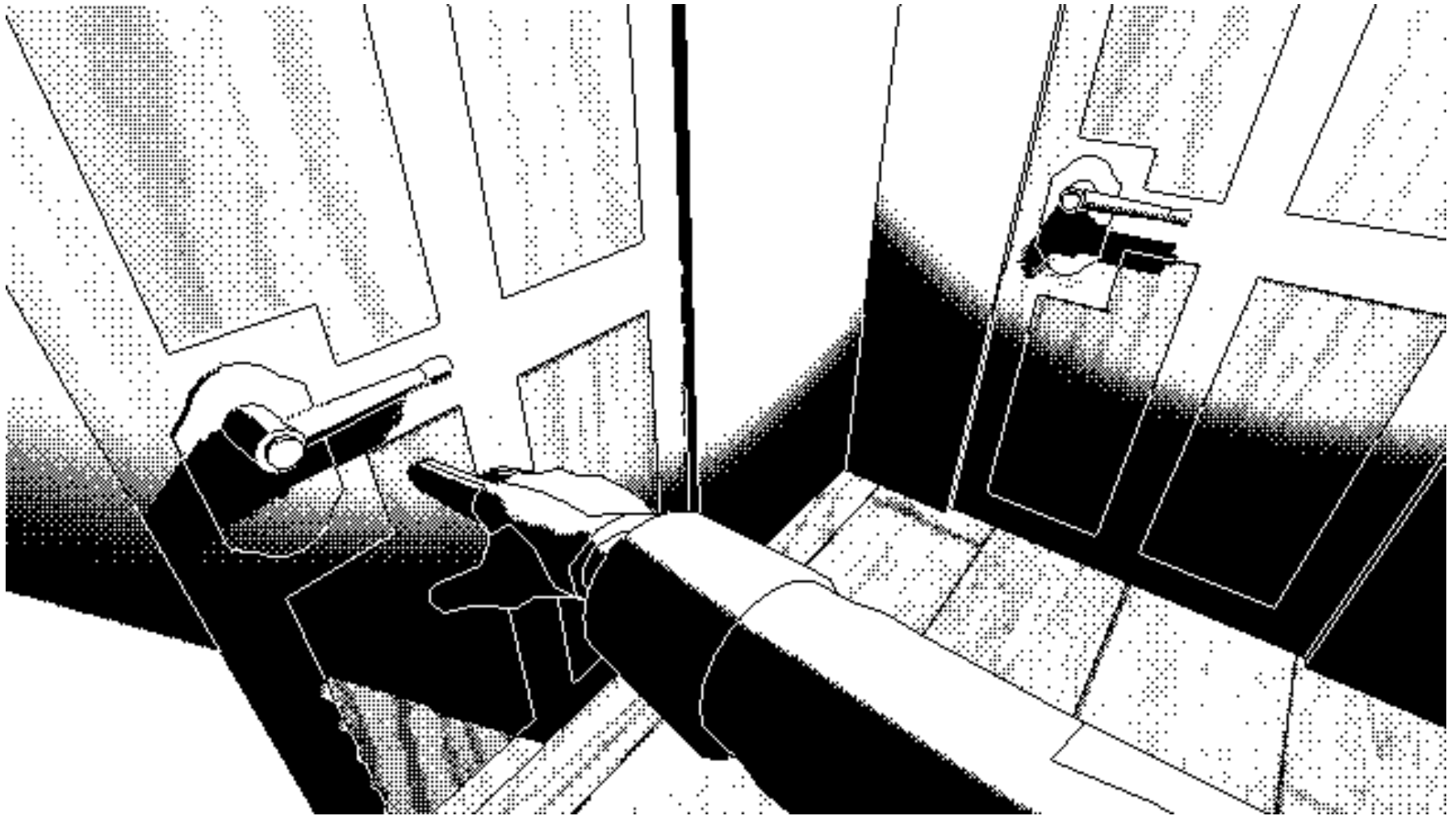
img by Howard Day (2015)

NPR rendering: simulated pixel art



img by Dukope

NPR rendering: simulated pixel art



Frame Breakdown

- Plain Unreal (2019)
<http://viclw17.github.io/2019/06/20/unreal-frame-breakdown-part-1/>
- GTA V (2015)
<https://www.adriancourreges.com/blog/2015/11/02/gta-v-graphics-study/>
- Una compilation
<https://www.adriancourreges.com/blog/2020/12/29/graphics-studies-compilation/>
- DemoScene
<https://www.iquilezles.org/prods/index.htm>
<https://www.pouet.net/>