


3D Videogames 2018/2019
Univ. degli Studi di Milano
Rendering in games
Part I: lighting & materials



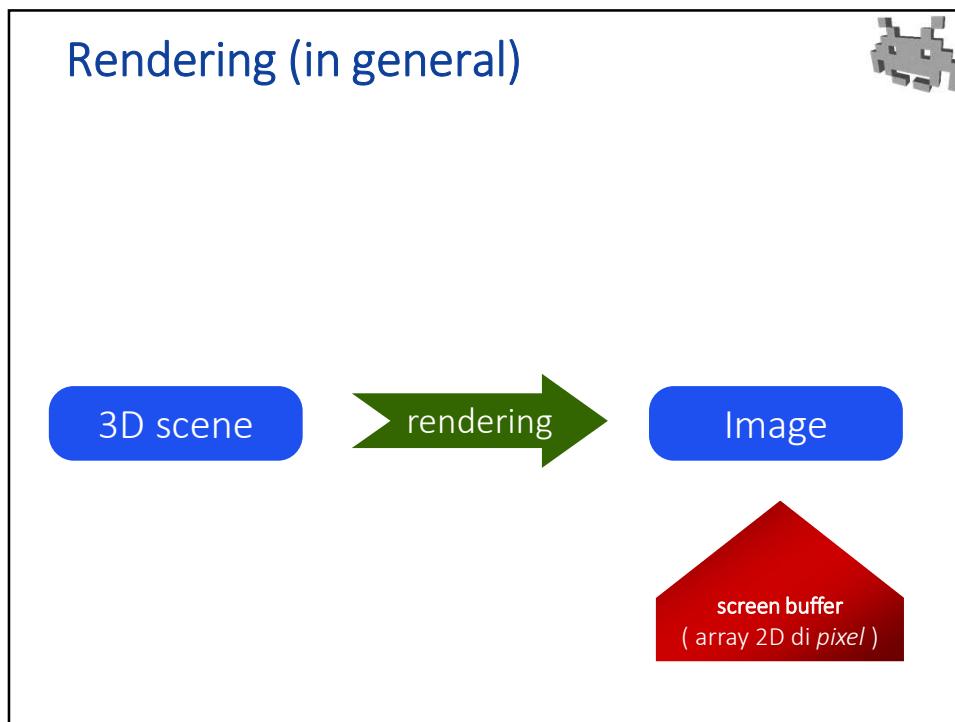
1

Course Plan




- lec. 1: **Introduction** ●
- lec. 2: **Mathematics** for 3D Games ●●●●●
- lec. 3: **Scene Graph** ●
- lec. 4: Game 3D Physics ●●● + ●●●
- lec. 5: Game Particle Systems ▶
- lec. 6: Game 3D Models ●◐
- lec. 7: Game Textures ●●
- lec. 8: Game 3D Animations ▶●●
- lec. 9: Game 3D Audio ●
- lec. 10: **Networking** for 3D Games ●
- lec. 11: **Artificial Intelligence** for 3D Games ●
- lec. 12: Game 3D Rendering Techniques ●●

2



3

- ### Rendering in 3D games
- Real time
 - (20 o) 30 o 60 FPS
 - Hardware (GPU) based
 - pipelined, stream processing
 - therefore: one class of algorithms (hardwired)
 - **rasterization** based algorithm
 - recent trend: switch to **ray-tracing** algorithms?
 - Complexity:
 - Linear with # of primitives
 - Linear with # of pixels
- 

4

This lecture and the next: a bird-eye view on...



- Graphic Hardware & HW based rendering
 - a brief summary
- Lighting
 - Local Lighting
 - Lighting equations notes
 - Lighting environments
 - Materials
 - Strategies to approximate Global Lighting
- Ad-hoc rendering techniques used in games
 - Multi-pass techniques in general
 - Screen space techniques in general
 - A summary of a few common game rendering techniques

5

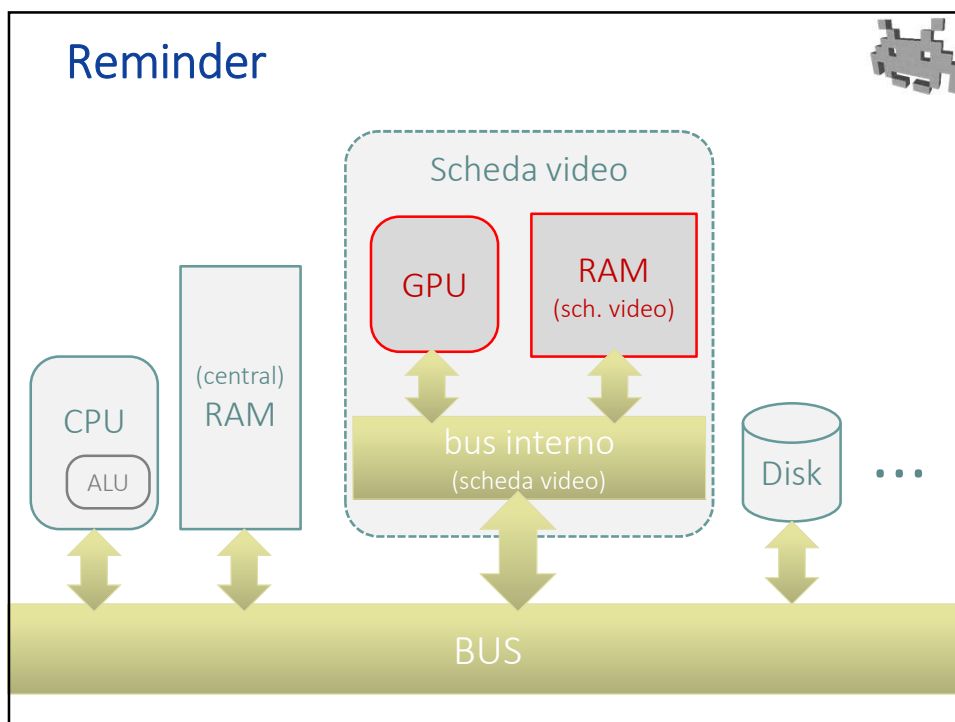
This lecture



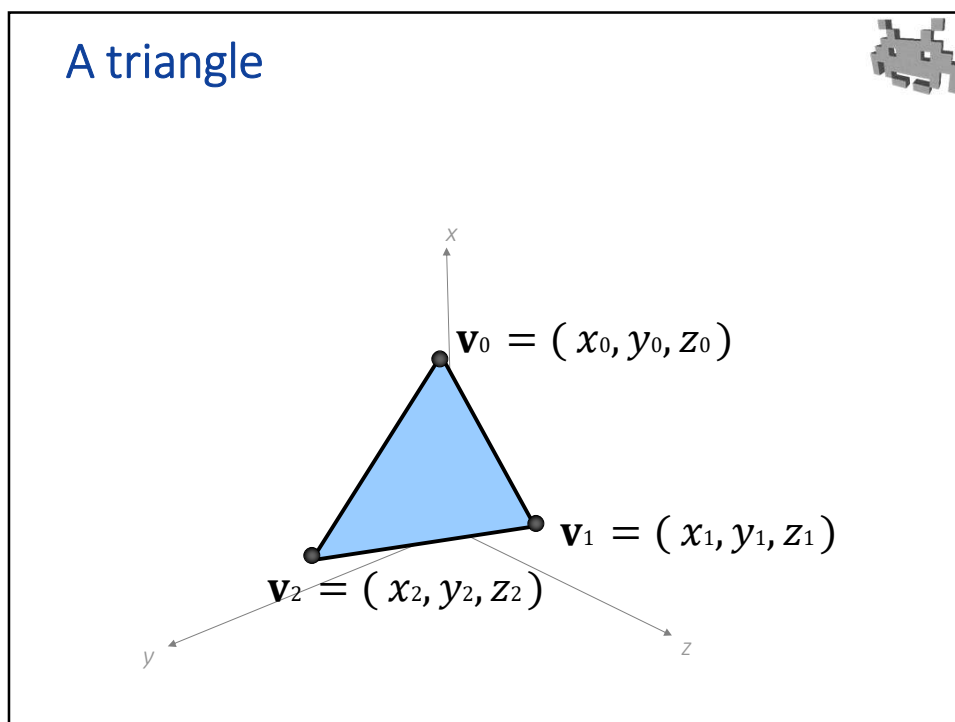
To learn more, see courses:

- **GID**
Grafica & Immagini Digitali
(the basis of 3D modelling and rendering)
- **RTGP**
Real-Time Graphics Programming
(advanced algorithms)

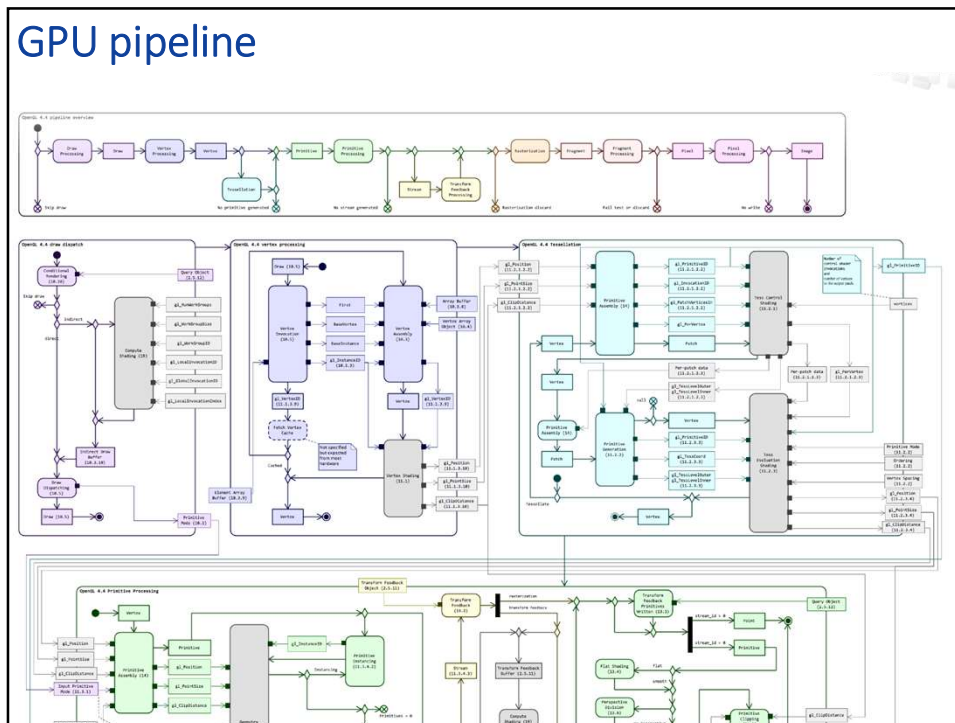
6



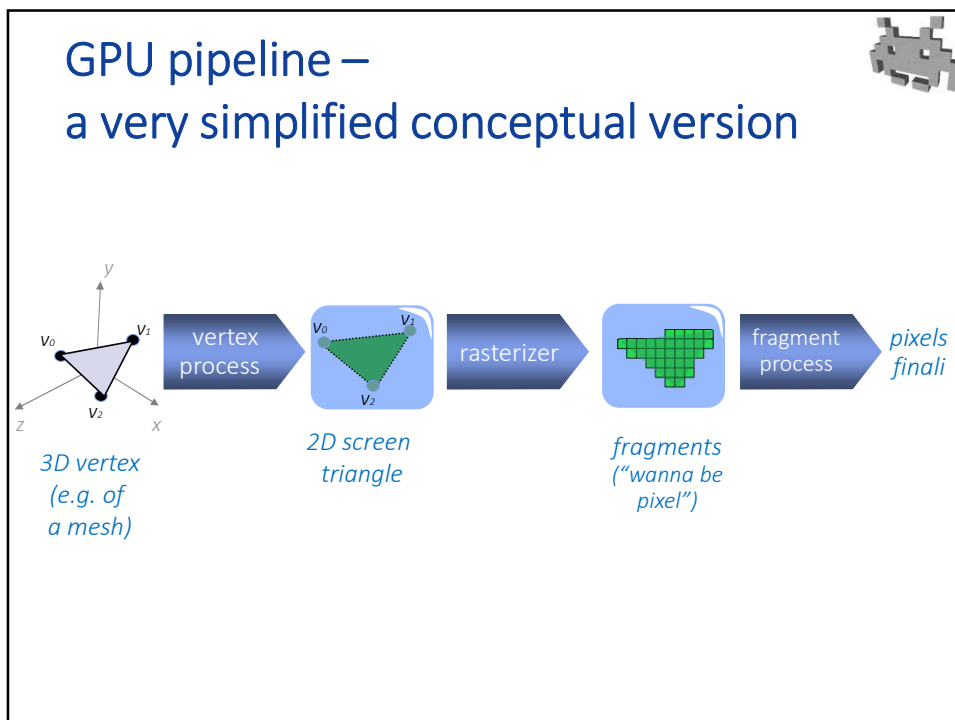
7



8



9



12

Rasterization based rendering: what is done in each step (examples)



- Per vertex: (**vertex shader**)
 - projection (transform from object space to screen space)
 - skinning (transform from rest pose to current pose)
- Per triangle: (**rasterizer**)
 - rasterization
 - interpolation of any per-vertex data
- Per fragment: (**fragment shader**)
 - lighting (from normal + lights + material to RGB)
 - texturing (textures are accessed)
 - alpha kill (fully transparent fragment – or almost – are removed)
- Per pixel: (**after the fragment shader**)
 - depth test (occluded pixels are removed)
 - alpha blend (semi-transparent fragments are mixed with background)

13

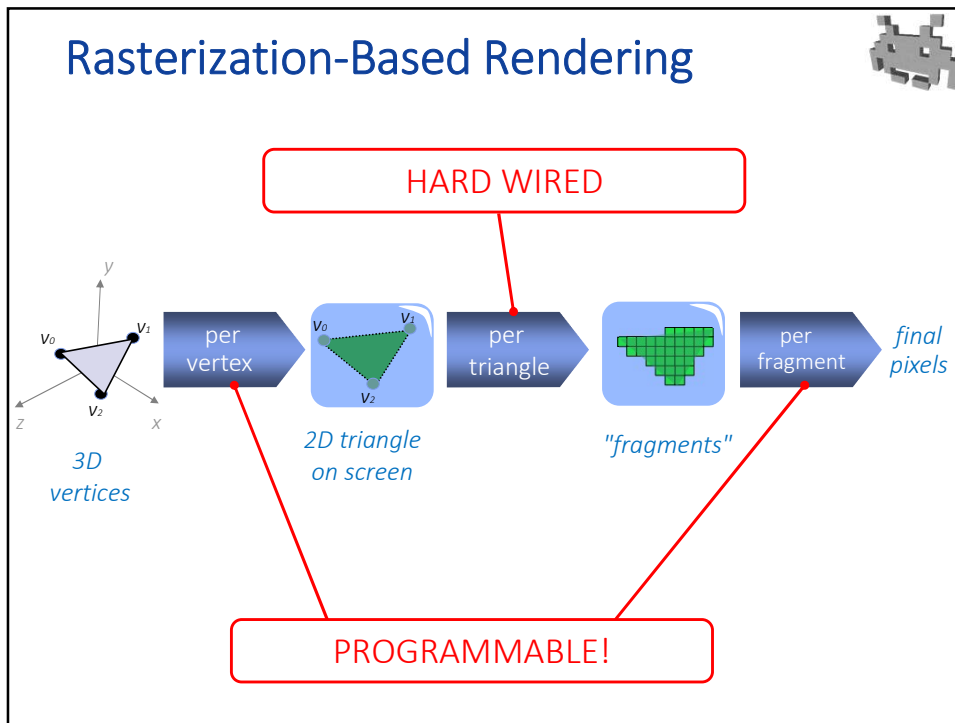
GPU pipeline – bottlenecks (notes and terminology)



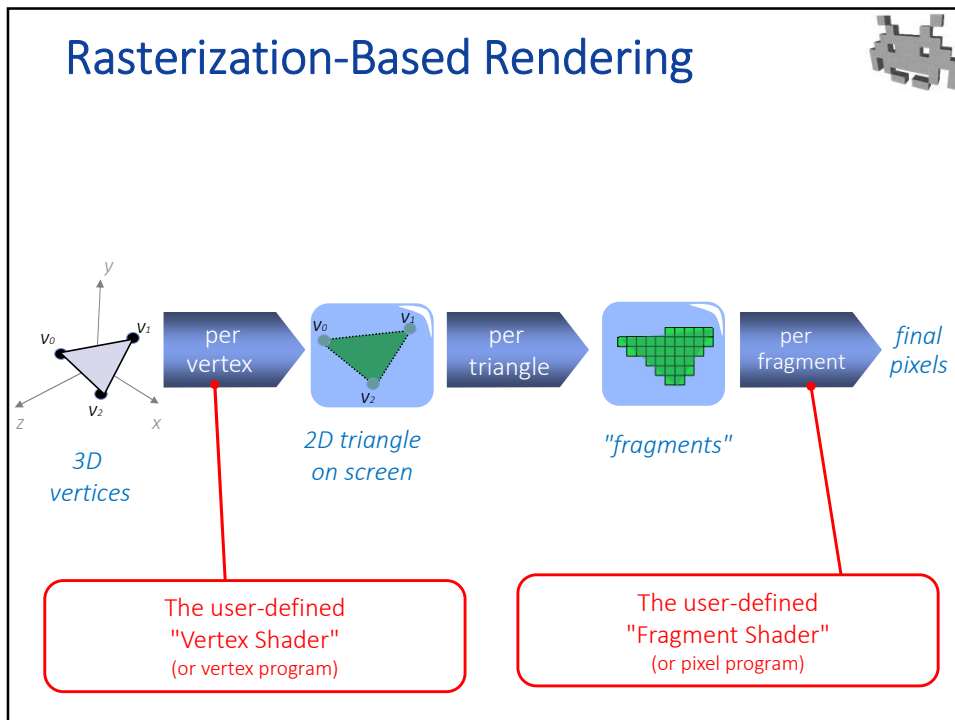
- Like in any pipeline, the process goes *as slow as its slower stage*
 - The «bottleneck» of the pipeline determines the total speed
 - Any other stage is idle for for part of the time (idle is always a waste)
 - stages before the bottleneck are «choked»
(cannot produce output because next stage is not ready)
 - stages after it are «starved» (they don't receive input from prev stage)
- Bottleneck terminology: (in CG)
 - If the bottleneck is per vertex, the app is **geometry-limited**
(it cannot process «geometry» fast enough)
 - If the bottleneck is per fragment, the app is **fill-limited**
(it cannot fill the buffer with pixel fast enough)
- Performances (rendering FPS) of a game only improves if computational load is removed from the bottleneck phase
 - Example:
using all meshes at LOD 1 instead of one does not help a fill limited app
 - Example:
reducing the resolution of the screen does not help a geometry-limited app
 - Using a simpler lighting model does not help a geometry limited app

← MORE COMMON
CASE, FOR GAMES

14



15



16

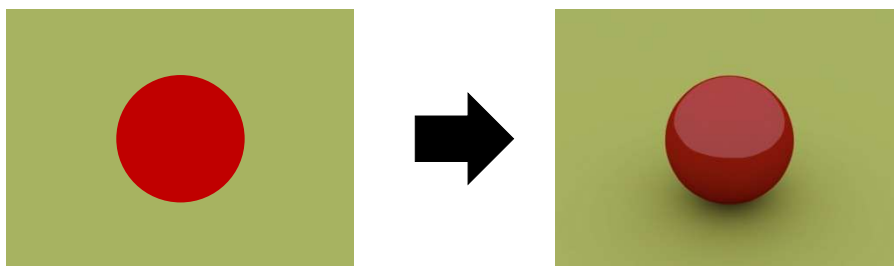
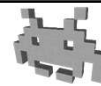
Shading languages examples



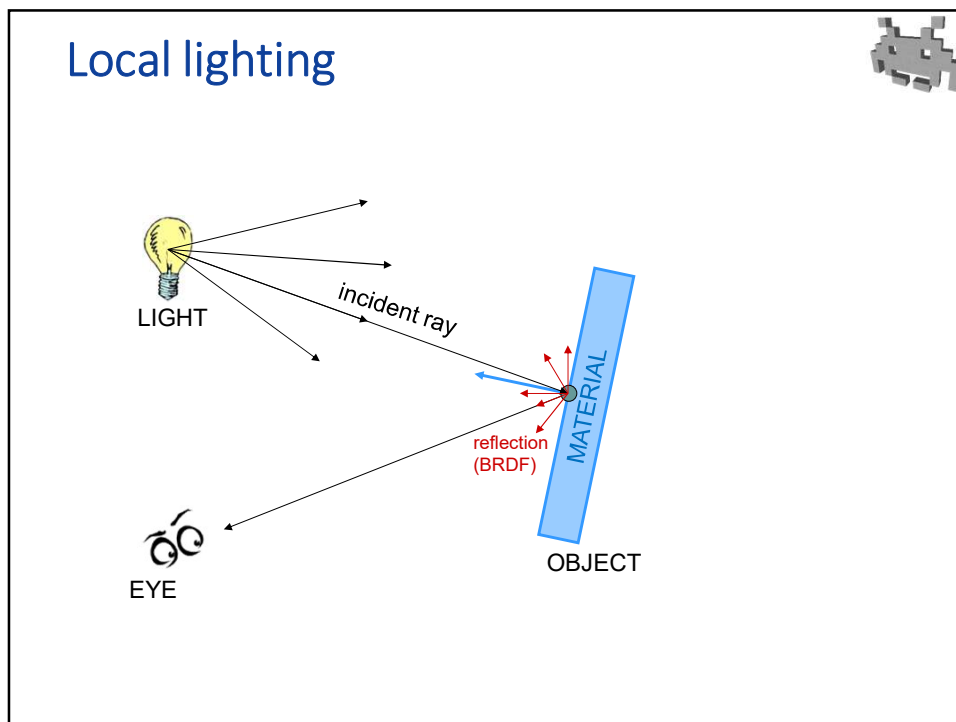
- High level:
 - **HLSL** (High Level Shader Language, Direct3D, Microsoft)
 - **GLSL** (OpenGL Shading Language)
 - **CG** (C for Graphics, Nvidia)
- Low lever:
 - **ARB** Shader Program
(the “assembler” of GPU – now deprecated)

17

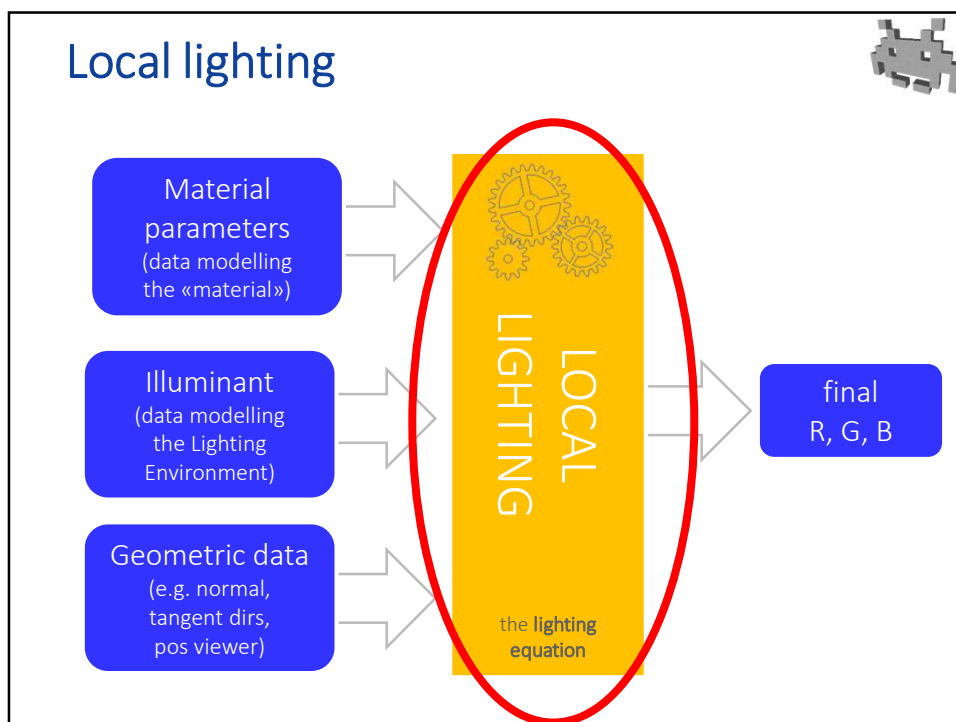
Lighting



18

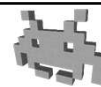


19



20

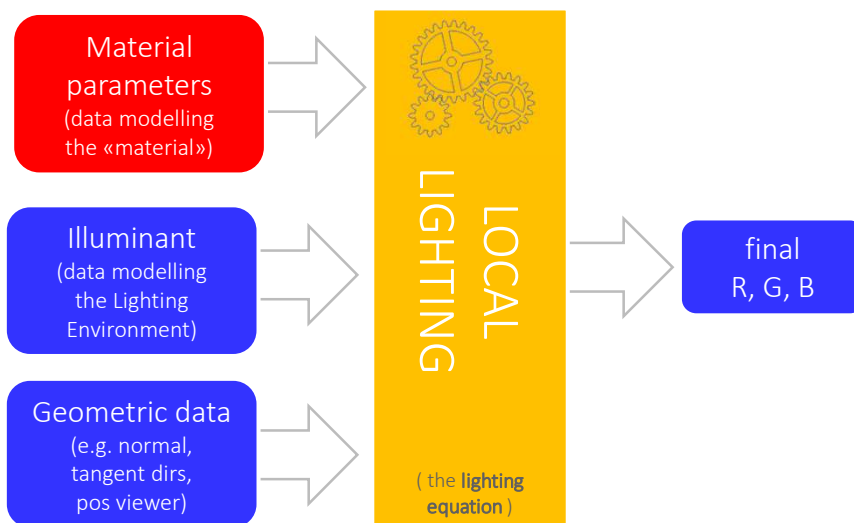
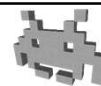
Lighting equations



- Different equations can be employed...
 - Lambertian
 - Blinn-Phong } the basic model, historically used in games for decades
 - Beckmann
 - Heidrich–Seidel
 - Cook–Torrance
 - Ward (anisotropic)
 - ...
 - + additional Fresnel effect
- Varying levels of
 - computational complexity
 - realism
 - some are *physically based*, some are... just tricks
 - material parameters required
 - richness / variety of effects

21

Local lighting



22

Materials



23



Terminology:

«material» can mean 2 different things



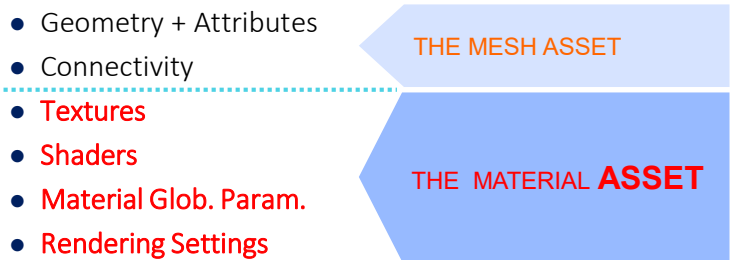
- The material **model**, i.e. the material **parameters**
 - a part of the input of the (local) lighting equation
 - the part that models the (local) optical behavior of a physical substance (plastic, wood)
- The material **asset**
 - a common abstraction used by game engines / dataset
 - an asset which combines:
 - a set of textures (e.g. diffuse + specular + normal map)
 - a set of shaders (e.g. vertex + fragment)
 - a set of global parameters (e.g. glossiness, ambient factors)
 - a set of rendering settings (e.g. back-face culling ON/OFF, or flags based on rendering order)
 - Basically, it corresponds to the **status** of the rendering engine when a mesh is drawn

24

Material Asset = status of renderer

To render a mesh: (reminder)

- Load...
 - make sure all data is stored in **GPU RAM**
 - Geometry + Attributes
 - Connectivity
 - **Textures**
 - **Shaders**
 - **Material Glob. Param.**
 - **Rendering Settings**
- ...and Fire!
 - issue the draw call (the command: "do it!")



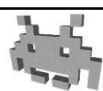
25

Material models: which parameters?

- Q: which set of parameters is a «material»?
- A: it is determined by the chosen lighting equation

material = the arguments of the lighting equation accounting for the physical substance that the surface is made of

- regardless of the answer, each parameter in the set can be stored:
 - per **Material Assets**, as a **global parameter**, or
 - per **Vertex** of a Mesh, as **attributes**, or
 - per **Texel** of a texture sheet (maximal freedom)



26

The most simple choice for Material-model & lighting-equation:



- see: “OpenGL material”, or OBJ material
- This **Lighting Equation** is the sum of 4 simple terms:
 - Ambient + Diffuse + Specular (+ Emission)
- The material is... a color multiplier for each term, therefore:
 - “Ambient” color (RGB)
 - “Diffuse” color (aka “Base Color”, aka “Albedo”)
 - “Specular” color (RGB)
 - plus one “Specular Exponent”, aka “glossiness” or “shininess” (an in > 1, often, max = 127)
 - “Emission” factor (RGB) (only for stuff *emitting* light – otherwise 0,0,0)

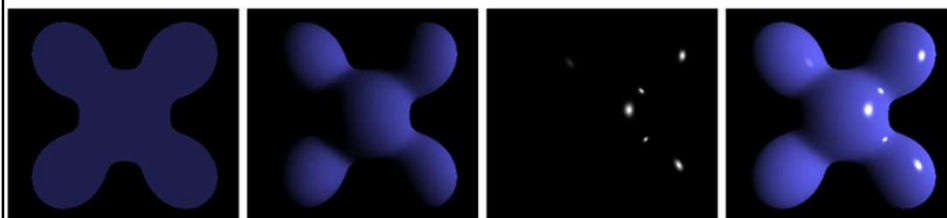
separate multiplier for R, G and B

27

A very basic lighting equation (basic or «Phong» lighting equation)



- Sum of 3 terms:



ambient + diffuse + specular = final
(or “Lambertian”) (or “Phong”)

(a constant additional term (“emission”) is added only for objects emitting light)

28

A very basic lighting equation: diffuse term (aka «Lambertian»)

- In formulas:

dot product
but zero if negative!

 $\hat{n} \cdot \hat{L}$

surface normal

light direction

$\begin{pmatrix} d_R \\ d_G \\ d_B \end{pmatrix}$

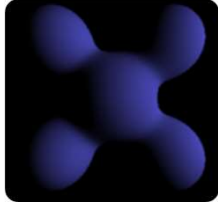
diffuse-color

\otimes

$\begin{pmatrix} L_R \\ L_G \\ L_B \end{pmatrix}$

light-color

component-wise
product



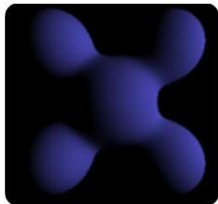
- material parameter
- light parameter
- geometry

See CG course for an explanation!

29

A very basic lighting equation: diffuse term (aka «Lambertian»)


- info:
 - it's physically based
 - exhibited by dull materials (e.g. plasters, untreated wood)
 - still used in any lighting equation
- material parameter used:
 - **base color**, aka **albedo** (when grayscale), aka **diffuse** color, aka **Lambertian** color, sometimes just **color**
 - the texture to specify it is called **diffuse-map** aka **color map** or just **RGB map**



implementation note: (applies to all formulas)
the versors in the dot-product must be in the same space! -- e.g. object space or world space

30

A very basic lighting equation: specular term (aka «Blinn-Phong»)



- In formulas:

specular exponent

dot product
but zero if negative!

surface normal

“half-way” versor:

$$\hat{H} = \text{mix}(\hat{V}, \hat{L}, 0.5)$$

view direction
(toward the light)

$$(\hat{n} \cdot \hat{H})^E$$

material parameter

$$\begin{pmatrix} S_R \\ S_G \\ S_B \end{pmatrix}$$

specular-color

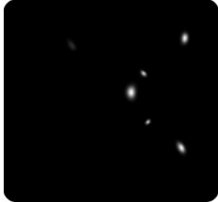
$$\otimes$$

light parameter

$$\begin{pmatrix} L_R \\ L_G \\ L_B \end{pmatrix}$$

light-color


component-wise product



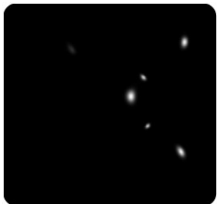
- material parameter
- light parameter
- geometry

31

A very basic lighting equation: specular term (aka «Blinn-Phong»)



- info:
 - it's just a trick
 - not physically based (not even energy conserving)
 - add simulated reflections (highlights)
- additional material parameters:
 - **specular color**
determines the intensity and color of the highlight
sometimes: diffuse color x a constant
often > 1 – oversaturated highlight
 - **specular exponent** (aka glossiness)
determines the SIZE of the highlight
larger numbers → smaller highlight
- textures:
 - **specular map** and **glossiness map**
 - e.g. in a 4 channel texture: RGB + glossiness



32

A very basic lighting equation: ambient term

- In formulas:

component-wise product

ambient-color

light-color

material parameter


light parameter

geometry

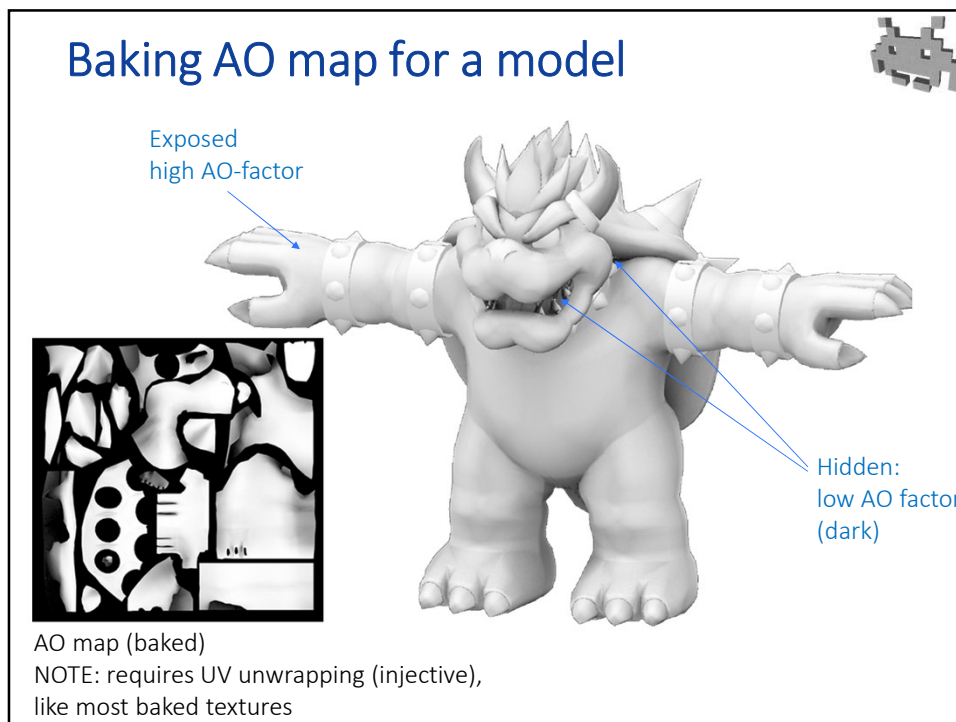
33

A very basic lighting equation: ambient term

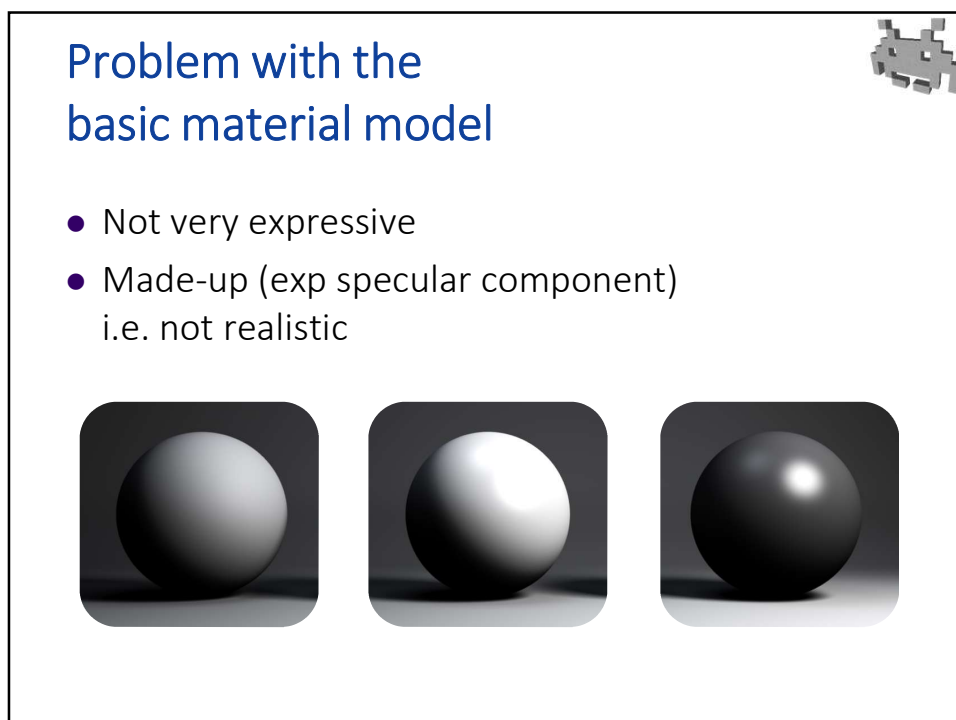
- info:
 - based on the assumption: a bit of light reaches the object from every direction (e.g. from light bounces, or unmodelled light sources)
 - important: without it, things not directly by lights are black! (e.g. negative cross verify)
 - Very simple to compute, so why not
- additional material parameters:
 - ambient color** – usually very small
 - usually, it's just the diffuse color times a constant (<1)
 - the constant occludes (negates the ambient) and is called **Ambient Occlusion**
- textures:
 - AO map** (stores AO terms)
 - other common solutions: store AO in vertices, and SSAO (see later)



34



35



36

Choice of material models: Chapter 1 ('90s)



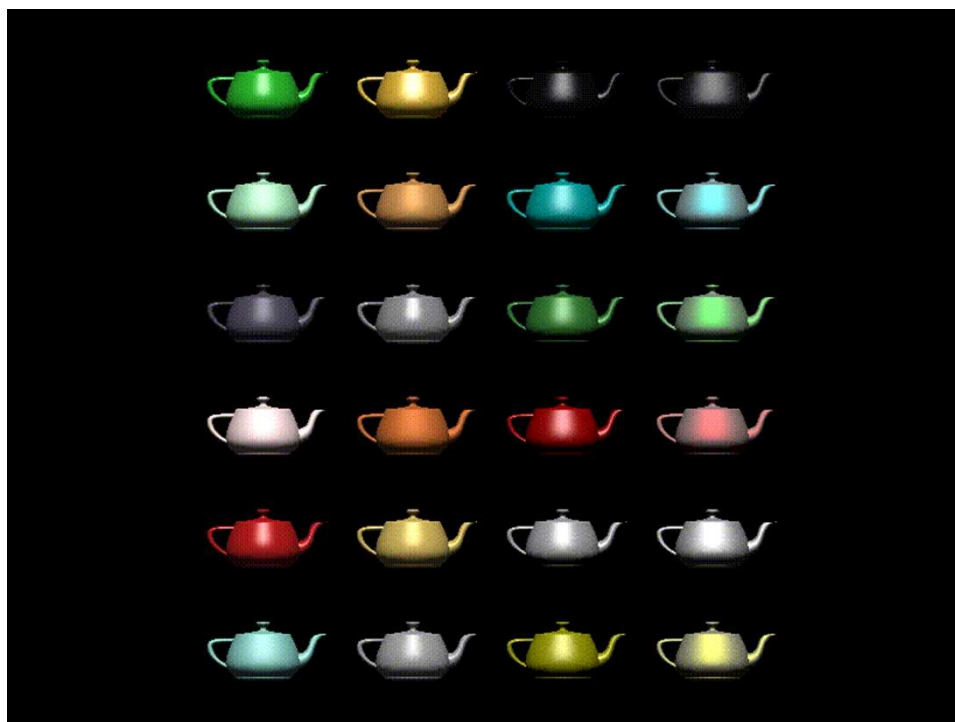
Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR	GL_SHININESS
Silver	0.19225	0.50754	0.508273	51.2
	0.19225	0.50754	0.508273	
	0.19225	0.50754	0.508273	
	1.0	1.0	1.0	
Polished Silver	0.23125	0.2775	0.773911	89.6
	0.23125	0.2775	0.773911	
	0.23125	0.2775	0.773911	
	1.0	1.0	1.0	
Emerald	0.0215	0.07568	0.633	76.8
	0.1745	0.61424	0.727811	
	0.0215	0.07568	0.633	
	0.55	0.55	0.55	
Jade	0.135	0.54	0.316228	12.8
	0.2225	0.89	0.316228	
	0.1575	0.63	0.316228	
	0.95	0.95	0.95	
Obsidian	0.05375	0.18275	0.332741	38.4
	0.05	0.17	0.328634	
	0.06525	0.22525	0.345435	
	0.82	0.82	0.82	
Pearl	0.25	1.0	0.296548	11.264
	0.20725	0.829	0.296548	
	0.20725	0.829	0.296548	
	0.922	0.922	0.922	
Ruby	0.1745	0.61424	0.727811	76.8
	0.01175	0.04136	0.626959	
	0.01175	0.04136	0.626959	
	0.55	0.55	0.55	
Turquoise	0.1	0.396	0.297254	12.8
	0.18725	0.74151	0.30829	
	0.1745	0.69102	0.306678	
	0.8	0.8	0.8	
Black Plastic	0.0	0.01	0.50	32
	0.0	0.01	0.50	
	0.0	0.01	0.50	
	1.0	1.0	1.0	
Black Rubber	0.02	0.01	0.4	10
	0.02	0.01	0.4	
	0.02	0.01	0.4	
	1.0	1.0	1.0	



not too expressive ☹️

Still used (sometimes).
MTL files (OBJ file format) is basically this.

37



38

Choice of material models: Chapter 2 ('00s)



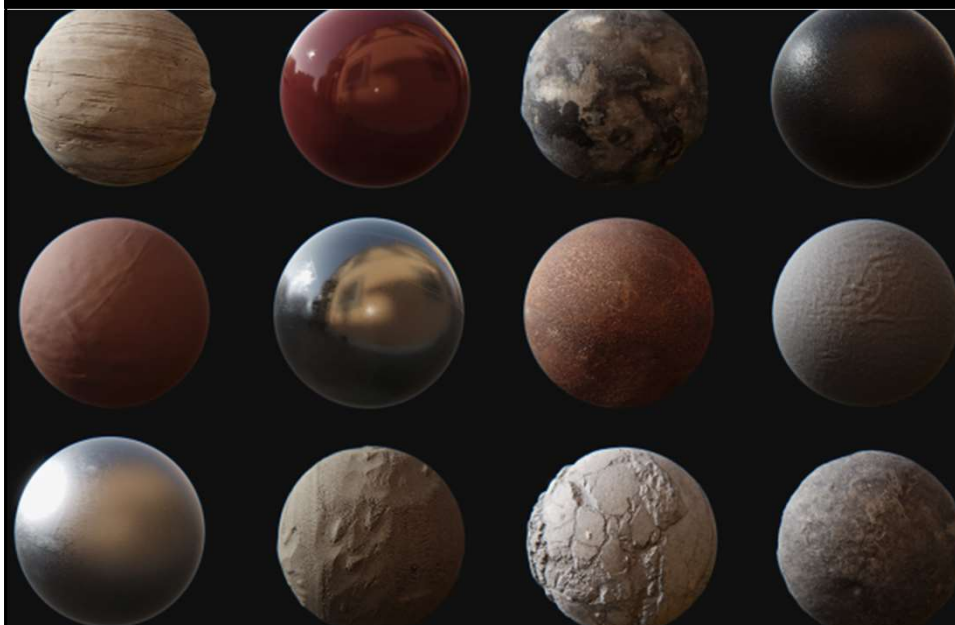
- The **Lighting Equation** becomes more complex
 - more terms are added
- It feeds more material **parameters**...
 - Such as: Fresnel effect, Anisotropic effect, Reflectivity – with environment maps, ...
- Authoring materials becomes an increasingly complex, and *ad-hoc* task
 - Difficult to port one material ...
 - ...from one engine to another, ...from one game to another, ...from one asset to another
 - Difficult to guess the right parameters for a given object
 - especially if it has to look good under widely different lighting conditions



the task of the
“material artist”

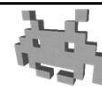
39

Much wider expressiveness



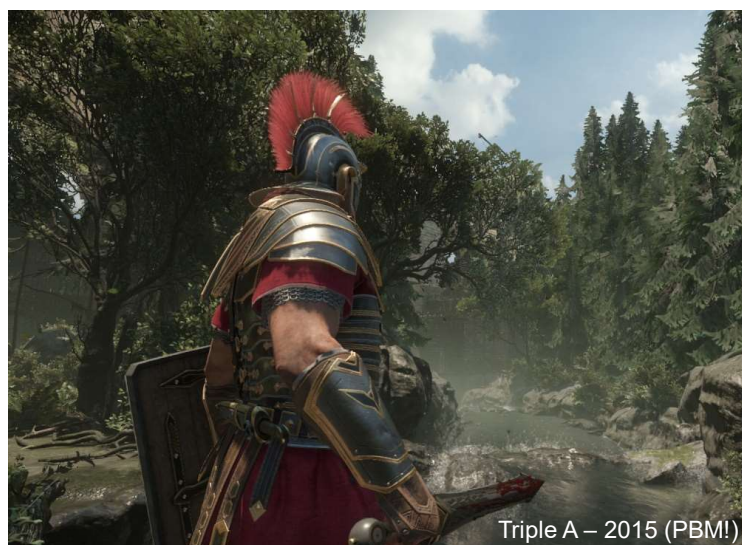
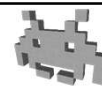
40

Material models improving



41

Material models are improving



42

Choice of material models: Chapter 3 ('10s)



- **Physically Based Materials (PBM)**
 - an ongoing trend!
- General characteristics and objectives:
 - increased intuitiveness:
 - provide Material Artist with a higher-level material description
 - eases the Material Authoring task
 - increased standardization:
 - makes materials more cross-engine / portable (almost)
 - increased generality:
 - accommodates for most established, modern lighting eq. terms, and lighting environment description (e.g. env maps)
 - increased realism / quality:
 - more faithful, physically justified model of real-world materials
 - result: material can even be captured from real-world samples
 - result: good-looking results under widely different lighting env

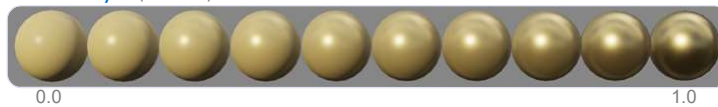
43

«Physically Based Materials» (PBM)

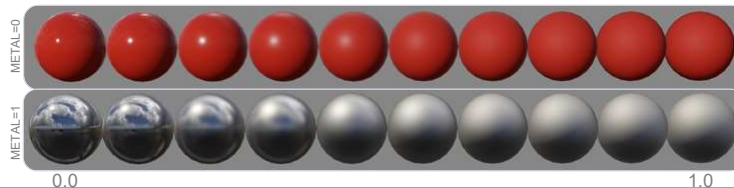


- Current popular choice of parameters:

- **Base color** (rgb – or “diffuse”, same as old school)
- **Specularity** (scalar – or rgb sometimes)
- **“Metallicity”** (scalar)



- **Roughness** (scalar)



images: unreal engine 4

44

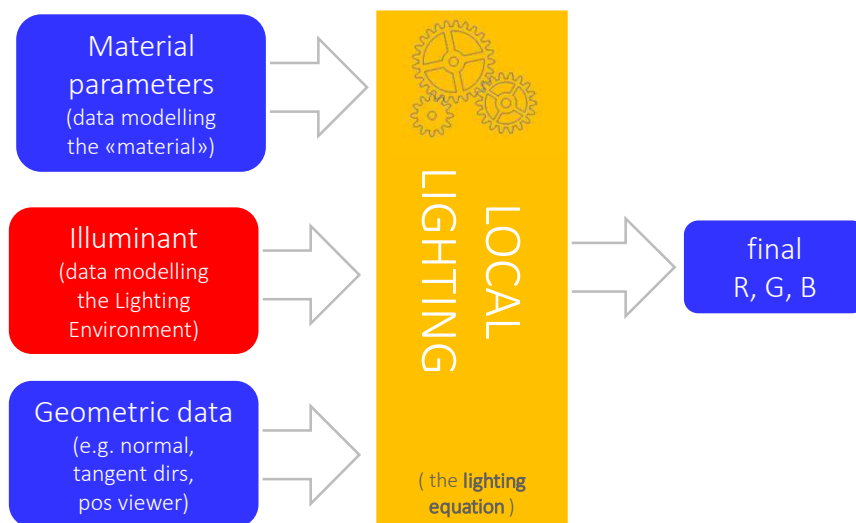
«Physically Based Lighting» (PBL)



- A **lighting model** accepting, as input, a **PBM**
 - note: this can be achieved with different equations
- Also, a **lighting model** taking fewer shortcuts than otherwise typical
 - Before PBM this shortcut was common. Instead of:
 - diffuse color: *one* texture
 - baked AO: *a separate* texture
 - Use one texture:
 - diffuse color × baked AO : one texture (cheaper!)
- Objectives: same as PBM (exp. under “realism”)
- Warning: PBM & PBL are, basically, buzzwords 😊

45

Next: modelling the Light environment



46