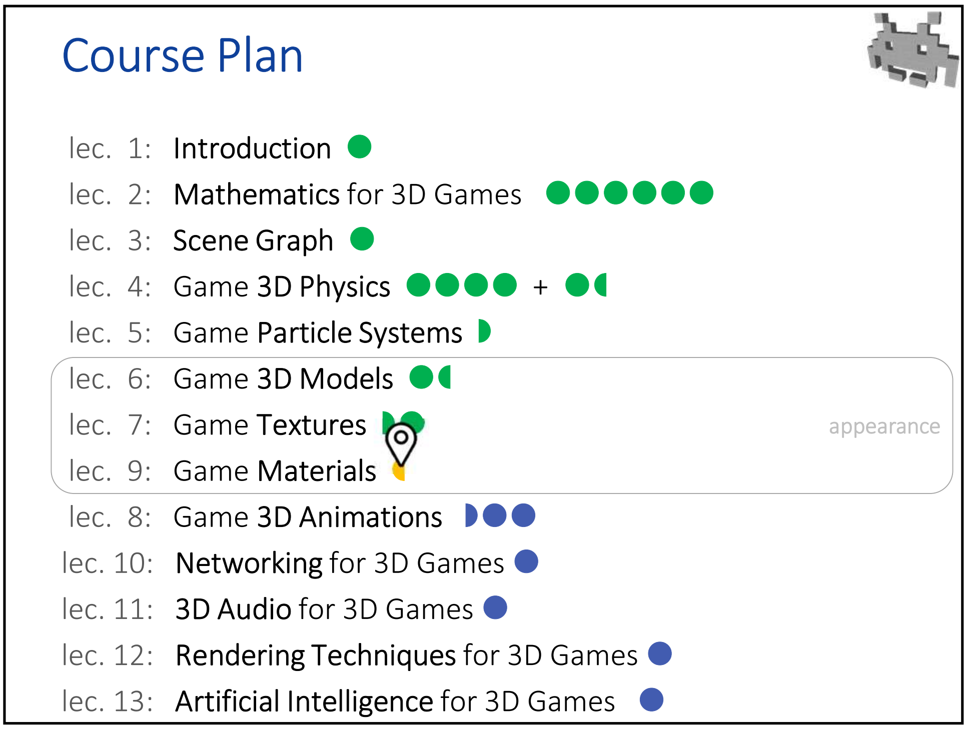


Course Plan

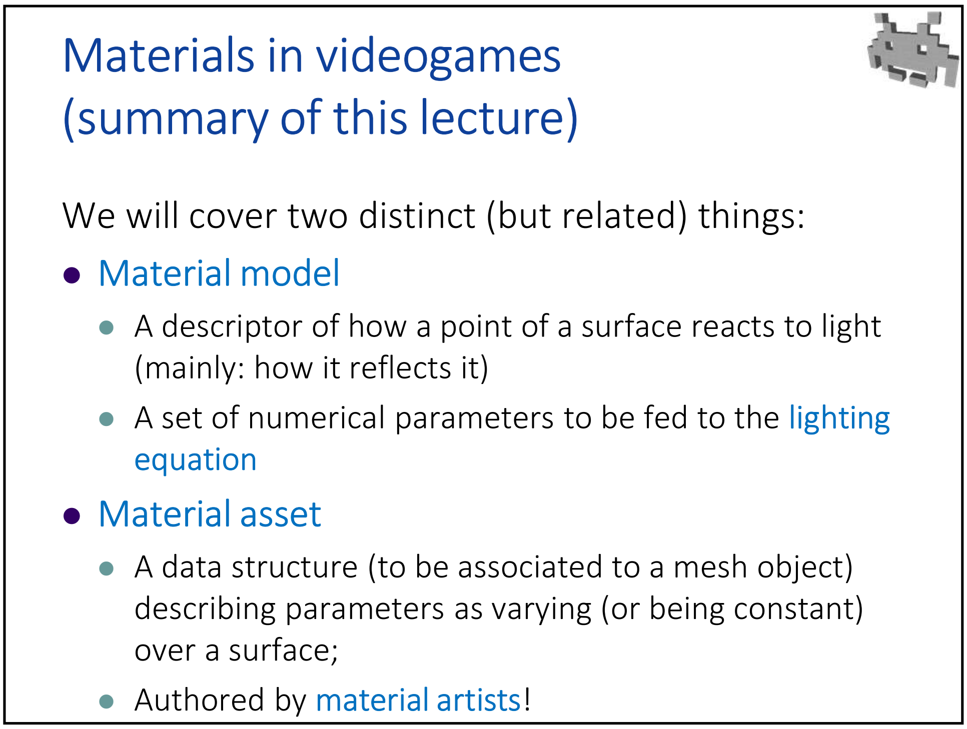
lec. 1: Introduction ●
lec. 2: Mathematics for 3D Games ●●●●●●
lec. 3: Scene Graph ●
lec. 4: Game 3D Physics ●●●● + ●◖
lec. 5: Game Particle Systems ◗
lec. 6: Game 3D Models ●◖
lec. 7: Game Textures ◗●
lec. 9: Game Materials ◖
appearance
lec. 8: Game 3D Animations ◗●●
lec. 10: Networking for 3D Games ●
lec. 11: 3D Audio for 3D Games ●
lec. 12: Rendering Techniques for 3D Games ●
lec. 13: Artificial Intelligence for 3D Games ●

1

Univ. degli Studi di Milano

Materials in videogames

2

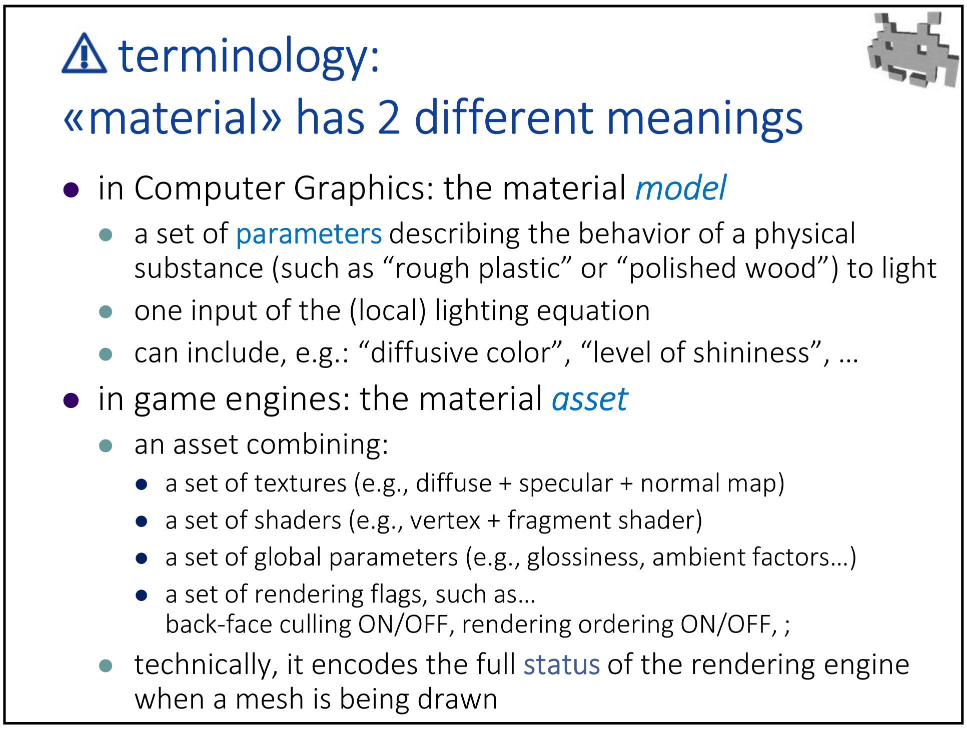


Materials in videogames (summary of this lecture)

We will cover two distinct (but related) things:

- Material model
  - A descriptor of how a point of a surface reacts to light (mainly: how it reflects it)
  - A set of numerical parameters to be fed to the lighting equation
- Material asset
  - A data structure (to be associated to a mesh object) describing parameters as varying (or being constant) over a surface;
  - Authored by material artists!

3

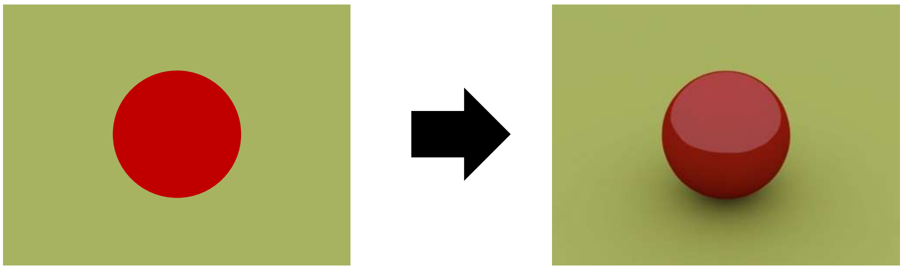


⚠ terminology: «material» has 2 different meanings

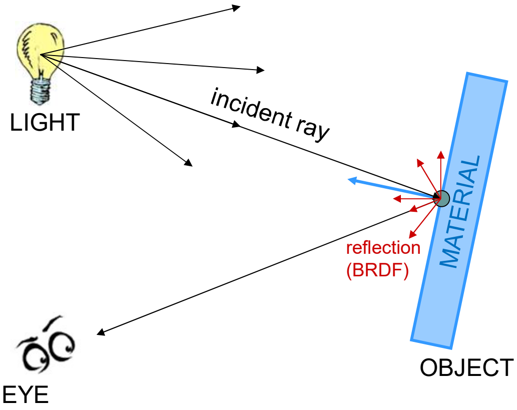
- in Computer Graphics: the material *model*
  - a set of parameters describing the behavior of a physical substance (such as “rough plastic” or “polished wood”) to light
  - one input of the (local) lighting equation
  - can include, e.g.: “diffusive color”, “level of shininess”, …
- in game engines: the material *asset*
  - an asset combining:
    - a set of textures (e.g., diffuse + specular + normal map)
    - a set of shaders (e.g., vertex + fragment shader)
    - a set of global parameters (e.g., glossiness, ambient factors…)
    - a set of rendering flags, such as…
      back-face culling ON/OFF, rendering ordering ON/OFF, ;
  - technically, it encodes the full status of the rendering engine when a mesh is being drawn

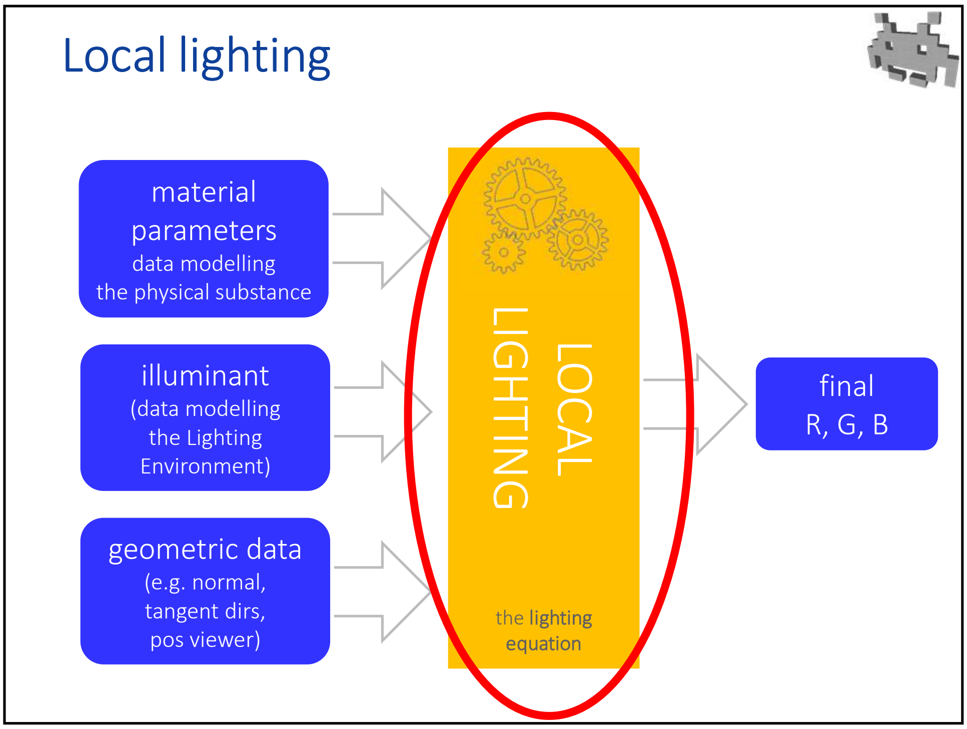
4

Background:
Lighting (task of the rendering)



Local Lighting





Local lighting

material parameters
data modelling the physical substance

illuminant
(data modelling the Lighting Environment)

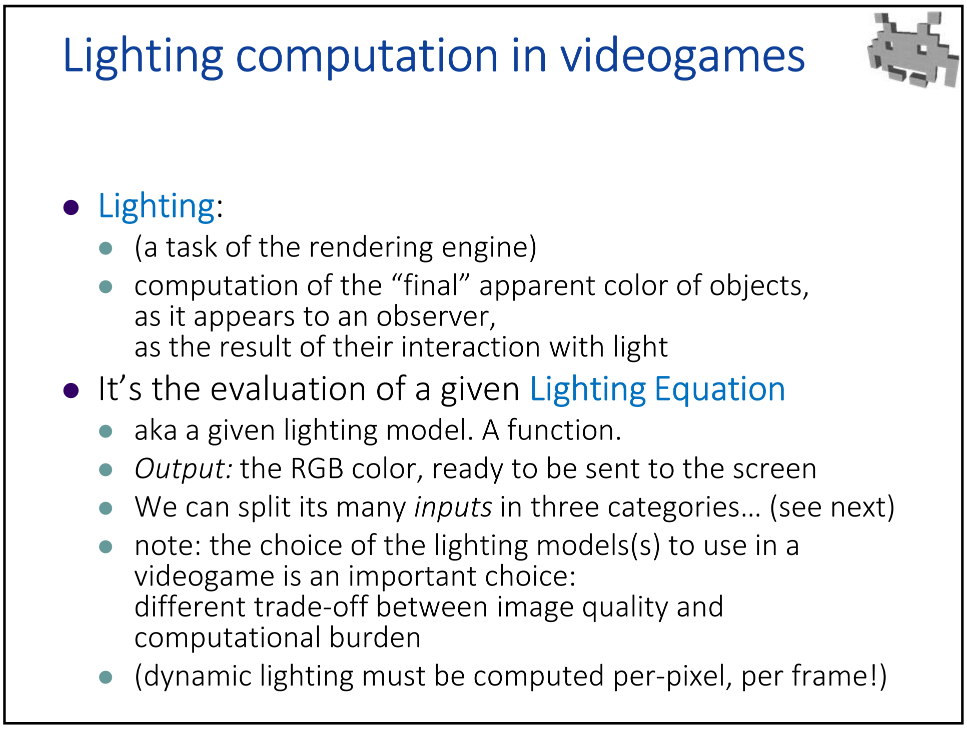
geometric data
(e.g. normal, tangent dirs, pos viewer)

LOCAL LIGHTING

the lighting equation

final
R, G, B

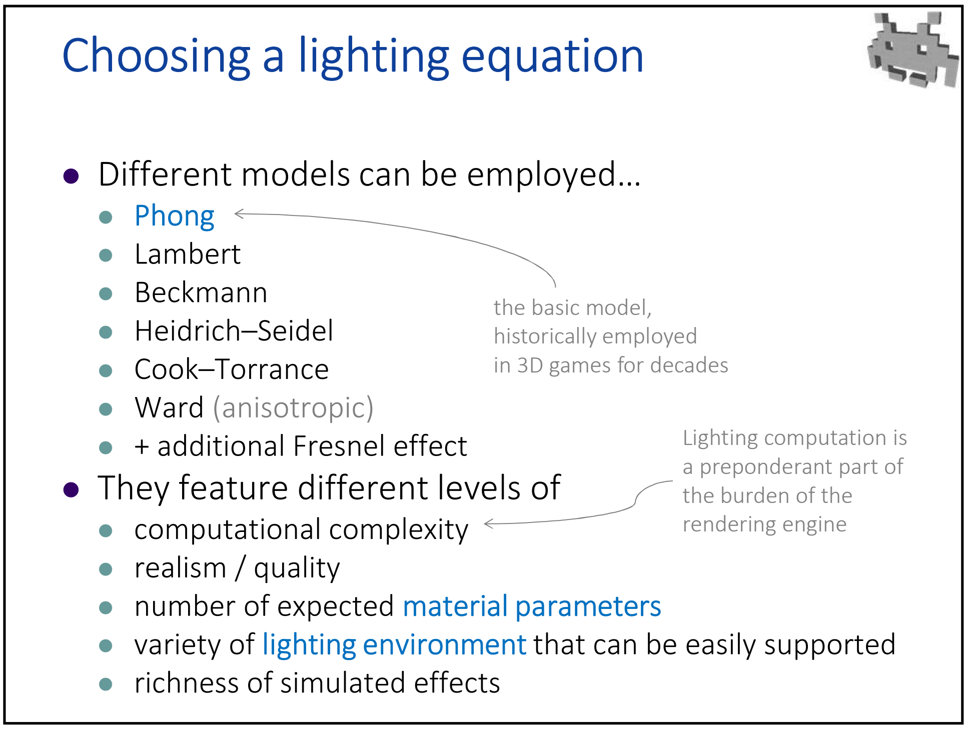
7



Lighting computation in videogames

- Lighting:
  - (a task of the rendering engine)
  - computation of the “final” apparent color of objects,
    as it appears to an observer,
    as the result of their interaction with light
- It’s the evaluation of a given Lighting Equation
  - aka a given lighting model. A function.
  - *Output:* the RGB color, ready to be sent to the screen
  - We can split its many *inputs* in three categories… (see next)
  - note: the choice of the lighting models(s) to use in a videogame is an important choice:
    different trade-off between image quality and computational burden
  - (dynamic lighting must be computed per-pixel, per frame!)

8



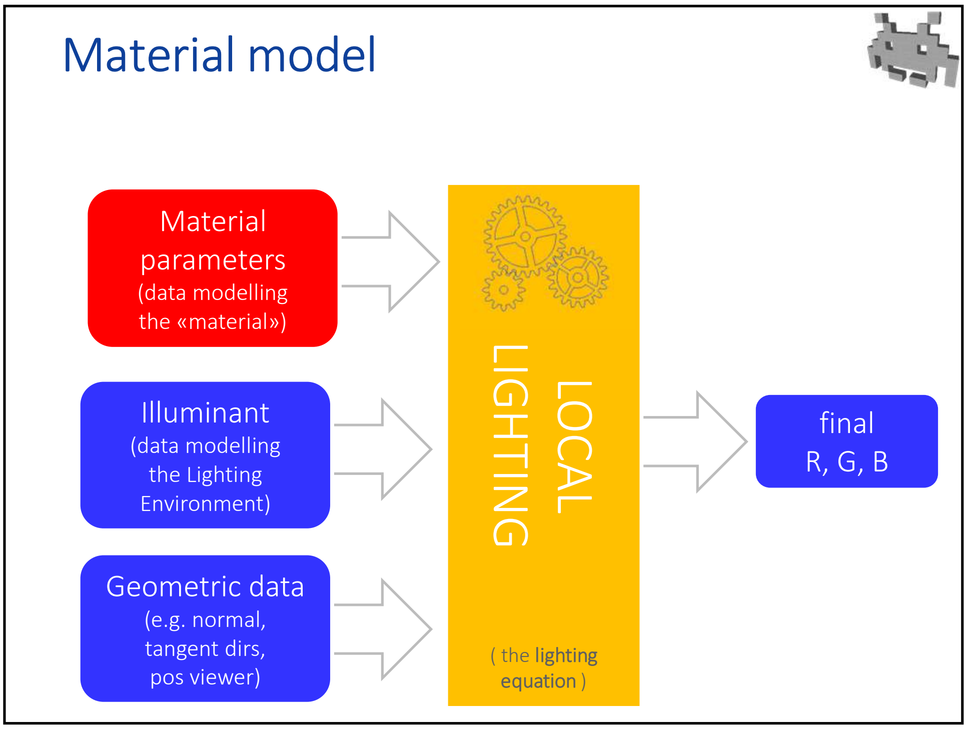
Choosing a lighting equation

- Different models can be employed…
  - Phong
  - Lambert
  - Beckmann
  - Heidrich–Seidel
  - Cook–Torrance
  - Ward (anisotropic)
  - + additional Fresnel effect
- They feature different levels of
  - computational complexity
  - realism / quality
  - number of expected material parameters
  - variety of lighting environment that can be easily supported
  - richness of simulated effects

the basic model, historically employed in 3D games for decades

Lighting computation is a preponderant part of the burden of the rendering engine

9



Material model

Material parameters (data modelling the «material»)

Illuminant (data modelling the Lighting Environment)

Geometric data (e.g. normal, tangent dirs, pos viewer)

LOCAL LIGHTING

(the lighting equation)

final R, G, B

10

Material *Model* = a description of how a physical surface / substance reacts to light

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

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

- whichever is the answer,
  each parameter can be stored:
  - as global parameters, or
  - per Vertex of a Mesh, as attributes, or
  - per Texel of a texture sheet (maximal freedom)

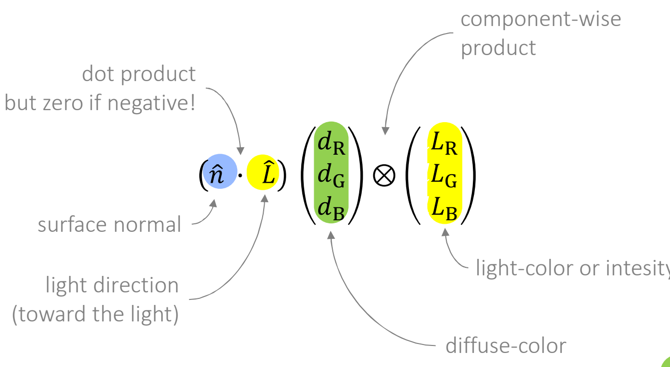
} uniform mat

} non-uniform, aka "spatially varying", material

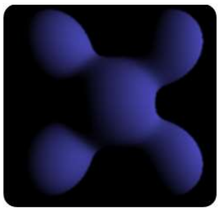
11

Most basic lighting equation: diffuse lighting («Lambertian»)

- The formula:



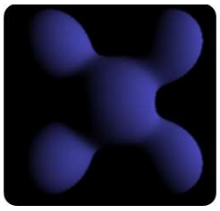
$$(\hat{n} \cdot \hat{L}) \begin{pmatrix} d_R \\ d_G \\ d_B \end{pmatrix} \otimes \begin{pmatrix} L_R \\ L_G \\ L_B \end{pmatrix}$$



● material parameter
● light parameter
● geometry

12

Most basic lighting equation: diffuse lighting («Lambertian»)



- info:
  - it's physically based
  - approximates well dull-looking stuff (e.g., plasters, untreated wood)
  - used as a term in most lighting equations
- expected material parameters:
  - base color, (called albedo, when grayscale), aka diffuse color, Lambertian color, or sometimes just color
  - remember non uniform materials, the texture used to specify it is called diffuse-map or 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

13

The intuition behind the diffuse formula

- When the normal direction is similar to the light direction, then the surface is oriented directly toward the light, so, it looks brighter
  - … from any viewing direction!
  - this is a view-independent effect: the view-direction is not involved in the formula

The local orientation of the surface $\hat{n}$

The dot product between versors is a measure of their similarity!

The direction $\hat{L}$ the light is coming *from*

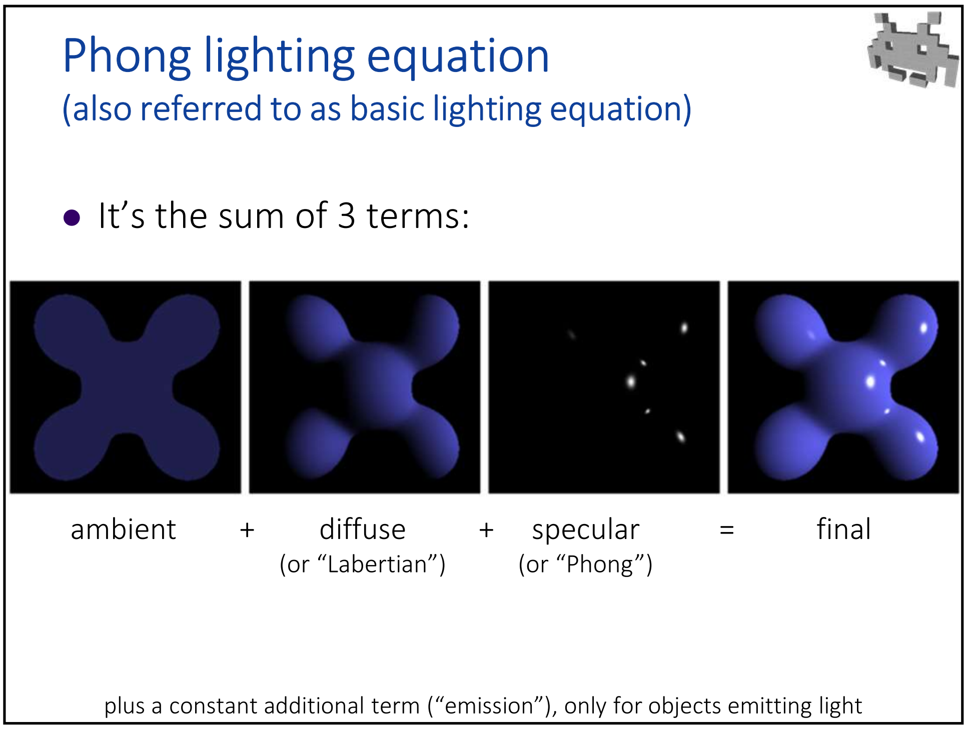
14

Material models

Phong lighting equation

$$\overbrace{\overbrace{(\hat{n}\cdot\hat{L})\begin{pmatrix}d_{\mathrm{R}}\\d_{\mathrm{G}}\\d_{\mathrm{B}}\end{pmatrix}\otimes\begin{pmatrix}L_{\mathrm{R}}\\L_{\mathrm{G}}\\L_{\mathrm{B}}\end{pmatrix}}^{\text{diffuse term}}+\overbrace{(\hat{n}\cdot\underbrace{\hat{H}}_{\substack{\text{nlerp}(\hat{V},\,\hat{L},\,0.5)\\ \text{the «half-way» vector}}})^{E}\begin{pmatrix}s_{\mathrm{R}}\\s_{\mathrm{G}}\\s_{\mathrm{B}}\end{pmatrix}\otimes\begin{pmatrix}L_{\mathrm{R}}\\L_{\mathrm{G}}\\L_{\mathrm{B}}\end{pmatrix}}^{\text{specular term}}}^{\text{repeat and sum for each light source}}+\overbrace{\overbrace{\begin{pmatrix}a_{\mathrm{R}}\\a_{\mathrm{G}}\\a_{\mathrm{B}}\end{pmatrix}\otimes\begin{pmatrix}A_{\mathrm{R}}\\A_{\mathrm{G}}\\A_{\mathrm{B}}\end{pmatrix}}^{\text{ambient term}}+\overbrace{\begin{pmatrix}e_{\mathrm{R}}\\e_{\mathrm{G}}\\e_{\mathrm{B}}\end{pmatrix}}^{\text{emission term}}}^{\text{add only once}}$$

- ● material parameter
- ● light parameter
- ● geometry



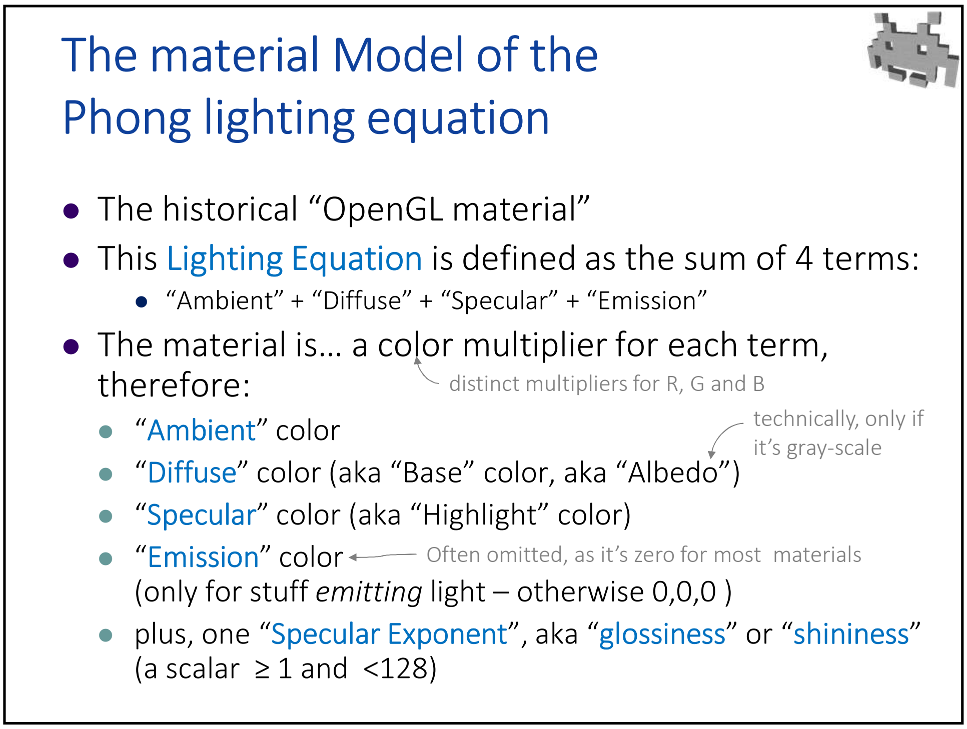
Phong lighting equation
(also referred to as basic lighting equation)

- It's the sum of 3 terms:

ambient + diffuse (or "Labertian") + specular (or "Phong") = final

plus a constant additional term ("emission"), only for objects emitting light

17

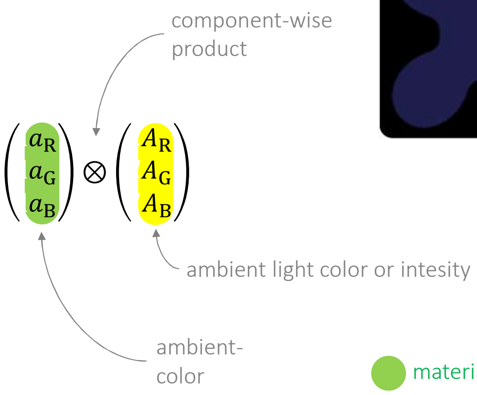


The material Model of the Phong lighting equation

- The historical "OpenGL material"
- This Lighting Equation is defined as the sum of 4 terms:
  - "Ambient" + "Diffuse" + "Specular" + "Emission"
- The material is… a color multiplier for each term, therefore:
  distinct multipliers for R, G and B
  - "Ambient" color
  - "Diffuse" color (aka "Base" color, aka "Albedo")
    technically, only if it's gray-scale
  - "Specular" color (aka "Highlight" color)
  - "Emission" color ← Often omitted, as it's zero for most materials
    (only for stuff *emitting* light – otherwise 0,0,0)
  - plus, one "Specular Exponent", aka "glossiness" or "shininess"
    (a scalar ≥ 1 and <128)

18

A basic lighting equation: ambient term

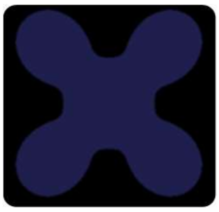


component-wise product

$$\begin{pmatrix} a_\mathrm{R} \\ a_\mathrm{G} \\ a_\mathrm{B} \end{pmatrix} \otimes \begin{pmatrix} A_\mathrm{R} \\ A_\mathrm{G} \\ A_\mathrm{B} \end{pmatrix}$$

ambient light color or intesity

ambient-color



● material parameter

● light parameter

● geometry

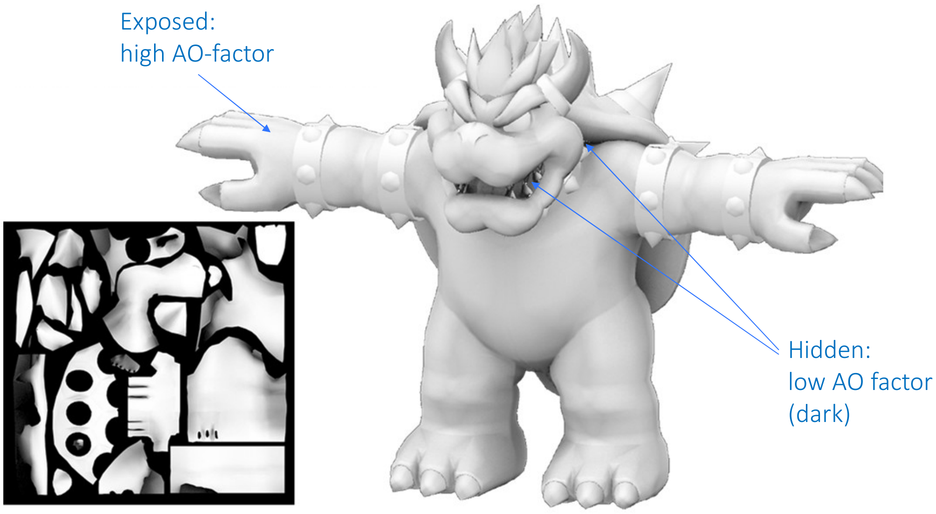
The intuition behind the ambient-term formula

- A bit of light will reach this point of the surface from *all* directions (so, surface normal does not count from this)
- In first approximation, this is proportional to: how exposed is this point of the surface (a constant), and how much light is around overall (another constant)

A 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,
  - e.g. from minor, unmodelled light sources
- without it, things not directly lit by lights are black (and it looks ugly)
  - and it’s very simple to include in the equation, so why not
- uses parameters in the material:
  - ambient-color (RGB)
  - or, ambient-factor (scalar),
    then ambient-color = diffuse-color · ambient-factor
  - also called Ambient Occlusion factor (AO)
- just like any material parameter, it can be :
  - stored in textures: AO map (typically baked)
  - stored in vertices as attributes
  - it can also be computed on the fly (see SSAO, last lecture)

Baking AO map for a model



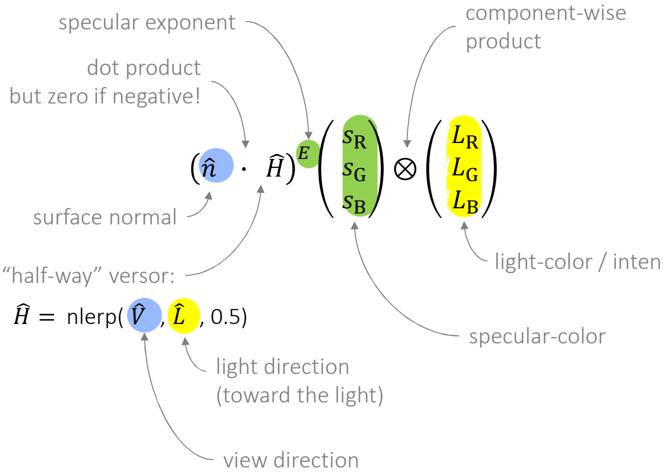
Exposed:
high AO-factor

Hidden:
low AO factor
(dark)

AO map (baked)
NOTE: this requires UV unwrapping (injective UV-map), like any baked texture

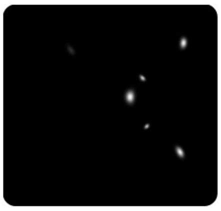
The specular term explained (aka the «Blinn-Phong» equation)

- In formulas:



$$(\hat{n}\cdot\hat{H})^{E}\begin{pmatrix}s_R\\s_G\\s_B\end{pmatrix}\otimes\begin{pmatrix}L_R\\L_G\\L_B\end{pmatrix}$$

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



● material parameter

● light parameter

● geometry

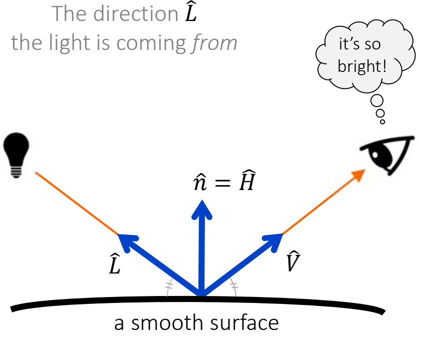
The intuition behind the specular-term formula

- When the normal direction matches the average of light direction and view direction , the light is reflected straight toward the eye, so, we see a bright reflection on the object
- By exponentiating the factor, I reduce it quickly toward 0, unless it was 1 or close
  - So, I only keep the really good matches

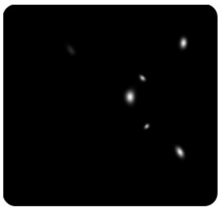
The local orientation of the surface $\hat{n}$

The dot product between versors is a measure in 0 to 1 of their similarity

The direction $\hat{L}$ the light is coming *from*



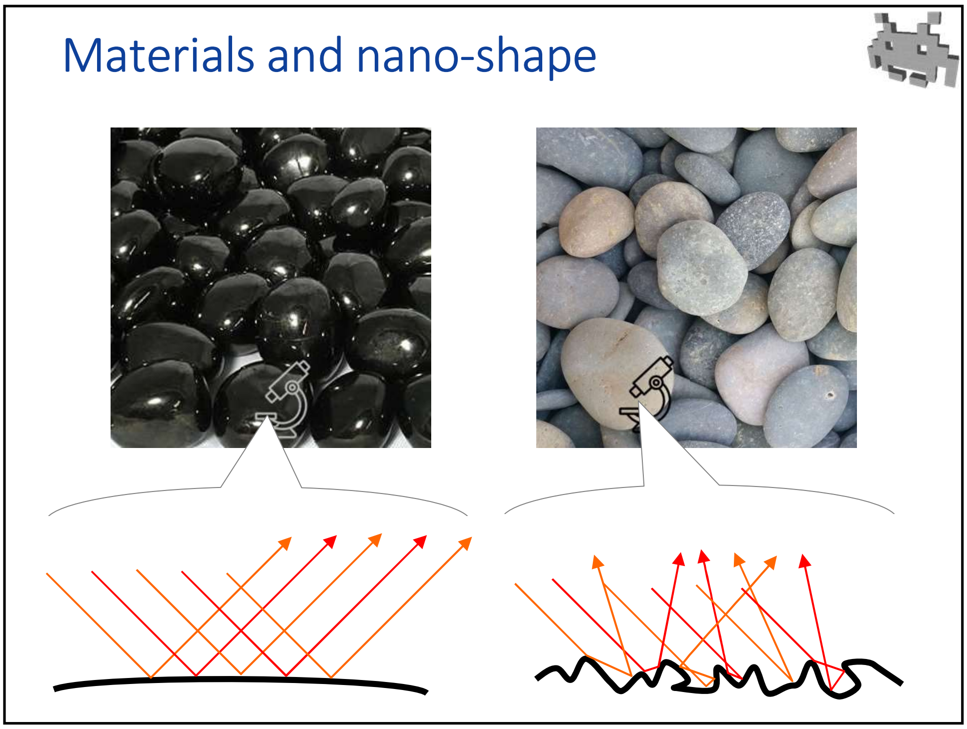
The specular term



- info:
  - not physically based
  - not even energy conserving
  - basically, just a trick
  - simulates reflections (“highlights”)
- material parameters used:
  - specular color
    determines the intensity and color of the highlight
    sometimes: it’s diffuse color x a constant
    often > 1 – oversaturated highlight
  - specular exponent (or “glossiness”)
    determines the SIZE of the highlight
    larger numbers → smaller highlight
- If stored in a texture, it is called:
  - specular map and glossiness map
  - e.g., a 4-channel texture can store: RGB spec color + glossiness

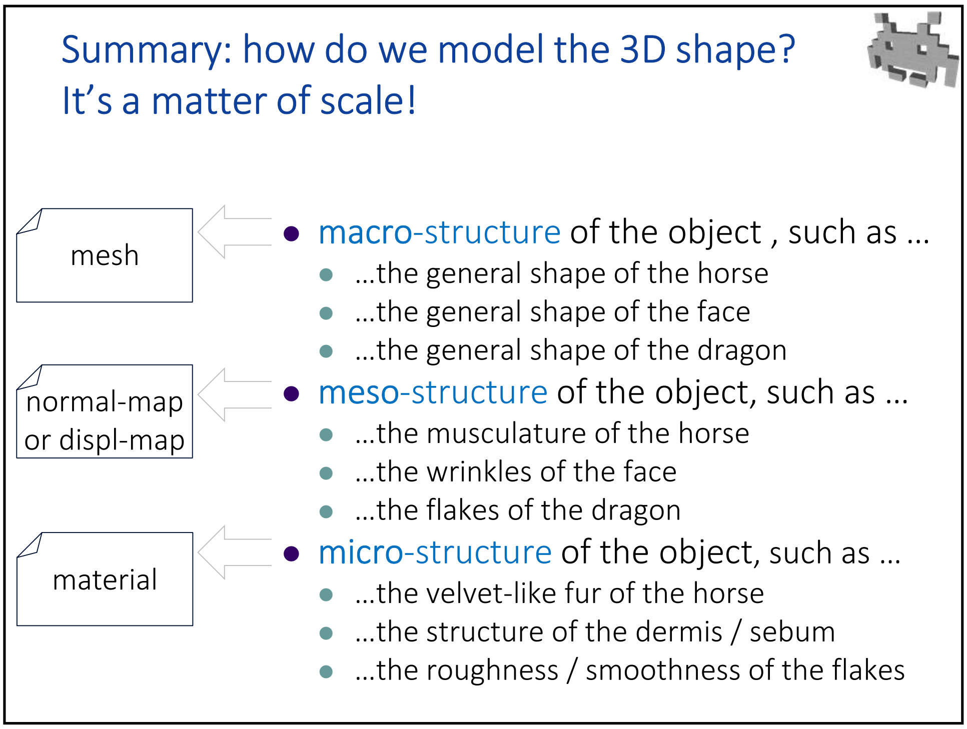
Material Model:
which physical characteristics does it represent?

1) The physical substance (of a point on the surface)
   - Does it absorb photons, does it bounce them away?
   - Is it transparent to photons? (i.e. can photons pass through it?)
   - How does that depend on the frequency, that is, color of the photon? (that’s what gives objects their “color”!)
   - These things depend, in turn, on electric conductivity
   - E.g.: metals look shiny, because (⚠ over-simplification warning ⚠) light bounces off a cloud of shared electrons surrounding them
2) The micro-shape of the surface (around a point), e.g.
   - A polished (smooth, at a micro-scale) surface looks more shiny
   - A wet surface (water layer is very smooth!) looks more shiny
   - A waxed surface (wax layer is smooth!) looks more shiny
   - A rough, unpolished surface looks dull / not shiny



Materials and nano-shape

30



Summary: how do we model the 3D shape? It’s a matter of scale!

mesh

- macro-structure of the object , such as …
  - …the general shape of the horse
  - …the general shape of the face
  - …the general shape of the dragon

normal-map or displ-map

- meso-structure of the object, such as …
  - …the musculature of the horse
  - …the wrinkles of the face
  - …the flakes of the dragon

material

- micro-structure of the object, such as …
  - …the velvet-like fur of the horse
  - …the structure of the dermis / sebum
  - …the roughness / smoothness of the flakes

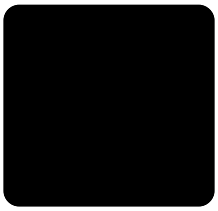
31

A basic lighting equation: emission term

- In formulas:

$$\begin{pmatrix} e_R \\ e_G \\ e_B \end{pmatrix}$$

emission-color



most objects have no emission term

● material parameter

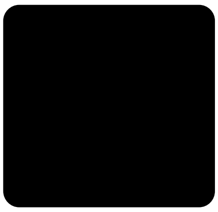
● light parameter

● geometry

32

A basic lighting equation - emission term: info

- it models light that is…
  - …emitted from an object, and
  - …reaches the camera (directly!)
- useful for, e.g., small led lights, that are visible in the dark
  - for any other object (i.e, most of them), it is zero
- note: the emitted light doesn’t illuminate other objects
  - for this, you need to add lights to the scene
- the texture storing it (if there’s one) is called emission map
- HDR values possible (that is, $e_{R,G,B} > 1$) to get a “glow” effect (see HDR, last lecture)



zero, in this case

33

Defining materials: Chapter 1 ('90s)

- The "Phong" lighting equation has been the *standard* for many years, because
  - It's cheap to compute
  - It's easy to control by material artists
  - Lighting equation was hardwired in graphics API (OpenGL and DirectX), and this was the only model which was provided
- Therefore, the material parameters it uses has been the standard way to define materials (in videogames)
  - Via global parameters, vertex attributes, or textures defining them
- Unfortunately, it's also crude, not realistic, and all materials look similar
  - It's only realistic if the Specular component is zero

How to author a Phong material: (how to pick the parameters)

By hand! Questions a *material artist* must ask him/herself

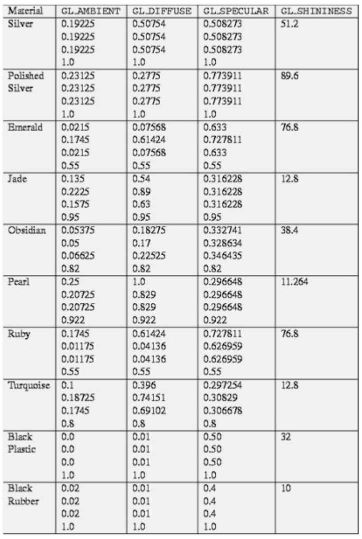
- Diffuse color (aka Base color, aka Albedo):
  - "Which color is this stuff?"
  - i.e., "Which color does it look like, if I shine a white light on it?"
- Specular color (aka Highlight color)
  - "Which color and how bright are its reflections?"
  - if it's a factor (and Specular-color = Base-color · Specular-factor): "How bright are its reflection?"
- Specular exponent (aka Glossiness) (in 1 to 128)
  - "How concentrated are its reflections?"
  - Larger value (e.g., 100) ==> more concentrated highlights
  - Smaller value (e.g., 4) ==> larger highlights

Doesn't make any physical sense,
it's just a crude way to simulate the effect.
Real reflections don't work this way!

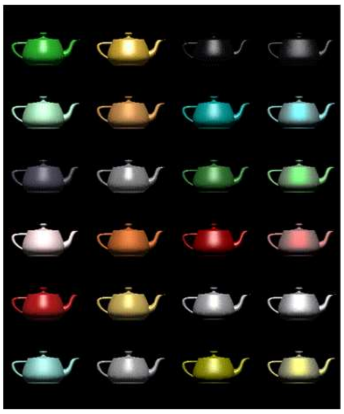
- Ambient color / Ambient (Occlusion) factor (in 0 to 1)
  - How easy it is to reach this point of by ambient light
  - Small values: this point is difficult to reach by light
  - Larger values: this point is well exposed, easy to reach

A way to remedy the fact that we are not modelling a realistic light environment

Defining materials: Chapter 1 ('90s)



Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR	GL_SHININESS
Silver	0.19225 0.19225 0.19225 1.0	0.50754 0.50754 0.50754 1.0	0.508273 0.508273 0.508273 1.0	51.2
Polished Silver	0.23125 0.23125 0.23125 1.0	0.2775 0.2775 0.2775 1.0	0.773911 0.773911 0.773911 1.0	89.6
Emerald	0.0215 0.1745 0.0215 0.55	0.07568 0.61424 0.07568 0.55	0.633 0.727811 0.633 0.55	76.8
Jade	0.135 0.2225 0.1575 0.95	0.54 0.89 0.63 0.95	0.316228 0.316228 0.316228 0.95	12.8
Obsidian	0.05375 0.05 0.06625 0.82	0.18275 0.17 0.22525 0.82	0.332741 0.328634 0.346435 0.82	38.4
Pearl	0.25 0.20725 0.20725 0.922	1.0 0.829 0.829 0.922	0.296648 0.296648 0.296648 0.922	11.264
Ruby	0.1745 0.01175 0.01175 0.55	0.61424 0.04136 0.04136 0.55	0.727811 0.626959 0.626959 0.55	76.8
Turquoise	0.1 0.18725 0.1745 0.8	0.396 0.74151 0.69102 0.8	0.297254 0.30829 0.306678 0.8	12.8
Black Plastic	0.0 0.0 0.0 1.0	0.01 0.01 0.01 1.0	0.50 0.50 0.50 1.0	32
Black Rubber	0.02 0.02 0.02 1.0	0.01 0.01 0.01 1.0	0.4 0.4 0.4 1.0	10



not very expressive ☹

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

Problems with the basic (aka Phong) material model

- It's not very expressive
- It's made-up (especially the specular component)
- It isn't realistic

Defining materials: Chapter 2 ('00s)

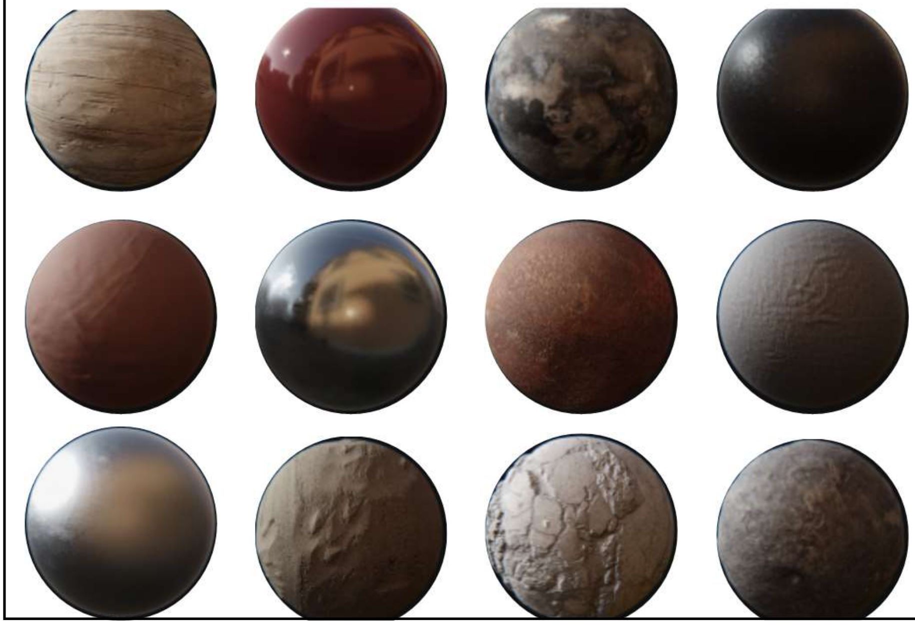
Main reasons:
1. more GPU processing power affords us more realism.
2. Programmable shaders.
3. More GPU RAM to store textures for parameters

- The Lighting Equation becomes more complex
  - more terms are added
- It feeds on more material parameters…
  - Factors for: 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"

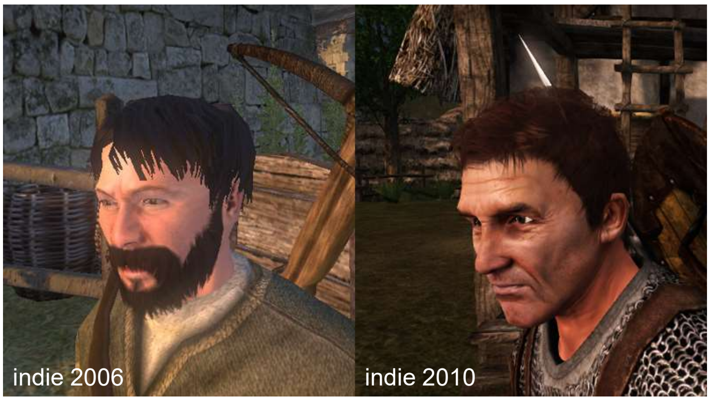
39

Much wider expressiveness is needed



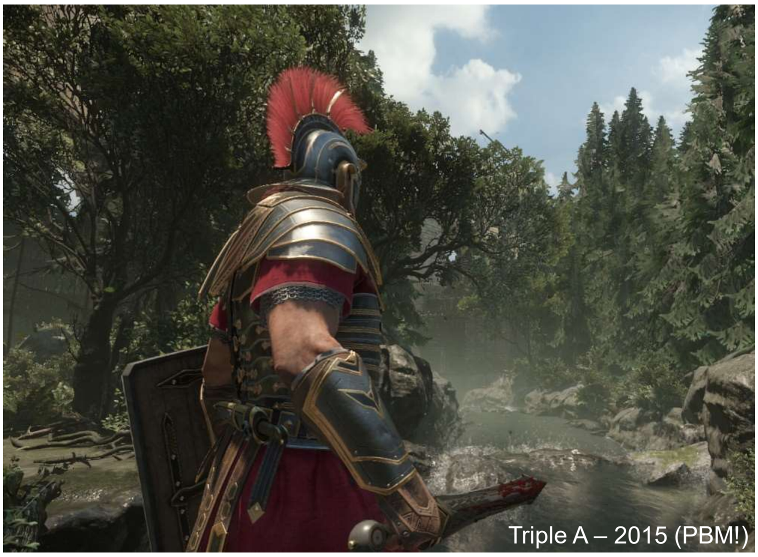
40

Material models are improving

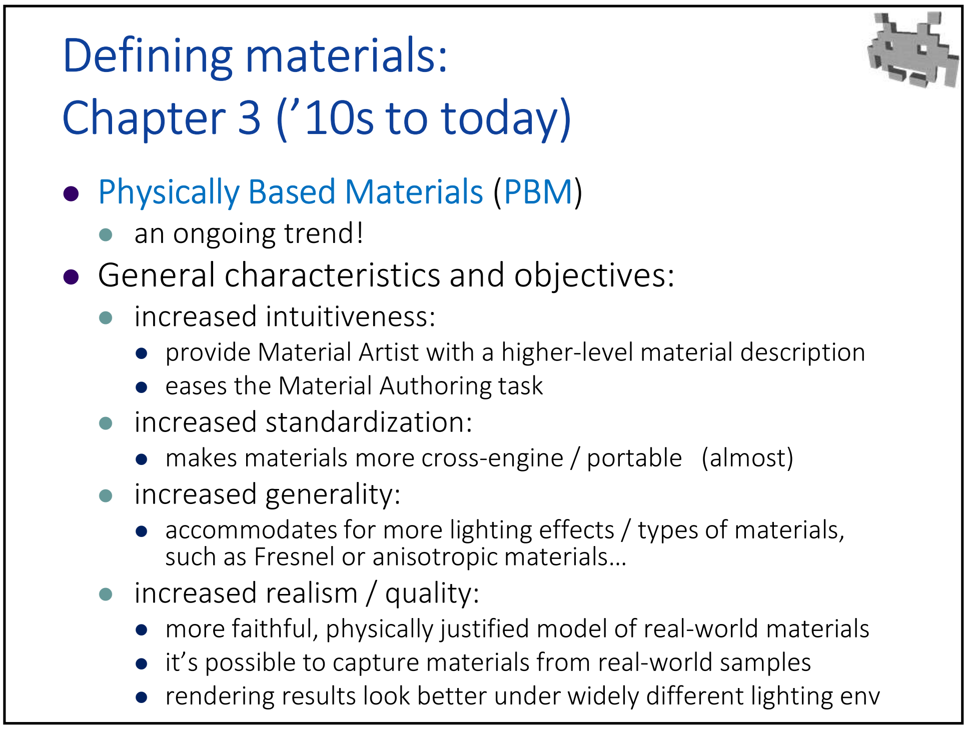


42

Material models are improving

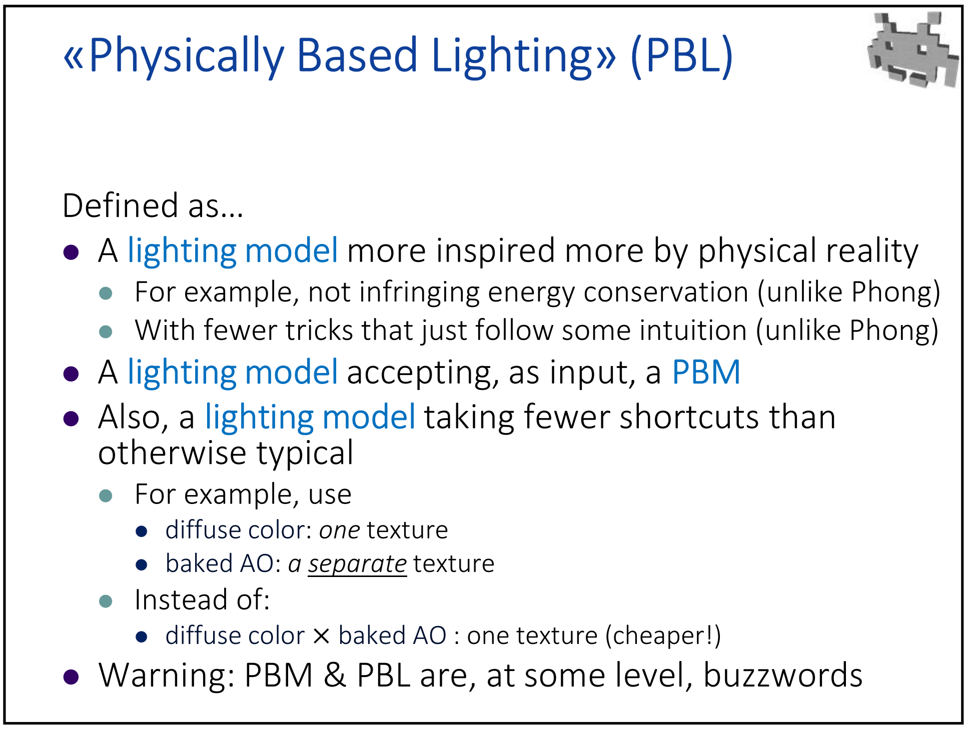


43



Defining materials: Chapter 3 ('10s to today)

- 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 more lighting effects / types of materials, such as Fresnel or anisotropic materials…
  - increased realism / quality:
    - more faithful, physically justified model of real-world materials
    - it's possible to capture materials from real-world samples
    - rendering results look better under widely different lighting env



«Physically Based Lighting» (PBL)

Defined as…

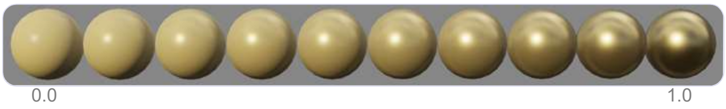
- A lighting model more inspired more by physical reality
  - For example, not infringing energy conservation (unlike Phong)
  - With fewer tricks that just follow some intuition (unlike Phong)
- A lighting model accepting, as input, a PBM
- Also, a lighting model taking fewer shortcuts than otherwise typical
  - For example, use
    - diffuse color: *one* texture
    - baked AO: *a <u>separate</u>* texture
  - Instead of:
    - diffuse color × baked AO : one texture (cheaper!)
- Warning: PBM & PBL are, at some level, buzzwords

PBM and PBL: objectives

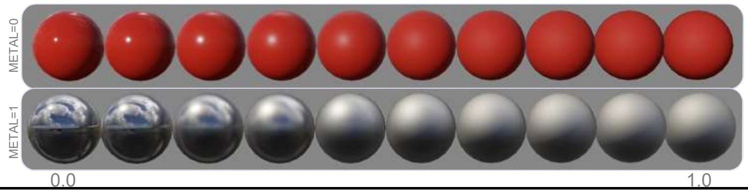
- Make realistic-looking materials easier to design (by material artists)
- Make it possible to *capture* materials from Real World samples
- Make it easier to that a reasonably realistic-looking lighting … under a wider range of simulated lighting conditions
- Standardize Materials, and make it easier to share a material description across different project / applications / games
- Ideally: more real-world materials can be described accurately using a unified model
- Use few parameters (to ease storage, authoring, editing), with each parameter standardized in a 0 to 1 range
- Ideally: no combination of Material parameters looks wrong (unlike, e.g., high spec. color & low specular exp, in phong)

Physically Based Materials (PBM). A good choice of parameters

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

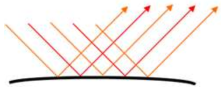
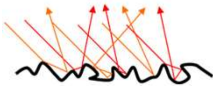


- Roughness (scalar)



images: unreal engine 4

Physically Based Materials (PBM)

- Base color: (a color)
  - Same as in base lighting model
- Specularity: (a scalar, in 0 to 1)
  - Total amount of light bouncing off the surface with reflections (regardless of how).
  - Barring exception, it is usually high (closer to 1 than to 0): "Everything is shiny", even if in different ways,
- Metallicity (or metallosity): (a scalar, in 0 to 1)
  - Is the surface a (conductive, dielectric) material or not?
  - In theory, either 0 or 1, but it's possible to interpolate results.
- Roughness: (a scalar, in 0 to 1)
  - Low =  High = 

48

Uniform VS non-uniform materials

- Uniform materials:
  the parameters are shared by the entire surface
  - They are stored as variables as a part of the material asset parameters
- Non-uniform Material:
  the parameters are different in each part of the surface
  - They can be stored as vertex attributes of the mesh *(for smooth, low-frequency variations)*
  - They can be stored as textures *(for highest-frequency variations)*
- This choice is done for each material parameter separately
  - and also for the normal / tangent dirs
  - for example, where to store
    (1) diffuse color, (2) specular color, (3) normal, (4) tangent dirs. ?
    (typical answers: 1,2, sometimes 3: textures. 3,4: per vertex attribute)

51

A material *Asset*

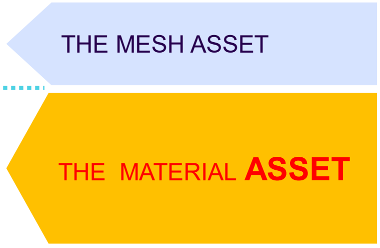
A data structure describing the parameters of a material model, and also:

- How are they stored, for example…
  - …as texels of *this* texture sheet?, or,
  - …as *these* global values (when the material is "uniform")?, or,
  - …as whichever attributes to be found in the rendered mesh?
- It can include bump-maps (any kind, e.g., normal-maps):
  - which technically, describe the object *shape*, not its *material model*
- In can describe algorithm used to compute the lighting, that is…
  - a set of "shader" programs that gather the parameters, and compute the lighting equation (see lecture on rendering)
  - flags and settings for the GPU rendering (e.g.: activation/deactivation of back-face culling, depth test, alpha blending, etc)
  - which rendering passes must be done
  - instruction affecting the rendering order, etc

Material Asset = status of renderer

To render a mesh…

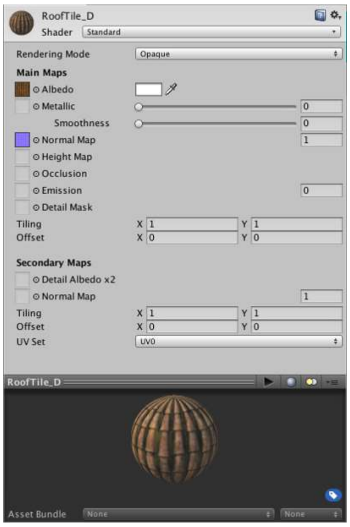
- Load…
  - make sure all data is ready in GPU RAM
    - Geometry + Attributes
    - Connectivity
    - Textures
    - Shaders
    - Material Global Params
    - Rendering Settings
- …and Fire!
  - issue the Draw Call



THE MESH ASSET

THE MATERIAL **ASSET**

Material Assets (examples)



RoofTile_D
Shader: Standard

Rendering Mode: Opaque

Main Maps

- Albedo
- Metallic: 0
- Smoothness: 0
- Normal Map: 1
- Height Map
- Occlusion
- Emission: 0
- Detail Mask

Tiling: X 1, Y 1
Offset: X 0, Y 0

Secondary Maps

- Detail Albedo x2
- Normal Map: 1

Tiling: X 1, Y 1
Offset: X 0, Y 0
UV Set: UV0

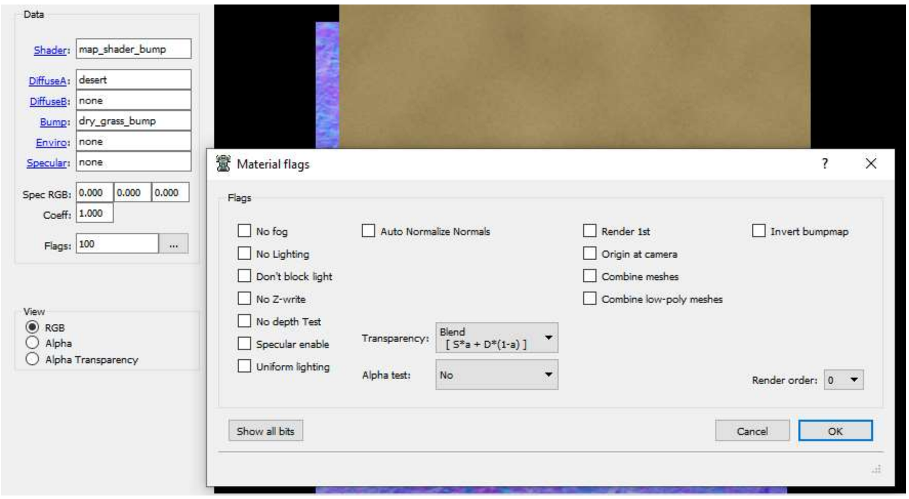
RoofTile_D

Asset Bundle: None, None

Panel for assets of type material in Unity

55

Material Assets (examples)



Data

Shader: map_shader_bump
DiffuseA: desert
DiffuseB: none
Bump: dry_grass_bump
Enviro: none
Specular: none
Spec RGB: 0.000 0.000 0.000
Coeff: 1.000
Flags: 100

View
● RGB
○ Alpha
○ Alpha Transparency

Material flags

Flags

□ No fog
□ No Lighting
□ Don't block light
□ No Z-write
□ No depth Test
□ Specular enable
□ Uniform lighting

□ Auto Normalize Normals

Transparency: Blend [S*a + D*(1-a)]
Alpha test: No

□ Render 1st
□ Origin at camera
□ Combine meshes
□ Combine low-poly meshes

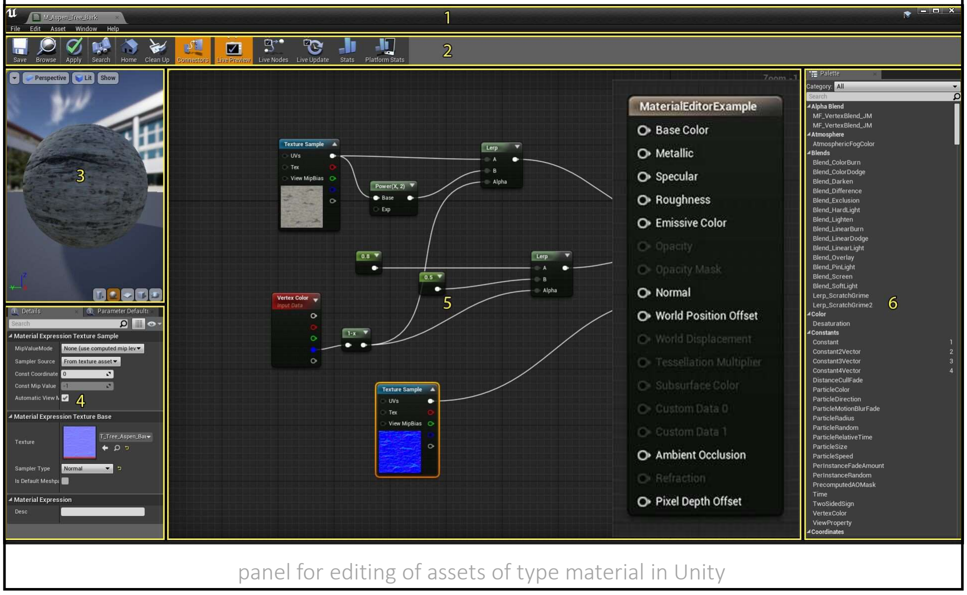
□ Invert bumpmap

Render order: 0

Panel for assets of type material in an indie game tool for asset editing (openBRF)

56

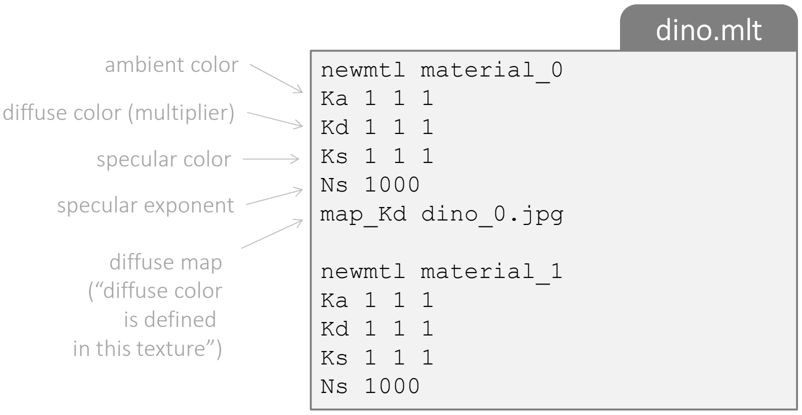
Material Assets (examples)



panel for editing of assets of type material in Unity

57

Material Assets (examples)



```
newmtl material_0
Ka 1 1 1
Kd 1 1 1
Ks 1 1 1
Ns 1000
map_Kd dino_0.jpg

newmtl material_1
Ka 1 1 1
Kd 1 1 1
Ks 1 1 1
Ns 1000
```

Material asset in format MLT – Material Template Library
(part of OBJ mesh file format)

58