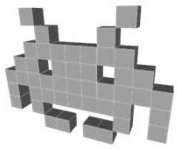
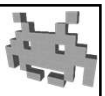
3D videogames – Marco Tarini
Università degli Studi di Milano



Materials in videogames

1

Course Plan



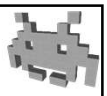
lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics** +
lec. 5: Game **Particle Systems**
lec. 6: Game **3D Models**
lec. 7: Game **Materials**
lec. 8: Game **Textures**

★ appearance

lec. 9: Game **3D Animations**
lec. 10: **3D Audio** for 3D Games
lec. 11: **Networking** for 3D Games
lec. 12: **Interactive Agents** for 3D Games
lec. 13: **Rendering Techniques** for 3D Games

2

«Materials» in videogames (summary of this lecture)



Last time, we have seen how meshes model the 3D shape of physical objects in a videogame (e.g. "which shape is this soccer ball?").

«Materials» complete that by modelling the appearance of mesh surfaces (e.g. "which color is this soccer ball?").

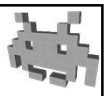
Today we will cover two distinct (but related) subjects:

- Material models
  - A point-wise descriptor of how a point of a surface reacts to light (mainly: how it reflects it)
  - That is: a set of (numerical) parameters to be fed to the lighting equation
- Material assets
  - Data structures describing an arrangement of materials varying over the surface of a mesh;
  - It's associated to a mesh object
  - It's an asset, authored by material artists

or not (constant) as a special case

3

Material model (in computer graphics)

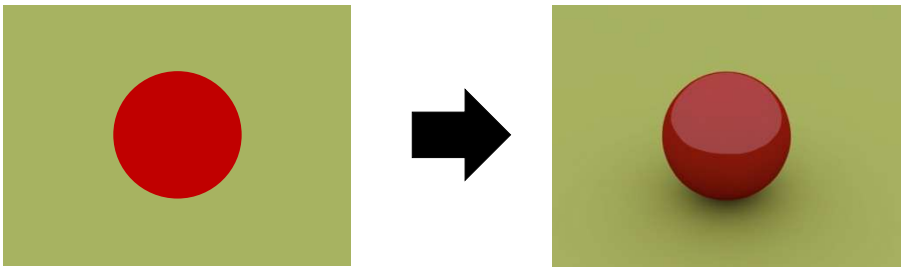


- a set of parameters modelling the pointwise behavior to light of a physical objects
  - basically, describes what a surface *looks like*
- a given material can model, for example, the appearance of…
  - rough plastic
  - polished wood
  - porcelain
  - a specific metal (e.g. gold or silver), etc
- it consists of parameters like:
  - "(diffusive) color", amount of "shininess", "roughness"…
- under the hood: it's a part of the inputs of the lighting equation

To understand this topic, we will need to briefly discuss the computation of lighting (in videogames)…

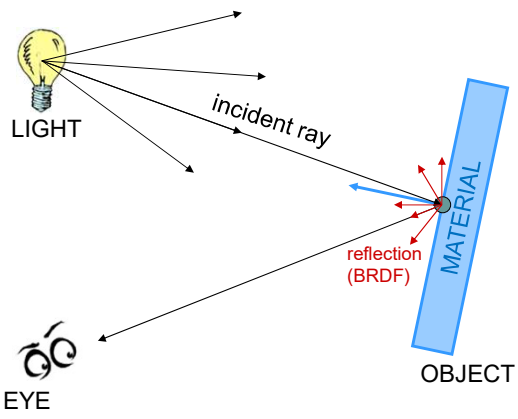
4

Background:
Lighting (task of the rendering)



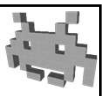
5

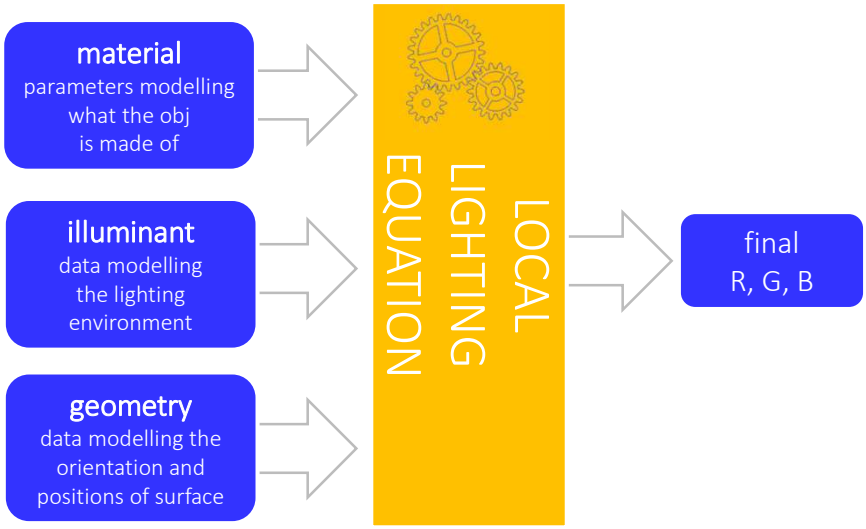
(Local) Lighting



6

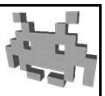
Lighting computation in practice





7

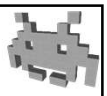
Lighting computation in videogames



- One 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 consists in the evaluation of a given Lighting Equation
  - aka a given lighting model. It's a function…
  - Its *output:* the RGB color, ready to be sent to the screen
  - We can split its many *inputs* in three categories (see previous slide)
- Repeated for each pixel, for every frame!
  - Extremely work intensive: hundreds million of computations needed per sec
- Which lighting models(s) to support in a videogame? An important technical choice (of the engine)!
  - different trade-off between image quality and computational burden
  - Note: most engines also support custom lighting models

8

Choosing a lighting equation



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

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

lighting computation is a huge part of the burden of the rendering engine

9

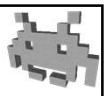
Material = a formal description of how a physical surface / substance reacts to light

- 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

10

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



- The formula:

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

dot product but zero if negative!

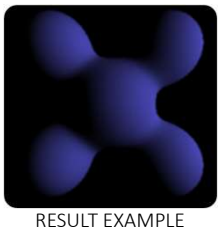
component-wise product

surface normal

light direction (toward the light)

diffuse-color

light-color or intensity



RESULT EXAMPLE

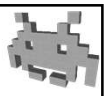
material parameter

light parameter

geometry

12

The intuition behind the diffuse lighting formula



$\hat{n}$, the local orientation of the surface:

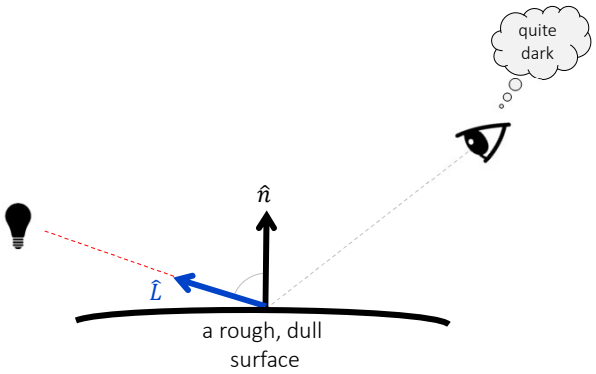
The dot product between versors is a measure of their similarity!

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

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

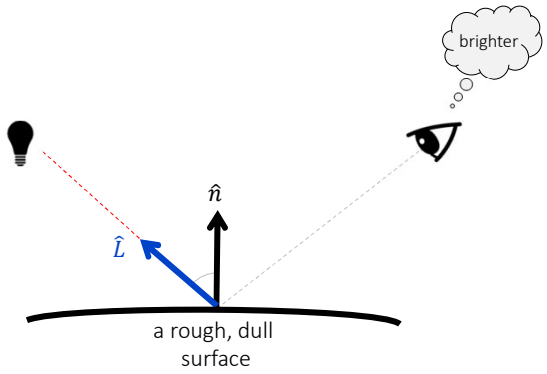
13

The intuition behind the Lambertian equation



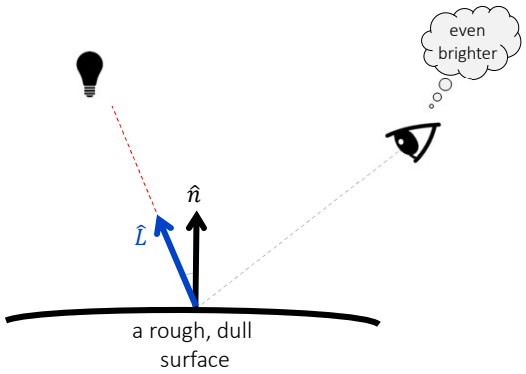
14

The intuition behind the Lambertian equation



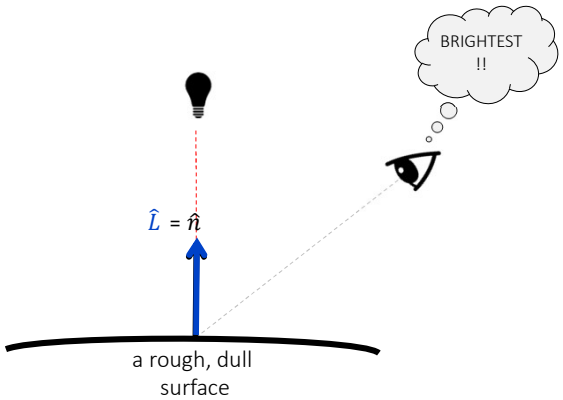
15

The intuition behind the Lambertian equation



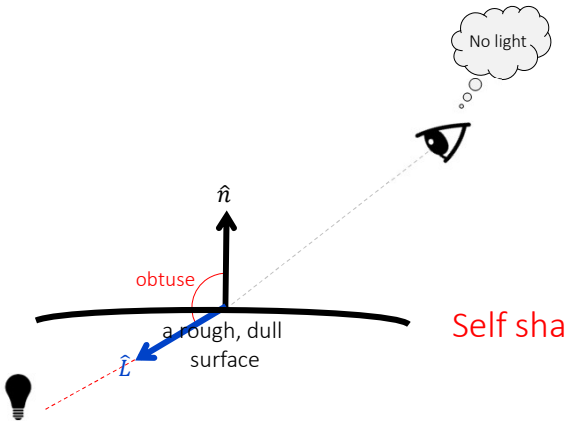
16

The intuition behind the Lambertian equation



17

The intuition behind the Lambertian equation

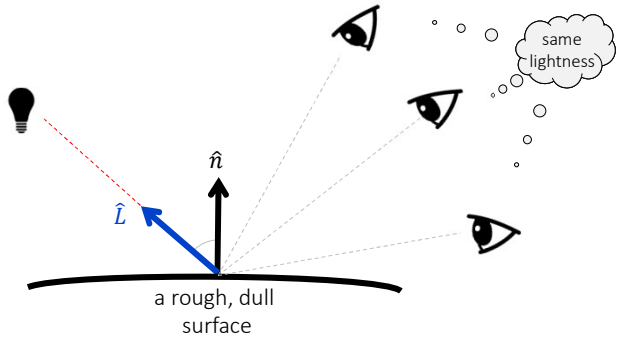


Self shadow!

24

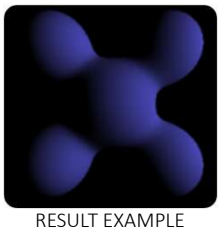
Lambertian equation: it's view-independent!

The view direction is NOT an input of the equation!



25

A most basic lighting equation: diffuse lighting («Lambertian»)


RESULT EXAMPLE

- info:
  - physically justified (it's not just intuitive: it precisely reflects a well-understood phenomenon)
  - approximates well some dull-looking stuff (e.g., plasters, untreated wood)
  - used as a term in most lighting equations
- expected material parameters:
  - base color,
  - aka albedo, when grayscale (one scalar, not three R-G-B)
  - aka diffuse color
  - aka Lambertian color,
  - aka (sometimes): just color

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

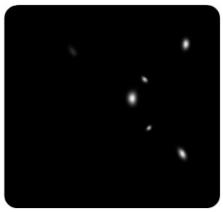
27

What about shiny objects?

- Diffuse lighting is physically based
- Problem: it only reproduces perfectly dull materials (not shiny)
  - Shininess / highlights : it's view-dependent effect
- To support shiny materials, we add a "specular term"
  - Final RGB: diffuse term + specular term
  - Let's see a made-up but simple equation used to compute it

29

A specular term: the «Blinn-Phong» equation


RESULT EXAMPLE

specular exponent

dot product but zero if negative!

component-wise product

surface normal

"half-way" versor:

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

light direction (towards the light)

view direction (towards the viewer)

light-color / intensity

specular-color

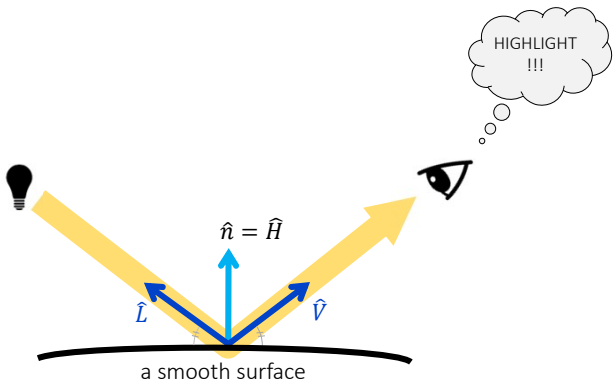
material parameter

light parameter

geometry

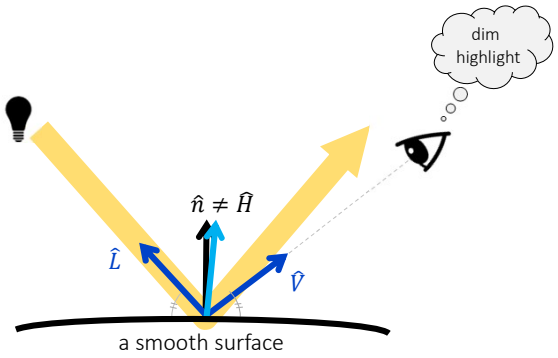
30

The intuition behind the specular-term formula



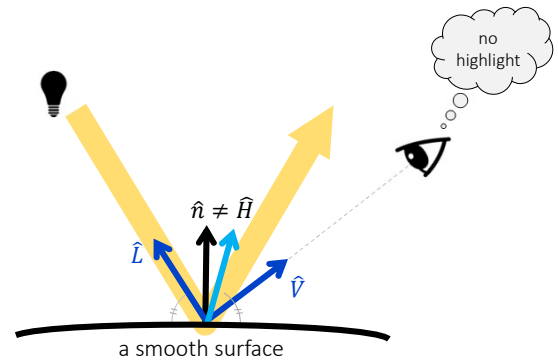
31

The intuition behind the specular-term formula



32

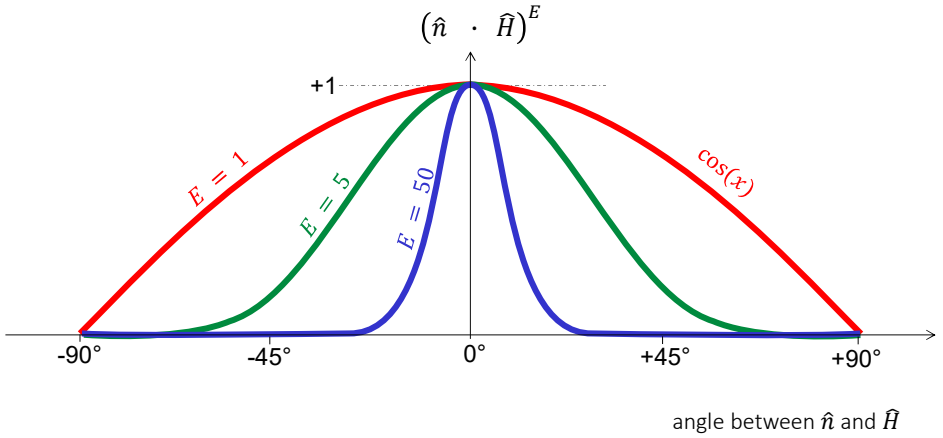
The intuition behind the specular-term formula



33

Specular exponent E

- Elevating by $E > 1$,
  highlights become smaller / more concentrated



34

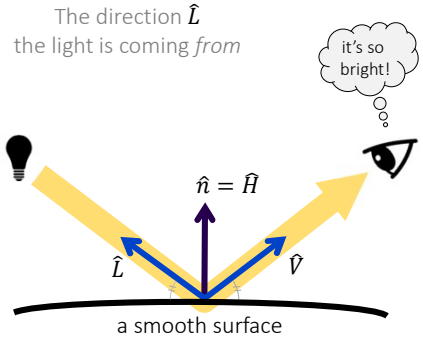
The intuition behind the Blinn-Phong formula

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

- 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

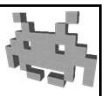
The dot product between versors is a measure (max = 1) of their similarity

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

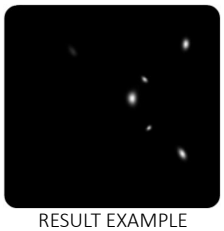


35

The Blinn-Phong equation



- info:
  - it's view-dependent (it uses V)
  - not based on real physics
  - no real material behaves like that
  - not even energy conserving!
  - intuitive, but just a made-up formula
  - yet, it simulates reflections ("highlights")

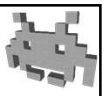


RESULT EXAMPLE

- introduces new material parameters:
  - specular color (rgb color)
    determines the intensity and color of the highlight
    sometimes: it's just diffuse color x a constant
    often > 1 – oversaturated highlights
  - specular exponent , aka "glossiness" (a scalar, in, e.g. 1 to 128)
    determines the SIZE of the highlights
    larger numbers → smaller highlights

36

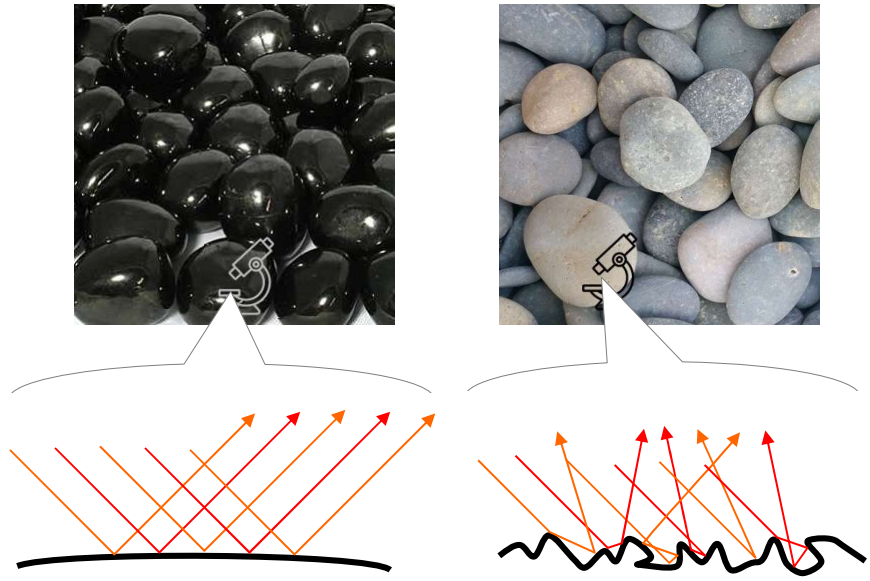
Which physical characteristics determine how "shiny" is a material, in the real world?



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

37

Materials and micro-details



38

Observation: both the mesh and the material model the 3D shape of the surface. But at different scales!

mesh

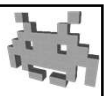
- **macro**-structure
  - the general shape of the surface
  - modelled explicitly

material

- **micro**-structure
  - the super tiny geometric details
  - modelled stochastically

39

Diffuse term + specular term: a practical problem



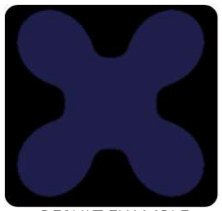
- Using just these two "direct" terms, we have a practical problems: pixels not reached by any light are BLACK (r,g,b = 0,0,0)
  - regardless of material
  - e.g., parts that are in shadow

Light is "blocked"

- This reflects a simplification of our lighting environment
  - in reality, some amount of light (which we don't model) reaches basically every surface (in most scenarios)
- Easy and brutal fix: add a new "ambient" term to the lighting equation
  - that is, we add a bit of light to everything
  - an additional term in the lighting equation

41

Ambient term



RESULT EXAMPLE

component-wise product

$$\begin{pmatrix} a_R \\ a_G \\ a_B \end{pmatrix} \otimes \begin{pmatrix} A_R \\ A_G \\ A_B \end{pmatrix}$$

ambient light color or intensity

ambient-color

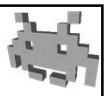
material parameter

light parameter

geometry

42

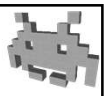
The intuition behind the ambient-term formula



- A bit of light will reach this point of the surface from *all* directions
  - so, surface normal does not count!
- In first approximation, the amount of light is the product of
  how much light is around overall
  and
  the ability of the material to reflect this light

43

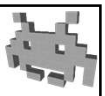
A basic lighting equation: ambient term: info



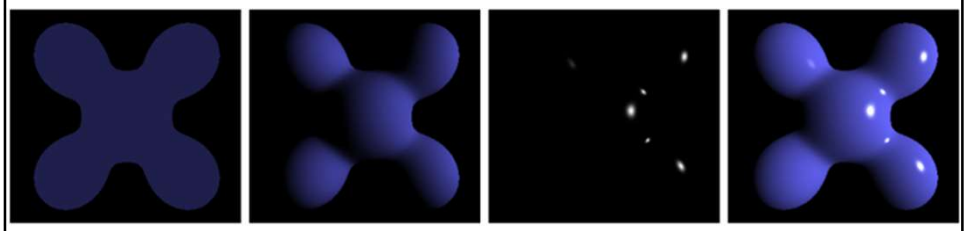
- based on the assumption:
  "a bit of light reaches the object from every direction"
  - e.g., from light bounces,
  - e.g. from minor, unmodelled light sources
- without it, things not directly lit by light(s) are black
  - and that looks ugly -- avoid
  - plus, it's very simple to include it in the equation, so why not
- used parameters in the material:
  - ambient-color (RGB)
  - variant: ambient-factor (a scalar),
    then ambient-color = diffuse-color · ambient-factor
  - or we can just reuse the diffuse-color

44

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



- It's the sum of 3 terms:

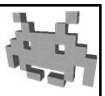


ambient + diffuse (or "Labertian") + specular (or "Phong") = final

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

46

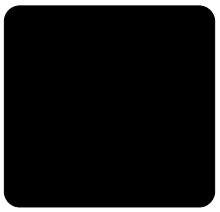
A basic lighting equation: emission term



- In formulas:

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

emission-color



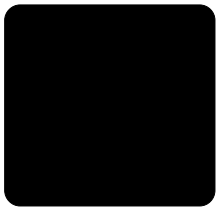
most objects have no emission term

material parameter
light parameter
geometry

47

A basic lighting equation: emission term

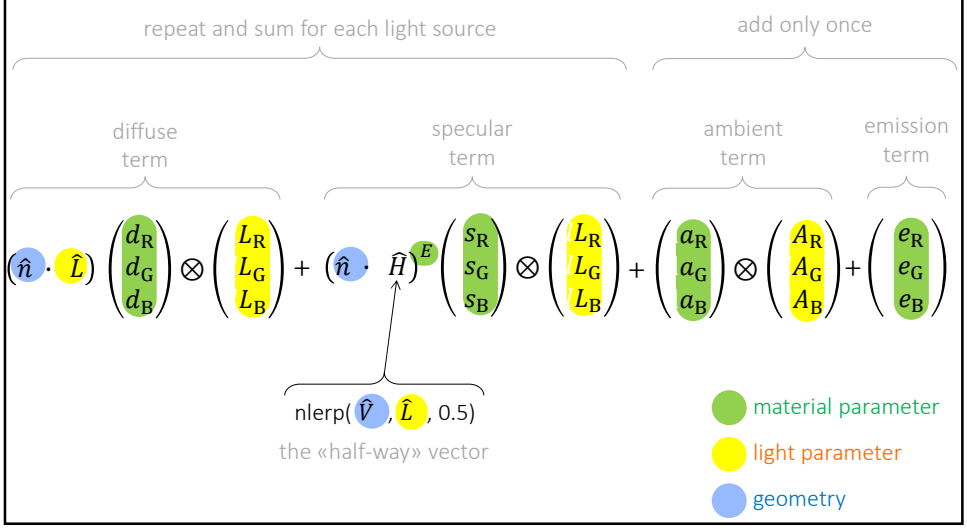
- it models any light that…
  - …is emitted from an object, and
  - …reaches the camera (directly!)
- useful for, e.g., small led lights, that are visible in the dark
  - for any other object (i.e, most of them), it is zero
- note: the emitted light doesn't illuminate other objects
  - for this, you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



zero, in this case

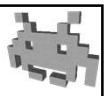
48

Phong lighting equation: complete



50

The material Model of the Phong lighting equation



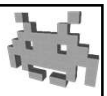
- The historical "OpenGL material"
- This Lighting Equation is defined as the sum of 4 terms:
  - "Ambient" + "Diffuse" + "Specular" + "Emission"
- The material is… a color multiplier for each term, therefore:
  - "Ambient" color
  - "Diffuse" color (aka "Base" color, aka "Albedo")
  - "Specular" color (aka "Highlight" color)
  - "Specular Exponent", aka "glossiness" or "shininess" (a scalar ≥ 1 and <128)
  - "Emission" color (only for stuff *emitting* light – otherwise 0,0,0)

distinct multipliers for R, G and B

technically, only if it's gray-scale

51

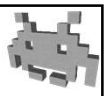
Phong lighting equation: remarks



- Lighting is *additive*: the Diffuse and Specular components…
  - must be added for each light in the scene (for discrete lights – see lecture on rendering)
  - must be integrated on the light environment (for continuous light environment – see lecture on rendering)
  - the Ambient Light and Emission components are added once
- The Specular factor is nowadays too crude for the quality expected from modern games
  - The other factors are still used, as they are realistic
  - See later for the improvements
- The geometric vector used can be expressed in any space
  - but it must be the same space! (e.g. global space or local space)

52

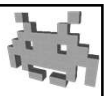
Blocked lights (that is, shadows)



- In this lecture, we are considering "local" lighting models, where we assume that the rest of the 3D scene doesn't count
  - Only (1) the bit of surface being lit, (2) light sources, and (3) the observer, count
- In reality, the lighting equation can account for light "blockers"
  - One example of a "global" (non-local) lighting effect
- The two "direct" terms (diffuse & specular) are to be negated (for each given light source) when a "blocker" prevents that light from reaching the surface
  - That is, the light is shadowed by the light blocker
  - In all or in part: the two terms are multiplied by a "shadow factor" (in 0 to 1)
  - See Lecture on rendering (or courses on *Real Time Graphics Programming*) for rendering algorithms to compute this factor
- The ambient term is to be negated when the point on the surface is surrounded by many blockers
  - the term is multiplied by an "ambient occlusion" factor (in 0 to 1)

53

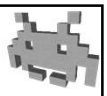
Defining materials: Chapter 1 (until the '90s)



- The "Phong" lighting equation has been the *standard in games* for many years, because
  - It's cheap to compute
  - Lighting equation was hardwired in graphics API (OpenGL and DirectX), making this was the only supported material model
- 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
  - Can only be considered accurate when the Specular component is zero

56

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



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

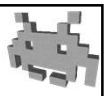
- Diffuse color (aka Base color, aka Albedo):
  - "Which color is this stuff?"
  - i.e., "Which color does it look like, if I shine a white light on it?"
- Specular color (aka Highlight color)
  - "Which color and how bright are its reflections?"
  - if it's a factor (and Specular-color = Base-color · Specular-factor): "How bright are its reflection?"
- Specular exponent (aka Glossiness) (e.g. in 1 to 128)
  - "How concentrated are its reflections?"
  - Larger value (e.g., 200) ==> more concentrated highlights
  - Smaller value (e.g., 4) ==> larger highlights
- 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

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

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

57

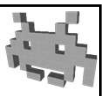
Problems with the basic (aka Phong) material model



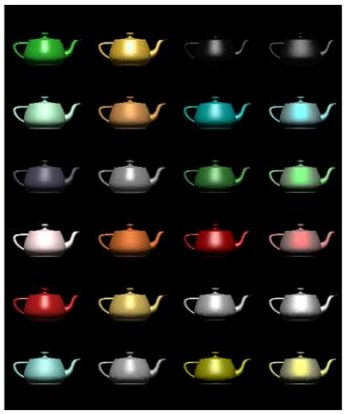
- It's not very expressive
- It's made-up (especially the specular component)
- It isn't realistic

59

Defining materials: Chapter 1 (until the '90s)



Material	Ambient Color	Diffuse Color	Specular Color	Specular Exponent
Silver	0.19225	0.50754	0.508273	51.2
	0.19225	0.50754	0.508273	
	0.19225	0.50754	0.508273	
Polished Silver	0.23125	0.2775	0.773911	89.6
	0.23125	0.2775	0.773911	
	0.23125	0.2775	0.773911	
Emerald	0.0215	0.07568	0.633	76.8
	0.1745	0.61424	0.727811	
	0.0215	0.07568	0.633	
Jade	0.135	0.54	0.316228	12.8
	0.2225	0.89	0.316228	
	0.1575	0.63	0.316228	
Obsidian	0.05375	0.18275	0.332741	38.4
	0.05	0.17	0.328634	
	0.06625	0.22525	0.346435	
Pearl	0.25	1	0.296648	11.264
	0.20725	0.829	0.296648	
	0.20725	0.829	0.296648	
Ruby	0.1745	0.61424	0.727811	76.8
	0.01175	0.04136	0.626959	
	0.01175	0.04136	0.626959	
Turquoise	0.1	0.396	0.297254	12.8
	0.18725	0.74151	0.30829	
	0.1745	0.69102	0.306678	
Black Plastic	0	0.01	0.5	32
	0	0.01	0.5	
	0	0.01	0.5	
Black Rubber	0.02	0.01	0.4	10
	0.02	0.01	0.4	
	0.02	0.01	0.4	



not very expressive ☹

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

60

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
  - new terms are added or replace simpler ones
- It supports 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 needs to look good under widely different virtual lighting conditions

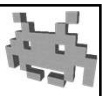
the task of the "material artist"

61

Much wider expressiveness is needed

62

Material models are improving



63

Material models are improving

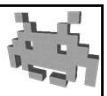
64

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

- Physically Based Materials (PBM) model
  - 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

65

«Physically Based Lighting» (PBL)

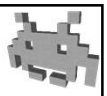


Defined as…

- A lighting model more closely inspired by physics
  - For example, not infringing energy conservation (unlike Phong)
  - With fewer tricks that just "follow an intuition" (unlike Phong)
- A lighting model accepting, as input, a PBM
- Also, a lighting model taking fewer shortcuts than otherwise typical
  - For example, store
    - ambient color: one value (e.g., in its own "texture", see later)
    - ambient occlusion (AO): *a separate value* (idem)
  - Instead of storing:
    - ambient color × AO : one value (cheaper!)
- Warning: PBM & PBL are, at some level, buzzwords

66

PBM and PBL: objectives



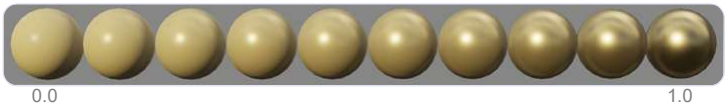
conflicting ones!
(to some extent)

- Make realistic-looking materials easier to design
  - by material artists
- Make it easier to capture materials from Real World samples
  - because the material description is closer to reality
- Make it easier to reach a reasonably realistic-looking lighting under a wider range of virtual lighting conditions
  - more approximated lighting equations used to require *ad hoc* tuning for different light conditions, e.g. dark cave VS sunny day
- Standardize Materials
  - make it easier to share a material description across different games
  - note: it helps to prescribe each parameter to be in a standard 0 to 1 range
- Unify: accurately describe more materials (useful ones) with one model
  - one lighting equation with different parameters for all materials, instead of …
  - … different lighting equations for different classes of materials
  - not need to switch equation -> more efficient rendering, see e.g. deferred shading (later)
  - (this is not possible 100% for wide enough class of materials)
- Fewer parameters…
  - eases storage, authoring, editing, capture
- …but the right ones - that is, represent a good "space of materials"
  - ideally: no combination of parameters should look wrong
  - unlike, e.g., high spec. color & low specular exp => terrible, with Phong

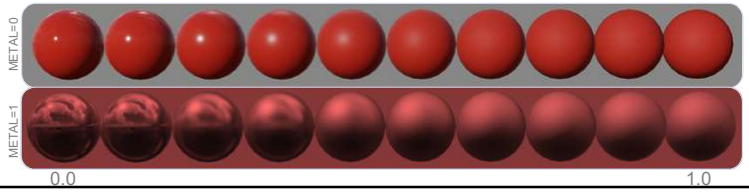
67

Physically Based Materials (PBM): a good choice of parameters (example)

- Base color (rgb – or "diffuse", same as old school)
- Specularity (scalar – or rgb sometimes)
- "Metallicity" (a bool in theory, but often a scalar in 0 to 1)



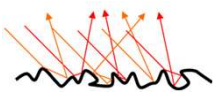
- Roughness (scalar)



images: unreal engine 4

68

Parameters of a Physically Based Material (PBM)

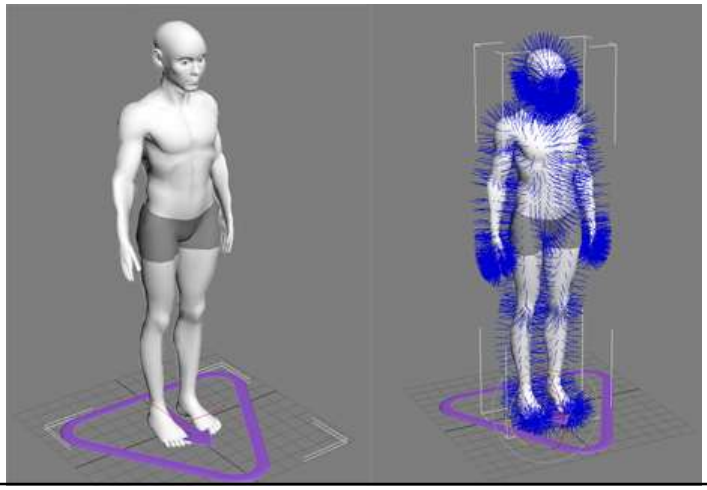
- Base color: (a RGB color)
  - Same as in basic lighting model
- Specularity: (a scalar, in 0 to 1)
  - Total amount of light bouncing off the surface with reflections (regardless of how). What is not reflected, is just absorbed!
  - Barring exception, it is usually high (closer to 1 than to 0): "Everything 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 = 

similar trick to "bounciness" for collisions!

69

The geometry in lighting: normals…

- Per-vertex attribute of meshes, and/or per texel (normal maps)

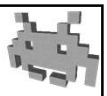


71

… and tangent directions too, used for *anisotropic* materials

72

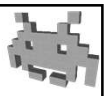
View-(in)dependent materials. (An)isotropic materials.



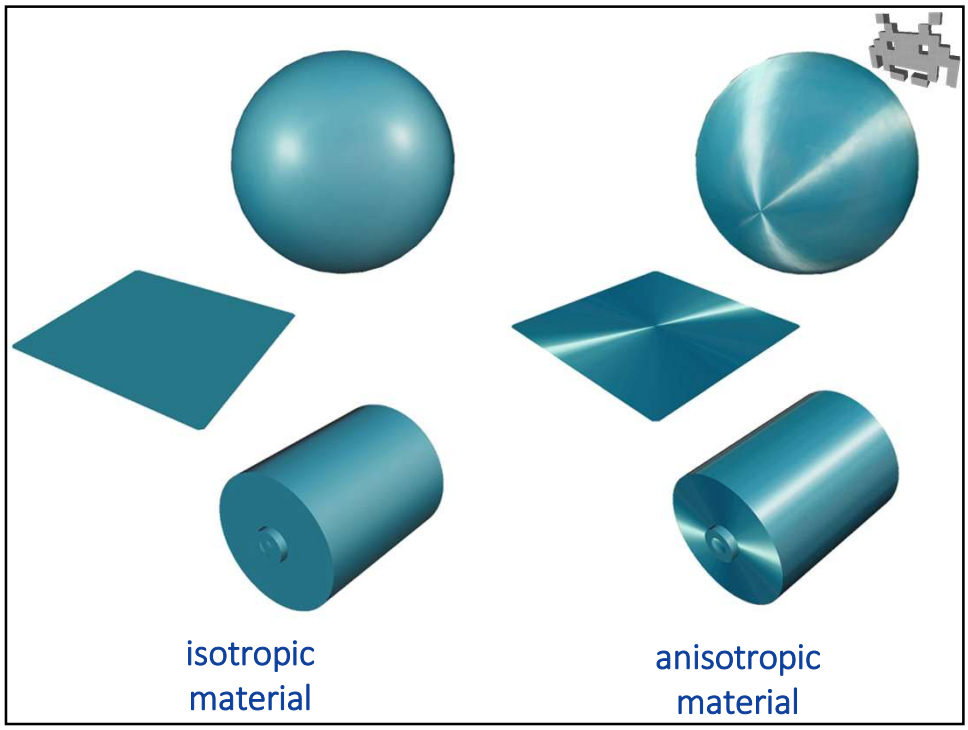
- A view-dependent lighting equation (or material) is one which uses the view direction $\hat{v}$
  - Q: which terms of the lighting equations seen above are "view-dependent"?
  - Otherwise, it's view-independent
- An anisotropic lighting equation (or material) is one which uses the tangent directions
  - For real-world materials such as where the microstructures have features aligned to one direction:
  - E.g. satin, velvet, woven fabric, scratched metal (if scratches in one preferred direction)
  - Otherwise, it's isotropic

74

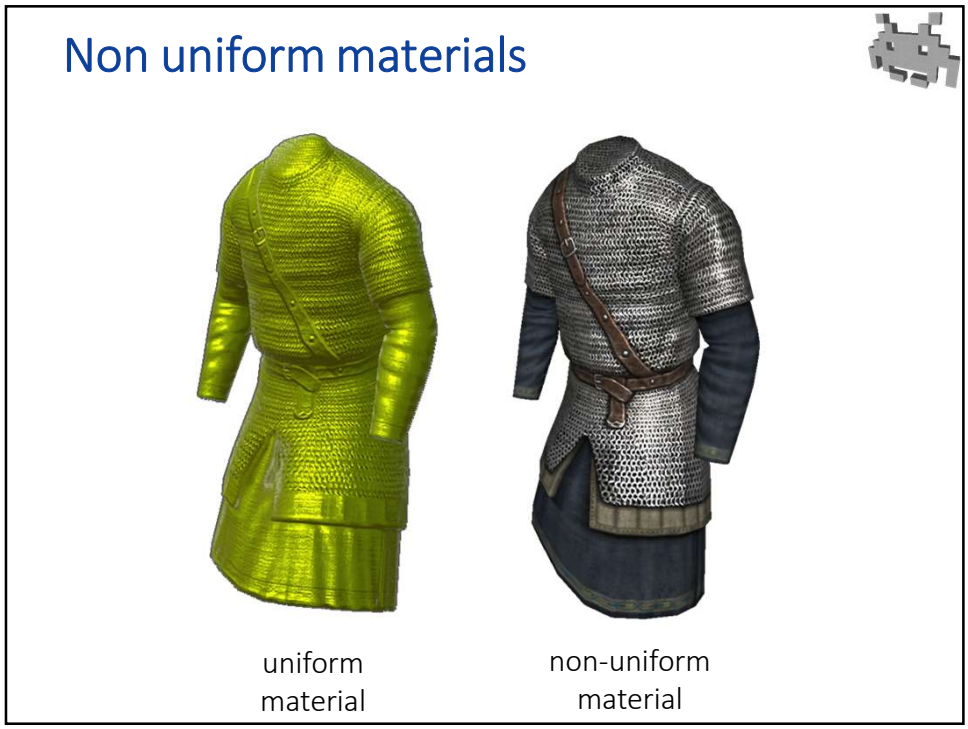
Materials / lighting models



75

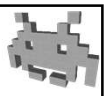

isotropic
material
anisotropic
material

76


Non uniform materials
uniform
material
non-uniform
material

77

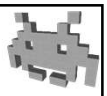
Uniform VS non-uniform materials



- So far, we assumed that the material is uniform
  - Material parameters defined once for the entire mesh
- Most objects in a videogame have non-uniform materials
  - aka a “spatially varying” material
  - the material parameters vary on the surface!
  - each point on the surface = different parameters
  - for example: different (basic) color
    (but also: different metallicity, etc)
- Note: some parameters can be uniform
  and others can be non-uniform
  - For example: constant base color but varying metallicity
    (because a few areas… rusted!)

78

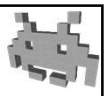
⚠ terminology: «material» has 2 different meanings



- in Computer Graphics jargon: the material *model*
  - a set of parameters describing the behavior of a physical substance to light
  - (part of) the input of the lighting equation
  - e.g.: “rough plastic” or “polished wood”
  - can be measured from samples, authored by artists, etc.
  - what we discussed so far
- in video-game jargon: the material *asset*
  - a data structure fully describing a non-uniform material
  - to be applied on one (or more) mesh
  - sometimes, tailored for one mesh

79

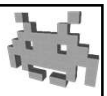
Material as an asset



- A compound data structure that describes all material parameters to be used on one or more mesh
- For uniform material parameters:
  - their values, directly (just constants)
- For non-uniform material parameters:
  - Are they stored as attributes?
    that is, they are to be found in the vertices of the mesh?
  - Or, are they Stored in a texture – see next lecture
    that is, they are to be found in which channel of a texture image?
  - Or, intermediate cases, e.g. the two above, multiplied together?
  - Or, should they be computed on the fly (procedurally)?
    If so, using which formula?
    (example: a formula to produce the base color from the pos)
    (for example:
    "player 1 knights have blue diffuse color in these area,
    player 2 knights have red diffuse color in the same areas –
    the player number will be specified for each model)

80

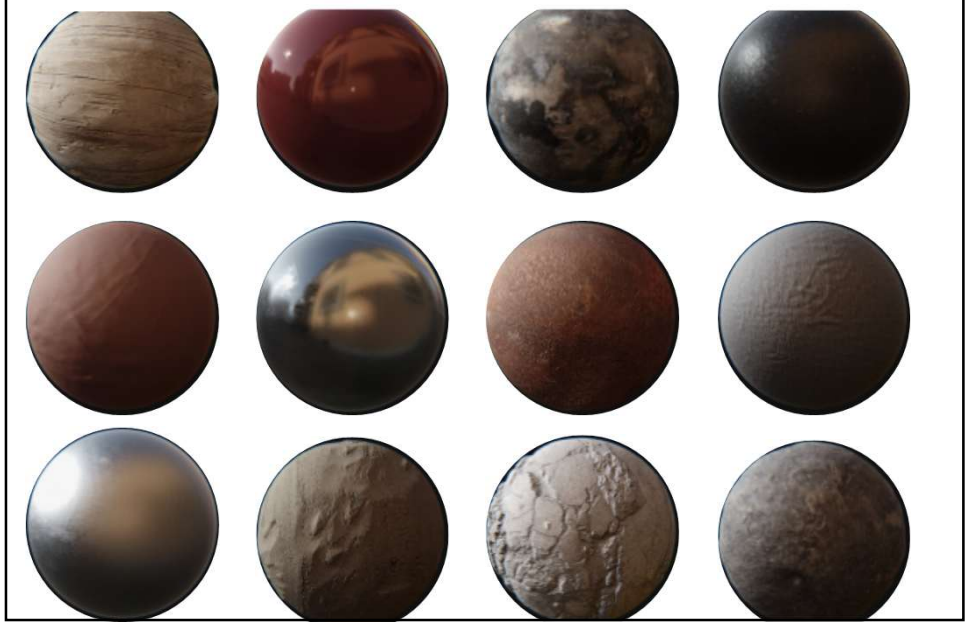
The *material asset* can also include…
(depending on the game engine)



- Which lighting model should be used
  - if the engine supports multiple ones
- Or even, a custom lighting model
  - in the form of… code to computes it
  - this program (aka the "shader") is formatted for GPU execution
- (references to) all needed textures
  - Describing how the materials parameters vary over the surface
  - See next lecture!
- Instructions about how the rendering engine should process the object, including…
  - Rendering flags, such as…
    is "back-face culling" ON or OFF?
    is "alpha testing" ON or OFF?
    Is "depth test" ON or OFF?
    } See the Computer Graphics course
  - Order of ordering?
    (does this object need to be drawn before / after the others, maybe?)

81

Material assets



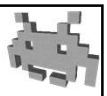
83

The *material asset* (high level view)

- From the perspective of the material artists (who author it), a material asset is a package of data (shaders, textures, parameters values, flags) describing (an arrangement of) materials on a surface
  - To be used on one or more meshes
  - Sometimes tailored for one specific mesh (and produced as part of the asset construction pipeline for that mesh)
  - Sometimes general and to be shared by multiple meshes (and produced independently from any mesh)
  - Try to download and look at a few ones! (of the latter kind)
- From the perspective of the game engine (that uses it) the material asset fully describes the state of the rendering engine needed when the draw-call is issued
  - Uniform material parameters to be sent to the GPU
  - Textures to be loaded in VRAM
  - Shaders (implementing a lighting equation) to be activated
  - Flags to be set (to control the hard-wired mechanisms of the GPU), etc

85

Material Asset = status of renderer



To render a mesh…

- Load…
  - make sure all data is ready in V-RAM
    - Geometry + Attributes
    - Connectivity
    - Textures
    - Shaders
    - Material Global Params
    - Rendering Settings

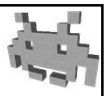
THE MESH ASSET

THE MATERIAL **ASSET**

- …and Fire!
  - issue the Draw Call

86

Material Assets: examples of interchange formats



dino.mlt

ambient color
diffuse color (multiplier)
specular color
specular exponent
diffuse map ("diffuse color is defined in this texture")

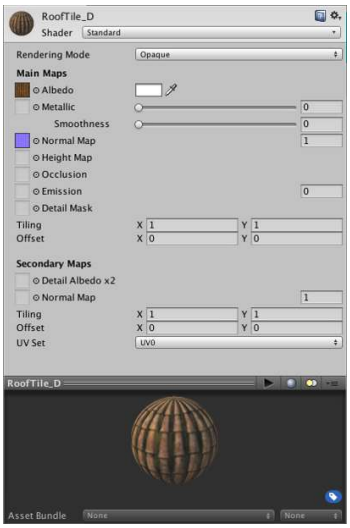
```
newmtl material_0
Ka 0.1 0.1 0.1
Kd 0.5 0.3 0.3
Ks 1 1 1
Ns 255
map_Kd dino_0.jpg

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)

87

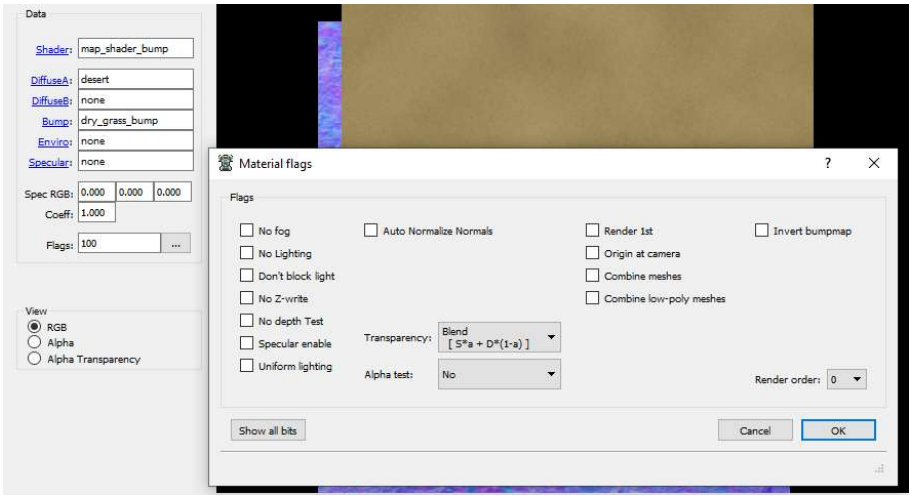
Material Assets: examples of GUI to view/edit them



GUI to edit assets of type “material” in Unity

88

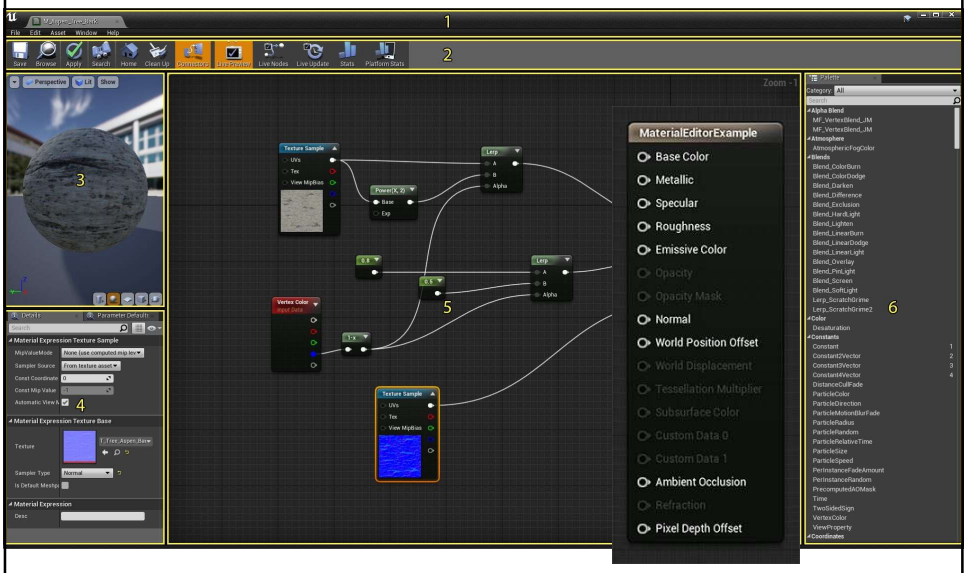
Material Assets: examples of GUI to view/edit them



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

89

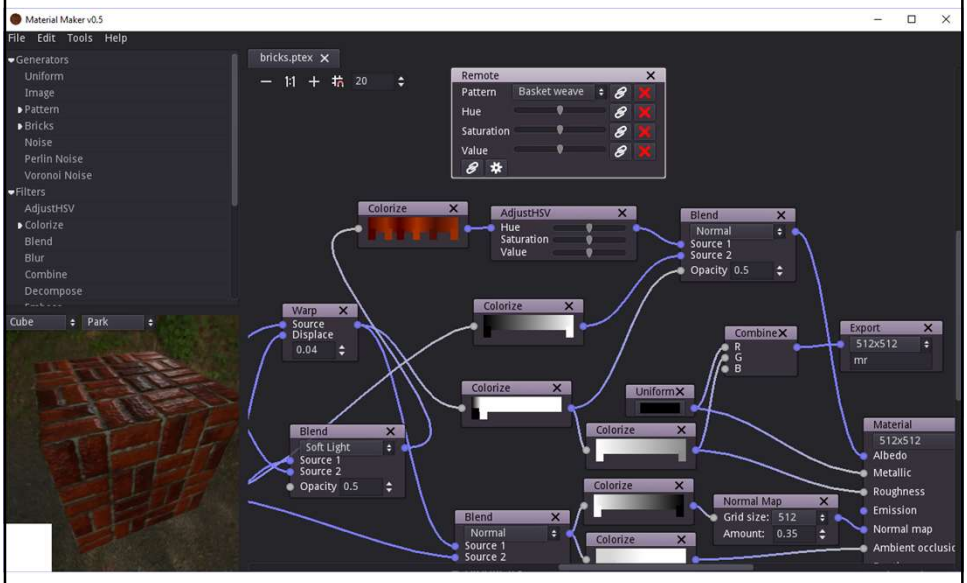
Material Assets: examples of GUI to view/edit them



GUI to edit assets of type material in Unreal Engine

90

Material Assets: examples of GUI to view/edit them



Material Maker: a (stand alone) tool to edit assets of type material for Godot

91