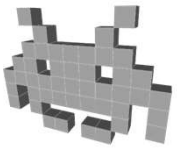
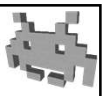
3D Videogames 2022/2023
Univ. degli Studi di Milano



Materials in videogames

1

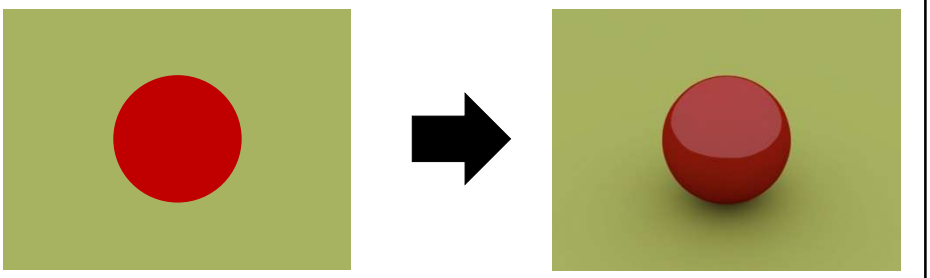
Course Plan



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 **Textures**
lec. 9: Game **Materials**
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

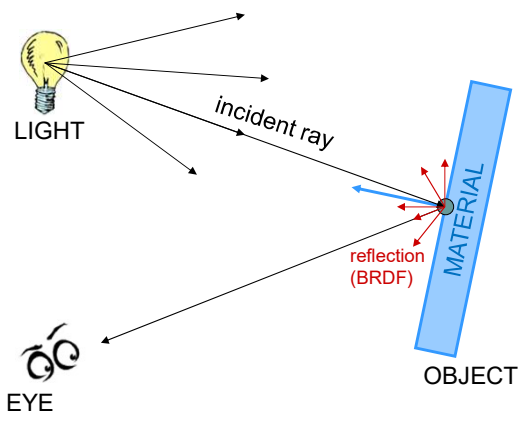
2

Lighting



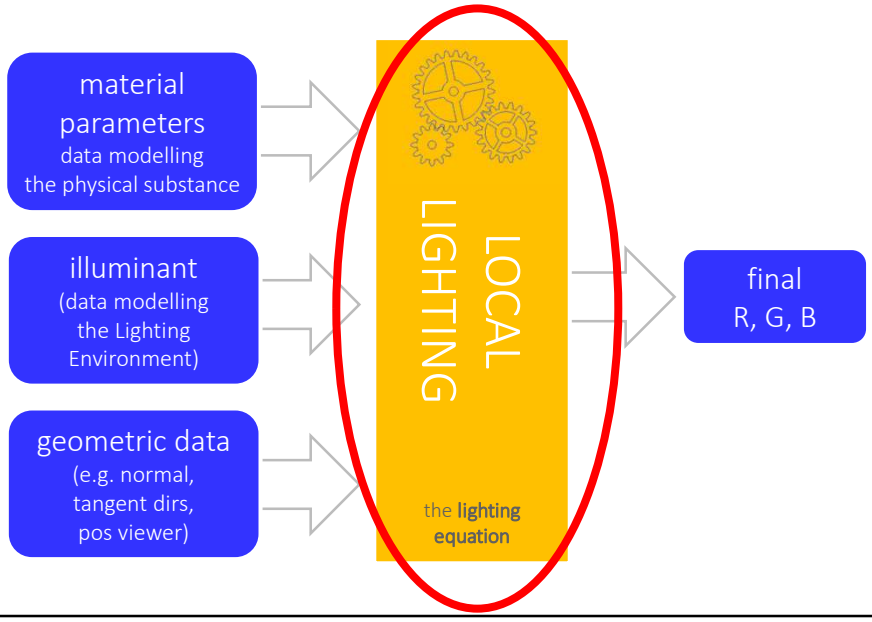
3

(Local) Lighting



4

Local lighting



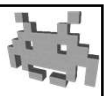
5

Lighting computation in videogames

- Lighting:
  - (a task of the rendering engine)
  - computation of the "final" 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 lighting model
  - *Output:* the RGB color, ready to be sent to the screen
  - We can divide its many *inputs* in three categories…
  - 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 is computed per-pixel, per frame!)

6

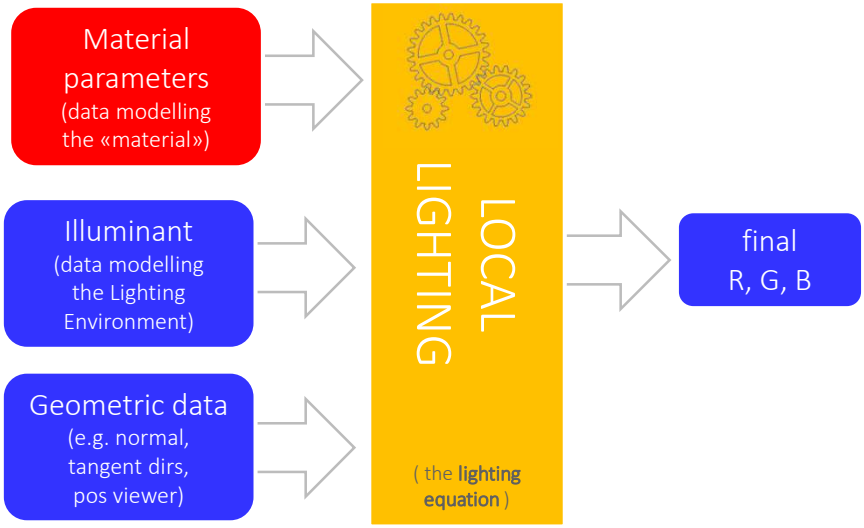
Choosing the lighting equation



- Different models can be employed…
  - Phong — the basic model, historically used in 3D games for decades
  - Beckmann
  - Heidrich–Seidel
  - Cook–Torrance
  - Ward (anisotropic)
  - …
  - + additional Fresnel effect
- with a varying levels of
  - computational complexity
  - realism
  - material parameters required
  - richness of simulated effects

7

Material model



8

Material models

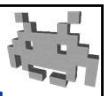
9

⚠ 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
  - a part of the 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

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

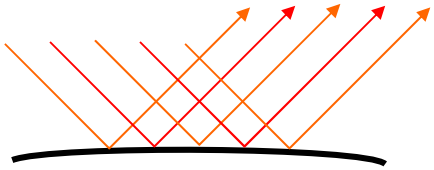
11

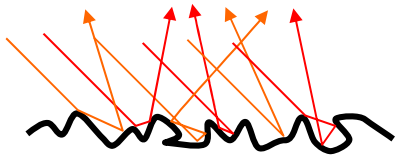
Material Model: which physical characteristics does it represent?

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, etc
   - 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

12

Material model and nano-shape

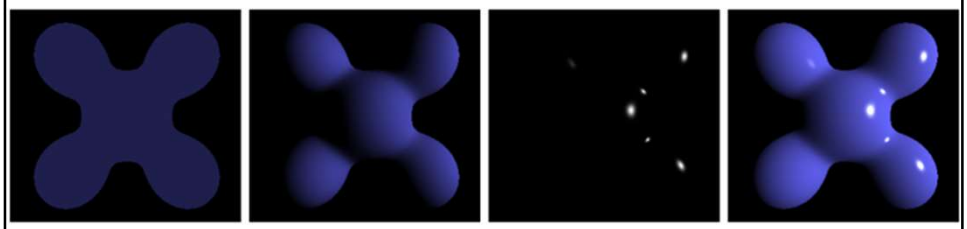




13

A basic lighting equation
(referred to as basic or «Phong» lighting equation)

- It's the sum of 3 terms:

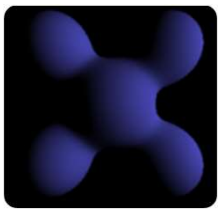


ambient + diffuse (or "Labertian") + specular (or "Phong") = final

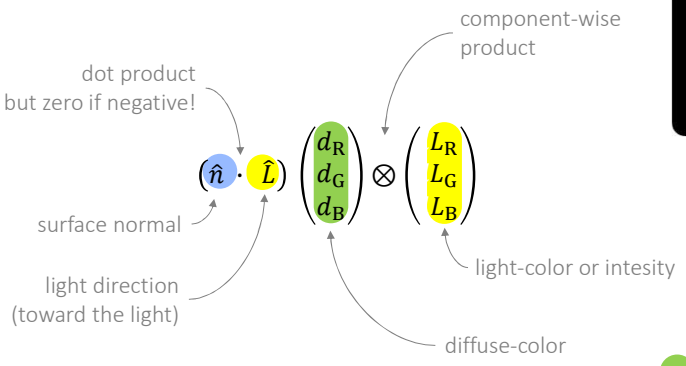
plus a constant additional term ("emission"), only for objects emitting light

14

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



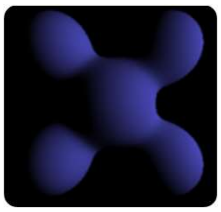
- In formulas:



See CG course!

15

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

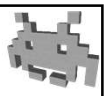


- info:
  - it's physically based
  - exhibited by dull materials (e.g., plasters, untreated wood)
  - used in any lighting equation
- material parameters used:
  - base color, (albedo, when grayscale), aka diffuse color, Lambertian color, or sometimes just color
  - with non uniform materials, the texture 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

16

The intuition behind the diffuse-term formula



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

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

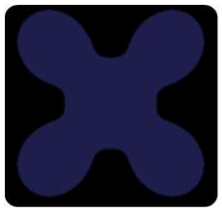
The direction $\hat{L}$ the light is coming *from*

- 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

17

A basic lighting equation: ambient term

- In formulas:



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 intesity

ambient-color

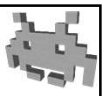
material parameter

light parameter

geometry

18

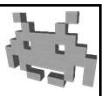
The simplest choice for lighting-equation (& Material-model) :



- Phong equation aka "OpenGL material"
- This Lighting Equation (aka phong model) = Σ 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)

19

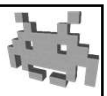
A basic lighting equation: ambient term: info



- based on the assumption: "a bit of light reaches the object from every direction"
  - e.g., from light bounces,
  - e.g. from unmodelled light sources
- without it, things not directly lit by lights are black (and it looks ugly)
  - it's very simple to compute, 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)
- as all parameters 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)

20

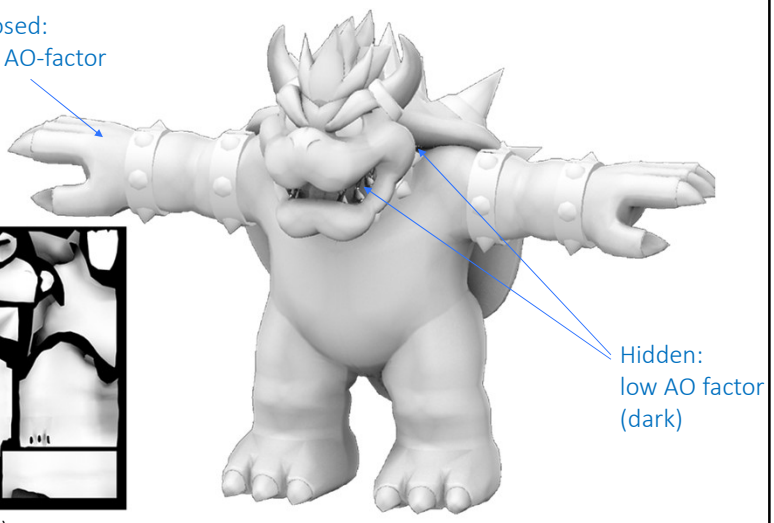
The intuition behind the ambient-term formula



- 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)

21

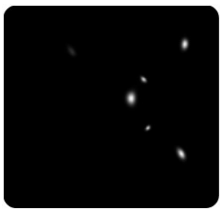
Baking AO map for a model



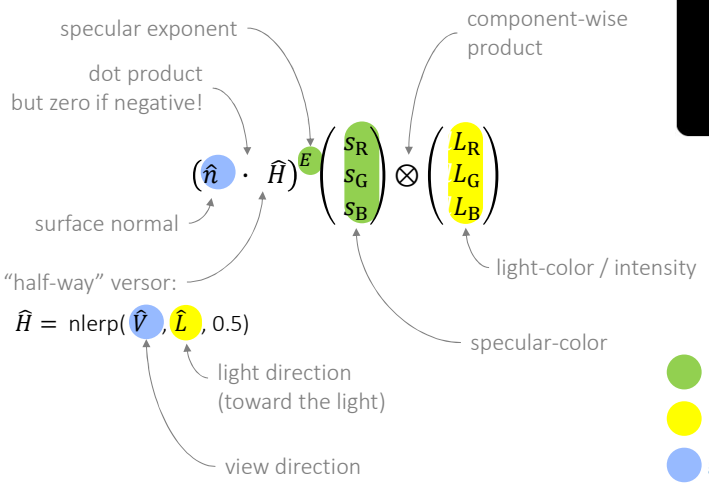
AO map (baked)
NOTE: this requires UV unwrapping (injective UV-map), like any baked texture

22

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

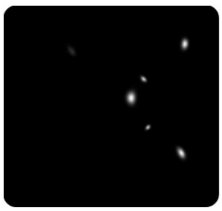


- In formulas:



23

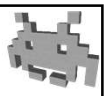
A basic lighting equation: specular term (aka «Blinn-Phong» term)



- 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
- textures:
  - specular map and glossiness map
  - e.g., a 4-channel texture can store: RGB spec color + glossiness

24

The intuition behind the specular-term formula

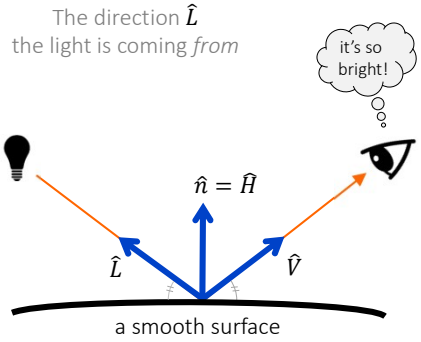


The local orientation of the surface $\hat{n}$

- 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 in 0 to 1 of their similarity

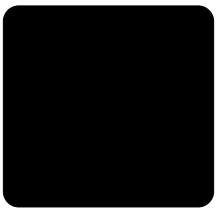
The direction $\hat{L}$ the light is coming *from*



25

A basic lighting equation: emission term

- In formulas:



most objects have no emission term

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

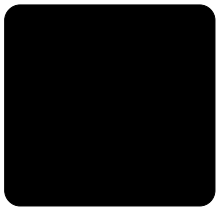
emission-color

material parameter
light parameter
geometry

26

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

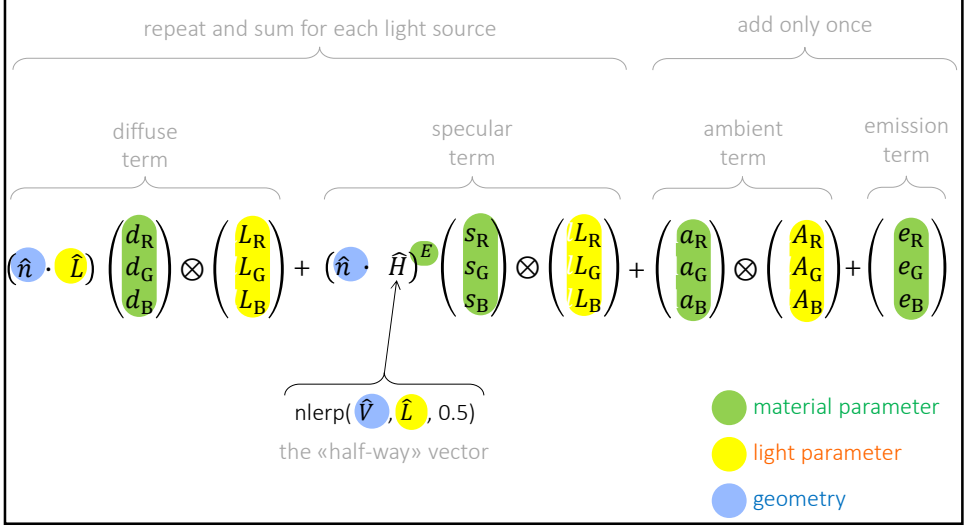


zero, in this case

- 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, $e_{\mathrm{R,G,B}} > 1$) to get a "glow" effect (see HDR, last lecture)

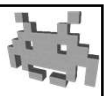
27

A basic lighting equation: in total



28

Lighting equation: remarks



- Lighting is *additive*: the Diffuse and Specular components…
  - must be added for each light in the scene (for discrete lights – see later)
  - must be integrated on the light environment (for continuous light environment – see later)
  - 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! (see later)

29

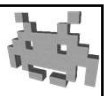
Applying the Lighting equation

- It's computed for each pixel (in the fragment shader, see lecture on rendering)
  - most game engines support a good set of choices of different lighting equations
  - custom new equations can be programmed in fragment shaders
- Material + geometry parameters can be stored …
  - …in textures *(for highest-frequency variations inside 1 obj)*
  - …in vertex attributes *(smooth variations inside 1 obj)*
  - …as material asset parameters *(no variation for 1 obj)*
  - For example, where are
    - diffuse color
    - specular color
    - normals
    - tangent dirs

  typically stored?

30

In which space to express the data for lighting?



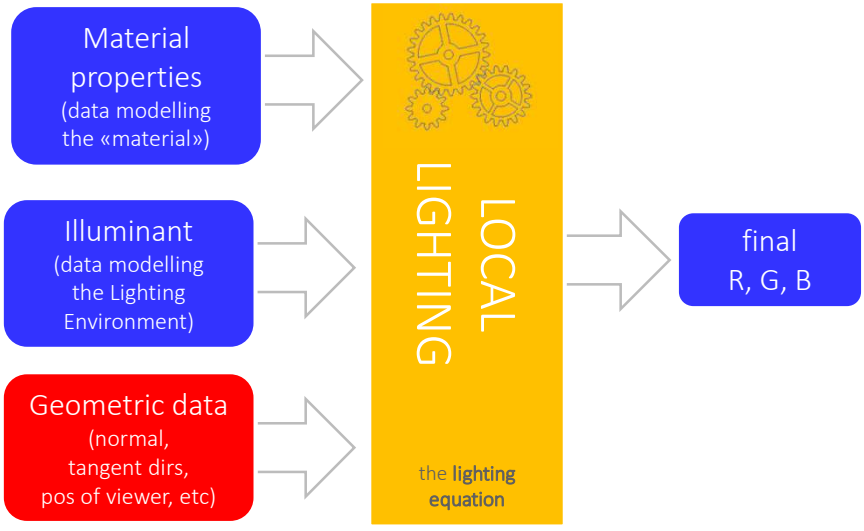
- All versors that used in operations of the lighting equation must be expressed in the same space
  - view direction, light directions, half-way vector, normals, tangent dirs…
- Choice: which space to use?
  - View space? (the space of the camera)
    (it's easy to express the view direction, and headlights)
  - World space?
  - Object (local) space? (the space of the object currently being rendered)

for anisotropic materials

- With normal maps, the most efficient solution is:
  - Use the same space the normals are expressed, in the texture
    For Tangent Space normal maps: the TBN space
  - All other versors must be transformed into this space… *per vertex!*
  - The normals accessed from the texture can be used right away… *per pixel!*
  - This minimizes the number of transformations needed

31

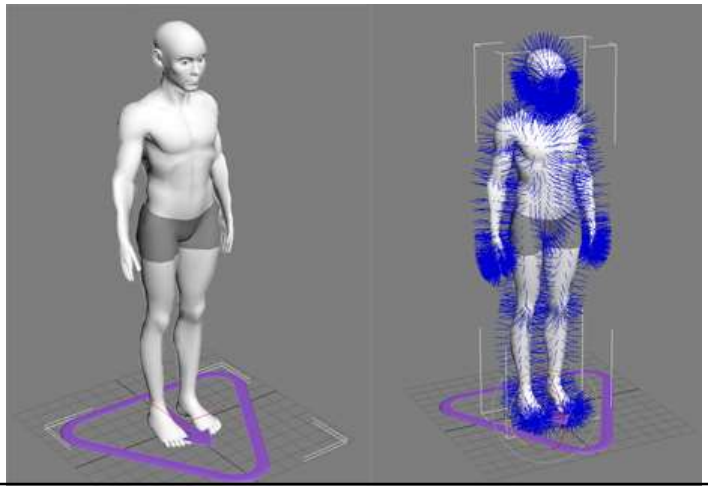
Geometric Data



32

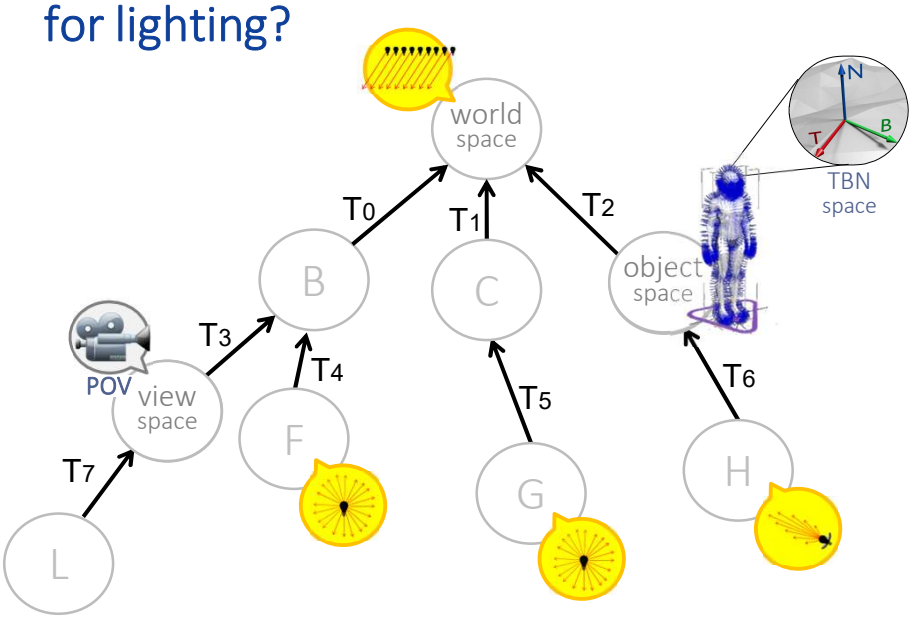
The geometry in lighting: normals

- Per-vertex attribute of meshes, and/or stored in normal maps



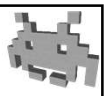
33

In which space to express the data for lighting?



36

Uniform and non-uniform materials



- Uniform materials:
  the parameters are shared by the entire mesh
  - 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 mesh
  - They can be stored as vertex attributes *(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 and other parameters
  - for example, where are (1) diffuse color, (2) specular color, (3) normal (3) tangent dirs typically stored?

37

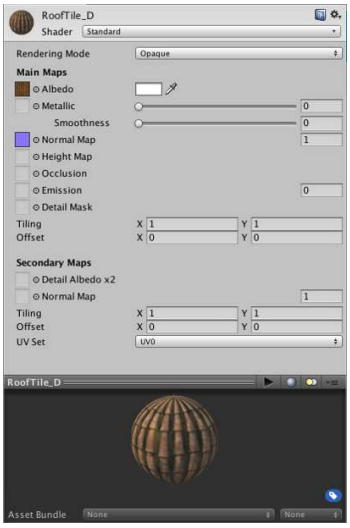
Material *Asset*

A data structure describing a material model, and also:

- How are the parameters stored, for example…
  - …in the texels of *this* texture?, or,
  - …as *this* global value (material is "uniform")?, or,
  - …as whichever attributes are stored in the rendered mesh?
- It includes bump-maps (any kind, e.g., normal-maps):
  - which technically, describe the object *shape*, not its *material*
- The algorithm used to compute the lighitng, that is
  - a set of "shader" program that gather the parameters, and compute the lighting equation (see lecture on rendering)
  - flags and settings for the GPU rendering (e.g. back-face culling, depth test)
  - which rendering passes must be done

38

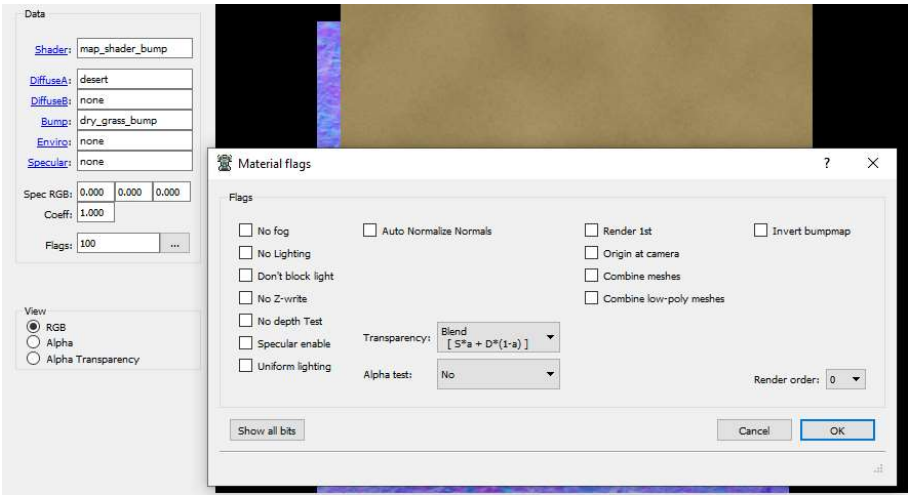
Material Assets (examples)



Panel for assets of type material in Unity

39

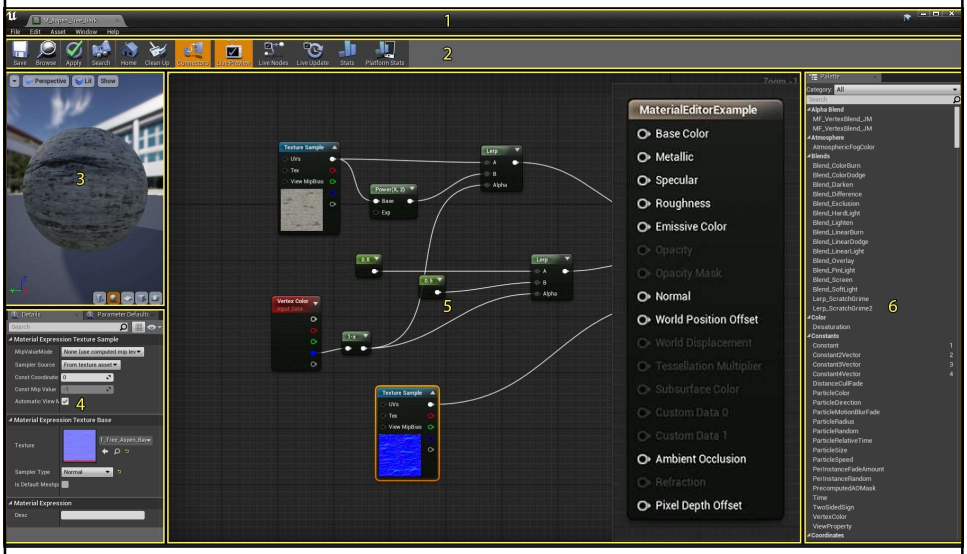
Material Assets (examples)



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

40

Material Assets (examples)



panel for editing of assets of type material in Unity

41

Material Assets (examples)

dino.mlt

```
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
```

ambient color

diffuse color (multiplier)

specular color

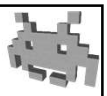
specular exponent

diffuse map ("diffuse color is defined in this texture")

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

42

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

THE MESH ASSET

THE MATERIAL **ASSET**

- …and Fire!
  - issue the Draw Call

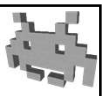
43

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

45

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



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

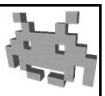
- Diffuse color (aka Base color, aka Albedo):
  - "Which color is this stuff?"
  - i.e., "Which color does it look like, if I shine a white light on it?"
- Specular color (aka Highlight color)
  - "Which color and how bright are its reflections?"
  - if it's a factor (and Specular-color = Base-color · 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
- 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

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

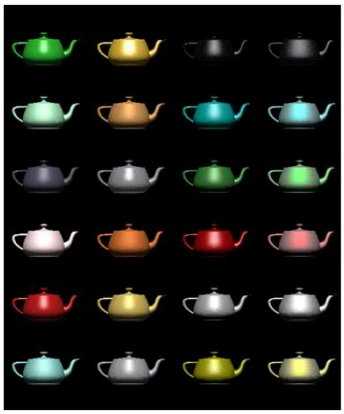
A way to remedy the fact that we are not modelling a realistic light environment

46

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

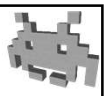


not very expressive ☹

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

48

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

49

Defining materials: Chapter 2 ('00s)

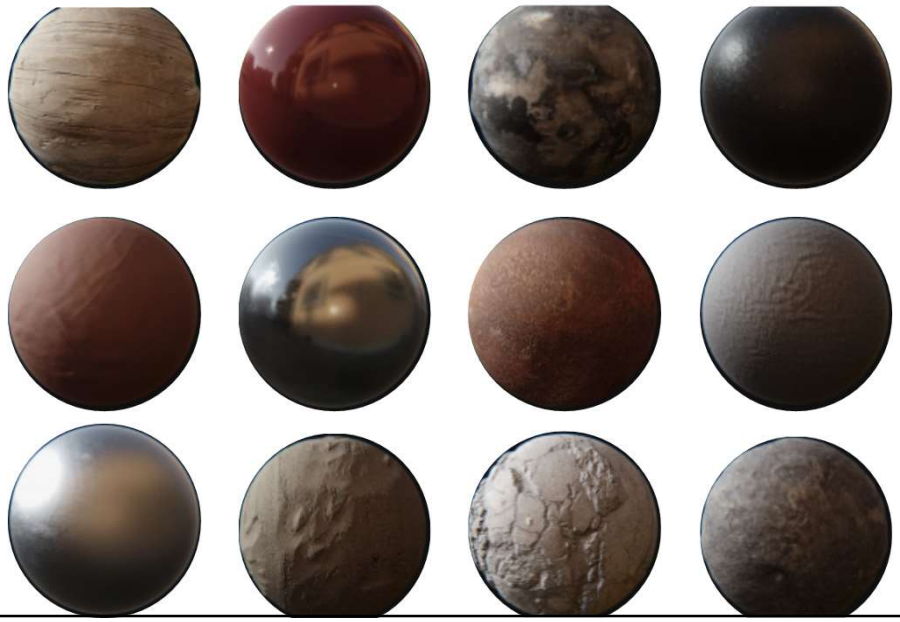
Main reasons:
1. more GPU processing power affords us more realism.
2. Programmable shaders.
3. More GPU RAM to store textures for parameters

- 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"

50

Much wider expressiveness is needed



51

Material models improving

53

Material models are improving

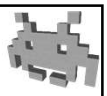
54

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

55

«Physically Based Lighting» (PBL)
aka «Physically Based Rendering» (PBR)



- A lighting model more inspired more by physical reality
  - For example, energy conservation
  - And less by tricks that just follow some intuition, see specular Phone reflections
- 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 separate* texture
  - Instead of:
    - diffuse color × baked AO : one texture (cheaper!)
- Warning: PBM & PBL are, basically, buzzwords ☺

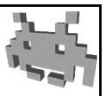
56

PBM and PBL: objectives

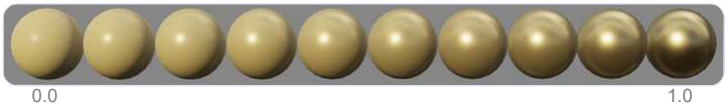
- Make realistic materials easier to design (by material artists)
- Make it possible to *capture* materials from real world samples
- Make it easier to make lighting look reasonably realistic… under many different lighting environment
- Standardize, and make it easier to share material description across different project applications
- Ideally: most real-world materials can be described accurately
- Ideally: no combination of Material parameters looks bad or wrong
- Use few parameters (ease storage, authoring), every parameter describe in a 0 to 1 range

57

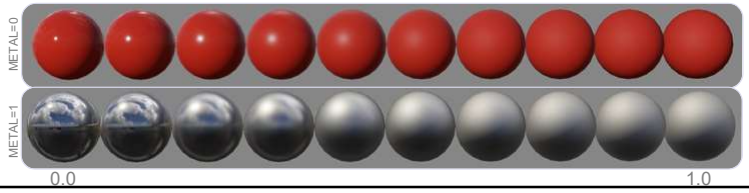
Physically Based Materials (PBM). A good choice of parameters



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



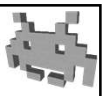
- Roughness (scalar)

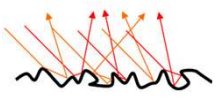


images: unreal engine 4

58

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 = 

59