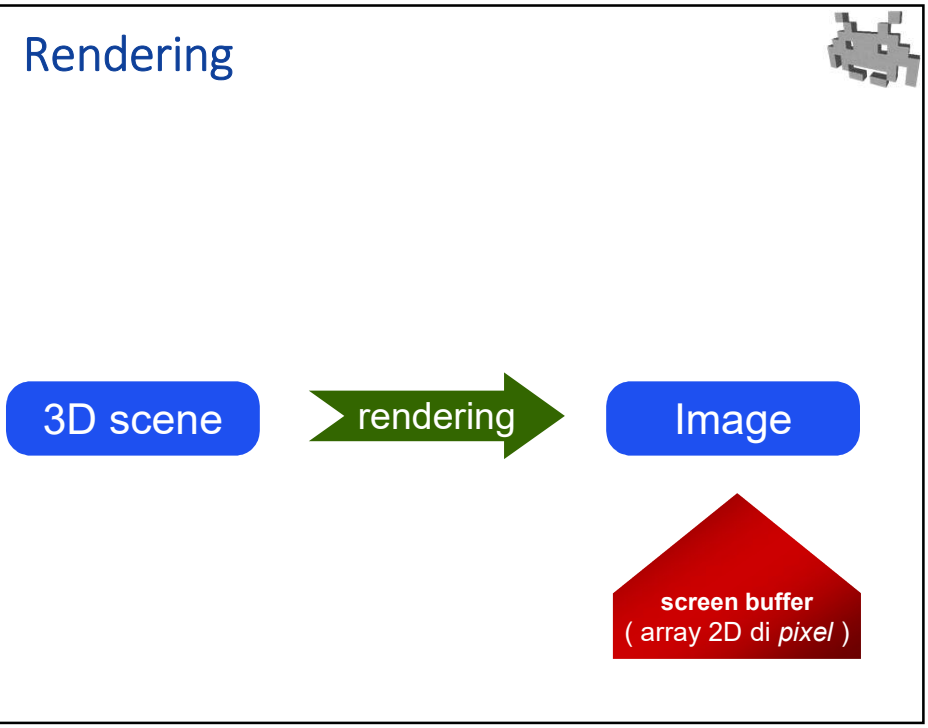
3D Videogames 2018/2019
Univ. degli Studi di Milano

Rendering in games

Part I: lighting & materials

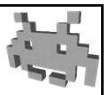
1

Rendering



2

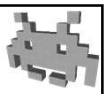
Rendering in games



- Real time
  - (20 o) 30 o 60 FPS
- Hardware based
  - Pipelined, stream processing
- therefore: class of algorithms:
  - rasterization based
- Complexity:
  - Linear with # of primitives
  - Linear with # of pixels

3

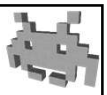
This lecture: a bird-eye view on…



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

4

This lecture

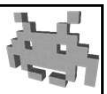


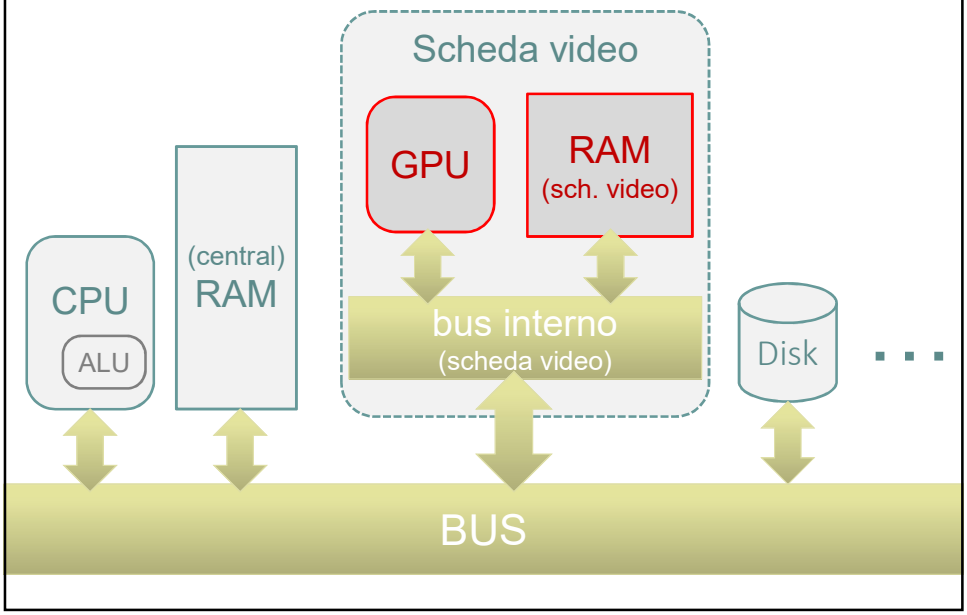
To learn more, see courses:

- GID
  Grafica & Immagini Digitali
- RTGP
  Real-Time Graphics Programming

5

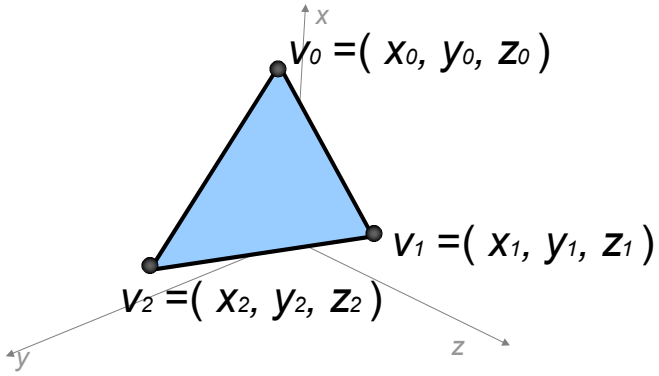
Reminder





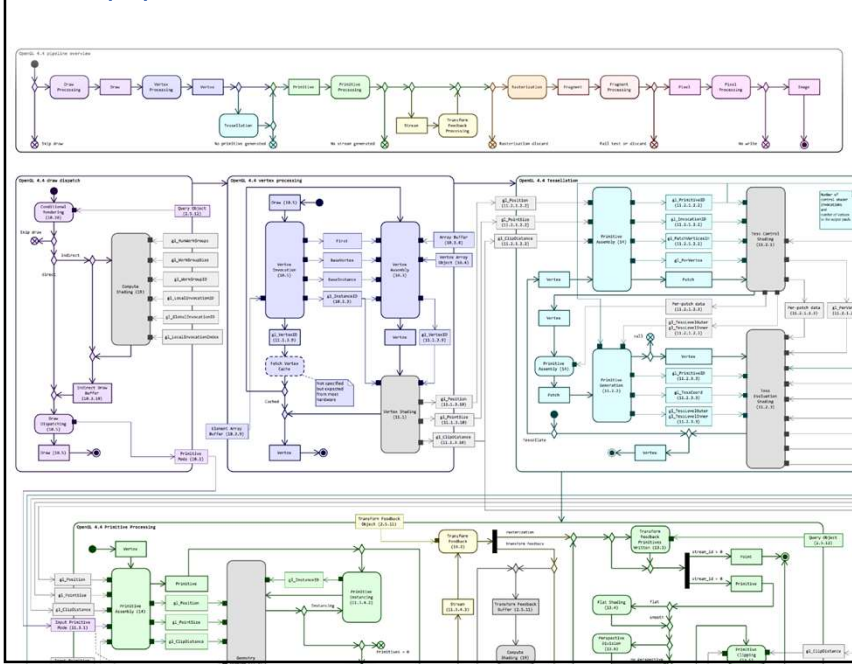
6

A triangle



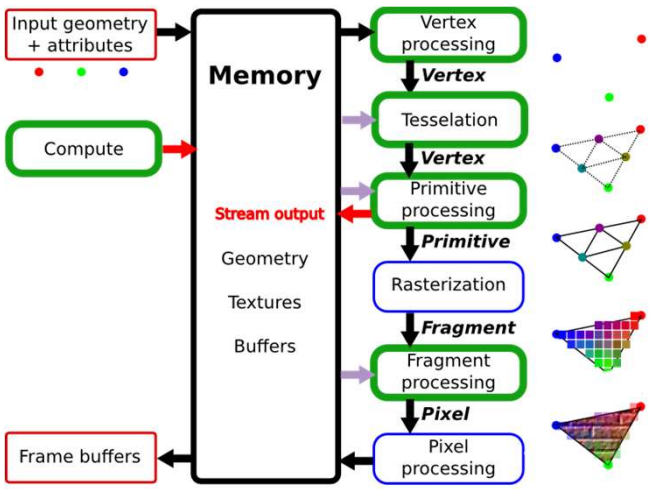
7

GPU pipeline



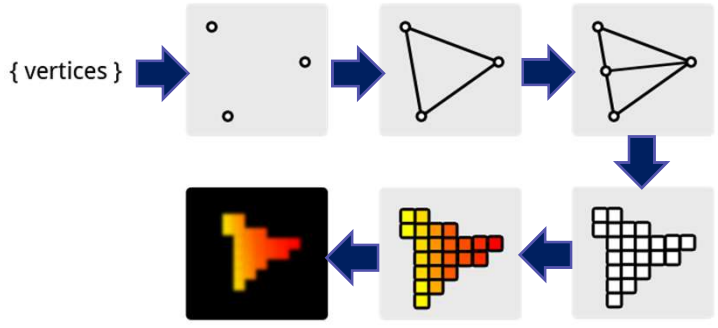
8

GPU pipeline – simplified



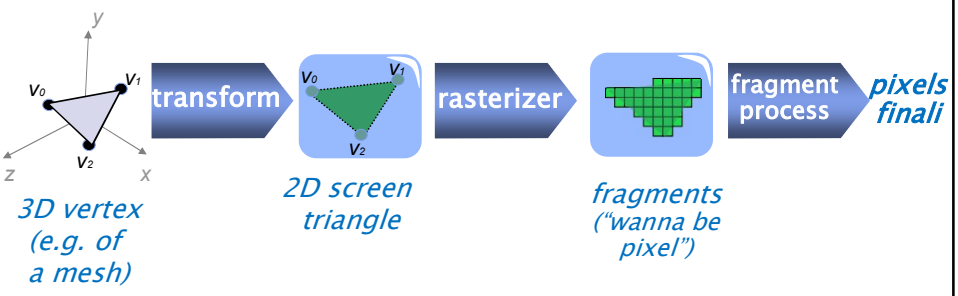
9

GPU pipeline – simplified *more*



10

GPU pipeline – simplified even *more*



3D vertex (e.g. of a mesh)

2D screen triangle

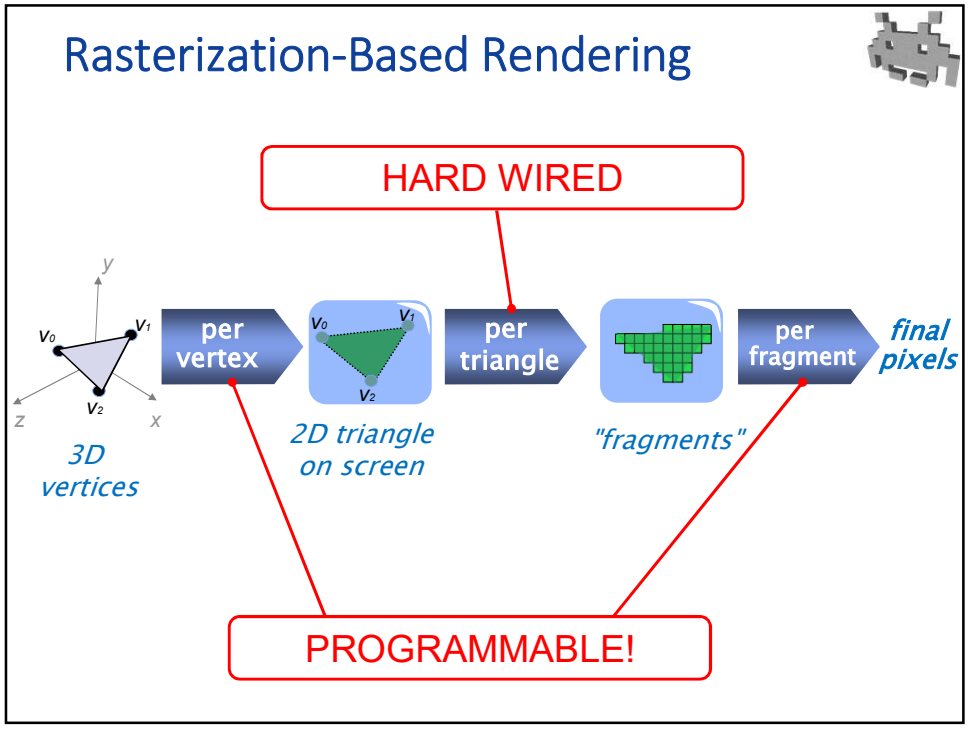
fragments ("wanna be pixel")

11

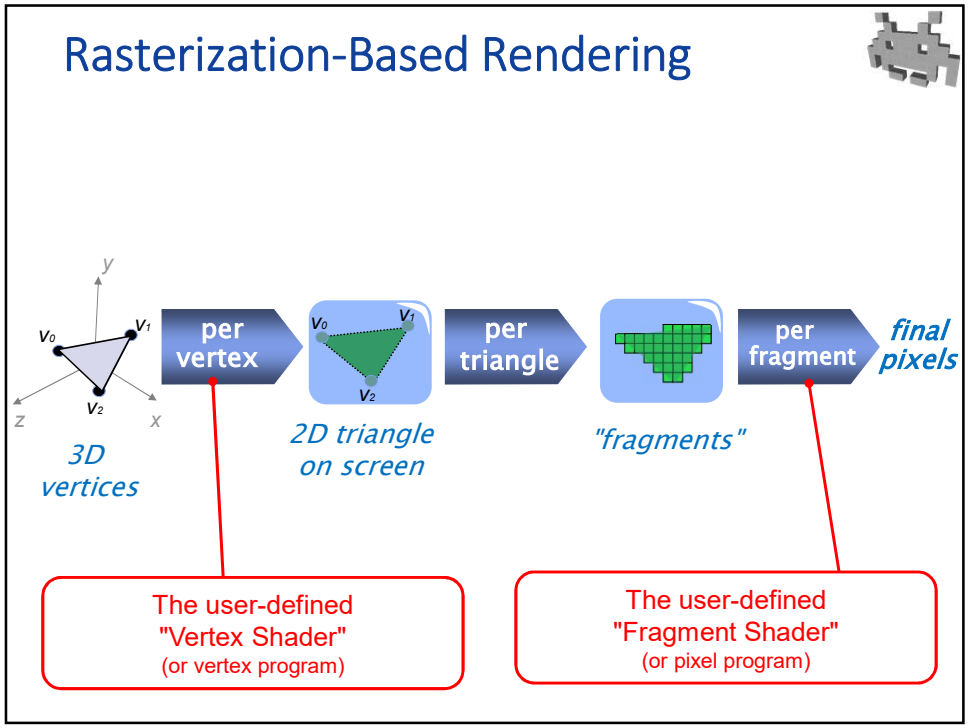
Rasterization based rendering: base schema

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

12

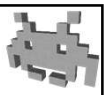

Rasterization-Based Rendering
HARD WIRED
per vertex
per triangle
per fragment
final pixels
3D vertices
2D triangle on screen
"fragments"
PROGRAMMABLE!

13


Rasterization-Based Rendering
per vertex
per triangle
per fragment
final pixels
3D vertices
2D triangle on screen
"fragments"
The user-defined "Vertex Shader" (or vertex program)
The user-defined "Fragment Shader" (or pixel program)

14

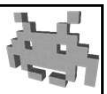
Shading languages examples

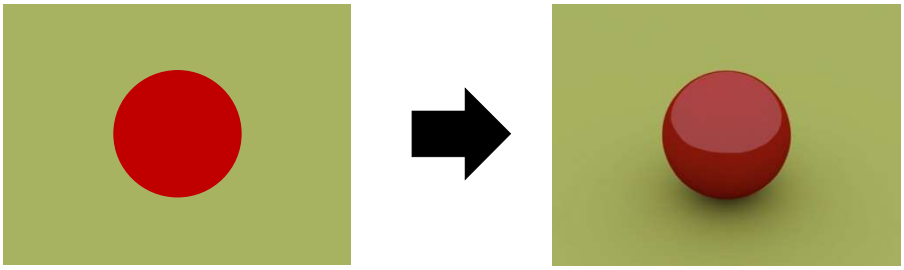


- High level:
  - HLSL (High Level Shader Language, Direct3D, Microsoft)
  - GLSL (OpenGL Shading Language)
  - CG (C for Graphics, Nvidia)
- Low lever:
  - ARB Shader Program
    (the "assembler" of GPU – now deprecated)

15

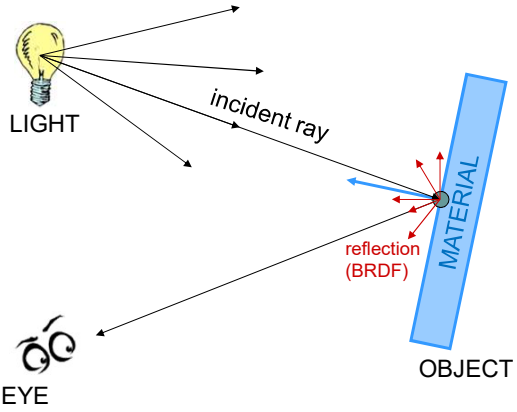
Lighting





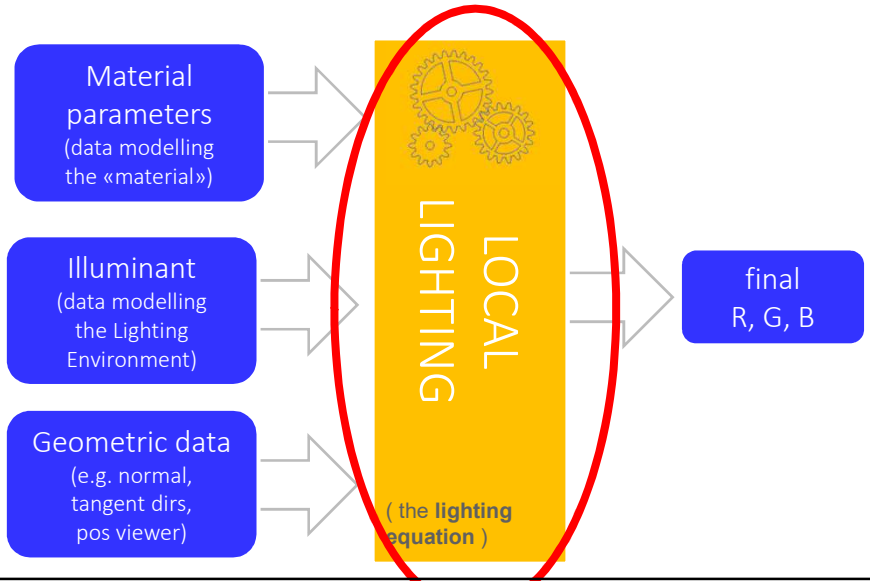
16

Local lighting



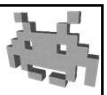
17

Local lighting



18

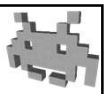
Lighting equations



- Different equations can be employed…
  - Lambertian
  - Blinn-Phong
  } basic
  - Beckmann
  - Heidrich–Seidel
  - Cook–Torrance
  - Ward (anisotropic)
  - …
  - + additional Fresnel effect
- Varying levels of
  - complexity
  - realism
    - (some are *physically based*, some are… just tricks)
  - material parameters allowed
  - richness of effects

19

Lighting equations: basics

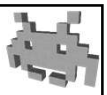


- Diffuse factor (aka Lambertian)
  - physically based
  - only dull materials
  - only material parameter:
    - base color
      (aka albedo, aka "diffuse" color)

- Specular factor (aka Blinn-Phong)
  - just a trick
  - add simulated reflections (highlights)
  - additional material parameters:
    - specular intensity (or, color)
    - specular exponent (aka glossiness)

20

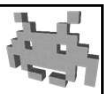
Lighting equations: basics



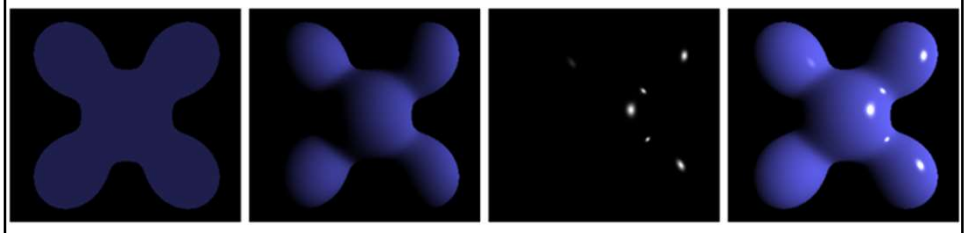
- Ambient factor:
  - assumption:
    a bit of light reaches the object
    from every dir
    - and is bounced by it in every dir
  - Pro: simple to compute:
    just an additive constant
    - the ambient light factor, a global
  - Con: terribly crude approx

21

Lighting equations: basics



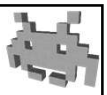
- Used together:

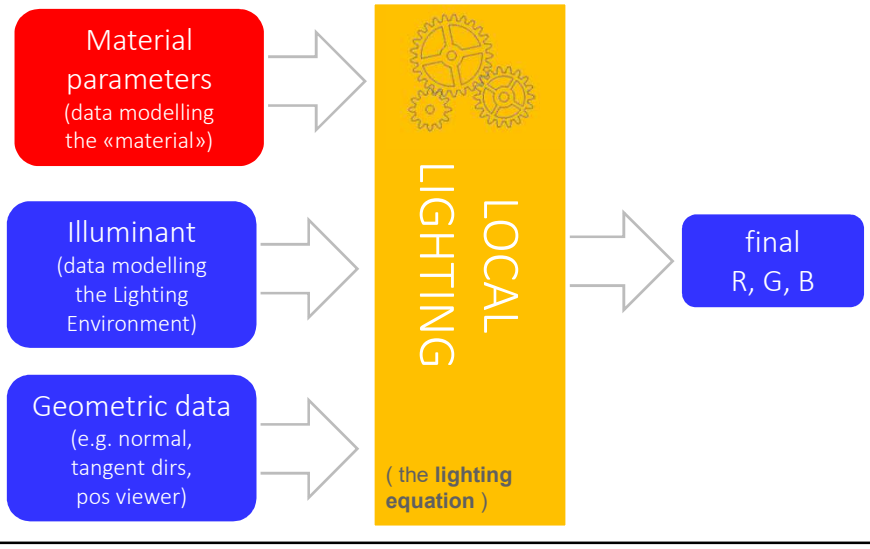


ambient + diffuse (or Labertian) + specular (or Phong) = final

22

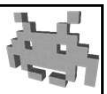
Local lighting





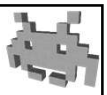
23

Materials



24

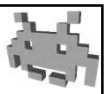
⚠ Terminology: «material» means two things



- Material Parameters
  - parameters modelling the local optical behavior of physical object
  - part of the arguments of the (local) lighting equation
- Material Asset
  - a common abstraction used by game engines
  - consisting of
    - a set of textures (e.g. diffuse + specular + normal map)
    - a set of shaders (e.g. vertex + fragment)
    - a set of global parameters (e.g. global glossiness)
    - rendering settings (e.g. back-face culling ON/OFF)
  - corresponds to the status of the rendering engines

25

Material Asset: remember how a mesh is rendered (hi-level)



- Load…
  - make sure all data is stored in GPU RAM
    - Geometry + Attributes
    - Connectivity
    - Textures
    - Shaders
    - Material Glob. Param.
    - Rendering Settings

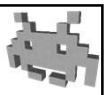
THE MESH ASSET

THE MATERIAL **ASSET**

- …and Fire!
  - send the command: *"do it"* !

26

Material parameters: Which ones?



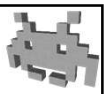
- Q: which set of parameters is a «material»?
- A: depends on the chosen lighting equation

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

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

27

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



- "OpenGL material", OBJ material
- The Lighting Equation is 4 simple terms:
  - Ambient + Diffuse + Specular (+ Emission)

see earlier

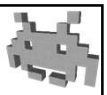
- The material is… a multiplier for each term, therefore:

usually, separate multiplier for R, G and B

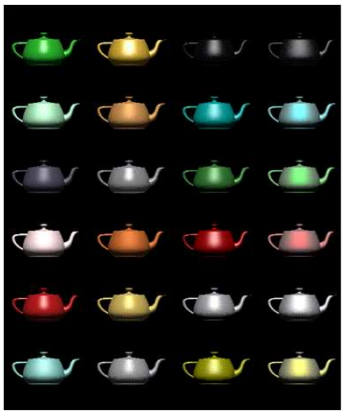
  - "Ambient" factor (RGB)
  - "Diffuse" factor (aka "Base Color", aka "Albedo")
  - "Specular" factor (RGB)
    (plus one "Specular Exponent", aka "glossiness")
    (a scalar param used in the formula for the specular term)
  - "Emission" factor (RGB)
    (only for stuff *emitting* light – otherwise 0,0,0)

28

Choice of material parameters: 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

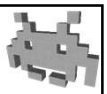


not too expressive ☹

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

29

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



- The Lighting Equation becomes more complex
  - in several different ways
- It feeds on more and more parameters…
  - Such as: Fresnel effect, Anisotropic effect, Reflectivity – with environment maps, …
- Authoring materials (task of the "material artist"): 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 right parameters for a given object
    - especially if it has to look good under widely different lighting conditions!

31

Material qualities: improving

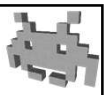
32

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

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

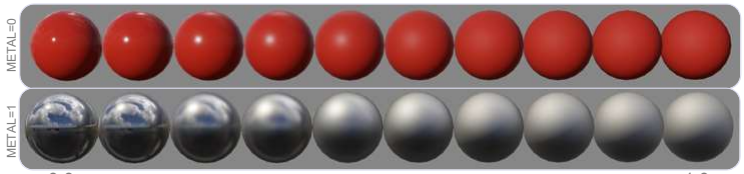
33

«Physically Based Materials» (PBM)



- Current popular choice of parameters:
  - Base color (rgb – or "diffuse", same as old school)
  - Specularity (scalar – or rgb sometimes)
  - "Metallicity" (scalar)

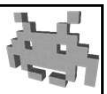
  - Roughness (scalar)



images: unreal engine 4

34

«Physically Based Lighting» (PBL)



- A lighting model accepting, as input, a PBM
  - note: this can be achieved with different equations
- Also, a lighting model taking fewer shortcuts than otherwise typical
  - e.g. using:
    - diffuse color: *one* texture
    - baked AO: *another* texture
  - instead of:
    - base color × baked AO : one texture

Ambient Occlusion Texture: see later

- Objectives: same as PBM (exp. under "realism")
- Warning: PBM & PBL are, basically, buzzwords ☺

35

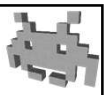
Material qualities: improving

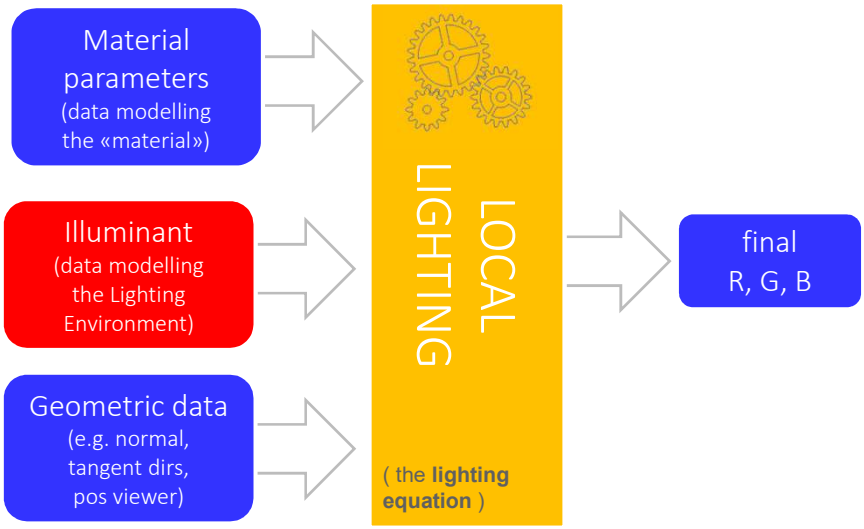
36

Much wider expressiveness

37

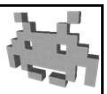
Local lighting





38

Illumination environments: types

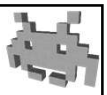


- Discrete
  - a finite set of individual light sources (plus a global ambient factor)
- Densely sampled
  - environment maps: textures sampling incoming light
- Basis functions
  - a spherical function stored as spherical harmonics coefficients

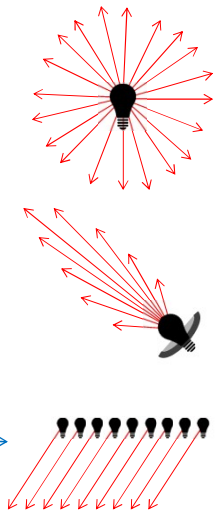
Also used jointly!

39

Illumination environments: discrete

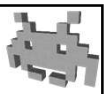


- a finite set of individual "light sources"…
  - few of them (usually 1-16)
- each one sitting in a node of the scene-graph
- each of a type:
  - point light sources
    - have: position
  - spot-lights
    - have: position, orientation, wideness (angle)
  - directional light sources
    - have: orientation only
- extra per light attributes:
  - color / intensity
  - fall-off function (with distance)
  - max range, and more



40

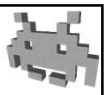
Illumination environments: discrete



- a finite set of "light sources"…
- …plus, one global "ambient light" factor
  - models other minor light sources + bounces
    - light incoming from every direction at every position
  - multiplier of the ambient term of the lighting equation
  - examples:
    - in a overcast outdoor scene: *high*
      - (dim shadows, flat looking lighting: every photographs' favorite for portraits!)
    - in realistic outer space: *zero*
    - in any other scenes : *something in between* (e.g. sunny day, or torch lit cave)

41

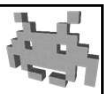
Illumination environments: discrete



- Pros:
  - simple to position / reorient individual light sources
    - both at design phase, or dynamically (at game exec)
    - as they just sit in nodes of the scene-graph!
  - quite faithfully model of certain illuminants, e.g.
    - explosions (positional lights)
    - car lights (spot-lights lights)
    - sun direction (directional light)
  - relatively easy to compute (hard, soft) shadows for them
    - (see shadow-map, later)
- Cons:
  - each discrete light requires extra processing … for each pixel!
    - this is why there's a hard limit on their number!
    - this is why they are often given a (physically unjustified) radius of effect
  - the don't model well:
    - area light sources (e.g. from back-lit clouds)
    - subtler illumination effects
    - environment to be seen mirrored in (e.g. metal) objects
- Often good for the main illuminants of the scene!

42

Illumination environments: densely sampled

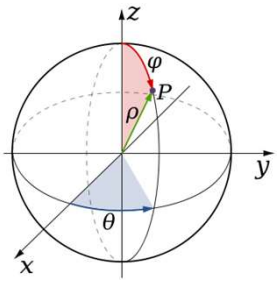


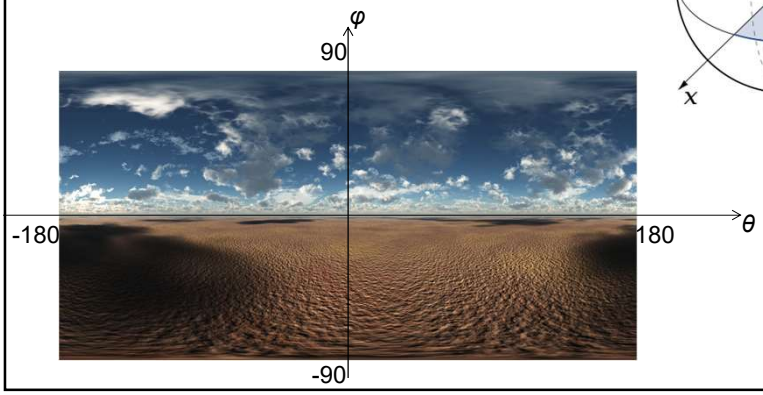
- A light intensity / color from each direction
- Asset to store that:
  "Environment map" texture

43

Illumination environments: densely sampled

- Latitude/longitude format





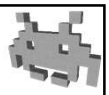
44

Illumination environments: densely sampled

- Also "sky-map" texture
  - when it's only / predominantly the sky to be featured
  - doubles as textures for "sky boxes"

45

Illumination environments: densely sampled

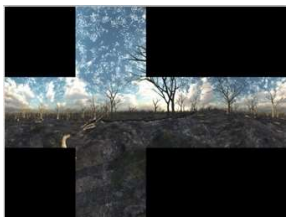


- Environment map: (asset)
  a texture with a texel *t* for each direction *d*
  - texel *t* stores the light coming from direction *d*
- Q: how to find u,v position of *t* for a given *d* ?
  - i.e. how to parametrize (flatten) the unit sphere
- Different answers are possible…

unit vector

latitude/longitude format

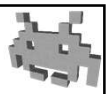
mirror sphere format



cube-map format
(ad hoc HW support!)

46

Environment map (asset)



- A texture with a texel *t* for each direction *d*
  - texel *t* stores the light coming from direction *d*
  - Used to compute reflections (on curved objects)
- Pro:
  - realistic, complex, detailed, hi-freq, light environments
    - best result for mirroring (e.g. shiny metal, glass, water) materials
  - can be captured from reality
- Con:
  - expensive
    - storage cost, lighting computation cost
  - hard for the engine to dynamically change
    - easy, for static environments only

47