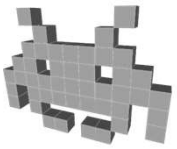
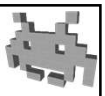
Università degli Studi di Milano
3D Videogames



Rendering in 3D games

1

Course Plan



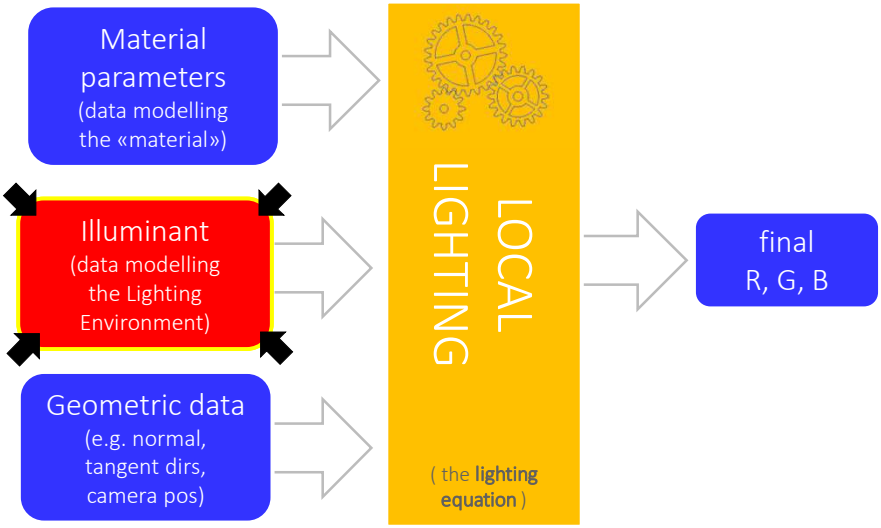
lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics** +
lec. 5: Game **Particle Systems**
lec. 6: Game **3D Models**
lec. 7: Game **Materials**
lec. 8: Game **Textures**
lec. 9: Game **3D Animations**
lec. 10: **3D Audio** for 3D Games
lec.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

For a more general, deeper discussion of many of the subjects of this lecture, see the courses CG «Computer Graphics» and RTGP «Real-Time Graphics Programming»

★ bridge lectures

2

Part I:
modelling the Light Environment



3

Approaches to model the light environment in 3D games

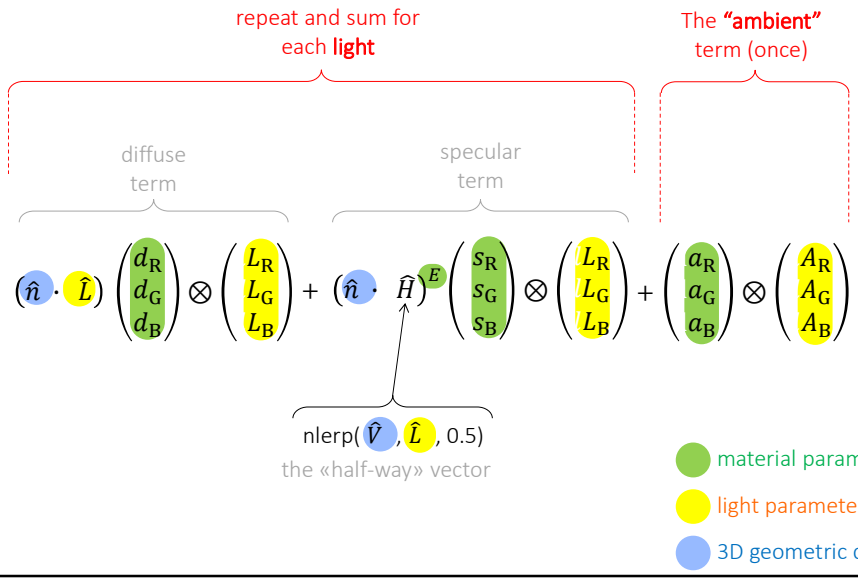
We are about to discuss 3 different approaches:

- Discrete lighting environment
  - a finite set of individual light sources (including one global ambient factor for the “leftovers”)
- Densely sampled lighting environment
  - environment maps: textures sampling incoming light from every dir
- Basis functions
  - a spherical function stored as spherical harmonics coefficients

(They can be, and are, used jointly)

4

A simple lighting equation (recap)



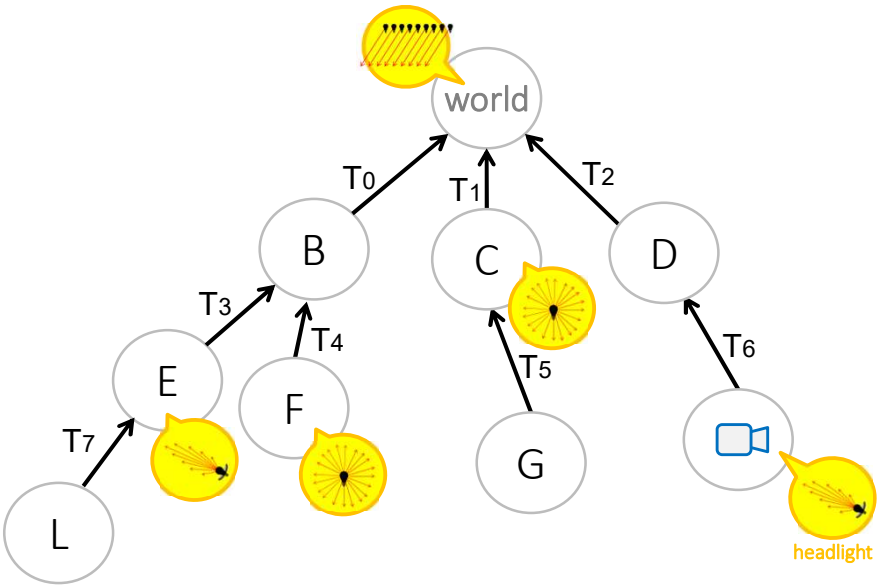
5

Discrete illumination environments: a set of individual *light sources*

- a finite set of "light sources"…
  - not too many (e.g. ≤16)
  - if more, can be assigned "priorities" to pick a subset for every object being rendered
- each light sits in a node of the scene-graph!
- each light is of one type…

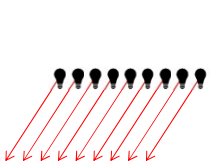
6

Discrete illumination environments

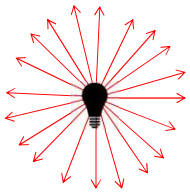


7

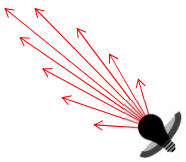
Lights sources types & their attributes



directional lights
- intensity / color
- direction



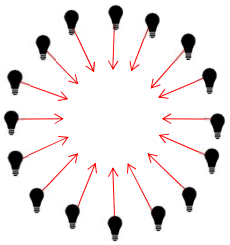
point lights
- intensity / color
- position
- falloff function (opt.)



spot-lights
- intensity / color
- position
- direction
- falloff function (opt.)
- angular falloff funct.
- “cookie” texture

8

Lights sources types & their attributes



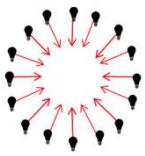
ambient light

- intensity / color
- *(that's it, no other attribute)*

Can be shadowed (blocked, negated, occluded) by an AMBIENT OCCULSION term .
E.g., backed per-texel (or per-vertex)

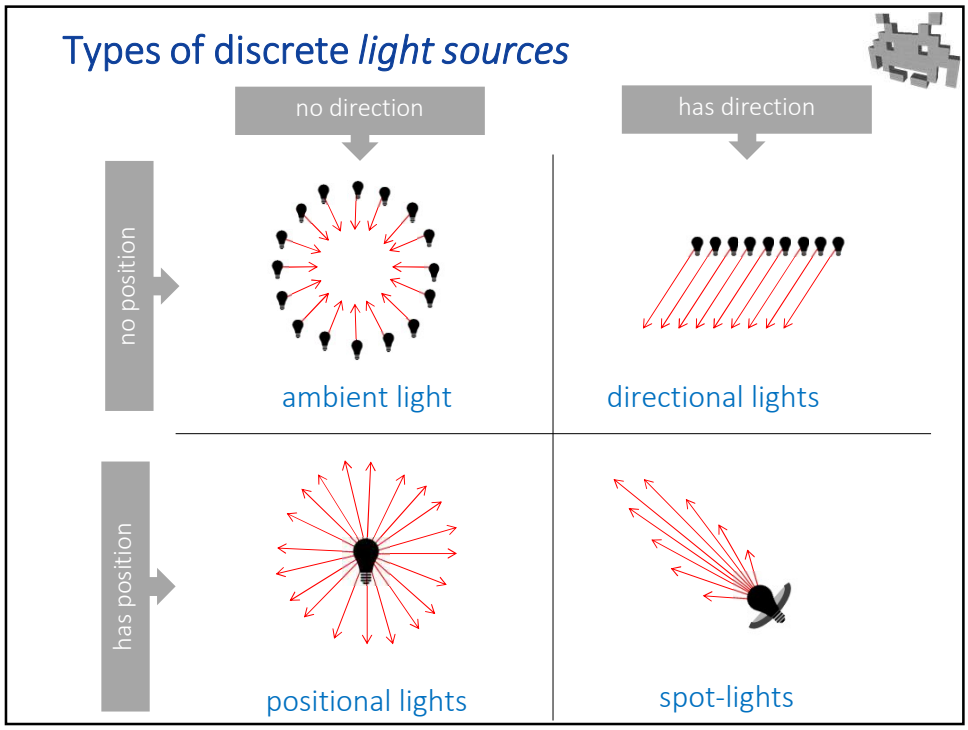
9

Ambient light

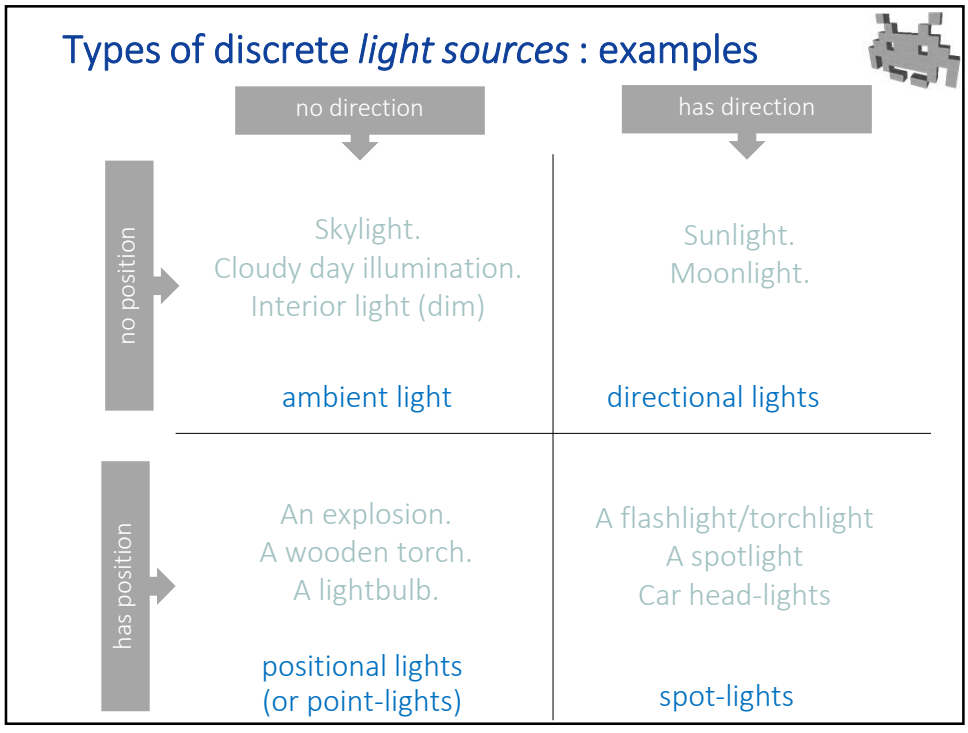


- models other all minor light sources + bounces
  - light incoming "from every direction at every position"
- examples:
  - in an overcast outdoor scene: it's *high*
    - dim shadows, flat looking lighting:
      every photographs' favorite for portraits!
  - in realistic outer space: it's *zero*
  - in any other scenes : *something in between*
    - e.g., sunny day, or a torch-lit cave
- the lighting env. includes only one (or zero) lights of this type
- All other light sources are spatialized
  - Thus, they reside in the scene graph!

10

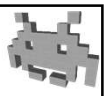

Types of discrete *light sources*
no direction
has direction
no position
has position
ambient light
directional lights
positional lights
spot-lights

11


Types of discrete *light sources* : examples
no direction
has direction
no position
has position
Skylight.
Cloudy day illumination.
Interior light (dim)
ambient light
Sunlight.
Moonlight.
directional lights
An explosion.
A wooden torch.
A lightbulb.
positional lights
(or point-lights)
A flashlight/torchlight
A spotlight
Car head-lights
spot-lights

12

Distance fall-off functions
for positional lights & spotlights

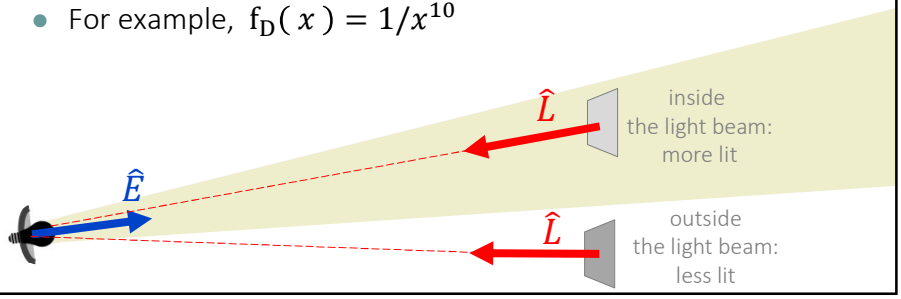


- The light intensity of positional lights and spotlights can be dimmed down with distance from light-pos $\mathrm{P_L}$ to the pos of the fragment being lit $\mathrm{P_P}$, scaling it by some positional «fall-off» function
$$\mathrm{f_P}(\|\mathrm{P_L} - \mathrm{P_P}\|)$$
  - In the real physical world, $\mathrm{f_P}(d) = 1/d^2$
  - Other functions can be used, for example $\mathrm{f_P}(d) = 1/d$

13

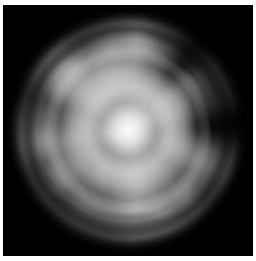
Angular fall-off functions
for spotlights

- For spotlights, the intensity is also dimmed down by an «angular fall-off» function, when the direction of the light emission $\hat{E}$ mismatches the light direction $\hat{L}$, scaling it by some function
$$\mathrm{f_D}(-\hat{E} \cdot \hat{L})$$
  - For example, $\mathrm{f_D}(x) = 1/x^{10}$



14

Spot-lights: they can use a "cookie texture"



as an alternative to angular fall-off functions

15

In the lighting equation…

repeat and sum for each **discrete light source**

the one **"ambient" light**

diffuse term

specular term

$$(\hat{n} \cdot \hat{L}) \begin{pmatrix} d_R \\ d_G \\ d_B \end{pmatrix} \otimes \begin{pmatrix} L_R \\ L_G \\ L_B \end{pmatrix} + (\hat{n} \cdot \hat{H})^E \begin{pmatrix} s_R \\ s_G \\ s_B \end{pmatrix} \otimes \begin{pmatrix} L_R \\ L_G \\ L_B \end{pmatrix} + \begin{pmatrix} a_R \\ a_G \\ a_B \end{pmatrix} \otimes \begin{pmatrix} A_R \\ A_G \\ A_B \end{pmatrix}$$

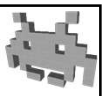
Light incoming direction
For directional lights: just a constant.
For a point- or spot-light: $\hat{L} = \frac{P_L - P_P}{\|P_L - P_P\|}$
pos of light, pos of lit point

Light RGB intensity
For a dir. light: a consts.
For point or spot-lights: attenuated by falloff functions (of angle, dist)

parameter = of the light

16

Types of discrete lights (a summary)

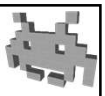





	Ambient light	Directional light	Positional light	Spotlight
geometry	*(nothing)* *(assumed at infinite)*	Direction (versor) *(assumed at infinite)*	Position (point)	Position (point) & Direction (versor)
can be dimmed by	-	-	Falloff function	Falloff function Angular falloff function "Cookie" texture
can be blocked by	Ambient Occlusion either baked (per-vertex or per-texel) or dynamically computed (see SSAO later)	Cast shadows (usually) dynamically computed (see shadow-map technique later)		
how many	0-1	0-N	0-N	0-N
parameters	Color/Intensity (RGB value) Priority?			

17

In which space to compute the lighting?



repeat for each light source

diffuse term + specular term + ambient term + emission term

$$(\hat{n}\cdot\hat{L})\begin{pmatrix}d_R\\d_G\\d_B\end{pmatrix}\otimes\begin{pmatrix}L_R\\L_G\\L_B\end{pmatrix} \quad (\hat{n}\cdot\hat{H})^E\begin{pmatrix}s_R\\s_G\\s_B\end{pmatrix}\otimes\begin{pmatrix}L_R\\L_G\\L_B\end{pmatrix}+\begin{pmatrix}a_R\\a_G\\a_B\end{pmatrix}\otimes\begin{pmatrix}A_R\\A_G\\A_B\end{pmatrix}+\begin{pmatrix}e_R\\e_G\\e_B\end{pmatrix}$$

$$\hat{L}=\frac{P_L-P_P}{\|P_L-P_P\|}$$

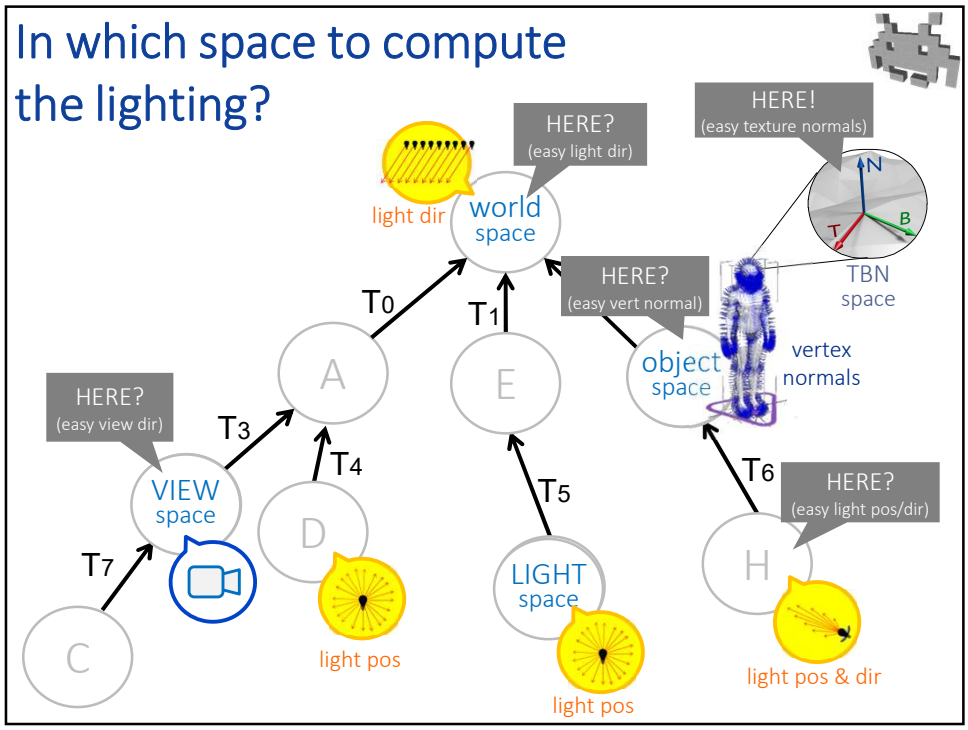
nlerp($\hat{V}$, $\hat{L}$, 0.5)

the «half-way» vector

3D Point / Vector / Versor

Q : in which space to express them (and the others like them)?
A: whichever!
As long as it's the *same space*

18



19

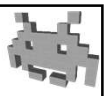
In which space to compute the lighting?

- All versors that used in 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)
  - World space?
  - Local object space? (the space of the object currently being rendered)
- With normal maps, usually the most efficient solution is:
  - Use the same space the normals are expressed
  - For normal stored as attribute: the Local Space (aka Object Space)
  - For Tangent Space normal maps: in the the TBN space. Then…
  - …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 amount of transformations needed

for anisotropic materials

20

Discrete illumination environments Summary



- Pros:
  - simple to re-position / reorient individual light sources
    - both at design phase, or dynamically (at game exec)
  - good model of illuminants as:
    - explosions (positional lights)
    - car lights (spot-lights lights)
    - sun direction (directional light)

    main illuminants of the scene!
  - relatively easy to compute (hard, soft) shadows for them

    see shadow map later
- Cons:
  - each light source requires extra processing … for each pixel!
    - therefore: hard limit on their number. Prioritize them
    - therefore: are often given a (physically unjustified) radius of effect
  - they don't model well:
    - area light sources (e.g., from back-lit clouds)
    - reflections on metal objects

21

Continuous lighting environments

- In general terms, a lighting environment is a function:

  set of unit directions

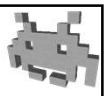
$$f : \Omega \longrightarrow \mathbb{R}$$

  or $f : \Omega \longrightarrow \mathbb{R}^3$ if we want colored lights

- $f(\hat{d}) \equiv$ intensity/color of light coming form dir $\hat{d} \in \Omega$
- With discrete light environments:
  - we define f only over a few $\hat{d}$, with individual lights
  - f is a constant for every other $\hat{d}$, with ambient light
  - note: f is potentially defined differently in every point in the scene (when positional light sources are used). Cool!
  - note: f is dynamic: it's easy to move & turn on/off & change lights from a frame to the next. Cool!
  - But its not super realistic lighting env.

22

Continuous lighting environments



- In general terms, a lighting environment is a function:

$$f : \Omega \longrightarrow \mathbb{R}$$

(Ω: set of unit directions)

or $f : \Omega \longrightarrow \mathbb{R}^3$ if we want colored lights

- $f(\hat{d}) \equiv$ intensity/color of light coming form dir $\hat{d} \in \Omega$
- Let's see two ways to encode a continuous f instead
  - Way 1: environment maps
    (just sample it, store it as a texture)
  - Way 2: with basis functions
    (good for very smooth lighting envs)
  - note: for now, we assume f is the same in everywhere in the scene
  - note: not easy to change f over time (there are ways – see later)
  - typically, a more realistic lighting env.

23

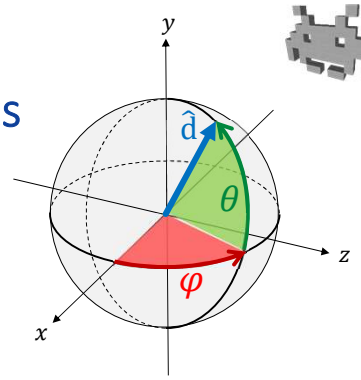
Densely sampled illumination environments

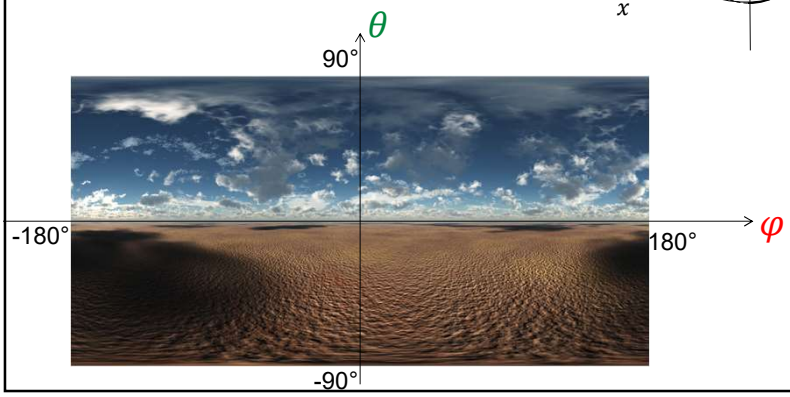
- A light intensity / color from each direction $\hat{d}$
- Asset to store that:
  "environment map" texture

24

Densely sampled illumination environments

- Latitude/longitude format (of a unit vector $\hat{d}$)





25

Densely sampled illumination environments

- Aka "sky-map" texture
  - because it doubles as a texture for the "sky boxes"

26

Densely sampled illumination environments

- Environment map: (asset)
  a texture with a texel $\mathbf{t}$ for each direction $\hat{\mathbf{d}}$ (unit vector)
  - $\mathbf{t}$ stores the intensity/color of the light coming from direction $\hat{\mathbf{d}}$
- Q: how to determine u, v position of $\mathbf{t}$ for a given $\hat{\mathbf{d}}$?
  - i.e. how to parametrize (flatten) the unit sphere
- Different answers are possible…


latitude/longitude format


mirror sphere format


cube-map format (ad-hoc HW support!)

27

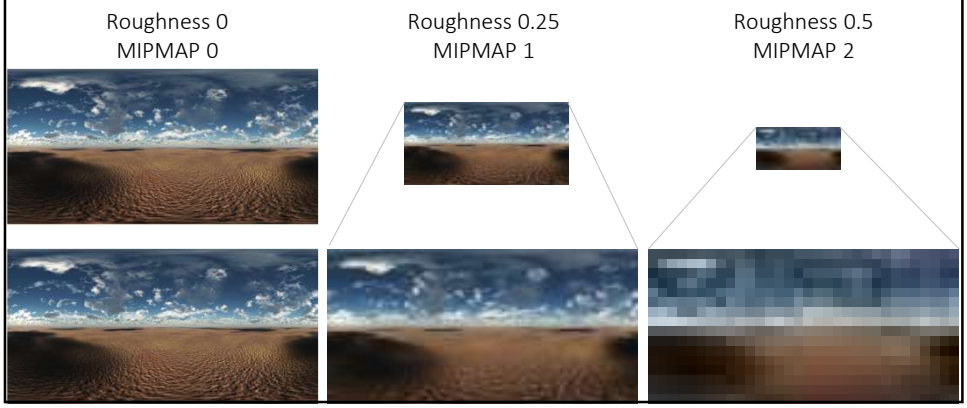
Environment map (asset)

- A texture with a texel $\mathbf{t}$ for each direction $\hat{\mathbf{d}}$
  - $\mathbf{t}$ stores the light coming from direction $\hat{\mathbf{d}}$
  - useful to compute reflections on (curved) metallic objects
  - often HDR (see later)
- Pro: realistic, complex, detailed, hi-freq, light env
  - best for mirroring materials (such as metal, glass, water)
- Pro: can be captured from reality
  - see “mat-cap”
- Con: expensive to update for dynamic scenes
  - no prob, for static environments only
- Con: assume far away illuminants
  - Not accurate for close illuminant

28

Environment map (asset): uses

1. Reflection mapping
   - metallic objects
   - in short: material roughness → mipmap level!



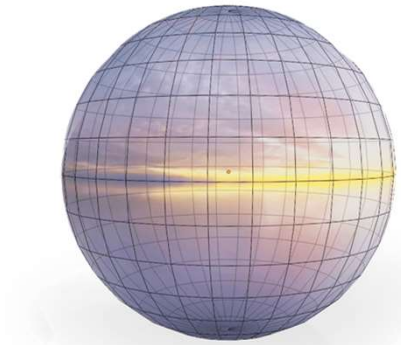
29

Environment map (asset): uses

1. Reflection mapping
   - metallic objects
   - material roughness → mipmap level!
2. More generally,
   description of the lighting env
   - for lighting computation
3. Doubles as a way to cover the background
   - as a texture for the
     3D “skybox” / “skydome” mesh

30

Skydome / skybox mesh



Its transform is defined as:

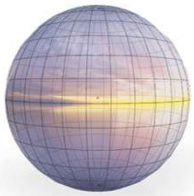
- Position (translation) as the camera node
- Rotation (orientation) as the world node

Textured with:

- The environment map

31

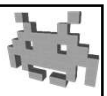
Skydome / skybox mesh



- A mesh encapsulating the entire scene
  - shows far away objects
  - note: for technical reason, the rendering doesn't show really far away objects – see "far plane clipping"
  - technically, it shows objects at ∞ distance
- Shaped as a large cube/sphere surrounding the scene
- Centered in the same node as the camera
  - so, it "follows the camera": "players can never reach it"
- But rotated as the world frame
  - not as camera!
  - (similar to: the transform of the microphone node)
- The environment map doubles as a color texture over this mesh
  - The sphere is painted with the background objects (stars, sky, mountains on the horizon, etc)

32

Light environments: using Basis Functions



- Lighting environment:
  a *continuous* spherical function $f : \Omega \rightarrow \mathbb{R}$
  - set of all unit vectors (i.e., surface of the unit sphere)
  - or R^3 if RGB colored light
- $f(\hat{v})$ = amount of (rgb) light coming from direction $\hat{v}$
- Store f through basis functions

fixed spherical "basis" functions (always the same ones)

$$f(\hat{v}) \cong w_0 \cdot f_0(\hat{v}) + w_1 \cdot f_1(\hat{v}) + w_2 \cdot f_{2,0}(\hat{v}) + w_3 \cdot f_3(\hat{v}) + \cdots$$

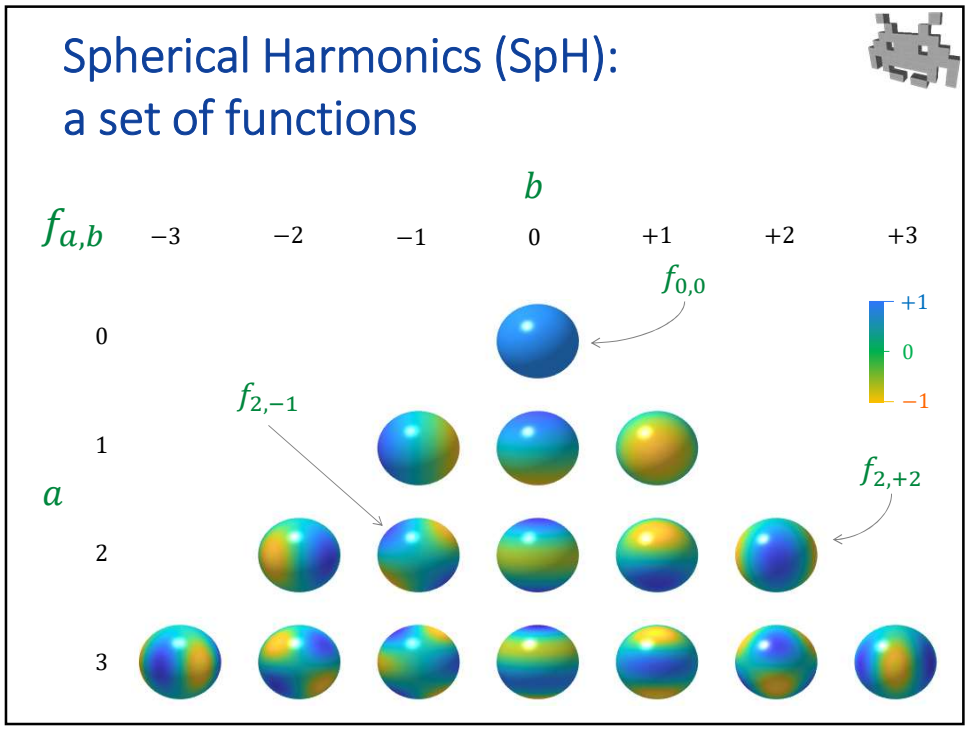
a few scalar values to be stored, in order to represent (an approx. of) f

34

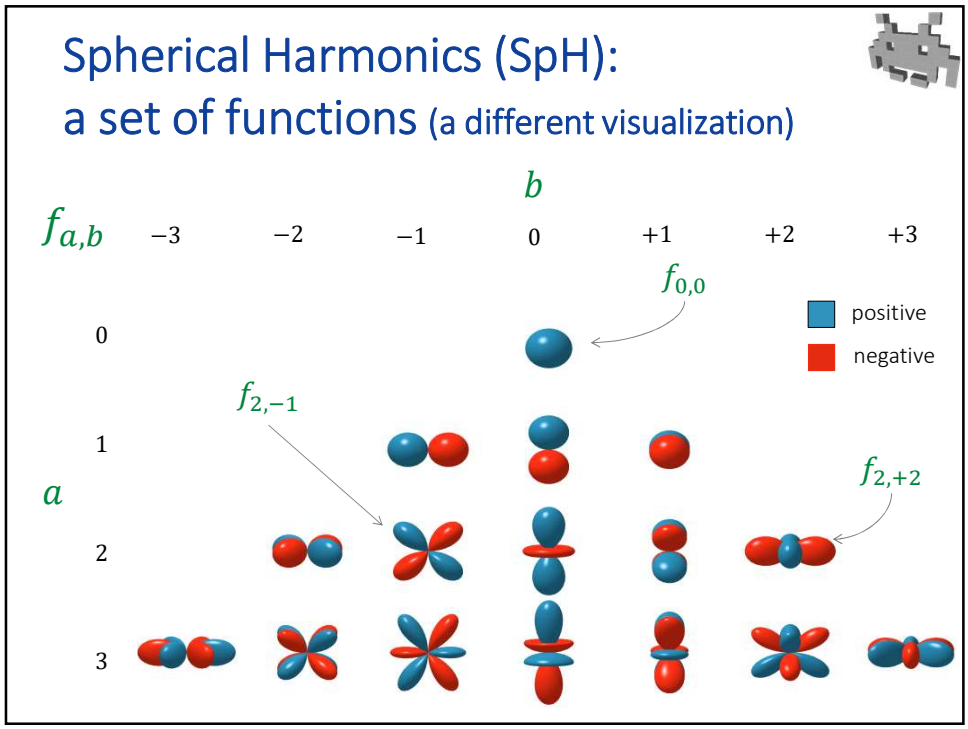
Spherical Harmonics (SpH): a good choice for the basis functions

- Spherical Harmonics is a good set of basis functions for spherical functions
- Each function in the set has two indices a, b (the degree)
  - $f_{a,b}(\hat{\mathrm{d}})$ with $a \geq 0$, $-a \leq b \leq +a$
- each function is (uniform) polynomial of some degree in the coordinates of $\hat{\mathrm{d}}$
  - so all function are easy to evaluate, to integrate, etc
- we can choose up to which degree store coefficients, e.g.
  - up to degree 1 => store 4 coefficients: for (0,0) (1,-1) (1,0) and (1,+1)
  - up to degree 2 => store 9 coefficient, for
    (0,0)
    (1,-1) (1,0) (1,+1)
    (2,-2) (2,-1) (2,0) (2,+1) (2,+2)
  - in general: up to degree N => store (N+1)^2 coefficients

35

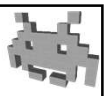

Spherical Harmonics (SpH):
a set of functions
$f_{a,b}$
b
−3
−2
−1
0
+1
+2
+3
a
0
1
2
3
$f_{0,0}$
$f_{2,-1}$
$f_{2,+2}$
+1
0
−1

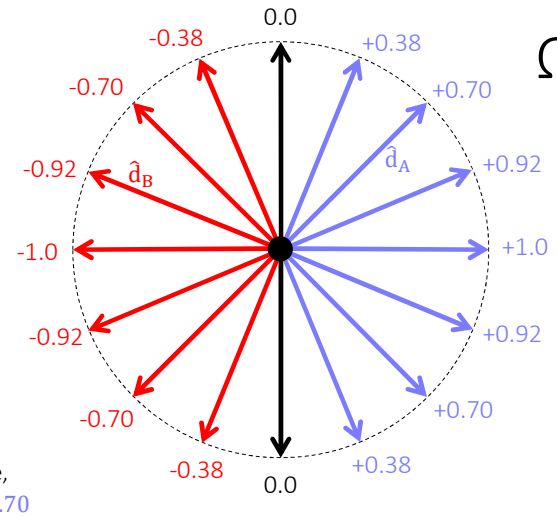
37


Spherical Harmonics (SpH):
a set of functions (a different visualization)
$f_{a,b}$
b
−3
−2
−1
0
+1
+2
+3
a
0
1
2
3
$f_{0,0}$
$f_{2,-1}$
$f_{2,+2}$
positive
negative

38

That graphical visualization (in a 2D example)

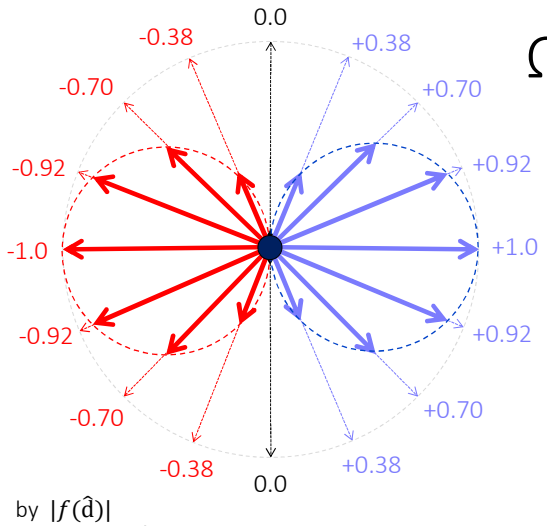




Here, for example,
$f_{1,-1}(\hat{d}_A) = +0.70$
$f_{1,-1}(\hat{d}_B) = -0.92$

39

That graphical visualization (in a 2D example)



scale each $\hat{d} \in \Omega$ by $|f(\hat{d})|$
color ●/● according to sign of $f(\hat{d})$

40

Spherical Harmonics (SpH): good properties

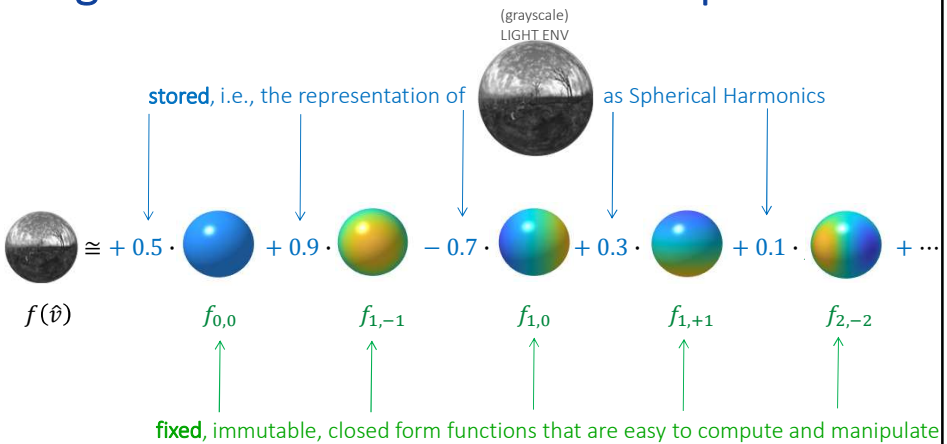
- $f_{0,0}$ is degree 0, so it's a just constant
  - so, it controls the total amount of light (irrespective of direction)
- They are "normalized"
  - the integral of each $f\hat{}2$ over Ω is 1
- They are "orthogonal" to each other
  - The integral of the product of any two different $f_{a,b}$ (which is kinda the "dot product" between functions) is 0
  - In particular, the product with $f_{0,0}$ has integral 0
    So, (except $f_{0,0}$) all have integral over Ω = zero
  - Intuition: orthogonality in practice means that each Sph. Harmonics expresses a mode of distribution of light that not other does
  - That is , functions are not redundant, coefficients are maximally informative
  - Also, it means that we can find the coefficient $w_{a,b}$ for $f_{a,b}$ for a given light environment by integrating its product with $f_{a,b}$
  - Analogy: it's like an orthonormal bases for vectors!

so, it's the Ambient light

so, they control the distrib. not the *quantity*, of light

41

Light probes: Light environment stored with SpH

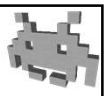


f is stored as (+0.5 , +0.9 , −0.7, +0.3, 0.1, …)

(if it's a colored Light Env, this is repeated for each R,G,B channel)

42

Light probes: Light environment stored with SpH



- Spherical Harmonics (SPH) in brief:
  - store Illumination Env as a small number (4, or 9, or 16…) of scalar weights of as many fixed spherical basis functions.
- Pros:
  - very compact representation
  - it models continuous functions well: good for smooth lighting environments
  - it allows for efficient computation of the Lighting equation
  - it's easy to interpolate between (two, or more) light envs!
- Cons:
  - continuous functions *ONLY*
    - not good for hi-freq details: for example, no hard lights
    - not sudden variations (unless very many coefficient used)
- Good for soft light env

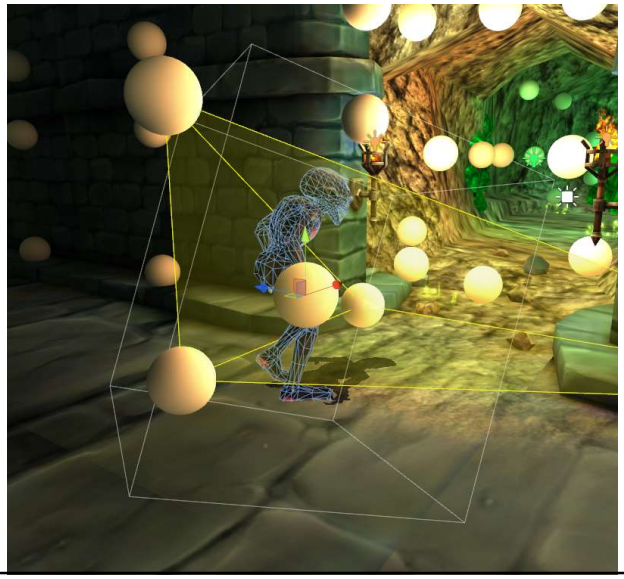
43

Light probes (position-dependent lighting env)

- A light probe == a (precomputed) lighting env. to be used around a given 3D position of the scene
- Light Probe lighting:
  - preprocessing: disseminate the scene with light probes
    - Store them as… low-res environment maps
    - …or, with SPH (the standard solution)
  - at rendering time, for an object currently in pos (xyz), use an interpolation of near-by "light probes"
    - note: two (or more) SPH function can be interpolated!
    - easy: just interpolate the weights

44

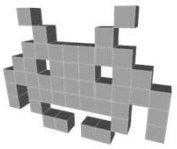
Light probes (position-dependent lighting env)



45

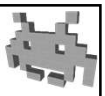
Light probes (position-dependent lighting env)

46



Rendering in 3D games
Common CG techniques in games

68

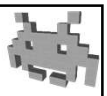


Rendering task for in 3D games: overview

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

69

High-level view of mesh rendering



To render a mesh:

- load in GPU RAM:
  - ✓ Geometry + Attributes
  - ✓ Connectivity
  - ✓ Textures
  - ✓ Vertex + Fragment Shaders
  - ✓ Global Material Parameters
  - ✓ Rendering Settings
- issue the Draw-call

THE MESH ASSET

THE MATERIAL ASSET

For this lecture, we need go lower level (a bit)

70

Off-screen buffers

- The rendering produces a screen buffer (2D array of RGB pixel) that is sent to the screen and is made visible to the player
- Any buffers that is used internally but and not sent to the screen is called an off-screen buffer
  - Example: the depth buffer (2D array of depth values)
  - Example: the back-screen buffer (double buffering techniques)
- Many rendering techniques are based on producing an off-screen buffer *then* using it in another pass
  - For example, to re-using it a texture in another rendering pass

71

Depth buffer (or Z-buffer) (or depth-map)

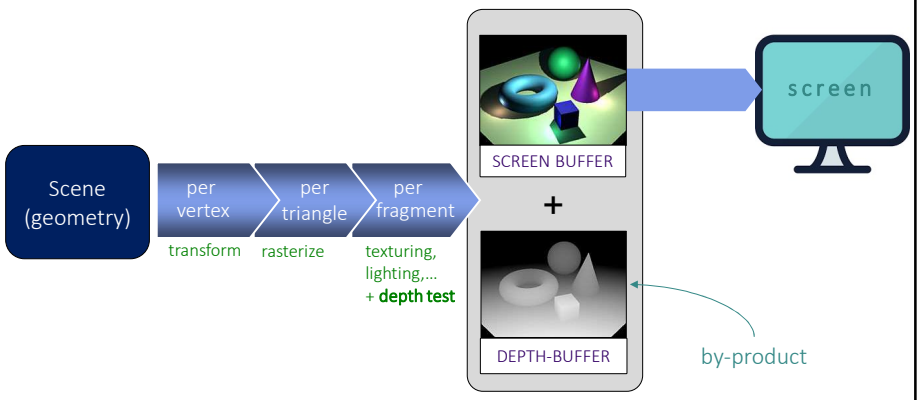
- Any rendering producing a screen-buffer …
  - which is sent to the screen
- …also produces a depth-buffer
  - as a by-product!
  - not sent to the screen: it's an "offline" buffer
  - it's used during the rendering to determine occlusions and remove "hidden surfaces" (i.e., make what is behind something else is not seen, because it's occluded by that something)
  - see the Computer Graphics course for more details
- many rendering algorithms exploit the depth-buffer
  - for different uses
  - for each pixel on the screen, we have not only its RGB value, but its depth value (a scalar from 0 – close to the camera, to 1 – far from the camera)

a 2D array of RGB values of some resolution

a 2D array of depth values (scalars in 0 to 1) of the same resolution

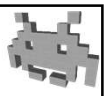
72

basics: Depth buffer



73

basics: Double Buffering



- To render a scene, all meshes are rendered succession
  - Filling the screen buffer
- Double-buffering is a basic technique to prevent any incomplete buffer to ever reach the screen
  - E.g., a rendering where some of the meshes is still not rendered
- How it works:
  - We have two RGB buffers: the front-buffer and the back-buffer
  - The front buffer shows the last complete rendering and is the one the screen shows
  - The back buffer is filled by the renderings, but it is not shown (it's yet another example of "off-screen buffer")
  - Screen Swap: When the back buffer is ready, the two buffer are swapped (instantaneously)
  - Info about variants: look up what "V-sync" means in 3D games settings
  - Observation: no need to double the depth-buffer

74

Some rendering techniques popular in games (we will not see how they are implemented)

- Double buffering (very basic, universally adopted)
- Shadowing
  - shadow mapping
  - Screen Space Ambient Occlusion
- Camera lens effects
  - Flares
  - limited Depth Of Field
- Motion Blur
- High Dynamic Range
- Non-Photorealistic Rendering
  - e.g., cell shading:
  - 1. contours
  - 2. lighting quantization
- Texture-for-geometry (beyond normal maps)
  - Parallax mapping

SSAO

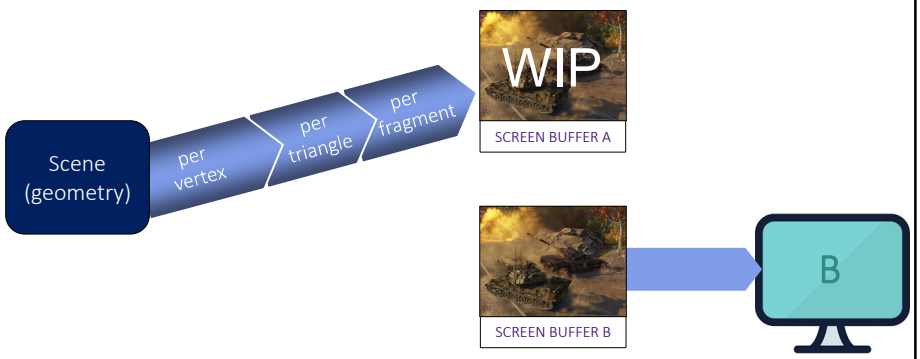
DoF

HDR

NPR

75

basics: Double Buffering

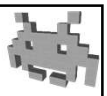


76

basics: Double Buffering

77

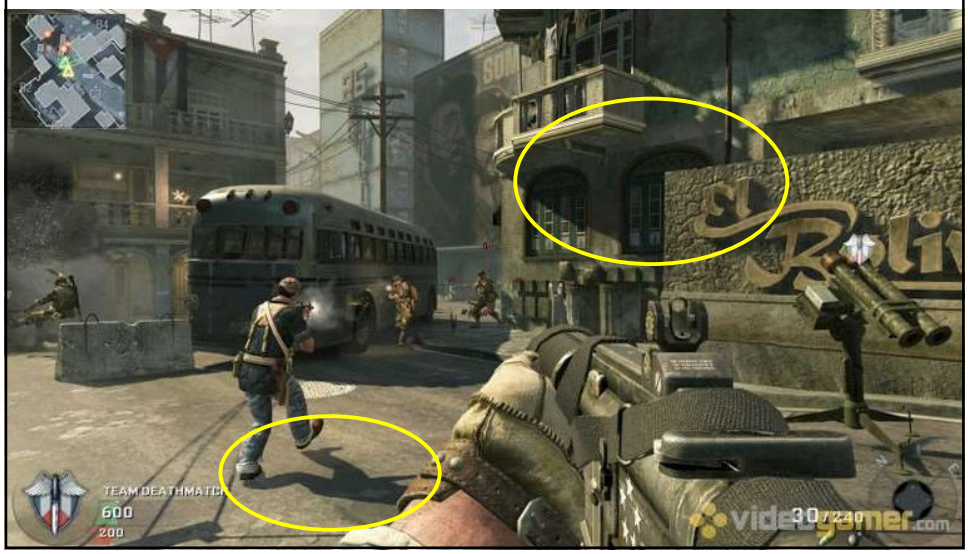
Multi-pass rendering techniques
(a wide class of rendering techniques)



- 1st pass: fill an internal buffer
  - an "off-screen" buffer
  - the output of this rendering, i.e. its "render target"
  - normally, the render target is the "screen buffer" (the buffer shown to the screen), but not in this case
- 2nd pass: fill the final screen buffer
  - using the just-computed off-screen buffer as a 2D texture
  - the 1st pass was… "rendering to texture"
- Note: good for GPU because…
  - the off-screen buffer is either only write-only (1st pass) or read-only (2nd pass). Never both!
  - the off-screen buffer is constructed and used in GPU RAM. No expensive swap of memory between CPU and GPU!

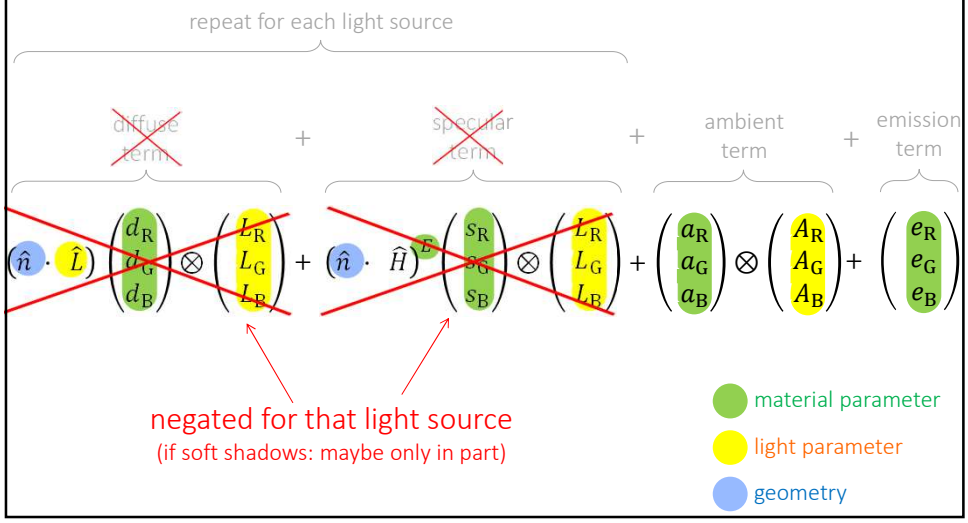
82

Shadow mapping



93

Shadow Mapping: effect of being in shadow



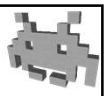
98

Shadow Mapping: effect of being in shadow

- Negates (zeroes) the light term of that (discrete) light-source (positional, directional, or spot- lights)
- Observe: the other lights are unaffected:
  - Other (non shadowed) positional / directional lights
  - Any ambient light
  - Also, the emission factor (if present)

99

Ambient occlusion (AO)



As we know…

- Cast shadows (computed by shadow-maps) negate the light coming from discrete light sources
- "Ambient occlusion", negates (occludes) the ambient component of lighting,
  - the AO is a factor (between 0 and 1) for each surface point
  - AO factor multiples the ambient component for that point
  - Intuitively, for a point **p**, its AO factor is a measure of how much **p** is exposed in the open
    - **p** is well exposed: AO ≈ 1.0
    - **p** is hidden, e.g. it is in the bottom of a crack: AO ≈ 0.0
  - Exact definition - not in this course. But keep in mind:
    - (1) it is an approximation
    - (2) it is a purely geometrical computation

100

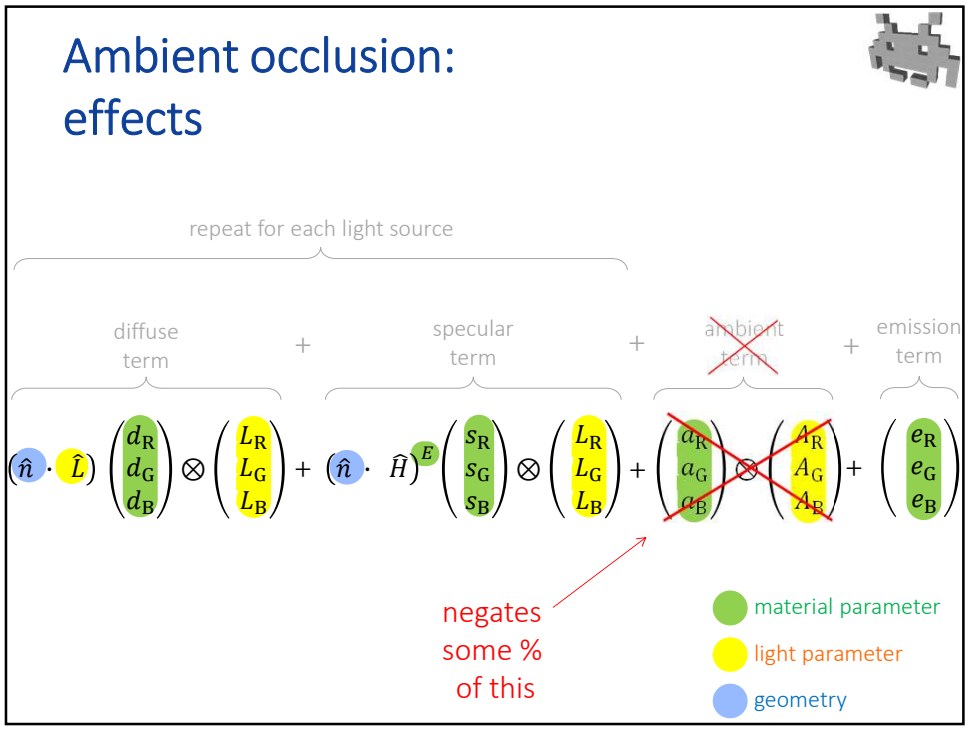
Two ways to compute AO: static AO versus SSAO

see lecture on textures

- Static Ambient Occlusion (or Baked AO)
  Baked in preprocessing on each mesh, in Object Space
  - Stored as a per-vertex attribute OR a texture (called "AO-map", or "light-map")
  - Pro: accurate & cheap (during rendering)
  - Con: static! Doesn't reflect current pos of the objects in the scene
- Screen Space Ambient Occlusion (SSAO)
  - It's a **screen-space** technique:
  - 1st pass: compute depth map (maybe normals too)
  - 2nd pass: compute AO map from the above (AO factor of each pixel, depends on neighboring depth values)
  - Final pass: use AO per-pixel from pass 2
  - Pro: dynamic! Reflect current position of objects in the scene
  - Con: less accurate
- The two can be combined!

101

Ambient occlusion: effects



103

Screen-Space techniques (in general)
(a class of multi-pass rendering techniques)

- 1st pass:
  - Render the scene from the same point of view as the final scene
  - Produce: final color buffer, plus a z-buffer (and/or other auxiliary buffers)
- 2nd pass:
  - render just one single "full screen" rectangle
  - (it fills the entire screens with two triangles)
  - for each produced fragment: apply 2D effects to the buffer
- Notes:
  - Basically, it's a way to apply "post-production" 2D image filters after the rendering.
  - Many of the techniques we will see are in this category

104

Screen Space Ambient Occlusion (SSAO)

SSAO only

105

Screen Space AO in a nutshell

- 1st pass: standard rendering
  - produces: rgb image
  - produces: depth image
- 2nd pass:
  screen space technique
  - for each pixel, look at its depth VS depths of its neighbors:
    - Neighbors are in front?
      difficult to reach pixel: low AO factor (closer to 0)
    - neighbors are behind?
      pixel exposed to ambient light: high AO factor (closer to 1)

108

(limited) Depth of Field

109

(limited) Depth of Field in a nutshell

- Screen space technique:
- 1st pass: standard rendering, producing
  - RGB image (kept off screen)
  - depth-buffer (as usual)
- 2nd pass:
  - pixel inside of focus range? Keep in focus
  - pixel outside of focus range? blur
    - Blur, way 1 = average with neighboring pixels
      kernel size ~= amount of blur
    - Blur, way 2 = compute MIP-map of RGB image,
      use lower MIP-map level with bilinear interpolation

110

HDR - High Dynamic Range (limited Dynamic Range)

111

HDR - High Dynamic Range in a nutshell

- Screen space technique:
- First pass: fill the off-screen buffer
  like a normal rendering,
  EXCEPT use lighting / materials value that are HDR
  - so, RGB of final pixel values not in [0..1]
  - e.g., sun *emits* light with RGB [15.0 , 15.0 , 15.0]:

>1 = "overexposed"!
i.e., "whiter than white"
(here: 15 times brighter
than the maximal screen brightness)

- Second pass:
  - Make values >1 *bleed* over neighboring pixels
  - i.e.: overexposed pixels lighten neighbors pixels
  - Result: halo effect

112

Motion Blur

113

Non-PhotoRealistic Rendering (NPR)

- Any rendering technique not aimed at realism
- Instead, the objective can be:
  - imitating a given style (imitative rendering), such as:
    - cartoons ("toon shading") ← most popular!
    - pen-and-ink drawings
    - pencil sketches
    - pixel art ← popular in nostalgic retro games (niche)
    - manga, comics, etc ← very common
    - pastels, oil paintings, crayons …
  - clarity/readability (illustrative rendering)
    - usually not for games

114

Toon shading / Cel Shading

(tweaked) Team Fortress II – Steam

116

Toon shading / Cel Shading in a nutshell

- Simulating “toons” / hand drawn effect
- At its basics, a combination of two effects:
  - addition contour lines
    - lines appearing at discontinuities of:
      1. depth,
      2. normals,
      3. materials
  - quantized lighting:
    - e.g., 2 or 3 tones: light, medium, dark instead of continuous shades
    - a simple variation of lighting equation: quantize its result

118

NPR rendering:
e.g.: simulated pixel art

img by Howard Day (2015)

119

Imitate “pixel art” (like an old, low res screen) in a nutshell

- Basics:
  - First pass:
    Normal rendering on a much smaller resolution
  - Second pass:
    Enlarge pixel on screen, no filters
- Extra flavors: in the first pass…
  (to make it look and feel like hand-made pixel arts)
  - Quantize rotations (of local or global transforms)
    (e.g.: express as Euler Angles, quantize angles)
  - Quantize frames
    (e.g.: in a skeletal animation, quantize time parameters, so to artificially limit the number of different in-betweens ever shown)
- More extra flavors: in the second pass…
  - Quantize colors (reduce number of different colors used)
  - Real time Dithering effects
  - Simulate CRT monitor effects (old monitor effects)

120