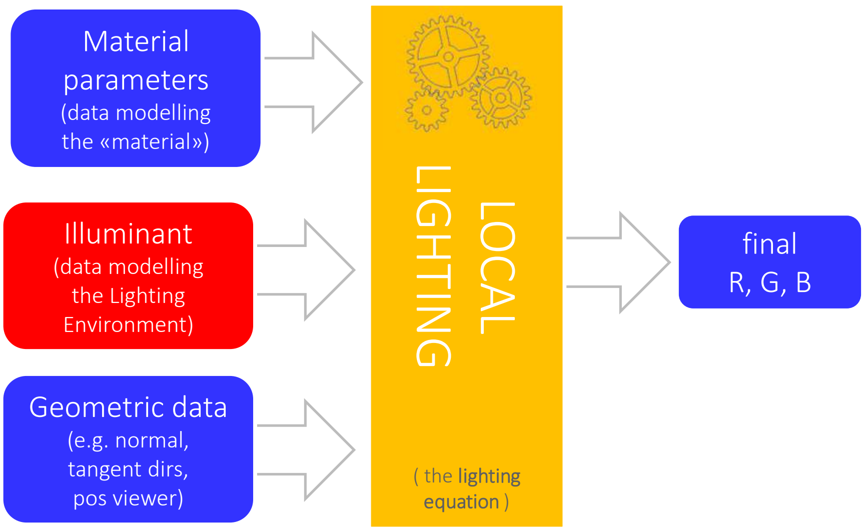
Course Plan

- 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. 8: Game 3D Animations
- lec. 9: Game 3D Audio
- lec. 10: Networking for 3D Games
- lec. 11: Artificial Intelligence for 3D Games
- lec. 12: Game 3D Rendering Techniques

47

Modelling the Light environment



48

Illumination environments: types

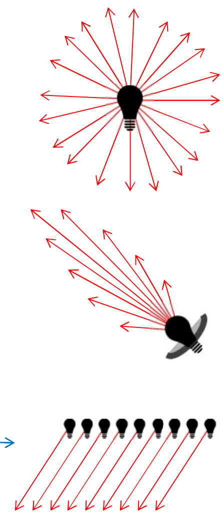
- Discrete
  - a finite set of individual light sources (plus a global ambient factor)
- Densely sampled
  - environment maps: textures sampling incoming light
- Basis functions
  - a spherical function stored as spherical harmonics coefficients

Also used jointly!

49

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



50

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 an 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)

51

Illumination environments: discrete

- Pros:
  - simple to position / reorient individual light sources
    - both at design phase, or dynamically (at game exec)
  - quite faithfully model of certain illuminants, e.g.
    - 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 discrete light requires extra processing … for each pixel!
    - therefore: hard limit on their number. Prioritize
    - therefore: are often given a (physically unjustified) radius of effect
  - the don't model well:
    - area light sources (e.g. from back-lit clouds)
    - reflections on (metal) objects

52

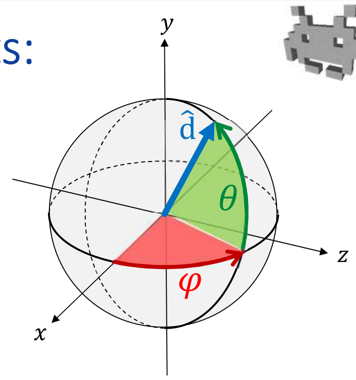
Illumination environments: densely sampled

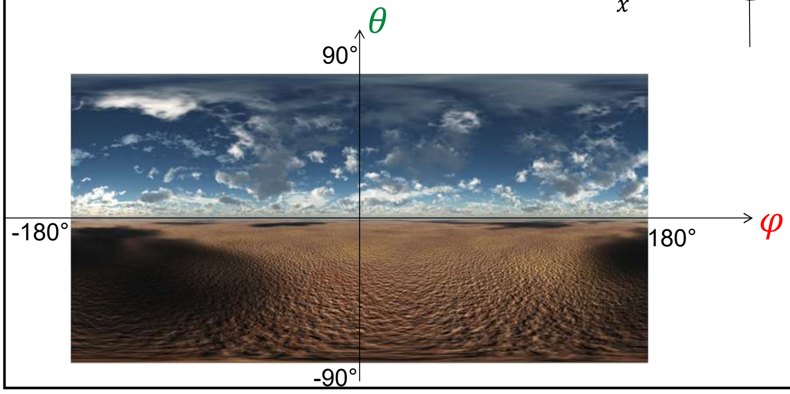
- A light intensity / color from each direction $\hat{d}$
- Asset to store that:
  "Environment map" texture

53

Illumination environments: densely sampled

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





54

Illumination environments: densely sampled

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

55

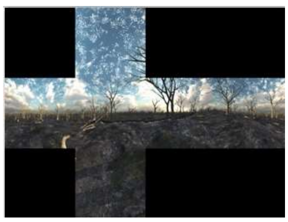
Illumination environments: densely sampled

unit vector

- Environment map: (asset)
  a texture with a texel $\mathbf{t}$ for each direction $\hat{\mathbf{d}}$
  - texel t stores the light coming from direction $\hat{\mathbf{d}}$
- Q: how to find 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!)

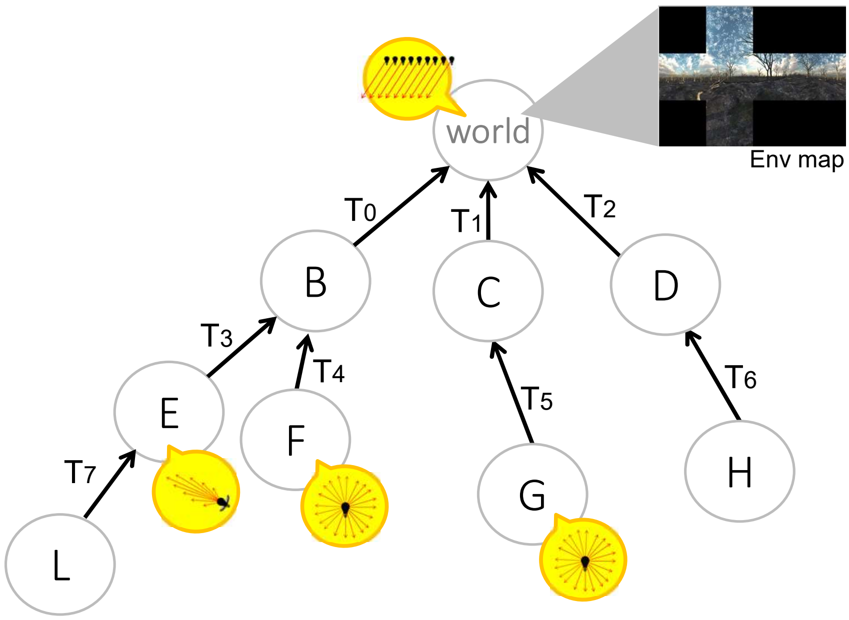
56

Environment map (asset)

- A texture with a texel $\mathbf{t}$ for each direction $\hat{\mathbf{d}}$
  - texel t stores the light coming from direction $\hat{\mathbf{d}}$
  - 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

57

Lighting env in the scene graph



58

Illumination environments: the Basis Functions way

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

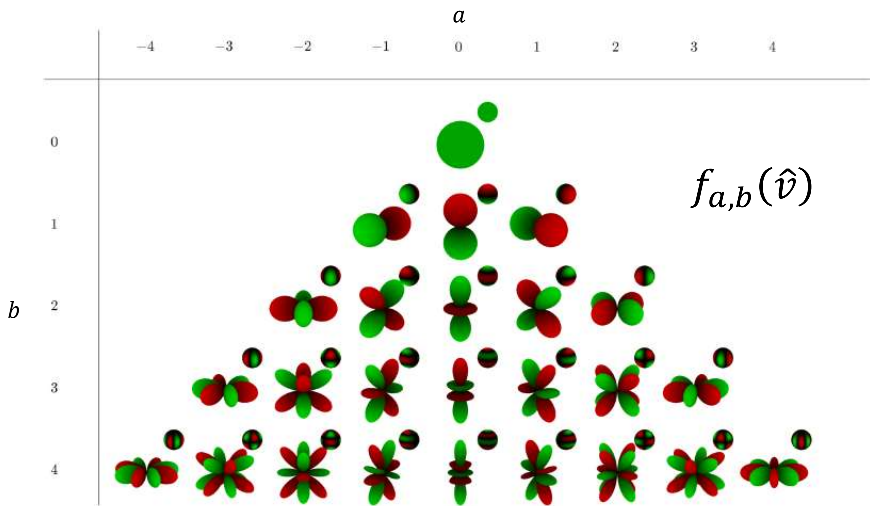
fixed spherical "basis" functions (always the same ones)

$$f(\hat{v}) = a_{0,0} \cdot f_{0,0}(\hat{v}) + a_{1,-1} \cdot f_{1,-1}(\hat{v}) + a_{1,0} \cdot f_{1,0}(\hat{v}) + a_{1,+1} \cdot f_{1,+1}(\hat{v}) + \cdots$$

a few scalar values to be stored, in order to model f

59

Illumination environments: with basis functions



60

Illumination environments: with basis functions

- Spherical Harmonics (SPH) in brief:
  - store Illumination Env as a small number (1,4,9,16…) of scalar weights of as many fixed spherical basis functions.
- Pros:
  - very compact
  - models continuous function well: smooth environments
  - allows for efficient computation of the Lighting equation
- Cons:
  - continuous functions *ONLY*
    - Bad for hi-freq details, e.g. no hard lights
    - not much variations (unless very many coefficient used)
- Often good for background lights

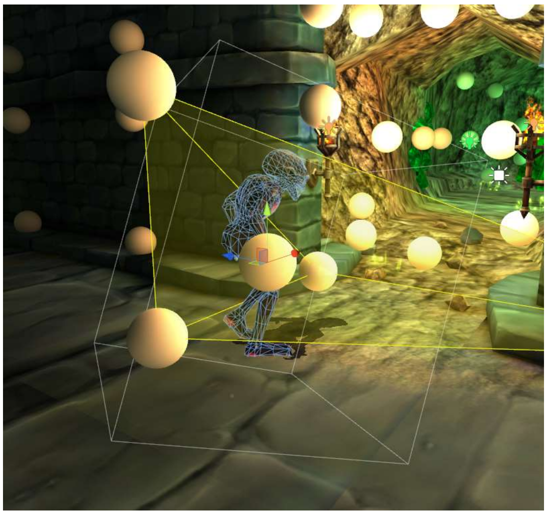
61

Light probes (position-dependent lighting env)

- A light probe == a (precomputed) lighting evn to be used near a given (xyz) position of the scene
- Light Probe lighting:
  - preprocessing: disseminate the scene with light probes
    - Store them as… low res environment maps
    - …or, with SPH (standard solution)
  - at rendering time, for a 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)
- Widely used !

62

Light probes (position-dependent lighting env)

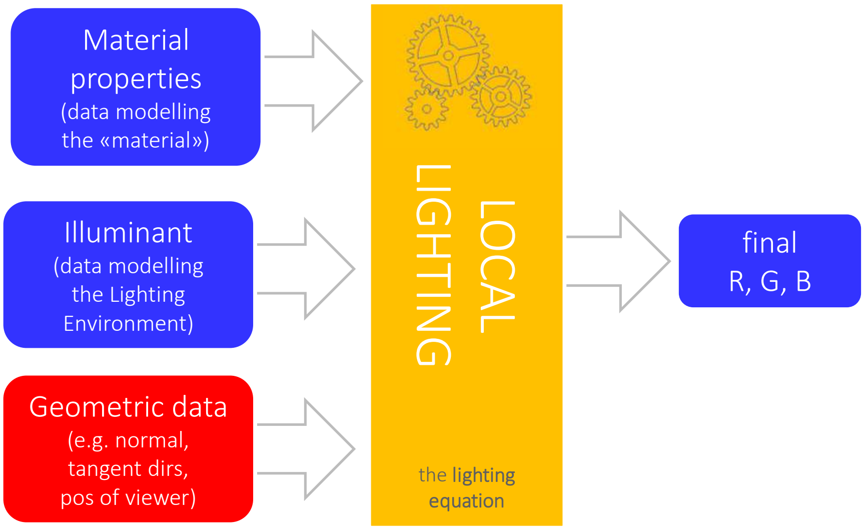


63

Light probes (position-dependent lighting env)

64

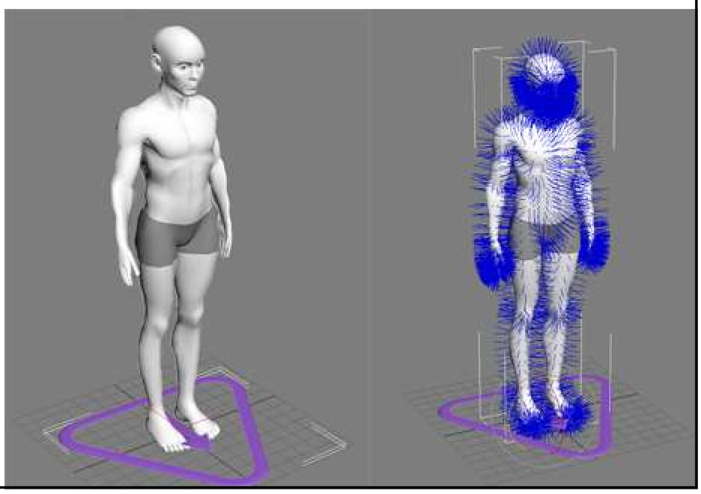
Local lighting in brief



65

Reminder: normals

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

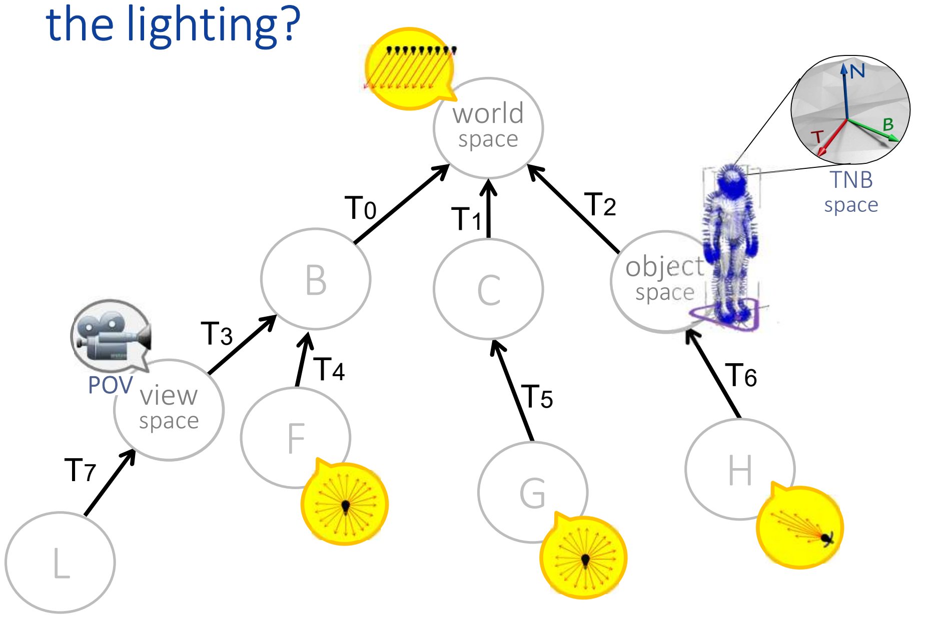


66

Reminder:
(per vertex) Tangent directions

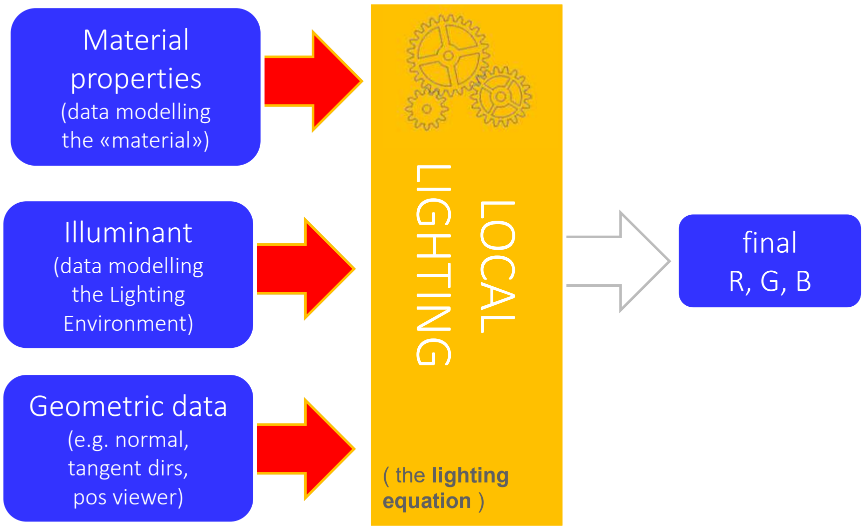
67

In which space to computue the lighting?



68

Local lighting in brief



69

Lighting equation: how

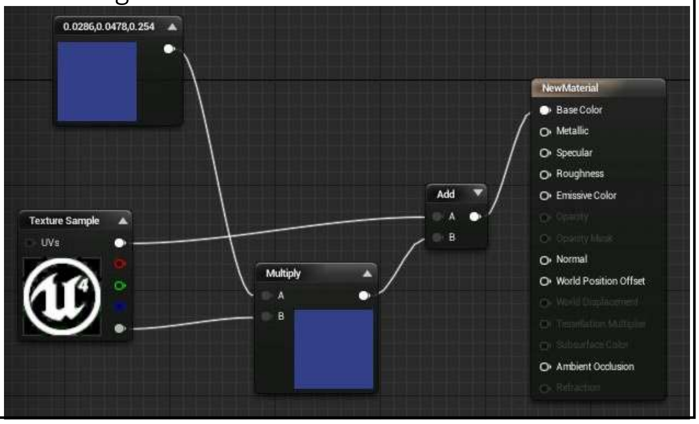
- Computed in the fragment shader
  - most game engines support a good set of choices
  - Custom new equations can be programmed in shaders
  - optimization: "lift" linear computations to the vertex shader
- Material + geometry parameters stored :
  - in textures *(for highest-frequency variations inside 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?

70

How to feed parameters to the lighting equation

- Hard wired choice of the game engine
  - but sometimes, a complex set of choices in the hand of the dev
- Specialized WYSIWYG game-tools not uncommon
  - E.g. in Unreal Engine 4:



71

Beyond local lighting

- Local lighting = only 3 things count:
  - light emitter(s) ← global variables / env textures
  - the *infinitesimal* part of surface hit by light, i.e.:
    - its local material (i.e. how does it bounces light) (aka: the BRDF)
    - its local shape
    - ← sampled from textures / interpolated from vertices / global variables
  - observer position ← global variables
- Anything else is part of Global lighting
  - The *rest of the scene* also affects the results
  - Global effects are considerably HARDER

72

Global lighting: two classes of approaches

- *Strategy 1:* use local lighting, but feed it a position-dependent lighting environment
  - *baked* (precomputed) i.e. in preprocessing
    good for static part scenes –
    problematic for dynamic scenes / lights
    usually too expensive for every frame
- *Strategy 2:* ad-hoc rendering techniques
  - basically, rendering algorithms that map well to existing HW pipeline
  - often, multi-pass techniques
  - see Part II of this lecture for a summarized list
- The two can be used jointly

73

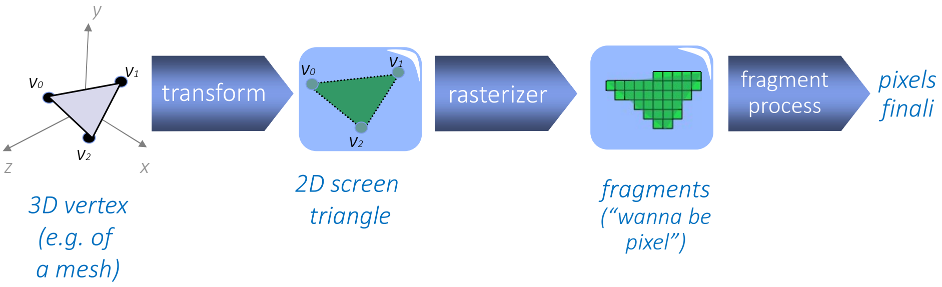
3D Videogames 2018/2019
Univ. degli Studi di Milano

Rendering in games

Part II: popular techniques in games

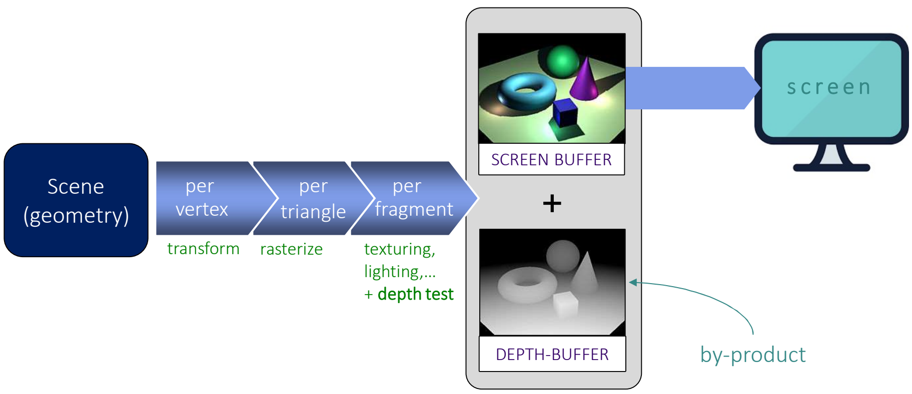
80

GPU pipeline – simplified even *more*



81

basics: Depth buffer



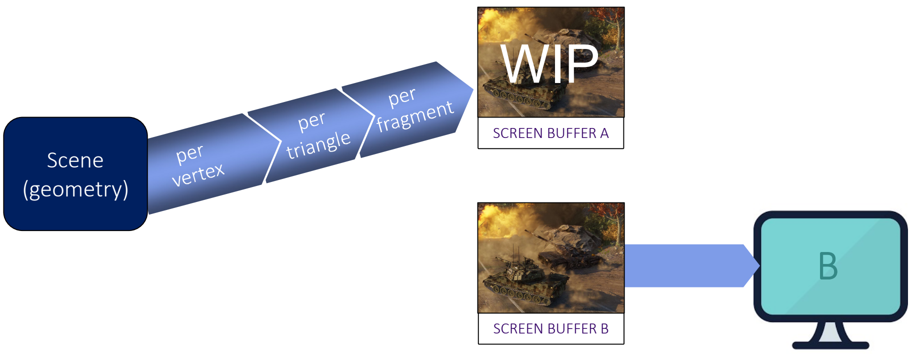
82

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
  - it's used during rendering to deterineocclusions (what covers what in a scene)
  - many algorithms exploit it that!

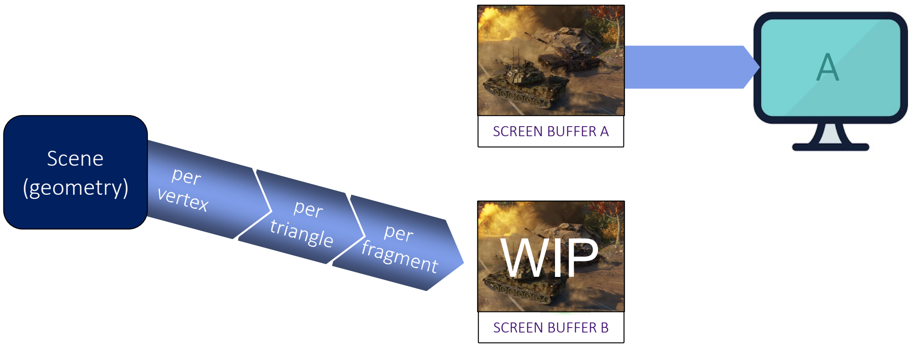
83

basics: Double Buffering



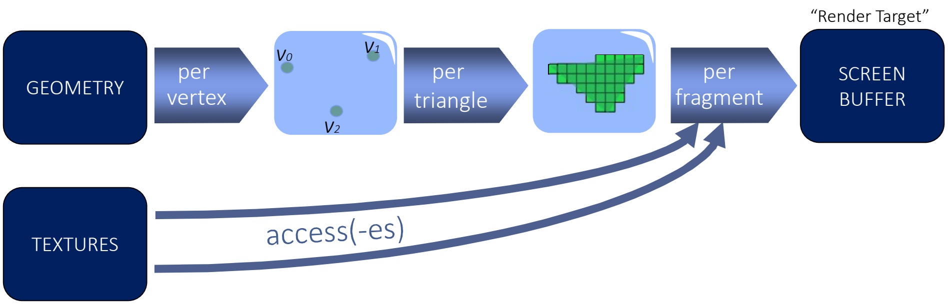
84

basics: Double Buffering



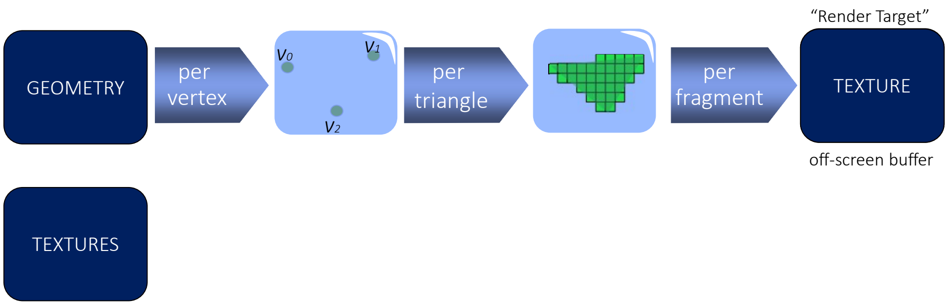
85

basics: Render to Texture



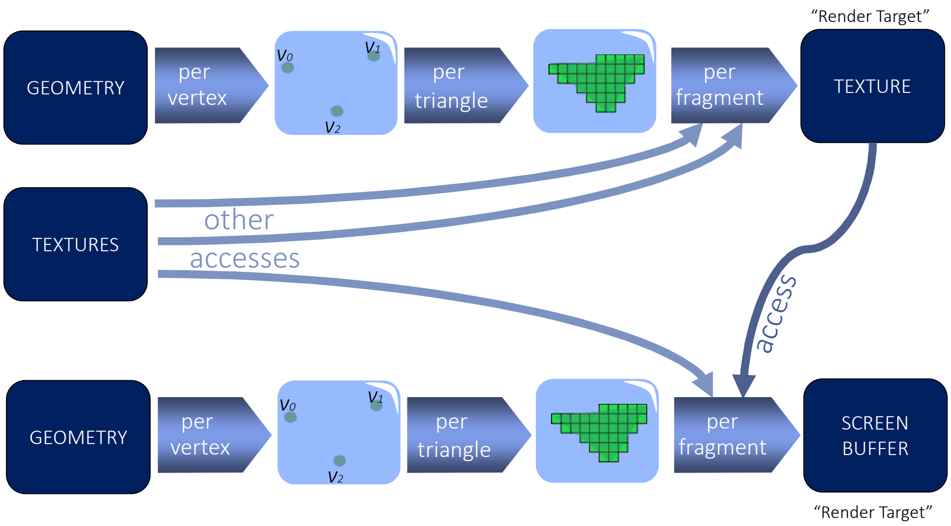
86

basics: Render to Texture



87

basics: Render to Texture



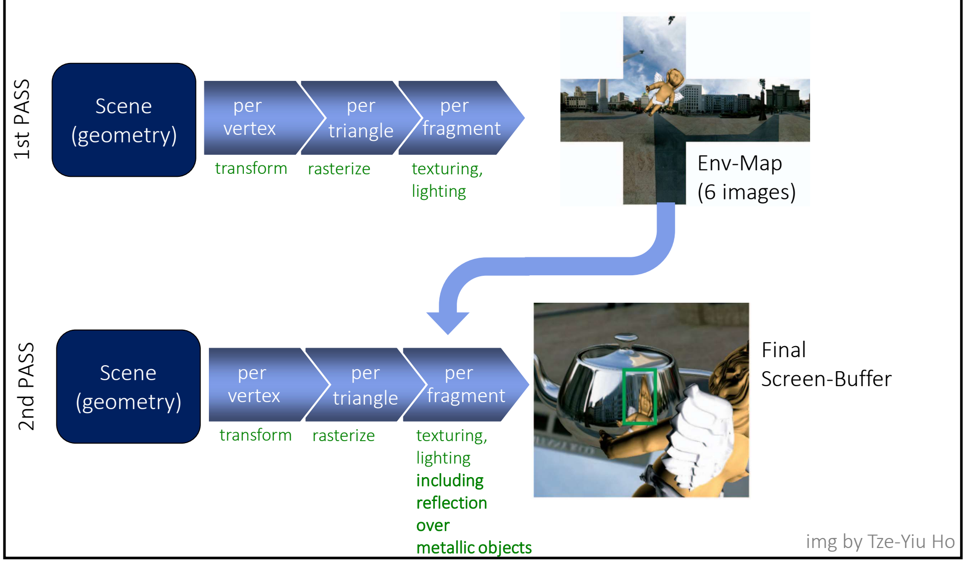
88

Multipass rendering techniques (general concept)

- 1st pass: fill an internal 2D buffer
  - i.e. An "off-screen" buffer (a buffer never shown to the user)
  - It's the output of this rendering, i.e.its "render target"
  - Normally, the render target is the "screen buffer" (buffer shown to the screen)
  - This technique is aka "render to texture"
- 2nd pass: fill the final screen buffer
  - Using the just-computed internal buffer as a 2D texture
- Note: efficient 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!

89

Example: metallic reflections of dynamic scenes



img by Tze-Yiu Ho

90

Main rendering algorithms: two classes of approaches

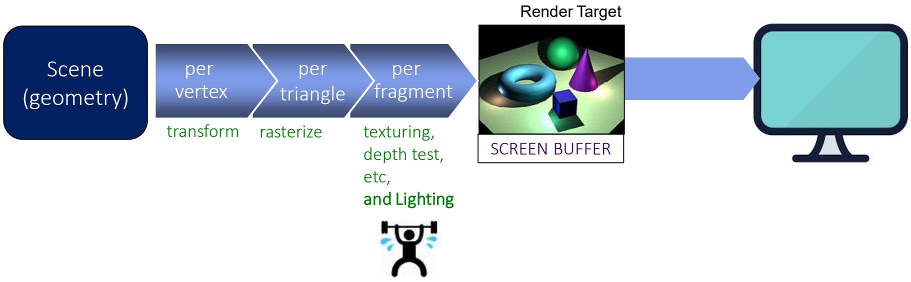
- Forward rendering
- Deferred shading ← aka Deferred lighting (actually, a variation) aka Deferred rendering (inappropriate?)

- Which approach to use?
  - Both are employed by games
  - Basilar choice! Implementation of all other rendering algorithms changes accordingly.

91

Main rendering algorithms: two classes of approaches

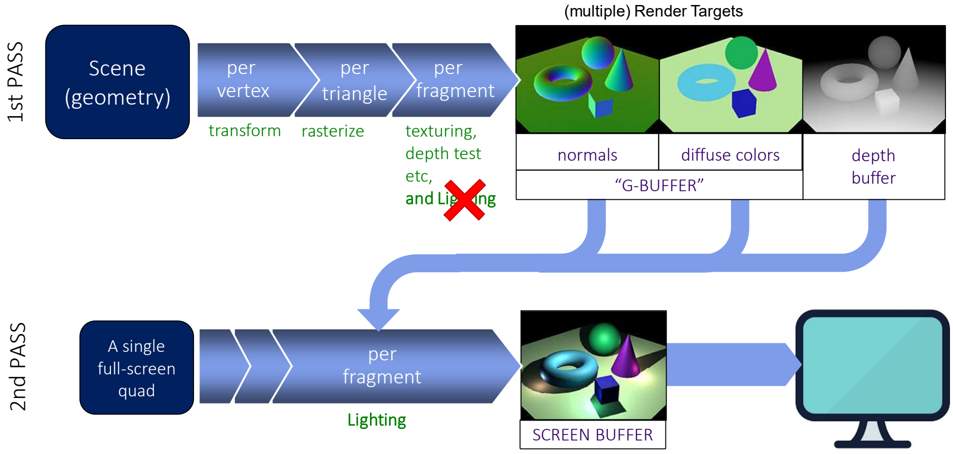
- Forward rendering



92

Main rendering algorithms: two classes of approaches

- Deferred shading ← aka Deferred lighting (actually, a variation)
  aka Deferred rendering (inappropriate?)



93

Deferred shading

- Advantage:
  lighting is computed only actually visible pixels
  - it's a huge saving if large depth complexity (aka overdraw) and/or lighting complexity – both common in 3D games
- Disadvantage:
  needs a separate buffer for every material parameter (or, sometimes, a material index)
  - Normal buffer
  - Depth bufferBase color buffer,
- Limits range of materials?
- Disadvantage: not very good with semi-transparency

94

Ad-hoc rendering techniques popular in games: a summary

- Shadowing
  - shadow mapping ← with PCF
  - Screen Space Ambient Occlusion ← SSAO
- Camera lens effects
  - Flares
  - limited Depth Of Field ← DoF
- Motion blur
- High Dynamic Range ← HDR
- Non Photorealistic Rendering ← NPR
  - contours
  - lighting quantization
- Texture-for-geometry
  - Bumpmapping
  - Parallax mapping

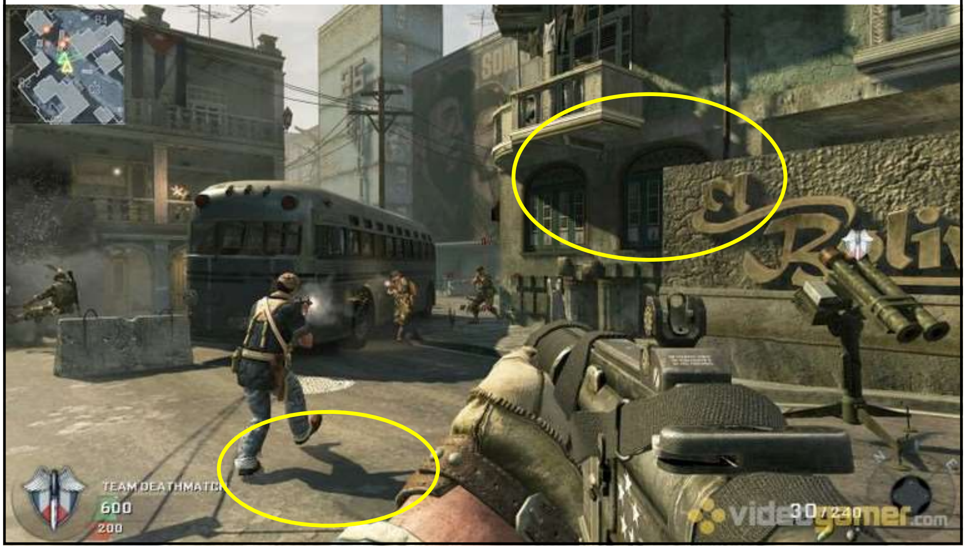
95

Screen-Space techniques (in general)
(a class of multi-pass 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 buffer)
- 2nd pass:
  - render just one single “full screen” rectangle
  - (it filling the entire screens with two triangles)
  - for each produced fragment: apply 2D effects to the buffer
- Notes:
  - Basically, apply image filters to the rendering.
  - Many of the techniques in the previous slides are like this

97

Shadow mapping



99

Shadow mapping

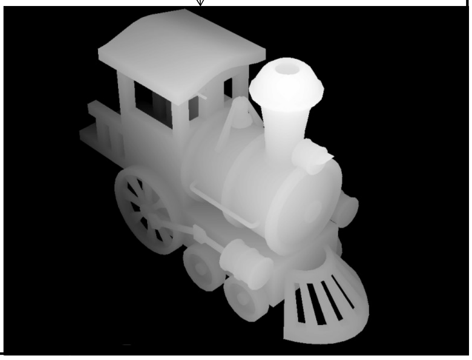
100

Shadow-mapping in a nutshell (a multi-pass technique for shadows)

1st pass:
- camera in light position
- render all light blockers
- produce a depth buffer *only* (known as the shadow map)
- (repeat for each discrete light casting a shadow)

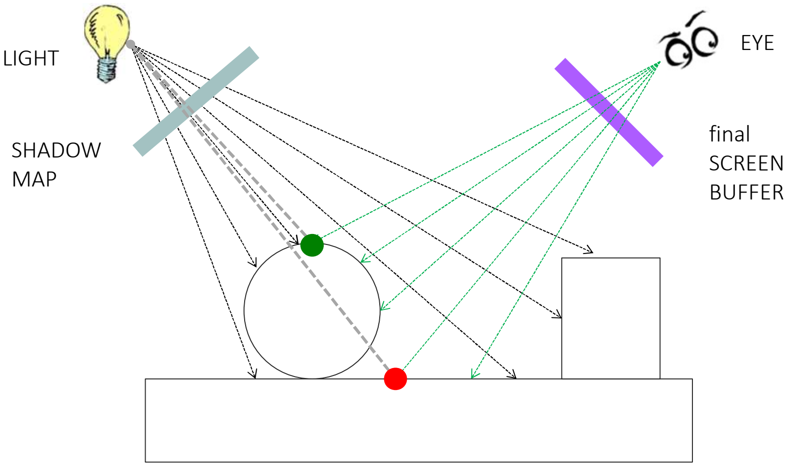
2nd pass:
- camera in final position
- for each fragment, access the shadow-map, determine if that if fragment is visible by light (or not)
- If not visible, negate contribution of that discrete light source

- Result:
  - Blockers cast ashadow



101

Shadow-mapping concept



102

Shadow mapping: issues

- Rendering shadow-map:
  - Must be redone every time object move
  - can be baked once and for all, for static objects only
  - (jet another reason to label static objects!)
- Shadow-map resolution:
  - it matters! aliasing effects
  - remedies: PCF, multi-res shadow-map

optional topics (no exam)

103

Shadow Mapping: results

- Negates (zeroes) the light term of discrete light-sources
- Other light components are still summed together…
  - Non blocked lights
  - Ambient factor
  - Background illumination (e.g. from light probes)

104

Screen Space AO (SSAO)

SSAO only

105

Ambient occlusion (AO)

- Cast shadows (computed by shadow-maps) negate the light coming from discrete light sources
- "Ambient occlusion", negates (occludes) the "ambient" component of lighting, instead
- Idea:
  - 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

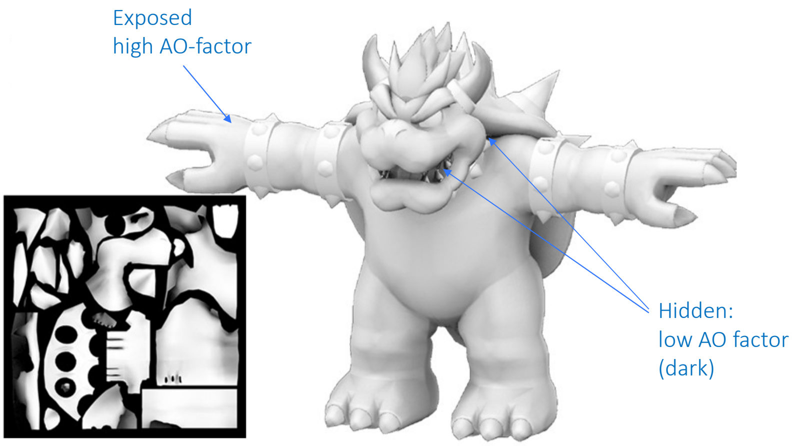
106

Two ways to compute AO: OSAO versus SSAO

- Object Space Ambient Occlusion (OSAO)
  - Baked in preprocessing on each mesh
  - Stored as a per-vertex attribute OR a texture ("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)
  - Screen space technique
  - 1st pass: compute depth map (maybe normal 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
- Can be combined!

107

Baking AO over a mesh (OSAO)



Baked AO map

108

No SSAO

109

With SSAO

110

Screen Space AO in a nutshell

- First pass: standard rendering
  - produces: rgb image
  - produces: depth image
- Second pass:
  screen space technique
  - for each pixel, look at depth VS its neighbors:
    - neighbors in front?
      difficult to reach pixel: darken ambient
    - neighbors behind?
      pixel exposed to ambient light: keep it lit

111

(limited) Depth of Field

112

(limited) Depth of Field in a nutshell

- Screen space technique:
- 1st pass: standard rendering, producing
  - RGB image
  - Z-buffer
- Second pass:
  - pixel inside of focus range? Keep in focus
  - pixel outside of focus range? blur
    - Blur, way 1 = average with neighbors pixels
      kernel size ~= amount of blur
    - Blur, way 2 = compute MIP-map of RGB image,
      use lower MIP-map level with bilinear interpolation

113

HDR - High Dynamic Range (limited Dynamic Range)

114

HDR - High Dynamic Range in a nutshell

- Screen space technique:
- First pass: like a normal rendering, BUT use lighting / materials with any values
  - RGB of final pixel values not in [0..1]
  - e.g. sun emits light with RGB [10.0,10.0,10.0]:
  - If >1 = “overexposed”! That is, “whiter than white”
- Second pass:
  - Make values >1 bleed over other pixels
  - i.e.: overexposed pixels lighten neighbors

115

Parallax mapping: in a nutshell

- Texture-for-geometry technique
- Texture used:
  - displacement maps
  - color / rgb map

116

Parallax Mapping

117

Parallax Mapping

118

Motion Blur

119

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, or, western comics ← not uncommon
    - pastels, oil paintings, crayons …
  - clarity/readability (illustrative rendering)
    - usually not for games

120

Toon shading / Cel Shading

121

Toon shading / Cel Shading

(tweaked) Team Fortress II – Steam

122

Toon shading / Cell Shading in a nutshell

- Simulating "toons"
- Two effects:
  - add contour lines
    - lines appearing at discontinuities of:
      1. depth,
      2. normals,
      3. materials
  - quantize lighting:
    - e.g. 2 or 3 tones: light, medium, dark instead of continuous
    - simple variation of lighting equation

123

NPR rendering: e.g.: simulated pixel art

img by Howard Day (2015)

124