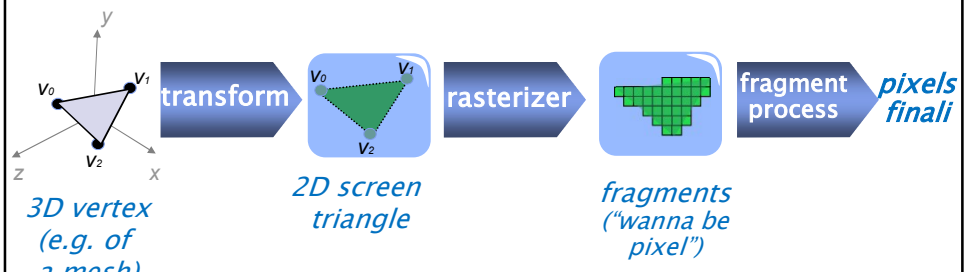
3D Videogames 2018/2019
Univ. degli Studi di Milano

Rendering in games

Part II: popular techniques in games

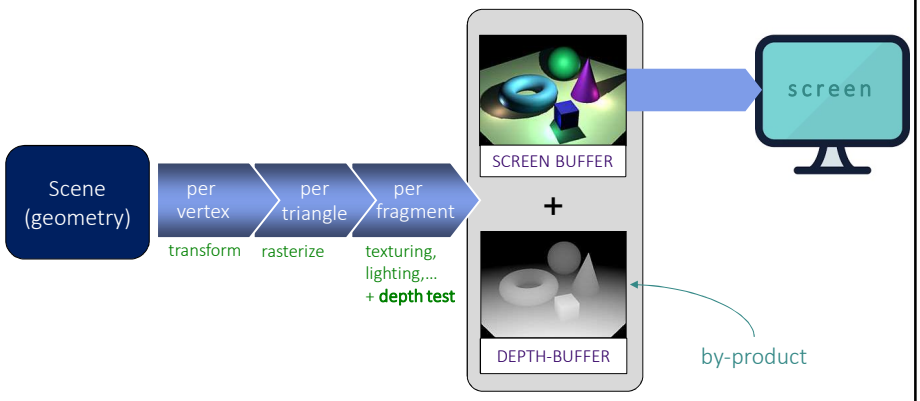
69

GPU pipeline – simplified even *more*



70

basics: Depth buffer



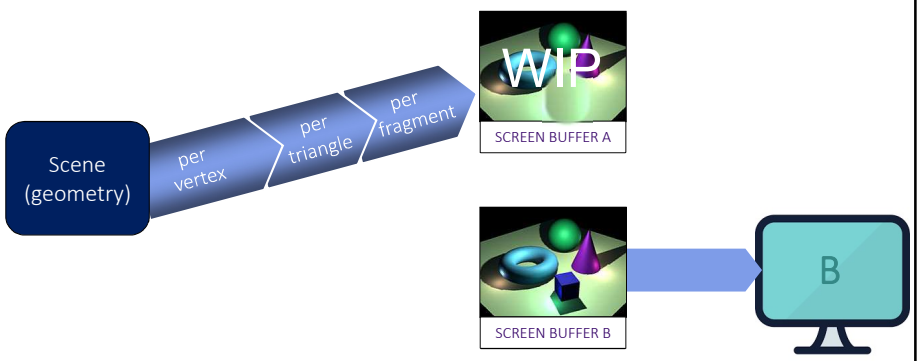
71

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

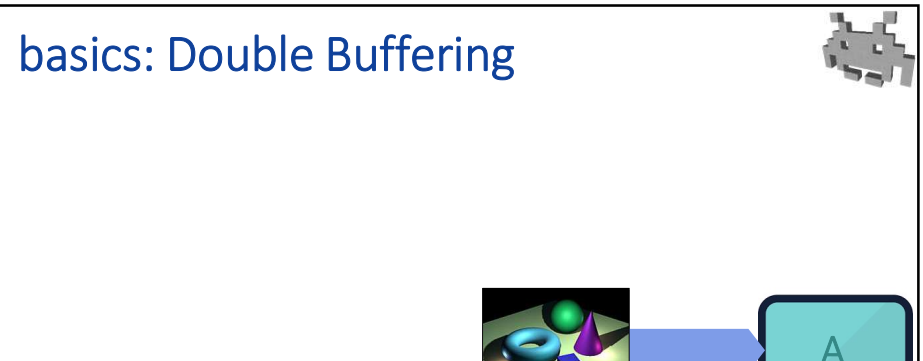
72

basics: Double Buffering

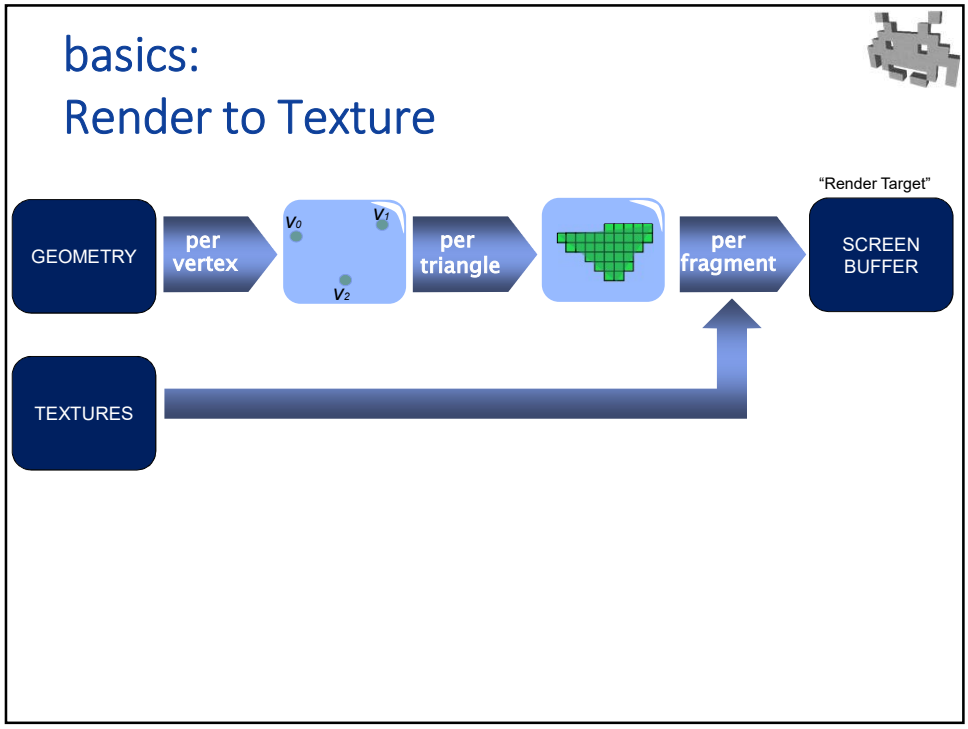


73

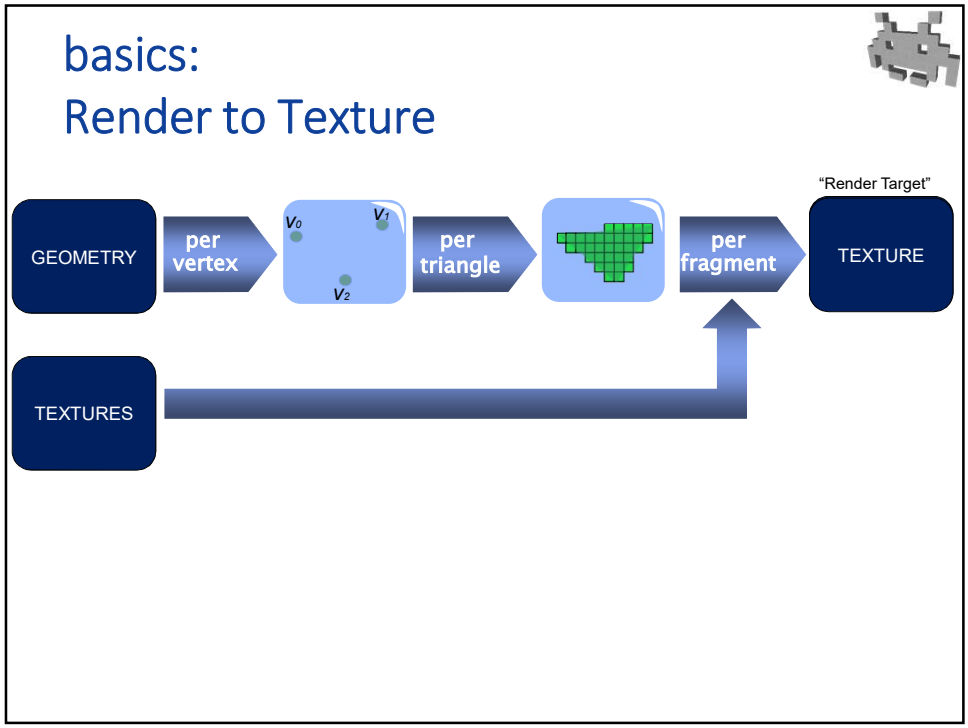
basics: Double Buffering



74

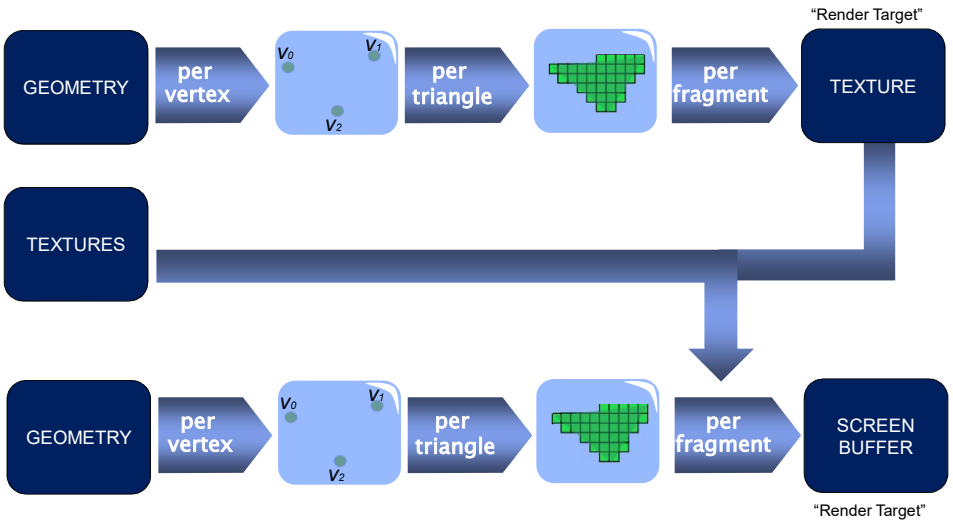

basics:
Render to Texture
"Render Target"
GEOMETRY
per vertex
v_0
v_1
v_2
per triangle
per fragment
SCREEN BUFFER
TEXTURES

75


basics:
Render to Texture
"Render Target"
GEOMETRY
per vertex
v_0
v_1
v_2
per triangle
per fragment
TEXTURE
TEXTURES

76

basics: Render to Texture



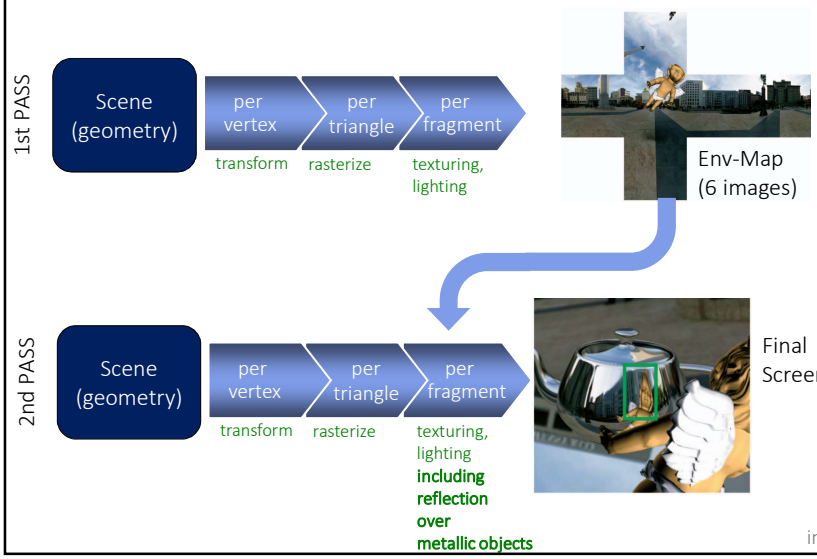
77

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 is also known as "render to texture"
- 2st pass: fill the actual screen buffer
  - Using the just-computed internal buffer as a 2D texture
- Notes:
  - the off-screen buffer is either only written (1st pass) or read-only (2nd pass). Not both at the same time! Good for efficiency.
  - the off-screen is constructed and used in GPU RAM. No expensive swap of memory between CPU and GPU

78

Example: metallic reflections of dynamic scenes



img by Tze-Yiu Ho

79

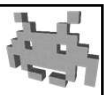
Main rendering algorithms: two classes of approaches

- Forward rendering
- Deferred shading

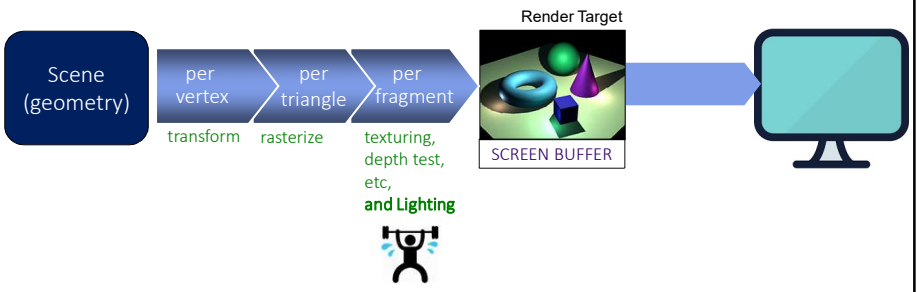
aka Deferred lighting (actually, a variation)
aka Deferred rendering (inappropriate?)

80

Main rendering algorithms: two classes of approaches

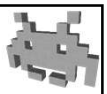


- Forward rendering

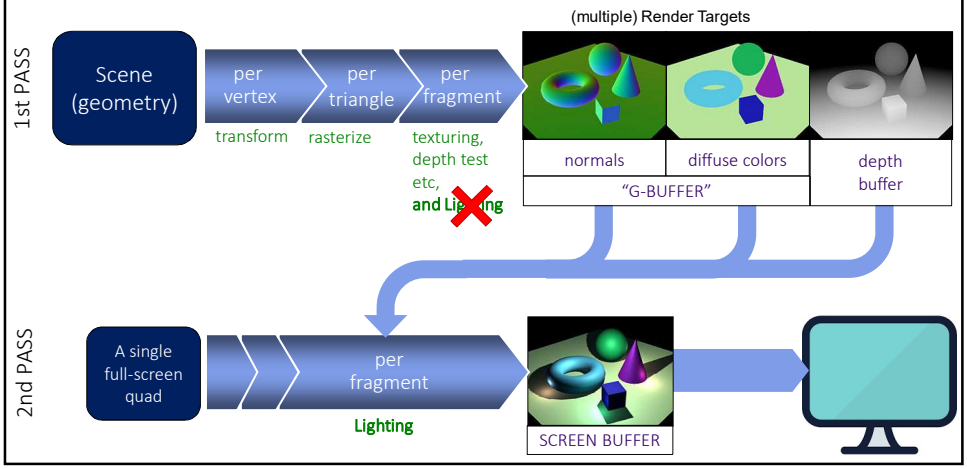


81

Main rendering algorithms: two classes of approaches

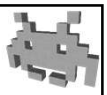


- Deferred shading ← aka Deferred lighting (actually, a variation)
  aka Deferred rendering (inappropriate?)



82

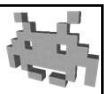
Main rendering algorithms: two classes of approaches



- Deferred shading
  - Advantage: lighting is computed only for what is actually visible (huge saving if large depth complexity and lighting complexity – safe assumptions is games)
  - Disadvantage: needs a separate buffer for every material parameter (sometimes, a material index)
  - Limits range of materials?
  - Disadvantage: not very good with semi-transparency
- Which approach to use?
  - Both are employed by games
  - Basilar choice! Implementation of all other rendering algorithms changes accordingly.

83

Ad-hoc rendering techniques popular in games: a summary



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

with PCF

SSAO

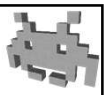
DoF

HDR

NPR

84

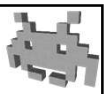
Screen-space techniques in general
(a class of multi-pass techniques)

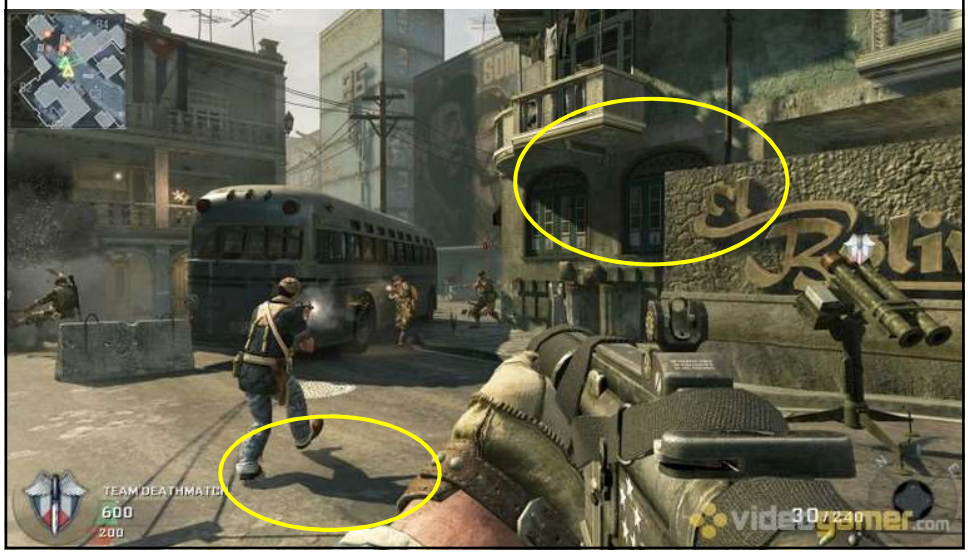


- 1st pass:
  - Render the scene from the same point of view as the final scene
  - Produce: final color buffer, plus a z-buffer (and/or other auxiliary buffer)
- 2nd pass:
  - render just one single “full screen” rectangle
  - (it filling the entire screens with two triangles)
  - for each produced 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

86

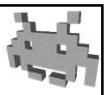
Shadow mapping





88

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



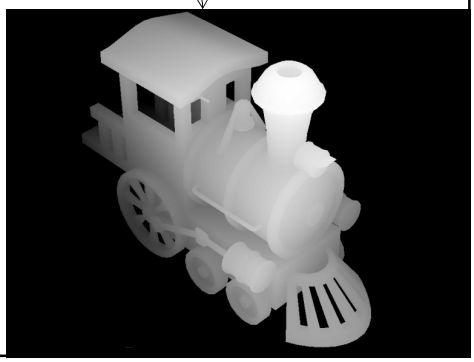
1st pass:

- 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:

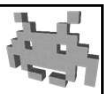
- camera in final position
- for each fragment, access the shadow-map, determine if that if fragment is visible by light (or not)
- If not visible, negate contribution of that discrete light source

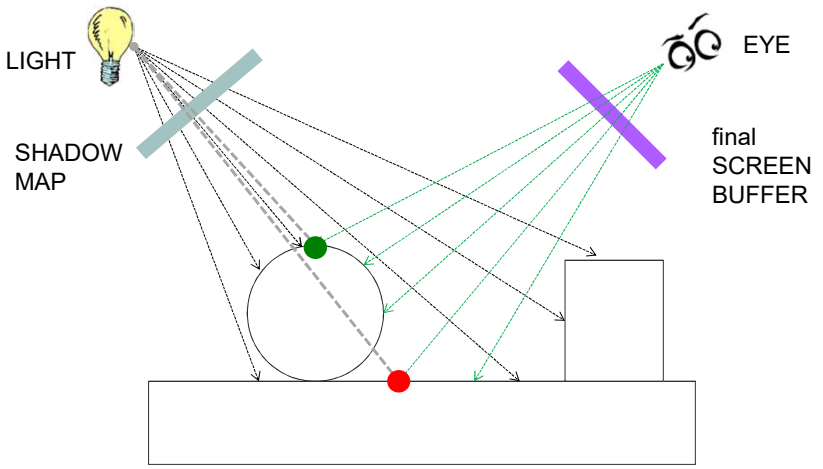
- Result:
  - Blockers cast ashadow



90

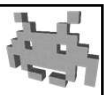
Shadow mapping in a nutshell





91

Shadow Mapping: issues

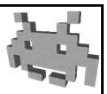


- Rendering shadow-map:
  - Must be redone every time object move
  - can be baked once and for all, for static objects only
  - (jet another reason to label static objects!)
- Shadow-map resolution:
  - it matters! aliasing effects
  - remedies: PCF, multi-res shadow-map

optional topics (no exam)

92

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)

93

Screen Space AO (SSAO)

SSAO only

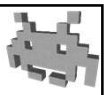
94

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

95

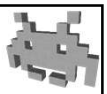
Two ways to compute AO: OSAO versus SSAO

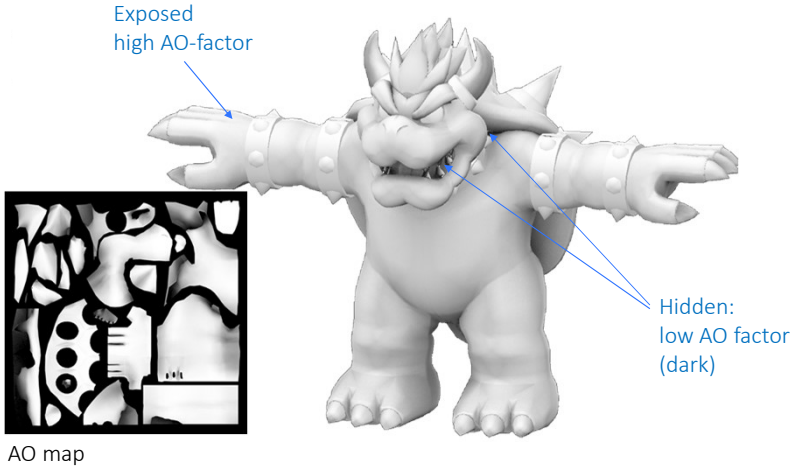


- Object Space Ambient Occlusion (OSAO)
  - Baked in preprocessing on each mesh
  - Stored as a per-vertex attribute OR a texture ("AO-map", or "light-map")
  - Pro: accurate & cheap (during rendering)
  - Con: static! Doesn't reflect current pos of the objects in the scene
- Screen Space Ambient Occlusion (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!

96

Baking AO over a mesh (OSAO)





97

With SSAO

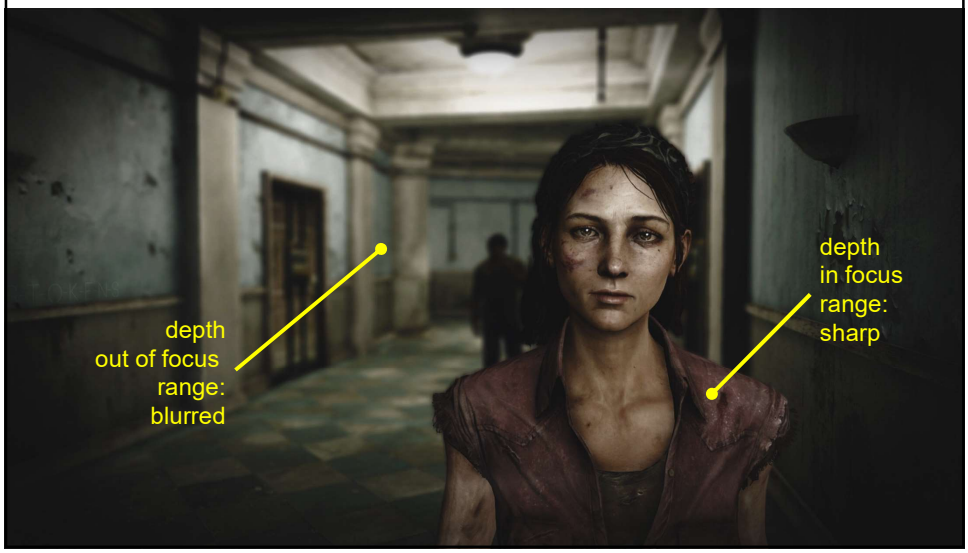
99

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

100

(limited)
Depth of Field



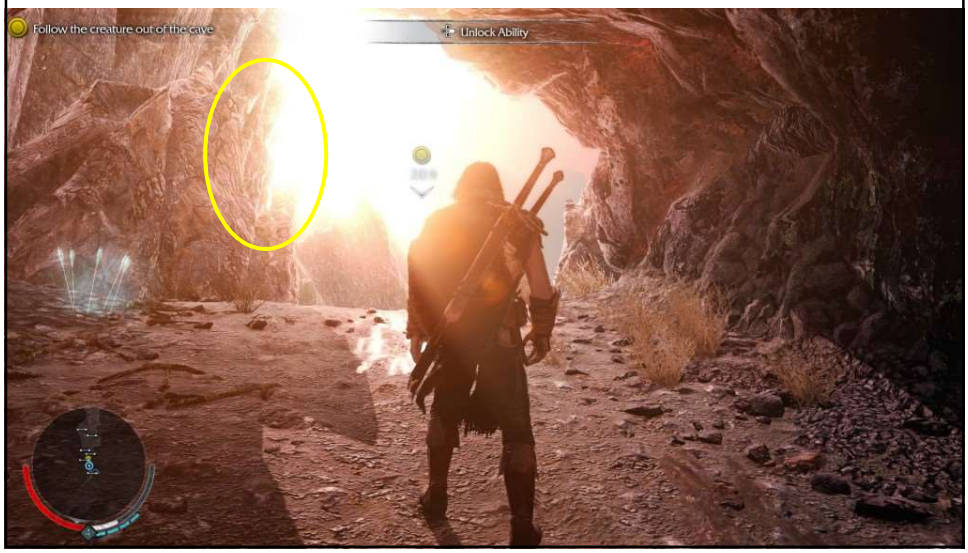
101

(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

102

HDR - High Dynamic Range (limited Dynamic Range)



103

HDR - High Dynamic Range in a nutshell

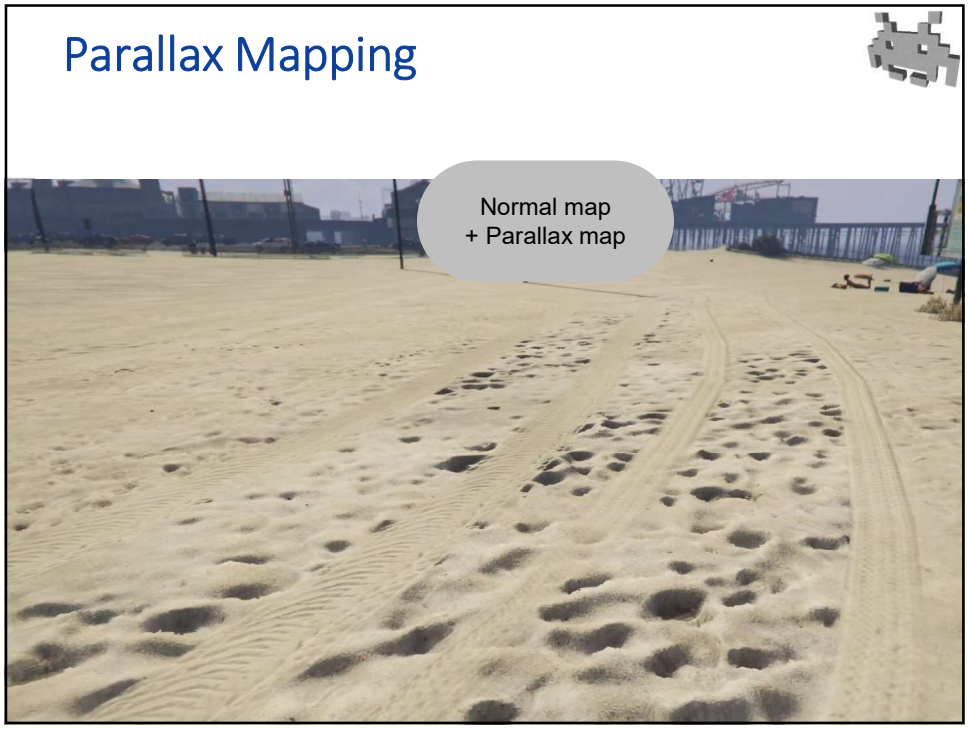
- Screen space technique:
- First pass: like a normal rendering, BUT use lighting / materials with any values
  - RGB of final pixel values not in [0..1]
  - e.g. sun emits light with RGB [10.0,10.0,10.0]:
  - If >1 = “overexposed”! E.g. “whiter than white”
- Second pass:
  - Make values >1 bleed over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

104

Parallax Mapping

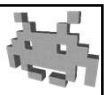
105

Parallax Mapping



106

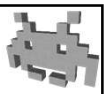
Parallax mapping: in a nutshell



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

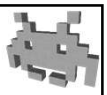
107

Motion Blur



108

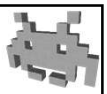
Non-PhotoRealistic Rendering (NPR)



- 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

109

Toon shading / Cel Shading



110

Toon shading / Cel Shading

(tweaked) Team Fortress II – Steam

111

Toon shading / Cel 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

112

NPR rendering:
e.g.: simulated pixel art

img by Howard Day (2015)

113

NPR rendering:
simulated pixel art

img by Lucas Pope

114