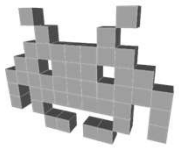
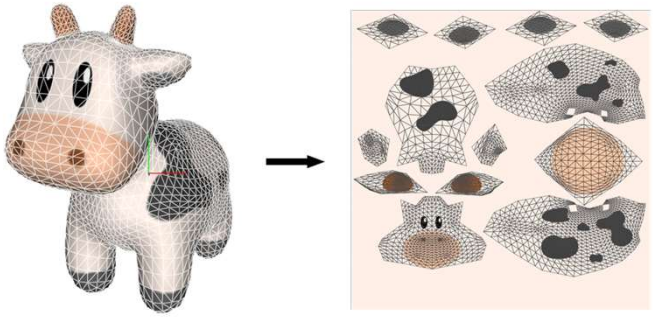
3D VideoGames

Textures in 3D Games

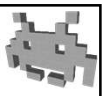


Marco Tarini



1

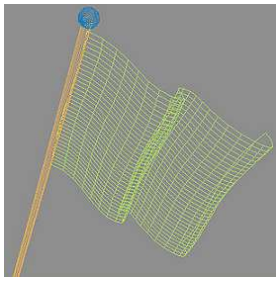
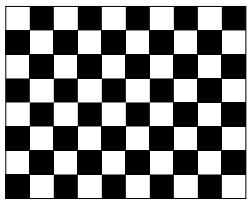
Course Plan



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**
★ appearance
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

2

Texture mapping (a process)


+

=

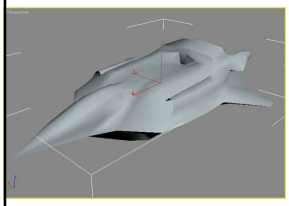
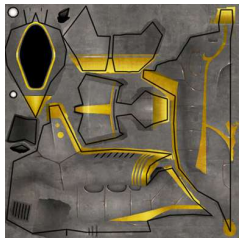
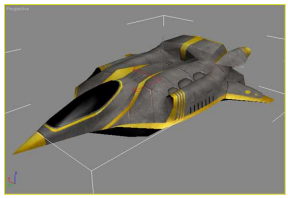

mesh

texture-map
(2D RGB image)
(here: a color-map)

result on screen

3

Texture mapping


+

=


mesh

texture-map
(2D RGB image)

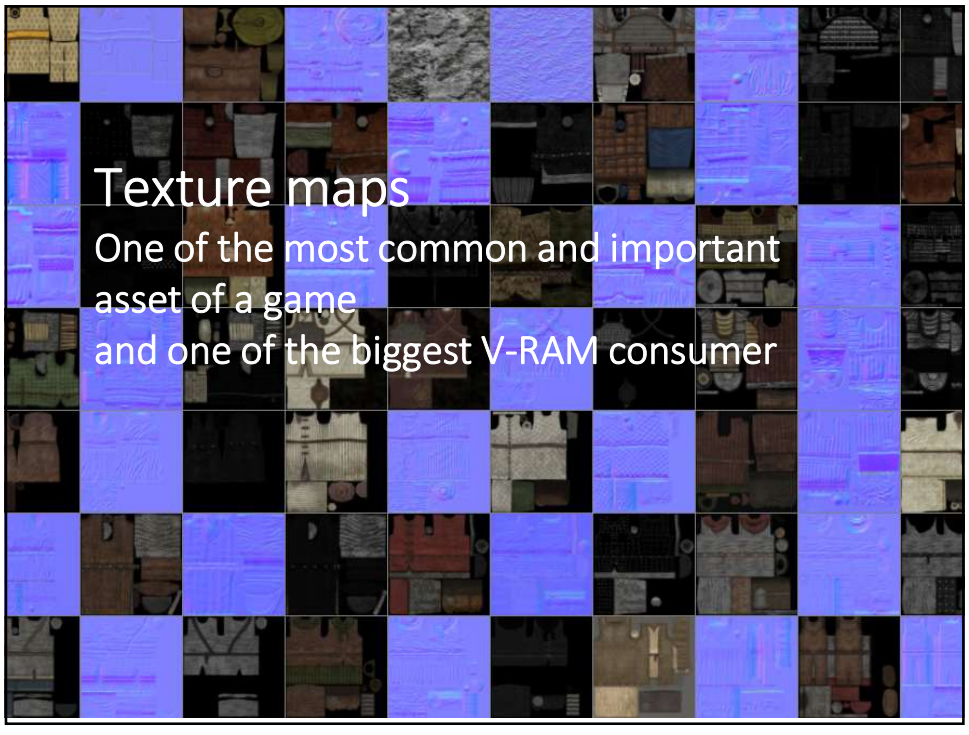
(here: a color-map)

result on screen

4

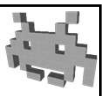
Example (color-maps)

5

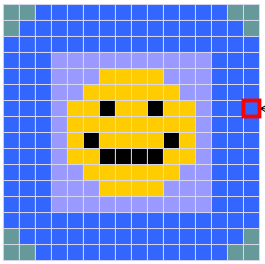


6

Texture maps: data structures

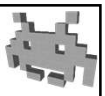


- In practice, a **rasterized image**



7

Textures (or texture maps)



- Multiple texture sheets (a raster image of texels)
- each defines a signal over the mesh
  - Similar purpose as per-vertex attributes!
  - but…
    - # texels >> # vertices
    - More complex signals!

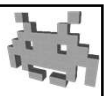
Texture: regular sampling, and dense

Attributes: irregular samplling (can be adaptive), and sparse

- A texel = a sample of that signal
  - Between samples: (bilinear) interpolation
- Signal sampling:
  - On a regular 2D grid (raster image)
  - At a given fixed resolution (NOT adaptive!)

8

GPU rendering of a Mesh in a nutshell (reminder)



- Load…
  - store all data on VRAM
    - Geometry + Attributes
    - Connectivity
    - Textures
    - Shaders
    - Uniform Parameters
    - Settings / flags

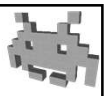
THE MESH ASSET

THE "MATERIAL" ASSET

- …and Fire!
  - issue the draw call (the command: "*do it*")!

9

Signals typically stored in textures (in videogames) 1/4

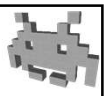


Material parameters (of a non-uniform material), such as:

- Each texel = a base-color (channels: *r*,*g*,*b*)
  - The texture sheet is a "diffuse-map" / "color-map" / "RGB-map"
- Each texel = a transparency factor (channels: α)
  - The texture sheet is a "alpha-map" or "cutout-texture" (exp. if 1bit)
- Each texel = a specular color value
  - The texture sheet is a "specular-map"
- Each texel = a glossiness value / specular exponent
  - The texture sheet is a "glossiness-map"
- Each texel = a metalness value
  - The texture sheet is a "metalness-map" or "metallicity-map"
- Each texel = a roughness value
  - The texture sheet is a "roughness-map"

10

Signals typically stored in textures (in videogames) 2/4



Custom values (rarely): user defined textures

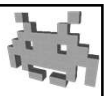
- How wet,
  or vulnerable,
  or snow covered, etc
  or blood splattered,
  each point on the surface is

See later for the rest of this list (parts 3/4, 4/4)!

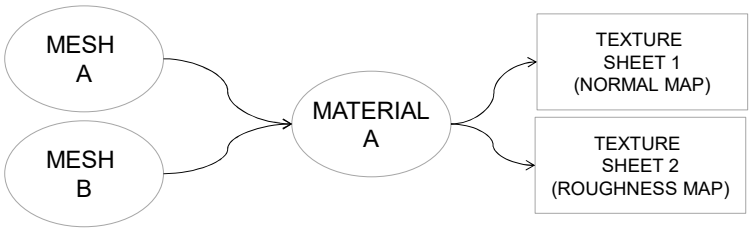
note: multiple channels of the same texture sheet can store different signals – e.g.
1st channel = roughness
2nd channel = specular

11

Texture maps assets and Mesh assets

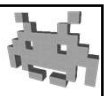


- Not necessarly 1:1
  - 1:N -- several textures «sheets» associated to a mesh
  - N:1 – more meshes on the same sheet (good)



13

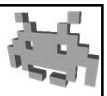
Texture maps assets and Mesh assets



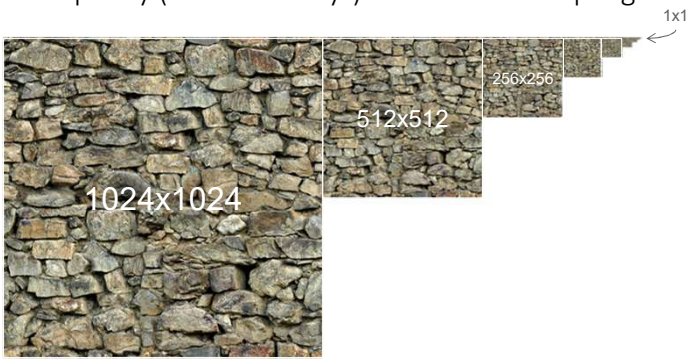
- Typical mechanism:
  - each mesh is associated to one material asset
  - a material asset is associated to multiple sheets, for example:
    - 1st sheet: diffuse-map
    - 2nd sheet: bumpmap
    - 3rd sheet: alphamap
- Note: more meshes can share the same material
  - convenient! Saves V-RAM
- what if different parts of the same object must be associated to different set of textures?
  - decompose the object into sub-mesh
  - associate each with a different material

14

MIP-map levels
(from latin: *Multum in Parvo*)

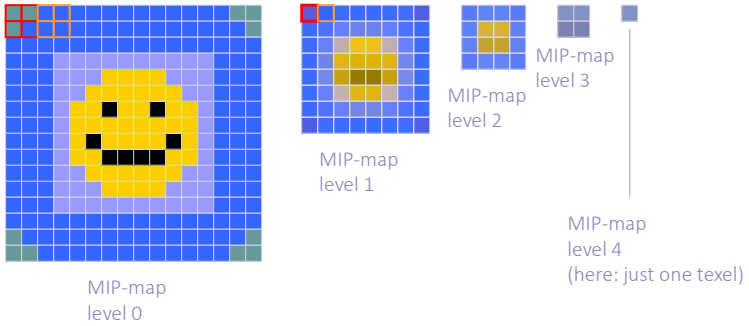


- Pre-filtering of textures
- A LOD-pyramid, for images
- Hardware picks the right level (for each screen pixel)
- Reason: quality (not efficiency!) - avoids subsampling artifacts



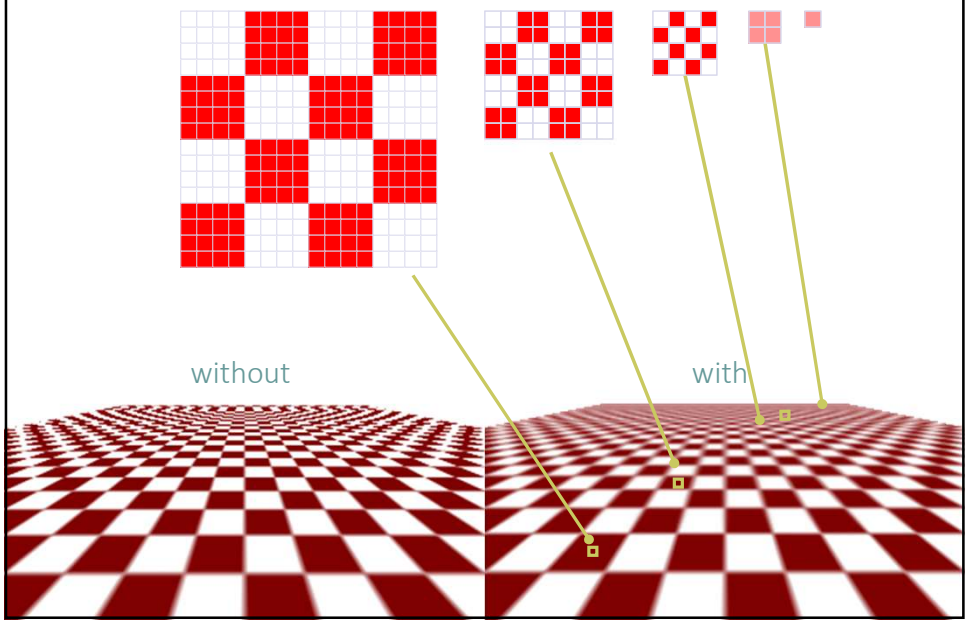
15

MIP-map = texture prefiltering



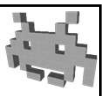
16

MIP-map = texture prefiltering



17

Texture maps as assets: characteristics



- Size:
  - resolution
  - channels (1,2,3,4)
- MIP-map levels
  - are they present?
  - how many
- Compression?
  - e.g., color quantization (“color-map” or “palette”)
  - compression schemas designed specifically for textures such as: DXT1-5 (DirectX Texture – Microsoft)

HW imposed constraints:

- Power of 2 for side (U and V)
  - e.g.: 256x256 or 1024x512
  - not a strict requirement any longer (for modern APIs)
- Hard-wired upper-bounds
  - today: 8K, 4K, or even just 2K

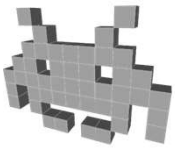
18

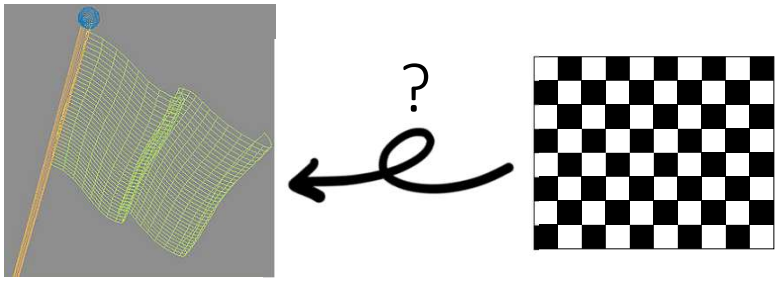
Most of the visual richness perceived in the typical videogame is due to textures!

Texture resolution has a bigger impact impact on quality than Meshes resolution!

19

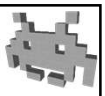
How do we define how a given texture is mapped over a given mesh?





20

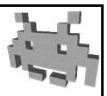
How is a texture mapped over a given mesh?



- **3D Models**
  i.e. tri-meshes with:
  - per vertex attrib
    - normals, color, AO, …
  - LODs
  - **uv-map**
  - keyframes
    - cyclic animations
    - face-morphs, …
  - "skinning"
- Materials
  - lighting model stats / flags
  - **textures**
    - **RGB maps**
    - **normal maps**
    - **alpha maps**…
  - shaders
    - vertex, fragments, …
- Animations
  - blend shapes
  - skeletal animations
  - kinematic animations
  - geometry caches
  - skeletons (rigs)
- Geometric proxies
  - hit-boxes
  - bounding objects
  - AI-meshes
- Particle systems
- Environments
  - 3d scenes
  - skydomes
  - env. maps
  - scene props

21

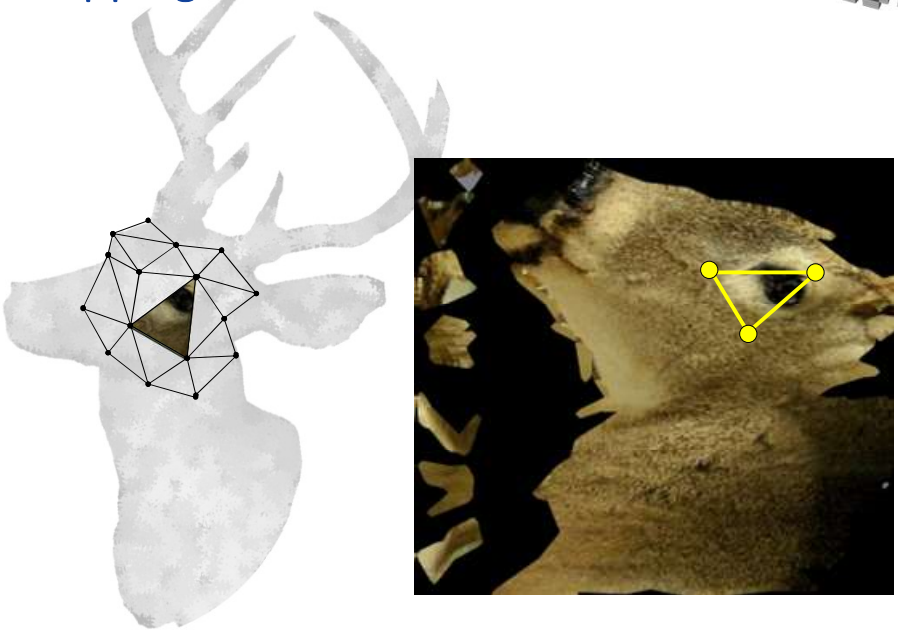
UV-Map of a mesh



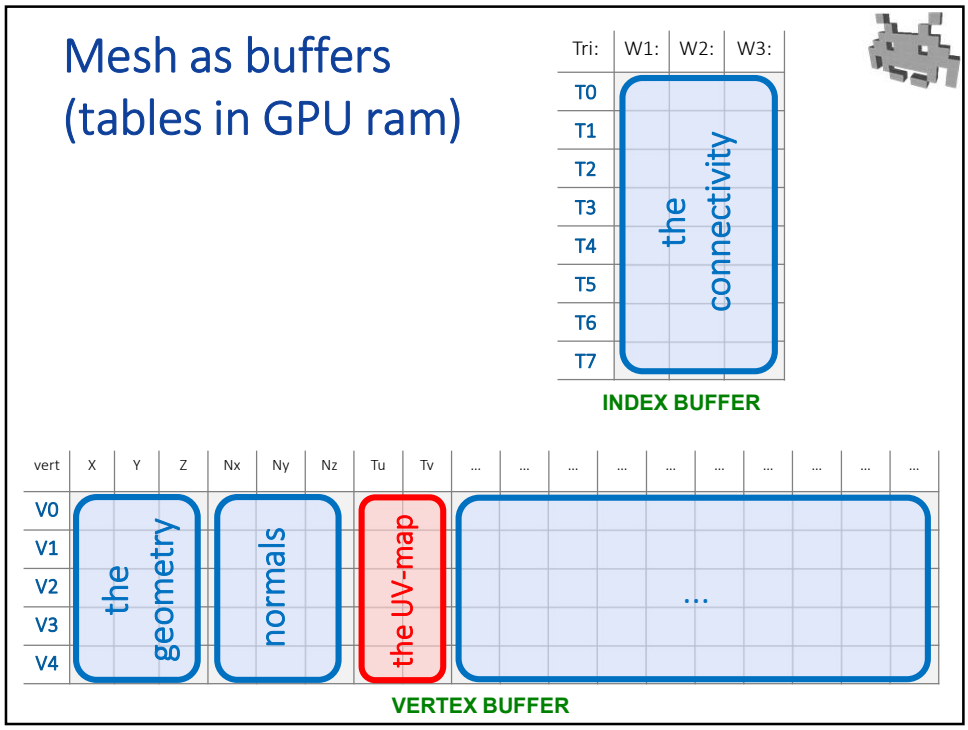
- A mapping : mesh surface → 2D texture space $[0..1]^2$
  - Aka
- It is stored as per vertex attribute : The «(u,v) position» (or «texture coordinate») of that vertex
- The set of UV positions is called the
  - The «u-v map» of the mesh
  - Or its «parametrization» (in Geometry Processing)

22

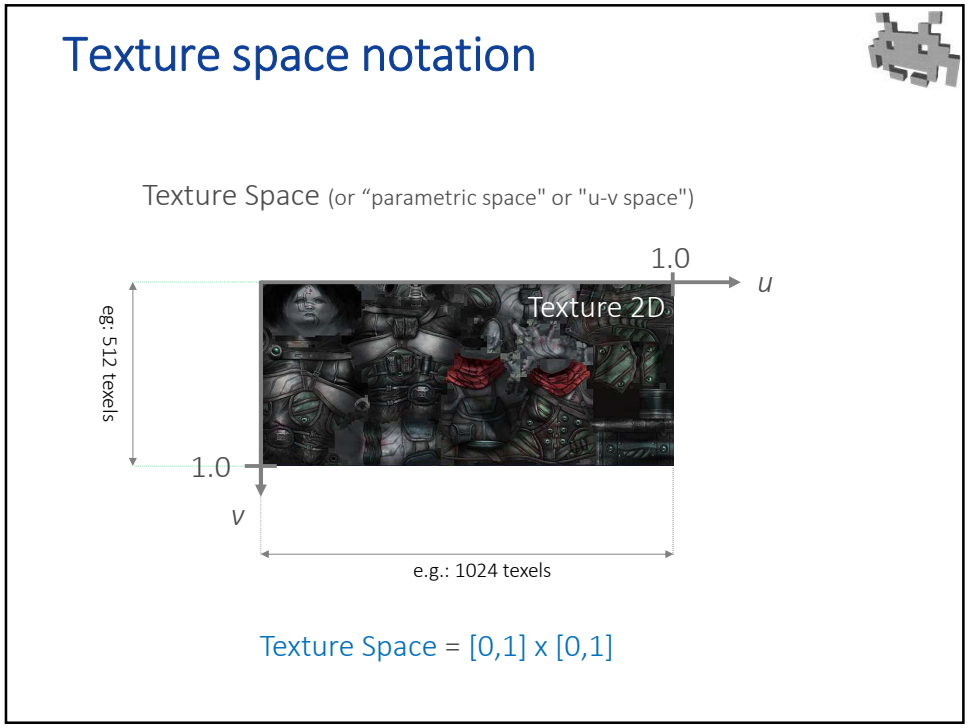
Mapping the texture over the mesh



23

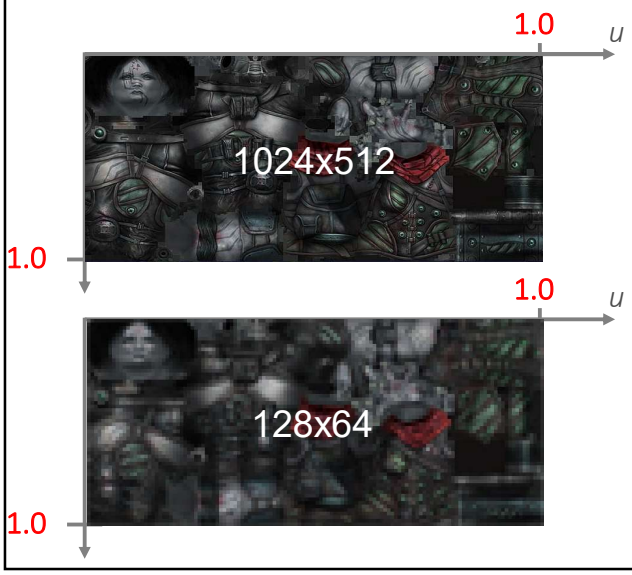

Mesh as buffers
(tables in GPU ram)
Tri:
W1:
W2:
W3:
T0
T1
T2
T3
T4
T5
T6
T7
the connectivity
INDEX BUFFER
vert
X
Y
Z
Nx
Ny
Nz
Tu
Tv
V0
V1
V2
V3
V4
the geometry
normals
the UV-map
...
VERTEX BUFFER

24


Texture space notation
Texture Space (or “parametric space" or "u-v space")
1.0
u
Texture 2D
eg: 512 texels
1.0
v
e.g.: 1024 texels
Texture Space = [0,1] x [0,1]

25

Note: Texture space independent from texture resolution (or aspect ratio)



Convenient!
We can reduce texture-sheet resolution (balancing quality / memory) without affecting the UV-map of the mesh.

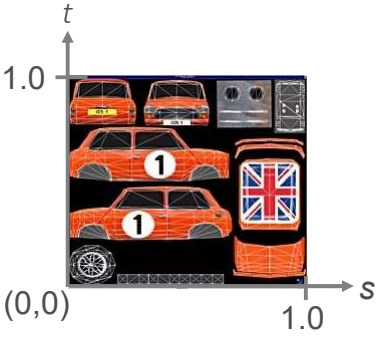
E.g.: load in GPU RAM only a few smaller MIP-map levels

26

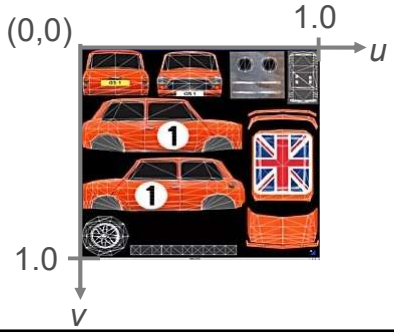
Two notations

The standard in games

s-t
(es OpenGL)

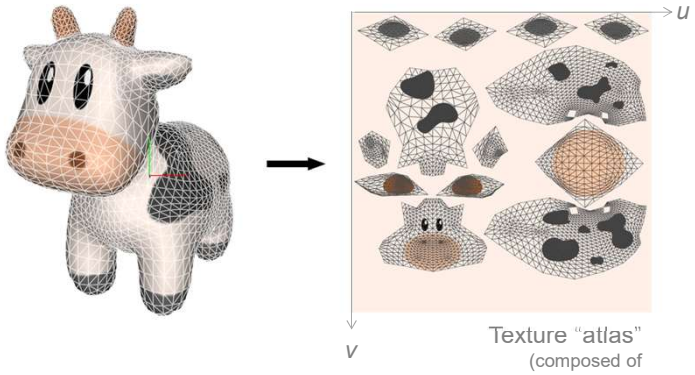


u-v
(es DirectX)



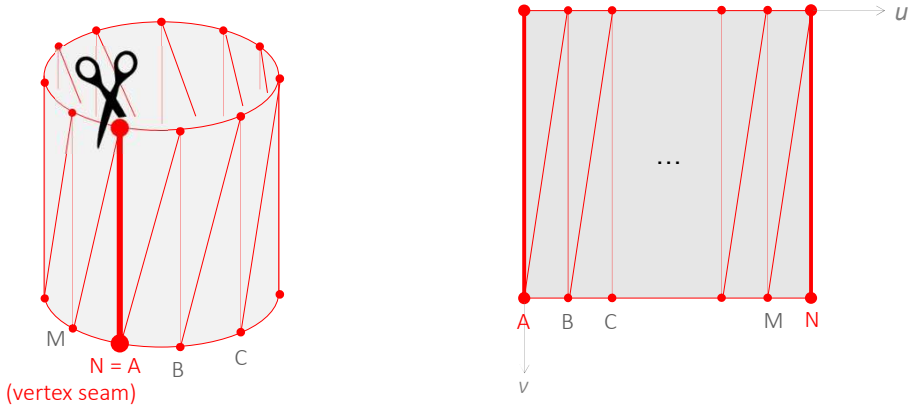
27

Modeling task: u-v map construction



28

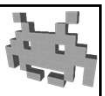
UV map: example

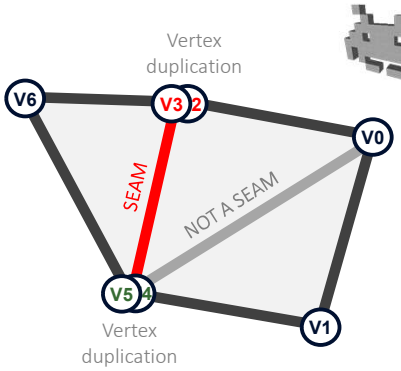


29

Texture seams
(or just texture "cuts")

- Texture seams are necessary to encode the UV-map





	X	Y	Z	U	V	…
V0	p_x0	p_y0	p_z0	$u0$	$v0$	…
V1	p_x1	p_y1	p_z1	$u1$	$v1$	…
V2	p_x2	p_y2	p_z2	$u2$	$v2$	…
V3	same		as	$u3$	$v3$	…
V4	p_x3	p_y3	p_z3	$u4$	$v4$	…
V5	same		as	$u5$	$v5$	…
V6	p_x4	p_y4	p_z4	$u6$	$v6$	…

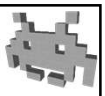
GEOMETRY + ATTRIBUTES

Tri:	Wedge 1:	Wedge 2:	Wedge 3:
T0	0	1	4
T1	4	2	0
T2	5	3	6

CONNECTIVITY

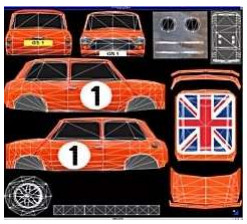
30

«Atlas» UV-map



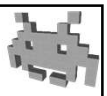
A (very common) class of UV-maps

- The mesh is split into "patches" (groups of triangles)
- Each patch is mapped to a separate "island" in UV space
- Island are packed in the texture rectangle
- In this setup,
  texture seams = any mesh edge separating two faces of two different patches.



32

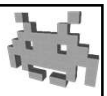
Constructing a UV-map for a mesh (or, «UV-mapping» a mesh)



- Typical task of the modeler (digital artists)
  - (semi-)automatic algorithms are deeply studied
- We need to find a spot in the (2D) texture space for each (3D) mesh triangle
- Similar to peeling an apple:
  - Cut the skin of the apple (cutting phase)
  - Lay each produced peel in 2D (unfolding phase)
  - Pack the peels inside a rectangular space (packing part)
- Cuts (aka "texture seams") are (almost) always required!
  - they are discontinuity of u,v attributes
  - stored in the mesh as vertex-seams (vertex duplications)

33

Modeling task: "UV mapping" (a verb!)

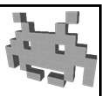


- The modeler..:
  - 1. selects of the cutting edge
    ...or...
    1. assigns faces to texture "charts"
    - either way, they decide where "texture seams" are
  - 2. unfolding
    - minimizing "distortion" (by automatic algorithms)
  - 3. charts packing (again, often automatized)
    - Minimize the empty space in textures
    - Assign areas according to necessities
      (important areas → bigger texture space)
      (the distribution of the texels becomes adaptive!)

See DEMO

34

Two types of UV-maps

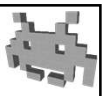


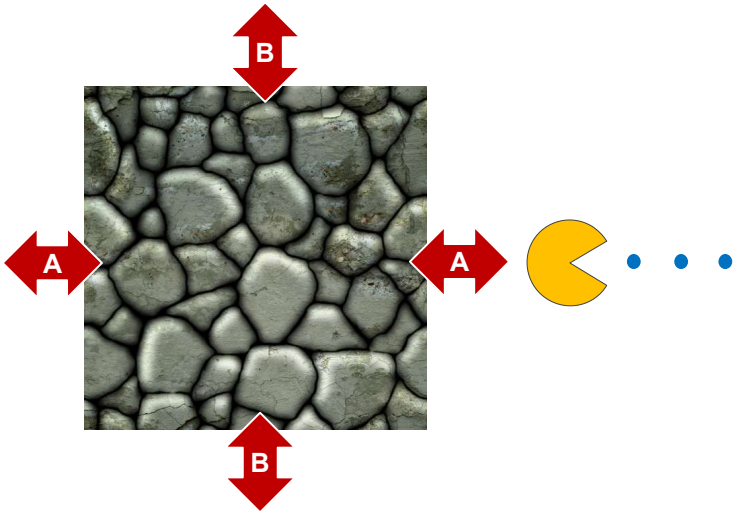
- NOT injective UV map ← aka: just "UV-map" (the standard)
  - Different zones of the mesh can be mapped to the same texture region
  - e.g.: charts of an atlas are overlapping
  - ☺ optimization of texture RAM
    - Can exploit of simmetries / repetitions
- Injective UV map ← aka: "unwrapping" or "unwrapped UVs" or "1:1 UV-map" or "lightmap" UV-map or "non-overlapping" UV-map
  - 1 (non-empty) point on the texture = 1 point on the mesh
  - non-overlapping charts! (& no self-overlap)
  - ☺ Generality / Flexibility
    - Used for several scopes (e.g., light-baking)
- Different objectives
  - both may be present
  - 2 distinct attributes (UV_A, UV_B) for each vertex

Q: which is the type of the UV-maps shown in prev slides?

35

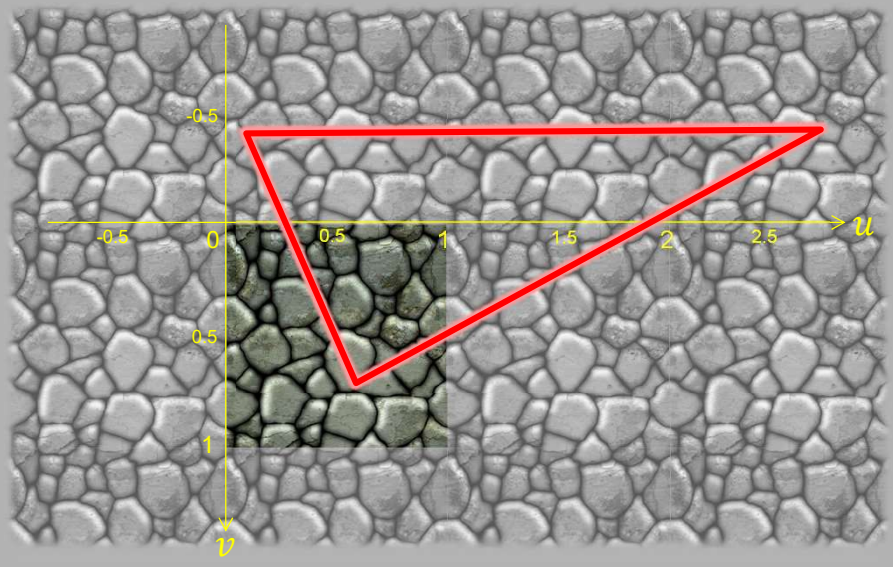
Tileable Textures





36

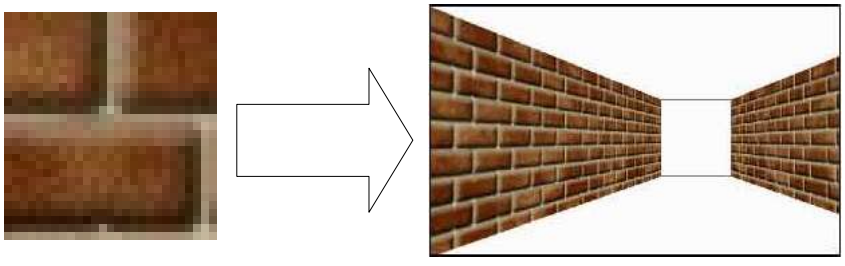
Tileable textures



40

Tileable textures

- Many uses

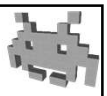


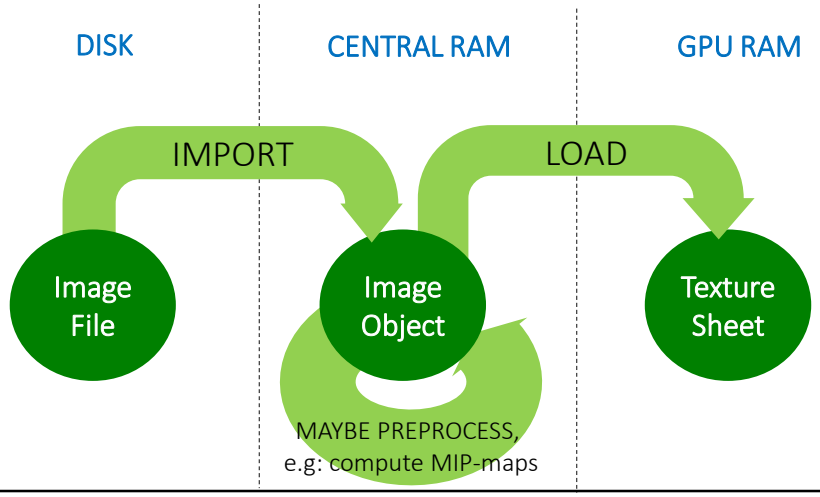
Very space-efficient (in VRAM too!)

41

Life of a Texture in a Game Engine

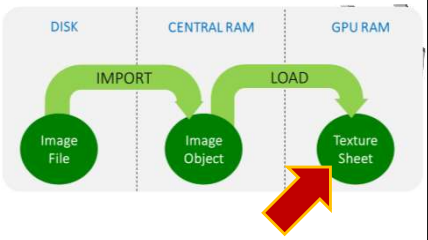
the same as, basically, any 3D assets





42

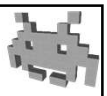
Texture Sheets (in GPU RAM)



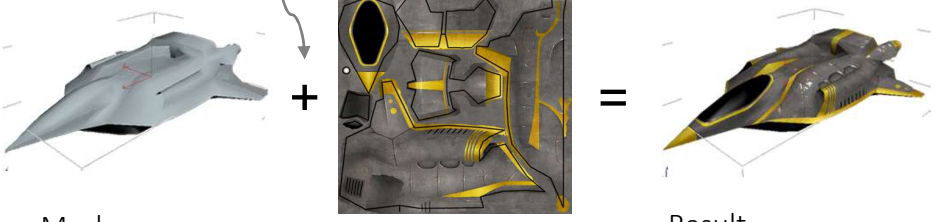
- Rasterized images, but with peculiarities …
  - MIP-map pyramid
  - channels per texel: 1,2, or (most typically) 4
  - texel format:
    fixed schemas, e.g.: 8 bits per channel,
    or 6+5+5+1
    or floating-point per channel (e.g. 16 bits)
  - compression: specific texture schemas (see next)
  - resolution: powers of 2 per side,
  - resolution: Hard Wired upper limits

43

Texture mapping is hard-wired *deep* in the GPU



Texture mapping **on the fly**, during rendering



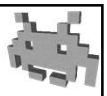
Mesh (with UV-map) **in VRAM**

Texture(-s) (with MIP-maps) (compressed!) **in VRAM**

Result **on Screen**

44

Texture mapping is hard-wired *deep* in the GPU

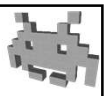


- "**Texture mapping**" uses per-pixel "**Texture access**" (aka "**texture fetch**")
  - The process of applying a texture over a mesh, according to its UV-map
  - The process of reading values from a texture stored in VRAM, by the GPU,
- Implementation of texture access is **hard-wired** in the GPU
- Good consequences:
  - Super fast!
  - Supports on-the-fly decompression
  - and much more functions (see later)
- Bad consequences:
  - Rigid, fixed formats for texutres (in VRAM) with hardcoded limitations (see later)

Can keep textures compressed in VRAM, see the results on screen, without ever decompressing the whole image or storing it decompressed, anywhere

45

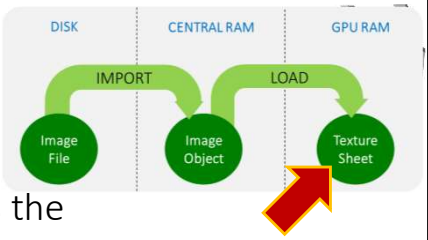
Per-fragment Texture fetch (during rendering, hardwired in GPU)



- Hard-wired GPU mechanisms to access the texture image at a given location: $(u, v) \rightarrow \mathbb{R}^4$ (number of channels)
- Includes many steps (per pixel!):
  1. Management of out-of-bound coordinates. E.g., repeat mode: $u \leftarrow u - \lfloor u \rfloor$ and $v \leftarrow v - \lfloor v \rfloor$
  2. De-normalization of coords, from normalized $[0..1]^2$ to texel coord $[0..res_X] \times [0..res_Y]$
  3. Selection of the appropriate MIP-map level On-the-fly decompression of compressed data
  4. Bilinear interpolation between 4 texels, plus linear across MIP-map levels

46

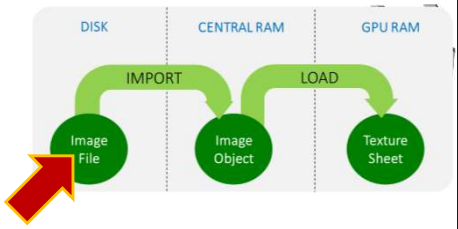
Texture compression (to save VRAM)



- Saves VRAM, but preserves the random-accessibility of texels. Examples:
  - color quantization
    - e.g., 5 red 5 green 5 blue 1 alpha = 16 bits per texel
  - color-table, or "palette"
    - e.g., 256 color table for texture, an 8-bit index per texel
  - specialized image-compression schemas. They are:
    - Lossy (very much so)
    - Fixed compression rates (e.g. ¼)
    - Unfavorable compression/loss ratio ☹
    - Most diffuse scheme S3TC, with variants: DXT-1 -2 -3 -4 -5 (yes/no alphas; uniform alphas; smooth alphas)

47

Textures as assets: file formats



For generic images
(decompress the entire image before accessing any pixels)

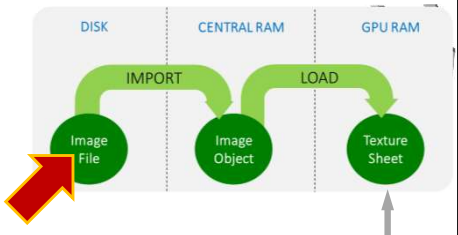
- ☺ compression: excellent
- ☹ loading: heavy
  - Decompress from RAM, (maybe) recompress in GPU-RAM
- ☹ MIP-map levels: Procedurally generated. Control by the engine
- ☺ Resolution: any

For textures
(random accessibility to texels, without uncompressing the entire image)

- ☹ compression: bad
- ☺ loading: light
  - direct streaming possible Disc => RAM => GPU RAM
- ☺ MIP-map levels: Baked. Control by the artist
- ☹ Resolution: sometimes, must be a pow of 2

48

Textures as assets: file formats



For generic images:

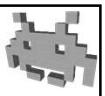
- .JPG / .JPEG
  - ☹ lossy,
  - ☺ good compression rate
  - ☺ "photographic" images: best
  - ☹ only 3 channels (no choice)
  - ☹ 8 bit per channel (no choice)
- .PNG
  - ☺ lossless
  - ☹ compression ratio (for natural images)
  - ☺ good for artificial images (logos)
  - ☺ alpha channel: also possible
  - ☺ 16bits: possible
- .TIFF e .RAW (rare)
  - ☺ lossless
  - ☹☹ no compression
  - ☺ max flexibility for channels, image depth
- .PNM (rarer, but useful for toy projects)
  - ☹☹☹ compression: verbose
  - ☺ Very easy parsing! (no lib needed)

Specialized for textures:

- .DDS («direct draw surface») same format used in GPU. Verbatim copy of data as it will be in GPU RAM Thus:
  - ☺ includes MIPmap levels (if needed)
  - ☹ compression: very lossy And bad compression rate (and fixed)
  - ☺ GPU ready! Just read from disk & load on GPU memory (no decompress / recompress!)

49

Type of textures by content: alpha maps / cutout textures



An alpha map (or transparency map):
a one-channel texture dictating
how transparent are the different parts of the 3D model

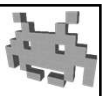
- Value = 1: **fully opaque** (as normal)
  During rendering:
  its pixels cover (overwrite) what's behind them
- 0 < value < 1: **semi-transparency**
  during rendering:
  the pixel are blended with the color behind them
- Value = 0: **fully transparent**
  the pixels are discarded (leaving what's behind unaffected)

A transparency map with 0-1 values (1 bit per texel)
is called a cutout texture

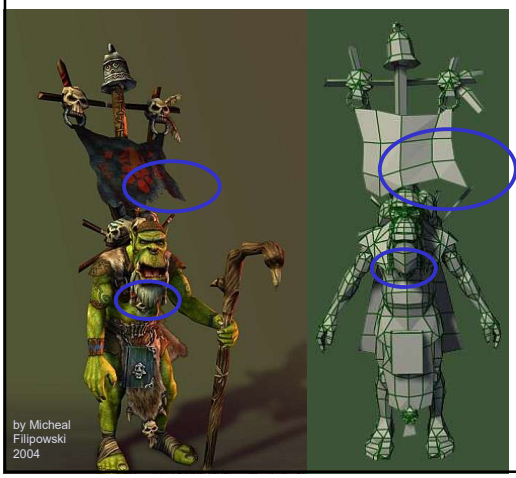
Let's see typical use examples…

50

Cutout textures
Texels = transparency level (0 or 1)



- e.g.: drapes, beard…



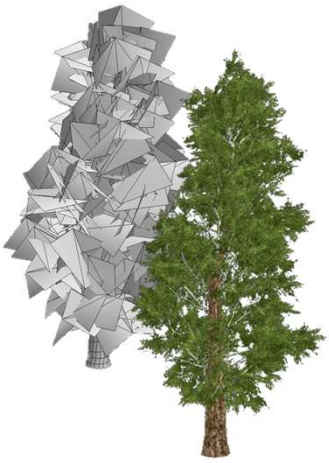
Texture
(RGBA, 4 channels)

52

Cutout textures
Texels = transparency level (0 or 1)

- e.g.: trees, foliage



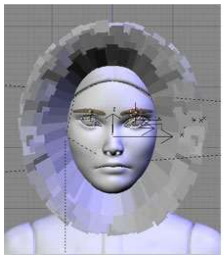
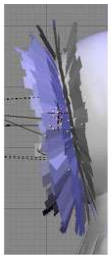
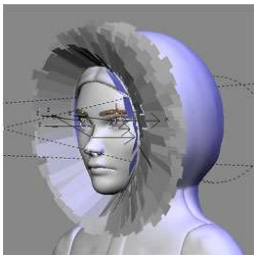


53

Texture mapping and Alpha Test

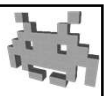
- Eg: fur, fur coats

The texture (horizontally tileable) Pink is transparent

54

Next topic: Bump-Map (*)



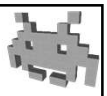
Any texture modelling geometric shape details
(that is, high-frequency geometric features)

- details not represented by the "real" geometry (the mesh)
- remember: meshes tend to be low-poly
  - not much detail in them
- this approach is also known as "Texture-for-Geometry"
- rationale:
  texels are much cheaper to render/store than vertices!
- geometric details may extrude out or be engraved in
  the "real" (mesh) surface
- in many cases: the detail affects lighting only
  - this is sufficient to trick the eye
  - especially with dynamic lighting

(*) This terminology not universal: e.g., «bump-map» can mean specifically «displacement map»

55

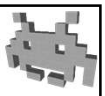
Other (typical) examples of usage of bump-maps…



- Wrinkles on a character skin
- Wrinkles on a skin orange
- Depression between bricks of a brick wall
- Pattern of concrete on a concrete wall
- Musculature on a bare torso
- Small wrinkles on a garment
- Veins on a hand
- In general, small, high frequency 3D details
  on an otherwise smooth surface

56

A summary: how do we model the 3D shape? It's a matter of scale!



mesh

- macro-structure of the object , such as …
  - …the general shape of the horse
  - …the general shape of the face
  - …the general shape of the dragon

bump-map

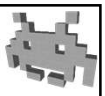
- meso-structure of the object, such as …
  - …the musculature of the horse
  - …the wrinkles of the face
  - …the flakes of the dragon

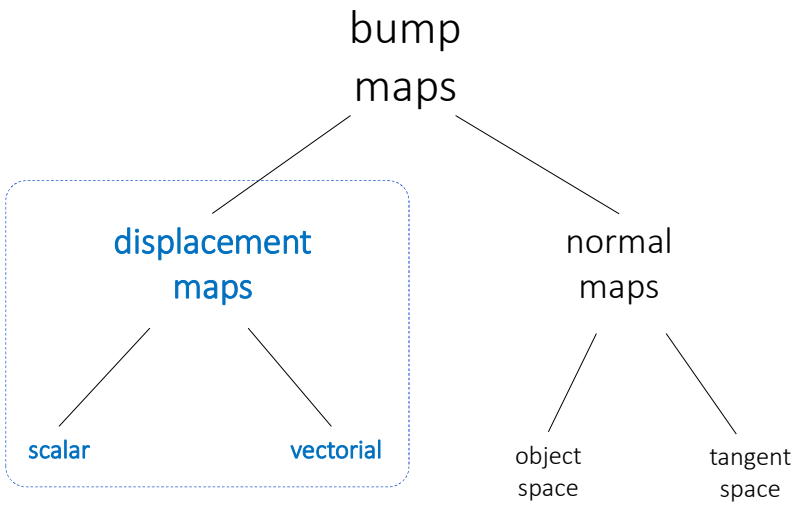
material params

- micro-structure of the object, such as …
  - …the velvet-like fur of the horse
  - …the structure of the dermis / sebum
  - …the roughness / smoothness of the flakes

57

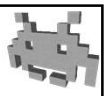
Types of Bump maps





59

Tassonomy of Bump maps (summary)



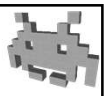
- Bump map:
  - Any texture encoding hi-frequency **geometric details**
- Displacement Map:
  - Details are encoded by storing geometric differences between the mesh geometry and the detailed surface:
  - either as scalars (distance along the normal), or as vectors
  - used for: on-the-fly re-tessellation, or *parallax mapping* technique
- Normal Map:
  - Details are encoded by storing the normals of the detailed surface
  - used for: lighting computation (affects lighting only)

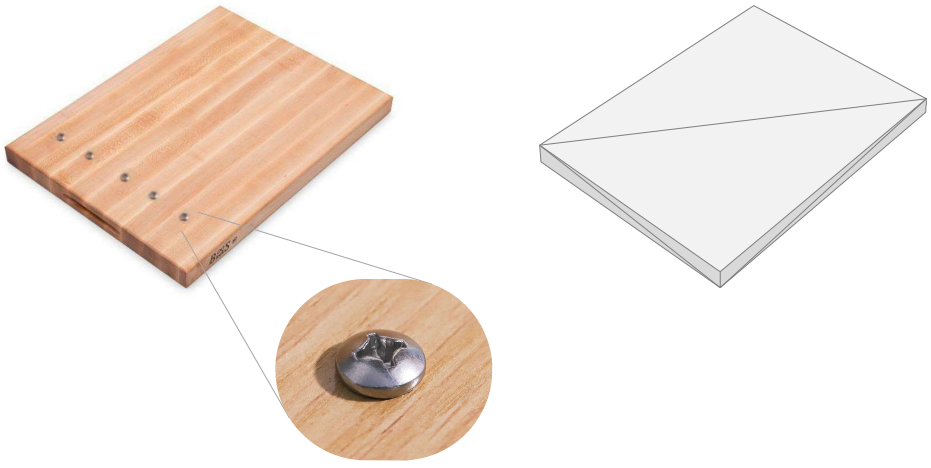
Note: they are not mutually exclusive!

- One 3D model can have both
- (different texture "sheets" encoding different things using the same uv-map)

60

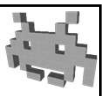
For example





61

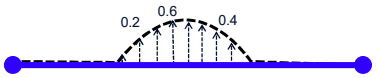
(Scalar) Displacement map : concept



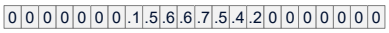
Stores the distance of the detailed surfaces from the plain geometry

- example: a bump-map for a screw-head  :

Detailed surfaces
(what I would like to represent)



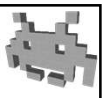
low-poly mesh
the approximation (here: it's flat ☹)



scalar displacement map
(1 texel = 1 scalar)

62

Scalar displacement map: notes



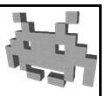
- Each texel stores: a distance of the detailed surface
  - Along the normal direction (of low-poly mesh)
  - 1 scalar per texel –> 1 channel texture
- Which way:
  - outwards (*extrusions*)
  - inwards (*excavations*)
  - or both (signed displacements)
- Storage:
  - gray-scale image (1 scalar per pixel)
  - remap values within the interval [0..1]
  - global scale factor (on the fly)
- Possible uses:
  - Direct lighting of implied normals: "embossing" effect (old effect: it's a bad approximation, not common anymore)
  - Global illumination (ambient occlusion) See later
  - «Parallax mapping» technique See later
  - Intermediate data for the construction of a normal map See later

white = sticks outwards
black = stays flat

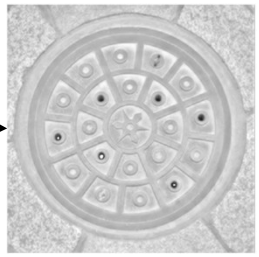
Easy to paint (by artists) / manipulate.

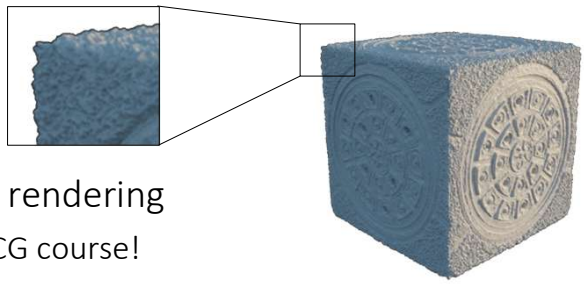
63

(scalar) Displacement map rendering: parallax mapping



- Technique used render a mesh with a Displacement Map
  - Bonus: the silhouette of the object can be affected



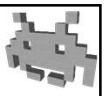


- See lecture on rendering
  - Or advanced CG course!

Image courtesy of https://cgcookie.com/articles/normal-vs-displacement-mapping-why-games-use-normals

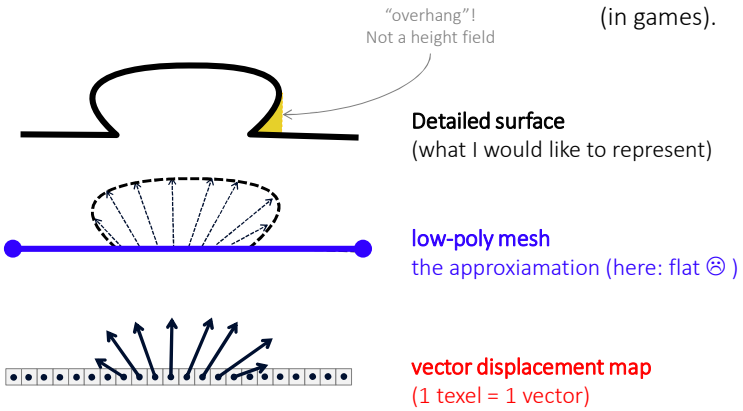
65

Vectorial displacement map : concept



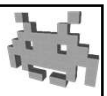
Store Vectors from the plain surface to the detailed surfaces

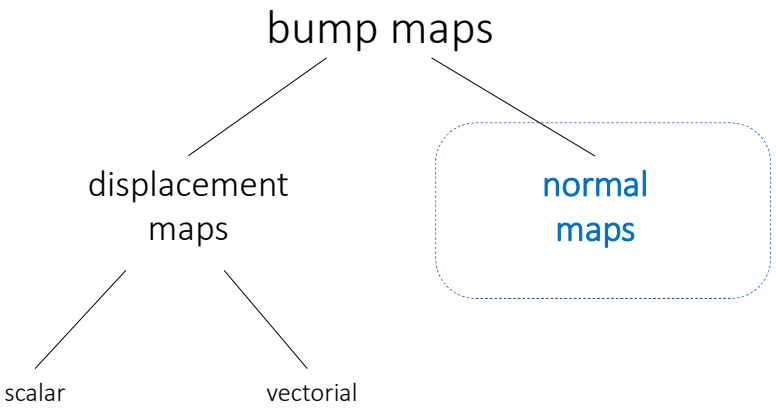
More expressive variant, but drastically more expensive and less usable. Not much used (in games).



66

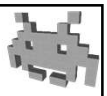
Tassonomy of Bump maps





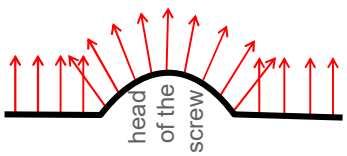
67

Normal Map: concept



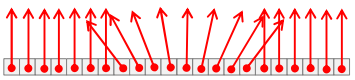
Store the Normals of the detailed surfaces

- example -- a normal-map for a screw-head



Detailed surface
(I would like to model)

low-poly mesh
(the approximation) (here: flat ☹)



normal map
(one normal per texel)
(no displacement = use geometric normal)

68