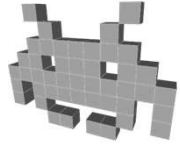
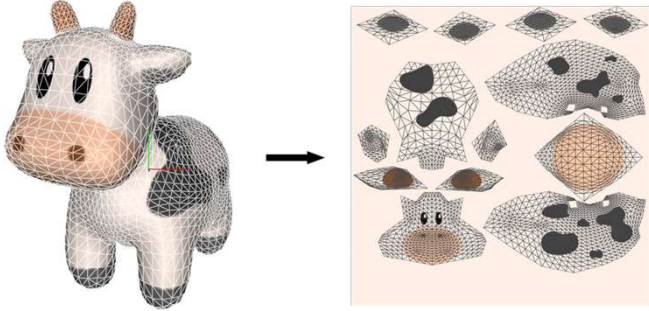


## 3D VideoGames Textures in 3D Games




---

Marco Tarini



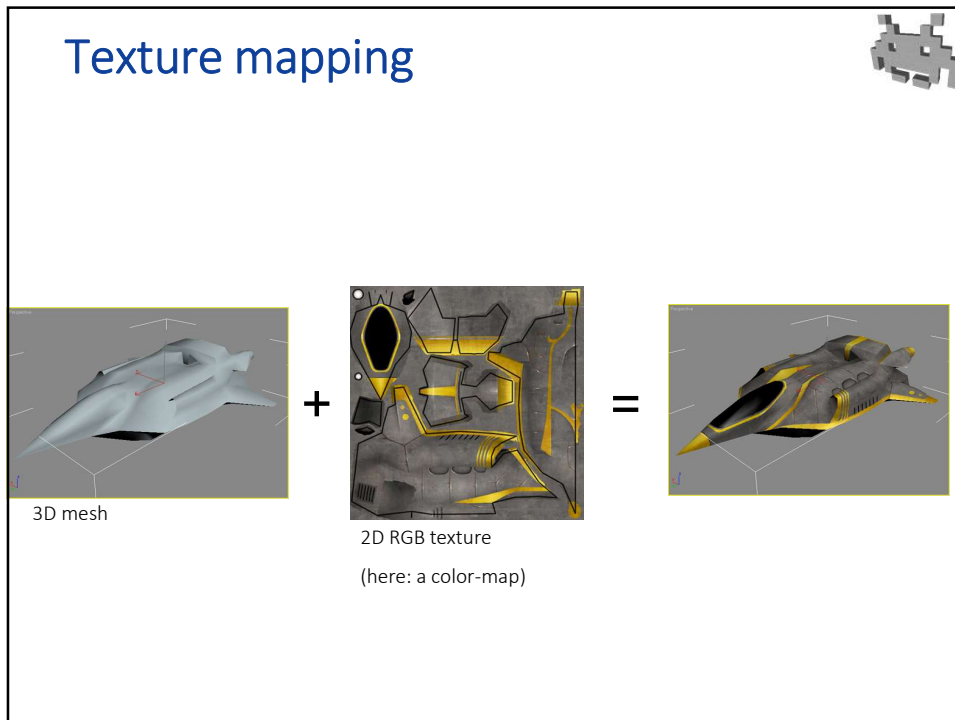
1

## Course Plan

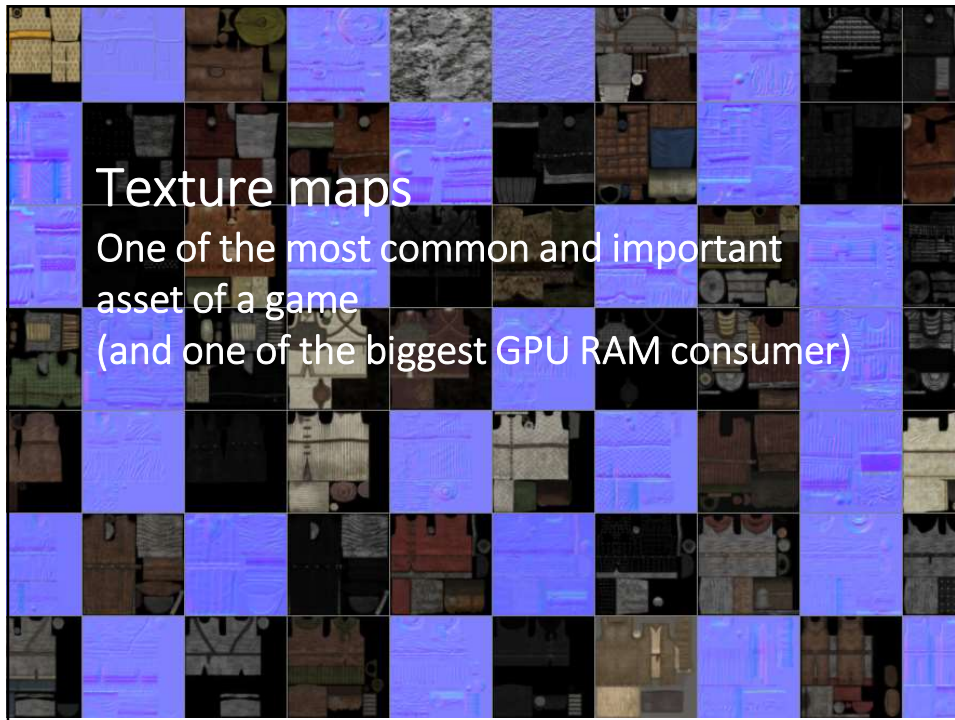


- lec. 1: **Introduction** ●
- lec. 2: **Mathematics** for 3D Games ●●●●●●
- lec. 3: **Scene Graph** ●
- lec. 4: **Game 3D Physics** ●●●●+●●
- lec. 5: **Game Particle Systems** ●
- lec. 6: **Game 3D Models** ●●
- lec. 7: **Game Textures** ●●
- lec. 9: **Game Materials** ●
- lec. 8: **Game 3D Animations**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

2



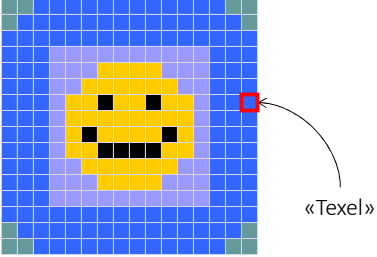
4



6

## Texture maps: data structures

- In practice, a **rasterized image**



Texture sheet

«Texel»

7

## Textures (or texture maps)

- Multiple **texture sheets** (1 raster image of texels) each defines a signal over the mesh
  - Similar purpose to per-vertex attributes!
  - but...
    - # texels  $\gg$  # vertices
    - More complex signals!
- 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!)

Texture: regular sampling, and dense

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

8

## Signals stored in textures (in games)



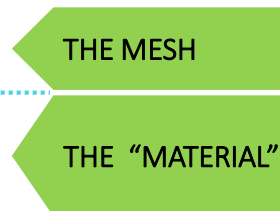
- Each texel = a base-color (components:  $r, g, b$ )
  - The texture sheet is a “diffuse-map” / “color-map” / “RGB-map”
- Each texel = a transparency factor (components:  $\alpha$ )
  - The texture sheet is a “alpha-map” or “cutout-texture” (exp. if 1bit)
- Each texel = a normal (versor, with components:  $x, y, z$ )
  - The texture sheet is a “normal-map” or “bump-map”
- Each texel = a specular coefficient value
  - The texture sheet is a “specular-map”
- Each texel = a glossiness value
  - The texture sheet is a “glossiness-map”
- Each texel = a *baked* lighting value...
  - The texture sheet is a (baked) “light-map”
- Each texel = a distance from a surface value
  - The texture sheet is a “displacement map” or “height texture”

9

## GPU rendering of a Mesh in a nutshell (reminder)



- Load...
  - store all data on GPU RAM
    - Geometry + Attributes
    - Connectivity
    - Textures
    - Shaders
    - Parameters / Settings
- ...and Fire!
  - send the command: “do it” !



10

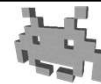
## Texture maps assets and Mesh assets



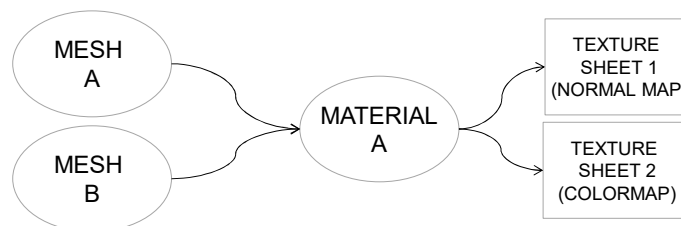
- Several texture «sheets» associated to a mesh
  - or also: more meshes on the same sheet (bene)
- Typical structure :
  - each mesh associated to a material
  - each material:
    - 1 sheet di diffuse-map
    - 1 sheet bumpmap (if needed)
    - 1 sheet di alphamap (if needed)
    - 1 vertex shaders + fragment shader
    - Several parameters
      - (e.g., shininess, ...)
  - If different parts of mesh associated to different textures:  
decompose the object in sub-mesh

11

## Texture maps assets and Mesh assets



- Not necessarily 1:1
  - 1:N -- several textures «sheets» associated to a mesh
  - N:1 – more meshes on the same sheet (goof)
  - If different part of mesh associated to different textures:  
decompose the object into sub-mesh



12

## MIP map levels

- Pre-filtering of textures
- LOD pyramid, for images
- Hardware picks the right level (for each screen pixel)
- Avoids subsampling artifacts



13

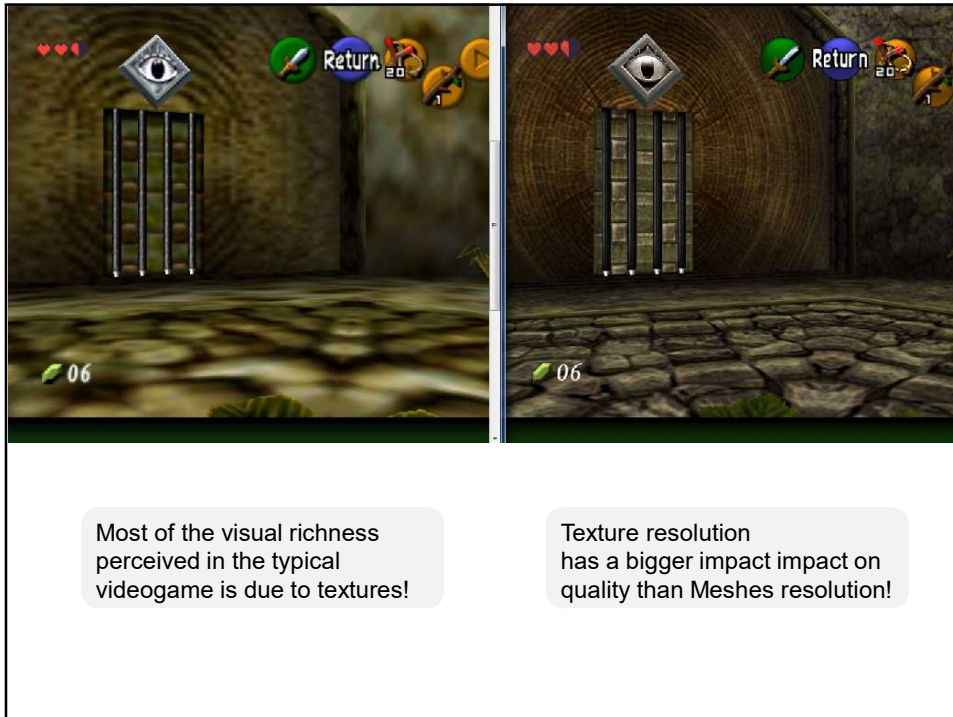
## Texture maps as assets: characteristics

- 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)

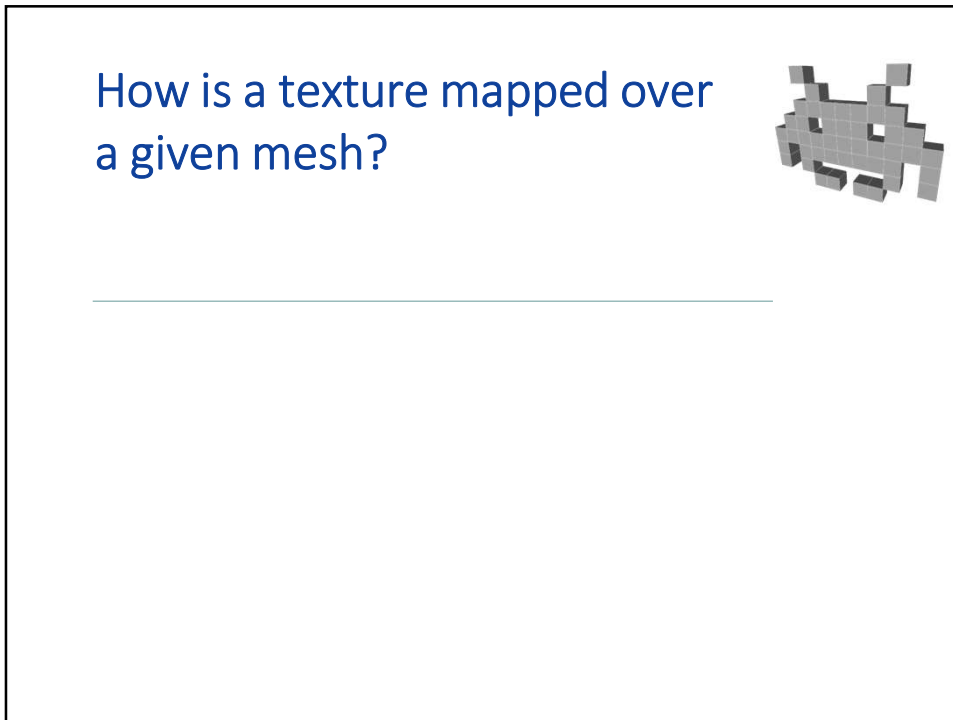
### HW imposed constraints:

- 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

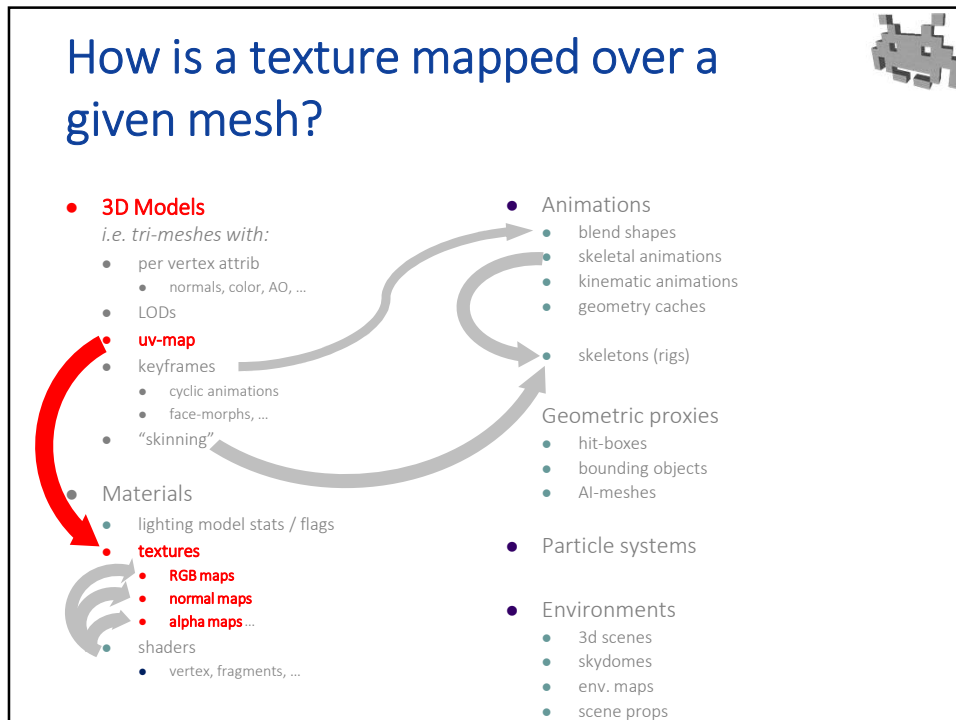
14



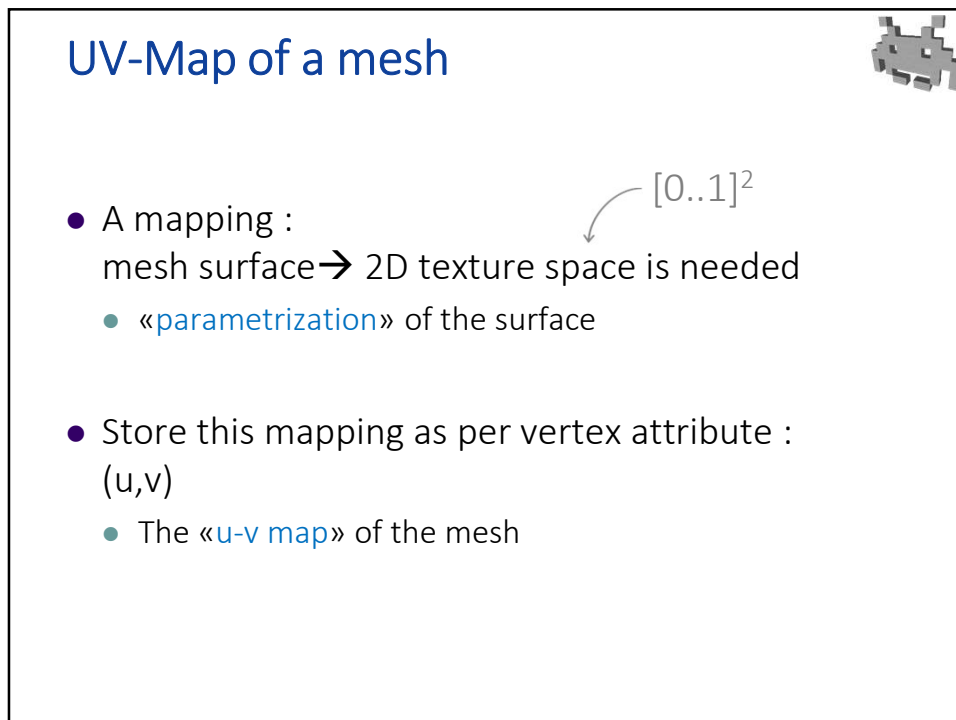
15



16

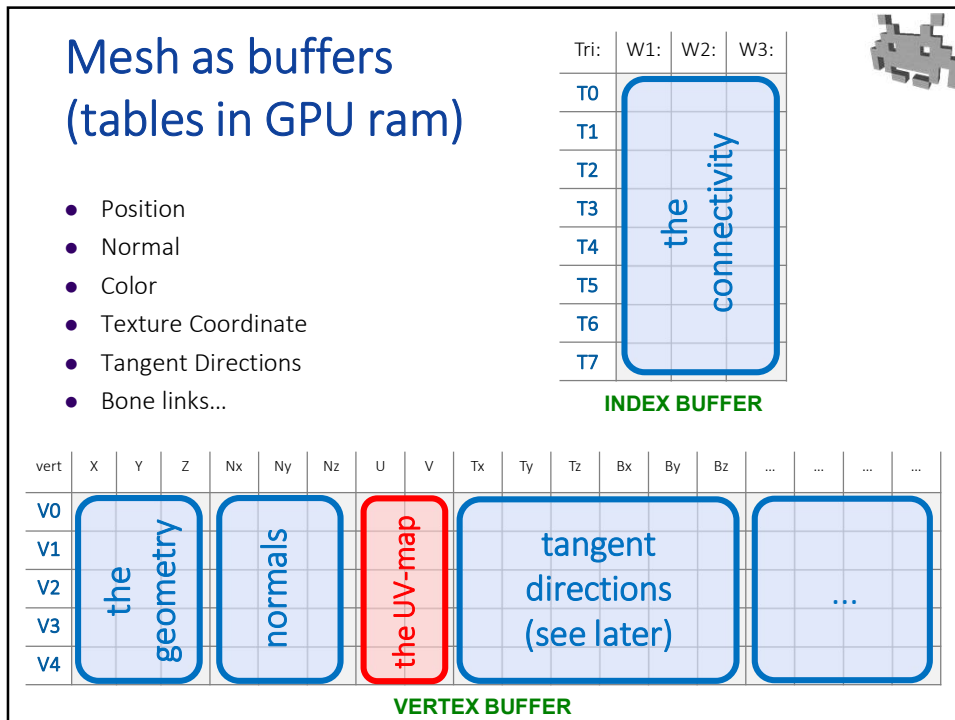


17

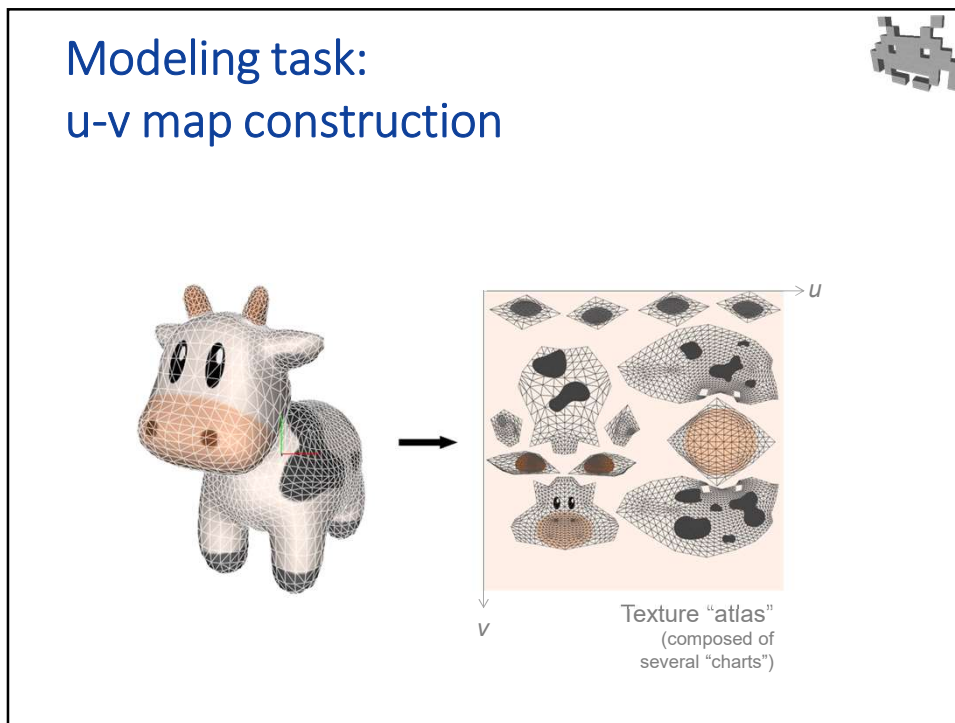


18





19



20

## Texture space notation

Texture Space (or "parametric space" or "u-v space")

Texture 2D

1.0  $u$

1.0  $v$

e.g.: 1024 texels

e.g.: 512 texels

Texture Space =  $[0,1] \times [0,1]$

21

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

1.0  $u$

1.0  $u$

1.0

1.0

1024x512

128x64

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

22

### Two notations

Most used  
(in game industry)

**s-t**  
(es OpenGL)

**u-v**  
(es DirectX)

23

### UV map: example

MESH

TEXTURE SPACE

24

## Texture seams (or just texture “cuts”)

- 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	$p_x2$	$p_y2$	$p_z2$	$u3$	$v3$	...
V4	$p_x3$	$p_y3$	$p_z3$	$u4$	$v4$	...
V5	$p_x3$	$p_y3$	$p_z3$	$u5$	$v5$	...
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

26

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

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

28

## Modeling task: “u-v mapping” (verb)



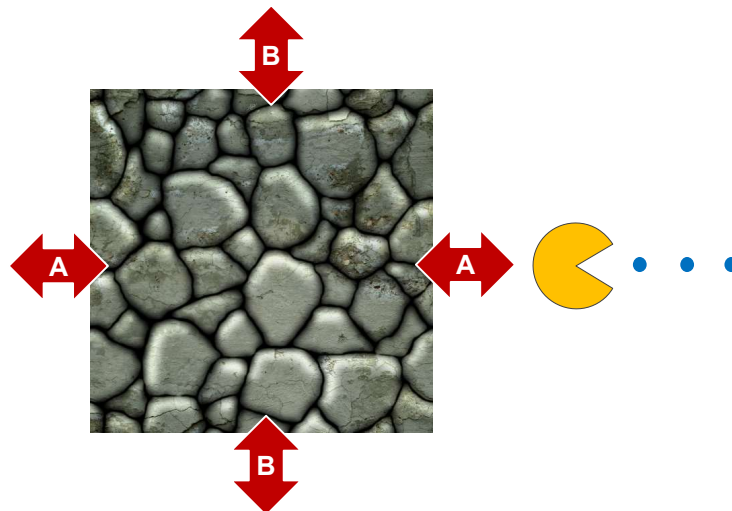
- Strategies:

- 1. select of the cutting edge  
...or...
  1. assign faces to texture “charts”
    - either way, 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 parts → bigger texture space) (sampling of the texels becomes [adaptive!](#))

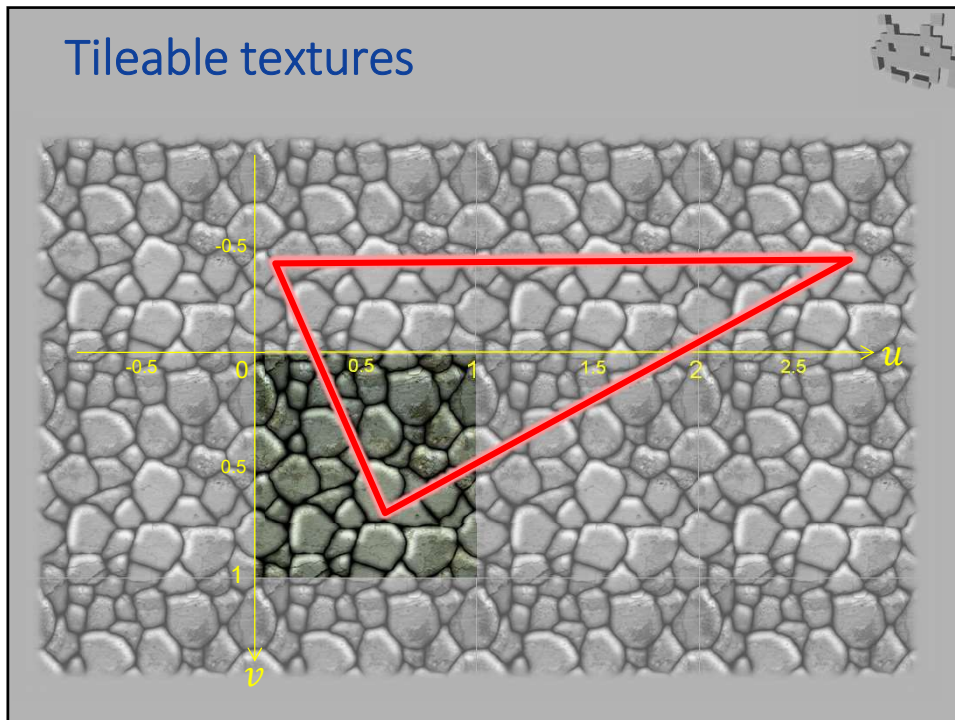
DEMO!

29

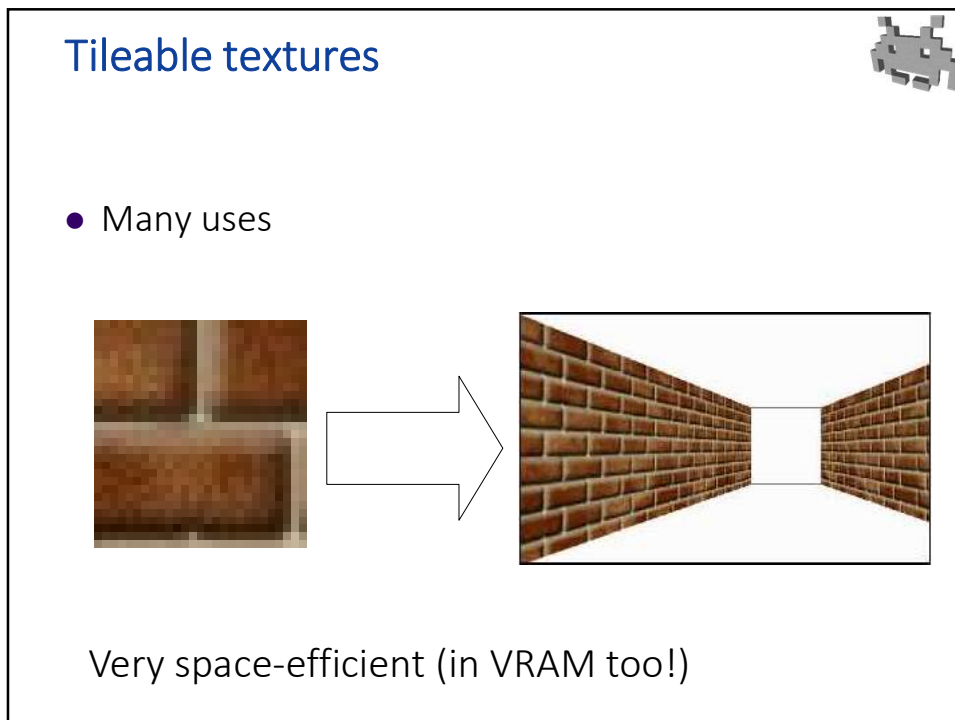
## Tileable Textures



30




33



34

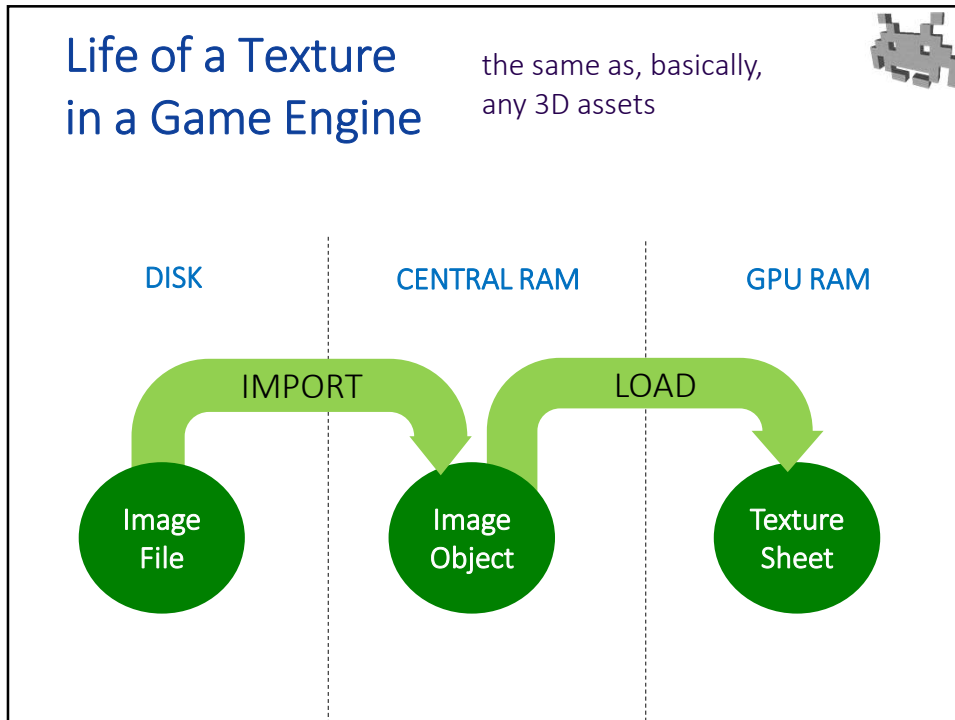
## Two types of UV-maps



- **NOT injective** UV map ← aka: just **"UV-map"** (the standard)
  - Different zones of the mesh mapped to the same texture region
  - e.g.: with overlapping charts
  - ☺ 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

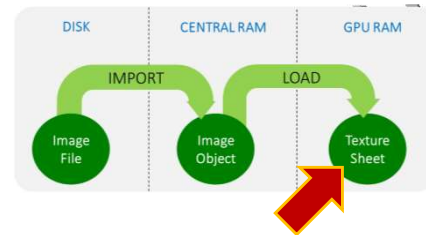
Q: which is the type of the UV-maps shown in prev slides?

35



36

## Texture Sheets (in GPU RAM)



- Rasterized images, but with peculiarities ...
  - MIP-map levels
  - channels per texel: 1,2,3, or (most typically) 4
  - bits per channels:
    - usually 8, fixed point
    - floating textures supported
  - compression: specific texture schemas (see next)
  - resolution: powers of 2 per side

37

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



- **Hard-wired GPU** mechanisms to access the texture image at a given location:  $(u, v) \rightarrow \mathbb{R}^4$
- Includes many steps (per pixel!):
  1. Management of out-of-bound coordinates.  
E.g., repeat mode:  $u \leftarrow [u]$  and  $v \leftarrow [v]$
  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 (how?)
  4. On-the-fly decompression of compressed image data
  5. Bilinear interpolation between 4 texels,  
plus linear across MIP-map levels

number of channels

38



## Texture compression (to save GPU RAM)

- Save RAM, but preserve the **random-accessibility** of texels
  - 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
      - yes/no alphas
      - uniform alphas: -2 -3
      - smooth alphas: -4 -5

39

## Textures as assets: file formats

<h3>For generic images</h3> <p>(decompress the entire image before accessing any pixels)</p> <ul style="list-style-type: none"> <li>😊 compression: excellent</li> <li>☹️ loading: heavy                     <ul style="list-style-type: none"> <li>Decompress from RAM, (maybe) recompress in GPU-RAM</li> </ul> </li> <li>☹️ MIP-map levels: Procedurally generated. Control by the engine</li> <li>😊 Resolution: any</li> </ul>	<h3>For textures</h3> <p>(random accessibility to texels, without uncompressing the entire image)</p> <ul style="list-style-type: none"> <li>☹️ compression: bad</li> <li>😊 loading: light                     <ul style="list-style-type: none"> <li>direct streaming possible Disc =&gt; RAM =&gt; GPU RAM</li> </ul> </li> <li>😊 MIP-map levels: Baked. Control by the artist</li> <li>☹️ Resolution: sometimes, must be a pow of 2</li> </ul>
---	---

40

## 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!)

41

## Type of textures

- Each texel is a base-color (components:  $r, g, b$ )
  - The texture is called a "diffuse-map" / "color-map" / "RGB-map"
- Each texel is a transparency factor (components:  $\alpha$ )
  - The texture is called a "alpha-map" or "cutout-texture" (exp. if 1bit)
- Each texel is a normal (versor, with components:  $x, y, z$ )
  - The texture is called a "normal-map" or "bump-map"
- Each texel is a specular coefficient value
  - The texture is called a "specular-map"
- Each texel contains a glossiness value
  - The texture is called a "glossiness-map"
- Each texel is a *baked* lighting value...
  - The texture is called a (baked) "light-map"
- Each texel stores a distance from a surface value
  - The texture is called a "displacement map" or "height texture"

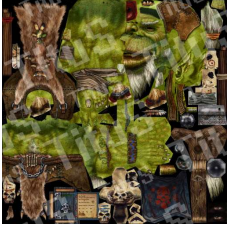

Let's see typical use examples

42

### Cutout textures

Texels = transparency level (0 or 1)

- e.g.: drapes, beard...



Texture  
(RGBA, 4 channels)


by Micheal Filipowski  
2004

44

### Cutout textures

Texels = transparency level (0 or 1)



- e.g.: trees, foliage



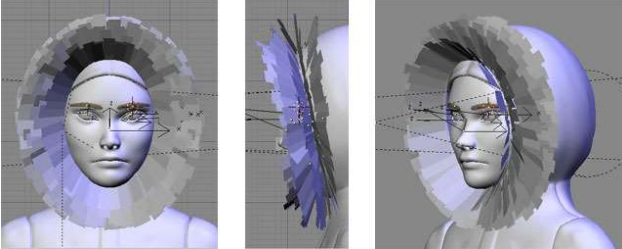
45

## Texture mapping and Alpha Test

- Eg: fur, fur coats



The texture (horizontally tileable)  
Pink is transparent



46

## Bump-Map (\*)


a **texture** representing (or, providing an illusion of) **shape details** (i.e., high-frequency geometric features)

- details not modeled 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 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
  - sufficient to trick the eye
  - especially with dynamic lighting

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

47

## Bump-Map: from the modeler perspective



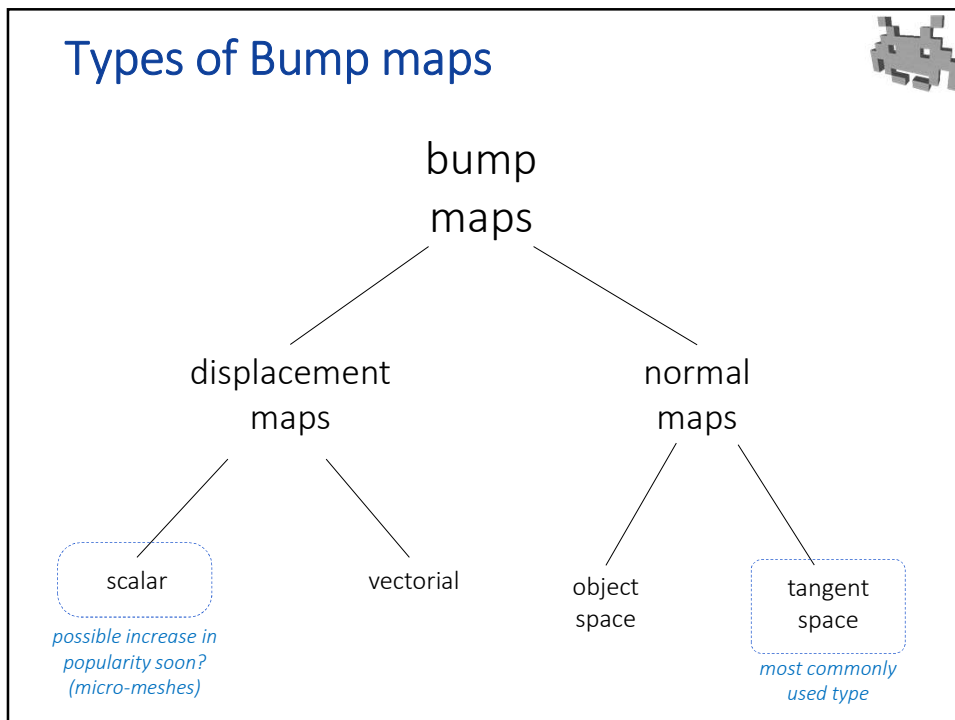
- **macro-structure** of the object , such as ...
  - ...the general shape of the horse
  - ...the general shape of the face
  - ...the general shape of the dragon
- **micro-structure** of the object, such as ...
  - ...the musculature of the horse
  - ...the wrinkles of the face
  - ...the flakes of the dragon
- **nano-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

low-poly mesh

bump-map

material

48



50

## Tassonomy of Bump maps (summary)



- **Bump map:**
  - A texture encoding hi-frequency geometric details
- **Displacement Map:**
  - Geometric details are encoded by storing differences between mesh geometry and detailed surface:
  - as *scalars* (distance along the normal), or as *vectors*
  - used for: on-the-fly re-tessellation, and *parallax mapping* technique
- **Normal Map:**
  - Details are encoded by storing the normals of the detailed surface
  - used for: lighting computation (and affects lighting only)
- **Note: they are not mutually exclusive!**
  - One model can have both
  - (as different texture sheets)

52

## Scalar displacement map : concept

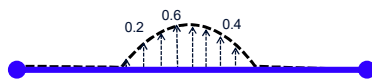


Stores the *distance* of the detailed surfaces from the plain geometry

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



**Detailed surfaces**  
(which I would like to represent)



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

0 0 0 0 0 0 0 1.5 6.6 6.7 5.4 2.0 0 0 0 0 0 0

**displacement map**  
(scalars)

53

## Vectorial displacement map : concept

Store **Vectors** from the plain surface  
to the detailed surfaces

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

"subsquare"!  
Not an height field

**Detailed surface**  
(I would like to model)

**low-poly mesh**  
(approx. of  $\wedge$ ) (here: flat ☹)

**displacement map**  
(vectorial)

54

## 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**  
(approximation of  $\wedge$ ) (here: flat ☹)

**normal map**  
(one normal per texel)

55

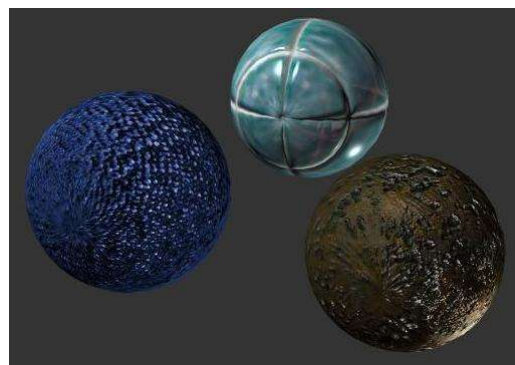
## Normal Map: notes

- Affects the lighting only
  - **not** the parallax
  - **not** the silhouette of the object
  - The lighting reflects the hi-freq detail of the object
    - dynamically (with variable lights!)
  - Total illusion: very convenient
    - If we are not trying to model a macro-structure
- In rendering: use the normal from the texture
  - (for lighting)
  - Instead of the interpolated per vertex normal
- Normals are expressed in cartesian coord
  - Often
    - But not always ( $\exists$  better ways to express unit vectors!)
  - Question: in which space? *See next lecture!*



57

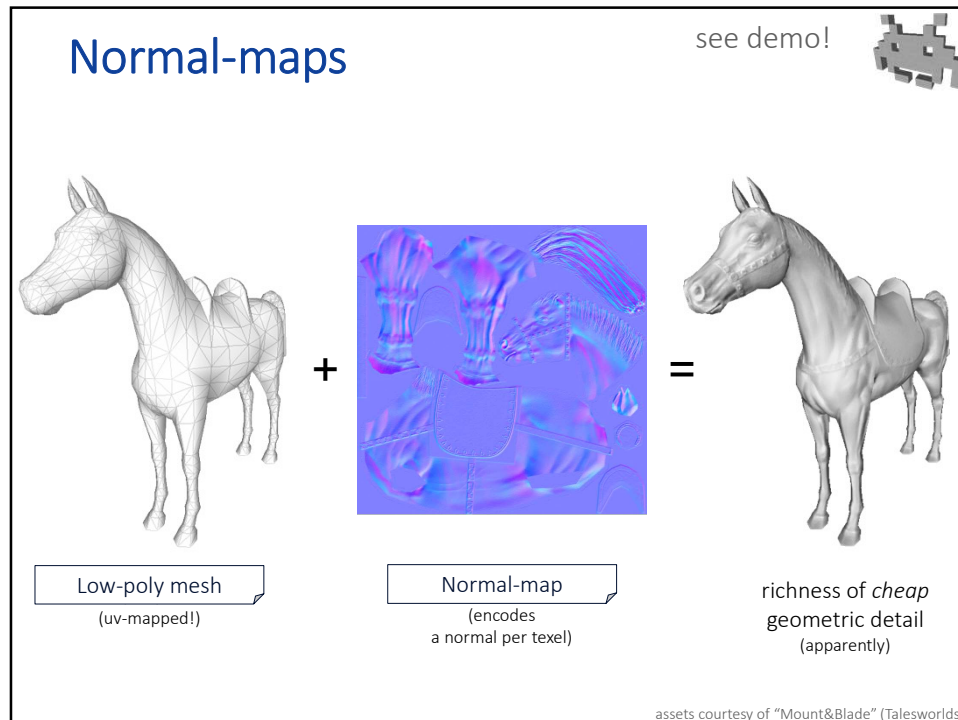
## Normal-maps



Same geometry (a sphere)  
Different normal-maps

58






59

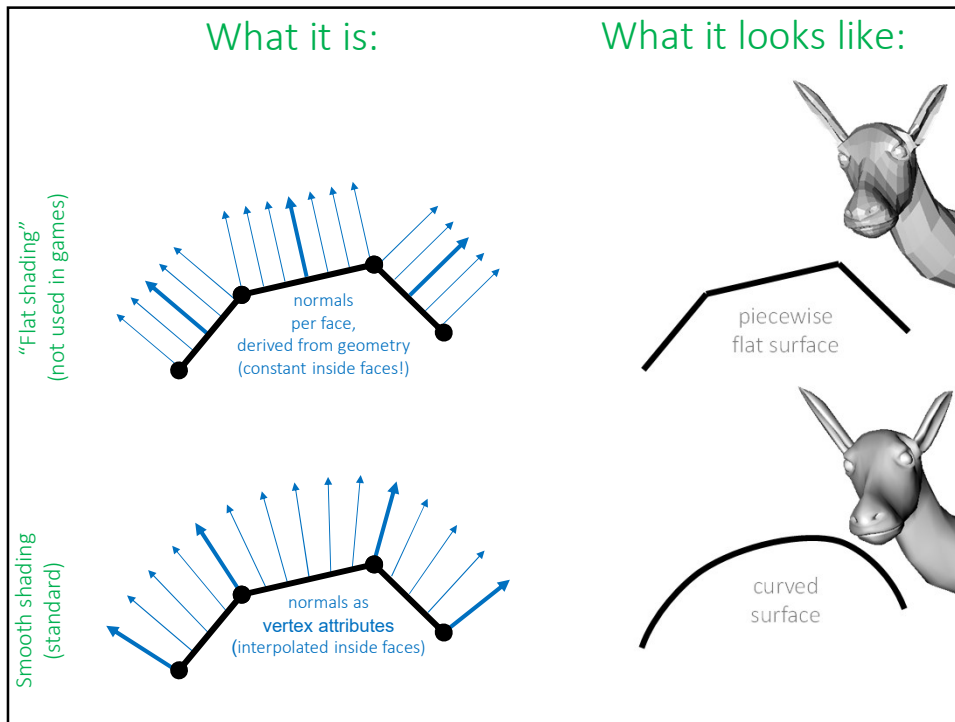
## But normal maps only affect lighting!

Do they provide a convincing illusion of 3D shape details?

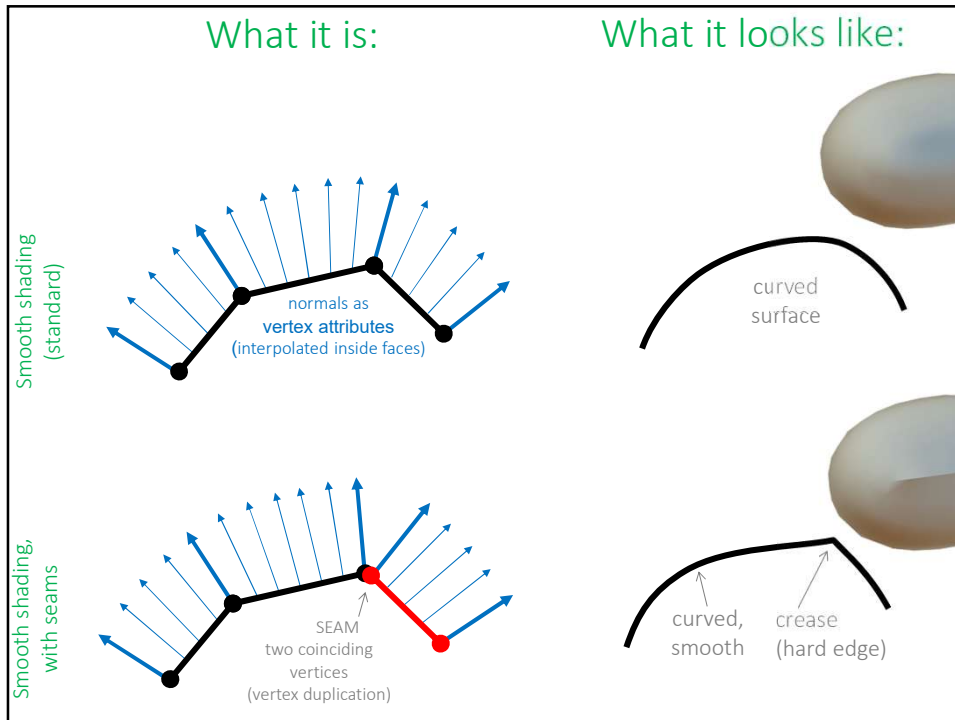
- Normal maps don't affect
  - Parallax, silhouette
- Still, illusion is very convincing
  - (for small scale details, and adequate resolution)
  - The lighting (think *chiaroscuro*) tricks the eye *over* the shape
- It's not the first time we observe this
  - We have seen other cases where the normal of the 3D model (used for lighting) determined independently from the actual 3D mesh shape, determines the appearance of the shape



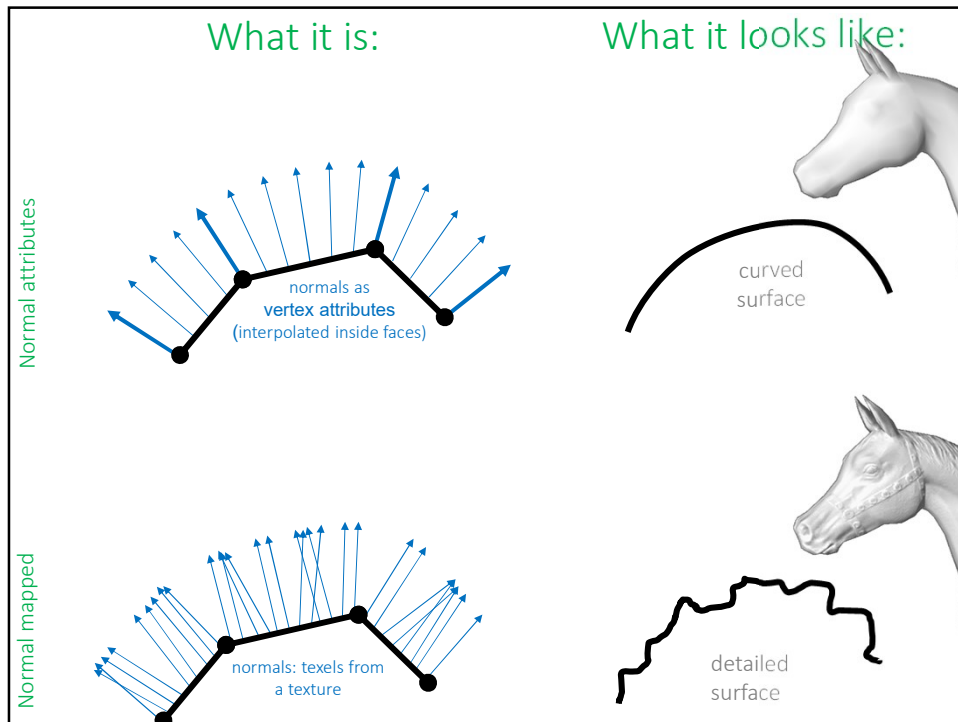
60



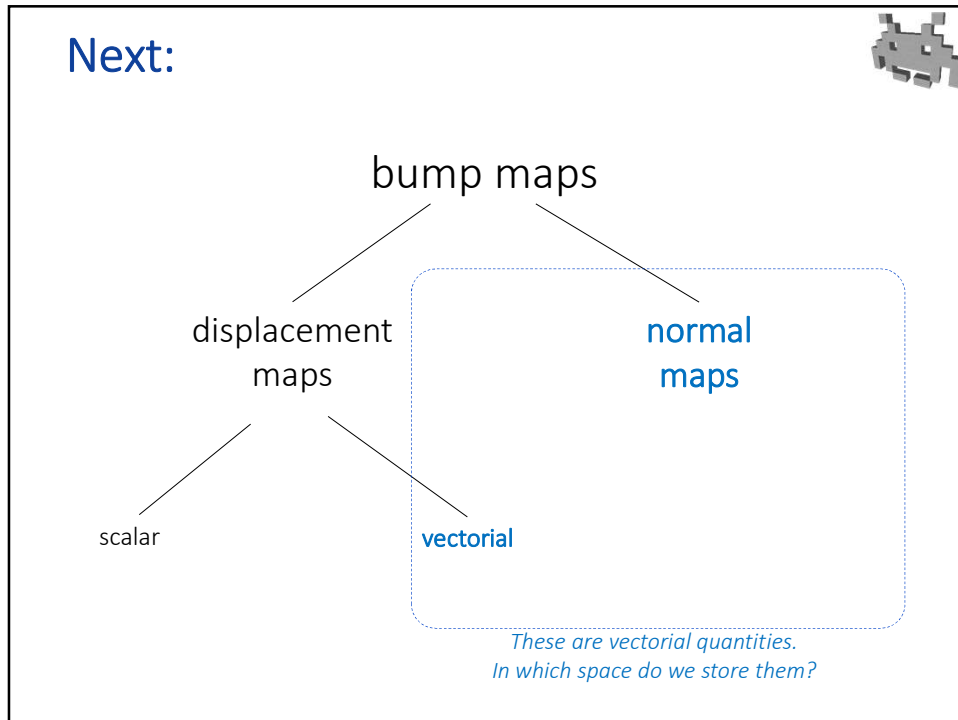
61



62



65



66