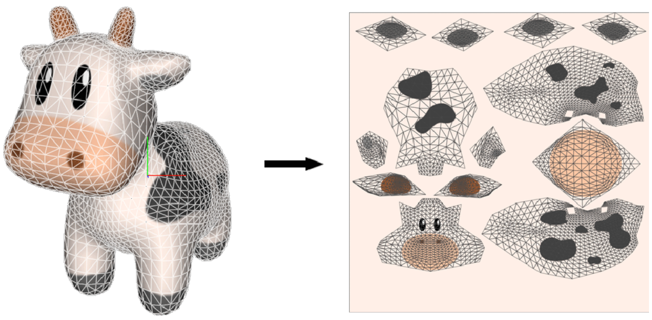
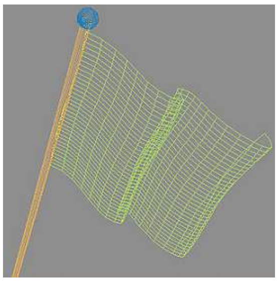
3D VideoGames

Textures

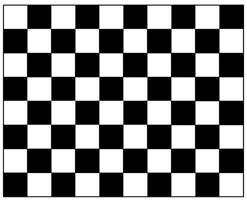
Marco Tarini



Texture mapping



+



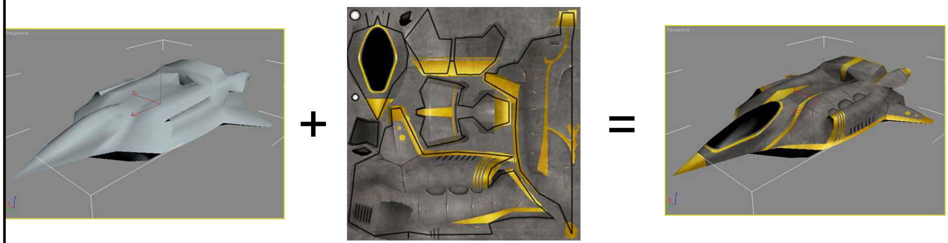
=

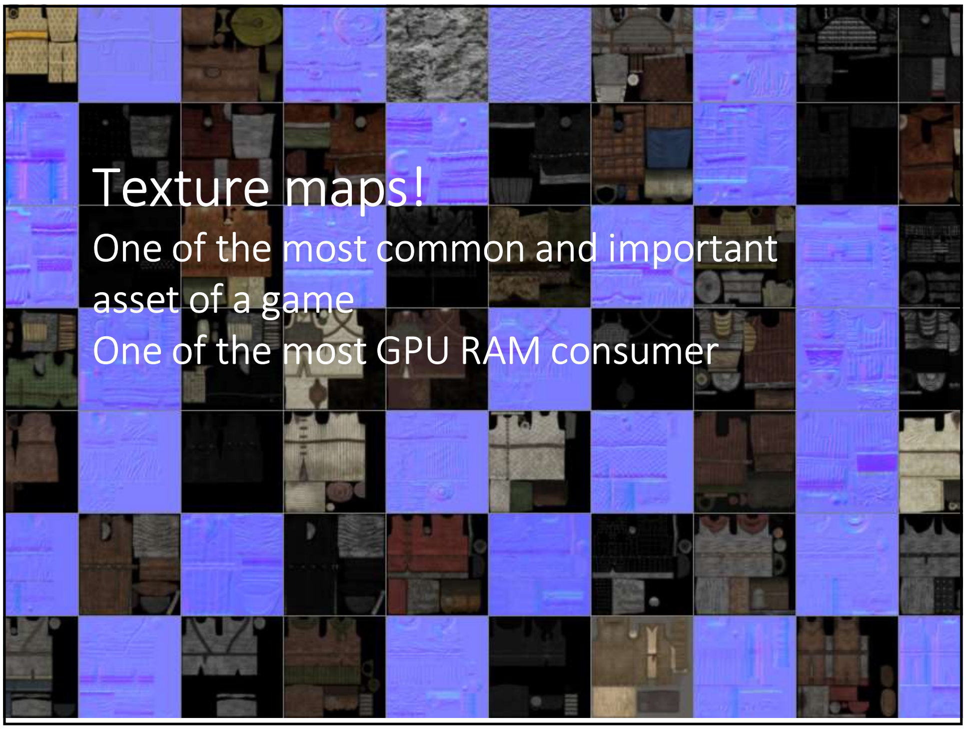
3D geometry
(set of quadrilaterals)

RGB texture 2D

(here: a color-map)

Example (color-map)





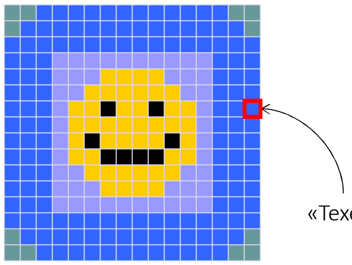
Texture maps!

One of the most common and important asset of a game

One of the most GPU RAM consumer

Texture maps: data structures

- In practice, a **rasterized image**



Texture sheet

Texutres (in games)

- Texture sheet =
  def. of a singale onto the surface (the mesh)
  - Similar purpose to the per-vertex attributes!
  - but...
    - # texels >> # vertices
    - More complex signals!

Texture: regular sampling, and dense (easy to get detail!)
Attributes: irregular samplling (adaptive), and sparse

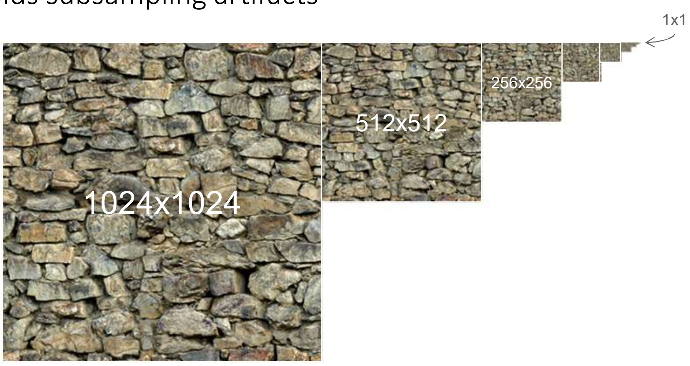
- A texel = a sample of that signal
  - Between samples: (bilinear) interpolation
- Signal sampling:
  - On a regular 2D grid (raster image)
  - At a given resolution (NOT adaptive!)

Signals stored in textures (in games)

- 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: α)
  - The texture is called a "alpha-map" or "cutout-texture" (exp. if 1bit)
- Each texel is a normal (components: *x,y,z*)
  - The texture is called a "normal-map" or "bump-map" (more later)
- Each texel is a value di specularity
  - 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"

MIP map levels

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



Texture maps as assets

- Characteristics:
  - Size:
    - resolution
    - channels (eg: alpha?)
  - MIP-map levels
    - present or not?
  - Compression?
    - e.g. color quantization (“color-map” or “palette”), or compression schemas designed specifically for textures

Constraints:

- Power of 2 for side (U and V)
  - e.g.: 256x256 or 1024x512
  - not so strict requirement today
- res < max
  - ever growing limit
  - today: 8K, 4K, 2K

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

Textures resolution have more impact (quality wise) than Meshes resolution!

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"* !

THE MESH (Geometry + Attributes, Connectivity)

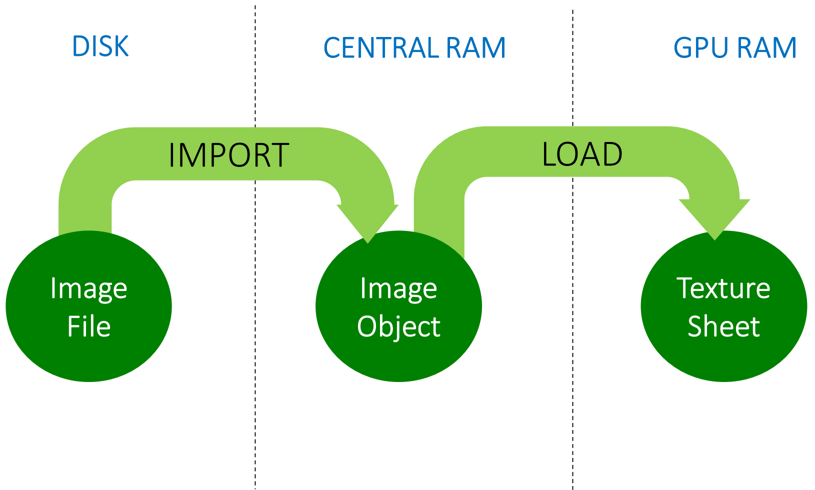
THE "MATERIAL" (Textures, Shaders, Parameters / Settings)

Texture fetch (during rendering, at each pixel)

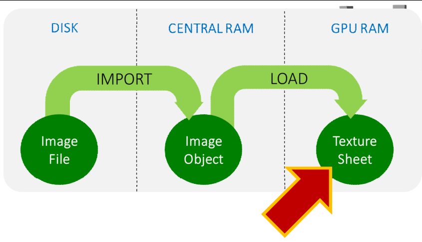
- GPU supported mechanisms to access the texture at a given location (u,v)
- Hardwired steps (can only be turned on/off):
  1. Management of out-of-bound coordinates
     repeat: $u \leftarrow \lfloor u \rceil$ and $v \leftarrow \lfloor v \rceil$
  2. De-normalization of coords, from normalized $[0..1]^2$
     to texel coord $[0..Res_X]\times[0..Res_Y]$
  3. Selection of the appropriate MIP-map level (how?)
  4. Decompression of compressed data
  5. Bilinear interpolation

Life of a Texture in a Game Engine

the same as, basically, any 3D assets

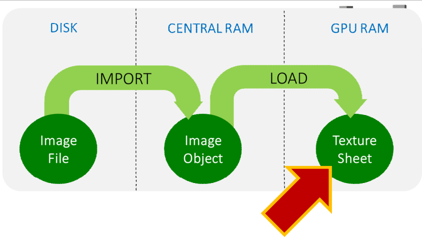


Texture Sheets (in GPU RAM)



- 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 («true-color»)
  - compression: specific texture schemas (see later)
  - resolution: powers of 2

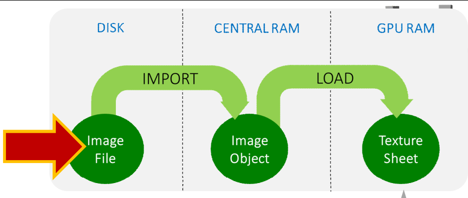
Texture compression (to save GPU RAM)



- We need to guarantee 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 (eg. ¼)
    - Unfavourable compression/loss ratio ☹
    - Most diffuse scheme S3TC, with variants: DXT-1 -2 -3 -4 -5

(DXT-1: yes/no alphas; -2 -3: uniform alphas; -4 -5: smooth alphas)

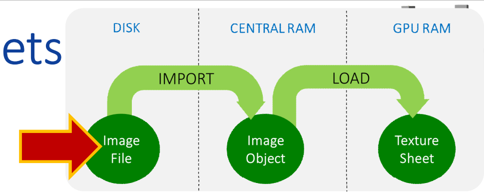
Texture Sheets: files format



- 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)
    ☹ ☹ ☹ compression: verbose
    ☺ Very easy parsing! (no lib needed)

- Specialized for textures: (very used option)
  - .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!)

Texture maps as assets file formats



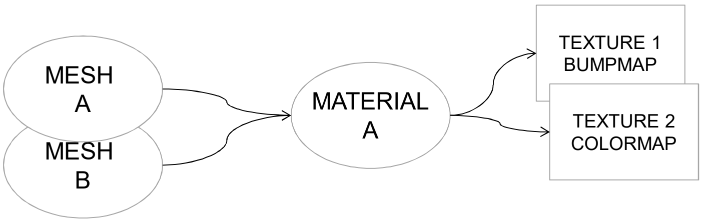
- For generic images
  (decompress the entire image before accessing any pixels)
  - ☺ compression: excellent
  - ☹ loading: heavy:
    - Decompress from RAM, (maybe) recompress in GPU-RAM
  - ☹ MIP-map lvls:
    Controlled by the engine
  - ☺ Resolution: any
    (can pad on load)
- For textures
  (random accessibility to texels, without uncompressing the entire image)
  - ☹ compression: bad
  - ☺ loading: light
    - direct streaming possible
      Disc => RAM => GPU RAM
  - ☺ MIP-map lvls etc:
    Controlled by the artist
  - ☹ Resolution:
    must be a pow of 2

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

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



How is a texture mapped over a a mesh?

The 3D part of game assets

- 3D Models
  i.e. tri-meshes with:
  - per vertex attrib
    - normals, color, AO, …
  - LODs
  - uv-mapping
  - 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

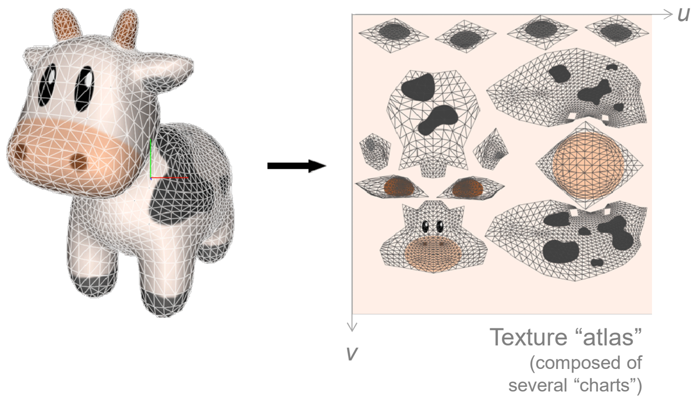
UV-Mapping of a mesh

- A mapping : mesh surface → 2D texture space is needed
  - «parametrization» of the surface

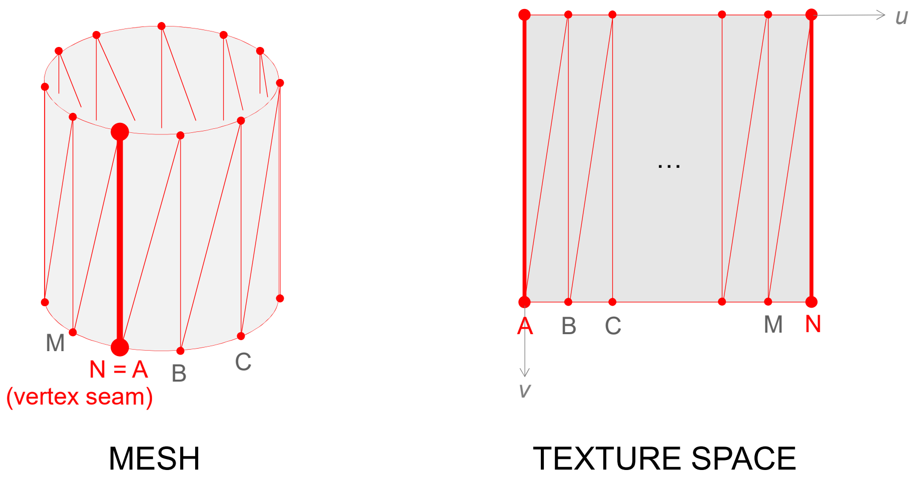
  (2D texture space: $[0..1]^2$)

- Store this mapping as per vertex attribute : (u,v)
  - The «u-v mapping» of the mesh

Modeling task: “u-v mapping” construction



UV mapping: example



Texture space notation

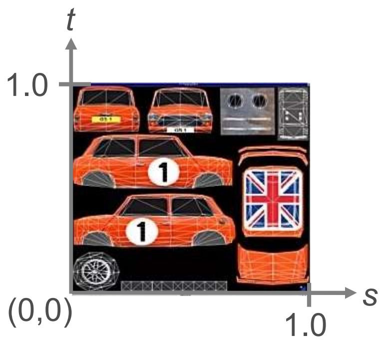
Texture Space (or “parametric space" or "u-v space")

Texture Space = [0,1] x [0,1]

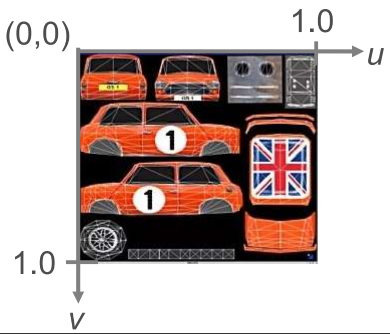
Two notations

Most used
(in game industry)

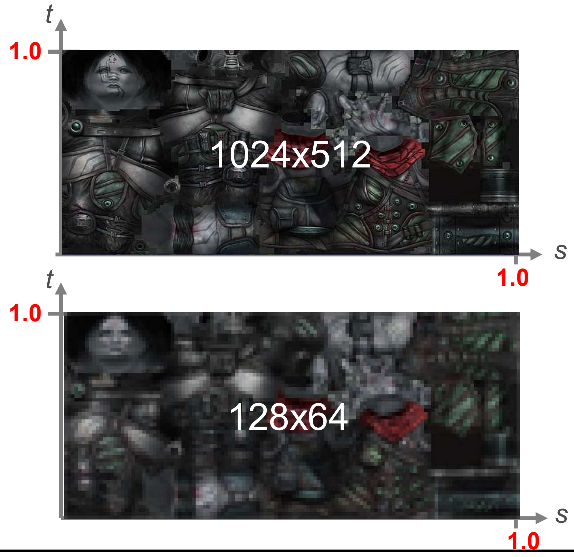
s-t
(es OpenGL)



u-v
(es DirectX)



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



Convenient!
We can reduce texture sheets res (balancing quality / memory) without affecting the mesh 'UV mapping.

Eg: load in GPU RAM only a few smaller MIP-map levels

Construction of a UV-map for a mesh (or, UV-mapping of a mesh)

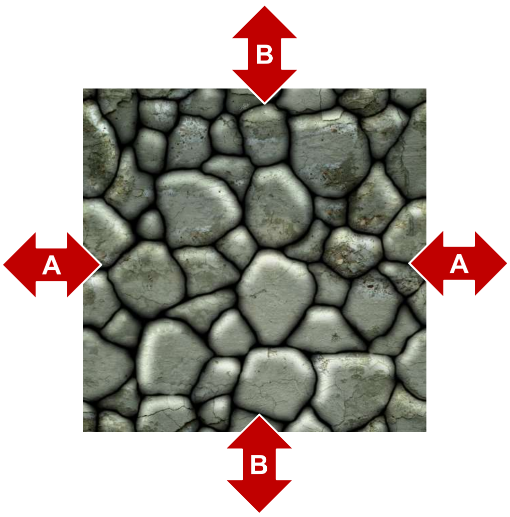
- Typical task of the modeler (digital artists)
  - (semi-)automatic algorithms very studied
- We need to find a spot in the (2D) texture space for each (3D) mesh triangle
- Similar to to:
  - Peel an apple (cutting part)
  - Lay each produced peel in 2D (unfolding part)
  - Pack the peels inside a rectangular 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)

Modeling task: "u-v mapping"

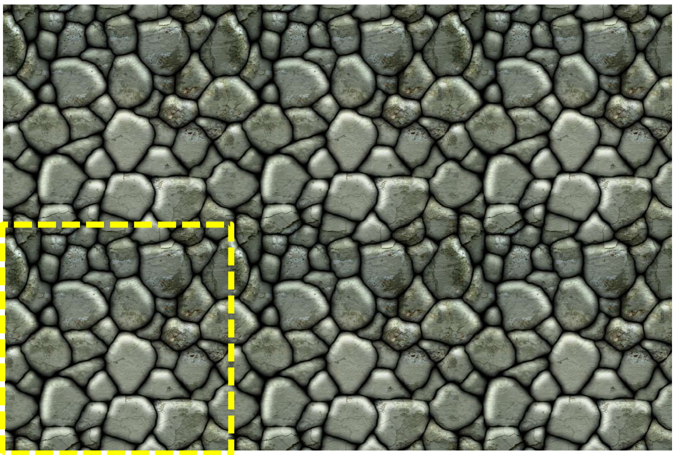
DEMO!

- 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 empty space
    - Assign areas according to necessities
      (important parts → bigger texture space)
      (sampling of the texels becomes adaptive!)

Tileable Textures

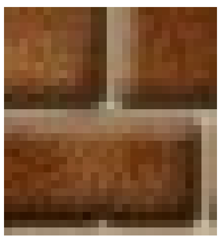


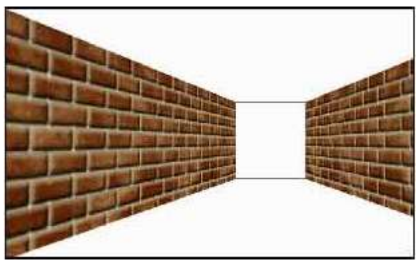
Tileable textures



Tileable textures

- Typical use





Very efficient in space