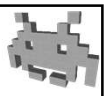
Course Plan

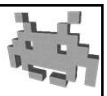


lec. 1: Introduction
lec. 2: Mathematics for 3D Games
lec. 3: Scene Graph
lec. 4: Game 3D Physics +
lec. 5: Game Particle Systems
lec. 6: Game 3D Models
lec. 7: Game Materials
lec. 8: Game Textures
lec. 9: Game 3D Animations
lec. 10: 3D Audio for 3D Games
lec. 11: Networking for 3D Games
lec. 12: Interactive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

★ appearance

69

Signals typically stored in textures (in videogames) 3/4

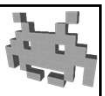


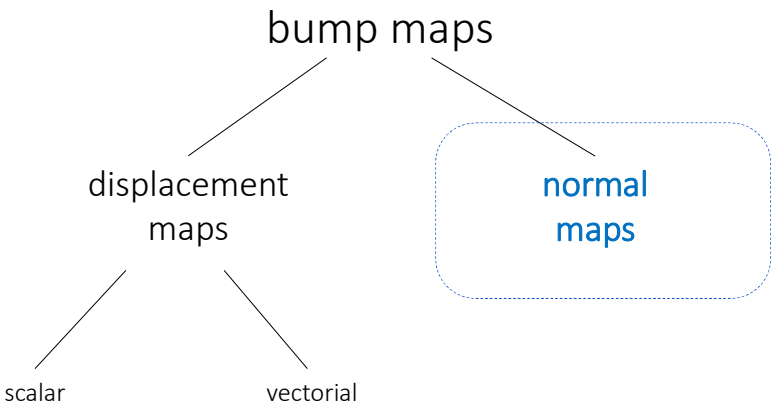
Small scale geometric detail (see Bumb-maps)

- Each texel = a distance from a surface value
  - The texture sheet is a "displacement map" (sometimes "height map")
- Each texel = a normal of the (wrinkled) surface
  - The texture sheet is a "normal map"
  - If not specified otherwise, a Tangent space (sometimes, Object space)

70

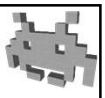
Tassonomy of Bump maps





71

Normal Map: notes

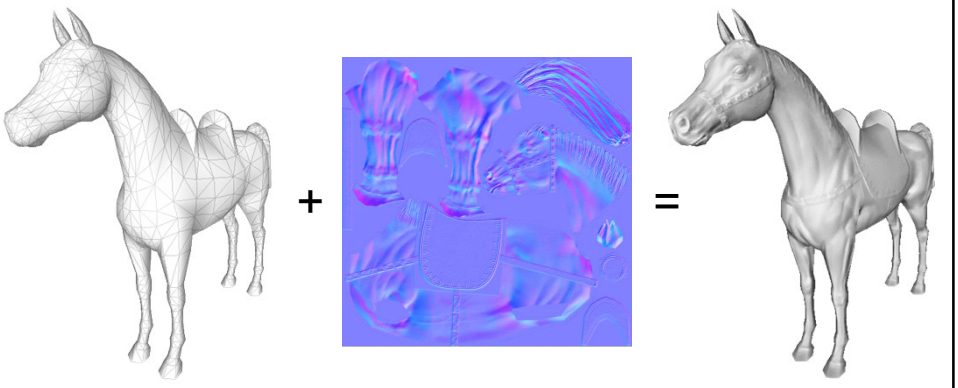


- Affects the lighting only
  - not the parallax
  - not the silhouette of the object
  - The lighting reflects the hi-freq detail of the object
    - dynamically (with variable lights!)
  - Resulting illusion: still convincent
    - If we are not trying to model a macro-structure
- In rendering: use the normal from the texture
  - (for lighting)
  - Instead of the interpolated per vertex normal
- Normals can be expressed simply in cartesian coord
  - But not always (∃ better ways to express unit vectors!)

72

Normal-maps

see demo!



Low-poly mesh
(uv-mapped!)

Normal-map
(encodes
a normal per texel)

richness of *cheap*
geometric detail
(apparently)

assets courtesy of "Mount&Blade" (Talesworlds)

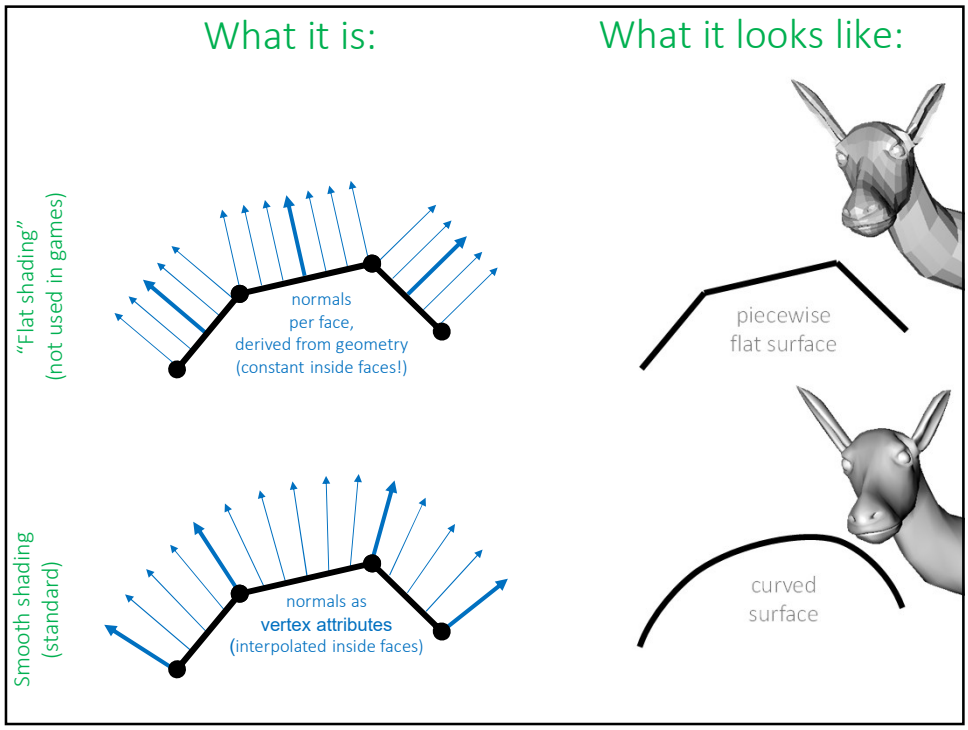
74

Normal maps only affect lighting

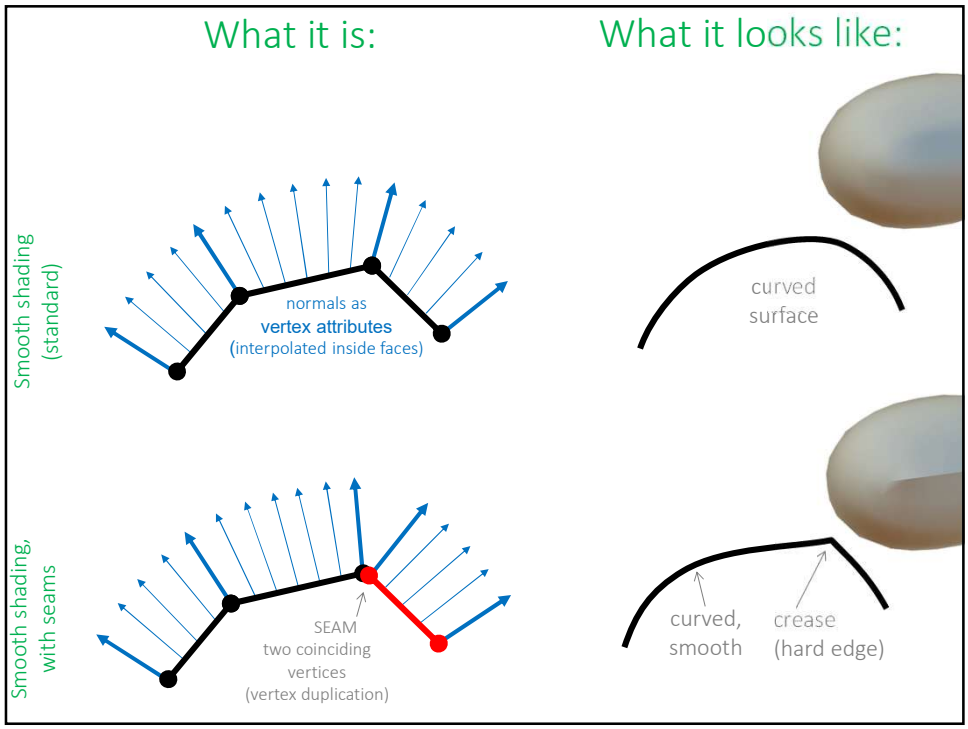
Do they provide a convincing illusion of 3D shape details?

- Normal maps won't affect:
  - Parallax, or object silhouette
- Still, illusion is very convincing
  - for small scale details, and adequate mesh/texture resolution
  - The lighting (think "*chiaroscuro*")
    tricks the eye notwithstanding the geometric shape
- It's not the first time we observe this in this course
  - We have seen other cases where the normal used for lighting is chosen independently from the actual 3D shape of the mesh
  - Normals (as vertex attributes, or texels value) don't have to be the real surface normal: small discrepancies are ok
  - For example…

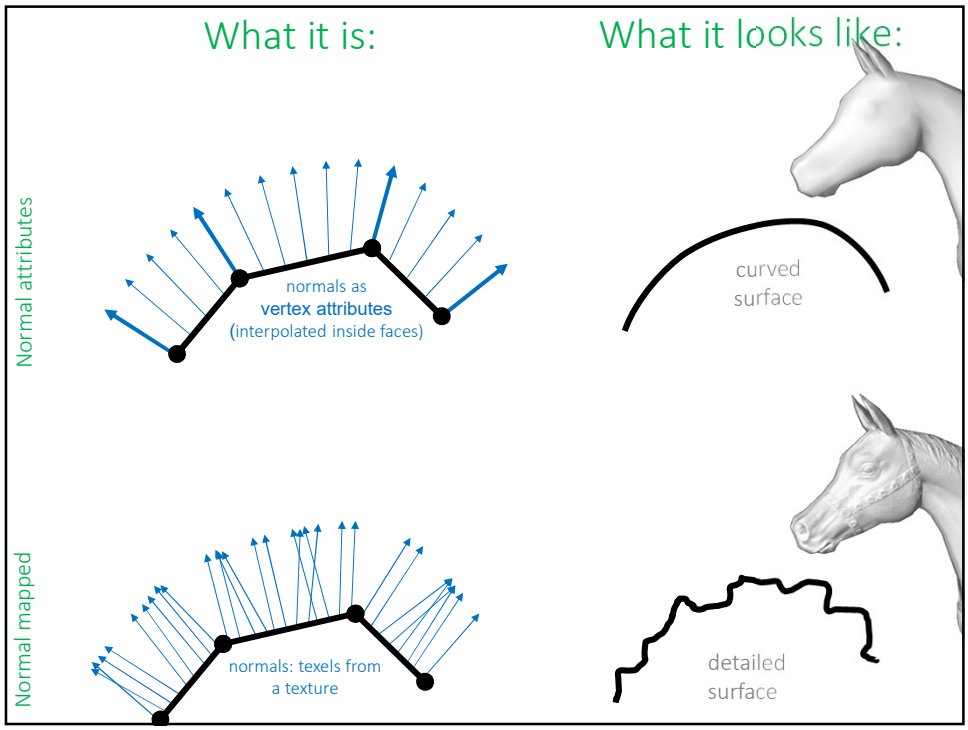
75


What it is:
What it looks like:
"Flat shading"
(not used in games)
normals
per face,
derived from geometry
(constant inside faces!)
piecewise
flat surface
Smooth shading
(standard)
normals as
vertex attributes
(interpolated inside faces)
curved
surface

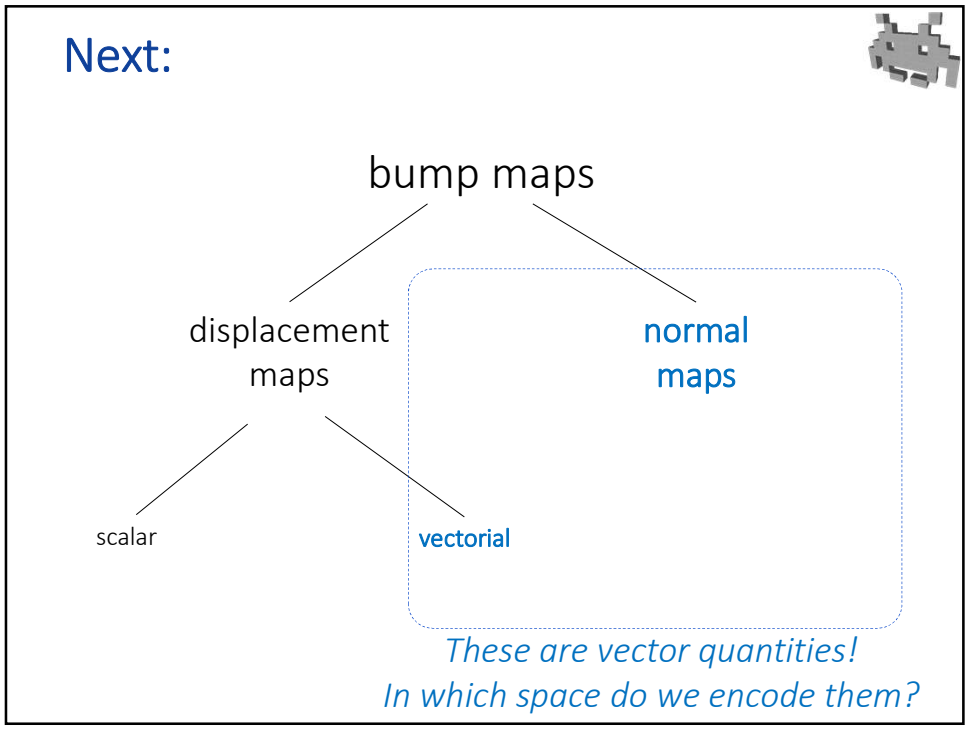
76


What it is:
What it looks like:
Smooth shading
(standard)
normals as
vertex attributes
(interpolated inside faces)
curved
surface
Smooth shading,
with seams
SEAM
two coinciding
vertices
(vertex duplication)
curved,
smooth
crease
(hard edge)

78

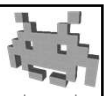

What it is:
What it looks like:
Normal attributes
normals as
vertex attributes
(interpolated inside faces)
curved
surface
Normal mapped
normals: texels from
a texture
detailed
surface

80


Next:
bump maps
displacement
maps
normal
maps
scalar
vectorial
These are vector quantities!
In which space do we encode them?

81

Normal Maps: in which space are the normals encoded?

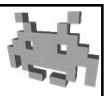


i.e, texture normals and mesh vertices are expressed in the same space

- If object space: Object-Space Normal-Maps
  - ☺ the per-vertex normal becomes unnecessary!
    - The normal from texture substitutes it
  - ☺ Trivial to apply (during rendering)
    - just use the normal fetched from the texture for lighting
  - ☹ normal-map is bound to a specific object
    - cannot be reused for different objects
  - ☹ Each region of the normal map is bounded to one specific area region of the object!
    - Injective UV-maps only!
    - e.g. no tiling, no exploitation of symmetries

82

Normal Maps: in which space are the normals encoded?



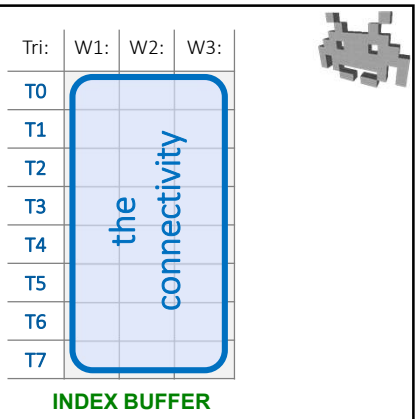
- Tangent space: Tangent Space Normal-Maps (the standard kind, in games)
  - ☹ extra attributes are now needed per vertex:
    - The tangent space:
      - Normal direction
      - Tangent direction
      - Bitangent direction

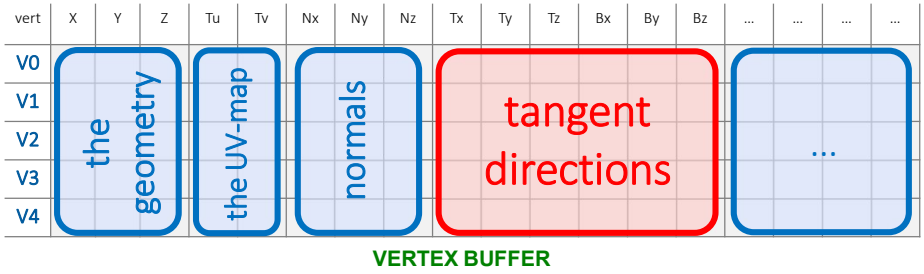
basically, a TS normal map specifies how to modify the per-vertex normal instead of replacing it

  - ☺ normal-map can be shared by different objects
  - ☺ non injective UV-maps can be used
    - e.g., the normal-map can be tiled
    - e.g., symmetries can be exploited
  - ☺ normal-map is independent from the mesh
    - e.g. can be constructed without knowing the mesh

83

Tangent directions are mesh-attributes

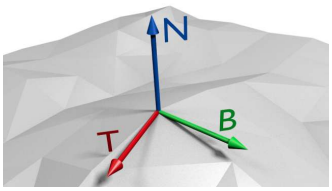




84

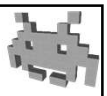
Tangent space (aka TBN space)

- A vector space defined ∀ point of the surface:
  - Z axis: Normal
    - orthogonal to surface
  - X and Y axis: tangent vectors
    - parallel to the surface
    - X = Tangent
    - Y = "Bi-Tangent"
      (sometimes, but inappropriately: *Bi-Normal)

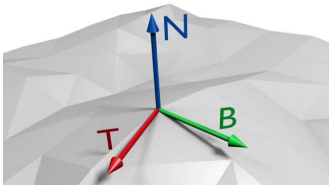


85

Tangent space (aka TBN space)



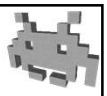
- How to store them?
  - As 3 versors stored as (per-vertex) **attributes**
    - So, they are interpolated inside faces (like any other attribute)



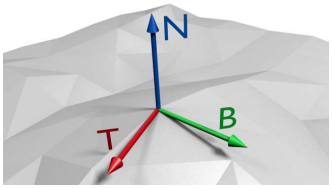
  - Optimizations are possible!
    - Not necessarily stored as 3 vectors (9 scalars)
    - E.g.: instead of storing B, we store N and T, then B = N × T
  - Note: they have discontinuities
    → seams (vertex duplications) are necessary
    - In first approximation, the same ones required by the UV-map (but non only! why?)

86

Tangent space (aka TBN space)



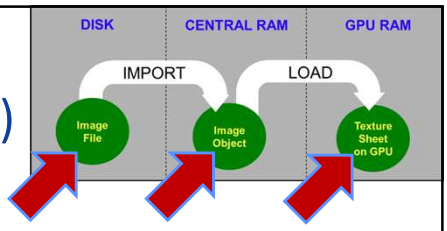
- How to compute them?
  - Normal
    - as usual (see lecture on mesh)
  - Tangent & Bi-Tangent
    - determined by the UV-map!
    - T = gradient of U coordinate
    - B = gradient of V coordinate



- details:
  - All three are defined (and constant) inside faces, then averaged at vertices (see per-vertex normal computation)
  - T,B,N can be *only approximatively* orthogonal to each other
  - T,B,N reference frame can be left-handed or right-handed (even different "handedness" in different parts of the same mesh)

87

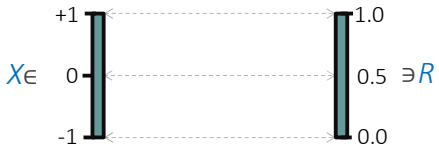
Normal-map: storage (as an rgb image)



- Idea: store it as an RGB texture
  - $R \leftrightarrow X$
  - $G \leftrightarrow Y$
  - $B \leftrightarrow Z$

(because normals are unit vectors)

- but $X,Y,Z \in [-1,+1]$ while $R,G,B \in [0,+1]$
  thus, a linear mapping is needed:

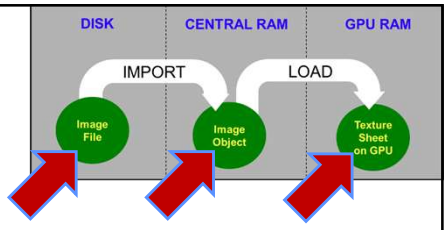


$$R = \tfrac{1}{2}(X + 1)$$
$$X = 2R - 1$$

- Advantage: we can use compressed interchange file format intended for RGB images (e.g. png, jpg) for textures
- Note: other, more efficient representations of versors exists

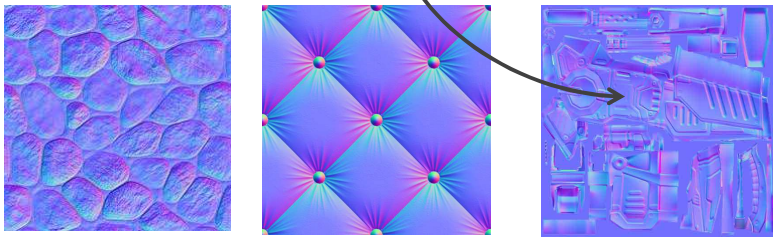
88

Normal-maps: storage (as a texture)



- Examples of tangent space normal-map

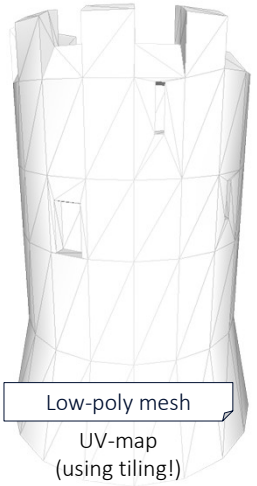
Prevailing normal : X=~0 , Y=~0 , Z=~1
⇒
Prevailing color: R =~0.5 , G=~0.5, B=~1
(~light blue)



89

E.g.: Tiled (tangent space) Normal Maps

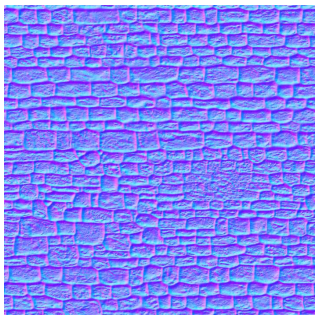
not even possible, with object-space NM!



Low-poly mesh

UV-map
(using tiling!)
Tangent dirs.

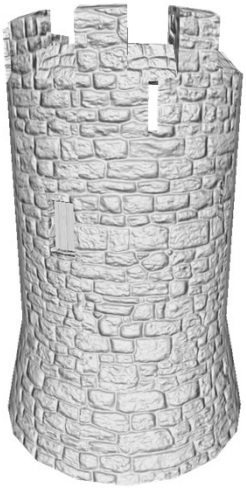
+



Normal-map

Tileable!

=



assets courtesy of "Mount&Blade" (Talesworlds)

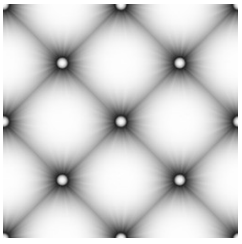
90

Bump-maps assets at a glance
(can you tell which is which?)

Tangent Space
Normal map

the default kind

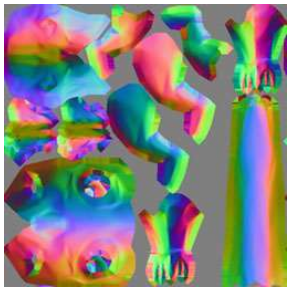
Object Space
Normal map



Displacement
Map (scalar)

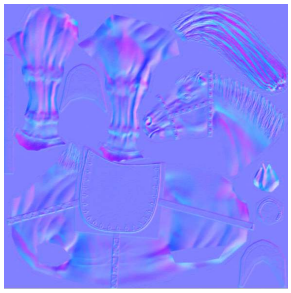
91

Spot the difference



Object Space Normal map

1:1 UV-map
right leg ≠ left leg



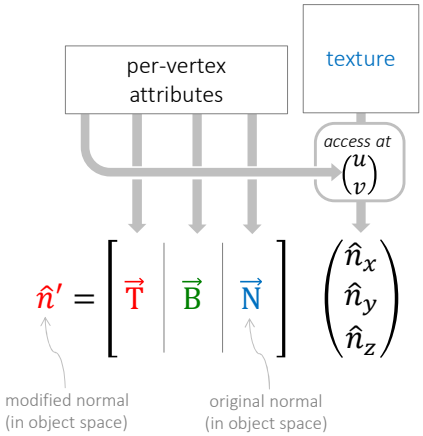
(Tangent Space) Normal map

UV-map NOT injective
Exploited symmetries!
Left side of head = right side of head

92

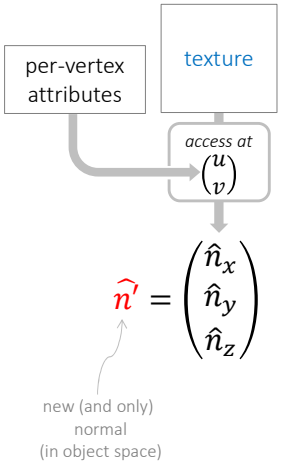
Normal Maps types: implementation

Tangent Space Normal-maps



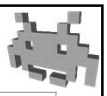
$$= \hat{n}_x \vec{T} + \hat{n}_y \vec{B} + \hat{n}_z \vec{N}$$

Object Space Normal-maps



93

Normal map comparison (a summary)

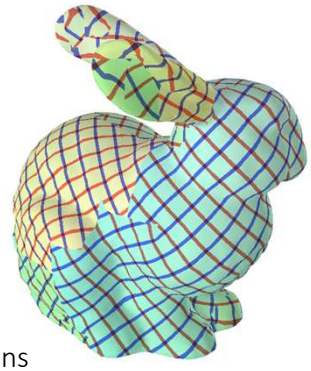


Object Space Normal map:	Tangent Space Normal map:
Replaces the normals of the object	Modifies the normals of the object
No normal attribute required on the mesh anymore	Requires two extra attributes on the mesh: T an B versors (in addition to the normal)
Constructing the texture requires to know the mesh it will be applied to	Textures can be constructed independently from the mesh (just like a color map!)
E.g., a normal map cannot be constructed from a displacement map (w/o the mesh)	E.g., a normal map can be constructed from a displacement map
It's impossible to share a normal map between models (barring exceptions)	Normal maps can be shared between different models
"unwrapping" UV-maps required (barring exceptions)	Can be applied to non-injective UV-maps
E.g., no tiled textures. E.g., no symmetry exploitation	E.g., tiled textures ok, E.g., symmetry exploitation ok
E.g., east-wall and south-wall of a castle: different normal maps required	E.g., east wall and south wall of a castle: same normal map.
Looks colorful (if encoded as RGB)	Looks azure-ish (if encoded as RGB)

THE STANDARD TYPE, IN GAMES

94

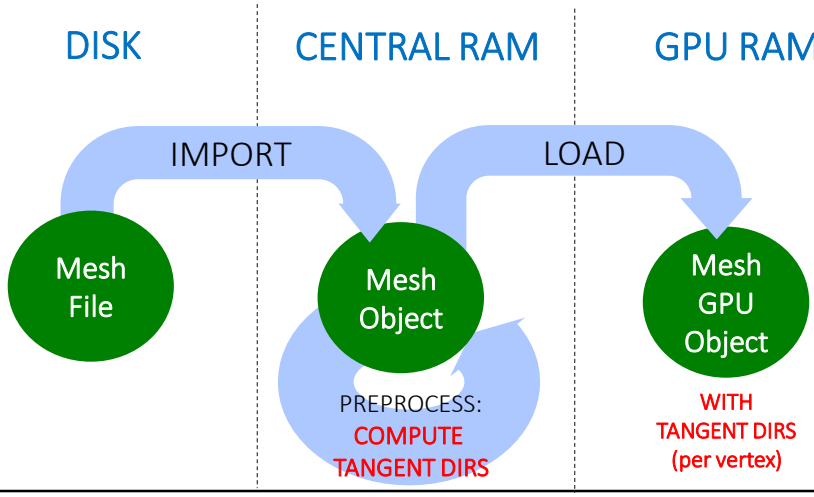
How to construct T and B vectors from the UV-map (a mesh preproc.)



- Concept (a mental experiment)
  - STEP 1: color a texture with a grid
    - horizontal blue lines = U direction
    - vertical red lines = V direction
  - STEP 2: apply it to the Mesh!
  - STEP 3: look at it:
    - the T vectors are the Blue lines directions
    - the B vectors are the Red lines directions
- T and B directions are defined in a triangular face
  - then, they are averaged at vertices
  - (just like the normal directions!)

95

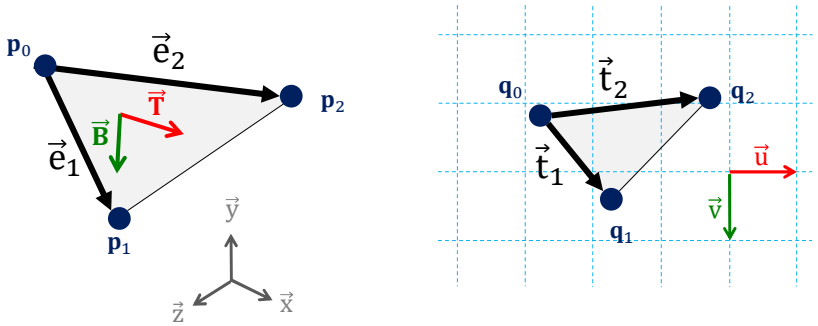
Extracting Tangent/Bitangent directions from the UV-map



96

Extracting T and B vectors from the UV-map (in a triangle)

- Object Space (3D)
- Texture Space (2D)



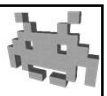
Idea:

$\vec{u}$ is some linear combination of $\vec{t}_1$ and $\vec{t}_2$ $\Rightarrow$ $\vec{T}$ is the same linear combination of $\vec{e}_1$ and $\vec{e}_2$

$\vec{v}$ is some linear combination of $\vec{t}_1$ and $\vec{t}_2$ $\Rightarrow$ $\vec{B}$ is the same linear combination of $\vec{e}_1$ and $\vec{e}_2$

97

Extracting T and B vectors from the UV-map (in a triangle)



- Input: 3D vertices $\mathbf{p}_{0,1,2}$ and 2D vertices $\mathbf{q}_{0,1,2}$
- Find 3D *edge vectors* $\vec{e}_{1,2}$
  and 2D *edge vectors* $\vec{t}_{1,2}$
- Find scalars a, b and c, d such that…

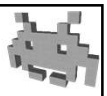
$$a\,\vec{t}_1 + b\,\vec{t}_2 = \vec{u} = \begin{bmatrix}1\\0\end{bmatrix} \qquad c\,\vec{t}_1 + d\,\vec{t}_2 = \vec{v} = \begin{bmatrix}0\\1\end{bmatrix}$$

- Then

$$\vec{T} = a\,\vec{e}_1 + b\,\vec{e}_2 \qquad \vec{B} = c\,\vec{e}_1 + d\,\vec{e}_2$$

98

Extracting T and B vectors from the UV-map (in a triangle)



- Input: 3D vertices $\mathbf{p}_{0,1,2}$ and 2D vertices $\mathbf{q}_{0,1,2}$
- Find $\vec{e}_1 = \mathbf{p}_1 - \mathbf{p}_0$ $\qquad \vec{t}_1 = \mathbf{q}_1 - \mathbf{q}_0$
  $\vec{e}_2 = \mathbf{p}_2 - \mathbf{p}_0$ $\qquad \vec{t}_2 = \mathbf{q}_2 - \mathbf{q}_0$
- Find scalars a, b and c, d such that…

in matrix form: solve with a 2x2 matrix inversion

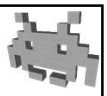
$$\left[\vec{t}_1 \,\middle|\, \vec{t}_2\right]\begin{bmatrix}a & c\\ b & d\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \Rightarrow \begin{bmatrix}a & c\\ b & d\end{bmatrix} = \left[\vec{t}_1 \,\middle|\, \vec{t}_2\right]^{-1}$$

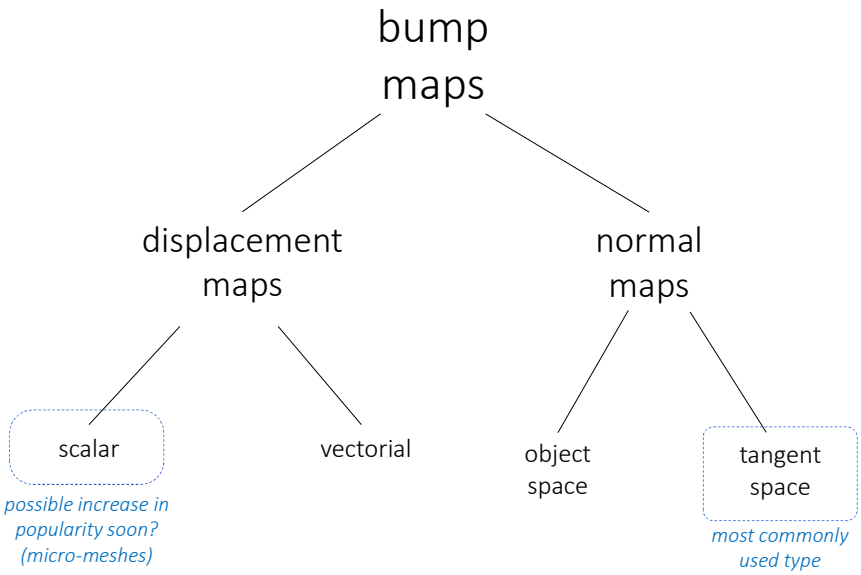
- Then

$$\vec{T} = a\,\vec{e}_1 + b\,\vec{e}_2 \qquad \vec{B} = c\,\vec{e}_1 + d\,\vec{e}_2$$

99

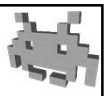
Types of Bump maps: summary

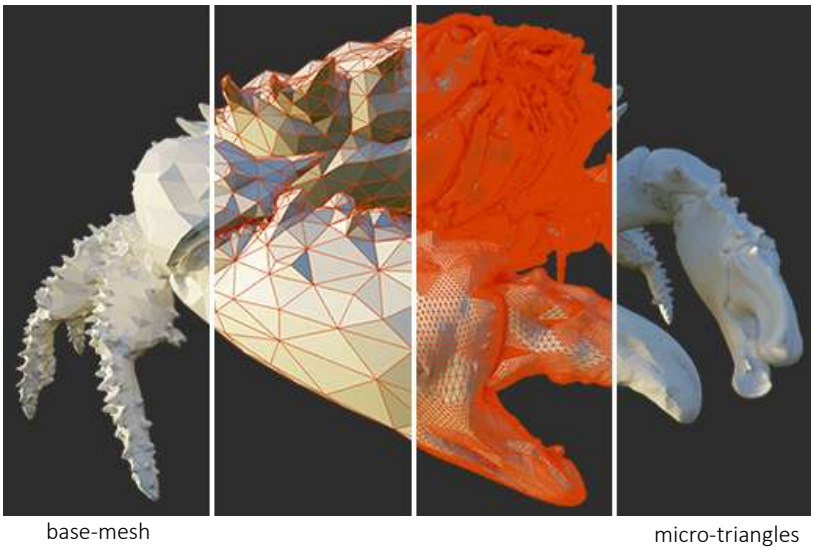




100

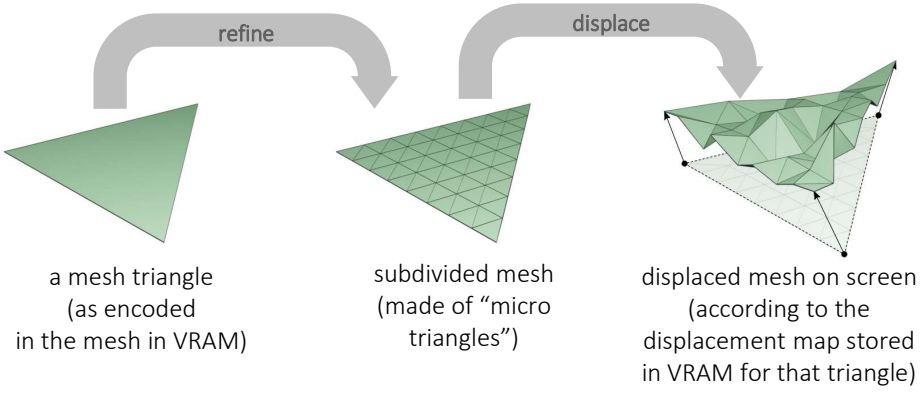
μ-meshes (micro-meshes)





101

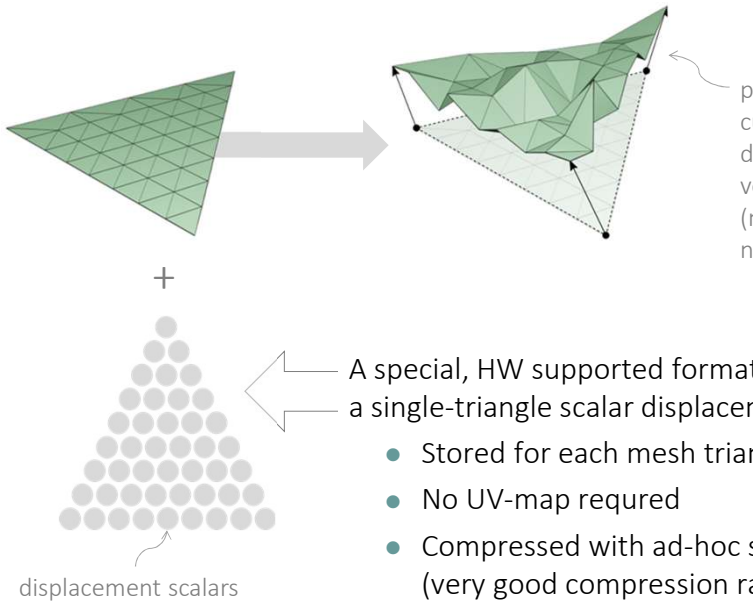
Micro-meshes (μ-meshes)



- A mesh with a specially formatted scalar displacement map
- Displacement map stored per triangle: no UV-map necessary
- During rendering, in GPU: refine + displace
- Proprietary by NVIDIA

102

μ-meshes displacement

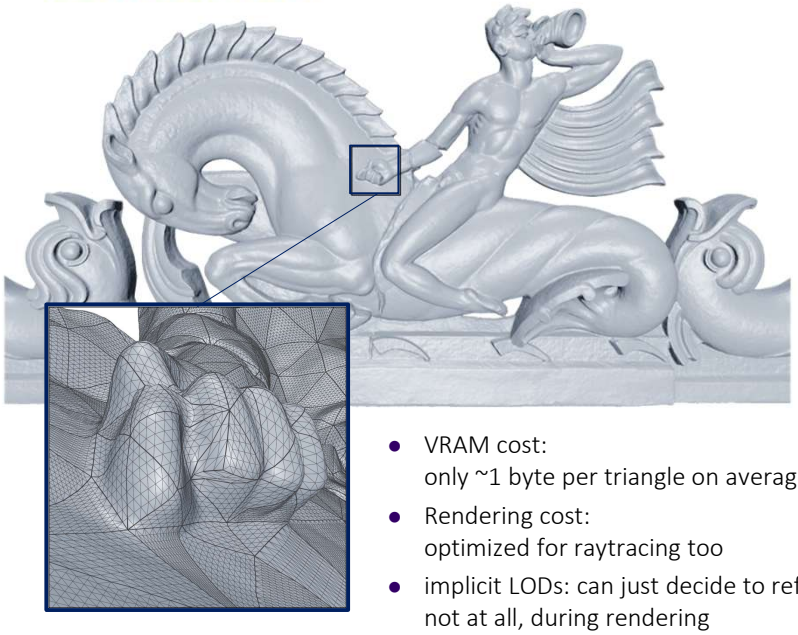


A special, HW supported format for a single-triangle scalar displacement-map

- Stored for each mesh triangle
- No UV-map requred
- Compressed with ad-hoc schema (very good compression rates)

105

MicroMeshes

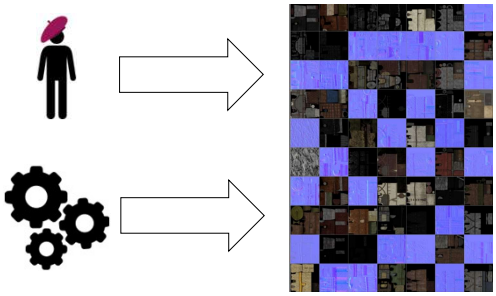


- VRAM cost:
  only ~1 byte per triangle on average!
- Rendering cost:
  optimized for raytracing too
- implicit LODs: can just decide to refine less, or not at all, during rendering

106

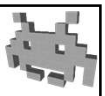
Something about authoring textures

- How are textures authored / created?
  - As a part of asset production pipelines
  - Let's see the basics on authoring of color-maps (or other material), normal maps, and more

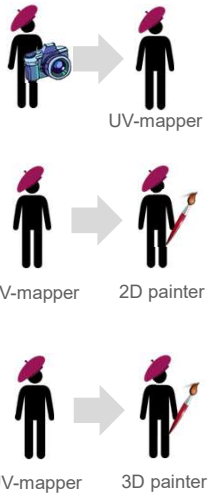


108

RGB-maps authoring: when (and how)?

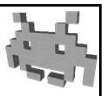


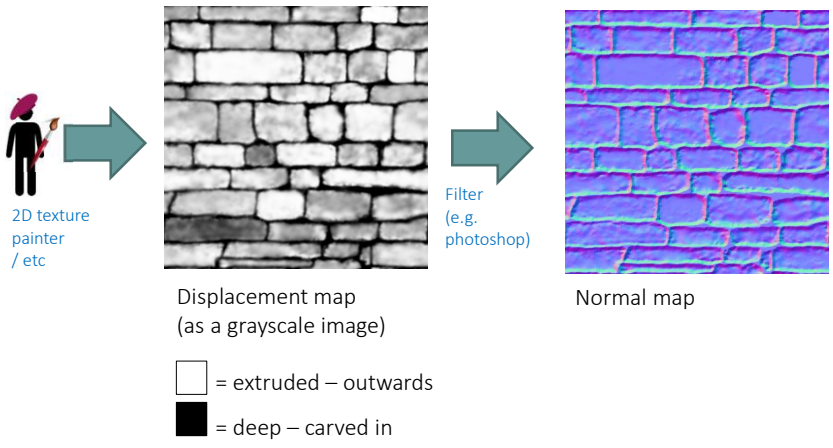
- Image *first, then* UV-map
  - e.g., images that are photos
  - e.g., tileable images
- UV-map *first, then* paint in 2D
  - paint with 2D app (e.g. photoshop)
- UV-map *first, then* paint in 3D
  - paint with-in 3D modelling software,
  - Paint strokes (e.g. brushes) done on the model, recorded in the texture



109

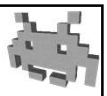
How are normal-maps obtained? (1/5) from a displacement map





111

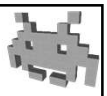
How are normal-maps obtained? (1/5) from a displacement map



- Input: a scalar displacement map
  Output: a normal map
  - a texel at coords *u,v* corresponds to a 3D point (*u* , *v* , height[*u*,*v*])
- Algorithm (2D image processing):
  - ∀ texel **t** of displacement map, compute best fitting plane around **t**
    - Consider all 3D points in a 3×3 patch surrounding **t**
      - or 5×5, or 7×7…
    - Find plane minimizing the summed squared distance from them
    - It's a least-squares minimization problem
  - The normal of this plane is the normal for **t**
- Resulting normal map is expressed in tangent-space
  - By definition! (one big advantage of Tangent Space NM)
  - Can be converted into Object-Space if needed (for a given UV-mapped mesh – injective maps only of course)

112

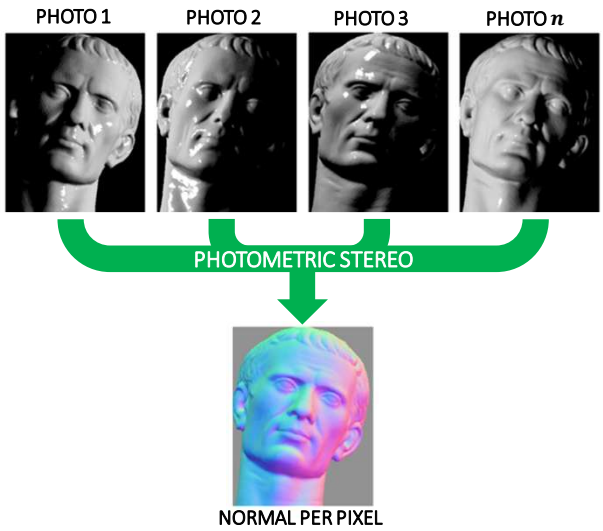
How are normal-maps obtained? (2/5) painting (sculpting?) on 3D



- Direct painting of normal- on the model
  - (can be done, e.g., with Z-brush, Blender…)
  - Similar to: 3D painting of color-maps
    - but artist paints geometric surface orientations, not colors
  - Similar to: mesh sculpting
    - but, for each stroke, the system updates the normal on the texture-map, not the geometry on the mesh

113

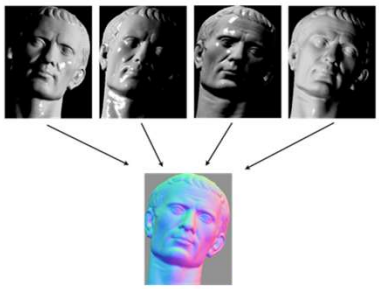
How are normal-maps obtained? (3/5) captured from reality



114

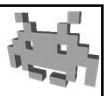
How are normal-maps obtained? (3/5) captured from reality

- Example: "Photometric Stereo"
  - a form of "inverse lighting"
  - Aka: "shape from shading"
  - a computer vision technique
- Input: *n* real images
  - Exact same viewpoint
  - Different illumination
  - (possibly, controlled and known)
- Output: a Normal Map
  - expressed in… image space
  - can be converted in object space, then in tangent space



115

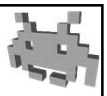
How are normal-maps obtained? (3/5) captured from reality



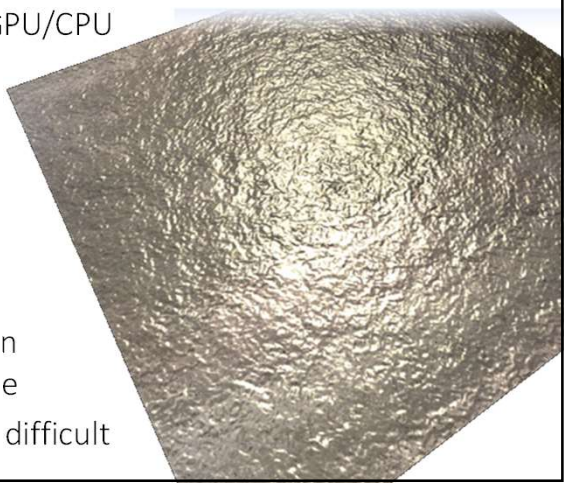
- Normal map estimation from images
  - Traditionally, many pictures are required in input
  - Traditionally, controlled illumination is required (I must place lights in known position)
  - With Machine Learning, it's becoming possible to use a single image with natural illumination
- Basic Idea:
  - input: a photo of a brick wall
  - output: a diffuse map + a normal map + a specular map etc. In short, an entire Material
- Emerging area. Many (commercial) tool already exists.

116

How are normal-maps obtained? (4/5) procedural generation (not frequent)

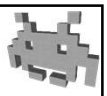


- Usual considerations about procedurality:
  - Saves RAM, costs GPU/CPU
  - Can be baked in preprocessing (becomes an asset)
  - Or, can be generated at run-time
  - Bonus: no repetition artifacts, animatable
  - Problem: control is difficult



117

How are normal-maps obtained? (5/5) from a high-resolution model



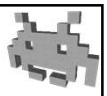
Texture baking

- aka: detail recovery / “detail texture” synthesis

- input:
  - a hi-res mesh A with per-vertex attributes
  - a low-poly mesh B, with an injective UV-map
  - with A and B representing the same object
- output:
  - one or more textures for B storing the attributes of A
- a fully automatic process!

118

Texture baking: texture synthesis from hi-res models

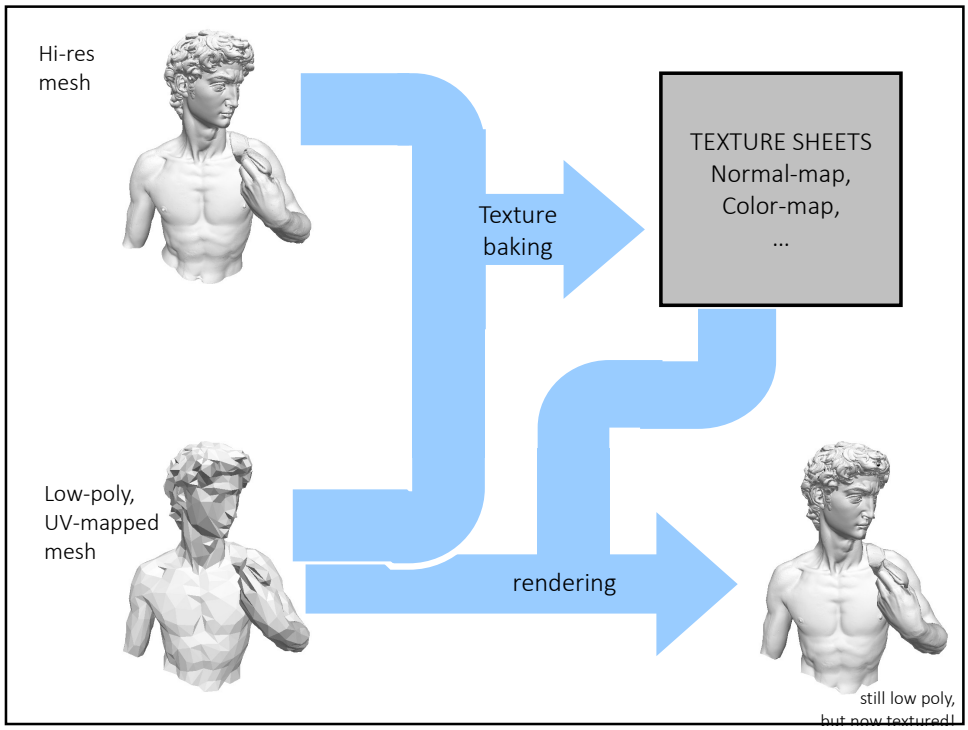


- examples of inputs A & B:
  - low-poly mesh A obtained from a hi-res mesh B via automatic simplification or manual retopology
  - hi-res mesh B obtained from low-poly mesh A via sculpting
- examples of output:
  - attributes = normals → an object-space normal map is produced
  - attributes = base colors → a diffuse maps is produced
  - store distances between A and B (no attribute required) → a displacement map is produced
  - attributes = baked (global) lighting / AO → a light-map / AO-map is produced (see later)

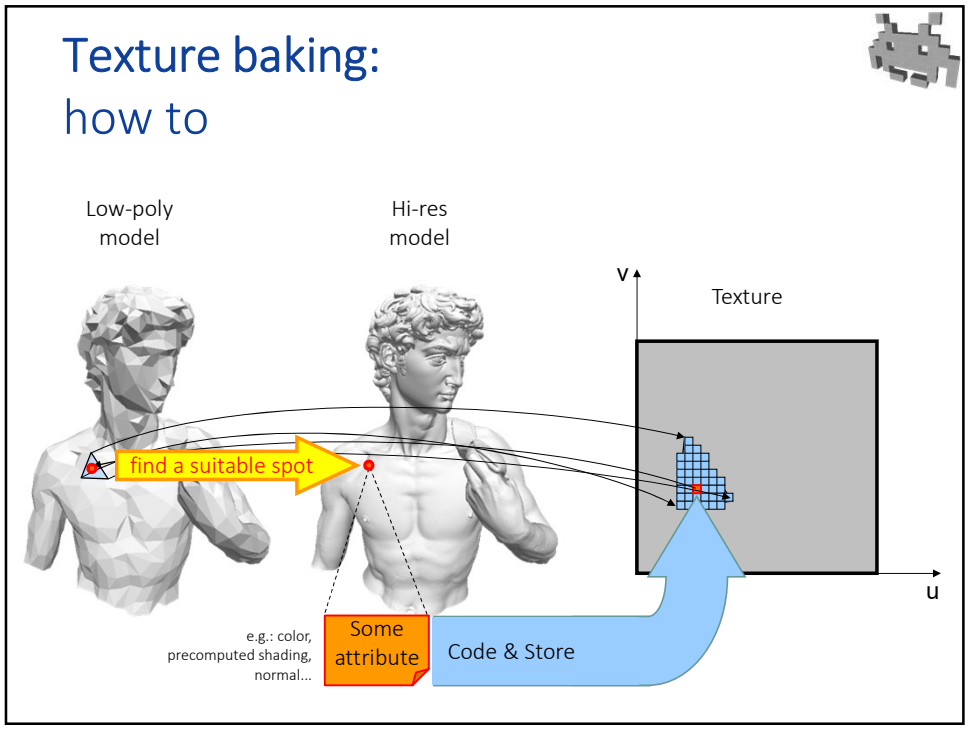
then converted to tangent space (using mesh A)

common case!

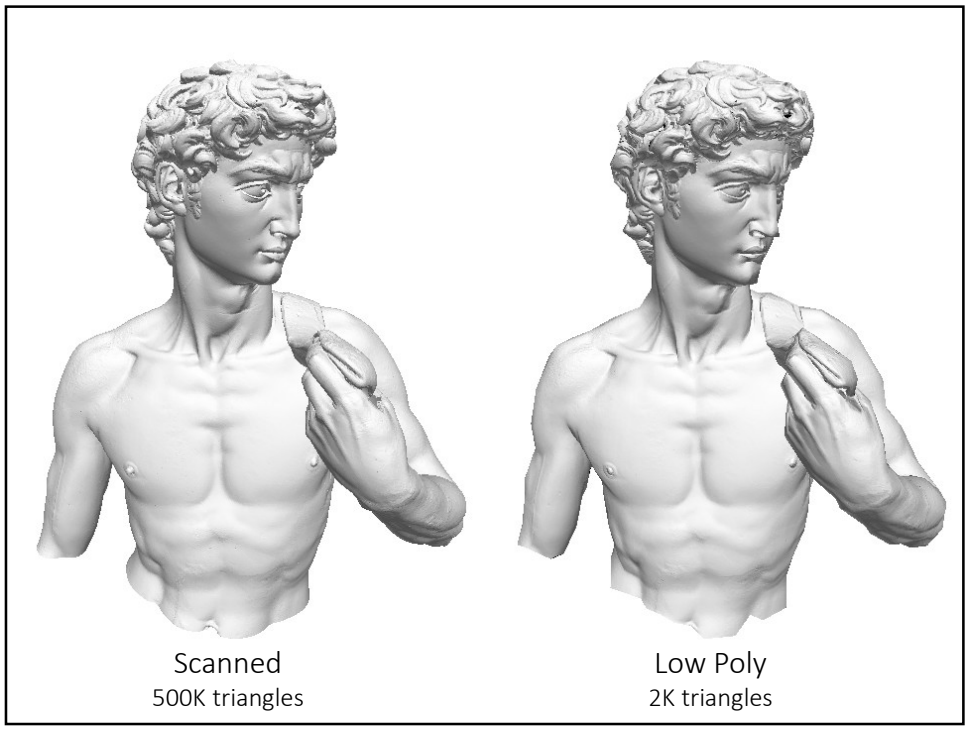
119


Hi-res
mesh
Texture
baking
TEXTURE SHEETS
Normal-map,
Color-map,
...
Low-poly,
UV-mapped
mesh
rendering
still low poly,
but now textured!

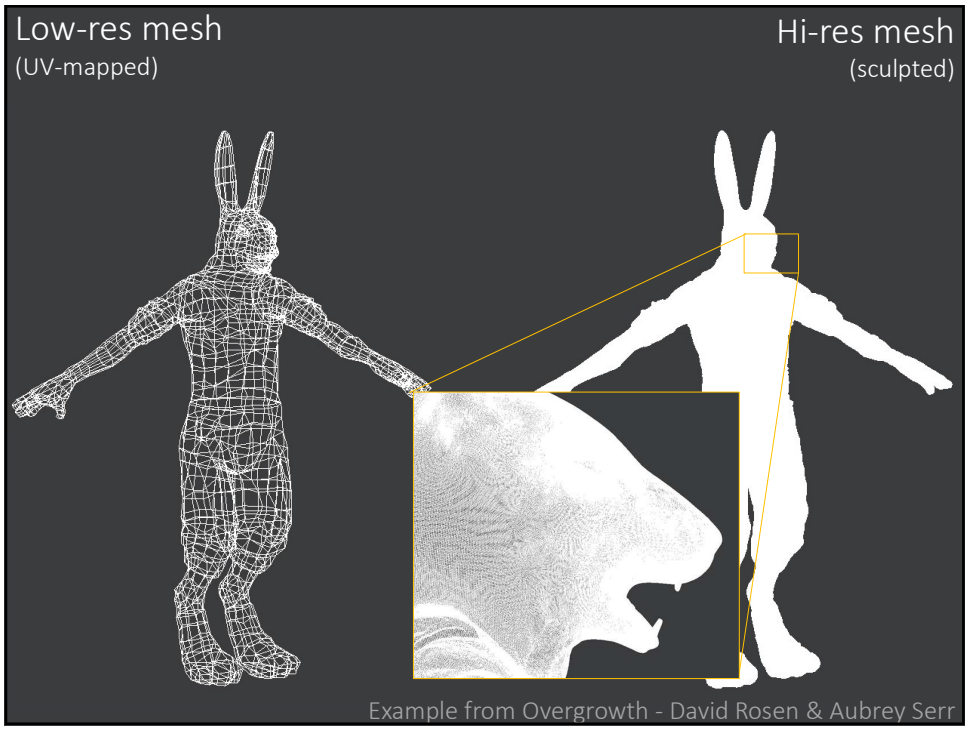
120


Texture baking:
how to
Low-poly
model
Hi-res
model
v
Texture
find a suitable spot
u
e.g.: color,
precomputed shading,
normal...
Some
attribute
Code & Store

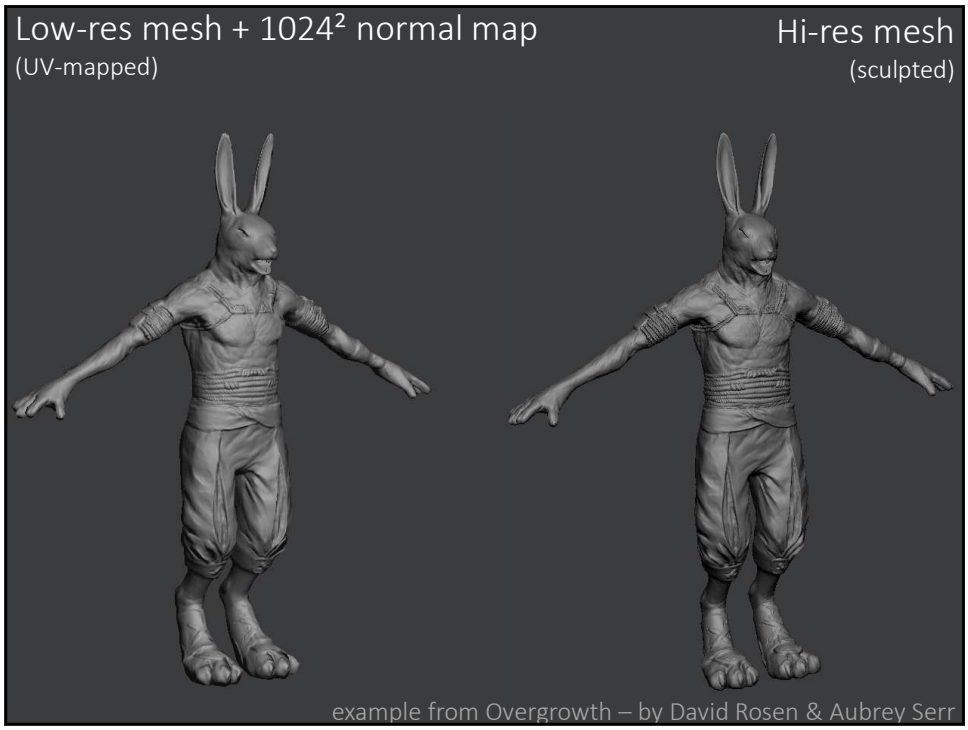
121


Scanned
500K triangles
Low Poly
2K triangles

122

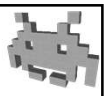

Low-res mesh
(UV-mapped)
Hi-res mesh
(sculpted)
Example from Overgrowth - David Rosen & Aubrey Serr

123



124

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



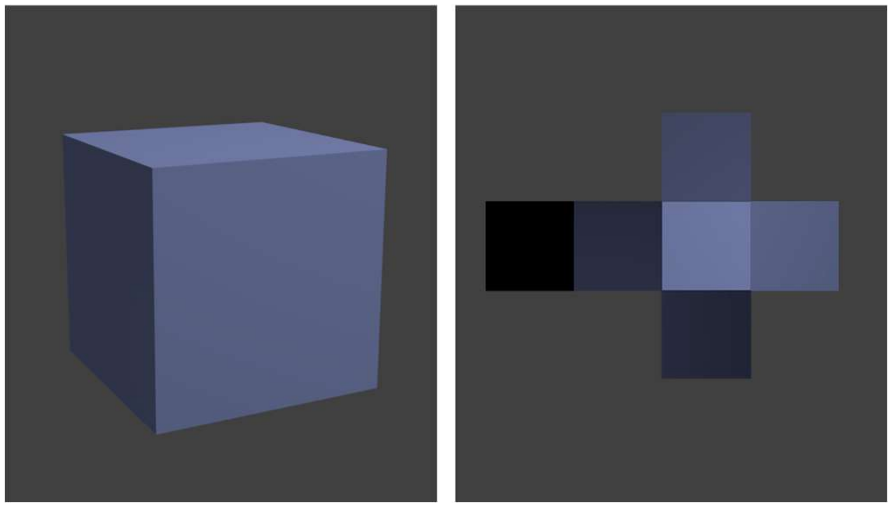
Any type of texture can be backed (see the list)
But these types can only be baked!

Baked values:

- Each texel = a *baked* lighting value
  the one computed by the lighting function
  - The texture sheet is a (baked) "light-map"
  - Instead of computing lighting on the fly during rendering, we bake in preprocessing
  - During rendering, we access only the texture instead of computing the lighting. Fast!
  - Advantage: lighting now can be arbitrarily complex. Good for global lighting! (e.g. shadows, multiple reflections, etc)
- Each texel = an *computed* ambient occlusion map
  - The texture sheet is an "AO-map" (see below)

130

Light-map (a toy example)



131

Lighting model (reprise):
Blocked lights (that is, shadows)

- In the lecture on materials, we considered "local" lighting models, where we assume that the rest of the 3D scene doesn't count
  - Only (1) the bit of surface being lit, (2) light sources, and (3) the observer, count
- In reality, the lighting equation can account for light "blockers"
  - One example of a "global" lighting effect
- The two "direct" terms (diffuse & specular) are negated (for each light source) when a "blocker" shadows the surface from that light
  - In all or in part: the two terms are multiplied by a "shadow factor" (in 0 to 1)
  - See Lecture on rendering (or courses on *Real Time Graphics Programming*) for rendering algorithms to compute this factor
- The ambient term is to be negated when the point on the surface is surrounded by many blockers
  - the term is multiplied by an "ambient occlusion" (AO) factor (in 0 to 1)
  - this term depends on the local arrangement of each point around the mesh
  - it can be backed into a texture: the AO-map!

132

Lighting model (reprise):
Ambient occlusion

CAST SHADOWS negate

AMBIENT OCCLUSION negates

direct light terms (per light) | ambient light

diffuse term | specular term | ambient term

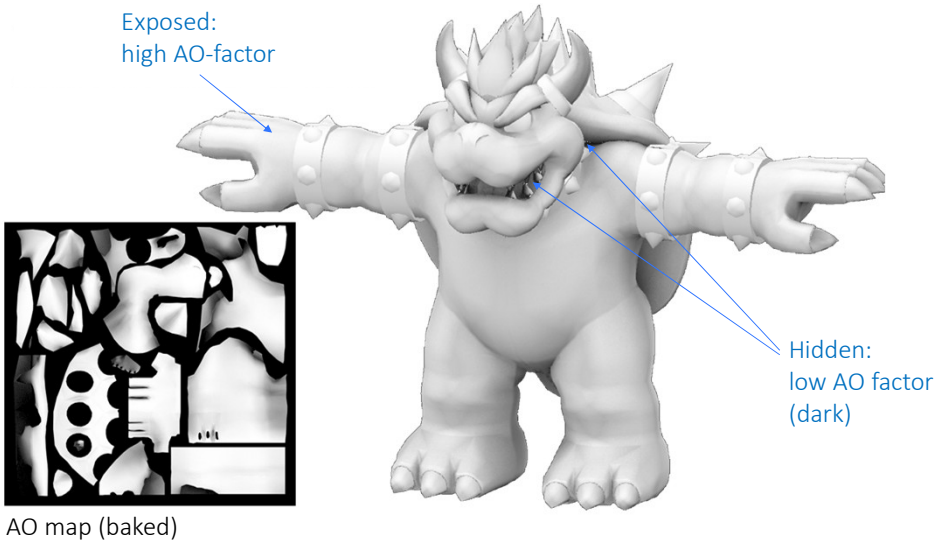
$$(\hat{n}\cdot\hat{L})\begin{pmatrix}d_R\\d_G\\d_B\end{pmatrix}\otimes\begin{pmatrix}L_R\\L_G\\L_B\end{pmatrix}+(\hat{n}\cdot\hat{H})^{E}\begin{pmatrix}s_R\\s_G\\s_B\end{pmatrix}\otimes\begin{pmatrix}L_R\\L_G\\L_B\end{pmatrix}+\begin{pmatrix}a_R\\a_G\\a_B\end{pmatrix}\otimes\begin{pmatrix}A_R\\A_G\\A_B\end{pmatrix}$$

$\hat{H}$ = nlerp($\hat{V}$, $\hat{L}$, 0.5)

material parameter
light parameter
geometry

134

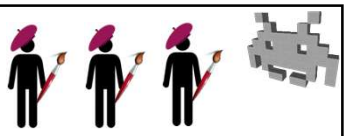
Baking AO-map for a model



AO map (baked)
NOTE: this requires UV unwrapping (injective UV-map), like any baked texture

135

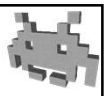
Asset production pipeline
(a general concept in game-dev)



- A sequence of stages used to produce assets. For each stage:
  - what is produced, starting from what
  - using which tool(s), by which artist(s)
  - storing which intermediate result(s), in which format, etc.
- Different pipelines for different classes of objects
  - E.g. characters ≠ sceneries (“props”) ≠ equippable armours ≠ …
  - Note: within a given game, all assets in a class are usually quite uniform (comparable resolution, same set of texture sheets, same formats, etc.)
- In the past lectures, we mentioned many possible steps
  - low poly modelling, sculpting, uv-mapping, LOD-ding…
  - texturing, geometric proxy construction, …
  - TODO: the parts about animations (skinning + rigging + animation…)
  - TODO: the parts about materials
- Identifying a good pipeline is not trivial!

136

Asset production pipeline: for example



1. Concept drawings
   - by 2D artists
2. Low-poly model A
   - by 3D modelers, using low-poly editing tools
3. UV-mapping of A
   - by UV-mappers, or by automatic tools. output: an injective UV-map of A
4. Subdivision, then digital sculpting of Hi-Res model B
   - by a 3D modeler, using digital sculpting tools
5. Painting over B
   - using a 3D painter, and producing per-vertex colors
6. Texture baking
   - Automatic construction of three Textures for A with attributes from B:
     - Normals from B, (produces a normal map)
     - Colors from B (produces a diffuse map)
     - Baked lighting from B (produces a light-map)

137