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. 8: Game 3D Animations
- lec. 9: Game 3D Audio
- lec. 10: Networking for 3D Games
- lec. 11: Artificial Intelligence for 3D Games
- lec. 12: Game 3D Rendering Techniques

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: α)
  - 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"

42

Cutout textures example
Texels = transparency level (0 or 1)

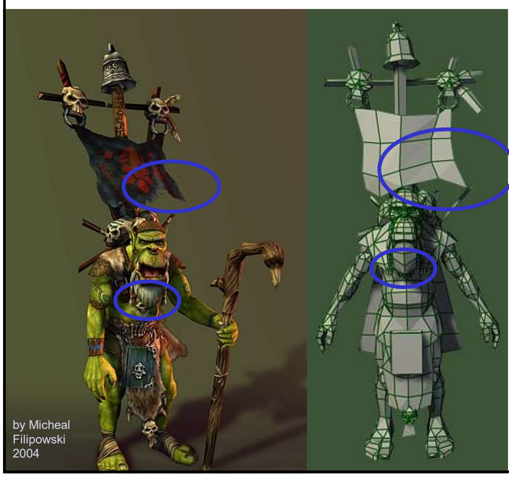
Alpha map

RGB map

43

Cutout textures
Texels = transparency level (0 or 1)

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

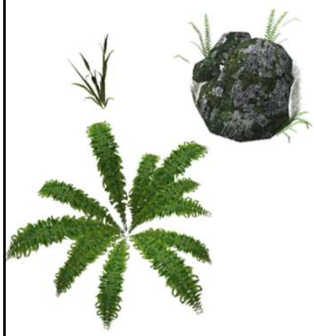


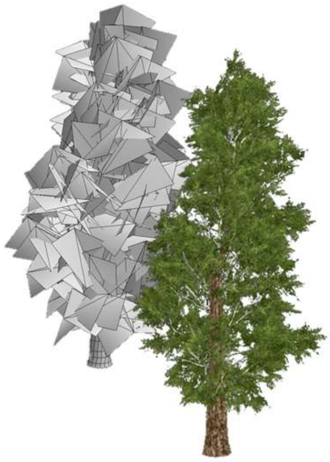
Texture
(RGBA, 4 channels)

44

Cutout textures
Texels = transparency level (0 or 1)

- e.g.: trees, foliage



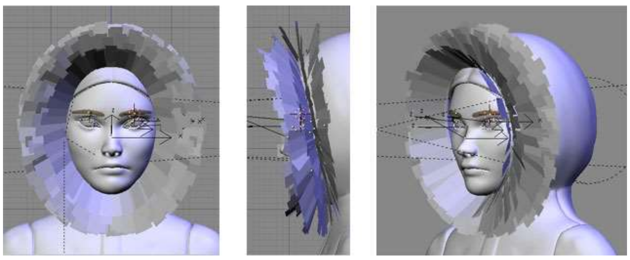


45

Texture mapping and Alpha Test

- Eg: fur, fur coats

The texture (horizontally tileable)
Pink is transparent

46

Bump-Map (*)

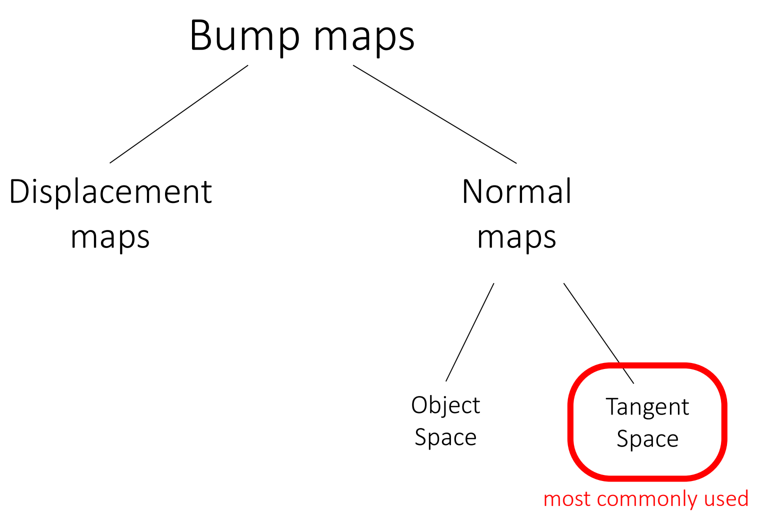
a texture modelling (or, providing an illusion of) shape details (i.e., high-frequency geometric features)

- details not modeled by the "real" geometry (the mesh)
- remember: meshes tend to be low-poly
  - not much detail in them
- approach 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»

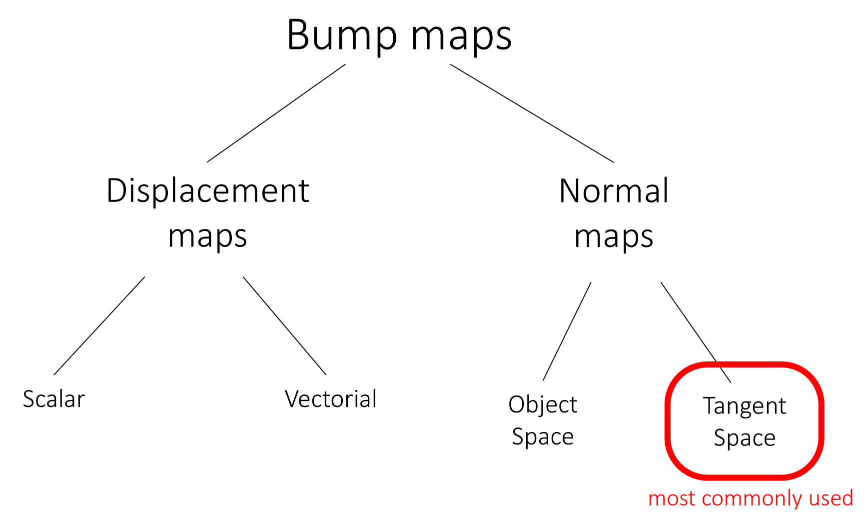
47

Types of Bump maps



48

Types of Bump maps



49

Types of Bump maps

- Bump map:
  - A texture encoding hi-frequency details
- Displacement Map:
  - 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: affecting the lighting
  - In which frame?
    - In Object Space: (requires 1:1 UV-map)
    - In Tangent Space: (TBN space)
      - Usable on more surfaces independently from the orientation
      - Requires Tangent-Bitangent direction and normals on surface

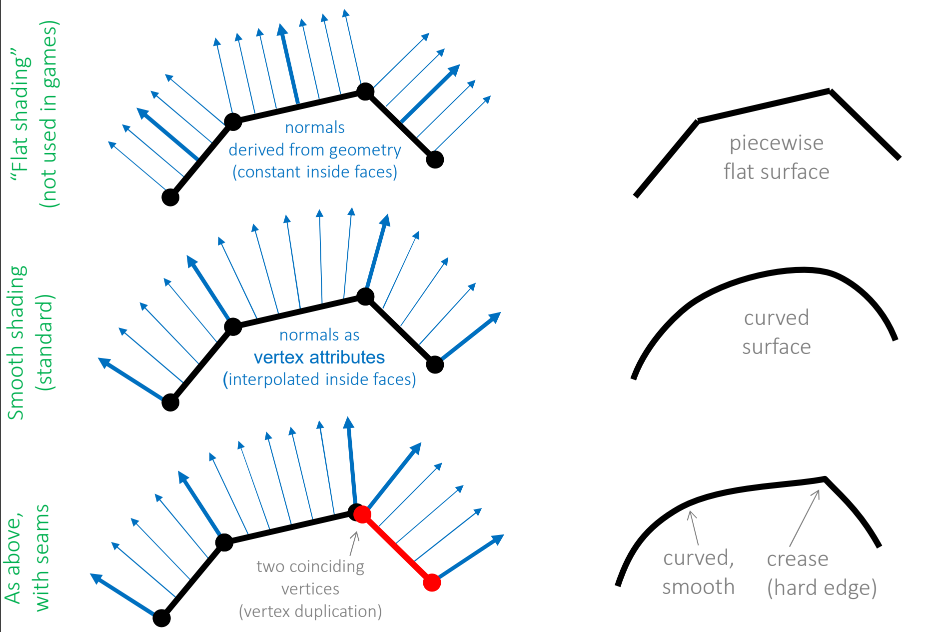
50

Bump-Map: from the modeler perspective

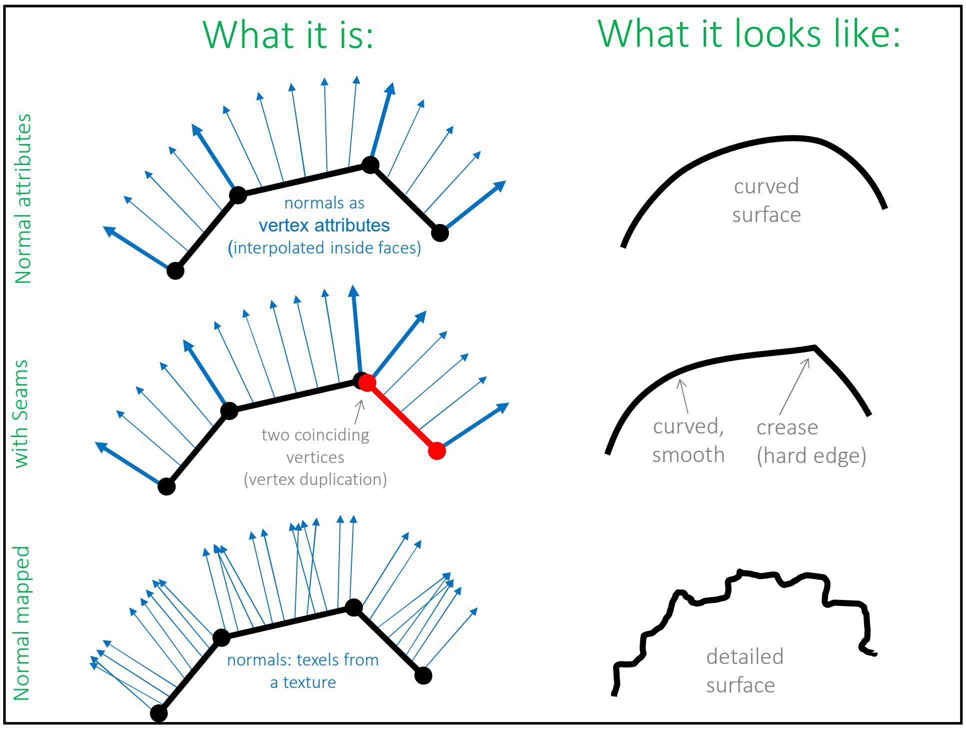
- macro-structure of the object → low-poly mesh
  - e.g.: the general shape of the horse
  - e.g.: the general shape of the face
  - e.g.: the general shape of the dragon
- meso-structure of the object → bump-map
  - e.g.: the musculature of the horse
  - e.g.: the wrinkles of the face
  - e.g.: the flakes of the dragon
- micro-structure of the object → material parameters
  - e.g.: the velvet-like fur of the horse
  - e.g.: the structure of the dermis / sebum
  - e.g.: the micro roughness / smoothnes of the flakes

51

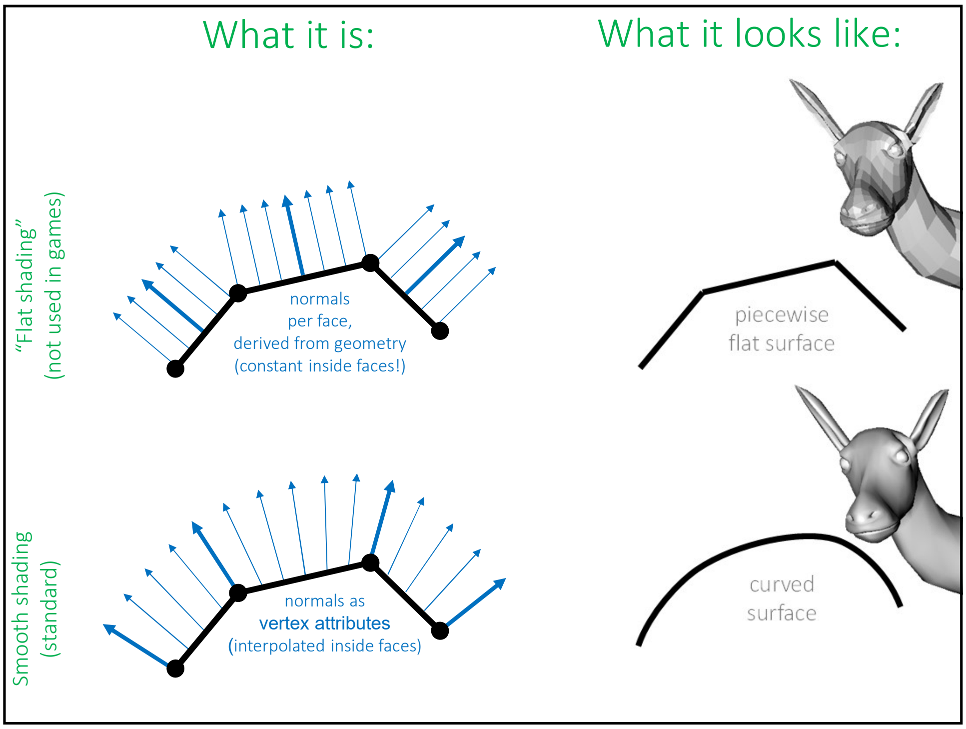
What it is: | What it looks like:



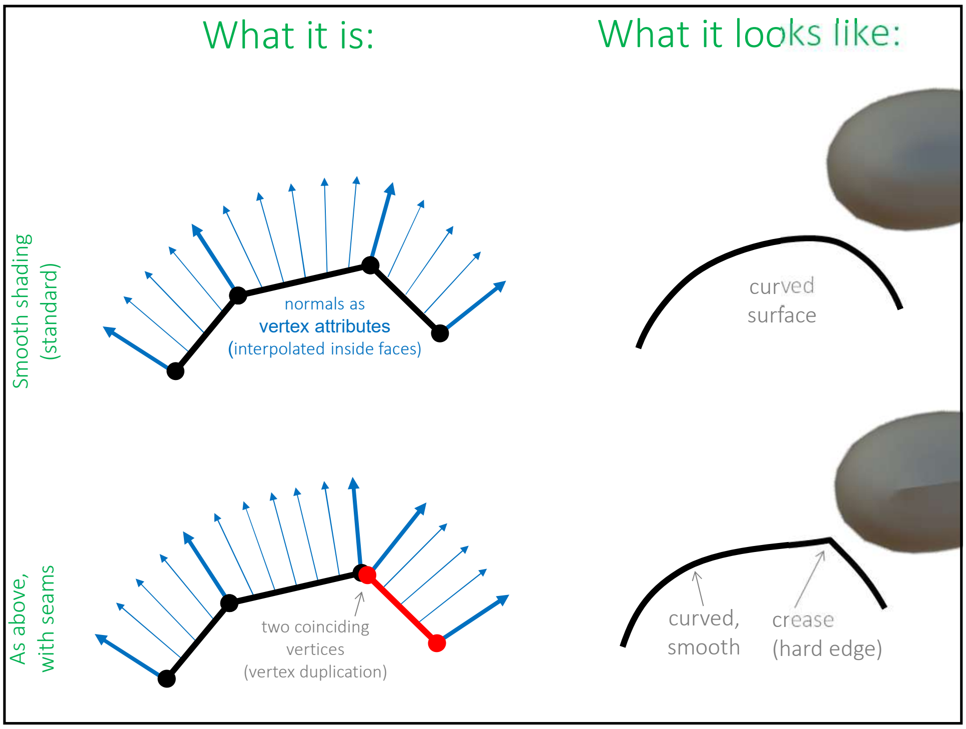
52


What it is:
What it looks like:
Normal attributes
normals as
vertex attributes
(interpolated inside faces)
curved
surface
with Seams
two coinciding
vertices
(vertex duplication)
curved,
smooth
crease
(hard edge)
Normal mapped
normals: texels from
a texture
detailed
surface

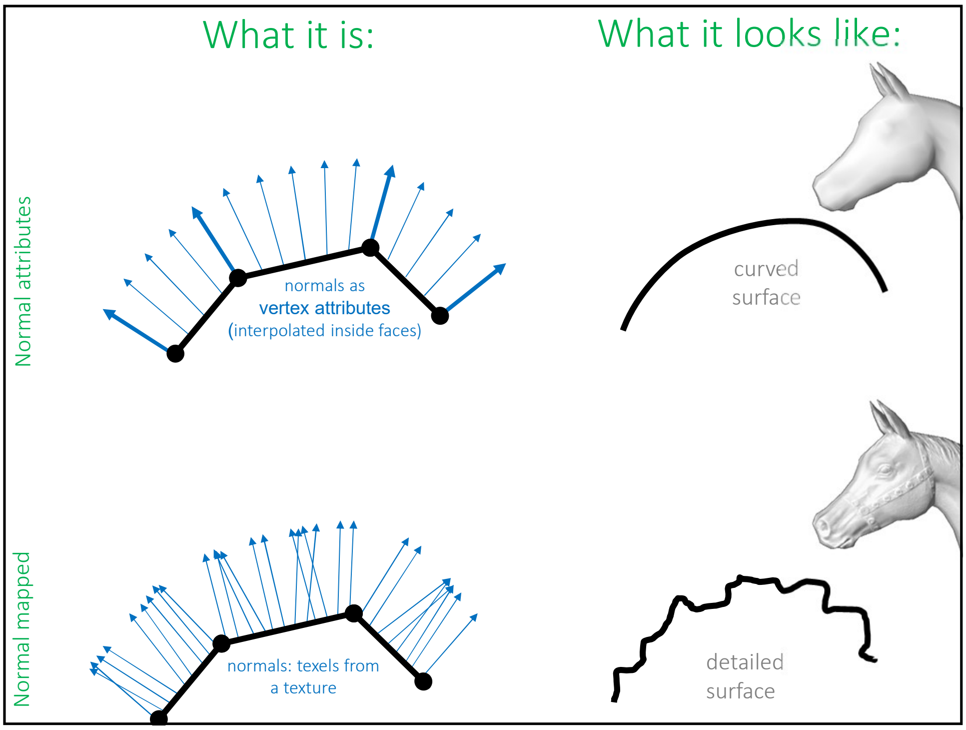
53


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

54


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

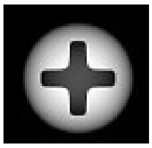
55

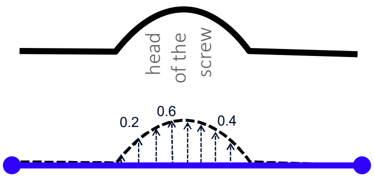

What it is:
What it looks like:
Normal attributes
normals as
vertex attributes
(interpolated inside faces)
curved
surface
Normal mapped
normals: texels from
a texture
detailed
surface

56

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 ☹)

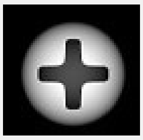
0	0	0	0	0	0	0	.1	.5	.6	.6	.7	.5	.4	.2	0	0	0	0	0	0	0

displacement map
(scalars)

57

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 = outwards
black = flat

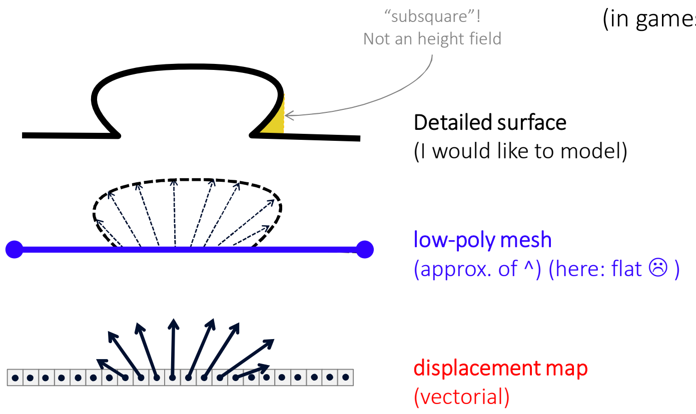
Easy to paint and manipulate!

58

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).



59

Displacement map (scalar): Rendering – embossing effect

½ · Displ.-map + ½·(1- Displ.-map) = lighting (approximated)

shifted: ↘ !

Image processing method for approximating the lighting onto a (scalar) displacement map

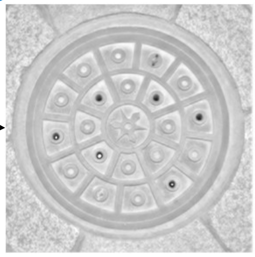
- *concept:*
  finite differences : approximate 2D gradient →
  approximate (X,Y) normal surface →
  approximate lighting

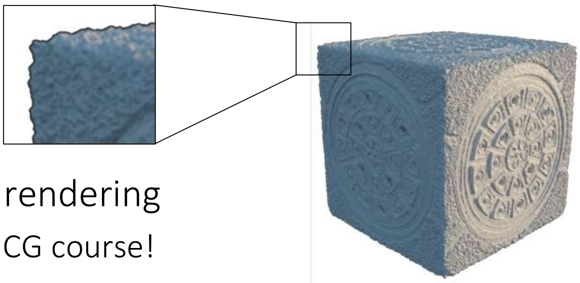
Approx. too rough: non used anymore (in games)

60

(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
  - And Real time CG course!

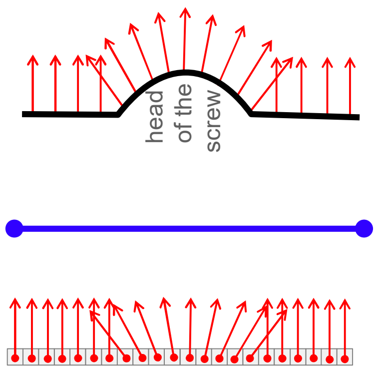
Image courtesy of https://cgcookie.com/articles/normal-vs-displacement-mapping-why-games-use-normals

61

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 ^) (here: flat ☹)

normal map
(one normal per texel)

62

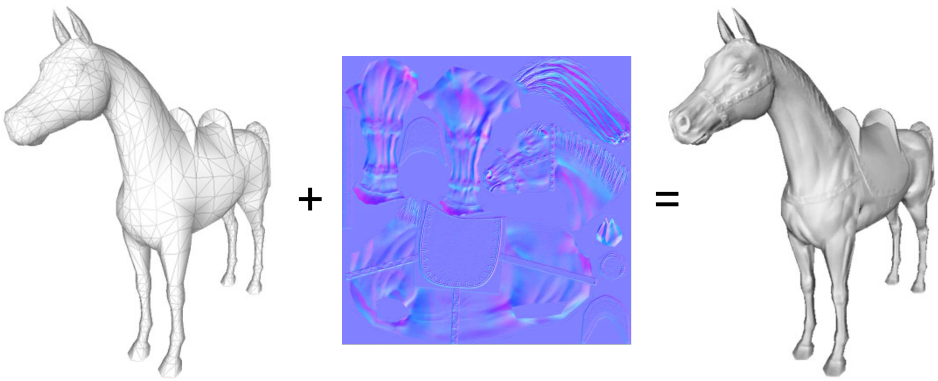
Normal Map: notes

- Affects the lighting only
  - not the parallax
  - not the shape 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 (∃ better ways to express unit vectors!)
  - Question: ok, but in which space??? *more later*

63

Normal-Mapping

see demo!



Low-poly mesh
(uv-mapped!)

Bump-map
(here: a
tangent space normal map)

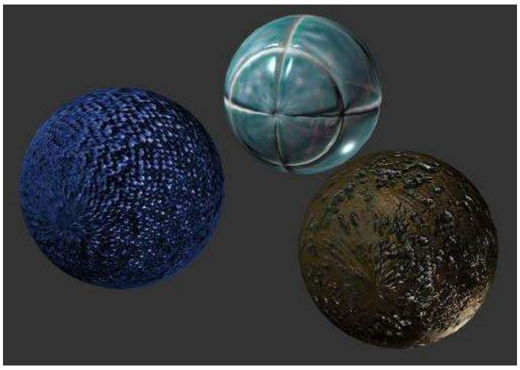
lots of *cheap*
geometric detail
(apparently)

assets courtesy of "Mount&Blade" (Talesworlds)

64

Marco Tarini
Università degli studi di Milano

Bump-Map



Same geometry (a sphere)
Different bump-maps

65

Normal Maps: in which space are the normals encoded?

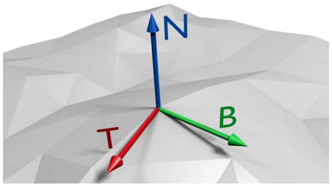
i.e, texture normals and mesh vertices are expressed in the same space

- Object space: Object-Space Normal-Maps
  - ☺ the per-vertex normal becomes unnecessary!
    - The normal from texture substitute 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 simmetries

66

Tangent space (aka TBN space)

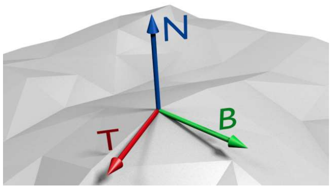
- A vector space defined ∀ point of the surface:
  - Z axis: Normal
    - orthogonal to surface
  - X and Y axis: tangent vectors
    - parallel to the surface
    - X = Tangent
    - Y = “Bi-Tangent”
      (sometimes, but inappropriately: *Bi-Normal)



67

Tangent space (aka TBN space)

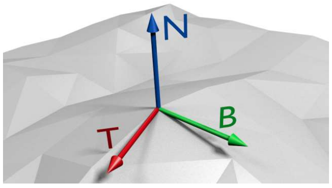
- How to store them?
  - As 3 versors stored as (per-vertex) **attributes**
    - So, they
      are interpolated inside faces
      (like any other attribute)
  - Optimizations are possible!
    - Not necessarily stored as 3 vectors (9 scalars)
    - E.g.: instead of storing B, we store N and T, then 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?)



68

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)

69

Normal Maps: in which space are the normals encoded?

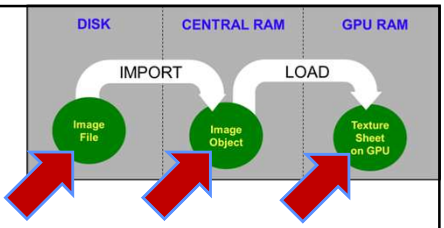
- Tangent space: Tangent Space Normal-Maps (the standard «bump-map», in games)
  - ☹ extra attributes are now needed per vertex:
    - Normal direction
    - Tangent direction
    - Bitangent direction

    (The tangent space)

    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

70

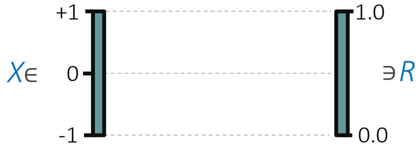
Normal-map: strorage



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

(normals are unit vectors)

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

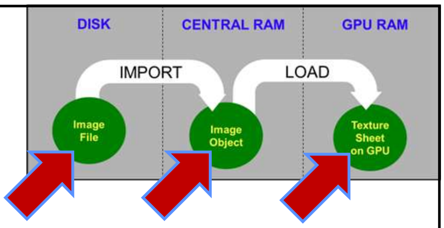


$$R = ½ (X + 1)$$
$$X = 2 R - 1$$

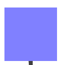
- Advantage: reuse compression of RGB textures/images
- Extra: store a (scalar) displacement map in 4th texture channel
- But, note: other, more efficient representations of versors exists

71

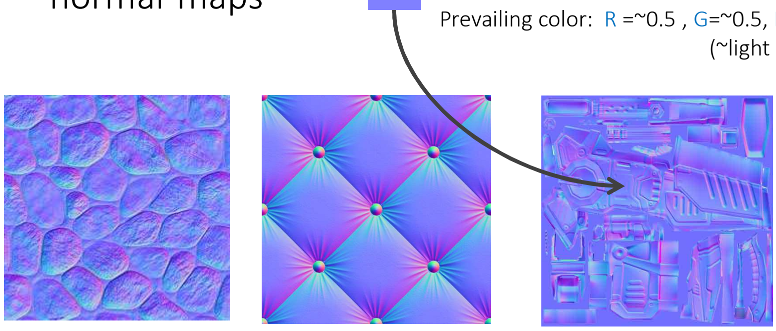
Normal-maps: Storage



- Examples of tangent space normal-maps



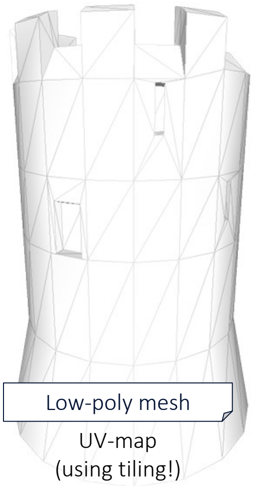
Prevailing normal : X=~0 , Y=~0 , Z=~1
⇒
Prevailing color: R =~0.5 , G=~0.5, B=~1
(~light blue)



72

Per e.g.: Tiled (tangent space) Normal Maps

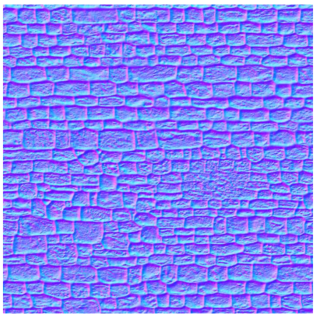
← not possible with object-space NM!



Low-poly mesh

UV-map
(using tiling!)
Tangent dirs.

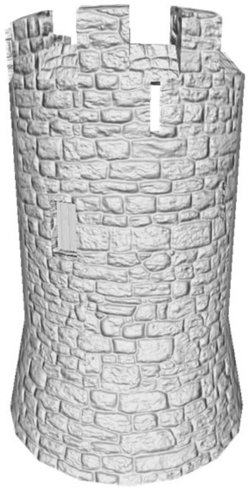
+



Normal-map

Tileable!

=



assets courtesy of "Mount&Blade" (Talesworlds)

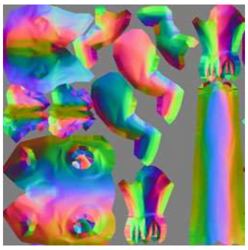
73

Bump-maps assets at a glance

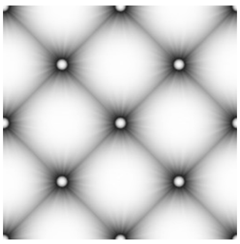
(can you tell which is which?)

Tangent Space
Normal map

the default kind



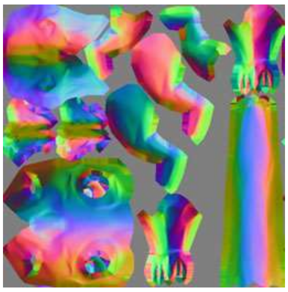
Object Space
Normal map



Displacement
Map (scalar)

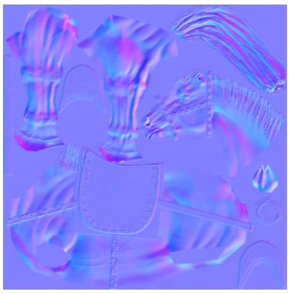
74

Observe



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

75

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 any more	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)

MUCH MORE USED IN GAMES

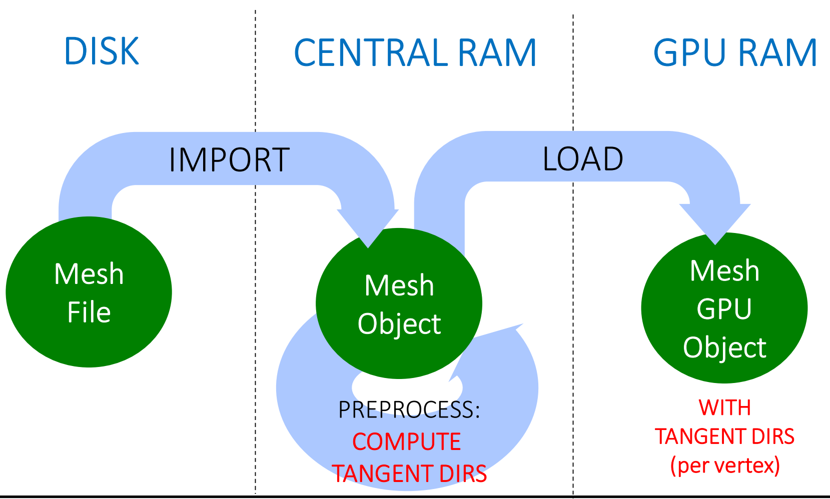
76

How to extract T and B vectors from the UV-map

- 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 trianglular face
  - then, they are averaged at vertices
  - (just like the normal directions!)

77

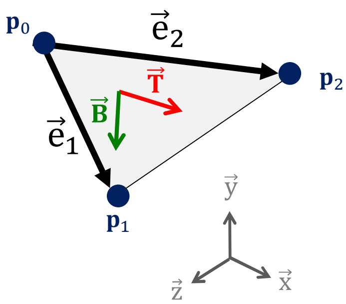
Tangent Dirs (Tangent and Bitangent) as per vertex attributes



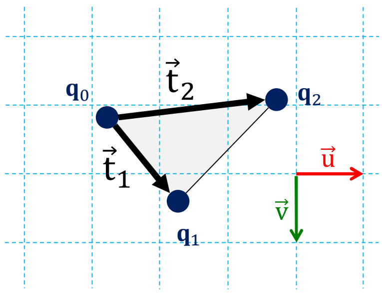
78

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

79

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

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

80

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 $\vec{e}_1 = \mathbf{p}_1 - \mathbf{p}_0$ $\quad \vec{t}_1 = \mathbf{q}_1 - \mathbf{q}_0$
  $\vec{e}_2 = \mathbf{p}_2 - \mathbf{p}_0$ $\quad \vec{t}_2 = \mathbf{q}_2 - \mathbf{q}_0$
- Find scalars a, b and c, d such that...

in matrix form: solve with a 2x2 matrix inversion

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

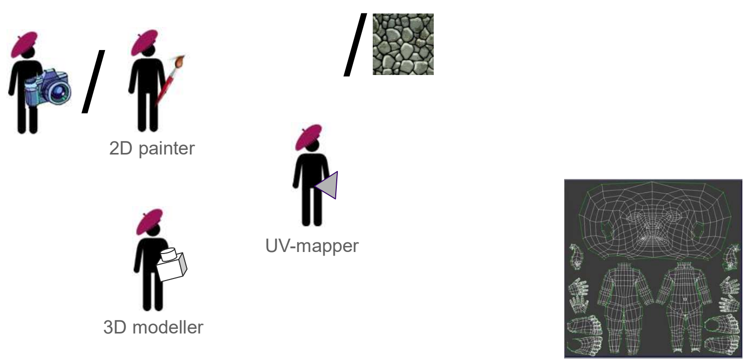
- Then

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

81

RGB maps: How are they obtained?

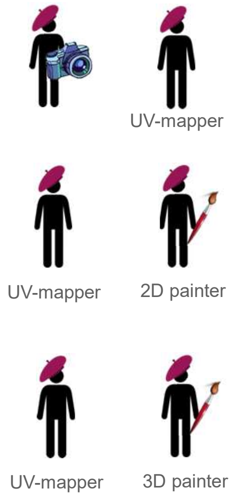
- Image first, then UV-mapping
  - e.g. Images from photos
  - e.g. tileable images



82

RGB maps:
How are they obtained?

- Image *first, then* UV-map
  - e.g., images that are photos
  - e.g., tileable images
- UV-map *first, then* paint 2D
  - paint with 2D app (e.g. photoshop)
- UV-map *first, then* paint 3D
  - paint within 3D modelling software,
  - *or:* 1. export 2D rendering,
    2. paint over with e.g. photoshop,
    3. reimport images
    4. goto 1



83

RGB maps:
How are they obtained?

...or:

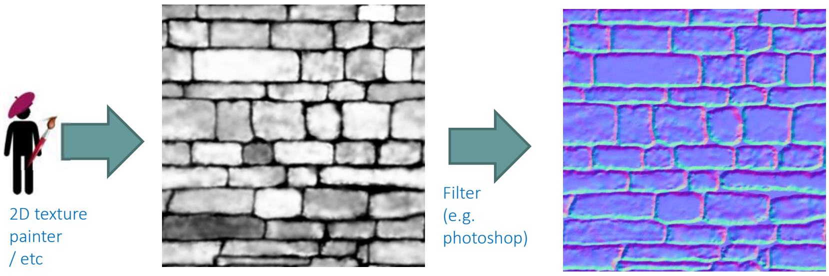
- *first* paint 3D
  - on hi-res model,
  - "paint" on vertex attributes
  - e.g. with Z brush...
- *then* coarsen
  - build / autobuild final low-poly version
- *then* UV-map
  - the low-poly model
  - must be a 1:1 UV-map!
- *then* texture backing
  - auto build texture

more about this later...

84

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

see demo!



Displacement map
come grayscale

Normal map

☐ = extruded – outwards

■ = deep – carved in

85

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

- Input: a scalar displacement map ← a texel at coords *u,v* corresponds to a 3D point (*u* , *v* , height[*u,v*])
  Output: a normal map
- 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)

86

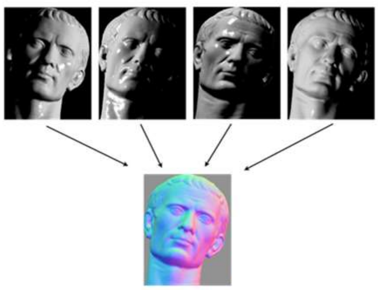
How are normal-maps obtained? (2/5) painting on 3D

- Direct painting of normal- on the model
  - (can be don, e.g., with Z-brush, Sculptris Alpha…)
  - Similar to a painting of color-maps
    - but artist paints geometric details not colors
  - Similar to mesh sculpting too
    - but, for each stroke, the system directly updates the normal on the texture-map, not the geometry on the mesh

87

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

- Captured form reality, using photos
- Example: "Photometric Stereo"
  - a form of "inverse lighting"
  - a computer vision technique
- Input: *n* real images
  - Same viewpoint
  - Different illumination
    - possibly, controlled and known
- Output: a Normal Map
  - expressed in image space
  - can be converted in object space, or in tangent space



88

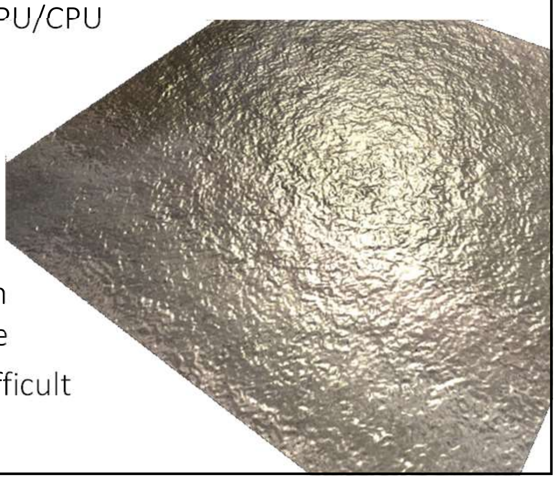
How are normal-maps obtained? (3/5) captured from reality

- 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
- Idea:
  - input: a photo of a brickwall
  - output: a diffuse map + a normal map + a specular map
- It's an active area of research!

89

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)
  - Can be build at run-time
  - Bonus: no repetition artifacts, animatable
  - Problem: control difficult



90

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

- textures baking / detail recovery / "detail texture" synthesis / texture for geometry
- input:
  - hi-res mesh A with per-vertex attributes
  - low-poly mesh B, with an injective UV-map
- output:
  - textures for B storing the attributes of A
- a fully automatic process!

91

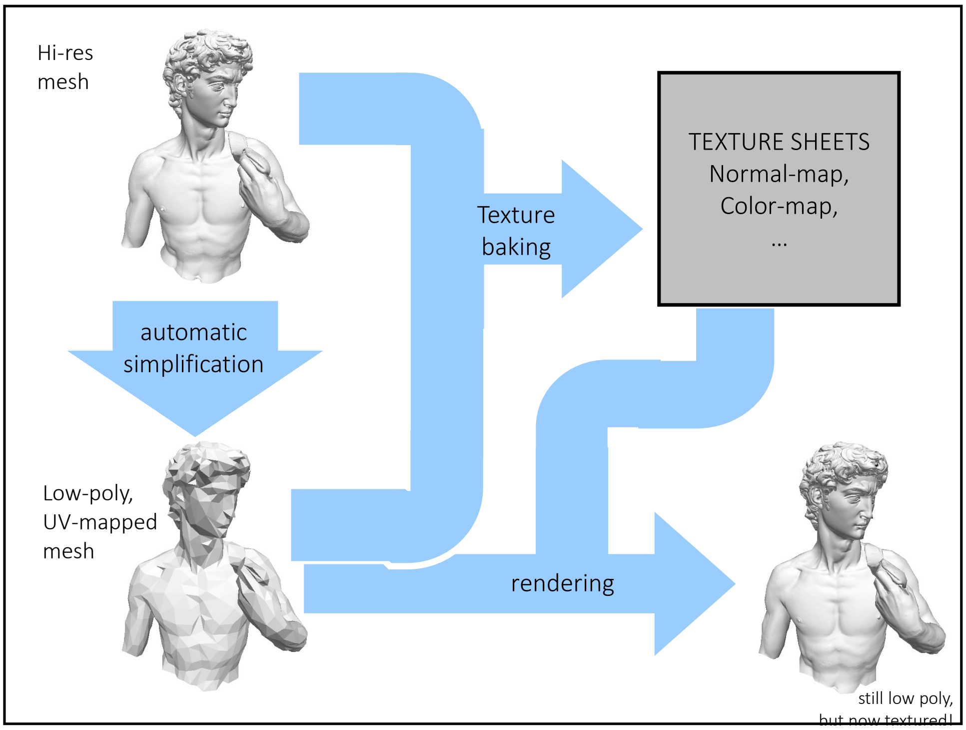
Texture baking: texture synthesis from hi-res models

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

then converted to tangent space (using mesh A)

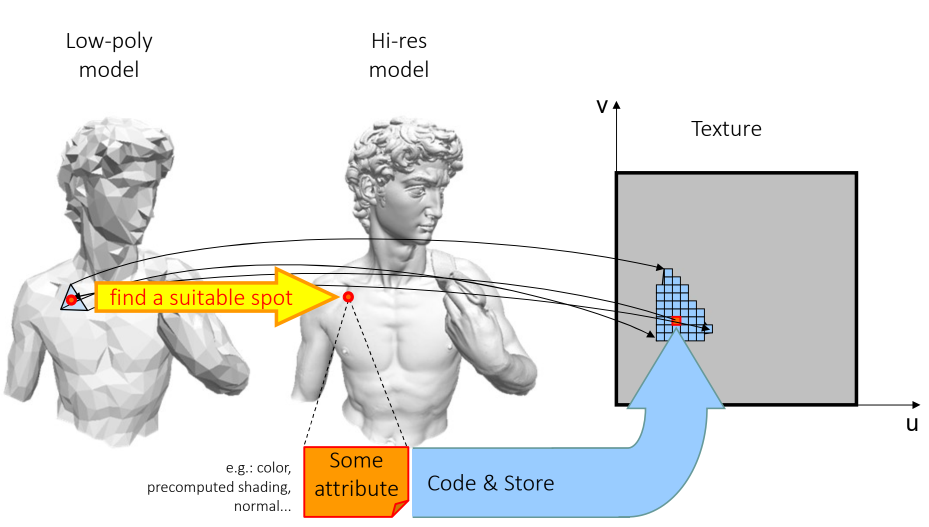
common case!

92

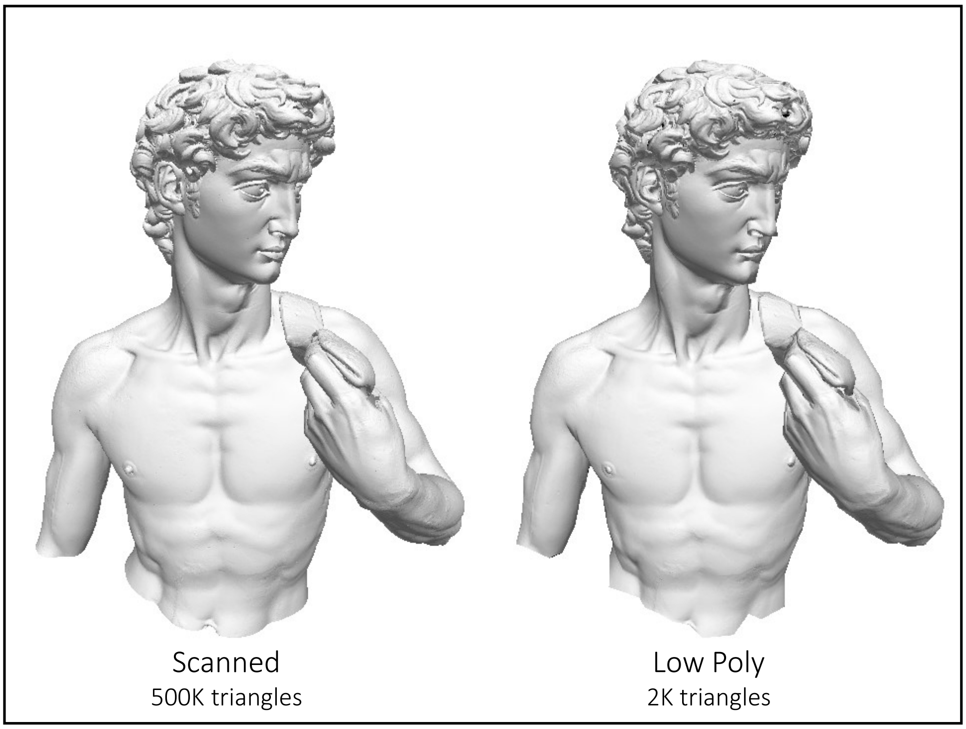


93

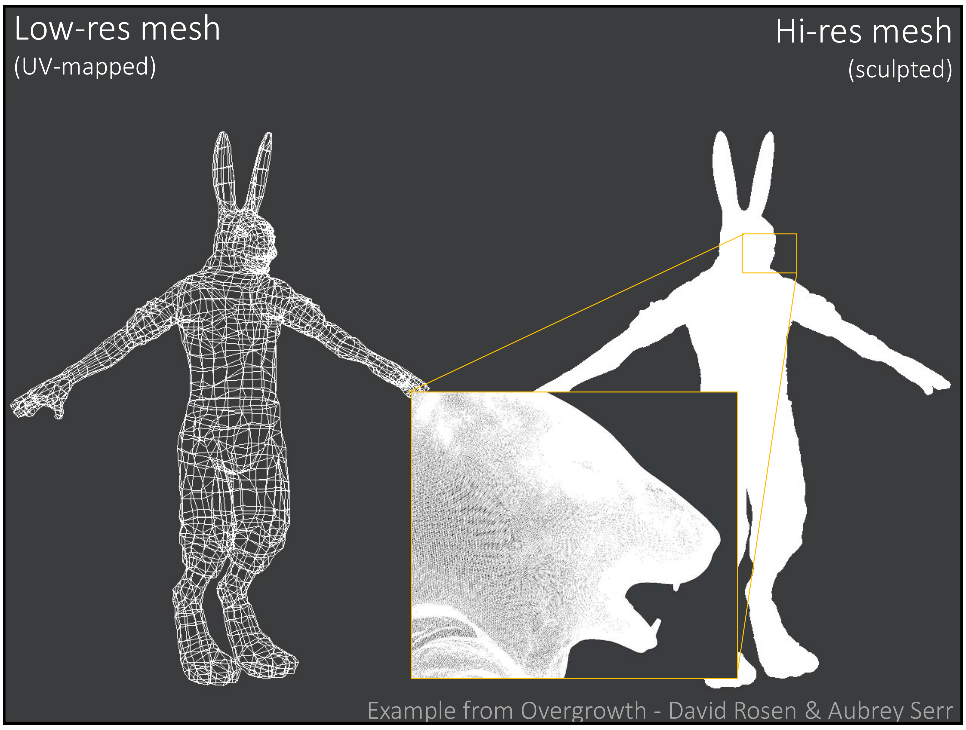
Texture baking: how to



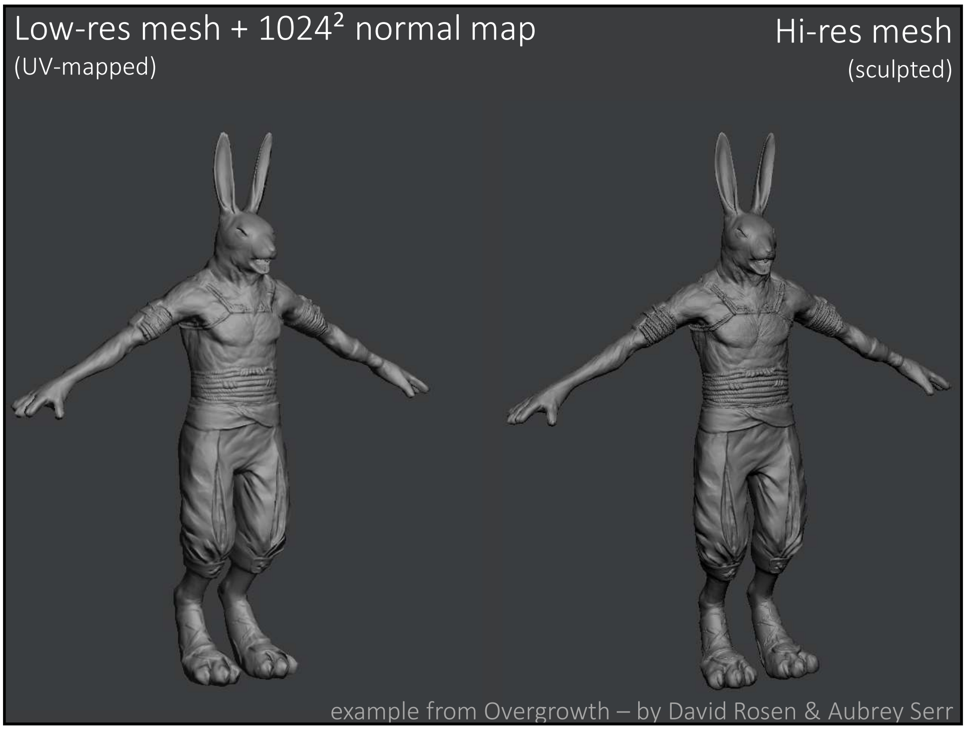
94


Scanned
500K triangles
Low Poly
2K triangles

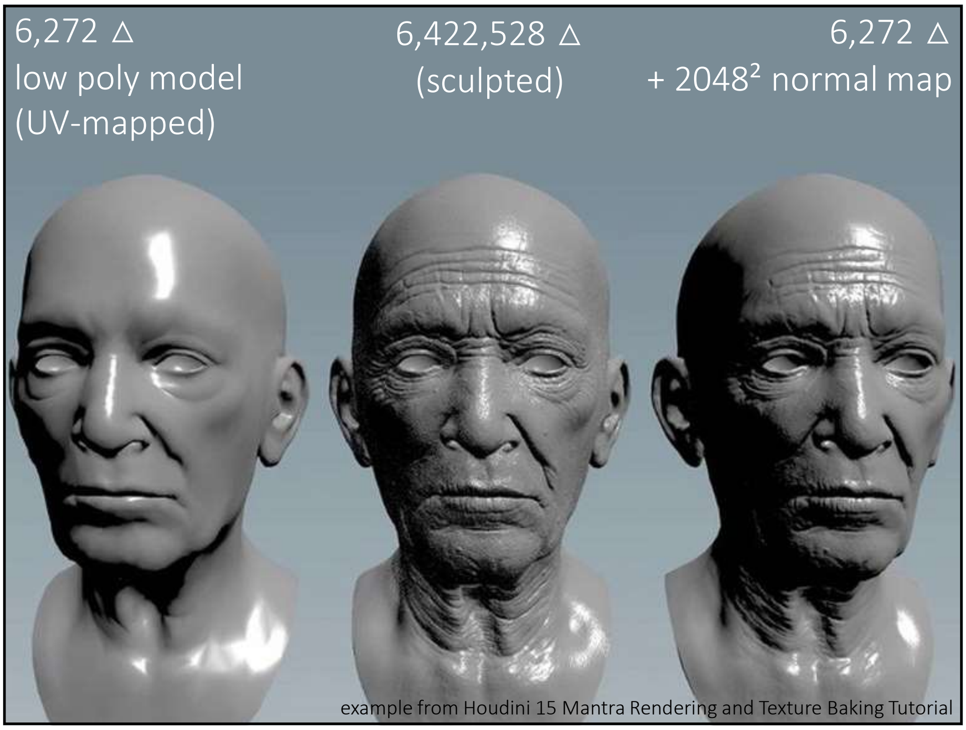
95


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

96


Low-res mesh + 1024² normal map
(UV-mapped)
Hi-res mesh
(sculpted)
example from Overgrowth – by David Rosen & Aubrey Serr

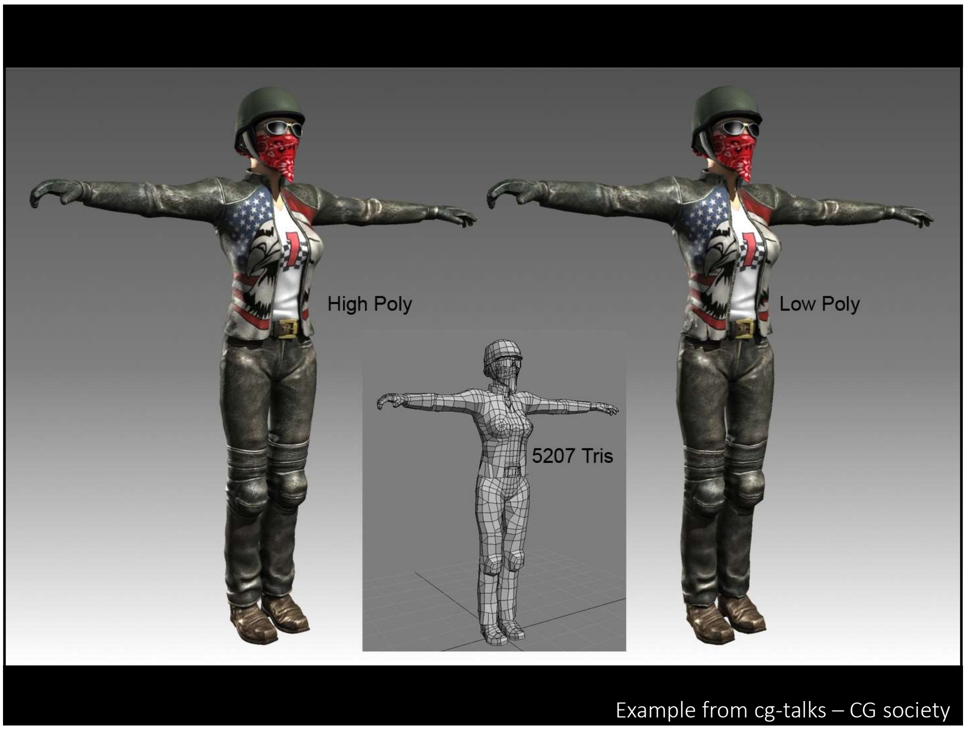
97


6,272 △
low poly model
(UV-mapped)
6,422,528 △
(sculpted)
6,272 △
+ 2048² normal map
example from Houdini 15 Mantra Rendering and Texture Baking Tutorial

98


480,000 △
(scanned)
example by
"Total Baker"
3D point software
640 △
+ 1024² diffuse map
+ 1024² normal map

99


High Poly
Low Poly
5207 Tris
Example from cg-talks – CG society

100


TEMPAR
GAME RESOLUTION
REAL TIME CGFX SHADED MODEL
2X 1024 NORM
2X 1024 SPEC
2X 1024 COL
7743 TRIS
High Res
Model
Untexured
Game Res
Normal mapped
Game Res
Final
Game Res
Example from da cg-talks – CG society

101


DAVID MACK'S
KABUKI
1X1024NORM
1X1024SPEC
1X1024COL
8000 TRIS
HIGH
GAME
FLAT
GAME
CGFX
REAL TIME CGFX SHADED MODEL
redfishimagery@yahoo.com
example from cg-talks – CG society

102

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

- A sequence of stages used to produce assets. 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
  - modelling (low poly modelling, sculpting, uv-mapping, LOD-ding…)
  - texturing, geometric proxies, …
  - TODO: the parts about animations (skinning + rigging + animation…)
  - TODO: the parts about materials
- Identifying a good pipeline is not trivial!

103

Asset production pipeline: an example

1. Concept drawings
   - by a 2D artists
2. Low-poly model A
   - by a 3D modeler, using low-poly editing tools
3. UV-mapping of A
   - by a UV-mapper, or by automatic tool. 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 3D painter, 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)

104

Procedural Textures (in general)

$$f\begin{pmatrix}u\\v\end{pmatrix} = \begin{pmatrix}r\\g\\b\end{pmatrix}$$

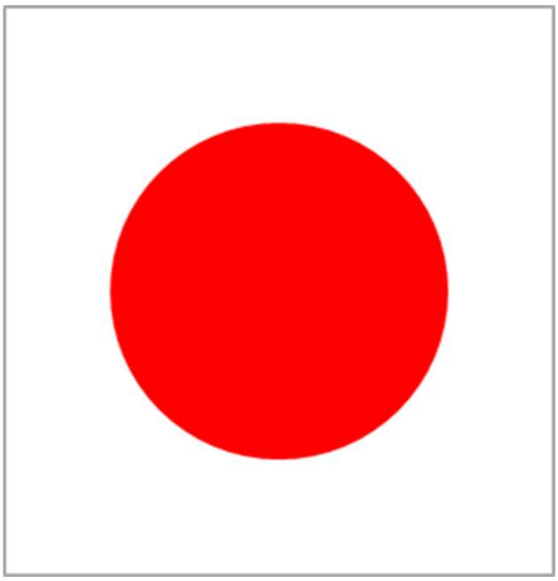
in [0..1] x [0..1]

e.g. diffuse colors, normals, transparency, etc

- A function from (u,v) to texel values
  - Plainly *replaces* a texture fetch!
  - Computed *during rendering* for each pixel (fragment shader)
  - Therefore, implemented in shader languages (e.g. GLSL, HSLS)
- Costs/benefits (the usual ones):

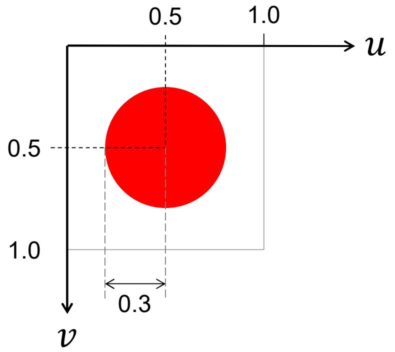
  see Lecture on Rendering and Real Time Graphics course
  - RAM / bandwidth / storage cost: reduces to almost nothing
  - GPU usage: can be substantial (it's per pixel!)
  - resolution independent (similarly to a vector image)
  - control / authoring: can be difficult to get the desired effect
- Usually limited to simple images

105



106

Example: the flag of Japan as a procedural RGB-map



$$f\begin{pmatrix}u\\v\end{pmatrix} = \begin{cases}\begin{pmatrix}1\\0\\0\end{pmatrix} & \text{if } \left\|\begin{pmatrix}u\\v\end{pmatrix} - \begin{pmatrix}0.5\\0.5\end{pmatrix}\right\|^2 < 0.3^2 \\ \begin{pmatrix}1\\1\\1\end{pmatrix} & \text{otherwise}\end{cases}$$

107

Solid Textures

108

Solid Textures

- Volumetric voxellized Texture: 3D array of texels
- 1 texel == 1 voxel
  - E.g. each voxel one color RGB → solid RGB textures
- As all the textures:
  - In video RAM
  - Fast access during rendering
  - filtering (tri-linear) in access, MIP-mapping …
- Model color onto volume
  - surface + internal
  - useful, e.g., for fractures
- Note: no need of UV-map!
  - Texture indexed by geometric mesh (rescaled)
- ⚠ Problem: ram space
  - Cubic wrt the resolution
  - Solution: procedural 3D texture?

109

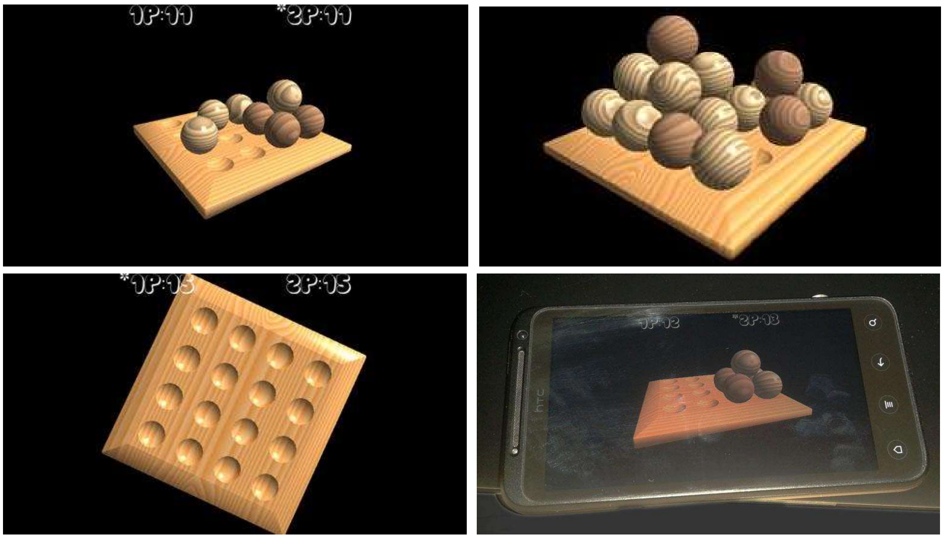
Procedural Solid Textures

$$f\begin{pmatrix}u\\v\\s\end{pmatrix}=\begin{pmatrix}r\\g\\b\end{pmatrix}$$

example by MODO

110

Procedural Solid Textures



Gyross – project by Paolo P. Slepoi

111