



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** ●● (with a red arrow pointing from the second dot to the right)
- 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 ●

68

Normal Maps: in which space are the normals encoded?



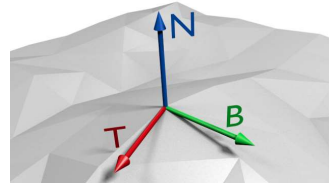
i.e., texture normals and mesh vertices are expressed in the same space

- Object space: **Object-Space Normal-Maps** (with a red arrow pointing from the text above to this bullet)
 - ☺ 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

69

Tangent space (aka TBN space)

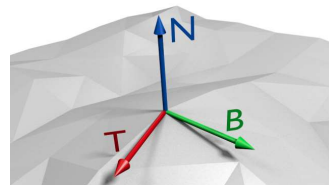
- A vector space defined \forall point of the surface:
 - Z axis: **Normal**
 - orthogonal to surface
 - X and Y axis: tangent vectors
 - parallel to the surface
 - X = **Tangent**
 - Y = "**Bi-Tangent**"
(sometimes, but inappropriately: *Bi-Normal)



70

Tangent space (aka TBN space)

- How to store them?
 - As 3 vectors 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 $B = N \times T$
 - Note: they have discontinuities
 - seams (vertex duplications) are necessary
 - In first approximation, the same ones required by the UV-map (but non only! why?)

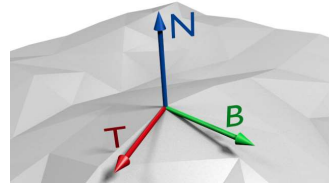


71

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 approximately* 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)

72

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:

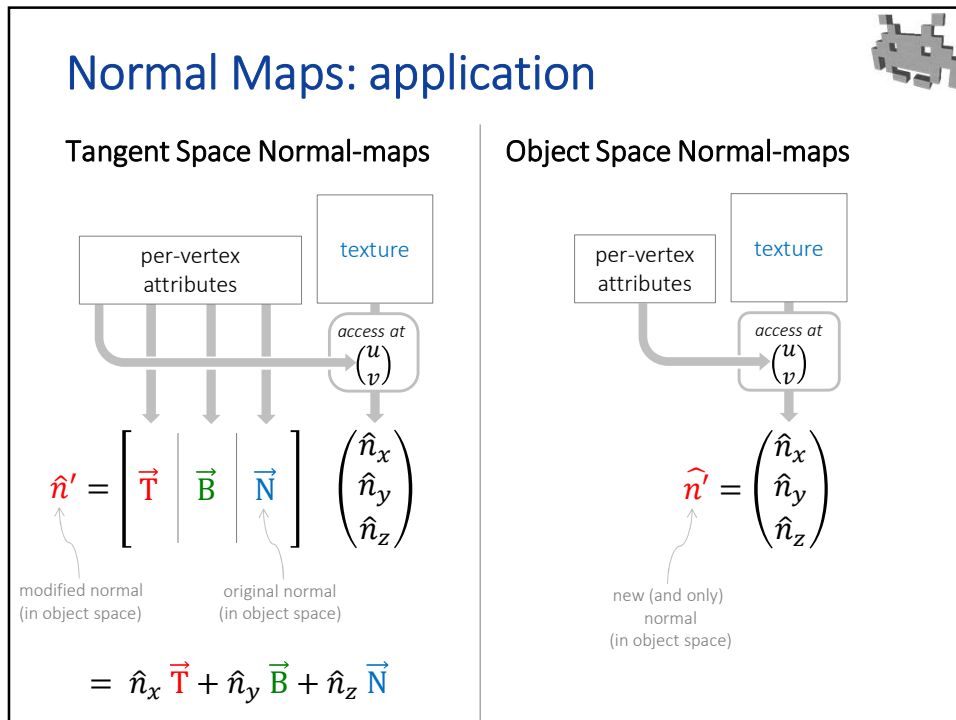
The tangent space

- 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

73



74

Normal-map: storage (as a texture)

- 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:

$X \in \begin{matrix} +1 \\ 0 \\ -1 \end{matrix}$

$\ni R$

$R \in \begin{matrix} 1.0 \\ 0.0 \end{matrix}$

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

$$X = 2R - 1$$
- Advantage: reuse compression of RGB textures/images
- Extra: store a (scalar) displacement map as 4th texture channel?
- Note: other, more efficient representations of versors exists

75

Normal-maps: storage (as a texture)

DISK

CENTRAL RAM

GPU RAM

IMPORT LOAD

Image File

Image Object

Texture Sheet on GPU

➔ ➔ ➔

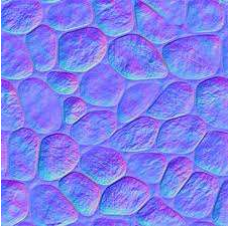
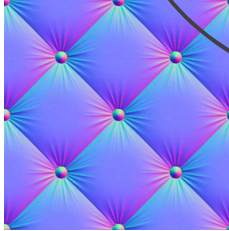
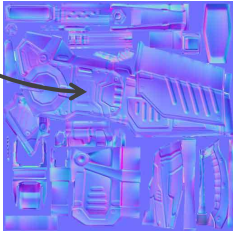
- Examples of tangent space normal-maps

Prevailing normal : $X \approx 0$, $Y \approx 0$, $Z \approx 1$

⇒

Prevailing color: $R \approx 0.5$, $G \approx 0.5$, $B \approx 1$

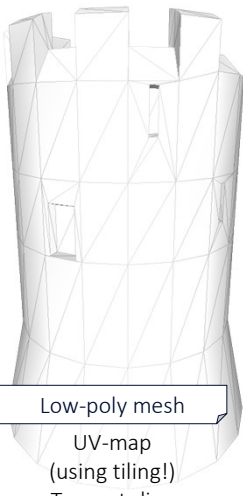
(~light blue)

76

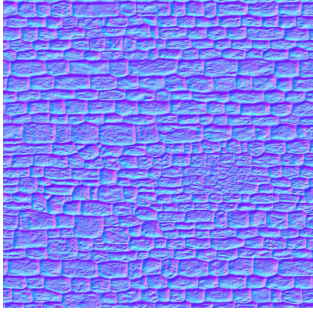
E.g.: Tiled (tangent space) Normal Maps

← not possible with object-space NM!



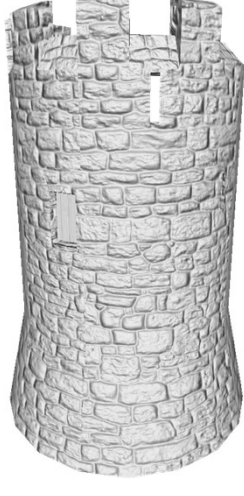
Low-poly mesh
UV-map
(using tiling!)
Tangent dirs.

+



Normal-map
Tileable!

=

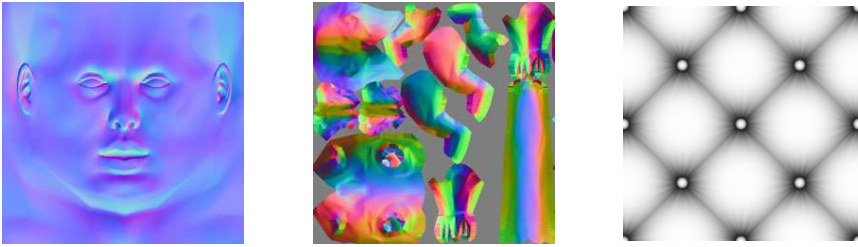


assets courtesy of "Mount&Blade" (Talesworlds)

77

Bump-maps assets at a glance

(can you tell which is which?)



Tangent Space Normal map

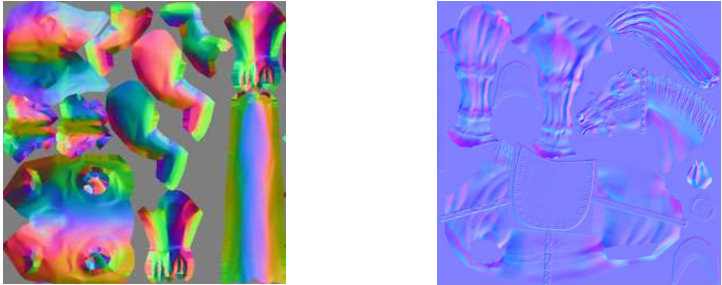
Object Space Normal map

Displacement Map (scalar)

the default kind

78

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

79

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 and B vectors (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)

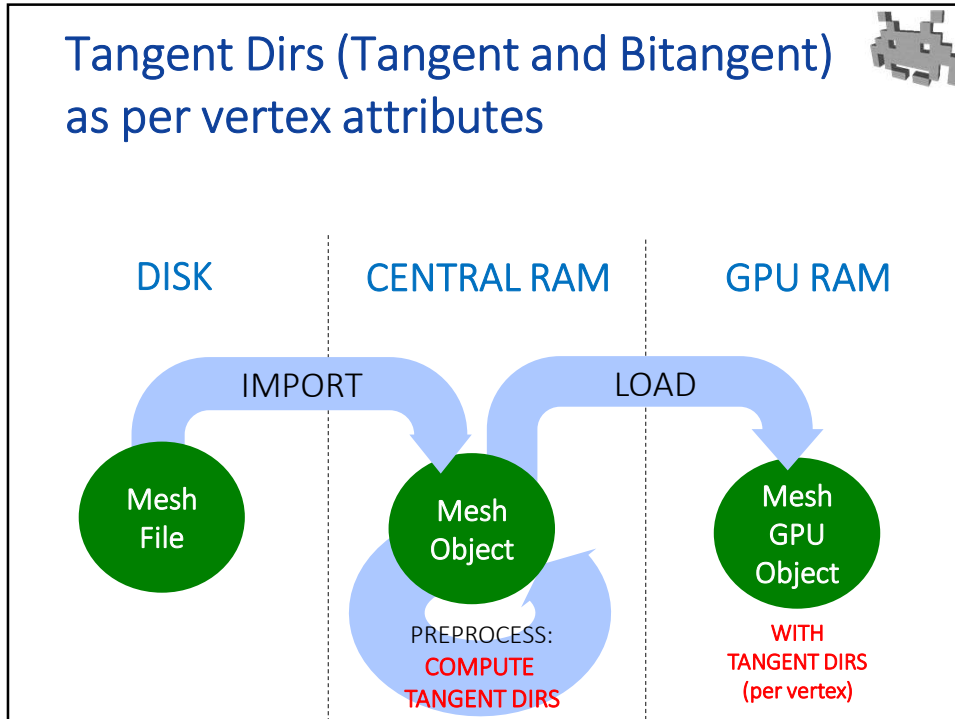
MUCH MORE USED IN GAMES

80

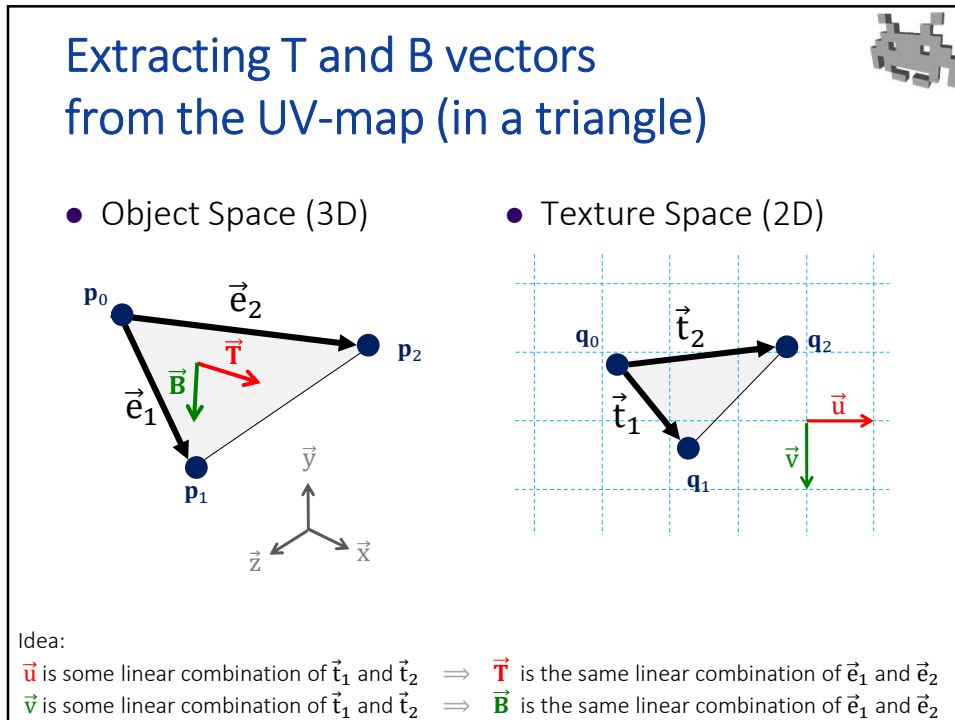
How to extract T and B vectors from the UV-map

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

81



82



83

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...

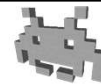
$$a \vec{t}_1 + b \vec{t}_2 = \vec{u} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad c \vec{t}_1 + d \vec{t}_2 = \vec{v} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Then

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

84

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

in matrix form:

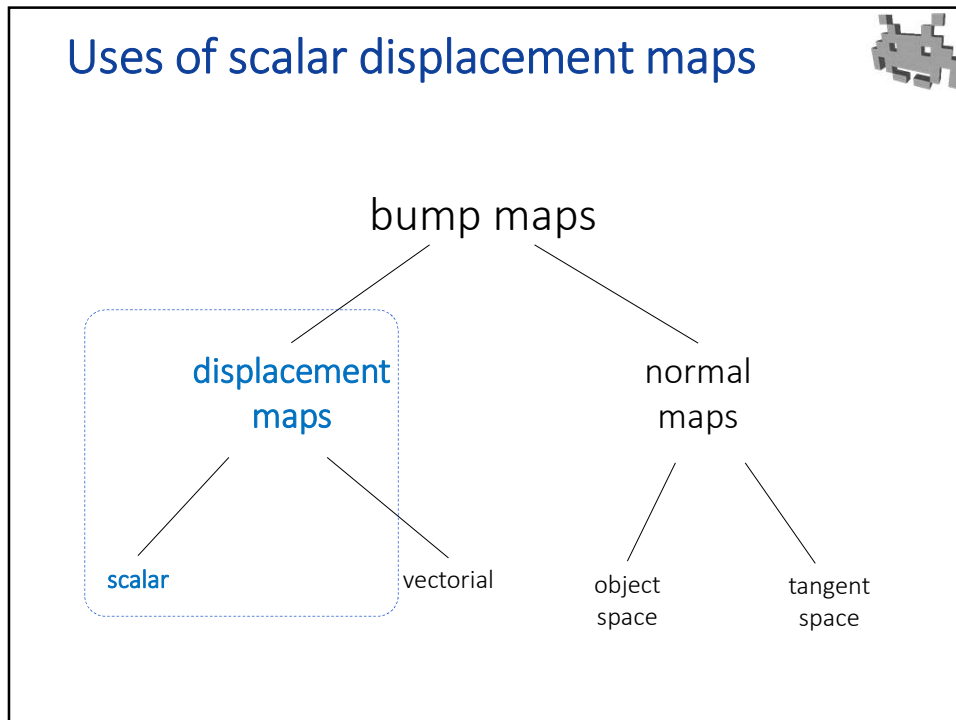
solve with a 2x2 matrix inversion

$$\begin{bmatrix} \vec{t}_1 & \vec{t}_2 \end{bmatrix} \begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} \vec{t}_1 & \vec{t}_2 \end{bmatrix}^{-1}$$

- Then

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


85



86

Scalar displacement map: notes

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



white = outwards
black = flat

Easy to paint and manipulate!

87

(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!

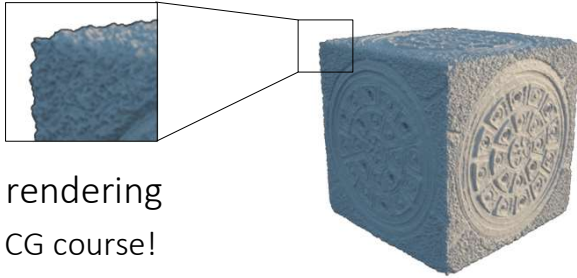
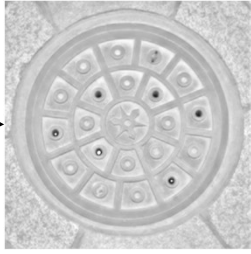
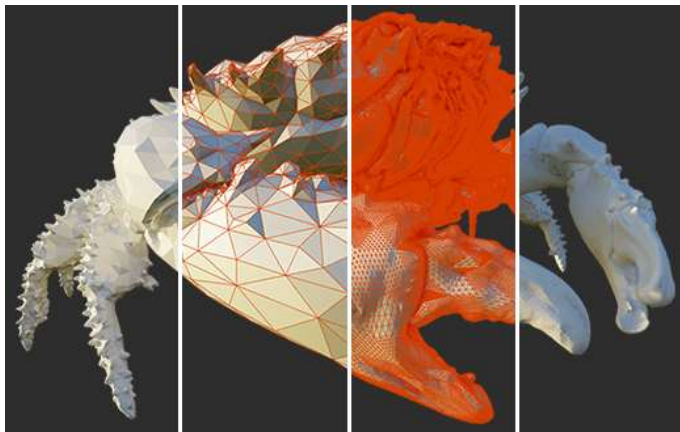


Image courtesy of <https://cgcookie.com/articles/normal-vs-displacement-mapping-why-games-use-normals>

89

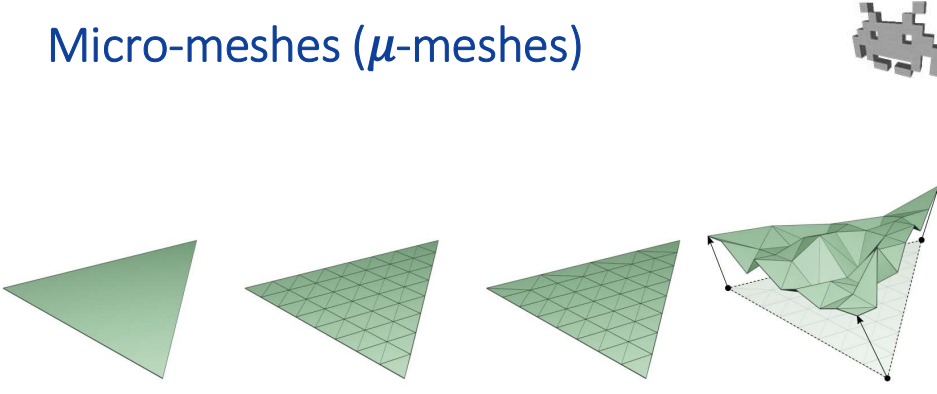
μ -meshes (micro-meshes)



base-mesh micro-triangles

90

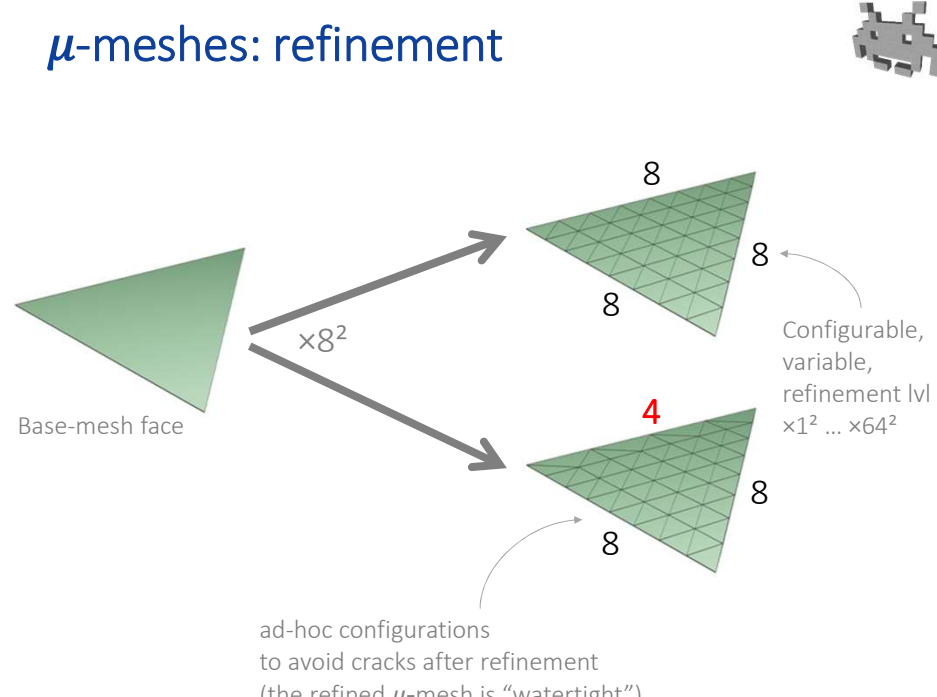
Micro-meshes (μ -meshes)



- A coarse mesh with a specially formatted scalar displacement map
- During rendering: refine + displace

91

μ -meshes: refinement



Base-mesh face

$\times 8^2$

8

8

8

8

8

8

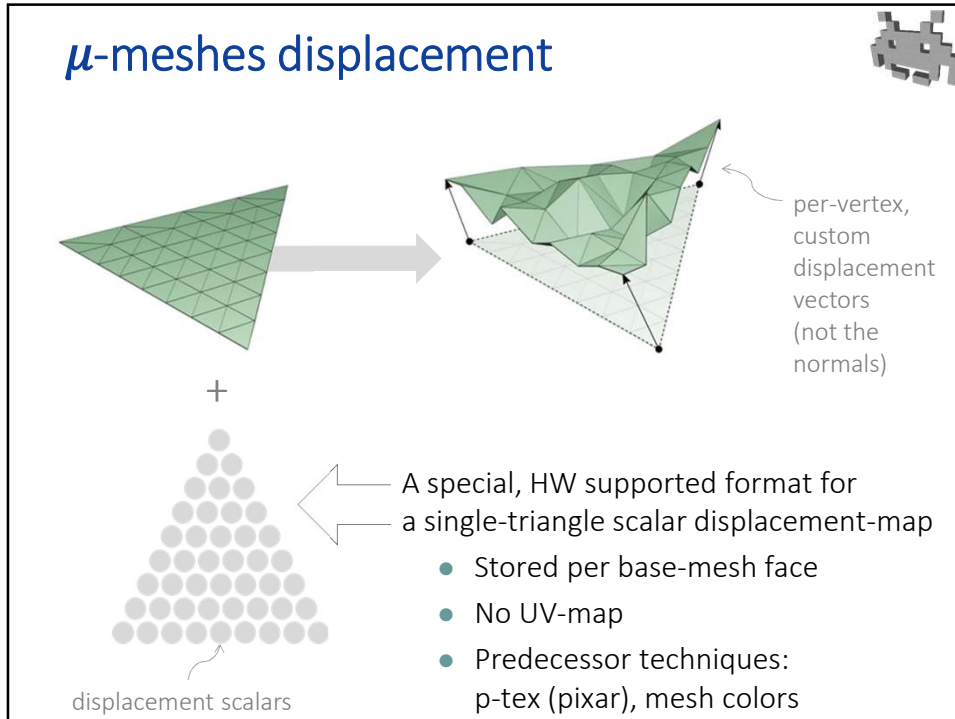
8

8

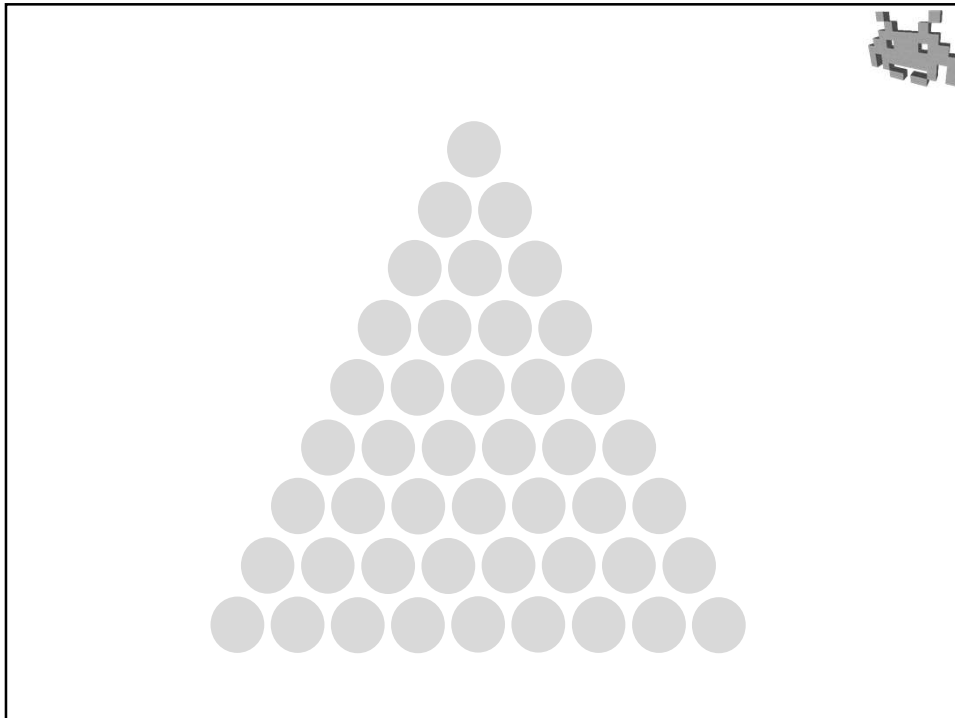
Configurable, variable, refinement lvl $\times 1^2 \dots \times 64^2$

ad-hoc configurations to avoid cracks after refinement (the refined μ -mesh is "watertight")

92

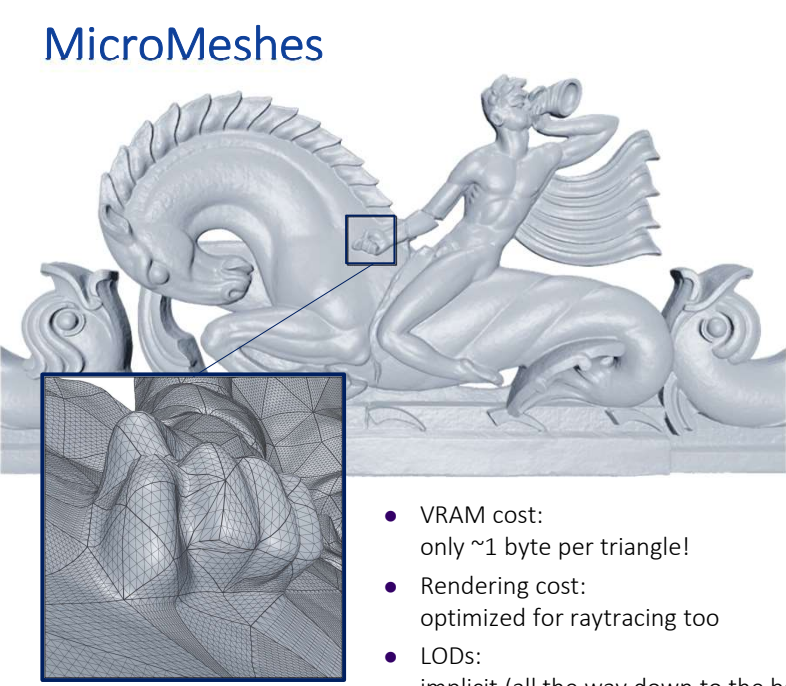


93



94


MicroMeshes



- VRAM cost:
only ~1 byte per triangle!
- Rendering cost:
optimized for raytracing too
- LODs:
implicit (all the way down to the base mesh)

95


Notes on authoring textures (an index of topics ahead)





- How are textures authored?
 - As a part of asset production pipelines?
 - Notes on authoring of color-maps, normal maps, and so on

97



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,
 - *or:* 1. export 2D rendering,
2. paint over with e.g. photoshop,
3. reimport images
4. goto 1



UV-mapper




UV-mapper 2D painter



UV-mapper 3D painter

98

RGB maps authoring: when (and how)?



...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...*

99

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

see demo!

2D texture painter / etc

Displacement map come grayscale

Filter (e.g. photoshop)

Normal map

= extruded – outwards
 = deep – carved in

100

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

- Input: a scalar displacement map
 Output: a normal map
- Algorithm (2D image processing):
 - \forall texel \mathbf{t} of displacement map, compute **best fitting plane** around \mathbf{t}
 - Consider all 3D points in a 3×3 patch surrounding \mathbf{t} (or 5×5 , or $7 \times 7 \dots$)
 - 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 \mathbf{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)

a texel at coords u, v corresponds to a 3D point $(u, v, \text{height}[u, v])$

101

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



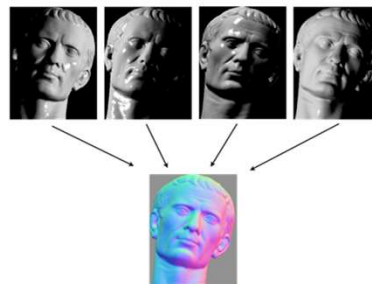
- Direct painting of normal- on the model
 - (can be don, e.g., with Z-brush, Sculptris Alpha...)
- Similar to a painting of color-maps
 - but artist paints geometric details not colors
- Similar to mesh sculpting too
 - but, for each stroke, the system directly updates the normal on the texture-map,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

102

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



103

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



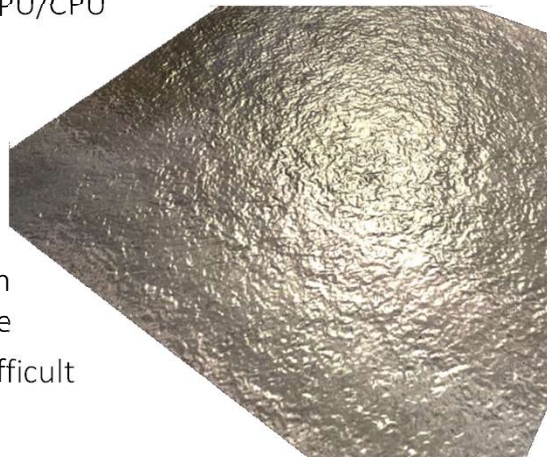
- Normal map estimation from images
 - Traditionally, many pictures are required in input
 - Traditionally, controlled illumination is required (I must place lights in known position)
 - With Machine Learning, it's becoming possible to use a single image with natural illumination
- Idea:
 - input: a photo of a brick wall
 - output: a diffuse map + a normal map + a specular map
- It's an active area of research!

104

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



105

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!

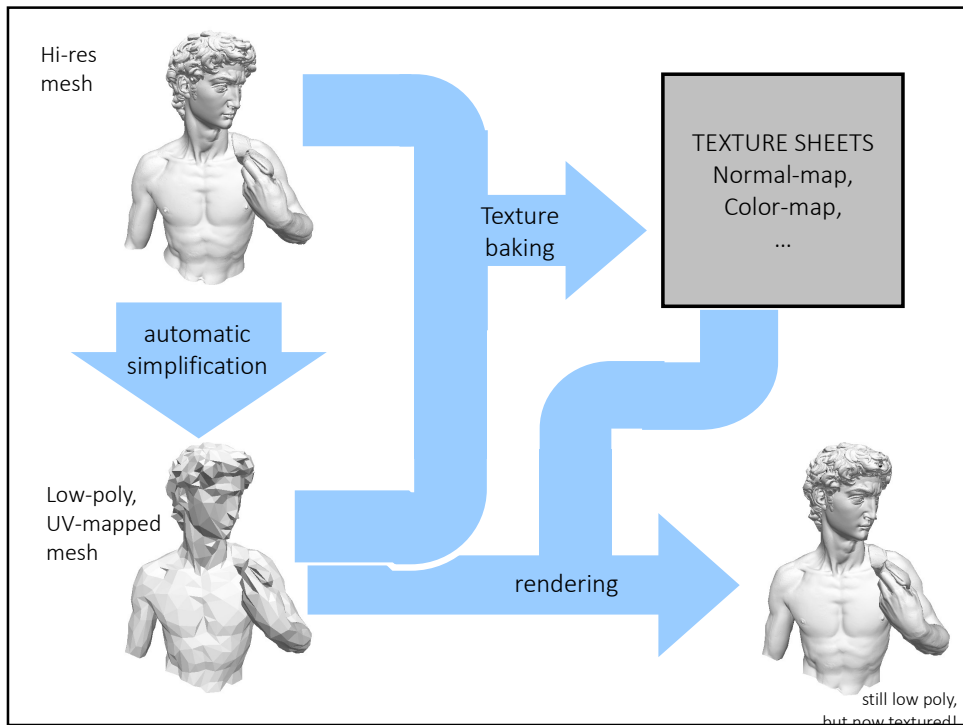
106

Texture baking: texture synthesis from hi-res models

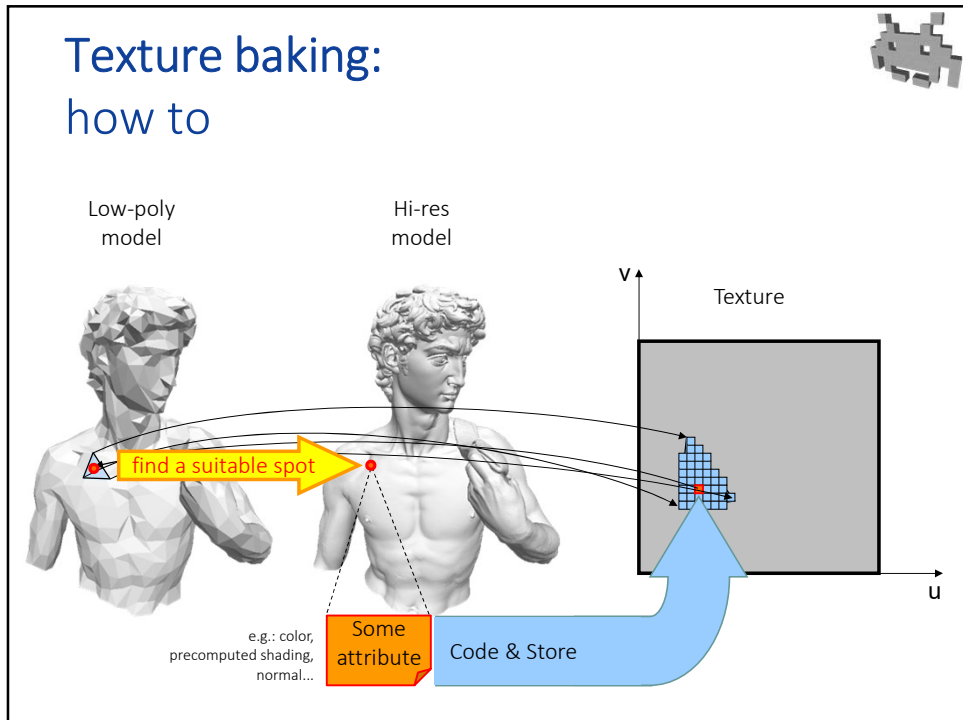


- input examples:
 - low-poly mesh A obtained from hi-res mesh B via **automatic simplification** or **manual retopology**
 - hi-res mesh B obtained from low-poly mesh A via **sculpting**
 - output examples:
 - attributes = normals
→ an **object-space normal map** is produced
 - attributes = base colors
→ 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!

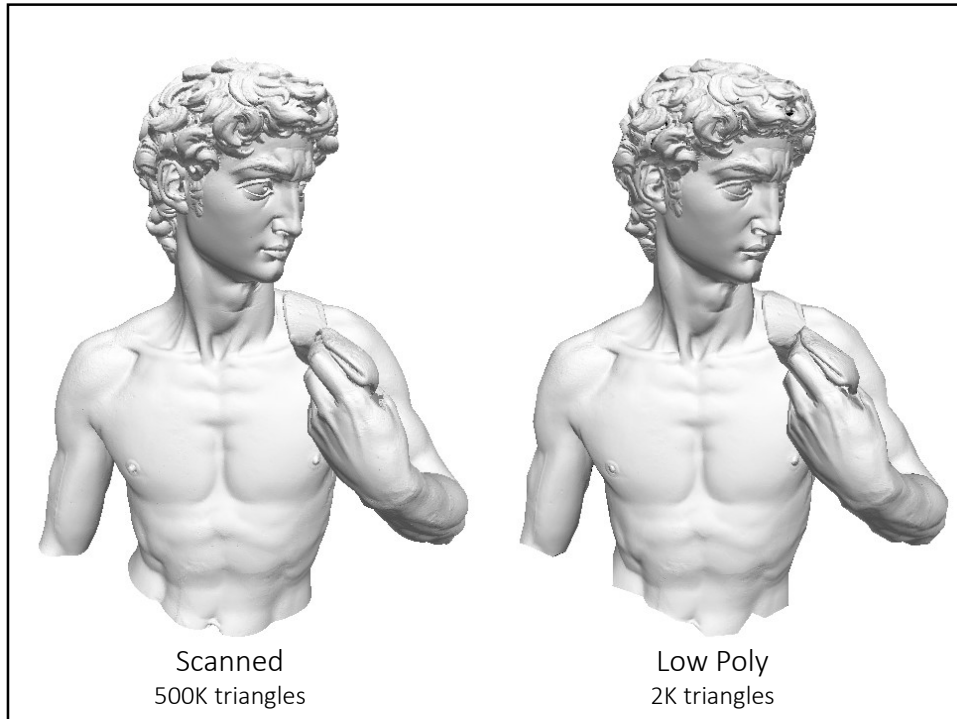
107



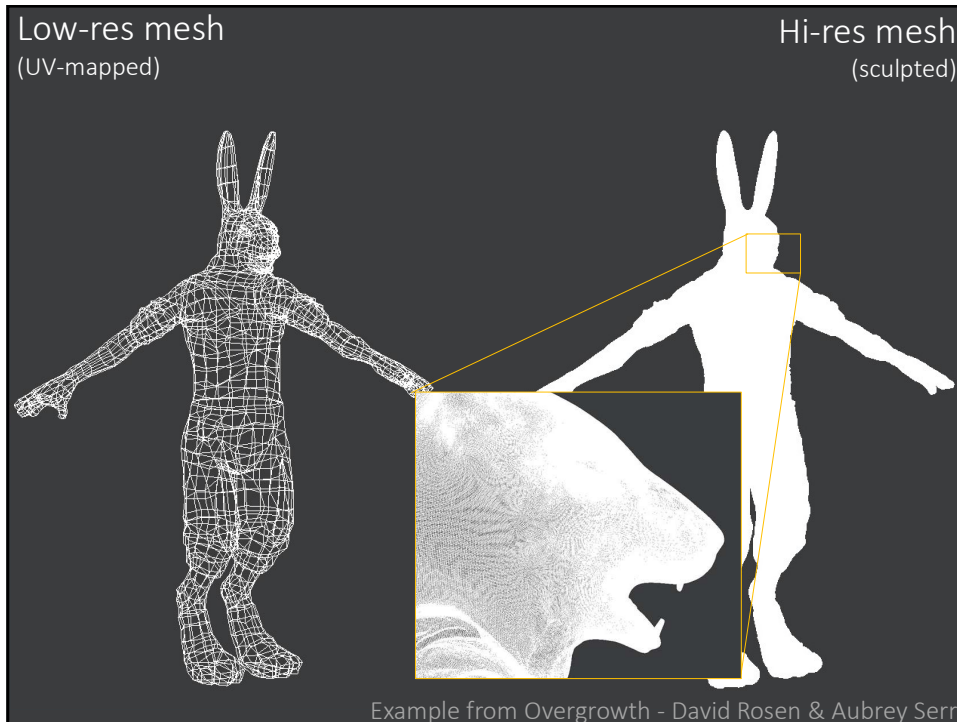
108



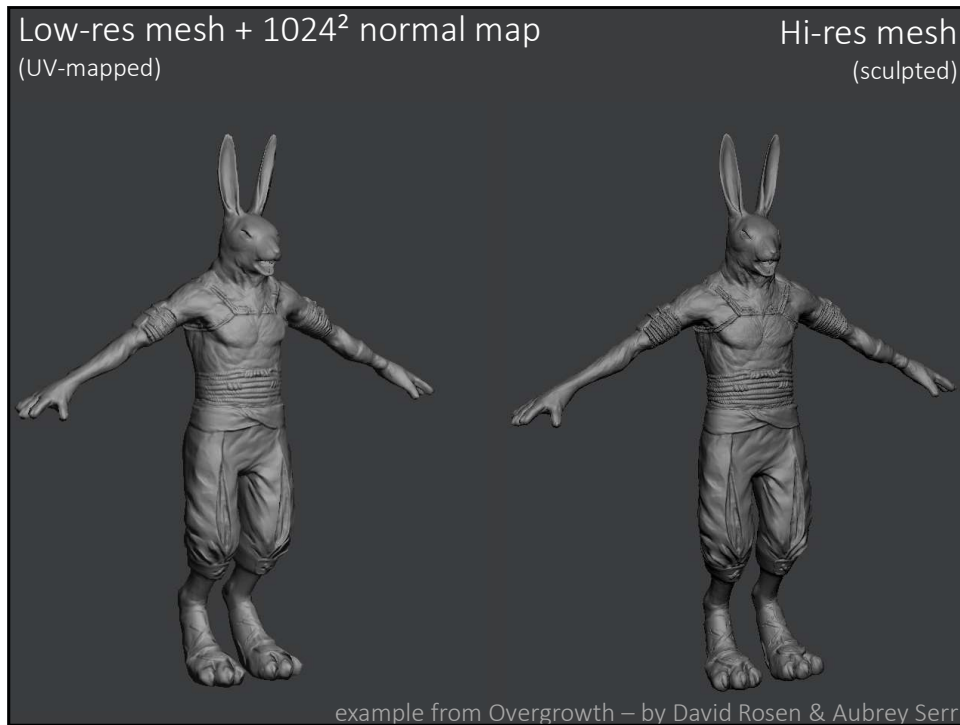
109



110



111



112

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!

118

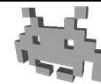
Asset production pipeline: an example



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)

119

Procedural Textures (in general)



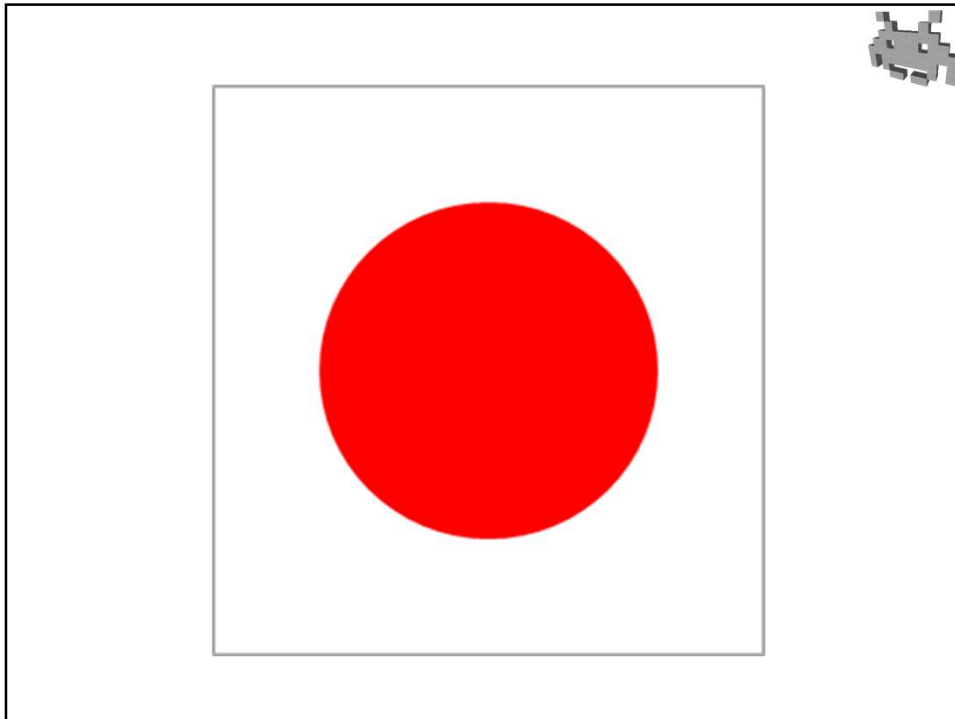
$$f \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

in $[0..1] \times [0..1]$

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

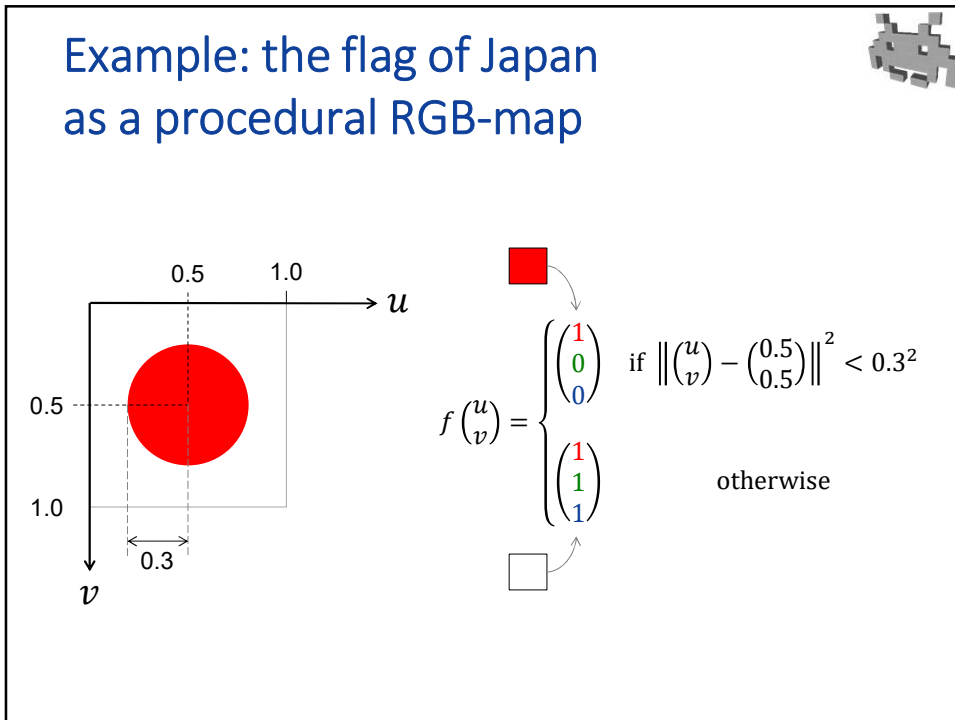
- A function from (u,v) to texel values
 - Plainly *replaces* a texture fetch!
 - Computed *during rendering* for each pixel (fragment shader)
 - Therefore, implemented in shader languages (e.g. GLSL, HLSL)
- Costs/benefits (the usual ones): see Lecture on Rendering and Real Time Graphics course
 - RAM / 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

120



121

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


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

122



123

Solid Textures


- Volumetric voxelized **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?

124

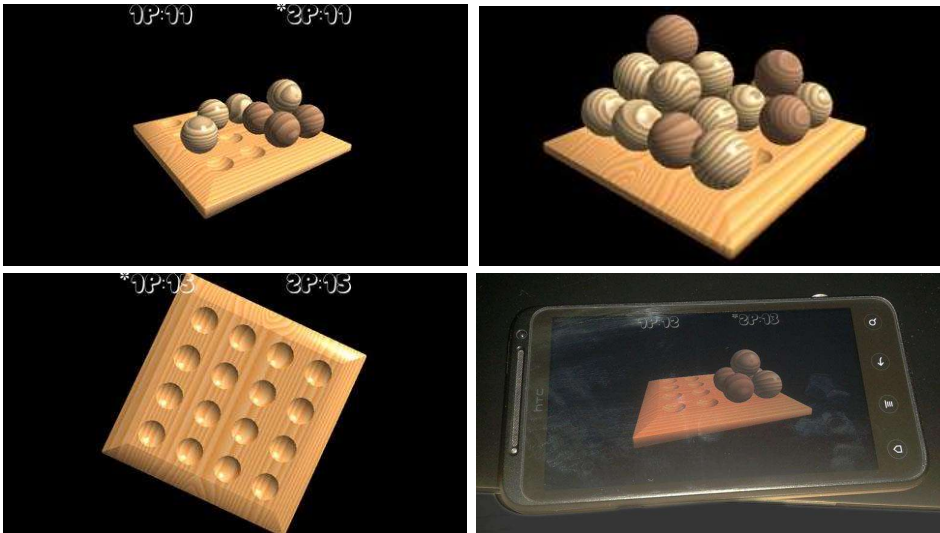
Procedural Solid Textures


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

example by 

125

Procedural Solid Textures



Gyross – project by Paolo P. Slepoi

126