




Course Plan

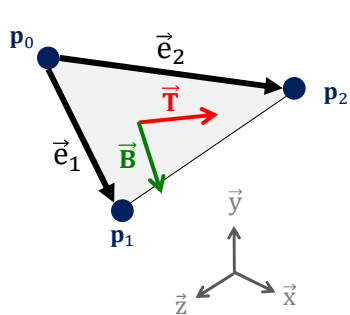
- lec. 1: **Introduction** ●
- lec. 2: **Mathematics** for 3D Games ●●●●●
- lec. 3: **Scene Graph** ●
- lec. 4: Game 3D Physics ●●● + ●●●
- lec. 5: Game Particle Systems ▸
- lec. 6: Game 3D Models ●●
- lec. 7: Game Textures ▸●●
- lec. 8: Game 3D Animations ▸●●
- lec. 9: Game 3D Audio ●
- lec. 10: **Networking**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

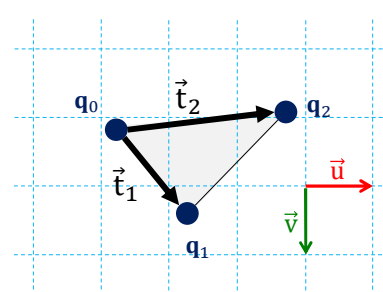
81

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



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





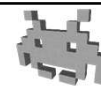
Idea:

\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

82

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 $\vec{\mathbf{e}}_{1,2}$
and 2D edge vectors $\vec{\mathbf{t}}_{1,2}$
- Find scalars a, b and c, d such that...

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

- Then

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

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

in matrix form:

solve with a 2x2 matrix inversion

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

- Then

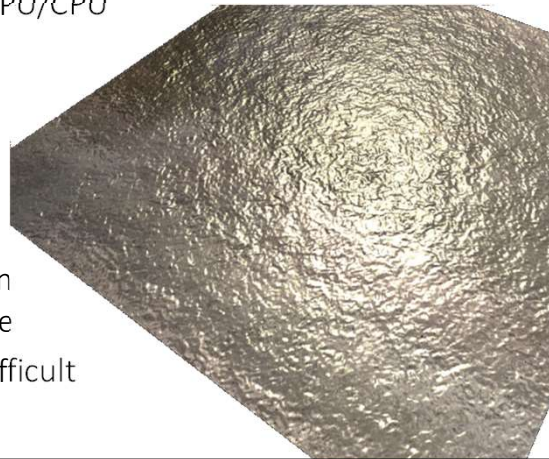
$$\vec{\mathbf{T}} = a \vec{\mathbf{e}}_1 + b \vec{\mathbf{e}}_2 \quad \vec{\mathbf{B}} = c \vec{\mathbf{e}}_1 + d \vec{\mathbf{e}}_2$$

84

How are normal-maps obtained? (4/5) procedural generation (rare)



- 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



85

Procedural Textures (in general)



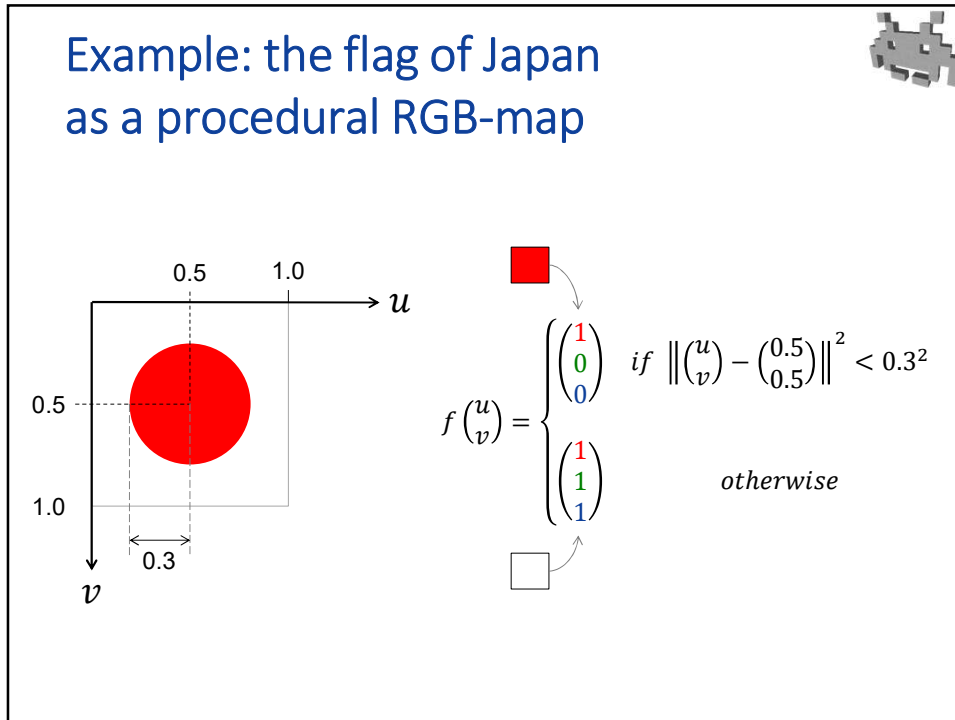
$$f \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$

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

86



87



88

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-mapping**!
 - Texture indexed by geometric mesh (rescaled)
- ⚠ Problem: ram space
 - Cubic wrt the resolution
 - Solution: procedural 3D texture?

89

Procedural Solid Textures

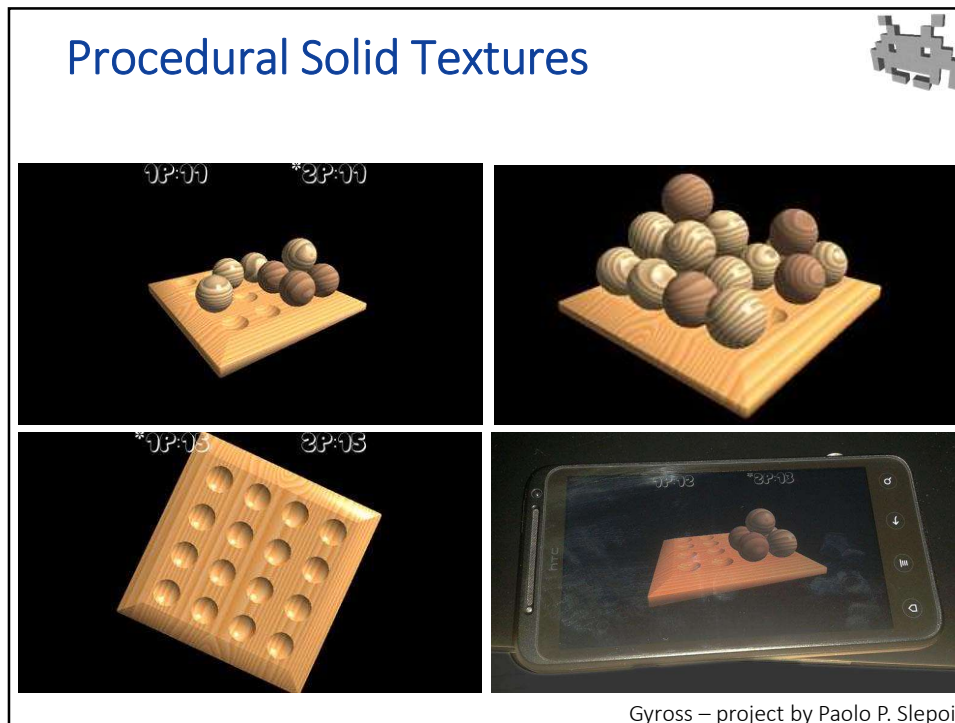


$$f \begin{pmatrix} u \\ v \\ s \end{pmatrix} = \begin{pmatrix} r \\ g \\ b \end{pmatrix}$$



example by  MODO ONLINE HELP

90



91

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!

92

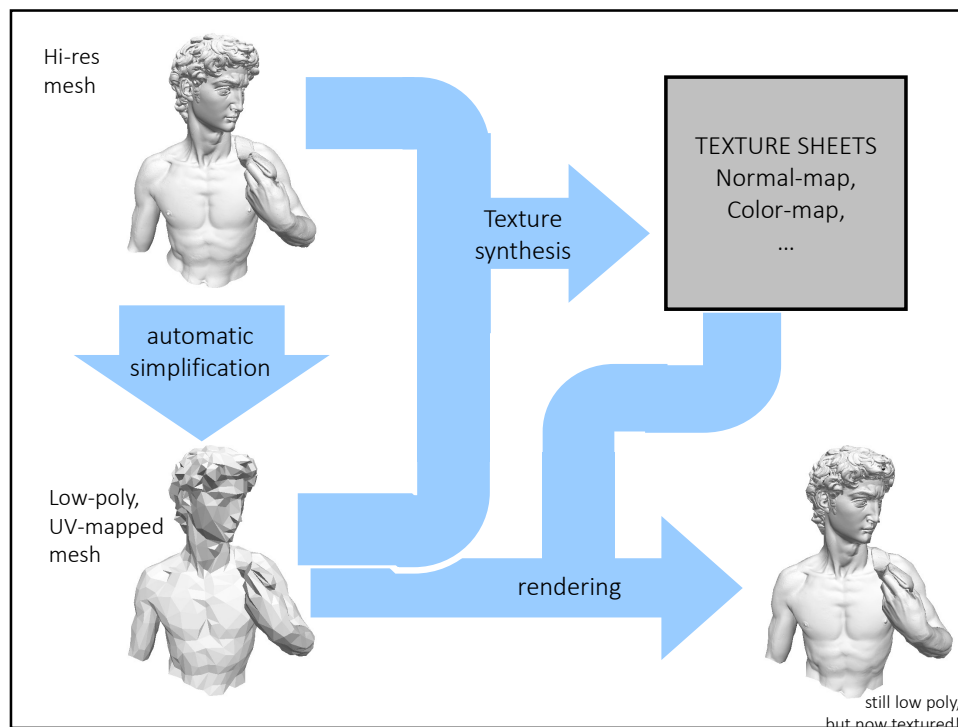
Texture baking: texture synthesis from hi-res models

- 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

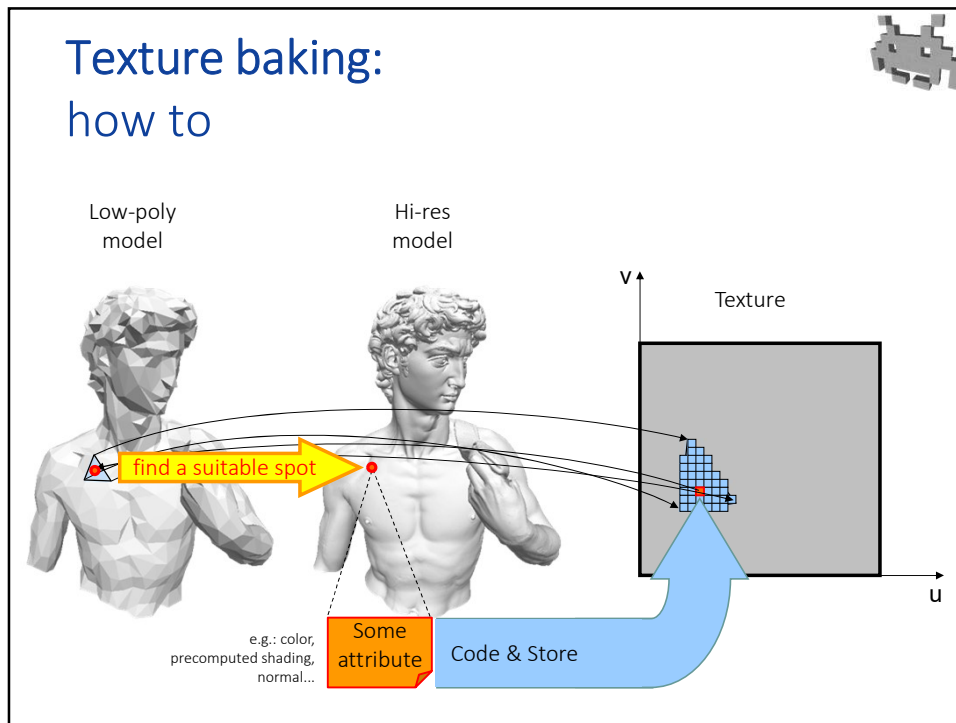
then converted to tangent space (using mesh A)

common case!

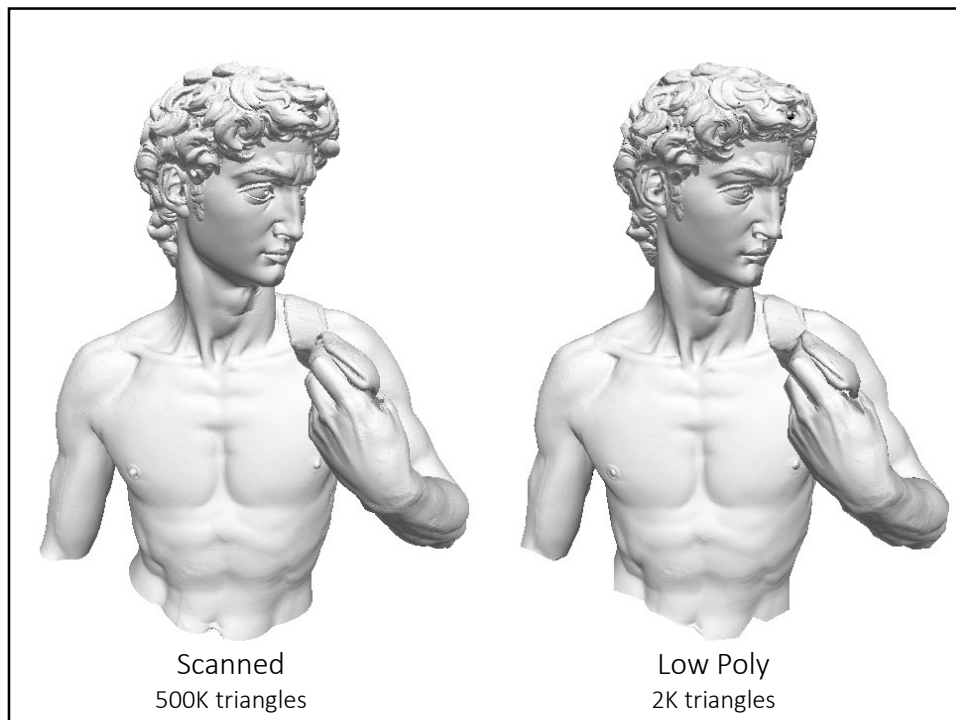
93



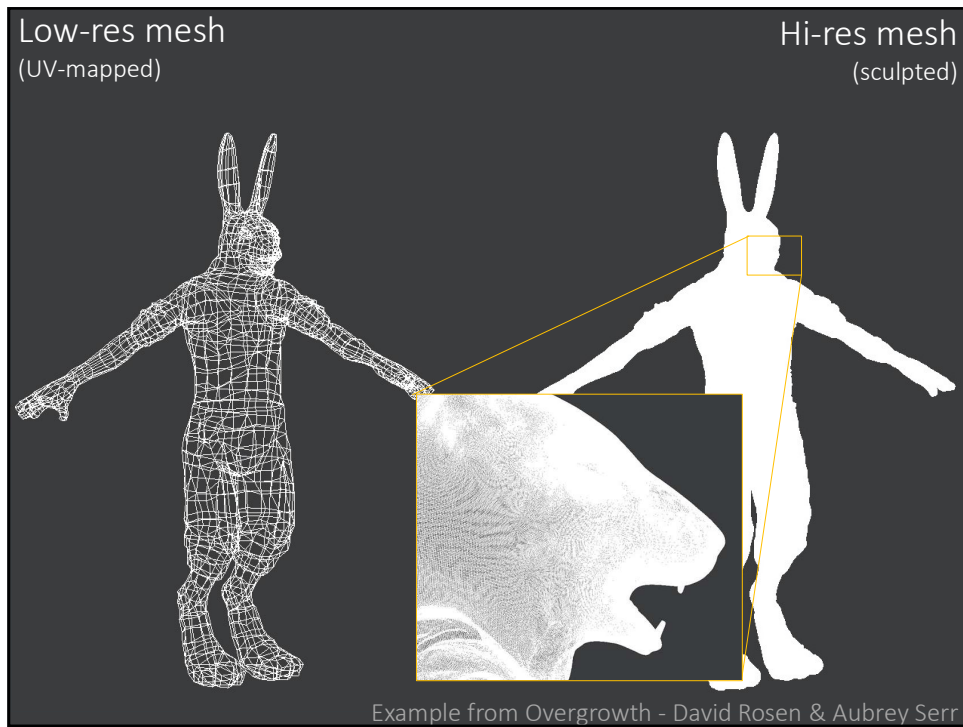
94



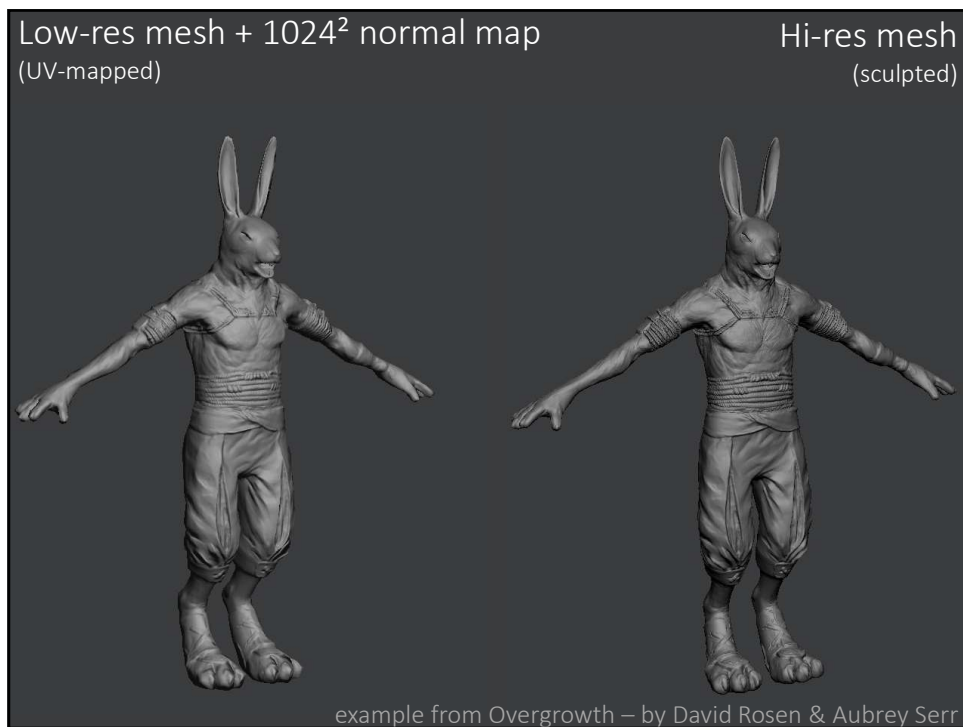
95



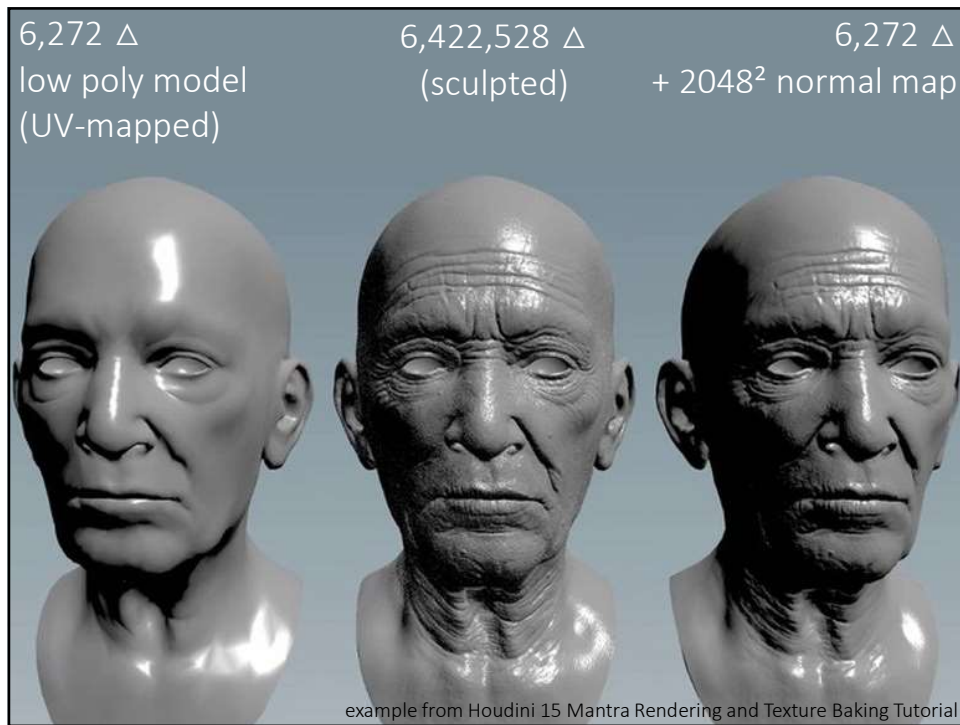
96



97



98



99



100

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!

105

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)

106