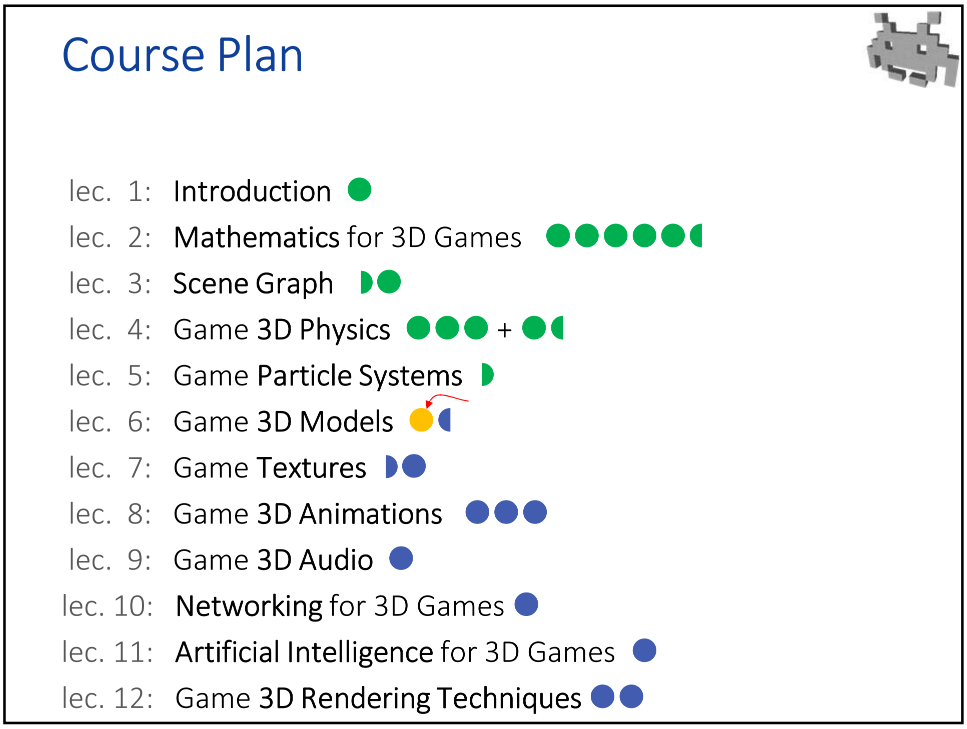
3D video games

Models for Games

Marco Tarini

REMOTE TEACHING!

1



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 ●●

2

Solomons's key
(1986, Temco)
on Z80

Metal Slug (1996, Nazca Copr), on Neo Geo (SNK)

reminder:
during the '80s – early '90s, the principal asset in games consisted in sprites / tilemaps authored by pixel artists ...

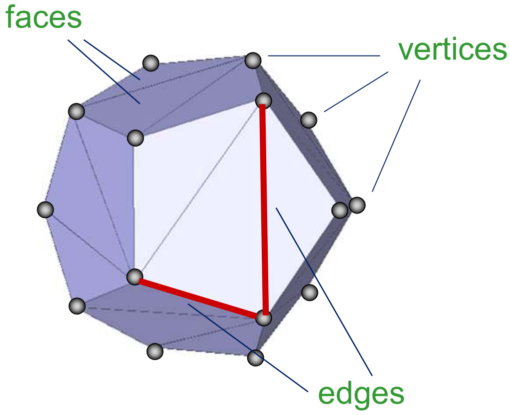
Triangle Meshes

The visual appearance of 3D objects

- Data structure for modelling 3D objects
  - GPU friendly
  - Resolution = number of faces
  - (Potentially) Adaptive resolution
- Used in games to represent the visual appearance of 3D objects
  - at least, the ones which can be represented by their surface
  - most solid objects (rigid or not)
- Mathematically: a piecewise linear surface
  - a bunch of surface samples "vertices"
    connected by a set of triangular "faces"
    attached side to side by "edges"

Triangle Mesh
(or simplicial mesh)

- A set of adjacent triangles



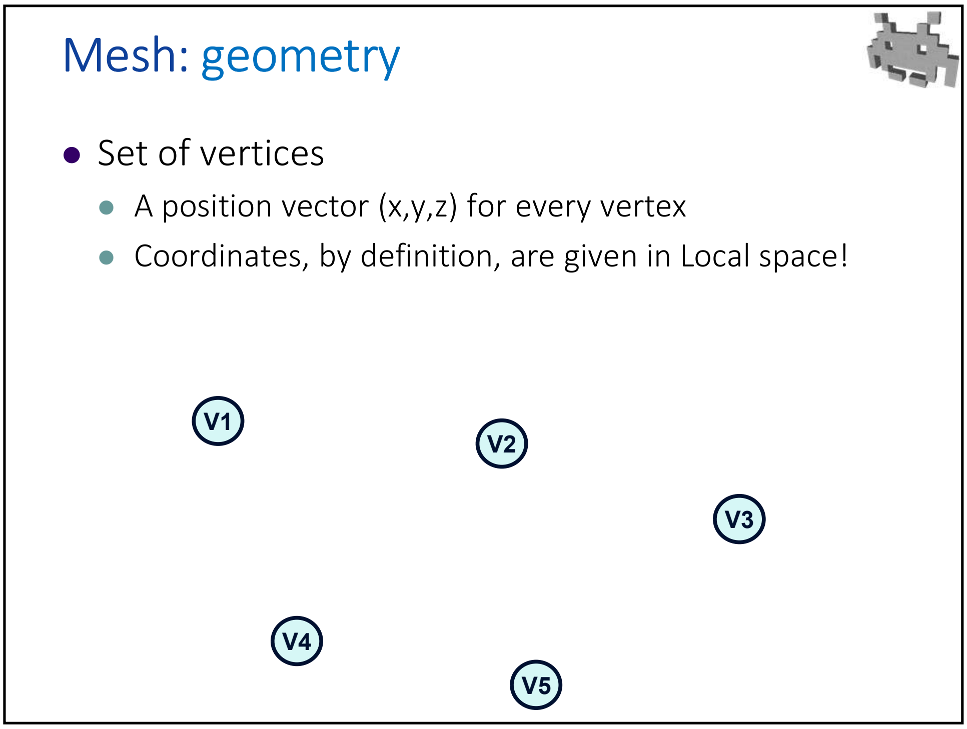
5

Mesh:
data structure

A mesh is made of

- geometry
  - The vertices, each with pos (x,y,z)
  - It's a sampling of the surface
- connectivity or topology
  - Faces connecting the vertices
    - Triangle mesh: faces are triangles (what the GPU is designed to render!)
    - (pure) quad mesh: faces are quadrilateral
    - Quad dominant mesh: most faces are quadrilateral
    - Polygonal mesh: faces are polygons (general case)
- attributes
  - Ex.: color, material, normal, UV, …

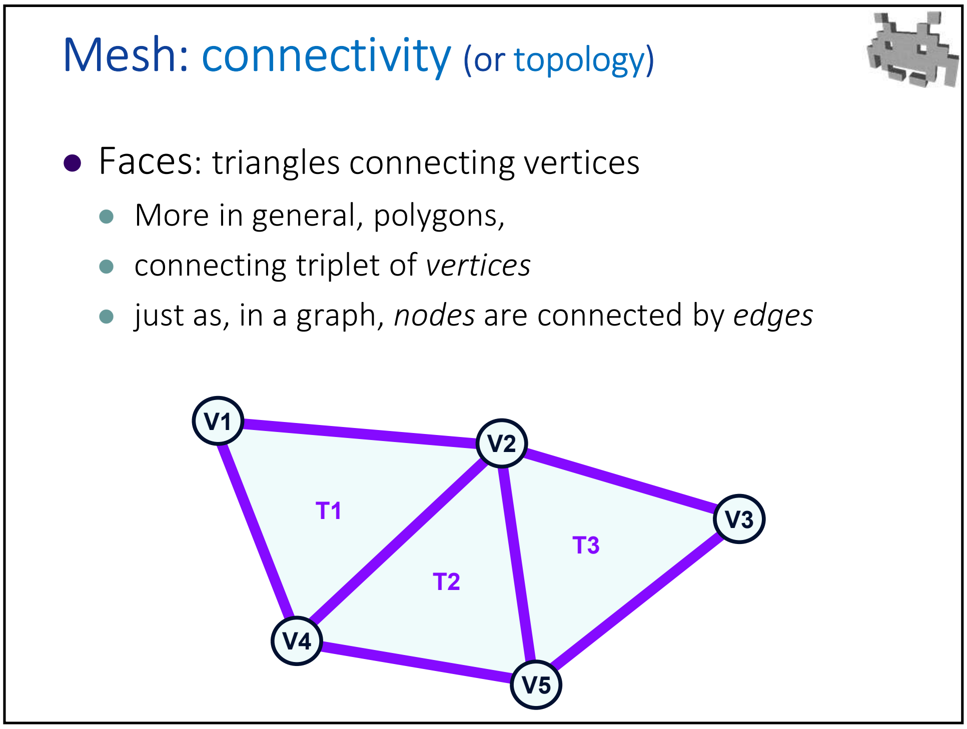
6



Mesh: geometry

- Set of vertices
  - A position vector (x,y,z) for every vertex
  - Coordinates, by definition, are given in Local space!

7



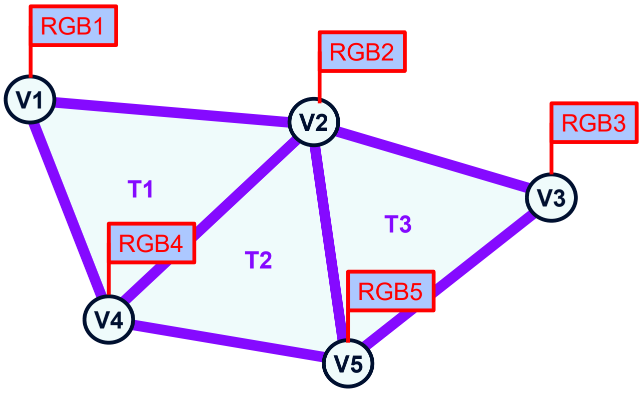
Mesh: connectivity (or topology)

- Faces: triangles connecting vertices
  - More in general, polygons,
  - connecting triplet of *vertices*
  - just as, in a graph, *nodes* are connected by *edges*

8

Mesh: attributes

- Any quantity that varies over the surface
  - sampled at vertices, and interpolated inside triangles



9

Mesh as a data structure: soup of triangles

- Simply, an array of triangles
- Each triangle stored as: sequence of 3 vertices
- Each vertex stored as:
  x,y,z coordinates + attributes
- Problem: data replication
  - Not very memory efficient
  - Inconvenient to update (e.g. to animate)
  - Not very used

most faces are adjacent to each other (adjacent faces share the same vertices)

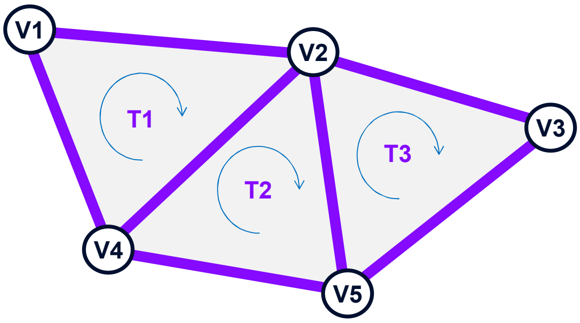
10

Mesh as a data strucuture: indexed meshes

- array of vertices
  - Each vertex stored as
    - *x,y,z* position (aka the "geometry" of the mesh)
    - attributes: (all vertices, the same ones)
      any data saved on the surface: e.g. color
- array of triangles
  - the "connectivity» (or, "topology") of the mesh
  - Each triangle stored as
    - triplet of indices (referring to a vertex in the array)
- The two arrays can be seen as tables

we can consider positions as attributes too

An indexed mesh in GPU ram = two buffers



vert	X	Y	Z	R	G	B
V1	x1	y1	z1	r1	g1	b1
V2	x2	y2	z2	r2	g2	b2
V3	x3	y3	z3	r3	g3	b3
V4	x4	y4	z4	r4	g4	b4
V5	x5	y5	z5	r5	g5	b5

GEOMETRY + ATTRIBUTES

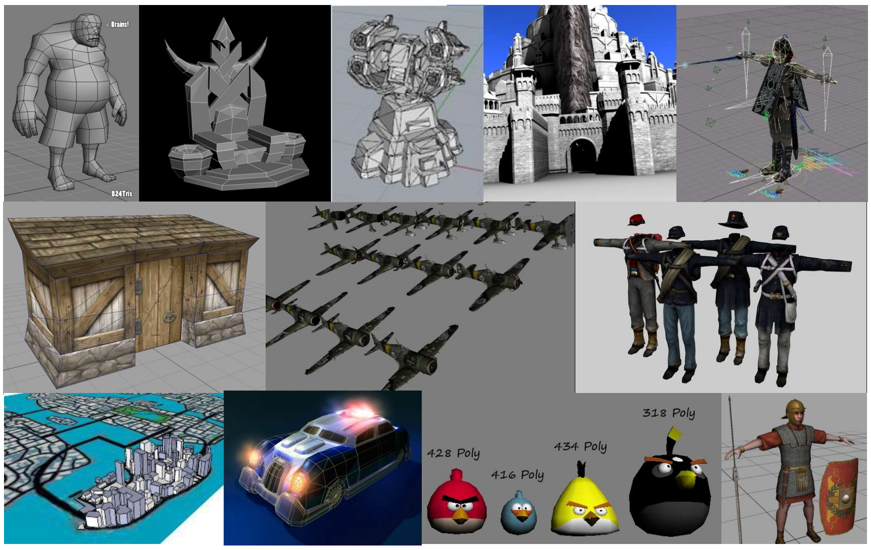
Tri:	Wedge 1:	Wedge 2:	Wedge 3:
T1	V4	V1	V2
T2	V4	V2	V5
T3	V5	V2	V3

CONNECTIVITY

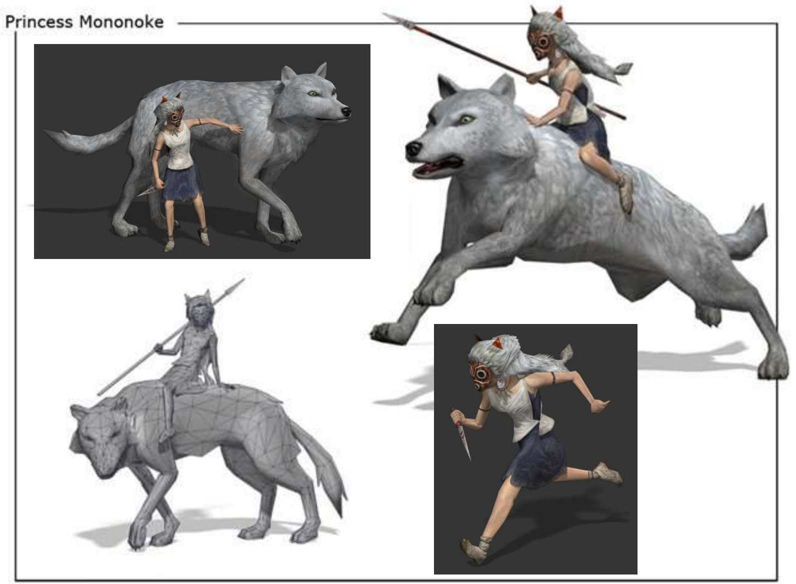
Mesh resolution

- Defined as the number of faces
  - or vertices, equivalent because typically #F ≈ 2 · #V)
- Rendering time is linear with resolution
  - therefore, in games, resolution is kept small
  - aka. «low-poly» models
- Resolution can be adaptive:
  - denser vertices & smaller faces in certain parts
  - sparser vertices & larger faces in other parts
- Resolution of typical models increases with time
  - e.g. 1990s: 10^5 △ is hi-res
  - 2000s: 10^{10} △ is hi-res

In games: "Low-Poly" models (low resolution meshes)



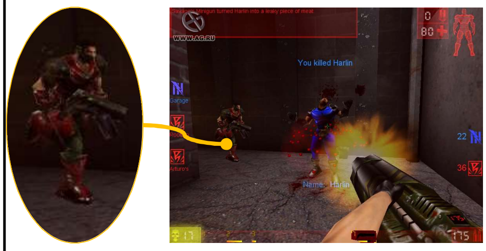
Low-poly models



by Phillip Heckinger (3D modeller)

16

Resolution increases over time

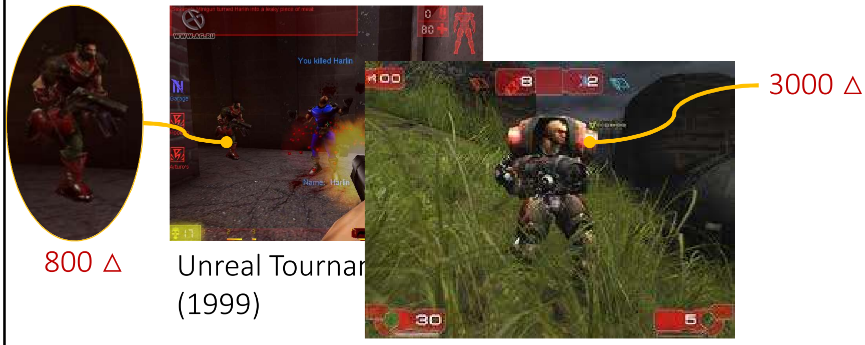


800 △

Unreal Tournament
(1999)

17

Resolution increases over time



800 △

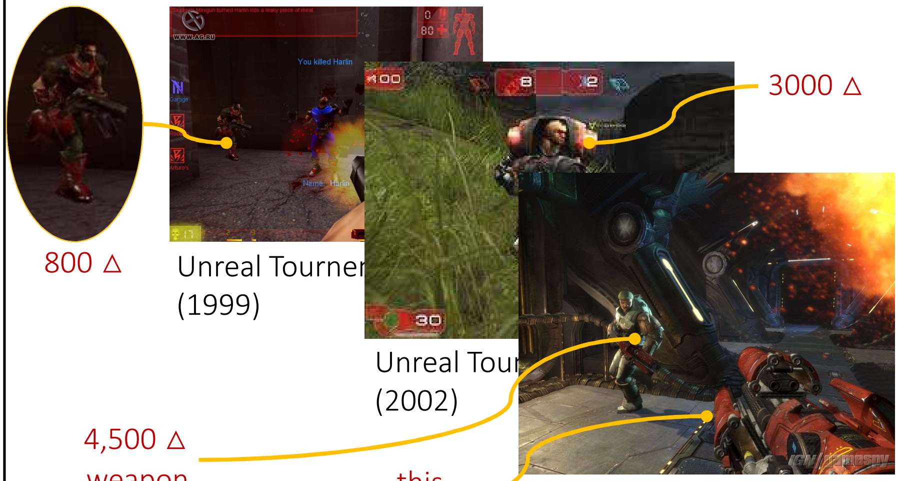
Unreal Tournar
(1999)

3000 △

Unreal Tournament 2K3
(2002)

18

Resolution increases over time



800 △

Unreal Tourner
(1999)

3000 △

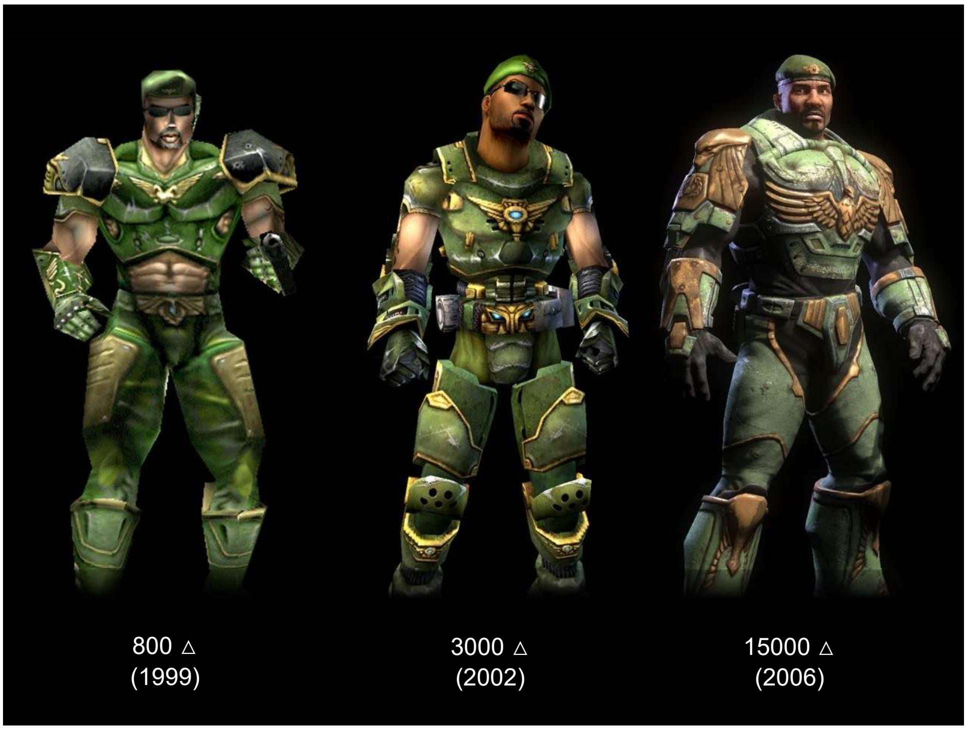
Unreal Tour
(2002)

4,500 △
weapon

this
12,000 △

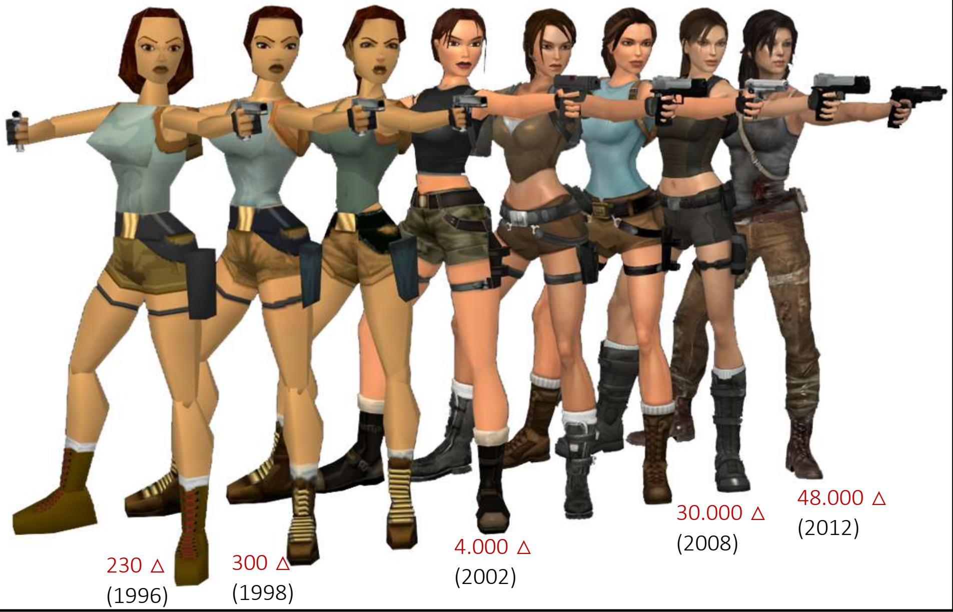
Unreal Tournament 3
(2007)

19

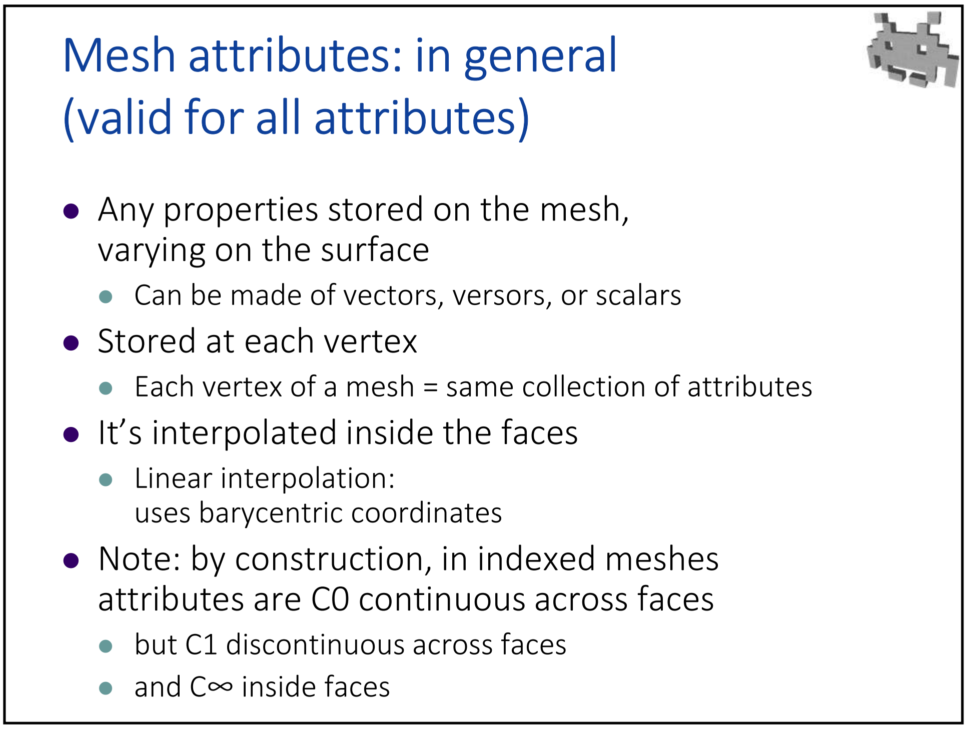


20

Resolution increases over time



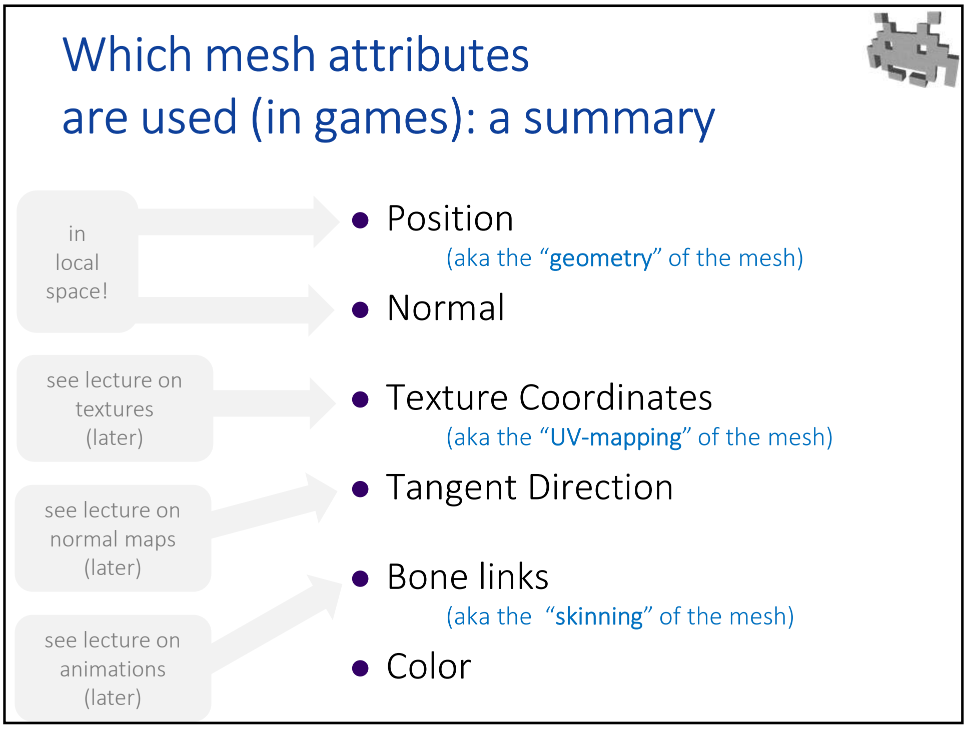
21



Mesh attributes: in general (valid for all attributes)

- Any properties stored on the mesh, varying on the surface
  - Can be made of vectors, versors, or scalars
- Stored at each vertex
  - Each vertex of a mesh = same collection of attributes
- It's interpolated inside the faces
  - Linear interpolation:
    uses barycentric coordinates
- Note: by construction, in indexed meshes attributes are C0 continuous across faces
  - but C1 discontinuous across faces
  - and C∞ inside faces

22



Which mesh attributes are used (in games): a summary

in local space!

see lecture on textures (later)

see lecture on normal maps (later)

see lecture on animations (later)

- Position
  (aka the "**geometry**" of the mesh)
- Normal
- Texture Coordinates
  (aka the "**UV-mapping**" of the mesh)
- Tangent Direction
- Bone links
  (aka the "**skinning**" of the mesh)
- Color

23

Which mesh attributes are used in games: a summary

- *Normal*
  - used for dynamic re-lighting
- *Texture coordinates*
  - aka the “uv-mapping” of the mesh
  - used for texture mapping
- *Tangent direction*
  - used for normal mapping
  - used for anisotropic lighting effects
- *Bone links*
  - aka the “skinning” of the mesh
  - used for skeletal animation
- *Color*
  - used for baked lighting (e.g. ambient occlusion)
  - used for «base» («diffuse») color (RGB)

SEE RENDERING LATER

SEE TEXTURES LATER

SEE TEXTURES LATER

SEE RENDERING LATER

SEE ANIMATIONS LATER

SEE RENDERING LATER

Mesh as tables

Tri:	W1:	W2:	W3:
T0			
T1			
T2			
T3			
T4			
T5			
T6			
T7			

CONNECTIVITY

- Position
- Normal
- Color
- Texture Coordinate
- Tangent Direction
- Bone links

vert	X	Y	Z	Nx	Ny	Nz	R	G	B	A	U	V	Tx	Ty	Tz	Bx	By	Bz
V0																		
V1																		
V2																		
V3																		
V4																		

GEOMETRY + ATTRIBUTES

Mesh attributes: colors

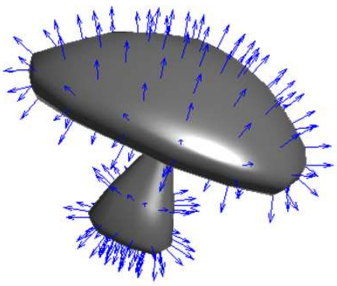
- In games, colors on 3D models are usually determined by textures (not by mesh colors)
  - reason: more resolution in signal
- Per vertex colors can be used…
  - To cheaply add variations models
    - Red guards, blue guards

      SEE RENDERING LATER
  - To bake lighting
    - e.g. baked per-vertex ambient occlusion see rendering later
  - To dynamically recolor mesh parts
    - e.g. redden the tip of a sword which is blood soaked
    - e.g. accumulate dirty

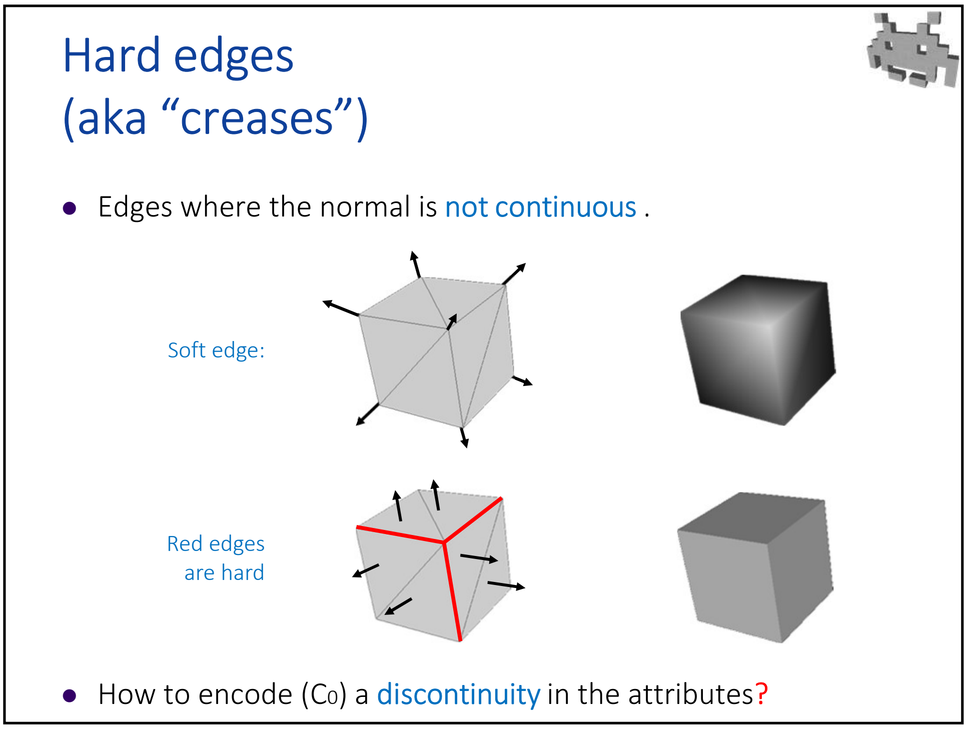
26

Mesh attributes: normals

- A versor
- Representing the surface orientation
- Main use: lighting computation
- Can be computed automatically from geometry…
- But it is a part of the mesh assets:
  - the artist is in control of which edges are *soft* and which are *hard*



27



Hard edges (aka “creases”)

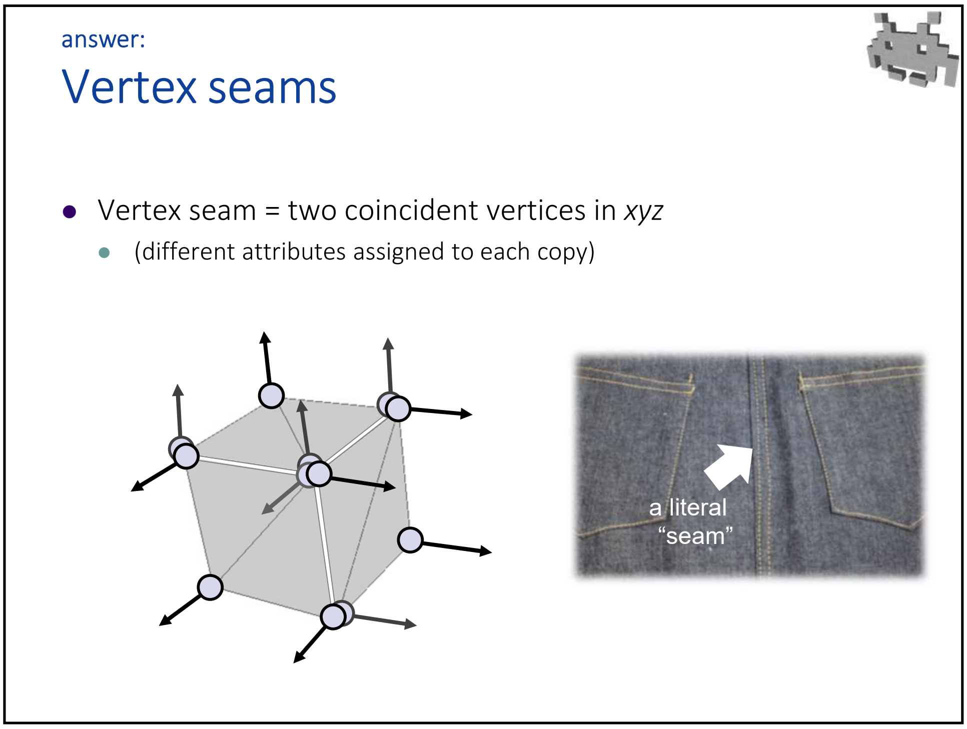
- Edges where the normal is not continuous .

Soft edge:

Red edges are hard

- How to encode (C_0) a discontinuity in the attributes?

28

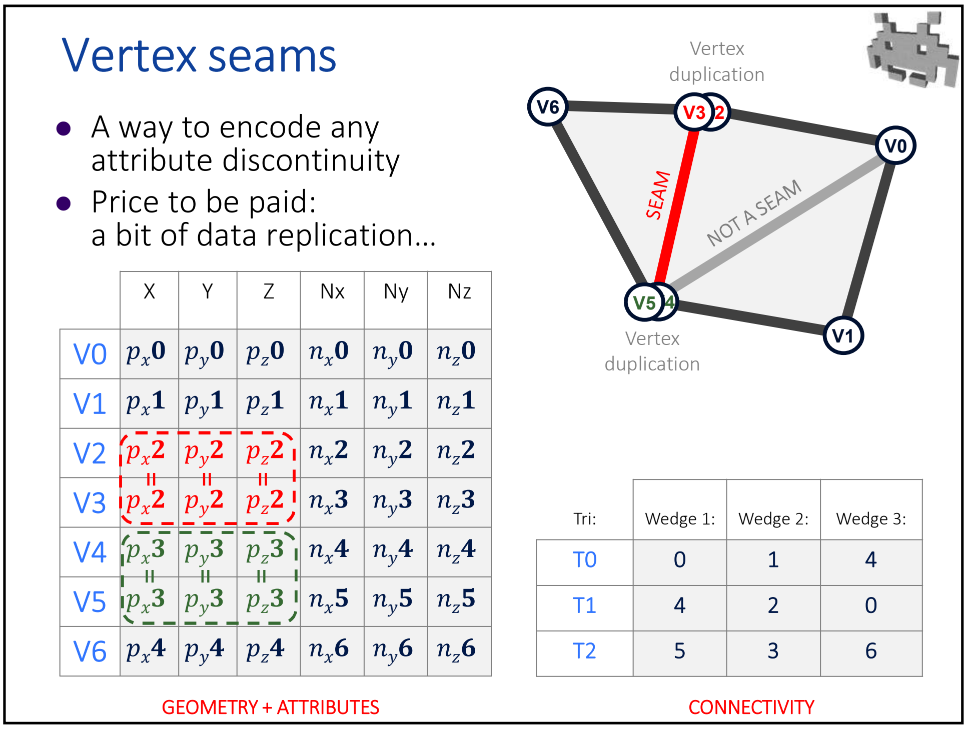


answer:

Vertex seams

- Vertex seam = two coincident vertices in *xyz*
  - (different attributes assigned to each copy)

29



Vertex seams

- A way to encode any attribute discontinuity
- Price to be paid: a bit of data replication…

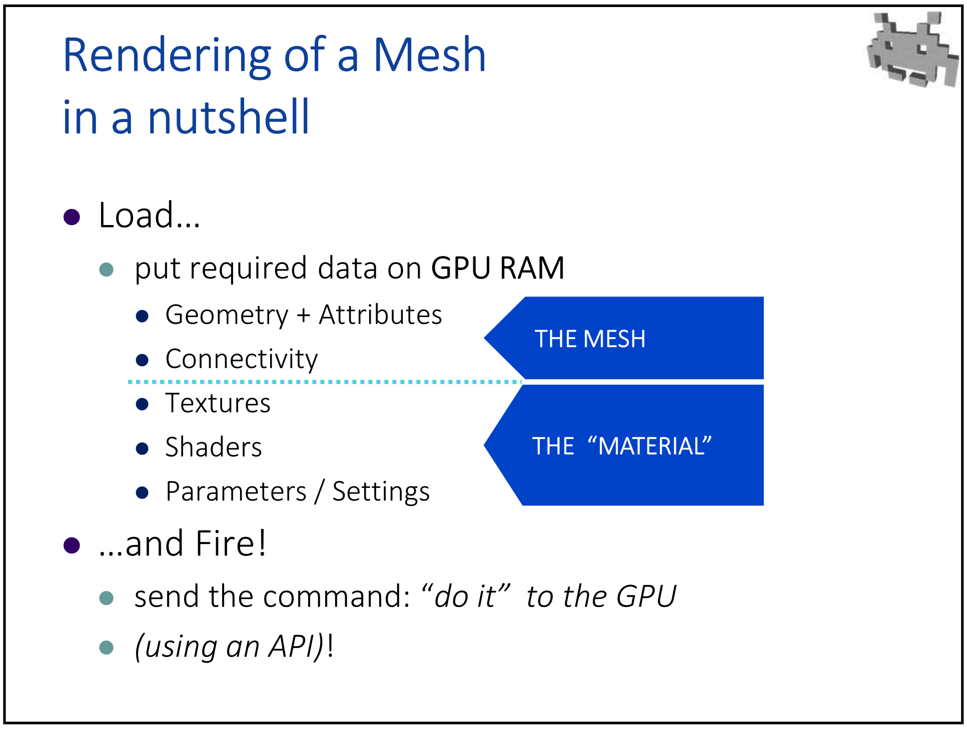
	X	Y	Z	Nx	Ny	Nz
V0	p_x0	p_y0	p_z0	n_x0	n_y0	n_z0
V1	p_x1	p_y1	p_z1	n_x1	n_y1	n_z1
V2	p_x2	p_y2	p_z2	n_x2	n_y2	n_z2
V3	p_x2	p_y2	p_z2	n_x3	n_y3	n_z3
V4	p_x3	p_y3	p_z3	n_x4	n_y4	n_z4
V5	p_x3	p_y3	p_z3	n_x5	n_y5	n_z5
V6	p_x4	p_y4	p_z4	n_x6	n_y6	n_z6

GEOMETRY + ATTRIBUTES

Tri:	Wedge 1:	Wedge 2:	Wedge 3:
T0	0	1	4
T1	4	2	0
T2	5	3	6

CONNECTIVITY

30



Rendering of a Mesh in a nutshell

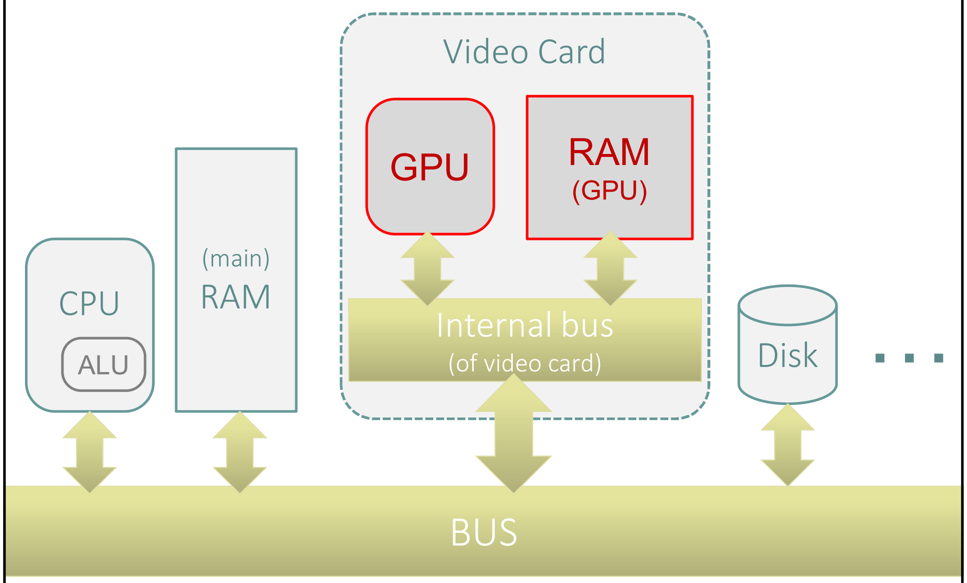
- Load…
  - put required data on GPU RAM
    - Geometry + Attributes
    - Connectivity
    - Textures
    - Shaders
    - Parameters / Settings
- …and Fire!
  - send the command: *“do it” to the GPU*
  - *(using an API)*!

THE MESH

THE “MATERIAL”

31

Simplified architecture of PC with Video Card



32

Rendering of a Mesh in a nutshell

Might change in the future?

- The algorithm to render a mesh (in games) is based on rasterization
  - It is outside the scope of this course. See CG course.
  - In brief, three phases in cascade:
    each vertex is projected on screen ("transform"),
    (find where the vertex will be seen on the screen)
    then each triangle is rasterized (converted into pixels)
    then each pixel is processed (find the final color)

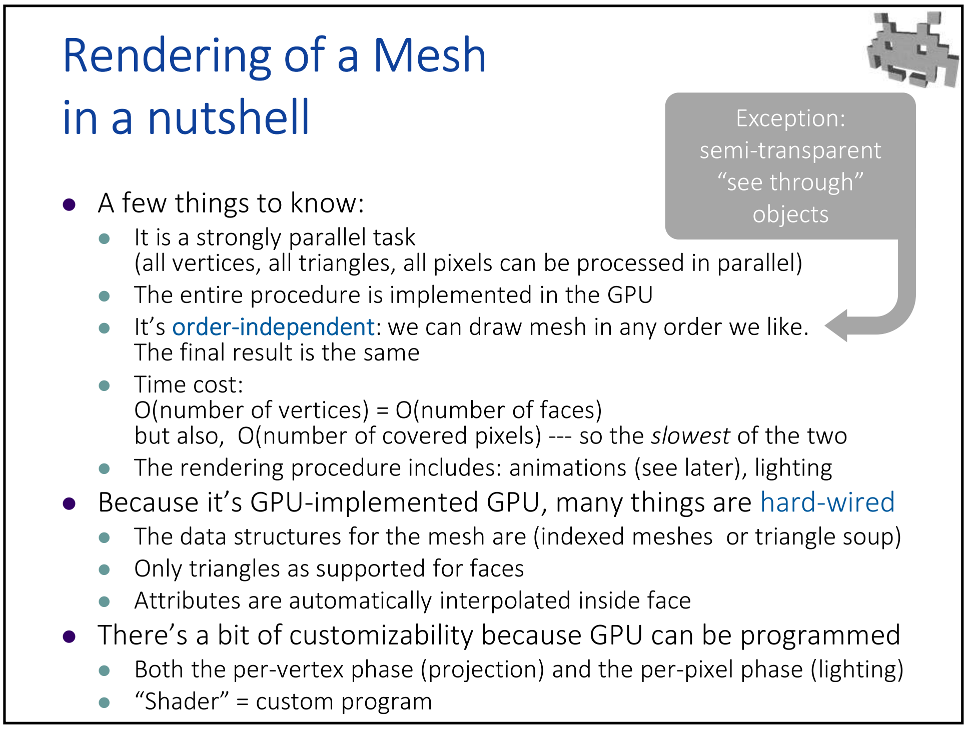
PER VERTEX PHASE

PER TRIANGLE PHASE

PER PIXEL PHASE

- For our purposes, rendering a mesh means just: load all required data on the card on the GPU and send the command to render it (the "draw call")
  - data includes the mesh itself (the two tables)
  - plus the current transformations (from local space to view space)
  - plus data describing the view: the "material", including textures

33

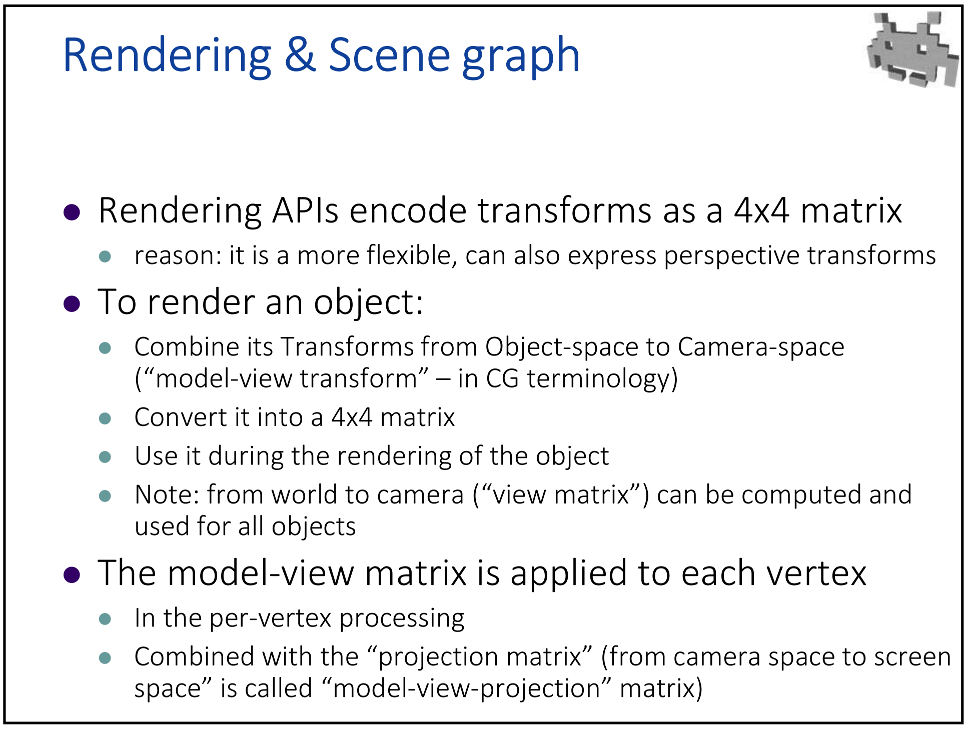


Rendering of a Mesh in a nutshell

Exception:
semi-transparent
“see through”
objects

- A few things to know:
  - It is a strongly parallel task
    (all vertices, all triangles, all pixels can be processed in parallel)
  - The entire procedure is implemented in the GPU
  - It’s order-independent: we can draw mesh in any order we like.
    The final result is the same
  - Time cost:
    O(number of vertices) = O(number of faces)
    but also, O(number of covered pixels) --- so the *slowest* of the two
  - The rendering procedure includes: animations (see later), lighting
- Because it’s GPU-implemented GPU, many things are hard-wired
  - The data structures for the mesh are (indexed meshes or triangle soup)
  - Only triangles as supported for faces
  - Attributes are automatically interpolated inside face
- There’s a bit of customizability because GPU can be programmed
  - Both the per-vertex phase (projection) and the per-pixel phase (lighting)
  - “Shader” = custom program

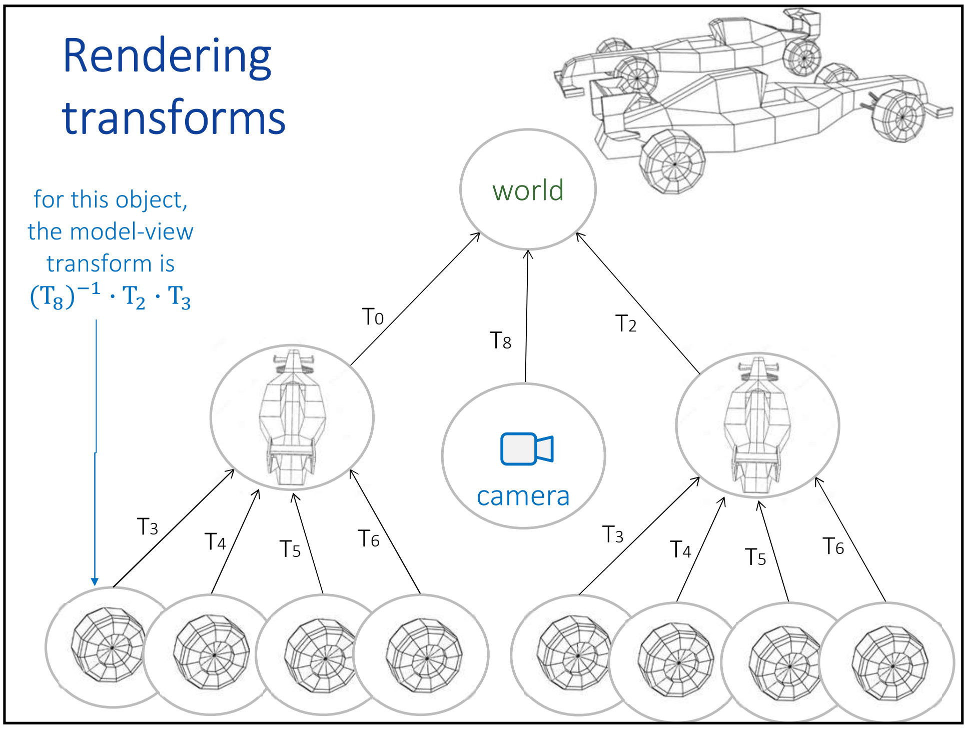
34



Rendering & Scene graph

- Rendering APIs encode transforms as a 4x4 matrix
  - reason: it is a more flexible, can also express perspective transforms
- To render an object:
  - Combine its Transforms from Object-space to Camera-space
    (“model-view transform” – in CG terminology)
  - Convert it into a 4x4 matrix
  - Use it during the rendering of the object
  - Note: from world to camera (“view matrix”) can be computed and used for all objects
- The model-view matrix is applied to each vertex
  - In the per-vertex processing
  - Combined with the “projection matrix” (from camera space to screen space” is called “model-view-projection” matrix)

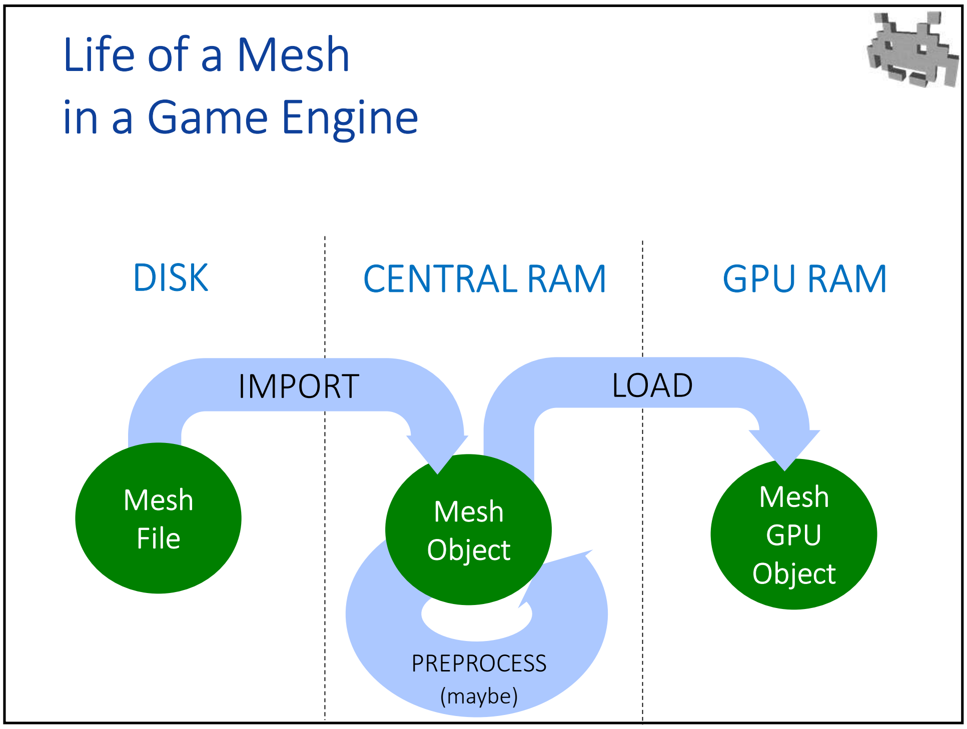
35



Rendering transforms

for this object, the model-view transform is $(T_8)^{-1} \cdot T_2 \cdot T_3$

36



Life of a Mesh in a Game Engine

DISK

CENTRAL RAM

GPU RAM

IMPORT

LOAD

Mesh File

Mesh Object

Mesh GPU Object

PREPROCESS (maybe)

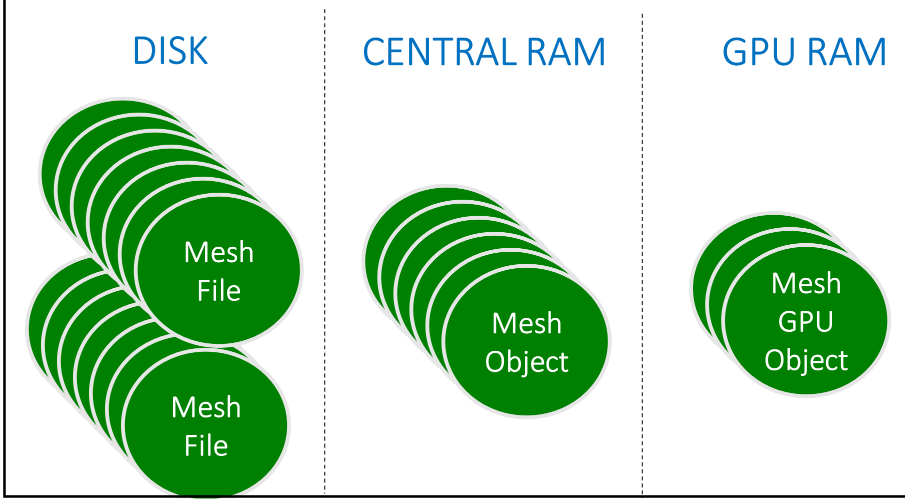
39

Life of a mesh in a game engine

- Import (from disk)
- Optionally, simple Pre-processing
  - e.g.: Compute Normals (if needed, i.e. rarely)
  - e.g.: Compute Tangent Dirs
  - e.g.: Bake Lighting (sometimes)
- Render (each frame)
  - GPU based
  - Meaning: mesh be loaded in GPU-ram first

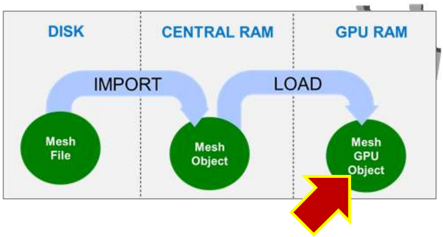
40

Memory Management (during game execution)



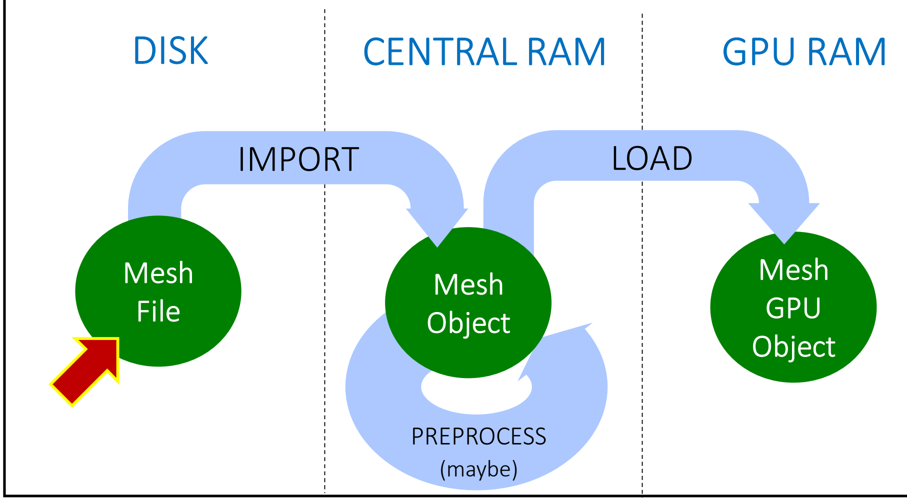
41

Mesh GPU Object (on Graphic Card)

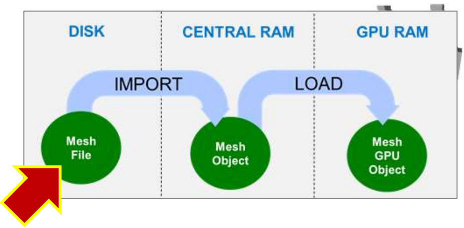


- Buffers storing the mesh
  - GPU APIs call them: Vertex Buffer Object or Vertex Arrays
- They are stored in GPU RAM
  - *The scarcest one !*
- Ready to render!
- Choices for a Game Engine:
  - storage formats, including precisions
  - trade-off between storage cost / accuracy
  - e.g.
    - color? 8 bit per channel
    - position? 16 bit per coordinate

Life of a Mesh in a Game Engine

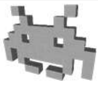


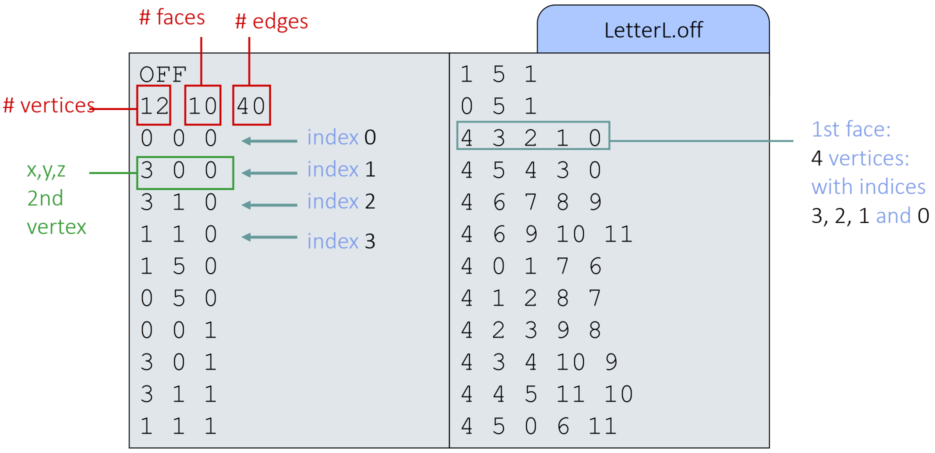
Mesh as an asset



- A file of a given format sitting on the disk
- Choices for the game engine:
  - which formats(s) to import?
    - proprietary, standard…
  - storing which attributes?
- Issues:
  - storage cost
  - loading time

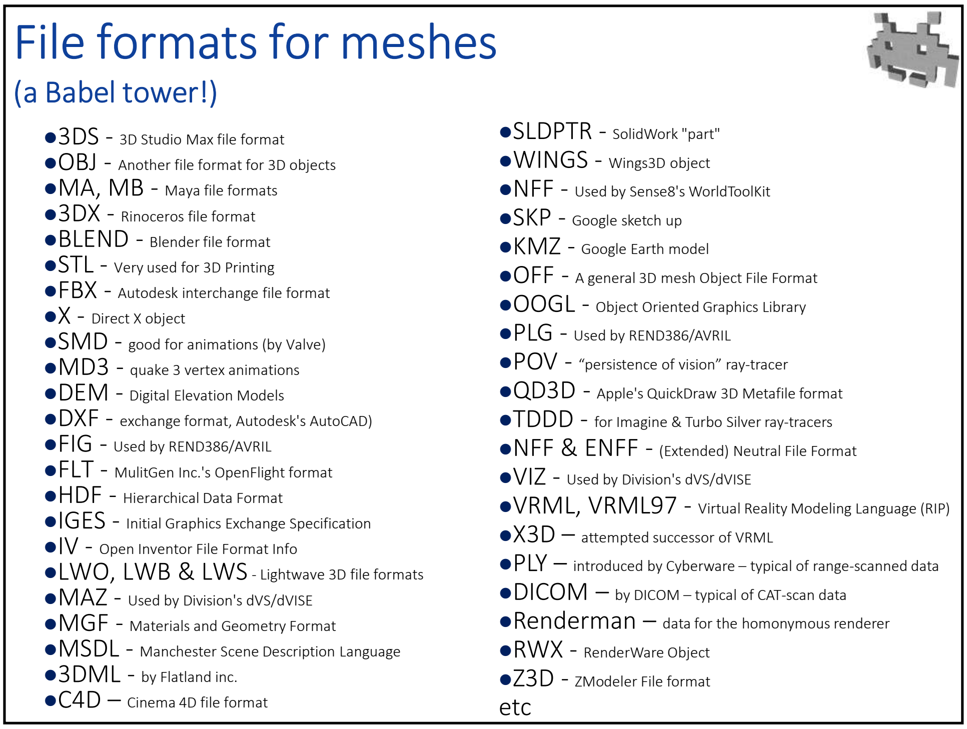
Example of file format for indexed meshes: OFF format





```
OFF              1  5  1
12 10 40         0  5  1
0  0  0          4  3  2  1  0
3  0  0          4  5  4  3  0
3  1  0          4  6  7  8  9
1  1  0          4  6  9  10  11
1  5  0          4  0  1  7  6
0  5  0          4  1  2  8  7
0  0  1          4  2  3  9  8
3  0  1          4  3  4  10  9
3  1  1          4  4  5  11  10
1  1  1          4  5  0  6  11
```

1st face:
4 vertices:
with indices
3, 2, 1 and 0



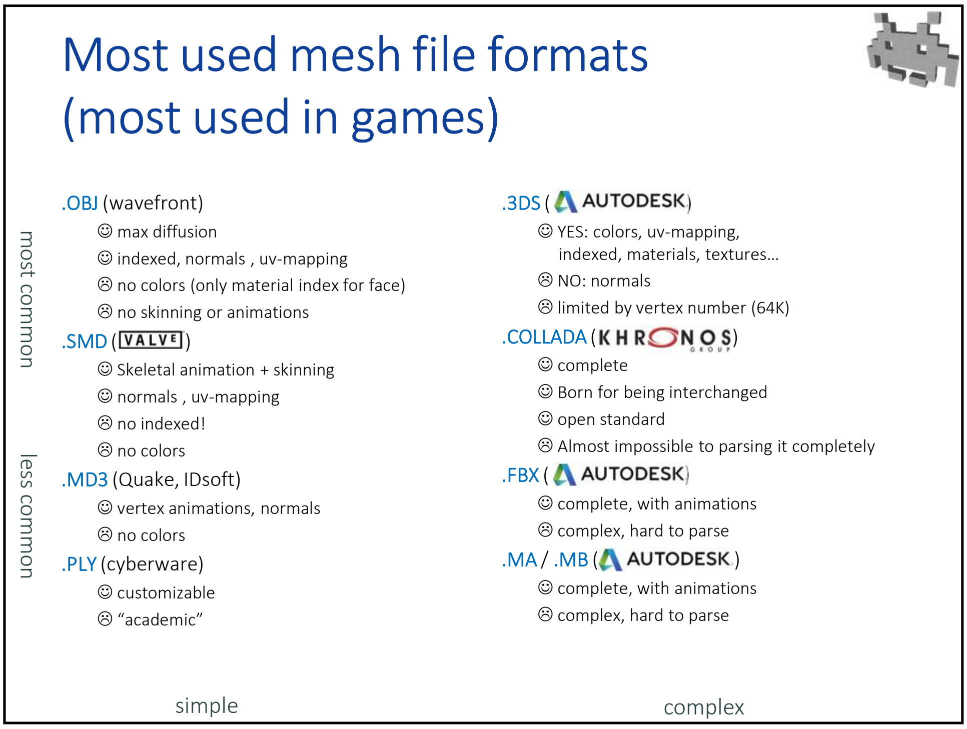
File formats for meshes

(a Babel tower!)

- 3DS - 3D Studio Max file format
- OBJ - Another file format for 3D objects
- MA, MB - Maya file formats
- 3DX - Rinoceros file format
- BLEND - Blender file format
- STL - Very used for 3D Printing
- FBX - Autodesk interchange file format
- X - Direct X object
- SMD - good for animations (by Valve)
- MD3 - quake 3 vertex animations
- DEM - Digital Elevation Models
- DXF - exchange format, Autodesk's AutoCAD)
- FIG - Used by REND386/AVRIL
- FLT - MulitGen Inc.'s OpenFlight format
- HDF - Hierarchical Data Format
- IGES - Initial Graphics Exchange Specification
- IV - Open Inventor File Format Info
- LWO, LWB & LWS - Lightwave 3D file formats
- MAZ - Used by Division's dVS/dVISE
- MGF - Materials and Geometry Format
- MSDL - Manchester Scene Description Language
- 3DML - by Flatland inc.
- C4D – Cinema 4D file format
- SLDPTR - SolidWork "part"
- WINGS - Wings3D object
- NFF - Used by Sense8's WorldToolKit
- SKP - Google sketch up
- KMZ - Google Earth model
- OFF - A general 3D mesh Object File Format
- OOGL - Object Oriented Graphics Library
- PLG - Used by REND386/AVRIL
- POV - “persistence of vision” ray-tracer
- QD3D - Apple's QuickDraw 3D Metafile format
- TDDD - for Imagine & Turbo Silver ray-tracers
- NFF & ENFF - (Extended) Neutral File Format
- VIZ - Used by Division's dVS/dVISE
- VRML, VRML97 - Virtual Reality Modeling Language (RIP)
- X3D – attempted successor of VRML
- PLY – introduced by Cyberware – typical of range-scanned data
- DICOM – by DICOM – typical of CAT-scan data
- Renderman – data for the homonymous renderer
- RWX - RenderWare Object
- Z3D - ZModeler File format

etc

47



Most used mesh file formats (most used in games)

most common

less common

.OBJ (wavefront)
- ☺ max diffusion
- ☺ indexed, normals , uv-mapping
- ☹ no colors (only material index for face)
- ☹ no skinning or animations

.SMD (VALVE)
- ☺ Skeletal animation + skinning
- ☺ normals , uv-mapping
- ☹ no indexed!
- ☹ no colors

.MD3 (Quake, IDsoft)
- ☺ vertex animations, normals
- ☹ no colors

.PLY (cyberware)
- ☺ customizable
- ☹ “academic”

.3DS (AUTODESK)
- ☺ YES: colors, uv-mapping, indexed, materials, textures…
- ☹ NO: normals
- ☹ limited by vertex number (64K)

.COLLADA (KHRONOS GROUP)
- ☺ complete
- ☺ Born for being interchanged
- ☺ open standard
- ☹ Almost impossible to parsing it completely

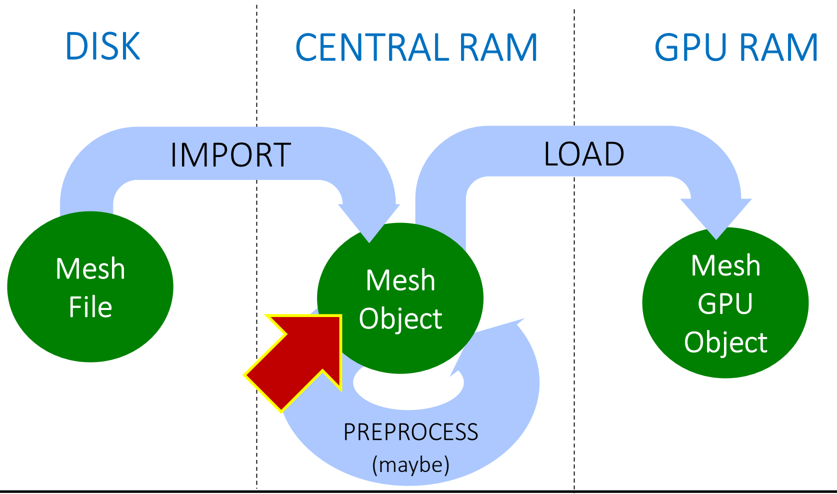
.FBX (AUTODESK)
- ☺ complete, with animations
- ☹ complex, hard to parse

.MA / .MB (AUTODESK)
- ☺ complete, with animations
- ☹ complex, hard to parse

simple complex

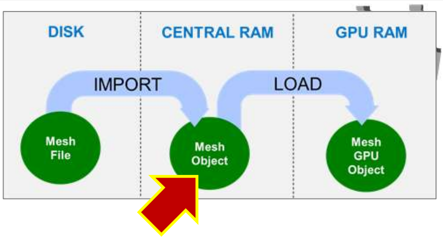
48

Life of a Mesh in a Game Engine



49

Mesh Object (in RAM)



- A (C++ / Javascript / etc) structure in main RAM
- Choices for a game engine:
  - which attribute to store?
  - storage formats… (floats, bytes, double…)
  - which preprocessing to offer (typically, at load time)

50

How to represent a mesh?
(which data structures)

- Indexed mode in C++ :

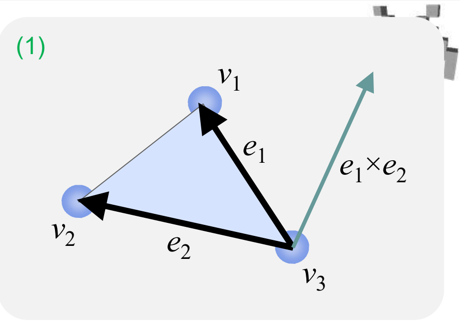
```
class Vertex {
  vec3 pos;
  rgb color;    /* attribute 1 */
  vec3 normal;  /* attribute 2 */
};

class Face{
   int vertexIndex[3];
};

class Mesh{
  vector<Vertex> verts;  /* geom + attr */
  vector<Face> faces;    /* connectivity */
};
```

51

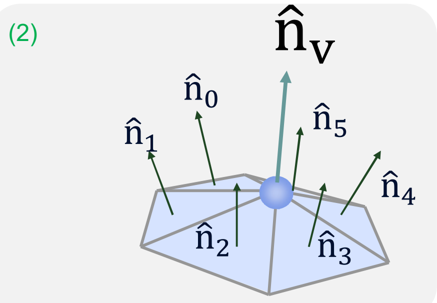
Computing normals from geometry

(1) compunte
normals of faces

(2) compute
normals of vertices



$$\hat{n}_v = \frac{\hat{n}_0 + \cdots + \hat{n}_k}{\|\hat{n}_0 + \cdots + \hat{n}_k\|}$$



54

Mesh processing:
(or, more in general, Geometry Processing)

- The algorithm above
  (for the computation of per vertex normal)
  is a tiny example of processing done over a mesh
- Mesh processing: the discipline of creating, transforming, computing meshes
  - inputs and/or outputs are meshes
- Part of, geometry processing:
  - when the input and output are other data structure for 3D models
  - See CG course for a very brief overview

55

Mesh processing:
typical tasks for the game industry

- Poly reduction / Retopology / Simplification
  - e.g. LOD construction
  - e.g. transition from (initial) hi-res to (final) low-poly
- Light baking LATER
  - Light precomputation
  - e.g.: Ambient Occlusion
- U-V map construction LATER
  - parametrization / unwrapping
- Texturing LATER
  - creation of different types of textures
- Rigging / Skinning / Animation LATER
  - to animate

58

Useful general tools: attribute transfer

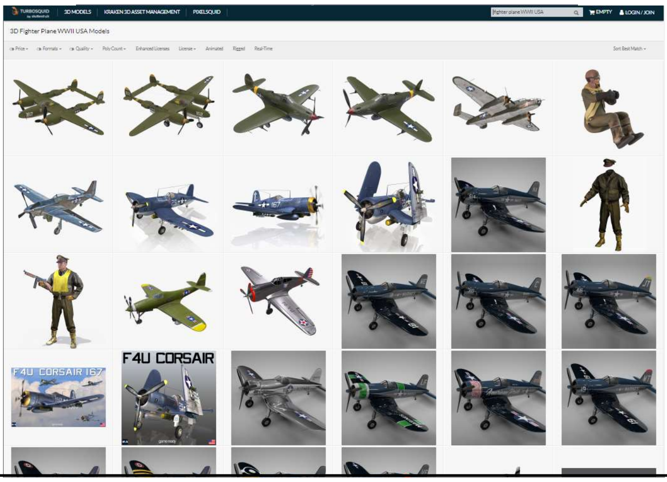
(any, see the list!)

- Given
  - a source mesh M_0 with attribute A
  - a target mesh M_1 similar (but not identical) to M_0 lacking that attribute
- Define attribute A in the vertices of M_1
  - Copying the attributes from M_0
- Result: "retargeting" of…
  - Animations, UV-mapping, textures, etc
- Results aren't always perfect, but can be useful as a starting point

59

3D models: suorces

- Like any asset, often just bought / off-sourced

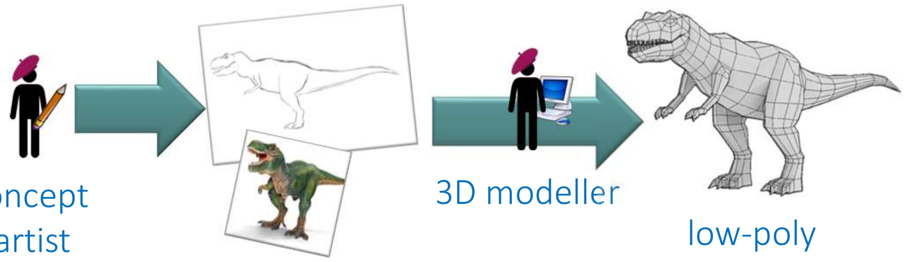


fighter plane WWII USA

3D Fighter Plane WWII USA Models

62

3D models: authoring pipeline



concept artist

2D concepts / sketches

3D modeller

low-poly mesh

63

Mesh: authoring

- Task of the 3D modeller
  - A type of digital artist
- Popular 3D modeling approaches:
  - Manual low-poly modelling
    - e.g. with wings3D
  - Subdivision surfaces
    - e.g. with blender
  - Digital sculpting
    - e.g. with Z-brush

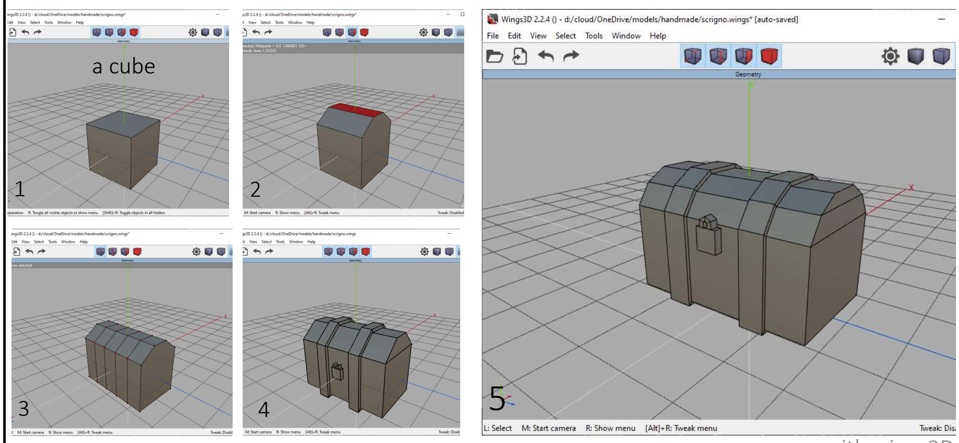
64

Mesh authoring (aka 3D modelling): a few applications

- 3D Studio Max (autodesk) , Maya (autodesk) , Cinema4D (maxon) Lightweight 3D (NewTek), Modo (The Foundry) , …
  - all-purpose, powerful, complete
- Blender
  - the same, plus open-source and freeware (compare: Gimp VS. Adobe Photoshop for 2D images)
- MeshLab
  - open-source, big collection of geometry processing algorithms …
- AutoCAD (autodesk), SolidWorks (SolidThinking)
  - for CAD
- ZBrush (pixologic) (+ Sculptris), Mudbox (autodesk)
  - Sculpting (inclusing texturing)
- Wings3D
  - low-poly modelling (& subdivision surfaces) open-source, small, specialized
- [Rhinoceros]
  - parametric surfaces (NURBS)
- FragMotion
  - small, specialized on animated meshes
- + a many more for specific contexts
  - editing of human models, of architectural interiors, environments, or specific editors for game-engines, etc...

65

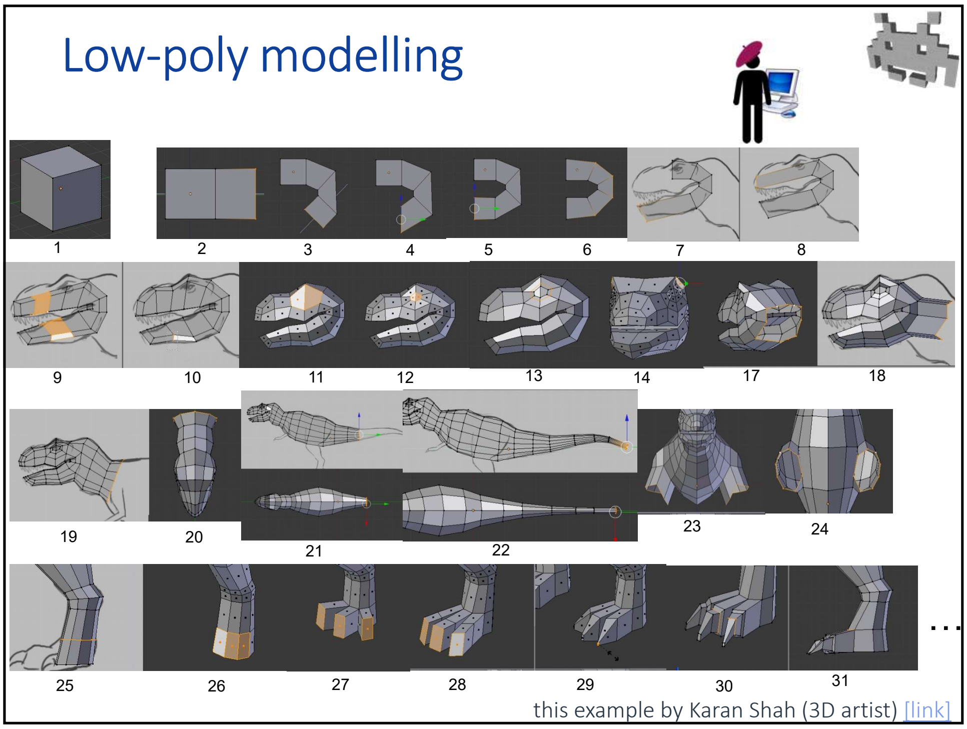
Low-poly modelling (demo)


a cube
1
2
3
4
5
Wings3D 2.2.4 () - d:/cloud/OneDrive/models/handmade/scrigno.wings* [auto-saved]
File Edit View Select Tools Window Help
Geometry
L: Select M: Start camera R: Show menu [Alt]+R: Tweak menu
Tweak: Dis

with wings3D

Note: during creation, the meshes can be polygonal instead of triangle based, but is simple to decompose any polygon into triangles
E.g. this can be done by the game engine as a simple preprocessing.

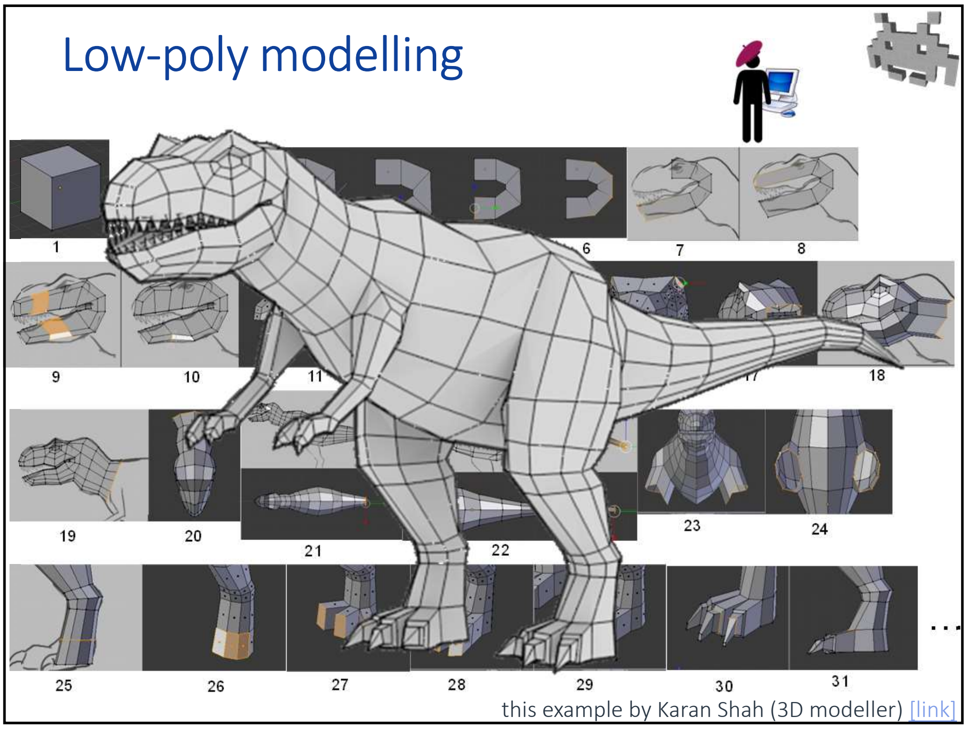
66



Low-poly modelling

this example by Karan Shah (3D artist) [link]

68



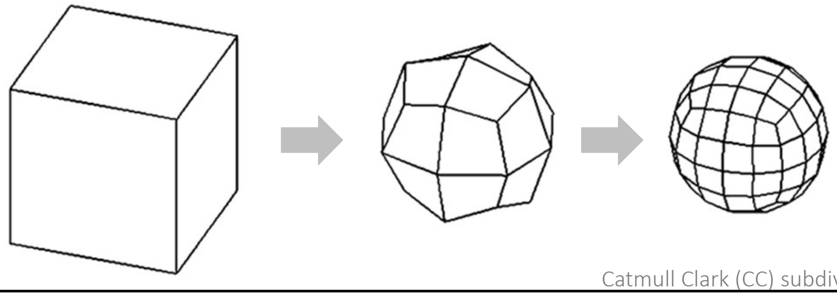
Low-poly modelling

this example by Karan Shah (3D modeller) [link]

69

3D mesh authoring techniques: subdivision surfaces

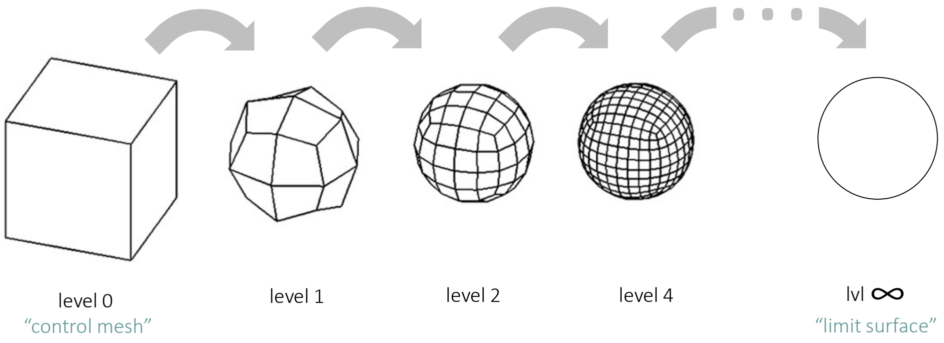
- **Subdivision step**:
  an algorithm that operates on a mesh
  and obtains a higher resolution, smoother mesh
- Can be iterated



Catmull Clark (CC) subdivision

70

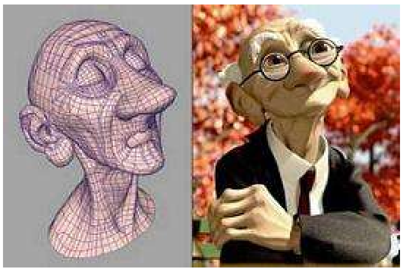
Example: with Catmull-Clark scheme



71

3D mesh authoring techniques: subdivision surfaces

- Many subdivision algorithms (schemas) exists
  - each with its own properties
- Produces clean, regular meshes
- Excellent for smooth, curved, organic looking objects



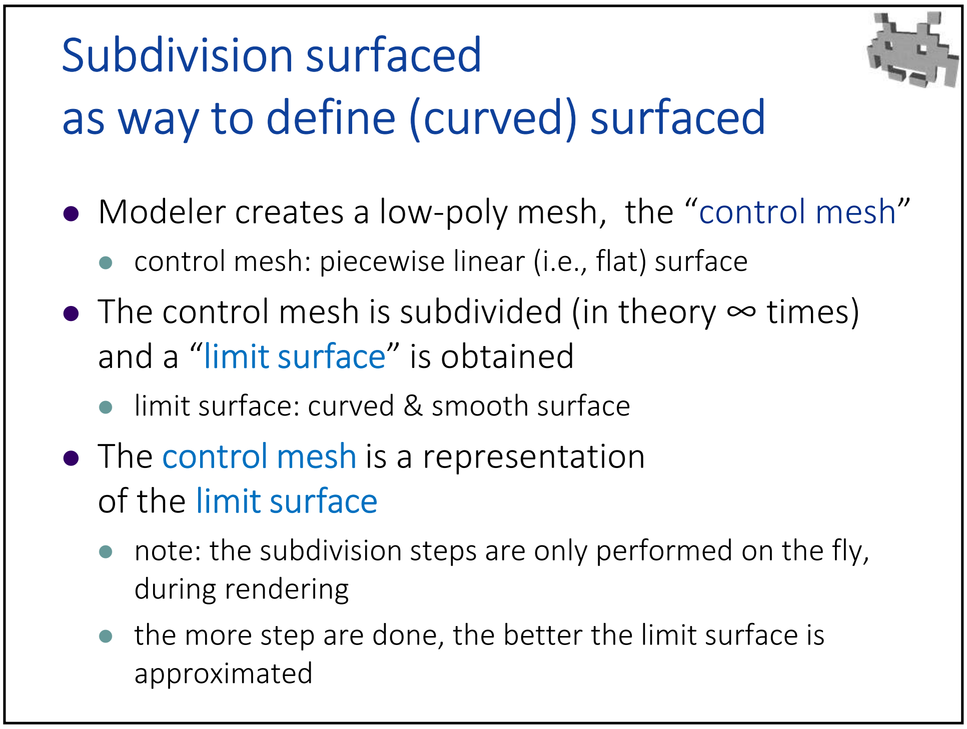
famously pioneered by movie industry (not games):

72

Subdivision surfaces as a tool…

- …to encode smooth surfaces
  - Idea: we encode the control mesh to represent the limit surface
  - use in games: rendering (now, rare – but popular around 2015)
    1. keep control mesh in GPU ram
    2. let 1-3 subdivision steps happen during rendering
- …to author 3D meshes
  - idea: alternate (low-poly) editing and subdivisions steps
  - at first steps: edit global shape
  - at last steps: edit minute details
  - use in games: during asset creation, by artists

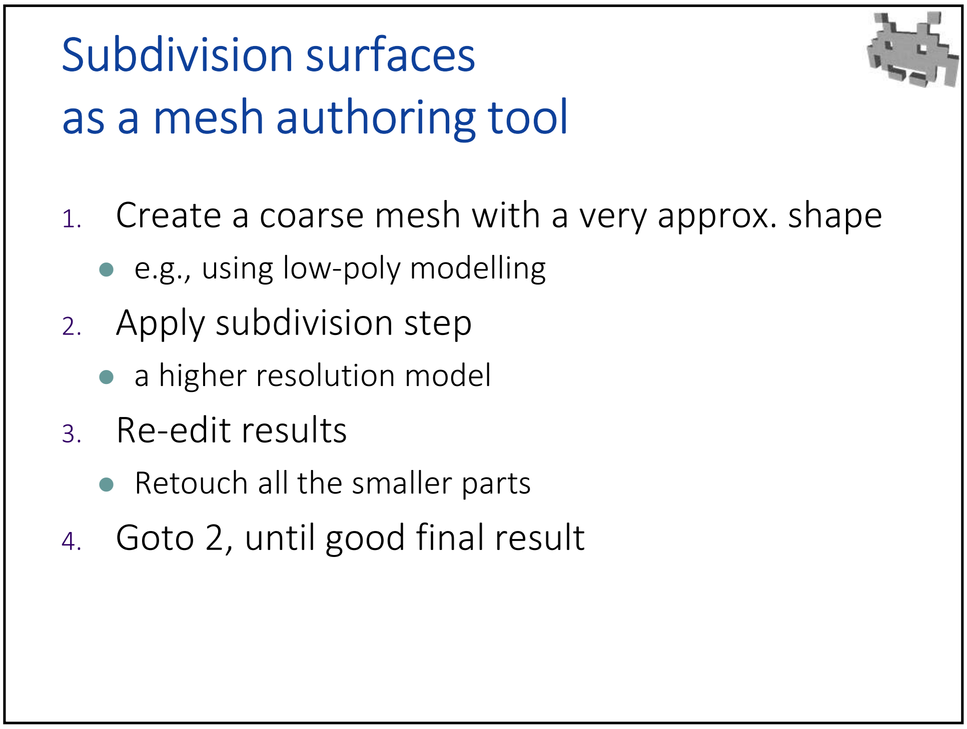
73



Subdivision surfaced as way to define (curved) surfaced

- Modeler creates a low-poly mesh, the “control mesh”
  - control mesh: piecewise linear (i.e., flat) surface
- The control mesh is subdivided (in theory ∞ times) and a “limit surface” is obtained
  - limit surface: curved & smooth surface
- The control mesh is a representation of the limit surface
  - note: the subdivision steps are only performed on the fly, during rendering
  - the more step are done, the better the limit surface is approximated

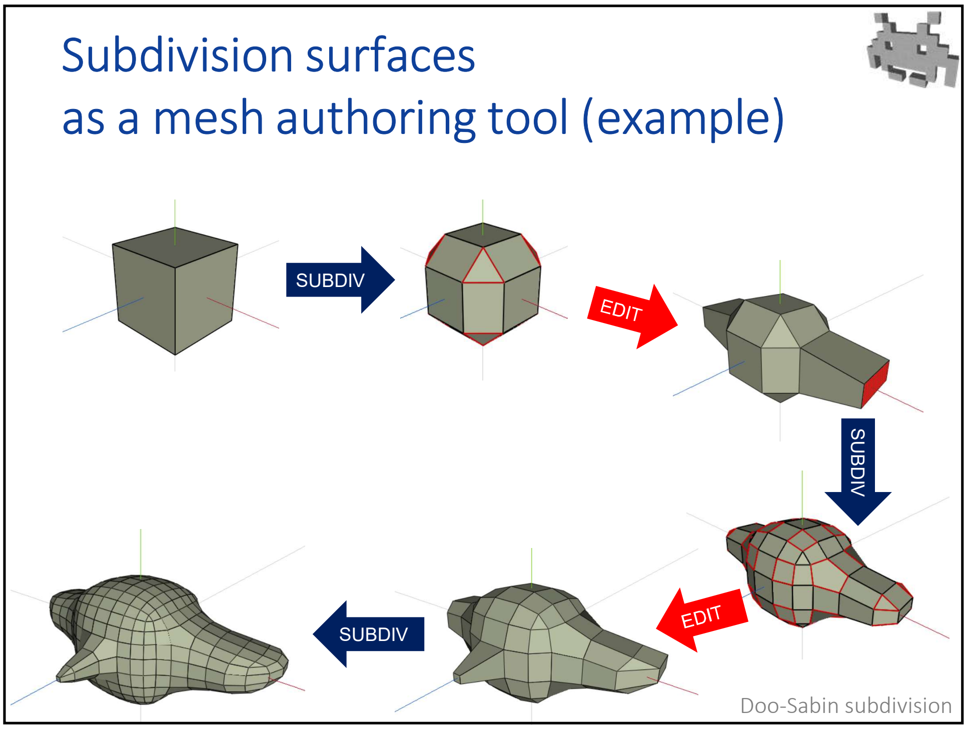
74



Subdivision surfaces as a mesh authoring tool

1. Create a coarse mesh with a very approx. shape
   - e.g., using low-poly modelling
2. Apply subdivision step
   - a higher resolution model
3. Re-edit results
   - Retouch all the smaller parts
4. Goto 2, until good final result

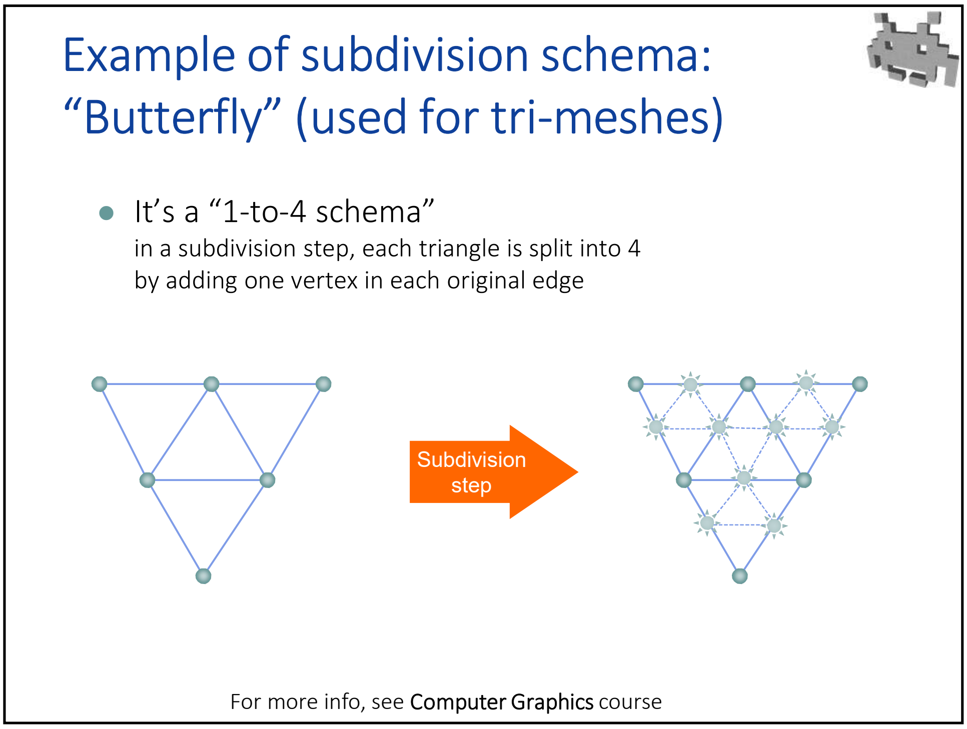
75



Subdivision surfaces as a mesh authoring tool (example)

Doo-Sabin subdivision

77



Example of subdivision schema: “Butterfly” (used for tri-meshes)

- It’s a “1-to-4 schema”
  in a subdivision step, each triangle is split into 4
  by adding one vertex in each original edge

For more info, see **Computer Graphics** course

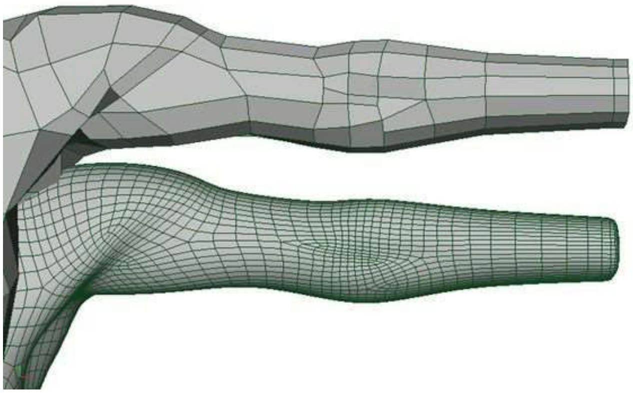
81

Subdivision surfaces in general

- A step typically increases resolution by a factor x4
- The geometry of the subidvided mesh (3D points) is computed according to a formula of the pos of their neighbors.
  - In some schemas (called interpolative), the old vertices are kept at the same positions
  - In other schemas (called approximative), old vertices are kept but moved into a new position
  - In other schemas (called dual) older vertices aren't kept
- Most created vertices are *regular*

82

An example with Catmull Clark



84

Some existing subdivision schemas

- Doo-Sabin
  - operates on any polygonal mesh
  - produces polygonal meshes
- Loop
  - 1-to-4 scheme for triangle meshes (only)
- Butterfly
  - 1-to-4 scheme for triangle meshes (only)
- Catmull-Clark
  - operates on any polygonal mesh
  - produces quad-meshes
  - traditionally, movie-industry favorite
  - a recent trend in games: use during mesh rendering

88

3D Mesh authoring: approaches

- Popular 3D modeling approaches:
  - Direct low-poly modelling
    - e.g. with wings3D
  - Subdivision surfaces
    - e.g. with blender
  - Digital sculpting
    - e.g. with Z-brush, (or Sculptris Alpha)

90

Digital Sculpting

mouse
(or stylus)

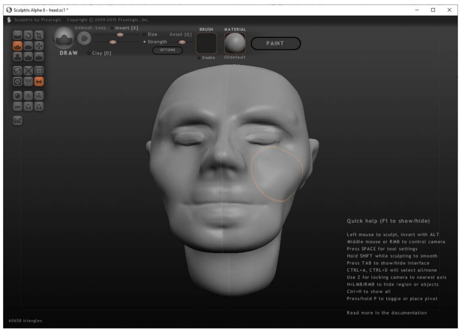
=

chisel

91

Digital Sculpting

- demo



Sculptris Alpha 6 - head.sc1 *

Sculptris by Pixologic   Copyright © 2009-2015 Pixologic, Inc.

Quick help (F1 to show/hide)

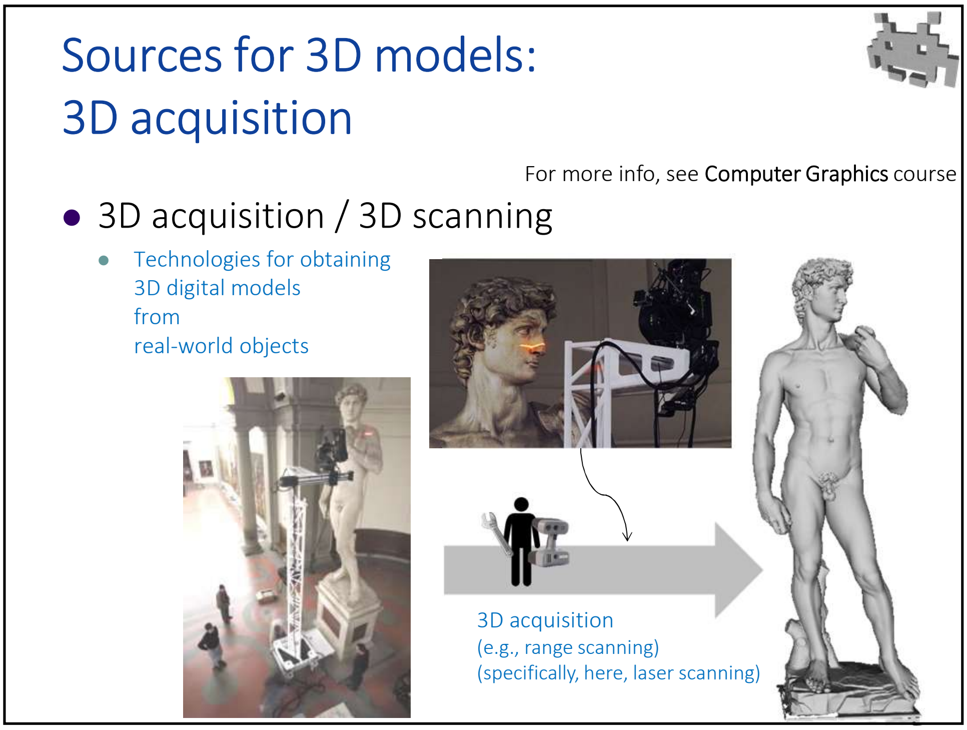
Left mouse to sculpt, invert with ALT
Middle mouse or RMB to control camera
Press SPACE for tool settings
Hold SHIFT while sculpting to smooth
Press TAB to show/hide interface
CTRL+A, CTRL+D will select all/none
Use Z for locking camera to nearest axis
H+LMB/RMB to hide region or objects
Ctrl+H to show all
Press/hold P to toggle or place pivot

Read more in the documentation

60658 triangles

with wings3D

92

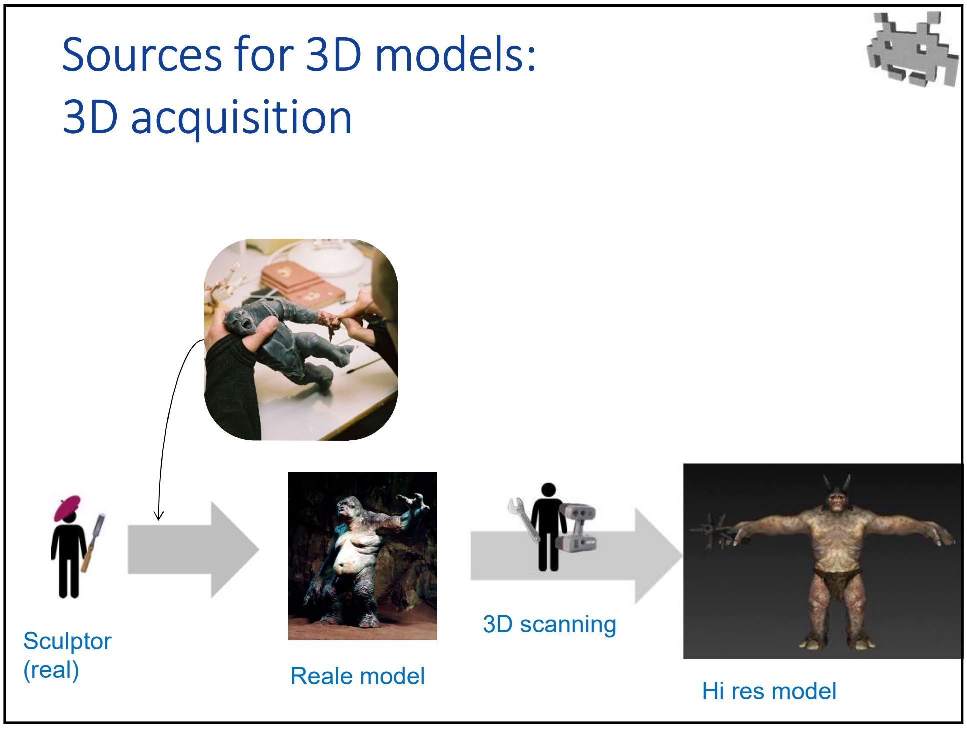


Sources for 3D models: 3D acquisition

For more info, see **Computer Graphics** course

- 3D acquisition / 3D scanning
  - Technologies for obtaining 3D digital models from real-world objects

93

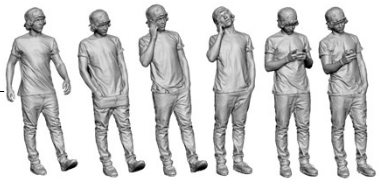


Sources for 3D models: 3D acquisition

94

Sources for 3D models: 3D acquisition

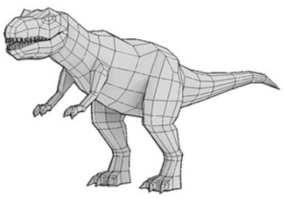
- 3D scanning
  - A.k.a. *automatic 3D model acquisition*
  - Lot of different technologies
    - Laser scanners
    - Time of flight
    - Structured light (kinect)
    - …
  - Different characteristics
    - Results quality
      - Noise / resolution
    - Automatism
    - Invasiveness
      - Markers? Powder?
    - Real time? (kinect)
    - Price
    - Max object dimension
      - (full body scanner?)



95

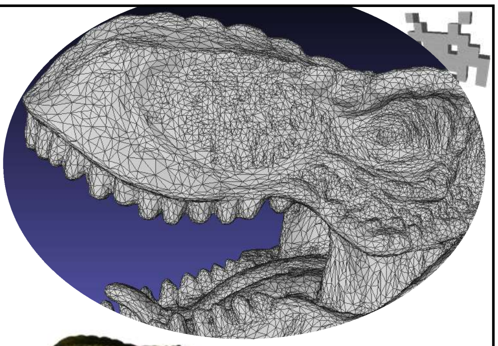
3D models sources: comparison

PERFECT for games!
(much easier to: animate, re-edit, uvmap, …)



manually edited
low-poly mesh
(2K triangles)

VS



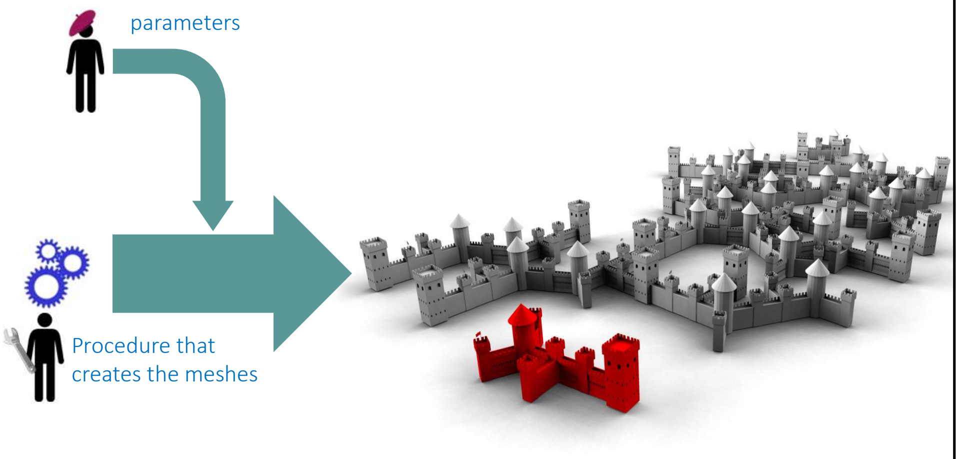
Dino, scanned by artec3d

scanned & cleaned
hi res mes
(30K triangles)

(sculpted meshes are similar)

96

Sources for 3D models: procedural modelling



97

Procedural modelling – see also…

EPC2021

EVERYTHING PROCEDURAL

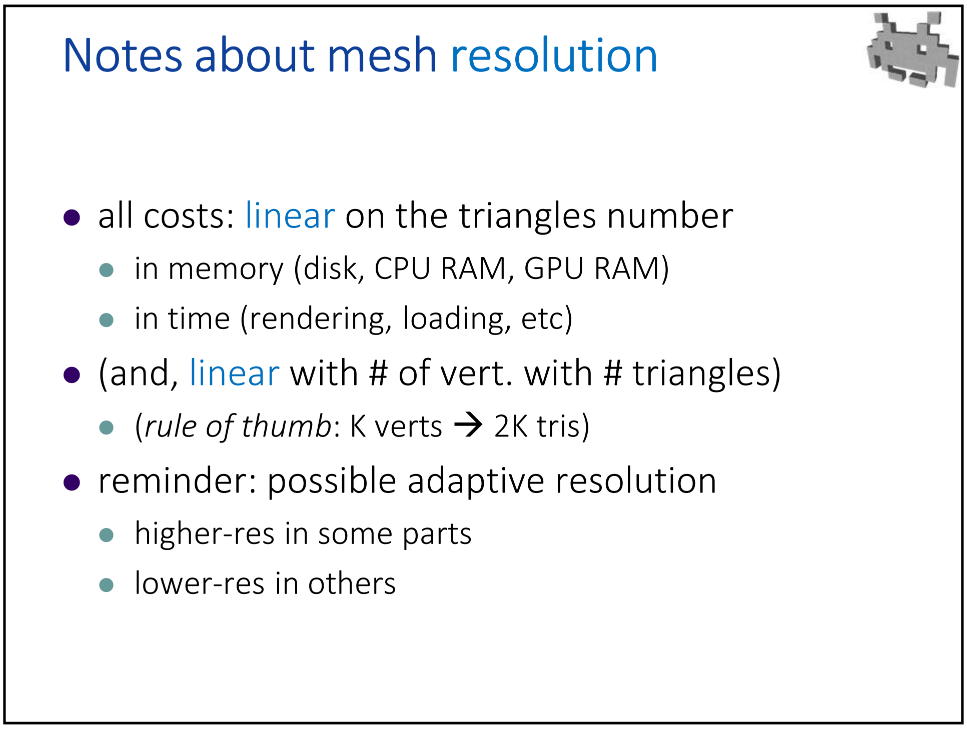
CONFERENCE ON PROCEDURAL CONTENT GENERATION FOR GAMES

30-04-2021

http://everythingprocedural.com/

this week
Game-of-the-Week

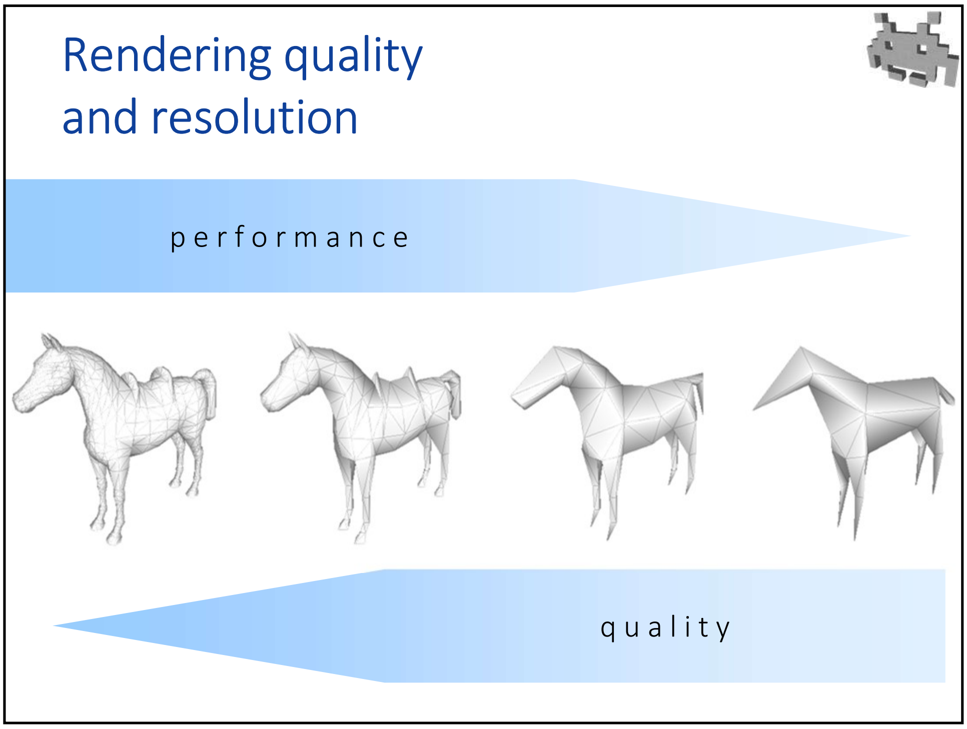
98



Notes about mesh resolution

- all costs: linear on the triangles number
  - in memory (disk, CPU RAM, GPU RAM)
  - in time (rendering, loading, etc)
- (and, linear with # of vert. with # triangles)
  - (*rule of thumb*: K verts → 2K tris)
- reminder: possible adaptive resolution
  - higher-res in some parts
  - lower-res in others

99



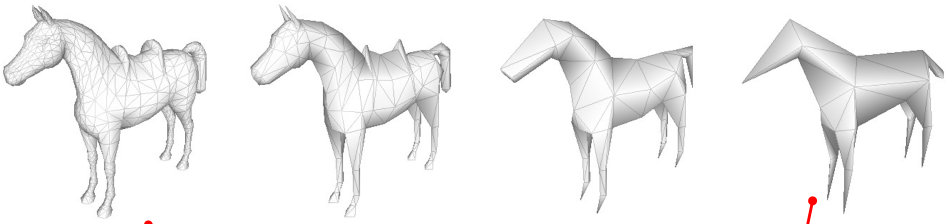
Rendering quality and resolution

performance

quality

100

LoD pyramids
(Level Of Detail)



LoD 1
2K faces

LoD 2
400 face

LoD 3
160 faces

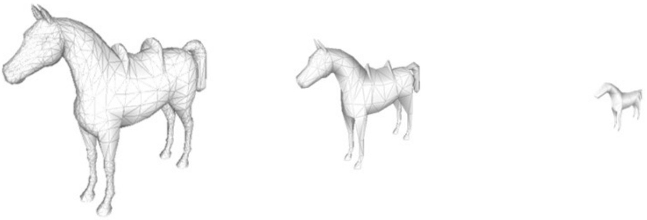
LoD 4
60 faces

use when seen up close

use when seen from afar

101

LoD pyramids
(Level Of Detail)



<5 m

5m ..10 m

10 .. 30 m

>30 m

102

LoD pyramids (Level Of Detail)

- Goal:
  - decrease the geometry budget (total number of vertices)
  - ideal: size of triangles in screen space (in pixel): constant
    - (if importance / complexity is the same)
- Task: determining the level to use (dynamically, at runtime)
  - depending on observer distance
  - and/or, depending on rendering workload
    - e.g.: rendering is lagging ⇒ decrease LoD
  - this is task of the rendering engine)
- Task: LOD creation or "LOD-ding" (during asset creation)
  - starting from LOD-0 (higher-res)
  - manual, or automatic *(see later on),* or assisted (mixed)
    - often manual
  - note: sometimes "LoD 0" is used only in special cases
    - e.g. during a cut-scenes

computed from
scene graph
(how?)

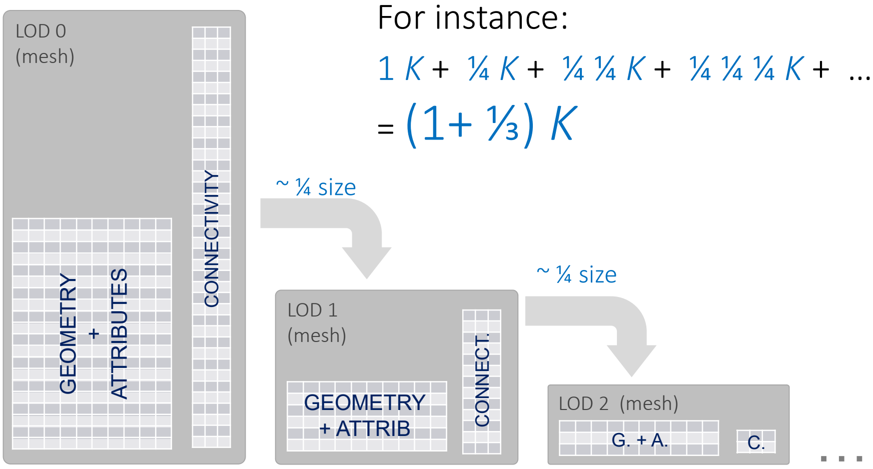
103

LoD pyramids (Level Of Detail)

Total memory usage: limited
For instance:

$$1\,K + \tfrac{1}{4}\,K + \tfrac{1}{4}\tfrac{1}{4}\,K + \tfrac{1}{4}\tfrac{1}{4}\tfrac{1}{4}\,K + \ldots$$

$$= \left(1+\tfrac{1}{3}\right) K$$



104