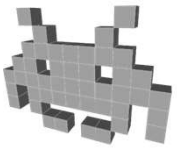
3D video games

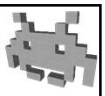
3D Meshes in Games



Marco Tarini

1

Course Plan

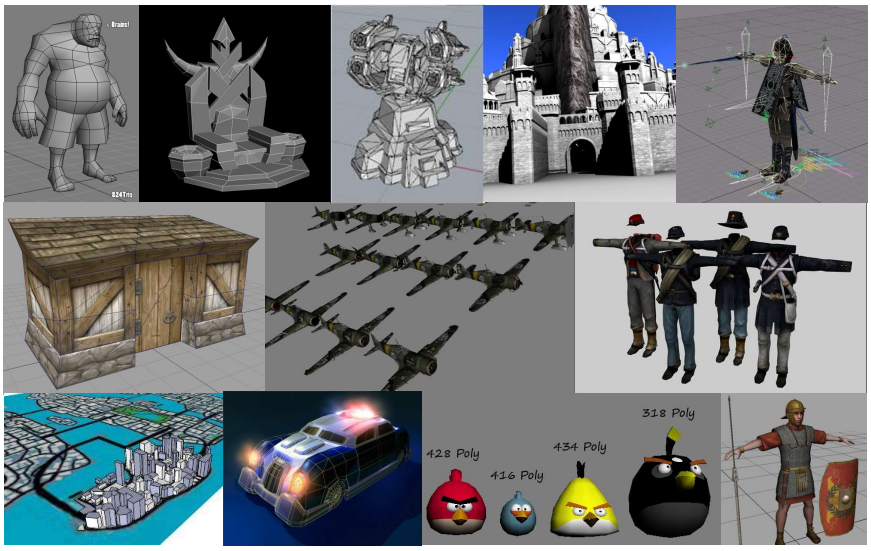


lec. 1: **Introduction**
lec. 2: **Mathematics** for 3D Games
lec. 3: **Scene Graph**
lec. 4: Game **3D Physics** +
lec. 5: Game **Particle Systems**
lec. 6: Game **3D Meshes**
lec. 7: Game **Materials**
lec. 8: Game **Textures**
lec. 9: Game **3D Animations**
lec. 10: **3D Audio** for 3D Games
lec.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

★ appearance

2

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



3

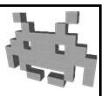
Solomons’s key (1986, Temco) on Z80

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

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

6

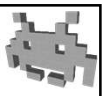
Triangle Meshes:
the visual appearance of 3D objects



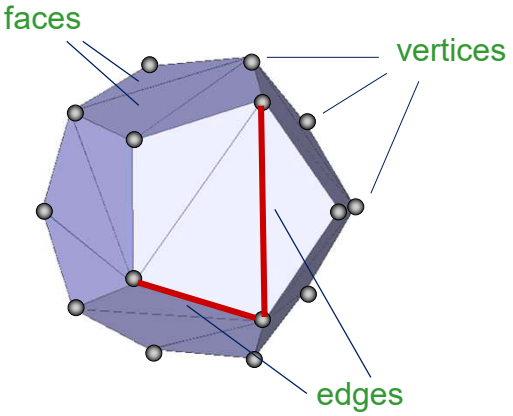
- Data structure for modelling 3D objects
  - GPU friendly
  - Resolution = number of faces
  - Resolution is (potentially) Adaptive (that is, more faces where needed)
- Used to model the visual appearance of 3D physical objects in the game
  - at least, the ones which can be represented by their surface
  - most solid objects (rigid or not)
- Mathematically: a piecewise linear approximation of the surface
  - a set of 3D samples, “vertices” connected by a set of triangular “faces” connected side to side by “edges”

7

Triangle Mesh
(or simplicial mesh)

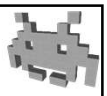


- A set of adjacent triangles



8

Mesh: data structure

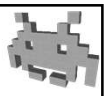


A mesh consists of

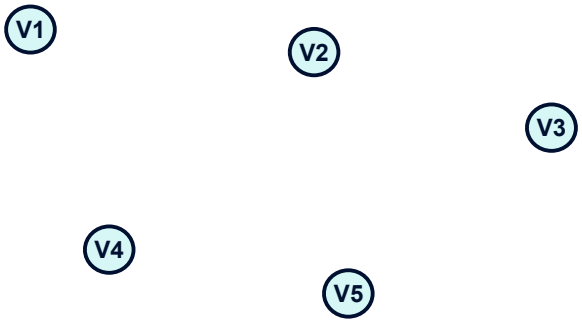
- geometry
  - The set of (x,y,z) positions of the vertices
  - It's a sampling of the surface
- connectivity (or topology)
  - The set of faces connecting the vertices
    - In a triangle mesh: faces are triangles (this is what the GPU is designed for!)
    - In a quad mesh: faces are quadrilateral
    - Quad dominant mesh: *most* faces are quadrilateral
    - Polygonal mesh: faces are polygons (general case)
- attributes
  - Data stored at vertices, such as: color, material, normal, …

9

Mesh: geometry

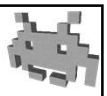


- Set of vertices
  - A position vector (x,y,z) for every vertex
  - Coordinates, by definition, are given in Local space!

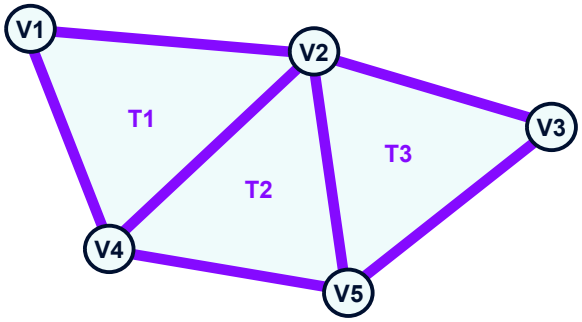


10

Mesh: connectivity (or topology)

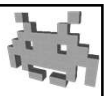


- Faces: triangles connecting vertices
  - More in general, polygons,
  - connecting triplet of *vertices*
  - just as, in a graph, *nodes* are connected by *edges*

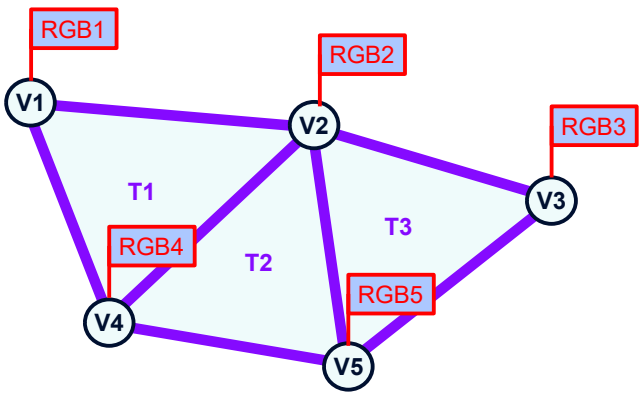


11

Mesh: attributes

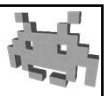


- Any quantity that varies over the surface
  - sampled at vertices, and interpolated inside triangles



12

Mesh as a data strucuture: indexed meshes



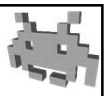
we can consider positions as attributes too

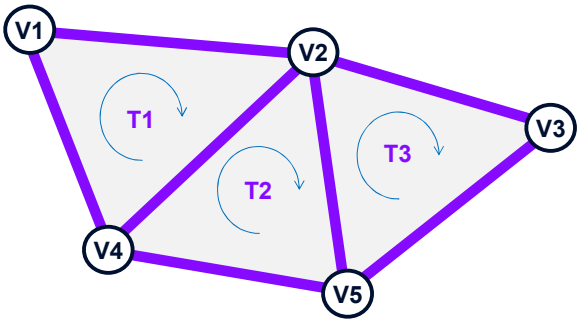
- array of vertices
  - Each vertex stored as
    - *x,y,z* position (aka the "geometry" of the mesh)
    - attributes: (all vertices, the same ones)
      any data saved on the surface: e.g. color
- array of triangles
  - the "connectivity»
  - Each triangle stored as
    - triplet of indices (referring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

These two arrays can be seen as tables (buffers)

14

An indexed mesh in VRAM : 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

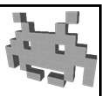
GEOMETRY + ATTRIBUTES

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

CONNECTIVITY

15

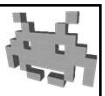
Mesh resolution

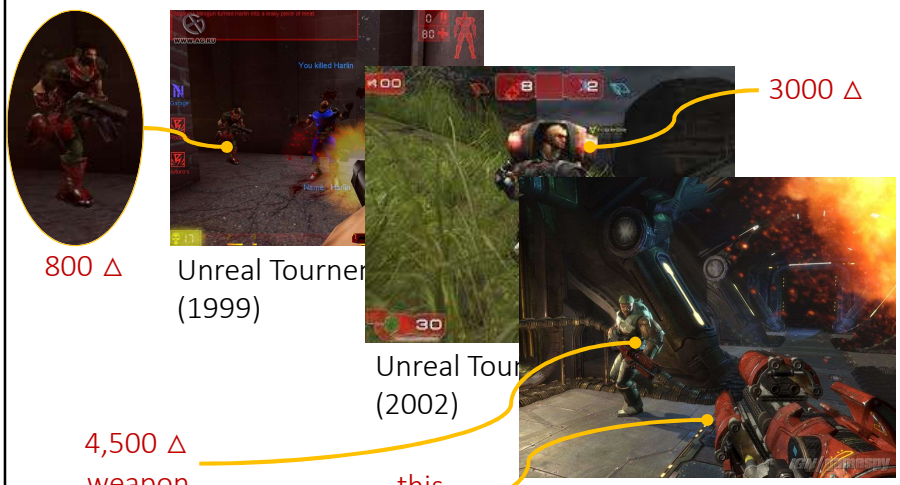


- Defined as the number of faces
  - or vertices, equivalent because typically #F ≈ 2 · #V)
- Rendering time is linear with resolution
  - therefore, in games, resolution is kept small
  - aka. «low-poly» models
- Resolution can be adaptive:
  - denser vertices & smaller faces in certain parts
  - sparser 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

16

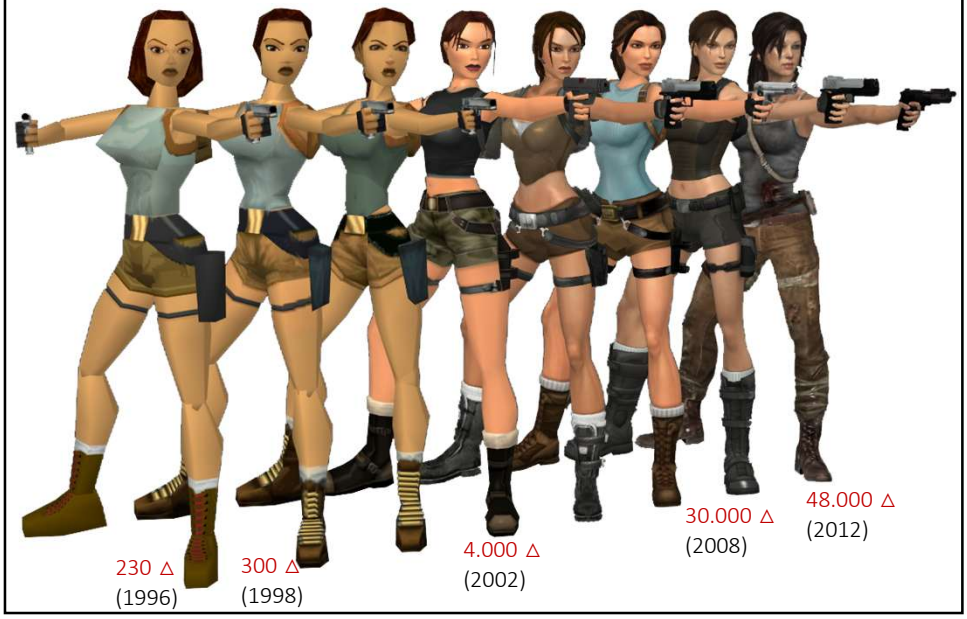
Resolution increases over time





19

Resolution increases over time



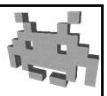
21

Mesh attributes: in general (this applies to any attribute)

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

22

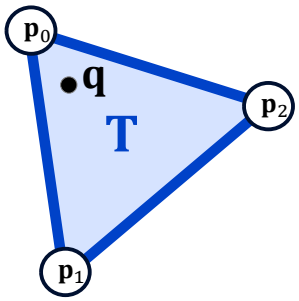
Interpolation of vertex attributes inside mesh triangles 1/2



- A triangle $\mathbf{T}$ with vertices $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$
- For every point $\mathbf{q}$ in $\mathbf{T}$ there are (unique!) k_0, k_1, k_2 with $k_0 + k_1 + k_2 = 1$ such that

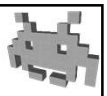
$$\mathbf{q} = k_0\,\mathbf{p}_0 + k_1\,\mathbf{p}_1 + k_2\,\mathbf{p}_2$$

- k_0, k_1, k_2 are called the barycentric coordinates of $\mathbf{q}$ in $\mathbf{T}$



23

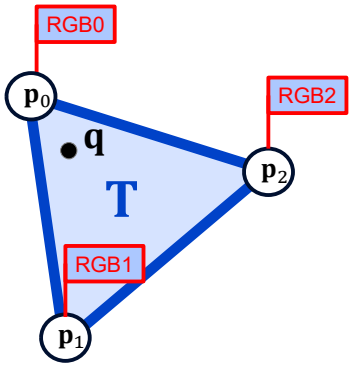
Interpolation of vertex attributes inside mesh triangles 1/2



- If we three attributes to the three vertices…
- a point $\mathbf{q}$ in $\mathbf{T}$ with baricentric coodinates k_0, k_1, k_2 is implicitly given the attribute value

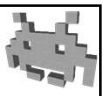
$$k_0\,\boxed{\text{RGB0}} + k_1\,\boxed{\text{RGB1}} + k_2\,\boxed{\text{RGB2}}$$

per vertice



24

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



in local space!

- Position
  (aka the "**geometry**" of the mesh)
- Normal

see lecture on textures (later)

- Texture Coordinates
  (aka the "**UV-mapping**" of the mesh)

see lecture on normal maps (later)

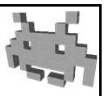
- Tangent Direction

see lecture on animations (later)

- Bone links
  (aka the "**skinning**" of the mesh)
- Color

25

Mesh as buffers



- Position (P)
- Normal (N)
- Color (r,g,b,a)
- Texture Coordinate (U,V)
- Tangent Direction (T,B)
- Bone links (…)

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

Ints (with some # of bits)

CONNECTIVITY

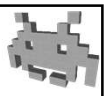
vert	Px	Py	Pz	Nx	Ny	Nz	r	g	b	a	U	V	Tx	Ty	Tz	Bx	By	Bz
V0																		
V1																		
V2																		
V3																		
V4																		

Floating points (with a given precision)

GEOMETRY + ATTRIBUTES

27

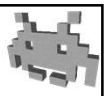
Mesh attributes: colors



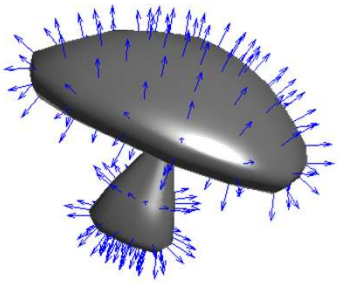
- In games, colors on 3D models are usually determined by textures (not by mesh colors)
  - reason: more detailed signals can be stored
- Per vertex colors can be used…
  - To cheaply add variations models
    - Red guards, blue guards SEE MATERIALS LATER
  - To bake lighting
    - e.g. baked per-vertex 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
  - …and more

28

Mesh attributes: normals

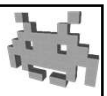


- A versor
- Representing the surface orientation
- Main use: lighting computation
- Can be computed automatically from geometry...
- But it is a part of the mesh assets:
  - the artist is in control of which edges are *soft* and which are *hard*



29

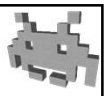
Mesh attributes: normals

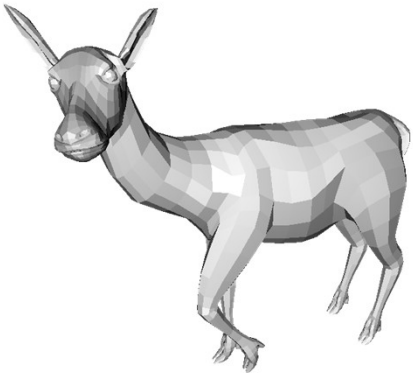


- Technically, mesh faces are *flat*
  - the normal is constant over a face
  - the normal is discontinuous across faces (each mesh edge is “sharp”)
- Usually, that’s not the surface we intend to represent
  - The flatness is just an artifact (a defect) of the mesh discretization
- By using a *continuously varying* normal (the per-vertex normal interpolated inside faces), the rendered images gives the illusion of a smooth, curved surface
  - which is (usually) what we want to represent
- But if we want, can we still represent “hard” (sharp) edges
  - With vertex seams: see below

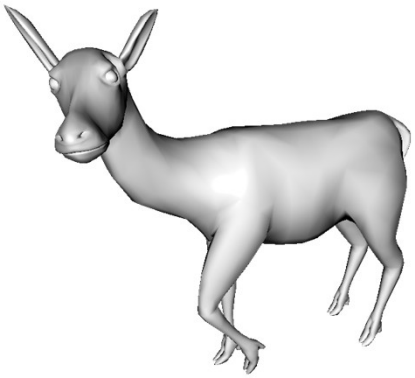
30

Mesh attributes: normals





if «real» normals where used («flat shading»)

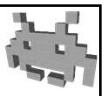


Using interpolated per vertex normals (smooth shading)

Note: normals are made visible to our eyes due to lighting (computation of how light reacts with the surface)

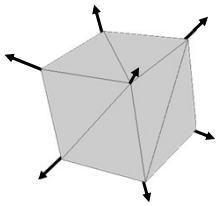
31

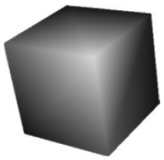
Hard edges (aka sharp edges) (aka “creases”)



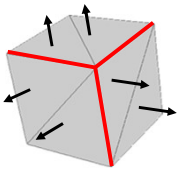
- Edges where the normal is not continuous .

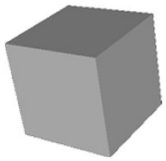
Soft edges:





Red edges are hard



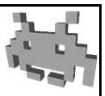


- How to encode (C_0) a discontinuity in any attributes?

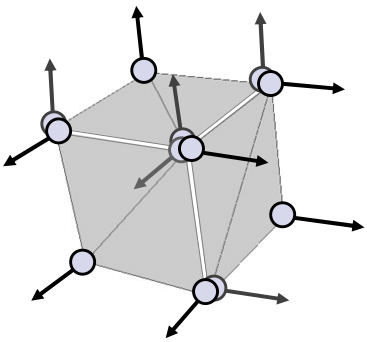
32

answer:

Vertex seams



- Vertex seam = two coinciding vertices. in *xyz*
  - different attributes assigned to each copy



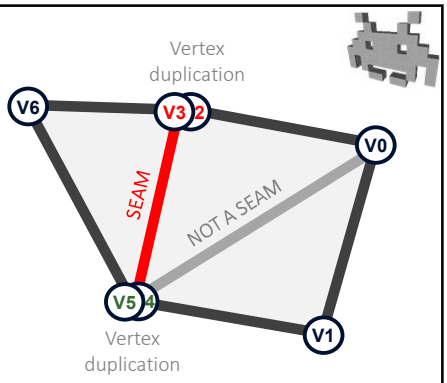
33

Vertex seams

- A way to encode any attribute discontinuity
- Price to be paid: a little 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	same	same	same	n_x3	n_y3	n_z3
V4	p_x3	p_y3	p_z3	n_x4	n_y4	n_z4
V5	same	same	same	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

34

Rendering of a Mesh in a nutshell

Load…

- get required data ready on **GPU RAM**
  - Geometry + Attributes buffer(s)
  - Connectivity buffer
  - Textures
  - Shaders
  - Parameters / Settings

THE MESH

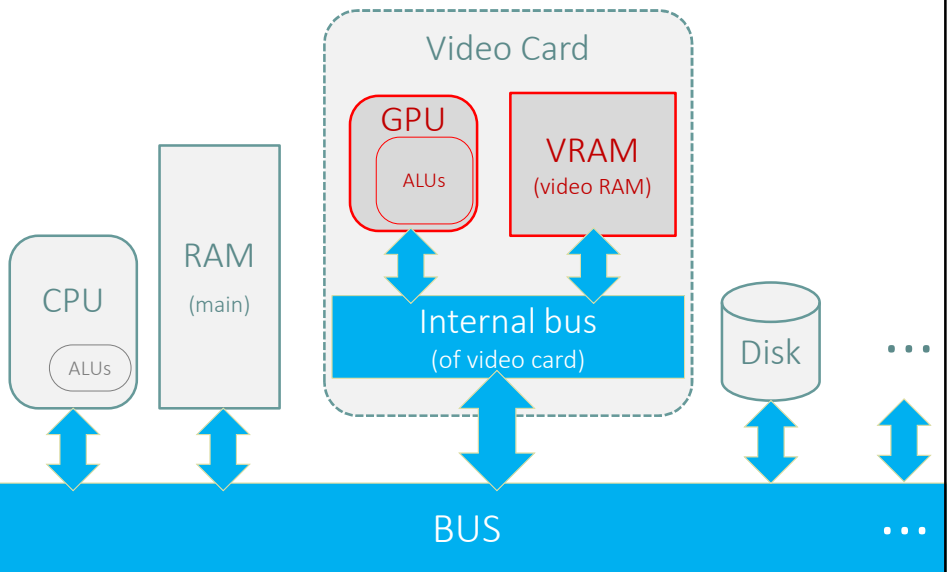
THE "MATERIAL"

…and Fire!

- send the "Draw-call" to the GPU
- using an API

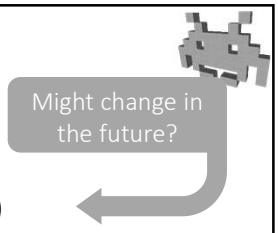
35

Simplified architecture of PC with Video Card



36

Rendering of a Mesh in a nutshell



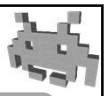
- The algorithm to render a mesh (in games) is based on rasterization
  - It is outside the scope of this course. See CG course.
  - In brief, three phases in cascade:
    each vertex is projected on screen ("transform"),
    (find where the vertex will be seen on the screen)
    then each triangle is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

37

Rendering of a Mesh in a nutshell

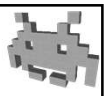


Exception: semi-transparent "see through" objects

- 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, many things are hard-wired
  - The data structures: indexed meshes (rarely: a triangle soup instead)
  - (Note: only triangle-shaped faces can be rendered – not quads/etc)
  - The interpolation of attributes inside faces
- 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

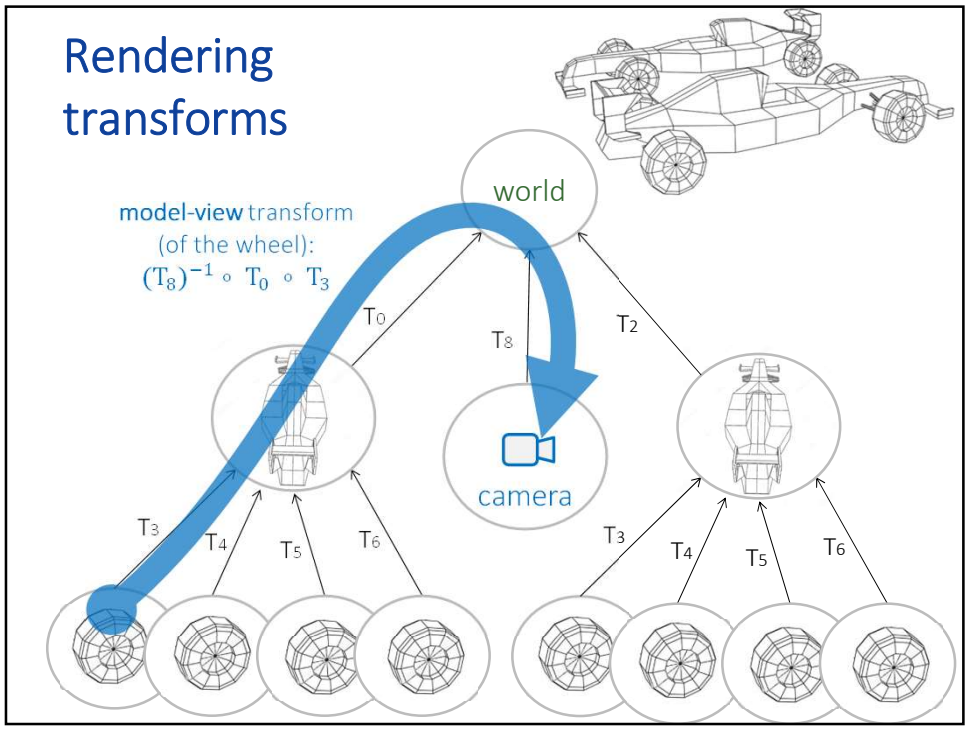
38

Rendering & Scene graph

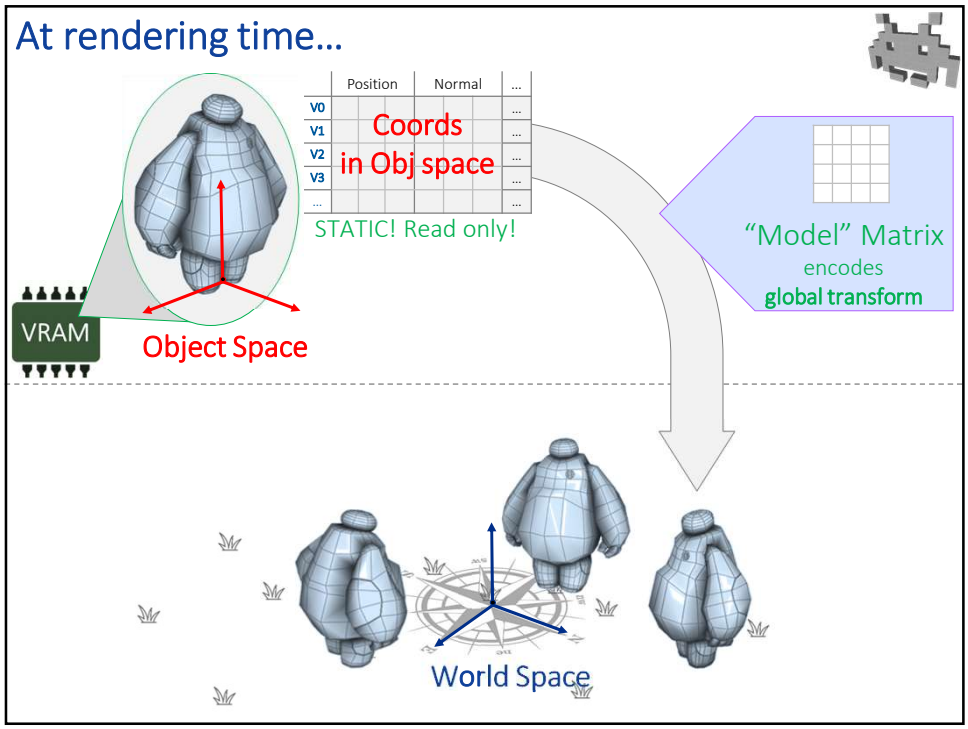


- Rendering APIs encode transforms as a 4x4 matrix
  - reason: it is a more flexible, can also express perspective transforms
- To render an object:
  - Combine its Transforms from Object-space to Camera-space ("model-view transform" – in CG terminology)
  - Convert it into a 4x4 matrix
  - Use it during 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)

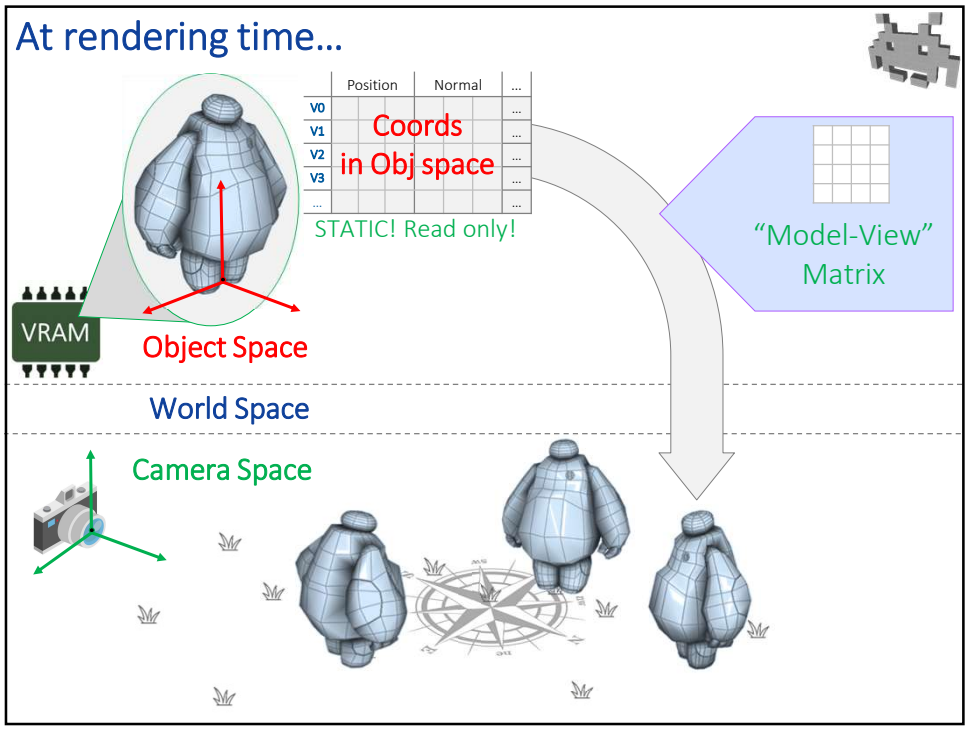
39


Rendering transforms
world
model-view transform (of the wheel):
$(T_8)^{-1} \circ T_0 \circ T_3$
T_0
T_8
T_2
camera
T_3
T_4
T_5
T_6
T_3
T_4
T_5
T_6

41


At rendering time...
Position
Normal
...
V0
V1
V2
V3
...
Coords in Obj space
STATIC! Read only!
VRAM
Object Space
"Model" Matrix
encodes
global transform
World Space

42



43

Rendering transofrms

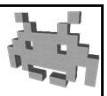
- The mesh buffers, once loaded in VRAM, are static.
  - When the object is rigid, at least
  - (Not a rope / a dynamic cloth for example)
- The only thing that changes between frames is the associated transform
- The transform:
  - Is sent to the GPU, to be applied to each vertex
  - Is applied on the fly to vertex position / normals, during rendering
  - Is typically expressed as a 4x4 matrix, which is ideal for the task (it's the quickest to apply)
  - Names of that matrix (in CG jargon):
    model matrix: from object space to global space
    model-view matrix: from object space to view space

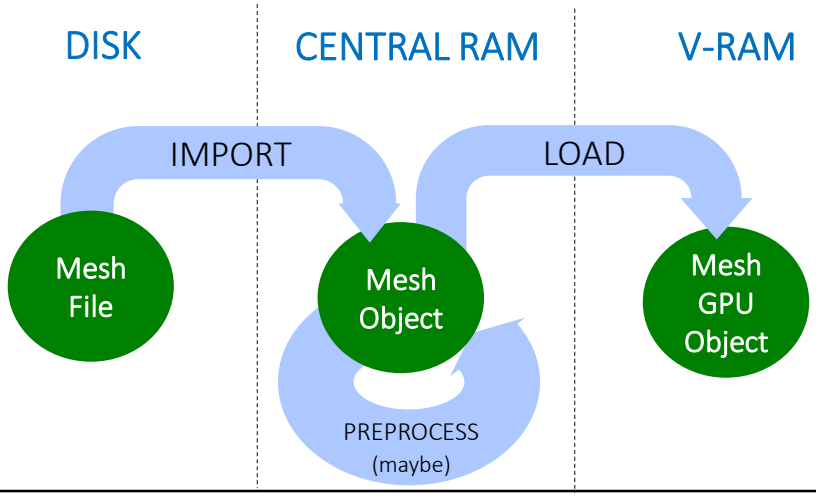
*that is, the matrix expressing the **global** transoform.*

*The **local space** of the **camera**. Ideal, for the rendering*

44

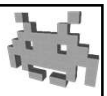
Life of a Mesh in a Game Engine





47

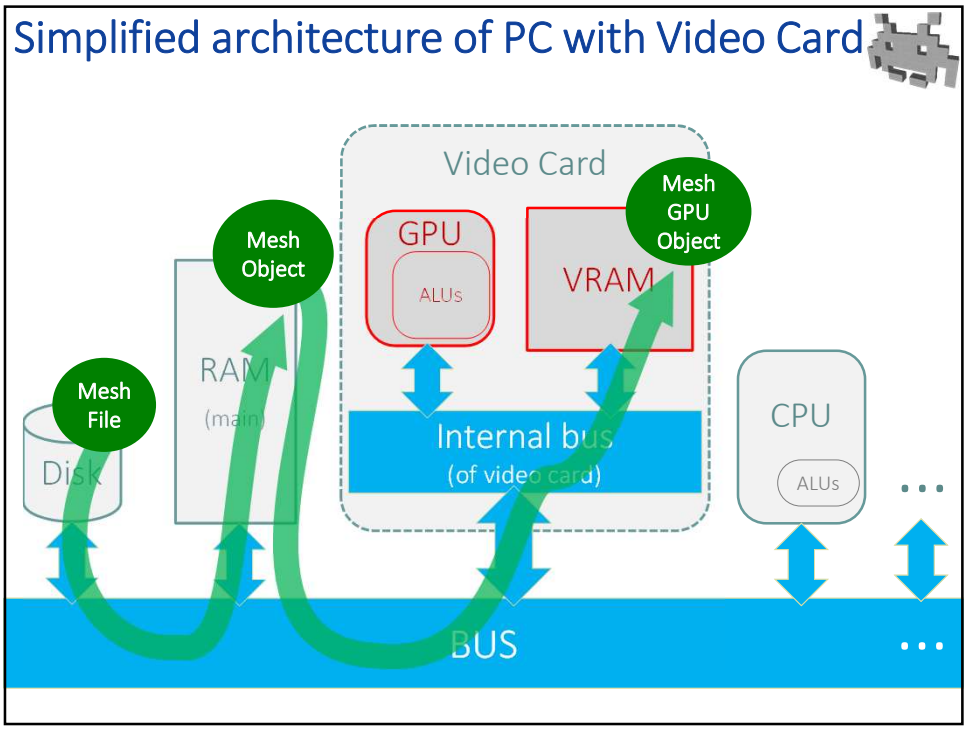
Life of a mesh in a game engine



- Import
  - from disk, or from remote
- 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
  - It must be loaded in GPU-ram first

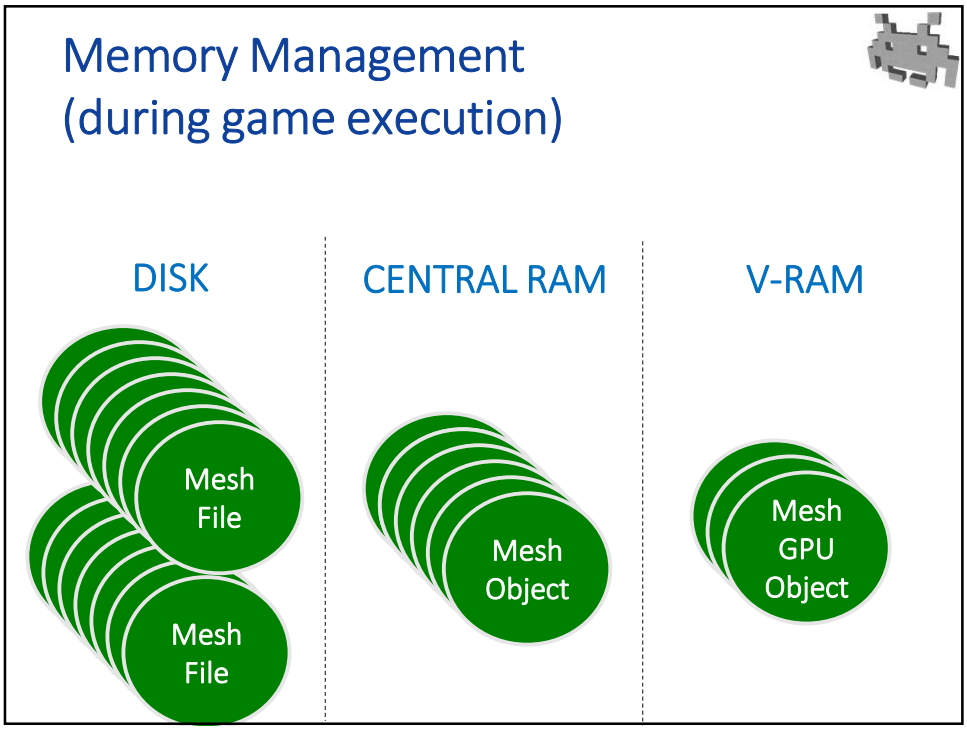
48

Simplified architecture of PC with Video Card



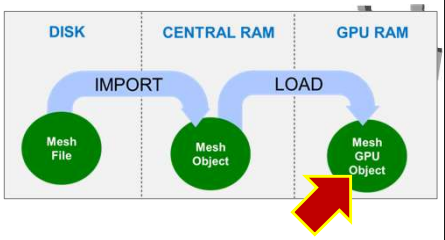
49

Memory Management (during game execution)



50

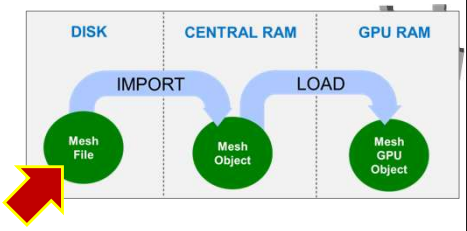
Mesh in VRAM



- Mesh stored as VRAM buffers
  - Named (by Graphics APIs): Vertex Buffer Object (VBO) or Vertex Arrays (VA)
- Coords (vertex pos, normal) in object space (by def)
- VRAM is a scarce, precious resource!
- Choices for a Game Engine:
  - storage formats, including precisions for each attribute, and how many buts per index. e.g.:
    - color? 8 bits per channel is usually fine
    - position? is 16 bit float per coordinate enough?
    - Vertex indices? Is 16 bit enough?
  - trade-off between storage cost / accuracy

52

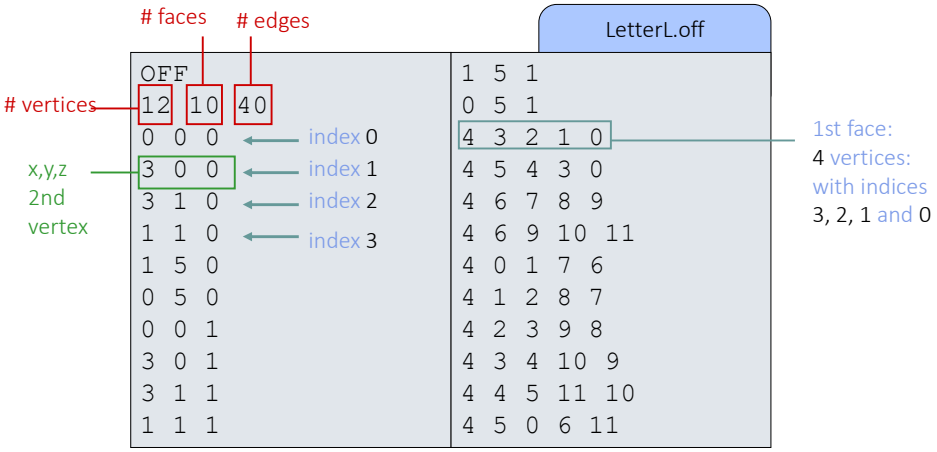
Mesh as an asset



- A file (or se of file) of a given format sitting on the disk
- Choices for the game engine:
  - which formats(s) to use?
  - which attributes are needed?
  - Etc.
- Issues:
  - storage cost
  - loading time

54

Example of file format for indexed meshes: OFF format




```
OFF
12  10  40
0  0  0
3  0  0
3  1  0
1  1  0
1  5  0
0  5  0
0  0  1
3  0  1
3  1  1
1  1  1
```

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

55

Most common file formats for meshes in games

.OBJ (wavefront)
- ☺ very broad diffusion
- ☺ mesh basics: indexed, normals , uv-mapping
- ☺ trivial to parse / read
- ☹ simple materials too
- ☹ material index for face (no colors)
- ☹ no skinning, animations, scenegraph…

.SMD (VALVE)
- ☺ Skeletal animations + skinning
- ☺ normals , uv-mapping
- ☹ meshes: not indexed, no colors…

.MD3 (Quake, IDsoft)
- ☺ good for blendshapes
- ☹ meshes: no colors

.PLY (cyberware)
- ☺ customizable
- ☹ "academic"

.DAE (collada: SONY + KHRONOS GROUP)

A format for the Exchange of Digital Assets
- ☺ complete:
  particle systems, physics attributes, scenegraph, skinned meshes, blend shapes, geometric proxies…
- ☺ open standard
- ☹ too complete? to parsing it completely

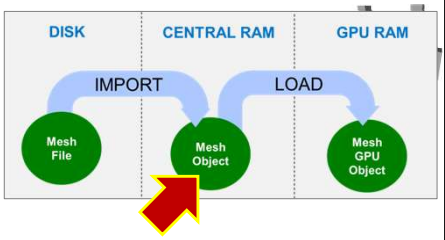
.FBX (AUTODESK.)
- ☺ abbastanza
- ☹ propretary

. glTF (Chronos, opensource)

Graphic Library Tranmission Format
A dump of memory structures for OpenGL rendering
- ☺ very complete, and customizable
- ☺ open standard
- ☹ includes animations, etc

58

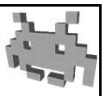
Mesh Object (in RAM)



- 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)

60

Data structure for a mesh (to be used, e.g., for processing)



- Indexed mode in C++ :

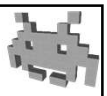
```
class Vertex {
  vec3 pos;
  rgb color;    /* attribute 1 */
  vec3 normal; /* attribute 2 */
};

class Face{
   int vertexIndex[3];
};

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

61

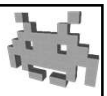
Mesh processing:
(or, more in general, Geometry Processing)



- The algorithm above
  for the computation of per vertex normal
  is one example of processing done over a mesh
- Mesh processing: the discipline of generating, processing, meshes
  - Algorithms having inputs and/or outputs as meshes
- See CG course for a very brief overview

66

3D Models: sources.
How to we get our 3D models?



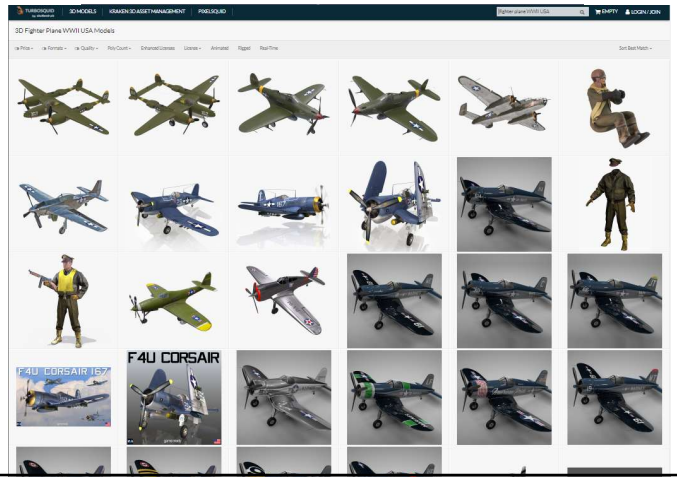
Will we'll discuss…

- Modelling by digital artists
  - Note: the 3D modelling phase is just a step of the Asset Creation pipeline
- Procedural modelling
  - As for any asset, the procedural approach is an option
  - Same trade-offs as usual for precedurality
  - Note: results can be baked (during production), or, generation can happen at game execution
- 3D acquisition
  - Scanning a real world, physical model

72

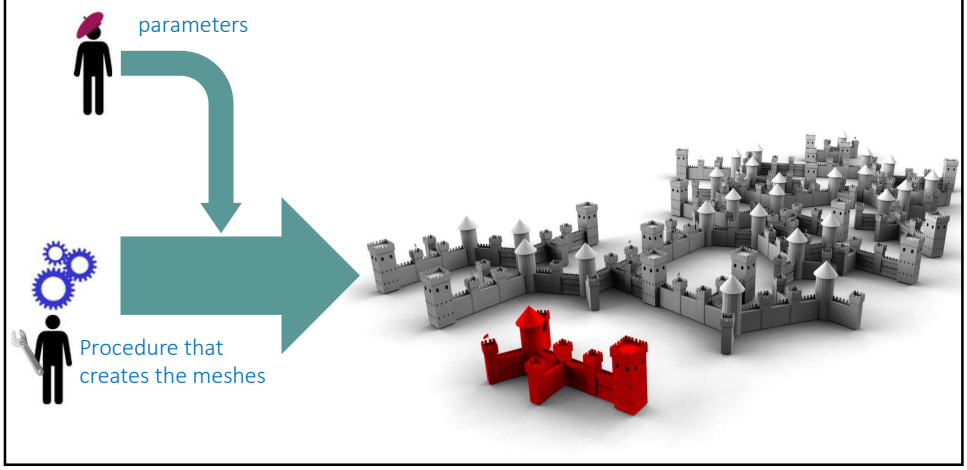
3D models: sources

- Or, like any asset, can be just bought / off-sourced
  - Try looking any repository for a given type of object



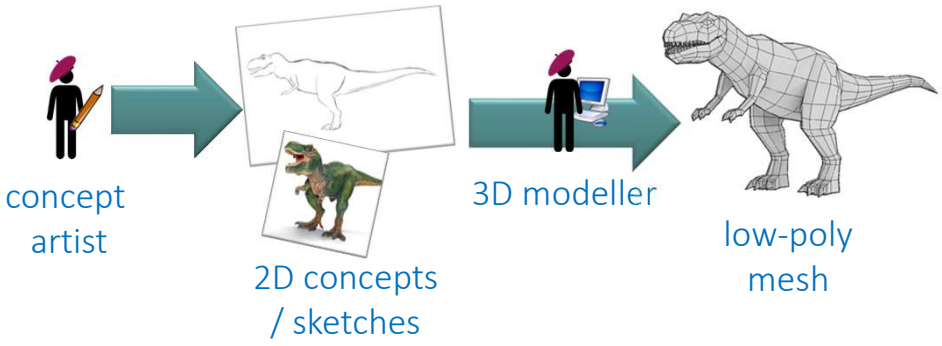
73

Sources for 3D models: procedural modelling



74

3D models: manual modelling



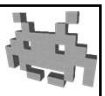
76

Mesh modelling

- Task of the 3D modeller
  - A type of digital artist
- Different approaches:
  - Direct low-poly modelling
  - Potentially, using subdivision surfaces too
  - Digital sculpting

77

Mesh modelling: a few popular suites

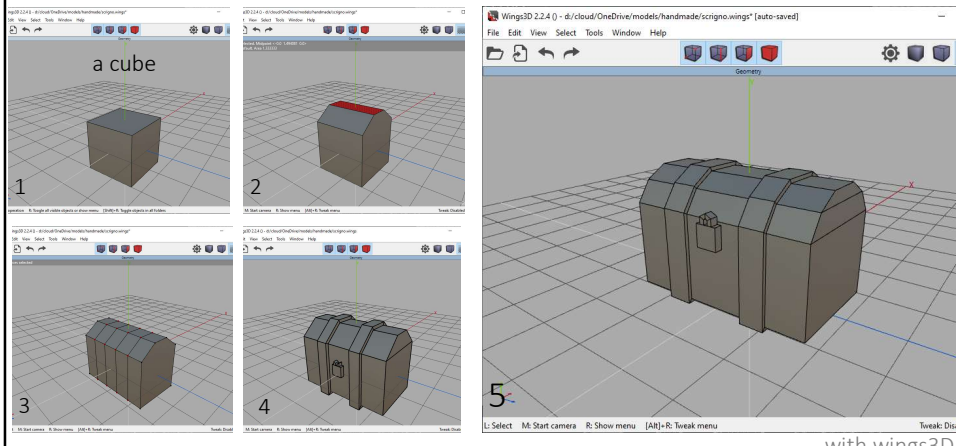


used in classroom demos

- 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
- AutoCAD (autodesk), SolidWorks (SolidThinking)
  - for CAD
- ZBrush (pixologic) (+ Sculptris alpha, a toy), 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…

78

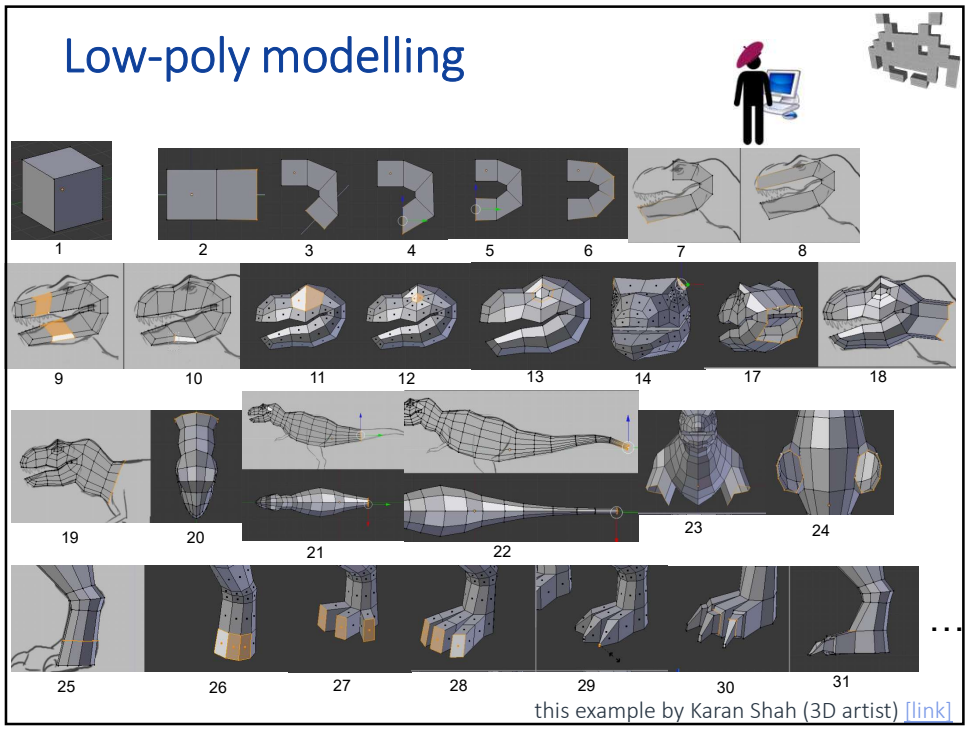
Low-poly modelling (demo)



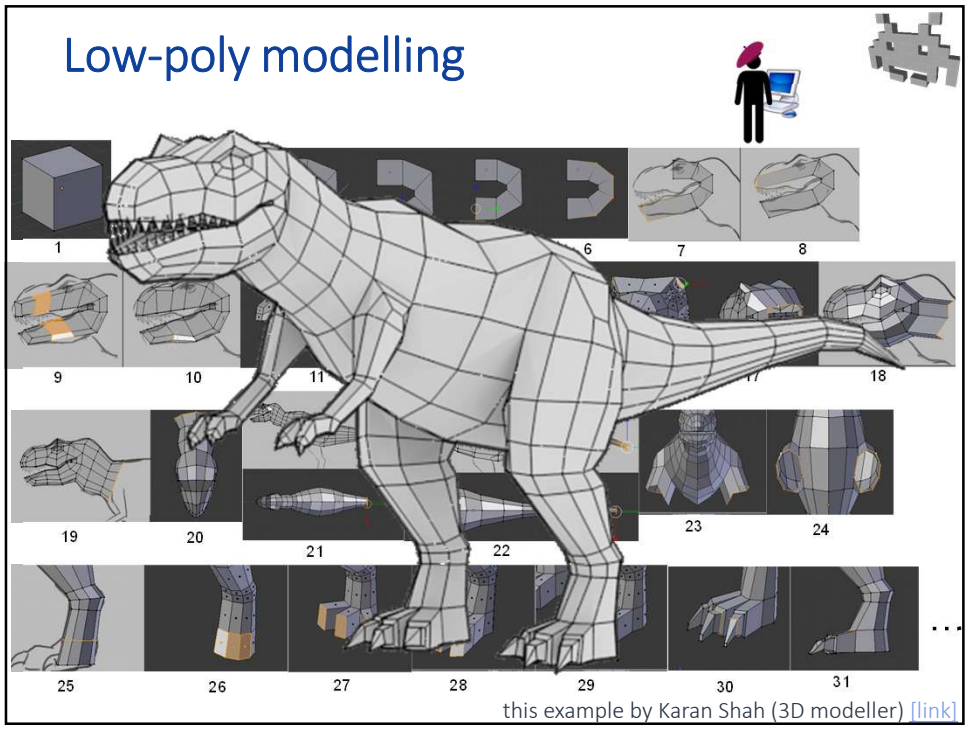
with wings3D

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

79

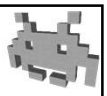

Low-poly modelling
1
2
3
4
5
6
7
8
9
10
11
12
13
14
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
this example by Karan Shah (3D artist) [link]

81


Low-poly modelling
1
6
7
8
9
10
11
18
19
20
21
22
23
24
25
26
27
28
29
30
31
this example by Karan Shah (3D modeller) [link]

82

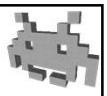
Mesh modelling



- Task of the 3D modeller
  - A type of digital artist
- Different approaches:
  - Direct low-poly modelling
  - Potentially, using subdivision surfaces too
  - Digital sculpting

102

Digital Sculpting



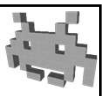
mouse
(or stylus)

=

chisel

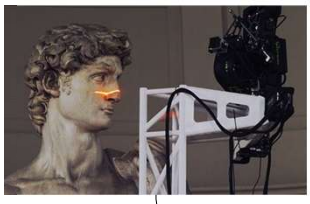
103

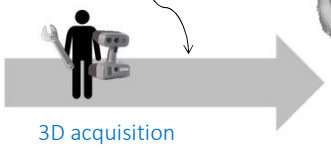
Sources for 3D models: 3D acquisition



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

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

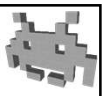




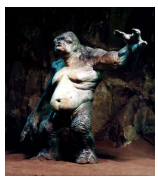
3D acquisition
(e.g., range scanning)
(specifically, here, laser scanning)

105

Sources for 3D models: 3D acquisition



Sculptor
(real)



Reale model

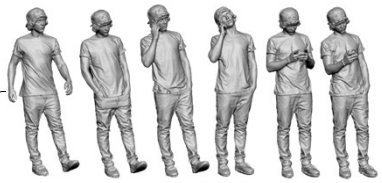
3D scanning

Hi res model

106

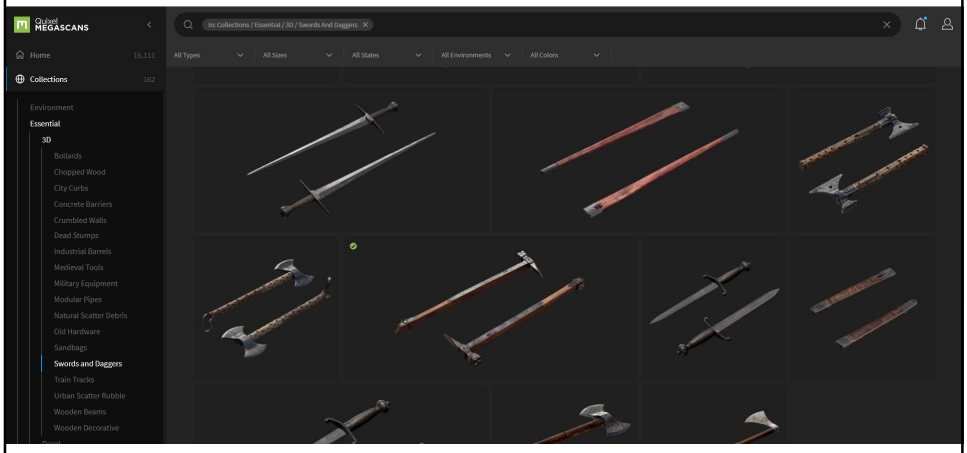
Sources for 3D models: 3D acquisition

- 3D scanning
  - A.k.a. *automatic 3D model acquisition*
  - Many 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?)



107

Example: a repository of production-ready 3D meshes for games scanned from real objects

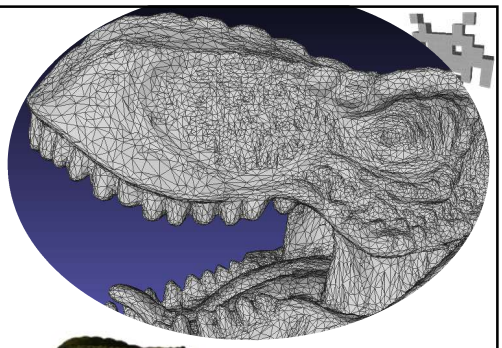


https://quixel.com/

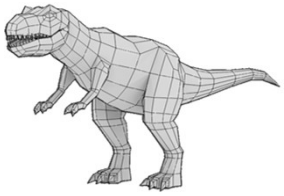
of course, scanning is just one phase of the Production pipeline

108

3D models sources: a comparison



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



manually edited
low-poly mesh
(2K triangles)

VS

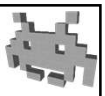
Dino, scanned by artec3d

scanned & cleaned
hi res mes
(30K triangles)

(sculpted meshes are similar)

109

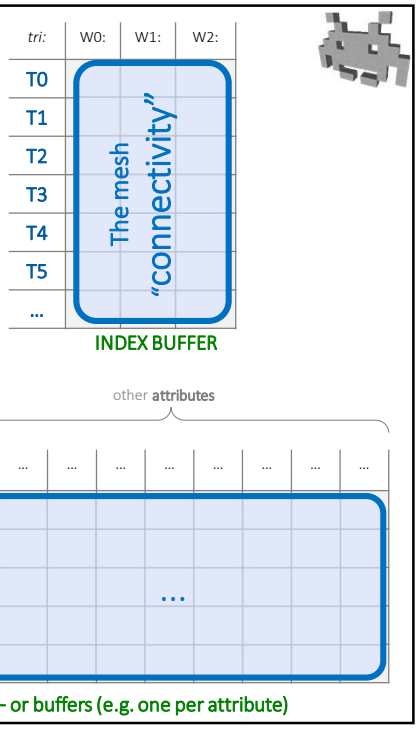
Notes about mesh resolution



- Costs: linear on the triangles number
  - in memory (disk, CPU RAM, GPU RAM)
  - in time (rendering, loading, etc)
- (also, linear with number of vertices)
  - (because, as a rule of thumb: n verts → $2n$ tris)
- adaptive resolution is possible in a mesh
  - higher-res (more numerous, smaller triangels) in some parts
  - lower-res (sparser vertices, connected by larger triangles) in others

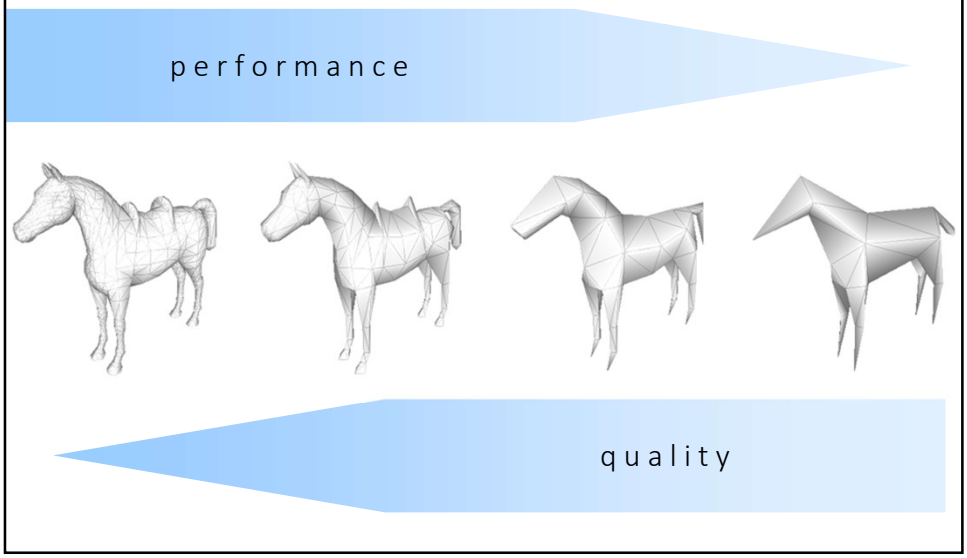
110

Summary: mesh as buffers (tables in GPU ram)



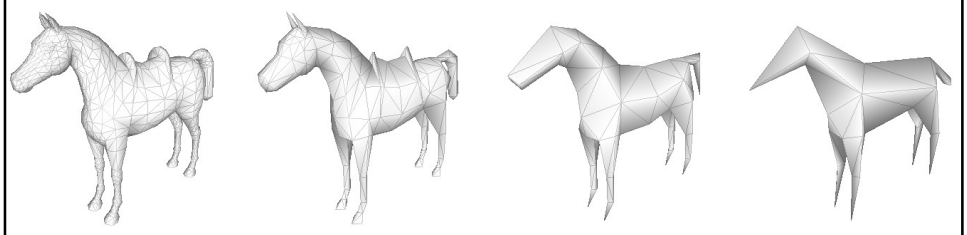
111

Rendering quality and resolution



112

LoD pyramids
(Level Of Detail)



LoD 0
2K faces

LoD 1
400 faces

LoD 2
160 faces

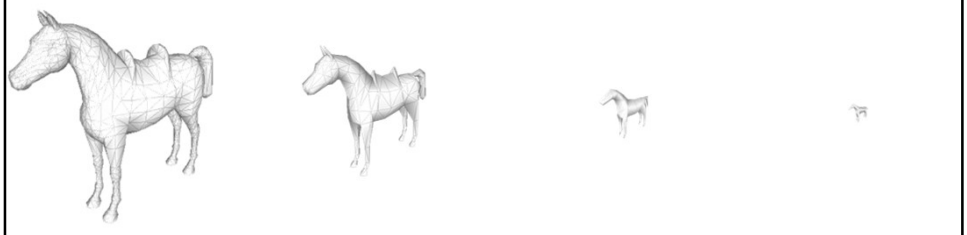
LoD 3
60 faces

use when seen up close

use when seen from afar

113

LoD pyramids
(Level Of Detail)



<5 m

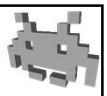
5m ..10 m

10 .. 30 m

>30 m

114

LoD pyramids (Level Of Detail)



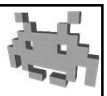
- Goal:
  - decrease the geometry budget (total number of vertices)
  - ideal: size of triangles in screen space (in pixel): constant
    - importance / geometrical complexity being the same
- Task: determining the level to use (dynamically, at runtime)
  - depending on observer distance
  - and/or, depending on rendering workload
    - e.g.: rendering is lagging ⇒ decrease LoD
  - this is task of the rendering engine)

computed from scene graph (how?)

- Task: LOD creation or "LOD-ding" (during asset creation)
  - starting from LOD-0 (higher-res)
  - manual, or automatic
    - difficult to automatize well, for very coarse LODs
  - note: sometimes "LoD-0" is used only in special cases
    - e.g., for cut-scenes

115

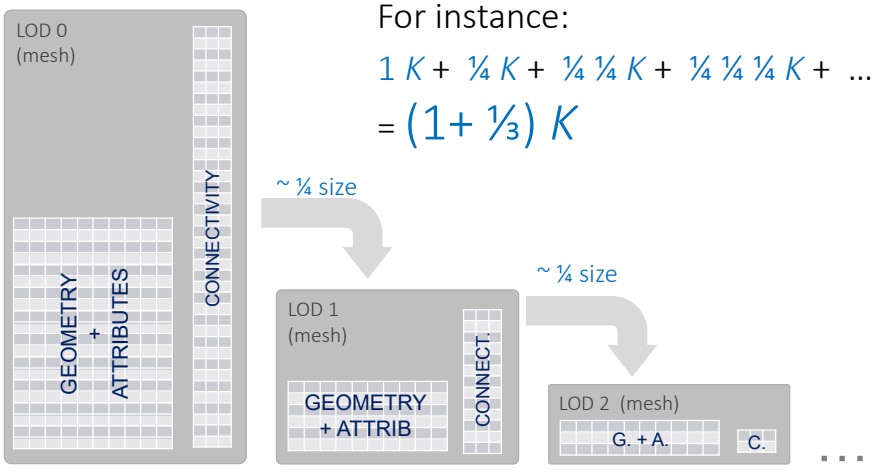
LoD pyramids (Level Of Detail)



Total memory usage: limited
For instance:

$1\ K + \frac{1}{4}\ K + \frac{1}{4}\frac{1}{4}\ K + \frac{1}{4}\frac{1}{4}\frac{1}{4}\ K + \ldots$

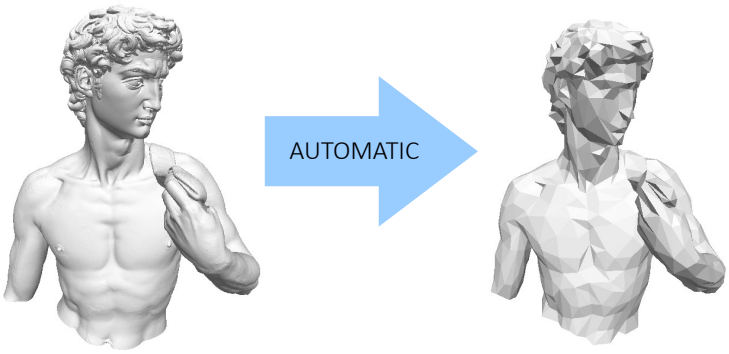
$= (1 + \frac{1}{3})\ K$



116

Poly-reduction
(aka Mesh "simplification" / "decimation" / "coarsening")

- parameters:
  - a maximum error
  - or number of faces objective



Original mesh
500K triangles

Simplified mesh
2K triangles

119

Poly-reduction
(aka Mesh "simplification" / "decimation" / "coarsening")

- Different approaches are studied (in Geometry Processing)
  - Can be adaptive or not
    - Adaptive = strive to use more triangles where needed
  - Maximum error introduced:
    - can be measured and/or limited
    - or not
  - Topology (e.g. holes, handles):
    - can be kept
    - or not
  - Streamable
    - Possible
    - or not
    - ...

120