

3D video games

Models for Games




Marco Tarini



1

Course Plan

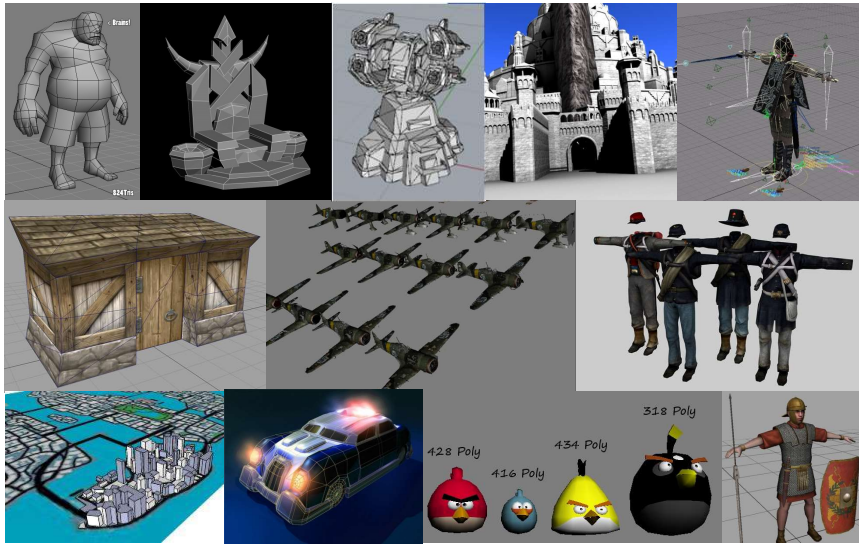


- lec. 1: **Introduction** ●
- lec. 2: **Mathematics** for 3D Games ●●●●●●
- lec. 3: **Scene Graph** ●
- lec. 4: **Game 3D Physics** ●●●● + ●●
- lec. 5: **Game Particle Systems** ▶
- lec. 6: **Game 3D Models** 📍
- lec. 7: **Game Textures** ▶●
- lec. 9: **Game Materials** ◀
- lec. 8: **Game 3D Animations** ▶●●
- 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 ●

★
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



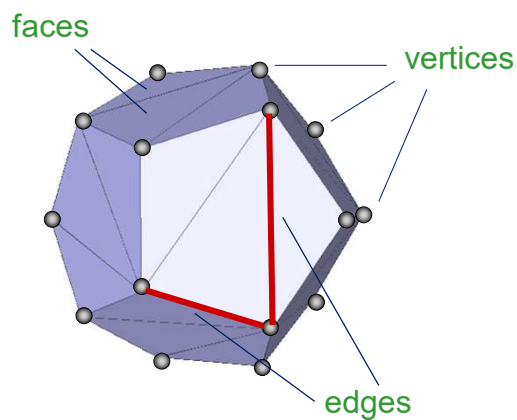
- Data structure for modelling 3D objects
 - GPU friendly
 - Resolution = number of faces
 - Resolution is (potentially) Adaptive (that is, more faces where needed)
- Used to model the **visual appearance** of 3D physical objects in the game
 - at least, the ones which can be represented by their surface
 - 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!

V1

V2

V3

V4

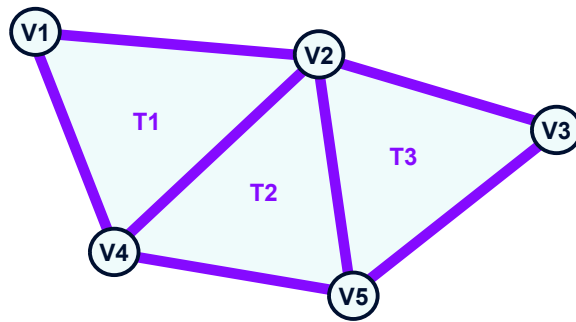
V5

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*

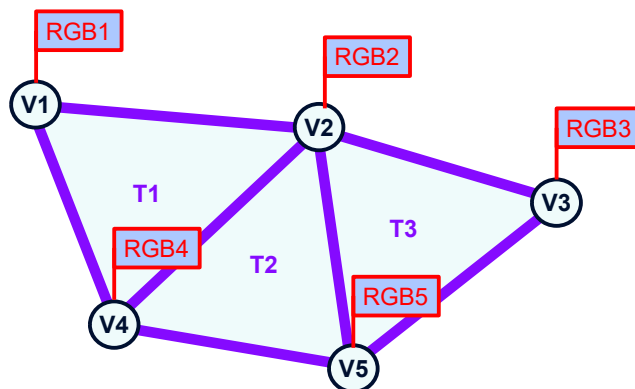


11

Mesh: attributes




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



12

Mesh as a data structure: 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»
 - Each triangle stored as
 - triplet of **indices** (referring to a vertex in the array)

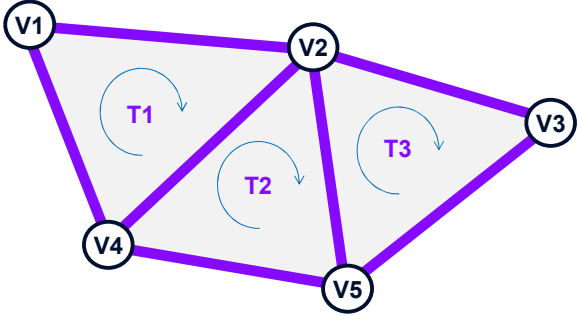
These two arrays can be seen as tables (buffers)

we can consider positions as attributes too

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 $\#F \approx 2 \cdot \#V$
- 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: $10^5 \Delta$ is hi-res
 - 2000s: $10^{10} \Delta$ is hi-res

16

Resolution increases over time



800 Δ Unreal Tournament (1999)

4,500 Δ weapon Unreal Tournament (2002)

this 12,000 Δ Unreal Tournament (2002)

3000 Δ Unreal Tournament 3 (2007)

19



21

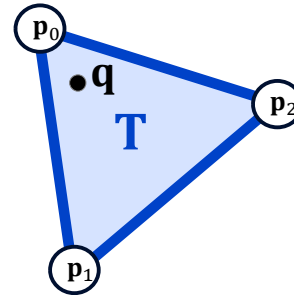
Mesh attributes: in general (this applies to any attribute)

- 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 C^0 continuous across faces
 - but C^1 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



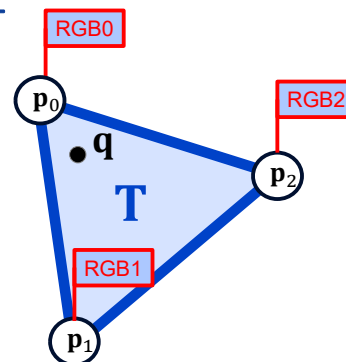
$$\mathbf{q} = k_0 \mathbf{p}_0 + k_1 \mathbf{p}_1 + k_2 \mathbf{p}_2$$

- 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 barycentric coordinates k_0, k_1, k_2 is implicitly given the attribute value




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

per vertex

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 buffer: a more realistic views



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

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

Ints
(with some # of bits)

CONNECTIVITY

| vert | X | Y | Z | 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)

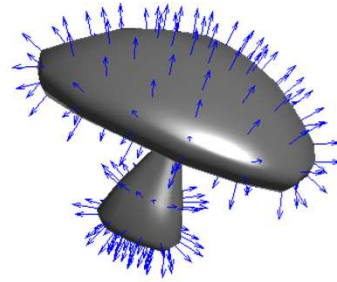
GEOMETRY + ATTRIBUTES

27

Mesh attributes: normals



- A vector
- 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

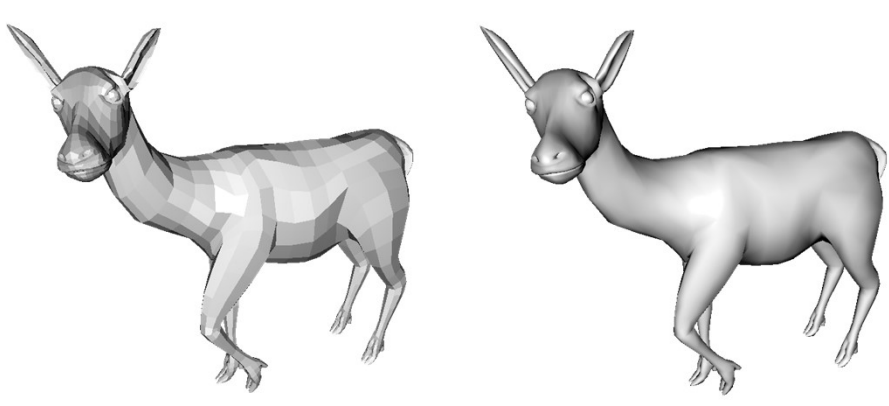
Mesh attributes: normals



- 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)

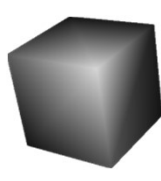
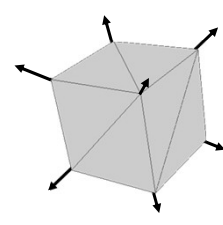
Note: normals are made visible to our eyes due to lighting (computation of how light reacts with the surface)

31

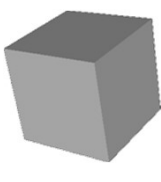
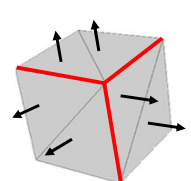
Hard edges (aka sharp edges) (aka “creases”)

- Edges where the normal is **not continuous**.

Soft edges:



Red edges are hard

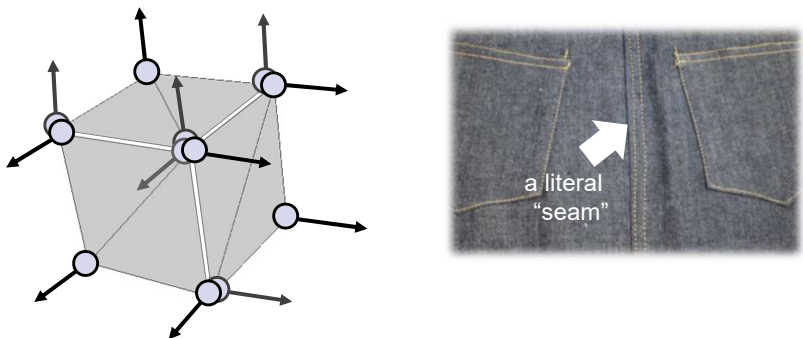


- How to encode (Co) a **discontinuity** in any attributes?

32

answer:
Vertex seams

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



The diagram on the left shows a mesh with a central vertex and several surrounding vertices. Arrows point outwards from each vertex, representing normal vectors. A central vertex is highlighted with a blue circle, and its position is duplicated, with the duplicate also highlighted with a blue circle. The diagram on the right is a photograph of a denim jacket with a white arrow pointing to a vertical seam, labeled "a literal 'seam'".

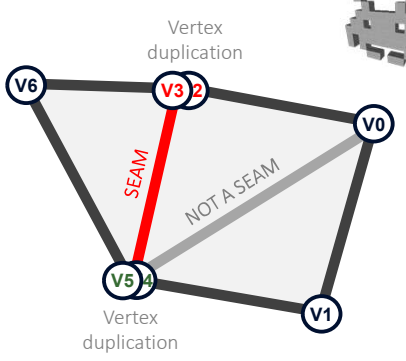
33

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 as V2 | same as V2 | same as V2 | n_x3 | n_y3 | n_z3 |
| V4 | p_x3 | p_y3 | p_z3 | n_x4 | n_y4 | n_z4 |
| V5 | same as V4 | same as V4 | same as V4 | n_x5 | n_y5 | n_z5 |
| V6 | p_x4 | p_y4 | p_z4 | n_x6 | n_y6 | n_z6 |

GEOMETRY + ATTRIBUTES



The diagram shows a mesh with vertices V0, V1, V2, V3, V4, V5, and V6. V3 and V4 are labeled as "Vertex duplication". A red line connects V3 and V4, labeled "SEAM". A grey line connects V3 and V4, labeled "NOT A SEAM".

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

CONNECTIVITY

35

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

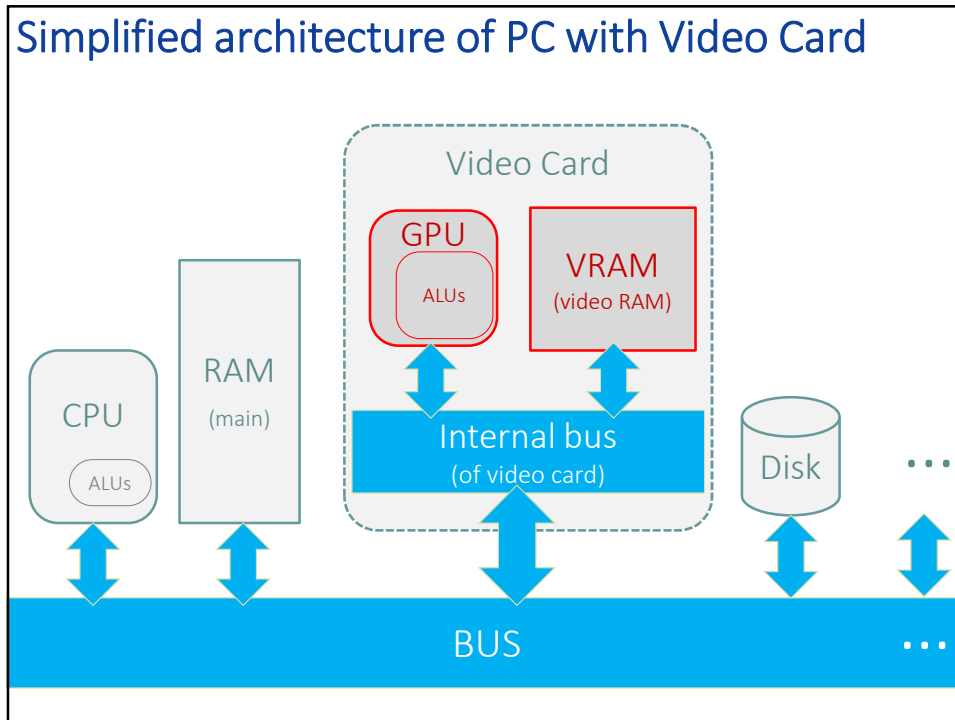
...and Fire!

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

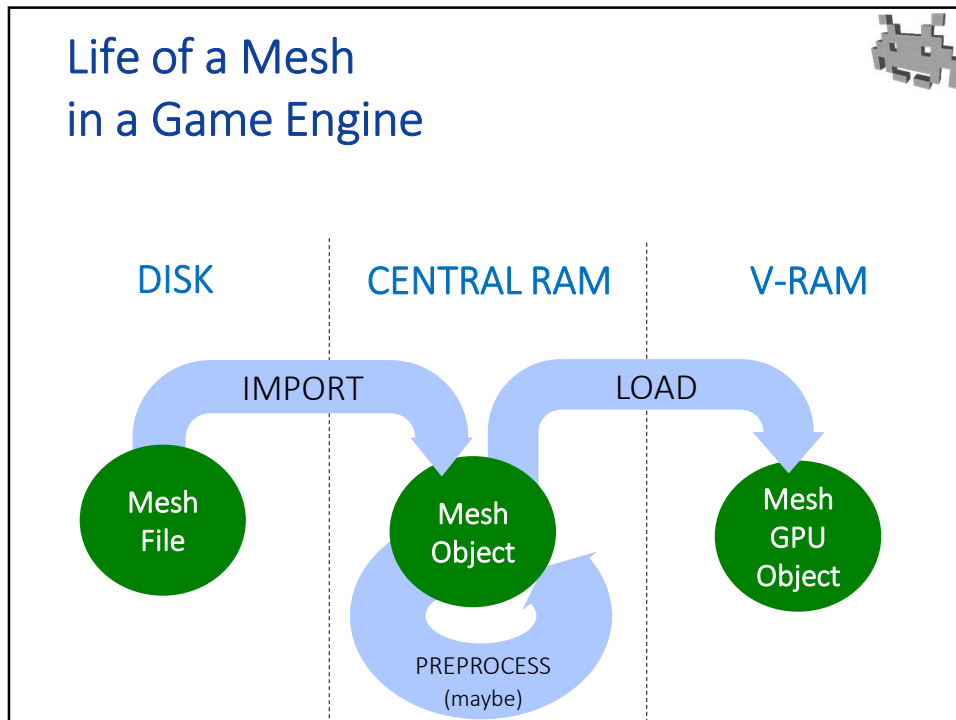
THE MESH

THE "MATERIAL"

36



37



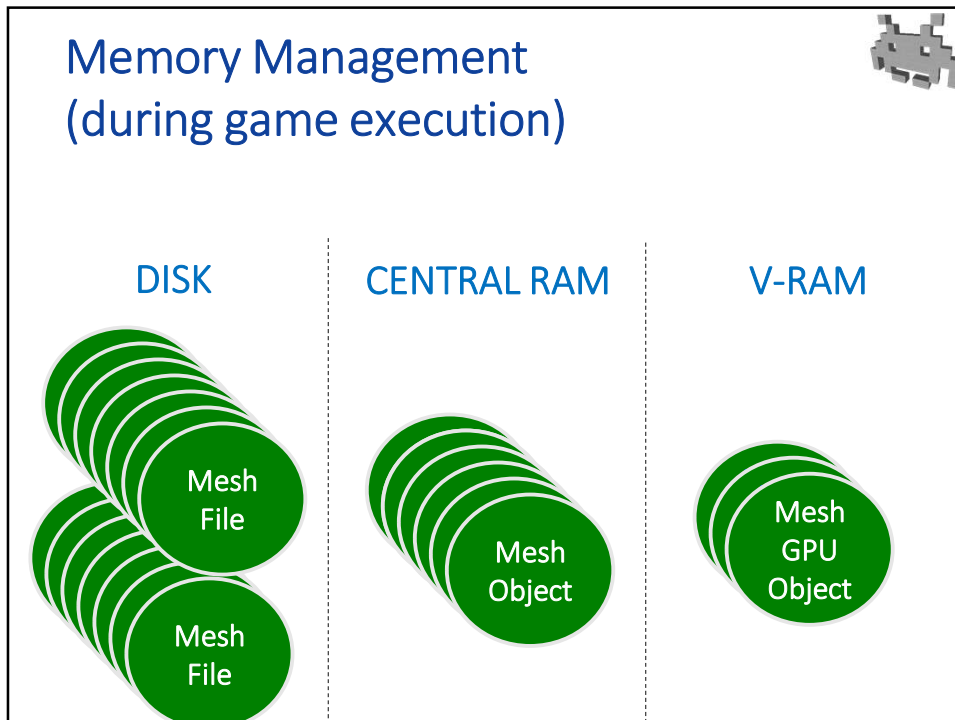
44

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

A small grey 3D mesh icon is in the top right corner.

45



46

Mesh on IN VRAM

DISK CENTRAL RAM GPU RAM

IMPORT LOAD

Mesh File Mesh Object Mesh GPU Object

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

47

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

49

Example of file format for indexed meshes: OFF format

```

LetterL.off
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
    
```

vertices: 12, 10, 40 (pointing to # faces, # edges)

x,y,z 2nd vertex: 3 0 0 (pointing to index 1)

index 0: 0 0 0

index 1: 3 0 0

index 2: 3 1 0

index 3: 1 1 0

1st face: 4 vertices with indices 3, 2, 1 and 0 (pointing to 4 3 2 1 0)

50

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

.3DS, .MA/.MB (AUTODESK)

- ⊕ customizable
- ⊕ "academic"

.DAE (collada: SONY + KHRONOS)

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)

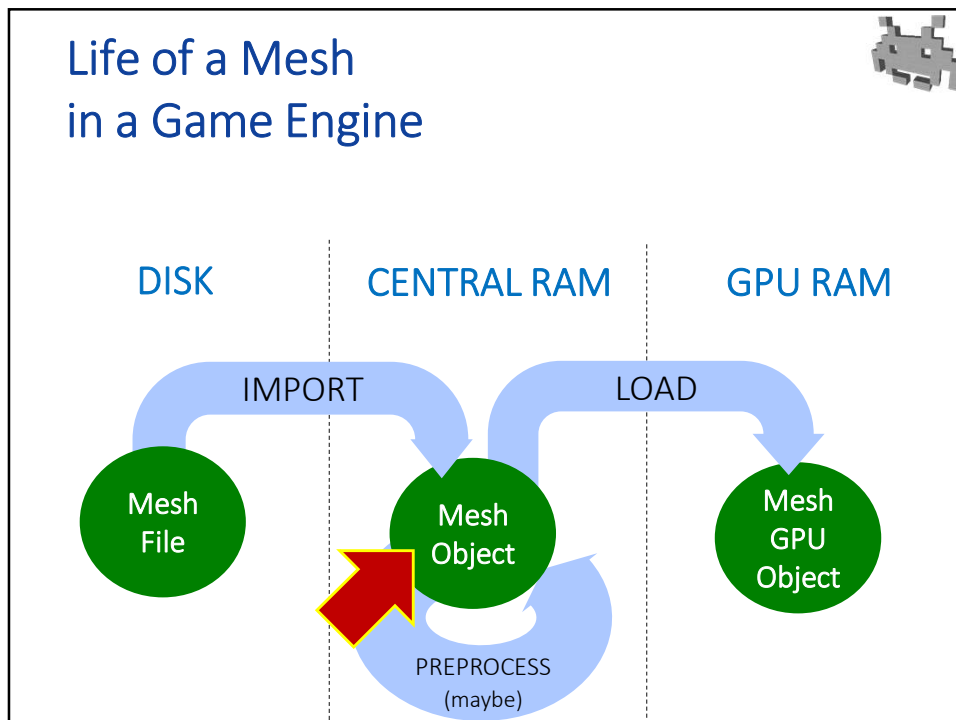
- ⊕ good for animations
- ⊕ proprietary

.glTF (Chronos, opensource)

Graphic Library Transmission Format
 A dump of memory structures for OpenGL rendering

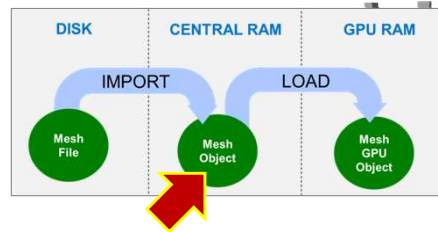
- ⊕ very complete, and customizable
- ⊕ open standard
- ⊕ includes animations, etc

53



54

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)

55

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



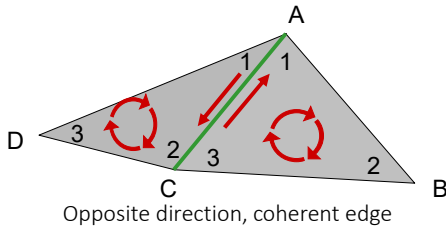
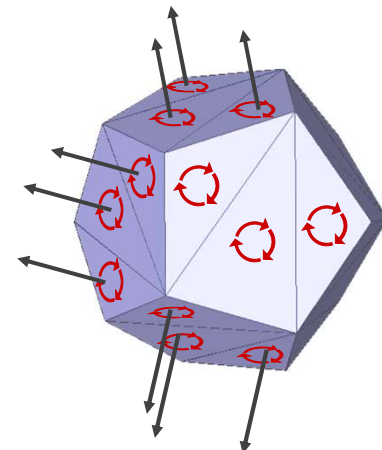
- Indexed mode in C++ :

```
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 */  
};
```

56

Compute normals from geometry

- Note: the faces **orientation** must be **coherent**

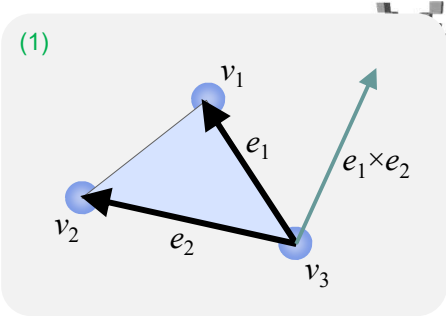
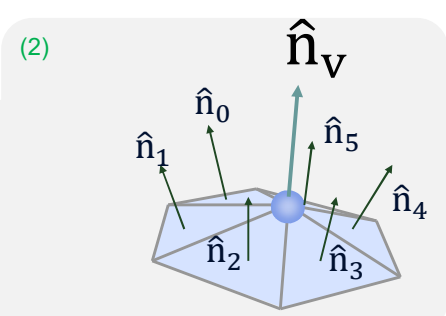



57

Computing normals from geometry


- compute the normal of each face
- cumulate the normals in each vertex

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

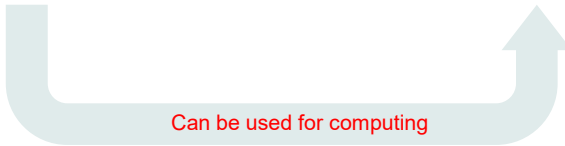



59

Note: surface normals




| | | |
|---|--------|--|
| geometric normals <ul style="list-style-type: none">• Defined per face• Implicitly defined by mesh geometry• The “real” orientation of the faces (which are flat!) | \neq | normals as attribute <ul style="list-style-type: none">• Defined per vertex• Explicitly stored, in the mesh data structure• A choice: which surface (e.g.) the artist is trying to represent? |
|---|--------|--|



Can be used for computing

60

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

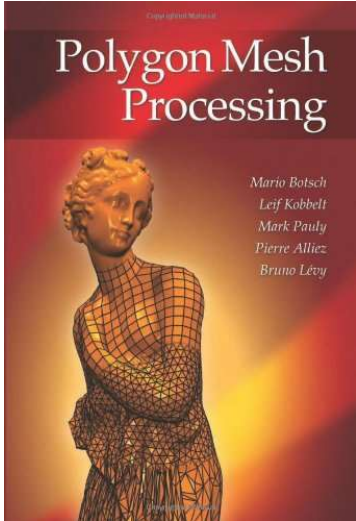


- The algorithm above for the computation of per vertex normal is one example of processing done over a mesh
- **Mesh processing**: the discipline of generating, processing, meshes
 - Algorithms having inputs and/or outputs as meshes
- See CG course for a very brief overview

61










Mesh processing: (or more in general Geometry Processing)

- A good textbook to mesh processing



62

C++ libraries for mesh processing

| | |
|--|--|
|  <p>VCG-Lib vision and computer graphic library CNR ()</p> |  <p>CGAL computational geometry algorithms library INRIA ()</p> |
|  <p>OpenMesh +  OpenFlipper RWTH ()</p> |  <p>libigl simple geometry processing library NYU ()</p> |

63

Mesh processing: a software suite



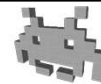
MeshLab

MeshLab

- open-source,
- A big collection of geometry processing algorithms
- ...

64

Mesh processing: examples of tasks commonly employed in games



- 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

65

3D Models: sources.

i.e.: where do a 3D models come from?

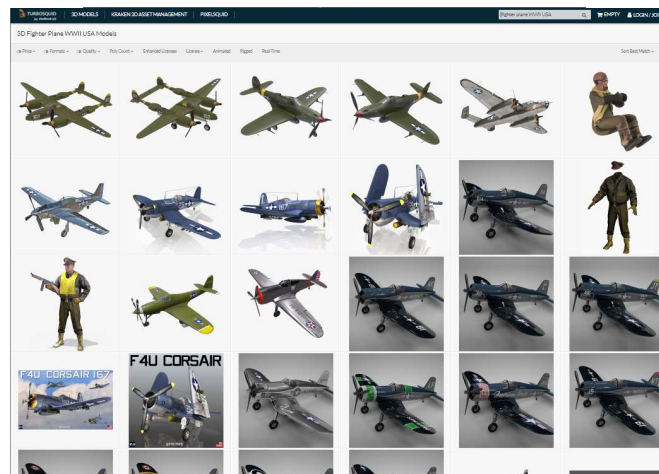
Will we'll see...

- **Modelling** by digital artists
 - Note: the 3D modelling part is just a step of the Asset Creation pipeline
- **Procedural modelling**
 - As for any asset, the procedural approach is an option
 - Usual trade-offs
- **3D acquisition**
 - Scanning a real world, physical model

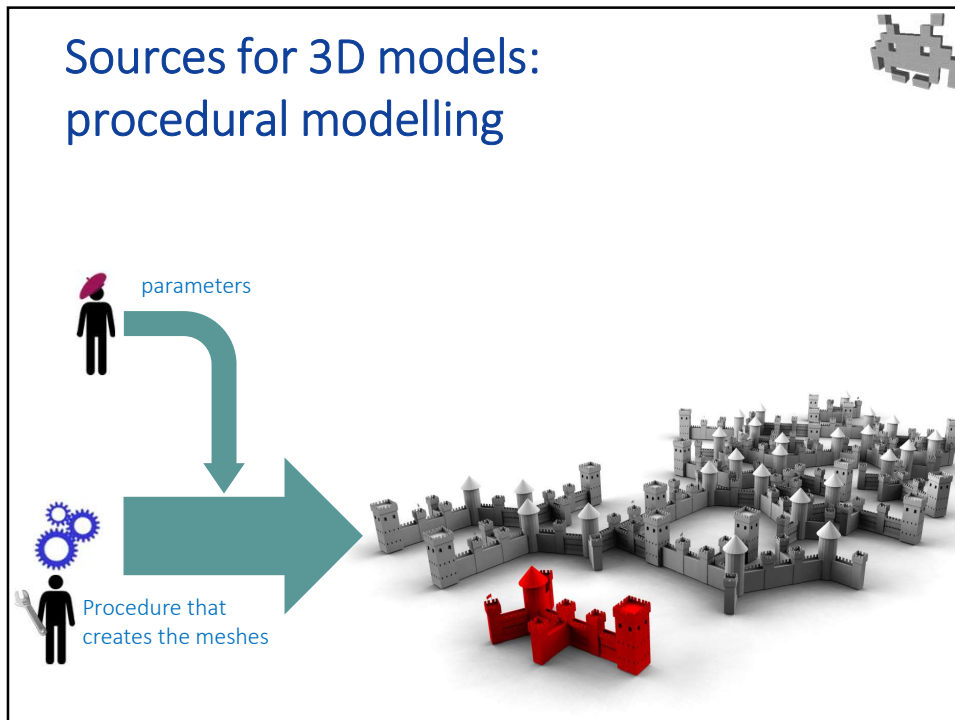
67

3D models: sources

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



68



69

Procedural modelling – see also...

EVERYTHING PROCEDURAL

EPC2024

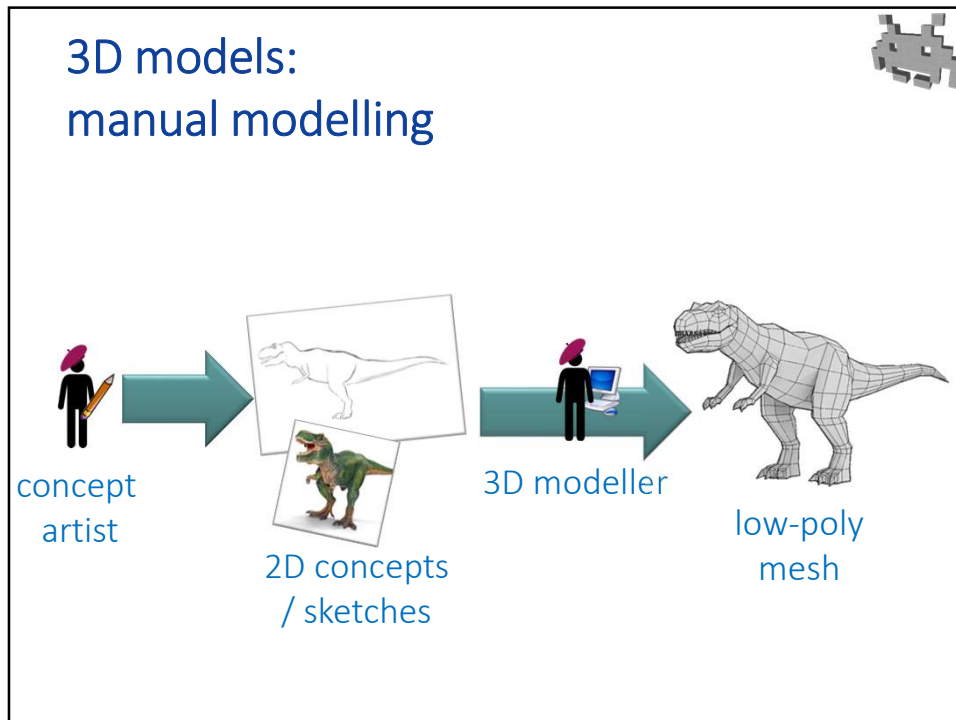
15-19 APRIL 2024

CONFERENCE ON PROCEDURAL GENERATION FOR GAMES

<http://everythingprocedural.com/>


this week
Game-of-the-Week


70



71


Mesh modelling

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



72

Mesh modelling: a few popular suites

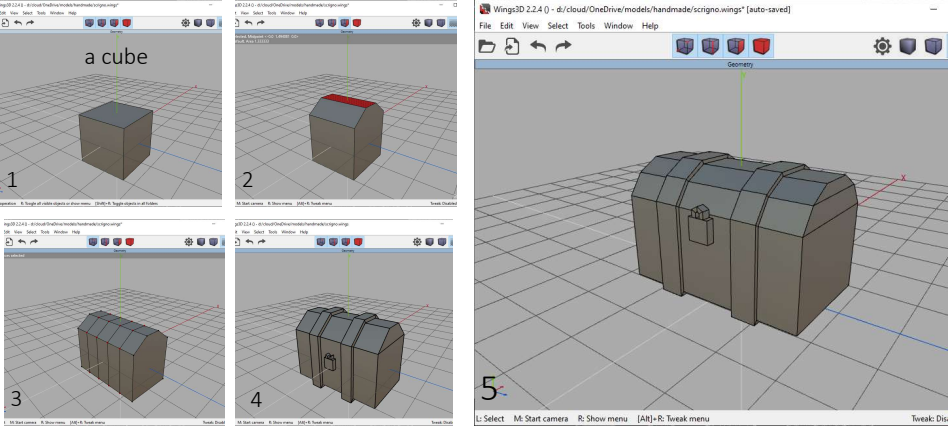




used in classroom demos

- **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 (including 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...

73

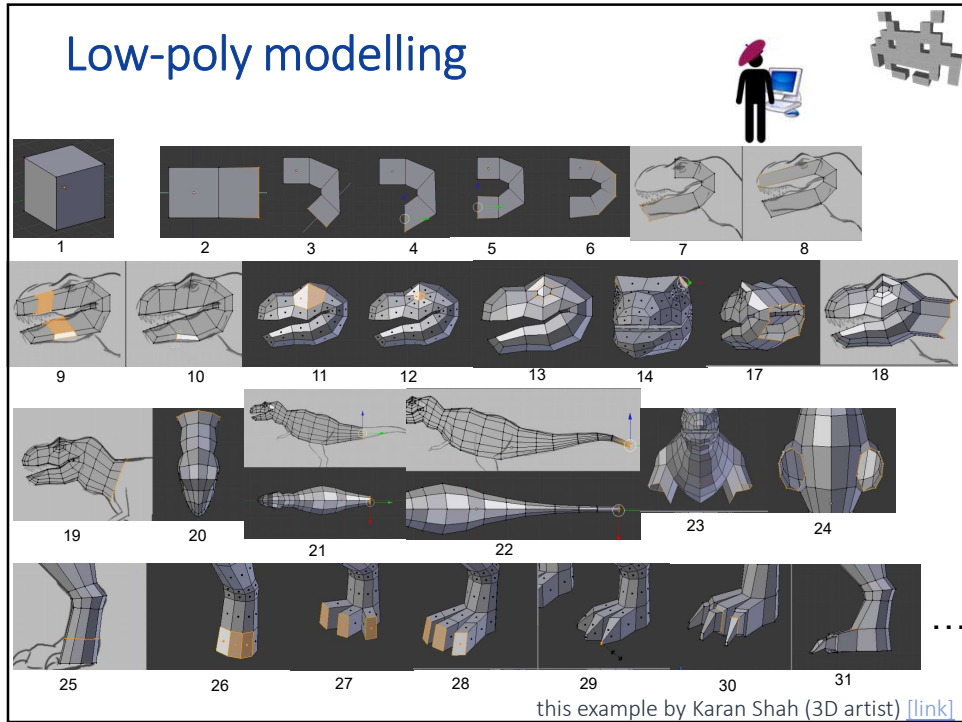
Low-poly modelling (demo)



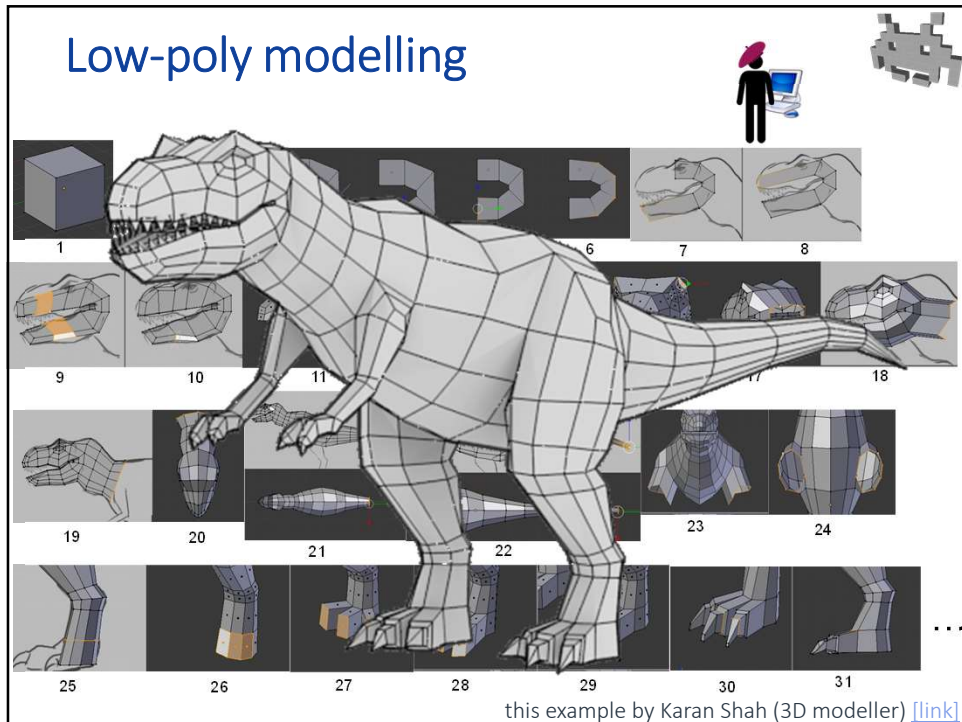
with wings3D

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

74



76

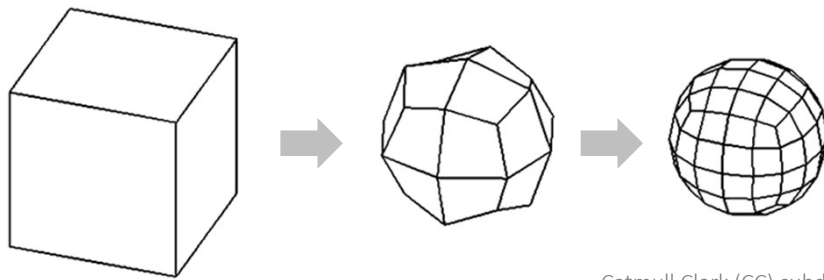


77

3D mesh authoring techniques: subdivision surfaces



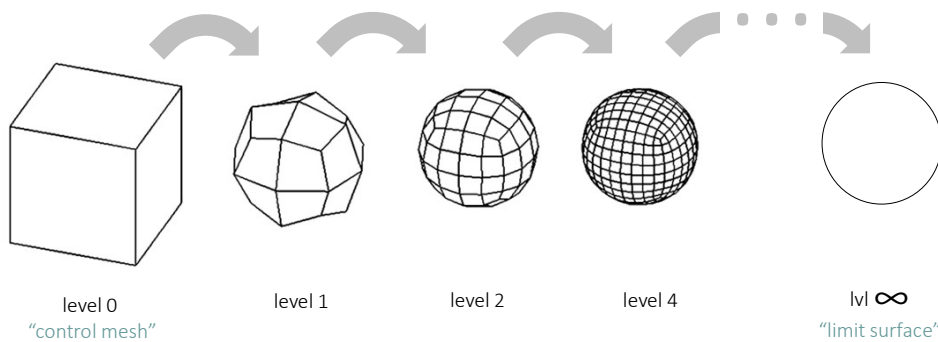
- **Subdivision step:**
an algorithm that operates on a mesh
and obtains a higher resolution, smoother mesh
- Can be iterated



Catmull Clark (CC) subdivision

78

Subdivision Surfaces Example: *Catmull-Clark* schema

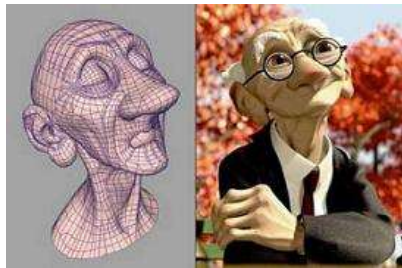


79

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):

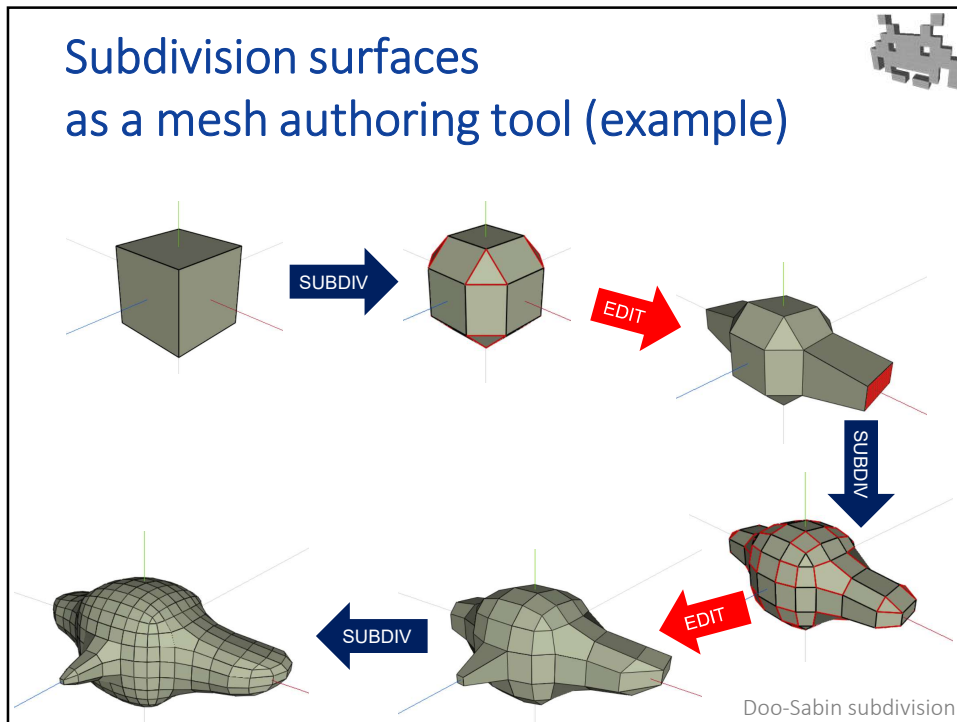


80

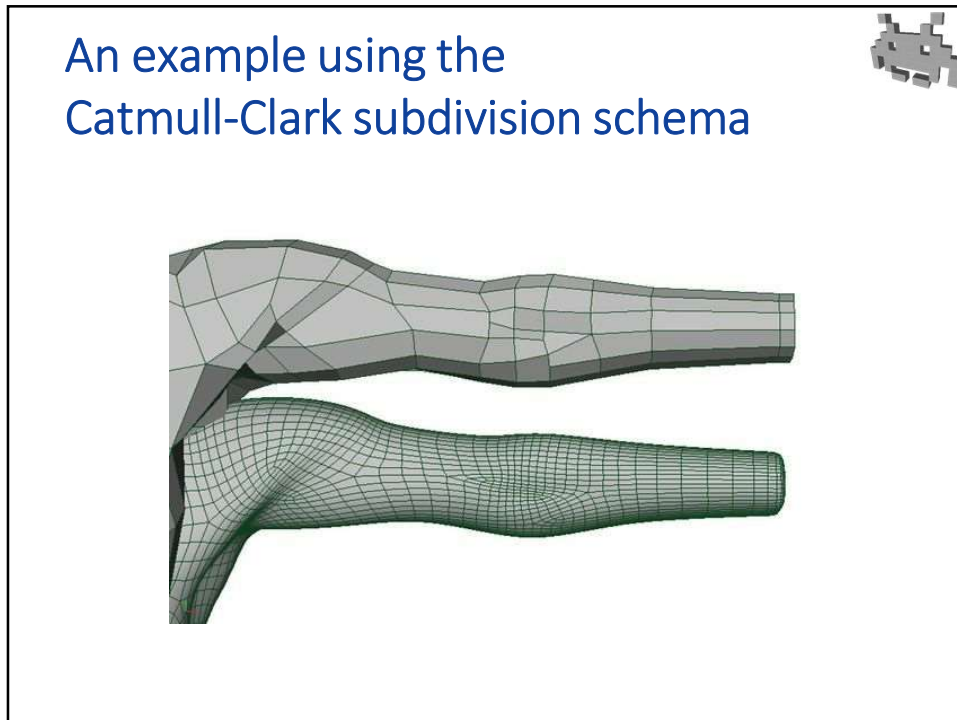
Subdivision surfaced as way to define (curved) surfaced

- 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 serves as a representation of the limit surface
 - note: the subdivision steps are only performed on the fly, for example, only during rendering (GPU support exists for this)
 - the more step are done, the better the limit surface is approximated

82



84



91

Some popular subdivision schemas



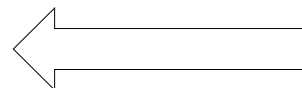
- **Doo-Sabin**
 - operates on any polygonal mesh
 - produces polygonal meshes
- **Loop**
 - 1-to-4 scheme for triangle meshes (only)
- **Butterfly**
 - 1-to-4 scheme for triangle meshes (only)
- **Catmull-Clark**
 - operates on any polygonal mesh
 - produces quad-meshes
 - traditionally, movie-industry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

95

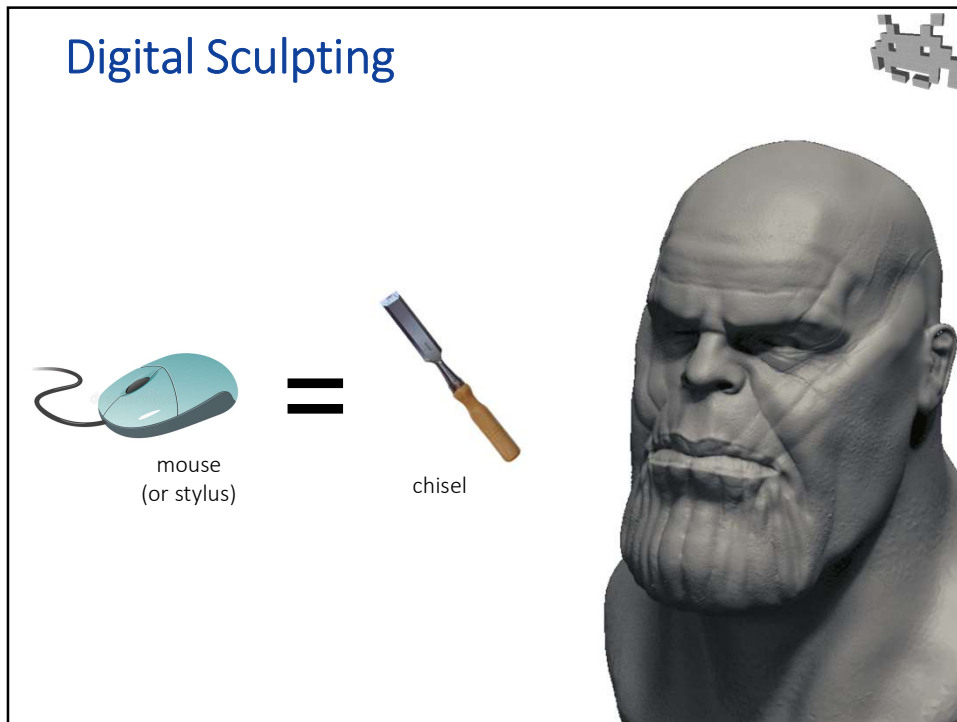
3D Mesh authoring: approaches



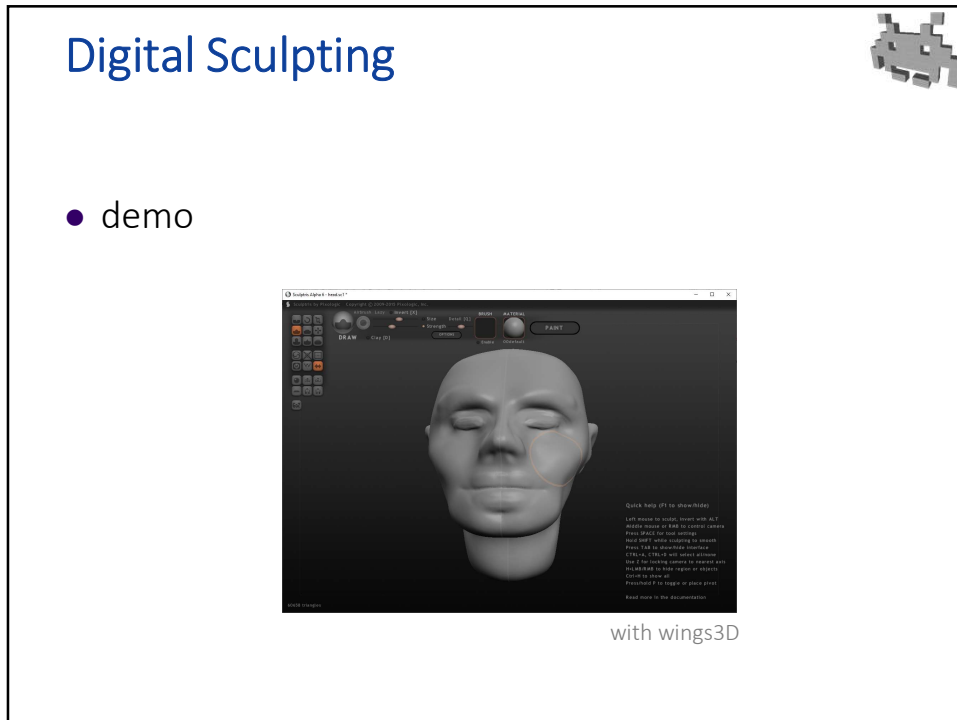
- Popular 3D modeling approaches:
 - Direct **low-poly modelling**
 - e.g. with wings3D
 - **Subdivision surfaces**
 - e.g. with blender
 - **Digital sculpting**
 - e.g. with Z-brush,
(or Sculpttris Alpha)



97



98

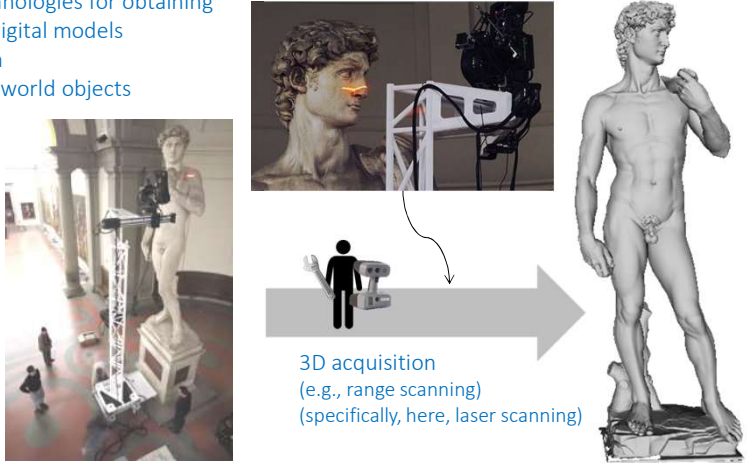


99

Sources for 3D models: 3D acquisition

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

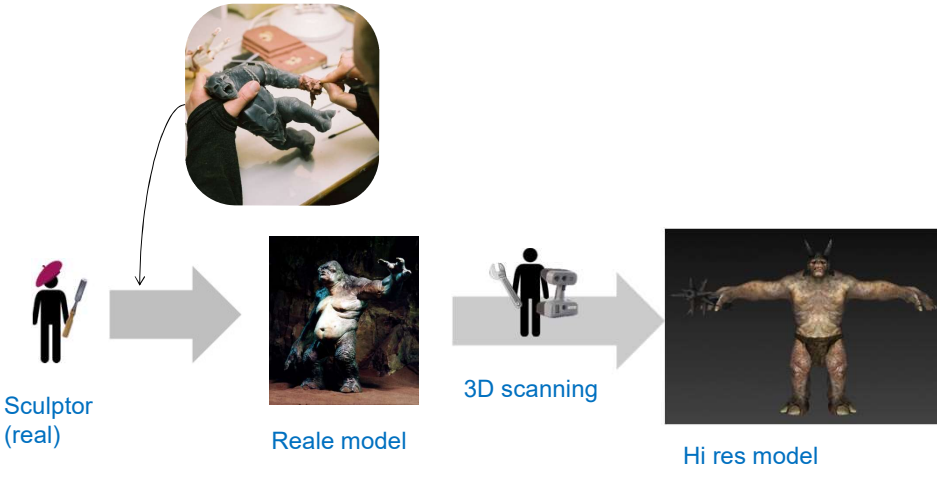
- 3D acquisition / 3D scanning
 - Technologies for obtaining 3D digital models from real-world objects



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

100

Sources for 3D models: 3D acquisition



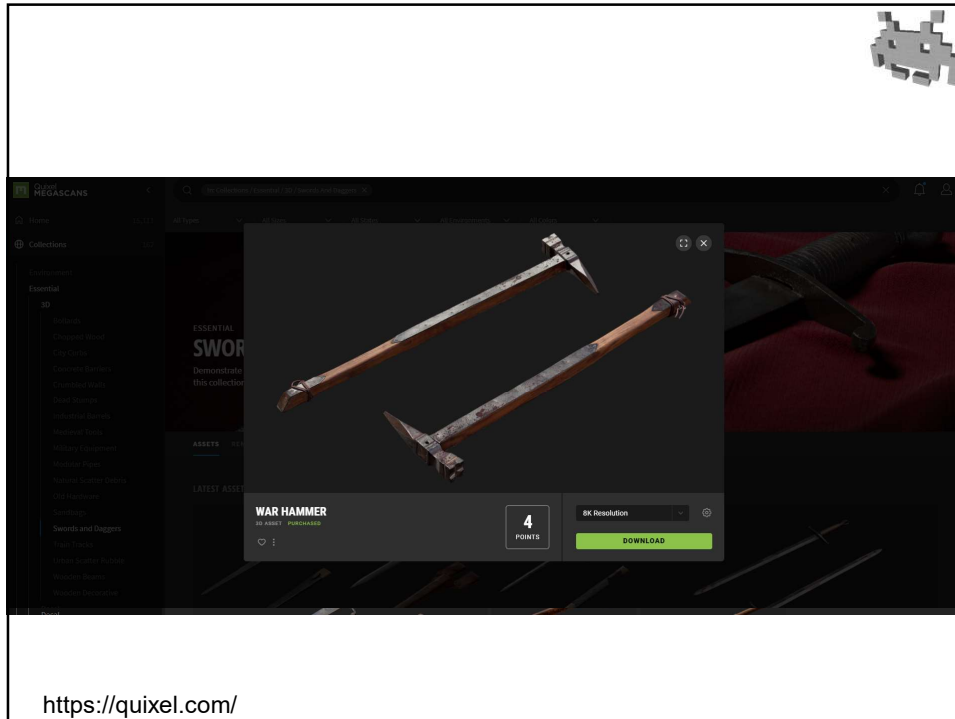
Sculptor (real)

Reale model

3D scanning

Hi res model


101



102

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?)



The image shows a Kinect sensor pointing towards a list of scanning technologies. Below the list, there are six 3D scanned human figures in various poses, with an arrow pointing from the 'Max object dimension' bullet point to them.

103

3D models sources: a comparison

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

manually edited
low-poly mesh
(2K triangles)

VS

scanned & cleaned
hi res mes
(30K triangles)

(sculpted meshes are similar)

Dino,
scanned
by artec3d

104

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

105