

3D video games

# Models for Games



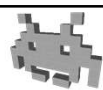
---

Marco Tarini



1

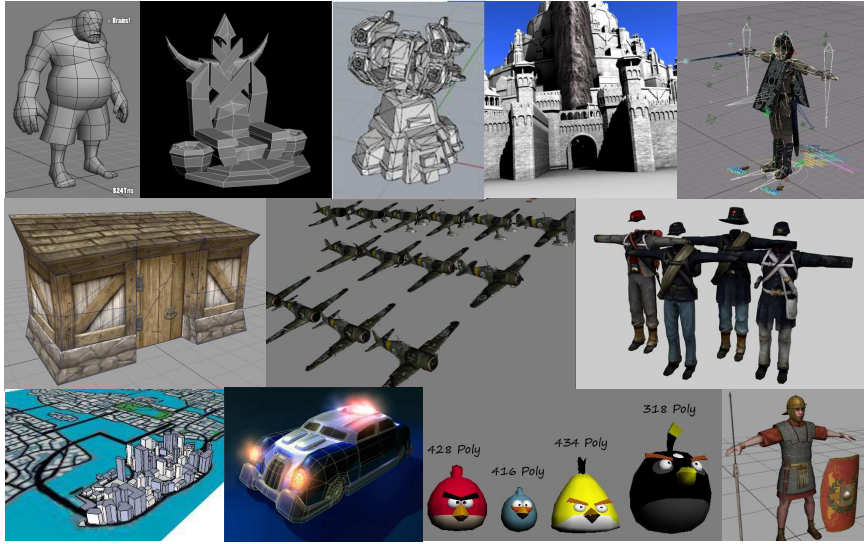
## Course Plan



- 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 ●

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

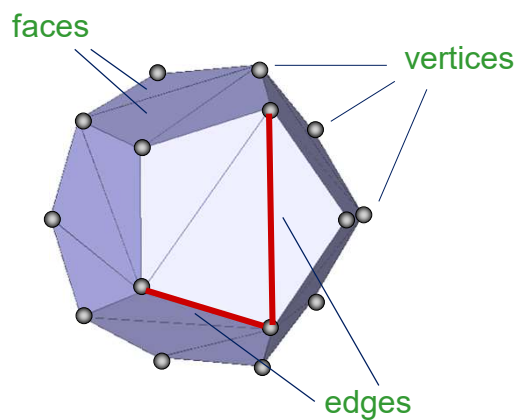
### The visual appearance of 3D objects

- Data structure for modelling 3D objects
  - GPU friendly
  - Resolution = number of faces
  - (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”

7

## Triangle Mesh (or simplicial mesh)

- A set of adjacent triangles



8

## 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, ...

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

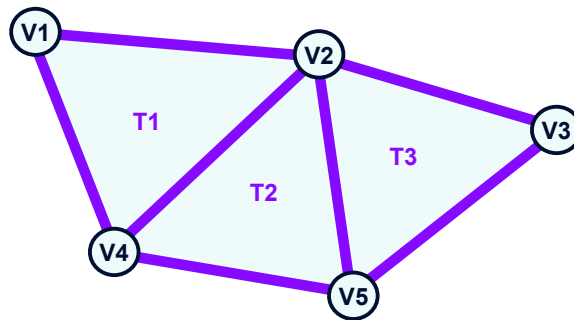
V5

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*

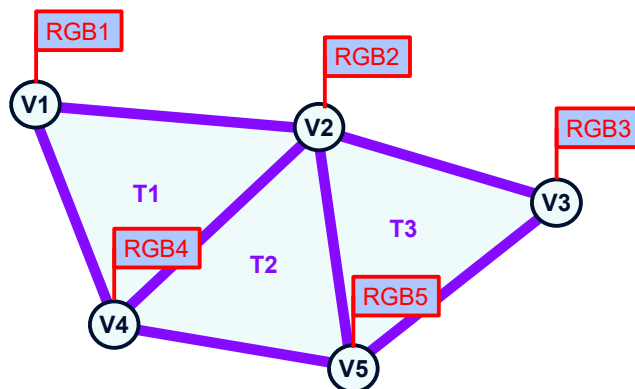


11

## Mesh: attributes



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



12

## Mesh as a data structure: “soup of triangles”

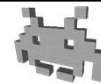


- Simply, a big array of triangles
- Each triangle stored as: a sequence of 3 vertices
- Each vertex stored as:  
its  $x,y,z$  coordinates + attributes
- Problem: data replication
  - Not memory efficient
  - Inconvenient to update  
(e.g., to animate)
  - Seldomly used

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

13

## Mesh as a data structure: indexed meshes

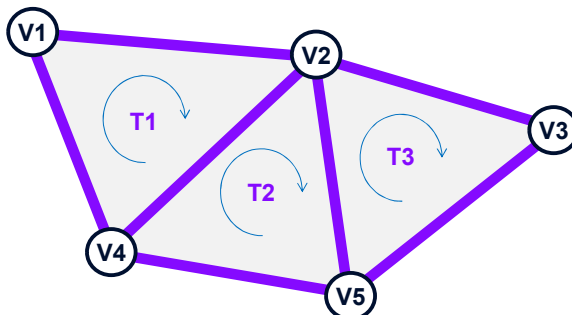


- array of vertices
  - Each vertex stored as
    - $x,y,z$  position (aka the “geometry” of the mesh)
    - attributes: (all vertices, the same ones)  
any data saved on the surface: e.g. color
- array of triangles
  - the “connectivity» (or, “topology”) of the mesh
  - Each triangle stored as
    - 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

14

## 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**

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  $\Delta$  Unreal Tournament (1999)

3000  $\Delta$  Unreal Tournament (2002)

4,500  $\Delta$  weapon

this 12,000  $\Delta$  Unreal Tournament 3 (2007)

The image shows a progression of four screenshots from the Unreal Tournament series. The first screenshot (1999) shows a character in a dark arena with a low polygon count. The second (2002) shows a character in a similar arena but with more detail. The third (2007) shows a character in a more complex environment with a higher polygon count. The fourth (2007) shows a character in a highly detailed environment with a very high polygon count. Yellow arrows point from the text labels to the corresponding screenshots.

19

### Resolution increases over time

230  $\Delta$  (1996)

300  $\Delta$  (1998)

4.000  $\Delta$  (2002)

30.000  $\Delta$  (2008)

48.000  $\Delta$  (2012)

The image shows a row of seven Lara Croft models from different Tomb Raider games, arranged from left to right in chronological order. Each model is shown in a similar pose, holding a handgun. The models become increasingly detailed and more realistic in appearance from left to right. Red text labels below each model indicate the polygon count and the year of the game.

21



## Mesh attributes: in general (valid for all attributes)



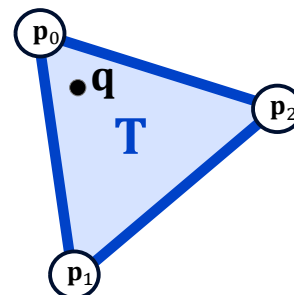
- Any properties stored on the mesh, varying on the surface
  - Can be made of vectors, versors, or scalars
- Stored at each vertex
  - Each vertex of a mesh = same collection of attributes
- It's interpolated inside the faces
  - Linear interpolation: uses barycentric coordinates (see next slides)
-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



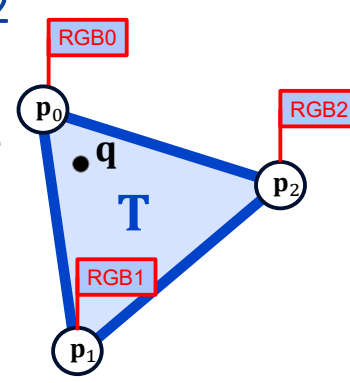
$$\mathbf{q} = k_0 \mathbf{p}_0 + k_1 \mathbf{p}_1 + k_2 \mathbf{p}_2$$

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

23

### Interpolation of vertex attributes inside mesh triangles 1/2

- Now assign three attributes to the three vertices
- A point  $\mathbf{q}$  in  $\mathbf{T}$  with barycentric coordinates  $k_0, k_1, k_2$  is implicitly assigned the attribute


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

per vertex

24

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

- 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

in local space!


see lecture on textures (later)

see lecture on normal maps (later)

see lecture on animations (later)

25

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



- *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

26

## Mesh as tables



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

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

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   |   |   |   |    |    |    |   |   |   |   |   |   |    |    |    |    |    |    |

GEOMETRY + ATTRIBUTES

27

## Mesh attributes: colors



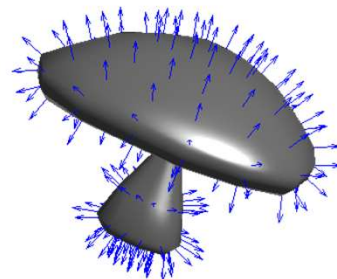
- In games, colors on 3D models are usually determined by textures (not by mesh colors)
  - reason: more resolution in signal
- Per vertex colors can be used...
  - To cheaply add variations models
    - Red guards, blue guards SEE RENDERING LATER
  - To **bake** lighting
    - e.g. baked per-vertex 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
  - ...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

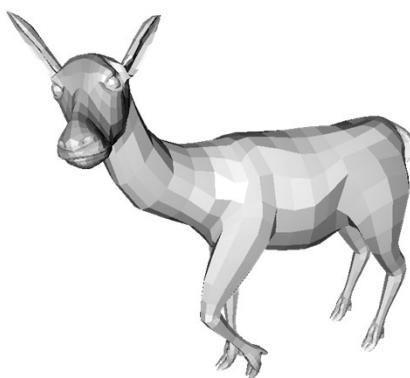
## Mesh attributes: normals



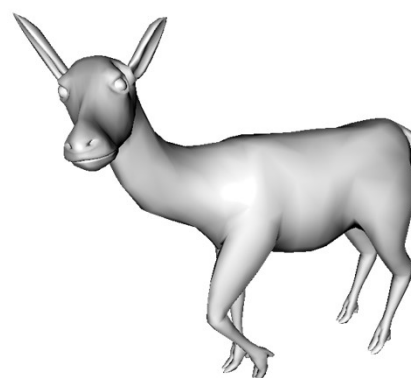
- Technically, mesh faces are *flat*
  - the normal is constant over a face
  - the normal is discontinuous across faces (each mesh edge is “sharp”)
- Usually, that’s not the surface we intend to represent
  - The flatness is just an artifact (a defect) of the mesh discretization
- By using a *continuously 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)

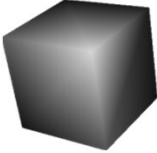
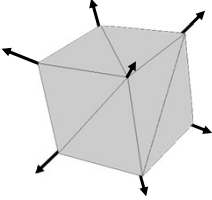
Note: normals are made visible to our eyes due to lighting  
(computation of how light reacts with the surface)

31


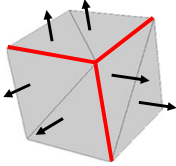
## Hard edges (aka sharp edges) (aka “creases”)

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

Soft edges:



Red edges are hard




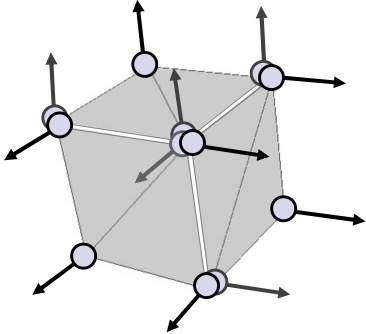
- How to encode (Co) a **discontinuity** in any attributes?

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      | N <sub>x</sub> | N <sub>y</sub> | N <sub>z</sub> |
|----|--------|--------|--------|----------------|----------------|----------------|
| 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        |

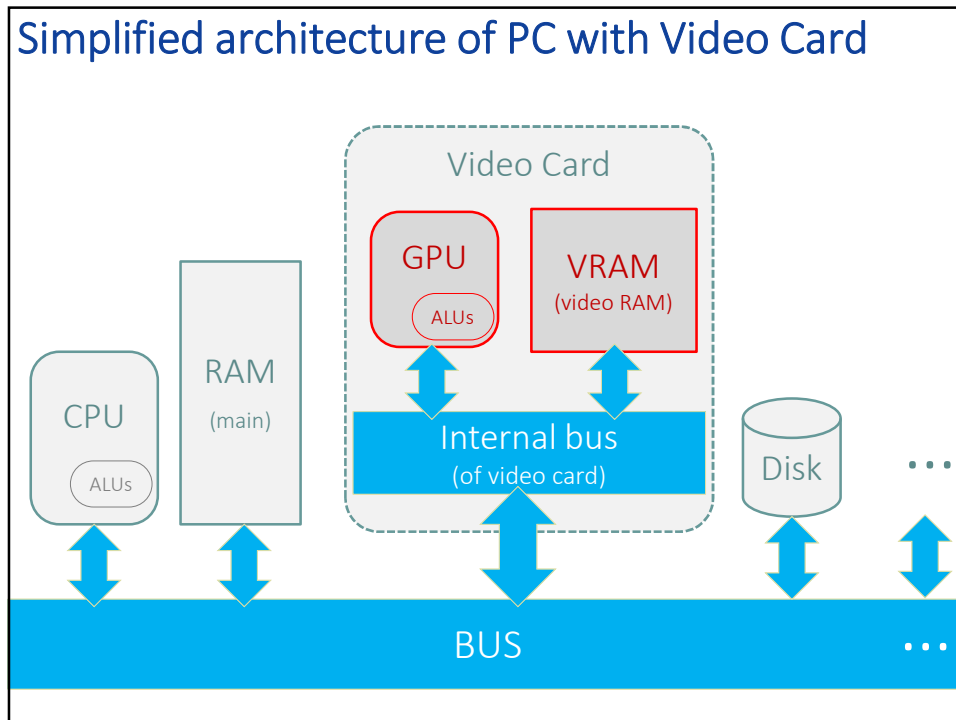
CONNECTIVITY

34

## Rendering of a Mesh in a nutshell

- Load...
  - get required data ready on GPU RAM
    - Geometry + Attributes table
    - Connectivity table
    - Textures
    - Shaders
    - Parameters / Settings
- ...and Fire!
  - send the "Draw-call" to the GPU
  - using an API

35



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)
- 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

Might change in the future?

PER VERTEX PHASE  
PER TRIANGLE PHASE  
PER PIXEL PHASE

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:  
 $O(\text{number of vertices}) = O(\text{number of faces})$   
but also,  $O(\text{number of covered pixels})$  --- 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 (or triangle soup)
  - (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

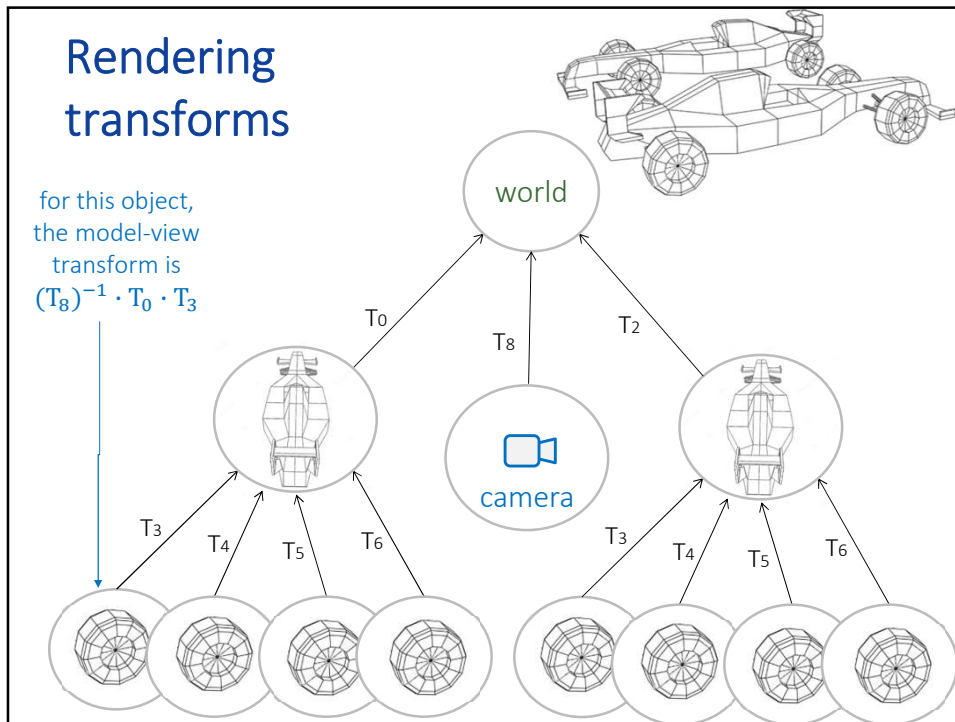
38

## Rendering & Scene graph

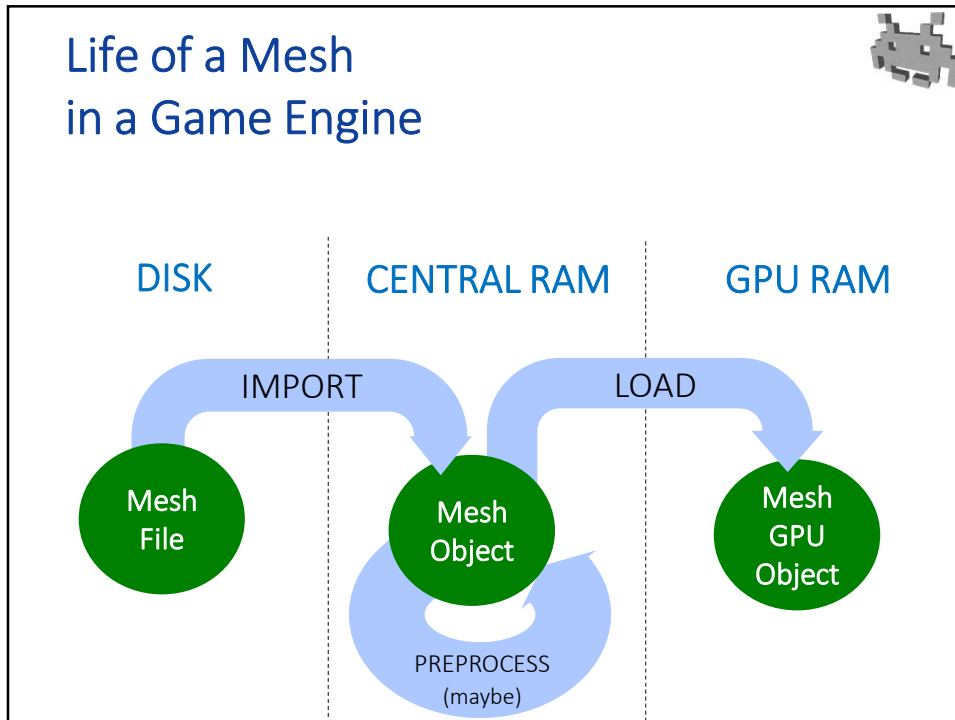


- Rendering APIs encode transforms as a 4x4 matrix
  - reason: it is a more flexible, can also express perspective transforms
- To render an object:
  - Combine its Transforms from Object-space to Camera-space (“model-view transform” – in CG terminology)
  - Convert it into a 4x4 matrix
  - Use it during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



40



43

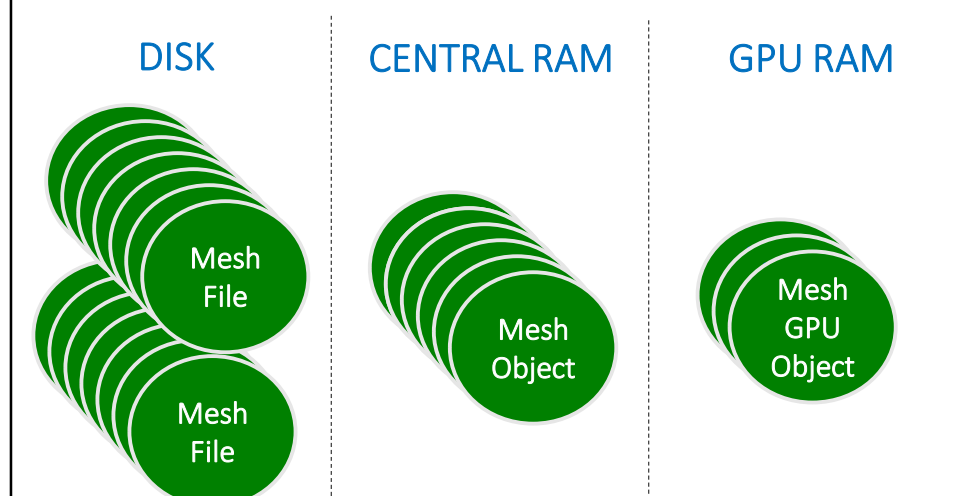
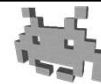
## Life of a mesh in a game engine



- **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

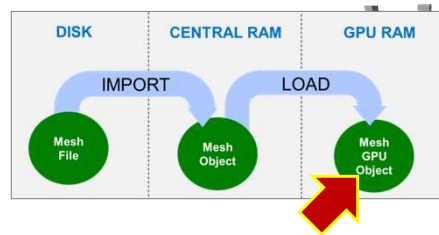
44

## Memory Management (during game execution)



45

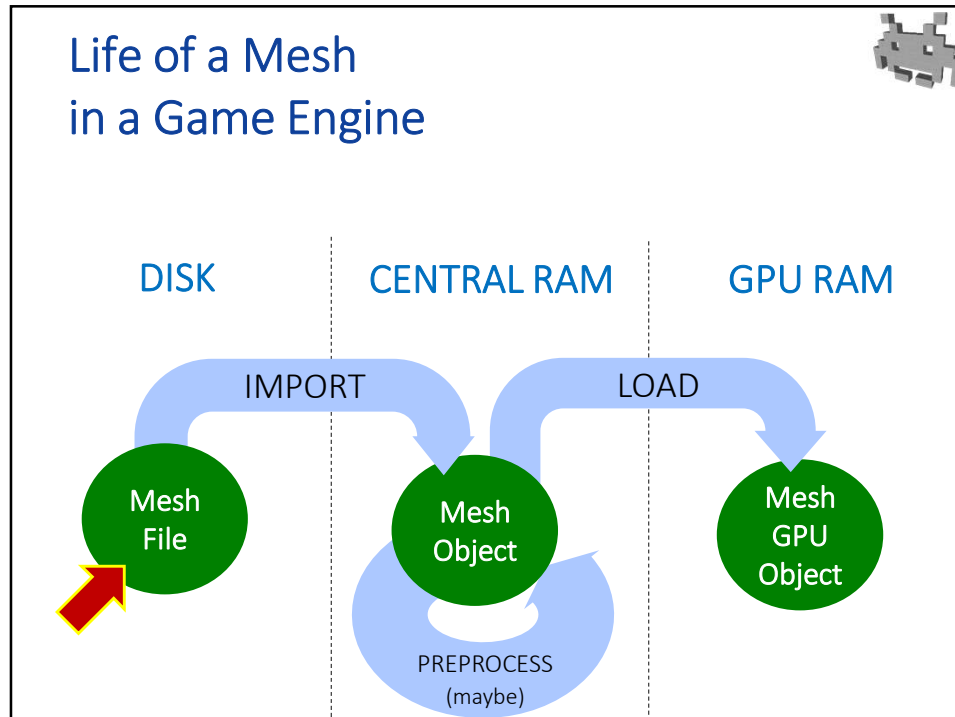
## Mesh GPU Object (on Graphic Card)



- 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

46

## Life of a Mesh in a Game Engine



47

## Mesh as an asset

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

48

## 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 (where 12 is # faces and 10 is # edges)

x,y,z 2nd vertex: 3 0 0

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

49

## 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** - Rhinoceros 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** - Multigen 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

51

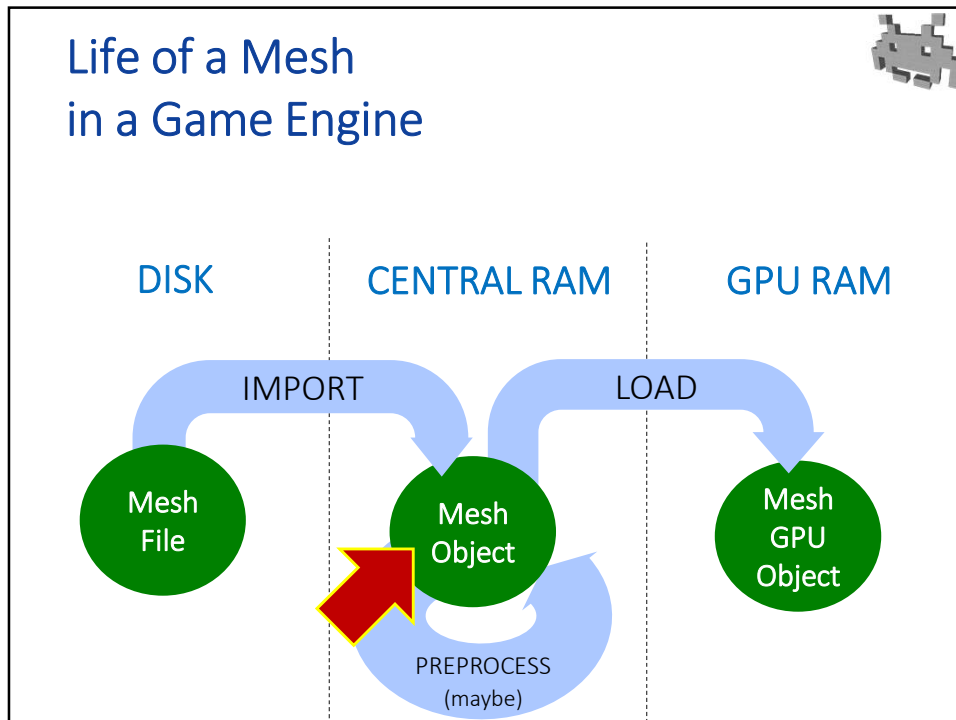
## Most used mesh file formats

(most used in games)



|             |  |  |
|-------------|--|--|
| most common | <ul style="list-style-type: none"> <li>● <b>.OBJ</b> (wavefront) <ul style="list-style-type: none"> <li>⊙ max diffusion</li> <li>⊙ indexed, normals, uv-mapping</li> <li>⊙ no colors (only material index for face)</li> <li>⊙ no skinning or animations</li> </ul> </li> <li>● <b>.SMD</b> (VALVE) <ul style="list-style-type: none"> <li>⊙ Skeletal animation + skinning</li> <li>⊙ normals, uv-mapping</li> <li>⊙ no indexed!</li> <li>⊙ no colors</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>● <b>.3DS</b> (AUTODESK) <ul style="list-style-type: none"> <li>⊙ YES: colors, uv-mapping, indexed, materials, textures...</li> <li>⊙ NO: normals</li> <li>⊙ limited by vertex number (64K)</li> </ul> </li> <li>● <b>.DAE</b> (collada: SONY + KHRONOS) <ul style="list-style-type: none"> <li>⊙ complete</li> <li>⊙ Born for being interchanged</li> <li>⊙ open standard</li> <li>⊙ Almost impossible to parsing it completely</li> </ul> </li> </ul> |
| less common | <ul style="list-style-type: none"> <li>● <b>.MD3</b> (Quake, IDsoft) <ul style="list-style-type: none"> <li>⊙ vertex animations, normals</li> <li>⊙ no colors</li> </ul> </li> <li>● <b>.PLY</b> (cyberware) <ul style="list-style-type: none"> <li>⊙ customizable</li> <li>⊙ "academic"</li> </ul> </li> </ul>  | <ul style="list-style-type: none"> <li>● <b>.FBX</b> (AUTODESK) <ul style="list-style-type: none"> <li>⊙ complete, with animations</li> <li>⊙ complex, hard to parse</li> </ul> </li> <li>● <b>.GLFW</b> (opensource) <ul style="list-style-type: none"> <li>⊙ very complete, and customizable</li> <li>⊙ includes animations, etc</li> </ul> </li> </ul>  |
|             | <div style="display: flex; justify-content: space-between; width: 100%;"> <span>simple</span> <span>complex</span> </div>  |  |

52



53

### Mesh Object (in RAM)

This diagram is a smaller version of the one on slide 53, but it focuses on the 'Mesh Object' in the CENTRAL RAM section. A red arrow points to the 'Mesh Object'.

- 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)

54

## How to represent a mesh? (which data structures)



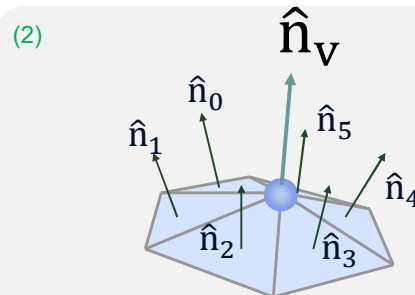
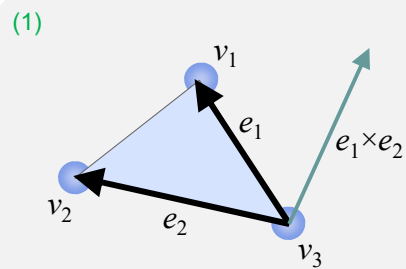
- Indexed mode in C++ :

```
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 */  
};
```

55

## Computing normals from geometry

- (1) compute normals of faces
- (2) compute normals of vertices



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

58



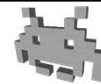
## Mesh processing: (or, more in general, Geometry Processing)



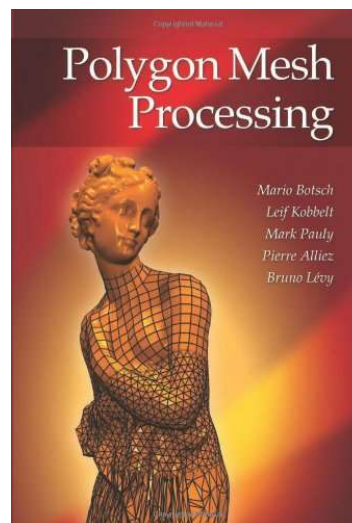
- The algorithm above (for the computation of per vertex normal) is a tiny example of processing done over a mesh
- **Mesh processing**: the discipline of creating, transforming, computing meshes
  - inputs and/or outputs are meshes
- Part of, geometry processing:
  - when the input and output are other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

59

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






- A good introduction to mesh processing








60



## Libraries for mesh processing



 **VCG-Lib**  
vision and computer graphic library  
CNR (  )


 **CGAL**  
computational geometry algorithms library  
INRIA (  )

 **OpenMesh**  
+  **OpenFlipper**  
RWTH (  )

 **libigl**  
simple geometry processing library  
NYU (  )

61

## 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

62