

3D video games

Models for Games

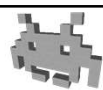


Marco Tarini



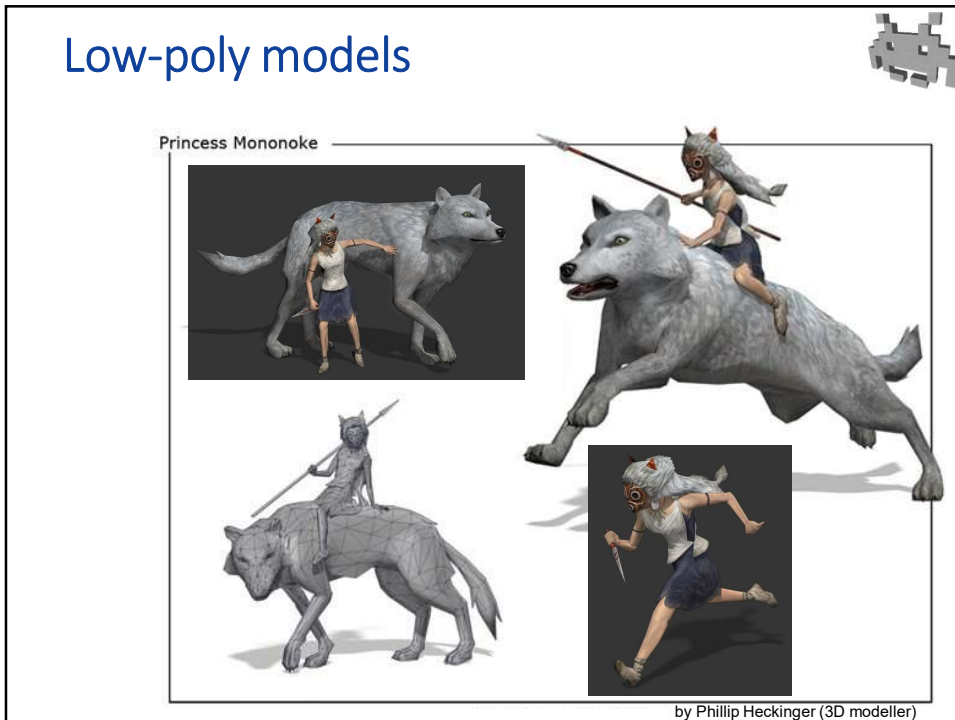
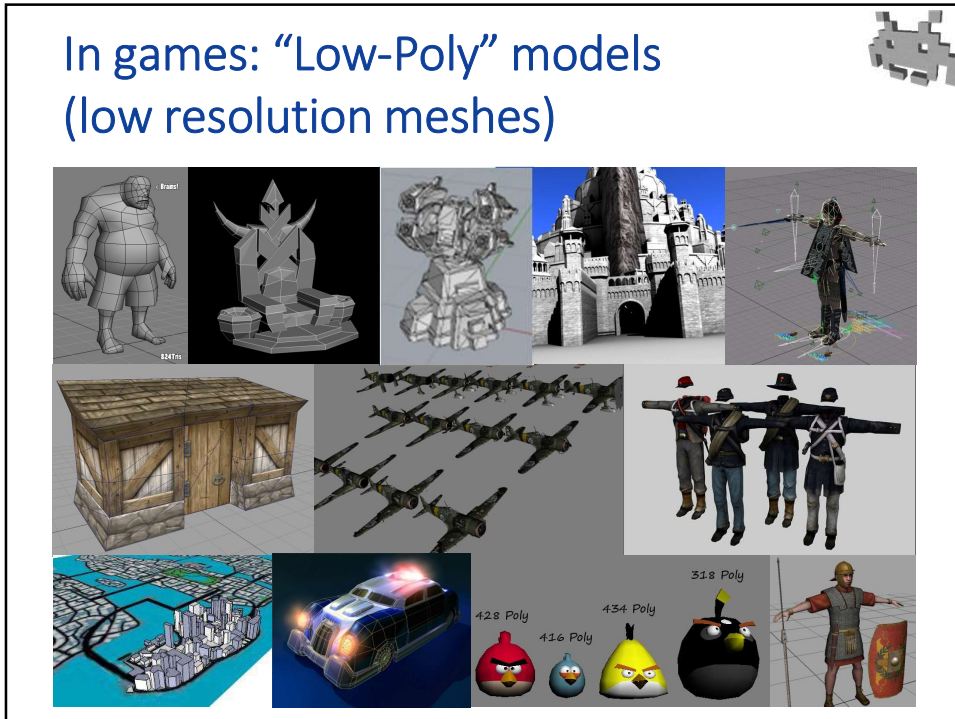
1

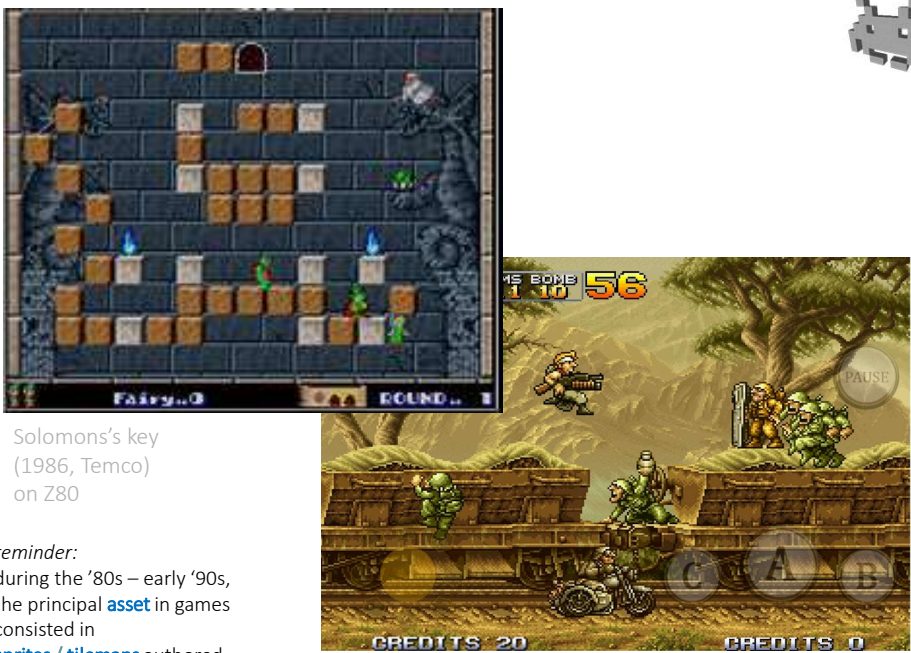
Course Plan



- 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. 8: **Game 3D Animations** ●●●
- lec. 9: **Game 3D Audio** ●
- lec. 10: **Networking** for 3D Games ●
- lec. 11: **Artificial Intelligence** for 3D Games ●
- lec. 12: **Game 3D Rendering Techniques** ●

2





Solomons's key
(1986, Temco)
on Z80

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

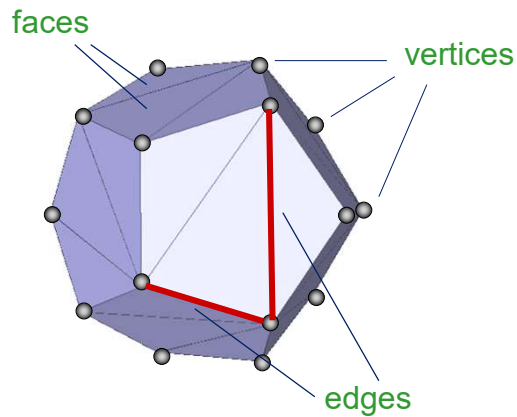
The visual appearance of 3D objects

- 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!

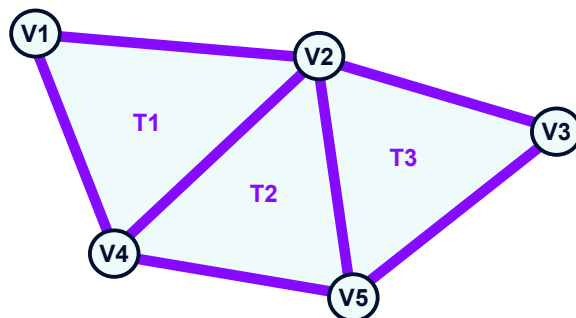


10

Mesh: connectivity (or topology)



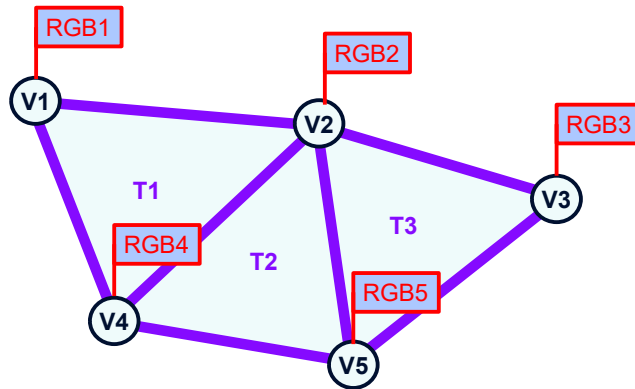
- Faces: triangles connecting vertices
 - More in general, polygons,
 - connecting triplet of *vertices*
 - just as, in a graph, *nodes* are connected by *edges*



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




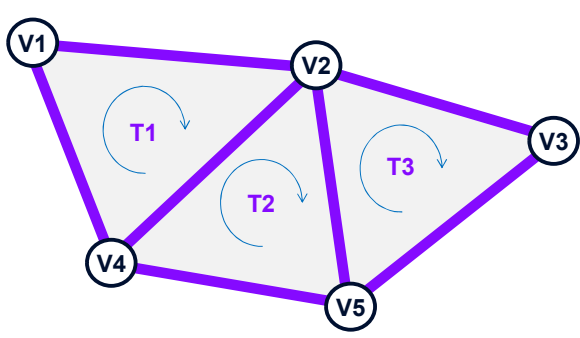
- array of vertices
 - Each vertex stored as
 - x,y,z position (aka the “geometry” of the mesh)
 - attributes: (all vertices, the same ones)
any data saved on the surface: e.g. color
- array of triangles
 - the “connectivity» (or, “topology”) of the mesh
 - Each triangle stored as
 - triplet of **indices** (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 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 (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 C^0 continuous across faces
 - but C^1 discontinuous across faces
 - and C^∞ inside faces

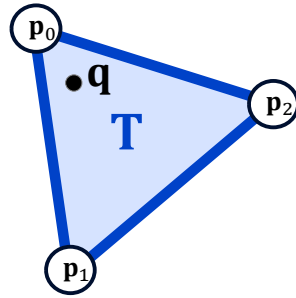
22

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}



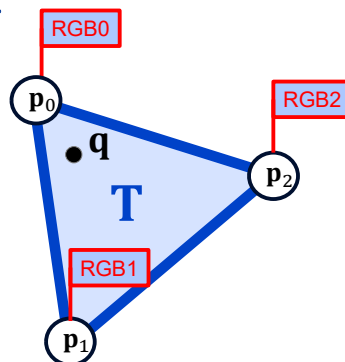
23

Interpolation of vertex attributes inside mesh triangles 1/2

- 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

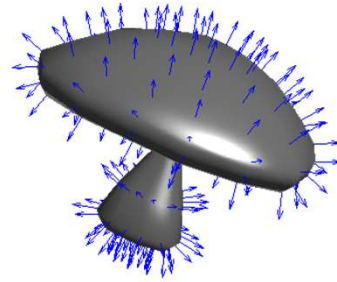


- In games, colors on 3D models are usually determined by textures (not by mesh colors)
 - reason: more resolution in signal
- Per vertex colors can be used...
 - To cheaply add variations models
 - Red guards, blue guards SEE RENDERING LATER
 - To **bake** lighting
 - e.g. baked per-vertex ambient occlusion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

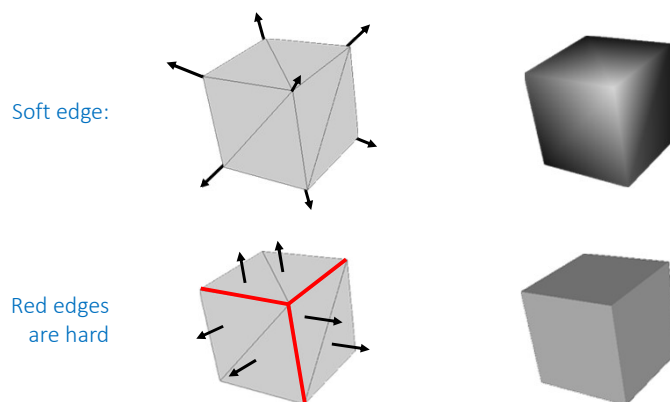
- A versor
- Representing the surface orientation
- Main use: lighting computation
- Can be computed automatically from geometry...
- But it is a part of the mesh assets:
 - the artist is in control of which edges are *soft* and which are *hard*



29

Hard edges (aka “creases”)

- Edges where the normal is *not continuous*.



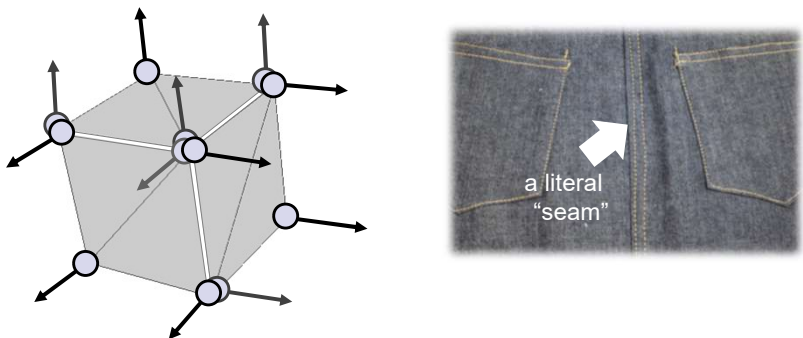
- How to encode (Co) a *discontinuity* in the attributes?

30

answer:

Vertex seams

- Vertex seam = two coincident 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 white circle, indicating a vertex seam where two coincident vertices share the same position but have different attributes. To the right, a photograph of a denim jacket shows a vertical seam, with a white arrow pointing to it and the text 'a literal "seam"'. A small 3D model of a character is in the top right corner.

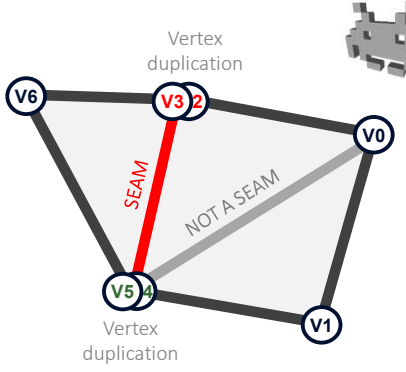
31

Vertex seams

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

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

GEOMETRY + ATTRIBUTES



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


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

CONNECTIVITY

32

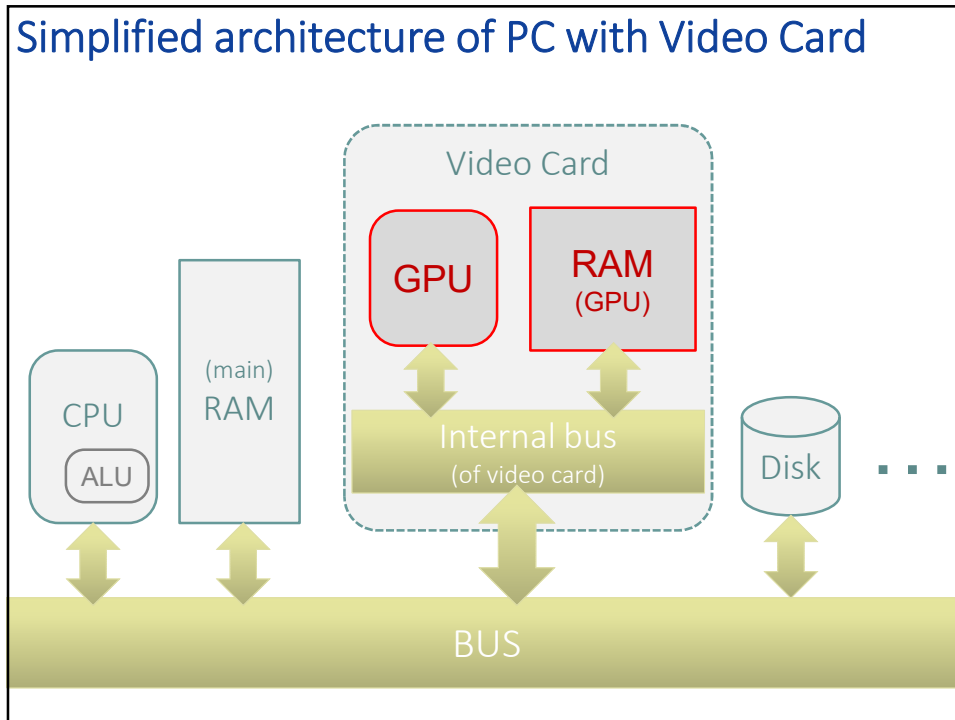
Rendering of a Mesh in a nutshell

- 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



The diagram shows two blue arrow-shaped boxes pointing to the right. The top one is labeled 'THE MESH' and the bottom one is labeled 'THE "MATERIAL"'. A horizontal dashed line separates the two boxes, with a dotted line extending from the 'Connectivity table' item in the list above to this dashed line.

33



34

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

35

Rendering of a Mesh in a nutshell

- 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 GPU, many things are **hard-wired**
 - The data structures for the mesh are (indexed meshes or triangle soup)
 - Only triangles as supported for faces
 - Attributes are automatically interpolated inside face
- There's a bit of customizability because GPU can be programmed
 - Both the per-vertex phase (projection) and the per-pixel phase (lighting)
 - “Shader” = custom program

Exception:
semi-transparent
“see through”
objects

36