3D video games

Models for Games

Marco Tarini

1

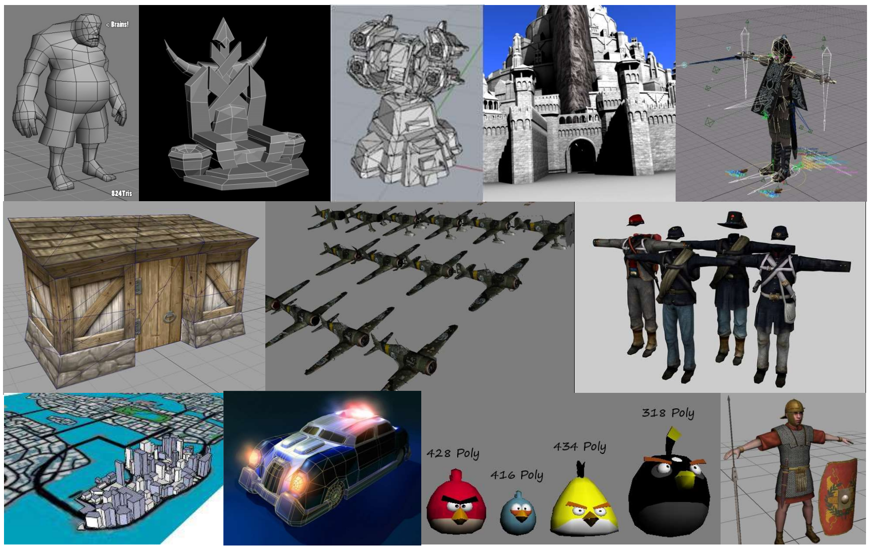
Course Plan

- 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 Materials
- lec. 9: Game Textures
- lec. 8: Game 3D Animations
- lec. 10: 3D Audio for 3D Games
- lec. 11: Networking for 3D Games
- lec. 12: Artificial Intelligence 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

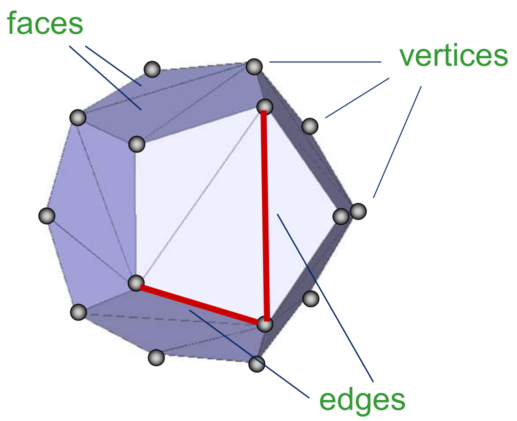
Triangle Meshes:
the visual appearance of 3D objects

- 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

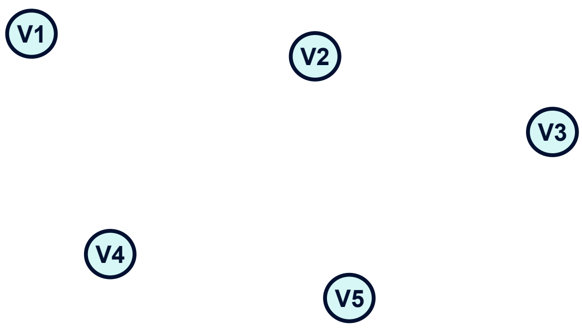
A mesh consists of

- 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

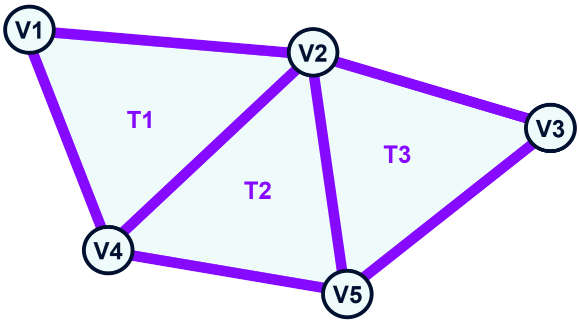
- Set of vertices
  - A position vector (x,y,z) for every vertex
  - Coordinates, by definition, are given in Local space!



10

Mesh: connectivity (or topology)

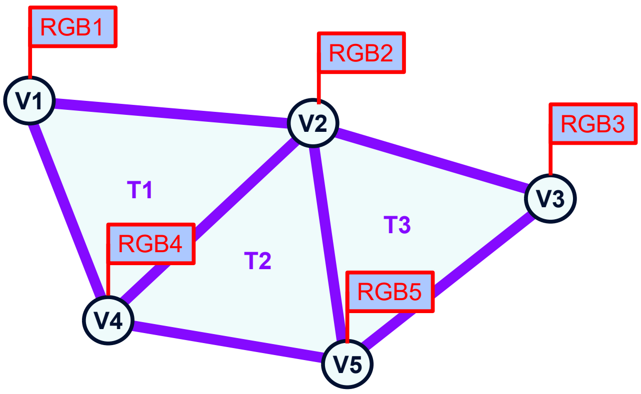
- Faces: triangles connecting vertices
  - More in general, polygons,
  - connecting triplet of *vertices*
  - just as, in a graph, *nodes* are connected by *edges*



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

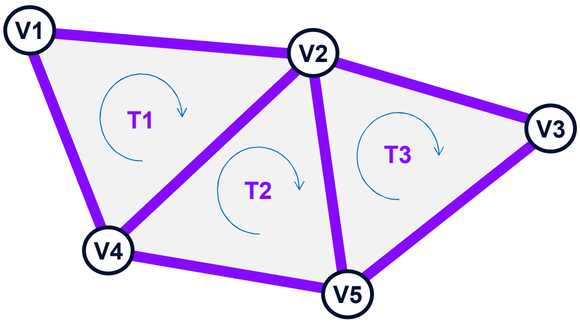
we can consider positions as attributes too

- 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

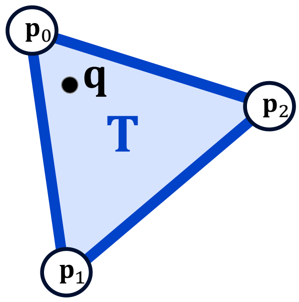
22

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

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

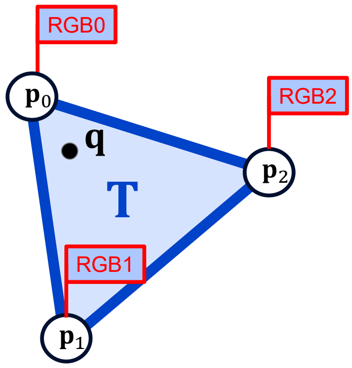
- k_0, k_1, k_2 are called the barycentric coordinates of $\mathbf{q}$ in $\mathbf{T}$



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



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

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 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

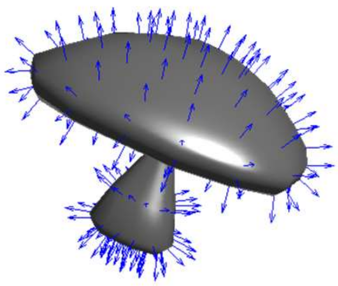
Mesh attributes: colors

- 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*



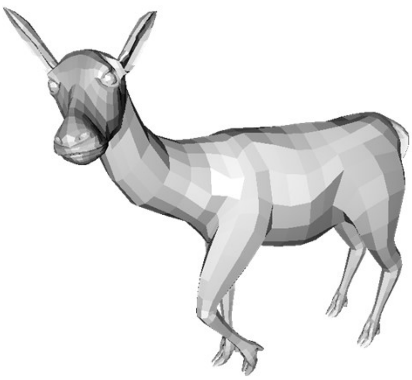
29

Mesh attributes: normals

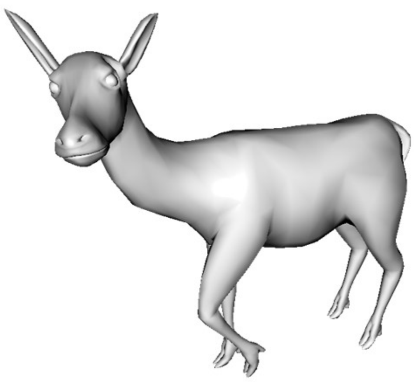
- Technically, mesh faces are *flat*
  - the normal is constant over a face
  - the normal is discontinuous across faces (each mesh edge is “sharp”)
- Usually, that’s not the surface we intend to represent
  - The flatness is just an artifact (a defect) of the mesh discretization
- By using a *continuously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

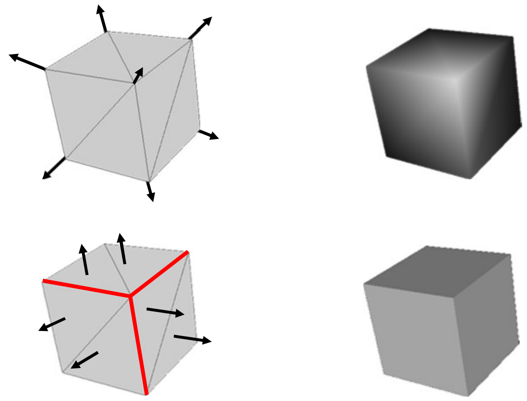
31

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

- Edges where the normal is not continuous .

Soft edges:

Red edges are hard



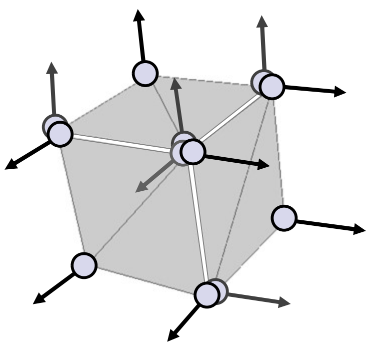
- How to encode (C_0) a discontinuity in any attributes?

32

answer:

Vertex seams

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

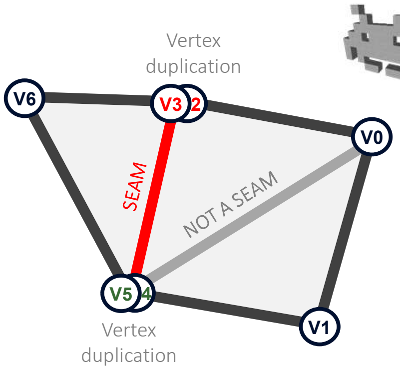


33

Marco Tarini
Università degli studi di Milano

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	s a	m e	a s	n_x3	n_y3	n_z3
V4	p_x3	p_y3	p_z3	n_x4	n_y4	n_z4
V5	s a	m e	a s	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

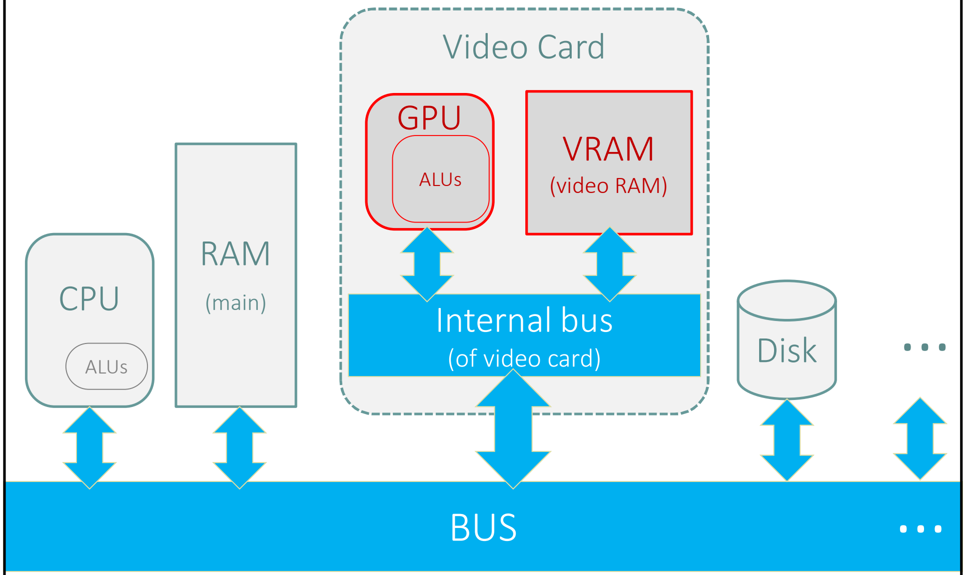
THE "MATERIAL"

…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

Might change in the future?

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

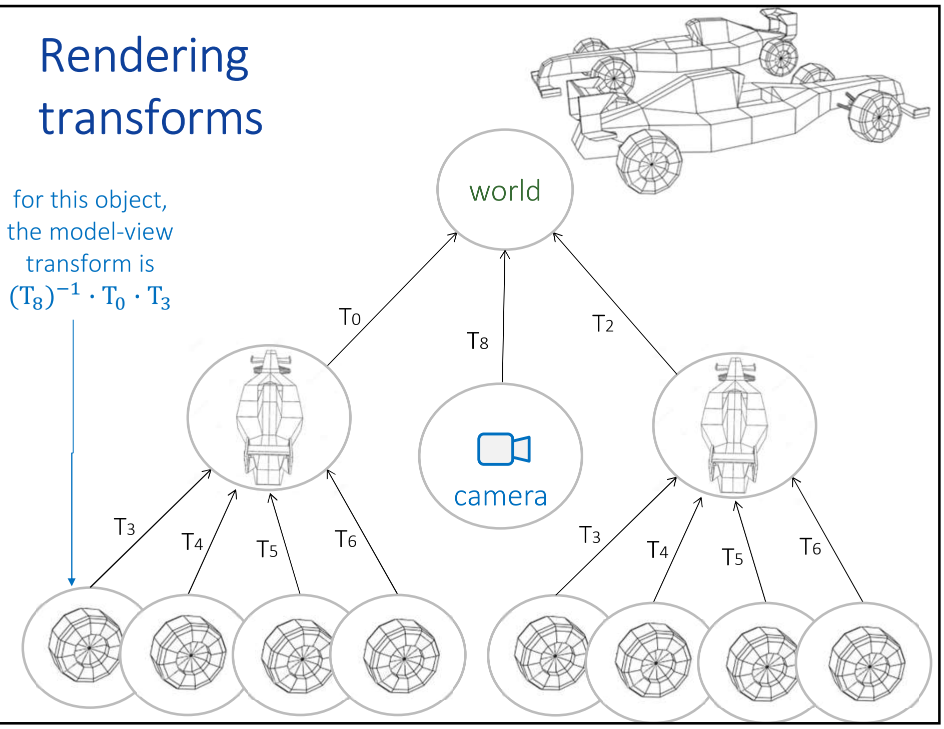
38

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



40

Rendering order of the meshes - notes
(and particle effects too)

- Idea 1: depth-first visit of the scene graph
  - Advantage: incremental update of the global transform (used by rendering)
  - Cumulate local transform when going down, popping or cumulate inverse when going back up
  - Not a big advantage, after all – popular in CG, not much used in games
- Idea 2: render meshes according to the material they use: "load a material, then render all meshes that use that material"
  - In the example above: render all 8 wheels consecutively
  - Advantage: consistent rendering state (material = state of the renderer)
  - Remember: all data needed by the rendering must reside in GPU ram
  - Very popular method
- Idea 3: sort meshes front-to-back (Z-order in camera space)
  - Render meshes from the back
  - Correct order is needed of semitransparent objects
  - Problem: expensive to sort – not much used
  - Instead, the correct order of semi-trasnsparent objects tweaked with tricks (postponing/preponing the rendering for certain materials – "render order")

Known as "painter's algorithm"

42