Games and Simulation

Fernando Birra

Scene Management

Why not simply draw everything?

- On current graphics cards, a complete Quake III level can be drawn using brute a force approach!

- Why isn’t this good enough?

- Because current games are even more demanding with current graphics cards than Quake III was at its time

- Players (and Editors) will keep wanting more and more…

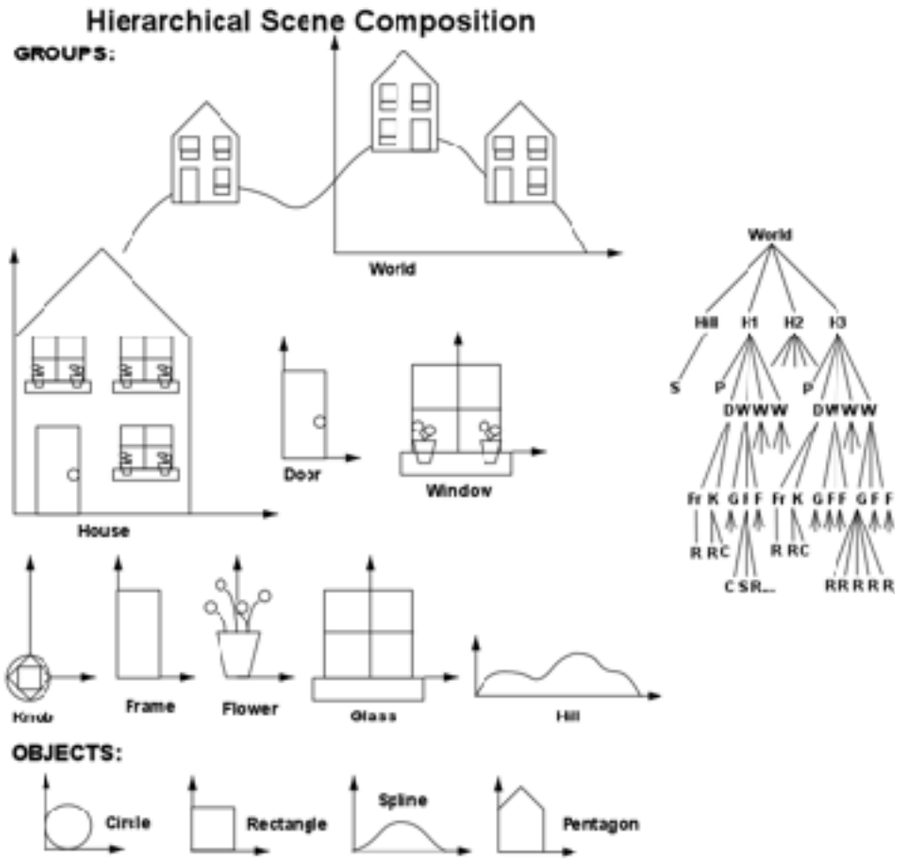
Scene Management

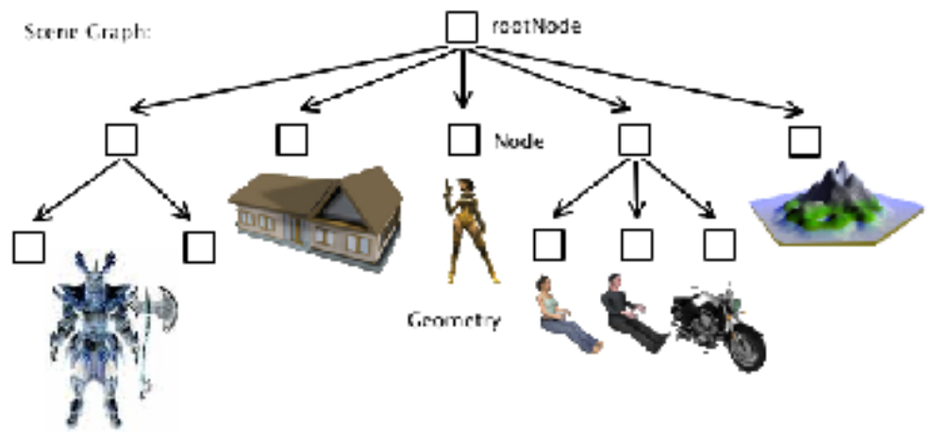
- **Goals**:
  - Select the subset of polygons that are needed to render the image corresponding to a certain location and orientation of the viewer
  - Execute other tasks such as collision detection in an efficient way
- **Tools**:
  - **Scene Graph**: A data structure that holds the entire scene and the relationships between its components
  - **Techniques**: An array of helpful algorithms that will reduce the scene complexity (**Culling, Level of Detail**, …)

Scene Graph

- We have a set of 3D models, probably coming from varied sources
- We want to put them together in a scene
- Models are surely transformed as they're placed in the scene (translation, scaling, rotation)
- Typically, each model is placed and adjusted in reference to another model
- Distinct models are grouped together (ex: a table and 4 chairs, all the bullets in the scene, etc.)
- All the objects in the scene graph (real or abstract) are called Spatials

Scene Graph


Hierarchical Scene Composition
GROUPS:
World
World
House
Door
Window
Knob
Frame
Flower
Glass
Hill
OBJECTS:
Circle
Rectangle
Spline
Pentagon
Hill H1 H2 H3
S P P
DWWW DWWW
R RC R RC
CSR...
RRRR


Scene Graph:
rootNode
Node
Geometry

Spatial

- Each model:
  - was created independently
  - is positioned in the scene with respect to a “parent” model (example: table in a room, chair near table, plate on table, fork and knife on each side of plate, …)
- Each model has a local transformation that defines its placement, size and orientation with respect to its parent
- Has a (cached) world transformation that determines its final location, size and orientation in the world.
- A Spatial keeps a reference to its parent, stores its local transformation and its world transformation
- A Geometry is a visible Spatial (one that represents a visible object)

Geometric Transformations

- Elementary Geometric Transformations (Scaling, Rotation and Translation) can be concatenated together to form a complex transformation

- Order is important (Non commutative)

- Using homogeneous coordinates allows to represent arbitrary sequences of transformations:

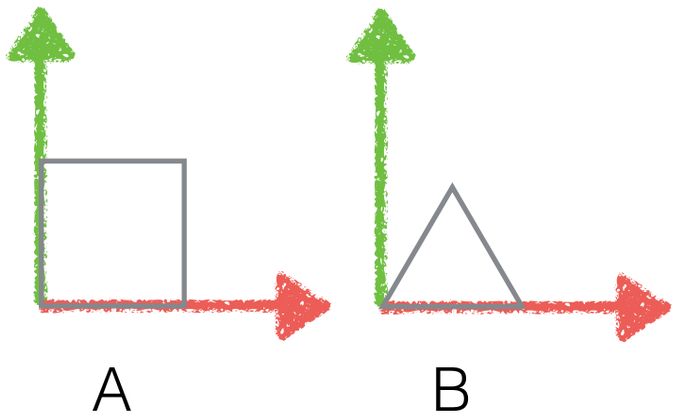
$$M = T_n \ldots T_2 T_1 P$$

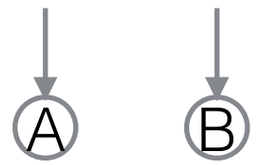
$P=[x\ y\ z\ w]^T$ is a vector column (in homogeneous coordinates) representing the point (x,y,z).
T_i is an simple geometric transformation (T, R or S)

- Some systems limit/favour general transformations to/in the order (Scale-Rotate-Translate):

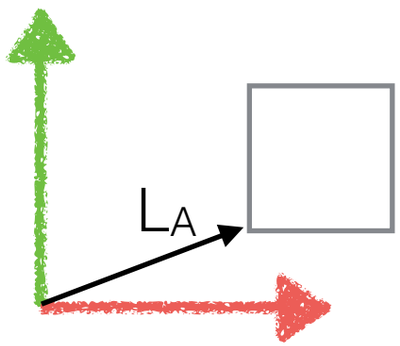
$$M = T.R.S$$

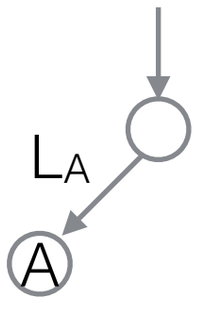
Due to our convention on representing points as vector columns, S is applied first and T is applied last





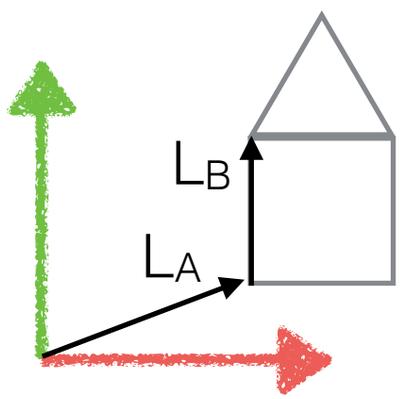
Each model in its own coordinate system

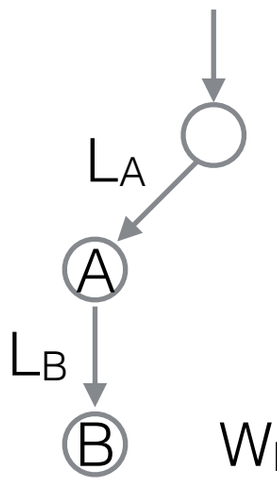




$W_A=L_A$

Placing A in the world with local transformation L_A





$W_B=W_AL_B=L_AL_B$

Adding B as a child of A with local transformation L_B. World transformation for B, W_B is the concatenation of all transformations up to the root.

Geometry: Spatial

- Vertices, stored in arrays (v[i], $0 \leq i \leq n$)

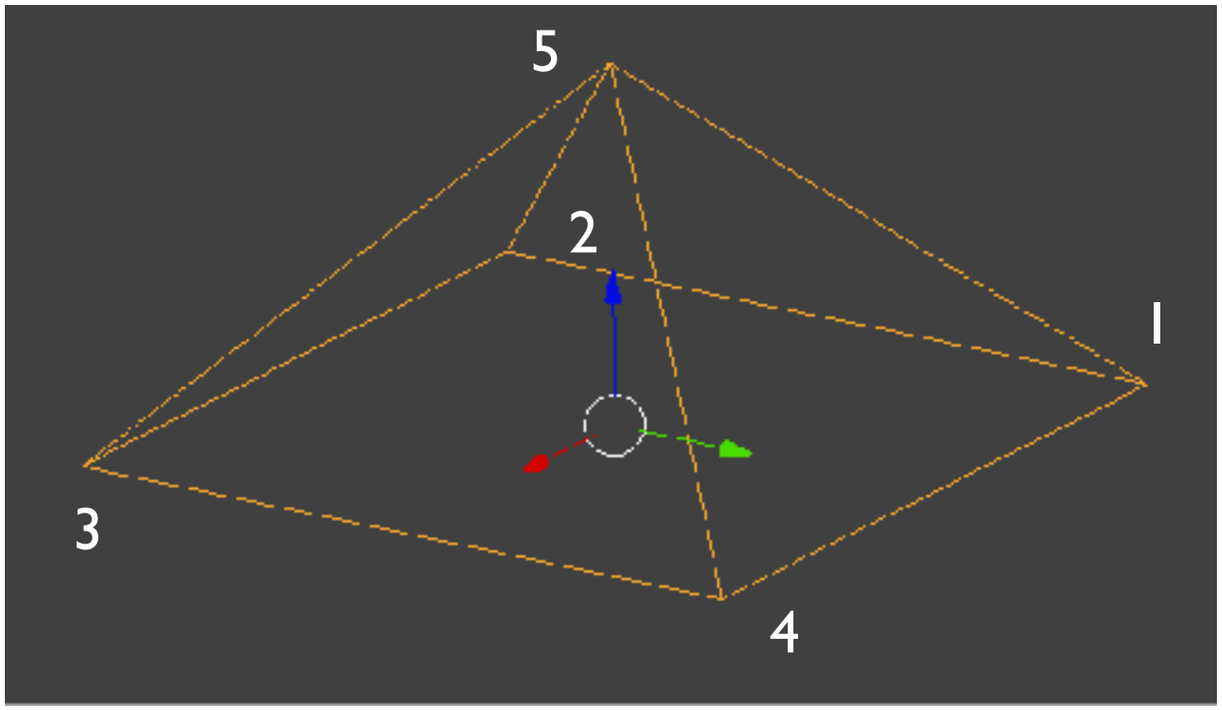
- Polygons, as indices to vertices, specified in a consistent way that let us distinguish between inside and outside faces. Ex: (i_0, i_1, i_2); (i_2, i_3, i_4)

- Vertex normals for dynamic lighting (modelled vs. procedurally computed)

- Bounds (A simplified volume that encloses the geometry), typically an AABB or a sphere (modelled vs. computed)

- Units of the model (Scale)

Topology and Geometry



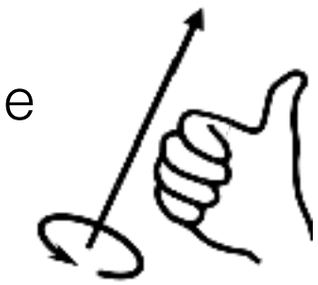
Topology

Faces	
1	1, 5, 4
2	5, 1, 2
3	5, 2, 3
4	5, 3, 4
5	1, 4, 3, 2

Geometry

Vertices	
1	(1,0,-1)
2	(-1,0,-1)
3	(-1,0,1)
4	(1,0,1)
5	(0,1,0)

- We are looking at the outside of a face if when follow the vertices of the polygon specification we see them in counterclockwise direction (right hand thumb rule)

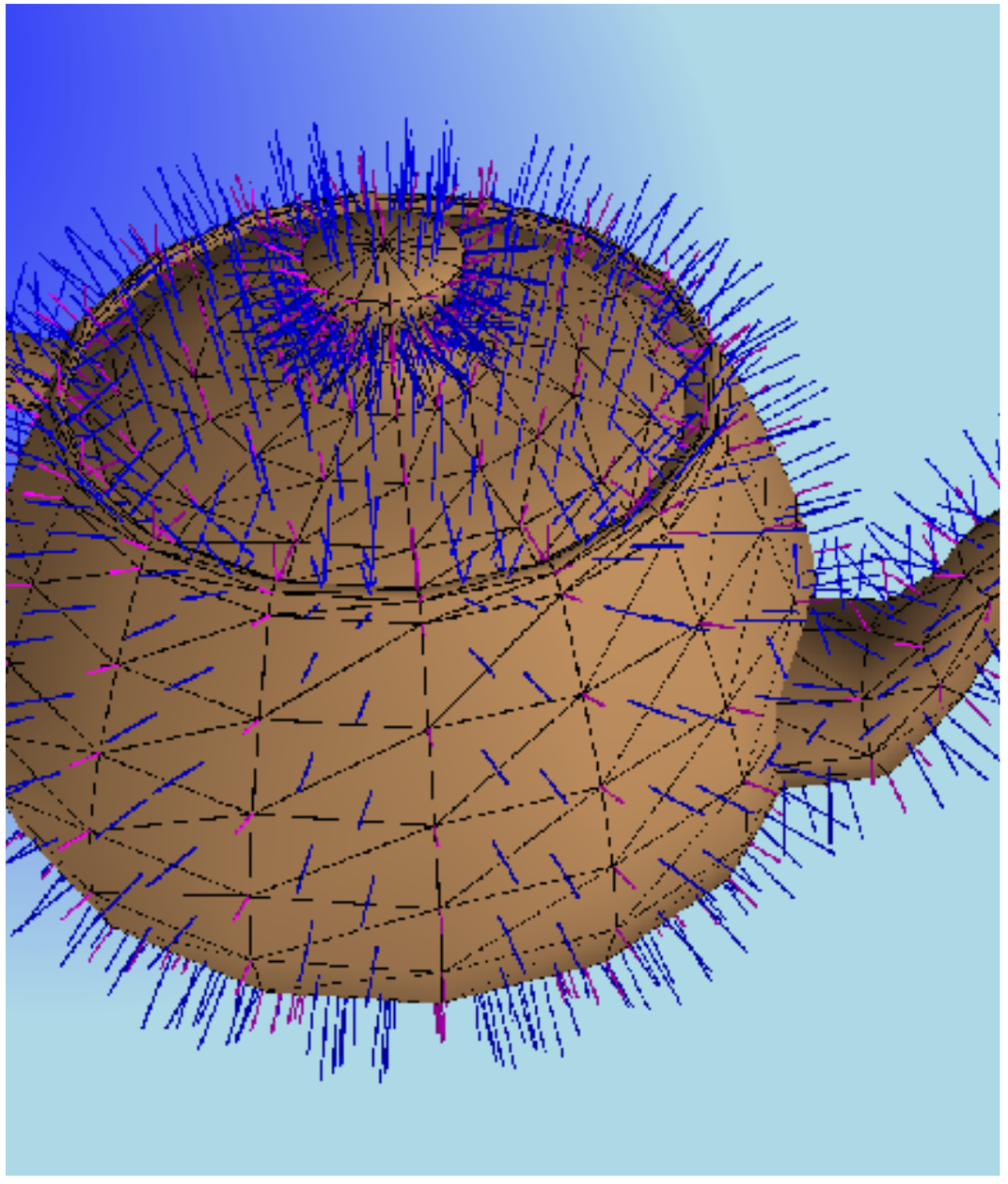


- It is better to break down general polygons into triangles (face 5 needs to be split) => simpler data structure and hardware support

Faces	
1	1, 5, 4
2	5, 1, 2
3	5, 2, 3
4	5, 3, 4
5	1, 4, 3
6	1, 3, 2

vertex normals

- Vertex normals are important for dynamic lighting
- The 3d Model may already have vertex normals or…
- … They can be computed from face normals
- Face normals are easily computed by cross product of 2 vectors along 2 edges of a triangle.



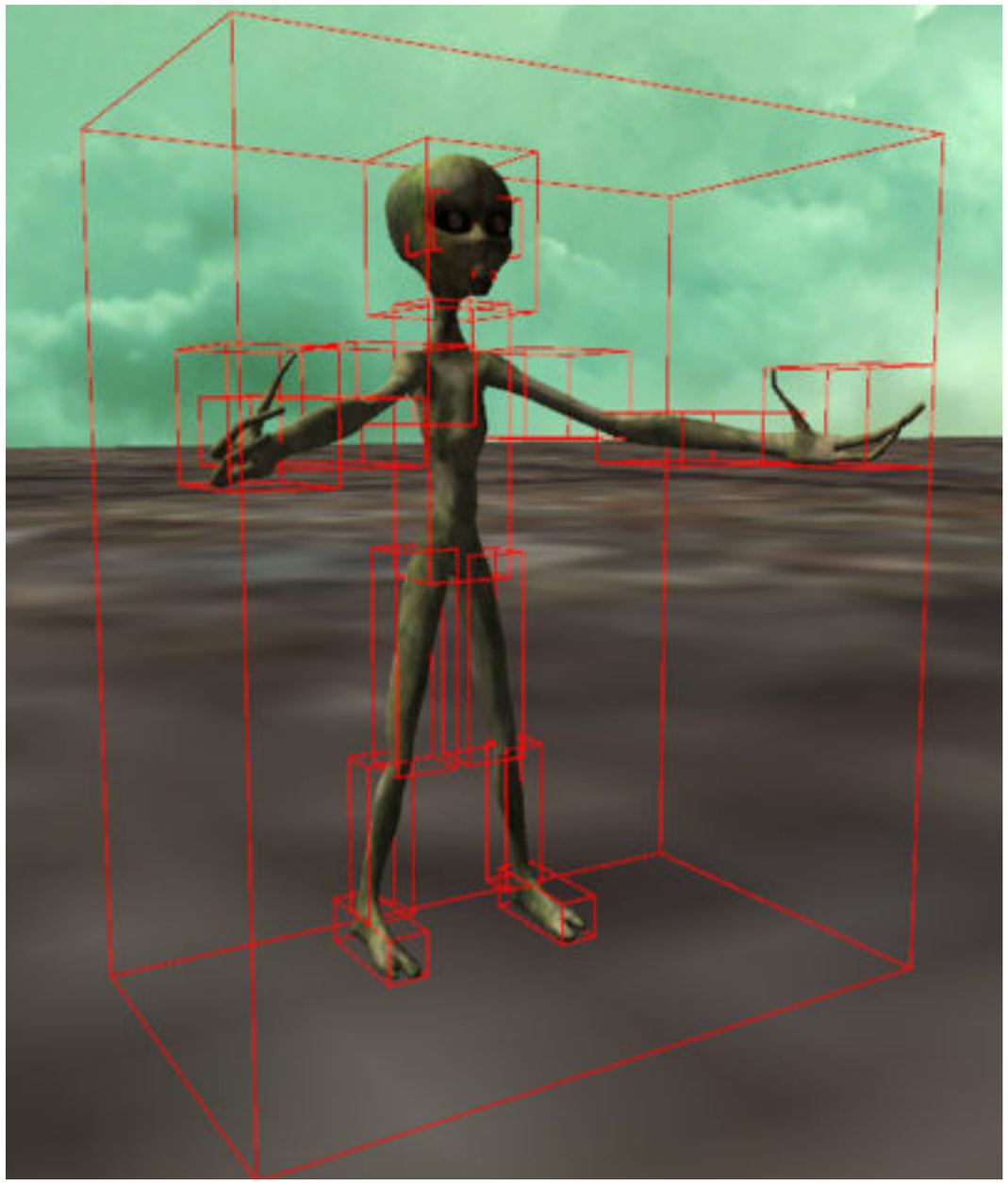
Bounding Volumes

- Bounding volumes are quite efficient for collision detection and culling

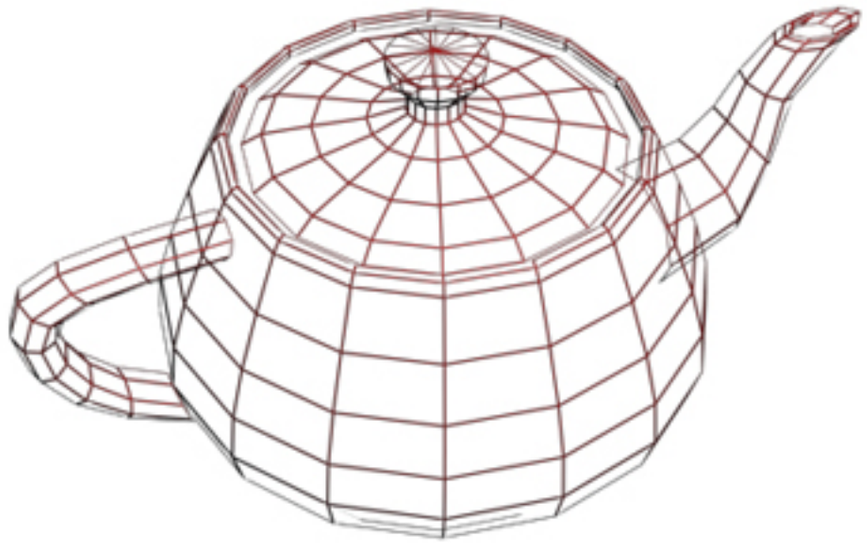
- The shape is used to minimize computation costs and collision/intersection tests

- Most frequent shapes are Bounding Spheres and Axis Aligned Bounding Boxes

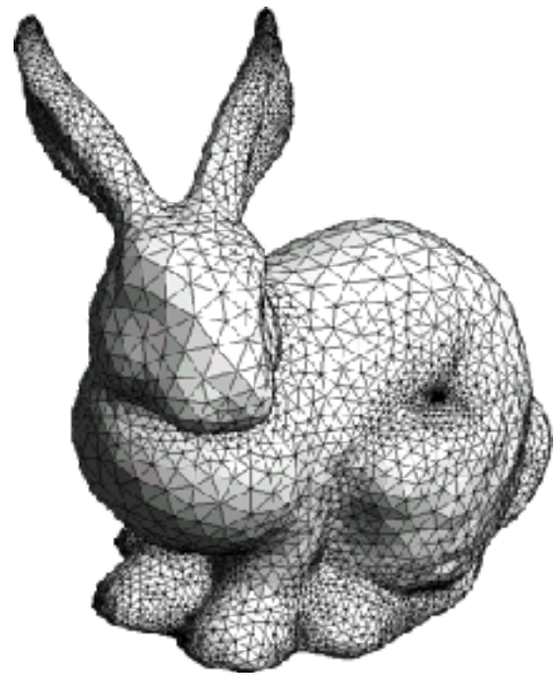
- They are transformed alongside with the geometry they enclose, in the exact same way



Geometry



Original teapot by Martin Newell 1975

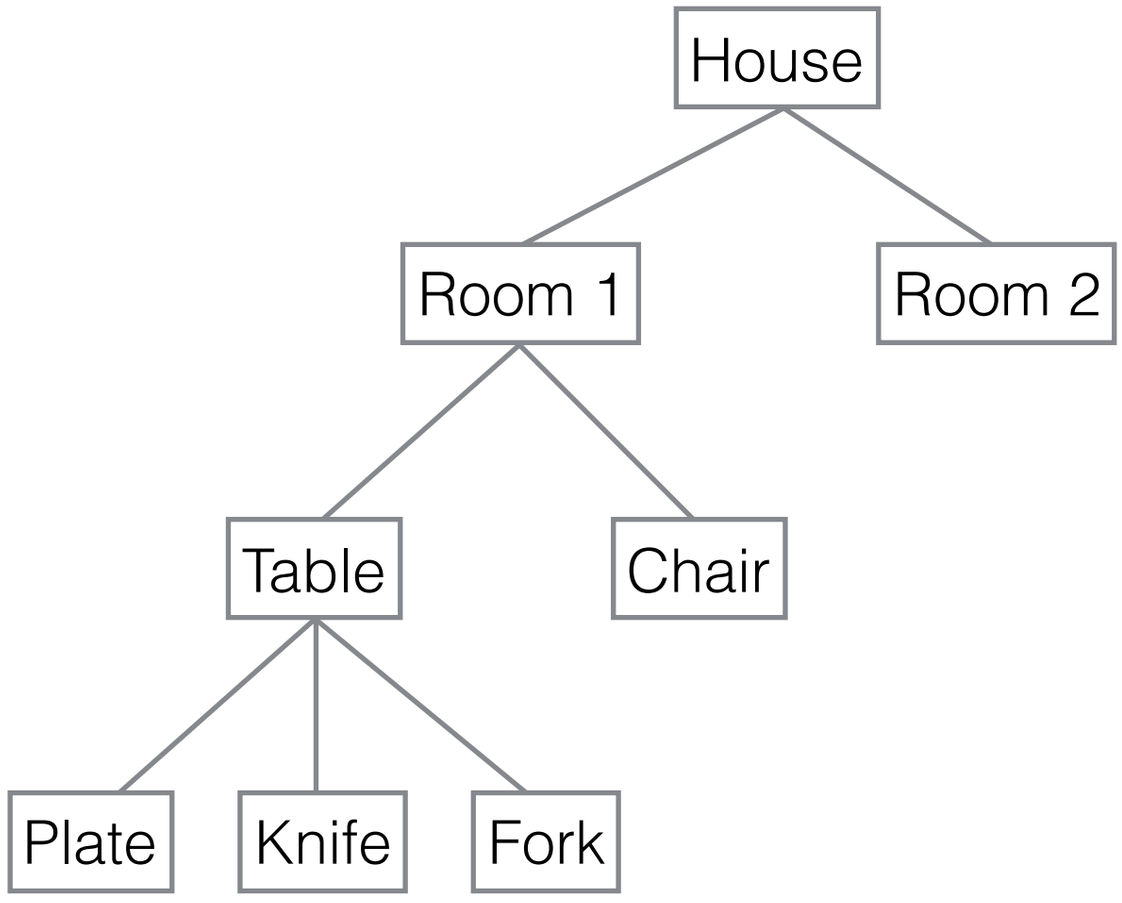


Original Stanford bunny by Greg Turk and Marc Levoy 1994

Note: in jMonkeyEngine the Geometry objects couple the meshes with the Materials that describe their appearance

Node: Spatial

- Is a Spatial that stores the parent-child relationships
  - A Node groups related elements in the scene (examples: plate, knife and fork in a room; rooms in a house)
- A Spatial already has a parent (a single one).
- A Node keeps record of each of its children



Up and Down the Hierarchy

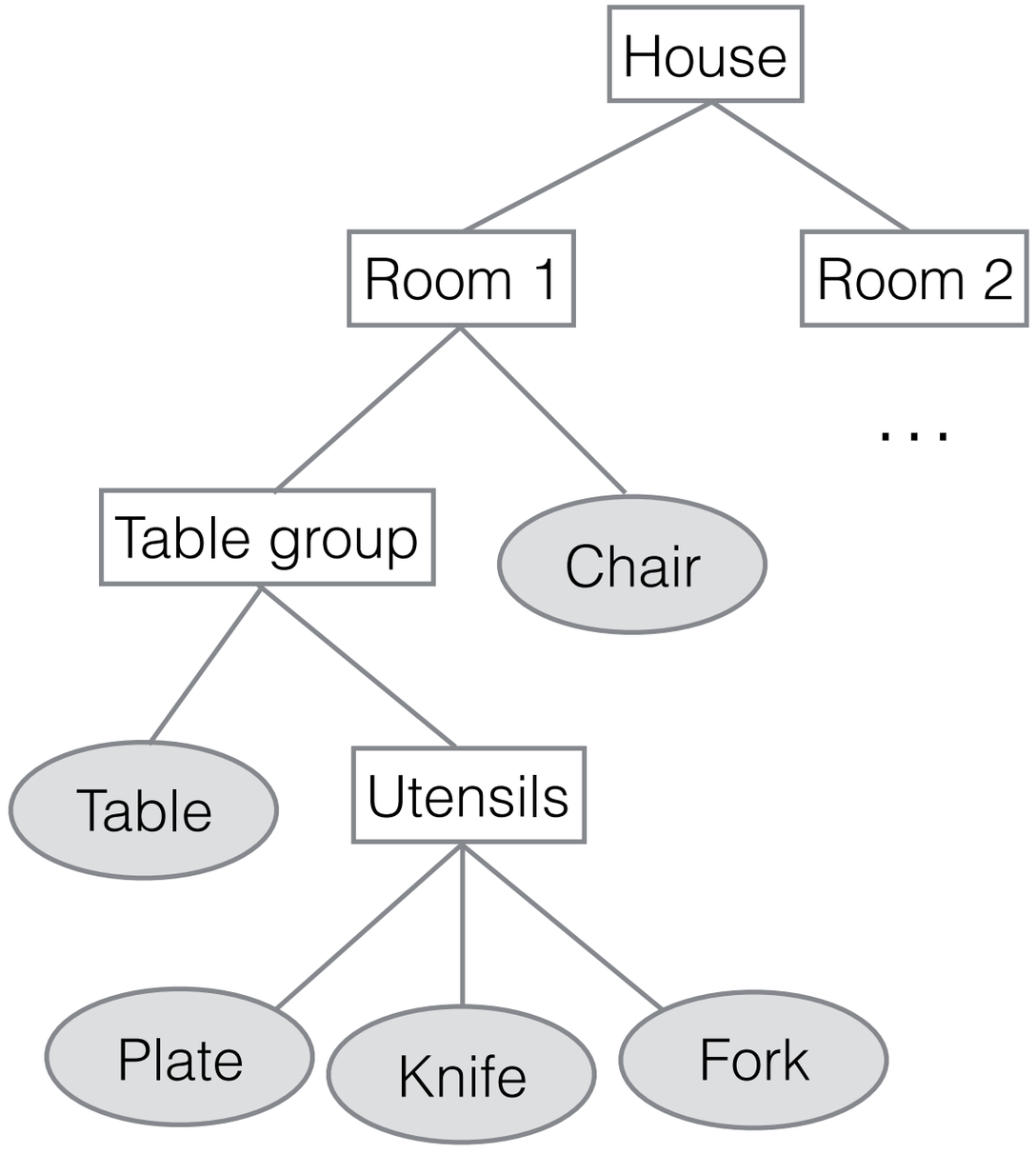
- (⬇) In the rendering pass, the hierarchy is traversed in a depth first way.
  - Each node receives its parent World transform and computes its own World transform: $W_{child} = W_{parent} * L_{child}$
    - With its own world transform, each node can be drawn
- (⬆) When a local transformation changes (e.g. animation) a depth first at the subtree is needed to update the World transforms, but a bottom up update may also be necessary (e.g. node bounds updates up to the root)
- (⬆*⬇*) Ultimately, the transformations in the Scene Graph allow us to go from any coordinate system to another (Model to World, World to Model, Model to Model)

Implementation

- We have two types of spatial
  - Geometry (real objects)
  - Nodes (grouping or abstract objects)
- It is better to keep Geometry at the leafs and Nodes in the interior, for grouping

Scene Graph (remade)

- Geometry spatials are only present as leaf nodes
- Node spatials appear for grouping purposes



Instancing

- So far, the Scene Graph is in fact a tree! Why call it a graph?
- Think of a room with multiple copies of the same geometric model, e.g. a chair or a desk…
- Each copy is in fact an instance of the same exact base model, with instancing attributes (different material, different local transformation, …)
- For each copy (instance) there is a spatial in the scene graph that points to the same base mesh/model. Hence the name graph!
- Advantages: less memory consumption

Controllers/Modifiers

- In a game, objets evolve over time, they animate!
- Controllers are attached to objects in the scene graph and change some of its attributes.
- Animation is here taken in a broader sense than the common character/ model articulated animation:
  - An opponent regularly checks if the hero is in range to attack it
  - A spark changes its brightness as it decays over time
  - A mesh is morphed over time, gradually assuming a different form
  - A detailed 3D model is replaced by a coarser one, as the object gets further away (and the other way around)

Controllers/Modifiers

- Transform Controllers: for key frame interpolation or inverse kinematics
- Vertex and Normal Controllers: for morphing and mesh deformation
- Render State Controllers: light color, intensity, texture coordinates, etc.
- Index Controllers: change mesh topology (level of detail)