Intermediate Graphics & Animation Programming

GPR-300
Daniel S. Buckstein

Intro to Skeletal Animation & Hierarchies
Week 11

License

Intro to Skeletal Animation

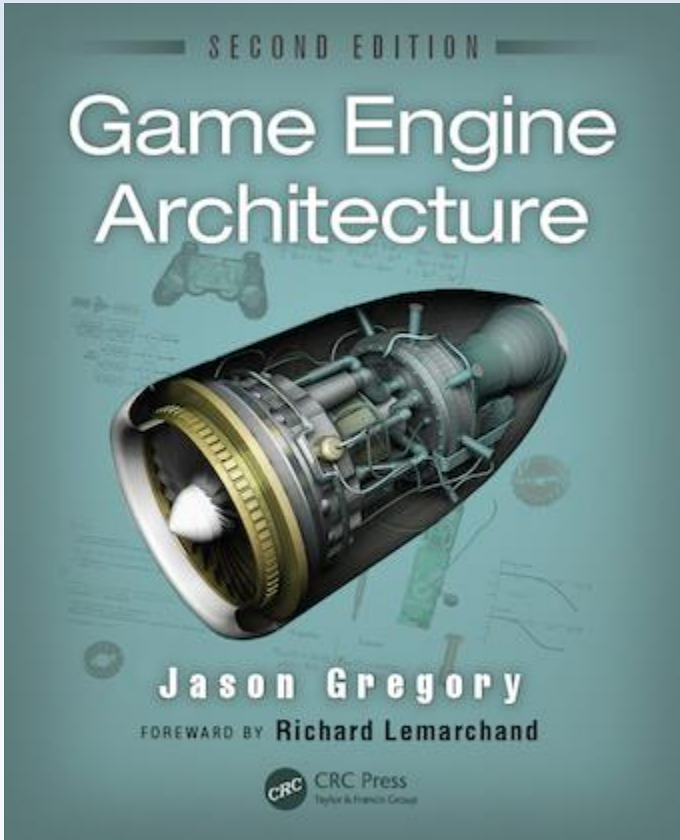
- Hierarchies
- Hierarchies & transformations
- Forward kinematics

Intro to Skeletal Animation

- Some of our systems are derived from here:

- Jason Gregory
- Naughty Dog
- www.gameenginebook.com

- This book is a gold mine.

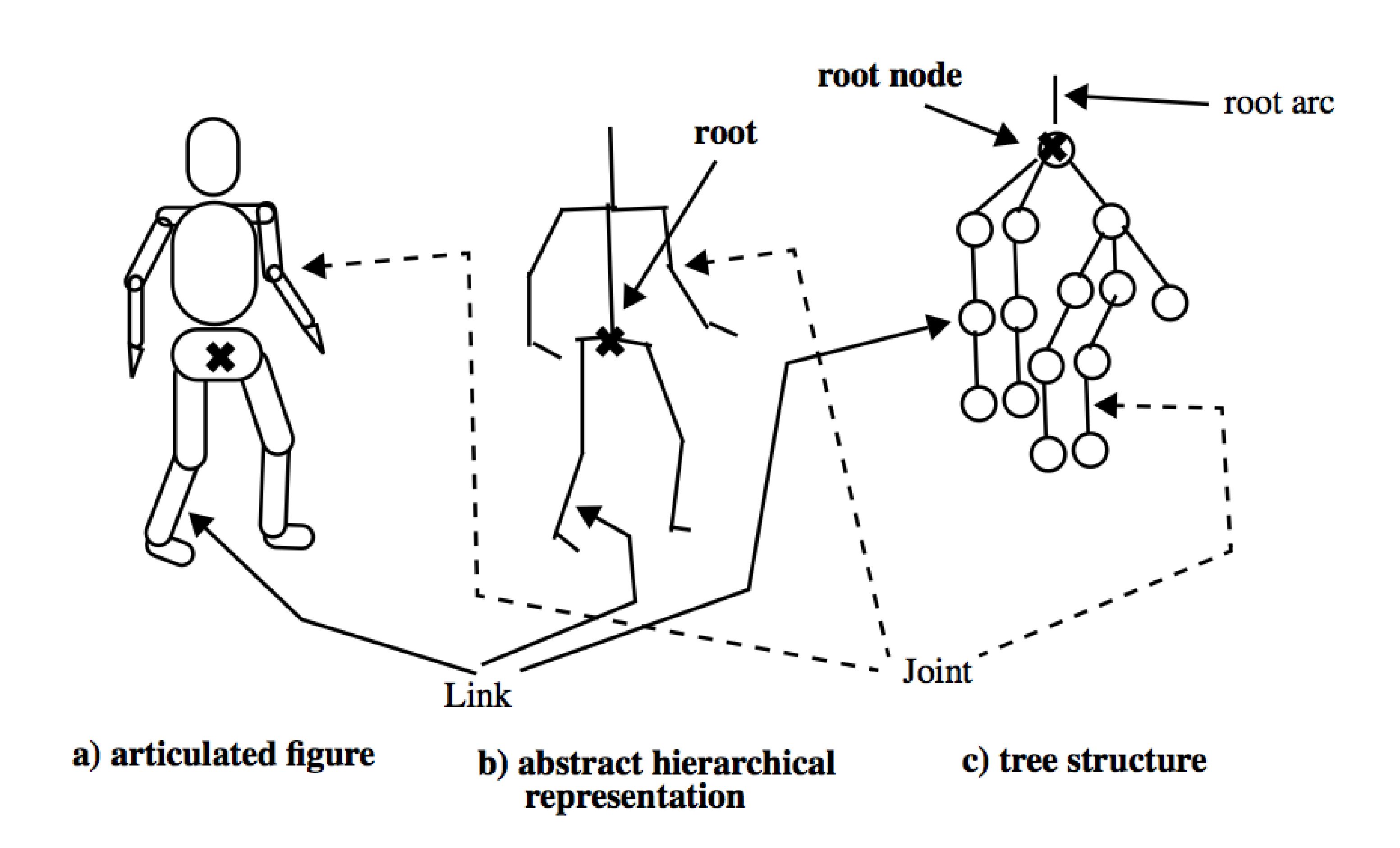


Hierarchies

- ***Kinematic linkages***: describe the motion of one object *relative* to another
- Example:

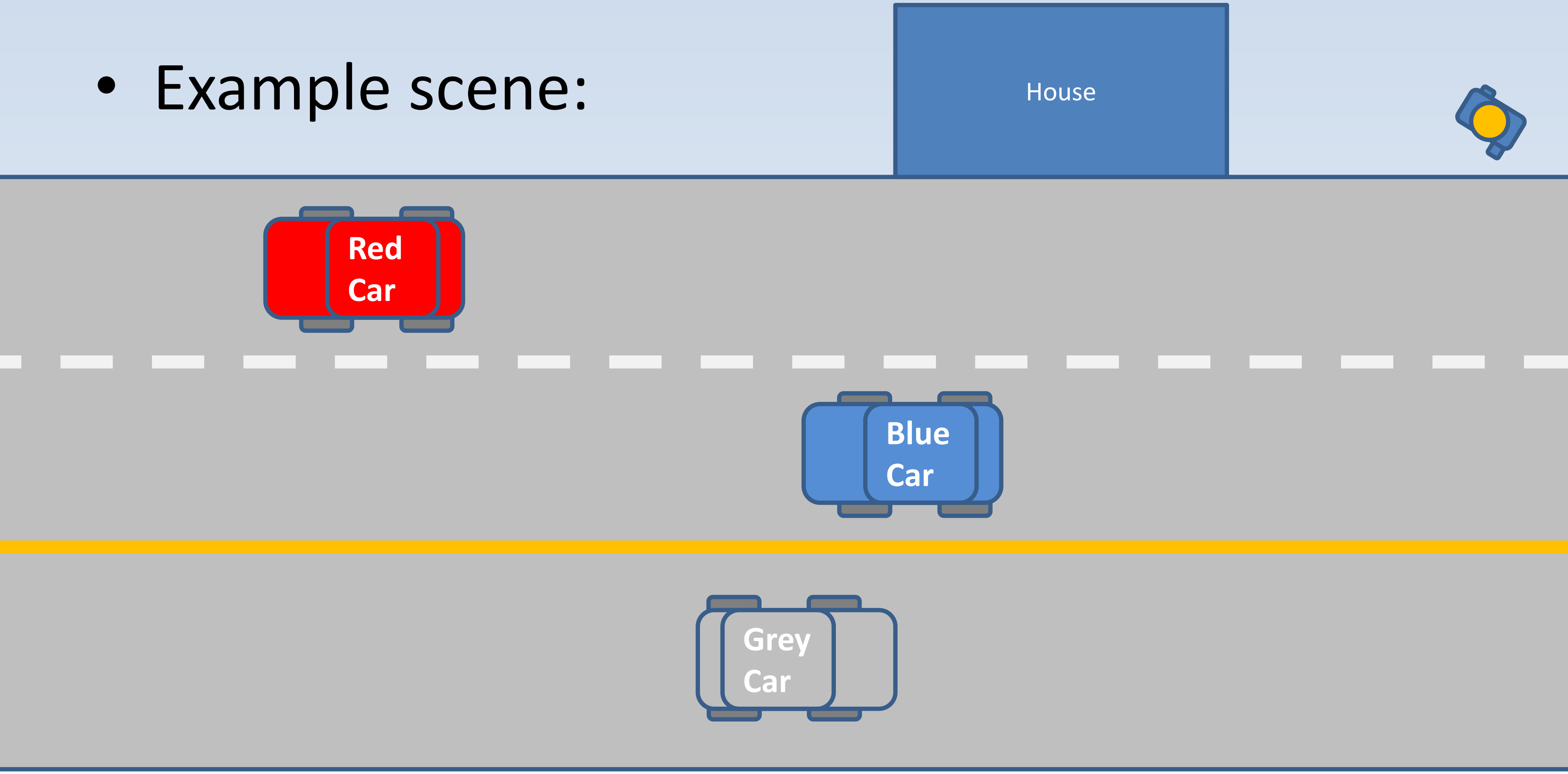
Hierarchies

- Human example:



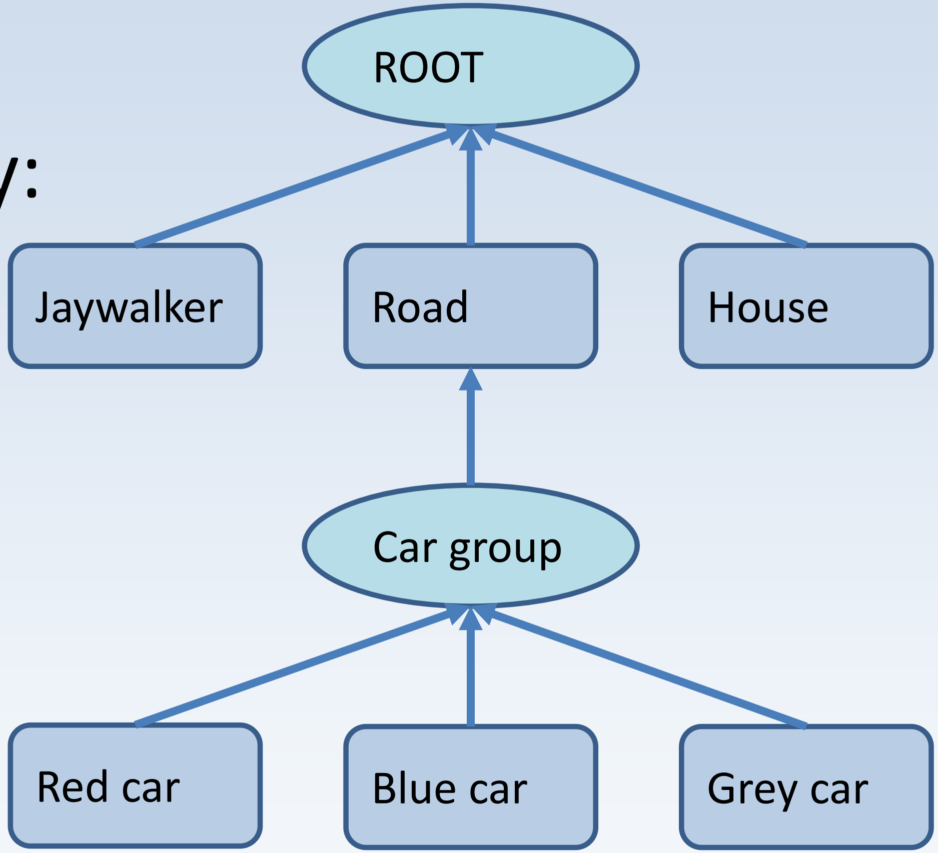
Hierarchies

- Example scene:



Hierarchies

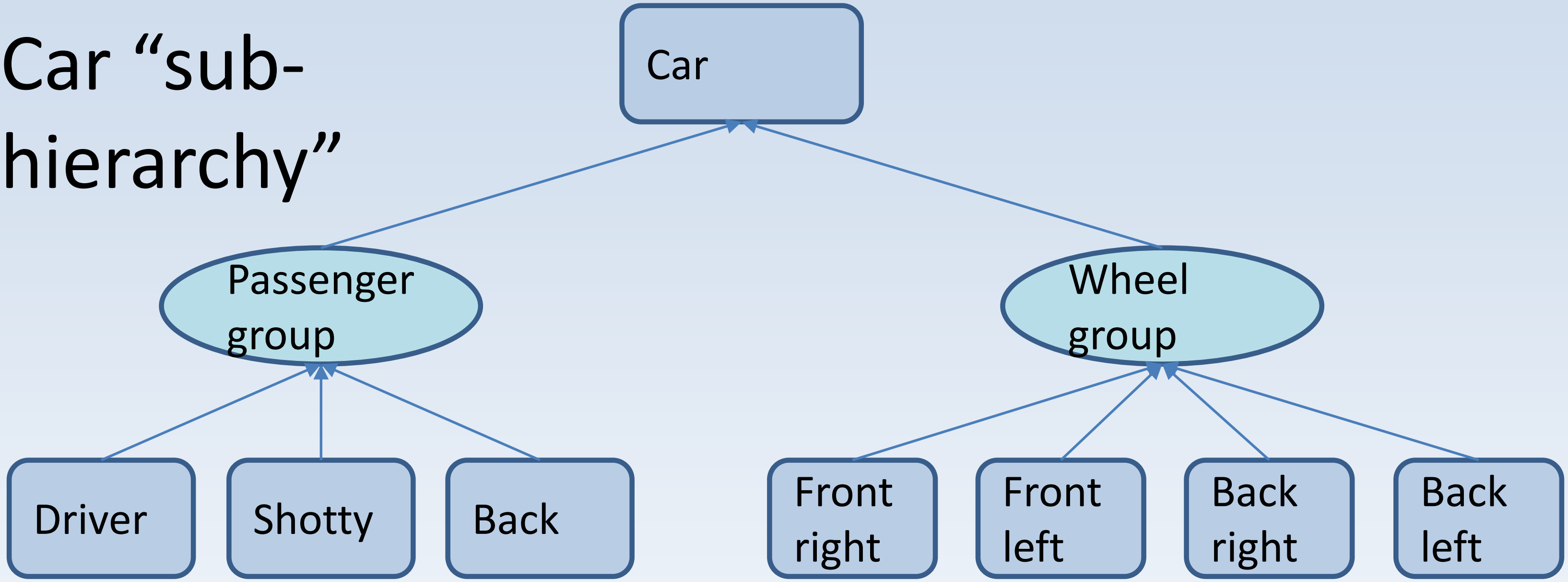
- Example hierarchy:



But wait… what about the cars?

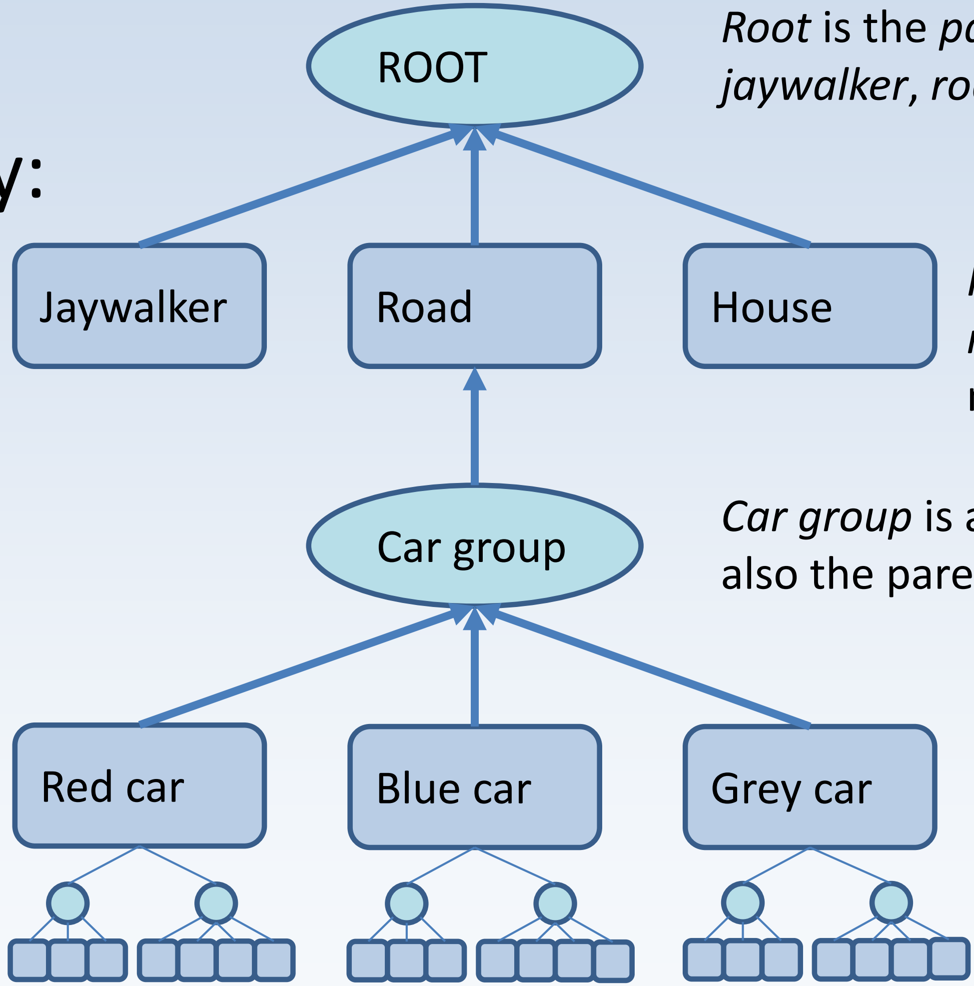
Hierarchies

- Car “sub-hierarchy”



Hierarchies

- Example hierarchy:



Root is the *parent node* of *jaywalker*, *road* and *house*

Road is a *child node* of *root*, also the parent node of *car group*

Car group is a *child node* of *road*, also the parent node of *cars*

Hierarchies

- Hierarchy is a ***tree*** data structure
  - Each node has a single parent node
  - Each node *may* keep track of child nodes
- Programming the hierarchy: efficiency vs. flexibility?
  - Efficient: keep track of parent node only
  - Flexible: keep track of parent *and* children for whatever reason

Hierarchies

- Flexible hierarchy node data structure:

```
struct HierarchyNode
{
    char name[MAX_NAME_CHARS];
    const HierarchyNode *parentNode;
    const HierarchyNode **childNode;
    unsigned int childCount;
};
```

- Nodes not necessarily contiguous in memory
  - …because pointers everywhere…

Hierarchies

- Efficient, contiguous memory approach:
- Refer to self and parent by index…

```
struct HierarchyNode
{
    char name[MAX_NAME_CHARS];
    unsigned int nodeIndex;
    unsigned int parentIndex;
};
```

- (cont’d next slide)

Hierarchies

- Simplified, contiguous memory approach:
- …and store array of nodes in wrapper struct:

```
struct Hierarchy
{
    HierarchyNode *nodeList;
    unsigned int nodeCount;
};
```

- Elements in node array are contiguous and organized by tree depth (you'll see why… soon)

Hierarchies

- Food for thought: The hierarchy only describes the ***general relationships*** between nodes…
- What about the ***spatial relationships***?
- This is more of a ‘*state*’ whereas the hierarchy itself is more of a ‘*resource*’
- Therefore, the two types of relationships are better off decoupled

Hierarchies & Transformations

- Local vs. global transformations (car example)
- Can describe the car relative to the road
- Road is relative to world (or root)
- Much easier to describe wheels relative to car than to the world: ***local transformation***

Hierarchies & Transformations

- Local vs. global transformations (car example)
- The relationship that we have here, *explicitly*, is: Wheel → Car → Road → World
- The wheel's ***local transform*** is relative to ***car***
- Which means that *implicitly*, we can determine the relationship between the *wheels* and the *world*:
  Wheel → World = wheel's *global transform*

Hierarchies & Transformations

- ***Transformation notation***:

Denotes a transformation

Ancestor node index/name $\ {}^{m}T_{n}$ Descendant node index/name

Describes:

"Node *n*'s transformation relative to node *m*'s local space."

Hierarchies & Transformations

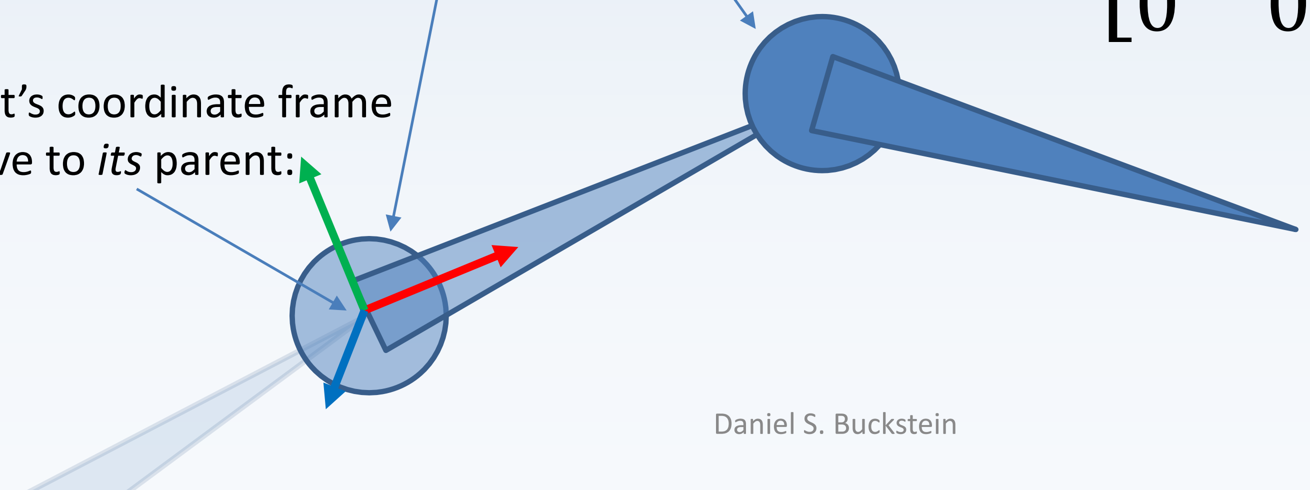
- ***Local transformations***:
- Mathematically a transformation represents a *coordinate frame*
- Each node has its own *local transformation* (its own space)

Hierarchies & Transformations

- Local transformations: each node has its own *transformation relative to its parent*
- Constructed using coordinate frame (TBN):

$$ {}^{\text{parent}}T_{\text{joint}_{(4\times 4)}} = \begin{bmatrix} \boldsymbol{\hat{x}} & \boldsymbol{\hat{y}} & \boldsymbol{\hat{z}} & \boldsymbol{\vec{t}} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

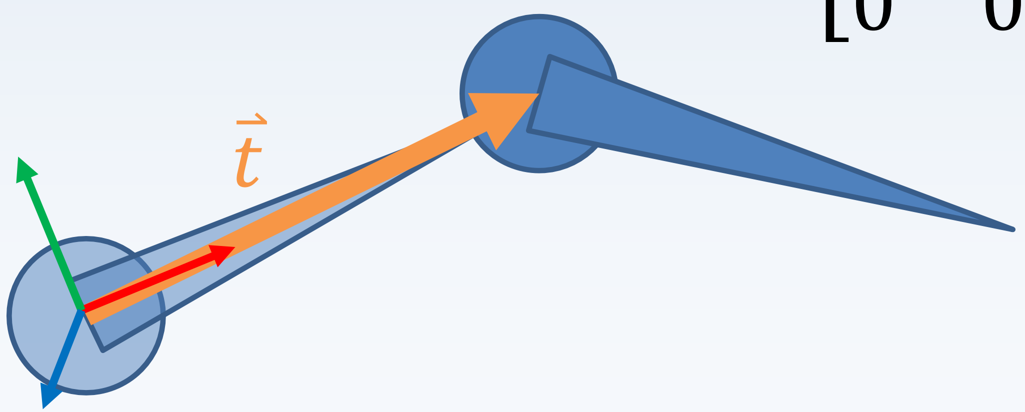
Parent's coordinate frame relative to *its* parent:



Hierarchies & Transformations

- Local transformations: the *translation vector* is the position of the joint relative to the parent:

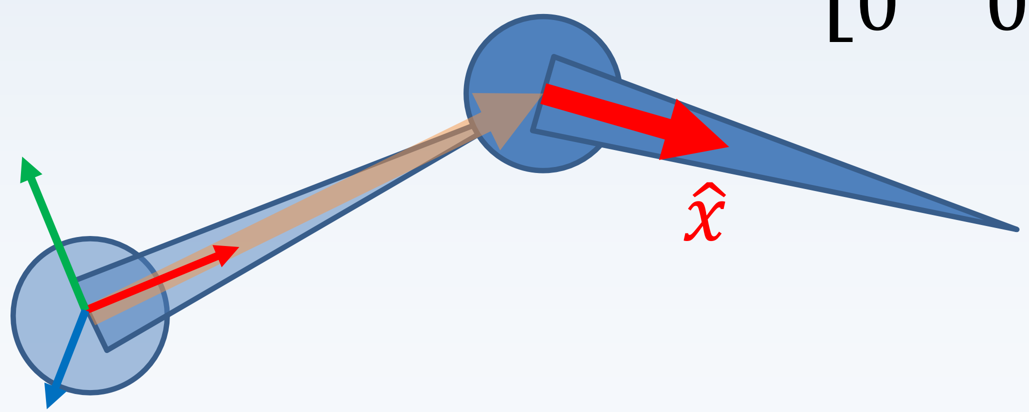
$$
{}^{\text{parent}}T_{\text{joint}_{(4\times4)}} = \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} & \vec{t} \\ \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$



Hierarchies & Transformations

- Local transformations: the *local x-axis* (a.k.a. "right/tangent") usually points *along the bone direction*:

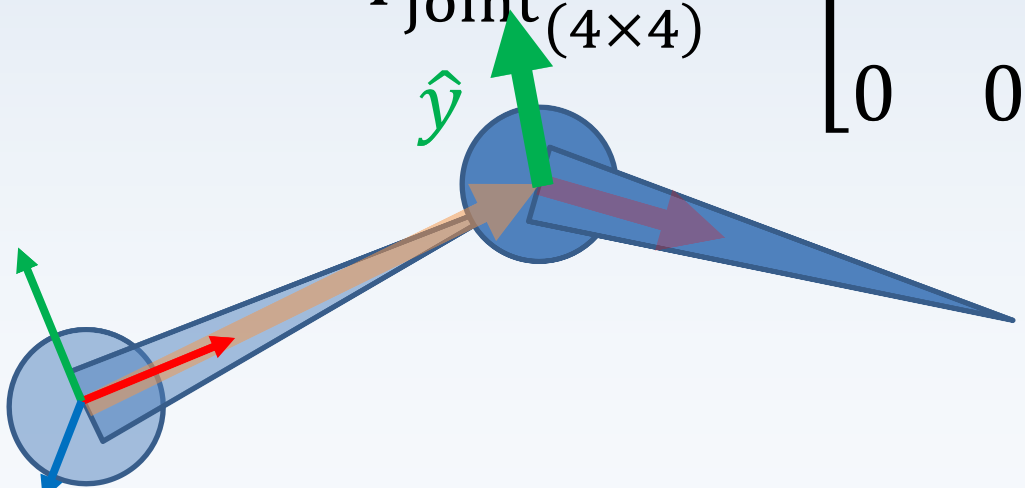
$$^{\text{parent}}T_{\text{joint}_{(4\times4)}} = \begin{bmatrix} \hat{\boldsymbol{x}} & \hat{y} & \hat{z} & \vec{t} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Hierarchies & Transformations

- Local transformations: the *local y-axis* (a.k.a. "up/binormal*") usually points along the *primary axis of rotation* for the joint:

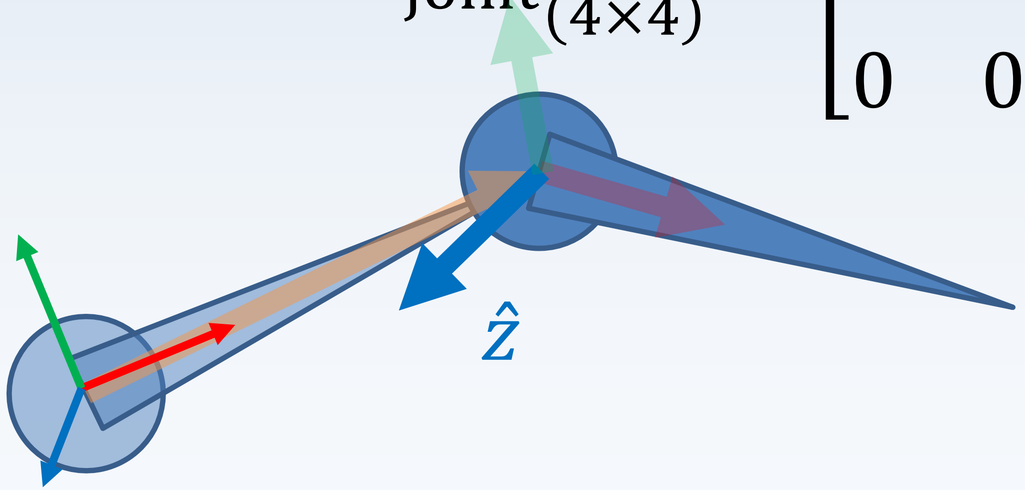
$$ {}^{\text{parent}}T_{\text{joint}_{(4\times 4)}} = \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} & \vec{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$



Hierarchies & Transformations

- Local transformations: the *local z-axis* (a.k.a. "negative direction/normal") completes the orthonormalized frame of reference:

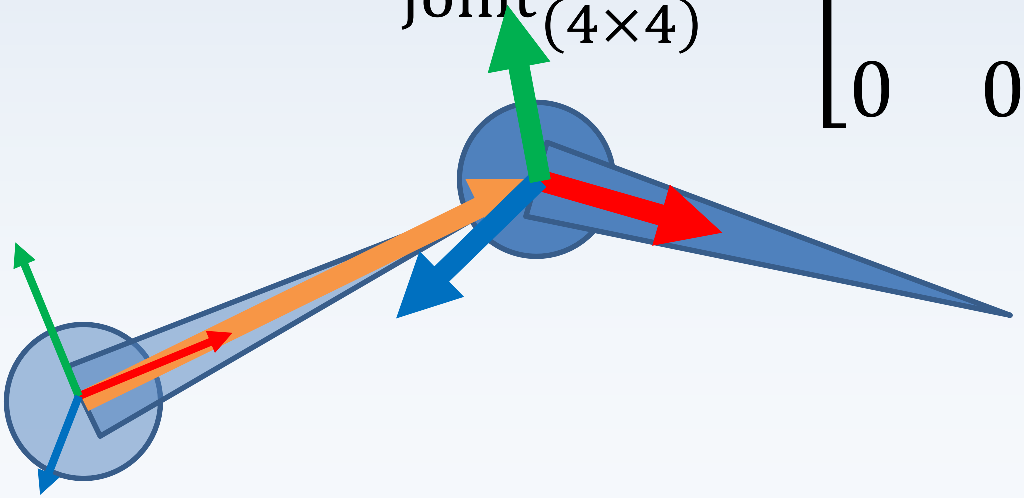
$$ {}^{\text{parent}}T_{\text{joint}_{(4\times4)}} = \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} & \vec{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$



Hierarchies & Transformations

- Local transformations: ***all axes' behaviours can interchange***, just make sure the coordinate frame is correctly orthonormal!

$$ {}^{\text{parent}}T_{\text{joint}_{(4\times4)}} = \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} & \vec{t} \\ \\ 0 & 0 & 0 & 1 \end{bmatrix} $$



Hierarchies & Transformations

- Local transformations: remember Frenet-Serret frame and basis vectors…
- …if you know two, you can solve the third:

$$^{\text{parent}}T_{\text{joint}_{(4\times4)}} = \begin{bmatrix} \hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}} & \vec{\boldsymbol{t}} \\ \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\hat{x} = \hat{y} \times \hat{z}$$

$$\hat{y} = \hat{z} \times \hat{x}$$

$$\hat{z} = \hat{x} \times \hat{y}$$

Hierarchies & Transformations

- The beginning of a skeletal animation system: *a hierarchy state with transformations*:

```
struct HierarchyState
{
    const Hierarchy *hierarchy;
    mat4 *localTransformList;
};
```

Forward Kinematics

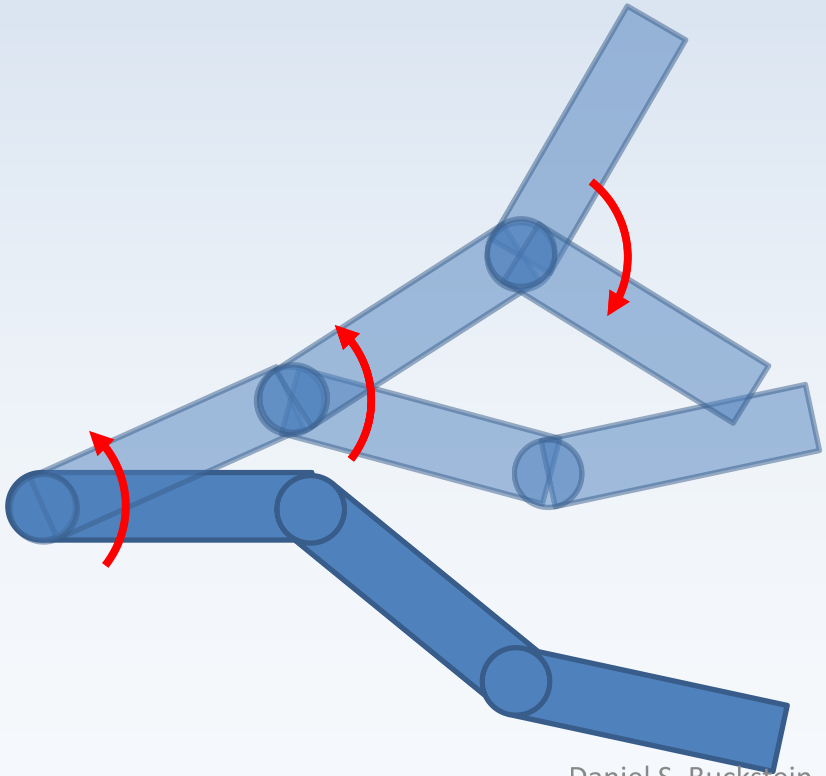
- What is the root joint's parent frame???
- *The world/scene* also has its own coordinate frame (i.e. where we define global 'up')
- Each node's *global transformation* is how it is oriented relative to the world!
- ***Forward kinematics***: Determine a node's transformation *relative to the world/scene*.
- I.e. "converting local to global"

Forward Kinematics

- Human example:
  - Shoulder
    - Elbow
      - Wrist
        - Thumb
        - Index finger
        - Middle finger…
- What happens when we rotate our *shoulder*?

Forward Kinematics

- Human example: rotating any parent node will have a *global* effect on all child nodes



Forward Kinematics

- *Forward kinematics (“FK”)*: we want *all* of our node transforms to be relative to the *world*
- Hard to do mathematical calculations when objects have their own coordinate systems!!!
- Get it all in the ***same frame of reference***, perfect for ***graphics, animation, physics***…
- How do we do that?
    - Psssssst hey… it’s math.

Forward Kinematics

- Transformation notation (cont’d):

$$^{\text{world}}T_n = {}^{\text{world}}T_{\text{gParent}} \, {}^{\text{gParent}}T_{\text{parent}} \, {}^{\text{parent}}T_n$$

“To get node *n*’s transformation relative to the world’s coordinate frame, take the product of all transforms between node *n* and the world.”

Forward Kinematics

- Transformation notation (cont'd):
- Let's apply this principle to calculate the thumb's spatial relationship with the world:

$$^{\text{world}}T_{\text{thumb}} = {}^{\text{world}}T_{\text{hip}}\,{}^{\text{hip}}T_{\text{spine1}}\,{}^{\text{spine1}}T_{\text{spine2}}\,{}^{\text{spine2}}T_{\text{spine3}}\,{}^{\text{spine3}}T_{\text{shoulder}}\,{}^{\text{shoulder}}T_{\text{elbow}}\,{}^{\text{elbow}}T_{\text{wrist}}\,{}^{\text{wrist}}T_{\text{thumb}}$$

😐

- Is there a less confusing way to do it?

Forward Kinematics

- **SolveRecursiveFK (*node*)**
  - If *node **is*** root (no parent)
    - Node's world transform ***is*** node's local transform
  - Else,
    - *Node*'s world transform
      = parent's world transform * node's local transform
    - For each child
      - SolveRecursiveFK (child)

Forward Kinematics

- If using efficient data structure, we don't track child nodes…
- …nodes should be ***ordered by tree depth*** so that their parents will always be updated first!
- Root node's index is 0
- Root's parent's index is -1 (doesn't exist)
- March through array of nodes and do the following algorithm:

Forward Kinematics

- **SolveOrderedFK (*hierarchy*)**
  – For each *node* in *hierarchy*
    - If node is root (parent index is -1)
      – Node’s world transform ***is*** node’s local transform
    - Else
      – Node’s world transform
        = parent’s world transform * node’s local transform

Forward Kinematics

- FK math (to compute global transforms):

Root node:

- ${}^{\text{world}}T_n$ (already known)

Everything else:

known

known

- ${}^{\text{world}}T_n = {}^{\text{world}}T_{\text{parent}} \; {}^{\text{parent}}T_n$

Unknown: solve by multiplying two things we do know!

- If you are not familiar with *transformations* by now, probably a good idea to review this…

Forward Kinematics

- Store local transform for animation
- Store world transform (FK result) for graphics

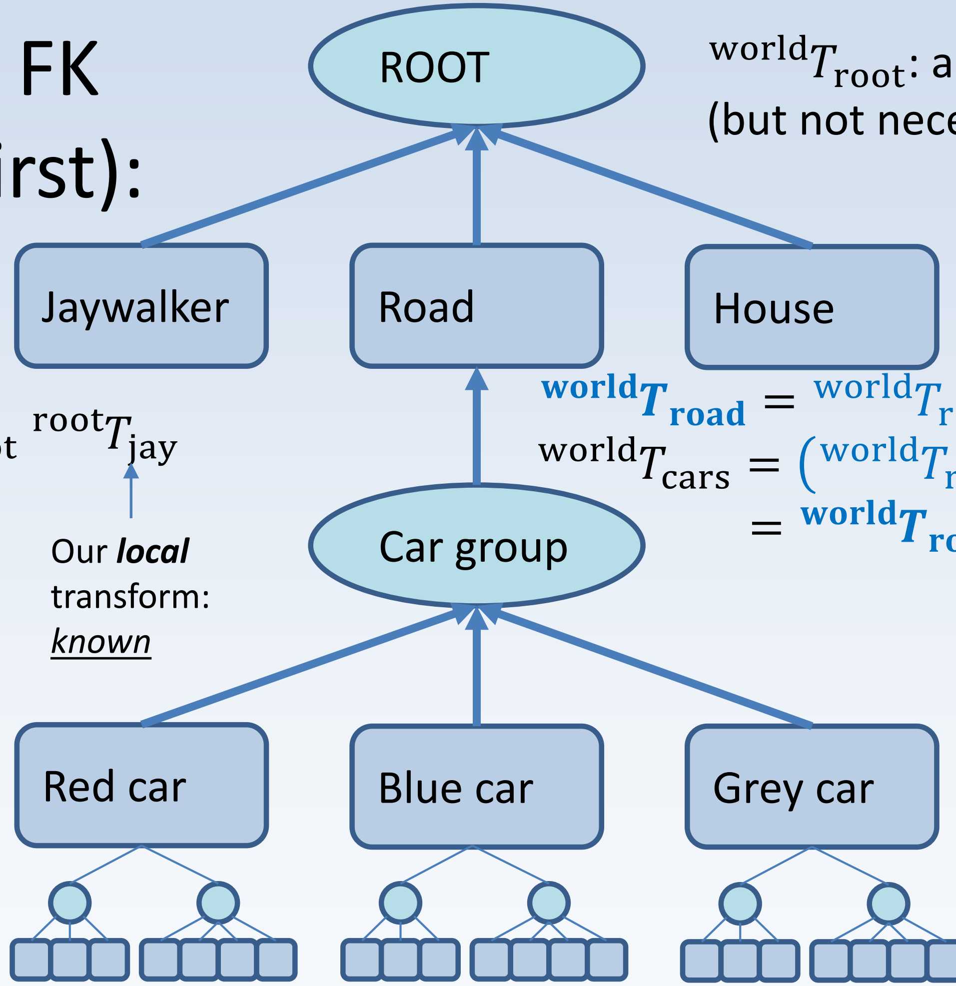
```
struct HierarchyState
{
    const Hierarchy *hierarchy;
    // ANIMATE using local transforms!
    mat4 *localTransformList;
    // RENDER using result of FK!
    mat4 *worldTransformList;
};
```

Forward Kinematics

- ***Motion hierarchy***: a tree that visualizes the spatial relationships between objects…
- Who moves relative to whom?
- Hierarchy simply defines *relationships*, but a component of each node is a *transformation*
- ***Local*** and ***global*** transformations… which is which???

Forward Kinematics

- Example FK (depth-first):



${}^{\mathrm{world}}T_{\mathrm{root}}$: already in world space!
(but not necessarily *identity*)

$$\boldsymbol{{}^{\mathrm{world}}T_{\mathrm{road}}} = {}^{\mathrm{world}}T_{\mathrm{root}}\ {}^{\mathrm{root}}T_{\mathrm{road}}$$

$$\begin{aligned} {}^{\mathrm{world}}T_{\mathrm{cars}} &= \left({}^{\mathrm{world}}T_{\mathrm{root}}\ {}^{\mathrm{root}}T_{\mathrm{road}}\right){}^{\mathrm{road}}T_{\mathrm{cars}} \\ &= \boldsymbol{{}^{\mathrm{world}}T_{\mathrm{road}}}\ {}^{\mathrm{road}}T_{\mathrm{cars}} \end{aligned}$$

$${}^{\mathrm{world}}T_{\mathrm{jay}} = {}^{\mathrm{world}}T_{\mathrm{root}}\ {}^{\mathrm{root}}T_{\mathrm{jay}}$$

Our ***global*** transform: <u>*unknown*</u>, solved by matrix mult.

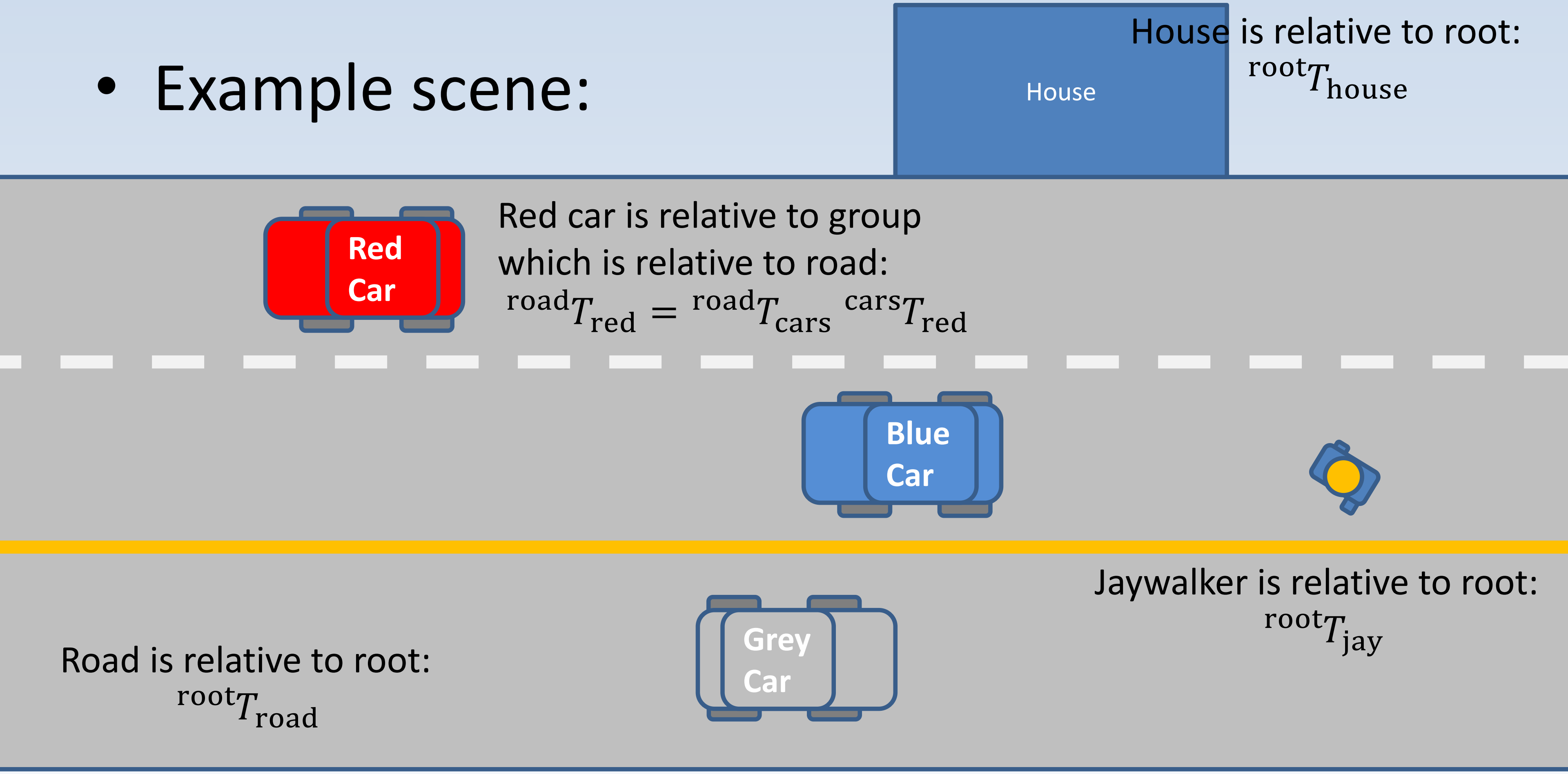
Parent's ***global*** transform: <u>*known*</u>

Our ***local*** transform: <u>*known*</u>

Depth-first traversal: do all child nodes first (complete when no more children)

Forward Kinematics

- Example scene:



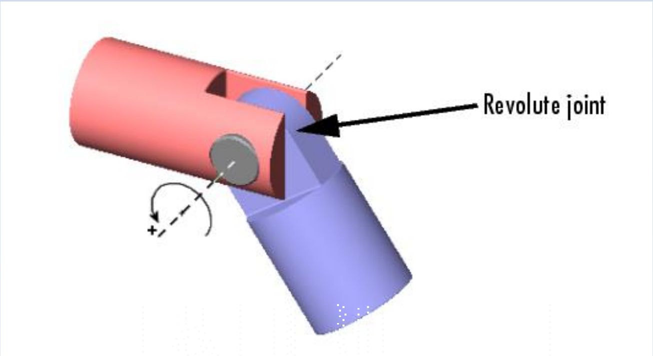
Forward Kinematics

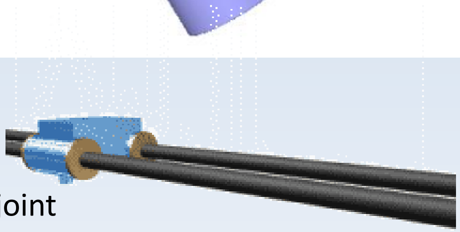
- Kinematic linkages for animation systems:
- Humans and robots can be thought of as a set of *bones* or *joints*
- Previously we have discussed “nodes”
- *Joints* are the animation term for *nodes*

- A hierarchy of *joints* is a *skeleton*

Forward Kinematics

- Different kinds of joints:
- “Revolute”: only rotates
- “Prismatic”: only translates




Prismatic joint

- *Most* human joints are considered “revolute”
- We have a set of angles per joint relative to the parent… update them all at once → ***FK***

Forward Kinematics

- “Forward kinematics” is a general term that applies to specific kinematic objects (human character) or entire scenarios (cars on road)
- Applied to characters, however, we are able to animate the *local joint orientations*, which will affect all of the child nodes’ *global orientations*…
- So how about that animation?!!

The end.

- Questions? Comments? Concerns?