ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

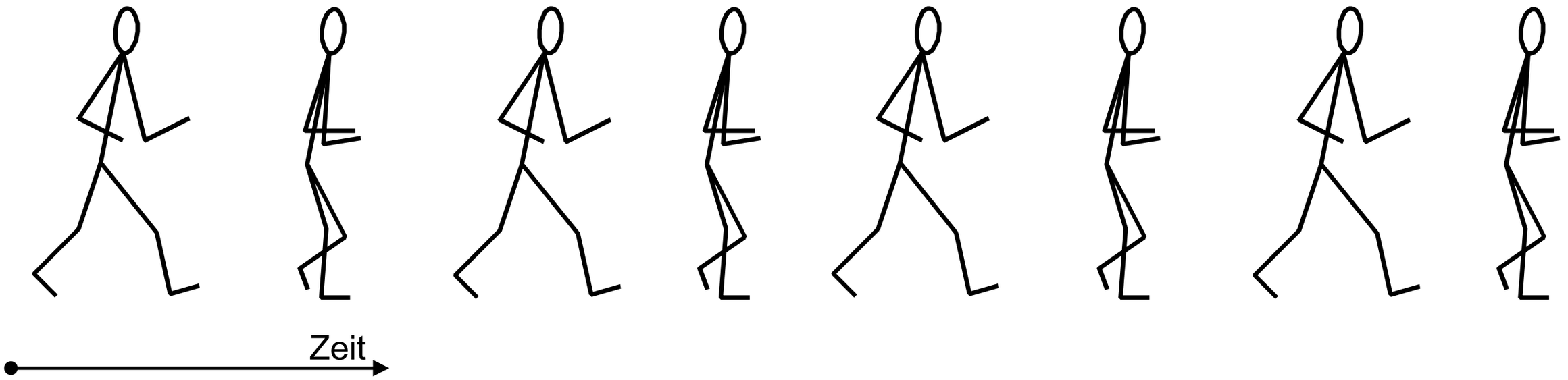
Hierarchical Scenes

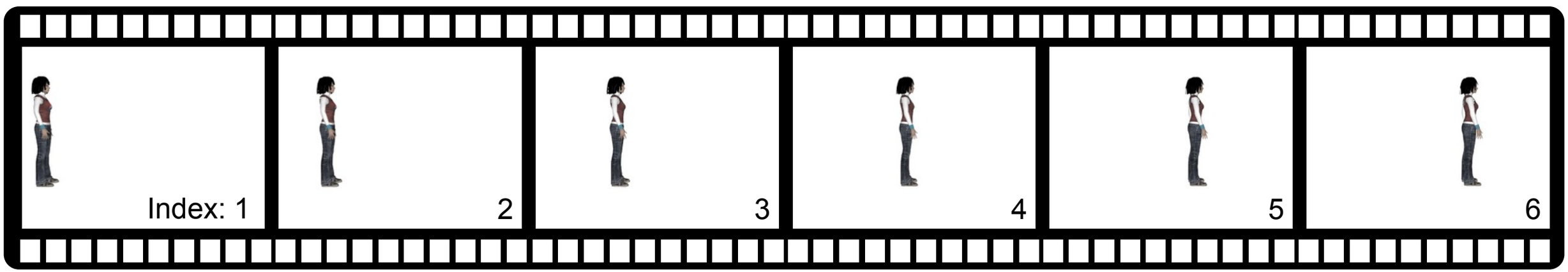
November 19th, 2015

Rafael Radkowski

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Animations





Animation is the process to generate moving objects on screen. The computer graphics rendering pipeline only generates static scenes. Animation techniques alter this scene from frame to frame. Thus, objects appear as moving objects.

Representing the keyframes

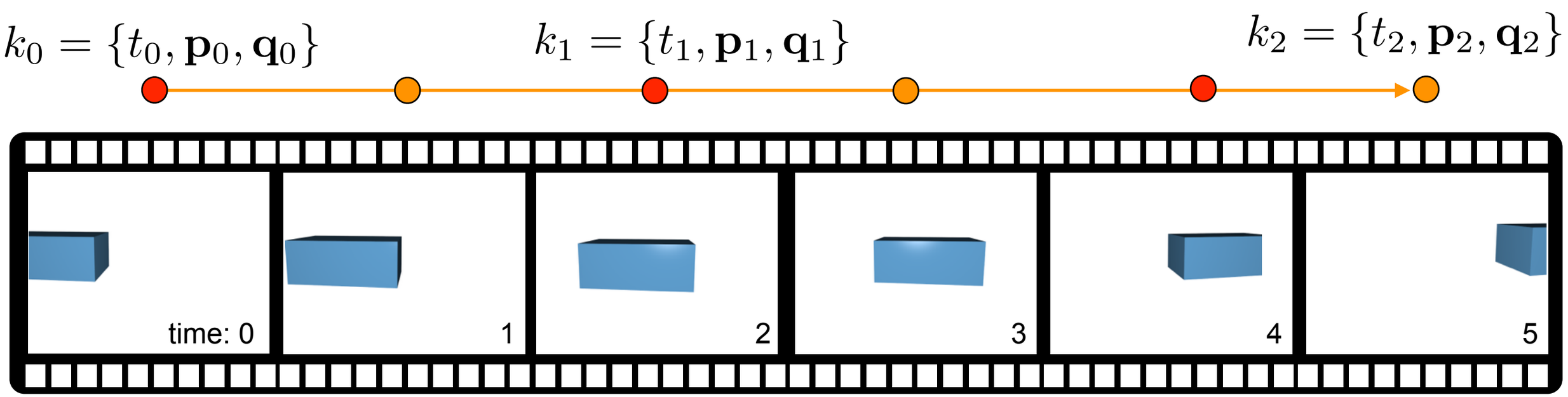
For every keyframe k_i we store

the time fraction in an interval between 0 and 1

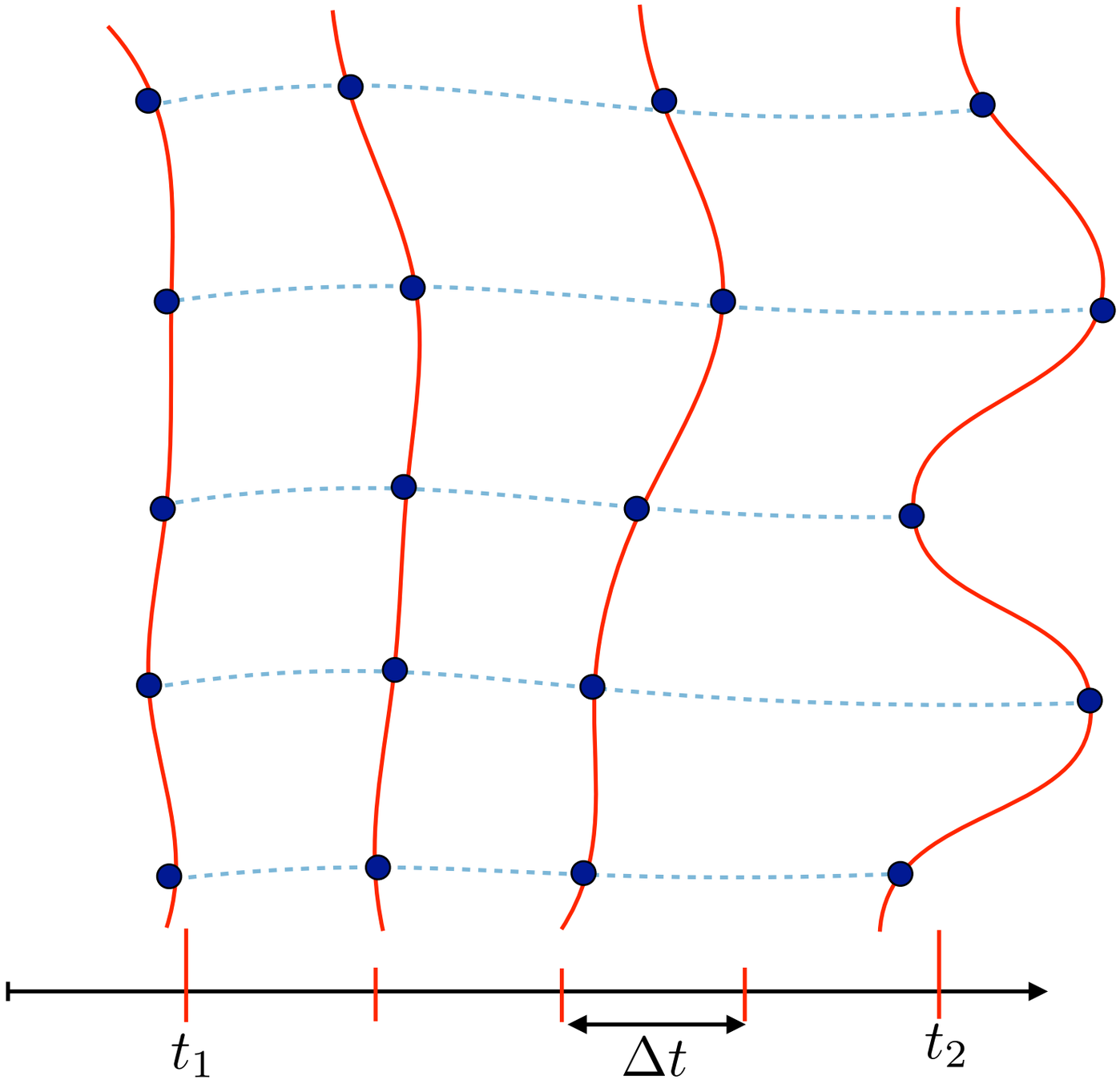
the position as vector with {x, y, z} coordinates

the orientation of the object as quaternion q = {x, y, z, w}

$$k_i = \{t_i, \mathbf{p}_i, \mathbf{q}_i\}$$



Constant Motion

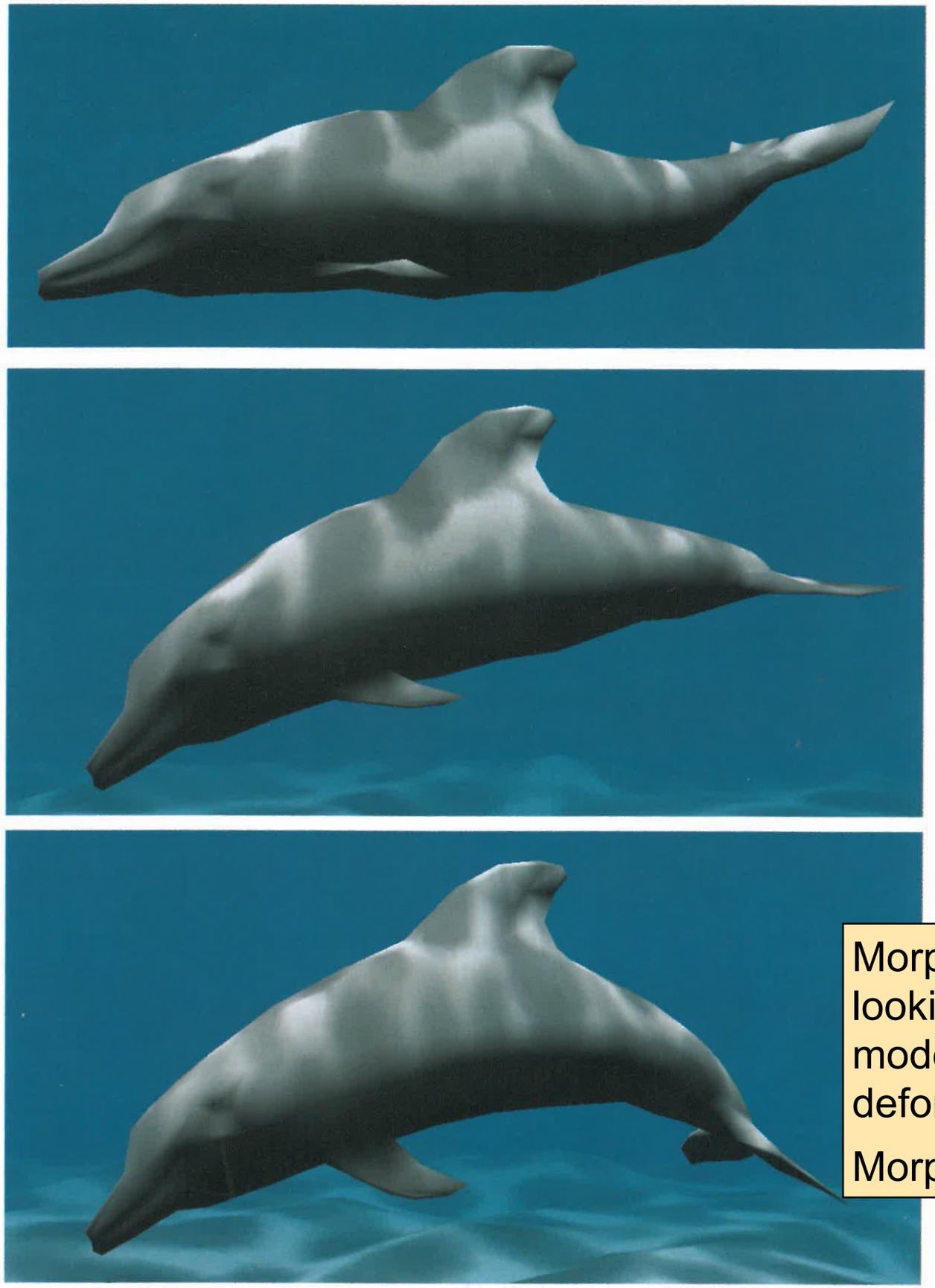


For constant motion, divide the time interval between the keyframes in n+1 subintervals

$$\Delta t = \frac{t_2 - t_1}{n + 1}$$

The time for the *j*th frame is:

$$t = t_1 + j\ \Delta t$$



Morphing

Morphing is a technique that facilitates natural looking motion. It deforms the mesh of a 3D model during runtime using one or multiple deformation models.

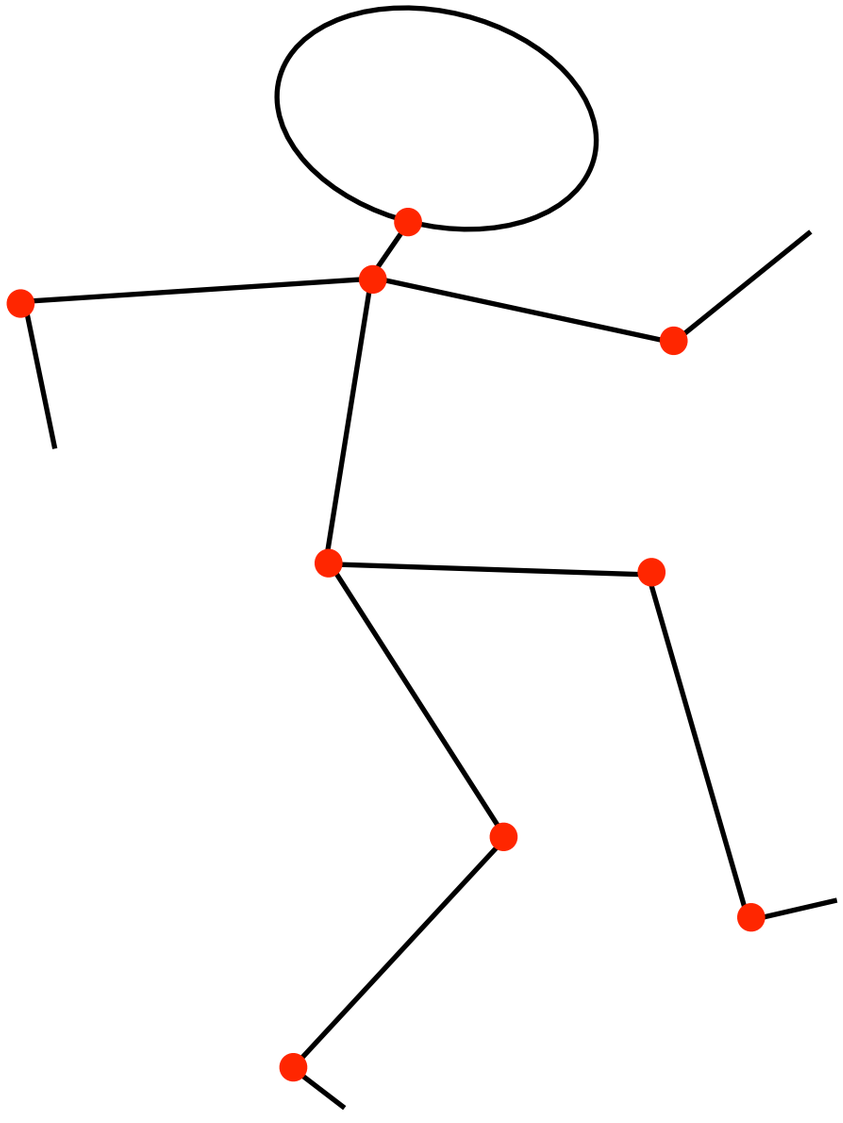
Morphing is a shorten term of metamorphosing

Content

- Hierarchical Modeling
- Graph data structures
- Scenegraphs and Scenegraph APIs

Video

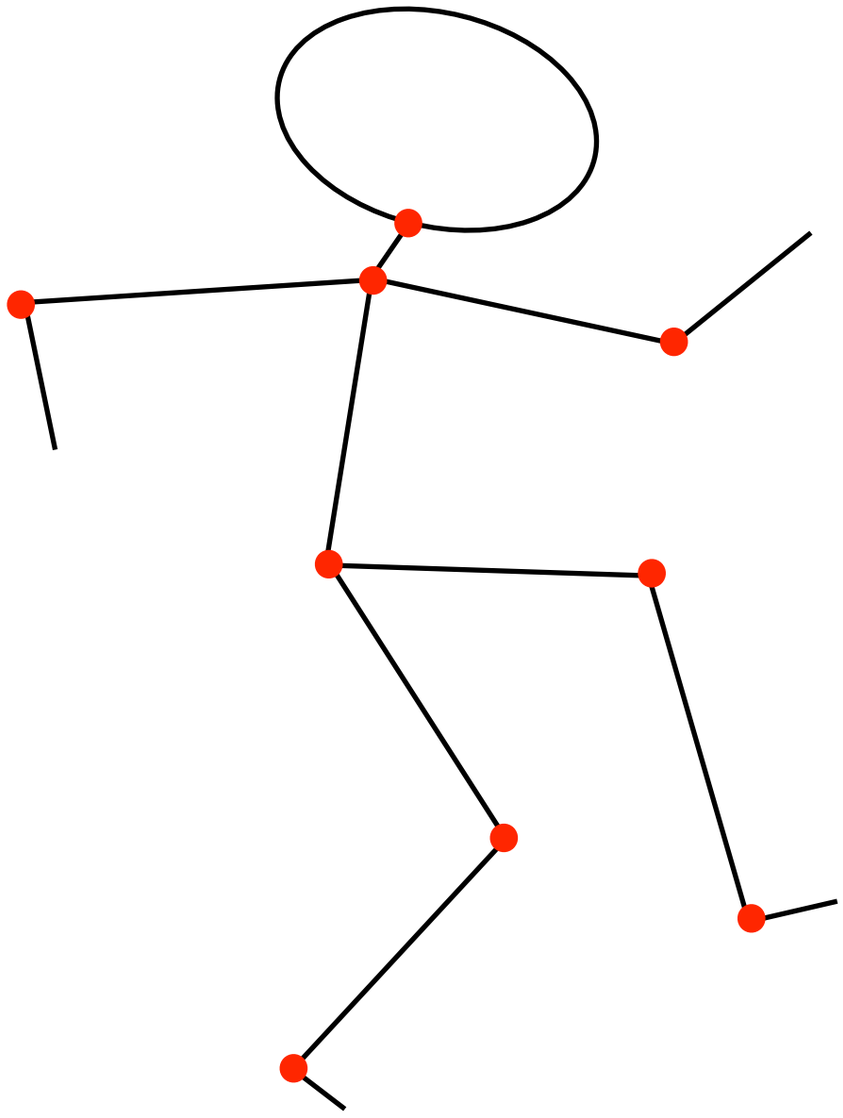
Articulated Figure Animation



The basic technique for animating people, insects, vehicles, etc. are **articulated figures**, which are a hierarchical structure of single objects connected by joints.

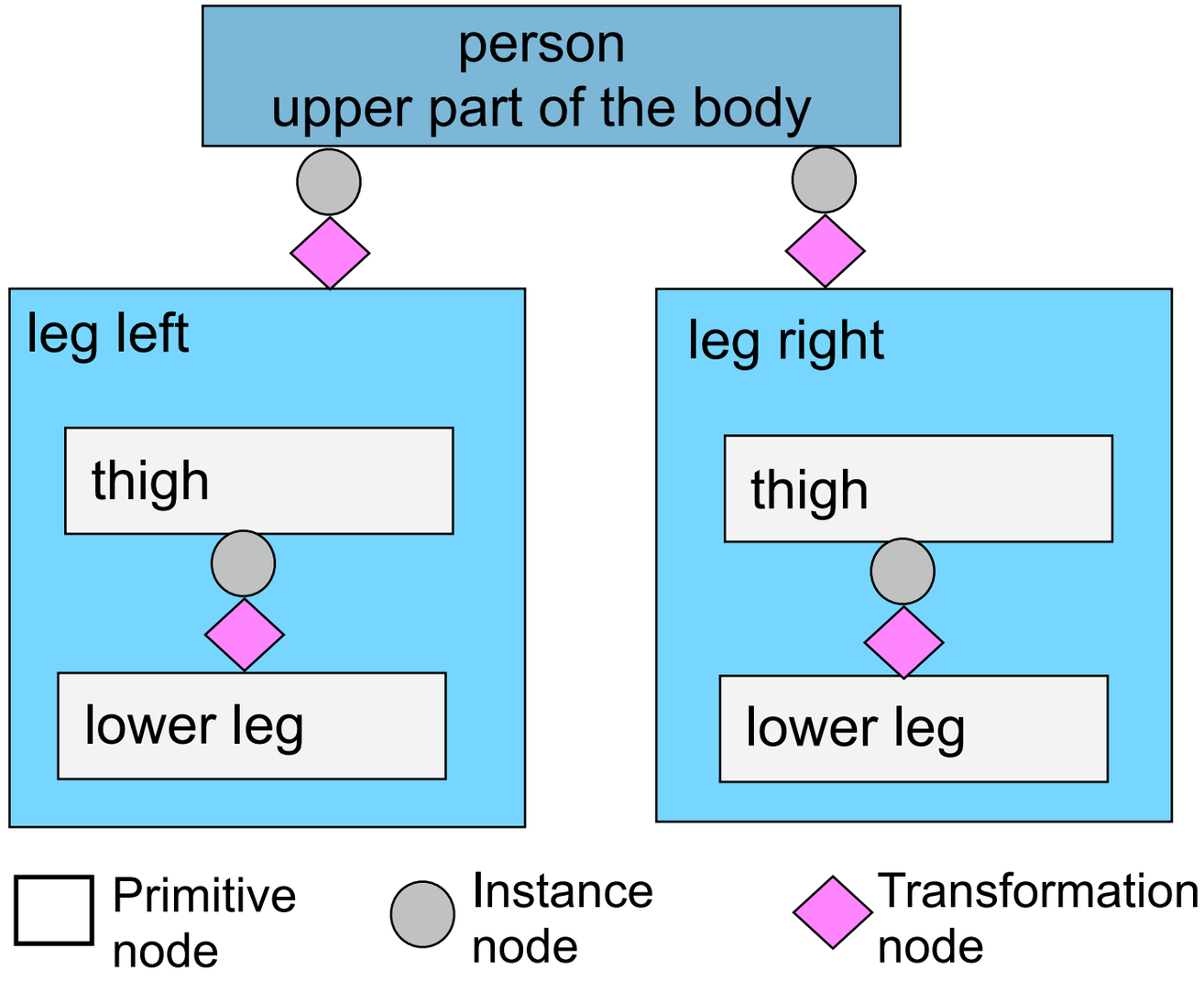
We formally create a stick figure also called a **skeleton** which we wrap into a skin.

Hierarchical Modeling



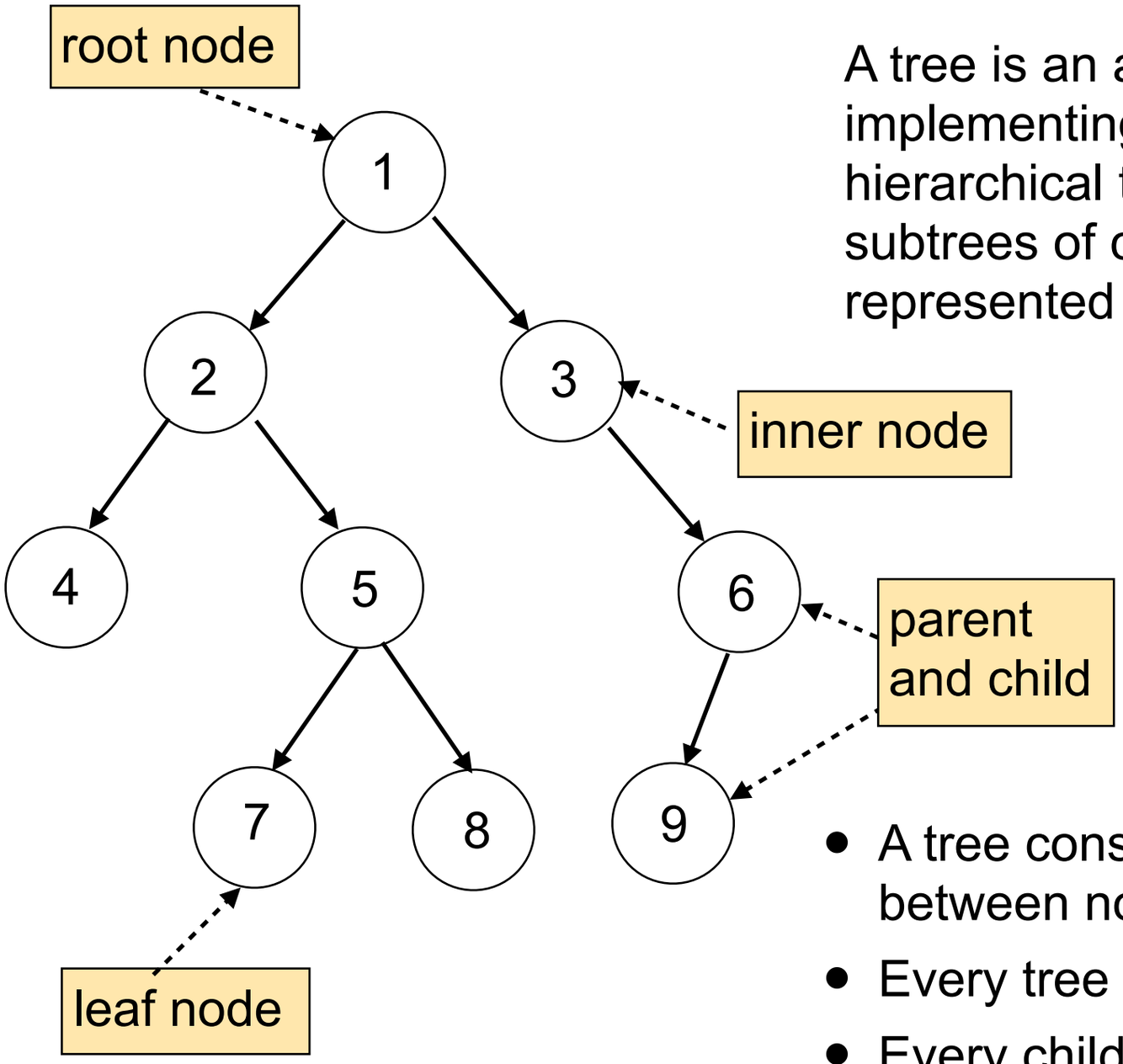
We design a complex model by modularizing the specification of the geometry into subcomponents.

One way is the **top-down design**



Whenever it is possible, construct your models hierarchically.

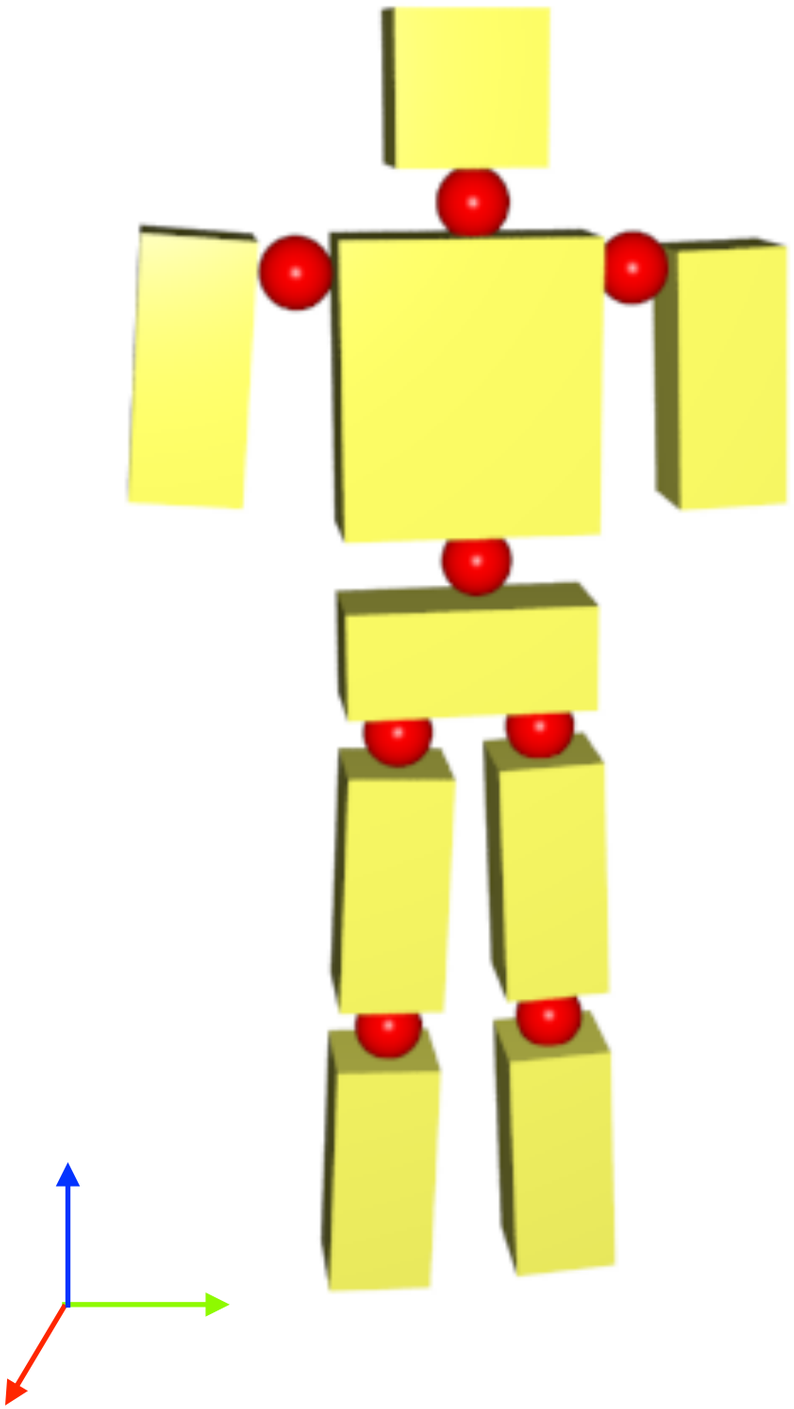
Tree Data Structure Notation



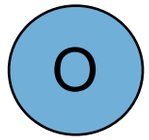
A tree is an abstract data type or data structure implementing this tree that simulates a hierarchical tree structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes.

- A tree consists of nodes and association between nodes
- Every tree has ONE root node
- Every child has only one parent
- Parent can have multiple children

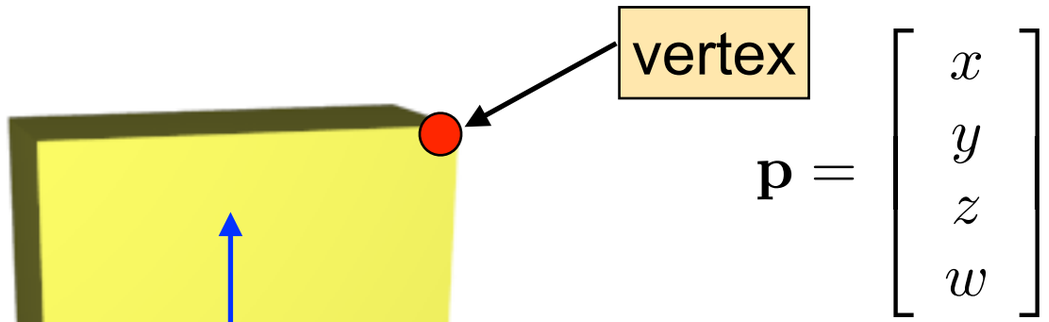
Primitive Nodes

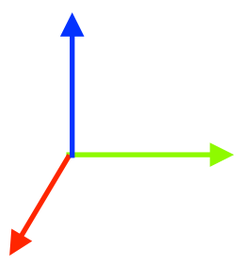


Primitive nodes are instances that render objects on screen. From a graph point of view, those nodes are leaf nodes, which have no further children.

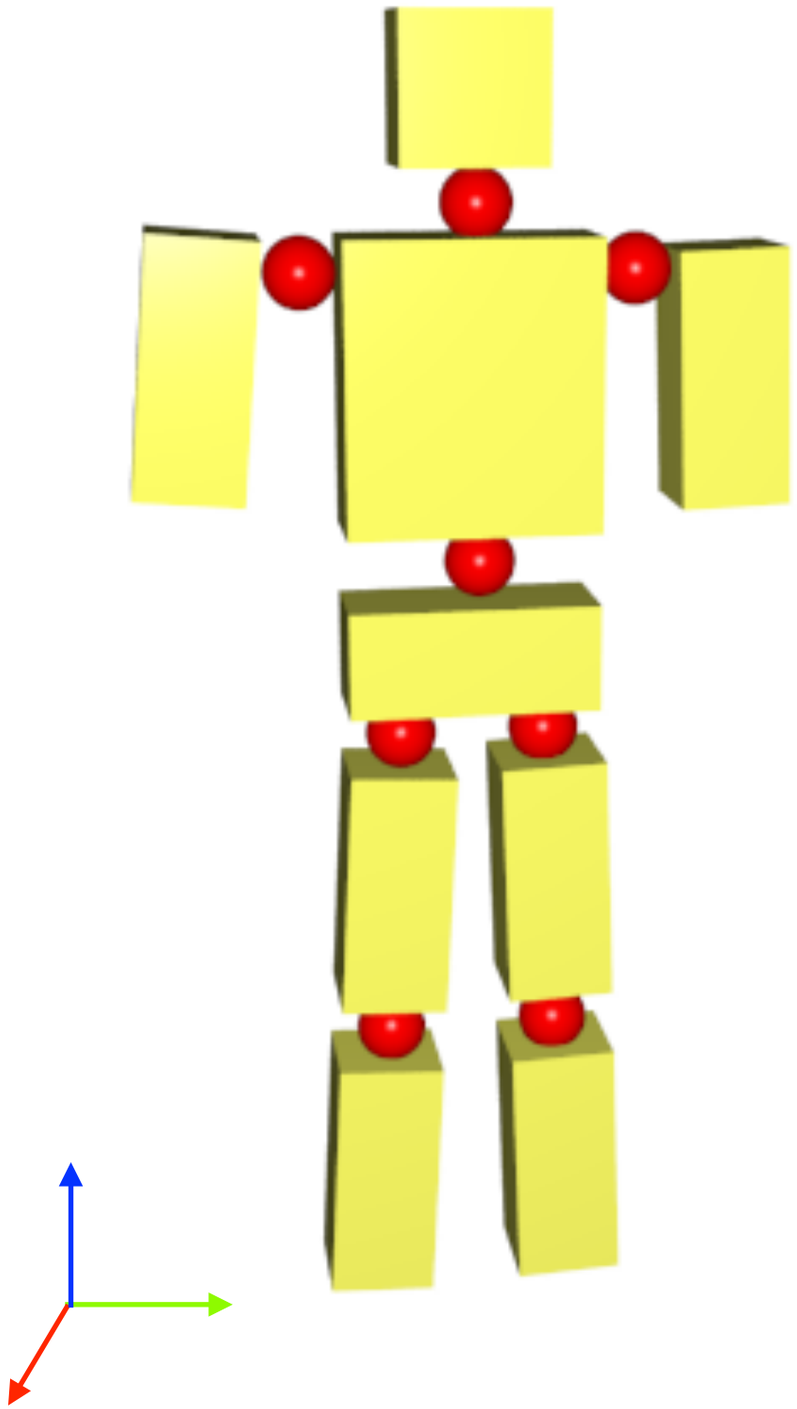
We represent them with a 

(object)



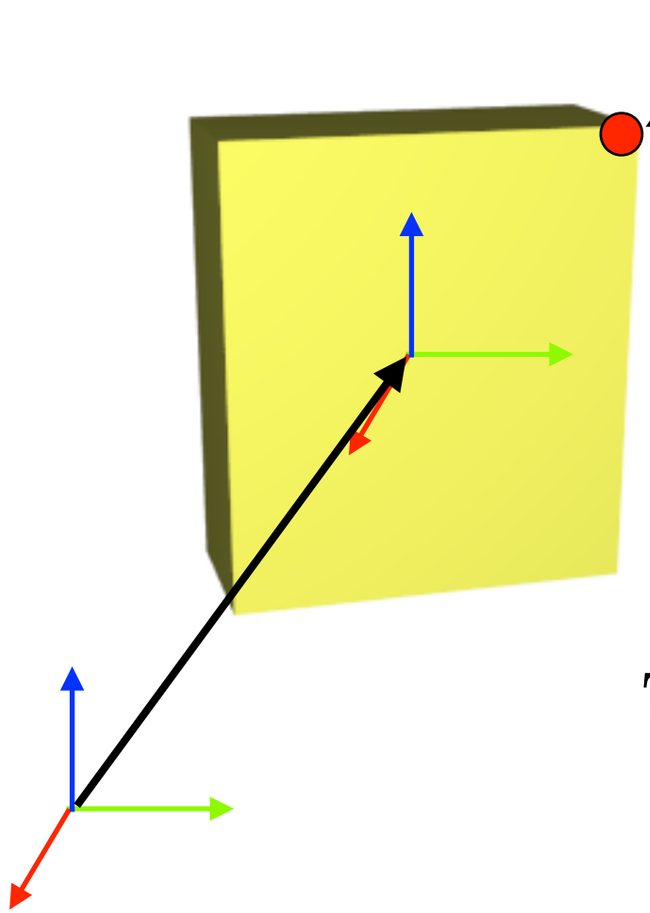


Instance Transformation



Primitive nodes are instances that render objects on screen. From a graph point of view, those nodes are leaf nodes, which have no further children.

We represent them with a T (object)

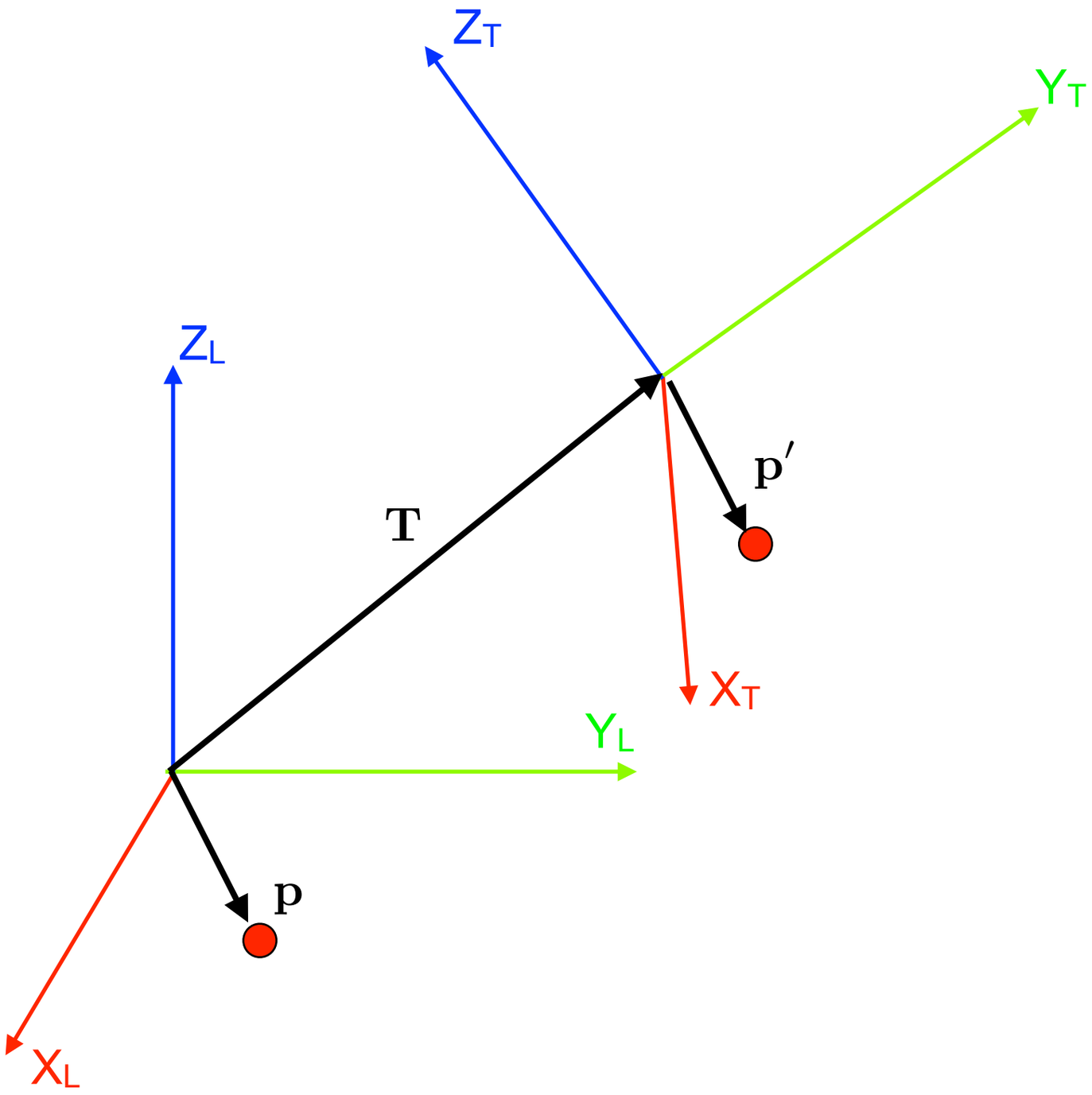


$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & x \\ r_{10} & r_{11} & r_{12} & y \\ r_{20} & r_{21} & r_{22} & z \\ 0 & 0 & 0 & w \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{T}\,\mathbf{p}$$

Transformation Reminder



Every transformation can be considered as a coordinate system transformation: the matrix represents a new coordinate system. Multiplying a point with a matrix moves this point into a new coordinate system T.

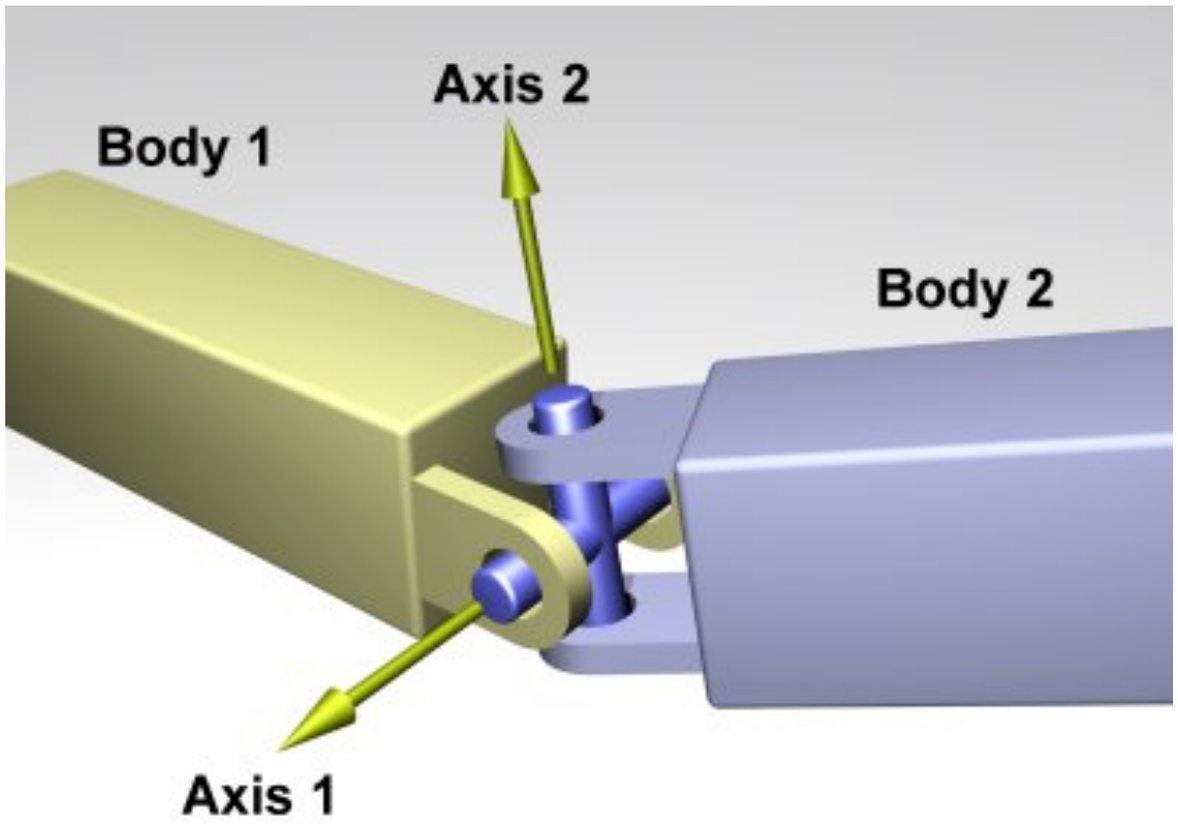
$$\mathbf{T} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & x \\ r_{10} & r_{11} & r_{12} & y \\ r_{20} & r_{21} & r_{22} & z \\ 0 & 0 & 0 & w \end{bmatrix}$$

$$\mathbf{p}' = \mathbf{T}\,\mathbf{p}$$

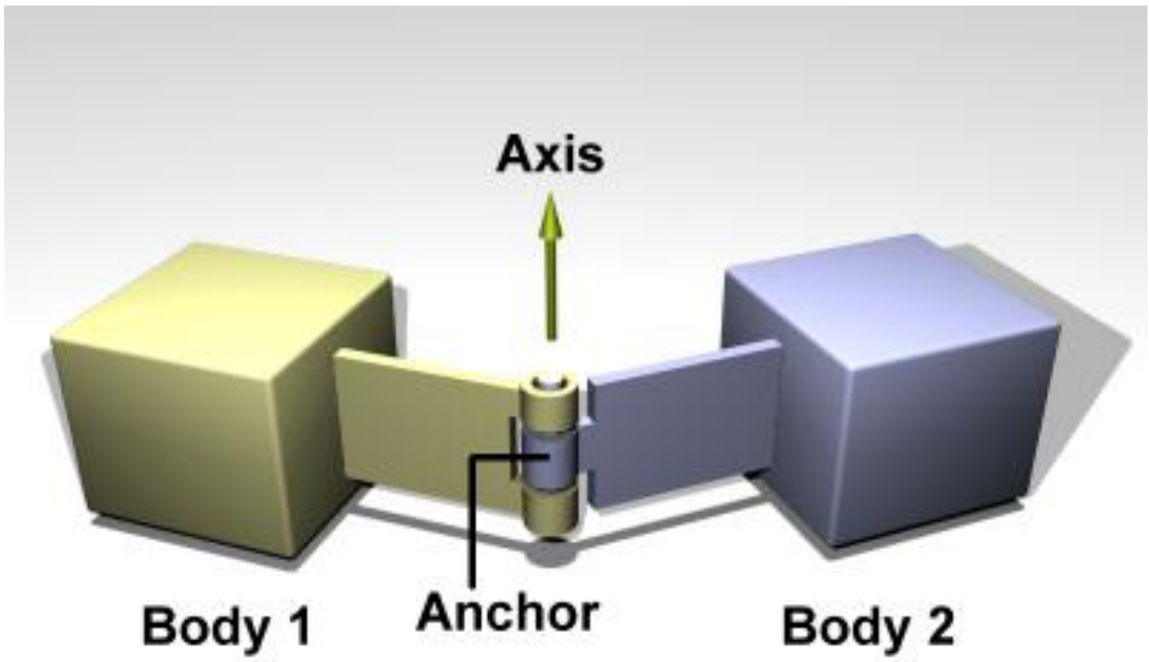
All further transformation are with respect to this coordinate system.

Joint Transformation

Joint transformations are used to simulate movement at a joint.



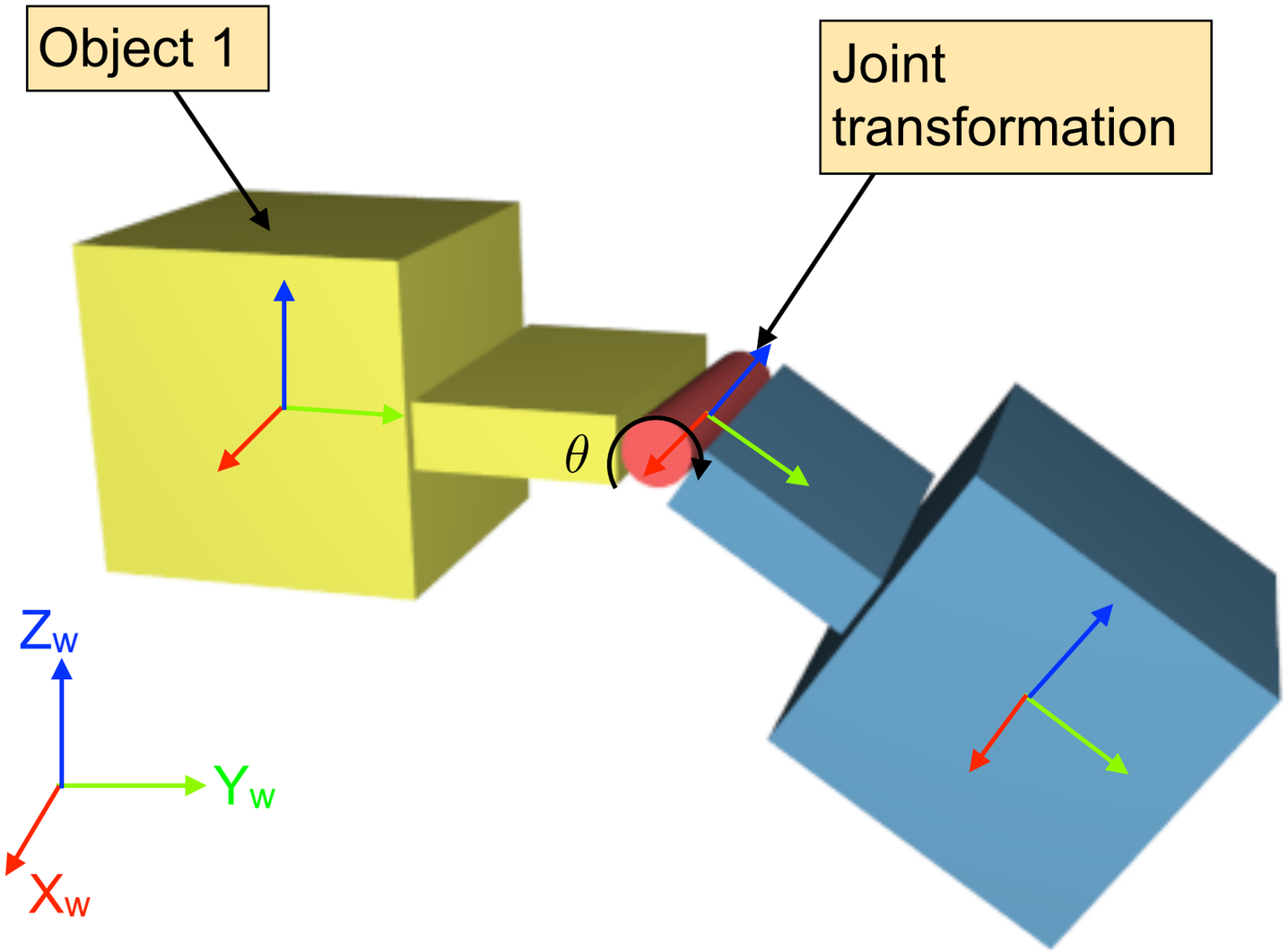
Universal joint (source: odm)



Hinge joint (source: odm)

- Different degrees of freedom (rotational and/or linear)
- Simulate the kinematics of an object
- Can be constraint

Joint Transformation

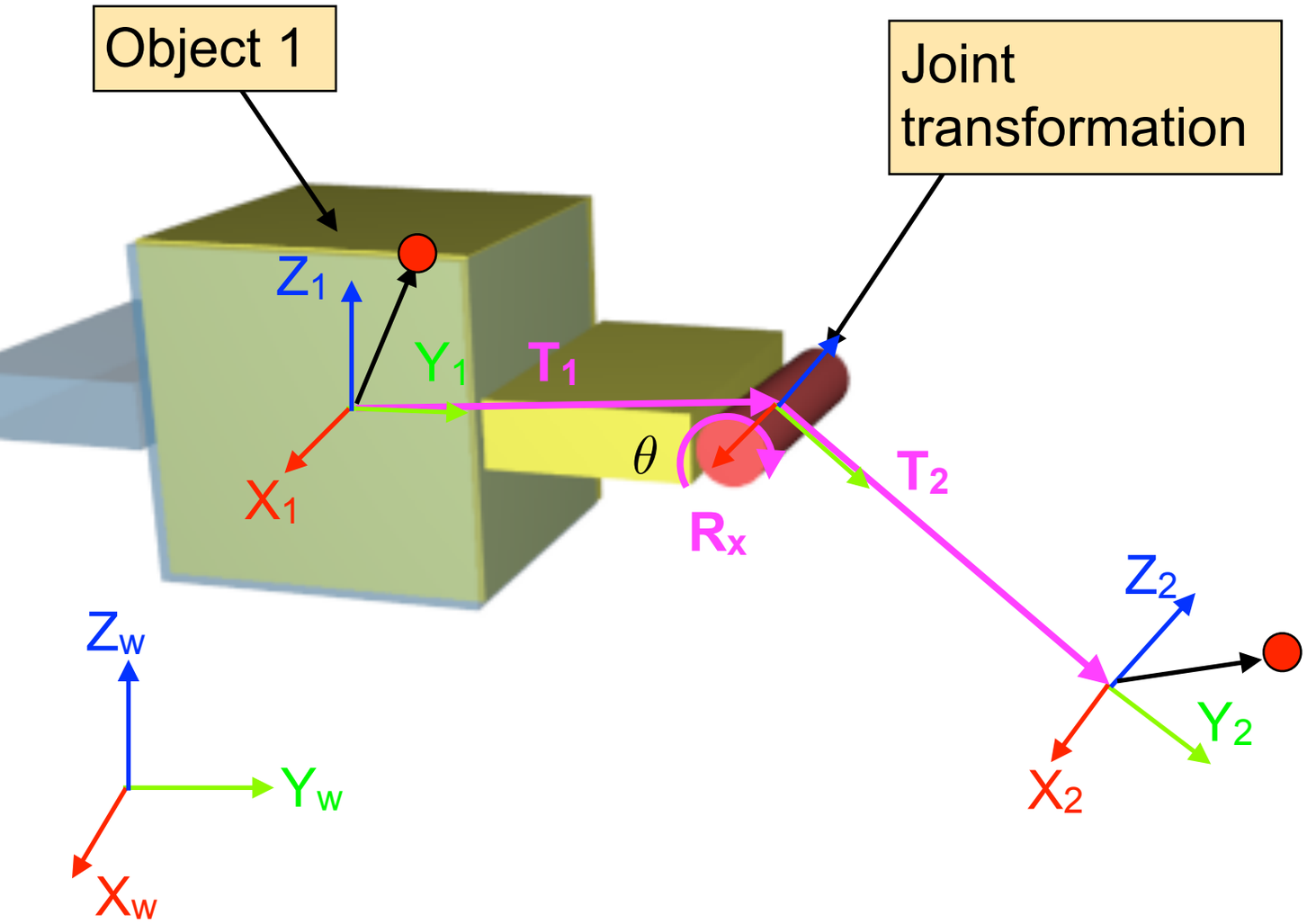


A joint transformation is a rotational and or linear transformation matrix along with a motion restriction, if required.

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin\theta & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\theta = \{\theta_{min}, \theta_{max}\}$$

Joint Transformation



A joint transformation is a rotational and or linear transformation matrix along with a motion restriction, if required.

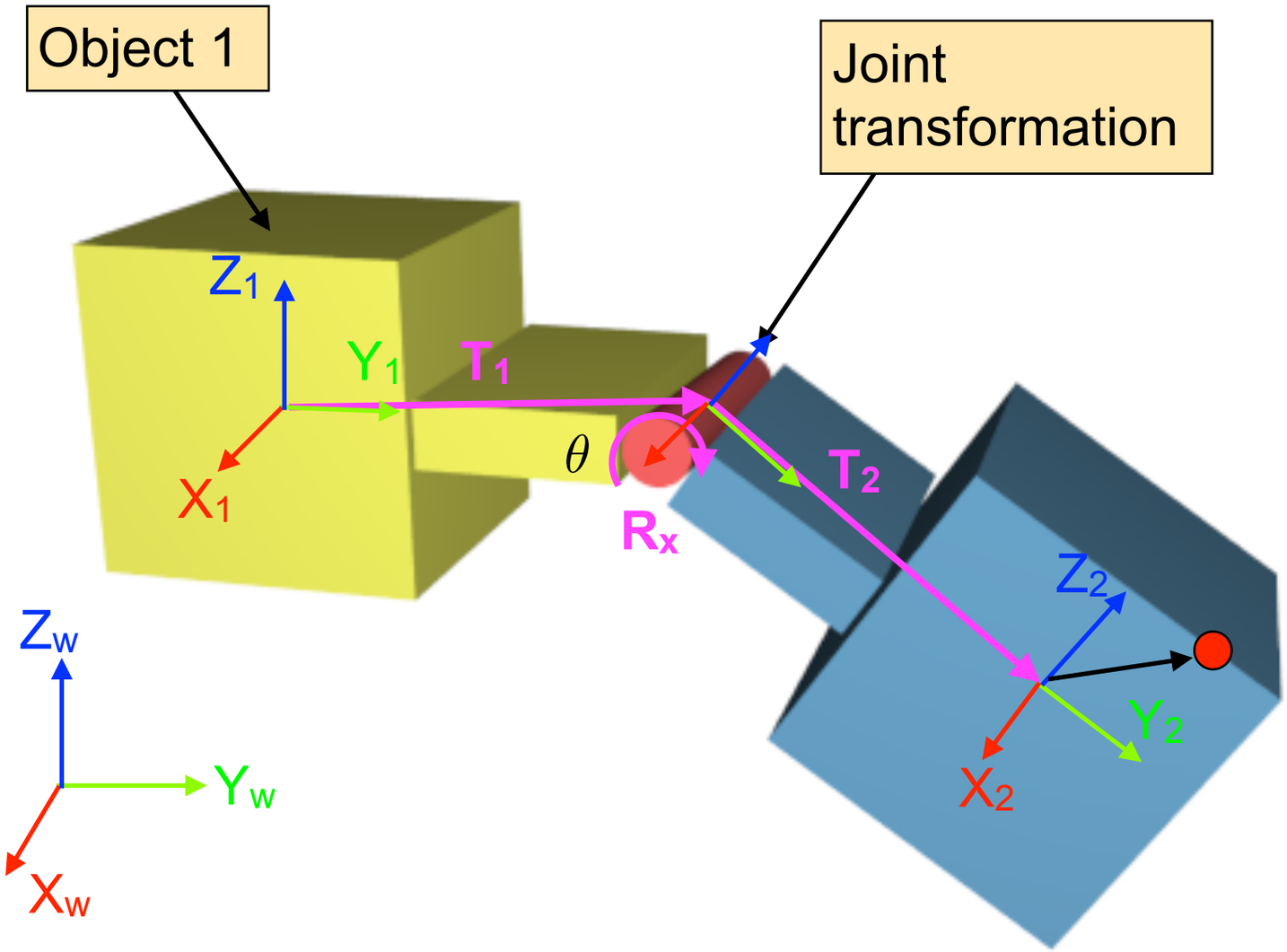
$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin\theta & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\theta = \{\theta_{min}, \theta_{max}\}$$

- Consider, the blue box is described in the first local coordinate system.
- We like to move it into the second one using a joint constraint, with

$$\mathbf{p}' = \mathbf{T}_2\ \mathbf{R}_x\ \mathbf{T}_1\ \mathbf{p}$$

Joint Transformation



A joint transformation is a rotational and or linear transformation matrix along with a motion restriction, if required.

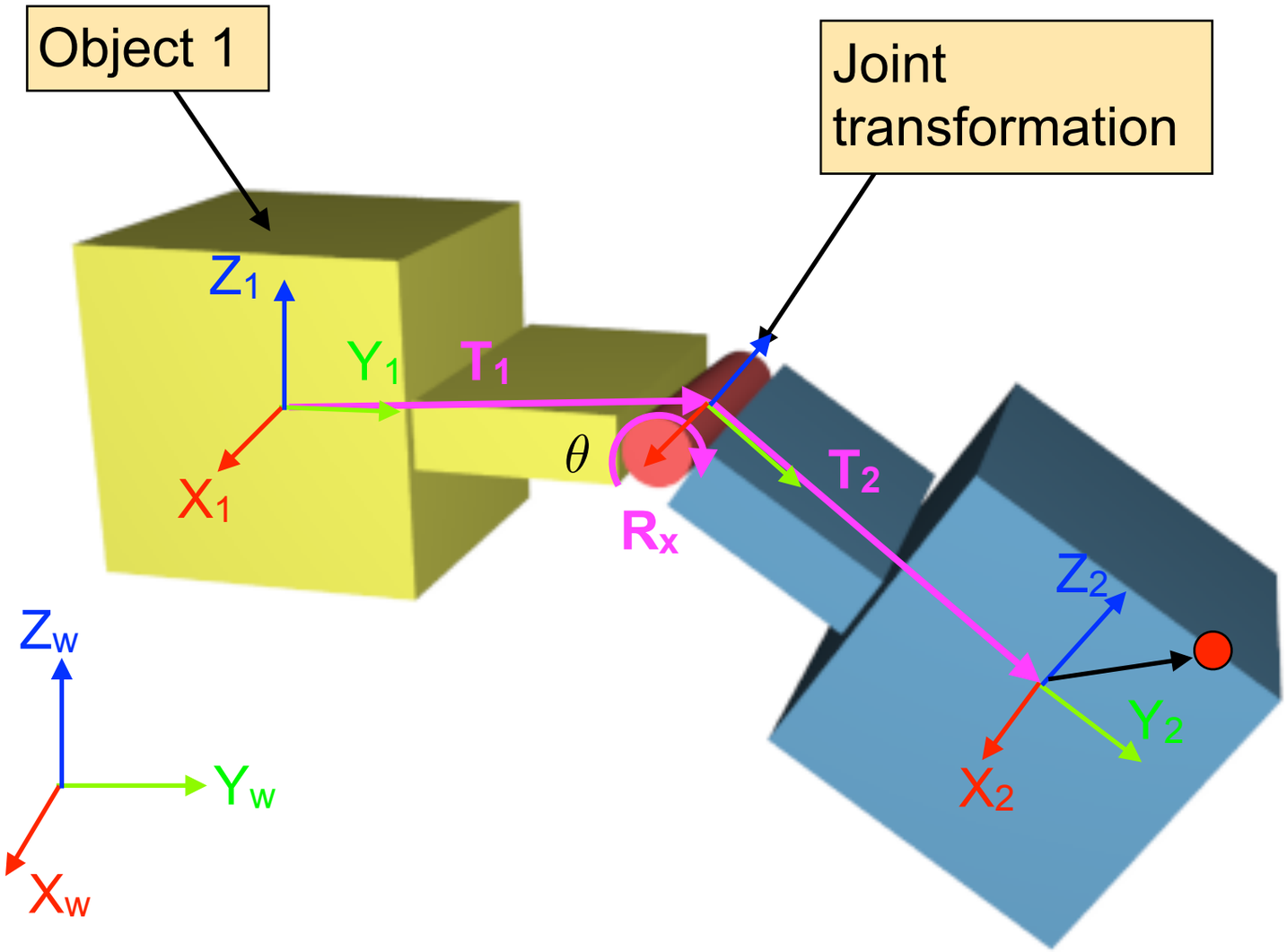
$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin\theta & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\theta = \{\theta_{min}, \theta_{max}\}$$

- Consider, the blue box is described in the first local coordinate system.
- We like to move it into the second one using a joint constraint, with

$$\mathbf{p}' = \mathbf{T}_2\ \mathbf{R}_x\ \mathbf{T}_1\ \mathbf{p}$$

Joint Transformation



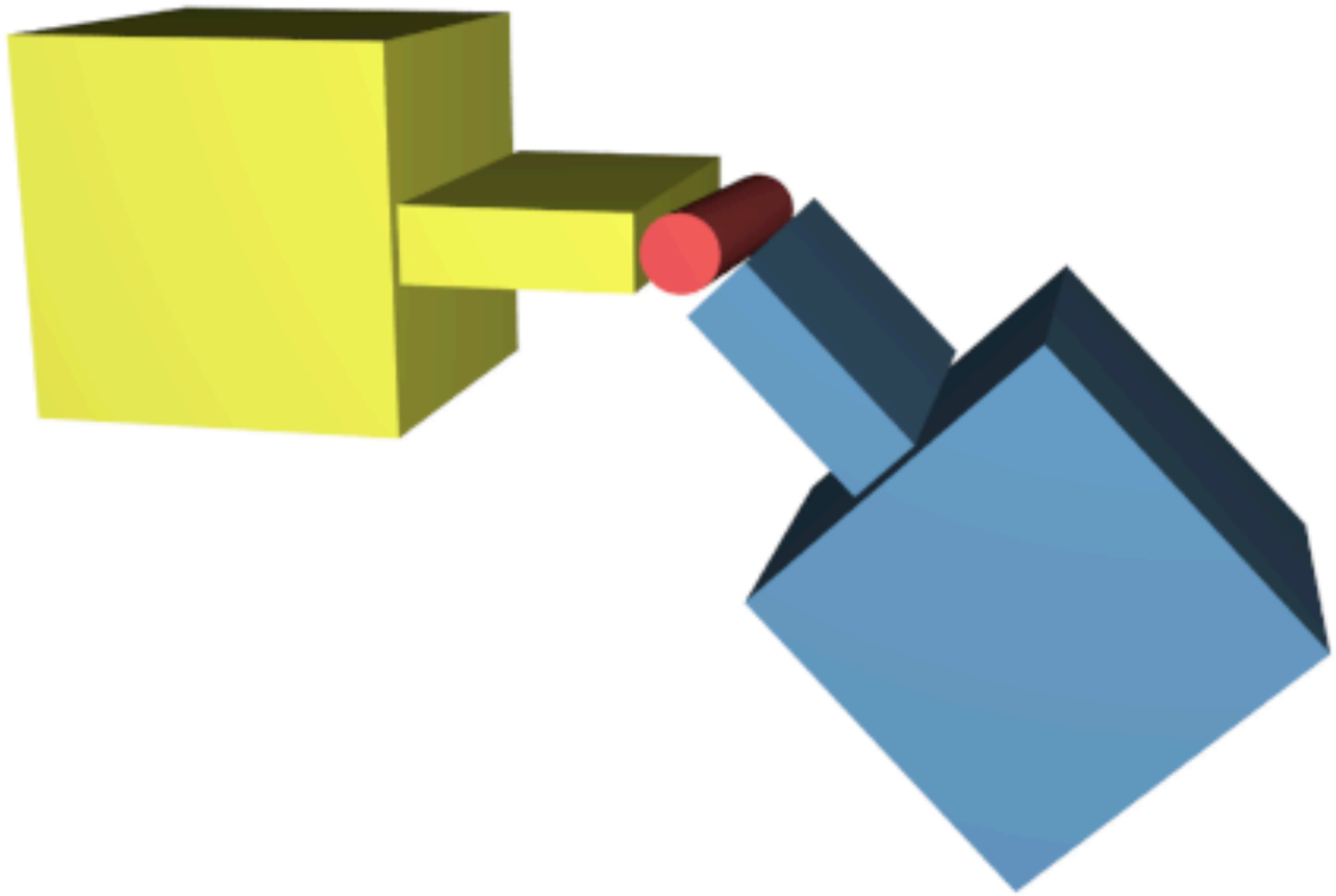
A joint transformation is a rotational and or linear transformation matrix along with a motion restriction, if required.

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin\theta & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

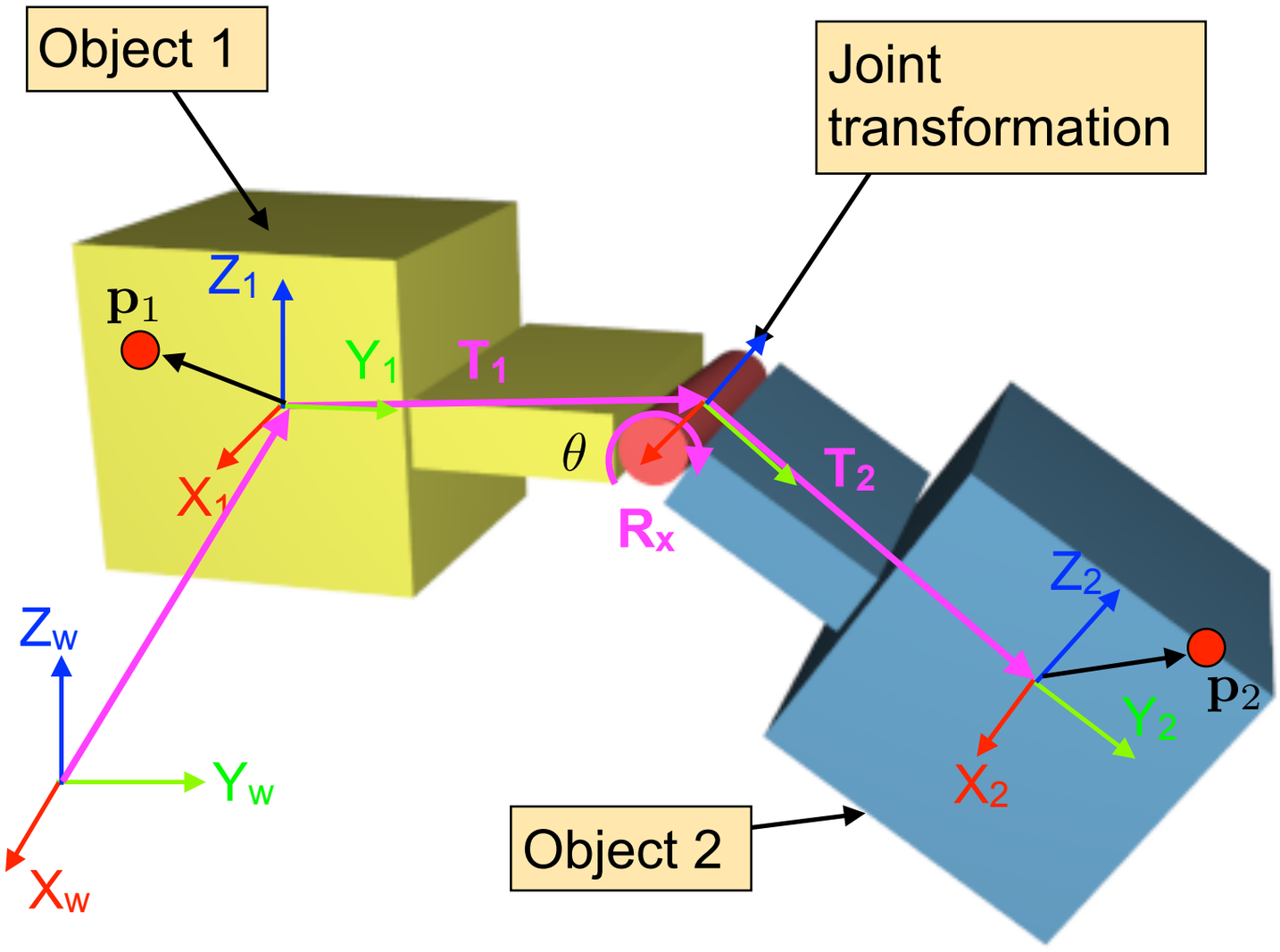
$$\theta = \{\theta_{min}, \theta_{max}\}$$

We represent them with a (J) (joint)

Video



Moving Object 1



Consider, we use a transformation matrix T_{w1} to move the first object around. All points of object 1 are transformed with

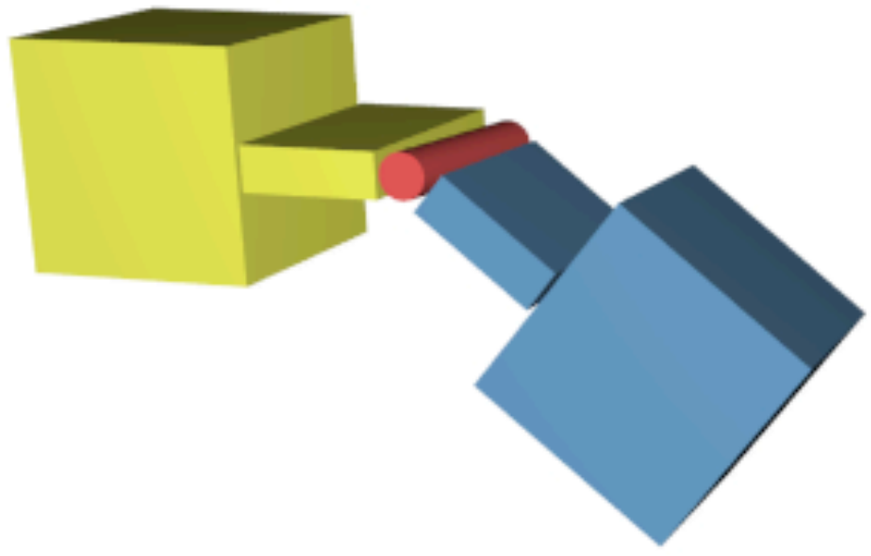
$$\mathbf{p}_1' = \mathbf{T}_{w1}\ \mathbf{p}_1$$

Since object 2, respectively all points of object two also depend on Tw1, all points follow the same transformation.

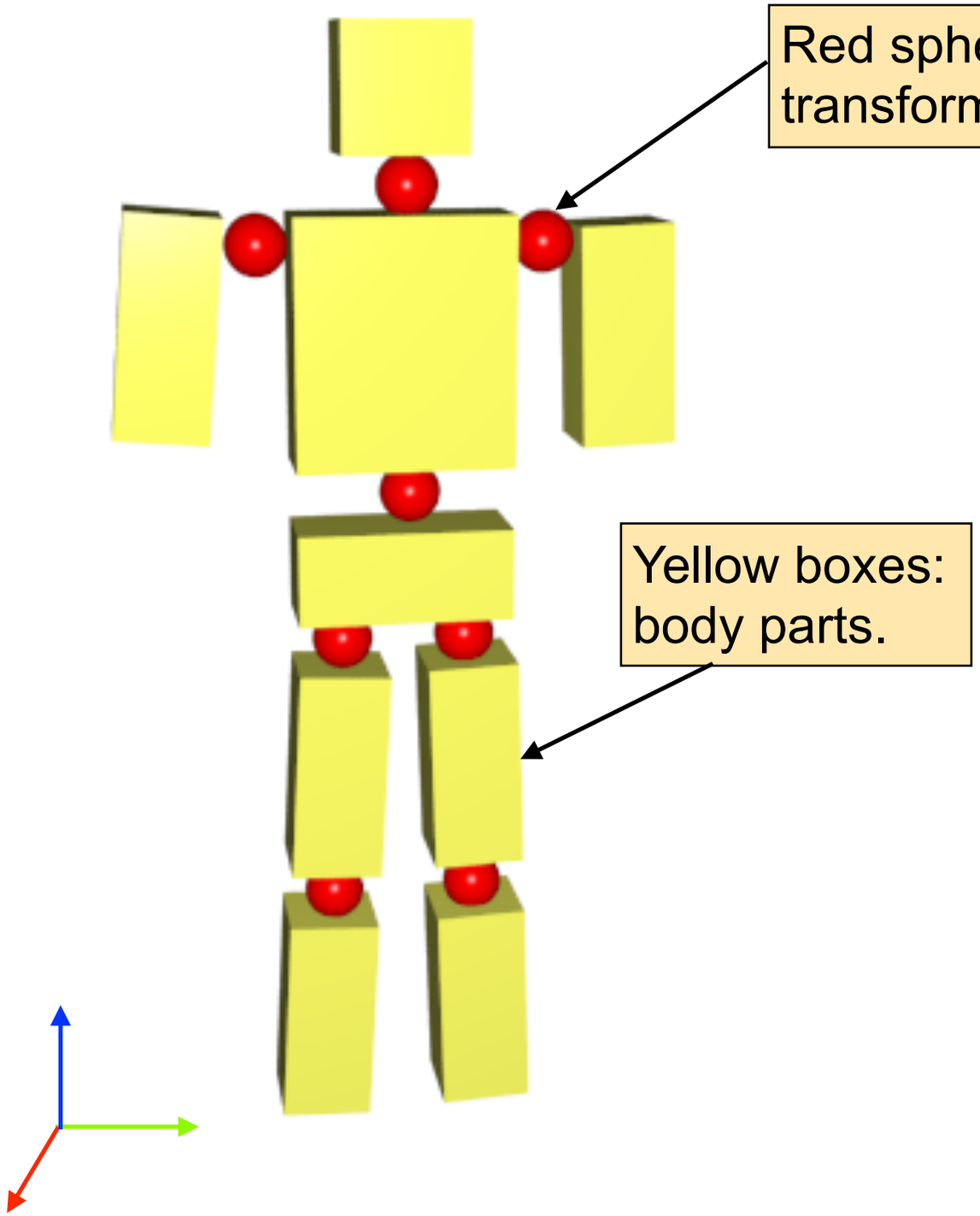
$$\mathbf{p}_2' = \mathbf{T}_2\ \mathbf{R}_x\ \mathbf{T}_1\ \mathbf{T}_{w1}\ \mathbf{p}_2$$

This allows us to build a hierarchical representation of transformed objects.

Video



Tree of objects



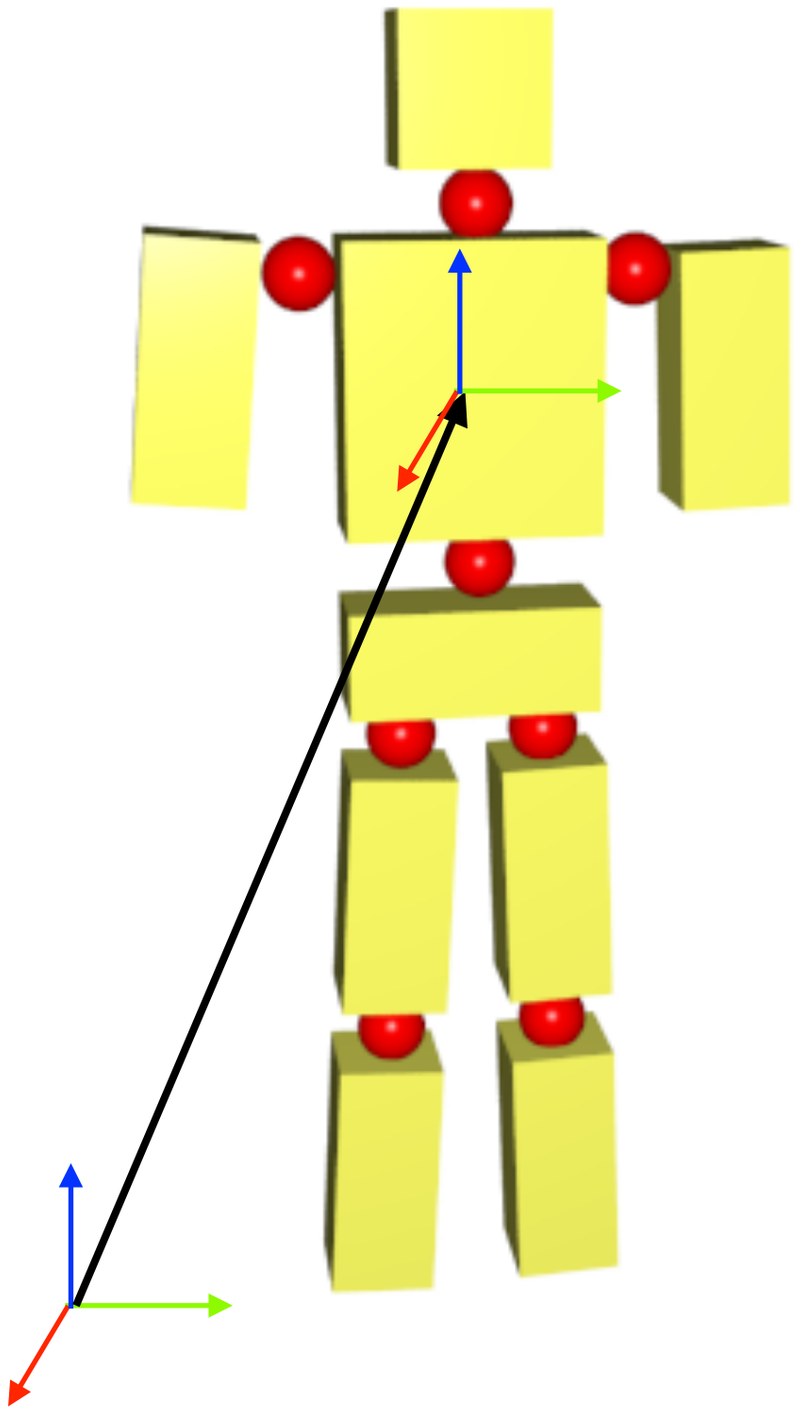
J joint

T transformation

O object

- Object nodes are leaf nodes! They do not take any additional children.
- Joint and transformation nodes are inner nodes.

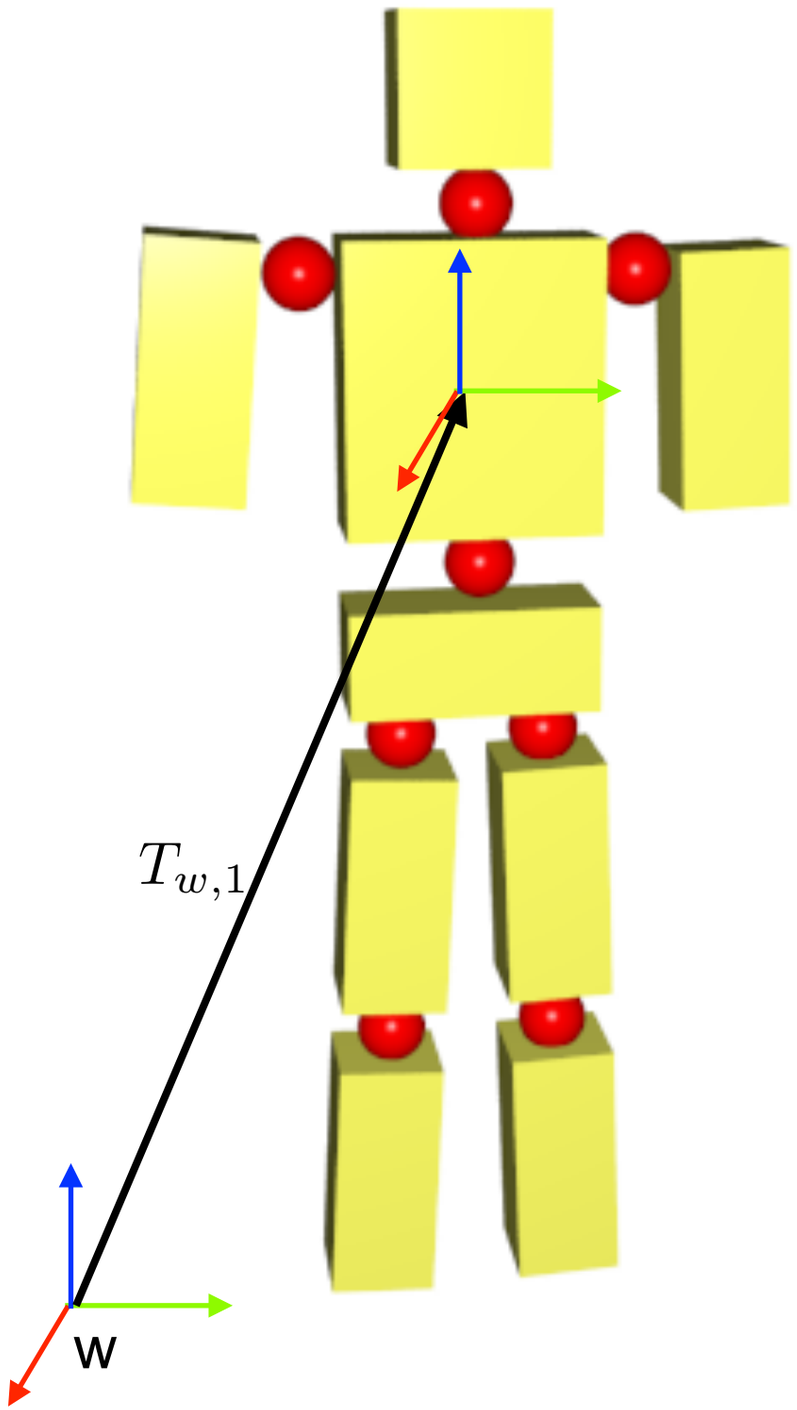
Tree of objects

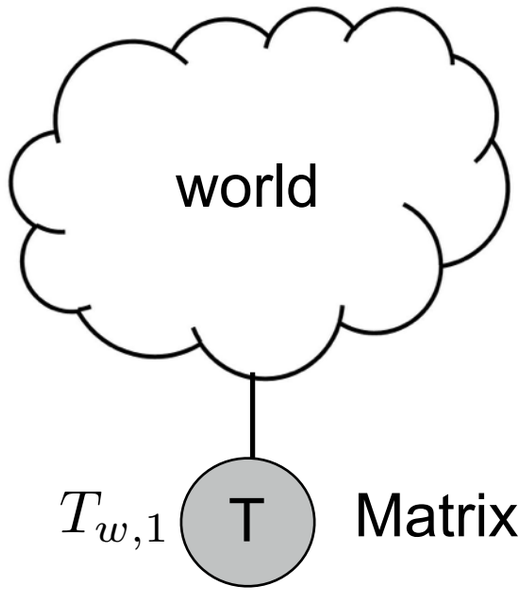


T

transformation for the world
(identity matrix)

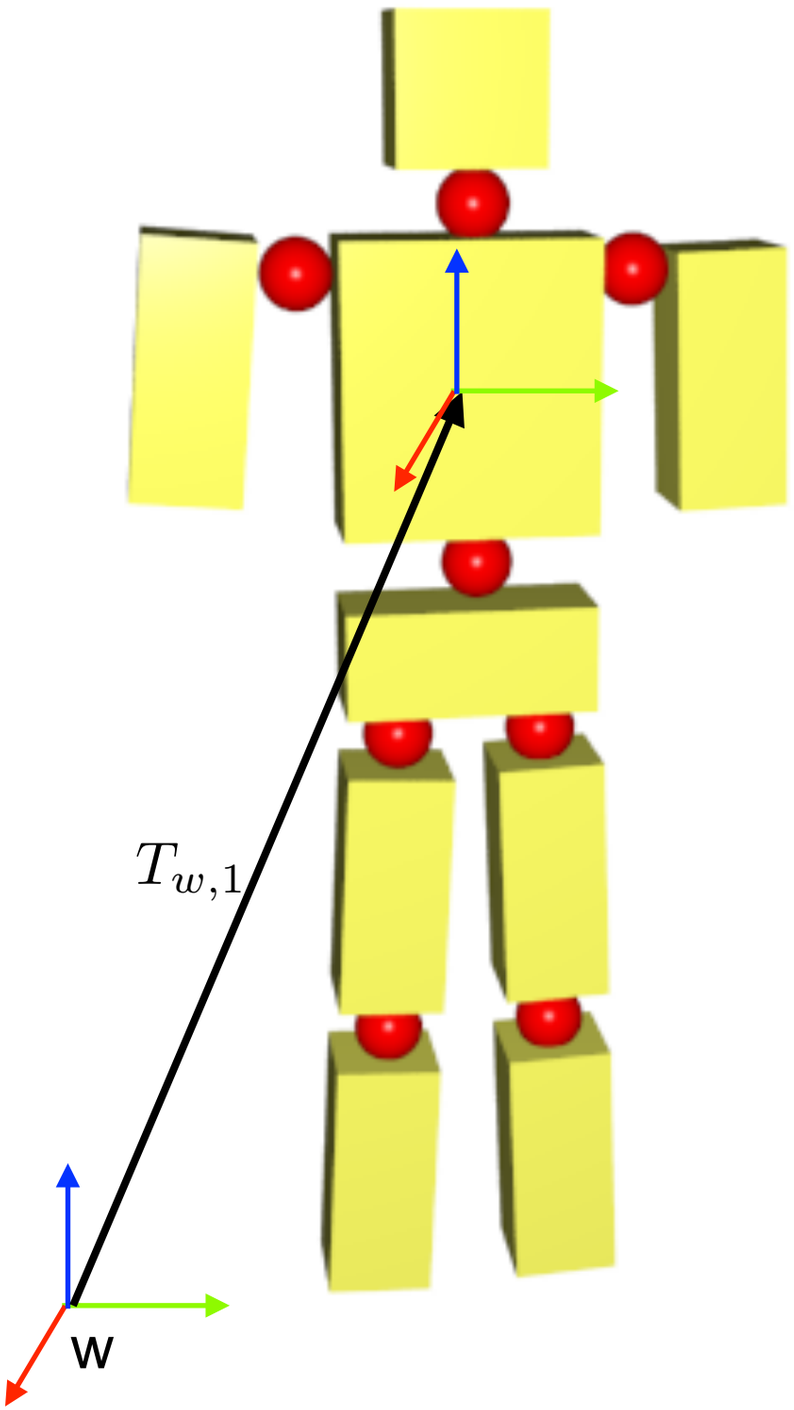
Tree of objects

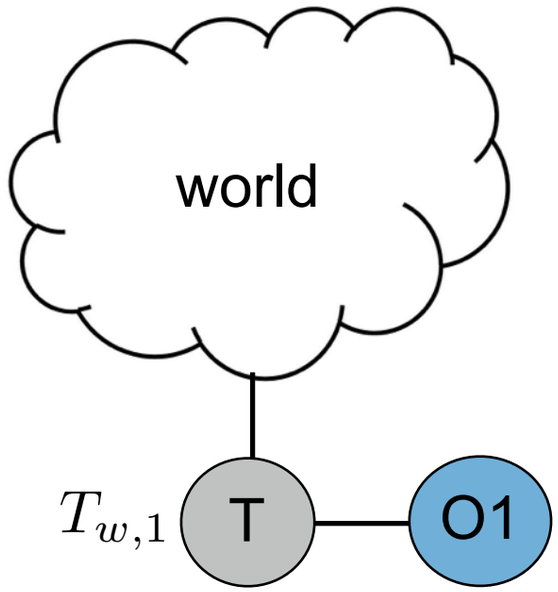




Matrix to place our object in the world

Tree of objects

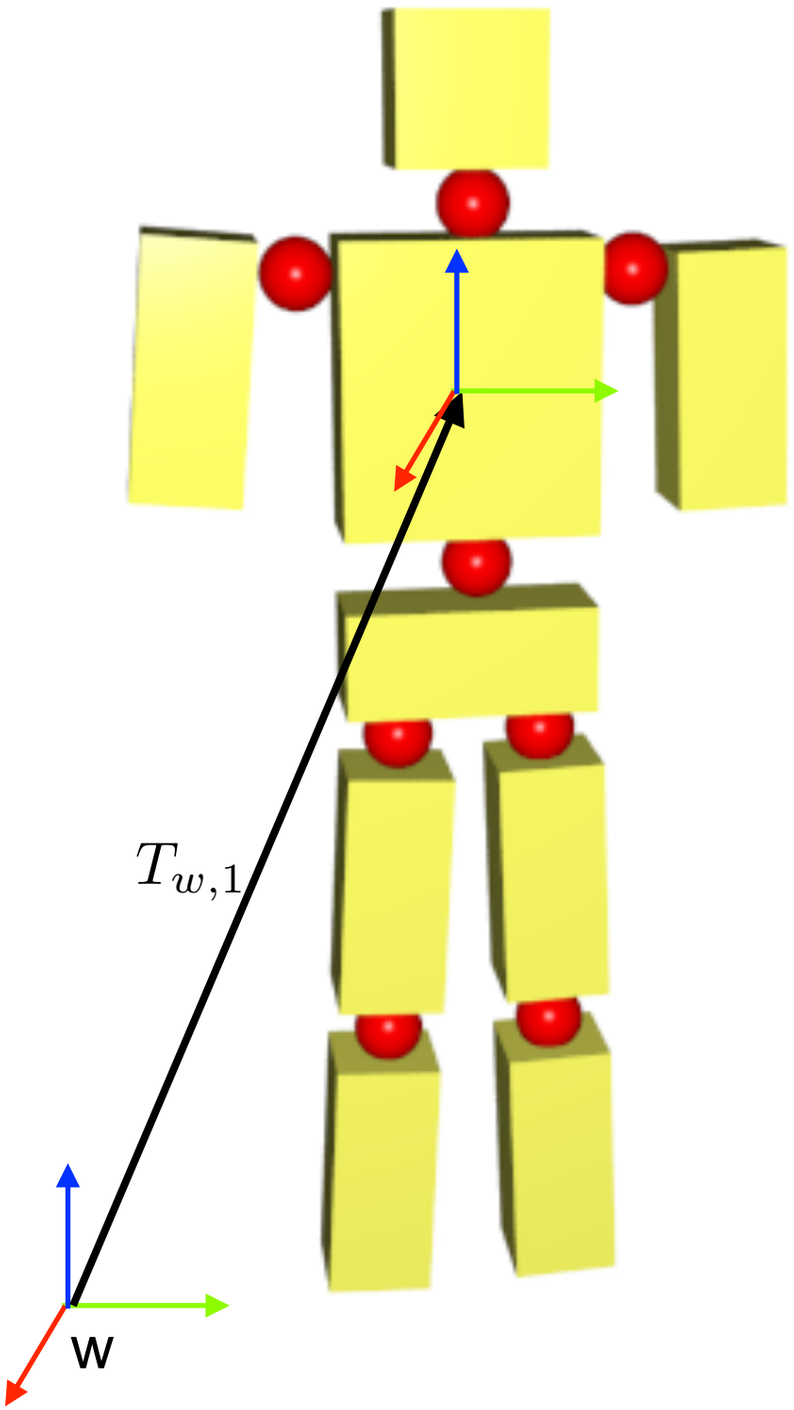


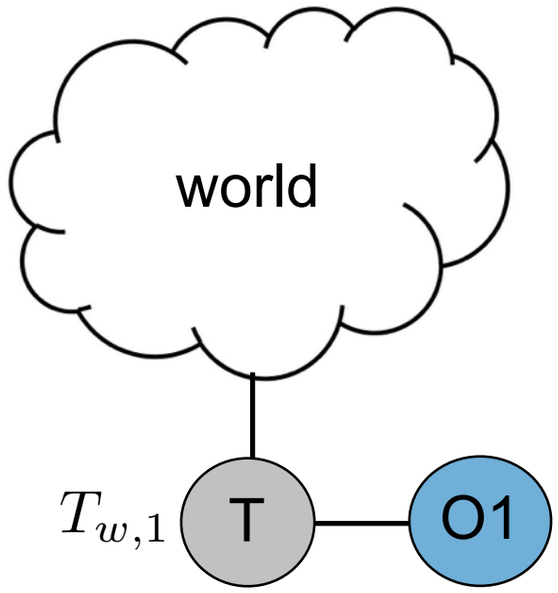


$$\mathbf{p}'_1 = T_{w,1}\ \mathbf{p}_1$$

Tree of objects

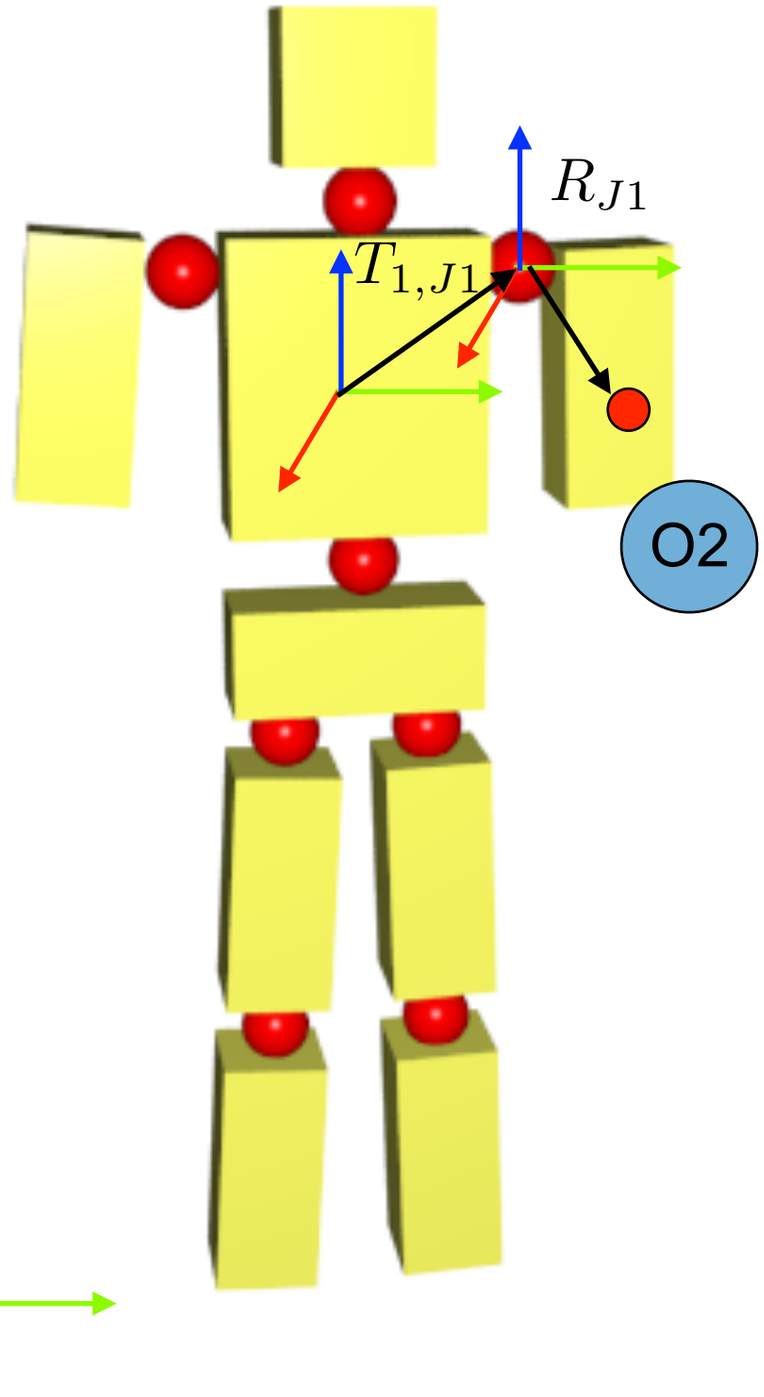
ARLAB

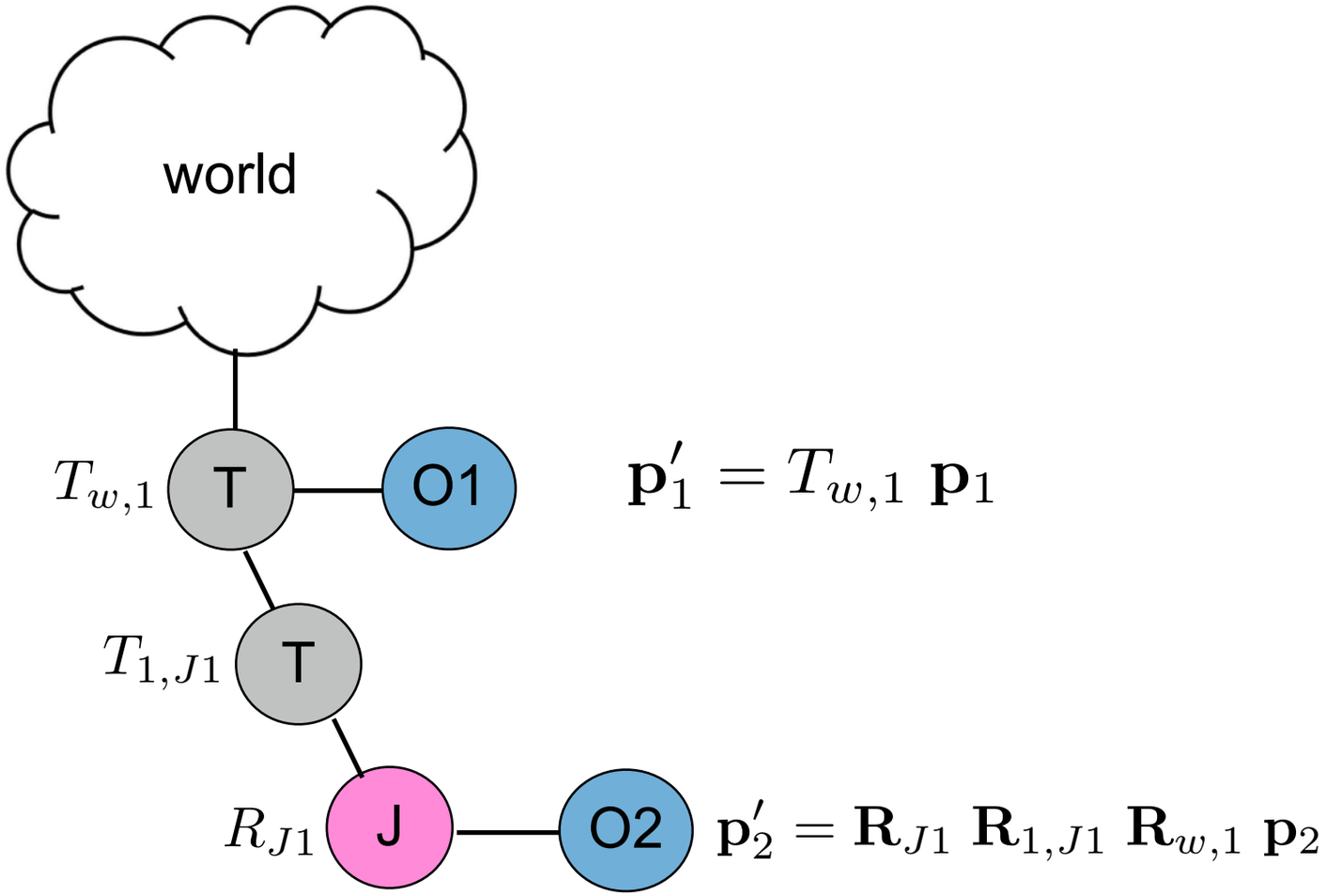




$$\mathbf{p}'_1 = T_{w,1}\ \mathbf{p}_1$$

Tree of objects



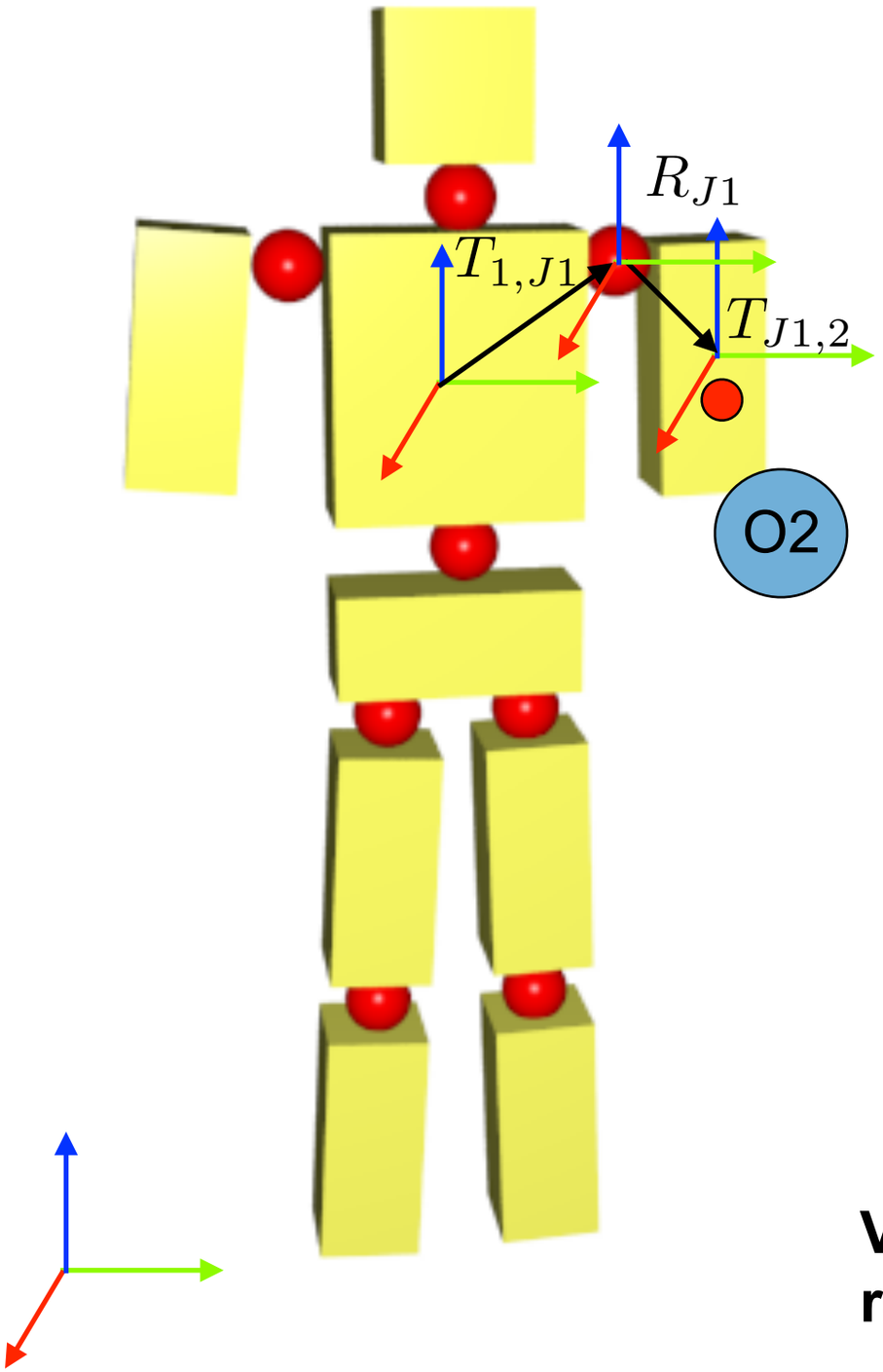


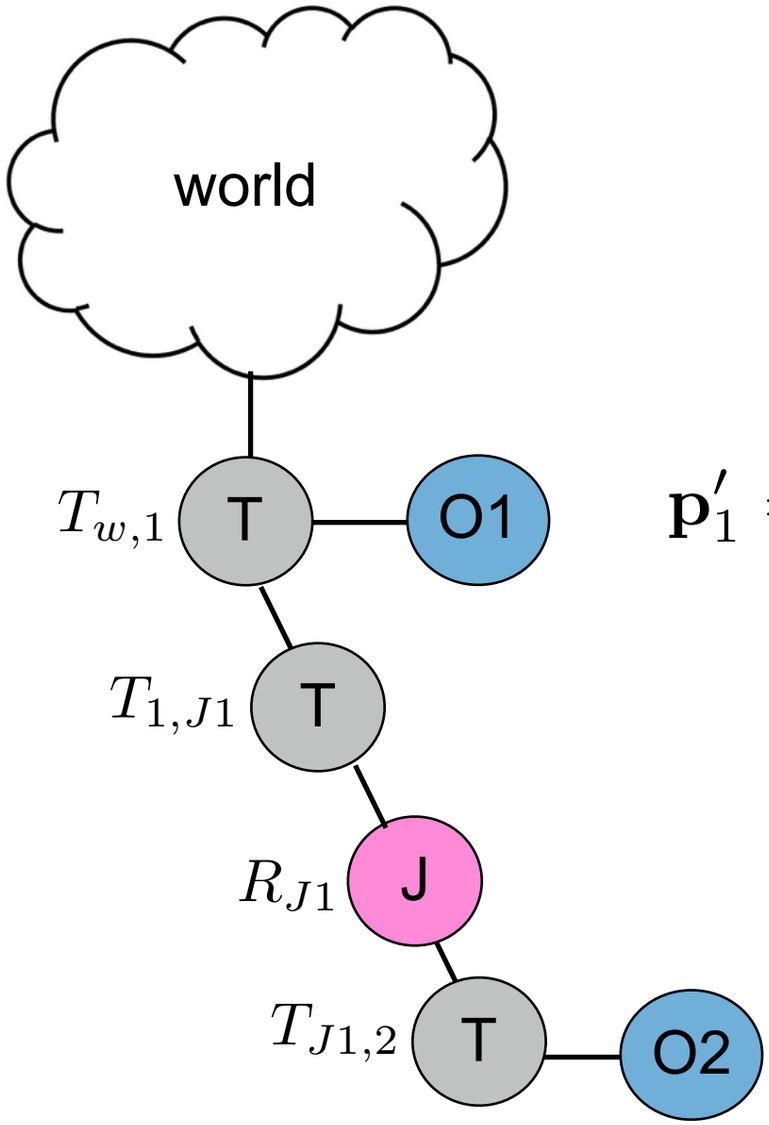
$$\mathbf{p}'_1 = T_{w,1}\ \mathbf{p}_1$$

$$\mathbf{p}'_2 = \mathbf{R}_{J1}\ \mathbf{R}_{1,J1}\ \mathbf{R}_{w,1}\ \mathbf{p}_2$$

Version 1: represent all point with respect to R_{J1}

Tree of objects



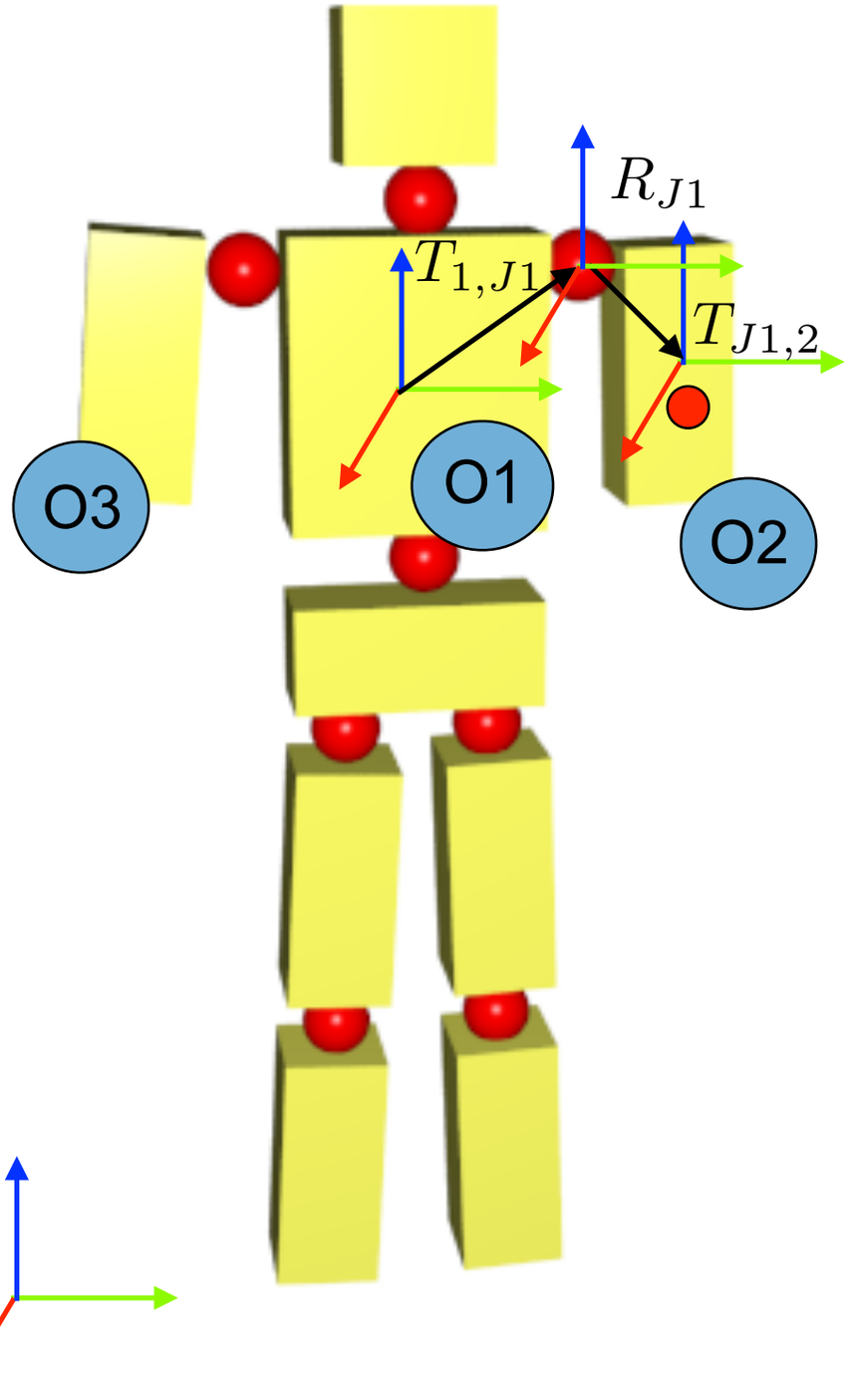


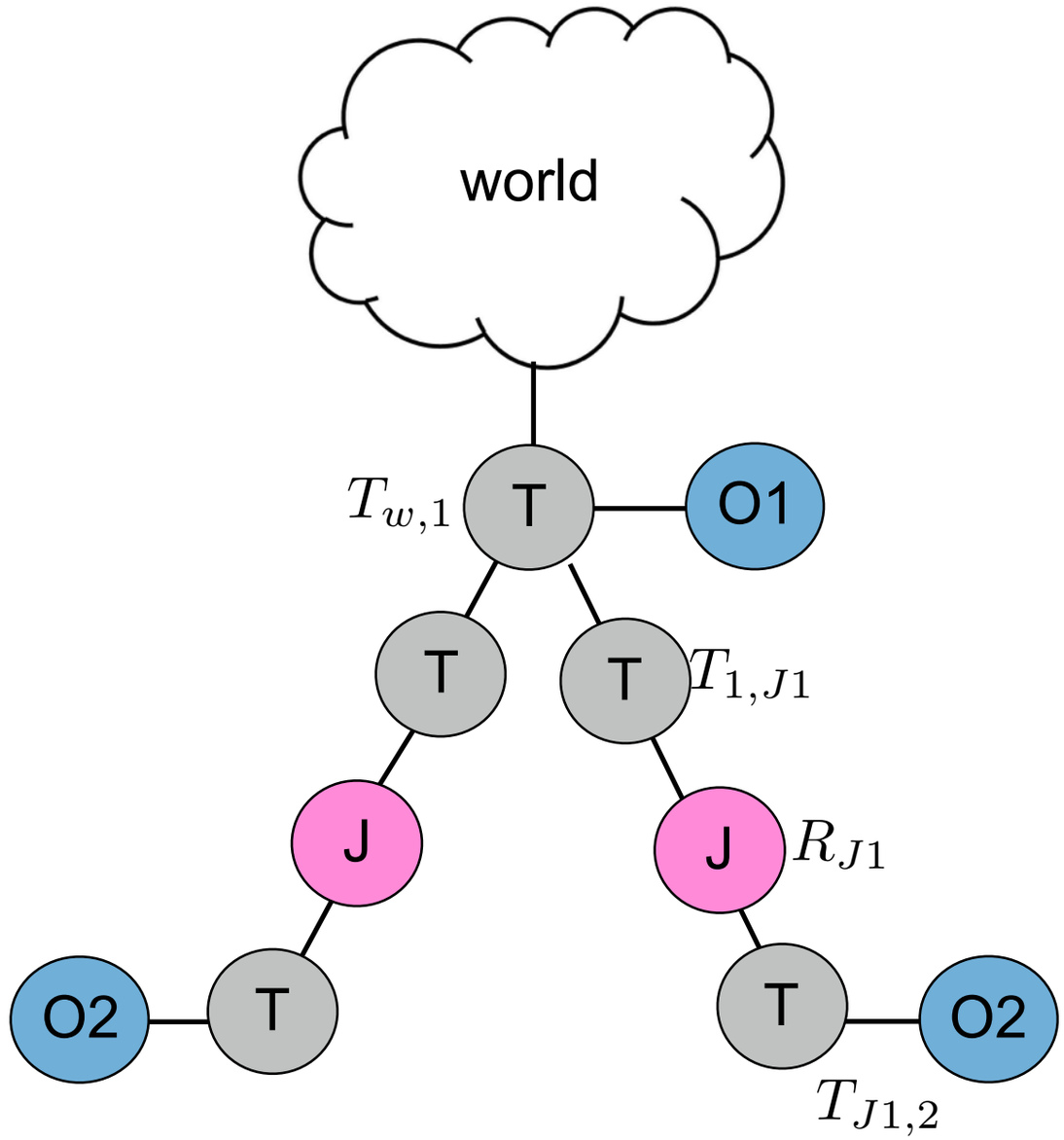
$$\mathbf{p}'_1 = T_{w,1}\ \mathbf{p}_1$$

$$\mathbf{p}'_2 = \mathbf{R}_{J1}\ \mathbf{R}_{1,J1}\ \mathbf{R}_{w,1}\ \mathbf{T}_{J1,2}\ \mathbf{p}_2$$

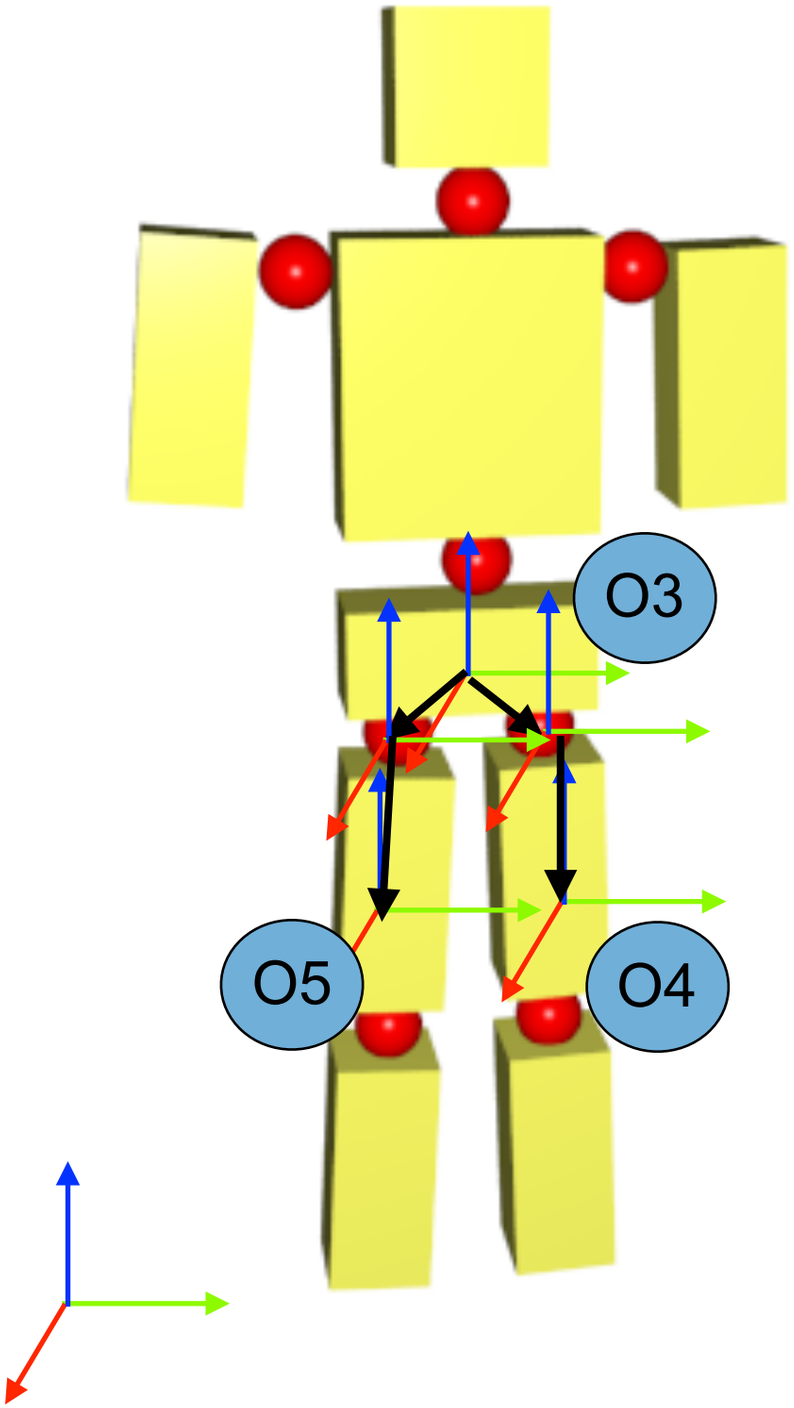
Version 2: add an additional transformation and represent all point with respect to $T_{J1,2}$

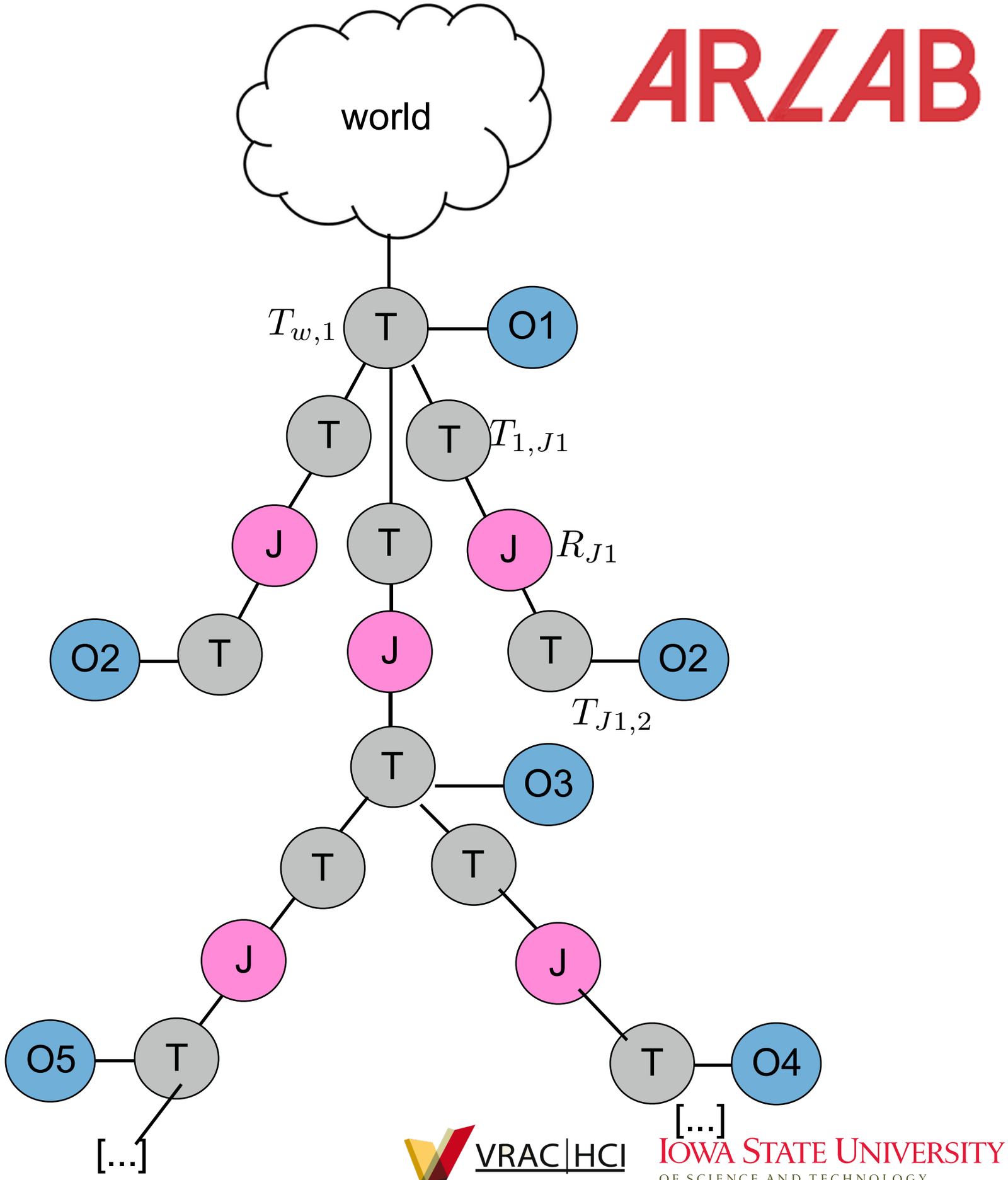
Tree of objects





Tree of objects



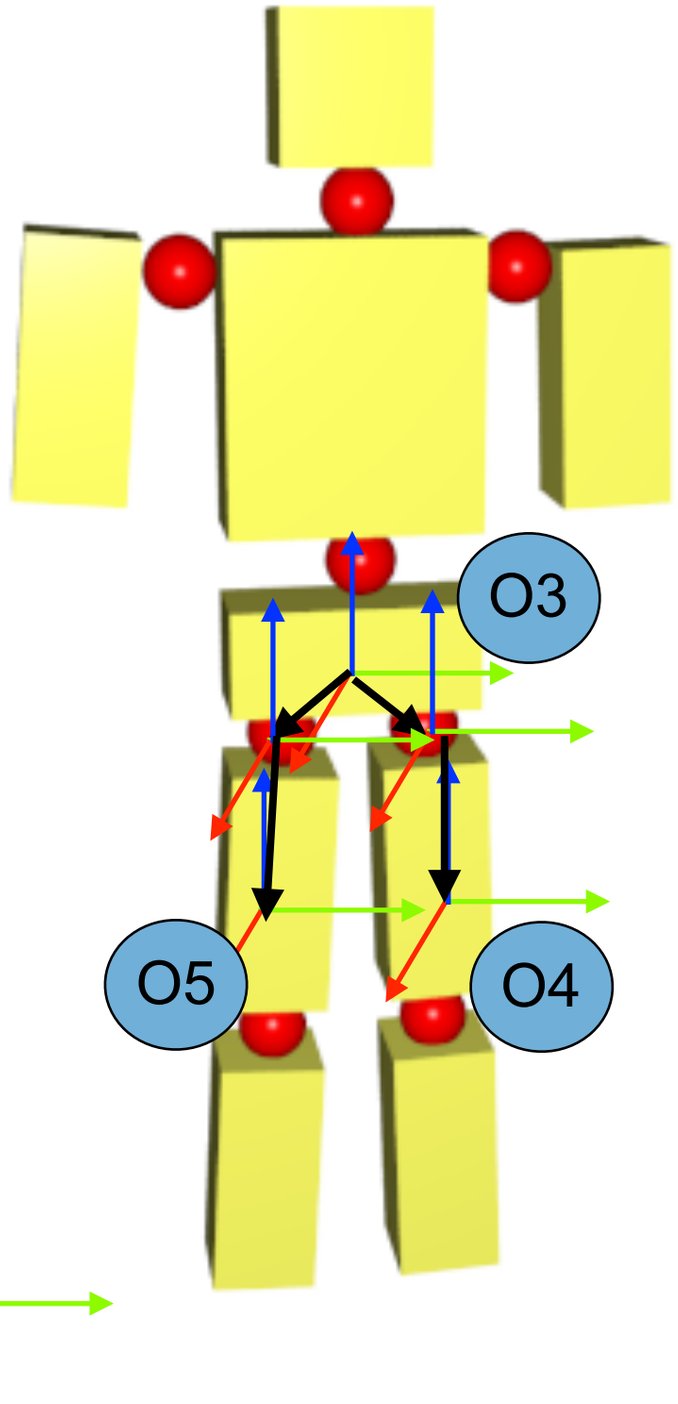


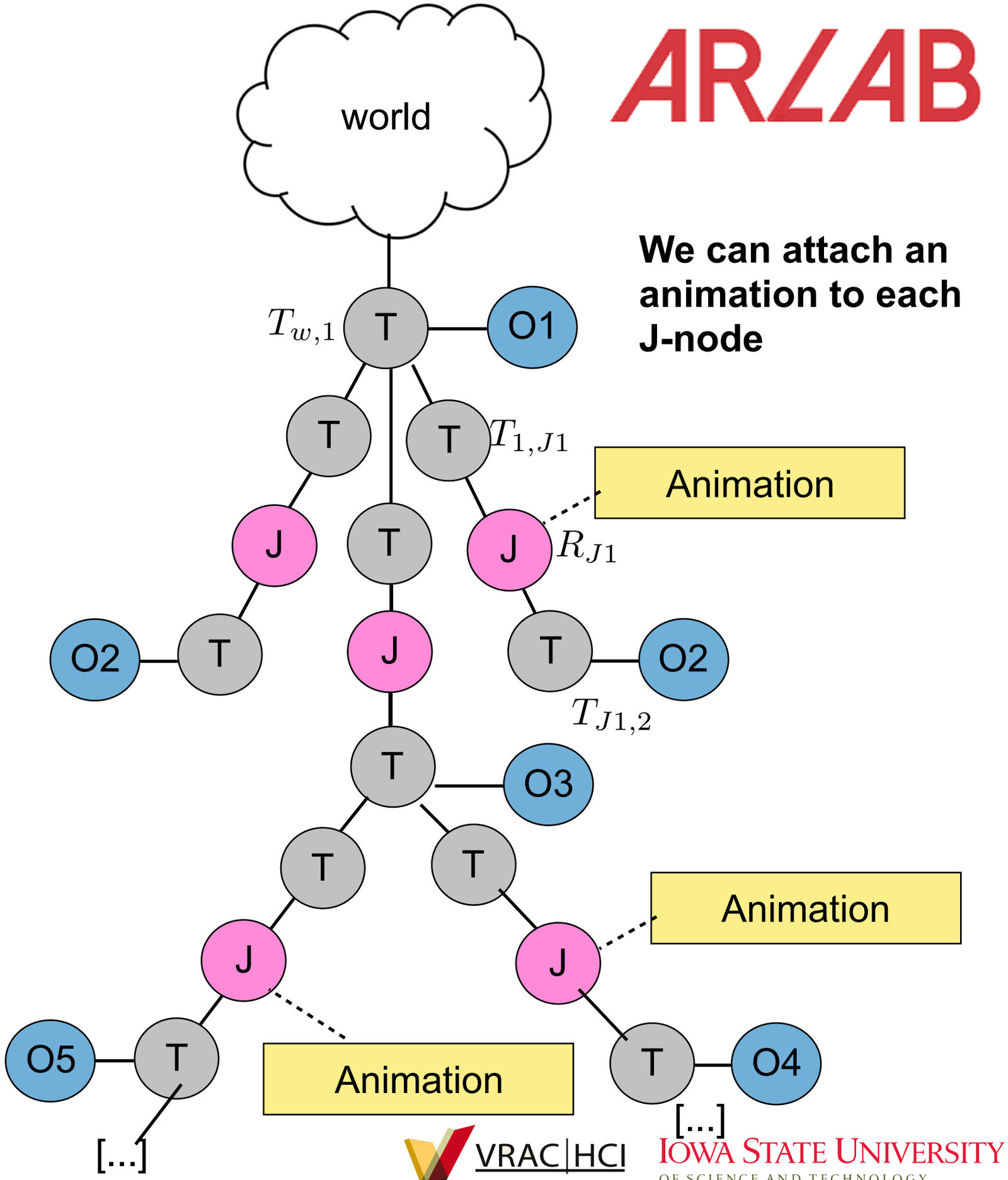
ARLAB

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Tree of objects





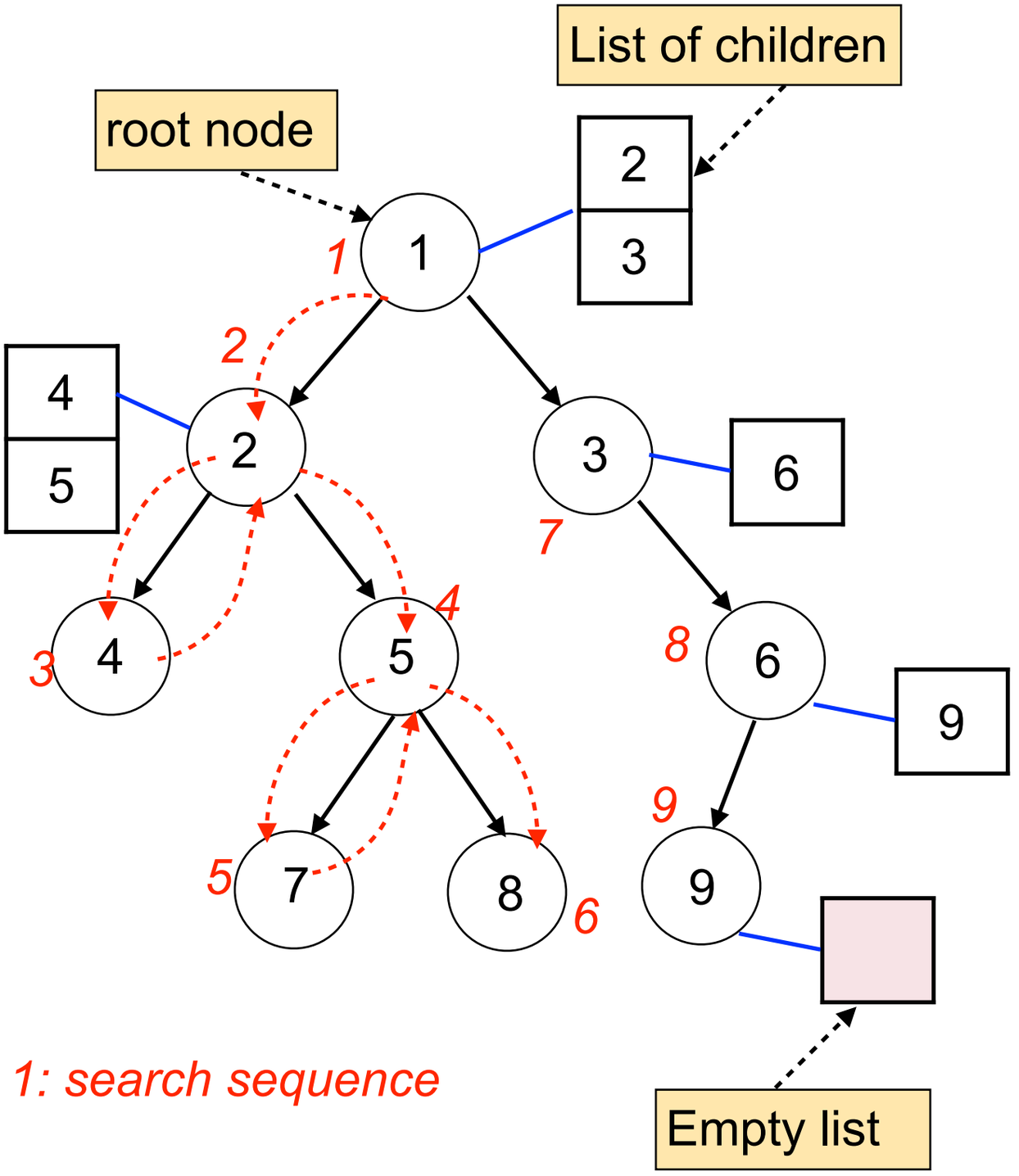
We can attach an animation to each J-node

ARLAB

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Tree Traversal Operation

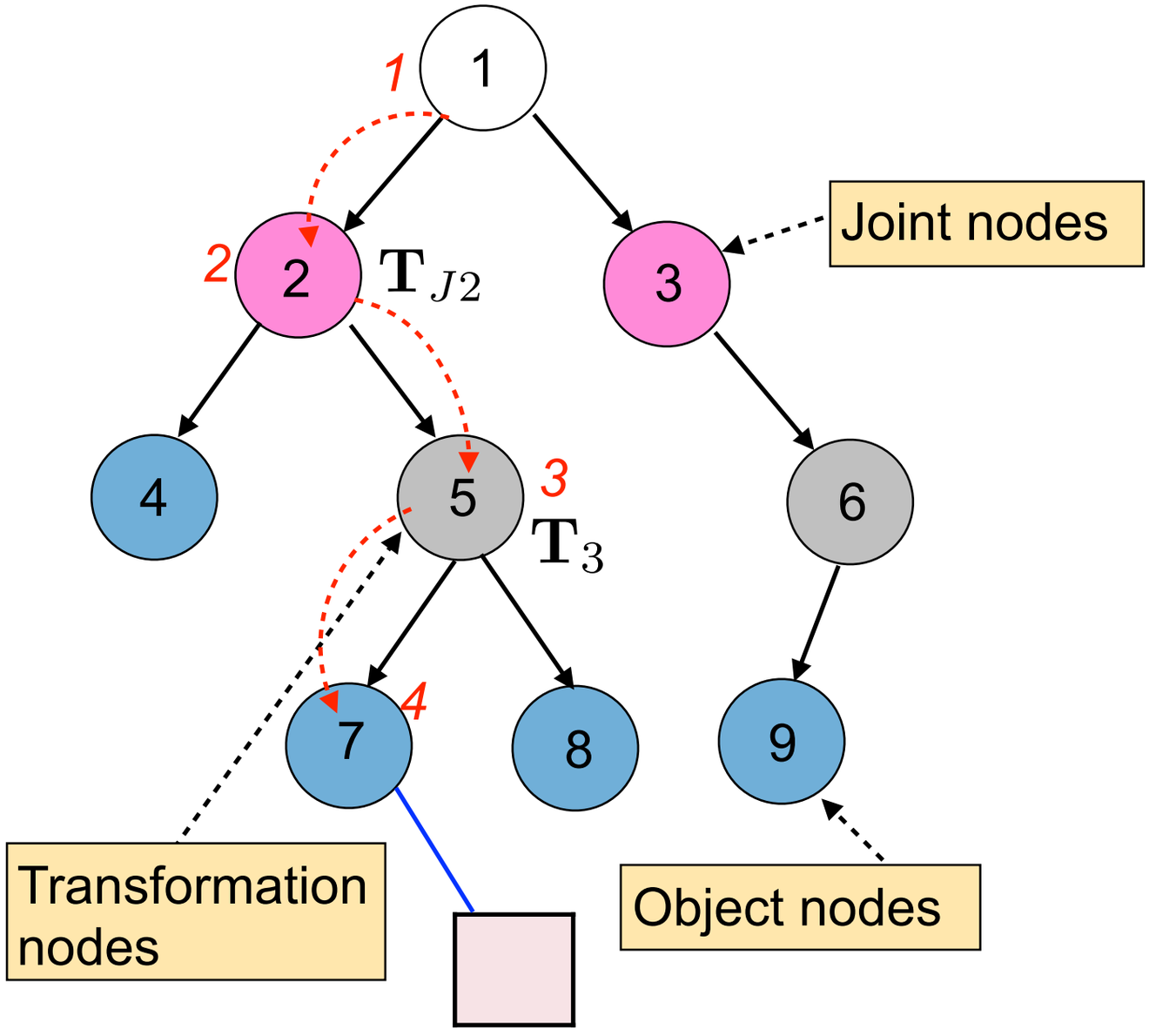


Depth First Search

is an algorithm for traversing or searching tree data structures

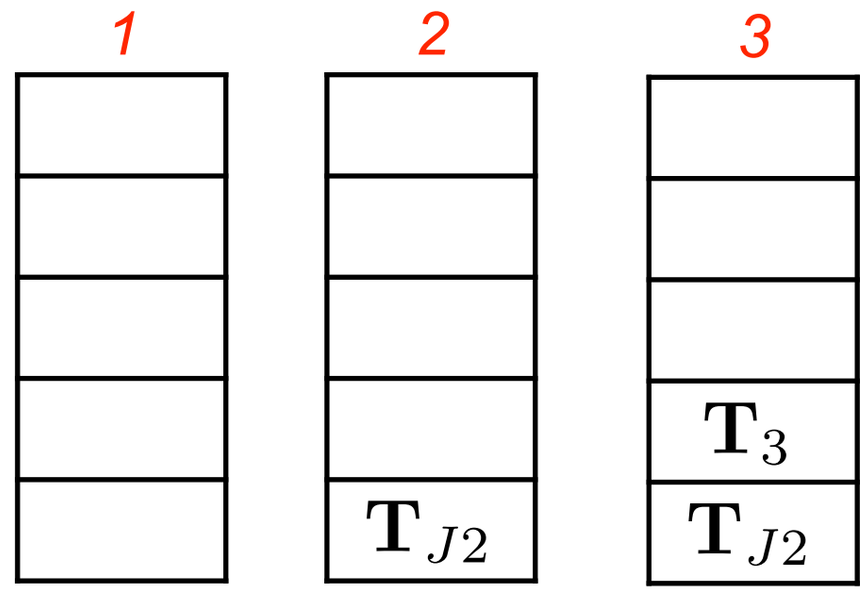
```
void traverse (Node& n)
{

   n-> do something

   int num_children = n.numChilds();

   for(int i=0; i< num_children; i++)
   {
      Node child = n->getChild(i);

      traverse(child);
   }

}
```

Tree Traversal Operation



Search for paths from root to empty child list

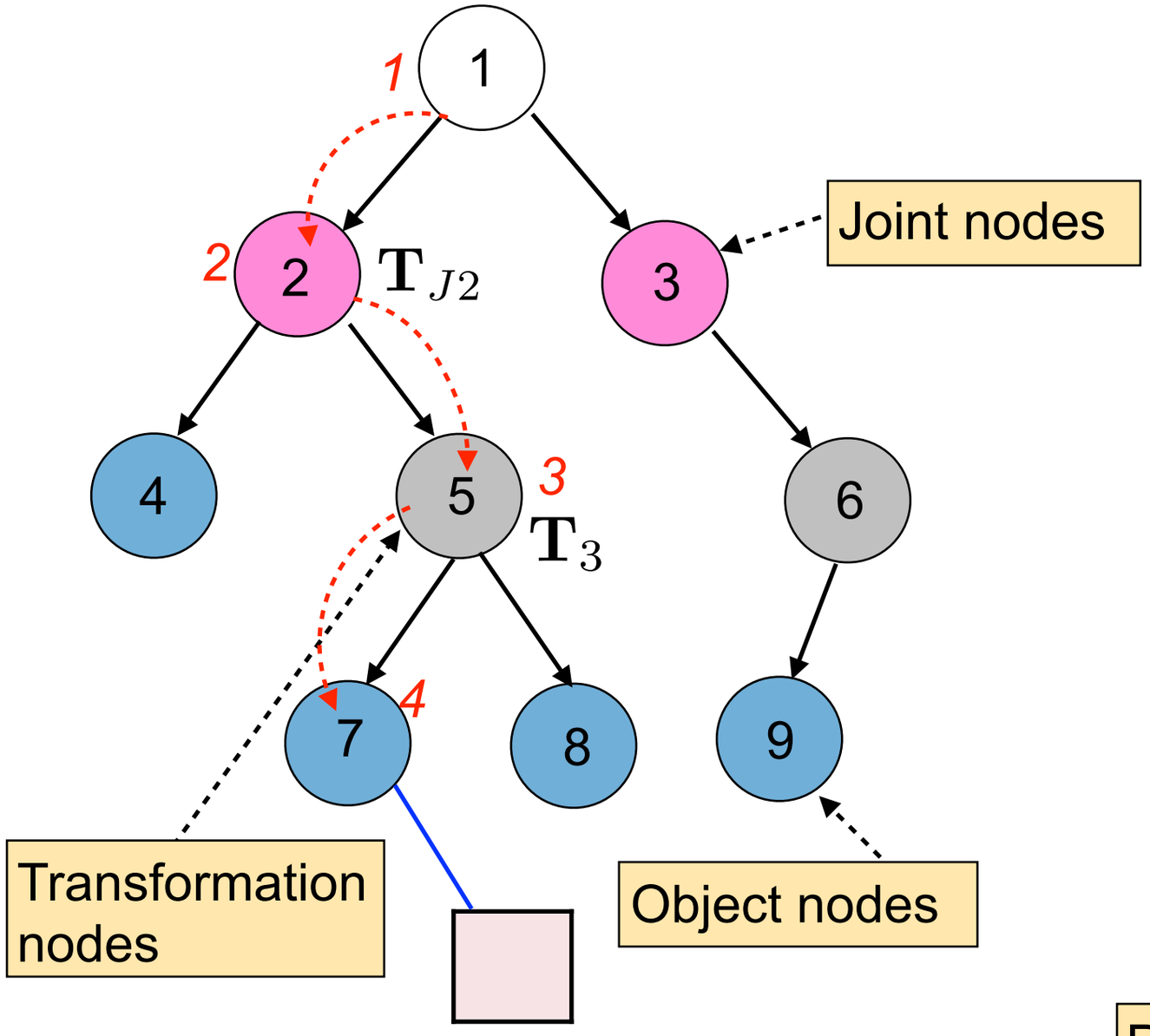
We can use a **stack** to remember the path.



Every time, we reach an empty child list, we can accumulate the path.

$$\mathbf{p}' = \mathbf{T}_3 \, \mathbf{T}_{J2} \, \mathbf{p}$$

Tree Traversal Operation



Search for paths from root to empty child list

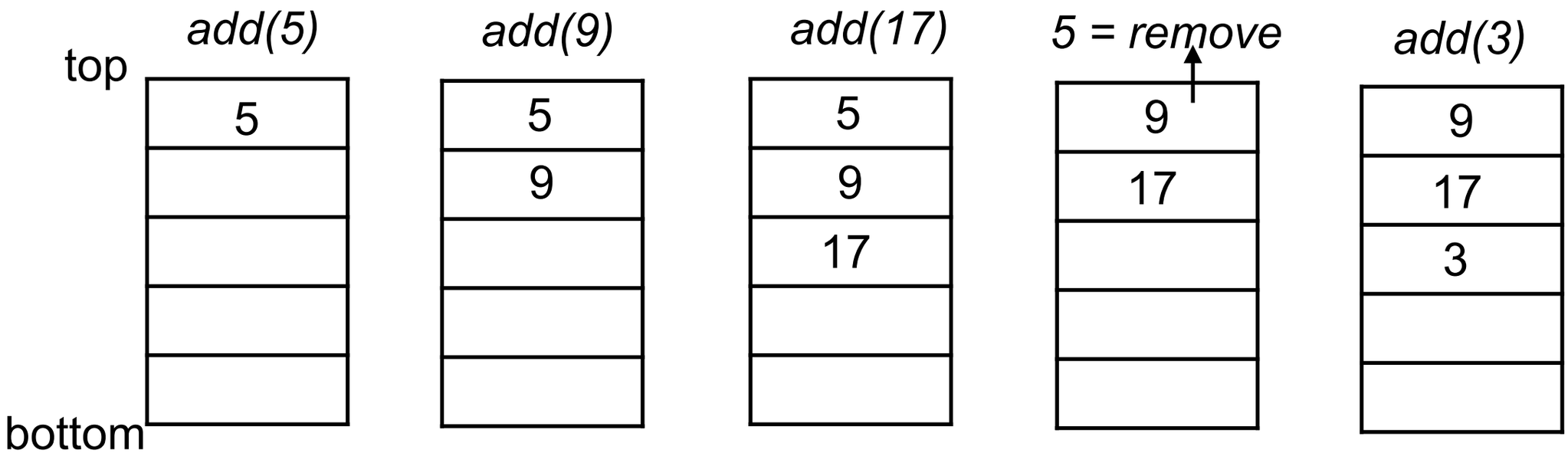
```
void traverse (Node& n)
{

   //remember the path
   stack.add(transformation)

   int num_children = n.numChilds();

   for(int i=0; i< num_children; i++)
   {
      Node child = n->getChild(i);

      traverse(child);
   }

   stack.remove(transformation)
```

Pseudo code, the syntax does not work

Stack and Queue

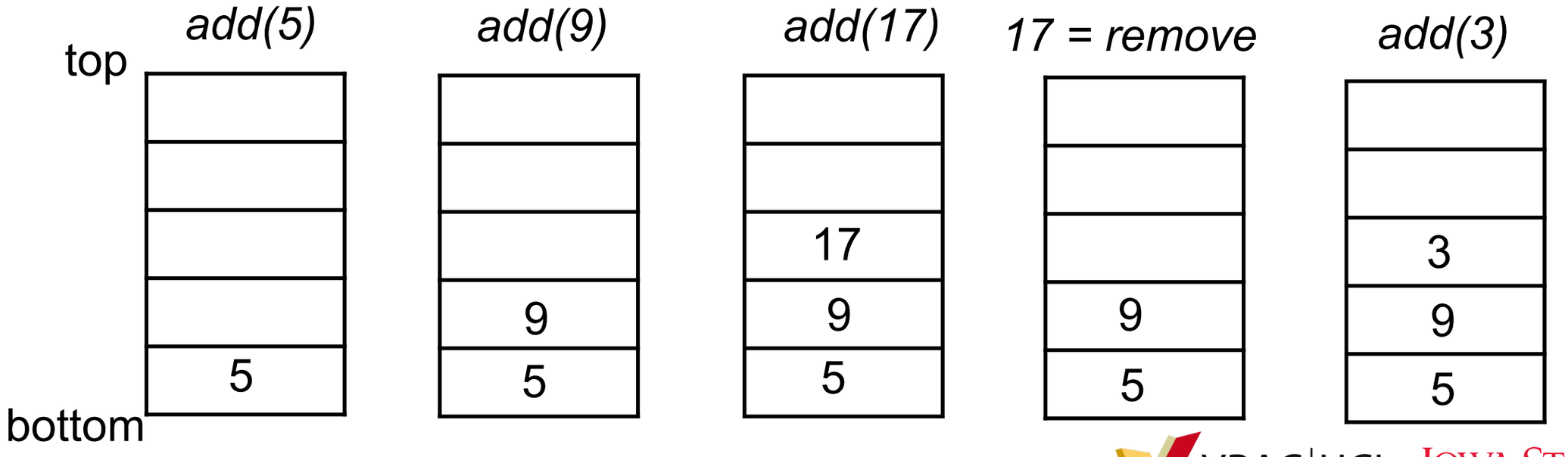
Queues: First-in-first-out

Elements are inserted at the bottom, but can only be removed from the top



Stack: Last-in-first-out

Elements are inserted at the top and can only be removed from the top



Note

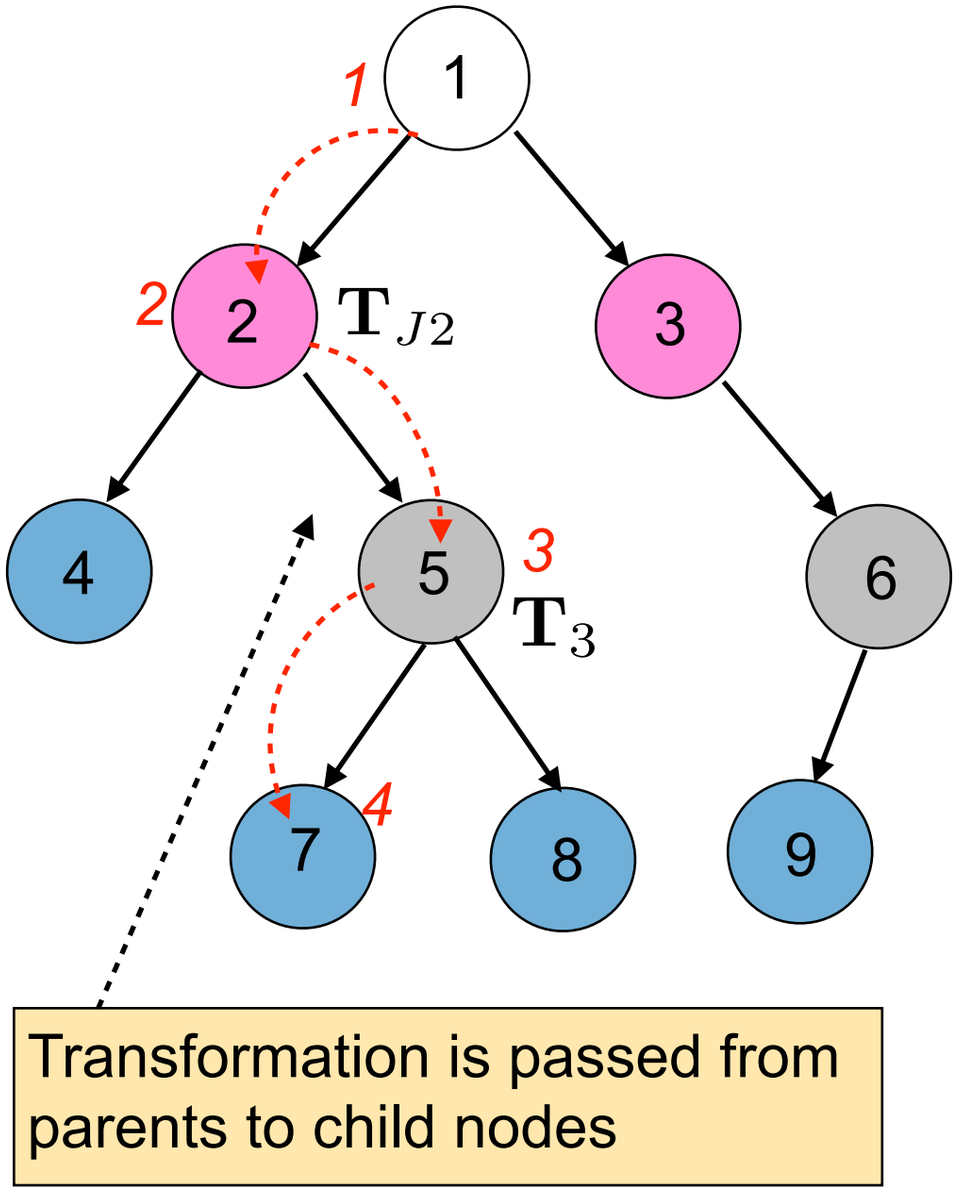
Disadvantages:

- Requires a lot of programming and understanding of abstract data structures, recursion, search algorithms, etc. Be encouraged, if you like to do this (I bet, you all are) !!!

Advantages:

- The new layer of abstractions (nodes and the tree structure) helps you to keep track of your objects in the scene, transformation, animations, etc.
- Almost all professional packages and open source graphics packages implement a hierarchical tree structure; so it is worth to understand how it works.
- We can extend this principle (node inheritance) to other graphics features such as light, material, transparency,

Node Inheritance



A node represents a graphics application property, i.e. transformation, light, etc.

The graph structure implements inheritance: **a parent passes its properties to its children.**

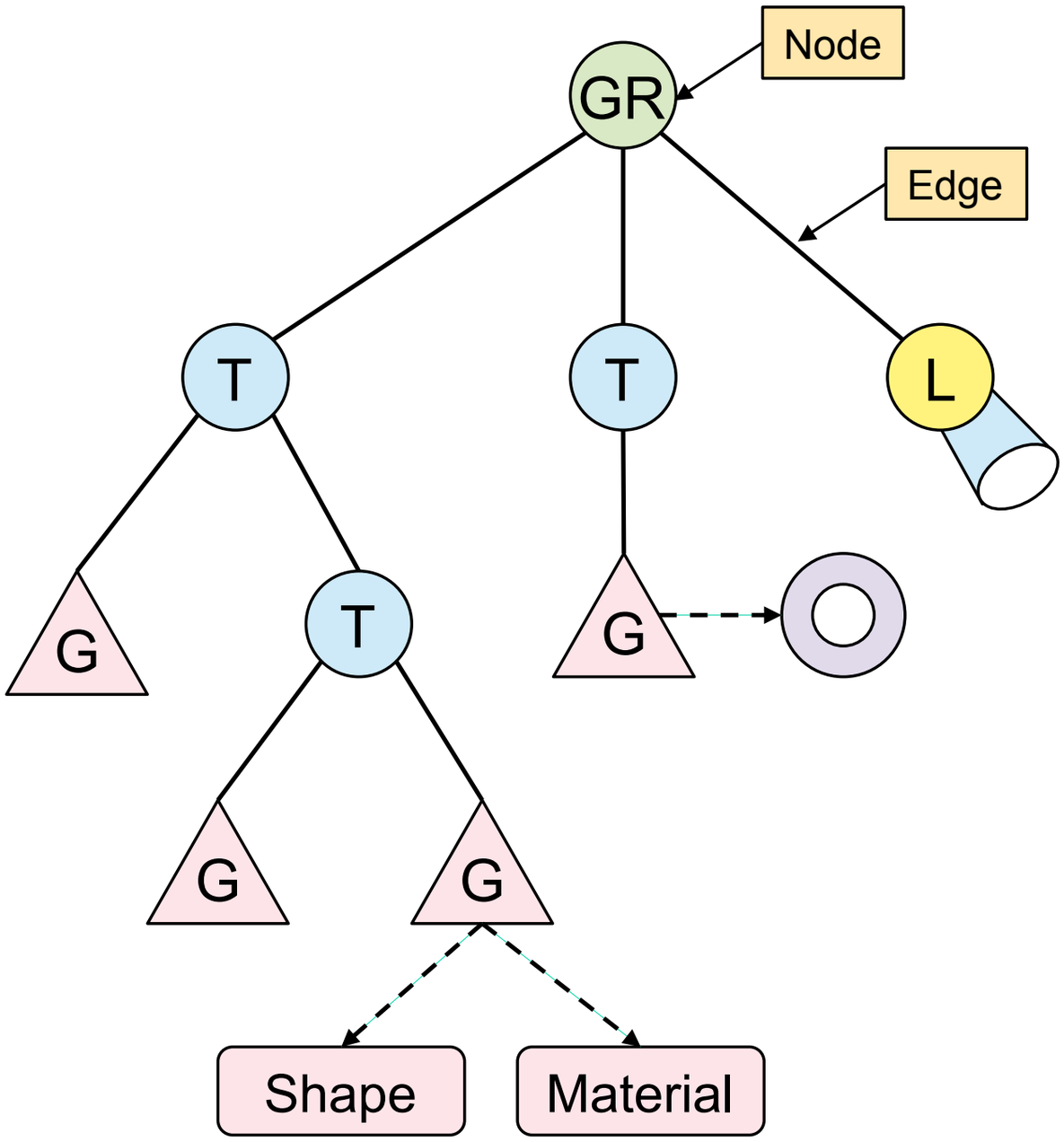
ARLAB

Scenegraphs and Scenegraph APIs

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

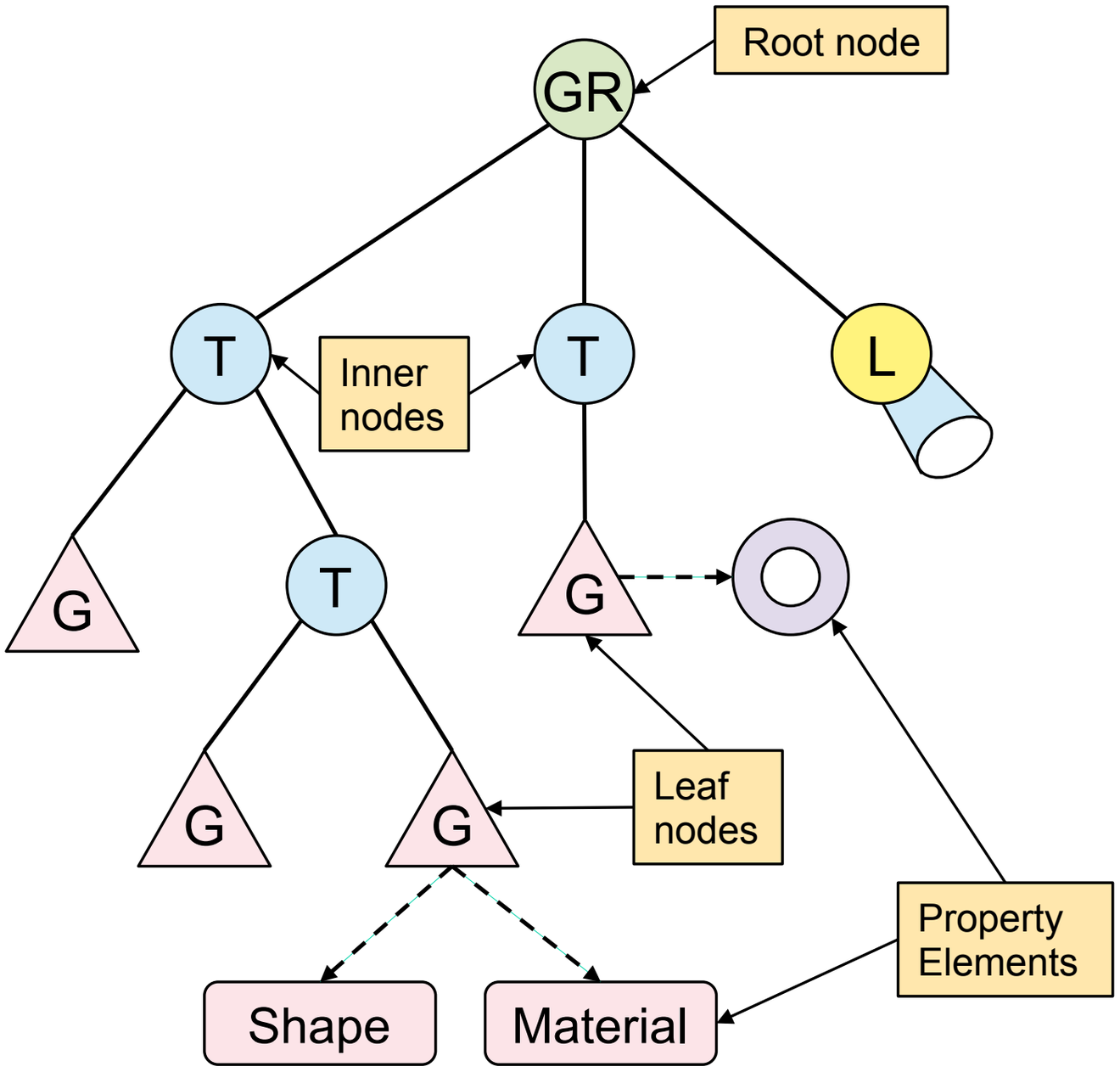
What is a scenegraph?



Schematic representation of a scene graph

A scene graph is a hierarchical data structure that facilitates the organization and the optimization of a 3D scene. It is represented by a tree structure (Direct Acyclic Graph). Its nodes represent scene elements, the edges between the nodes represent associations between the nodes that help to structure the scene.

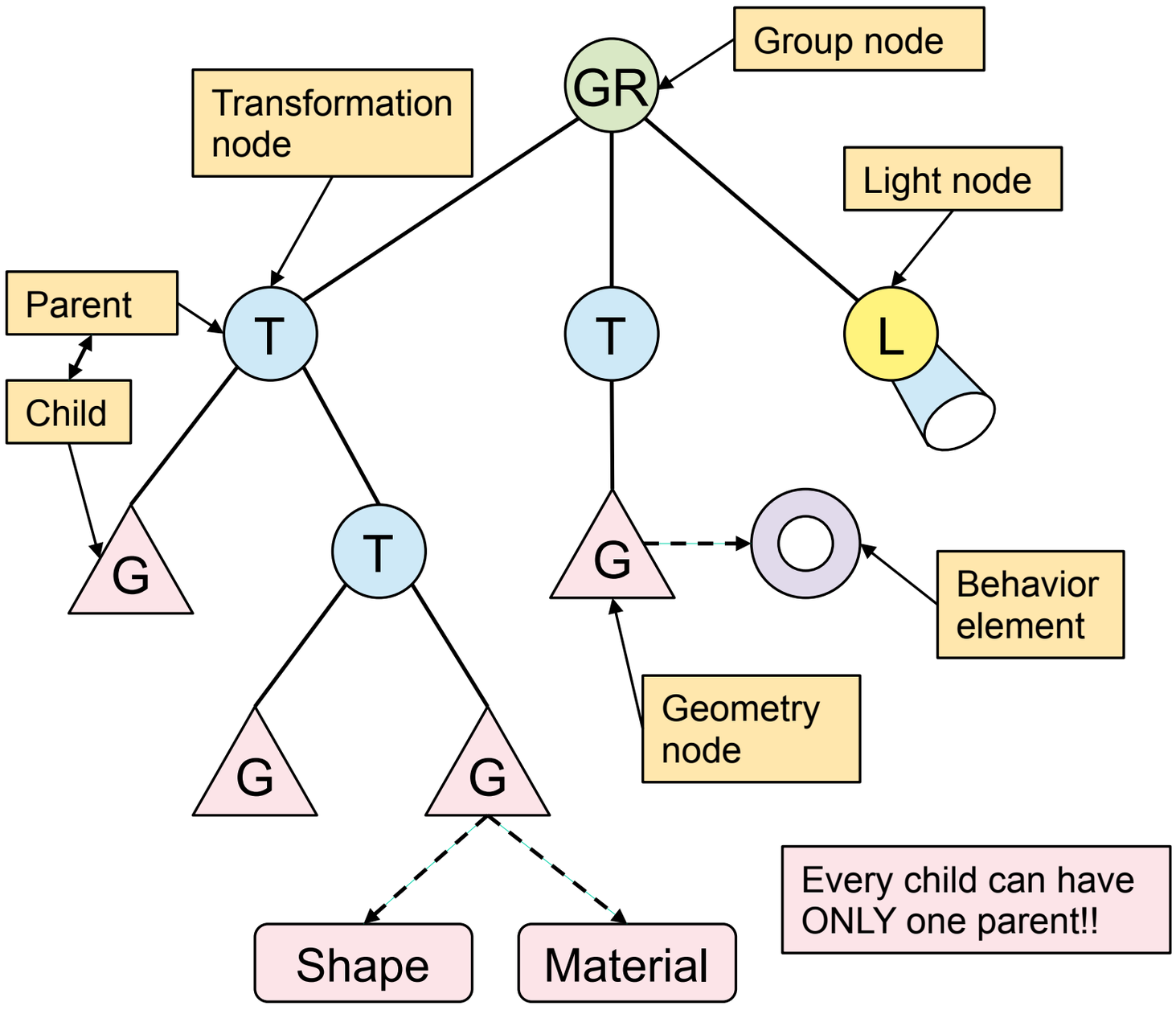
Scenegraph Node Types



Schematic representation of a scene graph

- Root node: this is the start node of every scene graph. It keeps "the entire scene."
- Leaf nodes represent geometry to render (e.g., 3D models). They do not accept further child nodes.
- Inner nodes represent relations between nodes, which are mainly used to structure the scene, e.g., transformations.
- Property elements describe the appearance and the behavior. They are not scene graph nodes.

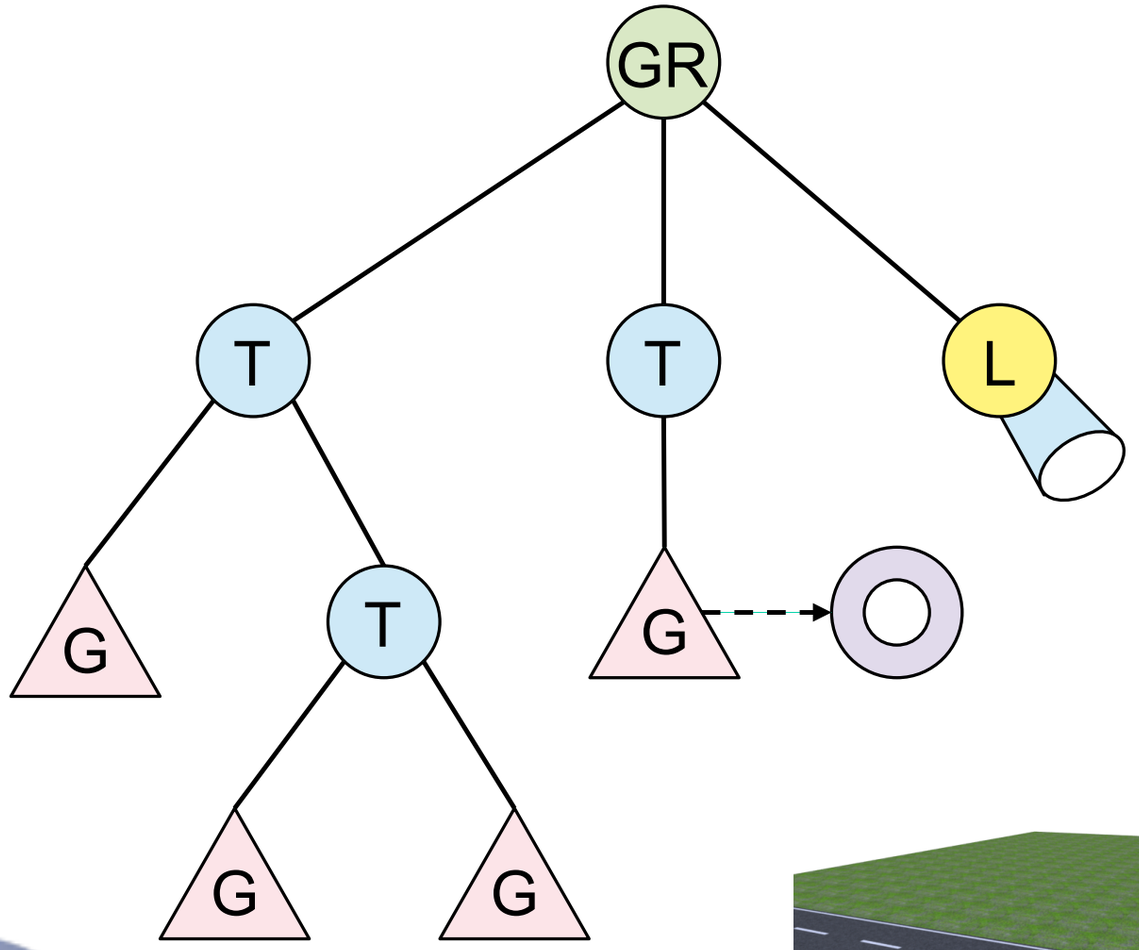
Scenegraph Nodes



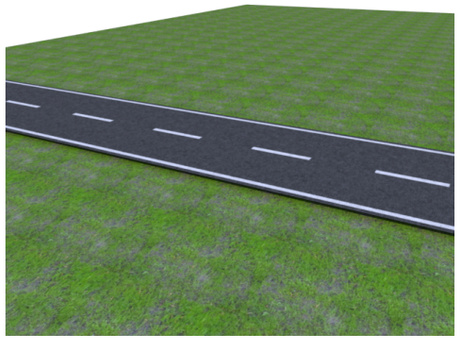
Schematic representation of a scene graph

- GR A Group node is used to structure the scene and to join multiple nodes to one logical unit.
- T A Transformation nodes represents a transformation matrix.
- G A Geometry node keeps OpenGL primitive information, thus, it can be used to represent 3D or 2D models.
- L A light node represents the OpenGL light sources.

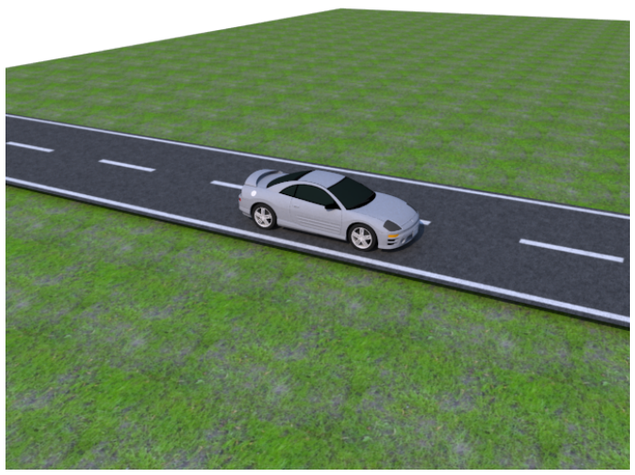
car body



wheels



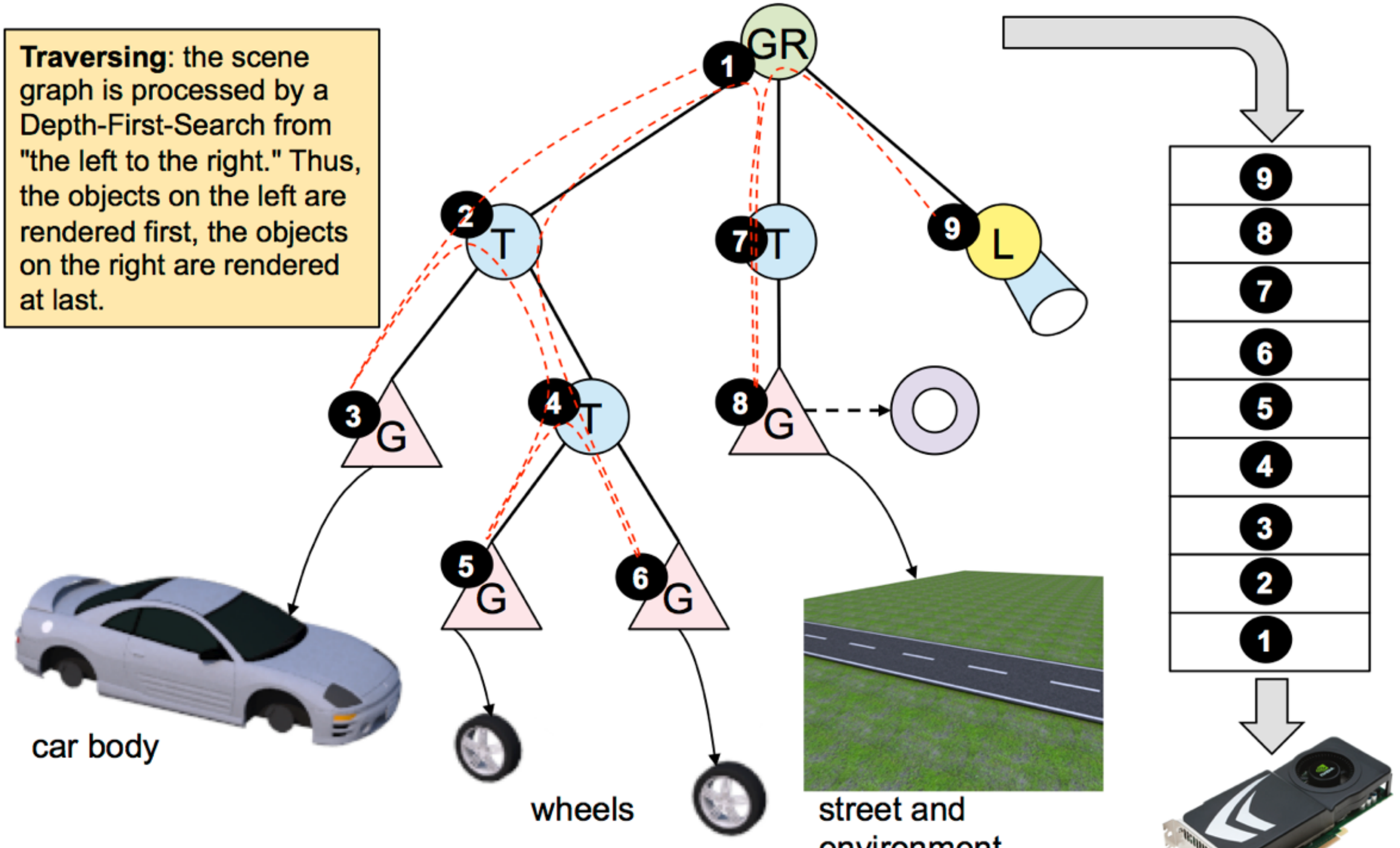
street and environment



Example

Traversing a Scenegraph

Traversing: the scene graph is processed by a Depth-First-Search from "the left to the right." Thus, the objects on the left are rendered first, the objects on the right are rendered at last.



1 order of visiting

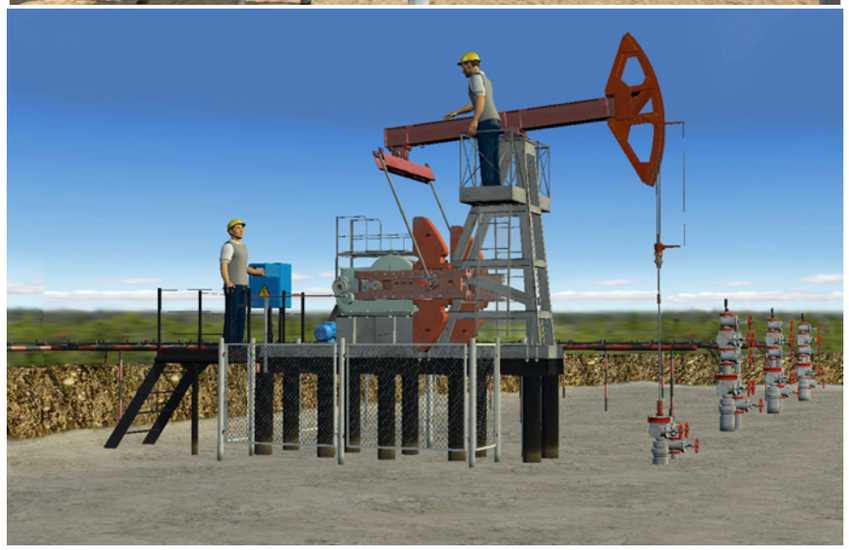
OpenSceneGraph

VRAC|HCI

OpenSceneGraph is an open source 3D graphics programming toolkit that can be used for a vast amount of 3D applications.

It provides functions to

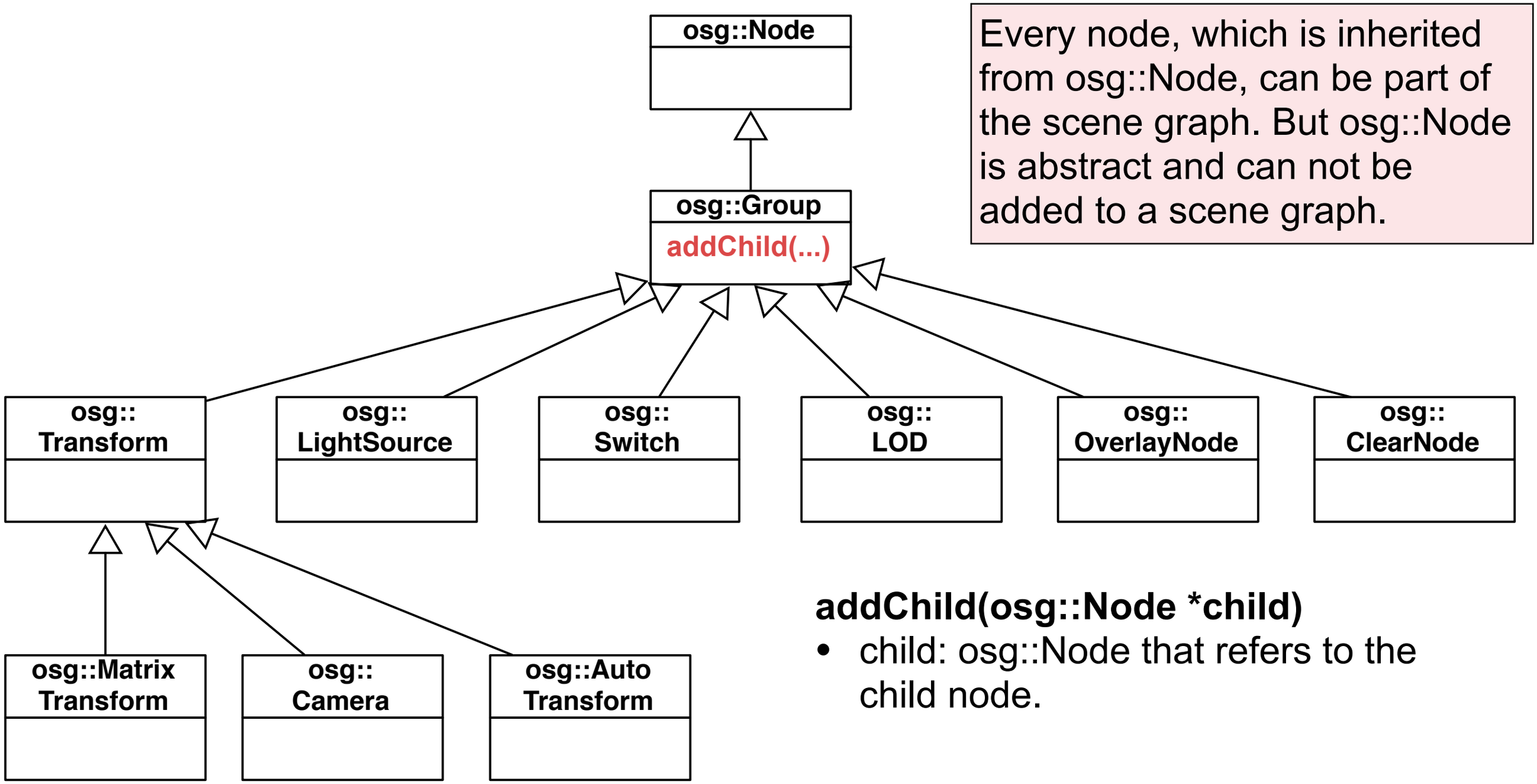
- build a scene graph
- loading and writing 3D models / images
- animation framework
- effects like environment mapping, shadows, particles.
- scene graph optimization
- interaction functions (e.g., Ray Intersection Test)
- basic collision detection.
- C++, Object Oriented Programming



http://www.openscenegraph.com

SITY

Node Hierarchy

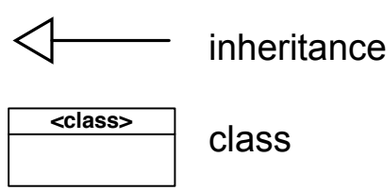


Every node, which is inherited from osg::Node, can be part of the scene graph. But osg::Node is abstract and can not be added to a scene graph.

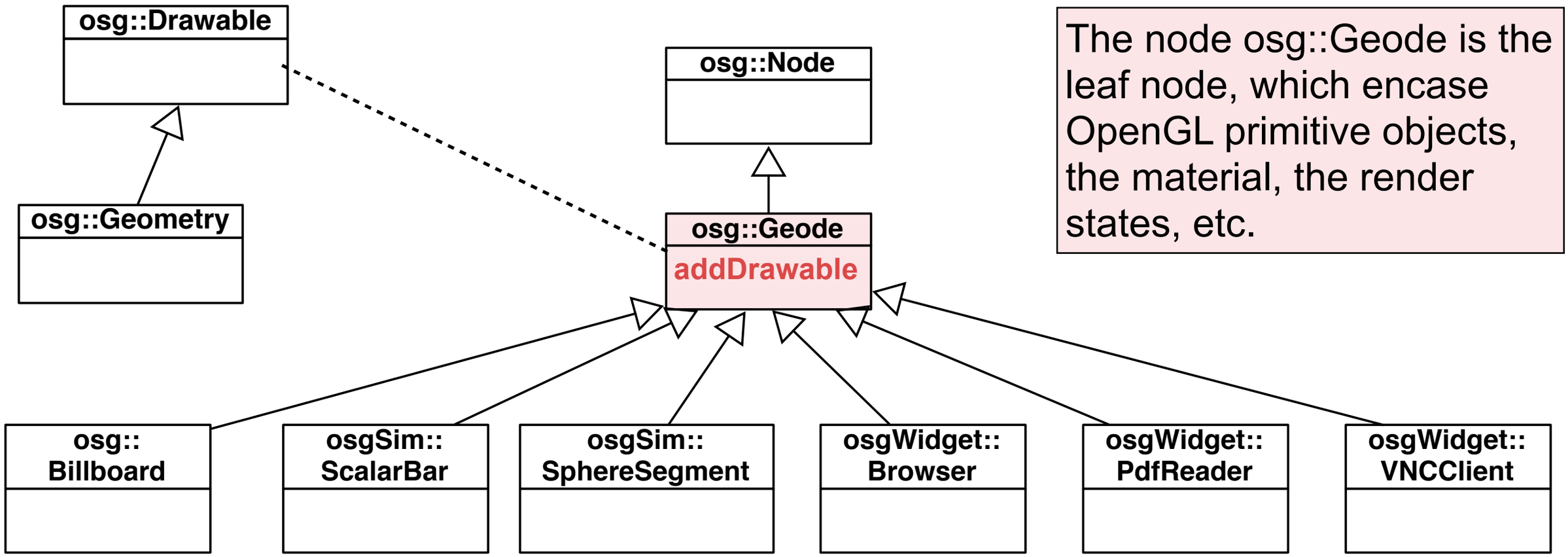
addChild(osg::Node *child)

- child: osg::Node that refers to the child node.

Subset of OSG nodes



Geometry Nodes



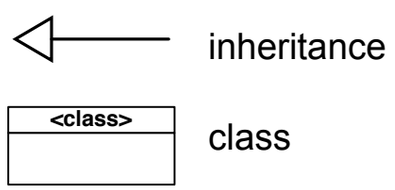
The node osg::Geode is the leaf node, which encase OpenGL primitive objects, the material, the render states, etc.

Add a Drawable to the Geode.

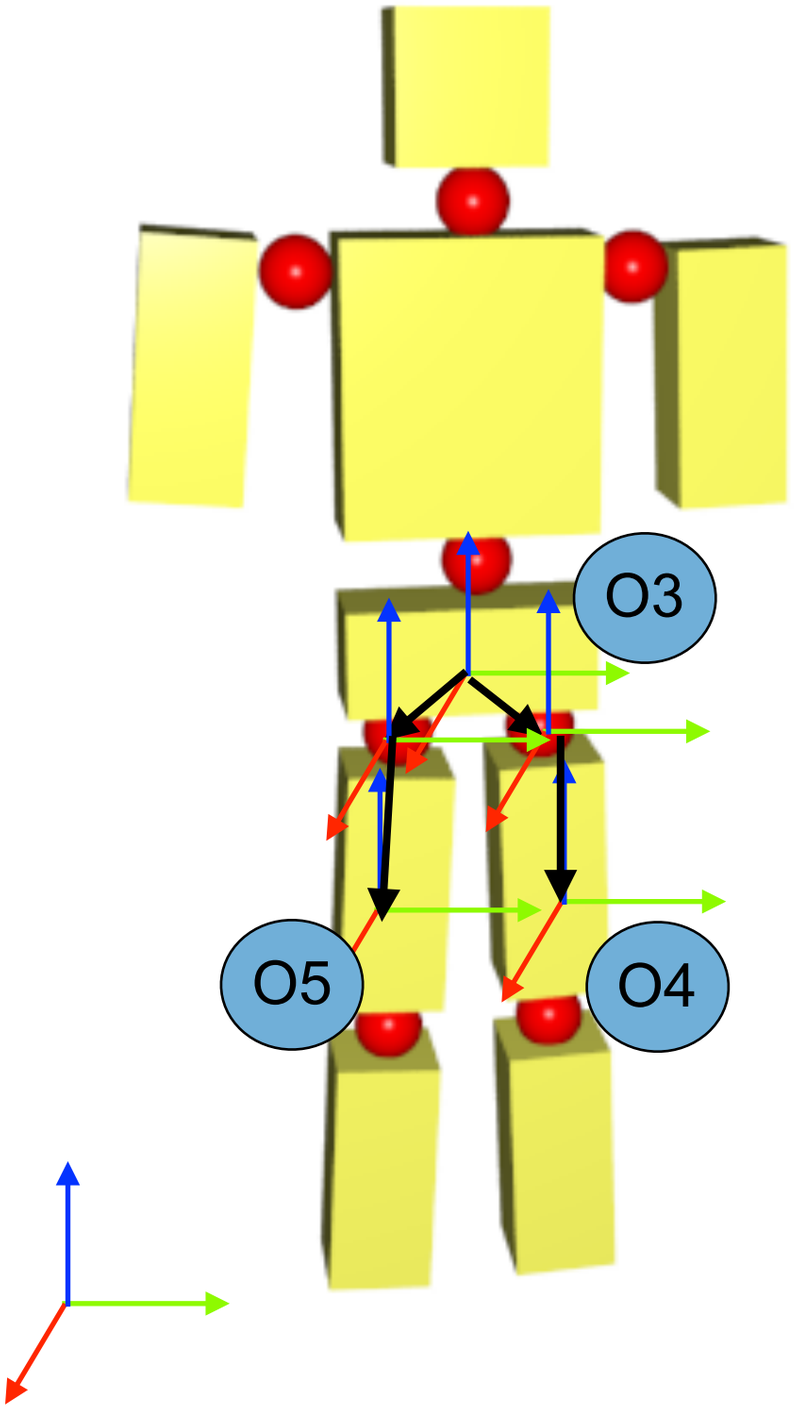
bool addDrawable (Drawable *drawable)

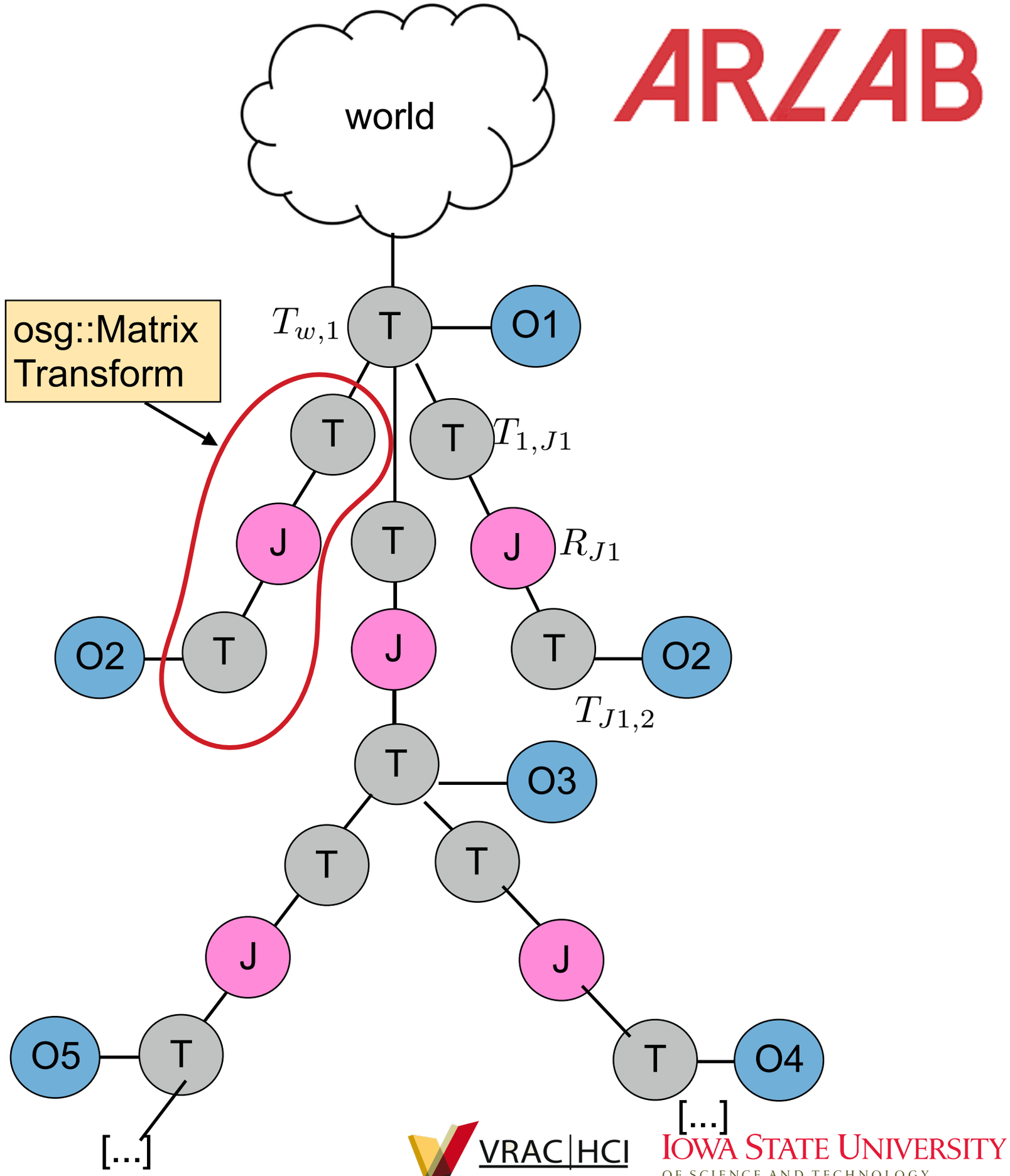
- drawable: a geometry object of type osg::Drawable

The class osg::Drawable stores OpenGL primitives. It cannot be part of the scene graph and need to be associated with an osg::Geode object.



Tree of objects



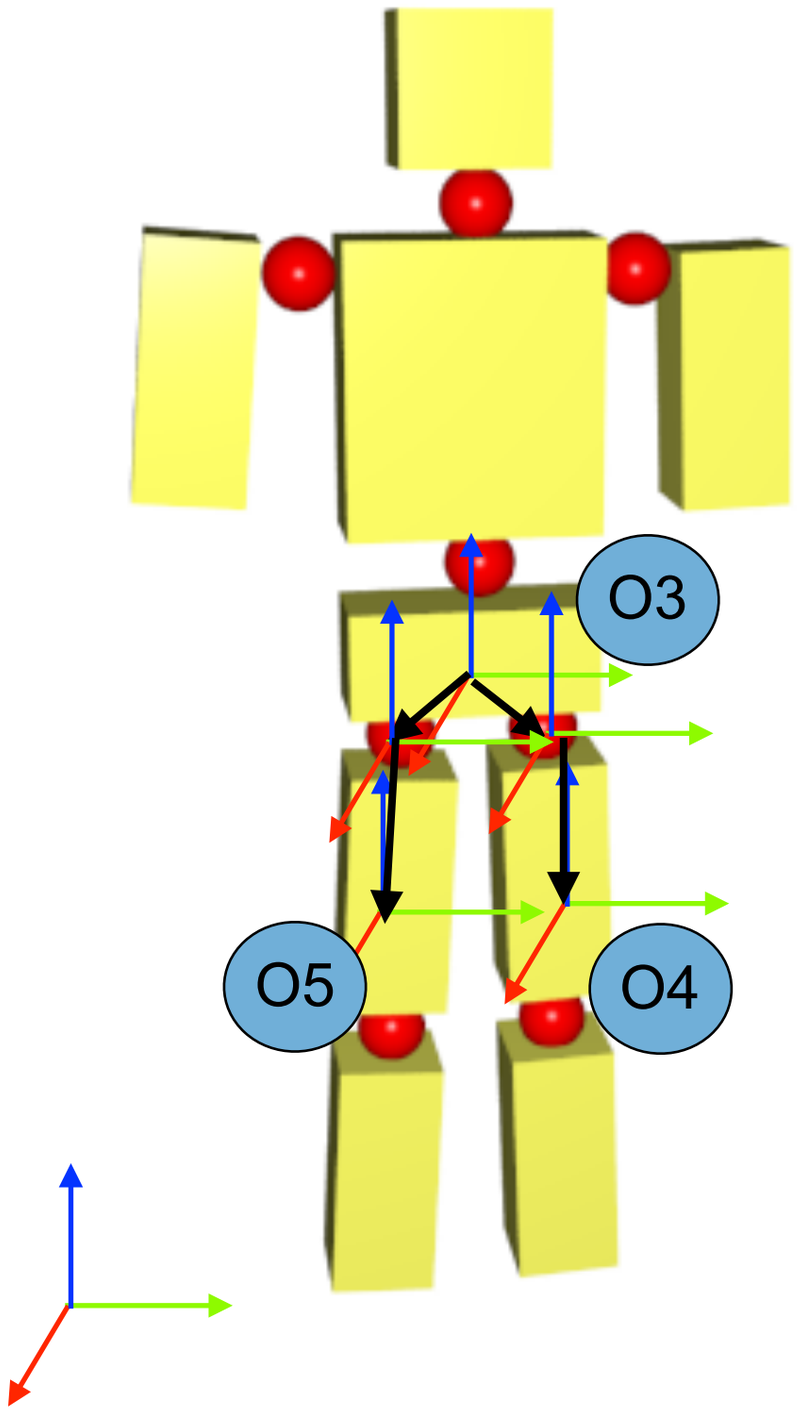


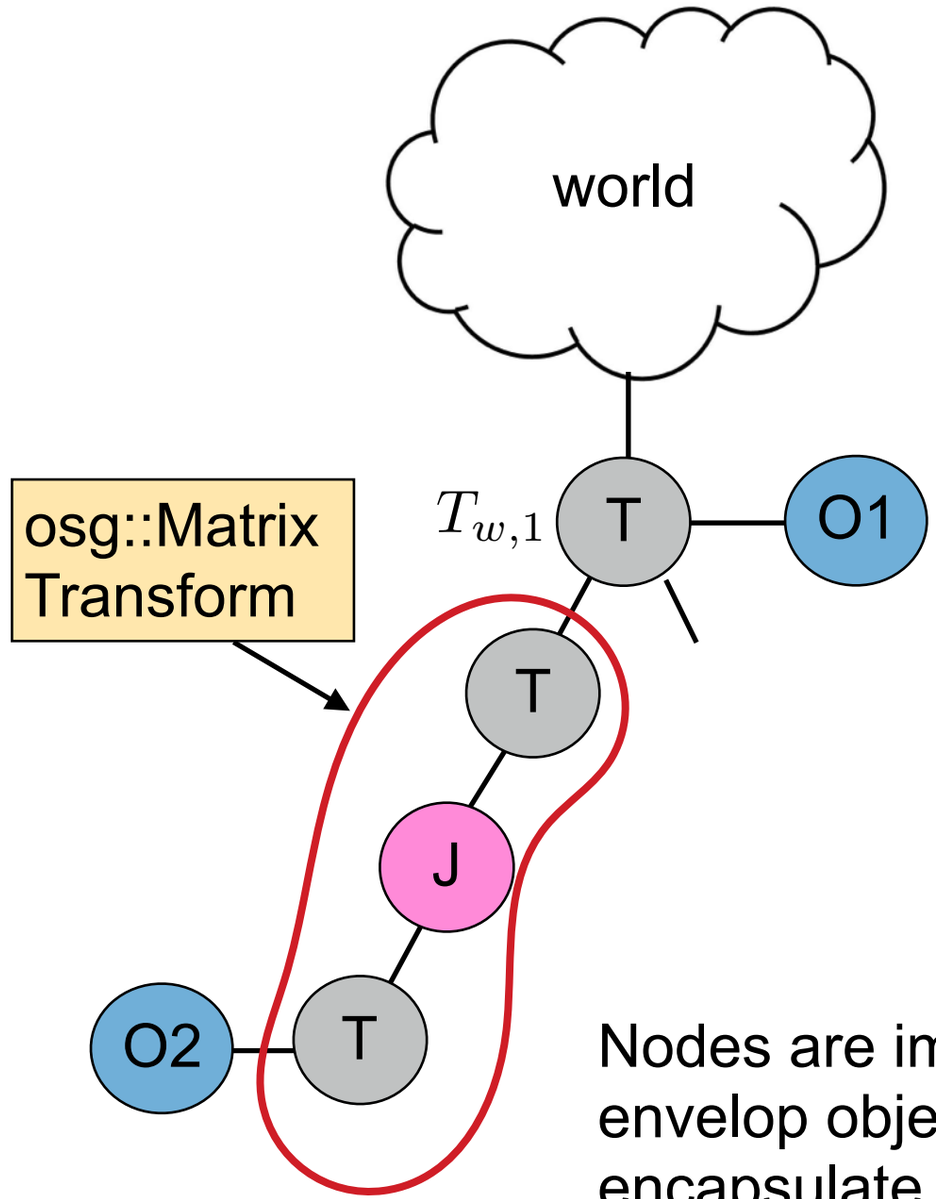
ARLAB

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Tree of objects





Nodes are implemented as envelop objects which encapsulate the graphics objects along with all graphics properties.

The nodes do not implement the graphics properties!

Distinguish between objects that do graphic and objects that implement the scene graph.

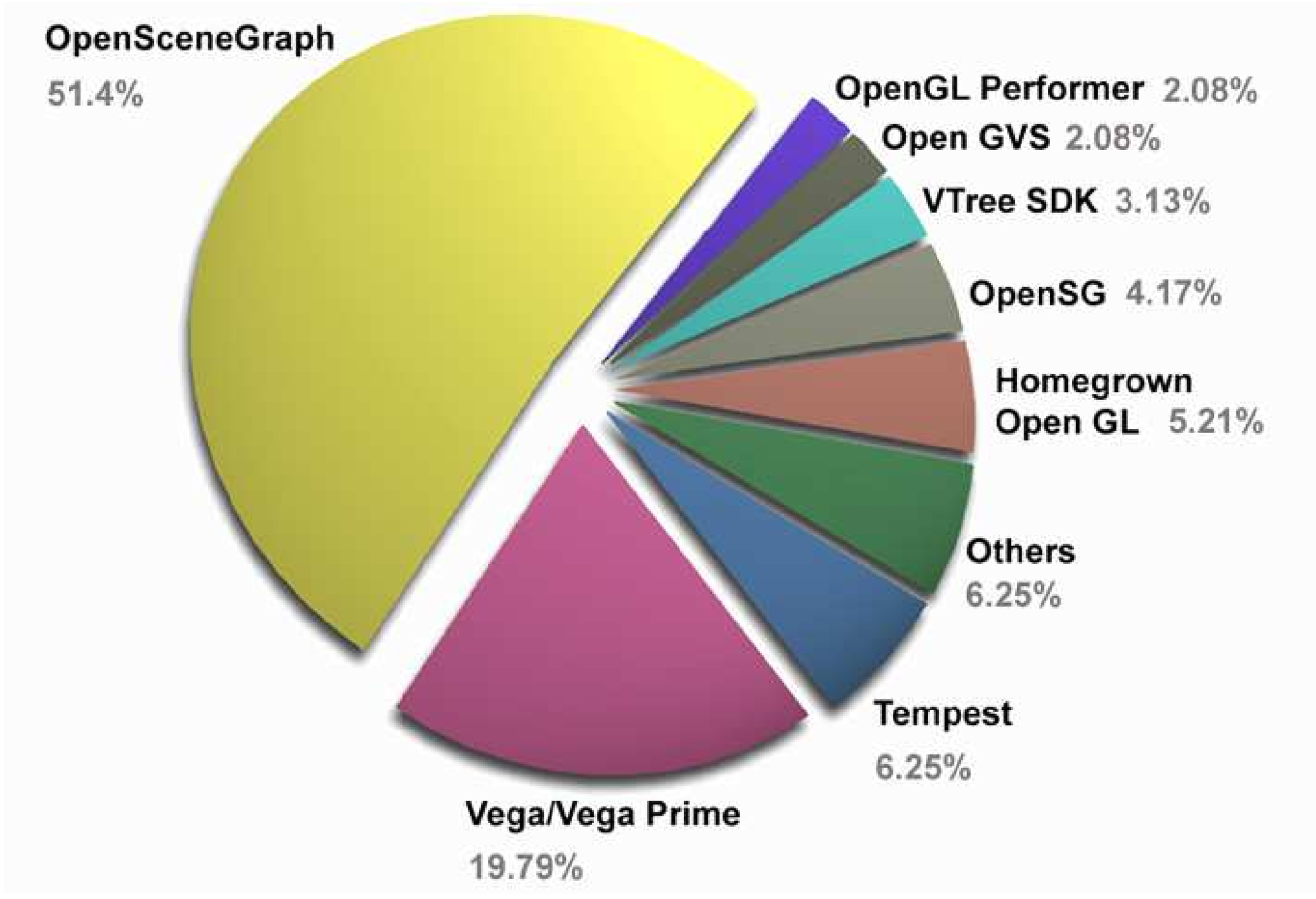
Scenegraph Optimization

Why we should use a scene graph?

Scene Graphs are used to optimize the processing of primitives.

- The policy is: as less primitives are passed to the graphics card as better it is.
- Typical optimization processes are
- Culling Operations
  (View Frustum Culling, Occlusion Culling)
- Level-of-Detail
- State sorting
- Helps you to keep track of you objects and scene elements

Scenegraph poll



MODSIM Poll on Prefered IG System (MODSIM.org, 2005)

Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 50011, USA

+1 515.294.5580
+1 515.294.5530(fax)

rafael@iastate.edu
http://arlab.me.iastate.edu

www.linkedin.com/in/rradkowski

ARLAB

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY