Scene Graph Hierarchy

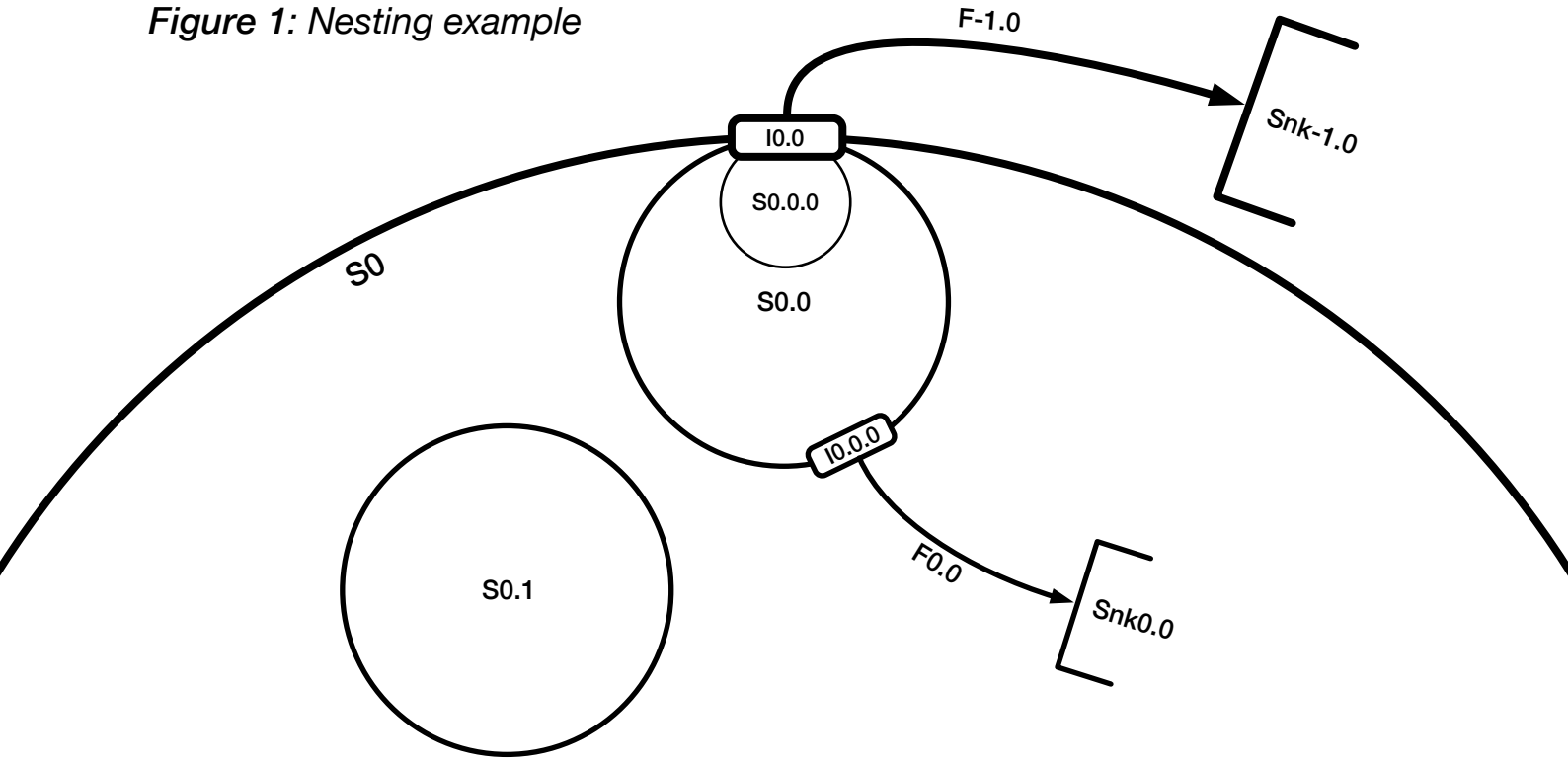
This shows how all the elements pictured below (*Figure 1*) are hierarchically related in the program.

Please note that, even though S0.0.0 is an interface subsystem, it is a child of S0.0 and not of I0.0 because it is a second level interface subsystem.

All labels (not illustrated here) are root level entities.

Entity	Description
Snk-1.0	External sink
F-1.0	External outflow
S0	Root system
Snk0.0	Internal Sink
F0.0	Internal flow
S0.1	Subsystem
I0.0	Interface
S0.0	Interface subsystem
I0.0.0	Nested interface
S0.0.0	Nested interface subsystem

Figure 1: Nesting example



Zoom, Scale, Visibility and Nesting Level

Zoom is stored and modified as a `Resource`. It is not applied to the camera's matrix in the form of a scaling factor. The camera is moved however to always be centered on the same point relative to the world entities.

The zoom value is multiplied to the translations x and y of all entities. The z component stays unchanged. To provide a base value at zoom level 1 the component `InitialPosition` is used.

Systems

Only system entities change size according to the current zoom value. This is done by drawing a circle with it's base radius multiplied by zoom. The transform's scale is not changed.

Nesting Level

Labels, external entities and interfaces all have a maximum size and are only scaled down when the zoom factor falls below a certain threshold. To compute their scale their nesting level together with the zoom factor is considered.

All of these types of entities have a component `NestingLevel` attached which is just a `u16`. The table below illustrates what nesting levels the entities shown in *Figure 1* have.

The scale is computed like the following:

$$s = \min\{f^l \cdot z, 1\}$$

where

- s — Resulting scale
- f — Nesting factor. A fixed constant < 1 called `SUBSYSTEM_SCALING_FACTOR`
- l — Nesting level
- z — Zoom factor

Entity	Nesting Level
Snk-1.0	0
F-1.0	0
S0 (No component)	0
Snk0.0	1
F0.0	1
S0.1	1
I0.0	1
S0.0	1
I0.0.0	2
S0.0.0	2

This same scale calculation is also used to determine the line width of all entities (including systems).

Visibility

It is also used to determine the visibility of an entity. If the scale falls below a certain threshold the entity is hidden (`SCALE_VISIBILITY_THRESHOLD`, `LABEL_SCALE_VISIBILITY_THRESHOLD`).

Draw Order aka Z Coordinate

With potentially infinite nesting a proper drawing order scheme has to be followed in order to ensure that the entities which are supposed to be in front of other entities stay that way at all nesting levels.

To make the logic independent of nesting every subsystem applies a scale of 0.9 to the z coordinate. Then inside that subsystem everything is positioned along the z axis between 0 and 100. The scaling along the z axis makes sure that subsystems and their contained entities never exceed z = 100 from the point of view of the parent system.

See the following table for the local z coordinates per entity type.

Entity Type	Local Z
Root System	0
Interaction	1
External Entity	1
Interface	100
Interface Subsystem	-90
Subsystem	10
Button	200
Label	150

Buttons usually have the same parent as the entity will have that they create when clicked.

On the next page you can see an example how this works together with the scene graph hierarchy and z scaling to make infinite nesting possible.

Entity	Local Z	Global Z	Z Scale
S0	0	0	1
Snk0.0	1	1	
F0.0	1	1	
S0.1	10	10	
I0.0	100	100	
S0.0	-90	-90 + 100 = 10	0.9
I0.0.0	100	100 · 0.9 + 10 = 100	
S0.0.0	10	10 · 0.9 + 10 = 19	0.9

Figure 2: Front view of Figure 3 to illustrate the z coordinates

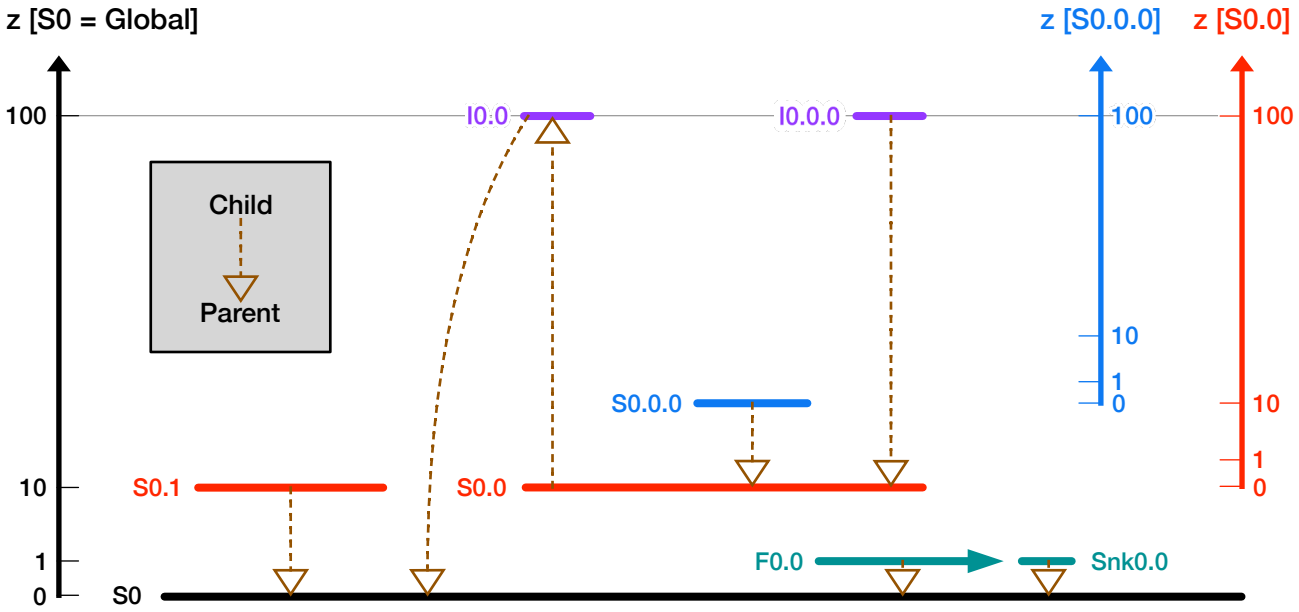
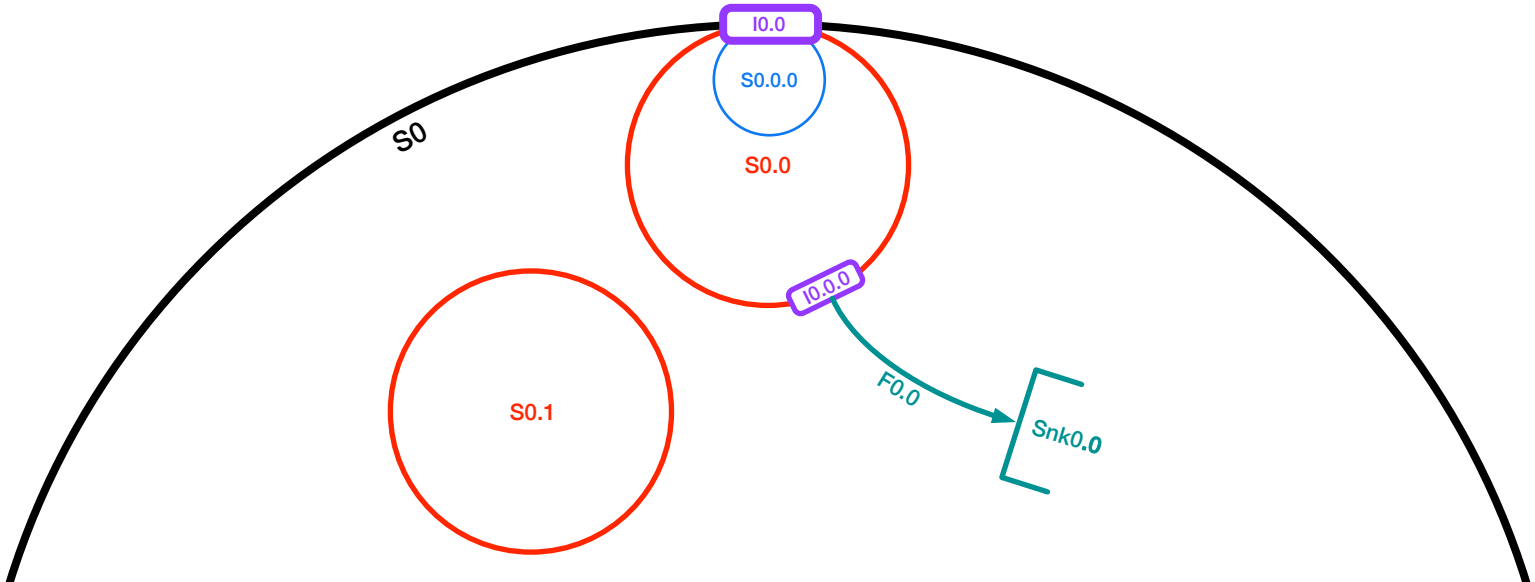


Figure 3: Top view of Figure 2 as it appears in the app. Colors are just a Visual help. They do not correspond to the colors in the table above



Interactions and Connected Entity Rotations

Interactions are displayed as cubic bezier curves. Their control points are computed automatically by the orientation and position of their connected entities. Let's look at an example.

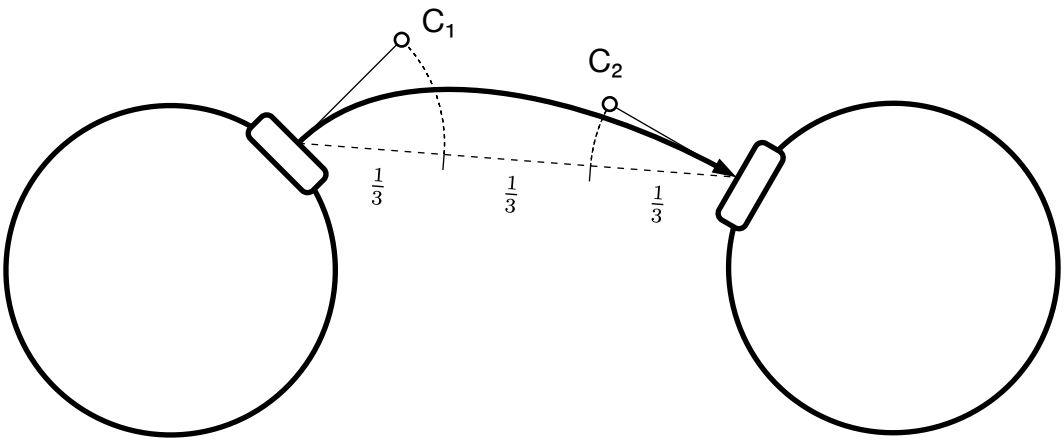


Figure 4: *Interaction curve between two interfaces with it's control points*

As you can see the direction of the tangents at the ends of the curve are determined by the rotation of the connected interfaces. In the code this direction can be retrieved by calling `interface_transform.right()` which returns a unit vector.

The length of these tangents, where the control points C_1 and C_2 are, is set to a third of the distance between the endpoints. This results in nice and smooth curves.

External Entity Rotation

To make aesthetic curves, external entities are automatically rotated when being dragged. The following figure shows how the rotation is computed.

The control point of the opposite end (in this case C_1) is computed. Then the vector from the external entity towards that control point (D) is determined which is finally used as the rotation of the external entity.

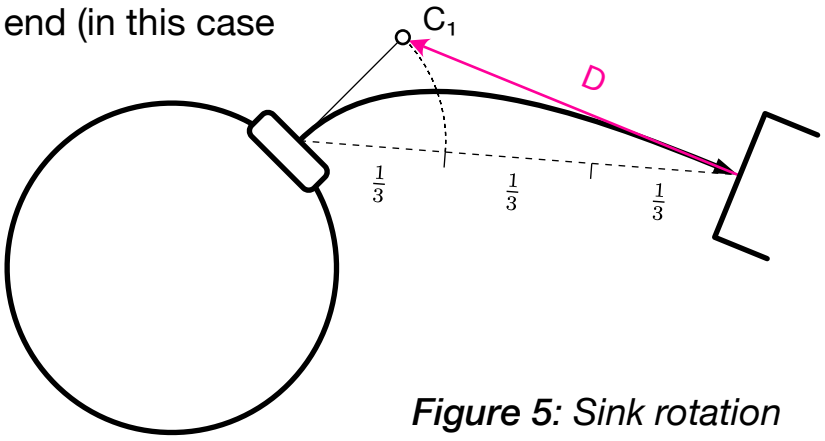


Figure 5: *Sink rotation*

The exact same method is used when determining the tangent of end of an interaction that is attached to the mouse cursor while selecting a source/sink.

Subsystem Connecting

When a subsystem is selected as the target of an interaction, the tangent of the newly connected end needs to be determined. This is done as illustrated in *Figure 6* below.

First, the smooth direction D is computed for the center of the subsystem that should be connected (the one to the right) exactly as in *Figure 5*. Then this direction is used together with the radius of this system to compute the target point P.

Finally (bottom row of *Figure 6*) this point P and direction D is used as the new target point and tangent of the interaction curve.

When a subsystem is created from several selected external entities, the interaction curves are connected to the fresh new subsystem are computed in the same way and the corresponding interfaces use the exact same position and orientation.

***Figure 6**: Connect to the right subsystem. Top: Step 1; Bottom: Step 2*