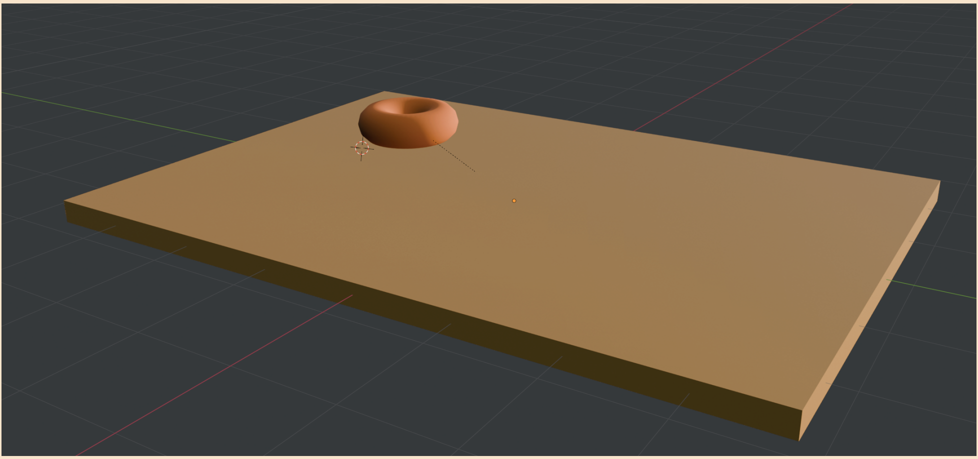
Parenting

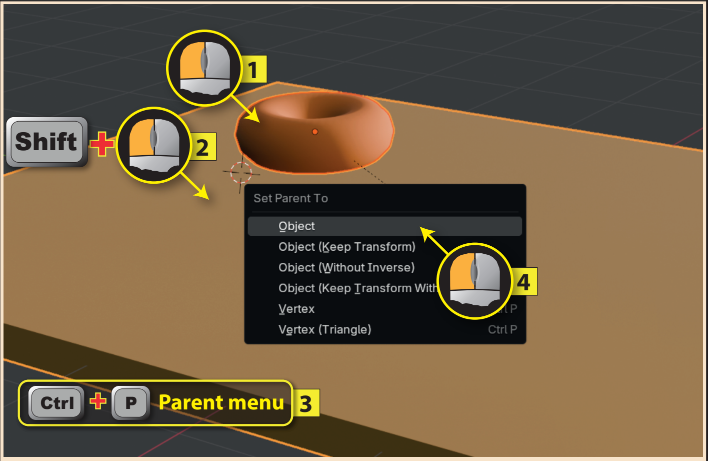
While using the join operation effectively merges two objects into a single object, parenting keeps the two objects involved separate but linked.

One object is referred to as the **parent** while the other is the **child**. Any operation performed on the parent is also performed on the child, but the child can be adjusted independently of the parent.

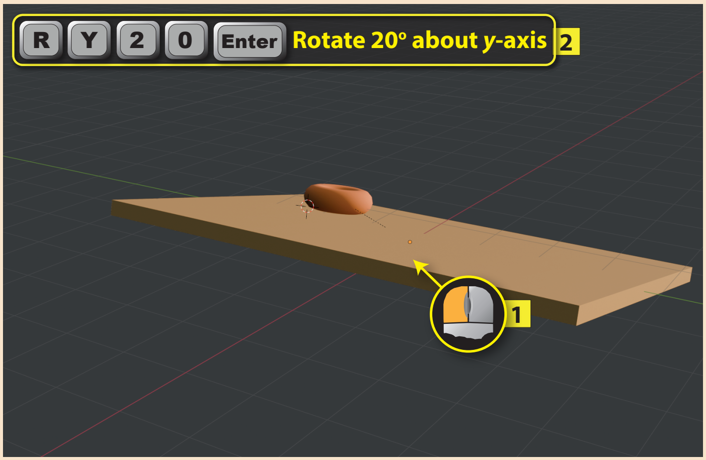
Imagine we have placed a doughnut on a flat piece of wood. If the wood is moved, we would expect the doughnut to move too. However, the doughnut can be moved without affecting the wood.



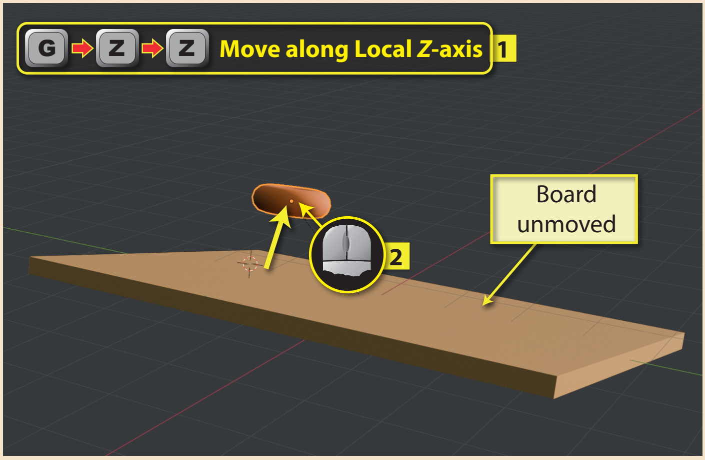
To create a parent/child relationship we must start by selecting what is to be the child object and then the parent. Next we need to press **Ctrl+P** and select **Object** from the popup menu.



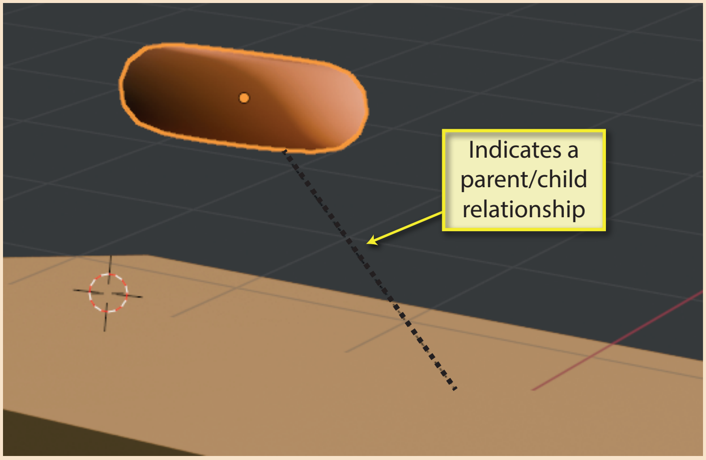
The two objects are now linked. If we move, rotate or scale the parent object, the child will perform the same operation. Here the *Board* has been rotated about the *y*-axis and the ***Doughnut*** has rotated too.



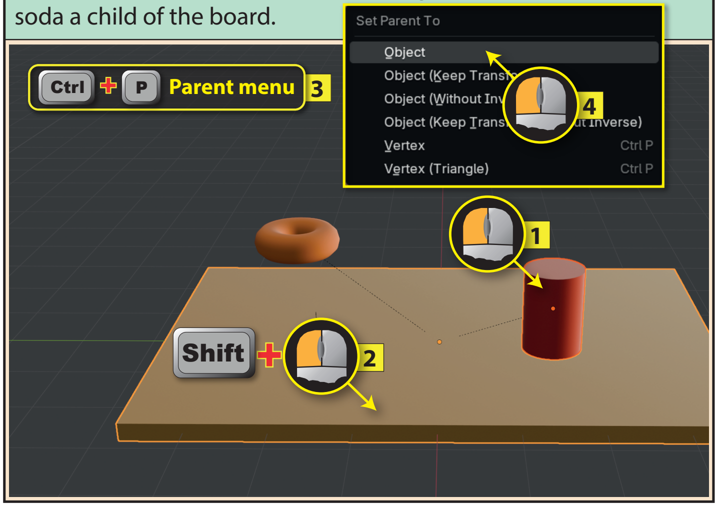
But the doughnut can be adjusted on its own. For example, we can move it away from the board.



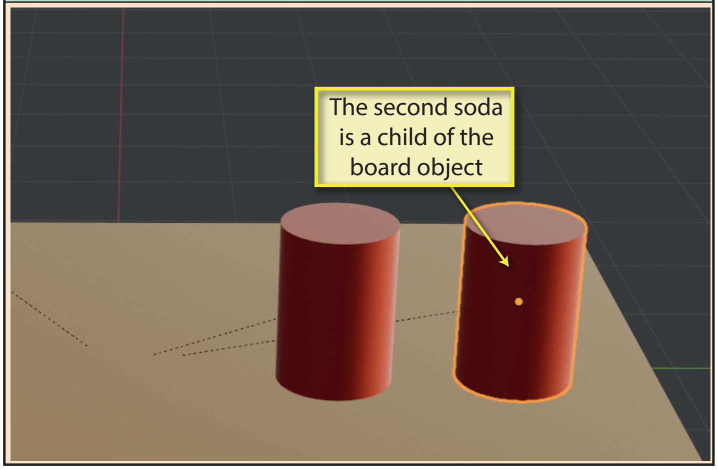
Notice that Blender adds a dashed line between the parent and child objects.



A parent can have more than one child. In our example, we've added a red cylinder to represent a soda can. If we follow the same sequence as we did with the doughnut, we can make the soda a child of the board.



If we make a copy of a child object (**Shift+D**), then the copy also becomes a child of the same parent as the original.

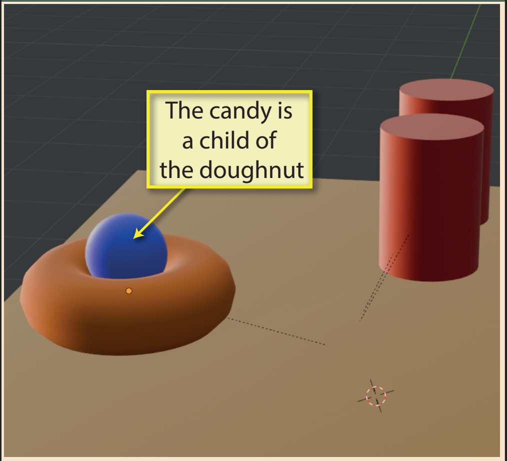


NOTE

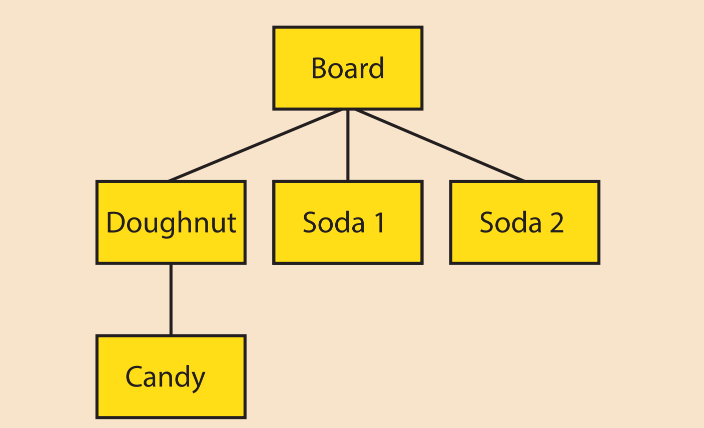
Although a parent can have many children, a child object can have only one parent.

If we attempt to assign a second parent to a child object, the new parent will replace the old parent.

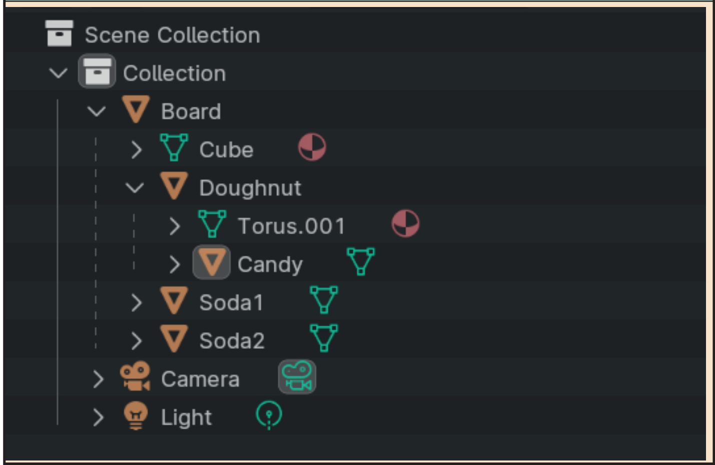
The child of one object can be the parent of another object. For example, we can add a round candy in the hole of the doughnut, and make the candy a child of the doughnut.



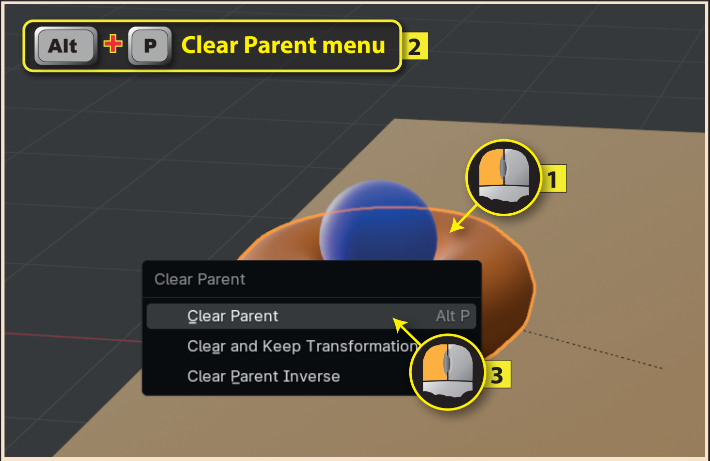
We can represent the parent/child relationships of our whole setup with the following "family tree". Making any changes to an object only affects those attached object further down the "tree". So moving the Board will move everything else, while moving the Candy or a Soda affects no other object.



We can see this tree structure reflected in the Outliner Editor where ***Doughnut***, ***Soda1***, and ***Soda2*** are listed within ***Board***, and ***Candy*** is listed within ***Doughnut***.

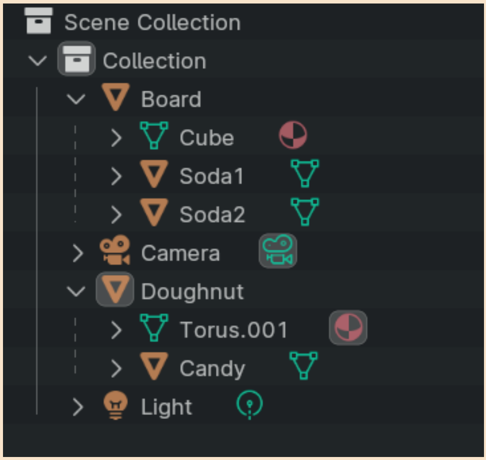


To remove a parent/child link, we must start by selecting the child object and then pressing **Alt+P**. In the popup menu that this produces, we need to select **Clear Parent**. The example, below shows the steps in breaking the link between ***Doughnut*** and ***Board***.

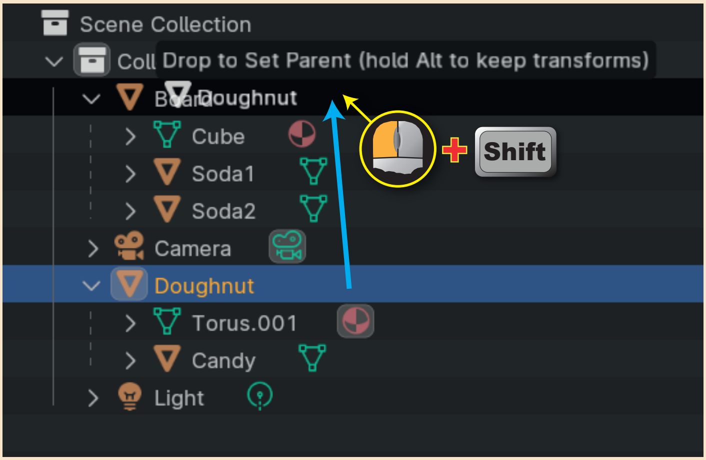


Transforming ***Board*** in any way will no longer affect ***Doughnut*** or ***Candy***, but ***Candy*** will still transform with ***Doughnut***.

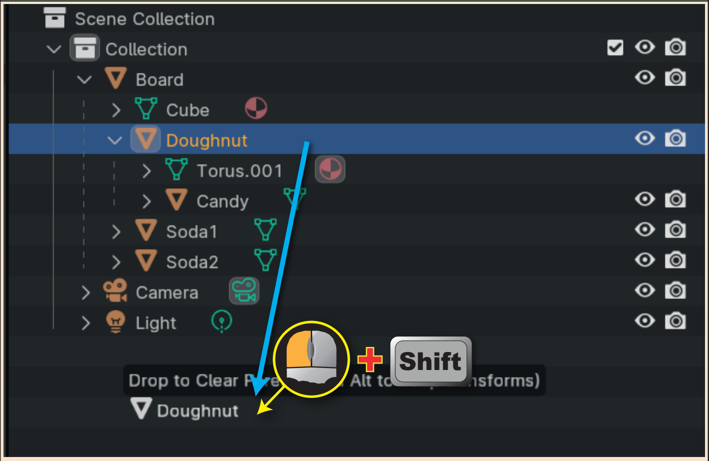
The change is also reflected in the Outliner Editor where ***Doughnut*** and ***Candy*** are no longer listed within ***Board***.



The Outliner Editor offers an alternative way of creating aparent/child relationship. All we need to do is hold down the **Shift** key while dragging the child object's name over the parent's name.

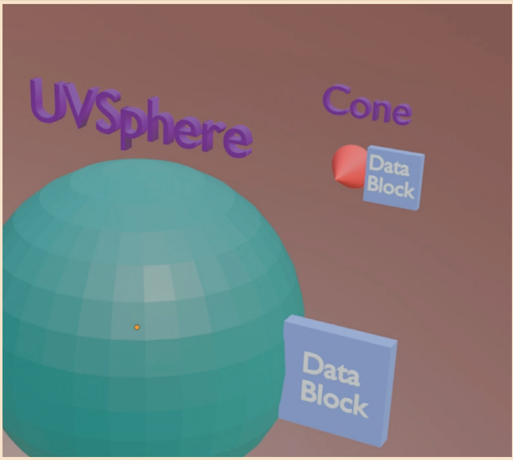


To break the link when working in the Outliner Editor, just hold down **Shift** and drag the child object into an unoccupied area of the Outliner Editor.

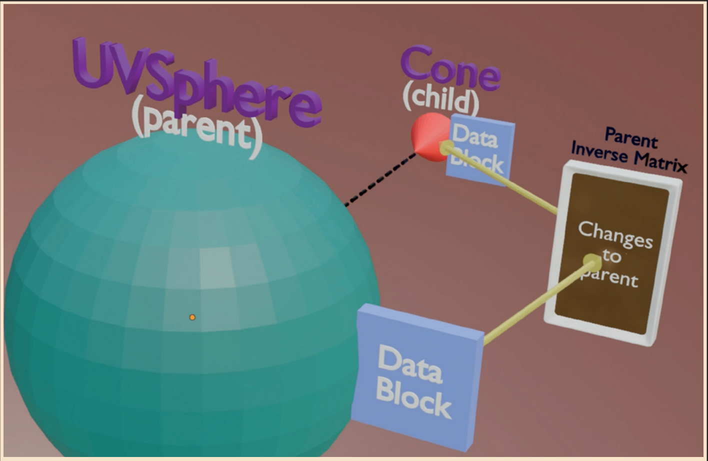


Before looking at further parenting options, it is necessary to delve a little deeper into how Blender organises the objects we create.

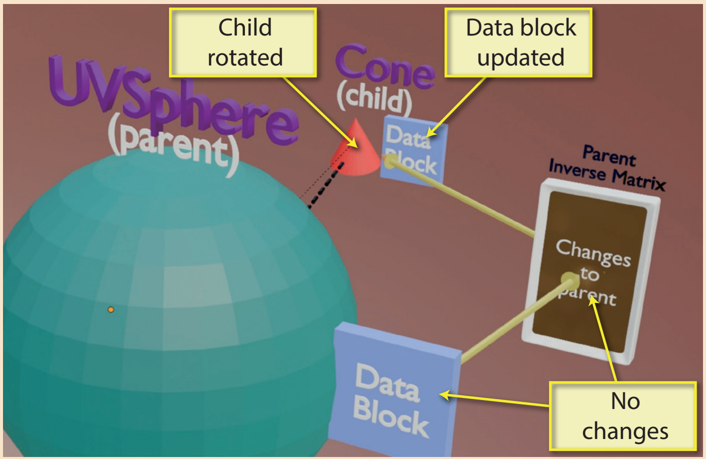
Every object we create is assigned a data block in computer memory where information about that object is stored.



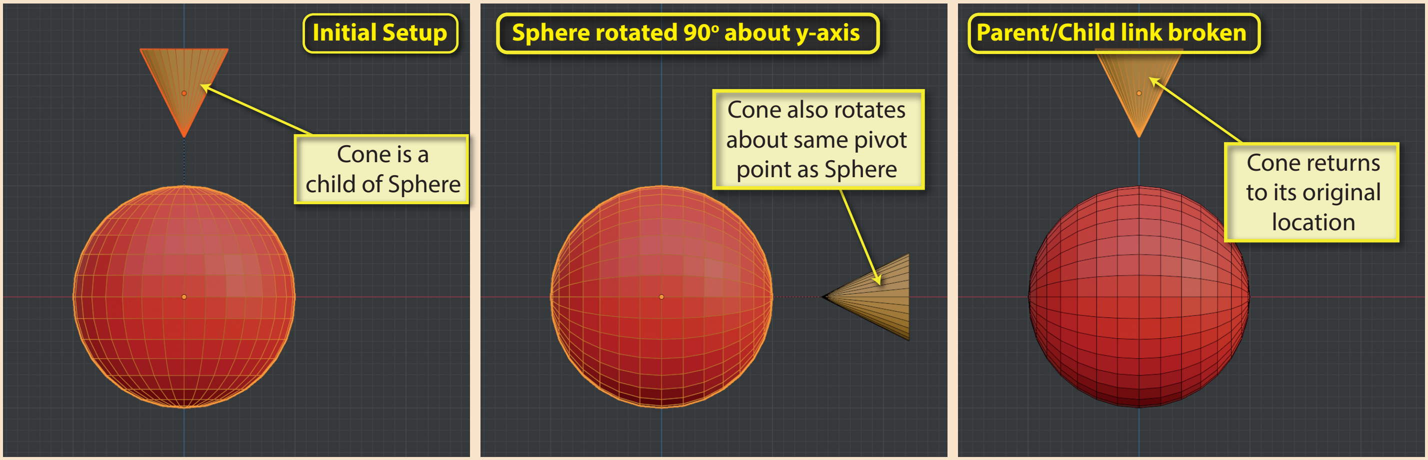
When we create a parent/child link, Blender sets up an extra data block known as the Parent Inverse Matrix which contains any changes to the parent and is resposible for relaying this information to the child.



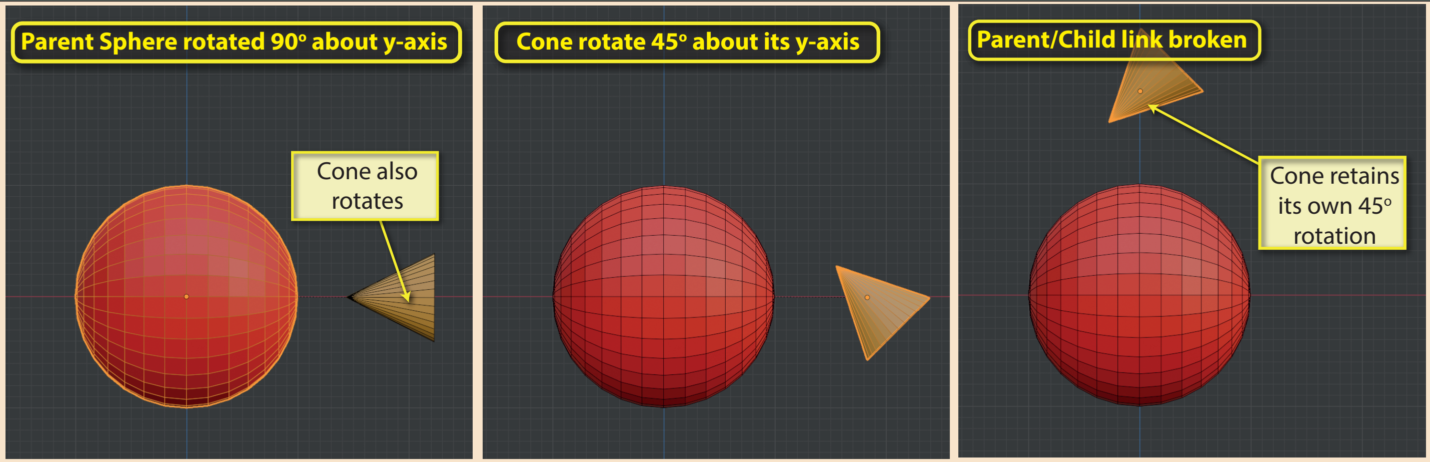
If any change is made directly to the child, only the Data Block of the child changes, the Inverse Matrix is unaffected.



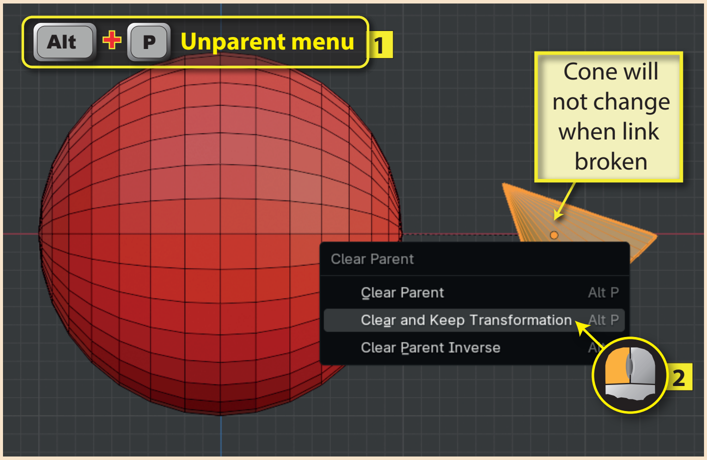
When we remove the parent/child relationship, the Inverse Matrix is deleted and its data no longer influences the child object. This means that any changes to the child that came via the matrix are removed and its state is determined solely by its own Data Block. Below we can see how a Cone which is a child of a sphere reacts to the link being broken.


Initial Setup
Cone is a child of Sphere
Sphere rotated 90° about y-axis
Cone also rotates about same pivot point as Sphere
Parent/Child link broken
Cone returns to its original location

If the child object is changed directly in any way while linked to the parent object, that change will be retained when the link is broken. Below the Cone is rotated directly while still linked to the Sphere. When the link is broken that rotation remains in effect.

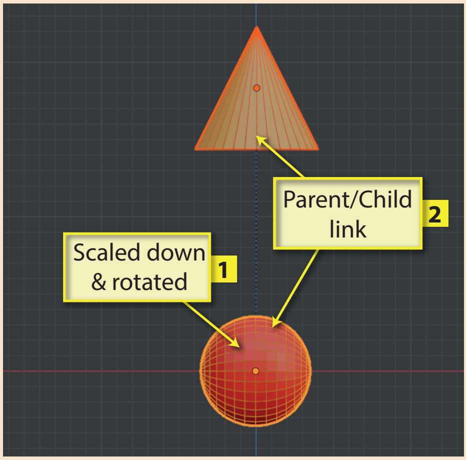

Parent Sphere rotated 90° about y-axis
Cone also rotates
Cone rotate 45° about its y-axis
Parent/Child link broken
Cone retains its own 45° rotation

If we want to retain the changes made by the parent to the child object, then, when we press **Alt+P**, we must choose **Clear and Keep Transformation** from the popup menu.

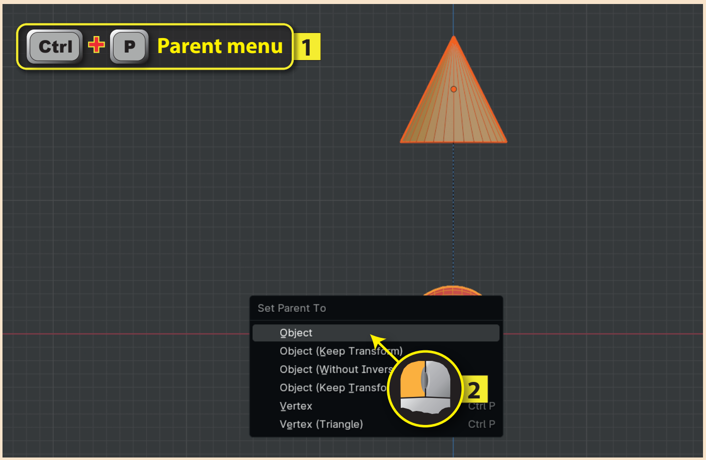


The third option in the **Clear Parent** popup menu, **Clear Parent Inverse**, does not, in fact, break the parent/child link. In fact, it clears the contents of the ***Parent Inverse Matrix***, making the child object react directly to the parent's Data Block.

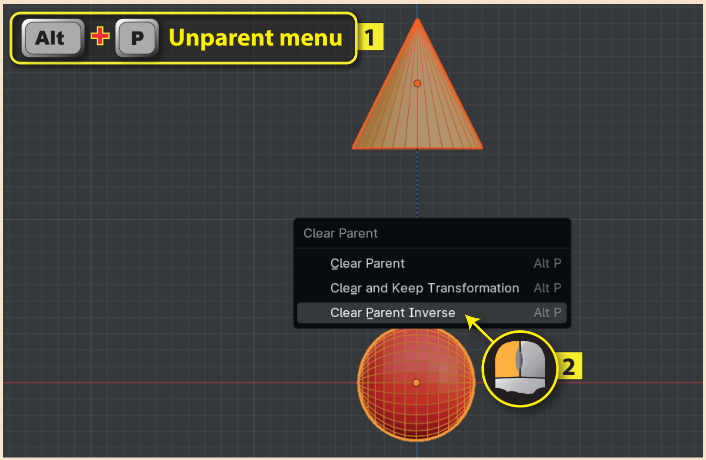
To demonstrate this effect, we'll start with our red Sphere, which we will scale down and rotate 90° about its y-axis before making it the parent of the Cone.



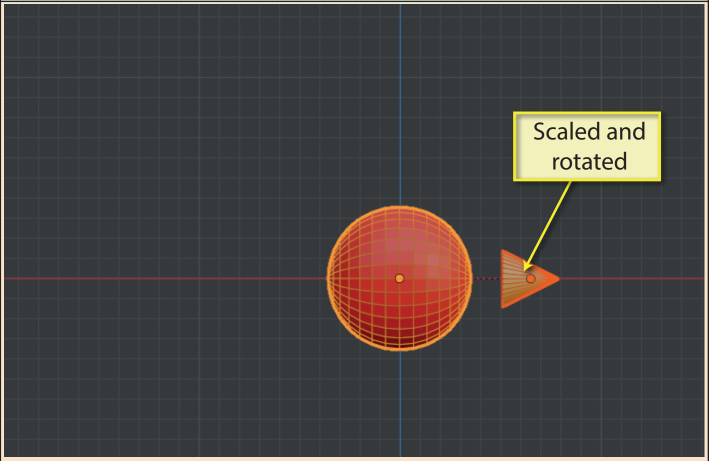
Now, with first the Cone and then the Sphere selected, we'll create a parent/child link.



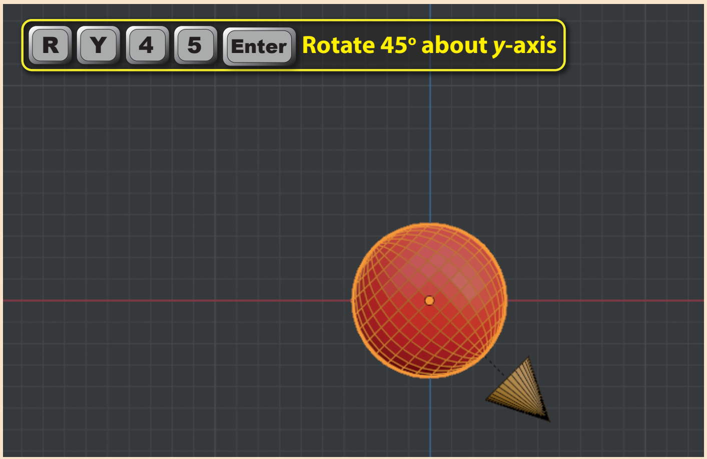
Without performing any other operations, we'll display the Clear Parent menu and select


Alt + P Unparent menu 1
Clear Parent
Clear Parent Alt P
Clear and Keep Transformation Alt P
Clear Parent Inverse Alt P
2

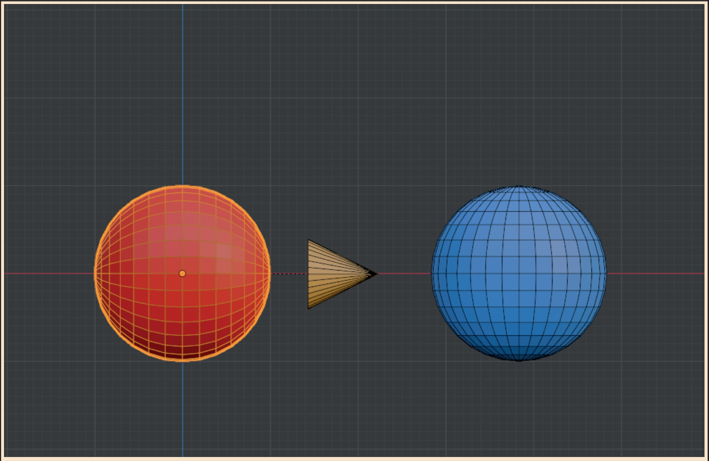
With the Inverse Matrix cleared, the earlier changes to the Sphere now affect the Cone directly causing it to resize and rotate.



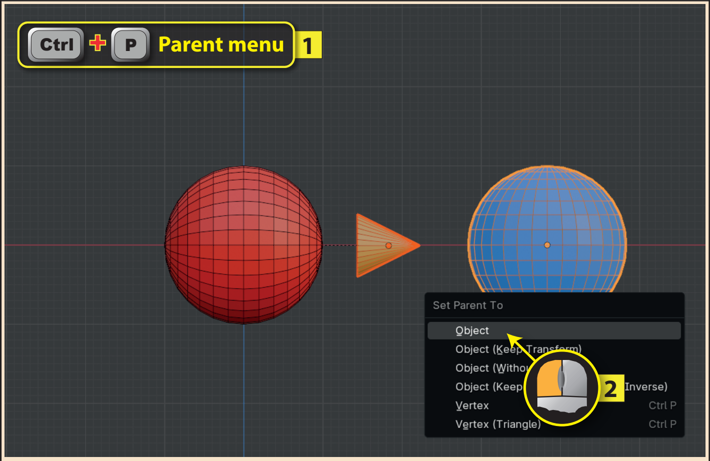
However, the parent/child relationship remains and any changes to the Sphere will also occur in the Cone. Below we can see the result of selecting the Sphere and rotating it by 45° about its *y*-axis.



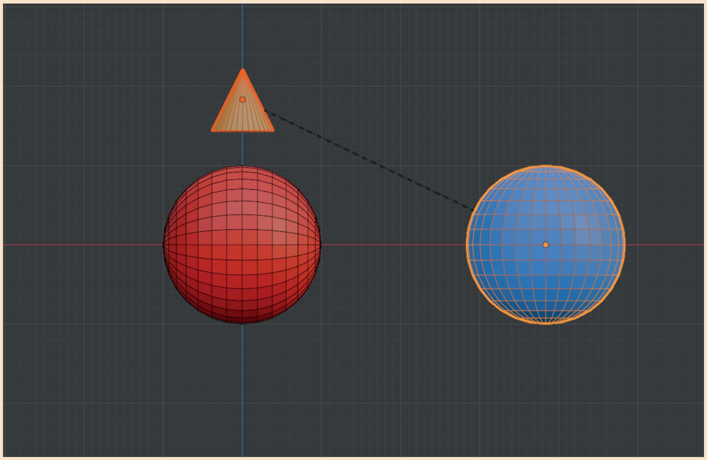
Next we'll look at the other options available in the **Set Parent To** menu. To find out the effects of each, we'll start with the setup shown below where the Cone is a child of the red Sphere which is then rotated by 90° about its y-axis.



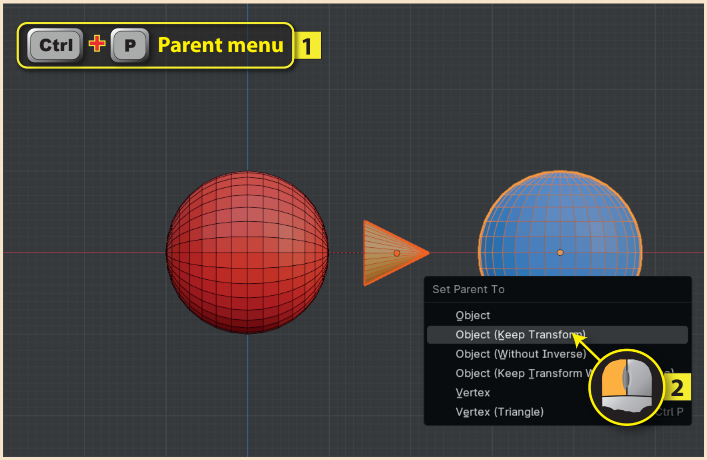
Now we want to change the Cone's parent to the blue Sphere. If, after selecting the Cone then the blue Sphere, we choose the usual option, **Object**, ...



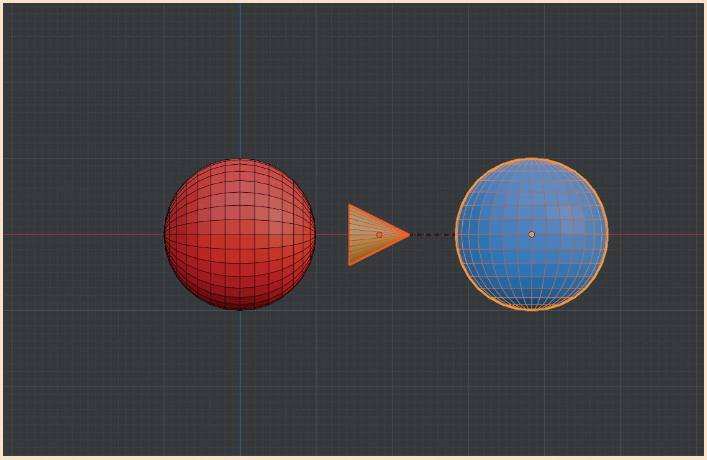
...the Cone will return to its original position - having discarded the changes coming from the Inverse Matrix - before linking to the blue Sphere.



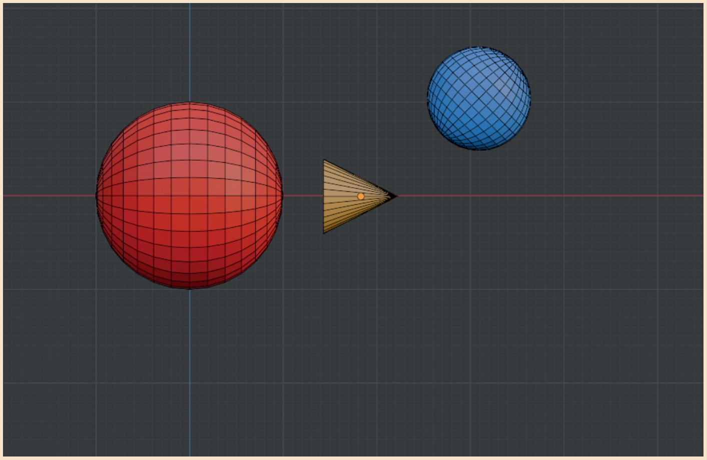
However, if we had selected the second entry in the **Set Parent To** menu, **Object (Keep Transform)**,...



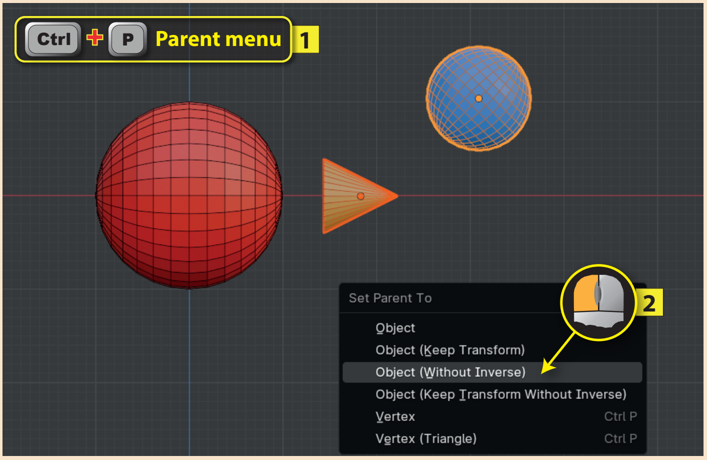
...the changes created by the red Sphere's Inverse Matrix would have been retained by the Cone.



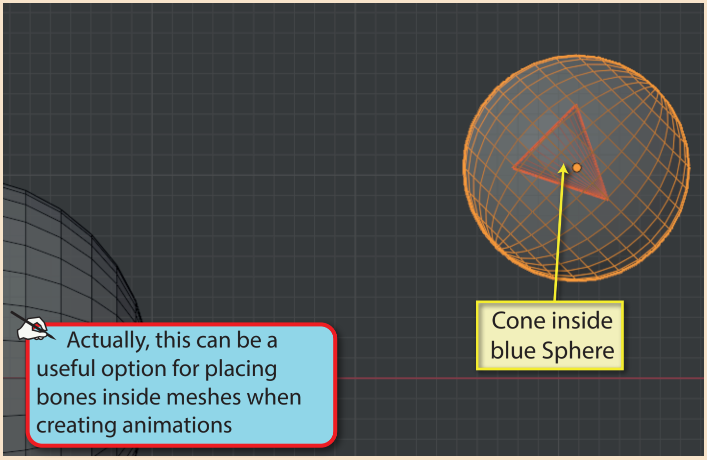
To demonstrate the third **Set Parent To** option, we'll start with the situation below where the blue Sphere has been scaled, moved and rotated before any link is made.



When, after selecting the Cone and blue Sphere, we select the third parenting option, **Object (Without Inverse)**...

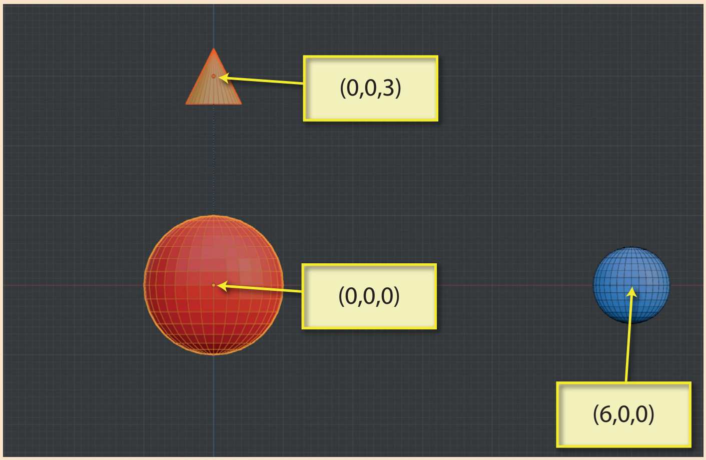


The Cone position, scale and rotation, change to match that of the blue Sphere. In fact, the Cone ends up inside the blue Sphere and we can only see it by switching on X-Ray mode.

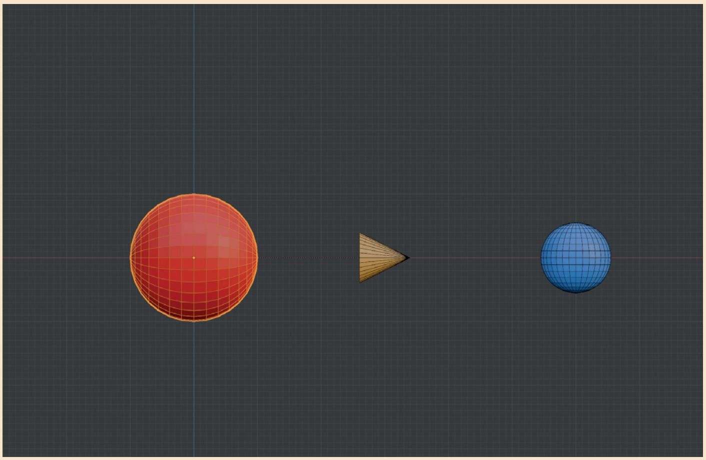


Actually, this can be a useful option for placing bones inside meshes when creating animations

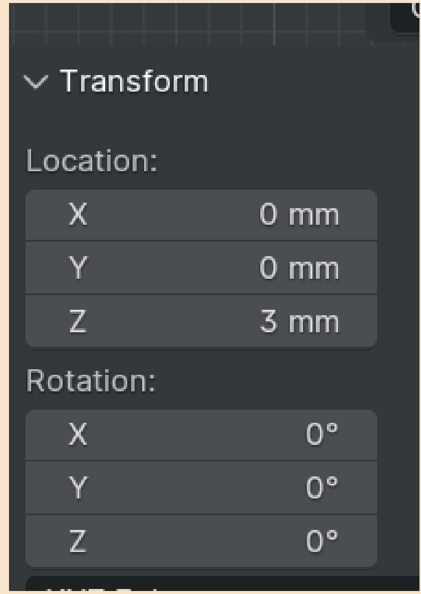
To understand the fourth entry in the **Set Parent To** menu, we'll look at the stated location of each object with the initial setup shown below.



We'll link the Cone to the red Sphere using **Set Parent To>Object**, then rotate the red Sphere by 90° about its y-axis.



But if we look at the Sidebar with the Cone selected, we'll see that its Location is still given as (0,0,3). This is because the Sidebar takes its information solely from the Cone's Data Block.



Transform

Location:

X 0 mm

Y 0 mm

Z 3 mm

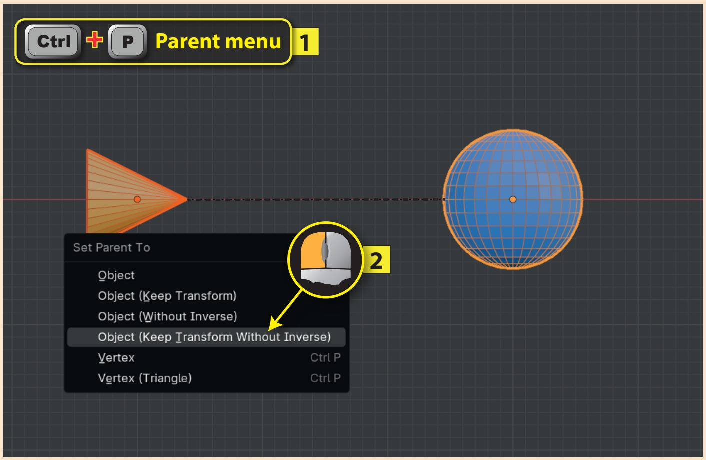
Rotation:

X 0°

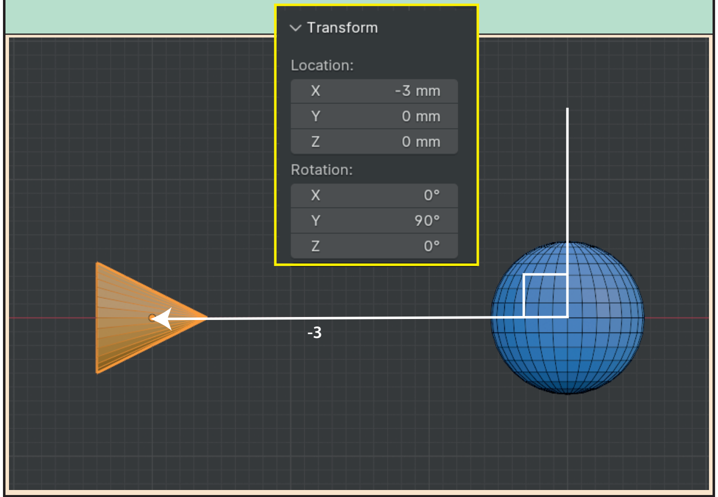
Y 0°

Z 0°

Now, if we link the Cone to the blue Sphere, using the **Object Keep Transform Without Inverse)** option...



...we'll see that the Location and Rotation values of the Cone have changed. In fact, the values are now measured from the origin of its new parent object.



Transform

Location:

X -3 mm

Y 0 mm

Z 0 mm

Rotation:

X 0°

Y 90°

Z 0°

And, because, the link has vaused the Cone's Data Block to change, if we break the parent/child link, the Cone will use these values to relocate itself.