3D Layout Groups

Thanks for purchasing 3D Layout Groups! Here are some tips to help get you started.

To create a layout group, just add the **LayoutGroup3D** component to any gameobject. The children of a gameobject with the **LayoutGroup3D** component will be arranged by the layout group just like the uGUI version.

IMPORTANT NOTE:
*Changing any of the LayoutGroup3D's public variables at runtime will NOT update / rebuild the layout. You must call the function **RebuildLayout()** to apply any changes made at runtime. This is for performance reasons, as rebuilding the layout every frame could become expensive if you have many complex layouts in your scene. This way, you can make as many changes as you want to the variables in a single frame, and then call RebuildLayout once to have all those changes applied, without rebuilding the layout for every individual change. You can force the layout to rebuild every frame with the bool ForceContinuousRebuild.*

All Layout Group Styles share two settings: the **ElementDimensions** Vector3 and the **StartPositionOffset** Vector3.

ElementDimensions specifies the 3D size of the child elements and is used to calculate the layouts. For instance, if the children are standard Unity cubes, the **ElementDimensions** would be (1, 1, 1). If you increased the x scale of the cubes to 2, then you should change the **ElementDimensions** to (2, 1, 1) etc. You can also play with the **ElementDimensions** to change the spacing of the layout in each axis separately. Having children with different dimensions isnt currently supported, but it is at the top of the priority list for future updates.

The **StartPositionOffset** allows you to change the local position of the first element in the layout, effectively shifting the layout group around within the groups local space. By default, all layouts are centered at transform with the **LayoutGroup3D** component on it.

All layout styles, with the exception of Radial, have some number of **Alignment** options, which allows you to change how the elements are aligned relative to the **LayoutGroup3D's** transform in each axis, similar to text alignment. The options are **Min, Center, Max**. By default, the elements start at the group transform's center, which is equivalent to the Min Alignment. Choosing Center will result in the mid point of all elements along that axis to be at the group's transform position. Max will result in the last element in that axis to be at the group's transform position. This is

effectively an automatic way of computing an additional StartPositionOffset such that all elements maintain their alignment when new ones are added or removed. Depending on which axis the Alignment controls, the meaning of Min, Center, Max varies. For example, if you have a GridLayout with the PrimaryAxis set to X, then the PrimaryAlignment would mean Min = Left Align, Center = Center, Max = Right Align. The SecondaryAlignment would then correspond to the Y axis, and thus Min = Bottom Align, Center = Center, and Max = Top. LinearLayouts will have a single Align option, GridLayouts will have two, and EuclideanLayouts will have three.

All layout styles, with the exception of Radial, have a Spacing float which determines the amount of distance between adjacent elements. This value is in the local space of the group, so any scaling applied to the layout group will affect this value. You can also use a negative spacing to reverse the direction of the layout.

All layout groups have a drop down to select the Layout Style which determines how the child elements are arranged.

The **LayoutGroup3D** class has four enums defined here:

enum LayoutStyle {Linear, Grid, Euclidean, Radial }

enum LayoutAxis2D {X, Y}

enum LayoutAxis3D {X, Y, Z}

enum Alignment {Min, Center, Max}

The currently supported layout styles are outlined below.

Linear Layout:

- Distributes all elements linearly along the chosen axis with equal spacing
- LayoutAxis3D **LayoutAxis**:
    - Determines which axis to distribute the elements along in the group's local space
    - If spacing is positive, the first element will be placed at the group's position plus the StartOffsetPosition Vector, with all subsequent elements being distributed in the positive direction along the selected axis. Negative spacing will reverse the direction.
- Alignment **Alignment**:

    - Determines how the linear elements are aligned relative to the group's position. Min is equivalent to Left Align and Max is equivalent to Right align, assuming the LayoutAxis is X.

Grid Layout:

- Distributes all elements in a rectangular grid in the local XY plane
- LayoutAxis2D **PrimaryLayoutAxis**:
    - The Grid layout will arrange the elements first in one axis, then in the other. The **PrimaryLayoutAxis** determines which axis (X or Y) is laid out first. It will ensure that the number of elements in the primary axis matches the **ConstraintCount** variable, with the number of elements along the other axis determined by the number of total elements in the group.
    - For example, if the group has 6 elements, the primary layout axis is the X axis and the constraint count is 3, then the grid would lay out 3 columns with 2 rows. If the count is changed to 8, there will still be 3 columns but two columns would have 3 rows and the last column would have 2.
- int **ConstraintCount**:
    - This value determines the desired number of elements along the selected **PrimaryLayoutAxis**. Just like the uGUI grid layout group, the resulting grid will attempt to create this number of columns (if X) or rows (if Y) and fill in the other axis dynamically.
- Alignment **PrimaryAlignment**:
    - Determines the element Alignment along the **PrimaryLayoutAxis.** The inspector will change the axis name to match the selected **PrimaryLayoutAxis** (X Alignment / Y Alignment)
- Alignment **SecondaryAlignment**
    - Determines the element Alignment along the **SecondaryLayoutAxis.** The inspector will change the axis name to match the selected **SecondaryLayoutAxis** (X Alignment / Y Alignment)

Euclidean Layout:

- Distributes elements in a 3D Euclidean space, i.e. in a cubic pattern
- This is essentially a GridLayout with an extra dimension and thus an extra LayoutAxis
- LayoutAxis3D **PrimaryLayoutAxis**:

    - Just like the grid layout, you can specify any axis to be the first that the constraint is enforced along.
- Int **PrimaryConstraintCount**:
    - The constraint count corresponding to the **PrimaryLayoutAxis**. This will be the number of elements to lay out along the selected axis.
- LayoutAxis3D **SecondaryLayoutAxis**:
    - The other axis to lay out elements along after the **PrimaryLayoutAxis** constraint is satisfied.
    - The number of elements in the last axis will be determined by a combination of the two constraint counts and the total number of elements in the group.
- Int **SecondaryConstraintCount**:
    - The constraint count corresponding to the **SecondaryLayoutAxis**. This will determine the number of elements along the secondary axis. If the total number of elements is less than **PrimaryConstraintCount** * **SecondaryConstraintCount**, this constraint might not be satisfied.
- Alignment **PrimaryAlignment**:
    - Determines the element Alignment along the **PrimaryLayoutAxis.**
- Alignment **SecondaryAlignment**:
    - Determines the element Alignment along the **SecondaryLayoutAxis.**
- Alignment **TertiaryAlignment**:
    - Determines the element Alignment along the **TertiaryLayoutAxis.**

Radial Layout

- Arranges elements evenly along a circle centered at the layout group's transform with a variable radius.
- bool **UseFullCircle**:
    - If true, the MaxArcAngle will be automatically calculated such that all elements are evenly distributed along the circle, with the first and last elements non-overlapping. Adding or removing elements will recalculate the spacing to keep the elements evenly distributed.
- float **MaxArcAngle**:
    - Determines the portion of the circle on which to distribute the elements along with the last element falling on the polar position determined by the angle. A value of 360 means both the first and last element are in the same position. Measured in degrees.
    - This will be overridden if **UseFullCircle** is true, and hidden in the inspector
- float **Radius**:

    - Determines the local space distance from the groups center to any element.
- float **StartAngleOffset**:
    - By default, the first element is positioned to the right of the group's center with subsequent elements filled counter clockwise. The **StartAngleOffset** shifts the starting position around the circle in either direction. Measured in degrees.
- float **SpiralFactor**:
    - Insets or outsets each element towards or away from the circles center incrementally.
    - Use this along with a **MaxArcAngle** value greater than 360 to create spirals
- bool **AlignToRadius**:
    - If false, the rotations of all elements are freely editable.
    - If true, then the element's transform.up will be aligned such that they point outwards from the center of the circle. Unchecking this box will restore their rotations to whatever they were before checking the box.