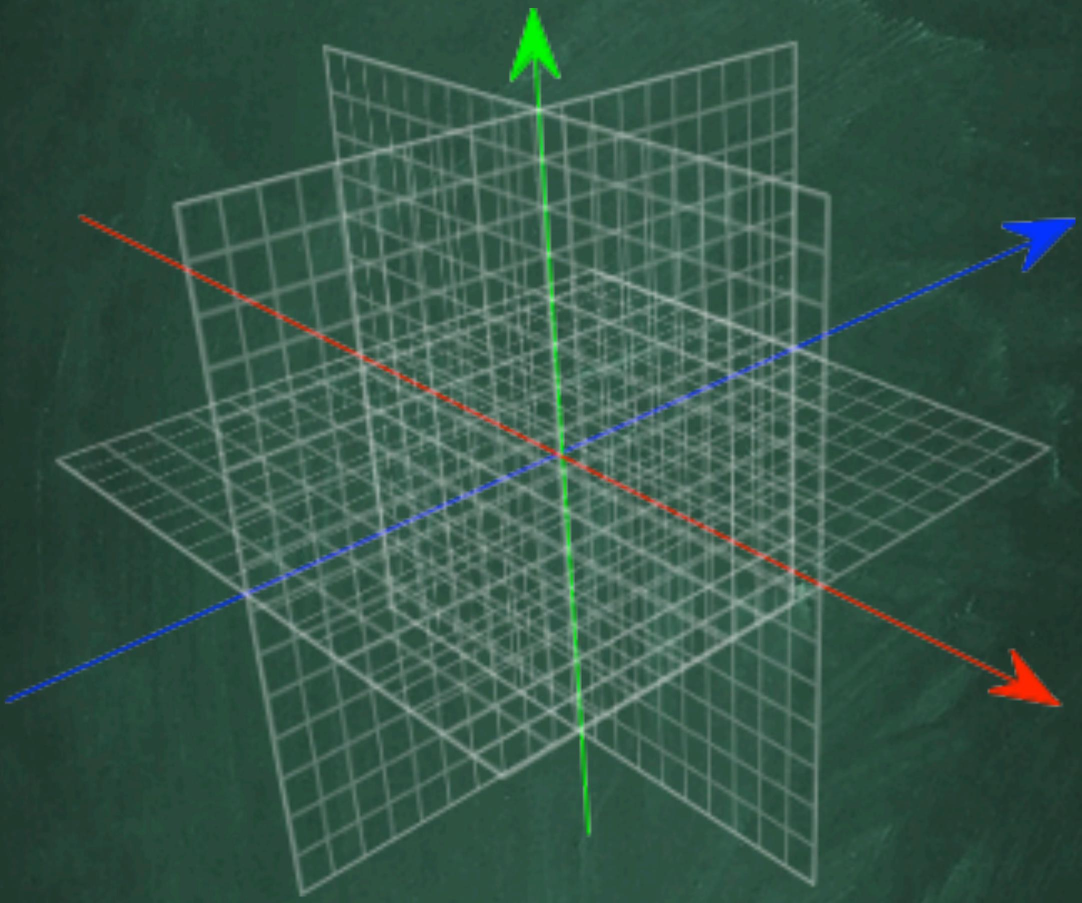
Grid Framework



Scripting Reference

Version 1.2.3

Grid Framework Scripting Reference

Each grid type has its own type and inherits from the abstract GFGrid class, which in return inherits from MonoBehaviour. I am not going to list any variables and functions inherited from MonoBehaviour since that would blow the size of this document out of proportions. Also, it would be redundant to list the same things for each grid when they all inherit from the same class, which in return only inherits from MonoBehaviour. See Unity's script reference for information on that class.

GFGrid (abstract)

Inherits from MonoBehaviour

This is the standard class all grids are based on. Aside from providing a common set of variables and a template for what methods to use, this class has no practical meaning for end users. Use this as reference for what can be done without having to specify which type of grid you are using.

Variables

relativeSize	whether the drawing/rendering will scale with spacing
size	the size of the visual representation of the grid
useCustomRenderRange	use you own values for the range of the rendering
renderFrom	custom lower limit for the rendering
renderTo	custom upper limit for the rendering
renderGrid	render the grid at runtime
renderLineWidth	the width of the lines used when rendering the grid
renderMaterial	the material for rendering, if none is given it uses a default
axisColors	colour of the axes when drawing and rendering
useSeparateRenderColor	whether to use the same colours for rendering as for drawing
renderAxisColors	separate colour of the axes when rendering (optional)
vertexColor	colour of vertices when drawing and rendering
hideGrid	don't draw the grid at all
hideAxis	hide just individual axes
hideOnPlay	hide the grid in play mode
drawOrigin	draw a little sphere at the origin of the grid

ownVertexMatrix	three-dimensional matrix for storing a list of grid vertices

Overridable Functions

WorldToGrid	converts world coordinates to grid coordinates
GridToWorld	converts grid coordinates to world coordinates
FindNearestVertex	returns the world position of the nearest vertex
FindNearestFace	returns the world position of the nearest face
FindNearestBox	returns the world position of the nearest box
GetVertexCoordinates	returns the grid position of the nearest vertex
GetFaceCoordinates	returns the grid position of the nearest face
GetBoxCoordinates	returns the grid position of the nearest box
BuildVertexMatrix	returns a Vector3[,,] containing the world position of grid vertices within a certain range of the origin
ReadVertexMatrix	returns the world position of a specified vertex in the vertex matrix
AlignTransform	fits a Transform inside the grid, but does not scale it
AlignVector3	similar to the above, except only for Vectors
ScaleTransform	scales a Transform to fit the grid but does not move it
ScaleVector3	similar to the above, except only for Vectors
RenderGrid	renders the grid at runtime
DrawGrid	draws the grid using gizmos
DrawVertices	draws the vertex matrix entries using gizmos
GetVectrosityPoints	returns an array of Vector3 ready for use with Vectrosity
GetVectrosityPointsSeparate	same as above, except all lines of the same direction are grouped together

Classes

GridPlane: enum	specifies on of three grid planes (XY, XZ and YZ)

GFGrid Variables

GFGrid.relativeSize

var **relativeSize** : boolean

If you disable this flag (default) then the values of *size*, *renderFrom* and *renderTo* will be interpreted as world unit length. If enabled they will be interpreted as grid unit lengths and the size of your drawing/rendering as well as the points returned by *GetVectrosityPoints* will depend directly on the spacing of the grid.

For example rectangular a grid with a *spacing* of (2, 0.5, 1), a *renderFrom* of (0, 0, 0) and a *renderTo* of (3, 4, 5) will be 3 x 4 x 5 world units large when the flag is disabled and 6 x 2 x 5 world units (3 x 4 x 5 grid units) large when the flag is enabled.

This affects only the drawing, coordinates in grid space and world space are still the same.

GFGrid.size

var **size** : Vector3

You can use this to set a limit for the grid. All grids are infinitely large, so use this to set limits. It also affects how much of the grid will be drawn. Note that none of the components can be less than 0.

GFGrid.useCustomRenderRange

var **useCustomRenderRange** : boolean

If false the grid will be rendered within its *size* limits, it true it will use custom limits.

GFGrid.renderFrom

var **renderFrom** : Vector3

Lower limit for the custom rendering range, can only be less or equal than *renderTo*.

GFGrid.renderTo

var **renderTo** : Vector3

Upper limit for the custom rendering range, can only be greater or equal than *renderFrom*.

GFGrid.renderGrid

var **renderGrid** : boolean

When set the grid will be rendered at runtime.

GFGrid.renderLineWidth

var **renderLineWidth** : int

The width of the rendered lines in pixels. If the width is one, then simple lines will be drawn, for higher numbers quads (rectangles) are used

GFGrid.renderMaterial

var **renderMaterial** : Material = null

The material used for rendering. If you don't specify any the system will use a default material:

```
defaultRenderMaterial = new Material( "Shader \"Lines/Colored Blended\" {" +
   "SubShader { Pass { " +
   "   Blend SrcAlpha OneMinusSrcAlpha " +
   "   ZWrite Off Cull Off Fog { Mode Off } " +
   "   BindChannels {" +
   "       Bind \"vertex\", vertex Bind \"color\", color }" +
   "} } }" );
```

GFGrid.axisColors

var **axisColors** : GFColorVector3

The colours for the gizmos when drawing and rendering the grid. You can set the colour for each axis individually.

GFGrid.useSeparateRenderColor

var **useSeparateRenderColor** : boolean

By default the same colours are used for drawing and rendering, however, if you prefer to have separate colours you can set this flag. This option is useful if you want the grid to be barely visible in the game but still clearly visible in the editor. Or, if you have several grids per level, you could have all grids looks the same in the game but different in the editor so you can distinguish them more easily.

GFGrid.renderAxisColors

var **renderAxisColors** : GFColorVector3

If the above flag is set these colours will be used for rendering, otherwise this does nothing.

GFGrid.vertexColor

var **vertexColor** : Color

The colour used by the function DrawVertices

GFGrid.hideGrid

var **hideGrid** : boolean

When this flag is set the grid will not be drawn or rendered. This will prevent all the for-loops from being fired, saving performance.

GFGrid.hideAxis

var **hideAxis** : GFBoolVector3

Same as above, but only for individual axes. Applies both to drawing and rendering.

GFGrid.hideOnPlay

var **hideOnPlay** : boolean

Same as hideGrid, but hides the grid only in play mode. That way you can judge performance without interference from the grid drawing.

GFGrid.drawOrigin

var **drawOrigin** : boolean

When set draws a small sphere at the centre of the grid.

GFGrid.ownVertexMatrix

var **ownVertexMatrix** : Vector3[,,]

A three-dimensional Vector3-array, intended for storing the vertex matrix. You can technically store the value returned by BuildVertexMatrix in any three-dimensional Vector3 array, this variable is just for your convenience.

GFGrid Classes

GFGrid.GridPlane

enum **GridPlane** {YZ, XZ, XY}

This represents a grid plane. The C# documentation contains detailed information about enumerations; the important part is that you can access the values of this class either as strings or integers. You can use this type in C# as well as Unity's Javascript.

The integer value corresponds to the plane's missing axis (X=0, Y=1, Z=2), i. e. the YZ plane has the number 0, since the X-axis (the first axis) is the missing one.

GFRectGrid

Inherits from GFGrid

Your standard rectangular grid, the characterising values are its spacing and the size, which can be set for each axis individually.

Variables

spacing	how large the grid boxes are

Functions

WorldToGrid	converts world coordinates to grid coordinates
GridToWorld	converts grid coordinates to world coordinates
FindNearestVertex	returns the world position of the nearest vertex
FindNearestFace	returns the world position of the nearest face
FindNearestBox	returns the world position of the nearest box
GetVertexCoordinates	returns the grid position of the nearest vertex
GetFaceCoordinates	returns the grid position of the nearest face
GetBoxCoordinates	returns the grid position of the nearest box
BuildVertexMatrix	returns a Vector3[,,] containing the world position of grid vertices within a certain range of the origin
ReadVertexMatrix	returns the world position of a specified vertex in the vertex matrix
AlignTransform	fits a Transform inside the grid, but does not scale it
AlignVector3	similar to the above, except only for Vectors
ScaleTransform	scales a Transform to fit the grid but does not move it
ScaleVector3	similar to the above, except only for Vectors
DrawGrid	draws the grid using gizmos
RenderGrid	renders the grid at runtime
DrawVertices	draws the vertex matrix entries using gizmos
GetVectrosityPoints	returns an array of Vector3 ready for use with Vectrosity
GetVectrosityPointsSeparate	same as above, except all lines of the same direction are grouped together

GFRectGrid Variables

GFRectGrid.spacing

var **spacing** : Vector3

How far apart the lines of the grid are. You can set each axis separately, but none may be less than 0.1 (please contact me if you *really* need lower values).

GFRectGrid Functions

GFRectGrid.WorldToGrid

function **WorldToGrid** (**worldPoint**: Vector3) : Vector3

Takes in a position in wold space and calculates where in the grid that position is. The origin of the grid is the world position of its GameObject. Rotation is taken into account for this operation.

GFRectGrid.GridToWorld

function **GridToWorld** (**gridPoint**: Vector3) : Vector3

The opposite of WorldToGrid, this returns the world position of a point in the grid. They cancel each other out.

GFRectGrid.FindNearestVertex

function **FindNearestVertex** (**fromPoint**: Vector3,

doDebug: boolean = false) : Vector3

Returns the world position of the nearest vertex from a given point in world space. If doDebug is set, a small gizmo sphere will be drawn at that position. That drawing is not affected by the variable DrawVertices.

GFRectGrid.FindNearestFace

function **FindNearestFace** (**fromPoint**: Vector3,

thePlane: GFGrid.GridPlane,

doDebug: boolean = false) : Vector3

Similar to FindNearestVertex, it returns the world coordinates of a face on the grid. Since the face is enclosed by four vertices, the returned value is the point in between all four of the vertices. You also need to specify on which plane the face lies. If doDebug is set, then a small gizmo face will drawn there.

GFRectGrid.FindNearestBox

function **FindNearestBox** (**fromPoint**: Vector3,

doDebug: boolean = false) : Vector3

Similar to FindNearestVertex, it returns the world coordinates of a box in the grid. Since the box is enclosed by eight vertices, the returned value is the point in between all eight of them. If doDebug is set, then a small gizmo box will drawn there.

GFRectGrid.GetVertexCoordinates

function **GetVertexCoordinates** (**fromPoint**: Vector3) : Vector3

Similar to FindNearestVertex, except you get grid coordinates instead of world coordinates.

GFRectGrid.GetFaceCoordinates

function **GetFaceCoordinates** (**fromPoint**: Vector3,
    thePlane: GFGrid.GridPlane) : Vector3

Similar to FindNearestFace, except you get grid coordinates instead of world coordinates.

GFRectGrid.GetBoxCoordinates

function **GetBoxCoordinates** (**fromPoint**: Vector3) : Vector3

Similar to FindNearestBox, except you get grid coordinates instead of world coordinates.

GFRectGrid.BuildVertexMatrix

function **BuildVertexMatrix** (**width**: float,
    height: float,
    depth: float) : Vector3[,,]

Builds the vertex matrix, a three-dimensional native .NET array of Vector3 values. The size of the matrix is specified as float, but the parameters get rounded to the nearest integers. The entries contain the world position of grid vertices within the specified range. The matrix starts in the upper left front corner (front in the sense of positive Z-coordinate).

Since .NET arrays cannot be resized, this function builds the matrix from scratch every time, so be aware of what you are doing if you decide to build this matrix every frame. Personally though, in that case I would recommend using the above functions which calculate positions on the fly.

GFRectGrid.ReadVertexMatrix

function **ReadVertexMatrix** (**x**: int,
    y: int,
    z: int,
    vertexMatrix: Vector3[,,],
    warning: boolean = false) : Vector3

Reads the vertex matrix in a cartesian way, i. e. the central entry has coordinates (0, 0, 0). You can pass any Vector3[,,], but the result makes most sense if you use a matrix created by BuildVertexMatrix. Setting warning will print out a warning to the console if you try to read something beyond the size of the matrix (like trying to read (7, -2, 1) in a 2x3x2 matrix). In any case the returned value will default to (0, 0, 0)

GFRectGrid.AlignTransform

function **AlignTransform** (**theTransform**: Transform,
    rotate: boolean = true
    lockAxis: GFBoolVector3 = new GFBoolVector3) : void

Tries to fit an object inside the grid by using the object's transform. If the object's scale is an even multiple (or close to one) the object's centre will be place on an edge between faces, else on a face. Setting doRotate makes the object take on the grid's rotation. The parameter lockAxis makes the function not touch the corresponding coordinate.

GFRectGrid.AlignVector3

```
function AlignVector3 (position: Vector3
        scale: Vector3 = Vector3.one
        lockAxis: GFBoolVector3 = new GFBoolVector3(false)) : Vector3
```

Works similar to AlignTransform but instead aligns a point to the grid. The *scale* parameter is needed to simulate the “size” of point, which influences the resulting position like the scale of a Transform would do above. By default it’s set to one on all axes, placing the point at the centre of a box. The lockAxis parameter works just like above.

GFRectGrid.ScaleTransform

```
function ScaleTransform (theTransform: Transform,
        lockAxis: GFBoolVector3 = new GFBoolVector3) : void
```

Scales an object’s transform to the nearest multiple of the grid’s spacing, but does not change its position. The parameter lockAxis makes the function not touch the corresponding coordinate.

GFRectGrid.ScaleVector3

```
function ScaleVector3 (scale: Vector3
        lockAxis: GFBoolVector3 = new GFBoolVector3(false)) : Vector3
```

Like the above, but takes a Vector3 instead and returns the scaled vector. The lockAxis parameter works just like above.

GFRectGrid.DrawGrid

```
function DrawGrid () : void
function DrawGrid (from: Vector3
        to: Vector3) : void
```

Simply draws the grid using gizmos, the size is based on the *size variable* of the grid or two points in local space, not affected by scale. Keep in mind that the grid is three-dimensional, to draw it there are three for-loops nested into each other, so drawing a huge grid can slow down the editor.

The grid you see is just a visual representation of an infinitely large grid, it is fine just to draw a small part, no matter how much of the grid you need.

GFRectGrid.RenderGrid

```
function RenderGrid (width: int = 0,
        cam: Camera = null) : void
function RenderGrid (from: Vector3,
        to: Vector3,
        width: int = 0,
        cam: Camera = null) : void
```

Renders the grid at runtime based either on its *size* or two points in local space, not affected by scale. Keep in mind that the grid is three-dimensional, to draw it there are three for-loops nested into each other, so rendering a huge grid can slow down the editor. You shouldn't call the function manually, the framework makes sure it gets called at the rights places. If you still want to call it yourself, I recommend the OnPostRender() function of a camera.

If no width or camera are given or the width is one, the lines will be one pixel wide. The camera passed will be used to calculate the screen position for the points of the rectangles.

GFRectGrid.DrawVertices

function **DrawVertices** (**vertexMatrix**: Vector3[,,],

        drawOnPlay: boolean = false) : void

Draws the entries from the vertex matrix. The same warning applies as for DrawGrid. Usually the vertices won't be drawn while playing, so set drawOnPlay to true if you wan to override this.

GFRectGrid.GetVectrosityPoints

function **GetVectrosityPoints** () : Vector3[]

function **GetVectrosityPoints** (**from**: Vector3,

        to: Vector3) : Vector3[]

Returns an array of Vector3 containing the points for a discrete vector line in Vectrosity. One entry is the starting point, the next entry is the end point, the next entry is the starting point of the next line and so on. The returned points represent the grid's size.

If no arguments are passed the grid's *size* is used, otherwise the two points are. These points are in local space but not affected by scale.

GFRectGrid.GetVectrosityPointsSeparate

function **GetVectrosityPointsSeparate** () : Vector3[3][]

function **GetVectrosityPointsSeparate** (**from**: Vector3

        to Vector3) : Vector3[3][]

Similar to above, except you get a jagged array. Each of the three arrays contains the points of lines for the same direction, i. e. the first entry contains only the lines along the grid's X-axis, the second one the Y-axis, the third one the Z-axis. Example:

```
var myLines: Vector3□□ = myGrid.GetVectrosityPointsSeparate();
myLine: VectorLine = new VectorLine("Y-Lines", myLines[1], Color.green, null, 3.0);
```

GFHexGrid

Inherits from GFGrid

A regular hexagonal grid that forms a honeycomb pattern. it is characterized by the radius (distance from the centre of a hexagon to one of its vertices) and the depth (distance between two honeycomb layers). Hex grids use a herringbone pattern for their coordinate system, please refer to the user manual for information about how that coordinate system works

Variables

radius	distance from the centre of a hex to a vertex
depth	distance between two grid layers
gridPlane	whether it's an XY-, XZ- or YZ-grid
hexSideMode	pointy sides or flat sides
gridStyle	the shape of the overall grid, affects only drawing and rendering, not the calculations

Functions

WorldToGrid	converts world coordinates to grid coordinates
GridToWorld	converts grid coordinates to world coordinates
FindNearestVertex	returns the world position of the nearest vertex
FindNearestFace	returns the world position of the nearest face
FindNearestBox	returns the world position of the nearest box
GetVertexCoordinates	returns the grid position of the nearest vertex
GetFaceCoordinates	returns the grid position of the nearest face
GetBoxCoordinates	returns the grid position of the nearest box
BuildVertexMatrix	returns a Vector3[,,] containing the world position of grid vertices within a certain
ReadVertexMatrix	returns the world position of a specified vertex in the vertex matrix
AlignTransform	fits a Transform inside the grid, but does not scale it
AlignVector3	similar to the above, except only for Vectors
ScaleTransform	scales a Transform to fit the grid but does not move it
ScaleVector3	similar to the above, except only for Vectors
DrawGrid	draws the grid using gizmos
RenderGrid	renders the grid at runtime

DrawVertices	draws the vertex matrix entries using gizmos
GetVectrosityPoints	returns an array of Vector3 ready for use with Vectrosity
GetVectrosityPointsSeparate	same as above, except all lines of the same direction are grouped together

Classes

HexOrientation: enum	pointy sides or flat sides
HexGridShape: enum	rectangular or compact rectangular

GFHexGrid variables

GFHexGrid.radius

var **radius** : float

Radius refers to the distance between the centre of a hexagon and one of its vertices. Since the hexagon is regular all vertices have the same distance from the centre. In other words, imagine a circumscribed circle around the hexagon, its radius is the radius of the hexagon. The value may not be less than 0.1 (please contact me if you *really* need lower values).

GFHexGrid.depth

var **depth** : float

As mentioned in the user manual all grids are three-dimensional. A honeycomb pattern on its own is just two-dimensional, so they get stacked on top of each other to form a three-dimensional grid. The depth is the distance between two honeycomb patterns an a lower values means a more dense grid. The value may not be less than 0.1 (please contact me if you *really* need lower values).

GFHexGrid.gridPlane

var **gridPlane** : GFGrid.GridPlane

This tells the grid o which of the three planes (XY, XZ or YZ) the honeycomb pattern lies. If for example you were to make a top-down game you could technically just take an XY-grid and rotate it, but it is more intuitive to use an XZ-grid, because the coordinates will be along the XZ-plane. This means that one unit into the Z-direction really changes the Z coordinate inside the grid's coordinate system instead of the Y-coordinate.

GFHexGrid.hexSideMode

var **hexSideMode** : GFHexGrid.HexOrientation

Whether the grid has pointy sides or flat sides. This affects both the drawing and the calculations.

GFHexGrid.gridStyle

var **gridStyle** : GFHexGrid.HexGridShape

The shape when drawing or rendering the grid. This only affects the grid's appearance, but not how it works.

GFHexGrid Functions

GFHexGrid.WorldToGrid

function **WorldToGrid** (**worldPoint**: Vector3) : Vector3

Takes in a position in world space and calculates where in the grid that position is. The origin of the grid is the centre of the central face. Rotation is taken into account for this operation. The coordinate system used is the “herringbone pattern”, please refer to the user manual to learn how the herringbone pattern works.

GFHexGrid.GridToWorld

function **GridToWorld** (**gridPoint**: Vector3) : Vector3

The opposite of *WorldToGrid*, this returns the world position of a point in the grid. They cancel each other out.

GFHexGrid.FindNearestVertex

function **FindNearestVertex** (**fromPoint**: Vector3
doDebug: boolean = false) : Vector3

Returns the world position of the nearest vertex from a given point in world space. If *doDebug* is set, a small gizmo sphere will be drawn at that position. That drawing is not affected by the variable *DrawVertices*.

GFHexGrid.FindNearestFace

function **FindNearestFace** (**fromPoint**: Vector3,
doDebug: boolean = false) : Vector3

Similar to *FindNearestVertex*, it returns the world coordinates of a face on the grid. Unlike rectangular grids you don’t need to specify a plane, the grid’s own plane is used. If *doDebug* is set, then a small gizmo face will drawn there.

GFHexGrid.FindNearestBox

function **FindNearestBox** (**fromPoint**: Vector3,
doDebug: boolean = false) : Vector3

Similar to *FindNearestFace*, it returns the world coordinates of a box in the grid. The Z-coordinate (or its equivalent for non XY-grids) is between two honeycomb patterns. If doDebug is set, then a small gizmo box will drawn there.

GFHexGrid.GetVertexCoordinates

function **GetVertexCoordinates** (**fromPoint**: Vector3) : Vector3

Similar to *FindNearestVertex*, except you get grid coordinates instead of world coordinates. Uses the herringbone pattern as the coordinate system.

GFHexGrid.GetFaceCoordinates

function **GetFaceCoordinates** (**fromPoint**: Vector3) : Vector3

Similar to *FindNearestFace*, except you get grid coordinates instead of world coordinates. Uses the herringbone pattern as the coordinate system.

GFHexGrid.GetBoxCoordinates

function **GetBoxCoordinates** (**fromPoint**: Vector3) : Vector3

Similar to *FindNearestBox*, except you get grid coordinates instead of world coordinates. Uses the herringbone pattern as the coordinate system.

GFHexGrid.BuildVertexMatrix

function **BuildVertexMatrix** (**width**: float
    height: float,
    depth: float) : Vector3[,,]

Builds the vertex matrix, a three-dimensional native .NET array of Vector3 values. The size of the matrix is specified as float, but the parameters get rounded to the nearest integers. The entries contain the world position of grid vertices within the specified range. The matrix starts in the upper left front corner (front in the sense of positive Z-coordinate).

Since .NET arrays cannot be resized, this function builds the matrix from scratch every time, so be aware of what you are doing if you decide to build this matrix every frame. Personally though, in that case I would recommend using the above functions which calculate positions on the fly.

GFHexGrid.ReadVertexMatrix

function **ReadVertexMatrix** (**x**: int,
    y: int,
    z: int) : Vector3

Reads the vertex matrix in a cartesian way, i.e. the central entry has coordinates (0, 0, 0). You can pass any Vector3[,,], but the result makes most sense if you use a matrix created by BuildVertexMatrix. Setting warning will print out a warning to the console if you try to read something beyond the size of the matrix (like trying to read (7, -2, 1) in a 2x3x2 matrix). In any case the returned value will default to (0, 0, 0)

GFHexGrid.AlignTransform

function **AlignTransform** (**theTransform**: Transform,
    rotate: boolean = true,
    lockAxis: GFBoolVector3 = new GFBoolVector3) : Vector3

Places on abject onto the grid by positioning its pivot point on the centre of the nearest face. Please refer to the user manual for more information. Setting doRotate makes the object take on the grid's rotation. The parameter lockAxis makes the function not touch the corresponding coordinate.

GFHexGrid.AlignVector3

function **AlignVector3** (**position**: Vector3,
    scale: Vector3 = Vector3.one,
    lockAxis: GFBoolVector3 = new GFBoolVector3) : Vector3

Works similar to AlignTransform but instead aligns a point to the grid. The lockAxis parameter works just like above.

GFHexGrid.ScaleTransform

function **ScaleTransform** (**theTransform**: Transform,

    lockAxis: GFBoolVector3 = new GFBoolVector3) : Vector3

Scales an object's transform to the nearest multiple of the grid's *radius* and *depth*, but does not change its position. The parameter lockAxis makes the function not touch the corresponding coordinate.

GFHexGrid.ScaleVector3

function **ScaleVector3** (**scale**: Vector3,

    lockAxis: GFBoolVector3 = new GFBoolVector3) : Vector3

Like the above, but takes a Vector3 instead and returns the scaled vector. The lockAxis parameter works just like above.

GFHexGrid.DrawGrid

function **DrawGrid** () : void

function **DrawGrid** (**from**: Vector3

    to: Vector3) : void

Simply draws the grid using gizmos, the size is based on the *size variable* of the grid or two points in local space, not affected by scale. Keep in mind that the grid is three-dimensional, to draw it there are three for-loops nested into each other, so drawing a huge grid can slow down the editor.

The grid you see is just a visual representation of an infinitely large grid, it is fine just to draw a small part, no matter how much of the grid you need.

GFHexGrid.RenderGrid

function **RenderGrid** (**width**: int = 0,

    cam: Camera = null) : void

function **RenderGrid** (**from**: Vector3,

    to: Vector3,

    width: int = 0,

    cam: Camera = null) : void

Renders the grid at runtime based either on its *size* or two points in local space, not affected by scale. Keep in mind that the grid is three-dimensional, to draw it there are three for-loops nested into each other, so rendering a huge grid can slow down the editor. You shouldn't call the function manually, the framework makes sure it gets called at the rights places. If you still want to call it yourself, I recommend the OnPostRender() function of a camera.

If no width or camera are given or the width is one, the lines will be one pixel wide. The camera passed will be used to calculate the screen position for the points of the rectangles.

GFHexGrid.DrawVertices

function **DrawVertices** (**vertexMatrix**: Vector3[,,],

drawOnPlay: boolean = false) : Vector3

Draws the entries from the vertex matrix. The same warning applies as for DrawGrid. Usually the vertices won't be drawn while playing, so set drawOnPlay to true if you wan to override this.

GFHexGrid.GetVectrosityPoints

function **GetVectrosityPoints** () : Vector3[]

function **GetVectrosityPoints** (**from**: Vector3,

to: Vector3) : Vector3[]

Returns an array of Vector3 containing the points for a discrete vector line in Vectrosity. One entry is the starting point, the next entry is the end point, the next entry is the starting point of the next line and so on. The returned points represent the grid's size.

If no arguments are passed the grid's *size* is used, otherwise the two points are. These points are in local space but not affected by scale.

GFHexGrid.GetVectrosityPointsSeparate

function **GetVectrosityPointsSeparate** () : Vector3[3][]

function **GetVectrosityPointsSeparate** (**from**: Vector3

to Vector3) : Vector3[3][]

Similar to above, except you get a jagged array. Each of the three arrays contains the points of lines for the same direction, i. e. the first entry contains only the lines along the grid's X-axis, the second one the Y-axis, the third one the Z-axis. Example:

```
var myLines: Vector3□□ = myGrid.GetVectrosityPointsSeparate();
myLine: VectorLine = new VectorLine("Y-Lines", myLines[1], Color.green, null, 3.0);
```

GFHexGrid classes

GFHexGrid.HexOrientation

enum **HexOrientation** {PointySides, FlatSides}

Information whether the grid has pointy sides or flat sides

GFHexGrid.HexGridShape

enum **HexGridShape** {Rectangle, CompactRectangle}

The shape of the drawn hex grid. *Rectangle* draws the hex grid as a rectangle where every second column (relative to the centre of the grid) is pulled upwards. *CompactRectangle* is similar, except that the upwards shifted columns have their topmost hexagon cut off.

GFColorVector3

This class groups three colours together, similar to how Vector3 groups three float numbers together. Just like Vector3 you can read and assign values using x, y, or an indexer.

Variables

x	X component of the colour vector
y	Y component of the colour vector
z	Z component of the colour vector
this[int index]	Access the X, Y or Z components using [0], [1], [2] respectively

Constructors

GFColorVector3	Creates a new colour vector with given X, Y and Z components

GFColorVector3 Variables

GFColorVector3.x

var **x** : Color

X component of the colour vector.

GFColorVector3.y

var **y** : Color

Y component of the colour vector.

GFColorVector3.z

var **z** : Color

Z component of the colour vector.

GFColorVector3.this [int index]

var **this[index** : int**]** : Color

Access the x, y, z components using [0], [1], [2] respectively. Example:

```
var c : GFColorVector3;
c[1] = Color.green; // the same as c.y = Color.green
```

GFColorVector3 constructors

GFColorVector3.GFColorVector3

static function **GFColorVector3(x**: Color**, y**: Color**, z**: Color**)** : GFColorVector3

Creates a new colour vector with given x, y, z components.

static function **GFColorVector3(color**: Color**)** : GFColorVector3

Creates a new colour vector where all components are set to the same colour.

static function **GFColorVector3()** : GFColorVector3

Creates a new standard RGB colour vector where all three colours have their alpha set to 0.5

GFBoolVector3

This class groups three booleans together, similar to how Vector3 groups three float numbers together. Just like Vector3 you can read and assign values using x, y, or an indexer.

Variables

x	X component of the bool vector
y	Y component of the bool vector
z	Z component of the bool vector
this[int index]	Access the X, Y or Z components using [0], [1], [2] respectively

Constructors

GFBoolVector3	Creates a new bool vector with given X, Y and Z components

GFBoolVector3 Variables

GFBoolVector3.x

var **x** : boolean

X component of the bool vector.

GFBoolVector3.y

var **y** : boolean

Y component of the bool vector.

GFBoolVector3.z

var **z** : boolean

Z component of the bool vector.

GFBoolVector3.this [int index]

var **this[index** : int**]** : boolean

Access the x, y, z components using [0], [1], [2] respectively. Example:

```
var b : GFBoolVector3;
b[1] = true; // the same as b.y = true
```

GFBoolVector3 constructors

GFBoolVector3.GFBoolVector3

static function **GFBoolVector3(x**: boolean**, y**: boolean**, z**: boolean**)** : GFBoolVector3

Creates a new bool vector with given x, y, z components.

static function**GFBoolVector3(condition**: boolean**)** : GFBoolVector3

Creates a new bool vector with all components set to *condition*.

static function **GFBoolVector3()** : GFBoolVector3

Creates a new standard all false vector.