GIMME

Instanced Health Bars

Manual

Contents

1 Dependencies and Setup

In order for the package to work, the following dependencies need to be installed either manually or via the package manager:

- **Burst 1.8.7** or above
- **Collections 2.1.4** or above
- **Mathematics 1.2.6** or above (might work with lower versions; untested)
- **Splines 2.2.1** or above (only for the demo scene)

1.1 Migration

Some custom editors use hard-coded paths for finding certain assets (e.g. textures, compute shader etc.). This prevents moving the folder around in the project. You can therefore find a tool called **Migration Helper** under **Window → Gimme → Instanced Healthbars → Migrate**. Simply select the directory you want the package to go into, and the tool will handle the rest. Make sure to close Visual Studio or any other resource-blocking programs before migrating.

2 Workflow

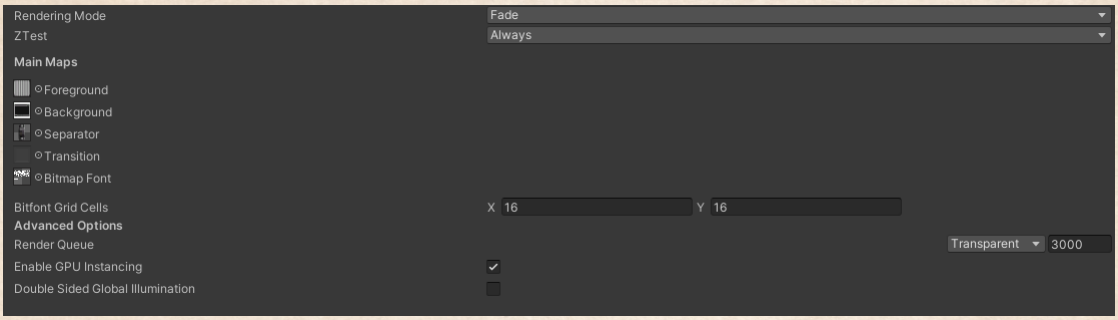
To render instanced healthbars we need four things:

- A material supporting GPU instancing
- The **HealthbarController** as a component attached to some GameObject in the scene
- At least one Health Bar Profile (which you can find an entry in the create menu)
- Some data for each instance you want to draw, sent via code to the controller

2.1 Material

You can find two shaders in the asset package: **HealthbarShader** and **HealthbarDecorationShader**. The controller expects a material created from the first shader, which also holds all the logic for rendering a health bar. The second shader is for decorating a health bar with an additional mesh and only scales and fades in the same way as the first shader.

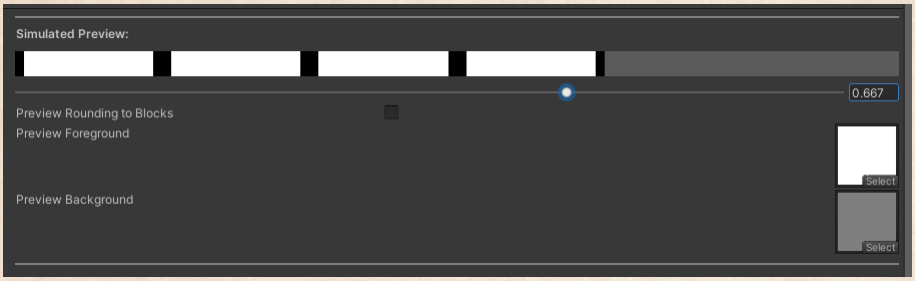
So the first step is to create a material from the **HealthbarShader**, which has an interface that looks like this:



- **Foreground**: The foreground texture. This is the actual bar that will go left and right
- **Background**: The background of a health bar. You can use alpha cutout or transparency to create different shapes other than rectangles
- **Separator**: An optional texture, allowing you to separate blocks of health from each other
- **Transition**: Another optional texture that is displayed before the background, but behind the foreground, when health is lost during a transition
- **Bitmap Font**: If numbers should be displayed in the health bar (see the Health Bar Profile section for more information), then the individual symbols are sampled from this texture. As the name suggests, only bitmap fonts (where each glyph has the same size) are supported. The individual symbols should adhere to an ASCII standard (i.e. 'A' should be placed at position 65 in the texture). If no text is rendered, the texture is ignored
- **Bitfont Grid Cells**: The number of cells in the Bitmap Font in each dimension - i.e. how many symbols are in each row and column of the texture

2.2 Profile

You can create a profile anywhere in your project in the create-menu under **GimmeInstancedHealthbars/Profile**. A profile contains data that is used by multiple instances. You can later assign a profile to each instance from your code and change them around as you desire.



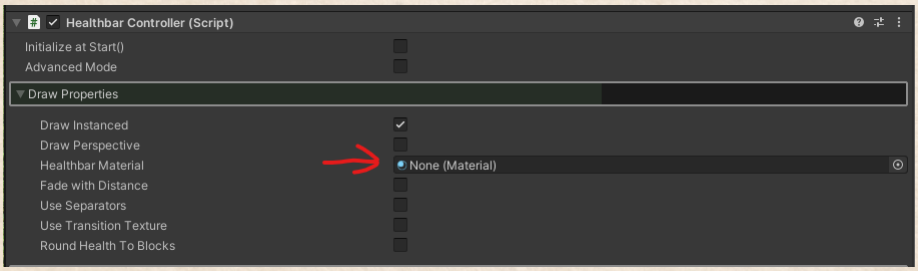
Create a profile anywhere in your project. You should already be able to see a simplified preview of the health bar.

You can find a more detailed explanation of the profile in Health Bar Data and Profile

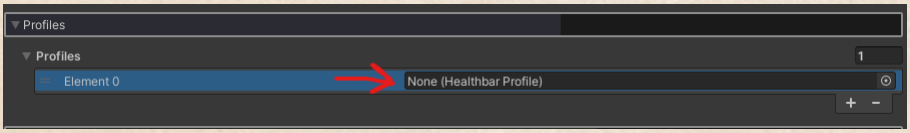
2.3 Controller

You can either attach the **HealthbarController**-component to an existing GameObject in your scene, or create it by dragging the BaseController-prefab into your scene. You can find it in **GimmeInstancedHealthBars/Prefabs/BaseController.prefab**.

After creating it, we need to assign the material and profile we created before. So drag your material into the **Healthbar Material**-field under **Draw Properties**



Next, we assign the profile under **Profiles**



Every controller needs at least one profile in order for it to work. This is all we need to do to get the system to start working. The only thing remaining to do is call methods from the controller in the script, to give it the information on *where* to draw the health bar on the screen and with how much health etc.

You can find a more detailed explanation of the controller in Health Bar Controller

2.4 Providing Data to the Controller

In essence, the only thing we need to do in the code is to create as many instances of **HealthbarData** as we want to draw. Each health bar data object should have assigned a transform (e.g. the hero), an offset (e.g. a few meters above the head), a maximum health amount (e.g. 100), a current health amount (e.g. 70) and a profile id (we only created one profile, so this should be 0)

Here an example declaration:

```
HealthbarData healthbarData = new HealthbarData()
{
        attachedObject = hero.transform,
        currentHealth = 70.0f,
        maxHealth = 100.0f,
        profileID = 0,
        offset = Vector3.up * 2.0f,
};
```

After we created the object and stored it somewhere, we need to give it to the controller that is referenced somewhere in the script, by calling the method **AddHealthbar()**. But first, we need to initialize it (it is not initialized automatically, in case you want to do it after loading the game for example).

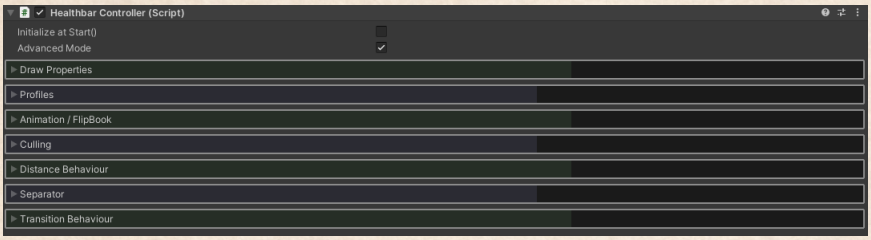
```
controller.Init();

controller.AddHealthbar(ref healthbarData);
```

And that is basically it. You should now see a health bar floating above the transform you provided to the data when starting the game. The reason the method requires **ref** is because internally a **HealthbarIdentifier** is assigned to the object. This is because of performance reasons to remove healthbars faster from the controller again (RemoveAtSwapBack).

If the explanation left something to explain to you, you can find a working example in the sample scene on how the data creation and controller methods work

3 Health Bar Controller

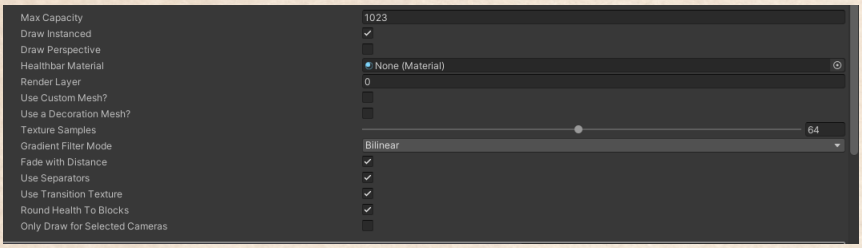
3.1 Inspector



Overview

- **Initialize at Start()**: Initialized the controller automatically upon Start() (same as any other MonoBehaviour) instead of you doing it via code
- **Advanced Mode**: Shows more advanced options in the inspector, that are only useful under special circumstances
- **Draw Properties**: Controls how and how many instances of health bars are drawn. This is also where you add and remove additional features you might or might not want to use.
- **Profiles**: Holds data that is used for multiple instances for each health bar. Each profile could represent a different type of enemy for example (which is drawn differently then).
- **Animation / FlipBook**: You can use Flipbook-Textures for the foreground or the background and control it here.
- **Culling**: Allows you to control the culling of the health bars
- **Distance Behaviour**: Controls the scale and alpha of all health bars based on their distance to the camera that is currently rendering
- **Separator**: Allows you to fade in or out the separators with distance, to prevent unreadable UI in some cases
- **Transition Behaviour**: Controls common behaviour of the transitions (when health is lost or gained) for all healthbars

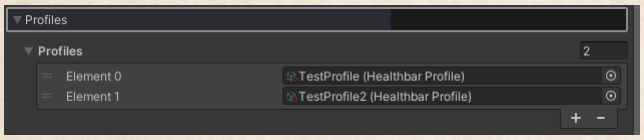
3.1.1 Draw Properties



- **Max Capacity**: Capacity with which the graphics buffers are initialized internally. This is the maximum amount of healthbars you can have simultaneously. If unsure how many you need, just set it to a high number (default is 1023, which amounts to a single draw call)
- **Draw Instanced**: You can also draw the health bars without instancing. I would not know why, but the option is here nonetheless
- **Draw Perspective**: By default, the bars are drawn as if they were UI elements. However, you can also draw them as if they were objects in the world by enabling this option
- **Health Bar Material**: Material used for rendering the health bars.
- **Render Layer**: Can be used to render the bars into a different layer (e.g. UI)
- **Custom Mesh**: Allows you to use a custom mesh, instead of a quad that is generated in the controller otherwise. As long as the UVs map correctly from 0 to 1, the health bar shader will work with any mesh you provide, still drawing instanced.
- **Decoration Mesh**: You can render an additional mesh on top of the health bar, using the health bar decoration shader. This allows you to for example encase the bar in a glass casket among other things.
- **Texture Samples**: Internally, some small texture are generated to be read again from in the shader. The higher this number, the more accurate is the distance behaviour / gradient colors etc. but the more memory the controller needs.
- **Gradient Filter Mode**: Sometimes you want to have smooth color transitions, sometimes you want them to be unfiltered.
- **Fade with Distance**: Allows you to fade in or fade away a health bar with a fade / transparent material

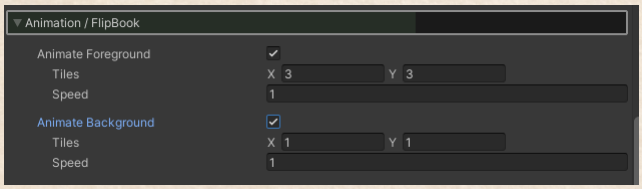
- **Use Separators**: If activated, uses the separator texture and the profile information to render separators on top of the foreground texture
- **Use Transition Texture**: When enabled, the shader will display the transition texture in the material when losing health
- **Round Health To Blocks**: Makes the health bar behave as if it has integers as health. You can get a preview on the behaviour in the profile
- **Render Text**: Activates or deactivates the rendering of text within the shaders (health numbers)
- **Only Draw for Selected Cameras**: You can tell the controller to render the health bars only for certain cameras. Can be occasionally useful for split screens or when doing some special rendering

3.1.2 Profiles



A list of Health Bar Profiles you can extend. You will need at least one in order for the controller to work. Each health bar data object has a profile ID assigned, which is just the order in this array. I.e. a health bar with profile ID of 1, uses the second element in this array as a reference for the profile information.

3.1.3 Animation / FlipBook

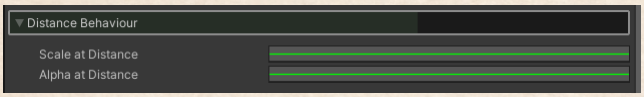


When enabling animations for either the foreground or the background, the texture are treated as flip books. The tiling specifies how many pictures are in each row or column. The speed is in frames per second.

3.1.4 Culling

By default, frustum culling is disabled for the health bars. This is because the scaling in the shader makes it difficult to calculate the bounds accurately. Another reason is that the performance gain is not very much in the first place, as the system either culls all health bars or none. If you enable it, the point positions of the health bars are taken to form a bounding box estimate for. You can then expand this box by the **Culling Box Expansion** such that culling is done correctly regardless of the scale of the health bars.

3.1.5 Distance Behaviour



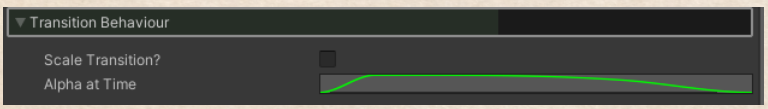
The names of the fields should explain the behaviour. In principle, the animation curves allow you to multiply the scale or the alpha of the health bars based on the distance. The X-Axis of the animation curves represents the distance, while Y-Axis represents either the scale- or the alpha-multiplier. If the distance is outside the specified keys, the value is equal to the closest key. I.e. if the leftmost key is at X = 5 and Y = 1.2, then a distance closer than 5m is still going to have a multiplier of 1.2.

3.1.6 Separator

This curve fades in or fades out the separators on top of the health bar foreground (only if **Use Separators** is enabled in the draw properties). This will only fade the separators, not the foreground or background.

Useful if there are reading problems from far away because of the amount of separators.

3.1.7 Transition Behaviour



- **Scale Transition**: When activated, scales the transition texture after the health transition, based on the transition background time defined in the profile. Otherwise, the transition texture is just rendered without moving.
- **Alpha at Time**: Adjusts the alpha value of the transition texture based on time. This allows you to create one or multiple flashes when losing health. If you do not want the transition texture to fade in or out, set the keys to have values of 1.

3.2 Code

The most important methods of the controller are:

- **Init()**: Initializes the controller. Should be called from one of your loading scripts. Upon initialization, all the necessary memory is created for storing all the information necessary for the shader later upon rendering.
- **AddHealthbar(ref HealthbarData)**: Adds a health bar to the controller, to be rendered the next frame
- **RemoveHealthbar(HealthbarData)**: Removes a health bar again from the controller. After the function is called, the health bar is not rendered anymore (the next frame)
- **UpdateHealthbar(HealthbarData)**: Updates the data in the controller with the new data. This is necessary, because the data has to be scheduled to be written to the GPU. Try to minimize the calls to the function as much as possible. The writing process is done using a job at the end of the frame.

Other methods which are likely less useful for the regular user:

- **SetMaterialPropertyBlockProperties()**: If you update properties of the controller at runtime that should affect the material data, you will have to call this function in order to see the effects
- **SetMaterialKeywords()**: If you want to change the draw behaviour at runtime (i. e. you want to change the shader keywords) you would need to call this method. However, note that you will need to handle all the possible shader variants in your project yourself in order for them to work in a build. That means you'll have to always include the shader in the project first with all its variants, and then strip the unnecessary ones later again. You can find a tutorial about shader variant stripping here: Shader Variant Stripping

- **UpdateProfileData()**: When you change profiles at runtime, make sure to call this function later on, so the changes are reflected on the GPU as well. This includes removing or adding them

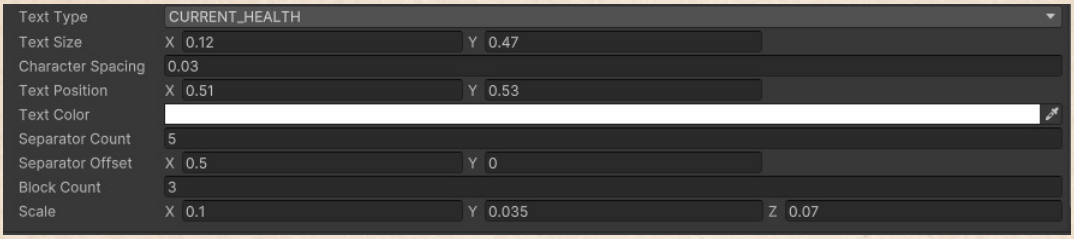
4 Health Bar Data and Profile

4.1 Health Bar Profile

- **Name**: Profile Name
- **Transition Time**: If transitions are enabled, this specifies how long they take (for the foreground texture). E.g. if the transition time is set to one, and if an entity has 50 health and gains 20 health, the bar will go from left to right and reach the point where 70 lies after one second.
- **Transition Background Time**: If transitions are used *and* a transition texture as well, this time is used for scaling and fading it. The total amount of time it takes for the transition is then the **Transition Time** plus the **Transition Background Time** if health is lost, and only the **Transition Time** otherwise.
- **Percentage Gradient**: Multiplies the foreground texture color by the current health percentage.

The properties in the profile for the foreground and background texture are the same:

- **Tint**: Additional Tint for the texture
- **Tiling**: Works in the same way as in a normal shader
- **Scroll Speed**: Allows you to scroll the texture coordinates / UVs over time. Can be used instead of a flip book.
- **Margins**: Adjusting the margin allows you to align the textures properly if they are not by default. You can also do some fun effects at runtime with them if you want



Back to the profile:

- **Text Type**: The way health numbers and percentages are shown on top of the health bars when rendering text is enabled (in the controller). Since everything is happening in shaders, the options are limited to displaying the current health (as integer), the current health percentage (as integer) or the current health as well as the maximum health (both casted to integers).
- **Text Size**: The size of each symbol / digit in the UV-Space of the rendered bar
- **Character Spacing**: The space between each symbol / digit
- **Text Color**: The coloring of the displayed numbers
- **Number of Separators**: Only has an effect if separators are also used in the controller. Defines the UV tiling of the separator texture in essence.
- **Number of Health Blocks**: Only has an effect if **Round Health to Blocks** is activated in the controller. This number basically represents the number of "blocks" the health bar has (i.e. the health bar acts as if the value internally was an integer rather than a floating point value)
- **Scale**: Scale of the health bar using this profile. Note that the scale has to be adjusted quite significantly if you change between drawing the health bars projected or like UI elements.

4.2 Health Bar Data

4.2.1 Data

- **float currentHealth**
- **float maxHealth**

- **int profileID**: The health bar will use the profile from the controller with the same index as specified in this field
- **Transform attachedObject**: The object the health bar is attached to. The render position is determined by the transform position and the offset (next field)
- **Vector3 offset**: Offset that is added to the transform position

4.2.2 Methods

The data can override some profile values using the override methods. The health bar will then use the values you provide instead. You can also clear the overrides again. The following properties can be changed in that way:

- **Foreground Tint**: **OverrideForegroundColor(Color)** overrides the profile foreground tint.
- **Background Tint**: **OverrideBackgroundColor(Color)** overrides the profile background tint
- **Scale**: **OverrideScale(Vector3)** overrides the profile scale

There are also Clear-methods for the overrides. **ClearOverrides()** clears all overrides previously made.

You can also create transitions. If you change the health without them, the change is instant instead. There are two functions: **CreateSimpleTransition** and **CreateCustomTransition**

CreateSimpleTransition creates a normal transition based on your parameters in the profile and the controller. It will linearly interpolate between a start and end health, calculating the required start and end times internally.

CreateCustomTransition instead requires you to calculate these times yourself. This can be used to create shorter or longer transitions, and also allows you to interrupt running transitions in a better way.

All methods in the health bar data require you to call the controller update health bar method again, in order for them to have any effect!

5 Contact

For any questions or suggestions, you can reach me anytime by the following email-adress:

blenderfan@gmx.at

There is also a discord server, which is usually the fastest way to reach me:

Parable Games - Discord

Alternatively, you can also find some social media links and contact information on my website:

https://parable-games.com

It is, for clarification, not required to credit my name when using my tools and packages. However, it is very much appreciated if you do!

6 Future Plans

Some additional features I plan to implement in the near future:

- Text-Support somehow
- Shader Optimization

Thank You!

Your purchase of **Gimme Instanced Healthbars** enables me to continue developing code and techniques for game-development in an independent way!