Texture Packer Plugin Tutorial

This is a tutorial on how to use the **TexturePacker** extension for Unity. There are two different versions of TexturePacker – The stand alone **TexturePacker**, and the **TexturePacker Unity Extension**. Both have a limited function **free** version in addition to the **full** version.

Basicly tutorial is splitted in 3 parts.

Introduction.

Why you should use texture atlases in your game:

- Reduced build size
- Reduced RAM consumption
- Increased performance

Why use TexturePacker?

TexturePacker will allow you to easily and quickly create texture atlases for your game.

- Rotation
- Trimming / cropping
- Scaling
- Smoothing
- Note: Some features are not available in the free version

Learn More here:

http://www.codeandweb.com/texturepacker

Why use the TexturePacker Unity Extension?

Unity GUI

- Draw textures from the atlas
- Supports TexturePacker features (rotation. crop, trim)
- GUI animations

3D object

- Auto generation of UV coordinates for Meshes

- Skinned Meshes support
- Rotated images for optimal space
- Play animations from atlas
- Preview in editor mode
- Full editor integration, custom windows and inspectors
- Solutions for sprite animation with batching and trim/crop support
- Note: Some features are not available in the free version

Using with 3D object's

TexturePacker is available for download here:

http://www.codeandweb.com/texturepacker/download

The TexturePacker Unity extension is available here:
Free
https://www.assetstore.unity3d.com/#/content/8905
Paid
https://www.assetstore.unity3d.com/#/content/8456

Creating the Texture Atlas:

For the first tutorial, we'll pack textures for our scene (Assets are available in the TexturePacker Extension package under **Assets/Extensions/TexturePacker/Tutorial/Art/AtlasSource**).

Open TexturePacker, and simply drag the textures inside. Now let's configure our atlas.

Output Options

Data Format: Unity3D

Data File: YourProject/Assets/Resources/TutorialAtlas_data.txt Texture File: YourProject/Assets/Resources/TutorialAtlas.png

Texture Format: png

Geometry Options

Check the “Force Squared” toggle

Layout Options

Algorithm: MaxRect (Full version only) or Basic (available in the free version)

Allow Rotation: checked (Full version only) unchecked (free version)

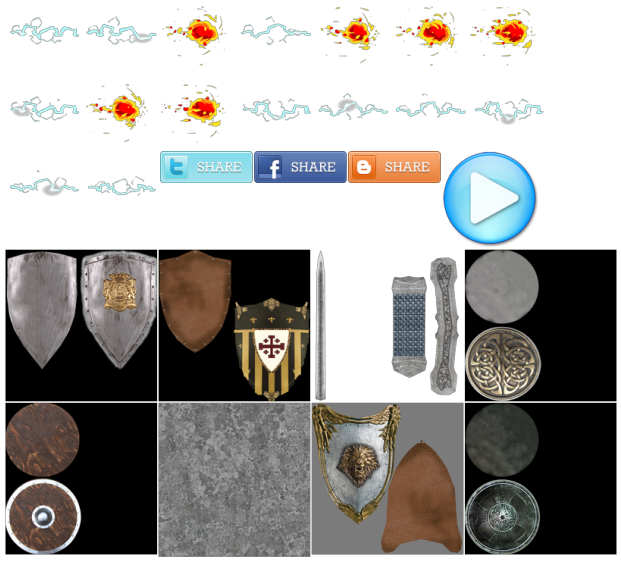
Trim mode: none (Trim mode is only supported for GUI rendering)

Note

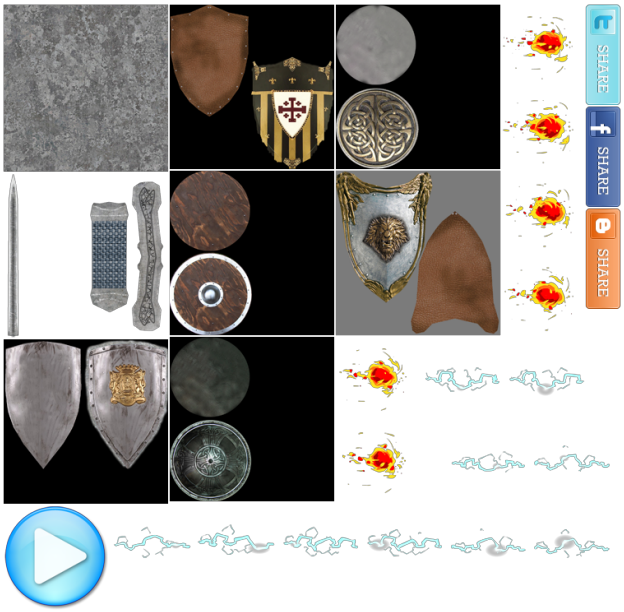
I has described most optimal options, but you can experiment with TexturePacker option to get the best result for your atlas.

Next, press the Publish button. Your altas should look like this:

Free version:



Paid version



Atlas is ready, so we need material for it.

In Resources Folder **Create->Material**. Set material name as **TutorialAtlasMaterial**, attach **TutorialAtlas.png** as main material texture and change shader to **Transparent->Diffuse**.

***Note:** The material will be created automatically by Asset Processor editor script. But you can create any number of other materials with different shader for atlas.

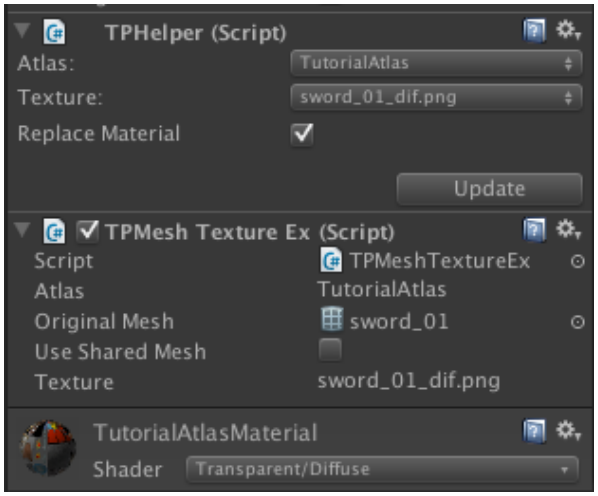
Using the texture atlas with a model:

Using the texture atlas with a model:

(**Assets/Extensions/TexturePacker/Tutorial/Art/Models/Sword/sword_01**)

It already has a texture and a material assigned. Replace this with a texture from the atlas:

1. Select the sword_01 gameobject (the one with the MeshRenderer component attached).
2. Attach TPMeshTextureEx component (TPHelper component will be added automatically)
3. Choose Atlas and Texture in TPHelper menu
    - Atlas: TutorialAtlas
    - Texture: sword_01_dif



The sword is now using the texture from the atlas.

***Note:** Do not attach TPMeshTexture component, it will cause component lost if you replace TexturePacker.dll. Use it only for extending.

Using the atlas with skinned model

The usage is similar to a simple model.

1. Select the sword_01 gameobject (the one with the SkinnedMeshRenderer component attached).
2. Attach TPSkinnedMeshTextureEx component (TPHelper component will be added automaticly)
3. Choose Atlas and Texture in TPHelper menu

Play an animation from the atlas:

This approach is only useful if you want play animation on YOUR mesh (not simple plane). Trim / Crop is not supported with this approach. If you just want to play animation on plane from atlas, please follow Creating Sprite Animation steps. It has additional features implemented as:

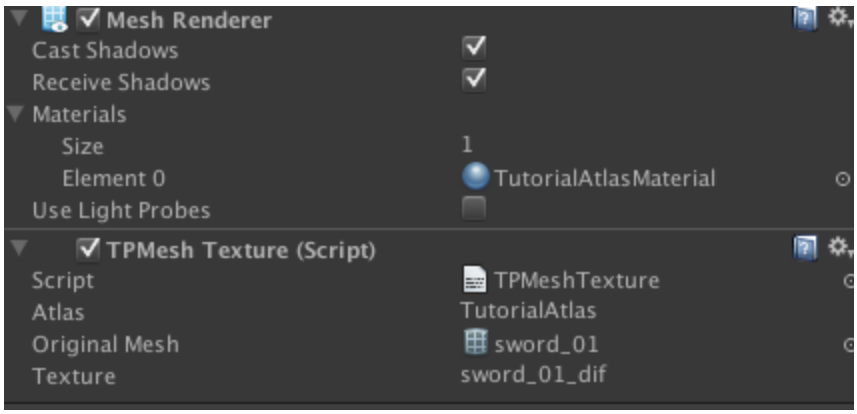
- Trim / Crop support
- Animation can contain frames from different atlases.

For the next tutorial, we will play an animation from the texture atlas on a cube:

1. Create a cube in the scene by selecting **GameObhect->Create Other->Cube**
2. Change the material on the cube to **TutorialAtlasMaterial**
3. Attach a TPMeshAnimation component
4. Configure the TPMeshAnimation component options as follows:
    - Atlas: TutorialAtlas
    - Frames: add 6 frames fireball_0001, fireball_0002, fireball_0003,

fireball_0004, fireball_0005, fireball_0006

The component should look like this:

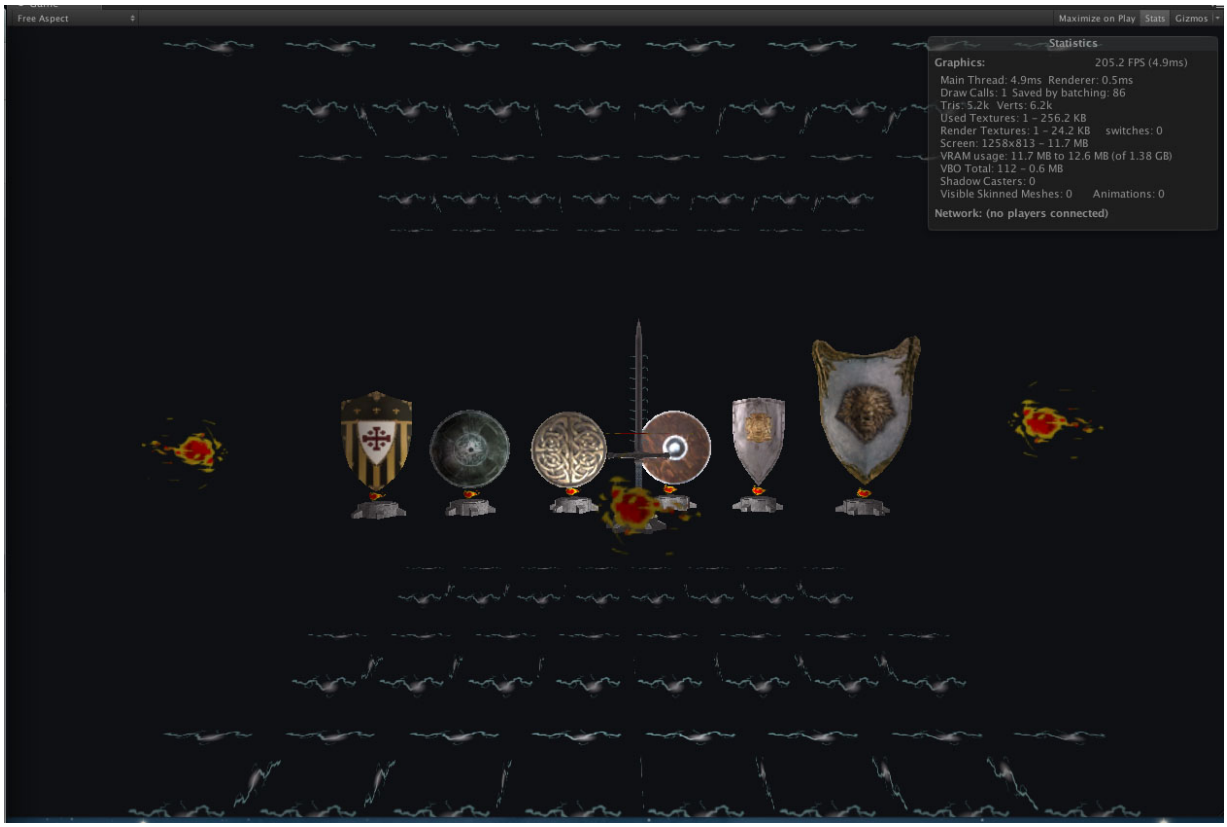


Now press play and you will see fireball animation playing on the cube. You can try other meshes and experiment with different TPMeshAnimation options.

You always can automate this process using extensions, there are a few examples of how do this inside the package. See the MyFireAnimation, MyTeslaAnimation and MyMeshTexture classes.

The complete scene can be found at
Assets/Extensions/TexturePacker/Tutorial/Scenes/AtlasesAnim

For this scene was used only one atlas, that's why whole this scene with almost 100 objects and 50 animations has only 1 Draw Call. 1 Texture, and 1MB of memory



Optimization.

Let's look at the optimization of weight in this example.

We have only one texture for the whole stage, which is already good benefit, but we can squeeze even more.

size of our texture is 754*754, if we choose RGBA 32 compression it will weight 2.2 MB

But if we will set PVRTC compression and stretch texture to the 1024 * 1024 (compression work only with POV2 textures) It will weight only 1MB. And if you do not mind against some quality lost, Unity3d can scale it to 512 *512 so it weight becomes only 256KB. Try it.

By the way you can use POV2 texturepacker options to always see how much texture you can add to your atlas.

Animation in Texture Packer 2.0

2.0 version contains more advanced function for sprite render and sprite animation. Main features:

- Trim / Crop, Rotation and Scale support
- Full Editor integration
- You can use frames for one animation from different atlases
- Animation preview in editor
- Draw calls batching

Note: This “on mesh” animation, it doesn't use Unity GUI system. To play animation using Unity GUI, please read about plugin GUI classes.

Creating Sprite Animation

You can use your or tutorial atlas to create 2D sprite animation.

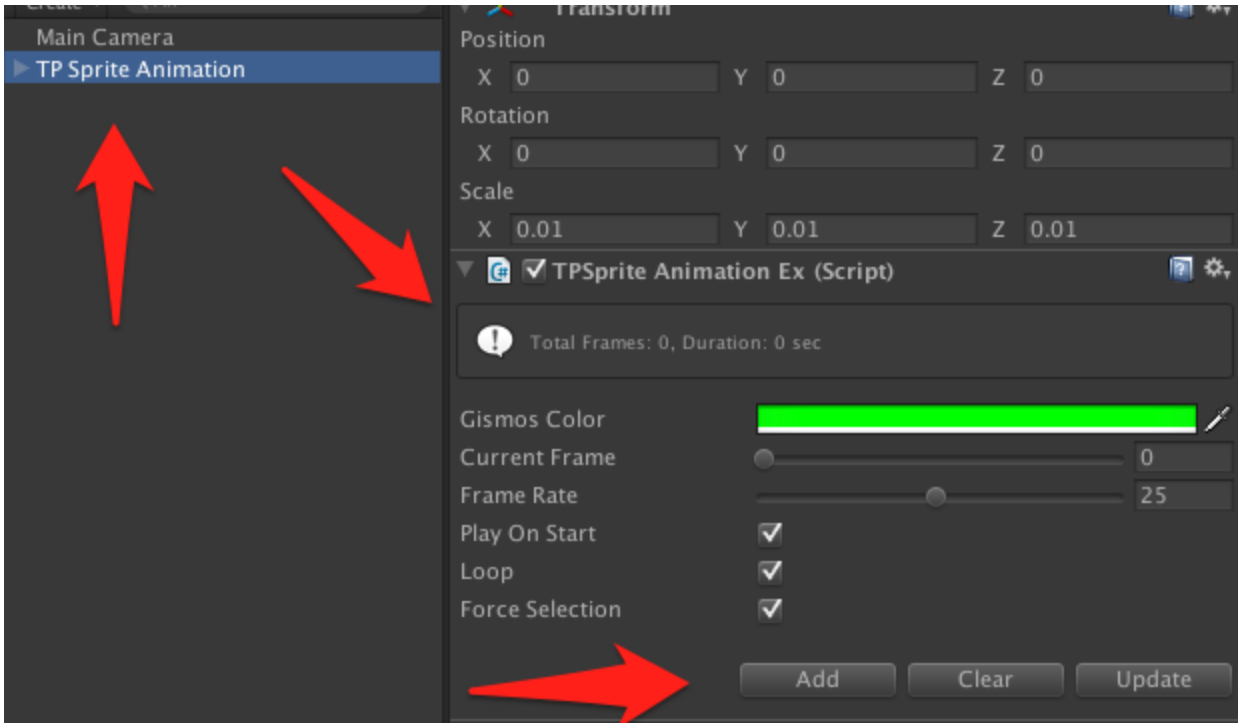
Go to the

Game Object -> Create Other -> Texture Packer -> Sprite Animation

This will create new TP Sprite Animation object.

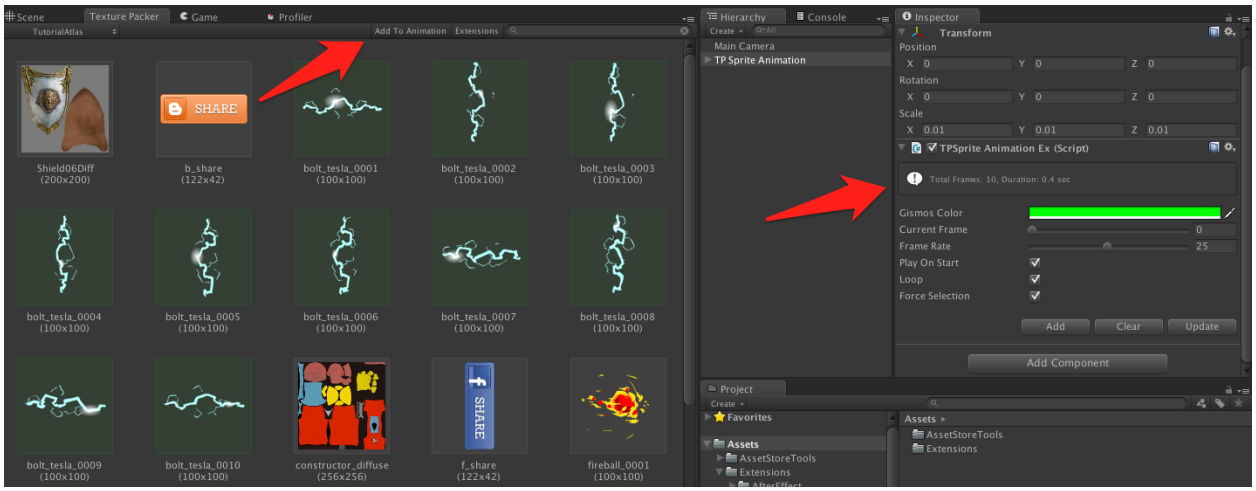
Now you can add some frame to the animation. Press **Add** button. **Add** button will call Texture Packer custom window or bring it up, if you already have it.

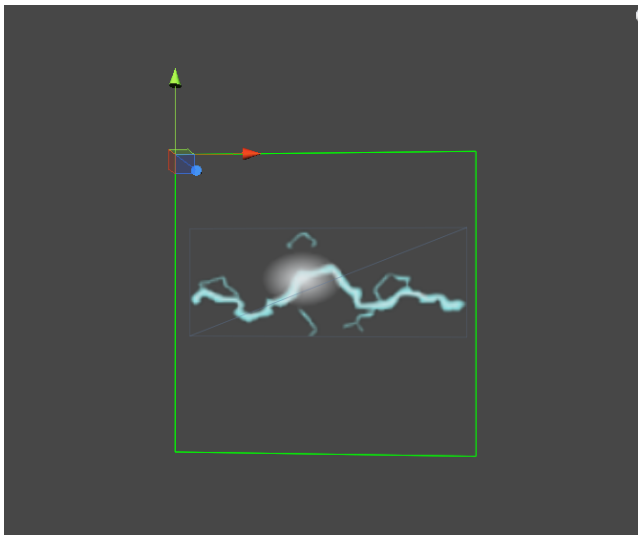
Note: For opening Texture Packer custom window go to the **Windows-> Texture Packer.**



In Texture Packer window, just select frames for your animation. You can use Ctrl/Cmd and Shift button to easily select frames you want for this animation. After frames are selected press “**Add to animation**” button.

Note: You can add more frames from different atlas (jsut switch) atlas in Texture Packer window and add another frames. And of course you can add the same frames multiple times to the animation if you want to.





That is it. That is all you have to do to set up the sprite animation from atlas.

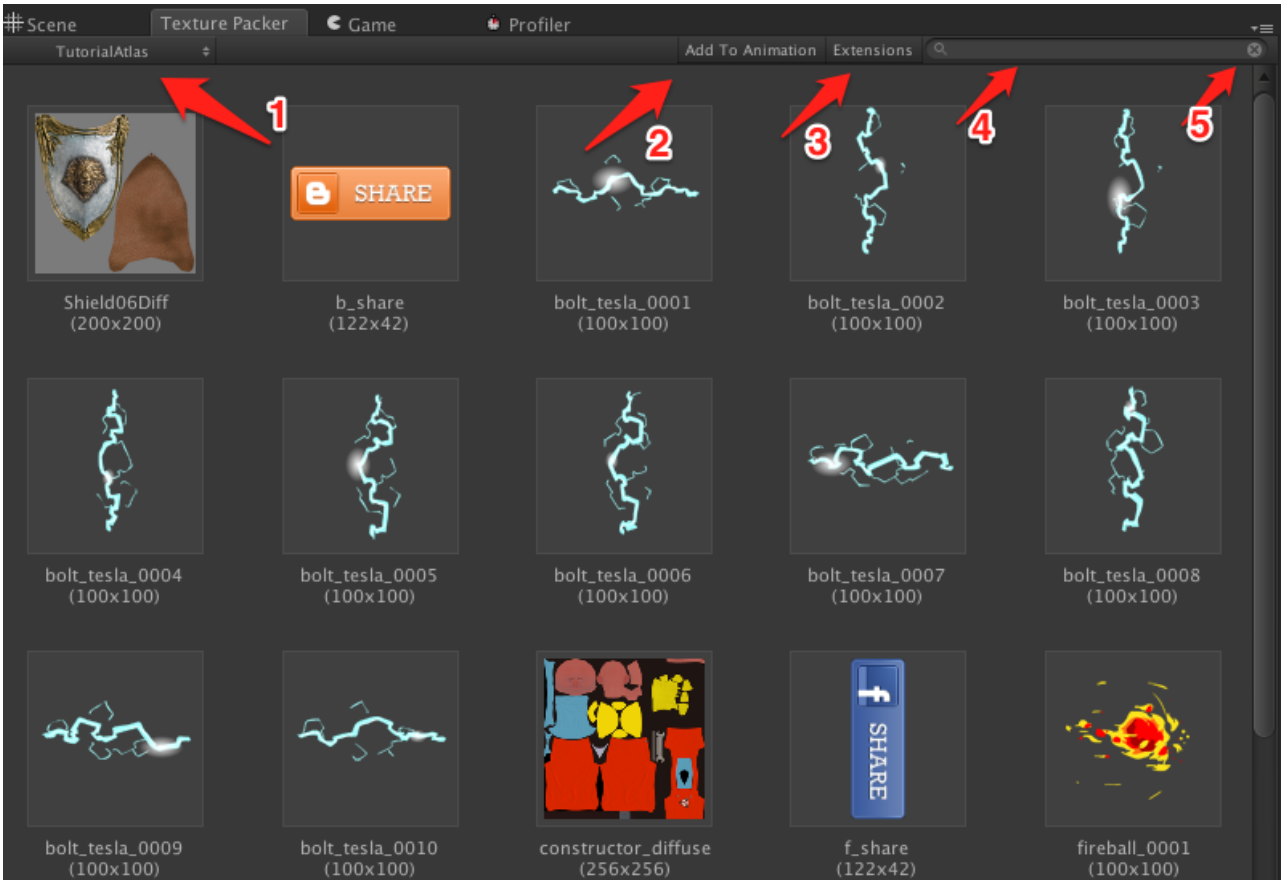
Creating Simple Sprite

That is useful if you want atlas texture to be rendered on simple plane Mesh, or if you want trim support. Go to

Game Object -> Create Other -> Texture Packer -> Sprite Texture

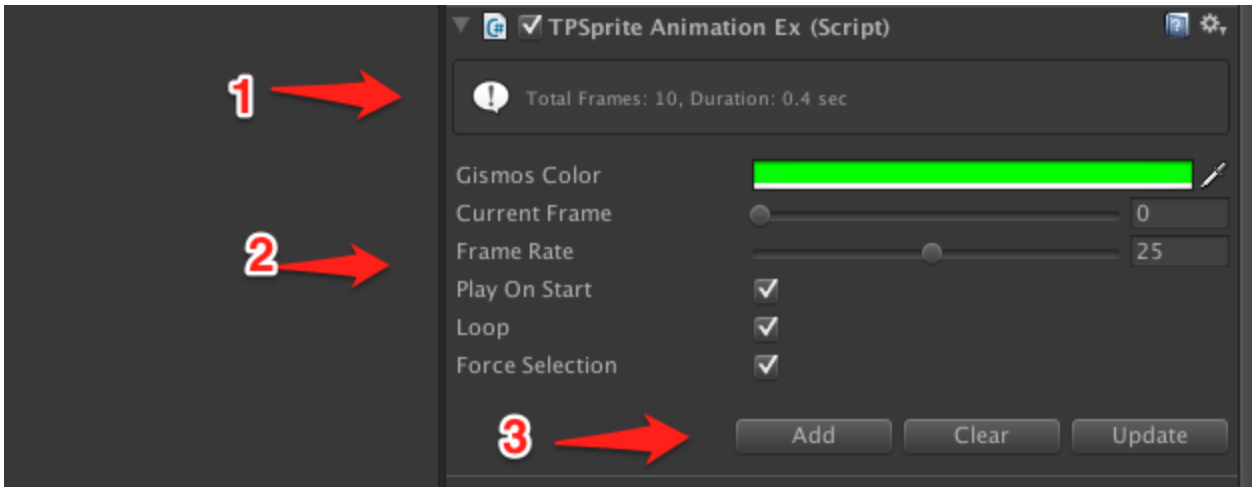
And follow the same steps as form Creating Sprite Animation.

Texture Packer Window



1. Atlas choose drop down. Use it to switch between atlases in your project.
2. Adds selected frames to the selected sprite animation.
3. Toggle between display or not display frames extension.
4. Search field. Start typing in this search field to filter frames result
5. Cancel search. Press this button to disable filter and clear search field.

Sprite Animation Inspector



1) Info Box. It will display useful information about the animation.
2) Animation settings:
    a) **Gizmos Color.** Color of the box around animation.
    b) **Current Frame**. Current frame of animation which is displayed in editor. When you press play animation will start playing from this frame.
    c) **Frame Rate**. Frame rate of selected animation.
    d) **Play On Start.** If this option is active, animation will start playing when animation game object will receive Unity Start event.
    e) **Loop**. If the options is active, animation will play a round.
    f) **Force Selection**. When you selecting animation by clicking on it in the editor scene, Game object with TPSpriteAnimation component will be selected instead animation sprite. To disable this behaviour uncheck this select box.
3) Control buttons:
    a) **Add**. Button will open Texture Packer window, or bring it up if already opened.
    b) **Clear**. Clear all animation frames.
    c) **Update**. Update animation view. Can be useful if something goes wrong, and view wasn't updated automatically.

Optimization tips

Last Optimization section was about RAM and Texture Atlases size saving. Let's talk about CPU. One of the main features of this extension is Draw Calls saving. This is possible thanks Unity Dynamic batching.

When you creating different animation they use the same material with make dynamic batching possible. To use it with the full power you should understand how the dynamic batching work, and render part of this extension work.

When the sprite animation is running it applying new scale and local position to the sprite every frame. It means that TP Sprite is always **non-uniform** object. With means it can be batched with other **non-uniform** objects, but if you will use the same material on object with uniform scale it will cause 1 additional draw call.

Using frames for different atlas is a great feature. But you should understand that animation sprite will switch materials while playing in this case. Every additional material on scene this is additional draw call.

Learn more about batching.

Using with GUI

I will use the same atlas as for our 3D sacene, but you can experement and create new one, using source gui textures which is located under Assets/Extensions/TexturePacker/Tutorial/Art/AtlasSource.

Difference between creating texture for 3D object and for GUI is, that for GUI atlases you can use Trim / Crop TexturePackre methods to save even more space (this option is available in paid version of TexturePackre)

Now we need to create our GUI drawing class. And assign it to gameobject.

Here is a little overview of function's that' is useful for work with the GUI part of TexturePacker extension.

To get Atlas, use:

```
TPackManager.getAtlas(MyAtlasName);
```

To get Texture, use:

```
TPAtlas atlas = TPackManager.getAtlas(MyAtlasName);
TPAtlasTexture texture =  atlas.getTexture("myTextureName");
```

or

```
TPAtlasTexture texture =
TPackManager.getAtlas(MyAtlasName).getPngTexture("myTextureNam
e");
```

To draw Texture, use:

```
void OnGUI() {
     TPackManager.draw(Rect rect, string atlas, string
textureName);}
```

Here is how our **`AtlasGUIExample`** class will look like

```
public class AtlasGUIExample : MonoBehaviour {

      private TPAtlasTexture play;

      private Texture2D unityTexture;


      void Awake() {
            play =
TPackManager.getAtlas(Atlases.EXAMPLE).getPngTexture("play");

            //Warning to Get Texture2D from atlas, should remain the same size
which was generated,
            //that's why we using another atlas here

            unityTexture =
TPackManager.getAtlas(Atlases.EXAMPLE2).getUnityTexture("play");
      }



      void OnGUI() {
            play.draw(new Rect(0, 0, play.width * 0.5f, play.height * 0.5f));

            play.draw(new Rect(0, 75, play.width, play.height));

            GUI.DrawTexture(new Rect(0, 225, unityTexture.width,
unityTexture.height ), unityTexture);



            TPackManager.draw(new Rect(200, 0, 122, 42), Atlases.EXAMPLE,
"f_share.png");
            TPackManager.getAtlas(Atlases.EXAMPLE).draw(new Rect(200, 100,
122, 42), "t_share.png");


      }
```

Proceed to the animation test (GUI animation available only in paid version)

We can use same atlas as for out 3D stage.

Let's Create "AtlasGUIAnimationExample" class and attach it to any gameobject on the stage. Here is how it will look like.

```
using UnityEngine;
using System.Collections;

public class AtlasGUIAnimationExample : MonoBehaviour {


	private TPGUIAnimation anim;
	private TPGUIAnimation Scaledanim;


	void Awake() {
		anim	=  TPGUIAnimation.Create();
		Scaledanim  =  TPGUIAnimation.Create();

		for(int i = 1; i < 7; i++) {
			TPAtlasTexture frame =
TPackManager.getAtlas(Atlases.EXAMPLE).getPngTexture("fireball_000" +
i.ToString());
			anim.addFrame(frame);
			Scaledanim.addFrame(frame);
		}

		anim.pos = new Vector2(200, 200);
		anim.loop = true;
		anim.fps = 25;
		anim.Play();
```

```
            Scaledanim.pos = new Vector2(300, 200);
            Scaledanim.loop = true;
            Scaledanim.fps = 25;
            Scaledanim.scale = 0.5f;
            Scaledanim.Play();



      }

      void OnGUI() {
            anim.draw();

            Scaledanim.draw();
      }
}
```

In conclusion, here is a few advantages to using this.

- Fast and easy atlas generation
- Easy to remove or add texture from atlas
- Built in animation solution
- reduced build size
- reduced RAM consumption
- increased performance