3dSprites

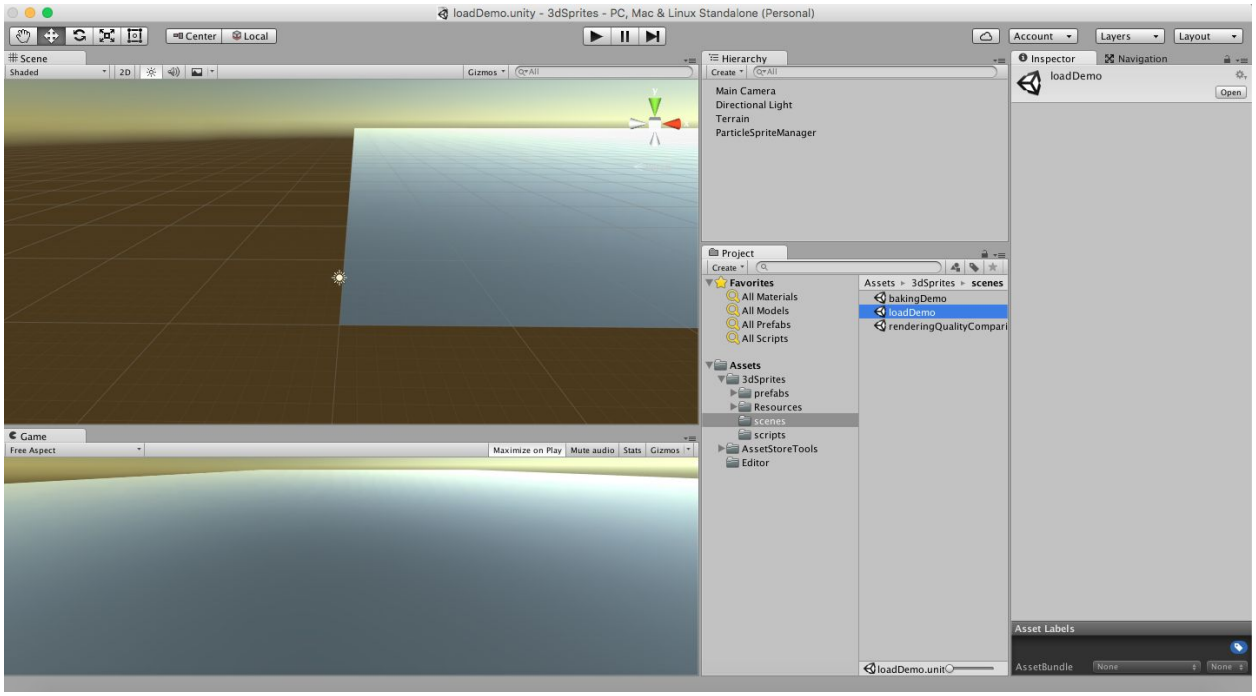
v1.0

Email: chanfort48@gmail.com

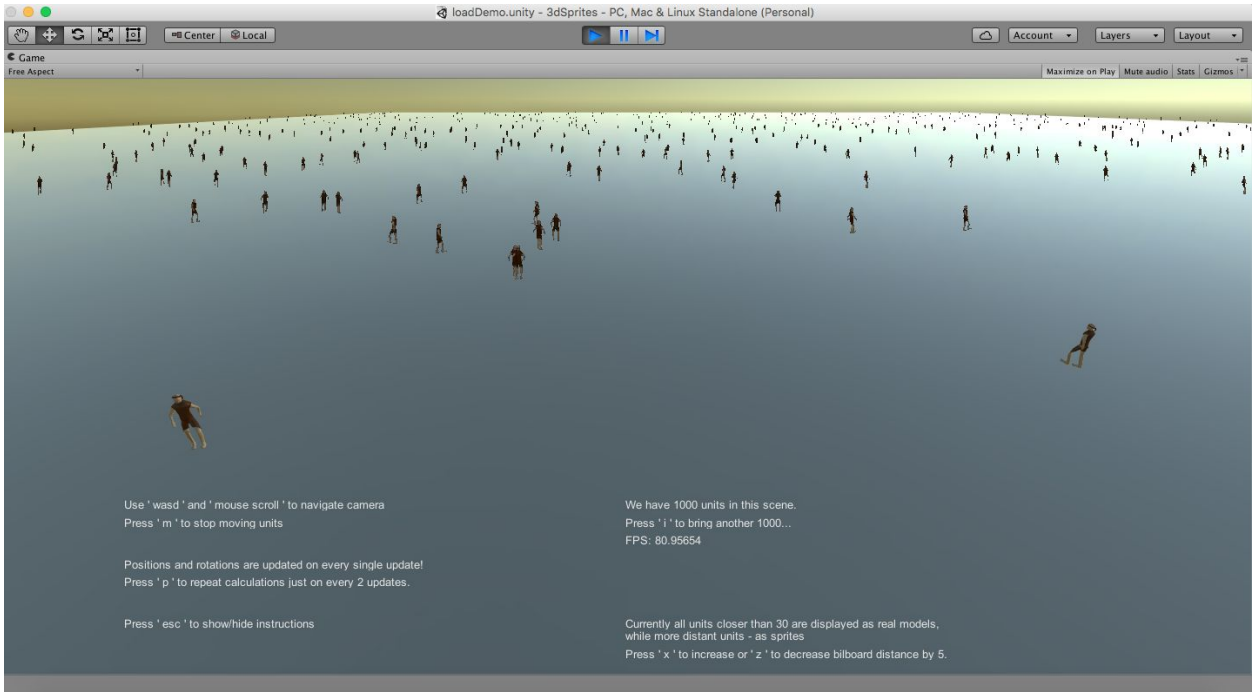
3dSprites allows you to bring thousands of animated 3d objects into the game. Only up to several hundreds of animated objects can be rendered using meshes in the runtime. So 3dSprites comes here into help. It uses Unity built in Particle System to render 3d objects, visible from different angles as sprites. Multiple thousands of sprites can be set onto the scene. To bring more realistic approach, nearby objects are rendered as regular meshes, while more distant ones - as sprites. User can bake any 3d object into 3dSprites.

First open

To see what 3dSprites really are, the best example scene to start would be “loadDemo”. The scene can be found in Assets/3dSprites/snenes. Double click on it and you should be able to see the first view:

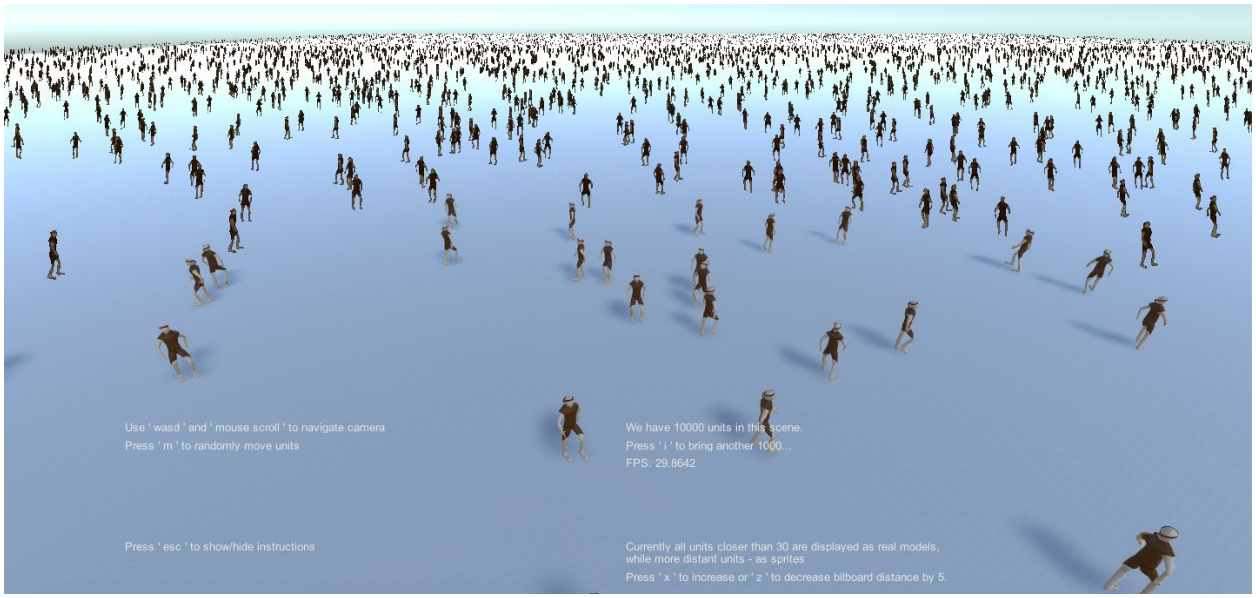


Click run and you will see the first 1000 units loaded and moving around:



Each unit is playing its individual animation. There are multiple things to test here.

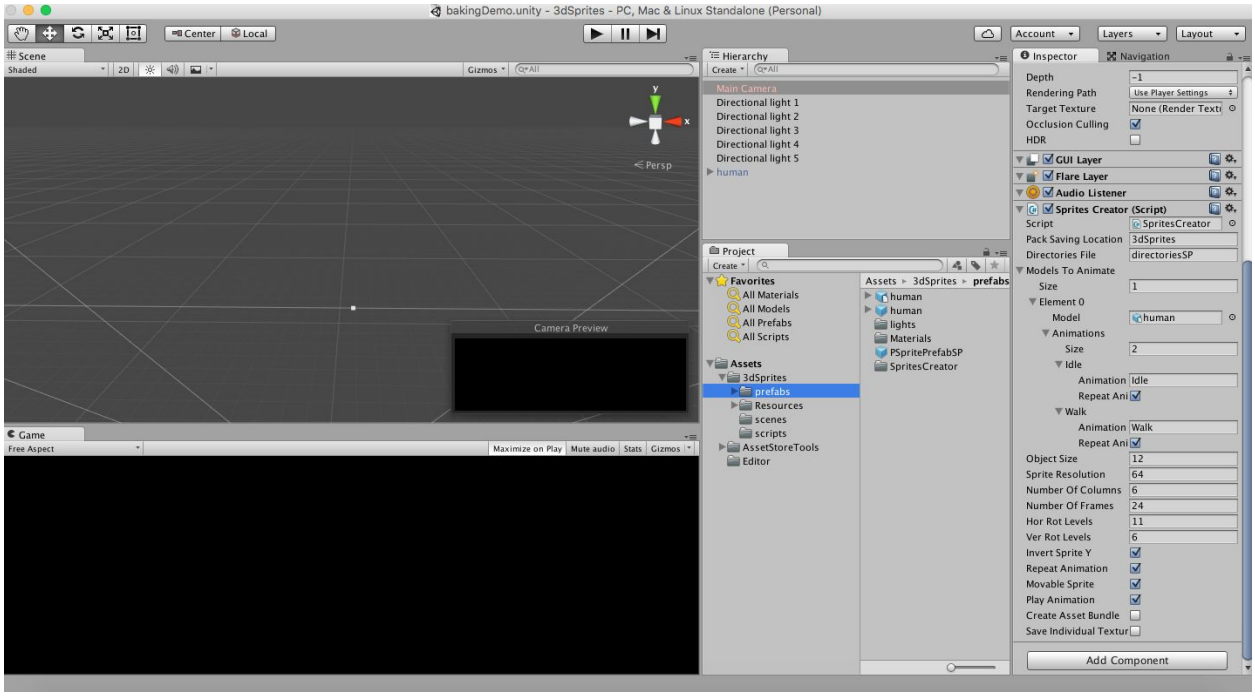
1. Increase the number of units in the scene. As there are “just” 1000 units visible, you can try to increase it to 2000 by pressing “i” in the keyboard. At this point there will appear 1000 additional units. By pressing “i” again you can keep rising this number up to 3000, 4000… and so on. There is also FPS counter visible, which tells at what FPS game is running.
2. Increase or reduce billboard distance. At the beginning there is set that units would be visible as meshes at distances lower than 30 from the camera. Units, which are at higher distances, are rendered as sprites. But this distance can be changed on the play mode by pressing ‘x’ or ‘z’ keys. Each time the key is pressed, original distance is increased or decreased by 5. The larger is this billboarding distance, the more objects are displayed as meshes. However, it comes at a cost of reduced FPS count. Play a bit to find out what is the most wanted billboarding distance.
3. Camera can be easily moved through the scene by using ‘wasd’ keys or bringing cursor close to the game screen edges. Mouse wheel can be used to zoom in and out. Move camera around the scene to see how everything changes and how sprites are adjusting to the camera.
4. Moving units is one of the most expensive operations, as each unit is linked with its individual GameObject. To stop moving units press ‘m’ and see how the scene looks. FPS counts should increase very significantly when units won’t be moving. You can bring up to 10000 units with around 30 FPS this way. You can bring units back to their movement by pressing ‘m’ again.



5. If units are not walking, the scene can be very boring. But many thousands GameObjects can't be moved at acceptable FPS rates. To find the balance between "instant freeze" and every update movement we use partial positions update feature. To use it enter regular mode, when everything moves in the scene. Press 'p' and and movements will be updated not on every single update, but every 2 updates. Pressing 'p' again will bring re-calculations to run on every 3, 4.. updates. At larger numbers you will start seeing 'lag' appearing in units movement. However, larger this number is, more FPS count you can win. Pressing 'o' you can decrease this number and bring more frequent calculations again. By increasing and decreasing update frequency you can explore which partial update frequency works the best for you.
6. There are some interactive commands and numbers written in play mode. You can completely hide them by pressing 'esc'.

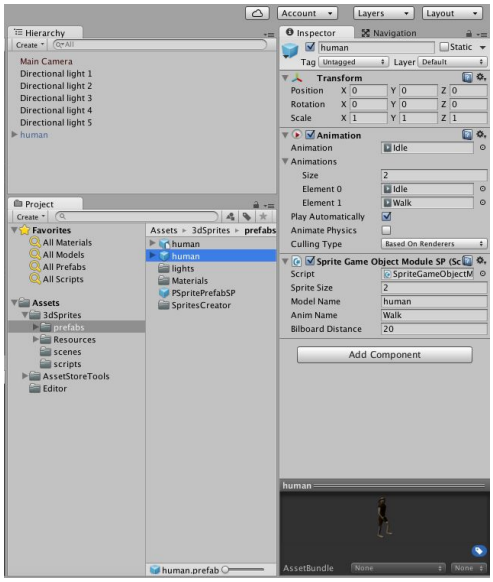
Creating 3dSprites

There would be no point to use such system as 3dSprites if there wouldn't be possible to set up your own models as 3dSprites! When you ready to start looking how to bring your model into 3dSprites, double click on "bakingDemo" scene. Here is a working example for our "human" model, which we rendered in "loadDemo" scene. Lets look how the scene is created.



Firstly there is Main Camera, which has “SpritesCreator” component attached. This component is the main script, which control camera when baking sprites is ongoing.

The next step is to find out how “human” prefab is made. The model is a simple fbx model imported from Blender. It has 2 animations “Idle” and “Walk” imported as Legacy animations. The object was dragged and dropped on the scene and then prefab was saved. Prefab also has a very important “SpriteGameObjectModuleSP” script attached, which links GameObject with 3dSprites system.



The next step is to set up SpritesCreator script, which in our case is attached to Camera. It has “ModelsToAnimate” list where our “human” GameObject (in the scene, not prefab) is assigned. There is also need to put exact animation names, which we want to bake. In this case we put “Idle” and “Walk” under “Animations” list.

Other more important SpritesCreator parameters are:
SpriteResolution - the resolution of the single sprite (not the animation sheet!).
NumberOfColumns - the number of columns in sprite animation sheet.
NumberOfFrames - the number of frames to be recorded through the entire animation.
Combination of NumberOfColumns and NumberOfFrames defines final sprite sheet resolution and number of rows is not needed to specify here.
HorRotLevels - number of levels in horizontal rotation.
VerRotLevels - number of levels in vertical rotation.
HorRotLevels and VerRotLevels defines a grid of points, from which snapshots of animated objects will be taken. As camera rotates around the object horizontally and vertically, it covers how the real object looks like viewing from different angles. Later these baked snapshots are used in “loadDemo” to display the right sprite based on where camera and the object really are in the scene. The larger number of HorRotLevels and VerRotLevels allows to cover larger sample of viewing angles and provides smoother transitions when rotating sprites. However, it comes at the cost that it creates large number of images. So developer can play with HorRotLevels and VerRotLevels to get the best balance between the quality and performance.

Most of other options in SpriteCreator at the moment are experimental.

Once you are happy with these options click play to enter the Play mode and the baking begins. Animation sheets are being saved as PNG images in Resources/3dSprites directory with subdirectories, corresponding to model name and animation name. If you are baking your own model sprites, it may need to look carefully what components your model has and always compare with how “human” model is set in the “bakingDemo” scene. If there are some components not included (i.e. no Animation component or animation names are different from these ones specified in SpritesCreator lists, baking will end up in errors).

Comparing differences between real models and 3dSprites

Once sprites are created, it can be not trivial to justify how well do they compare with real mesh models and what can be done to get best possible results for your project. To help with this question, we have the third scene, named as “renderingQualityComparison”. Double click it to open the scene and press play to enter Play mode.

The scene has camera, which can rotate around 0,0,0 point and two units brought in the front of camera to bring side by side comparison. Once you use “wasd” and “mouse scroll” to navigate camera, you can see easy side by side comparison between the object rendered as a mesh in the right and as a sprite in the left. The left object has very low billboarding distance and always

appears as a sprite, while the right object has very large billboard distance and always appears as a mesh. When camera is rotating, objects are re-positioned in space to create an illusion that both objects rotates around their own axis, bringing this nice side by side comparison possible. This comparison scene allow to find a best balance easier when working with number of rotational levels, sprite resolutions and considering memory usage in your project (due to loaded images when displaying sprites).