

Things to know when creating an iOS application

This section contains some important things to know when you create an application for the iPhone or the iPad. Please read it carefully and keep it in mind when programming your application.

Building for various devices

Multimedia Fusion 2 Developer allows you to build applications that can run on an iPhone, iPod Touch, or iPad. As a default, the properties of the XCode project are set to iPhone / iPad, meaning that your application will work on any of the available Apple devices at that date (March 2011).

If you want to restrict your application to iPod only, in XCode, right click on the name of your project in the project window (topmost item) and choose "Get info". This will display the properties of the project. Scroll down a few pages until you find the line called "Targeted Device Family". Just select iPod in the combo box.

If you want your application to only work on an iPad, just select iPad in the combo box.

Memory considerations

iOS devices are portable devices. As such, the amount of useable memory (RAM, not to be confused with the amount of Flash memory where the files are stored) is limited. On a 2gen device, like the original iPhone and the iPod 2nd Gen, the size of RAM is 128 Mbytes. On 3rd Gen devices and over, the amount of RAM is 256 Mbytes.

This might seem a lot, but if you know that iOS 4 consumes at least 80 Mbytes to function, you quickly realize that the amount of working memory on a 2nd Gen device can be evaluated to around 40 Mbytes.

The code of an iOS application must be copied into RAM before being launched by iOS : this takes 2 Mbytes out of your previous RAM.

The conclusion is that, if you want your application to work on a 2nd Gen device, you have to be very careful about the size of your graphics and sounds. The next chapter of this page will give you tips to reduce the memory inprint of your graphics.

If your game cannot run on such a device, then you should prevent it to be distributed in the app store for a 2nd Gen device. If you do so, you also save some memory by setting the build architecture of the application in the project properties to Arm7 only, avoiding the duplication of the compiled code.

How to find out exactly how much memory your application takes?

The amount of memory used by an application is shown in the debugger window (top-left of the screen) when you launch your application in Multimedia Fusion 2 Developer. The memory displayed is evaluated for a PC application, and does not reflect the amount of RAM really taken by your application under iOS.

To find out exactly how much memory your application needs on the device, in XCode, launch your application by choosing the menu option "Run / Run with performance tools / Allocations".

XCode will automatically open a window that keeps in real time track of all the memory allocations of your application. You will find the total amount of RAM used in the first line of the grid.

Be sure to check every frame of your application, as the amount of memory used depends on the amount of graphics and sounds used in each frame.

Also check that the number of allocations (not the temporary allocations, which keeps growing normally) does not increase as your game runs (and might indicate undestroyed objects).

How to save memory for your graphics?

Alpha-channels

Alpha-channels look awesome and give a visually pleasing effect but they will slow down your game if you overuse them, just as they do in the PC runtime (just more).

An image with an alpha-channel takes 33% more memory and space than an image without and it takes more power and time to render them.

Workaround:

Does your graphic really need the alpha-channel? Does the background underneath it ever change? If not you could probably "bake" the alpha-channel effect into the image itself so that the shadow is just part of the picture.

Huge pictures

Many are often tempted to import really large pictures into the game where it will only show very little of that picture at one time. A good advice is to break it up into smaller parts so that fewer of them are shown at one time.

Remember that any image you have in your game has to fit inside their own texture.

In a sample game there was a 960x480 sized background image on a frame. This picture alone had to fit into a 1024x512 texture taking up a huge amount of memory. What was even worse was that it had an alpha-channel even though no pixel in it was transparent.

Second, the same frame had two different huge animations sized 957x158 (each of them contained 17 frames). This already takes up $2 \cdot 17 \cdot 1024 \cdot 256 \cdot 3 = 26738688$ bytes = 25.5 MB of RAM which is way above what the iOS likes. That was even without the rest of the graphics and the huge background image we talked about before.

Workaround:

Most graphics don't need 100% crispness and quality. It will only take 1/4th of the previous memory usage if you cut your image size in half and then at runtime scale it to 2.0. It will also run much faster as it utilizes the graphics card RAM cache much better.

Tip:

Crop your images, remove any useless empty space around them. Some developers tend to use a single frame size for an entire animation with sometimes huge transparent parts in some images. It's probably better at design time, but at runtime it results in a loss of time and space.

Tip:

Try to make your graphics of a size close to something of a power-of-two. It will waste less precious memory.

A power of two size is one of the following: 8,16,32,64,128,256,512,1024 Images less than 8x8 pixels will be stored in a 8x8 texture as they cannot be smaller than that.

You can easily use different sizes for your width and height, they don't have to be equal.

Tip:

Be sure to set the "Image compression" property to "Color reduction" either globally for all objects in the App iOS properties or per object - as often as you can.

In many cases it will make it consume half the memory it did before.

Take for example the big background image before:

- Before it wasted 2 MB of texture memory. - With color compression and no alpha channel

it will only take 1 MB. And if it was cropped to only fill out the resolution of the iPhone device it would only take 30 KB.

The animations mentioned before that took 25 MB of RAM, can with color compression and half the resolution only take: $2 \times 17 \times 512 \times 128 \times 2 = 4456448$ bytes = 4.25 MB!

That is a huge difference.

Single color graphic

This is kinda an extension to the previous pitfall:

We have often seen in the sample games we have debugged some big actives (typical white) that are meant to cover the entire screen and fade to transparent. People need to realize that even though they are pure white (or whatever color) they are still stored in memory uncompressed at runtime and waste just as much memory as if it was a pretty picture.

If you need those fade-effects you can again scale a small active to fit the screen at runtime and gradually alter its semi-transparency.

Transitions on big graphics You should in general avoid using transitions on objects larger than the device screen resolution even though it works well enough. Using a transition on such an object will consume a lot of extra memory during the transition and can be slow.

You have to try it at runtime to see if it gives you a noticeable performance drop. Just remember that even though it doesn't hurt the FPS that much, it can still drain the battery much more than you know.

Speed considerations

Overuse of "text" counters/score and strings Though it may not be a big of a performance issue, it can hurt your game's performance a little bit if you use too many objects that draw text.

If the text doesn't change it should not be a performance problem, but if you update your string/score/counters often (as counters/scores often do) you will get a performance problem.

Workaround:

- Only use text whenever you know it will not change often (like every frame).
- Use bitmap counters/scores instead of text counters/scores. There you can also much better control the visual style of your objects and there is nearly no performance penalty.

Too many transitions at one time

Each object that needs to have a transition effect on it will consume more memory than usual and will use some special buffers on the device that aren't unlimited. Once you reach a certain limit you will begin getting errors.

Also if you have many transitions going on at one time it will slow down performance.

Workaround:

- Instead of using transitions try using animations or other effects to give the same effect.
- Instead of using the "Fade" transition, manually setting the semi transparency of your objects will be much much faster and will not consume any more memory.

Unnecessary Ink-effects

We have seen other games almost ready for release where INK effects are unnecessarily used very often.

Having a big background covering the screen and setting it to either transparent or ADD so that it will be added to some gradient behind it, will only make your game slower and

consume more power. Instead, make your graphics as they should look at runtime and then use them like that.

INK-effects are only meant for places where you cannot modify the original image to look like it should at edit time.

For example if you want a grayscale background from a picture, you make it grayscale before you import it into MMF2. You simply don't put the monochrome INK-effect on it and forget about it