Optimize Your Display List

One of the biggest bottlenecks when it comes to performance is the rendering of the display list. In comparison, your ActionScript code will outperform the rendering of your visual assets. So even if you have taken every step in order to make sure that your code is optimized to be as fast as possible, your visual assets may still be slowing your application down. There are several things to watch out for when creating your visual assets that will help you to make sure that your application will perform as fast as possible.

One tip is to make sure that you keep the display list as shallow as possible. Try not to have too many DisplayObject with many nested clips. When a touch event occurs on the screen, the display list is traversed to find all the elements that are underneath the user's finger.

The deeper the display list, the longer the traversal will take. Keeping the list shallow will return those objects faster.

Another tip is to reduce the amount of items that overlap each other. The more items that are overlapping, the more compositing has to occur. In addition, try to reduce the amount of alpha (transparency) in your images. The more alpha there is in your image, the more the renderer has to composite what is underneath the alpha.

My third tip is to remove all your masks. Masking items will cause them to be redrawn to the screen every frame, even when a redraw is not necessary. This will cause an extreme decrease in performance as your application will render more than it needs to.

The example in the steps below shows how you could organize your visual assets in order to create a scrollable list.

Optimize a Scrollable List

1. Create a MovieClip symbol to represent a cell in the list.

   Note: See Chapter 2, “Getting Started with Flash CS5,” for more information.

2. Stack the cells on the Stage exceeding the height of the Stage.

   Note: Make sure that the cells do not overlap and that they are on a whole pixel.

3. Click the Selection tool.

4. Select all the cells on the Stage.

   Note: You can also press the Ô+A (Ctrl+A) keyboard shortcut to select them all.

5. Convert the selected cells to a MovieClip symbol by pressing the F8 key.

   The example in the steps below shows how you could organize your visual assets in order to create a scrollable list.
Using vectors can have a big impact on the performance of your application. A good practice to follow is to eliminate the use of vector objects unless absolutely necessary. This may not always be possible — for example, if you are creating a paint application. In these instances, you may be able to get a good enough performance if the application is not that intensive. However, if you are creating a game and you have some illustrated characters and objects in vector, you will want to export them to images. The amount of detail and complex vector points that an animated character can have is so great that the renderer will have a hard time keeping up with a decent frame rate.

From within the Flash IDE, you can easily export each frame as a PNG or other image format. Reimporting your vector assets as images will greatly improve your chances of achieving a decent frame rate. Exporting a lot of frames from a Timeline animation can be a very time-consuming process. You can automate this by creating a JSFL script file, which will move the play head to each frame and export it to an image.

Test Your Movie

10. Press Ô + Enter (Ctrl+Enter).

- The extra cells are covered by the header and footer graphics.

Cells are not overlapped or masked, which improves performance.
Manage Mouse Events

As mentioned in the preceding section, “Optimize Your Display List,” keeping your display list shallow will have an impact on the performance of your application. One of the reasons is how mouse and touch events are handled within ActionScript 3.0.

There are three phases to a MouseEvent. The capturing phase is the first phase and occurs when an event is fired. The event starts with the topmost parent display object, or the Stage, and works its way down the display list hierarchy until it reaches the target in which the event originated. The second phase is the target phase, which occurs when the event reaches the display object from which the event originated. The third phase is the bubbling phase. During the bubbling phase of an event, the event follows the reverse path of the capturing event, all the way to the topmost parent display object.

With this knowledge, you can see how having deeply nested display lists can easily and quickly cause a decrease in performance, as the event will visit each item in the display list in both the capturing and bubbling phases.

There are a couple methods on the Event class that enable you to stop the event in its current phase. The stopPropagation() method prevents the processing of any event listeners in any nodes subsequent to the current node in the event flow. Calling this method will not have any effect on the current node, and it can be called during any phase of the event.

The stopImmediatePropagation() method prevents the processing of any event listeners in the current and subsequent nodes in the event flow. This method takes effect immediately and affects event listeners in the current node. However, you should note that this does not cancel the behavior of the associated event.

Manage Mouse Events

1. Add a Button symbol to the Stage.

   Note: See Chapter 2 for more information.

2. Give it an instance name, such as btn.

3. In the Actions panel, add a CLICK listener to the Button, such as btn.addEventListener(MouseEvent.CLICK, onClick);

4. Create an event handler function.

5. Output the phase of the event, such as trace("button clicked", e.eventPhase);

6. Stop the event from bubbling up, such as e.stopImmediatePropagation();
Chapter 15: Optimizing Performance

7 Listen for the bubbled event on the Stage, such as
    stage.addEventListener(MouseEvent.CLICK, onStageClick);

8 Create an event handler function.

9 Output the phase of the event, such as
    trace("stage clicked", e.eventPhase);

10 Press Ô+ Enter (Ctrl+Enter) to test your movie.

    • Check the Output panel to see the event phases.

    The propagation of the event has been stopped and will not
    bubble.

    Note: You can remove the line of code in order to see the
    event bubble up to the Stage.

Apply It

When the user drags his or her finger while it is pressed on the screen, a MouseEvent.MOUSE_MOVE event is fired. This event can be fired multiple times a frame as the finger moves quickly across the screen, which can decrease performance. The following is an example of how to use the Event.ENTER_FRAME event to track the position of a touch:

    stage.addEventListener(Event.ENTER_FRAME, onFrame);
    function onFrame( e:Event ):void{
        var x:Number = stage.mouseX;
        var y:Number = stage.mouseY;
        trace( x, y );
    }
Understanding cacheAsBitmap

The cacheAsBitmap property is a property that exists on all DisplayObject instances. If the cacheAsBitmap property is set to true, a bitmap representation of the display object will be cached internally. To get the highest performance increases in your application, use this method in conjunction with setting the rendering mode to GPU.

When your display object is cached using the cacheAsBitmap property, it is sent to the GPU. This enables you to perform simple translations along the x and y axis without having the display object reuploaded to the GPU. This works very well with display objects whose states remain fairly constant.

If there are any changes to the bounds of the display object — for example, if the object is scaled or rotated — the bitmap will be re-created and uploaded to the GPU.

For more information on scaling and rotating display objects, see the following section, “Understanding cacheAsBitmapMatrix.”

The cacheAsBitmap property is automatically set to true when a filter is applied to the display object. If you set cacheAsBitmap to false and then apply a filter, it will set the property to true. After all filters are removed from the display object or the filter array is empty, the property will be reset to its previous value.

When the cacheAsBitmap property is set to true, the rendering does not change. However, the display object snaps to the nearest pixel. To ensure that you do not see a visual shift in your display objects when setting the cacheAsBitmap property, make sure that all your elements are on whole pixels.

1. Create a MovieClip symbol and place it on the Stage.

   Note: See Chapter 2 for more information.

2. Give it an instance name, such as sushi_mc.

   Note: You can set the cacheAsBitmap property in the IDE as well as in code.

3. Click Cache As Bitmap.

4. In the Actions panel, cache the symbol, such as sushi_mc.cacheAsBitmap = true;

5. Register a listener for the enter frame event, such as stage.addEventListener( Event.ENTER_FRAME, onFrame );

6. Create an event handler.

7. Move the symbol incrementally every frame, such as sushi_mc.y -= 2;
8 Click File ➔ Open.
   The Open dialog box appears.
9 Click the application descriptor file.
10 Click Open.

The application descriptor file is opened.

11 Set the <renderMode> node to gpu.
   When you compile and install your application on your device, the cacheAsBitmap property will upload a representation of the display object to the GPU.

Apply It

The opaqueBackground property on a DisplayObject instance specifies whether it is opaque and the color value of the background. Setting this property to a number value creates an opaque background on the object to the color value that the value specifies. If it is set to null, the background of the display object will be transparent. When the cacheAsBitmap property is set to true, setting the display object to opaque can improve rendering performance. It is important to note that the opaque background region does not respond to mouse events.

```plaintext
myshape.opaqueBackground = 0x000000;
myshape.cacheAsBitmap = true;
```
Understanding cacheAsBitmapMatrix

The preceding section, “Understanding cacheAsBitmap,” explores how to use the cacheAsBitmap property to upload display objects to the GPU. This works well if the bounds of the display object do not change, such as if it is not scaled or rotated. If your display objects are required to be scaled or rotated, however, you can use the cacheAsBitmapMatrix property to prevent the bitmap from being re-created and uploaded to the GPU.

Setting the cacheAsBitmapMatrix property to a valid Matrix instance will define how the display object is rendered when cacheAsBitmap is set to true. Your application will use this Matrix as a transformation matrix when rendering the bitmap version of the display object. When the cacheAsBitmapMatrix property is set, the application will retain a cached version of the bitmap when a transformation of the display object occurs, such as rotation, scale, and translation. When the rendering mode of the application is set to GPU, the display object will be stored as texture in video memory. This allows all supported transformations to occur on the GPU, which can perform these transformations much faster than the CPU.

Simply setting the property to the identity matrix — for example, new Matrix(); — usually suffices, however, if you use any Matrix instance to upload a different bitmap to the GPU. It is best practice to set the Matrix to the size that it will appear in the application. Transformations of the display object are not required to occur on the matrix. After the cacheAsBitmapMatrix property is set, you can use the rotation and scale properties of the display object in order to transform the object.

1. Create a MovieClip symbol and place it on the Stage.

_Note: See Chapter 2 for more information._

2. Give it an instance name, such as sushi_mc.

3. In the Actions panel, cache the symbol, such as sushi_mc.cacheAsBitmap = true;.

4. Cache the matrix, such as sushi_mc.cacheAsBitmapMatrix = new Matrix();.

5. Register a listener for the enter frame event, such as stage.addEventListener( Event.ENTER_FRAME, onFrame );.
It is important to understand that the `cacheAsBitmapMatrix` property works only on 2D surfaces, as the texture that is uploaded is only a 2D surface. If you want to use a 3D surface, you simply set the `z` property to 0 or the value that you want. Setting the `z` property automatically sets your display object to be a 3D surface. The following syntax sets a `Sprite` instance to be a 3D surface:

```javascript
mysprite.z = 0;
```

There is also an easy way to set the `cacheAsBitmapMatrix` to the same `Matrix` as it currently is on the Stage:

```javascript
mysprite.cacheAsBitmapMatrix = sprite.transform.matrix.clone();
```

This is very useful when your display object has already been transformed before you set the matrix.