Flash Asset Analysis, Profiling and Optimization

A Case Study of Zynga's FarmVille

by Brett Bond

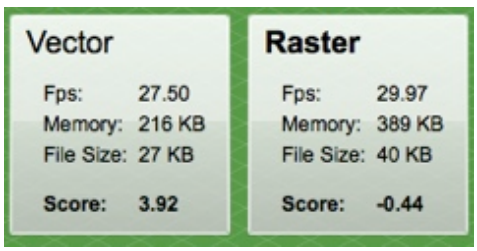
Back to the perennial topic of optimization for Flash games. There are many articles on the subject, and rightly so, because when writing code for games, sooner (hopefully) or later performance comes up. In this article, I'd like to relate some of my experience working for Zynga on the largest Flash game on Earth, FarmVille, and share the lessons of analysis, profiling and optimization that may shed some light on how we all can improve our games, no matter how big they may become.

Many Flash gaming articles start with an assumption that your team comprises 3D artists who just gave up their jobs at Pixar. And they advise rasterization, the process whereby only bitmap images are rendered, thus saving the CPU cycles required to calculate the screen coordinates of vector shapes created by the 2D tools such as Adobe Illustrator, InkScape, and Corel Draw.

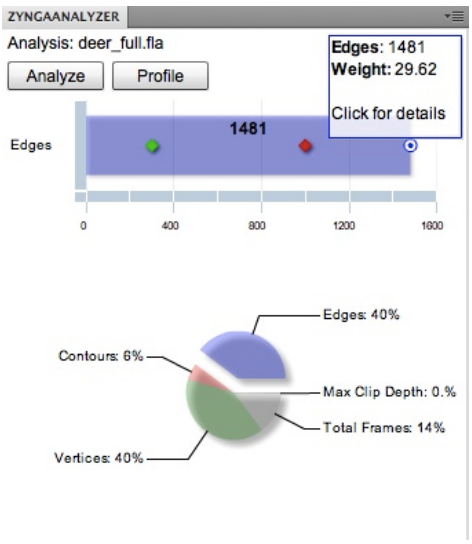
Though many top Flash games start with 3D models, created with Maya, 3ds Max, or Blender and the like, private citizens or even commercial enterprises, may have core competencies in the 2D vector drawing land and not in 3D. Not to mention that 2D illustration is often much faster and cheaper to create than 3D, and offers a wider range of possible styles.

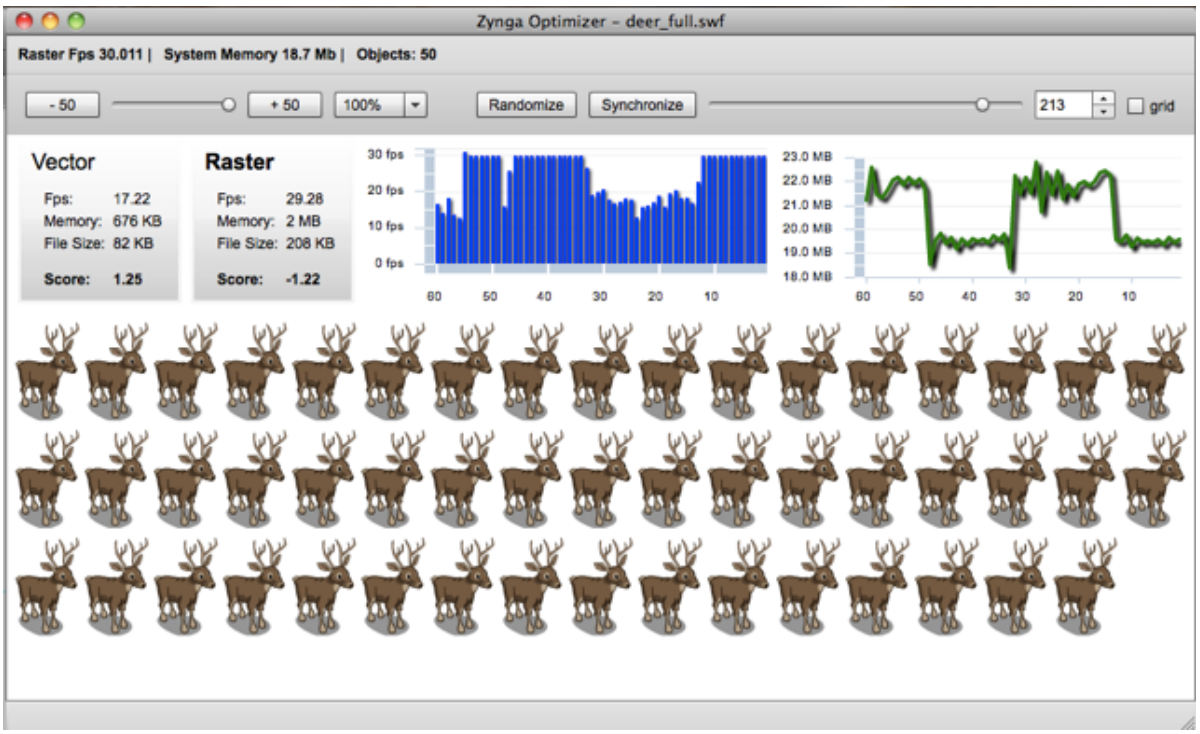
Well my vectorphilic friends, this article is for you. At its heart, the choice between whether to use vector and raster artwork is a size for speed trade-off: rasters usually require longer download times for better framerate. So let's get into a case study, looking at how Zynga's FarmVille rates raster vs. vector, and how you can produce better assets benefiting from all their hard work regardless of your game's size.

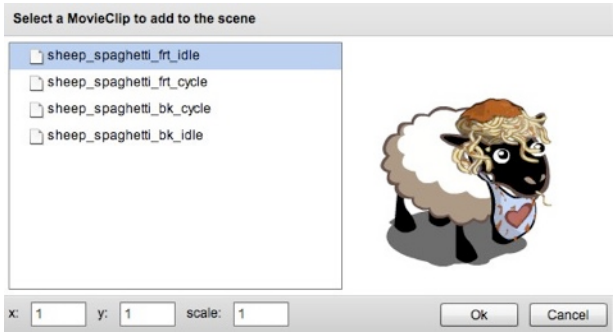
I built the Zynga Optimizer as a profiling and rasterization tool that scores artwork on a non-linear scale based on vector art framerate, memory and file size. The Optimizer also rasterizes vector art, creating a series of bitmaps that are bigger, but also generally faster, than the original vector art. The Optimizer then scores the raster, comparing its relative speed improvement with its relative size increase, ranking each asset in comparison to all other artwork in the game. That's very abstract, so let's look at some examples.



The FarmVille Deer contains 265 frames, with an average of 1481 edges per frame. The Optimizer scores it at a 1.11 out of 5.0. As you'll see, a very practical way to compare artwork is to add a large number of instances to the scene and check the framerate. So, I put 50 instances of the Deer in the scene and see a framerate of 16.5 frames per second (fps) on my Macbook Pro, using 680KB of memory and file size of 82KB. By rasterizing this animation we achieve a framerate of 30 fps, but now require 2MB of memory (each!), and a file size of 208KB. Because of the big memory jump the raster scores a relative -1.05 in the Optimizer. The negative score indicates it isn't worth using. In this case it's necessary to improve the original artwork. More on that later.







Now let's look at the Sheep Eating Spaghetti. This vector art weighs in at 6741 edges, has 600 frames, and similar to the Deer, scores 1.6 out of 5, rendering just 3 frames per second. When rasterized, framerate goes right up to 30 fps, with only slightly increased memory and filesize. Why does the sheep perform so badly as vector art, and why is it improved so much when rasterized? The answer is the number of edges being animated.

The cost of animating vectors is high. And the benefit of rasterization becomes apparent when artwork contains high vector density (a large number of edges per screen area).

Finally, let's look at the Duck. The Duck has just 370 edges, and its Idle animation contains just 117 frames. Yet, the Duck is cute, the Duck is lively. The Duck is such a nice duck that it makes us wonder whether the Sheep Eating Spaghetti and the Deer really need all those curves. The key lesson for those skilled in illustration is to reduce vector density and reduce the number of animated vectors in each frame. This will allow you to create vector animations that will perform efficiently without the need for rasterization.

But what about when rasterization is needed for inefficient vector artwork already created? If you have inefficient assets, sometimes rasterizing is the only way to get your framerate up. In this case, the use of frame differencing (as in video compression) is very helpful to reduce file and memory size. Open source png libraries such as Adobe's AS3CoreLib provide image compression on each frame for further file size reduction.