Overview:

BitmapTest is a simple program to test the speed of animating bitmaps on the screen. It also allows various optimizations to be turned on and off, to see how much they help. This program is intended as a demo program for developers, and is in the Public Domain. No claims are made to the usefulness or suitability of this program, or its source code.

This program demonstrates animating bitmaps, use of timer & event handling, use of the List object, limiting window resizing, and includes a simple generic object to handle moving a bitmap.

Comments are welcomed to help make this a better demo program. The code is intended to provide clear, correct use of the App Kit, rather than fast, but dirty code.

Basic Operation:

The simplest way to use the program is to click the "Create" button once, and then click the "Start" button. a small bitmap should appear and then begin bouncing

around the view. Clicking "Create" again will add another bitmap. The check boxes in the Control section allow various optimizations described below.

New Bitmap Options:

Create: Creates a new bitmap using the parameters described below.

Bitmap Size: Sets the size of the bitmap used for the next bitmap. The bitmaps were made with the Icon App, and added to the .nib file in the tiff files section of the project inspector. You can replace them and re-compile. Alpha values other than 0 and 1, and nicer pictures could easily have been used.

Remove Last One: Removes the last created bitmap.

Number: Shows how many bitmaps are on the screen.

Initial Velocity: Lets you set the initial bitmap velocities in units/move. The units are points, or screen pixels. The initial position is always the center of the view. The

"Random" check box will disable these inputs and assign random velocities from -5 to 5 on creation.

Control Options:

Start: Toggle to start and stop the motion of the bitmaps.

Allocate G State: Allocates/de-allocates a graphics state for the View. This takes a lot of memory, and is only recommended for computationally intensive and important Apps. Note that the Tight Draw Loop (below) is actually faster and more efficient.

NX_PING: Synchronizes the window server after each draw for smoother animation.

Erase Background: Erases the whole view before each redraw step. This is slower than erasing just the bitmaps for big views, and few numbers of bitmaps. Click this on and off to clear the view.

Erase Bitmaps: Erases the area behind each bitmap before each step of the animation.

This is faster than erasing the whole screen for a small number of bitmaps, but as the number gets larger, the overhead of messaging and erasing each bitmap has a bigger effect.

Tight Draw Loop: Makes the animation go in a continuous loop until an event is received. The graphics state is locked and unlocked just once. In this program, the timer is ignored when using this optimization, but that could have been easily allowed for. Notice that using a tight draw loop is the fastest way to keep the animation going. It will stop for events, and doesn't take a lot of Display PostScript memory like allocating a GState does.

Timer Period: Lets you specify the time in seconds between each redraw (decimal fractions are OK). Note that this will not have an effect if a tight drawing loop is used, although that funtionality could be easily added.