SpriteWorld — Scrolling

by Vern Jensen

What are the scrolling routines?

The scrolling routines enable you to write games with a scrolling background, such as Power Pete, Tubular Worlds, or nearly any Nintendo type game. Much care has been taken to make these routines as fast and efficient as possible, while also keeping them easy to use. Through the combination of being easy to use, fast, and free, it is my hope that they will spawn a new breed of cool scrolling games for the Mac.

How do they work?

Game developers usually use one of two techniques when creating a scrolling game: A) Make an offscreen area as large as the “virtual world” that they will scroll around in (the method used by Pac-In-Time), or B) Use an offscreen area slightly larger than the area visible on the screen, and scroll and redraw the background whenever the screen scrolls. The first method can take up a lot of memory and limit the size of the scrolling area considerably, while the second method can be quite slow, because of all the drawing that must be done offscreen each frame. There are other methods as well, but they each have their limitations.

SpriteWorld uses none of the methods described above, but rather a new “wrapping” technique. Here’s how it works: an offscreen area either the same size as the window or slightly larger is used for the background and work frames. To scroll the screen, we simply change the location of the rectangle that is copied from the offscreen area to the screen each frame. If part of this rectangle goes past the border of the offscreen area, it is clipped and “wrapped” to the other side of the offscreen area. So if the rectangle extends past the right and bottom side of the offscreen area, then four sections will be copied to the screen that frame: the main section after it is clipped, the right side which is wrapped to the left, the bottom side which is wrapped to the top, and the bottom-right corner, which is wrapped to the top-left corner. You can run the program “Wrapping Illustration” to see what happens offscreen during a scrolling animation. (This program is part of the SpriteWorld Extra Demos package, available from the SpriteWorld Web page.)

Because SpriteWorld uses this technique, it doesn’t need a huge offscreen area as large as the scrolling world, and it doesn’t have to do any drawing offscreen between frames, other than update the tiny portion that scrolls into view. Because the offscreen area is sometimes copied to the screen in several pieces, you may notice “seams” where one piece was copied to the screen before another. These can be eliminated by using a special blitter that draws the left and right rectangles at once. See the documentation for SWSetDoubleRectDrawProc for more info.

The good news is that SpriteWorld does all the dirty work for you; you don’t have to worry about how to “wrap” things yourself. You just create a SpriteWorld, set the boundaries of your scrolling world, and set up your sprites, and SpriteWorld takes care of everything for you. As far as you are concerned, it's just as if you had a huge offscreen area the size of your scrolling world, but without the memory it would take up!

The scrolling and tiling routines were meant to go hand-in-hand. Because of the complex method of scrolling that SpriteWorld uses, it would be quite difficult for the user to create their own routines to update the background as the SpriteWorld scrolls, because the routines would have to

be able to “wrap” from one side of the offscreen area to the other. Fortunately, the tiling routines do this wrapping for you, automatically. So if you want to make a scrolling game that uses tiles, SpriteWorld is for you! If you don’t want to use tiles, then you still have the option of making the offscreen areas as large as the scrolling world so you can draw the entire background in it before the animation starts, and then don’t have to worry about updating it while the animation is running. However, most scrolling games use tiling, so most people shouldn’t have to worry about this.

Why am I doing this?

I made these routines simply because no one else did. Many people have wanted to make a scrolling game with SpriteWorld but haven’t been able to, because SpriteWorld didn’t have support for scrolling in the past. I am interested in making scrolling games myself, which is why I wrote these routines. But it would be selfish for me to want to keep them all to myself, so I’m making them freely available to everyone. Of course, this means more work for me, since I have to document and support the routines, but it will be worth it if some good games are made as a result. Hopefully you will take the time to make a good, polished game that would make me proud of these routines.

How to contact me

If you have any comments, questions, suggestion, or bug reports, you can contact me at Jensen@loop.com. If you think that you have run into a bug, please do not e-mail me immediately. Take the time to try to isolate the problem as best as you can. If possible, build a little demo that shows the bug and send that to me. As a general rule, you should experiment with something for a few days before assuming that it is a bug in SpriteWorld, unless you are absolutely sure you know what the problem is. Also, if the problem is in your own code, and not in SpriteWorld, try to get help somewhere else, like comp.sys.mac.programmer.games, or the SpriteWorld mailing list. Although I'll certainly try to help if nobody else can.

Also, please let me know if you make a game that uses the scrolling or tiling routines, as I would be very interested in seeing it! (Hopefully you can provide a free copy. After all, it’s just a small way of saying “Thanks!”)

Getting Started

How to set up a Scrolling SpriteWorld

In order to use the scrolling routines, you must include the file “Scrolling.c” in your project. Setting up a scrolling SpriteWorld is very straightforward. You create the SpriteWorld, Layers, and Sprites, and put them together the way you normally would. You have to make only a few extra function calls for a scrolling SpriteWorld.

The first thing you need to do is to determine the size of the offscreen areas, which might need to be larger than the area that is visible on the screen. If you use the tiling routines, you will need to make sure that the width and height of the offscreen area is evenly divisible by the width and height of the tiles that you are going to use, so that the tiles fit perfectly in the offscreen area without any of them being clipped. This might make your offscreen areas slightly larger than the area on the screen. If you use the tiling routines, then you do not need to make the offscreen area as large as your “virtual scrolling world”, since with the wrapping technique described above,

SpriteWorld can use that small area to build the frame that is copied to the screen.

However, if you don't use the tiling routines, then you will need to create an offscreen area as large as your scrolling world (the area that you will be scrolling around in). Use the last parameter to SWCreateSpriteWorld to set the size of the offscreen areas. See the Scrolling Demo for some example code that demonstrates how to make sure tiles fit evenly in the offscreen area. Also, the Large Background Scrolling demo shows how to create offscreen areas that are as large as your scrolling world.

Other functions you might want to call while preparing for the animation would be SWSetScrollingWorldMoveBounds to set the boundaries of the scrolling world, SWMoveVisScrollRect to tell SpriteWorld where the animation is going to start, and SWSetScrollingWorldMoveProc to tell SpriteWorld what routine to use to control the scrolling. You would then call SWUpdateScrollingSpriteWorld to set things up, and then use SWProcessScrollingSpriteWorld and SWAnimateScrollingSpriteWorld to drive the animation. That's it!

After everything is in place, all you need to do to control the scrolling is to move around spriteWorldP->visScrollRect. The visScrollRect is a Rect structure that is part of the SpriteWorldRec that defines what part of the virtual scrolling world is currently visible on the screen. I refer to it as a "virtual" scrolling world because the world may be much larger than the offscreen area, thanks to the wrapping routines. (It can be up to 32767 pixels high and long!) But since SpriteWorld handles all the details for you, you can move the visScrollRect anywhere in your virtual world, and everything will appear properly on the screen just as if your offscreen area were really as large as your virtual world. There are three routines provided for you to move the visScrollRect: SWMoveVisScrollRect, SWOffsetVisScrollRect, and SWSetScrollingWorldMoveProc. Although you may use the first two functions every now and then, the most common way of moving the visScrollRect will be with the ScrollingWorldMoveProc. Here you can write your own function that controls the scrolling in whatever way you wish.

To assist you in moving the visScrollRect, SpriteWorld provides two variables which are also part of the SpriteWorldRec: horizScrollDelta and vertScrollDelta. The values in these variables are automatically added to the visScrollRect after the ScrollingWorldMoveProc has been called. This allows your ScrollingWorldMoveProc to change the scrollDelta values and have them reflected in the very next frame of the animation. You can change the scrollDelta variables either by accessing them directly from your ScrollingWorldMoveProc, or by calling SWSetSpriteWorldScrollDelta.

Improving the Look of the Animation

Due to SpriteWorld's scrolling technique, the screen is often updated in several separate pieces, rather than all at once. This results in ugly "seams" which are visible between the pieces that are copied to the screen when scrolling. However, by calling SWSetDoubleRectDrawProc, you can specify a DrawProc which draws the left and right pieces at the same time, eliminating any seams which would otherwise be visible when the pieces are drawn separately. See the documentation for SWSetDoubleRectDrawProc later in this file for more information.

A Note about Drawing in the Offscreen Areas

Because of the complicated technique SpriteWorld uses to achieve its fast scrolling, you can't simply draw something in the offscreen area. For instance, if you want to have text appear in front of the scrolling background, you can't simply draw the text directly into the offscreen area.

Instead, you need to create a Sprite big enough to hold the text, and draw the text into the Sprite and the Sprite's mask, and then the text will be drawn correctly into the offscreen area when the Sprite is drawn.

Additionally, if you make a custom drawProc for your Sprite (such as the one that draws the "white" version of a Sprite in the Split-Screen Demo when the Sprite is hit by a bullet), you must make sure that any drawing you do is clipped to the dstRect that is passed to your drawProc. Of course, you should do this anyway, but it is especially important during a scrolling animation, since if you draw outside of that rect, you can end up drawing where you shouldn't be drawing, if the Sprite is being "wrapped". This can cause fragments of your drawing to appear in strange places on the screen.

For an example of how to clip your drawing to the dstRect passed to your drawProc, see the SubHitDrawProc function in the SpriteStuff.c file in the Split-Screen Demo folder. Note that this clipping is only necessary when you make QuickDraw calls (such as PaintRgn) from your drawProc; it isn't necessary if you use a blitter, since the blitter automatically clips its drawing to the rect you pass to it.

Tips for Faster Scrolling

Everyone wants their scrolling game to go as fast as possible. Besides the obvious ways of speeding up a scrolling animation, such as using an interlacing DrawProc, making the view area smaller, and running in a lower bit depth, there are two additional ways you can make sure your scrolling game goes as fast as possible:

1) Make sure your worldRect's left side is positioned on an even horizontal long word boundary. (In 8-bits, this would be every 4th pixel, starting at 0. In 16-bits, it would be every 2nd pixel.) So if your SpriteWorld was created from a window and does not use a custom worldRect, make sure your window's left side is on a long world boundary. If you use a worldRect, make sure its left side in global coordinates is on a long word boundary. This is a good idea for any animation, scrolling or not. The vertical position of the worldRect doesn't matter.

2) Use one of the DoubleRectDrawProcs for the scrolling. (See SWSetSpriteWorldDoubleRectDrawProc for more information.) Not only will this improve the look of your animation, but it will keep it from slowing down when the right piece is copied to the screen at an odd location. If you can not use one of the DoubleRectDrawProcs for some reason, make sure to use CopyBits, not a custom blitter, as your screenDrawProc. This is because CopyBits handles copies to odd screen locations much better than SpriteWorld's standard rect blitters, which were designed only for copying Sprites, which are always aligned. However, if you can use a DoubleRectDrawProc, then go ahead and use a blitter as your screenDrawProc as well, as this will only be used to copy the left (aligned) rects.

Variables Used for Scrolling

There are certain variables in the SpriteWorldRec that are used for scrolling that you should become familiar with. You may wish to access these directly at times, or you may just want to know what they are for so you have a better understanding of how the scrolling routines work. This is not a list of all the variables, but only of the ones that might be useful to you. (For instance, you might want to access the visScrollRect to see what portion of the SpriteWorld is currently

visible on the screen.)

`Rect`	`visScrollRect`	This holds the current position of the visible scrolling rectangle; that is, the area of your scrolling world that the user sees on the screen. By moving this, you can scroll to a different part of the scrolling world.
`Rect`	`oldVisScrollRect`	The position of visScrollRect from the previous frame. Used internally by SpriteWorld.
`Short`	`horizScrollDelta`	The horizontal scrolling delta.
`Short`	`vertScrollDelta`	The vertical scrolling delta.
`Rect`	`scrollRectMoveBounds`	Move bounds for visScrollRect. See SWSetScrollingWorldMoveBounds.

Scrolling Function Reference

This section documents the scrolling routines. They are not in alphabetical order, but are grouped by the type of action they perform. The following is a list of each function in the order in which they are documented, so that you can find the function you want quickly. They are also listed with the parameters as you would pass them to each function, so you can also look here if you forget the order of the parameters in a particular function.

SWUpdateScrollingWindow(spriteWorldP)

SWUpdateScrollingSpriteWorld(spriteWorldP, updateWindow)

SWProcessScrollingSpriteWorld(spriteWorldP)

SWAnimateScrollingSpriteWorld(spriteWorldP)

SWFastAnimateScrollingSpriteWorld(spriteWorldP)

SWSetScrollingWorldMoveBounds(spriteWorldP, &scrollRectMoveBounds)

SWSetScrollingWorldMoveProc(spriteWorldP, MyScrollingWorldMoveProc, followSpriteP)

SWSetSpriteWorldScrollDelta(spriteWorldP, horizDelta, vertDelta)

SWSetDoubleRectDrawProc(spriteWorldP, doubleRectDrawProc)

SWMoveVisScrollRect(spriteWorldP, horizPos, vertPos)

SWOffsetVisScrollRect(spriteWorldP, horizOffset, vertOffset)

SWUpdateScrollingWindow

This function updates the window of a scrolling SpriteWorld from its work Frame.

void SWUpdateScrollingWindow(SpriteWorldPtr spriteWorldP)

`spriteWorldP`	A pointer to a SpriteWorld whose window needs updating.

Description:

This function will update the contents of the window of the spriteWorldP using the current position of visScrollRect to copy from the offscreen area. No drawing is done offscreen, so whatever was drawn last frame will be copied (as long as the visScrollRect has not been moved since the last frame). You would typically use this function in response to an update event.

See Also:

SWUpdateScrollingSpriteWorld

SWUpdateScrollingSpriteWorld

This function will copy the background area to the work area, render the current frame of the animation in it, and optionally copy the result to the screen.

void SWUpdateScrollingSpriteWorld(SpriteWorldPtr spriteWorldP,
Boolean updateWindow)

`spriteWorldP`	A pointer to a SpriteWorld to be updated.
`updateWindow`	A Boolean indicating whether to update the window or not.

Description:

This function does the same thing as SWUpdateSpriteWorld, but is for scrolling SpriteWorlds instead. It copies the background Frame to the work Frame, builds the current frame of the animation in the work Frame by drawing the sprites in it, and optionally copies the result to the screen. You should call this at the beginning of a scrolling animation. If you want to update the window in response to an update event, you should use SWUpdateScrollingWindow.

Before calling this function, you should position the visScrollRect in the area of your scrolling world that you want to be copied to the screen. If your scrollingWorldMoveProc does this, you can simply call it like so:

```
MyWorldMoveProc(spriteWorldP, spriteWorldP->followSpriteP);
```

Use the updateWindow parameter to control whether the work Frame is copied to the window. You would usually pass true as this parameter, but may wish to pass false if you want to update the window yourself so you can produce a special effect such as a screen wipe.

SWProcessScrollingSpriteWorld

This function will process a scrolling SpriteWorld.

void SWProcessScrollingSpriteWorld(SpriteWorldPtr spriteWorldP)

`spriteWorldP` A pointer to a SpriteWorld to be processed.

Description:

Use this function to drive the animation of a scrolling SpriteWorld. This function calls SWProcessSpriteWorld to process the sprites, then calls the ScrollingWorldMoveProc, and then offsets the visScrollRect by the value contained in the SpriteWorld's scrollDelta.

Note: Because the Sprites are processed before the visScrollRect is moved, if your Sprite relies on the current position of the visScrollRect for its own movement, you should not give the Sprite a MoveProc, so that the Sprite is not moved by SWProcessScrollingSpriteWorld, but rather, you should call the MoveProc yourself to move the Sprite after SWProcessScrollingSpriteWorld is called (but before SWAnimateScrollingSpriteWorld is called), so the sprite can get the latest version of the visScrollRect. For more information, see the section entitled "How to keep Sprites at a fixed position on the screen while scrolling" in the SpriteWorld Tips & Tricks file.

See Also:

SWAnimateScrollingSpriteWorld
SWSetScrollingWorldMoveProc

SWAnimateScrollingSpriteWorld

This function will render a frame of a scrolling animation on the screen.

void SWAnimateScrollingSpriteWorld(SpriteWorldPtr spriteWorldP)

`spriteWorldP` A pointer to a SpriteWorld to be animated.

Description:

This function is similar to SWAnimateSpriteWorld, but is for scrolling SpriteWorlds. It erases each Sprite and redraws it in its new location offscreen, and then copies the area defined by the visScrollRect to the screen. You would typically call this right after calling SWProcessScrollingSpriteWorld.

See Also:

SWAnimateSpriteWorld
SWFastAnimateScrollingSpriteWorld
SWProcessScrollingSpriteWorld

SWFastAnimateScrollingSpriteWorld

This function only updates the portions of the screen that have changed.

void SWFastAnimateScrollingSpriteWorld(SpriteWorldPtr spriteWorldP)

`spriteWorldP` A pointer to the SpriteWorld to be animated.

Description:

This function is essentially the same as SWAnimateScrollingSpriteWorld, except that it only updates the parts of the screen that have changed (such as animated Tiles or Sprites), rather than redrawing the entire screen. This function is not recommended for use during a game, since the animation will slow down dramatically when scrolling and suddenly speed up when not scrolling. It is, however, quite useful for applications such as a level editor, where you may wish to quickly update the screen after the user draws a Tile.

This function checks to see if the visScrollRect has moved since the previous frame. If it has, it calls SWAnimateScrollingSpriteWorld so that the entire screen gets updated; otherwise, this function simply updates the parts of the screen that have changed.

SWSetScrollingWorldMoveBounds

This will set the boundary of the scrolling area, and make sure the visScrollRect is still within that boundary.

void SWSetScrollingWorldMoveBounds(SpriteWorldPtr spriteWorldP, Rect* scrollRectMoveBounds)

`spriteWorldP` A pointer to a SpriteWorld
`scrollRectMoveBounds` The address of a rectangle specifying the moveBounds

Description:

You should call this function before the animation starts to tell SpriteWorld how big your scrolling area is. The maximum possible size of your scrolling world is 0 for the top and left sides, and 32767 for the bottom and right sides. These are the default movement boundaries. SpriteWorld automatically enforces the scrolling movement boundaries, so once you've set them, you don't need to worry about them any more.

If you do not use tiling, then you would generally make the offscreen areas of the SpriteWorld as large as your scrolling world, and then call `SWSetScrollingWorldMoveBounds(spriteWorldP, &spriteWorldP->backRect)` to set the boundary to the size of the offscreen areas.

If you do use tiling (and I expect that most people will), then you will want to set the bounds to the size of your tileMap. Scrolling past the bounds of the tileMap could cause SpriteWorld to crash, as it would be trying to read tiles that aren't there. If you use tiling, you might make a call like this:

```
SetRect(&moveBoundsRect, 0, 0,
  spriteWorldP->tileMapStructP->numCols * spriteWorldP->tileWidth,
  spriteWorldP->tileMapStructP->numRows * spriteWorldP->tileHeight);
```

```
SWSetScrollingWorldMoveBounds(spriteWorldP, &moveBoundsRect);
```

Note that this example would only work correctly if it was called after SWInitTiling was called and the TileMap was installed in the SpriteWorld.

After setting the new movement boundaries, this function also checks to make sure the visScrollRect is still within those boundaries. If it is not, it moves it so it is.

See Also:
SWSetScrollingWorldMoveProc

SWSetScrollingWorldMoveProc

This will set the procedure that moves the visScrollRect, which controls what portion of the scrolling world is currently visible on the screen.

void SWSetScrollingWorldMoveProc(SpriteWorldPtr spriteWorldP,
WorldMoveProcPtr worldMoveProcP, SpritePtr followSpriteP)

`spriteWorldP`	A pointer to a SpriteWorld
`worldMoveProcP`	The worldMoveProc
`followSpriteP`	An optional “follow Sprite”

Description:
SWSetScrollingWorldMoveProc provides a way for you to control the speed and direction of the scrolling. Since in most games, the scrolling “follows” a particular Sprite, such as the main character, an optional followSpriteP parameter is provided. This followSpriteP will then be passed to the WorldMoveProc each time it is called, so you can deal with it as you wish. If you do not need the scrolling to follow a Sprite, simply pass NULL as the followSpriteP (and a NULL will also be passed to the WorldMoveProc as the followSpriteP).

Your WorldMoveProc should be defined like this:

SW_FUNC void MyWorldMoveProc(SpriteWorldPtr spriteWorldP,
SpritePtr followSpriteP);

The visScrollRect structure of the SpriteWorld specifies the area in your scrolling world that will be copied to the screen. You need to move the visScrollRect in order to scroll. The standard way of doing this is by changing the values in the horizScrollDelta and vertScrollDelta variables of the SpriteWorld. You can also use SWOffsetVisScrollRect and SWMoveVisScrollRect, although these are less frequently used by the WorldMoveProc.

By changing the values of spriteWorldP->horizScrollDelta and spriteWorldP->vertScrollDelta, you can control the scrolling speed and direction. SpriteWorld will automatically offset the visScrollRect with these variables as soon as your moveProc is finished, so any changes your moveProc makes to these variables will be reflected in the very next frame of the animation. When SpriteWorld moves the visScrollRect, it is automatically kept within its moveBounds (see SWSetScrollingWorldMoveBounds).

If you want, you can access spriteWorldP->visScrollRect (a Rect structure) to see the current position of the visScrollRect, but you shouldn't move the visScrollRect by changing its values directly. Instead, use either the spriteWorldP->horizScrollDelta and spriteWorldP->vertScrollDelta variables, or the functions SWMoveVisScrollRect and SWOffsetVisScrollRect.

The WorldMoveProc is called after all the sprites in the SpriteWorld have been processed. So if your WorldMoveProc is set up to follow a particular Sprite, it will be given the Sprite's latest position as it will be seen in the next frame.

If you want to "turn off" a worldMoveProc so SpriteWorld doesn't call it anymore, you can simply call this function with a value of NULL:

```
SWSetScrollingWorldMoveProc(spriteWorldP, NULL, NULL).
```

See Also:
SWMoveVisScrollRect
SWOffsetVisScrollRect
SWSetSpriteWorldScrollDelta
SWSetScrollingWorldMoveBounds

SWSetDoubleRectDrawProc

This function sets the DoubleRectDrawProc, which copies side-by-side rectangles to the screen at the same time, eliminating any seams that might otherwise be visible between them.

```
SW_FUNC void SWSetDoubleRectDrawProc(SpriteWorldPtr spriteWorldP,
    DoubleDrawProcPtr doubleRectDrawProc);
```

`spriteWorldP`	A pointer to a SpriteWorld
`doubleRectDrawProc`	The DoubleRectDrawProc

Description:
This function allows you to eliminate the "tearing" that is visible when scrolling due to the fact that the screen is updated in several pieces rather than all at once. When you install a DoubleRectDrawProc, SpriteWorld can draw the left and right pieces at the same time, eliminating any seams that might otherwise be visible when drawing them separately.

The DoubleRectDrawProc is called whenever both left and right rects need to be copied to the screen. However, sometimes only one horizontal rect will need to be copied to the screen. When this is the case, the SpriteWorld's screenDrawProc is used. This means that the DoubleRectDrawProc is not a substitute for the normal screenDrawProc; it just compliments it.

Below is a listing of all the DoubleRectDrawProcs that are provided with SpriteWorld. To use any of them, you must add the file BlitPixieDoubleRect.c to your project. This file is in the Sources folder of the SpriteWorld Files folder.

BlitPixie8BitDoubleRectDrawProc An 8-bit DoubleRect blitter.

BP8BitDoubleRectInterlacedDrawProc Same as above, but interlacing.

BlitPixie16BitDoubleRectDrawProc A 16-bit DoubleRect blitter.

BP16BitDoubleRectInterlacedDrawProc Same as above, but interlacing.

By default, a SpriteWorld's doubleRectDrawProc variable is set to NULL. When it is NULL, the SpriteWorld's screenDrawProc is used to draw both rects separately. If, after installing a DoubleRectDrawProc, you wish it removed, simply call this function again, passing a value of NULL as the doubleRectDrawProc parameter.

Note: When running on PPC Macs, custom blitters such as the DoubleRectDrawProcs might be slower than CopyBits, the amount of difference depending on the machine. It's up to you to decide whether you want the extra bit of speed that CopyBits may provide, or a better looking animation with no seams.

SWSetSpriteWorldScrollDelta

This will set the scrolling delta, which is automatically added to the visScrollRect each frame by SWProcessScrollingSpriteWorld.

void SWSetSpriteWorldScrollDelta(SpriteWorldPtr spriteWorldP, short horizDelta, short vertDelta)

`spriteWorldP` A pointer to a SpriteWorld
`horizDelta` The horizontal delta
`vertDelta` The vertical delta

Description:
This function will set the scrolling delta of the SpriteWorld, which is used to control the speed and direction of the scrolling. You might want to call SWSetSpriteWorldScrollDelta at the beginning or during the animation. However, you would normally use a scrolling worldMoveProc to change the scrollDelta by accessing it directly. (See SWSetScrollingWorldMoveProc.)

The scrollDelta values are automatically added to the visScrollRect by SWProcessScrollingSpriteWorld. This is the last step performed by SWProcessScrollingSpriteWorld, done after calling the SpriteWorld's scrolling worldMoveProc, if there is one. When the scrollDelta values are added to the visScrollRect, SpriteWorld makes sure the visScrollRect remains within the boundaries set by SWSetScrollingWorldMoveBounds.

SWMoveVisScrollRect

This will set the position of the visScrollRect - the area that is copied to the screen.

void SWMoveVisScrollRect(SpriteWorldPtr spriteWorldP, short horizPos, short vertPos)

`spriteWorldP`	A pointer to a SpriteWorld
`horizPos`	The horizontal location of the left side of the visScrollRect
`vertPos`	The vertical location of the top of the visScrollRect

Description:

This function will move the visScrollRect to the specified location, so the top of visScrollRect will be located at vertPos and the left side of visScrollRect will be located at horizPos. This function also makes sure that the visScrollRect stays within its bounds. If you try to move it to a location outside of its bounds, it will be moved as close to the requested position as possible, while still remaining within its boundaries.

You would generally call SWMoveVisScrollRect only before the animation starts, and then use the ScrollingWorldMoveProc to control the scrolling after that, although SWMoveVisScrollRect might be useful every now and then, such as if the Sprite the scrolling follows was suddenly transported to a different spot in the scrolling world.

See Also:

SWOffsetVisScrollRect

SWSetScrollingWorldMoveProc

SWOffsetVisScrollRect

This will offset the visScrollRect

void SWOffsetVisScrollRect(SpriteWorldPtr spriteWorldP, short horizOffset, short vertOffset)

`spriteWorldP`	A pointer to a SpriteWorld
`horizOffset`	The horizontal offset
`vertOffset`	The vertical offset

Description:

This function will offset the visScrollRect the distance specified by horizOffset and vertOffset. This function also makes sure that the visScrollRect stays within its bounds. If you try to offset it to a location outside of its bounds, it will be moved as close to the requested position as possible, while still remaining within its boundaries.

You would generally use a ScrollingWorldMoveProc and the scrollDelta to control the scrolling, although SWOffsetVisScrollRect might be handy in some situations.

See Also:
SWMoveVisScrollRect
SWSetScrollingWorldMoveProc

Scrolling Function Prototypes

You might find it useful to print out this section of the documentation, or to save it as a separate document, for use as a “quick reference guide”.

```
void SWUpdateScrollingWindow(SpriteWorldPtr spriteWorldP)

void SWUpdateScrollingSpriteWorld(SpriteWorldPtr spriteWorldP,
  Boolean updateWindow)

void SWProcessScrollingSpriteWorld(SpriteWorldPtr spriteWorldP)

void SWAnimateScrollingSpriteWorld(SpriteWorldPtr spriteWorldP)

void SWFastAnimateScrollingSpriteWorld(SpriteWorldPtr spriteWorldP)

void SWSetScrollingWorldMoveBounds(SpriteWorldPtr spriteWorldP,                    Rect*
scrollRectMoveBounds)
void SWSetScrollingWorldMoveProc(SpriteWorldPtr spriteWorldP,
        WorldMoveProcPtr worldMoveProcP,   SpritePtr followSpriteP)
void SWSetSpriteWorldScrollDelta(SpriteWorldPtr spriteWorldP,                      short
horizDelta,  short vertDelta)
void SWSetDoubleRectDrawProc(SpriteWorldPtr spriteWorldP,
        DoubleDrawProcPtr doubleRectDrawProc);
void SWMoveVisScrollRect(SpriteWorldPtr spriteWorldP,    short horizPos,           short
vertPos)
void SWOffsetVisScrollRect(SpriteWorldPtr spriteWorldP,   short horizOffset,       short
vertOffset)
```