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 too, 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.

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. However, these seams are barely noticeable on the faster Macs (such as 68040’s and higher), if noticeable at all. However, you should understand how the scrolling routines work, in case a user asks you why there are “cracks” in the animation of your game.

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 is 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.

A note about idle sprites: I have bent over backwards to continue SpriteWorld's tradition of not redrawing idle sprites each frame. However, I'm not sure this is such a good idea for scrolling animations, and I may change this in the future, so that all sprites are redrawn each frame, whether idle or not. One of the reasons for this is that unlike normal SpriteWorlds, not all idle sprites will be visible on the screen all of the time, but SpriteWorld still has to cycle through each Sprite anyway to check to see if it is visible and needs to be drawn. Adding a lot of idle sprites to an animation could slow SpriteWorld down simply because it has to cycle through so many sprites each frame. And now with the addition of the tiling routines, you can often use tiles where you would otherwise use idle sprites, if their purpose is only to create a more interesting background. SpriteWorld also has to do a lot more checking when scrolling to see if the idle sprites need redrawing than it does during a non-scrolling animation. So SpriteWorld may be wasting more time by trying to avoid redrawing idle sprites that it is saving. And unless I'm overlooking something, most scrolling games would have hardly any idle sprites, if any at all. What do you think?

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. 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 Vern_Jensen@lamg.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, please let me know if you make a game, however great or small, 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. This file is the only SpriteWorld source file that is optional. If you do not use scrolling, then you do not need to include this file with your project. If you are using the SpriteWorld libraries instead of including individual source files in your project, then you need to include both libraries. The only file in the second library is "Scrolling.c". SpriteWorld was split up into two segments because if all

the source files were included in one segment, it would be larger than the 32k limit for 68k segments.

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, 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 another variable which is also part of the SpriteWorldRec: scrollDelta. The values in ScrollDelta.h and ScrollDelta.v are automatically added to the visScrollRect after the ScrollingWorldMoveProc has been called. This allows your ScrollingWorldMoveProc to change the values in scrollDelta and have them reflected in the very next frame of the animation. You can change the scrollDelta either from your ScrollingWorldMoveProc, or by calling SWSetSpriteWorldScrollDelta.

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, updateWindowMode)

SWProcessScrollingSpriteWorld(spriteWorldP)

SWAnimateScrollingSpriteWorld(spriteWorldP)

SWSetScrollingWorldMoveBounds(spriteWorldP, &scrollRectMoveBounds)

SWSetScrollingWorldMoveProc(spriteWorldP, MyScrollingWorldMoveProc, followSpriteP)

SWSetSpriteWorldScrollDelta(spriteWorldP, horizDelta, vertDelta)

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 would usually 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.

Use the updateWindow parameter to control whether the work frame is copied to the window. You would usually pass true as this paramater, 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 moves the visScrollRect according to the SpriteWorld's scrollDelta.

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 visScrollRect to the screen. You would typically call this right after calling SWProcessScrollingSpriteWorld.

A note about idle sprites:
In this version, idle sprites are not redrawn each frame in an effort to save time. However, doing this in a scrolling animation is much harder than it is in a non-scrolling animation, so more testing has to be done by SpriteWorld to determine whether the idle sprites need redrawing or not. Because of this, I'm not sure whether this special code to avoid redrawing idle sprites actually saves time or slows things down. I may decide to rip out this code later so idle sprites are redrawn each frame just like active sprites.

Because of this, I would suggest that you do not make scrolling games that rely on SpriteWorld's ability to avoid redrawing idle sprites. You should use tiles instead whenever possible. You can have tiles that are "above" sprites, so that the sprites appear to move underneath a tile or a part of a tile. This would be a much better way to make your sprites move "behind" something that does not move, such as a tree, since it is a way of adding depth to your animation without adding lots of idle sprites which have to be processed each frame, which could slow things down. See the tiling documentation for information on how to have tiles that are above sprites.

See Also:

SWAnimateSpriteWorld
SWProcessScrollingSpriteWorld

SWSetScrollingWorldMoveBounds

This will set the boundary of the scrolling area.

```
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->numTileMapCols * spriteWorldP->tileWidth,
  spriteWorldP->numTileMapRows * spriteWorldP->tileHeight);
SWSetScrollingWorldMoveBounds(spriteWorldP, &moveBoundsRect);
```

Note that this example would only work correctly if it was called after SWInitTiling was called, since SWInitTiling sets up the numTileMapCols, numTileMapRows, tileWidth, and tileHeight variables in the SpriteWorldRec.

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:

```
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 vertMoveDelta, which are variables in the SpriteWorldRec. You can also use SWOffsetVisScrollRect and SWMoveVisScrollRect, although these are not normally 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 moveProc 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

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.)

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. SpriteWorld automatically makes sure that the visScrollRect stays within its bounds. If you try to move it to a location outside of its bounds, then SpriteWorld will move it as close to the requested position as possible, while still making sure it stays within its bounds.

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 automatically makes sure that the visScrollRect stays within its bounds. If you try to offset it to a location outside of its bounds, then SpriteWorld will move it as close to the requested position as possible, while still making sure it stays within its bounds.

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