Technote 1182

NewGWorlds in VRAM and AGP Memory

By Geoff Stahl and Mike Marinkovich
Apple Worldwide Developer Technical Support

CONTENTS

T Technote describes the changes in `NewGWorldNewGWorld` with the release of Mac OS 9.

The `NewGWorldNewGWorld` routine now supports allocation of offscreen `GWorld`'s in AGP memory and VRAM. This allows the application programmer much more flexibility in deciding how to allocate their off-screen images. It also adds more complexity and can, if used incorrectly, result in significantly poorer application performance.

This Technote describes the new selectors, covers their basic use, then goes on to illustrate some of the basic problems associated with their use. Finally, the note discusses basic performance figures from a sample implementation.

The New NewGWorldNewGWorld ()

NewGWorldNewGWorld

Use the `NewGWorldNewGWorld` function to create an offscreen graphics world.

```
QDErr NewGWorldNewGWorld     (GWorldPtr   *   offscreenGWorld,
                              short            pixelDepth,
                              const Rect   *   boundsRect,
                              CTabHandle       cTable,    /* can be NULL */
                              GDHandle         aGDevice,  /* can be NULL */
                              GWorldFlags      flags)
```

`offscreenGWorld`

> `offscreenGWorld` is a pointer to the offscreen graphics world created by this routine.

`pixelDepth`

> `pixelDepth` is the pixel depth of the offscreen world; possible depths are 1, 2, 4, 8, 16, and 32 bits per pixel. The `NewGWorldNewGWorld` function uses the pixel depth of

the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you specify zero in this parameter, `NewGWorldNewGWorld` also uses the `GDevice` record from this device instead of creating a new `GDevice` record for the offscreen world. If you use `NewGWorldNewGWorld` on a computer that supports only basic QuickDraw, you may specify only zero or one in this parameter.

`boundsRect`

`boundsRect` is the boundary rectangle and port rectangle for the offscreen pixel map. This becomes the boundary rectangle for the `GDevice` record, if `NewGWorldNewGWorld` creates one. If you specify zero in the `pixelDepth` parameter, `NewGWorldNewGWorld` interprets the boundaries in global coordinates that it uses to determine which screens intersect the rectangle. (`NewGWorldNewGWorld` then uses the pixel depth, color table, and `GDevice` record from the screen with the greatest pixel depth from among all screens whose boundary rectangles intersect this rectangle.) Typically, your application supplies this parameter with the port rectangle for the onscreen window into which your application will copy the pixel image from this offscreen world.

`cTable`

`cTable` is handle to a ColorTable record. If you pass `NILNULL` in this parameter, `NewGWorldNewGWorld` uses the default color table for the pixel depth that you specify in the `pixelDepth` parameter. If you set the `pixelDepth` parameter to 0, `NewGWorldNewGWorld` ignores the `cTable` parameter, and instead copies and uses the color table of the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. If you use `NewGWorldNewGWorld` on a computer that supports only basic QuickDraw, you may specify only `NILNULL` in this parameter.

`aGDevice`

`aGDevice` is a handle to a `GDevice` record that is used in only two cases. First, when you specify the `noNewDevice` flag in the `flags` parameter, in which case `NewGWorldNewGWorld` attaches this `GDevice` record to the new offscreen graphics world. Second, when you specify `useDistantHdwrMem` and/or `useLocalHdwrMem` flags in the `flags` parameter, in which case `NewGWorld` uses this `GDevice`'s VRAM or AGP memory to store the `GWorld`. If you set the `pixelDepth` parameter to zero, or if you do not set the `noNewDevice`, noNewDevice flag,`useDistantHdwrMem`, and/or `useLocalHdwrMem` flag(s), `NewGWorldNewGWorld` ignores the `aGDevice` parameter, so you should set it to `NILNULL`. If you set the `pixelDepth` parameter to zero, `NewGWorldNewGWorld` uses the `GDevice` record for the graphics device with the greatest pixel depth among all graphics devices whose boundary rectangles intersect the rectangle that you specify in the `boundsRect` parameter. You should pass `NILNULL` in this parameter if the computer supports only basic QuickDraw. Generally, your application should never create `GDevice` records for offscreen graphics worlds. Lastly, if you set `useDistantHdwrMem` and/or `useLocalHdwrMem` flags you should always specify a `GDevice`, otherwise the behavior and device associated with the `GWorld`, is indeterminate.

`flags`

`flags` describes options available to your application. You can set almost any combination of the flags `pixPurge`, `noNewDevice`, `useTempMem`, `keepLocal`, `useDistantHdwrMem`, and `useLocalHdwrMem`. If you don't wish to use any of these flags, pass 0 in this parameter, in which case you get the default behavior for `NewGWorldNewGWorld`—that is, it creates an offscreen graphics world where the base address for the offscreen pixel image is unpurgeable, it uses an existing `GDevice` record (if you pass 0 in the depth parameter) or creates a new `GDevice` record, it uses memory in your application heap, and it allows graphics accelerators to cache the offscreen pixel image. You should not use `keepLocal` with either `useDistantHdwrMem` or `useLocalHdwrMem`, the results are in determinate. The available flags are described here:

```
enum
{
     pixPurge           = 1L << pixPurgeBit,
     noNewDevice        = 1L << noNewDeviceBit,
     useTempMem         = 1L << useTempMemBit,
     keepLocal          = 1L << keepLocalBit,
     useDistantHdwrMem  = 1L << useDistantHdwrMemBit,
     useLocalHdwrMem    = 1L << useLocalHdwrMemBit,
};
typedef unsigned longGWorldFlags;
```

`pixPurge`

Makes base address for offscreen pixel image purgeable.

`noNewDevice`

Stops the creation of an offscreen `GDevice` record.

`useTempMem`

Create base address for offscreen pixel image in temporary memory.

`keepLocal`

Keeps offscreen pixel image in main memory where it cannot be cached to a graphics accelerator card.

`useDistantHdwrMem`

Attempts to create the offscreen pixel image in VRAM.

`useLocalHdwrMem`

Attempts to create the offscreen pixel image in AGP memory.

DESCRIPTION

The `NewGWorldNewGWorld` function creates an offscreen graphics world with the pixel depth you specify in the `pixelDepth` parameter, the boundary rectangle you specify in the `boundsRect` parameter, the color table you specify in the `cTable` parameter, and the options you specify in the `flags` parameter. The `NewGWorldNewGWorld` function returns a pointer to the new offscreen graphics world in the `offscreenGWorld` parameter. You use this pointer when referring to this new offscreen world in other routines described in this chapter.

Typically, you pass 0 in the `pixelDepth` parameter, a window’s port rectangle in the `boundsRect` parameter, `NILNULL` in the `cTable` and `aGDevice` parameters, and—in the `flags` parameter—an empty set ([ ]) for Pascal code or 0 for C code. This provides your application with the default behavior of NewGWorldNewGWorld, and it supports computers running basic QuickDraw. This also allows QuickDraw to optimize the CopyBits, CopyMask, and CopyDeepMask procedures when your application copies the image in an offscreen graphics world into an onscreen graphics port.

The NewGWorldNewGWorld function allocates memory for an offscreen graphics port and its pixel map. On computers that support only basic QuickDraw, NewGWorldNewGWorld creates a 1-bit pixel map that your application can manipulate using other relevant routines described in this chapter. Your application can copy this 1-bit pixel map into basic graphics ports.

Unless you specify zero in the pixelDepth parameter--or pass the noNewDevice flag in the flags parameter and supply a GDevice record in the aGDevice parameter--NewGWorldNewGWorld also allocates a new offscreen GDevice record.

When creating an image, your application can use the NewGWorldNewGWorld function to create an offscreen graphics world that is optimized for an image's characteristics--for example, its best pixel depth. After creating the image, your application can then use the CopyBits, CopyMask, or CopyDeepMask procedure to copy that image to an onscreen graphics port. Color QuickDraw automatically renders the image at the best available pixel depth for the screen. Creating an image in an offscreen graphics port and then copying it to the screen in this way prevents the visual choppiness that would otherwise occur if your application were to build a complex image directly onscreen.

The NewGWorldNewGWorld function initializes the offscreen graphics port by calling the OpenCPort function. The NewGWorldNewGWorld function sets the offscreen graphics port's visible region to a rectangular region coincident with its boundary rectangle. The NewGWorldNewGWorld function generates an inverse table with the Color Manager procedure MakeITable, unless one of the GDevice records for the screens has the same color table as the GDevice record for the offscreen world, in which case NewGWorldNewGWorld uses the inverse table from that GDevice record.

The address of the offscreen pixel image is not directly accessible from the PixMap record for the offscreen graphics world. However, you can use the GetPixBaseAddr function (described in *Inside Macintosh*, pages 6-38) to get a pointer to the beginning of the offscreen pixel image.

For purposes of estimating memory use, you can compute the size of the offscreen pixel image by using this formula:

```
rowBytes * (boundsRect.bottom - boundsRect.top)
```

In the flags parameter, you can specify several options that are defined by the GWorldFlags data type. If you don't wish to use any of these options, pass zero here.

If you specify the pixPurge flag, NewGWorldNewGWorld stores the offscreen pixel image in a purgeable block of memory. In this case, before drawing to or from the offscreen pixel image, your application should call the LockPixels function (described in Inside Macintosh: Imaging With QuickDraw) and ensure that it returns TRUE. If LockPixels returns FALSE, the memory for the pixel image has been purged, and your application should either call UpdateGWorld to reallocate it and then reconstruct the pixel image, or draw directly in a window instead of preparing the image in an offscreen graphics world. Never draw to or copy from an offscreen pixel image that has been purged without reallocating its memory and then reconstructing it. If you specify the noNewDevice flag, NewGWorldNewGWorld does not create a new offscreen GDevice record. Instead, it uses the GDevice record that you specify in the aGDevice parameter—and its associated pixel depth and color table—to create the offscreen graphics world. (If you set the pixelDepth parameter to 0, NewGWorldNewGWorld uses the GDevice record for the screen with the greatest pixel depth among all screens whose boundary rectangles intersect the rectangle that you specify in the boundsRect parameter—even if you specify the noNewDevice flag.) The NewGWorldNewGWorld function keeps a reference to the GDevice record for the offscreen graphics world, and the SetGWorld procedure (described in Inside Macintosh: Imaging With QuickDraw) uses that record to set the current graphics device. If you set the useTempMem flag, NewGWorldNewGWorld creates the base address for an offscreen pixel image in temporary memory. You generally would not use this flag, because you should use temporary memory only for fleeting purposes and only with the AllowPurgePixels procedure (described in Inside Macintosh: Imaging With QuickDraw). If you specify the keepLocal flag, your offscreen pixel image is kept in Macintosh main memory and is not cached to a graphics accelerator card. Use this flag carefully, as it negates the advantages provided by any graphics acceleration card that might be present. Specifying useDistantHdwrMem and/or useLocalHdwrMem attempts to allocate the offscreen pixel image in VRAM or AGP memory respectively. If both flags are specified NewGWorldNewGWorld will first attempt to allocate in AGP memory, if that fails, it will attempt to allocate in VRAM. When using useDistantHdwrMem, useLocalHdwrMem or both and NewGWorldNewGWorld cannot allocate the offscreen pixel image in the requested area of memory NewGWorldNewGWorld will fail and return a memFullErr error code.

As its function result, NewGWorldNewGWorld returns one of four result codes enumerated below. noErr will always be returned from successful calls to NewGWorldNewGWorld any other returns values should be considered a failure to allocate the requested offscreen pixel image.

SPECIAL CONSIDERATIONS

If you supply a handle to a ColorTable record in the cTable parameter, NewGWorldNewGWorld makes a copy of the record and stores its handle in the offscreen *PixMap* record. It is your application's responsibility to make sure that the ColorTable record you specify in the cTable parameter is valid for the offscreen graphics port's pixel depth.

If when using NewGWorldNewGWorld you specify a pixel depth, color table, or GDevice record that differs from those used by the window into which you copy your offscreen image, the CopyBits, CopyMask, and CopyDeepMask procedures require extra time to complete. These will likely cause buffers allocated in AGP memory or VRAM to be unable to utilize hardware blitting acceleration, possibly resulting in extremely poor copy performance.

There are two important things to note about GWorld's allocated in VRAM. First, the base address retrieved through GetPixBaseAddr or read directly from the PixMap structure can become invalid anytime memory is allocated in VRAM. This can occur either by explicit allocations, such as calls to NewGWorldNewGWorld, or by implicit ones, such as those associated with the internal texture allocation of OpenGL. The stored pixel images themselves will still be valid but may have been moved in VRAM, thus rendering any stored base addresses invalid. You should never store an image's base address for longer than is necessary and especially never across calls to NewGWorldNewGWorld or texture-creation routines.

Secondly, an offscreen pixel image allocated in VRAM can be purged at system task time by the display driver. This means any time your application yields time such by calling WaitNextEvent or SystemTask you can lose your VRAM GWorld contents. While this happens infrequently, usually associated with display resolution or pixel depth changes you must code for this eventuality. This purge can occur whether or not the GWorld is locked or not. A return value of false from LockPixels, a NULL return value from GetPixBaseAddr or NULL in the baseAddr field of the PixMap mean that the pixel image has been purged. To reallocate it you can either call UpdateGWorld or Dispose your current GWorld through DisposeGWorld and reallocate it via NewGWorldNewGWorld. Either way you must then rebuild the pixel image.

To use a custom color table in an offscreen graphics world, you need to create the associated offscreen GDevice record, because Color QuickDraw needs its inverse table.

Currently, NewGWorld does not do exhaustive error checking on the combination of parameters you supply to it. It assumes these parameters make sense. This is especially true when working with the new flag parameters. For example, you could legally pass keepLocal, useDistantHdwrMem, and useLocalHdwrMem in flags and NULL in aGDevice, although this makes no sense and the behavior is undefined. You must ensure the flags and other parameters supplied to `NewGWorld` actually work together and not rely on the OS checking these kinds of errors.

The `NewGWorldNewGWorld` function may move or purge memory blocks in the application heap. Your application should not call this function at interrupt time.

RESULT CODES

`noErr`	0	No error
`paramErr`	-50	Illegal parameter
`memFullErr`	-108	Out of memory error
`cDepthErr`	-157	Invalid pixel depth

See also Inside Macintosh: Imaging With QuickDraw.

Listing 6-1 on page 6-5 and Listing 6-2 on page 6-10 illustrate how to use `NewGWorldNewGWorld` to create offscreen graphics worlds.

If your application needs to change the pixel depth, boundary rectangle, or color table for an offscreen graphics world, use the `UpdateGWorld` function, described on pages 6-23 of *Inside Macintosh.*

Using the New `NewGWorldNewGWorld ()`

The basics of using the new `NewGWorldNewGWorld` remain the same. The two additional flags `useDistantHdwrMem` and `useLocalHdwrMem` allow the user to control where the offscreen pixel image is allocated. If you use either `useDistantHdwrMem` or `useLocalHdwrMem` by themselves, `NewGWorldNewGWorld` will attempt to allocate the image on the device specified in `aGDevice` in only VRAM or AGP memory respectively. If this allocation fails, `NewGWorldNewGWorld` will fail and return a `memFullErr` error. If both `useDistantHdwrMem` and `useLocalHdwrMem` are specified, `NewGWorldNewGWorld` will first attempt to allocate in VRAM first, then AGP memory of the device specified in `aGDevice`. If both fail, `NewGWorldNewGWorld` will fail and return a `memFullErr` error. `aGDevice` should never be `NULL` when allocating in AGP memory or VRAM since the device used for the allocation will be indeterminate, which is almost never what the developer intended.

`useDistantHdwrMem` is useful to allocate pixel images that are set once (or few) and used many times. It is relatively slow to write to a VRAM pixel image from system memory, but it is very quick to do a hardware copy from VRAM to VRAM or VRAM to the screen. Since there is currently no mechanism to determine if a copy operation will use hardware acceleration, it is recommended that all pixel images allocated in VRAM be the same bit depth and have the same color table as the screen. Additionally, you should use simple copy operations that do not use masks or resizing to maximize the possibility of a specific copy being accelerated.

`useLocalHdwrMem` attempts to allocate the pixel image in AGP memory. If the system does not have an AGP graphics system or `NewGWorldNewGWorld` is unable to allocate the pixel image, the allocation will fail, returning a `memFullErr` error. Since AGP allocations are in system memory, these do not suffer from the same problems associated with pixel images allocated in VRAM. AGP memory can however have some limitations, such as being uncacheable, that make it slightly slower than regular system memory in copies to and other system memory. While this is minor, developers should be aware of the slight performance degradation.

It is very important to understand where allocations happen and the general caveats (described in the preceding paragraphs) associated with allocations in other than standard system memory. The key to improving an application's performance using offscreen `GWorld`'s with `useDistantHdwrMem` or useLocalHdwrMem flags is identifying which pixel images are used often and can be transferred with hardware accelerated copies. Images used very often and modified infrequently can be placed in VRAM to optimize their copy performance. For general purposes, you can use AGP memory. Since the amount of AGP memory is limited and other graphics services, such as OpenGL, use it, care should be used to allocate more frequently used images first. In addition, note that AGP memory is not swapped out to disk by the virtual memory system and the amount available may vary with the amount of physical memory installed on a system.

To recap, if a copy does not use hardware acceleration, performance from AGP memory to the screen can be expected at best to be equal to system memory. In the same non-accelerated case performance from VRAM to the screen will be significantly slower than from system memory. So allocate you images carefully and use AGP memory and VRAM judiciously.

As a final note, when NewGWorld allocates memory outside of your application heap (i.e., in AGP memory space or VRAM) it is extremely important that you properly dispose of that memory with DisposeGWorld prior exiting your application. Failure to do so will leak memory, making either the VRAM or AGP memory unavailable for future use. In many cases, this leaked memory will only be recovered at restart.

One more reminder: when developing your implementation, ensure you note the above special considerations for GWorld's allocated in VRAM. It is critical that you handle moved and purged pixel images for GWorld's in VRAM correctly, ensuring you will not display garbage or access invalid memory when trying to use VRAM GWorld's.

A Sample Implementation

Implementing this is actually very easy. Below is some simple code that will assist you. First, we will test for the availability of the new flags. In this case, we just need to look at the system version (since there are no specific selectors for the new version of NewGWorldNewGWorld). If the Mac OS in use is later than 8.6, we can expect useDistantHdwrMem and useLocalHdwrMem to be available.

```
Boolean gNewNewGWorld = false;
long versionSystem;
// this will only work with Mac OS later than 8.6
Gestalt (’sysv', &versionSystem);
if (0x00000860 < (versionSystem & 0x00000FFFF))
     gNewNewGWorld = true; // system is greater than version 8.6
```

Next, we want to encapsulate the functions required to allocate and reallocate our GWorld. We can use the same logic for both, checking the GWorldPtr, then checking the pixel image's baseAddr and finally checking to see if the window's GDevice is still the same as the offscreen's GDevice. When allocating the GWorld we take the location input parameter and use this to determine in which memory space to allocate (VRAM, AGP memory, or application heap). If an allocation fails, we fall through to the next type. Obviously, you can modify this behavior to suit your needs. The function returns true if the GWorld is allocated or reallocated and false if the existing GWorld is valid. This is shown in the following listing.

```
Boolean BuildOffscreen (GWorldPtr * ppGWorld, WindowPtr pWindow,
                        short * plocation)
{
     GDHandle hgdWindow = NULL;
     Boolean fMustRebuild = false;

if (NULL == *ppGWorld) // if GWorld passed in is not allocated
     fMustRebuild = true;
else
{
     PixMapHandle hPixmap = GetGWorldPixMap (*ppGWorld);
     // if pixmap handle is NULL or pixmap base address is NULL
     if ((NULL == hPixmap) || (!GetPixBaseAddr (hPixmap)))
         fMustRebuild = true;
     // if GWorld not on same device as window
         else if (GetGWorldDevice(*ppGWorld) != GetWindowDevice (pWindow))
         fMustRebuild = true;
}

if (fMustRebuild) // must rebuild
{
     // window pixel depth
     short wPixDepth = (**((CGrafPtr)pWindow)->portPixMap).pixelSize;
     GDHandle hgdWindow = GetWindowDevice (pWindow);// window GDevice
     if (NULL != *ppGWorld) // if we have an allocated GWorld
     {
         DisposeGWorld (*ppGWorld);// dump our current GWorld
         *ppGWorld = NULL;
     }
     switch (*plocation) // where to we want to put it
     {
         case kInVRAM:
             if (noErr == NewGWorld (ppGWorld, wPixDepth, &pWindow->portRect,
                                     NULL, hgdWindow,
                                     noNewDevice | useDistantHdwrMem))
              break;
         // we failed with VRAM, signal that and drop to AGP
         SysBeep (30);
         *plocation = kInAGP;
     case kInAGP:
         if (noErr == NewGWorld (ppGWorld, wPixDepth, &pWindow->portRect,
                                 NULL, hgdWindow,
                                 noNewDevice | useLocalHdwrMem))
              break;
         // we failed with AGP, signal that and drop to system memory
         SysBeep (30);
         *plocation = kInSystem;
     case kInSystem:
     default:
         if (noErr != NewGWorld (ppGWorld, wPixDepth,
                                 &pWindow->portRect, NULL, hgdWindow,
                                 keepLocal | noNewDevice))

          {
              // we failed with system thus, we can’t allocate our GWorld,
              // signal that, indicate no location and drop to debugger
              SysBeep (30);
              *plocation = kNoWhere;
              DebugStr ("\pUnable to allocate off screen image");
              return false; // nothing was allocated
         }
         *plocation = kInSystem;
    }
     return true; // we rebuilt our GWorld
   }
   return false; // everything is okay
}
```

This previous function uses standard Macintosh Toolbox functions except the call to `GetWindowDevice` that determines the `GDevice` on which the majority of the window resides. Note that it is up to the individual application developer to handle the case where windows span multiple devices. `GetWindowDevice` is listed below.

```
GDHandle GetWindowDevice (WindowPtr pWindow)
{
     Rect rectWind, rectSect;
     short wFrameHeight, wTitleHeight;
     long greatestArea, sectArea;
     GDHandle hgdNthDevice, hgdZoomOnThisDevice;

     rectWind = pWindow->portRect;
     LocalToGlobal ((Point*)& rectWind.top); // convert to global coordinates
     LocalToGlobal ((Point*)& rectWind.bottom);
     // calculate height of window’s title bar
     wFrameHeight = rectWind.left - 1 –
                   (**(((WindowPeek)pWindow)->strucRgn)).rgnBBox.left;
     wTitleHeight = rectWind.top - 1 –
                   (**(((WindowPeek)pWindow)->strucRgn)).rgnBBox.top;
     rectWind.top = rectWind.top - wTitleHeight;
     hgdNthDevice = GetDeviceList ();
     greatestArea = 0; // initialize to 0
     // check window against all gdRects in GDevice list and remember
     //  which gdRect contains largest area of window}
     while (hgdNthDevice)
     {
         if (TestDeviceAttribute (hgdNthDevice, screenDevice))
              if (TestDeviceAttribute (hgdNthDevice, screenActive))
              {
                   // The SectRect routine calculates the intersection
                   //  of the window rectangle and this GDevice
                   //  rectangle and returns TRUE if the rectangles intersect,
                   //  FALSE if they don’t.
                   SectRect(&rectWind, &(**hgdNthDevice).gdRect, &rectSect);
                   // determine which screen holds greatest window area
                   //  first, calculate area of rectangle on current device
                   sectArea = (long)(rectSect.right - rectSect.left) *
  (rectSect.bottom - rectSect.top);
                   if ( sectArea > greatestArea )
                   {
                        greatestArea = sectArea;// set greatest area so far
                        hgdZoomOnThisDevice = hgdNthDevice;// set zoom device
                   }
                   hgdNthDevice = GetNextDevice(hgdNthDevice);
              }
         } // of while
     return hgdZoomOnThisDevice;
}
```

Once we have the buffer allocated, we just need to fill it and blit it to our window. The process to do this remains unchanged. The following listings demonstrate this. Note, `FillOffscreen` assumes the `GWorldPtr` passed in is valid, while `BlitToWindow` is more general purpose and runs a check on the `GWorld`.

```
// fills offscreen buffer with random bright color

void FillOffscreen (GWorldPtr pGWorld)
{
     GDHandle hGDSave;
     CGrafPtr pCGrafSave;
     Rect rectSource = (pGWorld->portRect);
     RGBColor rgbColor;

     rgbColor.red   = (Random () + 32767) / 2 + 32767;
     rgbColor.green = (Random () + 32767) / 2 + 32767;
     rgbColor.blue  = (Random () + 32767) / 2 + 32767;

     GetGWorld (&pCGrafSave, &hGDSave);
     SetGWorld (pGWorld, NULL);
     if (LockPixels (GetGWorldPixMap (pGWorld)))
     {
         // draw some background
         EraseRect (&rectSource);
         RGBForeColor (&rgbColor);
         PaintRect (&rectSource);
         UnlockPixels (GetGWorldPixMap (pGWorld));
     }
     SetGWorld (pCGrafSave, hGDSave);
}

// checks offscreen and blits it to the front

void BlitToWindow (GWorldPtr pGWorld, WindowPtr pWindow, short * pLocation)
{
    Rect rectDest = ((GrafPtr)pWindow)->portRect;
     Rect rectSource = ((GrafPtr)pWindow)->portRect;
     GrafPtr pCGrafSave;

     // check to ensure we have a valid offscreen and rebuild if required
     if (BuildOffscreen (&pGWorld, pWindow, pLocation))
     FillOffscreen (pGWorld);

     // blit
     GetPort (&pCGrafSave);
     SetPort ((GrafPtr) pWindow);
     if (LockPixels (GetGWorldPixMap (pGWorld)))
     {
         CopyBits (&((GrafPtr)pGWorld)->portBits,
                   &pWindow->portBits,
 &rectSource, &rectDest, srcCopy, NULL);
         UnlockPixels (GetGWorldPixMap (pGWorld));
     }
     SetPort (pCGrafSave);
}
```

Lastly, we need to ensure the memory allocated by `NewGWorld` is disposed of properly. The follow code demonstrates this.

```
// this is VERY important since the GWorld may not be in the application heap
if (pGWorld)
     DisposeGWorld (pGWorld);
pGWorld = NULL;
```

Summary

Using the new `NewGWorld` greatly enhances your options for creating performance-oriented applications. By allocating pixel images in either VRAM or AGP memory space, one can achieve levels of graphics performance previously unavailable. Using these new features though does impose some requirements on the application developer to ensure their code functions properly under all conditions. A checklist to consider when using the new `NewGWorld` is as follows:

- Check system version for availability of `useDistantHdwrMem` and `useLocalHdwrMem` flags.
- Provide a `GDevice` when using `useDistantHdwrMem` and/or `useLocalHdwrMem`.
- Check your return values for errors.
- When retrieving your pixel image's base address check for `NULL`.
- Implement a restoration scheme to handle purged pixel images.
- Do not cache base address across functions that yield time to the system or that could allocate or deallocate VRAM.
- Ensure `GWorlds` are disposed of properly to prevent memory leaks in VRAM or AGP memory.

The flags `useDistantHdwrMem` and `useLocalHdwrMem` provide the developer with more options for handling offscreen graphics but must be used with complete understanding of the additional burdens placed on the application.

Further References

- Apple's Technote web site
- Inside Macintosh: Imaging With QuickDraw; Chapter 6: Offscreen Graphics Worlds

Back to top

Downloadables

Acrobat version of this Note (how many K?)

Back to top

Acknowledgments

Thanks to Tim Carroll, Kent Miller, and Fernando Urbina.

To contact us, please use the Contact Us page.
Updated: 05-October-1999

Technotes | Contents
Previous Technote | Next Technote