SmartErase.pbi

by Stephen Rodriguez.

General.

SmartErase.pbi is a Purebasic source code include file containing a small Windows utility to assist in reducing flicker when dynamically resizing controls in response to the user resizing an application window etc.

This software is written in Purebasic 4.3 and has been tested on Win XP and Vista. It will run only on Windows platforms.

Package, licence and terms of use.

This package contains all the relevant source files, six demo programs and this very short instruction manual.

The software contained within this package is free to use in any project (commercial or otherwise) or as a learning tool. I do, however, assert my moral right to be identified as the creator of this software (except where acknowledgements are given) and thus ask that due acknowledgement is given within any product/creation in which my source code forms a part. Use the software for any purpose whatsoever.

This software is provided on an *as is* basis, with no warranty either given or implied, meaning that I am not liable for any damage caused by its use (or misuse!) nor by damage caused by other programs based on its source code.

To use "SmartErase.pbi" within a Purebasic program.

Simply ensure that the following command resides within your Purebasic source code file before any attempt is made to make use of this utility :

```
XincludeFile "SmartErase.pbi"
```

That's it.

What causes flicker and how can we combat it?

When the user resizes an application window, flicker generally results from the program dynamically resizing controls to keep pace with the changes in the window etc.

In these cases, the vast majority of flicker is caused by controls erasing their contents prior to repainting.

Imagine a container gadget, for example, containing a few buttons. When the container is resized (through code) following the user resizing the main application window, the container will generally erase it's own contents by filling it's client area with some colour or other. When it does this, the child button controls will appear to flicker as they are momentarily blanked out during this repeating operation. This is of course more noticeable when you increase the number of such containers and the number of child controls etc.

Certain kinds of controls (e.g. Purebasic Frame3Dgadgets and PanelGadgets) exacerbate this further through their very nature and the ways in which they operate.

Removing this flicker completely would be a matter for *double-buffering* (off-screen rendering), but with the Windows OS, our options in this regard are very limited indeed.

Sure we can double-buffer our own drawing operations and even our own custom controls, but double-buffering the pre-built common controls is an altogether different proposition.

No, the best we can usually hope for is to be able to take steps to reduce this flicker to far more manageable levels and so the question is generally not one of removing all flicker, but of reducing it, and even this can be a lot of work on occasion.

Strategies for reducing flicker stretch from the sublime to the ridiculous on occasion; from simply setting a few Window's styles (and hoping for the best) to subclassing and 'hooking' individual control's painting and erasing processes. It can be a very involved and time consuming business indeed!

And that is where SmartErase steps in with it's size thirteen hiking boots to kick those flickeringly annoying controls firmly into touch!

It is not very sophisticated (at least nowhere near as sophisticated as my first attempt at this, but which failed to deal with themed buttons and the like!) But it is damn effective in a lot of cases!

What does SmartErase do and how does it work?

First, SmartErase (like the Purebasic SmartWindowRefresh() command – which never seems to work for me anyhow!) is not guaranteed to give good results in any individual situation. It is a simple tool which may or may not help with individual applications or indeed individual windows etc.

What I will say is that in my tests it is proving very effective indeed! There is a little bit of trial and error involved (as you will see from the demo programs) but, if like me you find flickering to be very annoying, then it is well worth a shot - you have nothing to lose!

Myself, I will generally combine this tool with other techniques (generally more specialised) as appropriate depending on the underlying situation etc.

I am finding, for example, that this tool has only a limited success with Purebasic panel-gadgets, which is of no surprise really as nothing else seems to work with these beasties thanks to the themed nature of this swine! Not even the #WS_EX_COMPOSITED style works. However, this tool can certainly reduce flicker even with these gadgets.

So, how exactly does SmartErase work?

Well, first you would generally apply '*SmartErase*' to a parent window/control (e.g. a container gadget) whose child gadgets flicker inordinately when dynamically resized etc. There are occasions when you might apply SmartErase to other controls as well (e.g. Frame3Dgadgets), but we'll come to that later.

Once SmartErase has been applied to a window or control, then all erasing of that window/control is handled by the SmartErase library itself. Its default behaviour is to simply erase only those parts of the window/control which do not house any child controls. That is, all child controls are excluded from the erasing process. [1]

[1] You may think that this is the same as applying the #WS_CLIPCHILDREN window style to the window/control, but this is not the case. However, in combination with this style, SmartErase could prove even more effective in certain circumstances (it could also prove totally ineffective as well!)

You can refine this behaviour by excluding not just the regions containing child controls, but also any regions occupied by any overlapping (non-child) controls etc. Further, you can opt to remove all erasing from the window/control which is useful in cases where the window's client area is completely covered by child controls etc.

There you are; simple, and, some would argue, pretty crude. However, the proof of the pudding is of course in the eating and I would advise you to take a quick peek at the demo programs to judge for yourself just how effective this can be!

Tech note.
I did originally attempt to simply adjust the window/control's update region to remove all child controls and whilst this worked very well in principal, it could not deal with themed buttons and the like and in fact it completely ruined their appearance! Hence my falling back to what is a far more simplistic approach. Still, the simplest ideas are often the best!

Applying *SmartErase* to a window/control.

To apply *SmartErase* to a window or control simply use the SetSmartErase() function which has the following prototype :

Procedure.i ***SetSmartErase***(hWnd [, color [, flag]])

with the following parameters :

hWnd
the Window's handle of the window/control in question (usually a parent control)

color
Optional.

The background colour of the window/control in question.
The default value is *#SmartErase_USESTANDARDWINDOWCOLOR* which will use the default window color.

flag
Optional.

One of the following values :

#SmartErase_NOERASEATALL
Remove all erasing from the window/control. Useful if the window/control is completely covered by child controls etc.

#SmartErase_REMOVESMARTERASE
Remove *SmartErase* from the window/control.

The following values determine whether only the regions occupied by child controls are excluded from the erasing process or, in addition, whether other overlapping controls (not just child controls) are also excluded :

#SmartErase_MAXERASELEVEL
All overlapping regions are excluded.

#SmartErase_ERASELEVEL0

Only regions occupied by child controls are excluded.
This is the default setting.

#SmartErase_ERASELEVEL1

All regions occupied by child controls and all Childs of the most immediate parent are excluded (siblings).

#SmartErase_ERASELEVEL2

As for *#SmartErase_ERASELEVEL1* but with the addition of all Childs of the grand-parent etc.

etc. up to *#SmartErase_ERASELEVEL6*.

NOTE that SmartErase does not count any window/control whose client area completely shadows that of our window/control when looking for overlapping controls etc.

A non-zero return from this function means that the operation completed successfully.

Removing individual child controls from the exclusion.

Now that you understand how SmartErase works, you may find that there are occasions when, having applied SmartErase to a parent window, for example, that certain child controls are misbehaving because they are being excluded from their parents erasing process. This might effect toolbars in particular.

For this you can opt to remove any such child-controls from the exclusion process by simply setting a window property as follows :

```
SetProp_(ChildhWnd, #SmartErase_IGNORECHILD, 1)
```

This means that, having applied SmartErase to the parent of *ChildhWnd*, the region occupied by *ChildhWnd* will be included (not removed) when the parent is erased etc. This may mean of course that this region will appear to flicker, but then you would have good reason for doing this anyhow!

As I say, this may be useful in particular for toolbars.

You will of course have to remove this window property yourself before destroying the control as Windows does not do this automatically. SmartErase will also not remove this property automatically as doing so would require SmartErase to subclass the control in question which is, to be honest, overkill in such circumstances.

To remove the property simply issue the command :

```
RemoveProp_(ChildhWnd, #SmartErase_IGNORECHILD)
```

Notes on using SmartErase.

1. Using SmartErase to excess can slow down an application's responsiveness in terms of it's GUI etc.

   In particular, any GUI containing say hundreds of controls would probably have to use SmartErase sparingly; say on a couple of controls only.

2. The default flag for use with the SetSmartErase() function, namely *#SmartErase_ERASELEVEL0* will be the most commonly used. All other values from *#SmartErase_MAXERASELEVEL* onwards will of course slow things down accordingly as the library will then have to search through more controls etc.

   Thus far I have had to use one of these constants once only with a Frame3Dgadget (which are poorly behaved at the best of times!)

3. Applying *SmartErase* to a parent window/control (e.g. a container gadget) whose child gadgets flicker inordinately when dynamically resized etc. is a bit of a hit and miss affair. For example, with the Frame3Dgadget demo program I found myself having to apply *SmartErase* to the Frame3D itself (as well as it's parent window!)

   Just be prepared for some trial and error!

4. I reiterate that this library may or may not alleviate any flickering problems related to any individual application. You will ideally need to combine this utility with other techniques as well.

Stephen Rodriguez.