

The CSprite Class

Overview

The CSprite class provides an easy to use interface for animating sprites.

Supported features are:

Transparency:

This works with any given color as transparent color (not only black).

Animated sprites:

A Sprite can have more then one fixed image. You may assign as many images as you wish (and memory can hold). This images are iterated every n-th moving action.

Ownerdraw sprites:

A sprites image is not a fixed one but can be changed with standard GDI functions during every redraw action of the sprite.

Mouse dragging:

The CSprite class owns some helper functions to make implementing mouse dragging a little bit easier.

Flicker-free animation:

Moving sprites and calculating transparency occurs in an off-screen dc to prevent from flickering

The interface of the CSprite class

CSprite(int nID = -1, BOOL bVisible = TRUE, BOOL bDraggable = FALSE, BOOL bOwnerDraw = FALSE)

int nID	the id of this sprite; it's the users responsibility to pass a unique id to this sprite if it's managed by a CSpriteMgr object else undefined behavior of the sprite would take place
BOOL bVisible	the initial state of the sprite. This value is only used if the sprite is managed by a CSpriteMgr object.
BOOL bDraggable	if this boolean is set TRUE the sprite could be dragged with the mouse. this value is only used if the sprite is managed by a CSpriteMgr object
BOOL bOwnerDraw	this value specifies wheter the sprite has fixed number of images or is an ownerdraw sprite.

Constructor of the CSprite object. One of the InitSpriteInfo functions has to be called after the constructor. Which of the two functions has to be called depends on bOwnerDraw.

virtual void **InitSpriteInfo**(CDC* pDC, CBitmap& newBitmap, int nLoops, BOOL bTransparent, COLORREF transparent, CPoint& spritePos, BOOL bAutoSkip, CSize* pStretchSize = NULL)

CDC* pDC	pointer to the windows client device context
CBitmap& newBitmap	the first sprite image
int nLoops	number of movements this image should be shown; if there is only one image for this sprite it should be set to 1
BOOL bTransparent	boolean value determing wheter the image uses a transparent color or not
COLORREF transparent	the RGB value of the transparent color; if bTransparent is FALSE this value is ignored
CPoint& spritePos	the initial sprite position
BOOL bAutoSkip	this value determines wheter the sprites frame counter is incremented automatically on every movement of the sprite; if there is only one image for this sprite it should be set to FALSE
CSize* pStretchSize	if the sprite should have an diferent size then the bitmap, it could be passed as pointer to a CSize object; if not, the pointer should be

NULL; the given bitmap and any additional bitmaps are stretched to this size.

This function initializes a sprite with a given image at a specified position. The sprite is not displayed due to this function call. This function should be used for fixed-images sprites only, if called for an ownerdraw sprite the debug version of the library asserts.

virtual void **AddSpriteImage**(CDC* pDC, CBitmap& newBitmap, int nLoops)

CDC* pDC	pointer to the windows client device context
CBitmap& newBitmap	an additional sprite image
int nLoops	number of movements this image should be shown

This function adds another image to the sprite. This function should be used for fixed-images sprites only, if called for an ownerdraw sprite the debug version of the library asserts.

virtual void **InitSpriteInfo**(CDC* pDC, CRect& spriteRect)

CDC* pDC	pointer to the windows client device context
CRect& spriteRect	the initial position of the sprite; the CRect object implicitly defines the size of the image buffer to be prepared.

This function initializes a owner draw sprite, if called for fixed-images sprites the debug version of the library asserts. The sprite is not displayed due to this function call but the first call to OnRenderSprite is done to retrieve the first image to display.

virtual BOOL **OnRenderSprite**(CDC* pDC, CRect positionInfo, COLORREF& transparent, int nDrawingType)

CDC* pDC	pointer to memory device context used as buffer for the image
CRect positionInfo	CRect object containing the position of the sprite
COLORREF& transparent	the RGB value of the transparent color, should be set by the function
int nDrawingType	this is a hint for what drawing action resulted in the call to this function. it is one of the following constants:
DT_SHOWSPRITE	call to CSpriteMgr::ShowSprite
DT_MOVESPRITE	call to CSpriteMgr::MoveSprite or CSprite::MoveSprite
DT_INITSPRITE	call to CSprite::InitSpriteInfo
DT_KEEPIIMAGE	this is an internal constant that prevents the function from being called, so this value is never passed to the function
DT_ONPAINT	call to CSpriteMgr::OnPaint

To create an ownerdraw sprite this member function must be overwritten to render the sprite's image. The default implementation asserts. Do not call CSprite::OnRenderSprite out of your function. Ownerdraw sprites call this function every time a drawing action occurs. The function should return a boolean specifying if the rendered image uses a transparent color. If the function returns TRUE it should also change the transparent-variable to the RGB value of the transparent color used.

virtual void **DrawSprite**(CDC* pDC, CSize offset = CSize(0, 0))

CDC* pDC	pointer to the windows client device context
CSize offset	offset of the bitmap position within the given dc

This function renders the sprite to the given dc. If used on a buffer dc with extensions other than the client dc it may be useful to specify an offset to the bitmap position. Usually you will call `CSprite::StoreBkg` first to save the sprites background.

virtual void **StoreBkg**(CDC* pDC, CSize offset = CSize(0, 0))

CDC* pDC pointer to the windows client device context
CSize offset offset of the bitmap position within the given dc

This function stores the background of the sprite. The background is copied from the sprites current position. If used on a buffer dc with extensions other than the client dc it may be useful to specify an offset to the bitmap position.

virtual void **RestoreBkg**(CDC *pDC, CSize offset = CSize(0, 0))

CDC* pDC pointer to the windows client device context
CSize offset offset of the bitmap position within the given dc

This function restores the stored background in the given dc. If used on a buffer dc with extensions other than the client dc it may be useful to specify an offset to the bitmap position.

virtual BOOL **IntersectSprite**(CRect& rect)

CRect& rect rectangle to check for

This function returns TRUE if the given rectangle intersects with the sprites position else it returns FALSE.

virtual BOOL **HitTest**(CPoint& pos, BOOL bExact = FALSE, CDC* pDC = NULL)

CPoint& pos point to check for
BOOL bExact boolean specifying whether the sprites image transparency should be involved or not
CDC* pDC if bExact is TRUE this has to be a pointer to the windows client dc else it is ignored

This function checks if the given point hits the sprites image. If the given point lies within the sprites rectangle the function returns TRUE else it will return FALSE. If bExact is TRUE and the given point hits the sprite on a transparent pixel the function will return FALSE.

virtual void **MoveSprite**(CDC* pDC, CPoint &pos)

CDC* pDC pointer to the windows client dc
CPoint &pos new sprite position

This function moves the sprite to the given position restoring the previous positions background.
Important: This function should only be used if there is no other sprite intersecting with the previous or the new sprite position. A second limitation is that this function only works correctly if the client area is completely visible. To work around this limitations let the sprite be managed by a CSpriteMgr object and use CSpriteMgr::MoveSprite instead.

virtual void **AnimateSprite**(CDC *pDC, CPoint& newPos, int nStepFactor)

CDC *pDC	pointer to the windows client dc
CPoint& newPos	end position of the animation
int nStepFactor	number of animation steps

This function animates the sprite from the current position to the given position drawing nStepFactor frames. **Important: This function should only be used if there is no other sprite intersecting with one of the positions the sprite is animated through. A second limitation is that this function only works correctly if the client area is completely visible. To work around this limitations let the sprite be managed by a CSpriteMgr object and use CSpriteMgr::AnimateSprite instead.**

virtual void **BeginSpriteDragging**(CWnd* pWnd , CPoint &pos)

CWnd* pWnd	pointer to the window containing the sprite
CPoint &pos	current mouse position

This function starts a mouse dragging action by setting intern variables and calling SetCapture to make sure the given window will receive all the mouse messages until EndSpriteDragging is called. **Important: This function does not check if the mouse position hits the sprite. To check this out use CSprite::HitTest.**

Usually you will implement mouse dragging like this:

```
void CMyWindow::OnLButtonDown(..., CPoint point)
{
    if (pSprite->HitTest(point, ....)
    {
        pSprite->BeginSpriteDragging(this, point);
        m_bDragging = TRUE;
    }
    CWnd::OnLButtonDown(..., CPoint point);
}

void CMyWindow::OnMouseMove(..., CPoint point)
{
    if(m_bDragging)
    {
        CClientDC dc(this);
        pSprite->MoveSprite(&dc, point);
    }
    CWnd::OnMouseMove(..., CPoint point);
}

void CMyWindow::OnLButtonUp(..., CPoint point)
{
    if(m_bDragging)
    {
        CClientDC dc(this);
        pSprite->EndSpriteDragging(&dc, point);
    }
    CWnd::OnLButtonUp(..., CPoint point);
}
```

virtual void **EndSpriteDragging**(CDC* pDC, CPoint& newPos)

CDC* pDC	pointer to the windows client dc
CPoint& newPos	current mouse position

This function positions the sprite and releases the mouse capturing.

virtual void **SkipFrames**(int nOffset = 1)

int nOffset number of frames to skip

Call this function to increment or decrement the sprites frame counter. The function does not redraw the sprite. The number of frames a sprites image is displayd is given by the parameters of InitSpriteInfo and AddSpriteImage. If bAutoSkip is TRUE calling CSprite::InitSpriteInfo the frame counter is incremented automatically on every call to CSprite::MoveSprite, SpriteMgr::MoveSprite and between the animation steps from calls to CSprite::AnimateSprite and CSpriteMgr::AnimateSprite.

virtual void **FreeBitmaps**(BOOL bDeleteObjectsOnly = FALSE)

BOOL bDeleteObjectsOnly If TRUE, only CBitmap::DeleteObject() is called, else the delete operator is used on the Cbitmaps containing the sprites images

This function deletes the GDI bitmap objects and depending on bDeleteObjectsOnly it removes the CBitmap objects from memory. This function usually is not called from the user of the library but from the CSprite destructor and CSprite::InitSpriteInfo function.

virtual void **GetSpritePos**(CRect& pos)

CRect& pos CRect object to receive the coordinates

This function places the sprites current position in the given CRect object.

virtual void **GetBoundingRect**(CRect& rect)

CRect& rect CRect object to receive the coordinates

This function places the sprites boundig rectangle in the given CRect object. Called from outside the library this would return the same coordinates then CSprite::GetSpritePos.

The CSpriteMgr class

Overview

The CSpriteMgr class provides you an object that is capable of managing an unlimited (limited by memory, of course) number of CSprite objects. Usually you may not use two or more CSprite objects without having them managed by a CSpriteMgr object.

Managing CSprite object means:

- preventing from painting errors if two or more sprites positions intersect
- preventing from painting errors if the sprites are partially overlapped by other windows
- providing an easy way to access information about sprite position intersections
- implementing a Z-order on CSprite objects
- making standard operations like handling mouse dragging or OnPaint calls a lot easier
- providing the possibility to move several sprites with intersecting boundaries without flickering

The interface of the CSpriteMgr class

virtual BOOL **AddSprite**(CSprite* pSprite, int nPlaceInFrontOfID = TOPSPRITE)

CSprite* pSprite	pointer to sprite that should be added
int nPlaceInFrontOfID	the new sprite should be placed in Z-order in front of the sprite identified by this id

Adds a sprite to the sprite manager.

You may also pass one of the following constants as nPlaceInFrontOfID:

TOPSPRITE	positions the sprite in front of all other sprites
BOTTOMSPRITE	positions the sprite beneath all other sprites

virtual void **RemoveSprite**(CDC* pDC, int nID, BOOL bFreeMem = TRUE)

CDC* pDC	pointer to the windows client dc
int nID	id of sprite to remove
BOOL bFreeMem	boolean that specifies whether the sprite should only be removed from the CSpriteMgr's sprite list (FALSE) or if it should be removed from memory (TRUE) too.

Removes a sprite from the sprite manager.

virtual void **ShowSprite**(CDC* pDC, int nID, BOOL bShow)

CDC* pDC	pointer to the windows client dc
int nID	id of sprite to manipulate
BOOL bShow	boolean specifying whether the sprite should be shown (TRUE) or hidden (FALSE)

This function shows or hides a sprite. If the sprite has already the desired state nothing happens.

virtual BOOL **IsSpriteVisible**(int nID)

int nID	id of sprite
---------	--------------

Returns TRUE if the sprite is visible else FALSE. „Visible“ does not mean that the sprite is not hidden by another window but if it has been made visible by passing TRUE as bVisible in the sprites construtor or calling ShowSprite with bShow = TRUE.

virtual BOOL **IntersectSprites**(CObArray* pObArray, CRect& rect)

CObArray* pObArray pointer to a CObArray object holding the function results
CRect& rect rectangle to intersect with

Calculates for all sprites if there positions are intersecting with the given rectangle. The intersecting sprites are returned in pObArray in Z-order (top-most sprite at index 0 and so on). The function returns TRUE if there are intersecting sprites.

Important: The CObArray object receiving the result has to be constructed by the user before calling this function.

virtual BOOL **IntersectSprites**(CObArray* pBeneath, CObArray* pAbove, CSprite* pSprite,
CRect additionalRect = CRect(0,0,0,0))

CObArray* pBeneath pointer to a CObArray object holding the function results
CObArray* pAbove pointer to a CObArray object holding the function results
CSprite* pSprite pointer to the CSprite that should be manipulated
CRect additionalRect an additional rect to intersect with

Calculates for all sprites if there positions are intersecting with the given rectangle or the bounding rectangle of pSprite. The intersecting sprites are returned in pBeneath and pAbove in Z-order (top-most sprite at index 0 and so on). pBeneath holds the sprites with lower Z-order then pSprite. pAbove holds the sprites with higher Z-order then pSprite.

Important: The CObArray objects receiving the result have to be constructed by the user before calling this function.

virtual void **MoveSprite**(int nID, CDC* pDC, CPoint &pos)

int nID id of sprite to manipulate
CDC* pDC pointer to the windows client dc
CPoint &pos new position

This function moves the given sprite to the given position.

virtual void **MoveSprite**(CSprite* pSprite, CDC* pDC, CPoint &pos)

CSprite* pSprite pointer to the CSprite that should be manipulated
CDC* pDC pointer to the windows client dc
CPoint &pos new position

This function moves the given sprite to the given position.

virtual void **AnimateSprite**(int nID, CDC *pDC, CPoint& newPos, int nStepFactor)

int nID id of sprite to manipulate
CDC *pDC pointer to the windows client dc
CPoint& newPos end position of the animation

int nStepFactor number of animation steps

This function animates the given sprite from the current position to the given position drawing nStepFactor frames.

virtual void **AnimateSprite**(CSprite* pSprite, CDC *pDC, CPoint& newPos, int nStepFactor)

CSprite* pSprite	pointer to the CSprite that should be manipulated
CDC *pDC	pointer to the windows client dc
CPoint& newPos	end position of the animation
int nStepFactor	number of animation steps

This function animates the given sprite from the current position to the given position drawing nStepFactor frames.

virtual void **SkipFrames**(int nID, BOOL bShow, int nOffset = 1, CDC *pDC = NULL)

int nID	id of sprite to manipulate
BOOL bShow	boolean that specifies wether the new selected images should be displayd or not
int nOffset	number of frames to skip
CDC *pDC	pointer to the windows client dc

For an explanation of this function look at the description of CSprite::SkipFrames. The differences to CSprite::SkipFrames are:

The new image could be displayed automatically using parameter bShow

If you set bShow to TRUE you also have to pass a valid pointer to the device context in which the changing should be displayed in parameter pDC.

virtual void **OnPaint**(CDC* pDC)

CDC* pDC pointer to the windows client dc

This function calls StoreBkg and DrawSprite for alle sprites managed by the CSpriteMgr object. This is usefull in the CWnd::OnPaint or the CView::OnDraw method of the window containing the sprites. Calling this function all sprites read there new background and draw themself on the given dc.

Example:

```
void CMyView::OnDraw(CDC* pDC)
{
    ...
    // some drawing code of your window
    ...
    m_SpriteMgr.OnPaint(pDC);
}
```

virtual BOOL **HitTest**(CObArray* pObArray, CPoint& point, BOOL bExact, CDC* pDC)

CObArray* pObArray	pointer to a CObArray object holding the function results
CPoint& point	point to check for
BOOL bExact	boolean specifying wheter the sprites image transparency should be involved or not
CDC* pDC	pointer to the windows client dc

This function checks if the given point hits one of the sprites. If the given point lies within the sprites rectangle the function returns TRUE else it will return FALSE. If bExact is TRUE and the given point hits the sprite on a transparent pixel the function will return FALSE. The sprites that are hit by the given point are returned in pObArray in there Z-order (top-most sprite at id 0 and so on).

Important: The CObArray object receiving the result has to be constructed by the user before calling this function.

virtual int **StartDrag**(CWnd* pWnd, CPoint& pos, BOOL bExact, CDC* pDC)

CWnd* pWnd pointer to the window containing the sprite
 CPoint& pos current mouse position
 BOOL bExact boolean specifying wheter the sprites image transparency should be involved or not
 CDC* pDC pointer to the windows client dc

This function starts a mouse dragging action by setting intern variables and calling SetCapture to make sure the given window will receive all the mouse messages until EndDrag is called. The function checks which sprite has been hit and returns the sprite id or the constant SPRITE_ERROR_ID if no sprite has been hit.

Usually you will implement mouse dragging like this:

```
void CMyWindow::OnLButtonDown(..., CPoint point)
{
    CClientDC dc(this);
    if(m_SpriteMgr.StartDrag(this, point, TRUE, &dc) == SPRITE_ERROR_ID)
        ::MessageBeep(MB_OK); // no sprite has been hit!

    CWnd::OnLButtonDown(..., CPoint point);
}

void CMyWindow::OnMouseMove(..., CPoint point)
{
    if(m_SpriteMgr.IsDragging())
    {
        CClientDC dc(this);
        m_SpriteMgr.OnDrag(&dc, point);
    }
    CWnd::OnMouseMove(..., CPoint point);
}

void CMyWindow::OnLButtonUp(..., CPoint point)
{
    if(m_SpriteMgr.IsDragging())
    {
        CClientDC dc(this);
        m_SpriteMgr.EndDrag(&dc, point);
    }
    CWnd::OnLButtonUp(..., CPoint point);
}
```

virtual void **OnDrag**(CDC* pDC, CPoint& pos)

CDC* pDC pointer to the windows client dc
 CPoint& pos current mouse position

Moves the dragging sprite to the new position

virtual void **EndDrag**(CDC* pDC, CPoint& pos)

CDC* pDC pointer to the windows client dc
CPoint& pos current mouse position

Moves the dragging sprite to the new position and releases the mouse capturing.

virtual BOOL **IsDragging**()

This function returns TRUE if one of the sprite managers sprites is currently dragged with the mouse.

virtual CSprite* **GetSpriteFromID**(int nID)

int nID id of sprite to get pointer from

This function returns a pointer to the sprite with the given id or NULL if there is no such sprite.

CSprite* **operator[]**(int nID)

int nID id of sprite to get pointer from

This operator returns a pointer to the sprite with the given id or NULL if there is no such sprite.

virtual void **ChangeZOrder**(CDC* pDC, int nID, int nPlaceInFrontOfID = TOPSPRITE)

CDC* pDC pointer to the windows client dc
int nID id of sprite to manipulate
int nPlaceInFrontOfID the sprite should be placed in Z-order in front of the sprite identified by this id

Changing the given sprites Z-order by placing it in front of the sprite given by nPlaceInFrontOfID.

You may also pass one of the following constants as nPlaceInFrontOfID:

TOPSPRITE positions the sprite in front of all other sprites
BOTTOMSPRITE positions the sprite beneath all other sprites

virtual BOOL **BeginDeferSpritePos**()

This function starts a block of sprite movements which are combined so that they could be executed all at once. Doing so prevents flickering if the sprites moved have overlapping bounding rectangles. There can only be one DeferSpritePos-Block at a time. The function returns TRUE if successful.

virtual void **DeferSpritePos**(int nID, CPoint &pos)

int nID id of sprite to manipulate
CPoint &pos new position

This function works like a call to MoveSprite but with the difference that the drawing itself will not occur until the next call to EndDeferSpritePos.

virtual void **DeferSpritePos**(CSprite* pSprite, CPoint &pos)

CSprite* pSprite pointer to the CSprite that should be manipulated
CPoint &pos new position

This function works like a call to MoveSprite but with the difference that the drawing itself will not occur until the next call to EndDeferSpritePos.

virtual void **EndDeferSpritePos**(CDC* pDC)

CDC* pDC pointer to the windows client dc

This function realizes all DeferSpritePos-calls to the given dc.

Example:

```
if(!m_SpriteMgr.BeginDeferSpritePos())      // begin the DeferSpritePos-block
    return;

m_SpriteMgr.DeferSpritePos(5, CPoint(100, 100)); // no painting is done
m_SpriteMgr.DeferSpritePos(3, CPoint(100, 200)); // no painting is done
m_SpriteMgr.DeferSpritePos(4, CPoint(200, 100)); // no painting is done

CClientDC dc(this);
m_SpriteMgr.EndDeferSpritePos(&dc);      // all movements are displayed at once
```