```
// The following function is only for use with FPD data files
// coming from the SVS2000
// Update done by MLH to accept void* and data type
BOOL CImageObject::ApplyVector(void* pWorig, void* pWnew, const char *pszFilename, WORD dataType)
{
	LFileHdr		hdr;		// Standard lumitron file header
	VectorFileHdr		m_VectorFileHeader;
	HGLOBAL			hVectMemory;
	DWORD			*pVectorData;
	DWORD			bytes;
	CFile			m_File;
	BOOL			m_FileOpened;
	CFileException		theException;
	CString			tempStr;
	float*			fOrigPtr;
	float*			fNewPtr;
	WORD*			wOrigPtr;
	WORD*			wNewPtr;

	/* Alert if no Vector file is loaded */
	if ((strcmp(pszFilename, "") == 0) || ((strstr(pszFilename, ".vct") == NULL) &&
		(strstr(pszFilename, ".VCT") == NULL)))
	{
		AfxMessageBox("Please Load Vector File for Scan Conversion!");
		return(FALSE);
	}

	switch (dataType)
	{
	case (IMAGE_FLOAT_DATA):
		fOrigPtr = (float*)pWorig;
		fNewPtr = (float*)pWnew;
		break;
	case (IMAGE_WORD_DATA):
		wOrigPtr = (WORD*)pWorig;
		wNewPtr = (WORD*)pWnew;
		break;
	}


	m_FileOpened = m_File.Open(pszFilename, CFile::modeRead | CFile::typeBinary,
&theException);
	if (!m_FileOpened)
	{
		tempStr = "Error opening file: ";
		tempStr += *pszFilename;
		AfxMessageBox(tempStr);
		return(FALSE);
	}

	/* Store vector file name to header */
	strcpy(m_SRSHeader.vector_file, pszFilename);

	// now read in the vector table from file
	bytes = m_File.Read(&hdr, sizeof(hdr));	// Get the standard lumitron header
	if ((hdr.eof != 26) || /*(strcmp(hdr.signature, LUM_SIGNATURE) != 0) ||*/
		(hdr.product != LUM_SVS2000) || (hdr.filetype != LUM_VECTOR_FILE) ||
		(bytes != sizeof(hdr)))
	{
			AfxMessageBox("Invalid Vector file");
			if (hdr.eof != 26)
				AfxMessageBox("eof problem");
			if (strcmp(hdr.signature, LUM_SIGNATURE) != 0)
				AfxMessageBox("signature problem");
			if (hdr.product != LUM_SVS2000)
				AfxMessageBox("product id problem");
			if (hdr.filetype != LUM_VECTOR_FILE)
				AfxMessageBox("filetype problem");
			if (bytes != sizeof(hdr))
				AfxMessageBox("bytes read <> size of hdr problem");
```

```
                m_File.Close();
                return(FALSE);
        }

        bytes = m_File.Read(&m_VectorFileHeader, sizeof(m_VectorFileHeader));
        if (bytes != sizeof(m_VectorFileHeader)) {
                AfxMessageBox("Invalid Vector file");
                m_File.Close();
                return(FALSE);
        }

        if ((m_FPAwidth == 0) || (m_FPAheight == 0))
        {
                m_FPAwidth = m_VectorFileHeader.fullFrameHoriz;
                m_FPAheight = m_VectorFileHeader.fullFrameVert;
        }

        hVectMemory = ::GlobalAlloc( GMEM_MOVEABLE | GMEM_ZEROINIT,
                m_VectorFileHeader.fullFrameHoriz * m_VectorFileHeader.fullFrameVert *
sizeof(DWORD) );

        if (hVectMemory == NULL)
        {
                AfxMessageBox("Memory error allocating vector table memory");
                ::GlobalFree( hVectMemory );
                ::GlobalUnlock( m_hDib );
                return( FALSE );
        }

        pVectorData = (DWORD *) ::GlobalLock(hVectMemory);
        if (pVectorData == NULL)
        {
                AfxMessageBox("Memory Lock error in ApplyVector");
                ::GlobalFree( hVectMemory );
                return( FALSE );
        }
        bytes = m_File.Read(pVectorData, m_VectorFileHeader.dataSize);
        m_File.Close();

        DWORD   acqcard;
        DWORD   position;
        DWORD   NumPixels;
        DWORD   PixPerCard;

        NumPixels = (DWORD) m_VectorFileHeader.fullFrameHoriz*m_VectorFileHeader.fullFrameVert;
        // Line Updated by MLH
        //PixPerCard = (DWORD) NumPixels /
(m_VectorFileHeader.videoOutputs/m_VectorFileHeader.videoChannels);

        // Line above does not work for linear arrays with inter/intra clip pixels added to mix
        PixPerCard = m_VectorFileHeader.inScanPix * m_VectorFileHeader.videoChannels *
m_VectorFileHeader.crossScanPix;

        DWORD i;
        for (i = 0; i < NumPixels; i++)
        {
                acqcard = (DWORD)((pVectorData[i] >>19) & 0x07);
                position = (DWORD)((pVectorData[i]) & 0x7FFFF);

                position = (DWORD)PixPerCard * acqcard + position;

                //*pWnew++ = pWorig[position];
                switch (dataType)
                {
                case (IMAGE_FLOAT_DATA):
                        *fNewPtr++ = fOrigPtr[position];
                        break;
                case (IMAGE_WORD_DATA):
                        *wNewPtr++ = wOrigPtr[position];
                        break;
                }
```

```
        }
        //::GlobalUnlock( hNewMemory );
        ::GlobalUnlock(hVectMemory);
        ::GlobalFree(hVectMemory);

        return (TRUE);
}
```