Raster Methods for 2-D Geometric Translations

- See CS-460/560 Notes:
  http://www.cs.binghamton.edu/~reckert/460/bitmaps.htm
- Can translate image in frame buffer by:
  - Embed it in a rectangle
  - Copy color of each pixel in rectangle to a destination rectangle
    - A simple double loop

- Assume coordinates of opposite corners of embedding rectangle are (x1,y1) and (x2,y2)
- Want to translate enclosed image by (tx,ty)
- Following pseudocode will do the job:

```
for (x=x1 to x2)
  for (y=y1 to y2)
    {
    color = GetPixel(x,y);   // get pixel color
    SetPixel(x+tx, y+ty, color);   // paint pixel
    }
```

- Known as a Bit Block Transfer (BiTBLT, or "blitting")
- Can also be used to move offscreen images to screen
- Can be very fast if we have direct access to the frame buffer
- Windows API uses Device Dependent Bitmaps to achieve effect of Bit Blitting
  - Also provides functions to stretch bitmaps

Introduction to Windows Bitmaps

- See CS-360, CS-460/560 Notes & Programs:

http://www.cs.binghamton.edu/~reckert/460/bitmaps. htm
http://www.cs.binghamton.edu/~reckert/360/class4a.htm
http://www.cs.binghamton.edu/~reckert/360/bitmap1_cpp.htm
http://www.cs.binghamton.edu/~reckert/360/bitmap3_cpp.htm

Bitmap: An Off-screen Canvas

- Rectangular image, can be created with a paint program
- Data structure that stores a matrix of pixel values in memory
  - Pixel value stored determines color of pixel
- Windows supports 4-bit, 8-bit (indirect) & 24-bit (direct) pixel values
- Can be stored as .bmp file (static resource data)
- Can be edited; can save any picture
- Takes up lots of space
  - No compression

- A bitmap is a GDI object
  - must be selected into a DC to be used
- A “resource” (like menus, dialog boxes, etc.)
- Think of it as the canvas of a DC upon which drawing takes place
- Must be compatible with the display device
- Can be manipulated invisibly and apart from physical display device
- Fast transfer to/from physical device ==> flicker-free animation
- Does not store information on drawing commands
  - Windows Metafiles do that

Using Device Dependent Bitmaps in MFC

A. Create and save bitmap using a paint editor --> image.bmp file

Add to program's resource script file

– e.g.: IDB_IMG BITMAP image.bmp

– easier to: “Project | Add Resource | Bitmap”

B. Instantiate a CBitmap object

```
CBitmap bmp1;
```

C. Load bitmap from the program's resources:

```
bmp1.LoadBitmap(IDB_IMG);
```

D. Display the bitmap

0. Get a ptr to the screen DC (as usual): pDC
1. Create a memory device context compatible with the screen DC

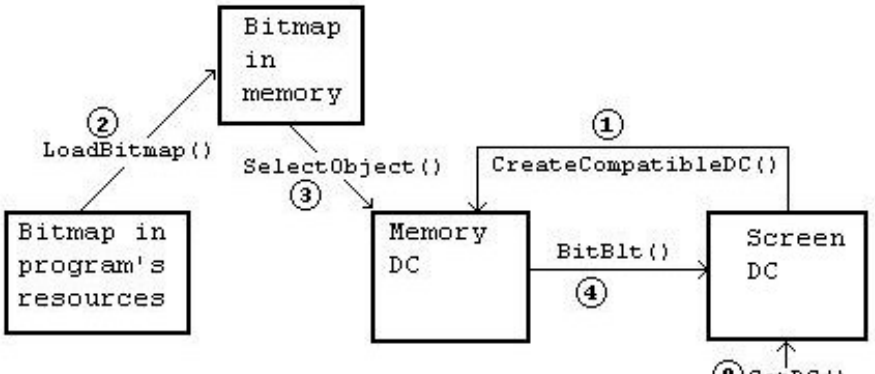
```
CDC  dcMem;
dcMem.CreateCompatibleDC(pDC);
```

2. Select bitmap into the memory DC

```
CBitmap* pbmpold = dcMem.SelectObject(&bmp1);
```

3. Copy bitmap from memory DC to device DC using pDC’s BitBlt() or StretchBlt()
4. Select bitmap out of memory DC

Using Bitmaps




```
0. hDC=GetDC(hWnd);
1. hMDC=CreateCompatibleDC(hDC);
2. hBM=LoadBitmap(hInst,"bmp");
3. SelectObject(hMDC,hBM);
4. BitBlt(hDC,...,hMDC,...);
```

A Memory DC

- Like a DC for a physical device, but not tied to device
- Used to access a bitmap
- Bitmap must be selected into a memory DC before displayable on physical device
- CreateCompatibleDC(pDC) --> a memory DC with same attributes as the device DC
- SelectObject() selects bitmap into DC
  - copying from memory DC is fast since data sequence is same as on the device

Blitting in Windows

- pDC->BitBlt (x, y, w, h, &dcMem,
  xsrc, ysrc, dwRop)
  - Copies pixels from bitmap in source DC (dcMem) to destination DC (pDC)
  - x,y: upper left hand corner of destination rectangle
  - w,h: width, height of rectangle to be copied
  - xsrc, ysrc -- upper left hand corner of source bitmap
  - dwRop -- raster operation for copy

Raster Ops

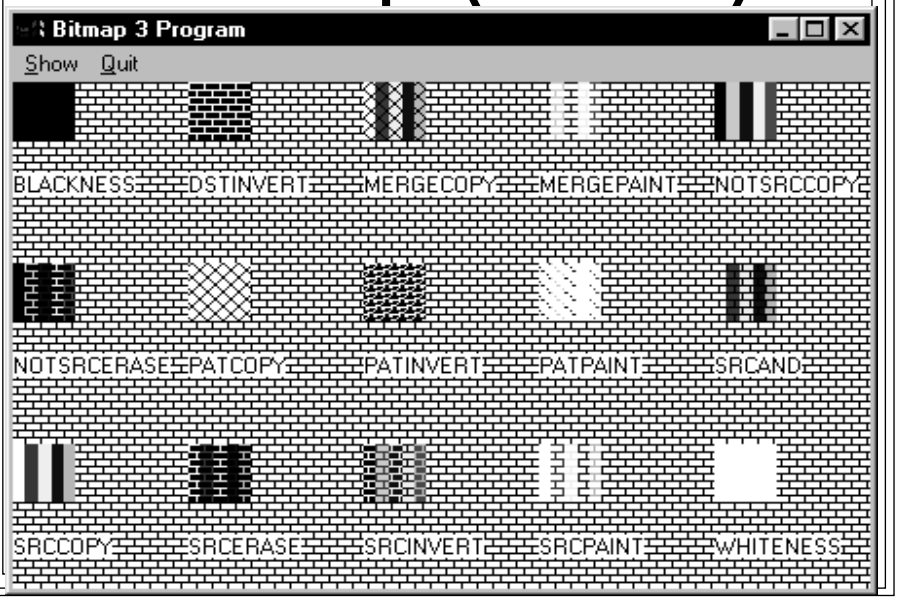
- How source pixel colors combine with current pixel colors
- Boolean logic combinations (AND, NOT, OR, XOR, etc.)
  - Currently-selected brush pattern also can be combined
  - so 256 different possible combinations
  - 15 are named
- Useful for special effects

Named Raster Ops

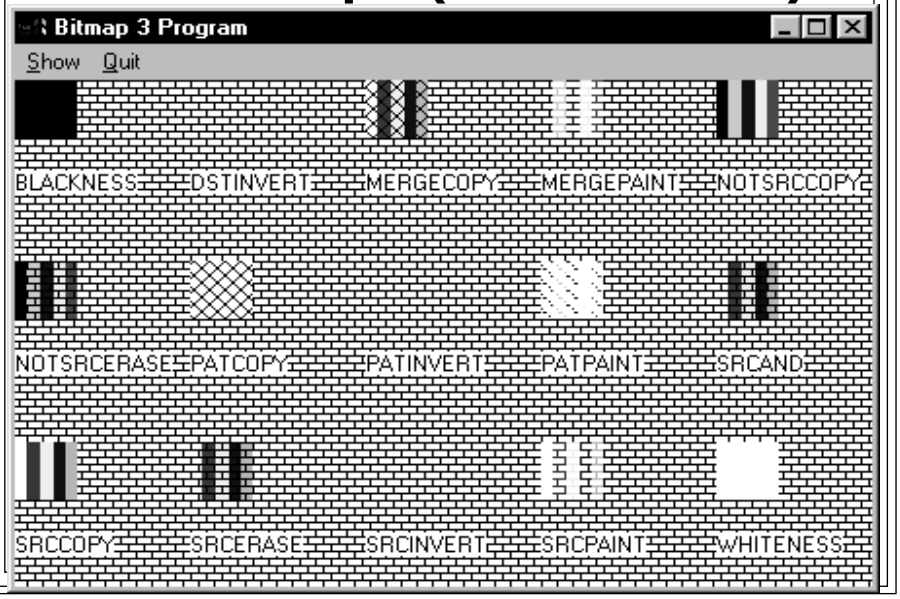
- (S=source bitmap, D=destination, P=currently-selected brush, i.e., the current Pattern)

BLACKNESS	0 (all black)	DSTINVERT	~D
MERGECOPY	P & S	MERGEPAINT	~S | D
NOTSRCCOPY	~S	NOTSRCERASE	~(S | D)
PATCOPY	P	PATINVERT	P ^ D
PATPAINT	(~S | P) | D	SRCAND	S & D
SRCCOPY	S	SRCERASE	S & ~D
SRCINVERT	S ^ D	SRCPAINT	S | D
WHITENESS	1 (all white)		

Raster Ops (first time)



Raster Ops (second time)



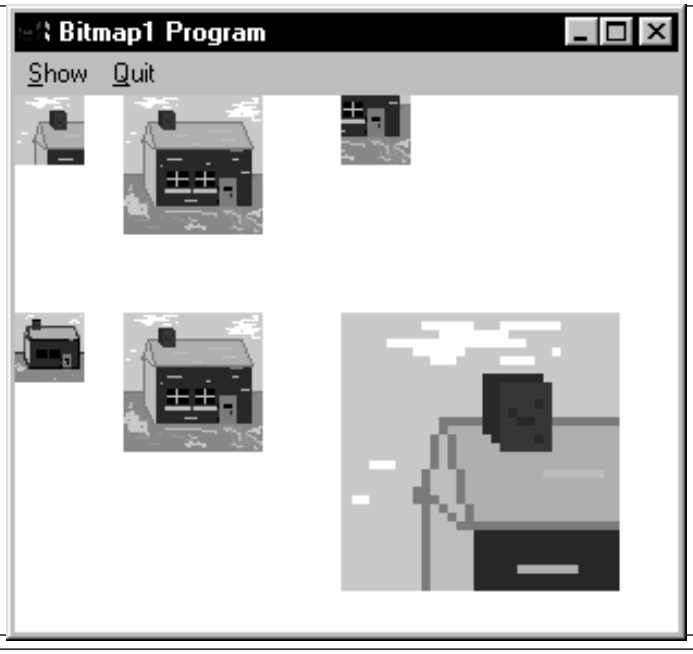
StretchBlt()

- Same as BitBlt() except size of copied bitmap can be changed
- Source & destination width/height given

  pDC->StretchBlt (x,y,w,h,&dcMem, xsrc,ysrc,wsrc,hsrc,RasterOp);

Examples of BitBlt & StretchBlt

```
CBitmap bmpHouse; CDC dcMem;
BITMAP bm; int w,h;
bmpHouse.LoadBitmap(IDB_HOUSE);
bmpHouse.GetObject(sizeof(BITMAP), &bm);
w = bm.bmWidth; h = bm.bmHeight;
dcMem.CreateCompatibleDC(pDC);
CBitmap* pbmpOld = dcMem.SelectObject(&bmpHouse);
pDC->BitBlt(0, 0, w/2, h/2, &dcMem, 0, 0, SRCCOPY);
pDC->BitBlt(50, 0, w, h, &dcMem, 0, 0, SRCCOPY);
pDC->BitBlt(150, 0, w/2, h/2, &dcMem, w/2, h/2, SRCCOPY);
pDC->StretchBlt(0,100,w/2,h/2,&dcMem,0,0,w,h,SRCCOPY);
pDC->StretchBlt(50,100,w,h,&dcMem,0,0,w,h,SRCCOPY);
pDC->StretchBlt(150,100,2*w,2*h,&dcMem,0,0,w/2,h/2,SRCCOPY);
dcMem.SelectObject(pbmpOld);
```



Loading and Displaying Image Files in Windows

- CImage Class
  - In the Active Template Library (ATL)
    - Primarily used to build components & services
  - Before .NET, MFC and ATL could not be used together
  - Under .NET there is a set of shared classes that can be used in both
  - CImage is one of the shared classes
    - Encapsulates functionality to read/draw/save JPEG, GIF, BMP, and PNG images
    - To use it: #include <atlimage.h> at top of stdafx.h file

Some Useful CImage Member Functions

BOOL Create(int w, int h, int nBPP, DWORD dwFlags);

- Creates a Cimage bitmap and attaches it to a previously constructed Cimage object
  - w = width of image, h = height of image
  - nBPP: number of bits per pixel
  - DwFlags: 0 or creqateAlphaChannel (32 only)

Load(LPCTSTR strFileName);

- Loads the image from a file
  - strFileName can be a CString

Save(LPCTSTR strFileName, REFGUID guidFileType);

- guidFileType:
  - GUID_NULL: determined from file extension
  - GUID_JPEGFILE: JPEG image
  - GUID_GIFFILE: GIF image
  - Same idea for BMP and PNG images

Easy Image Conversion using CImage

```
CImage m_image;

// Read in GIF Image
m_image.Load("myimage.gif");

// Write out image as a JPEG
m_image.Save("myimage.jpg", GUID_NULL);
```

Drawing an image with CImage

- CImage::Draw(…) member function
  - Draw(HDC, int xdest, int ydest, int wdest, int hdest, int xsrc, int ysrc, int wsrc, int hsrc);
  - Lot's of other possible arguments
    - See online help
- CImage::BitBlt(…)
- CImage::StretchBlt(…);

CImage Enquire Functions

- int w = GetWidth();
- int h = GetHeight();

Example of Using CImage with a Common File Dialog Box

```
CImage  m_image;
CString  m_strImage;
CDC*  pDC;
CFileDialog  dlgFile(TRUE);  // TRUE➔file open, not file save DBox
if (dlgFile.DoModal() == IDOK)
{
        m_strImage = dlgFile.GetPathName();
        m_image.Load(m_strImage);
        int w = m_image.GetWidth();
        int h = m_image.GetHeight();
        pDC=GetDC();
        m_image.Draw(pDC->m_hDC,10,10,w,h,0,0,w,h);
}
```

Bitmaps in OpenGL

- Define a binary array pattern with:
  - glBitmap(w,h,x0,y0,xoffset,yoffset,array)
    - w,h: # of columns and rows
    - X0,y0: position of origin in array
      - Relative to lower left corner of array
    - xoffset,yoffset: new current raster position after bitmap is displayed
    - array: binary array that defines the bitmap
      - 1: pixel to be set to current color
      - 0: pixel unaffected
- Use glRasterPos2i(x,y) first to set position in frame buffer of lower left hand corner of destination
- See example on page 144 of Hearn & Baker

Pixmaps in OpenGL

- Fundamental Pixel-writing function is: glDrawPixels(width, height, dataFormat, dataType, pixMap);
- First define an RGB image as, for example, in:

  GLuByte image[ROWS][COLS][3];
- Then display at current raster position as, for example, with:

  glDrawPixels(ROWS, COLS, GL_RGB, GL_UNSIGNED_BYTE, image);

BitBlitting in OpenGL

glReadPixels(x0, y0, w, h, GL_RGB, GL_UNSIGNED_BYTE, array);

- Retrieves pixels from frame buffer
  - x0,y0: lower left corner
  - w,h: number of columns and rows
  - array: Destination array in memory

glCopyPixels(x0, y0, w, h, pixelValues);

- Copies the block of w x h pixels starting at (x0,y0)
  - destination location starts at current raster position
  - Source & destination buffers chosen w/ glReadBuffer(buf) & glDrawBuffer(buf) , buf=GL_FRONT, GL_BACK, etc.

- Image file support in OpenGL – Not Much!
  - Must write your own functions to load/save standard image file formats in OpenGL

Animated Graphics

Notes from CS-360 Web Pages

Course Notes:

Class 4 -- Windows Bitmaps,Animation, and Timers
http://www.cs.binghamton.edu/~reckert/360/class4a.htm

Sample Programs:

Example 4-3: Bouncing Ball Animation using PeekMessage() (ball.cpp)
http://www.cs.binghamton.edu/~reckert/360/ball_cpp.htm

Example 4-4: Bouncing Ball Animation with Bitblt() to Preserve Background (ballblt.cpp)
http://www.cs.binghamton.edu/~reckert/360/ballblt_cpp.htm

Example 4-5: Bouncing Ball Animation using a Timer (balltime.cpp)
http://www.cs.binghamton.edu/~reckert/360/balltime_cpp.htm

Animated Graphics

- Creating a moving picture
  - Give illusion of motion by continual draw / erase / redraw
  - If done fast, eye perceives moving image
- In a single-user (e.g., DOS) application, we could do the following:

```
Do Forever{
   /* compute new location of object */
   /* erase old object image */
   /* draw object at new location */  }
```

- In Windows, other programs can't run while this loop is executing
- Need to keep giving control back to Windows so other programs can operate
- One method:
  - Use a Windows Timer

Windows Timers

- An input device that notifies an application when a time interval has elapsed
  - Application tells Windows the interval
  - Windows sends WM_TIMER message each time interval elapses

Using a Timer

- Allocate and set a timer with:

  CWnd::SetTimer(timerID, interval, NULL);
  - Interval in milliseconds

- From that point on, timer repeatedly generates WM_TIMER messages and resets itself each time it times out
  - Could be used to signal drawing the next frame of an animation!!!
- WM_TIMER handler: OnTimer(timerID)
- When app is done using a timer, stop timer messages and remove it with:

  KillTimer(timerID);
- Example: Balltime
  - See CS-360, Example Programs: Example 4-6 for .NET

Drawing on a Memory Bitmap (Improving an Animation)

- If many objects are drawn during each frame of an animation, we get flicker
  - Because of multiple accesses to frame buffer during each frame
- Best way to eliminate flicker:
  - Have just one access to frame buffer per frame
  - Windows: use off-screen memory bitmaps

Drawing on Off-screen Bitmaps

- Use GDI functions to "draw" on a bitmap selected into a memory DC
- Just like using a "real" DC
  - So we can do many drawing operations “offscreen”
- When finished, BitBLt() result to real DC
  - Fast, so no flicker in animations
- This technique is called double buffering

Getting a Bitmap to Draw on

- Create a blank bitmap in memory with:

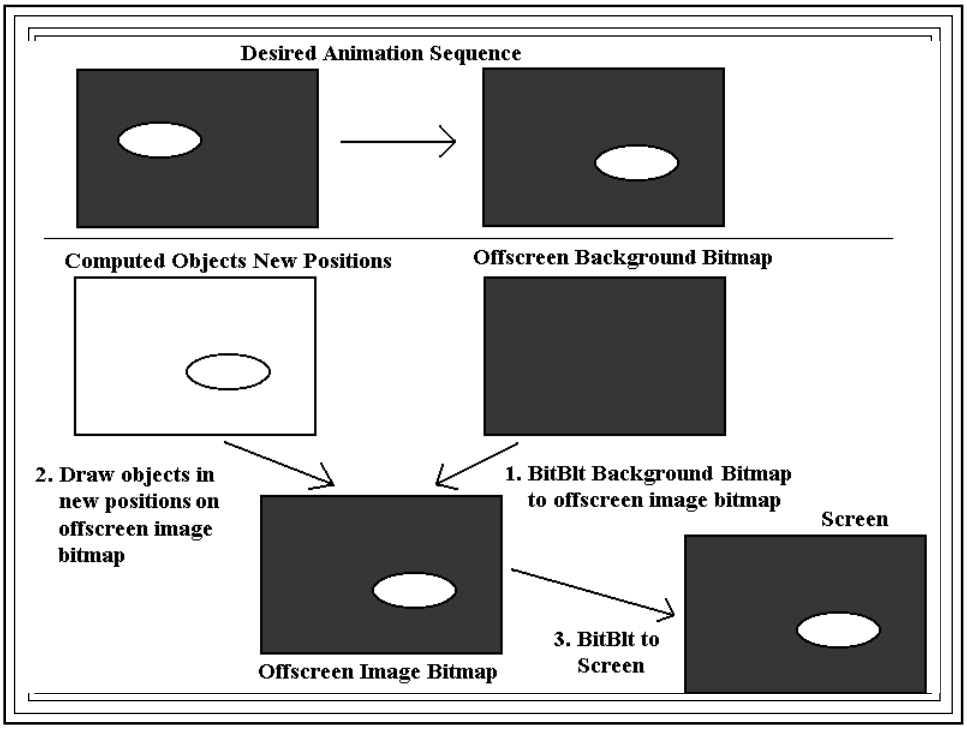
CBitmap::CreateCompatibleBitmap (pDC, w, h);

  - An alternative to LoadBitmap()
- After it’s selected into a memory DC, use GDI graphics functions to draw on it without affecting real device screen
  - All the GDI drawing operations are now invisible to the user

- When drawing is all done, BitBLt() it to real device
  - So just one screen access
  - No flicker
    - (drawing directly to screen device context ==> many accesses to screen
      - produces flicker for complex images
- Process is an example of double buffering
  - Draw next frame on an offscreen canvas
  - while current frame is displayed on real screen
  - OpenGL also provides for Double Buffering
    - glutSwapBuffers(); // Replaces glFlush();
    - First: glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);

Animation of moving objects over a stationary background

- Set up an offscreen image bitmap and select it into a memory DC
- Set up an offscreen background bitmap
- For each frame (each timer timeout):
  - Calculate new positions of objects
  - BitBLt() background bitmap to the offscreen image bitmap
  - Redraw/BitBlt objects (in new positions) on the offscreen image bitmap
  - BitBLt() entire offscreen image bitmap to screen
- Example: imageblt



- For a large image field, this BitBLt() covers a large area
  - could be too slow
- Better method: compute affected area
  - (rectangle encompassing old and new object positions)
- BitBLt() to that area only

Sprites

- Little bitmaps that move on screen
- Frequently used in game programs
- Could restore background and just BitBLt() the sprite over it
- But there's a problem
  - sprite consists of desired image enclosed in a rectangle
  - so when blitting is done, background color inside enclosing rectangle will wipe out the background area on destination bitmap
  - moving object will have a "halo" around it
  - will also always have a rectangular shape

Solution (Sprite Animation)

1. Set up a "mask bitmap" in which sprite pixels are black and the rest of the enclosing rectangle is white
2. BitBLt() this over background using SRCAND (AND) raster op
3. Use the "image bitmap" in which the sprite pixels are set to the image colors they should be (whatever colors are in the sprite object) and the rest of enclosing rectangle pixels (the “halo”) are black
4. BitBLt() this to the result of step 2 using the SRCINVERT (XOR) raster op

   Result will make sprite move to its new location with the background around it intact

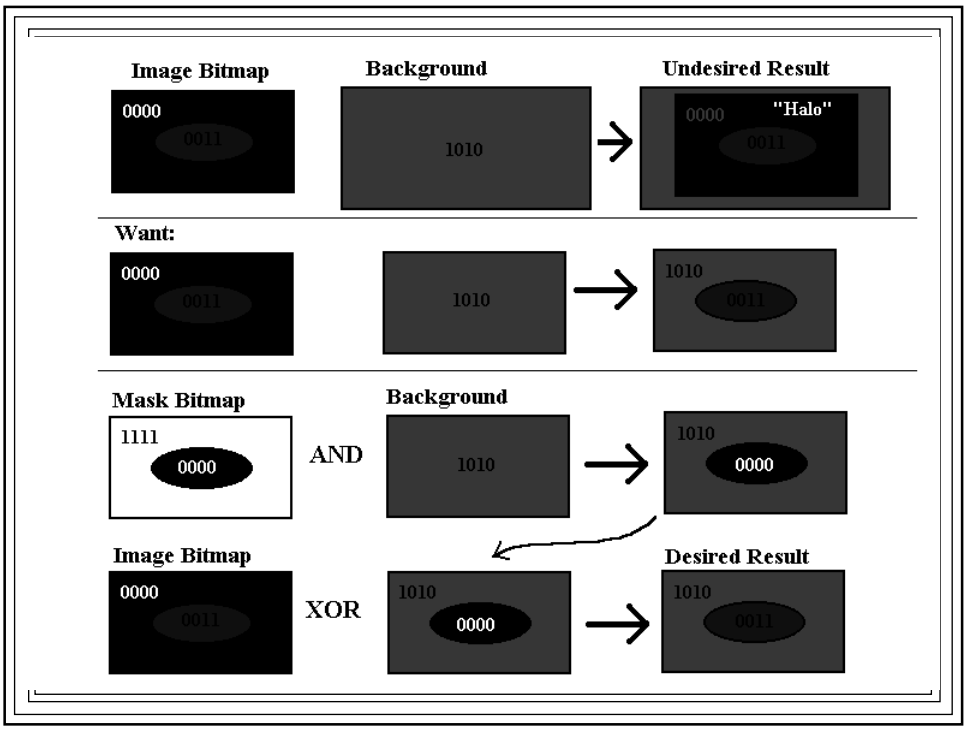


Image Bitmap
0000
0011
Background
1010
Undesired Result
0000
"Halo"
0011
Want:
0000
0011
1010
1010
0011
Mask Bitmap
1111
0000
AND
Background
1010
1010
0000
Image Bitmap
0000
0011
XOR
1010
0000
Desired Result
1010
0011