Blending

Motivation

- Images are stored as rectangular shape for simplicity
  - Not all parts of an image should be visible
- What to do or what should happen when several images are overlapping?
- How to handle transparency
  - Scene through a window
  - Looking through water

Blending

- Blending is a process in which two or more images are used to create the “final” image
  - Usually one “blends” new drawn pixels with those that were already drawn
- An overlap operation where an operator is used to determine the value of each pixel location
  - Output = inputA ⊕ inputB
- Effect of blending
  - Mixing
  - Transparency

Blending Forumula

- Output = Source Pixel ⊗ Source Blend Factor ⊕
  Dest Pixel ⊗ Dest Blend Factor
- Output – the blended pixel
- Source Pixel – the pixel that is currently been rendered
- Source Blend Factor – specifies the weight (percent) to be used in the blending operation
  - range [0..1]
- Dest Pixel – the pixel in the back buffer
- Dest Blend Factor – the blending weight in the range of [0..1].

Blend Factors

- The blend factors are defined using device function
- SetRenderState(STATE, value)
  - d3dDevice->SetRenderState(D3DRS_SRCBLEND, value)
  - d3dDevice->SetRenderState(D3DRS_DESTBLEND, value)
- Each of the source and destination can have several values
- The blend factors operate on each of the colour channels – Red, Green, and Blue

Blending Factors Values

- D3DBLEND_ZERO – blend factor = (0,0,0,0)
- D3DBLEND_ONE – blend factor = (1,1,1,1)
- D3DBLEND_SRCCOLOR – blend factor = (r_s,g_s,b_s,a_s)
- D3DBLEND_INVSRCCOLOR – blend factor = $(1-r_s,1-g_s,1-b_s,1-a_s)$
- D3DBLEND_SRCALPHA – blend factor = (a_s,a_s,a_s,a_s)
- D3DBLEND_INVSRCALPHA – blend factor = $(1-a_s,1-a_s,1-a_s,1-a_s)$
- D3DBLEND_DESTALPHA – blend factor = (a_d,a_d,a_d,a_d)
- D3DBLEND_INVDESTALPHA – blend factor = $(1-a_d,1-a_d,1-a_d,1-a_d)$
- D3DBLEND_DESTCOLOR – blend factor = (r_d,g_d,b_d,a_d)
- D3DBLEND_INVDESTCOLOR – blend factor = $(1-r_d,1-g_d,1-b_d,1-a_d)$
- D3DBLEND_SRCALPHASAT – blend factor = $(f,f,f,1)$ where $f = \min(a_s, 1-a_d)$
- D3DBLEND_BLENDFACTOR – predefined blend factor
- D3DBLEND_INVBLENDFACTOR – inverse of predefined blend factor

- The default values are
  - Src – D3DBLEND_SRCALPHA
  - Dest – D3DBLEND_INVSRCALPHA

Example 1

Draw the new image (source pixel) on top of the background image pixel (dest pixel)

d3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_ONE)
d3dDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ZERO)
Output = Source Pixel ⊗ Source Blend Factor ⊕ Dest Pixel ⊗ Dest Blend Factor
Output = Source Pixel ⊗ (1,1,1,1) ⊕ Dest Pixel ⊗ (0,0,0,0)
Output = Source Pixel ⊕ (0,0,0,0)
Output = Source Pixel

Example 2

Mix the new image (source pixel) and the back buffer image (destination pixel) using equal weights

d3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_ONE)
d3dDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ONE)
Output = Source Pixel ⊗ Source Blend Factor ⊕ Dest Pixel ⊗ Dest Blend Factor
Output = Source Pixel ⊗ (1,1,1,1) ⊕ Dest Pixel ⊗ (1,1,1,1)
Output = Source Pixel ⊕ Dest Pixel

Example 3

Assume that the new image has an alpha channel and that the alpha channel represents transparency level

0 – means transparent (colour 0)

1 – means opaque (colour 255)

d3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA)
d3dDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA)
Output = Source Pixel ⊗ Source Blend Factor ⊕ Dest Pixel ⊗ Dest Blend Factor
Output = Source Pixel ⊗ (S_a, S_a, S_a, S_a) ⊕ Dest Pixel ⊗ (1-S_a,1-S_a,1-S_a,1-S_a)

For example assume that $S_a = 0.3$ on all three channels (R, G, B) then output on each channel
Output = Source Pixel * 0.25 + Dest Pixel * 1-0.25
Output = Source Pixel * 0.25 + Dest Pixel * 0.75

The source pixel value is reduced by 75% and the back buffer pixel is reduced by 25%.

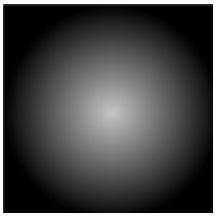
Transparency

- Transparency is obtained by using the alpha channel
- The alpha channel specifies the level of transparency
  - Black colour (colour 0) fully transparent (0%)
  - White colour (colour 255) completely opaque (100%)
  - Gray colour (colour 127) semi transparent (50%)
- Using the alpha channel to specify the transparency level is accomplished by specifying
  - Source – D3DBLEND_SRCALPHA
  - Dest – D3DBLEND_INVSRCALPHA

Alpha channel

- Represents the transparency level of the image.
- RGB determine the colours

Image greyscale



Alpha channel

Result

Alpha Source

- Default behaviour
  - The alpha value is taken from the texture (assuming the texture contains an alpha channel)
  - Otherwise the alpha is taken from the vertex colour
- Manual setting - Alpha channel can be explicitly specified
  - Vertex colour -
    d3dDevice->SetTextureStageState(0,D3DTSS_ALPHAARG1, D3DTA_DIFFUSE);
  - Alpha channel –
    d3dDevice->SetTextureStageState(0,D3DTSS_ALPHAARG1, D3DTA_TEXTURE);

Drawing images

- Sprites and textures use background colour in the images
  - The background colour should be invisible
- Setting the background colour to invisible can be done in two ways
  - Alpha channel
  - Specifying the background colour

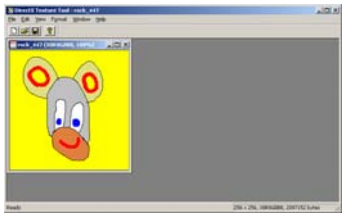
Removing background colour – alpha channel

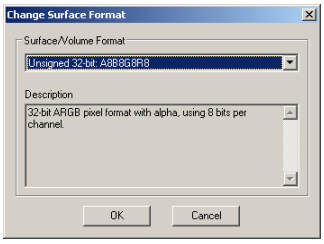
- Use DirectX Texture tool
- Create from the original image a black and white image such that
  - The background colour is coloured black
  - All other parts are coloured white
  - Add the black and white image as an alpha channel



Adding alpha channel steps

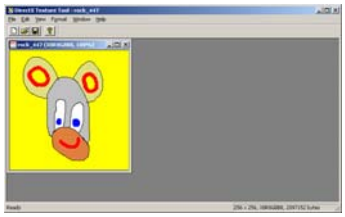
- Open the DirectX texture tool
- Open the image file
- Under the format menu item select “change surface format”
- Select the unsigned 32bits: A8R8G8B8 option

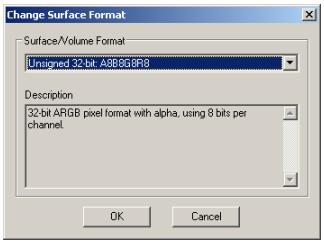




Adding alpha channel steps

- Under the file menu select “open onto alpha channel of this texture”
- Select the black and white image that was created
- Save the file
  - I usually use “save as” in order to keep the original file.
- One can see the alpha channel under the view menu item





Declaring the background colour

- This is done when the file is opened

```
HRESULT D3DXCreateTextureFromFileEx(
LPDIRECT3DDEVICE9 pDevice, -                    // the device
LPCTSTR pSrcFile,                               // the file name
UINT Width, UINT Height,
UINT MipLevels,
DWORD Usage,
D3DFORMAT Format,
D3DPOOL Pool,
DWORD Filter,
DWORD MipFilter,
D3DCOLOR ColorKey,                              // the background colour
D3DXIMAGE_INFO * pSrcInfo,
PALETTEENTRY * pPalette,
LPDIRECT3DTEXTURE9 * ppTexture                  // the texture structure
);
```



```
rc = D3DXCreateTextureFromFileEx(
d3dDevice,
L"rock_445.bmp",
0,0,5,0,
D3DFMT_UNKNOWN,
D3DPOOL_MANAGED,
D3DX_DEFAULT,
D3DX_DEFAULT,
D3DCOLOR_XRGB(255,255,0),
NULL, NULL,
&SpriteRock::spriteTexture[0]);
```

Sprites - Transparency

- Obtained using the draw function.
  - Manipulating the colour parameter by adding an alpha component

```
HRESULT Draw(
pTexture,                    // the loaded image
RECT pSrcRect,
D3DVECTOR3 pCentre,
D3DVECTOR3 pPosition,
D3DCOLOR colour              // add an alpha component
)

d3dSprite->Draw(myRockTexture,&r, &refPoint, &location,
    D3DCOLOR_ARGB(127,255, 255, 255)); // 50% transparent
```


Sprites - Transparency

- Placing or drawing the binary image on the screen

```
HRESULT Draw(
pTexture,                    // the loaded image
RECT pSrcRect,
D3DVECTOR3 pCentre,
D3DVECTOR3 pPosition,
D3DCOLOR colour
)

d3dSprite->Draw(myRockTexture,&r, &refPoint, &location,
    D3DCOLOR_XRGB(255, 255, 255));
```

```
d3dSprite->Begin(D3DXSPRITE_ALPHABLEND);    // begin sprite drawing


SetRect(&r,0,0,256,256);

// Draw at 50% transparency and 50% intensity
d3dSprite->Draw(myRockTexture,&r, &relPlace, &place,
     D3DCOLOR_ARGB(127,100, 100, 100));

d3dSprite->End();    // end sprite drawing
```


Texture Transparency

- Using a sequence of function calls
- Instructing the system to use blending
  - d3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, true);
- Setting up the source alpha
  - d3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
- Setting up the destination alpha
  - d3dDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);
- Draw the objects
- Disable the blending
  - d3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, false);

```
d3dDevice->SetTexture(0, this->myRockTexture);        // set the texture

d3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, true);  // enable blending

// set sources
d3dDevice->SetRenderState(D3DRS_SRCBLEND, D3DBLEND_SRCALPHA);
d3dDevice->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_INVSRCALPHA);

// render object
d3dDevice->SetStreamSource( 0, pVtxBuf, 0, sizeof(RockVertex) );
d3dDevice->SetFVF(D3DFVF_ROCK_VERTEX );
d3dDevice->DrawPrimitive( D3DPT_TRIANGLELIST, 0, 2 );

d3dDevice->SetTexture(0, NULL);  // unset the texture


d3dDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, false); // disable blending
```