# How to create a DirectX texture with a picture

This article explains how to create a DirectX texture with a picture loaded using a .NET API.

## Overview

DirectX typically uses the Windows Imaging Component API to load a picture. Unfortunately, this API is unavailable for Windows Phone 8.

This article explains how you can create the texture by loading the picture into a buffer in C# and passing it to C++ code (where it can be loaded into a texture).

## Solution

Solution has four parts :

### Read a picture to get its buffer data

To load a picture you must use BitmapImage class. This class can decode different image formats but it doesn't give an access to pixel.

You must convert this bitmap to a WritableBitmap class.

```
using System.Windows.Media.Imaging
...
BitmapImage bitmap = new BitmapImage();
bitmap.SetSource("images source");//load an image
WriteableBitmap bmp = new WriteableBitmap(bitmap);//convert BitmapImage to
WriteableBitmap
```

`WriteableBitmap` have three interesting properties :

- `PixelWidth` : picture width.
- `PixelHeight` : picture height.
- `Pixels` : pixels buffer.

Warning: Picture orientation of pixel buffer could be wrong. You can find a solution to get the true picture orientation of pixel buffer here: Ensuring correct orientation of loaded image.

### Give Buffer to C++ code

Once you load a picture and create a `WriteableBitmap`, you must give picture size and pixel buffer to C++ code. C# array is compatible with the C++/CX Platform::Array class.

First you must add a public function to your `WinPRT` component with equivalent parameters:

```
void func (const Platform::Array<int>^ buffer, int with, int height);
```

In the C++ code, the buffer pointer is accessible from the Platform::Array::Data property.

To call this function from your C# code, you just need to give the pixels array.

```
m_d3dInterop.CreateTexture(bmp.Pixels, bmp.PixelWidth, bmp.PixelHeight);
```

### Create a DirectX texture

To use a picture with DirectX, you must create a ID3D11Texture2D. Texture size and format is defined by a CD3D11_TEXTURE2D_DESC structure.


(C) Copyright Nokia 2014. All rights reserved.

```
CD3D11_TEXTURE2D_DESC textureDesc(
    DXGI_FORMAT_B8G8R8A8_UNORM, // Texture use ARGB pixel
    static_cast<UINT>(width), //picture width
    static_cast<UINT>(height), //picture height
    1,
    1,
    D3D11_BIND_SHADER_RESOURCE
   );
```

When you create a texture you can give pixel data. To do it, you must initialize a D3D11_SUBRESOURCE_DATA structure with pixel buffer and its memory organisation.

```
int pixelSize = sizeof(int);//pixel size. Each pixels are represented by a int 32bits.

D3D11_SUBRESOURCE_DATA data;
    data.pSysMem = buffer; //pixel buffer
    data.SysMemPitch = pixelSize *width;// line size in byte
    data.SysMemSlicePitch = pixelSize *width*height ;// total buffer size in byte
```

Now you can create the DirectX texture with ID3D11Device::CreateTexture2D method

```
m_d3dDevice->CreateTexture2D(
    &textureDesc, //texture format
    &data,          // pixel buffer use to fill the texture
    &m_Texture  // created texture
 );
```

## Image with alpha layer

WritableBitmap use premultiplied_ARGB format, i.e. RGB values are ever multiplied by alpha coefficient. This format is used to seep-up blending computation.

If you load a picture with an alpha layer you must remove alpha compensation on RGB values.

```
//use uint32 buffer
uint32 * uBuffer = (uint32 *)buffer;

//ARGB buffer
std::vector<uint32> ARGBBuffer(width*height);
//for each pixel
for (int i =0; i <width*height;++i)
{
 //extract alpha value
 uint8 a = uBuffer[i] >>24;
 //alpha = 0   => can't compensate RGB value
 //alpha = 255 => ARGB == premultiplied_ARGB
 if(a ==0 || a ==255)
 {
  ARGBBuffer[i] = uBuffer[i];
 }
 else
 {
  //compute alpha coefficient
  double aCoef = (uBuffer[i] >>24)/255.;
```

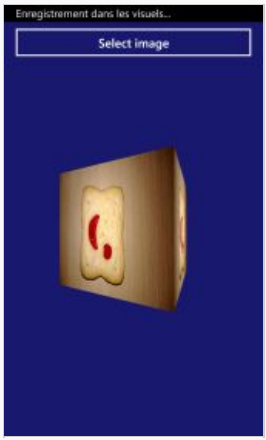

(C) Copyright Nokia 2014. All rights reserved.

```
  //extract RGB value and remove alpha compensation with alpha coeficient
  uint8 r = (uBuffer[i] >>16 & 0xFF) /aCoef +.5;
  uint8 g = (uBuffer[i] >>8 & 0xFF) /aCoef +.5;
  uint8 b = (uBuffer[i]  & 0xFF) /aCoef +.5;

  //recreate ARGB value to uint32
  ARGBBuffer[i] = (a <<24) + (r <<16) + (g <<8) + b;
 }
}
```

## Test code

**Texture created with an alpha layer**




**Texture created with a photo**

A code example/test code can be downloaded from here: Media:DisplayPictureDX.zip

The example app uses previous functions to generate a texture and displays a cube. A button is used to select a picture from your photo gallery.

## References

- Lesson 11: Direct3D 11 Textures (Braynzar Soft)
- Tutorial 26: Transparency (Raster Tek)


(C) Copyright Nokia 2014. All rights reserved.