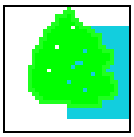
Color Key in Direct3D

Doug Rogers
NVIDIA Corporation
drogers@nvidia.com

Introduction

Colorkey is used as a method to eliminate texels from being rendered. If a color in a texture map matches the colorkey color, it is discarded, effectively making that part of the texture transparent.

For this tree, the colorkey is black. When the texture is rendered, the black part is not rendered, allowing what is behind to show through.



In this case a blue rectangle shows through the colorkeyed area of the texture. The actual polygon that is draw is shown as a black rectangle.

Colorkeys are specified the same format as the texture. If the texture is RGB format, then the coloykey value is also specified in RGB format.

Pixel Pipeline

If you examine the pixel pipeline in Direct3D, color key values are implemented using alpha transparency values.

This is the Direct3D Pixel Pipeline from the Device Driver Development Kit from Microsoft:

The pixel pipeline description shown below is how the hardware should be set up to conform to the Direct3D standard. States are set up and all the information is passed to the hardware which carries out the operations of the pixel pipeline. This only shows what the hardware does; there is no software component.

1 Stipple test (simple check against stipple pattern), scissor test
   - Failed test = don't draw pixel
2 Texture mapping, repeat for as many samples as required by step 3
   a) Lookup RGB texture value from q, u, v corresponding to x, y
   b) If the texture has an alpha channel, read the A value, otherwise set A to 1.0
   c) If chroma-key transparency is enabled and the RGBA texture color lies inclusively within the chroma key range, set the A value to 0.0
3 Perform bi- or tri-linear filtering operation on RGBA values
4 Perform alpha test. If test fails, reject pixel (updating interpolants)
5 Perform Z-buffer test
   a) If test fails, reject pixel (updating interpolants)
   b) If test succeeds, update Z-buffer value at x, y

6 Modulate (see **blending** below) texture color with diffuse color to get srcColor. If **MODULATE_ALPHA** is set, do the same for the alpha channel as well. Update color interpolants after this step

7 Add specular color to **srcColor**. The alpha channel is not affected by this stage. This stage may produce results greater than 1.0. Update specular color interpolants

8 Apply RGB fog. This stage may produce results greater than 1.0

9 Clamp colors to range [0.0,1.0)

10 Alpha blend **srcColor** with **destColor** to get **finalColor**

a) Alpha test **srcAlpha** value against reference value, if test fails, reject pixel

b) If stippled alpha, check alpha value against stippled alpha pattern, if test fails reject pixel

11 Convert **finalColor** to destination pixel format, applying dither pattern

12 Apply raster operation

newDest = src Rop dest

13 Write final color to destination at x, y

Colorkey Issues

From this we can see that colorkey is really alpha based. If the texture color matches the colorkey color, the alpha value is set to zero (step 2c). This means that we have some issues to contend with.

Issue one. What do we do with the color key color when we perform bilinear filtering?
Issue two. How do we control the blending of this pixel into the background?
Issue three. How do we combine alpha textures and color key?
Issue four. There is only one colorkey color control.
Issue five. How and when do I specify the colorkey color?
Issue six. The TNT implements colorkeying with alpha testing.

Issue one. What do we do with the color key color when we perform bilinear filtering?

The colorkey is filtered right along with the other texels. The implication of this is that you should either select a colorkey value that is not different from the surrounding pixels, is black, or you should discard a pixel that has any colorkey color in the texture.

Let's take a 2x2 color-keyed texture map with no alpha values in the texture map with 565 format.

C = colors
K = key

C1	K
C2	C3

The alpha values generated for this map from step 2b and 2c are:

1	0
1	1

Next we apply step 3. Let's assume that these four texels are combined into one pixel using bilinear filtering.

"the weighted average of the four texels around the pixel position are interpolated to determine what color the pixel is."

Let's assume an even weighting, so:

Pixel = (0.25 * C1 + 0.25 * C2 + 0.25 * K + 0.25 * C3)
Alpha = 0.25 * 1 + 0.25 * 1 + 0.25 * 0 + 0.25 * 1 = 0.75

Issue two. How do we control the blending of this pixel into the background?
We now have a pixel that has a colorkey component filtered in and a transparency value even though we did not have alpha values in the texture map.

We have several options:
1) Enable alpha blending and alpha blend the pixel into the background. This will require you to sort your colorkeyed textures like any transparent object. You might want to use a black colorkey color, but you might get a dark fringe around your textures.

2) Use alpha test to discard alpha values. The following are useful alpha test and reference values to achieve different results. Alpha blending and alpha testing will be functional even though there is no alpha component in the texture. This is another reason why we don't recommend colorkey.

To remove this pixel from being rendered, use the alpha test function (on texture with no alpha component) with the following values to cull pixels:

Value	comment
1.0	0xFF alphafunc to EQUAL Render the pixel when none of the four bilinear filtered texels is the colorkey value.
0.75	0xBF alphafunc to GREATER This will allow pixels to be rendered when one of the four texel is the colorkey value.
0.5	0x7F alphafunc to GREATER This will allow pixels to be rendered when two of the four texels are the colorkey value.

0.25 0x3F
alphafunc to GREATER
This will allow pixels to be rendered when three of the four texels are the colorkey value.

0.00 0x00
alphafunc to GREATER
This will discard only pixels where all four texels are the colorkey value.

If you do not enable alphatesting when you are using colorkey with a texture that has no alpha component, the device driver will enable it for you and use the default case of 0.00 listed above.

Issue three. How do we combine alpha textures and color key?
For textures that have an alpha component, the alpha is used from the texel for transparency and colorkeying is ignored.

Issue four. There is only one colorkey color control.
Colorkeying is included in DX6 as a legacy feature for DX5. There is no separate control for colorkey for each separate texture stage. There ***is*** a separate alpha blending stage. We strongly encourage you to use the alpha blending facilities (you are anyway) of DX6 instead of the old colorkey method. This can be accomplished by using a texture format that has an alpha component 1:5:5:5 for example. During texture load time, where there is a colorkey, set the alpha values to zero, otherwise set the alpha value to 1.

Issue five. How and when do I specify the colorkey color?
Because the colorkey is alpha based, you must set the colorkey color for the surface *before* you load data in it.

You can set the colorkey color for the surface in the following manner. The colorkey color is in the same format as the texture, so you need to convert it into that format. If the texture is palletized, the color key is an index. If the texture is RGB, the colorkey color must be an RGB value in the same pixel format as the testure.

```
if (ptcTexture->m_colorkeyed)
  {
   DDCOLORKEY          ck;
   DWORD key;

   key = texture_color(ptcTexture->m_colorkey.color);
   ck.dwColorSpaceLowValue  = key;
   ck.dwColorSpaceHighValue = key;

   res = ptcTexture->m_pddsSurface->SetColorKey( DDCKEY_SRCBLT, &ck );
}
```

```
/*
 takes a 32 bit color and converts it to a texel
 pf. Is pixelformat
*/
unsigned long texture_color(unsigned int color32)
{
  int r,g,b,a;
  unsigned long color;

  if (pf.red_scale == 0) // pixel format is not set
    return 0;
  a = (color32 >> 24) & 0xFF;  // get individual colors
  r = (color32 >> 16) & 0xFF;
  g = (color32 >> 8) & 0xFF;
  b = (color32) & 0xFF;

  r /= pf.red_scale;
  g /= pf.green_scale;  // divide them by the scale
  b /= pf.blue_scale;
  a /= pf.alpha_scale;

  if (pf.dwAlphaBitDepth == 0 || pf.dwRGBAlphaBitMask == 0)
    a = 0;

  // shift them to the right place and mask them
  r = (r << pf.red_shift) & pf.dwRBitMask;
  g = (g << pf.green_shift) & pf.dwGBitMask;
  b = (b << pf.blue_shift) & pf.dwBBitMask;
  a = (a << pf.alpha_shift) & pf.dwRGBAlphaBitMask;

   color = r | g | b | a;              // combine them all together

   return color;
}
```

Issue six. The TNT implements colorkeying with alpha testing.
This can have ramifications on your application. In step four above, the colorkeyed color is removed by alpha testing. You use the alpha testing to control how much how this alpha value is discarded.

If you enable alpha testing, and do not discard the value zero in the alpha test function then colorkeying will not be performed. If alpha testing is disabled, the driver will enable it to perform the colorkey.