Matrix Operations for Image Processing

Paul Haeberli

Nov 1993

Introduction

Four by four matrices are commonly used to transform geometry for 3D rendering. These matrices may also be used to transform RGB colors, to scale RGB colors, and to control hue, saturation and contrast. The most important advantage of using matrices is that any number of color transformations can be composed using standard matrix multiplication.
Please note that for these operations to be correct, we really must operate on linear brightness values. If the input image is in a non-linear brightness space RGB colors must be transformed into a linear space before these matrix operations are used.

Color Transformation

RGB colors are transformed by a four by four matrix as shown here:

```
xformrgb(mat,r,g,b,tr,tg,tb)
 float mat[4][4];
 float r,g,b;
 float *tr,*tg,*tb;
 {
    *tr = r*mat[0][0] + g*mat[1][0] +
                b*mat[2][0] + mat[3][0];
    *tg = r*mat[0][1] + g*mat[1][1] +
                b*mat[2][1] + mat[3][1];
    *tb = r*mat[0][2] + g*mat[1][2] +
                b*mat[2][2] + mat[3][2];
 }
```

The Identity

This is the identity matrix:

```
float mat[4][4] = {
    1.0,    0.0,    0.0,    0.0,
    0.0,    1.0,    0.0,    0.0,
    0.0,    0.0,    1.0,    0.0,
```

```
    0.0,    0.0,    0.0,    1.0,
  };
```

Transforming colors by the identity matrix will leave them unchanged.

Changing Brightness

To scale RGB colors a matrix like this is used:

```
  float mat[4][4] = {
      rscale, 0.0,    0.0,    0.0,
      0.0,    gscale, 0.0,    0.0,
      0.0,    0.0,    bscale, 0.0,
      0.0,    0.0,    0.0,    1.0,
  };
```

Where rscale, gscale, and bscale specify how much to scale the r, g, and b components of colors. This can be used to alter the color balance of an image.
In effect, this calculates:

```
        tr = r*rscale;
        tg = g*gscale;
        tb = b*bscale;
```

Modifying Saturation

Converting to Luminance

To convert a color image into a black and white image, this matrix is used:

```
  float mat[4][4] = {
      rwgt,   rwgt,   rwgt,   0.0,
      gwgt,   gwgt,   gwgt,   0.0,
      bwgt,   bwgt,   bwgt,   0.0,
      0.0,    0.0,    0.0,    1.0,
  };
```

Where rwgt is 0.3086, gwgt is 0.6094, and bwgt is 0.0820. This is the luminance vector. Notice here that we do not use the standard NTSC weights of 0.299, 0.587, and 0.114. The NTSC weights are only applicable to RGB colors in a gamma 2.2 color space. For linear RGB colors the values above are better.
In effect, this calculates:

```
        tr = r*rwgt + g*gwgt + b*bwgt;
        tg = r*rwgt + g*gwgt + b*bwgt;
        tb = r*rwgt + g*gwgt + b*bwgt;
```

Modifying Saturation

To saturate RGB colors, this matrix is used:

```
float mat[4][4] = {
    a,      b,      c,      0.0,
    d,      e,      f,      0.0,
    g,      h,      i,      0.0,
    0.0,    0.0,    0.0,    1.0,
};
```

Where the constants are derived from the saturation value s as shown below:

```
a = (1.0-s)*rwgt + s;
b = (1.0-s)*rwgt;
c = (1.0-s)*rwgt;
d = (1.0-s)*gwgt;
e = (1.0-s)*gwgt + s;
f = (1.0-s)*gwgt;
g = (1.0-s)*bwgt;
h = (1.0-s)*bwgt;
i = (1.0-s)*bwgt + s;
```

One nice property of this saturation matrix is that the luminance of input RGB colors is maintained. This matrix can also be used to complement the colors in an image by specifying a saturation value of -1.0.
Notice that when s is set to 0.0, the matrix is exactly the "convert to luminance" matrix described above. When s is set to 1.0 the matrix becomes the identity. All saturation matrices can be derived by interpolating between or extrapolating beyond these two matrices.
This is discussed in more detail in the note on Image Processing By Interpolation and Extrapolation.

Applying Offsets to Color Components

To offset the r, g, and b components of colors in an image this matrix is used:

```
float mat[4][4] = {
    1.0,    0.0,    0.0,    0.0,
    0.0,    1.0,    0.0,    0.0,
    0.0,    0.0,    1.0,    0.0,
    roffset,goffset,boffset,1.0,
};
```

This can be used along with color scaling to alter the contrast of RGB images.

Simple Hue Rotation

To rotate the hue, we perform a 3D rotation of RGB colors about the diagonal vector [1.0 1.0 1.0]. The transformation matrix is derived as shown here:
If we have functions:

identmat(mat)
that creates an identity matrix.
xrotatemat(mat,rsin,rcos)
that multiplies a matrix that rotates about the x (red) axis.
yrotatemat(mat,rsin,rcos)
that multiplies a matrix that rotates about the y (green) axis.
zrotatemat(mat,rsin,rcos)
that multiplies a matrix that rotates about the z (blue) axis.
Then a matrix that rotates about the 1.0,1.0,1.0 diagonal can be constructed like this:
First we make an identity matrix

```
  identmat(mat);
```

Rotate the grey vector into positive Z

```
  mag = sqrt(2.0);
   xrs = 1.0/mag;
   xrc = 1.0/mag;
   xrotatemat(mat,xrs,xrc);

   mag = sqrt(3.0);
   yrs = -1.0/mag;
   yrc = sqrt(2.0)/mag;
   yrotatemat(mat,yrs,yrc);
```

Rotate the hue

```
  zrs = sin(rot*PI/180.0);
   zrc = cos(rot*PI/180.0);
   zrotatemat(mat,zrs,zrc);
```

Rotate the grey vector back into place

```
  yrotatemat(mat,-yrs,yrc);
   xrotatemat(mat,-xrs,xrc);
```

The resulting matrix will rotate the hue of the input RGB colors. A rotation of 120.0 degrees will exactly map Red into Green, Green into Blue and Blue into Red. This transformation has one problem, however, the luminance of the input colors is not preserved. This can be fixed with the following refinement:

Hue Rotation While Preserving Luminance

We make an identity matrix

```
  identmat(mmat);

Rotate the grey vector into positive Z
   mag = sqrt(2.0);
    xrs = 1.0/mag;
    xrc = 1.0/mag;
    xrotatemat(mmat,xrs,xrc);
    mag = sqrt(3.0);
    yrs = -1.0/mag;
    yrc = sqrt(2.0)/mag;
    yrotatemat(mmat,yrs,yrc);
    matrixmult(mmat,mat,mat);

Shear the space to make the luminance plane horizontal
   xformrgb(mmat,rwgt,gwgt,bwgt,&lx,&ly,&lz);
    zsx = lx/lz;
    zsy = ly/lz;
    zshearmat(mat,zsx,zsy);

Rotate the hue
   zrs = sin(rot*PI/180.0);
    zrc = cos(rot*PI/180.0);
    zrotatemat(mat,zrs,zrc);

Unshear the space to put the luminance plane back
   zshearmat(mat,-zsx,-zsy);

Rotate the grey vector back into place
   yrotatemat(mat,-yrs,yrc);
    xrotatemat(mat,-xrs,xrc);
```

Conclusion

I've presented several matrix transformations that may be applied to RGB colors. Each color transformation is represented by a 4 by 4 matrix, similar to matrices commonly used to transform 3D geometry.
Example C code that demonstrates these concepts is provided for your enjoyment. These transformations allow us to adjust image contrast, brightness, hue and saturation individually. In addition, color matrix transformations concatenate in a way similar to geometric transformations. Any sequence of operations can be combined into a single matrix using matrix multiplication.

http://www.graficaobscura.com/matrix/