Color-based per-pixel blending of detail textures

Papavasiliou Dimitris
dpapavas@gmail.com

November 18, 2011

1 Introduction

Detail texture blending is a technique that can be used to add high frequency detail to a base texture when directly using higher resolution versions of this base texture is either impractical or not worth the cost in terms of added texture memory usage and bandwidth [1]. In many cases as for example when texturing terrain, more than one detail maps may be necessary to represent a usually small number of discrete categories of small-scale detail, such as grass, soil and rock. The solution to this problem, also known as splatting [2], has traditionally been to use a grayscale mask texture for each detail map to control the way it is blended over the base texture to produce the final result. These masks can either be automatically calculated based on analysis of the surface geometry (for example the height, slope or aspect of the terrain) or manually created by an artist.

The technique to be described in the next section is similar to the latter approach but instead of using additional mask textures for each detail map the base map itself is used. This eliminates the problem of creating separate masks as well as the added cost in terms of the needed memory bandwidth and GPU texture units. Perhaps the biggest gain though is that the blending function is evaluated per-pixel and the resolution of the transition between different detail maps is not bound by the resolution of the used mask textures.

This presentation does not imply that the technique is novel, it may have been employed by others. In fact the notion of using the base texture color to select detail maps seemed to the author so straightforward that it was taken for granted that a quick research on the internet would yield the details needed to carry it out. The fact that this did not in fact prove true was the main incentive for publishing this work in the hope that it will be useful to others.

2 Blending based on color

The basic idea of the method is the following. Instead of using masks to define how much of each texture to blend into a given fragment of the terrain the color of the base texture is used. A color is assigned to each detail map used, for example some shade of green for grass, a shade of brown for soil, gray for rock and so on. Then, when rendering the terrain, at each output fragment each detail map is blended with a contribution inversely proportional to the 'distance' of the base texture color from the color assigned to the detail map in question. Conceptually this would mean that where the color of the base map is green the prevailing detail texture would be that of grass but as the base texture fades towards brown say so would the detail map fade towards a mixture of grass and soil and finally towards mostly soil. Smooth transitions in the base map ensure smooth transitions between detail textures and one can utilize as many detail textures as the number of texturing units available.

In terms of artistic control the basic advantage of this method is that it is very easy to use and extremely versatile: the base map allows the definition of the general layout of the terrain texture and can either be created by hand or sourced from aerial and satellite imagery. Once this is defined any number of detail maps can be used, provided that the hardware has enough texturing units of course, but older hardware can be supported by leaving out detail maps based on the number of units available. For example in addition to the basic set of soil, grass and rock detail maps one could use a separate map of sparse grass and assign it to a green-brown shade halfway between the colors assigned to grass and soil. By varying the color as well as the power parameter that will be explained in the next section, an infinity of variations can be achieved. Using the same set of maps the terrain can be made to look more grassy or more barren by changing the colors assigned to each detail map (also making it possible to provide multiple 'skins' of the same terrain by specifying different color configurations). By changing these colors in real-time it would be possible to animate the terrain shading. The result is almost guaranteed to be aesthetically pleasing (provided suitable base and detail maps are used) since at every point all maps are mixed in a smooth and controllable fashion. The only downside is that you lose some control over the way the textures are applied since you do not specify it directly.

3 Detail texture synthesis through multivariate interpolation

As it turns out the method is in fact a straight application of an existing mathematical method in the field of multivariate interpolation named inverse distance weighting [3]. Let us therefore cast the problem into a more mathematical formulation.

Let f be a function $f : R^3 \rightarrow R^N$ mapping three dimensional points representing colors to N-dimensional points representing weights for N discrete detail maps. That is the value of f is a vector with each component lying in $[0, 1]$ and defining how much one of the detail maps contributes to the final detail texture. Given then the color $\mathbf{x_b} \in R^3$ of the base map at some point and the colors $D_i,\ i = 1, ..., N$ of each detail map at this point it is straightforward to calculate the final blended color C as

$$C = \sum_{i=1}^{N} f_i(\mathbf{x_b}) D_i \tag{1}$$

In the equation above f_i is the i-th component of the value of f, which is the weight of the i-th detail map.

In order to calculate the needed mapping f we specify an arbitrary number of known points for it. In general one can specify values for f with arbitrary weights for each detail map but in the current implementation it was assumed that each point specified would have one component equal to one and all others equal to zero. In other words the initial values map colors to pure detail maps instead of mapping them to mixtures. This choice was made for the sake of simplicity but it can readily avoided should more control over the blending be required.

According to the inverse distance weighting method, given a set of initial values u_i for the mapping f on some points $\mathbf{x_i}$ the values on any other point can be interpolated through the equation

$$u(\mathbf{x}) = \sum_{i=1}^{N} \frac{w_i(\mathbf{x}) u_i}{\sum_{j=1}^{N} w_j(\mathbf{x})} \tag{2}$$

where

$$w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^p} \tag{3}$$

Here $\mathbf{x}$ denotes an interpolated (arbitrary) point, $\mathbf{x_i}$ is an interpolating (known) point, d is a given distance metric from the known point $\mathbf{x_i}$ to the unknown point $\mathbf{x}$, N is the total number of known points and p is a positive real number, called the power parameter.

In our case, where f maps colors to detail map mixing weights, what this equation means intuitively is that for some point on the terrain the weight of each detail map is the inverse of the distance between the color assigned to the detail map and the color of the base map at this point so that a detail map contributes more to the final result over the areas of the terrain where the base color is closer to the color assigned to it. The normalization of each weight through division by the sum of all weights ensures that the final blended detail map will have the same mean brightness as the input detail maps.

The power parameter is particularly interesting as greater values of p assign greater influence to values closest to the interpolated point resulting in sharper separation of the detail maps and more well-defined map transitions. Furthermore as the weighting function is evaluated per-pixel the definition of the transitions is not bound by the resolution of the base map so the latter needn't be overly large (assuming it is sampled using bi-linear interpolation). A base map at a 1 m resolution for example only allows you to directly specify the mixture of detail maps that is to be applied every 1 m but since the mapping will be evaluated per pixel yielding a different set of detail map weights based on the interpolated base map values one can get pleasantly sharp detail map transitions by making p sufficiently high.

The distance function used was the usual Euclidean distance metric but instead of specifying the colors and calculating their distance in the RGB color space the HSL color space [4] was chosen. This choice was made on the very scientific basis that 'it seemed reasonable' given that the HSL color space is more intuitive in terms of what a distance along each axis means. It might be possible to use the RGB color space instead which would incur a minor saving on the calculations needed for the conversion but the author didn't have much luck with a quick implementation.[1]

Apart from the fact that the colors assigned to the detail maps have to be specified in the HSL space and that the color sampled from the base map has to be transformed into that space in order to calculate the distance there's an additional implication of this choice that should be noted. The HSL space is cylindrical since the hue represents an angle. When calculating the distance this should be taken into account as the difference between a hue of 350° and a hue of 20° is 30° and not 330°. Using the Euclidean metric the distance between two colors (h_a, s_a, l_a) and (h_b, s_b, l_b) can be calculated as

$$d = \sqrt{\left[\min\left(h_a - h_b, 1 - h_a + h_b\right)\right]^2 + (s_a - s_b) + (l_a - l_b)} \tag{4}$$

4 Implementation details

Once the detail map for the given terrain fragment has been calculated there are a number of possibilities for blending them onto the terrain such as using additive blending as in

$$f = b + d - 0.5 \tag{5}$$

where f is the final pixel color, b the base color and d the detail map color. That would work with both color and grayscale detail maps. Assuming monochrome detail maps one could simply use some form of multiplicative blending such as

$$f = bd \tag{6}$$

In the case of color detail maps another possibility (which is the one chosen by the author) is to use the luminance of the base color to provide intensity variation to the detail maps as well as baked in lighting if the base map is lit:

$$f = \frac{b_r + b_g + b_b}{3} d \tag{7}$$

[1]It seemed to work in general but it proved difficult to map gray to a detail map such as rock or asphalt. The grass kept creeping into the road.

There are some possible modifications to the scheme outlined so far. For one, one could modify the distance metric by weighting each of the axes, which correspond to hue, saturation and lightness, with an arbitrary weight. This would allow one more control since it would be possible to specify how important each component of the input colors is. For example when mapping a shade of green to a map of grass specifying a higher weight for hue would make it more significant by assigning larger distances to points that differ by some distance in hue than to points that differ by the same distance in saturation or lightness. This allows more fine-tuning based on intuitive color qualities and was one of the reasons the HSL color space was chosen. Although initially implemented the author chose to disallow user-setting of weights for the sake of simplicity after observing that you can do quite well without them.

Another area that deserves more exploration is the weighting function used. Instead of using the inverse Euclidean distance one could, for example, assign a weight of one to the detail map with the least distance and zero to all others. This would yield a sharp mixing of detail maps suitable for cel-like shading. Of course a function with more quantization steps would be preferable.

Finally it should be noted that this method could scale well to more levels of hierarchy. For example for a flight simulator it should be possible to use two (or more) palettes of detail maps one with geotypical detail that is visible from a relatively low height such as fields, cities and the like and one with the usual close-range detail, that is grass, rock, etc. One could then use the base map to blend the first palette and the result to blend the second and blend these yet again based on the height to get the final fragment color. Of course when the viewing point is high enough only the first palette needs to be used.

An implementation of the algorithm using GLSL is given in the appendix.

5 Results

The web page cited in [5] contains images as well as a few videos captured from a motorcycle simulation where this method is employed. Although the base texture and detail maps employed are quite rudimentary and hardly any time has been taken experimenting with the colors assigned to the detail maps which guide the process of mixing they still allow the basic qualities of the method to be assessed. There is probably a lot of room for improvement though, through proper fine-tuning of the various parameters as well as the base map.

References

[1] Detail textures
http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node88.html

[2] Terrain Texture Compositing by Blending in the Frame-Buffer
http://www.cbloom.com/3d/techdocs/splatting.txt

[3] Inverse distance weighting
http://en.wikipedia.org/wiki/Inverse_distance_weighting

[4] HSL and HSV color spaces
http://en.wikipedia.org/wiki/HSL_and_HSV

[5] Color-based per-pixel blending of detail textures sample page
http://www.nongnu.org/techne/research/blending

A GLSL source code

The implementation is quite straightforward and simple, although there are a few details such as converting the base texture texel to the HSL color space and computing a distance in this color space which is cylindrical. The following listing contains the code used by the author to implement this shader in GLSL. The hue in this implementation has been remapped to lie in the range [0,1) although it is customarily given in degrees. Also in the code below `N` is an integer constant holding the number of detail maps used, `pigments` is a uniform array of N 3-vectors holding the colors assigned by the user to each map and `detailScale` is a uniform array of N 2-vectors specified by the user to control the scale of each detail map.

```
/* Sample the base map. */

texel = vec3(texture2D(sampler, gl_TexCoord[0].st));
```

```
/* Convert texel to HSL. */

M = max(max (texel.r, texel.g), texel.b);
m = min(min (texel.r, texel.g), texel.b);
C = M - m;

/* If C is zero the color is fully desaturated so
   its hue is undefined.  We just set it to zero. */

if (C > 0.0) {
    if (texel.r == M) {
        H = mod((texel.g - texel.b) / C, 6.0) / 6.0;
    } else if (texel.g == M) {
        H = ((texel.b - texel.r) / C + 2.0) / 6.0;
    } else {
        H = ((texel.r - texel.g) / C + 4.0) / 6.0;
    }
} else {
    H = 0.0;
}

hsv = vec3(H, C / M, M);

/* Now compute the distance between the base texture
color and the color assigned to each detail map. Also
comput the sum of the distances and store it in C. */

for (i = 0, C = 0.0 ; i < N ; i += 1) {
    vec3 v;

    v = hsv - pigments[i];

    /* Take into account that v.x is the hue
    and hence an angle so we want the modular
    distance between the two numbers. */

    v.x = min(v.x, 1.0 - v.x);
    distances[i] = 1.0 / pow(dot(v, v), power);
    C += distances[i];
}
```

```
/* Weight each detail map based on its normalized
distance and sum to get the final detail texel. */

for (i = 0, detail = vec3(0.0) ; i < N ; i += 1) {
    detail += distances[i] / C *
              texture2D(detailSampler[i],
                        detailScale[i] * gl_TexCoord[0].st).rgb;
}

/* Modulate the detail texel by the luminance of
the base map to get the final value. */

result = (texel.r + texel.g + texel.b) / 3.0 * detail;
```