Blend Maps: Enhanced Terrain Texturing

ALEXANDRE HARDY
University of Johannesburg
and
DUNCAN ANDREW KEITH MC ROBERTS
University of Johannesburg

Very large textures are prohibitive for most interactive terrain rendering due to the use of excessive resources. Alternatively, tiled textures can be linearly blended to produce interesting transitions between predefined textures. However, these transitions seldom work well when the texture frequency content is not high. This paper introduces blend maps, a technique used to enhance the blending of textures so that features of the underlying texture are maintained or emphasised. The resulting texture appears more realistic in many cases, and adds to the perceived detail of the terrain. Blend maps are constructed by identifying important features in a texture. These features are often highlighted under certain lighting conditions. The blend maps are created manually, but some automation is possible. The importance is then used to control a blending operation in such a way that features can be preserved. The proposed algorithm is simple enough to allow realtime rendering on modern graphics hardware.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Colour, shading, shadowing, and texture*

General Terms: Algorithms

Additional Key Words and Phrases: Blending, texturing, terrain texture, texture tiling

1. INTRODUCTION

Texturing is an important technique to increase the realism of rendered objects. Texturing is particularly important for the realtime rendering of terrain. The textures assist in conveying geographical and vegetation information as well as a sense of motion for moving cameras due to optic flow. Terrain data sets are often very large, with accompanying textures. The use of large textures often results in large penalties during rendering. For synthesised terrain, a smaller set of textures may be blended to create variable texture depending on the attributes of the terrain. Linear blending provides fast, smooth transitions between texture maps, but when viewed up close, the blending appears artificial. Specifically created blending tiles consume extra texture memory, and limit the range of transitions that can practically be implemented. This paper introduces blend maps to increase the control and observed richness of tiled textures as applied to terrain rendering. The advantages of blend maps are discussed, as well as the impact on performance. Computation of blend coefficients are also addressed.

The application of texturing to terrain rendering, and texturing techniques that encode features of the image are discussed in the next section, followed by a discussion of our method.

2. RELATED WORK

Clipmaps [Tanner et al. 1998] provide an efficient technique for displaying very large textures. However, a large amount of texture memory is used, and the large texture must be obtained or created before use. Shenas and Interrante [Shenas and Interrante 2005] introduce compositing techniques for combining colour and texture. One of the applications is to colour terrain according to some predefined data. The technique is well suited to visualising data, but poor for realistic terrain rendering. Döllner et al. [Döllner et al. 2000] present a flexible technique for texturing terrain with multiple textures. The technique presented allows portions of different textures to be viewed selectively over different portions of terrain. However, the texture covers the entire terrain and is not built from tiles.

A. Hardy, Academy for Information Technology, University of Johannesburg, P.O. Box 524, Auckland Park, 2006, South Africa, ahardy@uj.ac.za
D.A.K. Mc Roberts, Academy for Information Technology, University of Johannesburg, P.O. Box 524, Auckland Park, 2006, South Africa

Linear blending has been used in various applications to control the transitions between images or textures. Texture splatting [Bloom 2000] is an application of linear blending to the smooth transition between tiled textures in terrain. More recently, this approach has been implemented efficiently with modern graphics hardware [Luna 2003]. Linear blending makes no attempt to ensure that features are consistent or visible.

Nailboards [Schaufler 1997] assign a depth value to each location in the texture, so that textures mapped onto surfaces are correctly hidden according to their depth values. This approach is primarily of interest for impostors (quadrilaterals that are rendered in place of more complex geometry). Relief textures [Oliveira et al. 2000] add an orthogonal displacement to each pixel of a texture, that allows textures to be rendered with view motion parallax that enhances the three dimensional appearance of the texture.

Controlling the blending of textures can significantly improve the recognition of features [Baudisch and Gutwin 2004]. For terrain texturing we would like to include some kind of importance for each feature in the texture map that will allow blending while still keeping features to some extent. A natural candidate would be the texton map [Guo et al. 2001] or texton channel [Malik et al. 1999] to specify the features of a texture map. The texton map has been applied successfully to control procedural texture synthesis so that features of the texture are preserved [Zhang et al. 2003; Wu and Yu 2004].

Recent work in realtime texturing of terrain includes the work of [Lefebvre and Hoppe 2005] for the procedural synthesis of terrain texture in realtime. The techniques presented can easily be applied to the work presented here. The work of Lai et al. [Lai et al. 2005] has many similar features to our work, except that our algorithm is applied in realtime whereas Lai et al. compute blend textures beforehand. The algorithm they present identifies the features of an image and provides a visually pleasing border between tiled textures on the terrain. The technique is limited by the number of textures that are precomputed, and so the visual richness of the terrain may be reduced. The textures that blend between regions also require extra texture memory, over and above the tiles already in use. Neyret and Cani [Neyret and Cani 1999] create an aperiodic tiling based on triangular textures. The triangular textures must be generated or provided and satisfy certain boundary constraints. In other words, transitions should be provided with the triangles. Triangulated terrain may be textured using this algorithm [Neyret and Cani 1999]. Lefebre and Neyret [Lefebvre and Neyret 2003] use modern graphics hardware to produce an aperiodic grid of tiles which can be used for terrain rendering in realtime. The transitions in their scheme are created using pixel maps or transition tiles. Transition tiles could be produced by the technique of Lai et al. [Lai et al. 2005] whereas the pixel map provides a mask determining sharp boundaries between different textures. In contrast, blending helps to soften the transition at the expense of features.

In the following sections we present our realtime blending algorithm for terrain, in which we attempt to preserve the features of underlying textures.

3. THE BLEND MAP

It is important in terrain texturing to allow features to be evident. For example, we consider pebbles and grass. The pebbles in a terrain texture are the most important part. That is, when viewing pebbles, the shape and prominence of the pebble is of more importance to us than whether the pebbles are lying on sand or grass. We thus assign an importance value for each sample in the texture map. This importance value will determine how prominent the sample is when textures are blended. The importance value is stored in the alpha channel of the texture map. This importance map will be used to control blending, and so the alpha channel will be referred to as the blend map from this point on. We will not attempt to address the computation of importance of visual features, but rather assume that this is an artist selected attribute. A few simple examples of how these attributes may be computed are presented in a later section. An example of a pebble texture and corresponding blend map are shown in figure 1.

Now, given that pebbles are important, but the space in between the pebbles is less important, we would assume that blending pebbles with grass (or sand) will leave the pebbles relatively untouched and the spaces in between would be filled by grass (or sand). We could take the process a step further, and regard some pebbles as more important than others. On the other hand, if grass becomes more important, less pebbles will be visible. From this example, it is evident that the geometry must have an importance value assigned for each texture. Gradual changes in importance of textures will produce gradual changes between different environments.

Now that importance values have been assigned we have to determine how these values may be used to control blending. In the next section we address this problem for two textures, which is a useful specialisation, followed by a generalised algorithm for blending an arbitrary number of textures with their blend maps.

Figure 1. Alpha channel of the pebble map and result of blending.

Figure 2. Comparison between linear blending and blend maps with $k = 1$ (blend maps below).

4. BLENDING BETWEEN 2 TEXTURES

Traditional linear blending between two textures is performed using the equation

$$\mathbf{c}(u, v) = \alpha\mathbf{d}(u, v) + (1 - \alpha)\mathbf{s}(u, v),$$

where $\mathbf{c}$ is the final colour, $\mathbf{s}$ is the first texture, $\mathbf{d}$ is the colour of the second texture and $0 \leq \alpha \leq 1$ controls the transition between the two textures. This form of blending pays no attention to the relative importance of each texel. The relative importance of a texel in $\mathbf{d}$ with respect to $\mathbf{s}$ can be represented as $d_\alpha - s_\alpha$. The importance gives an indication of how prominent the texture is and can be used directly as a blending value. We may wish to emphasise the relative importance, and so we use the following formula to compute the blending factor:

$$\alpha(u, v) = \text{clamp}(b_\alpha + k(d_\alpha - s_\alpha), 0, 1).$$

The factor k determines the strength of the importance in determining the final weight, and b_α determines the relative importance of the texture $\mathbf{d}$. The useful range of b_α varies according to the two textures. For the examples presented here $-1 < b_\alpha < \frac{3}{2}$ proves to be sufficient. The colour is given by

$$\mathbf{c}(u, v) = \alpha(u, v)\mathbf{d}(u, v) + (1 - \alpha(u, v))\mathbf{s}(u, v).$$

We note that this differs from alpha blending directly on graphics hardware, since the relative weight depends on both textures, and the sum of the weights of the textures is 1. If we were to simply use the alpha values in the textures directly, we do not have a guarantee that the sum of the weights is 1. It is important in terrain rendering that the weights sum to 1 so that definite ground composition can be determined. Even though sand may not be very important, if there is nothing but sand then we should still see the sand. Figure 2 provides a comparison between linear blending and blend maps.

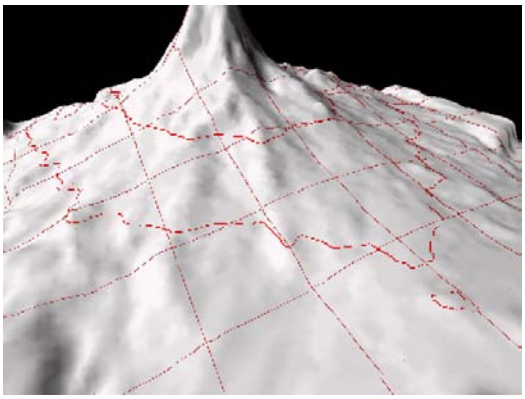


Figure 3. Errors in mipmap selection (incorrect mipmap level in red).

4.1 Texture atlasses

When using only two textures, the textures to be blended must be selected from many textures representing terrain. Current graphics hardware cannot select the texture to render from a list of textures. Instead, the textures are packed into a texture atlas [Wloka 2005]. Two options are available: a two dimensional texture atlas, or a one dimensional atlas. By packing the textures horizontally into a one dimensional texture atlas, we only need one texture coordinate to identify a particular ground cover texture. The texture coordinates for a subtexture in the texture atlas is computed using (for a 2D texture atlas)

$$b_u = (\text{frac}(t_u) + c_i)/n_w,$$
$$b_v = (\text{frac}(t_v) + r_i)/n_h,$$

where n_w (n_h) are the number of textures horizontally (vertically), c_i and r_i are the column and row indices of the subtexture and t_u and t_v are the original texture coordinates. The function frac is defined as $\text{frac}(x) := x \bmod 1.0$.

A mipmap produced directly from the texture atlas produces cross-pollution in the subtextures [Wloka 2005]. The mipmap levels are thus generated separately for each subtexture, and these individual mipmap levels are packed into one mipmap level for the texture atlas. In addition to these problems, the texture hardware uses the change in texel coordinates in a 2×2 pixel quad to select the mipmap level for texturing. When computing texture coordinates for a subtexture, we compute the wrapped texture coordinates $\text{frac}(t_u)$ and $\text{frac}(t_v)$. Tiled textures may result in extreme distances between texels at the boundary. For example, two interior texture coordinates 0.11 and 0.21 have distance $|\text{frac}(0.21) - \text{frac}(0.11)| = 0.1$. At the border of a tile we might have coordinates 0.91 and 1.01, giving a distance of $|\text{frac}(1.01) - \text{frac}(0.91)| = |0.01 - 0.91| = 0.9$. This rapid change in texture coordinates at the boundary of a tiled subtexture causes the graphics hardware to select the wrong mipmap level (illustrated in figure 3). Using the functions `ddx` and `ddy` available on modern graphics hardware, the mipmap level can be manually computed, with a penalty in frame rate.

Texture filtering can also produce cross-pollution in the subtextures. When texture samples are very close to the border of the subtexture, neighbouring subtextures may be sampled for the purpose of filtering. Adding a border around the texture (created by tiling the texture) alleviates this problem [Wloka 2005].

4.2 Selection of subtextures

The previous section discussed the retrieval of textures from the texture atlas. In this section we discuss the selection of textures to be used, as well as the relative weights for the textures. We will not concentrate on the calculation of weights, but postpone that discussion for a later section.

We have chosen to use the gradient and altitude to select textures and weights. To simply use these values directly to determine texture coordinates would limit the use of subtextures. Instead, the values are used to perform a texture lookup in a texture map (the index texture). The contents of the index texture are as follows:

— Offset in texture atlas for first subtexture.

— Offset in texture atlas for second subtexture.

— The weight b_α for controlling the transition between the two textures.

— The parameter k, used to control the strength of the importance value.

Either a 32-bit per pixel texture map can be used, or a 128-bit per pixel texture map. We have opted for the 128-bit per pixel floating point texture map. The size of the index texture can have a significant impact on performance, so it is important that the texture not be too large. Performance results for different index texture sizes are presented in the results section. Naturally, the index texture should not be filtered.

Although it is often sufficient to blend between only two textures, in some circumstances it is desirable to blend between more than two textures. This is the topic of the next section.

5. BLENDING BETWEEN N TEXTURES

When blending multiple textures, the relative importance is not as simple to calculate. In particular, it is very difficult to extend the formula for $\alpha(u, v)$ to many textures using addition or subtraction. The solution is to use a slightly less efficient measure of the relative importance related to all the other textures simultaneously. To begin we compute the total importance of all samples:

$$t_\alpha(u, v) = \sum_i b_{i,\alpha} m_{i,\alpha}(u, v)^{k_i},$$

where $b_{i,\alpha}$ is the computed importance of the texture map, and $m_{i,\alpha}$ is the value in the blend map for texture $\mathbf{m}_i$. The importance of each texel, is the importance stored in the blend map multiplied by the importance of the texture. The factor k_i provides another parameter which can control the strength of a particular texture. This parameter helps to compensate for blend maps that have the correct distribution, but the importance values are not necessarily correct in relation to the other textures. Using the value t_α, the weights can be normalised so that they sum to 1. We compute

$$\alpha_i(u, v) = b_{i,\alpha} m_{i,\alpha}(u, v)/t_\alpha(u, v),$$

for each texture i. It is not necessary to clamp the α_i, as long as the $b_{i,\alpha}$ values are not negative. The final colour is a weighted average of the input textures:

$$\mathbf{c}(u, v) = \sum_i \alpha_i(u, v)\mathbf{m}_i(u, v).$$

Modern graphics hardware can rapidly blend textures using this formula. One of the remaining questions is the choice of the blend coefficients $b_{i,\alpha}$.

6. CREATING BLEND MAPS

Creation of blend maps is not an automatic process. It depends on the application and features that are regarded as important. Nevertheless, it is possible to obtain reasonable blend maps using conventional image editing software. For terrains, we observe that height often correlates with importance. However, the algorithm presented does not simply do a depth comparison, rather a smooth blend between the textures is performed based on importance value (which may be depth).

The colour of objects also sometimes indicate importance. In the pebble texture map, pebbles are brighter than the material in between, and so luminance can be used as a measure of importance. In the case of grass, we find that the grass blades that are higher usually receive more light, and so luminance is an indication of the depth at each texel.

A combination of techniques were used to produce the blend maps presented. Typical steps in the production of a blend map are:

— Convert the texture to a gray scale image.

— Threshold the values.

— Apply a Gaussian filter to smooth the blend map.

— Adjust brightness and contrast to produce the desired range of importance values.

Some steps may be omitted. For example, the pebble image benefits from the threshold operation, whereas the grass image does not. The Gaussian filter reduces the clear boundaries around texture features, but it also helps to smooth the transition between features so that lighting inconsistencies or other problems may be reduced. For example, grass blades often have shadows beneath them. If we omit the Gaussian filter operation, then blend maps fail to capture the shadow under the blade of grass. These simple techniques were relatively effective in the sample applications. In practice, more sophisticated techniques may be necessary.

7. SELECTION OF BLENDING COEFFICIENTS

The selection of blend coefficients can be left to the artist or programmer and stored in a texture. This may be very expensive for large terrain. We have chosen to provide a formula for computing blend coefficients for blend maps. There are a number of parameters that control how prominent a particular texture is. The programmer controls

— The altitude at which the texture starts, a_b.
— The altitude at which the texture ends, a_e.
— The gradient at which the texture starts, g_b.
— The gradient at which the texture ends, g_e.
— The importance of altitude, c_a.
— The importance of gradient, c_g.

The altitude and gradient are independent parameters. That is, the texture does not only appear where the altitude meets the criteria and the gradient meets the criteria, but rather, where the altitude meets the criteria or the gradient meets the criteria.

The altitude weight should reach a maximum at $m = 0.5(a_b + a_e)$ and should be zero at the ends. A smooth interpolation is also desirable. We thus use the function

$$w_a(alt) = \text{smoothstep}(a_b, m, alt) \times \text{smoothstep}(-a_e, -m, -alt),$$

where the function smoothstep is given by

$$\text{smoothstep}(a, b, x) = t^2(3 - 2t), \quad \text{with} \qquad t = \text{clamp}\left(\frac{x - a}{b - a}, 0, 1\right).$$

The effect of multiplying two smoothstep functions results in a curve that peaks in the centre. By construction, both w_a and smoothstep are C^2.The cubic polynomial provides a smooth transition between the edges and midpoint of the interval. The altitude *alt* is a value between 0 and 1, where 0 is the minimum altitude and 1 is the maximum altitude. The altitudes should be scaled into this range for the purposes of this computation, by either the CPU or vertex shader.

To compute the gradient, we use the normal provided by the application. We will use gradient values from 0 to 1, where 0 is perfectly level and 1 is a 90° cliff. The gradient is computed using the formula

$$grad = 0.5 - \frac{n_y}{2},$$

where $\mathbf{n}$ is the surface normal, and the y-axis points upwards. This formula also allows downward facing polygons and thus the range of n_y is $[-1, 1]$. As long as the normals are of unit length, we have the desired values for the gradient. We then compute the gradient weight w_g in the same way as the altitude weight.

Once the weights for each component have been identified, the weights are multiplied by the importance coefficients to provide a final weight for the texture:

$$w = c_a w_a + c_g w_g.$$

These weights are then passed through to the blending stage (in the fragment shader).

8. FILTERING

One of the concerns with blend maps is the effect of mipmapping and other filtering techniques on the importance values stored in the mipmap. During mipmapping and filtering, a weighted average of the colours and alpha values are produced. It seems sensible that a weighted average of the importance be used. The results confirm that existing filtering and mipmapping techniques do not decrease the quality of results produced by blend maps.

9. RESULTS

Blending coefficients can be precomputed and stored in texture coordinates. Alternatively, the vertex shader can compute these values in realtime. We have implemented both techniques and found no visible difference in performance. We can thus avoid the overhead of storing these values in texture coordinates, and rather use the texture coordinates to allow more textures to be blended.

The blend maps algorithm was implemented in a fragment shader and compared to the performance of conventional linear blending (implemented as a fragment shader) and direct use of the multi-texturing hardware (no

	Blend maps	Linear blending	Fixed function pipeline
Low detail terrain (Radeon 9600 128Mb)	34 fps	35 fps	39 fps
High detail terrain (Radeon 9600 128Mb)	7 fps	7 fps	8 fps
Low detail terrain (GeForce 7800 GTX 256Mb)	365 fps	375 fps	595 fps
High detail terrain (GeForce 7800 GTX 256Mb)	54 fps	55 fps	103 fps

Table I. Performance of blending 4 textures.

Blending method	Average frame rate	Relative frame rate
No blending	380	100%
Linear blending	340	89%
Blend maps	321	84%

Table II. Performance of blend maps with two textures (GeForce 7800 GTX 256Mb).

Texture size	Average frame rate	Relative frame rate
64x64	330	100%
128x128	321	97%
256x256	294	89%
512x512	228	69%

Table III. Impact of index texture size (GeForce 7800 GTX 256Mb).

shader programs). These tests were run on an AMD Athlon 2600 CPU with an ATI Radeon 9600 (128Mb) video card. The program was also tested on an AMD Athlon 3800 CPU with an NVIDIA GeForce 7800 GTX (256Mb) video card. No lighting is applied, only the respective texturing algorithms. All frame rates are listed for a resolution of 1024×768. The textures used are displayed in figure 4, each texture was loaded into a separate texture unit. The results in table I indicate that there is little difference between the performance of linear blending and blend maps, and also that the performance degrades gracefully.

Blending only two textures can potentially speed up the terrain rendering. The following results for two textures, stored in a texture atlas, were obtained on an AMD Athlon 3800 CPU with an NVIDIA GeForce 7800 GTX (256Mb) video card. For these tests, lighting was also computed and frame rates are measured at a resolution of 1280×1024. Specular lighting and texture filtering were disabled. Table II shows the impact of blend maps. Blend maps are slower than linear blending, but still within realtime rates. No blending is significantly faster, since only one texture is sampled. The effect of the index texture size is illustrated in table III. An index texture of 128×128 provides a good compromise between accuracy and speed. For these examples, the index texture was produced by a program using a simple algorithm.

Figure 5 compares linear blending and blend maps with four textures sampled per pixel. The textures used for these images are displayed in figure 4. Blend maps allow the sand to be visible between the grass blades, although shadows are not taken into account. The grass in a real photo (figure 7, a mixture of mud and grass) seems to match blend maps slightly more closely than linear blending. However, if the application requires something similar to the dark portions created by linear blending, the parameters of blend maps can be adjusted to provide similar results. In the second part of figure 5, a combination of gray pebbles, brown rock and grass is displayed. Blend maps allow the grass to "grow" into the cracks between the rocks. The gray pebbles protrude out of the surrounding terrain, and usually an entire pebble is visible. However, the material between the rocks is seldom evident. Due to the high importance of the rock, the entire rock is usually visible, providing greater coherence of features.

Two textures can also be quite effective. In figure 6 green and yellow grass has been blended, and the blades of grass remain evident in the mixture. The yellow grass becomes more prominent in the distance. Additionally, blend maps were used to control the specular component in these images as well. For these images, a mud texture with high specular coefficients was added. The reflective properties of the mud are seen in the centre of the image where the mud is visible between grass blades. The second image demonstrates how effective the algorithm is when two textures are selected from several. For every pixel, only two textures are selected for blending. Once again, the specular reflection is also affected by blend maps.

In figure 8, we compare linear blending and blend maps (using two textures per sample). Linear blending is clearly superior to using a single texture. Even though only two textures are used, blend maps still provide useful transitions between different regions.

Figure 4. Textures for blend maps with four textures (importance values below).

10. CONCLUSIONS AND FUTURE WORK

An algorithm for enhancing features in realtime, tiled, synthetic texture maps was introduced. This algorithm has proven to provide pleasing transitions between textures on synthetic terrain, in realtime on modern graphics hardware. The expense of implementing this technique is minimal. Several avenues can still be explored to improve blend maps.

Noise can be used to perturb weights with linear blending [Bloom 2000]. Applying noise helps to break the transition lines between textures and provide a greater variety of transitions. However, no attempt is made to keep larger features of the texture (such as a pebble) visible. The effect of noise functions on blend maps should be investigated to see if the results are superior, or decrease quality. Three dimensional index textures based on gradient, altitude and sun direction would provide an even richer terrain. The effect of blend maps on other surface attributes (such as specular reflection coefficients, or normal maps) could potentially provide interesting results. Lastly, filtering of the index map should be investigated. It may be useful to interpolate some values while not interpolating others.

ACKNOWLEDGMENT

Paul Bourke kindly granted use of the textures in this paper.

REFERENCES

Baudisch, P. and Gutwin, C. 2004. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press, New York, NY, USA, 367–374.

Bloom, C. 2000. Texture splatting. http://www.cbloom.com/3d/techdocs/splatting.txt.

Döllner, J., Baumman, K., and Hinrichs, K. 2000. Texturing techniques for terrain visualization. In *VIS '00: Proceedings of the conference on Visualization '00*. IEEE Computer Society Press, Los Alamitos, CA, USA, 227–234.

Guo, C., Zhu, S. C., and Wu, Y. 2001. Visual learning by integrating descriptive and generative methods. In *International Conference on Computer Vision*. 370–377.

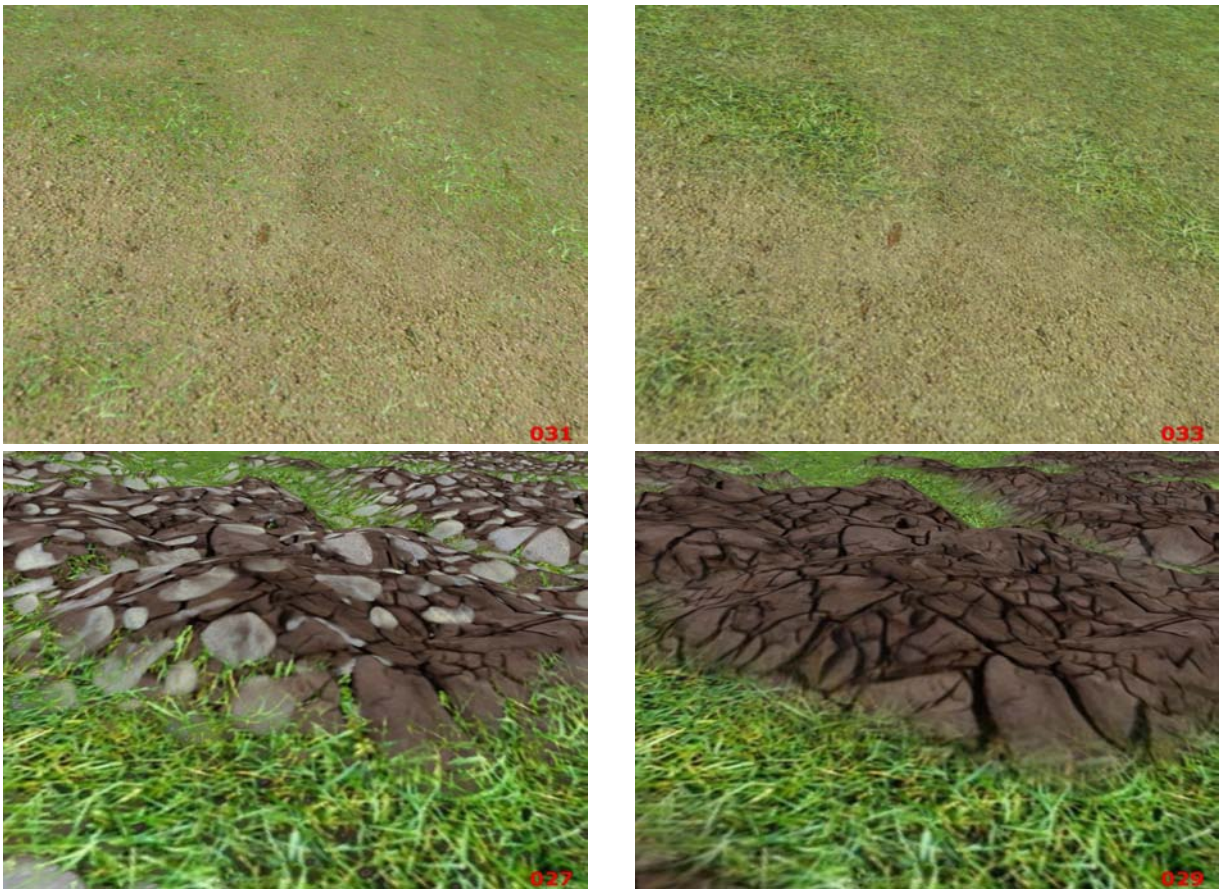
Lai, Y.-Y., Tai, W.-K., Chang, C.-C., and Liu, C.-D. 2005. Synthesizing transition textures on succession patterns. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. ACM Press, New York, NY, USA, 273–276.

Lefebvre, S. and Hoppe, H. 2005. Parallel controllable texture synthesis. *ACM Trans. Graph. 24,* 3, 777–786.

Lefebvre, S. and Neyret, F. 2003. Pattern based procedural textures. In *SI3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM Press, New York, NY, USA, 203–212.

Luna, F. D. 2003. *Introduction to 3D Game Programming with DirectX 9.0*. Wordware Publishing, Inc.

Malik, J., Belongie, S., Shi, J., and Leung, T. 1999. Textons, contours and regions: Cue integration in image segmentation. In *Seventh International Conference on Computer Vision (ICCV'99)*. Vol. 2. 918–925.



Blend maps Linear blending

Figure 5. Comparison between linear blending and blend maps with four textures.

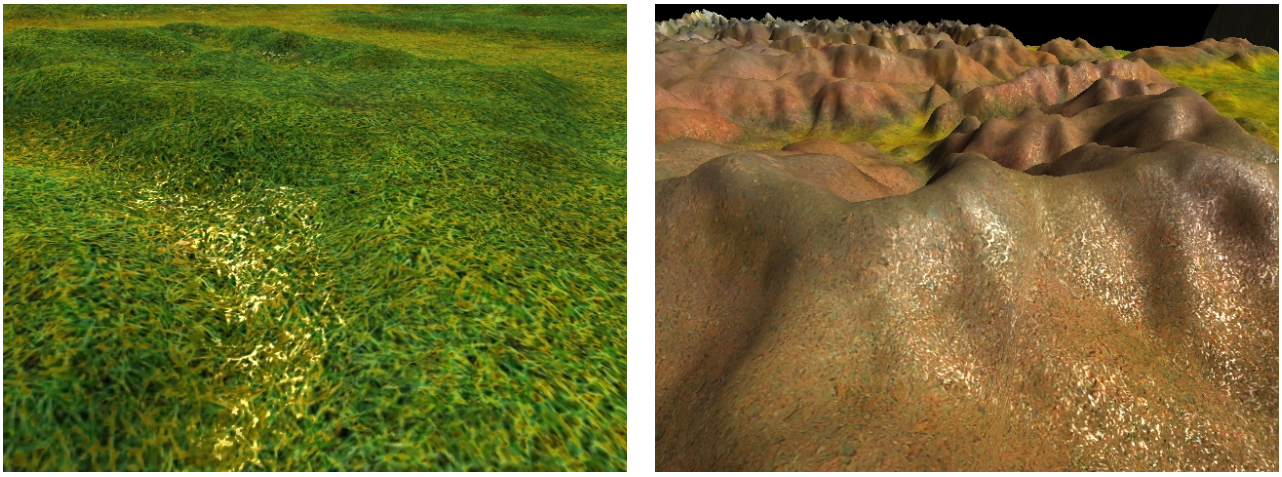


Figure 6. Blend maps using two textures per sample.

Neyret, F. and Cani, M.-P. 1999. Pattern-based texturing revisited. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 235–242.

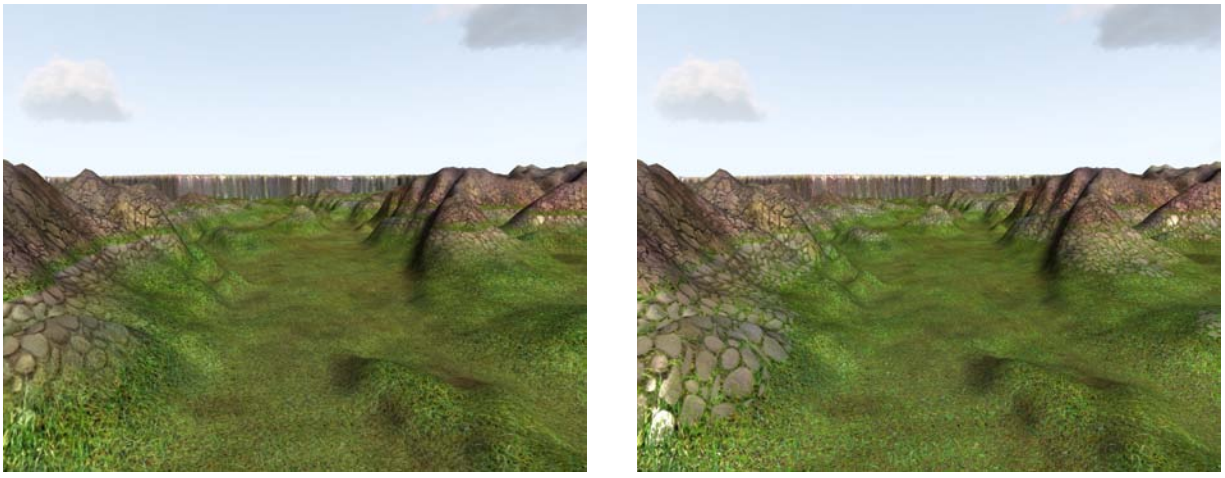
Oliveira, M. M., Bishop, G., and McAllister, D. 2000. Relief texture mapping. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 359–368.

Schaufler, G. 1997. Nailboards: A rendering primitive for image caching in dynamic scenes. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97*. Springer-Verlag, London, UK, 151–162.

Shenas, H. H. and Interrante, V. 2005. Compositing color with texture for multi-variate visualization. In *GRAPHITE '05:*

Linear blending | Real grass | Blend maps

Figure 7. Comparison between linear blending, real grass and blend maps.



Linear blending | Blend maps

Figure 8. Comparison between linear blending and blend maps with two textures per sample.

Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia. ACM Press, New York, NY, USA, 443–446.

Tanner, C. C., Migdal, C. J., and Jones, M. T. 1998. The clipmap: a virtual mipmap. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM Press, New York, NY, USA, 151–158.

Wloka, M. 2005. Improved batching via texture atlases. In *ShaderX3 Advanced Rendering with DirectX and OpenGL*. Charles River Media.

Wu, Q. and Yu, Y. 2004. Feature matching and deformation for texture synthesis. *ACM Trans. Graph. 23,* 3, 364–367.

Zhang, J., Zhou, K., Velho, L., Guo, B., and Shum, H.-Y. 2003. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph. 22,* 3, 295–302.