Reconstructing a Height Field from a Normal Map

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

Created: May 3, 2006
Last Modified: March 1, 2008

Contents

1 Introduction

A *height field* is a function $z = H(x, y)$ for real-valued continuous variables x, y, and z. Given a point (x, y) in the plane, $H(x, y)$ is the height of the graph of the function above that point ($z \geq 0$) or below that point ($z < 0$). For our purposes, the domain of the function is a rectangle, $x_{\min} \leq x \leq x_{\max}$ and $y_{\min} \leq y \leq y_{\max}$.

The graph of $H(x, y)$ is a surface defined implicitly by $0 = F(x, y, z) = z - H(x, y)$. From calculus, we know that the gradient of $F(x, y, z)$ is a normal vector to the surface at (x, y, z). The gradient is

$$\nabla F = (F_x, F_y, F_z) = (-H_x, -H_y, 1)$$

where F_x, F_y, F_z, H_x, and H_y are the first-order partial derivatives of the functions. Unit-length normals for the height field are

$$\mathbf{N}(x, y) = (N_0(x, y), N_1(x, y), N_2(x, y)) = \frac{(-H_x, -H_y, 1)}{\sqrt{1 + H_x^2 + H_y^2}} \tag{1}$$

where the functional notation on the left-hand side indicates that $\mathbf{N}(x, y)$ is a normal vector to the graph at the point $(x, y, H(x, y))$.

The normal vector components are in the interval $[-1, 1]$. For the purposes of bump mapping, the components are *range-compressed* to the interval $[0, 1]$ to be stored as the color channels in a texture image to be used by a fragment program. The range-compressed vector is

$$\mathbf{C}(x, y) = \left(\frac{N_0(x, y) + 1}{2}, \frac{N_1(x, y) + 1}{2}, \frac{N_2(x, y) + 1}{2} \right) \tag{2}$$

The texture image is referred to as a *normal map* for the height field. In a vertex program, the vertices are assigned (x, y) values as texture coordinates for the normal map. These are passed through to the rasterizer for interpolation on a per-pixel basis. The (x, y) texture coordinate for a single pixel is passed to the fragment program, the normal map texture is sampled to produce $\mathbf{C}(x, y)$, and the value is uncompressed to a normal vector $\mathbf{N}(x, y)$. This vector is considered to be the surface normal for the graph and is used in dot product computations to modulate the lighting at the surface point.

2 Constructing a Normal Map

The previous section introduces the concept of height fields and normal maps using *continuous variables* x, y, and z. In practice, the height fields are represented using discrete variables. That is, a height field is is defined on a rectangular lattice with r rows and c columns. Each lattice point (i, j), where $0 \leq i < c$ and $0 \leq j < r$, has an associated height h_{ij}. These values are stored in a two-dimensional gray-scale image. In terms of the problem statement using continuous variables, you may think of the discrete version as a sampling of the continuous height field. Define

$$x_i = x_{\min} + \frac{(x_{\max} - x_{\min})i}{c - 1}, \quad y_j = y_{\min} + \frac{(y_{\max} - y_{\min})j}{r - 1}, \quad h_{i,j} = H(x_i, y_j)$$

for $0 \leq i < c$ and $0 \leq j < r$.

In the continuous setting, the normal vector construction requires computing partial derivatives of $H(x, y)$. We have only samples in the discrete setting, so we may rely on finite-difference approximations to these partial derivatives. There are many ways to do this of which two are covered here.

2.1 One-Sided Difference Approximations

From calculus, the partial derivatives are defined as limits,

$$H_x(x,y) = \lim_{\Delta x \to 0} \frac{H(x+\Delta x, y) - H(x,y)}{\Delta x}, \quad H_y(x,y) = \lim_{\Delta y \to 0} \frac{H(x, y+\Delta y) - H(x,y)}{\Delta y}$$

For sufficiently small Δx and Δy, reasonable approximations to the derivatives are the *one-sided differences*

$$H_x(x,y) \doteq \frac{H(x+\Delta x, y) - H(x,y)}{\Delta x}, \quad H_y(x,y) \doteq \frac{H(x, y+\Delta y) - H(x,y)}{\Delta y}$$

In the discrete setting, the sample spacing defines the smallest possible Δx and Δy values; namely,

$$\Delta x = \frac{x_{\max} - x_{\min}}{c-1}, \quad \Delta y = \frac{y_{\max} - y_{\min}}{r-1}$$

If $x_{\min}$, $x_{\max}$, $y_{\min}$, and $y_{\max}$ are unknown, it is reasonable to choose $x_{\min} = 0$, $x_{\max} = c-1$, $y_{\min} = 0$, and $y_{\max} = r-1$, in which case $\Delta x = 1$ and $\Delta y = 1$. However, in my own software I allow the user to specify Δx and Δy. The one-sided difference approximations are

$$\begin{aligned} H_x(x_i, y_j) &\doteq \tfrac{H(x_{i+1},y_j) - H(x_i,y_j)}{\Delta x} = \tfrac{h_{i+1,j} - h_{i,j}}{\Delta x} \\ H_y(x_i, y_j) &\doteq \tfrac{H(x_i,y_{j+1}) - H(x_i,y_j)}{\Delta y} = \tfrac{h_{i,j+1} - h_{i,j}}{\Delta y} \end{aligned} \tag{3}$$

The normal vectors are computed by substituting the approximations of Equation (3) into Equations (1) and (2), obtaining a two-dimensional array of normal vectors

$$\mathbf{N}_{i,j} = \mathbf{N}(x_i, y_j) \doteq \frac{(-\sigma_x(h_{i+1,j} - h_{i,j}), -\sigma_y(h_{i,j+1} - h_{i,j}), 1)}{\sqrt{1 + [\sigma_x(h_{i+1,j} - h_{i,j})]^2 + [\sigma_y(h_{i,j+1} - h_{i,j})]^2}} \tag{4}$$

where $\sigma_x = 1/\Delta x$ and $\sigma_y = 1/\Delta y$. The corresponding range-compressed vectors are $\mathbf{C}_{i,j} = \mathbf{C}(x_i, y_j)$, which are stored as an RGB image.

2.2 Centered Difference Approximations

The one-sided difference are biased in the sense that they favor derivative measurements with increasing x and y. An unbiased alternative is the following. From calculus, the partial derivatives are defined as limits,

$$H_x(x,y) = \lim_{\Delta x \to 0} \frac{H(x+\Delta x, y) - H(x-\Delta x,y)}{2\Delta x}, \quad H_y(x,y) = \lim_{\Delta y \to 0} \frac{H(x, y+\Delta y) - H(x,y-\Delta y)}{2\Delta y}$$

For sufficiently small Δx and Δy, reasonable approximations to the derivatives are the *centered differences*

$$H_x(x,y) \doteq \frac{H(x+\Delta x, y) - H(x-\Delta x,y)}{2\Delta x}, \quad H_y(x,y) \doteq \frac{H(x, y+\Delta y) - H(x,y-\Delta y)}{2\Delta y}$$

The centered difference approximations are

$$\begin{aligned} H_x(x_i, y_j) &\doteq \tfrac{H(x_{i+1},y_j) - H(x_{i-1},y_j)}{2\Delta x} = \tfrac{h_{i+1,j} - h_{i-1,j}}{2\Delta x} \\ H_y(x_i, y_j) &\doteq \tfrac{H(x_i,y_{j+1}) - H(x_i,y_{j-1})}{2\Delta y} = \tfrac{h_{i,j+1} - h_{i,j-1}}{2\Delta y} \end{aligned} \tag{5}$$

The normal vectors are computed by substituting the approximations of Equation (5) into Equations (1) and (2), obtaining a two-dimensional array of normal vectors

$$\mathbf{N}_{i,j} = \mathbf{N}(x_i, y_j) \doteq \frac{(-\sigma_x(h_{i+1,j} - h_{i-1,j}), -\sigma_y(h_{i,j+1} - h_{i,j-1}), 1)}{\sqrt{1 + [\sigma_x(h_{i+1,j} - h_{i-1,j})]^2 + [\sigma_y(h_{i,j+1} - h_{i,j-1})]^2}} \tag{6}$$

where $\sigma_x = 1/(2\Delta x)$ and $\sigma_y = 1/(2\Delta y)$. The corresponding range-compressed vectors are $\mathbf{C}_{i,j} = \mathbf{C}(x_i, y_j)$, which are stored as an RGB image.

2.3 Left-Handed Height Image

The previous construction assumes the height field lives in a right-handed coordinate system for (x, y, z). When the height field is computed from a texture image (the *height image*), and if the texture image is based on left-handed coordinates for (x, y) (x increases rightward, y increases downward), a conversion needs to be made when computing normal vectors to obtain a right-handed system. This can be accomplished by negating the height field itself, $z = -H(x, y)$, in which case $0 = F(x, y, z) = z + H(x, y)$. This leads to sign changes in the first two components of the normal vectors in Equations (4) and (6). In the implementation, it is sufficient to use $+\sigma_x$ and $+\sigma_y$ in the formulas rather than $-\sigma_x$ and $-\sigma_y$.

2.4 Boundary Conditions

Notice that the one-sided difference approximations involve indices $(i + 1, j)$ and $(i, j + 1)$. The differences at $i = c - 1$ are $h_{c,j} - h_{c-1,j}$ and the differences at $j = r - 1$ are $h_{i,r} - h_{i,r-1}$. Both expressions involve out-of-range indices. You need a convention for choosing *boundary conditions* for the height field. One possibility is to force a duplication of the derivative estimates from the adjacent locations, say

$$\begin{aligned} h_{c,j} &= 2h_{c-1,j} - h_{c-2,j} \\ h_{i,r} &= 2h_{i,r-1} - h_{i,r-2} \end{aligned} \tag{7}$$

Another possibility is to use wrap-around,

$$\begin{aligned} h_{c,j} &= h_{0,j} \\ h_{i,r} &= h_{i,0} \end{aligned} \tag{8}$$

The centered differences involve indices $(i \pm 1, j)$ and $(i, j \pm 1)$. The indices are out of range when $i = 0$, $i = c - 1$, $j = 0$, and $j = r - 1$. You may force a duplication of the derivative estimates from the adjacent locations,

$$\begin{aligned} h_{-1,j} &= h_{0,j} + h_{1,j} - h_{2,j} \\ h_{c,j} &= h_{c-1,j} + h_{c-2,j} - h_{c-3,j} \\ h_{i,-1} &= h_{i,0} + h_{i,1} - h_{i,2} \\ h_{i,r} &= h_{i,r-1} + h_{i,r-2} - h_{i,r-3} \end{aligned} \tag{9}$$

or you can use one-sided differences at the boundary, both forward and backward,

$$\begin{aligned} h_{c,j} &= 2h_{c-1,j} - h_{c-2,j} \\ h_{-1,j} &= 2h_{0,j} - h_{1,j} \\ h_{i,r} &= 2h_{i,r-1} - h_{i,r-2} \\ h_{i,-1} &= 2h_{i,0} - h_{i,1} \end{aligned} \tag{10}$$

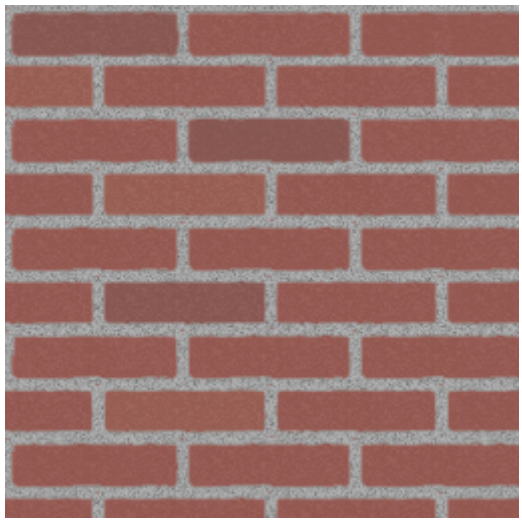
or you can use wrap-around,

$$\begin{aligned} h_{-1,j} &= h_{c-1,j} \\ h_{c,j} &= h_{0,j} \\ h_{i,-1} &= h_{i,r-1} \\ h_{i,r} &= h_{i,0} \end{aligned} \tag{11}$$

2.5 An Example

The classic example to illustrate normal maps involves rendering a rectangle that has a brick texture applied to it. Figure 1 shows such a texture, a tweaked subimage of the `Brkrun.JPG` file that ships with the maps in 3dsmax 8.

Figure 1. An RGB brick texture.

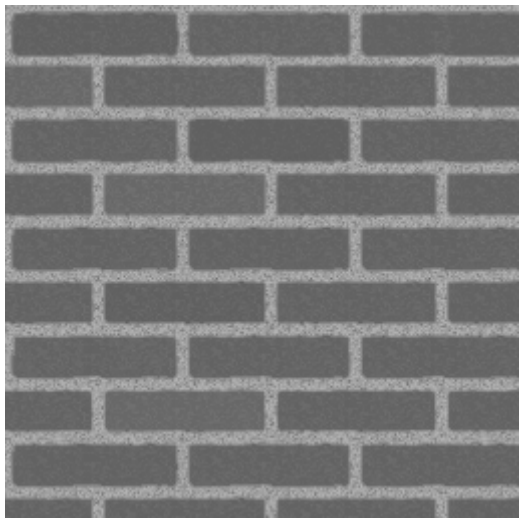


The first problem you have to deal with is creating a height field that corresponds to the brick texture. There are a number of ways to convert an RGB image to a gray-scale image. I chose this simple scheme to generate an intensity,

$$I = (0.2125R + 0.7154G + 0.0721B)/\mu \in [0, 1]$$

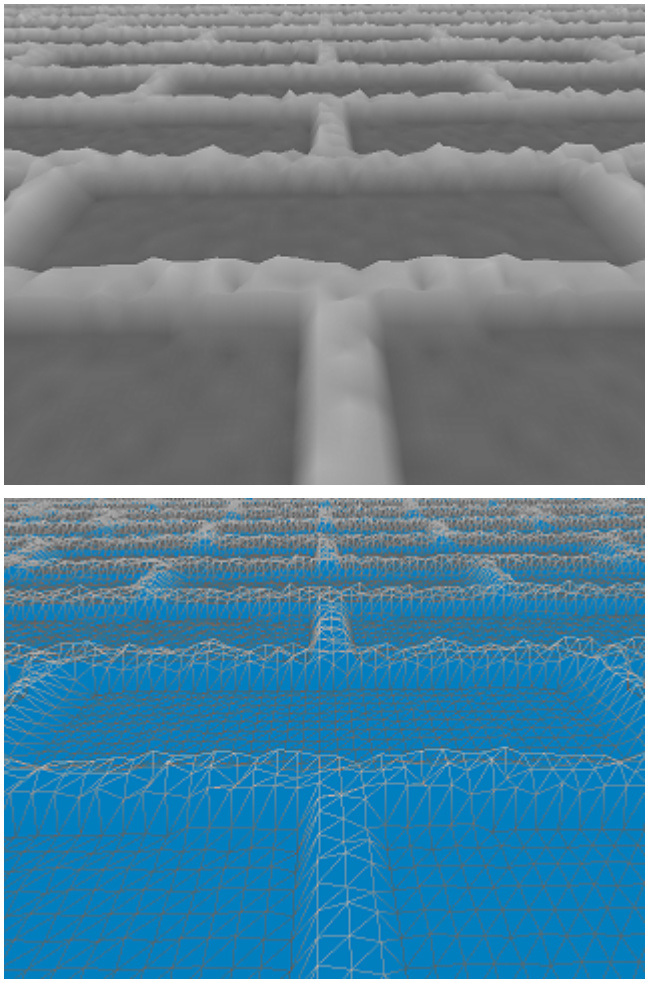
from the RGB channels, where each color channel has values in $[0, \mu]$. For 8-bit images, the maximum channel value is $\mu = 255$. The resulting image is shown in Figure 2.

Figure 2. A gray-scale version of the brick texture of Figure 1.



The height field can be visualized by actually mapping the gray-scale image values as heights over a rectangular region in the plane. Figure 3 shows such a rendering of a small portion of the brick height field.

Figure 3. A rendering of the height field, both as a solid mesh and in wireframe mode.



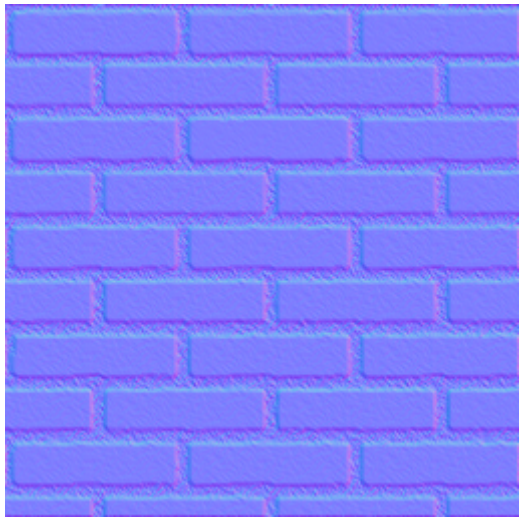
The gray-scale image of Figure 2 has its heights in the interval $[0, 1]$. The image was processed by the following code. I used centered differences from Equation (6), wrap-around boundary conditions from Equation (11), and sign changes on the first two normal components to map from a left-handed image to a right-handed normal vector field. I also choose input scales of $\sigma_x = \sigma_y = 4$.

```
unsigned char* CreateNormalMap::GenerateNormals (int iXMax, int iYMax,
    float* afHeight, float fXScale, float fYScale)
{
    // The values afHeight[i] are required to be in [0,1].

    int iXMaxM1 = iXMax -1, iYMaxM1 = iYMax - 1;
    unsigned char* aucNormal = WM4_NEW unsigned char[3*iXMax*iYMax];

    for (int iY = 0; iY < iYMax; iY++)
    {
        // Use wrap-around.
        int iYm1 = (iY > 0 ? iY-1 : iYMaxM1);
```

```
        int iYp1 = (iY < iYMaxM1 ? iY+1 : 0);

        for (int iX = 0; iX < iXMax; iX++)
        {
            // Use wrap-around.
            int iXm1 = (iX > 0 ? iX-1 : iXMaxM1);
            int iXp1 = (iX < iXMaxM1 ? iX+1 : 0);

            // The approximation dH/dx is in [-xscale,xscale] and the
            // approximation dH/dy is in [-yscale,yscale] since H is in
            // [0,1].
            float fHXm1Y = afHeight[iXm1 + iY*iXMax];
            float fHXp1Y = afHeight[iXp1 + iY*iXMax];
            float fHXYm1 = afHeight[iX + iYm1*iXMax];
            float fHXYp1 = afHeight[iX + iYp1*iXMax];
            float fDHDX = fXScale*(fHXp1Y - fHXm1Y);
            float fDHDY = fYScale*(fHXYp1 - fHXYm1);

            // Left-handed (x,y,z) coordinates are used, so no minus sign
            // occurs in the first two components of the normal vector.
            Vector3f kNormal(fDHDX,fDHDY,1.0f);
            kNormal.Normalize();

            // Transform the normal vector from [-1,1]^3 to [0,255]^3 so it
            // can be stored as a color value.
            unsigned char* aucCompressed = &aucNormal[3*(iX+iY*iXMax)];
            aucCompressed[0] = (unsigned char)(127.5f*(kNormal[0]+1.0f));
            aucCompressed[1] = (unsigned char)(127.5f*(kNormal[1]+1.0f));
            aucCompressed[2] = (unsigned char)(127.5f*(kNormal[2]+1.0f));
        }
    }

    return aucNormal;
}
```

Figure 4 shows the resulting color image that is stored in the array aucCompressed.

Figure 4. The normal map corresponding to the gray-scale height image of Figure 2.



The bumpmapping was applied to a square with a tiling factor of 2 in each dimension. Figure 5 shows a rendering of the square with only the base texture.

Figure 5. A square with the brick texture as the base texture but no bumpmapping is applied.

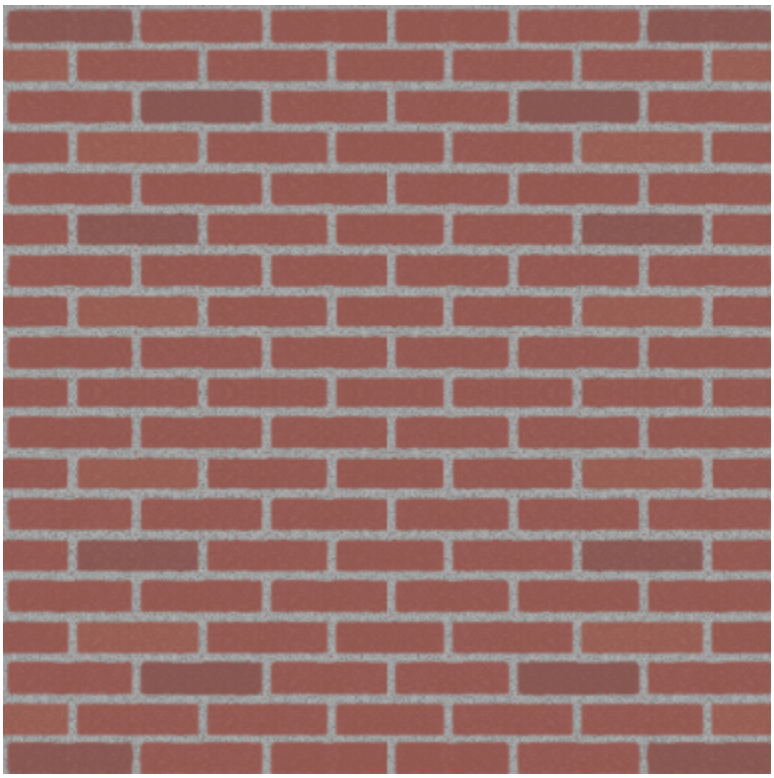
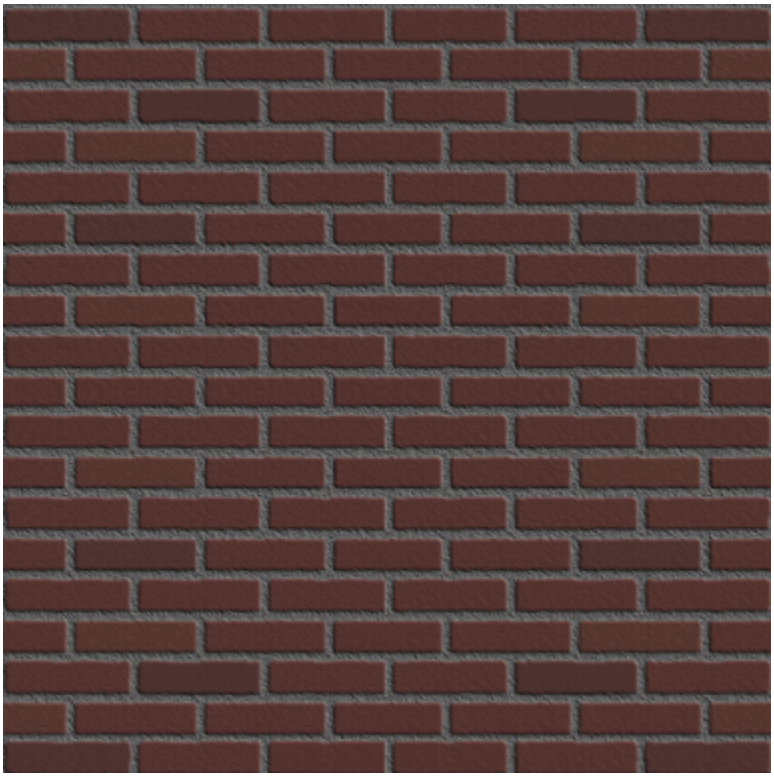


Figure 6 shows the rendered square with the brick texture and normal mapping used to modulate the (directional) lighting.

Figure 6. A square with the brick texture and bumpmapping applied.



3 Reconstructing a Height Field

The process of generating a normal map from a height field involved choosing:

1. A transformation from RGB values to intensities.
2. A finite-difference scheme for approximating derivatives, including what to use for Δx and Δy.
3. A scheme for boundary conditions.
4. Whether or not to switch from left-handed image coordinates to right-handed normal vector fields.

It is not possible to invert the intensities to RGB values. The best you can do is reconstruct the gray-scale height image. If you are given a normal map and all this information is known to you, it is not possible to reconstruct *exactly* the height field. The first problem is that if $z = H(x, y)$ is a height field that generated the normal map, the height fields $z = H(x, y) + c$ for any constant c also generate the same normal map. Thus, the best you can do is reconstruct the height field modulo a translation. Along with the normal map, if you were to be told the height $h_{0,0}$, then you can remove the translational degree of freedom.

Whether exact reconstruction is possible also depends on the difference scheme and boundary conditions. If you do not have all this information, it is possible to reconstruct a height field that is similar to the original, but not exactly the original. The following discussion illustrates the procedure for reconstruction. In all of this, I will assume that the conversion was made from left-handed image coordinates to right-handed normal vector fields.

3.1 One-Sided Differences, Wrap-Around Boundary

The normal map image has c columns and r rows. I assume here that c and r are power-of-two since texture images tend to have that restriction. The normal map values are the compressed quantities

$$\mathbf{C}_{i,j} = (C_{i,j;0}, C_{i,j;1}, C_{i,j;2})$$

for $0 \le i < c$ and $0 \le j < r$. The corresponding normal vectors are

$$\mathbf{N}_{i,j} = (N_{i,j;0}, N_{i,j;1}, N_{i,j;2}) = (2C_{i,j;0} - 1, 2C_{i,j;1} - 1, 2C_{i,j;2} - 1)$$

Solving Equation (4), but with the signs changed on the first two components to indicate the conversion from left-handed image coordinates to a right-handed normal vector field, we have

$$\begin{aligned} h_{i+1,j} - h_{i,j} &= \tfrac{1}{\sigma_x}\,\tfrac{N_{i,j;0}}{N_{i,j;2}} = A_{i,j} \\ h_{i,j+1} - h_{i,j} &= \tfrac{1}{\sigma_y}\,\tfrac{N_{i,j;1}}{N_{i,j;2}} = B_{i,j} \\ h_{c,j} &= h_{0,j}, \quad h_{i,r} = h_{i,0} \end{aligned}$$

where $A_{i,j}$ and $B_{i,j}$ are defined by the last equalities in the first two equations. These are both known quantities since we have been supplied with the normal vectors $\mathbf{N}_{i,j}$.

As indicated earlier, $h_{0,0}$ is a degree of freedom. Its value and the difference equations may be used to compute the first row of the height image. Specifically,

$$h_{k+1,0} - h_{0,0} = \sum_{i=0}^{k} (h_{i+1,0} - h_{i,0}) = \sum_{i=0}^{k} A_{i,0}$$

The summation of the height differences is a *telescoping sum*. All but two of the terms cancel out. Thus,

$$h_{k+1,0} = h_{0,0} + \sum_{i=0}^{k} A_{i,0}, \quad 0 \le k \le c-2 \tag{12}$$

Using the wrap-around boundary condition, we also have a *consistency condition* when $k = c - 1$. By the wrap-around, $h_{c,0} = h_{0,0}$, so the evaluation at $k = c - 1$ produces

$$\sum_{i=0}^{c-1} A_{i,0} = 0 \tag{13}$$

Knowing the normal map was constructed as it was, the consistency condition must be true (in theory, but modulo numerical round-off errors in practice).

We can reconstruct the second row in a similar manner, but we need to know $h_{0,1}$ to start the process. This is obtained by the finite-difference equation in the y direction,

$$h_{0,1} = h_{0,0} + B_{0,0} \tag{14}$$

The remaining terms in the row are

$$h_{k+1,1} = h_{0,1} + \sum_{i=0}^{k} A_{i,1}, \quad 0 \le k \le c-2 \tag{15}$$

with consistency condition

$$\sum_{i=0}^{c-1} A_{i,1} = 0 \tag{16}$$

The remaining rows are constructed in the same manner. The first term of each row is constructed by a finite difference across two rows, the remaining terms are constructed by finite differences across two columns, and a consistency equation occurs for each row.

3.2 Centered Differences, Wrap-Around Boundary

The normal map image has c columns and r rows. I assume here that c and r are power-of-two since texture images tend to have that restriction. When using centered differences, the parity (even or odd) of c and r affect the reconstruction. The normal map values are the compressed quantities

$$\mathbf{C}_{i,j} = (C_{i,j;0}, C_{i,j;1}, C_{i,j;2})$$

for $0 \le i < c$ and $0 \le j < r$. The corresponding normal vectors are

$$\mathbf{N}_{i,j} = (N_{i,j;0}, N_{i,j;1}, N_{i,j;2}) = (2C_{i,j;0} - 1, 2C_{i,j;1} - 1, 2C_{i,j;2} - 1)$$

Solving Equation (6), but with the signs changed on the first two components to indicate the conversion from left-handed image coordinates to a right-handed normal vector field, we have

$$\begin{aligned}
&h_{i+1,j} - h_{i-1,j} = \tfrac{1}{\sigma_x} \tfrac{N_{i,j;0}}{N_{i,j;2}} = A_{i,j} \\
&h_{i,j+1} - h_{i,j-1} = \tfrac{1}{\sigma_y} \tfrac{N_{i,j;1}}{N_{i,j;2}} = B_{i,j} \\
&h_{c,j} = h_{0,j}, \quad h_{i,r} = h_{i,0} \\
&h_{-1,j} = h_{c-1,j}, \quad h_{c,j} = h_{0,j}, \quad h_{i,-1} = h_{i,r-1}, \quad h_{i,r} = h_{i,0}
\end{aligned}$$

where $A_{i,j}$ and $B_{i,j}$ are defined by the last equalities in the first two equations. These are both known quantities since we have been supplied with the normal vectors $\mathbf{N}_{i,j}$.

As indicated earlier, $h_{0,0}$ is a degree of freedom. Under the assumption that c is even, we actually have a second degree of freedom when attempting to compute the first row of the height image. Notice that

$$h_{i+1,0} - h_{i-1,0} = A_{i,0}$$

involves height samples whose column indices, $i+1$ and $i-1$, always differ by two. Given $h_{0,0}$, this restricts you to reconstructing only the height samples for even i. Using the idea of the telescoping sum as shown for one-sided differences,

$$h_{2k+2,0} = h_{0,0} + \sum_{i=0}^{k} A_{2i+1,0}, \quad 0 \le k \le c/2 - 2 \tag{17}$$

with consistency condition at $k = c/2 - 1$,

$$\sum_{i=0}^{c/2-1} A_{2i+1,0} = 0 \tag{18}$$

A similar *cycle* of height samples occurs for odd i, but to start the reconstruction we need to know one of the values, say $h_{1,0}$. Then

$$h_{2k+3,0} = h_{1,0} + \sum_{i=0}^{k} A_{2i,0}, \quad 0 \le k \le c/2 - 2 \tag{19}$$

with consistency condition at $k = c/2 - 1$,

$$\sum_{i=0}^{c/2-1} A_{2i,0} = 0 \tag{20}$$

As long as we choose values for $h_{0,0}$ and $h_{1,0}$, we can reconstruct the first row of the height image when centered differences are used. Now let's try to reconstruct the second row. The same issue arises. The centered difference equations lead to two disjoint systems of linear equations, each requiring a known starting value. You may as well choose values for $h_{0,1}$ and $h_{1,1}$, neither of which is derivable from the centered differences in the y direction (making this unlike the one-sided difference case). The even-numbered terms are constructed by

$$h_{2k+2,1} = h_{0,1} + \sum_{i=0}^{k} A_{2i+1,1}, \quad 0 \le k \le c/2 - 2 \tag{21}$$

with consistency condition at $k = c/2 - 1$,

$$\sum_{i=0}^{c/2-1} A_{2i+1,1} = 0 \tag{22}$$

and the odd-numbered terms are constructed by

$$h_{2k+3,1} = h_{1,1} + \sum_{i=0}^{k} A_{2i,1}, \quad 0 \le k \le c/2 - 2 \tag{23}$$

with consistency condition at $k = c/2 - 1$,

$$\sum_{i=0}^{c/2-1} A_{2i,1} = 0 \tag{24}$$

Now when you attempt to reconstruct row $j = 2$, the centered differences in the y direction do allow you to compute the starting quantities

$$h_{0,2} = h_{0,0} + B_{0,1}, \quad h_{1,2} = h_{1,0} + B_{1,1}$$

The remaining terms of the row are constructed as before.

In summary, the centered difference reconstruction, using wrap-around and applied to an image with an even number of rows and an even number of columns, has four degrees of freedom: $h_{0,0}$, $h_{1,0}$, $h_{0,1}$, and $h_{1,1}$. It is unlikely you will know these values, so a reasonable substitute is to treat the normal vectors at these locations as if they were computed using one-sided differences, and then

$$\begin{aligned}
h_{0,0} &= 0 \text{ or some other arbitrary choice} \\
h_{1,0} &= h_{0,0} + A_{0,0} \\
h_{0,1} &= h_{0,0} + B_{0,0} \\
h_{1,1} &= ((h_{0,1} + A_{0,1}) + (h_{1,0} + B_{1,0}))/2 = h_{00} + (A_{0,0} + A_{0,1} + B_{0,0} + B_{1,0})/2
\end{aligned}$$

The expression for $h_{1,1}$ was chosen to be the average of what the one-sided differences produce (one in the row direction, one in the column direction) to avoid bias that might occur using only one direction.