Normal Map to Height Map

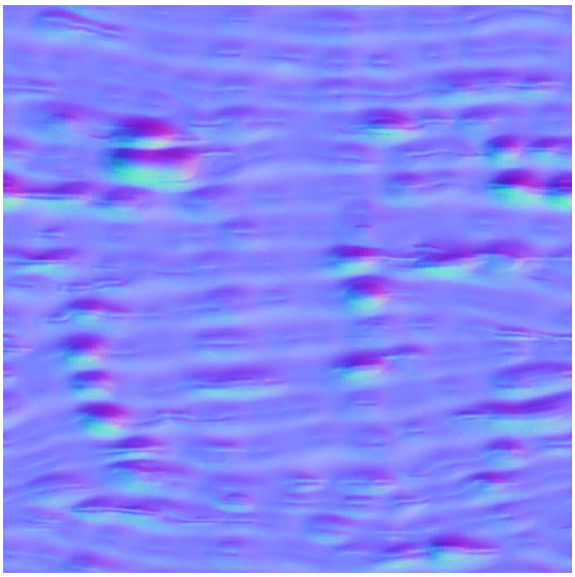
Laurent Cancé 05 avril 2017
(ajout 04 octobre 2017)

I. Introduction.

From a height map is easy to convert to a normal map, but the reverse is something using a straight of clue.
Even if it's imperfect this algorithm works for the most normal maps.

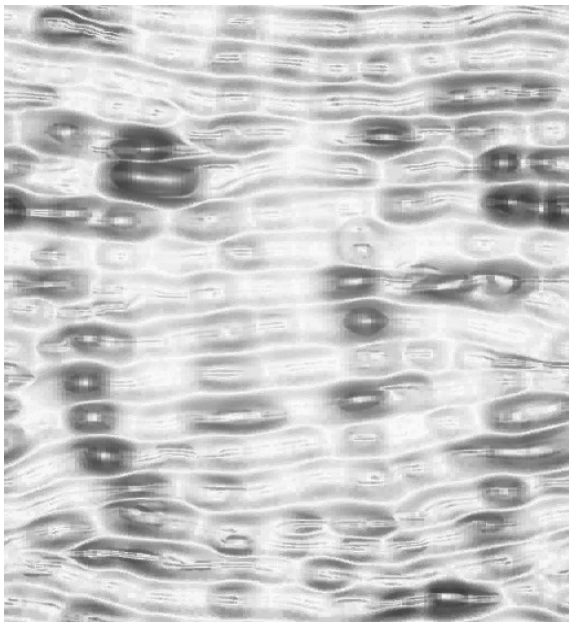
II. Seeking.

How to retrieve height map from this wonderfull Skyrim texture ?



A.

This can be the solution:



B.

III. How it works ?

First the texture space must be divided in cell of fixed scale. Then we apply a simple parametrizable envmap to calculated normals :

The envmap :

```
phongmap=(unsigned char *)malloc(tilephongmap*tilephongmap*4);
for (y=0;y<tilephongmap*2;y++)
	for (x=0;x<tilephongmap*2;x++)
	{
		xx=((float)(x-tilephongmap))/tilephongmap;
		yy=((float)(y-tilephongmap))/tilephongmap;
		r=sqrtf(xx*xx + yy*yy);
		if (r>1) r=0; else r=1-r;
		phongmap[x+y*tilephongmap*2]=255*r*r;
	}
```

The Normals :

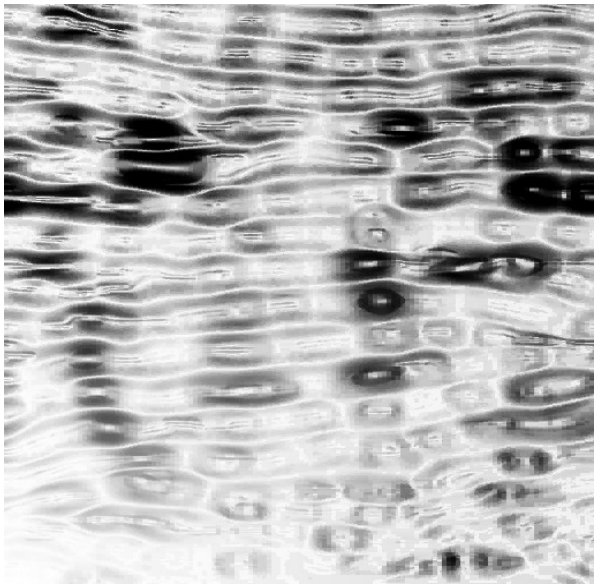
```
for(y=0;y<h;y++)
	for(x=0;x<w;x++)
	{
		int r=ptr[4*(x+w*y)+0];
		int g=ptr[4*(x+w*y)+1];
		int b=ptr[4*(x+w*y)+2];

		nx=(((float)r / 255.0f) - 0.5) * 2.0f;
		ny=(((float)g / 255.0f) - 0.5) * 2.0f;
		nz=(((float)b / 255.0f) - 0.5) * 2.0f;

		float nn=sqrtf(nx*nx+ny*ny+nz*nz);
		nx/=nn;
		ny/=nn;
		nz/=nn;

		normals[3*(x+w*y)+0]=nx;
		normals[3*(x+w*y)+1]=ny;
		normals[3*(x+w*y)+2]=nz;
	}
```

Then parsing all cells with the normal deformations centered on the cell, sum up and normalize !

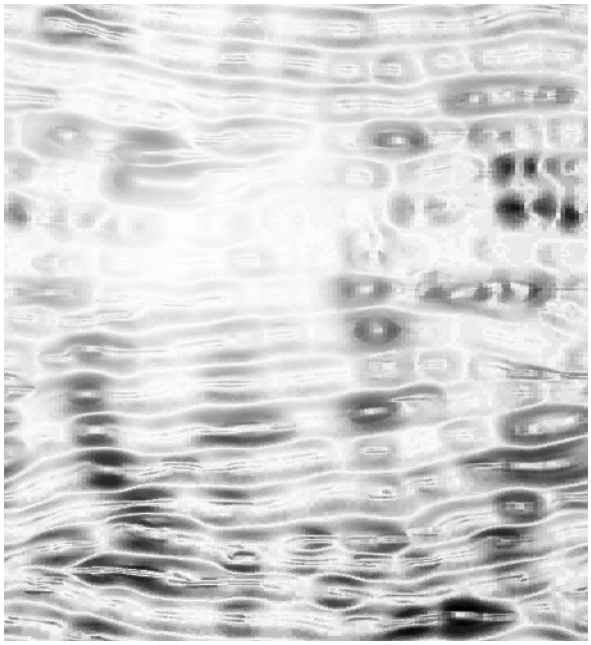
```
for(y=0;y<h;y++)
	for(x=0;x<w;x++)
	{
		float dx=normals[(x+w*y)*3+0]*2.0f;
		float dy=normals[(x+w*y)*3+1]*2.0f;

		int xxx=tilephongmap+(x-cx)*dx;
		int yyy=tilephongmap+(y-cy)*dy;

		if (xxx<0) xxx=0;
		if (yyy<0) yyy=0;

		if (xxx>=tilephongmap*2) xxx=tilephongmap*2-1;
		if (yyy>=tilephongmap*2) yyy=tilephongmap*2-1;

		int a=phongmap[xxx+yyy*tilephongmap*2];

		res[x+w*y]=a;
	}
```

Cell samples :

1.



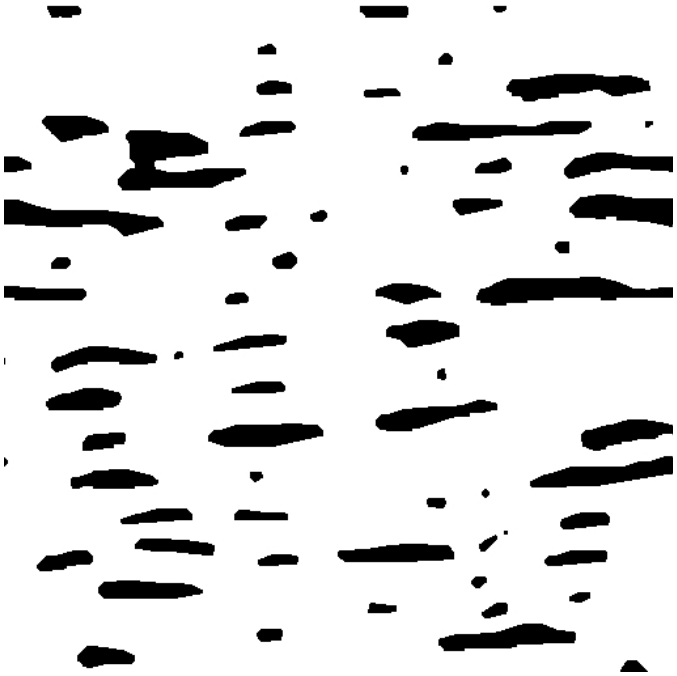
2.



IV. Deeper

If you look at the result (B) and partial cells, you can find that normal map describes some kind of miscellaneous depth forms.
At this point you can proceed in finding this forms by **silhouette** processing that gives the following :

Silhouette :

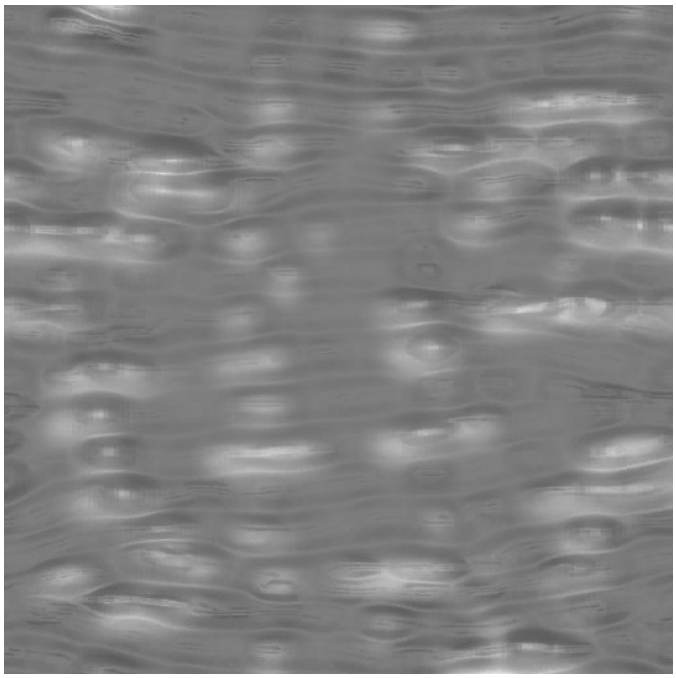


C.

Some kind of processing 0 or 1 from middle arbitrary value, then apply the binary of a convolution at a binary discretisation :

```
for (x=0;x<w;x++)
	for (y=0;y<h;y++)
	{
		int cc=0;
		float nb=0;
		int DEF=8;
		for (int i=-DEF;i<=DEF;i++)
			for (int j=-DEF;j<=DEF;j++)
			{
				int xx=x+i;
				int yy=y+j;
				if (sqrtf(i*i+j*j)<DEF)
				{
					if ((xx>=0)&&(yy>=0)&&(xx<w)&&(yy<h))
					{
						a=ptr[(xx+w*yy)+0]&255;
						cc+=a;
						nb+=1;
					}
				}
			}

		cc/=nb;
		if (cc<128) cc=0; else cc=255;
		ptrImage[(x+w*y)]=cc;
	}
```

Then blur the whole in order to modulate cells integrate with partial form detection and i gives the result of a cool depth map calculated with normal maps :



(Blur C) % (B)