heman Documentation

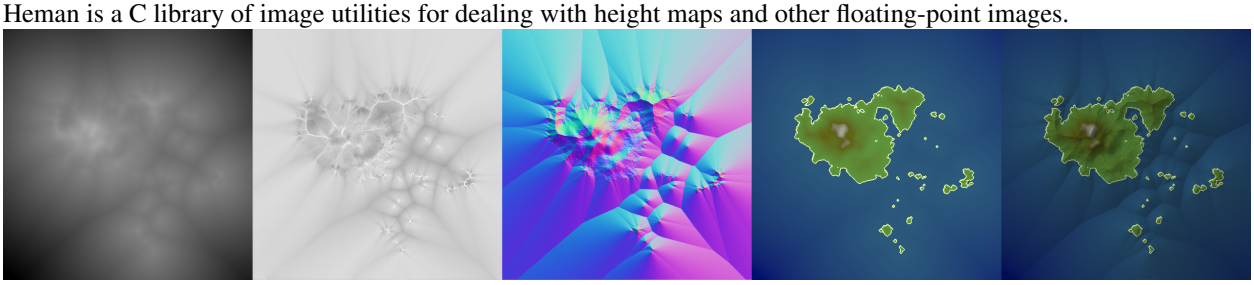
Release r1

Philip Rideout

August 17, 2015

Contents

Heman is a C library of image utilities for dealing with height maps and other floating-point images.



Heman can be used for:

- Creating random height fields using simplex noise and FBM.
- Generating a normal map from a height map using forwarding differencing.
- Efficiently computing ambient occlusion from a height map.
- Generating a signed distance field (SDF) using a fast algorithm.
- Exporting a 3D mesh in PLY format.
- Applying a color gradient to a heightmap (LUT).
- Generating a color gradient, given a list of control points.
- Computing diffuse lighting with an infinite light source.

CHAPTER 1

Why the name “heman”?

It’s a subset of letters taken from *height map* and *normal map*.

CHAPTER 2

Source code

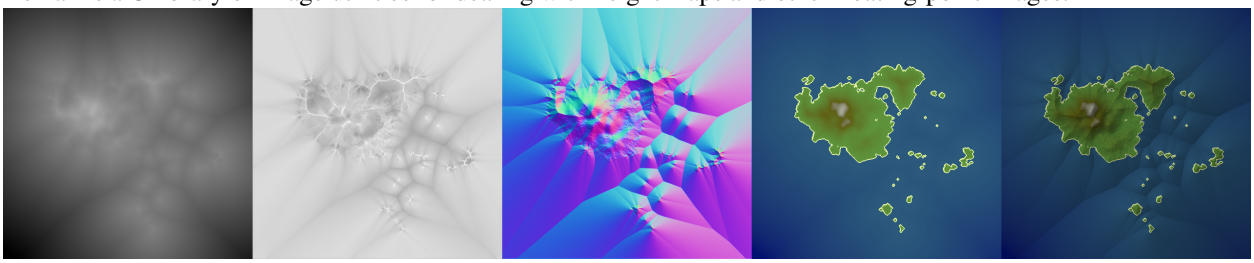
You can access the source code at: https://github.com/prideout/heman

CHAPTER 3

Documentation

3.1 Heman Overview

Heman is a C library of image utilities for dealing with height maps and other floating-point images.



Heman can be used for:

- Creating random height fields using simplex noise and FBM.
- Generating a normal map from a height map using forwarding differencing.
- Efficiently computing ambient occlusion from a height map.
- Generating a signed distance field (SDF) using a fast algorithm.
- Exporting a 3D mesh in PLY format.
- Applying a color gradient to a heightmap (LUT).
- Generating a color gradient, given a list of control points.
- Computing diffuse lighting with an infinite light source.

3.1.1 Why the name "heman"?

It's a subset of letters taken from *height map* and *normal map*.

3.1.2 Source code

You can access the source code at: https://github.com/prideout/heman

3.2 Heman Images

All functions with the `heman_image_` prefix are meant for creating empty images, freeing memory, or examining image contents.

Images are simply arrays of floats. By default, the value type is `float`, but this can be overriden by setting the `HEMAN_FLOAT` macro to `double`. By design, integer-typed images are not allowed, although heman provides some conversion utilities (see Import / Export).

Each image has a specified number of *bands*, which is usually 1 (height maps, distance fields) or 3 (colors, normal maps).

heman_image
: Encapsulates a flat array of floats and its dimensions. The struct definition is not public, so clients must refer to it using a pointer.

3.2.1 Creating and Destroying

```
// Allocate a floating-point image with dimensions width x height x nbands.
heman_image* heman_image_create(int width, int height, int nbands);

// Obtain image properties.
void heman_image_info(heman_image*, int* width, int* height, int* nbands);

// Free memory for a image.
void heman_image_destroy(heman_image*);
```

3.2.2 Examining Texels

```
// Peek at the stored texel values.
float* heman_image_data(heman_image*);

// Peek at the given texel value.
float* heman_image_texel(heman_image*, int x, int y);

// Find a reasonable value for the given normalized texture coord.
void heman_image_sample(heman_image*, float u, float v, float* result);
```

3.3 Lighting and AO

All functions with the `heman_lighting_` prefix are meant for doing things that are useful for lighting, like generating normals or ambient occlusion.

3.3.1 Normal Maps

Normal maps are generated using a simple forward differencing algorithm.

*heman_image** **heman_lighting_compute_normals**(*heman_image* heightmap*)
: Given a 1-band heightmap image, create a 3-band image with surface normals. The resulting image values are in **[-1, +1]**.

3.3.2 Ambient Occlusion

Ambient occlusion is computed by doing 16 sweeps across the height map to find horizon points, as described by Sean Barrett here.

*heman_image** **heman_lighting_compute_occlusion** (*heman_image** *heightmap*)
Compute occlusion values for the given heightmap, returning a new single-band image with values in **[0, 1]**.

3.3.3 Complete Lighting

*heman_image** **heman_lighting_apply** (*heman_image** *heightmap*, *heman_image** *colorbuffer*, float *occlusion*, float *diffuse*, float *diffuse_softening*, float* *light_position*)
High-level utility that generates normals and occlusion behind the scenes, then applies simple diffuse lighting.

Parameters

- **heightmap** (heman_image*) – The source height map, must have exactly one band.
- **colorbuffer** (heman_image*) – RGB values used for albedo; must have 3 bands, and the same dimensions as **heightmap**.
- **occlusion** (*float*) – Desired strength of ambient occlusion in **[0, 1]**.
- **diffuse** (*float*) – Desired strength of diffuse lighting in **[0, 1]**.
- **diffuse_softening** (*float*) – Used to flatten the normals by lerping them with **+Z**. Set to **0** to use unaltered normal vectors.
- **light_position** (*float**) – Pointer to three floats representing the light direction.

Heman automatically un-applies gamma to the albedo, then re-applies gamma after lighting. This behavior can be configured using *heman_color_set_gamma*.

3.4 Distance Fields

All functions with the `heman_distance_` prefix are meant for creating distance fields. This is also known as a Euclidean Distance Transform.

3.4.1 API

```
// Create a signed distance field based on the given input, using the very
// fast algorithm described in Felzenszwalb 2012.
heman_image* heman_distance_create_sdf(heman_image* monochrome);
```

3.4.2 Example

Here's an example of a starting image (the "seed") and its resulting signed distance field (SDF).

The above image was generated with the following program:

```
#include <heman.h>
#include "hut.h"

#define OUTFOLDER "build/"
#define INFOLDER "test/"

int main(int argc, char** argv)
{
    heman_image* seed = hut_read_image(INFOLDER "sdfseed.png", 1);
    heman_image* sdf = heman_distance_create_sdf(seed);
    heman_image_destroy(seed);
    hut_write_image(OUTFOLDER "sdfresult.png", sdf, -0.1, 0.1);
    heman_image_destroy(sdf);
}
```

3.5 Color Gradients

This page covers functions with the `heman_color_` prefix.

One-pixel tall images can be interpreted as *gradients*, also known as *lookup tables*.

3.5.1 Creating Gradients

Clients can import a gradient using *heman_import_u8*, or they can create one from scratch:

*heman_image** **heman_color_create_gradient** (int *width*, int *num_colors*, const int* *cp_locations*, const heman_color* *cp_colors*)

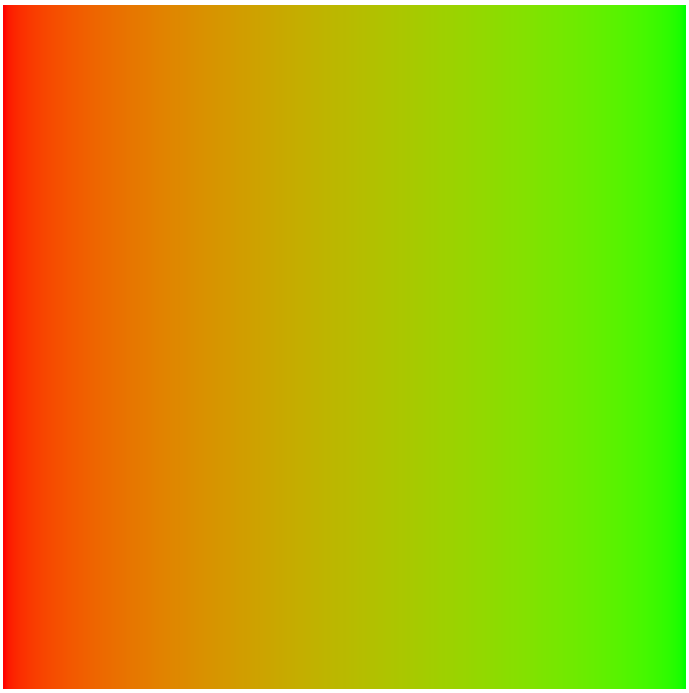
Create a 1-pixel tall, 3-band image representing a color gradient that lerps the given control points.

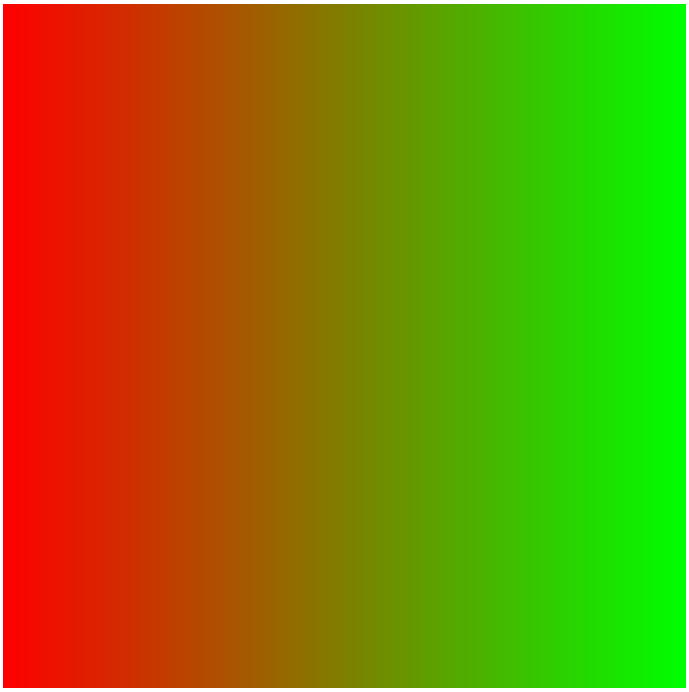
Parameters

- **width** (*int*) – Desired number of entries in the lookup table.
- **num_colors** (*int*) – Number of control points. Must be at least 2.
- **cp_locations** (*int**) – The X coordinate of each control point. The first value must be **0** and the last value must be **width - 1**.
- **cp_colors** (*heman_color**) – The RGB values of each control points.

3.5.2 Gamma Correctness

The following two images both depict the interpolation of **(1,0,0)** to **(0,1,0)**. Can you tell which one is more correct?





The image on the left is more correct; it interpolates the colors by first unapplying gamma to the control point colors, then performing linear interpolation, then re-applying gamma. Heman does this automatically when you call *heman_color_create_gradient*, but the behavior can be controlled with the following function.

void **heman_color_set_gamma** (float *f*)
This sets some global state that affects lighting and color interpolation. The default value is **2.2**.

3.5.3 Applying Gradients

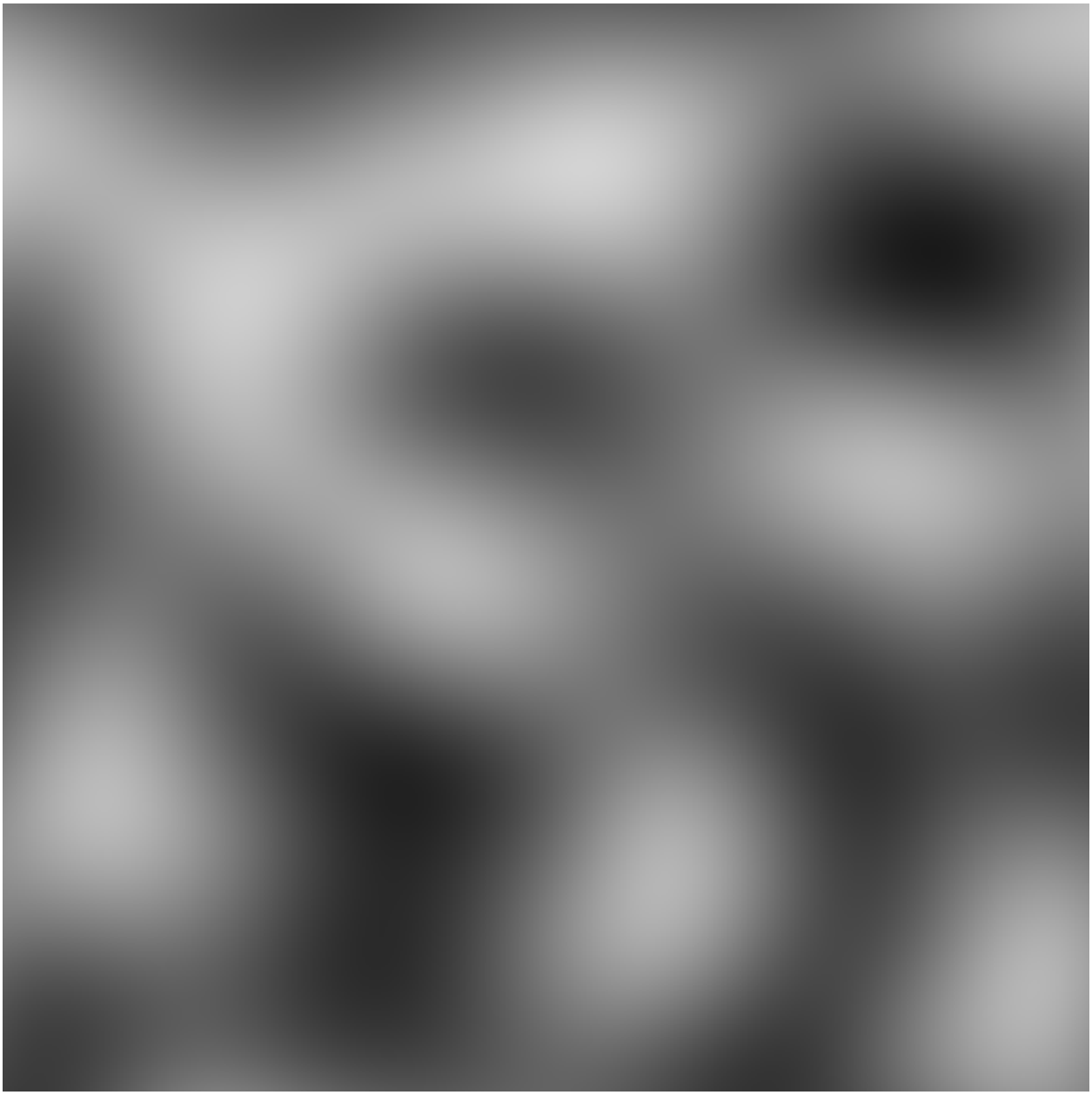
*heman_image** **heman_color_apply_gradient** (*heman_image** *heightmap*, float *minheight*, float *maxheight*, *heman_image** *gradient*)
Create a 3-band image with the same dimensions as the given heightmap by making lookups from a 1-pixel tall color gradient. The heightmap values are normalized using the given minheight, maxheight range.

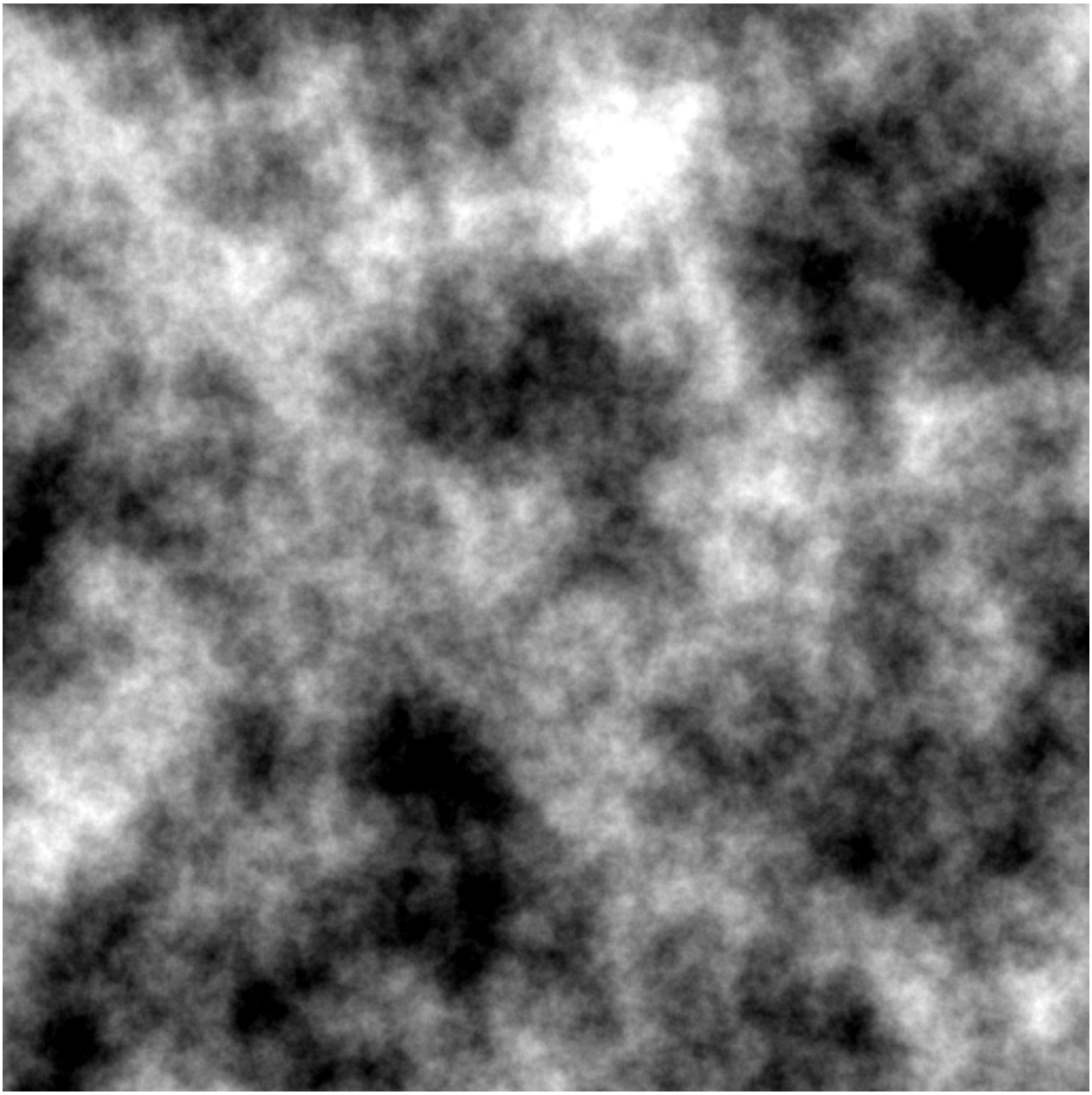
3.6 Image Generation

All functions with the `heman_generate_` prefix are meant to help in the creation of interesting procedural imagery.

3.6.1 Noise and FBM

The image on the left is Ken Perlin's simplex noise function, which is nice and continuous, but non-fractal. The image on the right adds up several octaves of that same noise function; this is known as *Fractional Brownian Motion* (FBM). This provides a way of generating fractal-like images that look cool when interpreted as a height map.





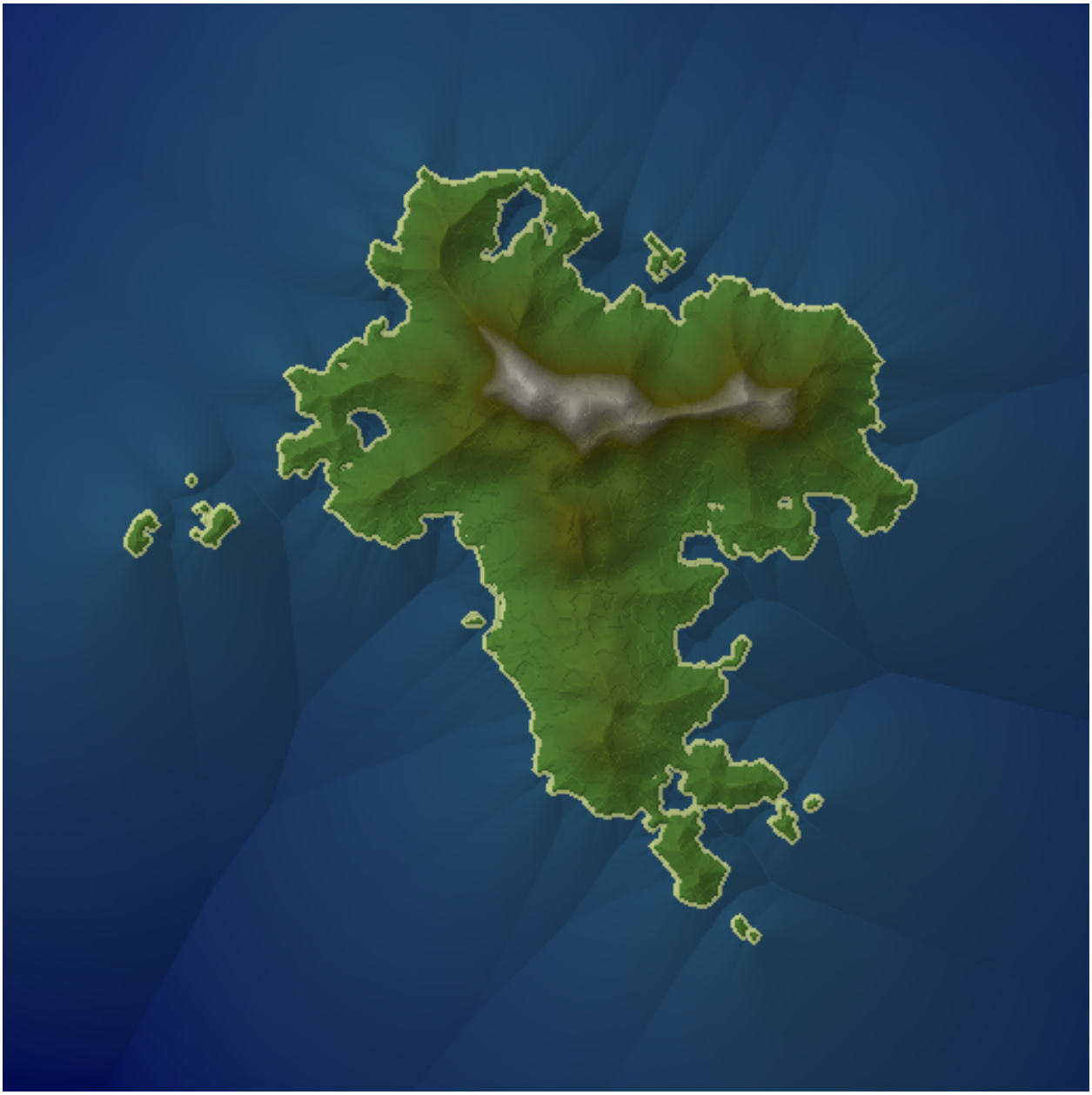
*heman_image** **heman_generate_simplex_fbm**(int *width*, int *height*, float *frequency*, float *amplitude*, int *octaves*, float *lacunarity*, float *gain*, int *seed*)

Sums up a number of noise octaves and returns the result. A good starting point is to use a lacunarity of 2.0 and a gain of 0.5, with only 2 or 3 octaves.

3.6.2 Islands

*heman_image** **heman_generate_island_heightmap**(int *width*, int *height*, int *seed*)

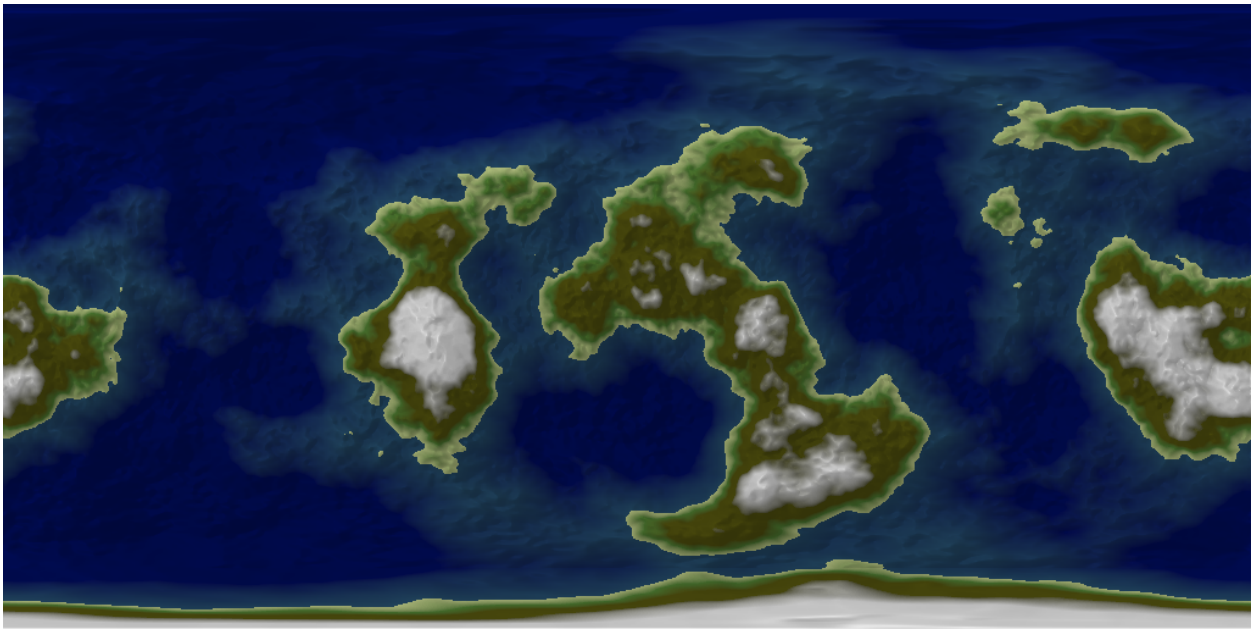
High-level function that uses several octaves of simplex noise and a signed distance field to generate an interesting height map.



3.6.3 Planets

*heman_image** **heman_generate_planet_heightmap** (int *width*, int *height*, int *seed*)

High-level function that sums up several octaves of OpenSimplex noise over a 3D domain to generate an interesting lat-long height map. Clients should specify a **width** that is twice the value of **height**.



3.7 Image Operations

All functions with the `heman_ops_` prefix are meant for doing very simple image operations that are outside of heman's core functionality.

```
// Given a set of same-sized images, copy them into a horizontal filmstrip.
heman_image* heman_ops_stitch_horizontal(heman_image** images, int count);

// Given a set of same-sized images, copy them into a vertical filmstrip.
heman_image* heman_ops_stitch_vertical(heman_image** images, int count);

// Transform texel values so that [minval, maxval] map to [0, 1] and return the
// result.  Values outside the range are clamped.  The source image is
// untouched.
heman_image* heman_ops_normalize_f32(
    heman_image* source, HEMAN_FLOAT minval, HEMAN_FLOAT maxval);

// Generate a monochrome image by applying a step function.
heman_image* heman_ops_step(heman_image* image, HEMAN_FLOAT threshold);

// Generate a height x 1 x 1 image by averaging the values across each row.
heman_image* heman_ops_sweep(heman_image* image);
```

3.8 Import / Export

Heman only knows how to work with in-memory floating-point images. It doesn't know how to read and write image files, although the test suite uses stb for handling image files. See the heman utility header (hut.h) for an example of this.

Heman can, however, convert floating-point to unsigned bytes, or vice versa, using one of the following functions.

*heman_image** **heman_import_u8** (int *width*, int *height*, int *nbands*, const heman_byte* *source*, float *minval*, float *maxval*)

Create a single-channel floating point image from bytes, such that [0, 255] maps to the given [minval, maxval] range.

void **heman_export_u8** (*heman_image** *source*, float *minval*, float *maxval*, heman_byte* *dest*)

Transform texel values so that [minval, maxval] maps to [0, 255], and write the result to "dest". Values outside the range are clamped.

3.8.1 Example with STB

This function uses `stbi_load` to load the given PNG file and convert it into a floating-point image in the range **[0, 1]**.

```
heman_image* read_image(const char* filename, int nbands)
{
    int width = 0, height = 0;
    stbi_uc* bytes;
    heman_image* retval;
    bytes = stbi_load(filename, &width, &height, &nbands, nbands);
    assert(bytes);
    printf("%4d x %4d x %d :: %s\n", width, height, nbands, filename);
    retval = heman_import_u8(width, height, nbands, bytes, 0, 1);
    stbi_image_free(bytes);
    return retval;
}
```

3.8.2 3D Mesh Data

Heman can export a binary mesh file representing height field data, where each grid cell in the mesh corresponds to a single texel in the height field:

```
// Create a mesh with (width - 1) x (height - 1) quads.
void heman_export_ply(heman_image*, const char* filename);

// Create a mesh with (width - 1) x (height - 1) quads and per-vertex colors.
void heman_export_with_colors_ply(
    heman_image* heightmap, heman_image* colors, const char* filename);
```

H