Creating Height Maps from Normal Maps

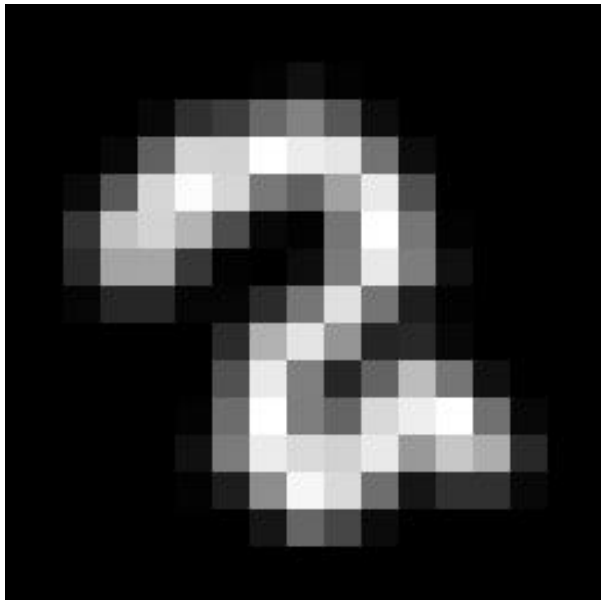
Nathan Cournia

acnatha@vr.clemson.edu

Clemson University

Goal

- Goal: Given a normal map, produce a height map



Motivation

- Original height maps may not be available
- Height maps are easier to edit than normal maps
- Constructing a surface from a height map is trivial

Background: Normal Vector

- The normal vector of a surface is a vector perpendicular to the surface
- Usually a unit vector
- Given a surface define by the bivariate vector function $\mathbf{P}(u, v)$ the surface normal for a point on $\mathbf{P}$ is defined by:

$$\mathbf{N}(u, v) = \frac{\partial \mathbf{P}(u,v)}{\partial u} \times \frac{\partial \mathbf{P}(u,v)}{\partial v}$$

Background: Normal Map

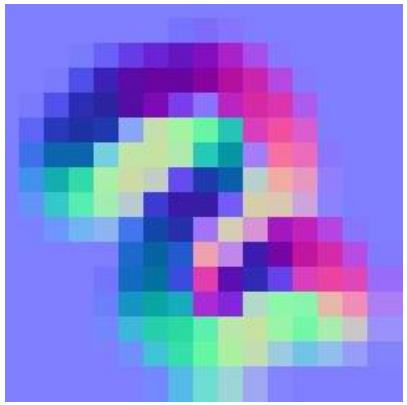
- A normal map is a RGB image, each pixel of which represents a normal
- Each channel of each pixel represents a component of the normal

$$r \rightarrow x$$
$$g \rightarrow y$$
$$b \rightarrow z$$

Background: Normal Map (cont.)

- Normals are scaled from $[-1, 1]$ to $[0, 255]$ as follows:

$$RGB_i = 255 \times \frac{\mathbf{N}_i + 1.0}{2.0}$$



Background: Normal Map Creation

- Given a height map H, generate tangent vectors alight to the s and t directions as follows:

$$\mathbf{S}(i, j) = \langle 1, 0, aH(i+1, j) - aH(i-1, j) \rangle$$
$$\mathbf{T}(i, j) = \langle 1, 0, aH(i, j+1) - aH(i, j-1) \rangle$$

- Where a is a scaling factor

Background: Normal Map Creation

- The normal can then be compute as follows:

$$\mathbf{N}(i,\, j) = \frac{\mathbf{S}(i,j)\times\mathbf{T}(i,j)}{\|\mathbf{S}(i,j)\times\mathbf{T}(i,j)\|}$$

- The normal is then scaled to $[0, 255]$ and stored in the map

Simple Algorithm

- A seemingly simple way to convert the normal map to a height map is:
  - Seed all height values to 0.0
  - For each pixel in the height map:

$$H_k(i, j) = H_k(i, j-1) - \Delta H_k(i, j)$$

  - Where $k \in \{x, y\}$
  - and $\Delta H_k(i, j) = \frac{\mathbf{N}_k(i,j)}{\mathbf{N}_z(i,j)}$

Simple Algorithm (cont.)

- Processes x and y components separately
- Works for simple images
- Noise causes error accumulation (ex: JPEG compression artifacts)

Better Solution

- Create a linear system:

$$\begin{bmatrix} a_{1,0} & a_{1,1} & \cdots & a_{1,w\times h} \\ a_{2,0} & a_{2,1} & \cdots & a_{2,w\times h} \\ a_{3,0} & a_{3,1} & \cdots & a_{3,w\times h} \\ a_{4,0} & a_{4,1} & \cdots & a_{4,w\times h} \\ \vdots & \vdots & \ddots & \vdots \\ a_{w\times h\times 2,0} & a_{w\times h\times 2,1} & \cdots & a_{w\times h\times 2,w\times h} \end{bmatrix} \begin{bmatrix} H(1,1) \\ H(1,2) \\ \vdots \\ H(w,h) \end{bmatrix} = \begin{bmatrix} N_x(1,1) \\ N_y(1,1) \\ \vdots \\ \vdots \\ N_x(w,h) \\ N_y(w,h) \end{bmatrix}$$

Linear System

- System follows:

$$\mathbf{N}_x(i, j) = H(i, j) - H(i, j + 1)$$
$$\mathbf{N}_y(i, j) = H(i, j) - H(i + 1, j)$$

- Right and bottom borders follow:

$$\mathbf{N}_x(i, j) = H(i, j)$$
$$\mathbf{N}_y(i, j) = H(i, j)$$

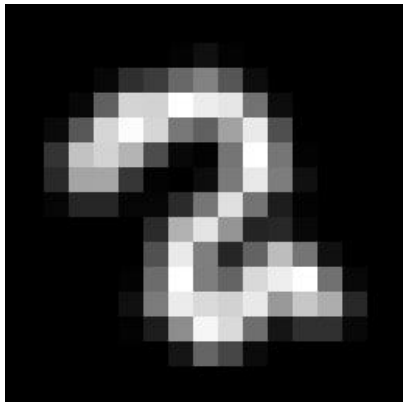
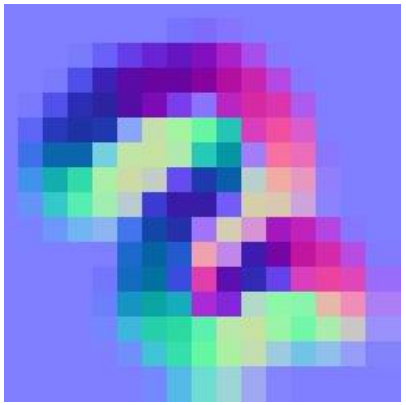
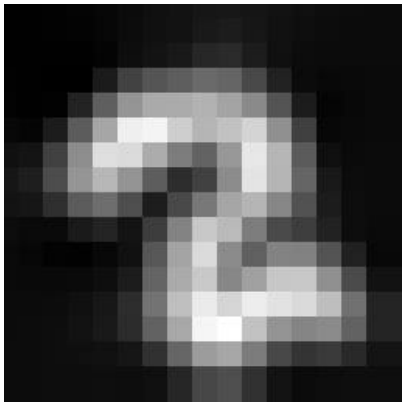
System Example

- A 2×2 normal map would lead to the following augmented matrix:

$$\left[\begin{array}{cccc|c} 1 & -1 & 0 & 0 & \mathbf{N}_x(1,1) \\ 1 & 0 & -1 & 0 & \mathbf{N}_y(1,1) \\ 0 & 1 & 0 & 0 & \mathbf{N}_x(1,2) \\ 0 & 1 & 0 & -1 & \mathbf{N}_y(1,2) \\ 0 & 0 & 1 & -1 & \mathbf{N}_x(2,1) \\ 0 & 0 & 1 & 0 & \mathbf{N}_y(2,1) \\ 0 & 0 & 0 & 1 & \mathbf{N}_x(2,2) \\ 0 & 0 & 0 & 1 & \mathbf{N}_y(2,2) \end{array}\right]$$

Linear System (cont.)

- System is:
  - Overdetermined
  - Nonhomogeneous
- Results:

Linear System Notes

- Results are acceptable but not perfect
- Scaling term and the normal map creation method introduce error into the linear system
- Larger the map, better the results

Linear System Problems

- Solving the system is resource intensive
- 16×16 image: 5 seconds to solve
- 32×32 image: 5 minutes to solve
- 128×256 image: 16GB required to store system

Timings on Pentium 4 3.2GHz

Future

- Break image up into tiles
- Exploit sparse matrix