February 18-22, 2008
San Francisco

www.gdconf.com

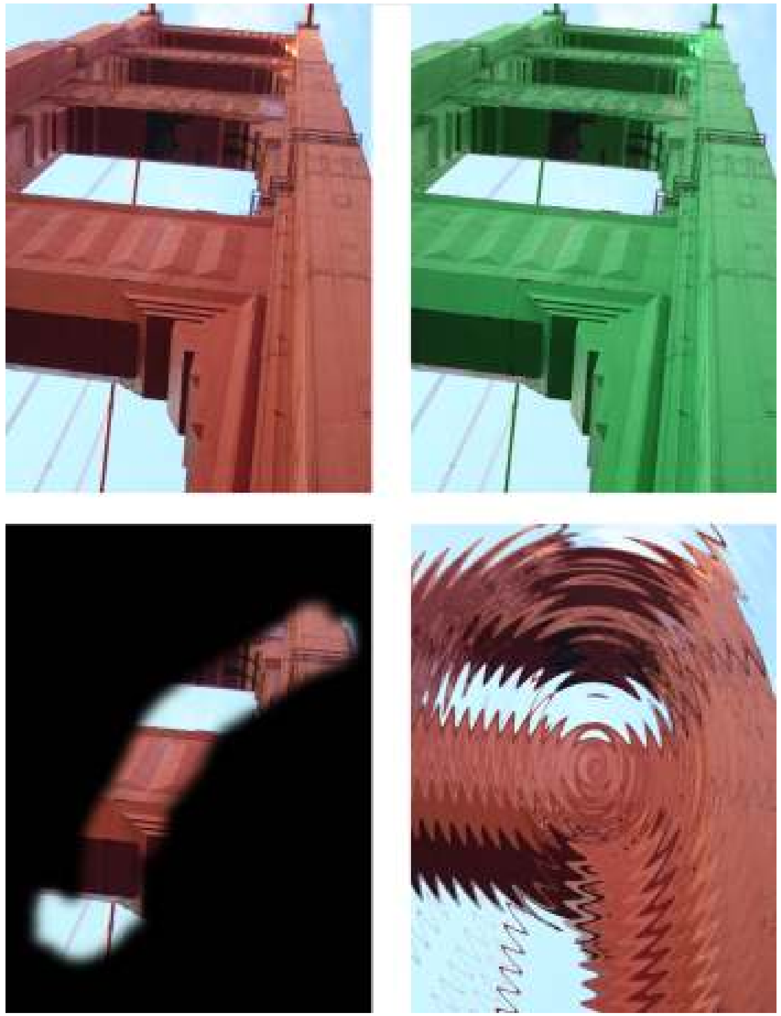
Care and Feeding of Normal Vectors

Jörn Loviscach
Hochschule Bremen (University of Applied Sciences)
Bremen, Germany

joern.loviscach@hs-bremen.de

What is this about? (1)

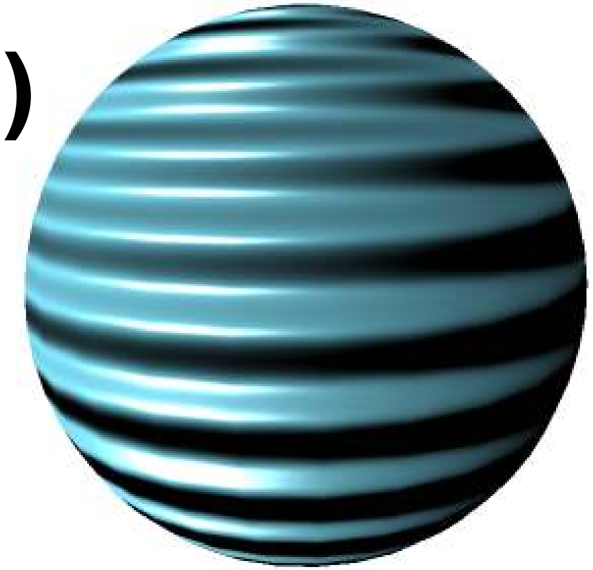
- We can do lots of image editing operations on the GPU
- Let's employ them to make more out of normal maps
- To save memory or to add real-time effects



What is this about? (2)

- But: Standard image editing operations cause unnatural normals.

Original

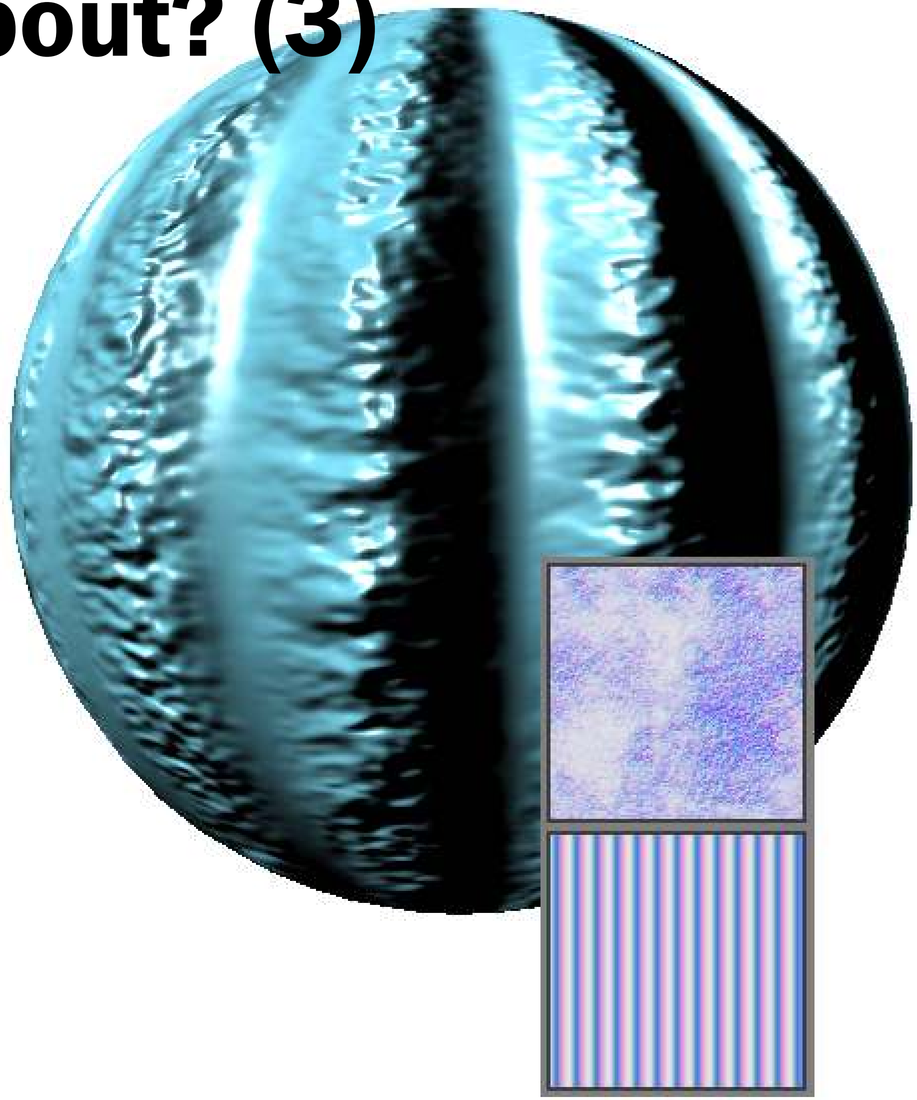


Improved

Standard deformation

What is this about? (3)

- Do this fast and correctly:
  - Scale, add, blend
  - Modulate
  - Apply curves
  - Deform uv



Outline

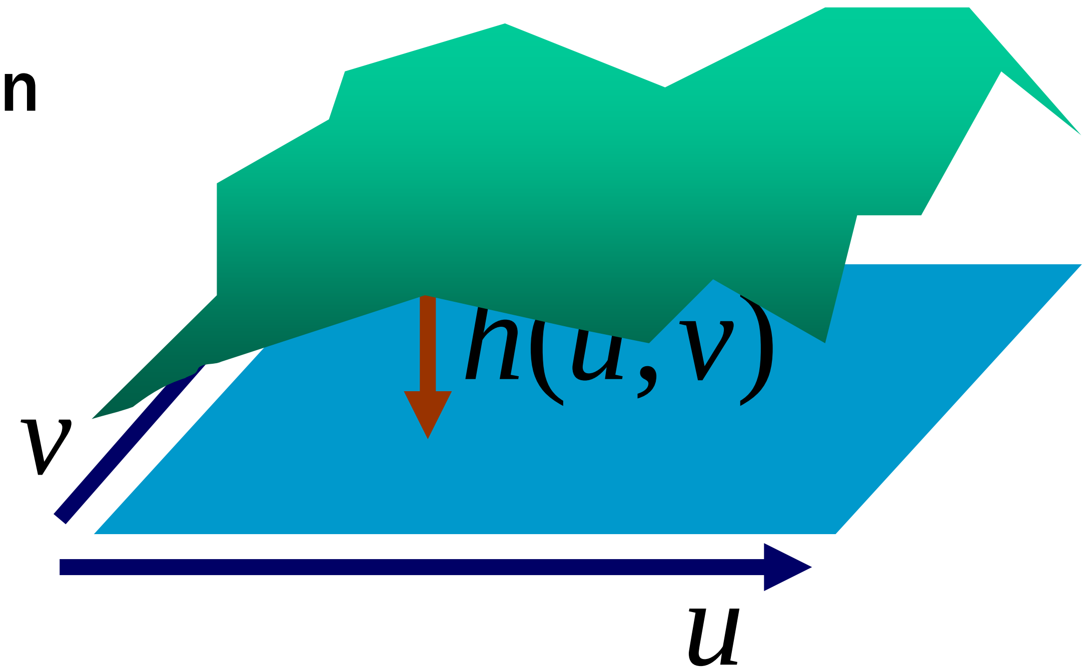
- Birth
- Feeding
  - Standard image processing operations done right
- Care
  - Generate tangents; store and antialias normals

Birth of Normal Vectors

- Height field
- Tangent frame
- Partial derivative

Height Field

- For starters: a *planar* base
- Elevation given per texel



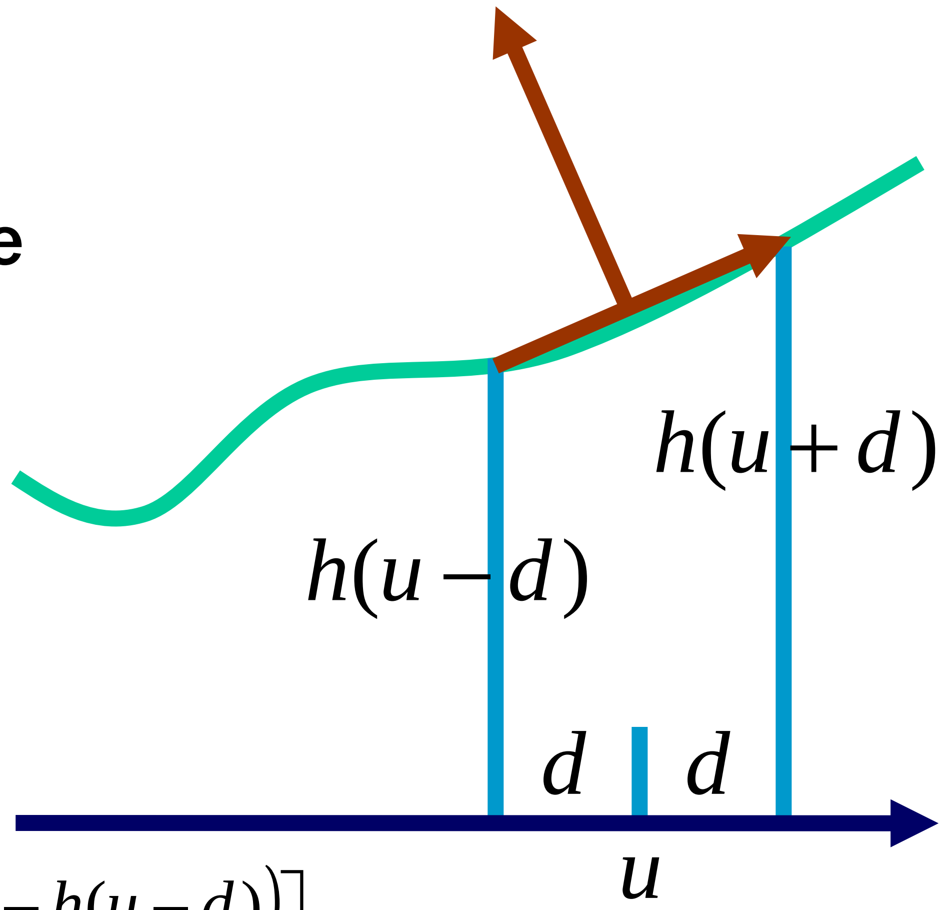
Normal in 1D

- Direction along the surface in 1D:

$$\begin{bmatrix} 2d \\ h(u+d)-h(u-d) \end{bmatrix}$$

- Normal to the surface in 1D:

$$\text{normalize}\begin{bmatrix} -\left(h(u+d)-h(u-d)\right) \\ 2d \end{bmatrix}$$



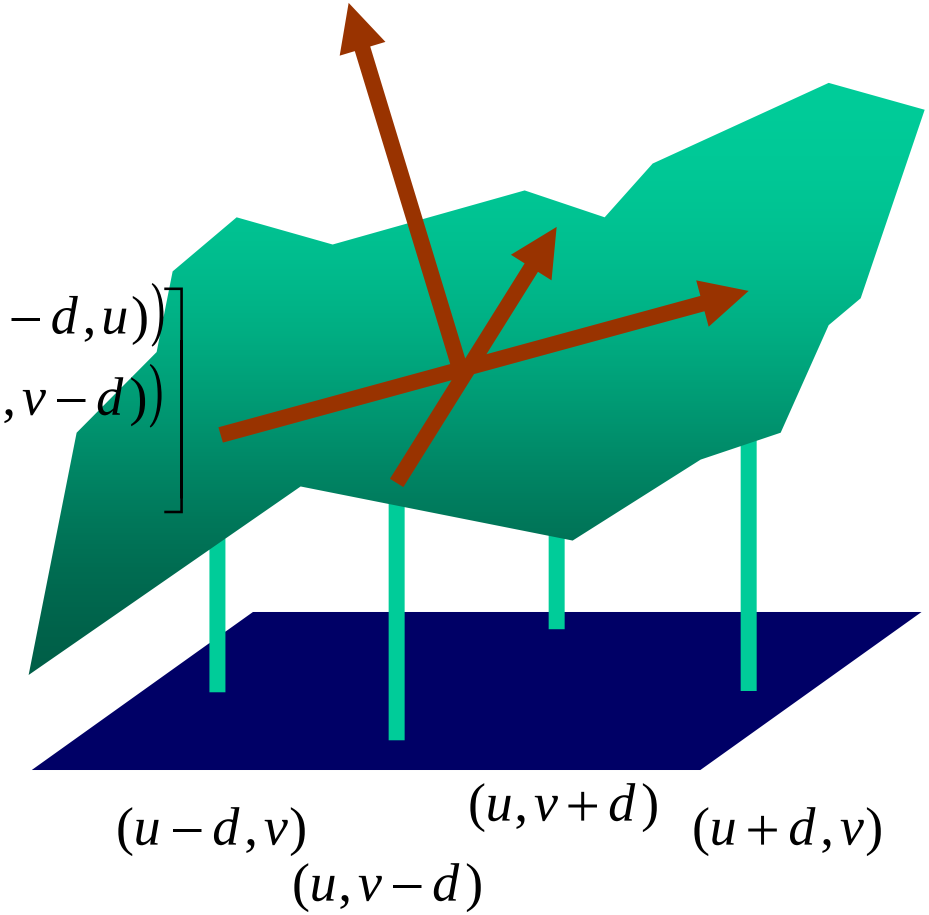
Normal in 2D

- Normal to the surface in 2D:

$$\text{normalize}\begin{bmatrix} -\left(h(u+d,v)-h(u-d,u)\right) \\ -\left(h(u,v+d)-h(u,v-d)\right) \\ 2d \end{bmatrix}$$

- This is mostly what ends up in the normal map.

$$\textbf{map}(u,v) = \text{as above}$$

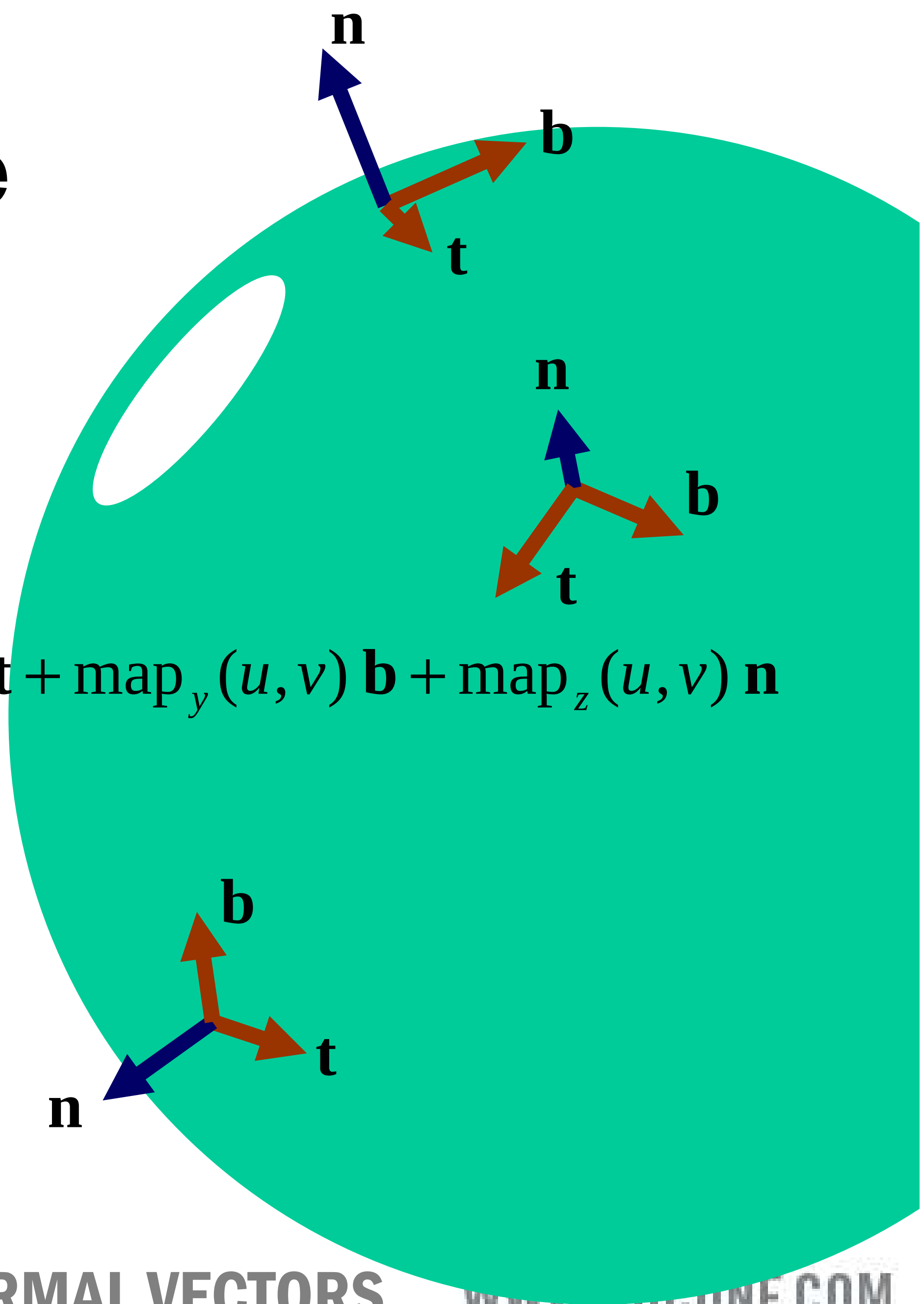


Tangent Frame

- For curved surfaces transplant the normal to the local tangent frame:

$$\mathbf{n}_{\text{map}}(u,v) = \text{map}_x(u,v)\,\mathbf{t} + \text{map}_y(u,v)\,\mathbf{b} + \text{map}_z(u,v)\,\mathbf{n}$$

- Geometrically inaccurate;
we'll return to this.



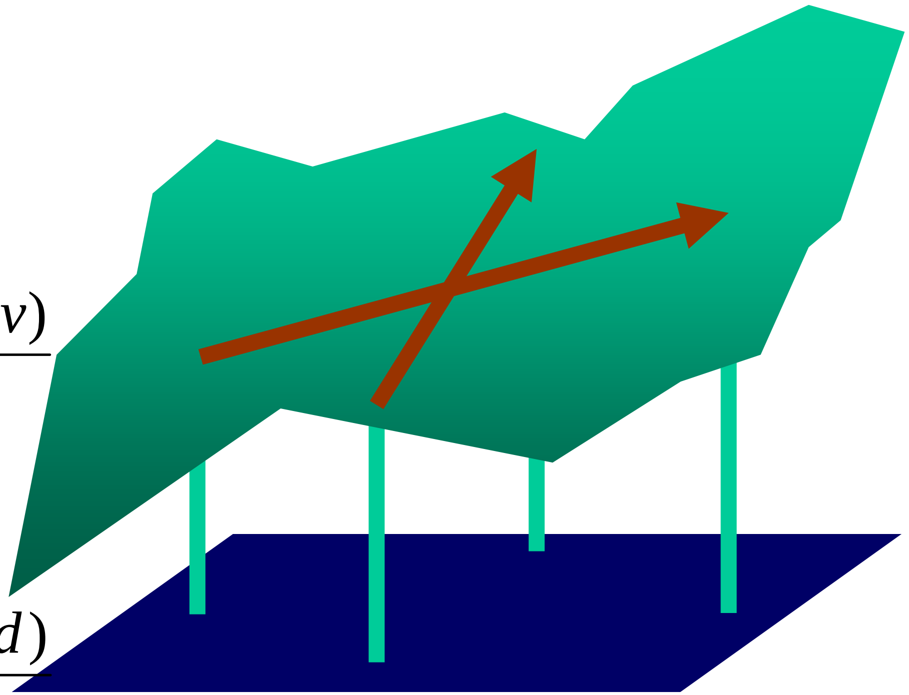
Shorthand: Partial Derivative

- Slope (rate of change) as we move along *u*:

$$\frac{\partial h}{\partial u} = \frac{h(u+d,v) - h(u-d,v)}{2d}$$

- And along *v*:

$$\frac{\partial h}{\partial v} = \frac{h(u,v+d) - h(u,v-d)}{2d}$$



Normal in Short Form

- Normalization cancels the denominator

$$\mathbf{map}(u,v) = \text{normalize}\begin{bmatrix} -\big(h(u+d,v)-h(u-d,v)\big) \\ -\big(h(u,v+d)-h(u,v-d)\big) \\ 2d \end{bmatrix}$$

$$= \text{normalize}\begin{bmatrix} -\big(h(u+d,v)-h(u-d,v)\big)/2d \\ -\big(h(u,v+d)-h(u,v-d)\big)/2d \\ 1 \end{bmatrix} = \text{normalize}\begin{bmatrix} -\partial h/\partial u \\ -\partial h/\partial v \\ 1 \end{bmatrix}$$

Getting the Derivative Back

- Later we need to uncover the derivatives from the normal map

$$\mathbf{map}(u,v) = \text{normalize}\begin{bmatrix} -\partial h / \partial u \\ -\partial h / \partial v \\ 1 \end{bmatrix}$$

$$\frac{\partial h}{\partial u} = -\frac{\text{map}_x(u,v)}{\text{map}_z(u,v)}$$

$$\frac{\partial h}{\partial v} = -\frac{\text{map}_y(u,v)}{\text{map}_z(u,v)}$$

Normalization does not change the ratio.

Feeding of Normal Vectors

- Central idea
- Scaling
- Adding & blending
- Modulation
- Curves
- Deformation

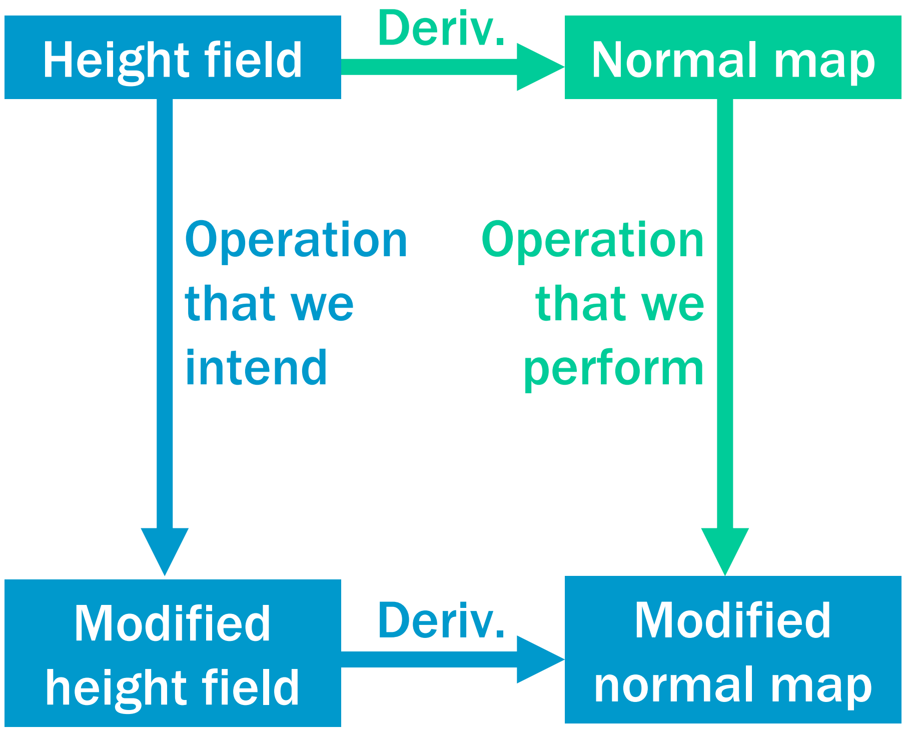
Birth
Height Field
Tangent Frame
Derivative

Feeding
Scaling
Adding
Modulation
Curves
Deformation

Care
Geometry
Channels
Antialiasing

Central Idea

- Apply any operation to the height field, not to the normal vectors.
- **But** compute the result from the normal map.
- Better don't form the derivatives on the GPU.



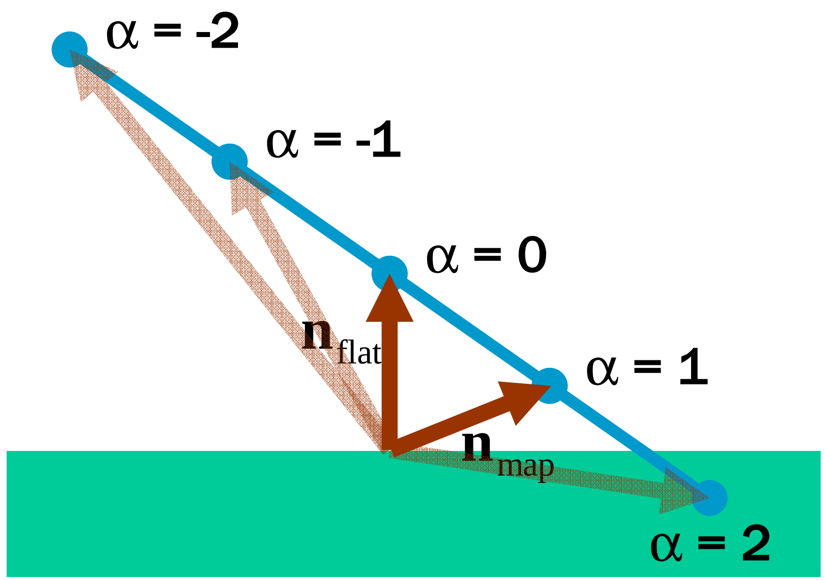
Scaling

Scaling (1)

- Naïve approach:

$$\mathbf{n}_{new} = \mathrm{normalize}(\alpha \mathbf{n}_{map} + (1-\alpha)\mathbf{n}_{flat})$$

- α = -1 does not invert the bump!?
- α = 2 can lead to normals pointing to the inside!?



Scaling (2)

- Apply the scaling to the height field, not directly to the normal map.

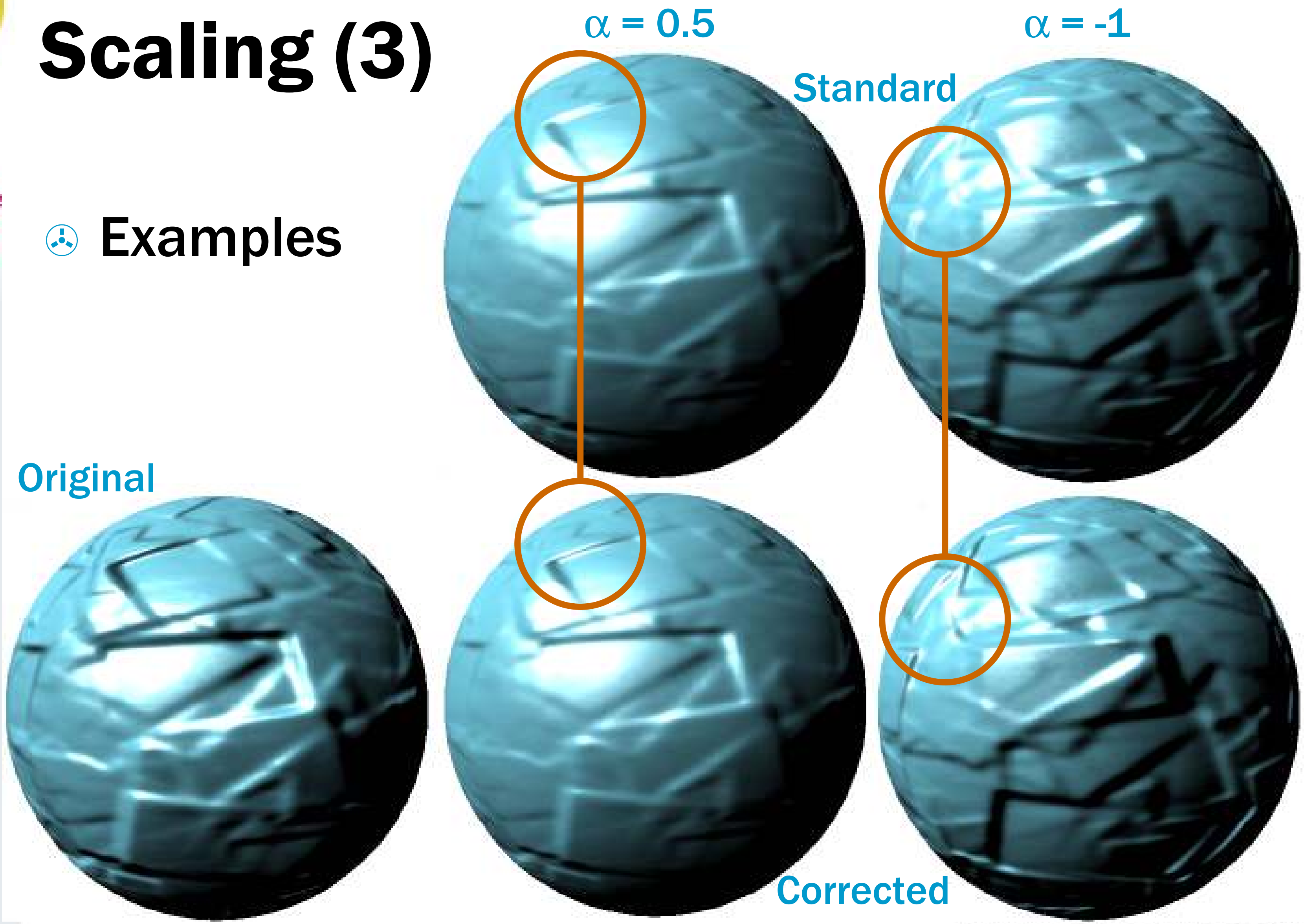
$$\mathbf{scaledMap}(x,y) = \text{normalize}\begin{bmatrix} -\partial\alpha h/\partial u \\ -\partial\alpha h/\partial v \\ 1 \end{bmatrix}$$

$$= \text{normalize}\begin{bmatrix} \alpha\,\text{map}_x(u,v)/\text{map}_z(u,v) \\ \alpha\,\text{map}_y(u,v)/\text{map}_z(u,v) \\ 1 \end{bmatrix} = \text{normalize}\begin{bmatrix} \alpha\,\text{map}_x(u,v) \\ \alpha\,\text{map}_y(u,v) \\ \text{map}_z(u,v) \end{bmatrix}$$

Normalization lets us get rid of the division

Scaling (3)

- Examples



Adding and Blending

Adding and Blending (1)

- Combine two height fields additively: $\alpha h_1(u,v) + \beta h_2(u,v)$

Normalization lets us get rid of the division

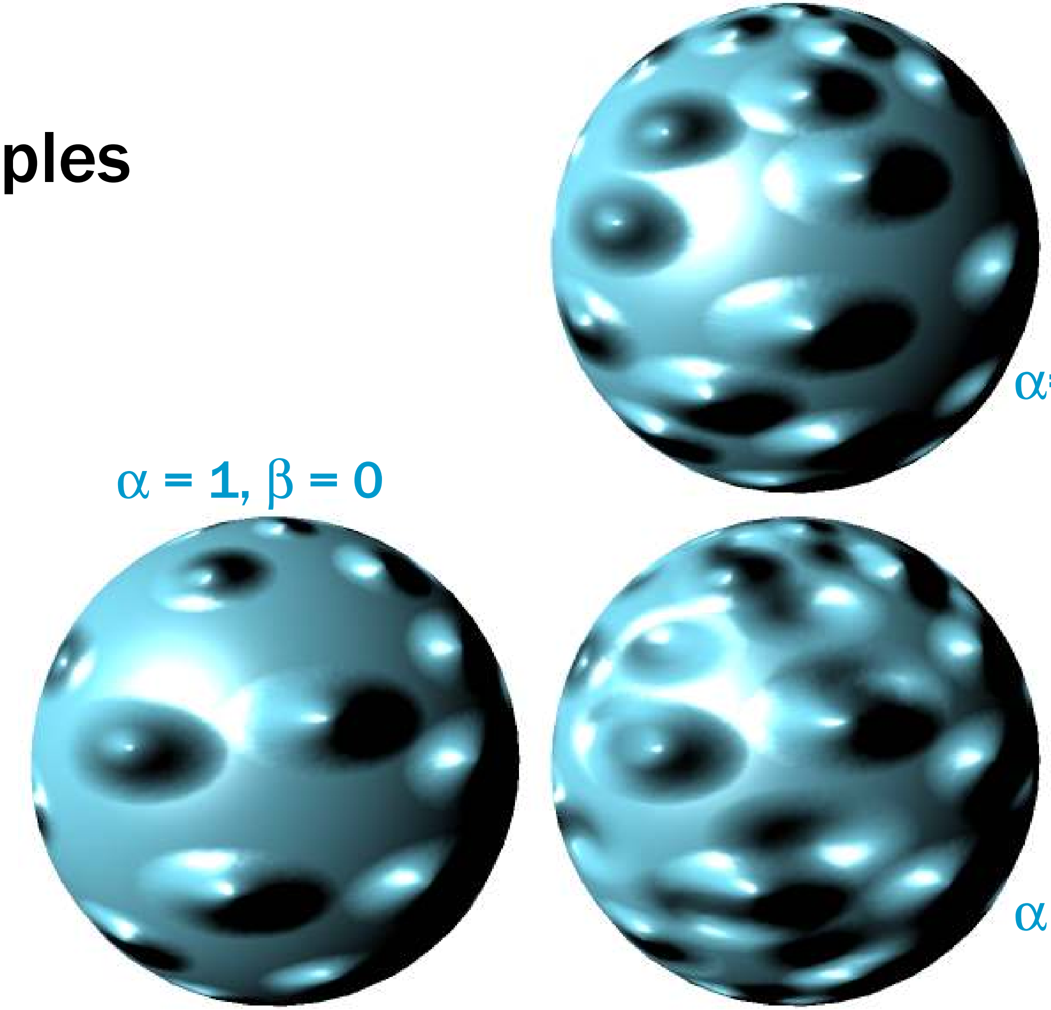
$$\mathbf{summedMap}(u,v)$$

$$= \text{normalize}\begin{bmatrix} \alpha\, \text{map}_{1,x}(u,v) / \text{map}_{1,z}(u,v) + \beta\, \text{map}_{2,x}(u,v) / \text{map}_{2,z}(u,v) \\ \alpha\, \text{map}_{1,y}(u,v) / \text{map}_{1,z}(u,v) + \beta\, \text{map}_{2,y}(u,v) / \text{map}_{2,z}(u,v) \\ 1 \end{bmatrix}$$

$$= \text{normalize}\begin{bmatrix} \alpha\, \text{map}_{1,x}(u,v) \cdot \text{map}_{2,z}(u,v) + \beta\, \text{map}_{2,x}(u,v) \cdot \text{map}_{1,z}(u,v) \\ \alpha\, \text{map}_{1,y}(u,v) \cdot \text{map}_{2,z}(u,v) + \beta\, \text{map}_{2,y}(u,v) \cdot \text{map}_{1,z}(u,v) \\ \text{map}_{1,z}(u,v)\, \text{map}_{2,z}(u,v) \end{bmatrix}$$

Adding and Blending (2)

- Examples



$\alpha = 0, \beta = 1$

$\alpha = 1, \beta = 0$

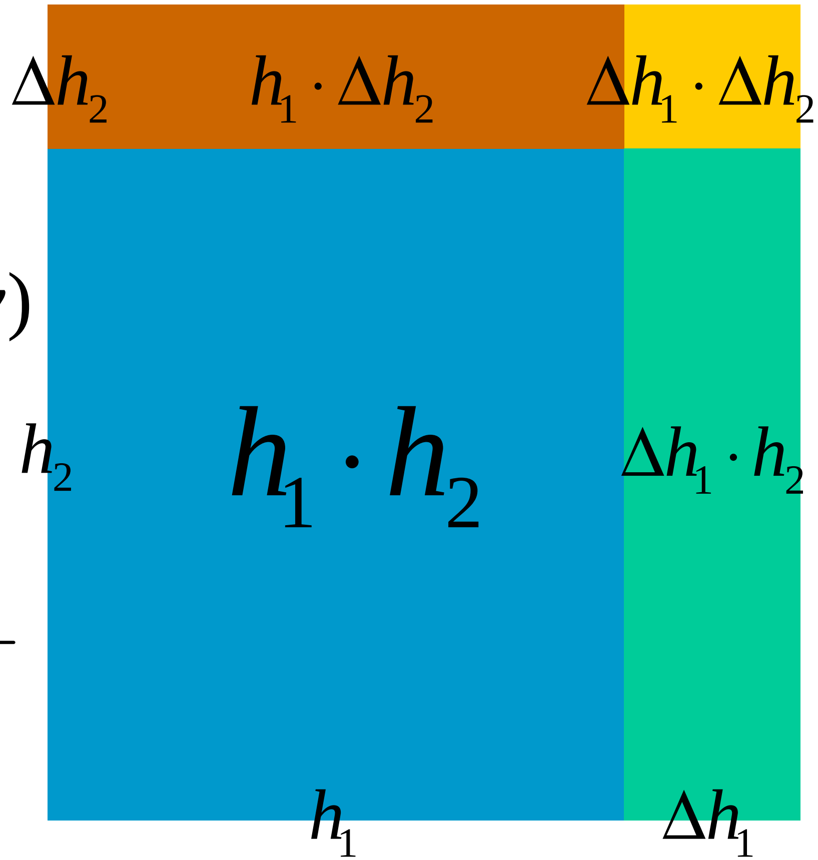
$\alpha = 1, \beta = -0.5$

Modulation

Modulation (1)

- Scale a height field up or down, depending on the position: $h_1(u,v) \cdot h_2(u,v)$
- The rate of change of a product:

$$\frac{\partial h_1 \cdot h_2}{\partial u} = \frac{\partial h_1}{\partial u} h_2 + h_1 \frac{\partial h_2}{\partial u}$$



Modulation (2)

- Result:

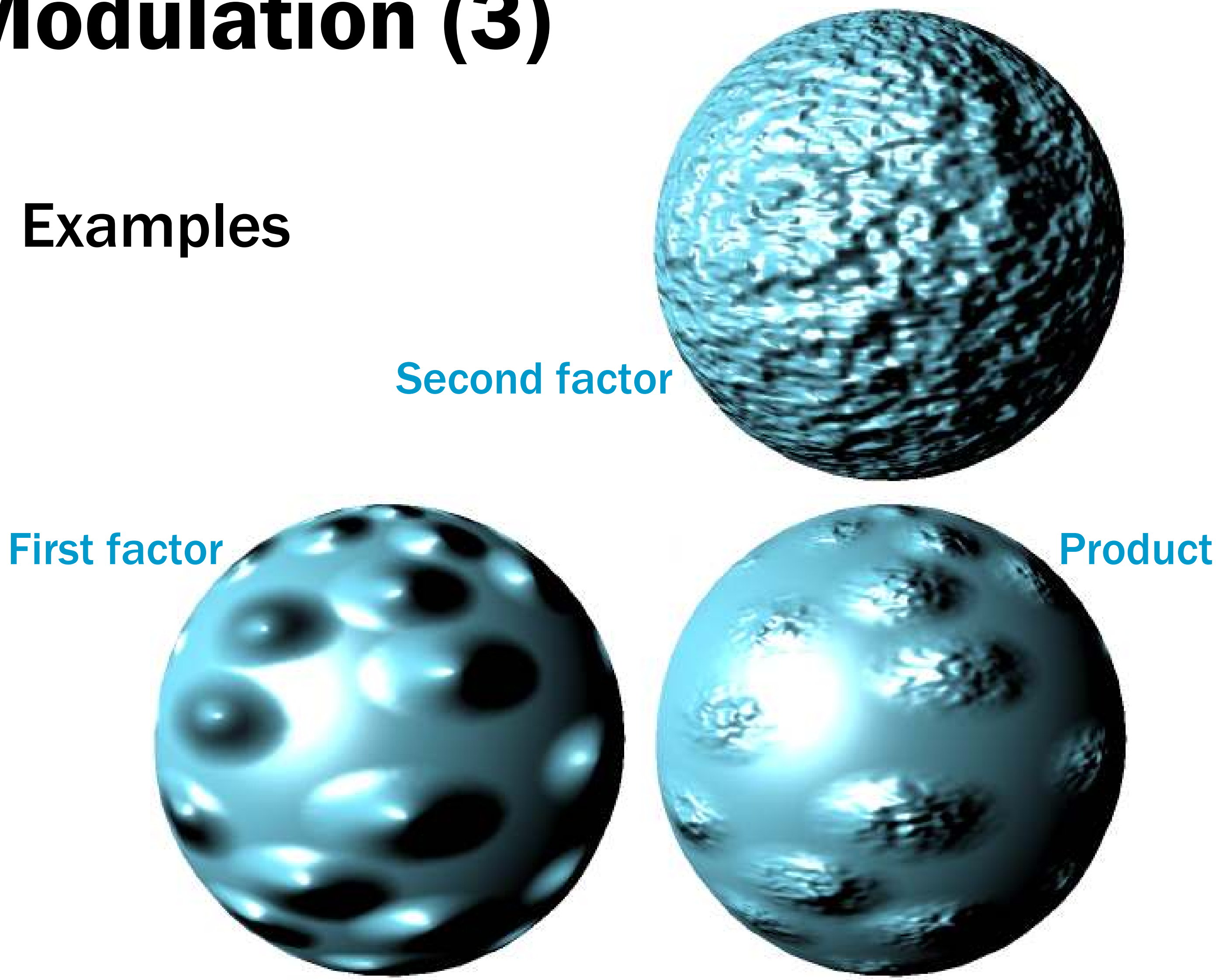
$$\mathbf{modulatedMap}(u,v) = \text{normalize}\begin{bmatrix} \text{map}_{1,x}\, h_2 / \text{map}_{1,z} + \text{map}_{2,x}\, h_1 / \text{map}_{2,z} \\ \text{map}_{1,y}\, h_2 / \text{map}_{1,z} + \text{map}_{2,y}\, h_1 / \text{map}_{2,z} \\ 1 \end{bmatrix}$$

$$= \text{normalize}\begin{bmatrix} \text{map}_{1,x}\text{map}_{2,z}\, h_2 + \text{map}_{2,x}\text{map}_{1,z}\, h_1 \\ \text{map}_{1,y}\text{map}_{2,z}\, h_2 + \text{map}_{2,y}\text{map}_{1,z}\, h_1 \\ \text{map}_{1,z}\text{map}_{2,z} \end{bmatrix}$$

Value of height field needed: use fourth texture channel

Modulation (3)

- Examples

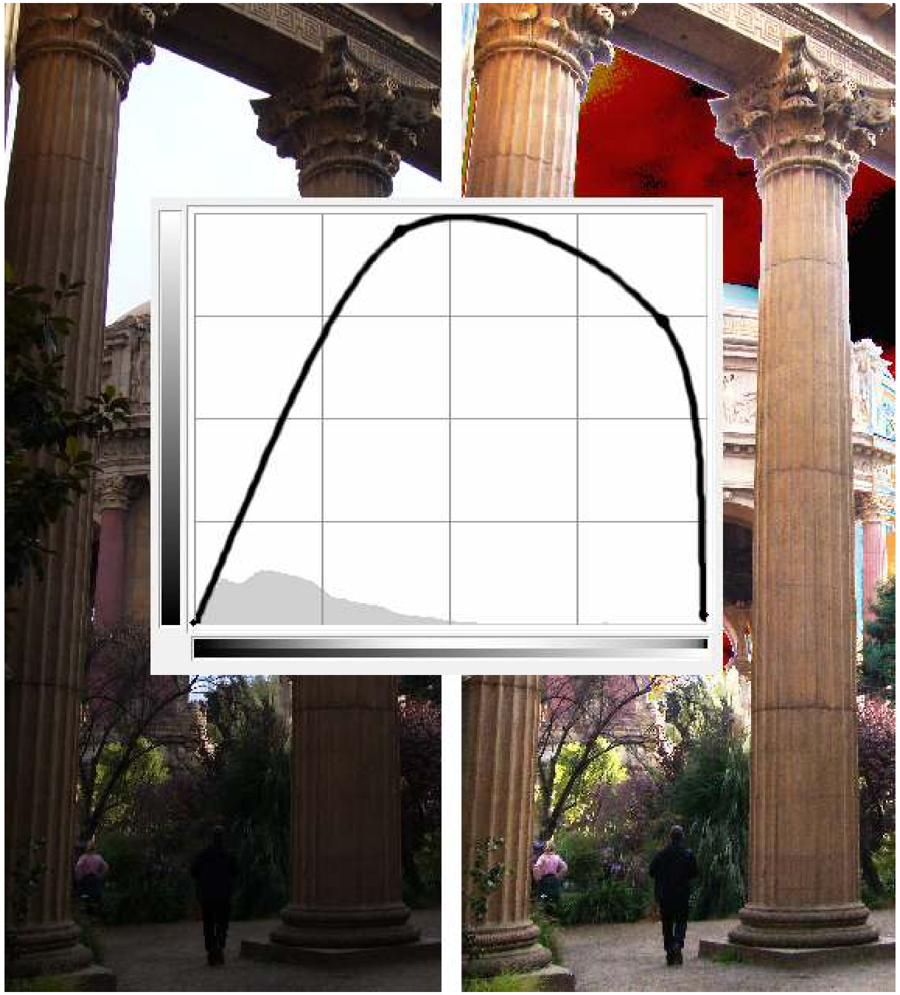


Curves

Curves (1)

- Curves in image processing: Feed every pixel's color through a function:

$$f\left(\text{texture}(u,v)\right)$$



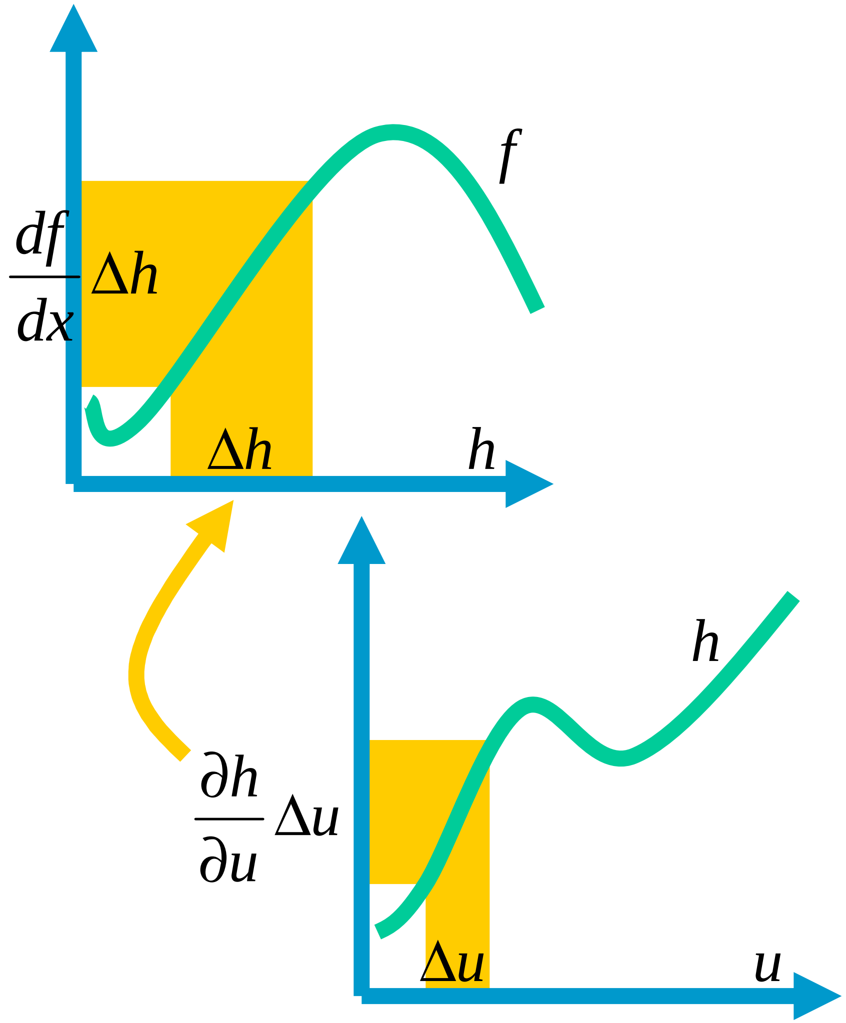
Curves (2)

- **Now feed the *elevation* through a curve:**

  $$f\left(h(u,v)\right)$$

- **The rate of change of a composed function:**

  $$\frac{\partial f(h)}{\partial u} = \left.\frac{df}{dx}\right|_{x=h} \cdot \frac{\partial h}{\partial u}$$



Curves (3)

- The amplitude is modulated by $f'(h) = \left.\frac{df}{dx}\right|_{x=h}$

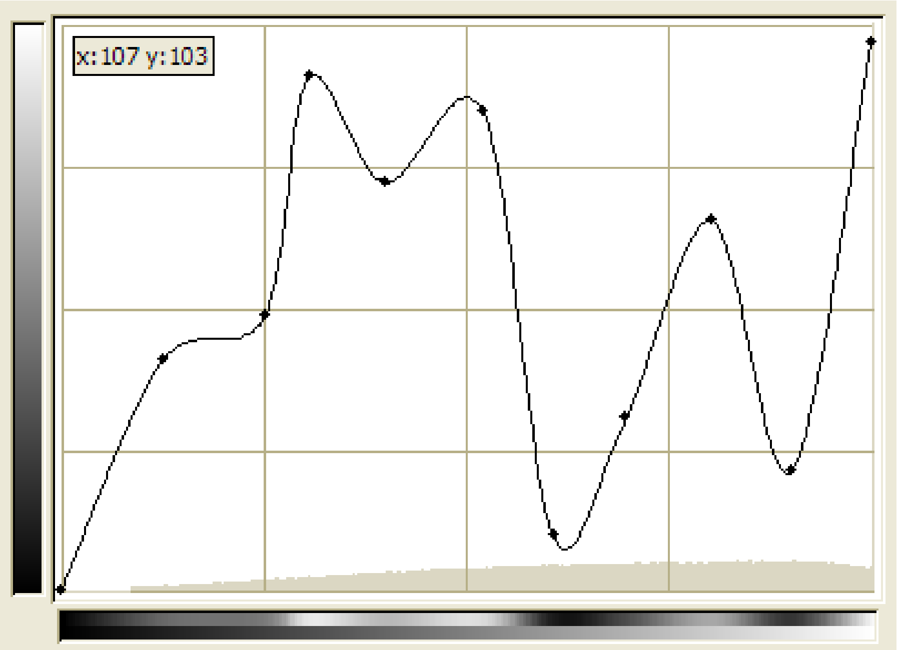
$$\mathbf{composedMap}(u,v) = \text{normalize}\begin{bmatrix} -\partial f\left(h(u,v)\right)/\partial u \\ -\partial f\left(h(u,v)\right)/\partial v \\ 1 \end{bmatrix}$$

Need the elevation (fourth texture channel?) **and** the derivative of the curve

$$= \text{normalize}\begin{bmatrix} -f'(h)\,\partial h(u,v)/\partial u \\ -f'(h)\,\partial h(u,v)/\partial v \\ 1 \end{bmatrix} = \text{normalize}\begin{bmatrix} f'(h)\,\text{map}_x(u,v) \\ f'(h)\,\text{map}_y(u,v) \\ \text{map}_z(u,v) \end{bmatrix}$$

Curves (4)

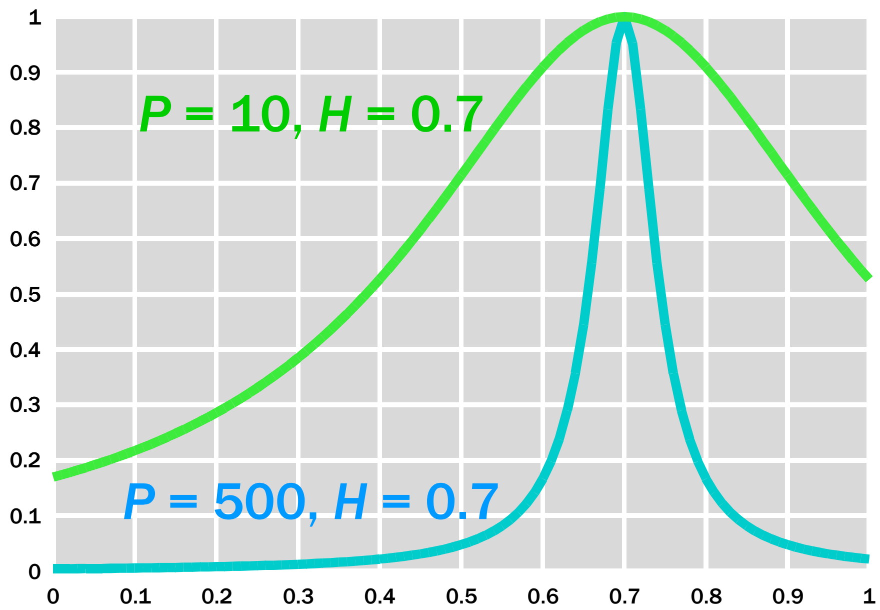
- Where to get f' from:
  - Dependent texture lookup: versatile, possibly slow
  - Algorithmic expression: hard to tune, possibly fast



Birth
Height Field
Tangent Frame
Derivative
Feeding
Scaling
Adding
Modulation
Curves
Deformation
Care
Geometry
Channels
Antialiasing

Curves (5)

- Simple algorithmic expression: boost elevation values around H, control the affected range by P

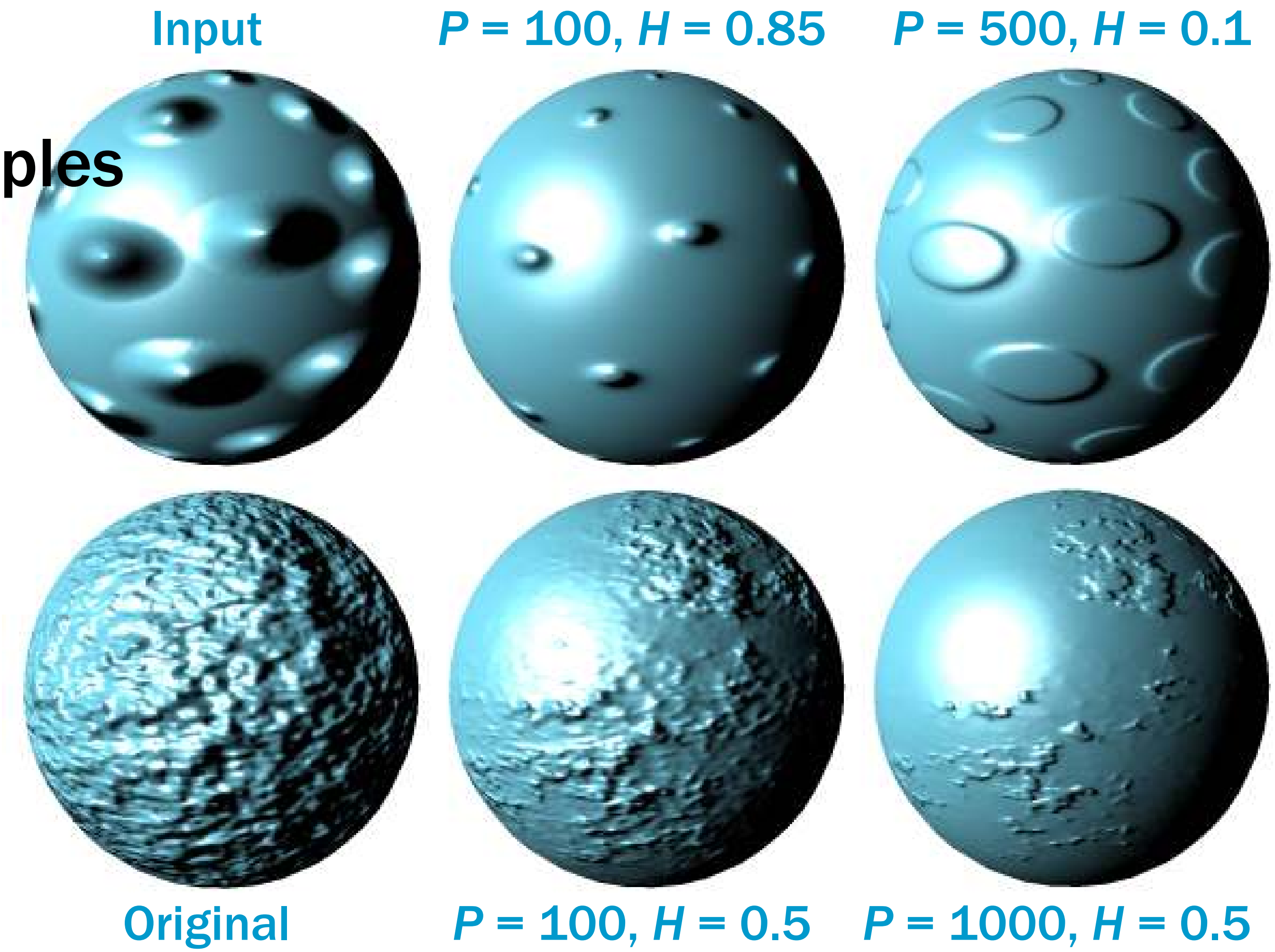


$$f'(h) = \frac{1}{1 + P \cdot (h - H)^2}$$

This is the derivative!

Curves (6)

- Examples

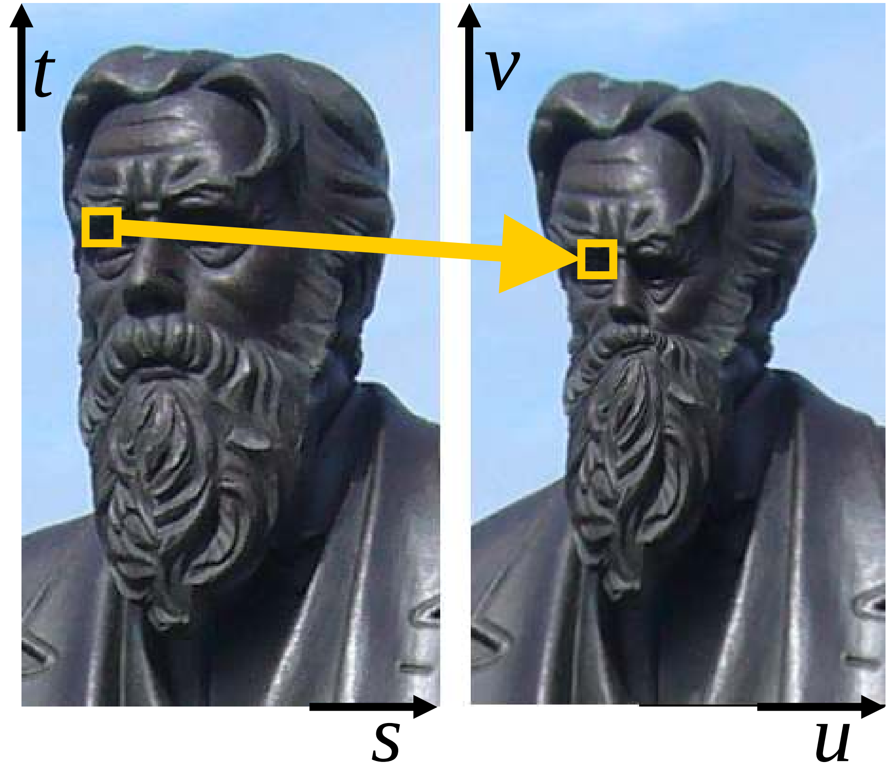


Deformation

Deformation (1)

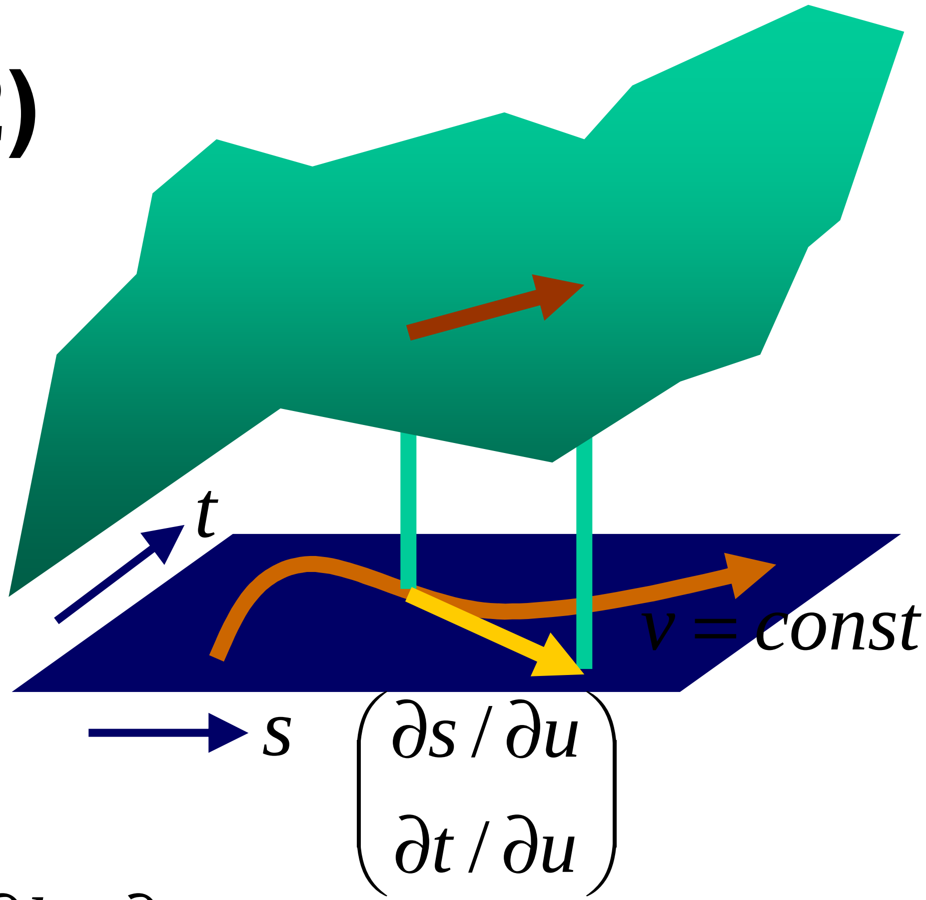
- In image processing: Pick the color for a pixel from another place:

$$\text{texture}\big(s(u,v),t(u,v)\big)$$



Deformation (2)

- Do the same for a height field:
  $$h(s(u,v),t(u,v))$$
- The rate of change of the deformed height field:
  $$\frac{\partial h(s,t)}{\partial u} = \frac{\partial h}{\partial s} \cdot \frac{\partial s}{\partial u} + \frac{\partial h}{\partial t} \cdot \frac{\partial t}{\partial u}$$



Deformation (3)

- This turns out to be a variant of amplitude modulation:

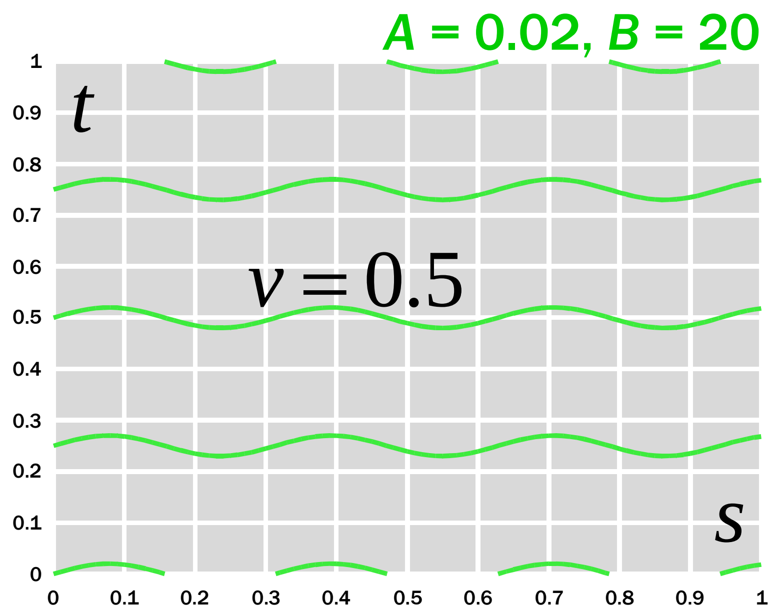
$$\textbf{deformedMap}(u,v) = \mathrm{normalize}\begin{bmatrix} -\partial h(s,t)/\partial u \\ -\partial h(s,t)/\partial v \\ 1 \end{bmatrix}$$

$$= \mathrm{normalize}\begin{bmatrix} \mathrm{map}_x(s,t)\cdot\partial s/\partial u + \mathrm{map}_y(s,t)\cdot\partial t/\partial u \\ \mathrm{map}_x(s,t)\cdot\partial s/\partial v + \mathrm{map}_y(s,t)\cdot\partial t/\partial v \\ \mathrm{map}_z(s,t) \end{bmatrix}.$$

Deformation (4)

- Example:

$$s(u,v) = u$$
$$t(u,v) = v + A\sin(Bu)$$



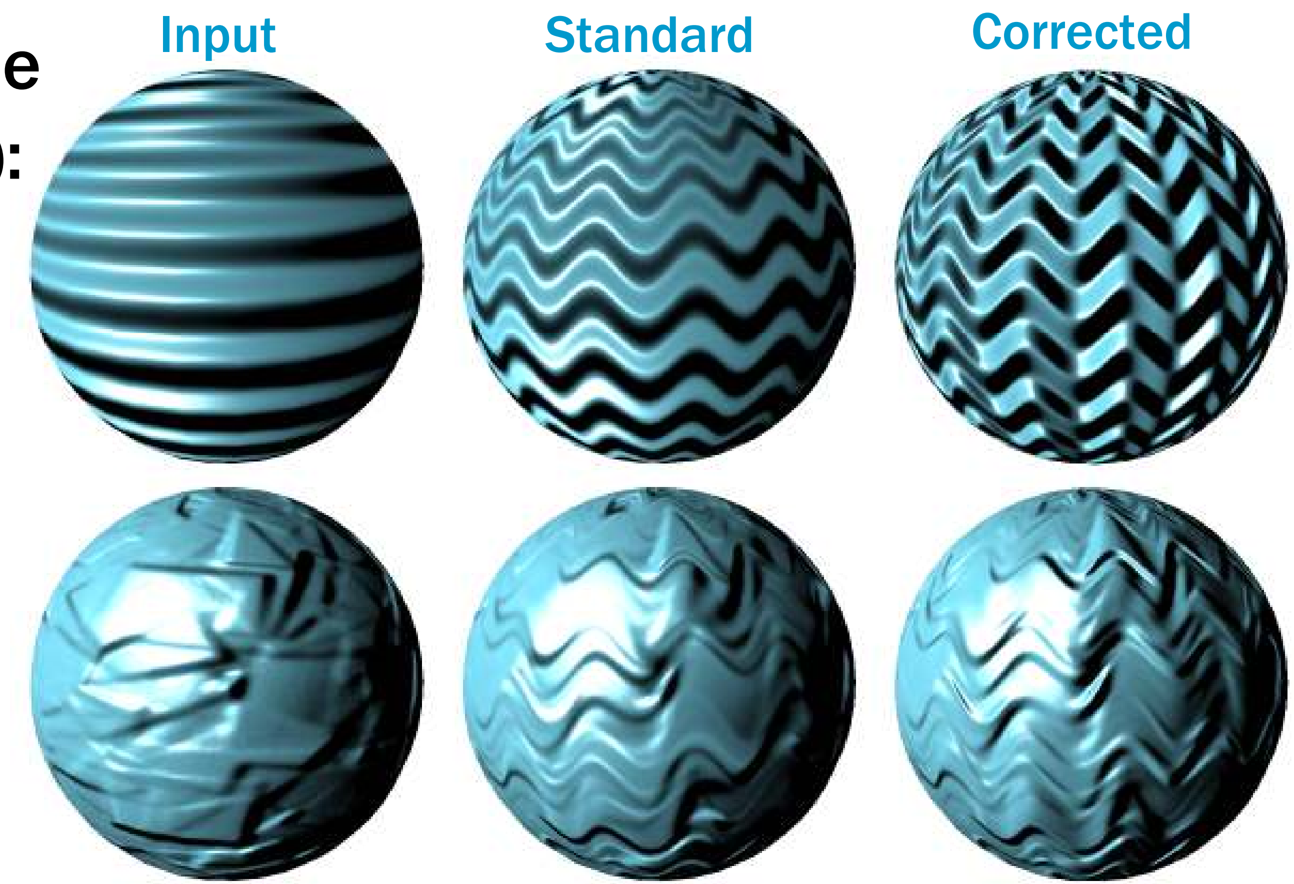
$$\textbf{deformedMap}(u,v) =$$

$$\text{normalize}\begin{bmatrix} \text{map}_x(s,t) + \text{map}_y(s,t)\cdot AB\cos(Bu) \\ \text{map}_y(s,t) \\ \text{map}_z(s,t) \end{bmatrix}$$

Deformation (5)

$A = 0.02$, $B = 100$

- Example (cont'd):

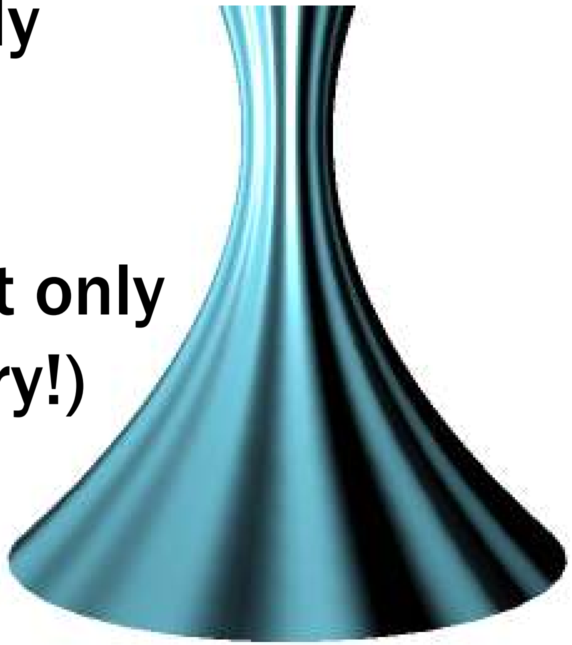


Care of Normal Vectors

- Accurate Geometry
- Two, three, or four channels
- Antialiasing

Accurate Geometry (1)

- If we actually deform the surface, the shading (not only the geometry!) differs from normal mapping.



Normal map: amplitude varies with scale

Height field: constant amplitude

Accurate Geometry (2)

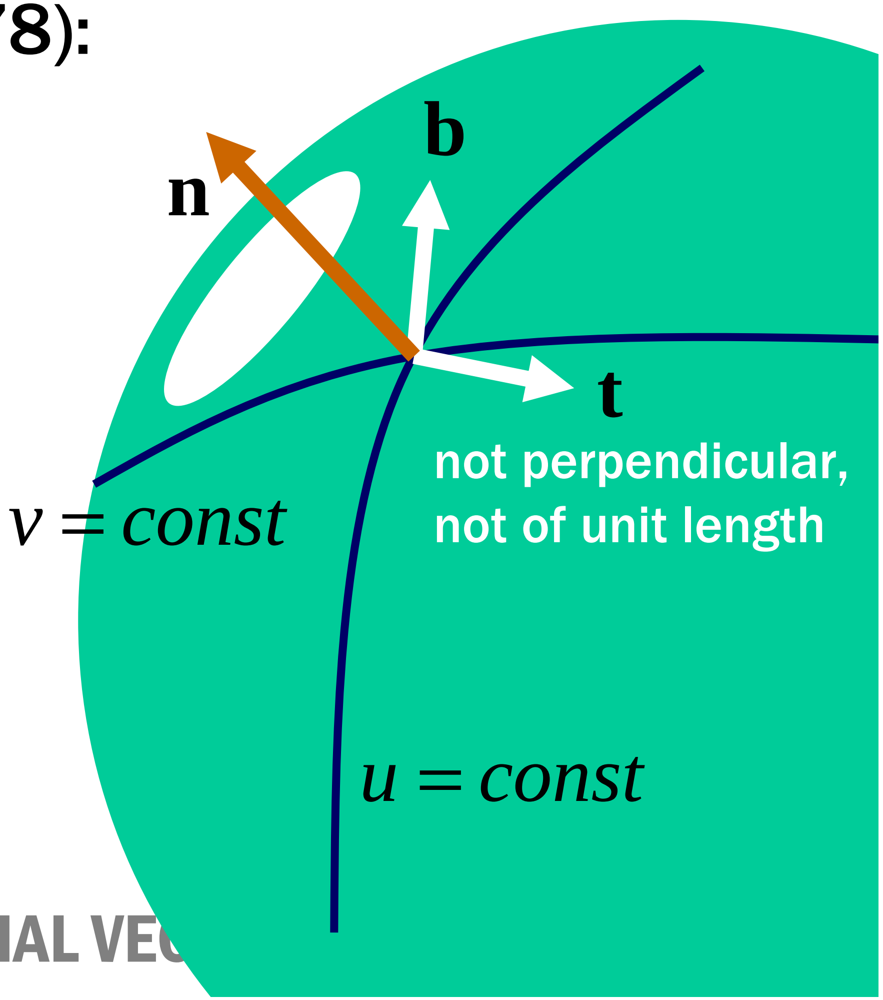
- Accurately (Blinn 1978):

$$\mathbf{t} = \frac{\partial \mathbf{p}}{\partial v} \times \mathbf{n} \Big/ \left| \frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v} \right|,$$

$$\mathbf{b} = \mathbf{n} \times \frac{\partial \mathbf{p}}{\partial u} \Big/ \left| \frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v} \right|,$$

no normalization

for the normal map $\begin{bmatrix} -\partial h / \partial u \\ -\partial h / \partial v \\ 1 \end{bmatrix}$



Two, three, or four channels (1)

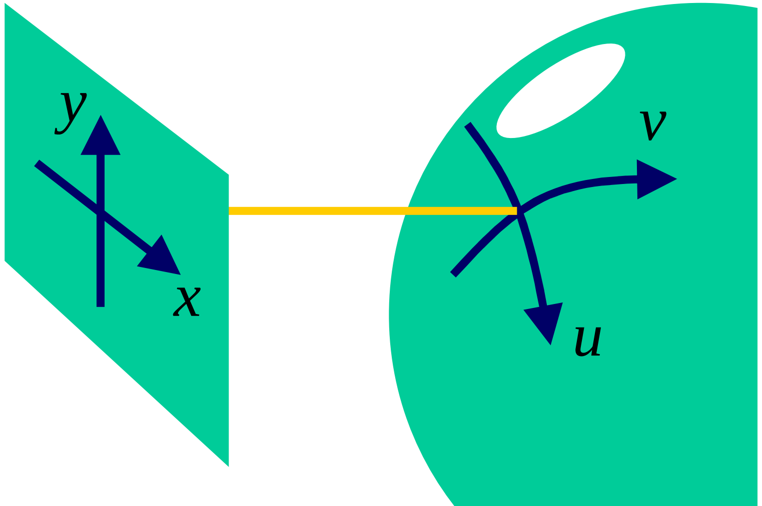
- 3 x 8 bits: $(\text{map}_x, \text{map}_y, \text{map}_z)$
- 2 x 16 bits: $(\text{map}_x, \text{map}_y)$

$$\text{map}_z = \sqrt{\text{map}_x^2 + \text{map}_y^2}$$

- Alternative: $(\partial h / \partial u, \partial h / \partial v)$
  Benefit: sharpen and blur through MIP level bias
- Add height as another channel if needed

Two, three, or four channels (2)

- Alternative: Compute normals on the fly from a height texture.



$$\frac{\partial h}{\partial u} = \frac{\partial h}{\partial x}\cdot\frac{\partial x}{\partial u} + \frac{\partial h}{\partial y}\cdot\frac{\partial y}{\partial u} = \left(\frac{\partial h}{\partial x}\cdot\frac{\partial v}{\partial y} - \frac{\partial h}{\partial y}\cdot\frac{\partial v}{\partial x}\right) \Big/ \left(\frac{\partial u}{\partial x}\cdot\frac{\partial v}{\partial y} - \frac{\partial u}{\partial y}\cdot\frac{\partial v}{\partial x}\right)$$

Convert to expression that can be evaluated with ddx, ddy

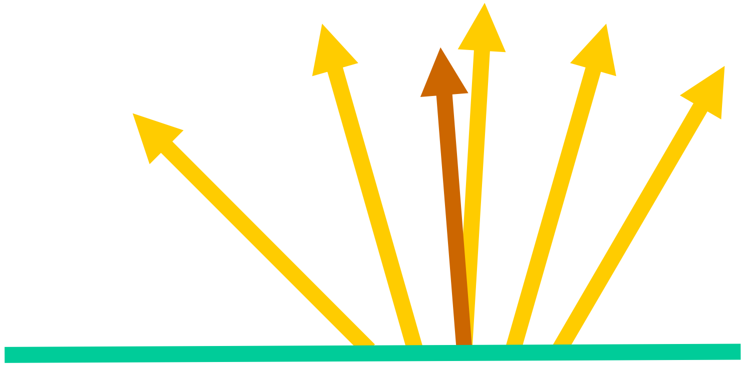
- Cons: slow; ugly patterns of 2x2 pixels due to ddx, ddy; high resolution needed

Antialiasing (1)

- Want to average the resulting pixel colors
- ... but **not** the normals!
- One could filter the height field by MIP mapping $(\partial h/\partial u, \partial h/\partial v, h)$.
- But that's not the resulting color either.

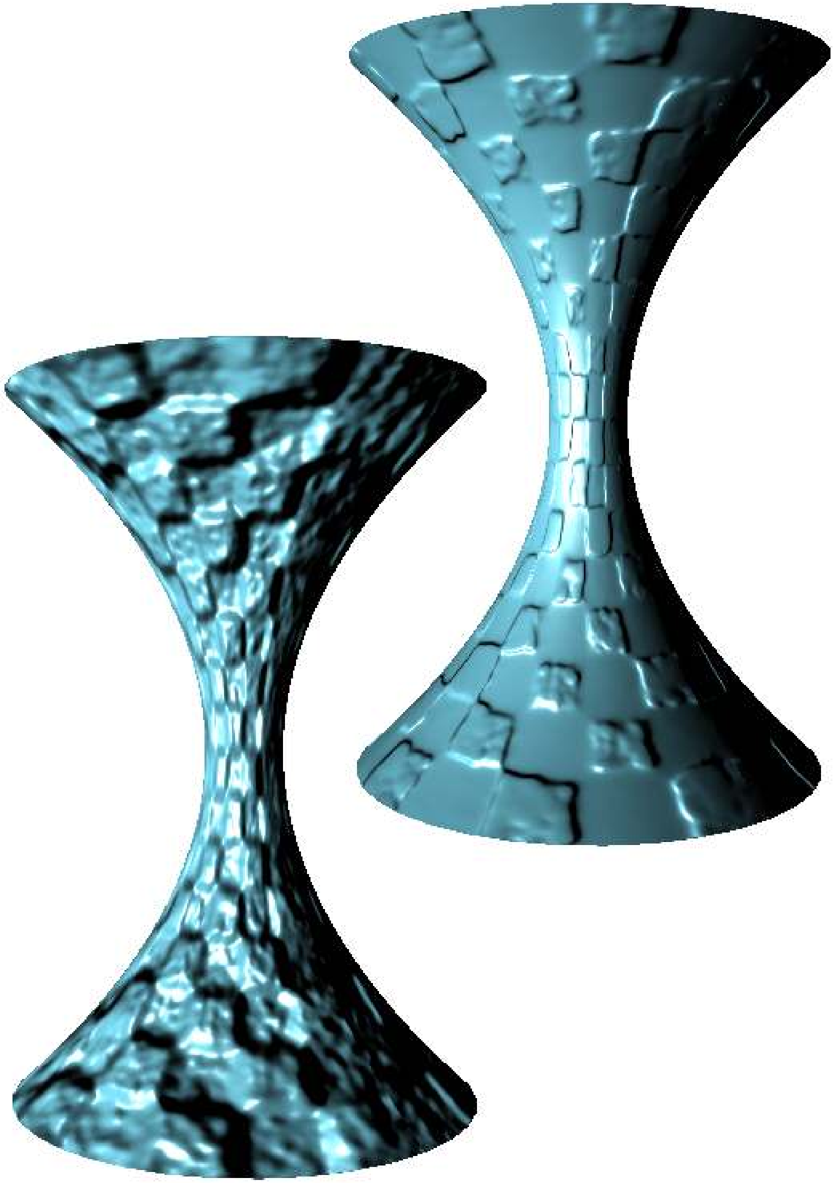
Antialiasing (2)

- Measure length loss to find divergence of normals (Toksvig 2005)
- Compute the distribution of normals (Han et al. 2007)



Conclusion

- To modify normals, keep the height field in mind.
- Virtually all image editing operations can be carried over.



Questions?