Intermediate Graphics & Animation Programming

GPR-300
Daniel S. Buckstein

Normal Mapping & Parallax Occlusion Mapping
Advanced Topics: Modern Techniques

License

Normal Mapping

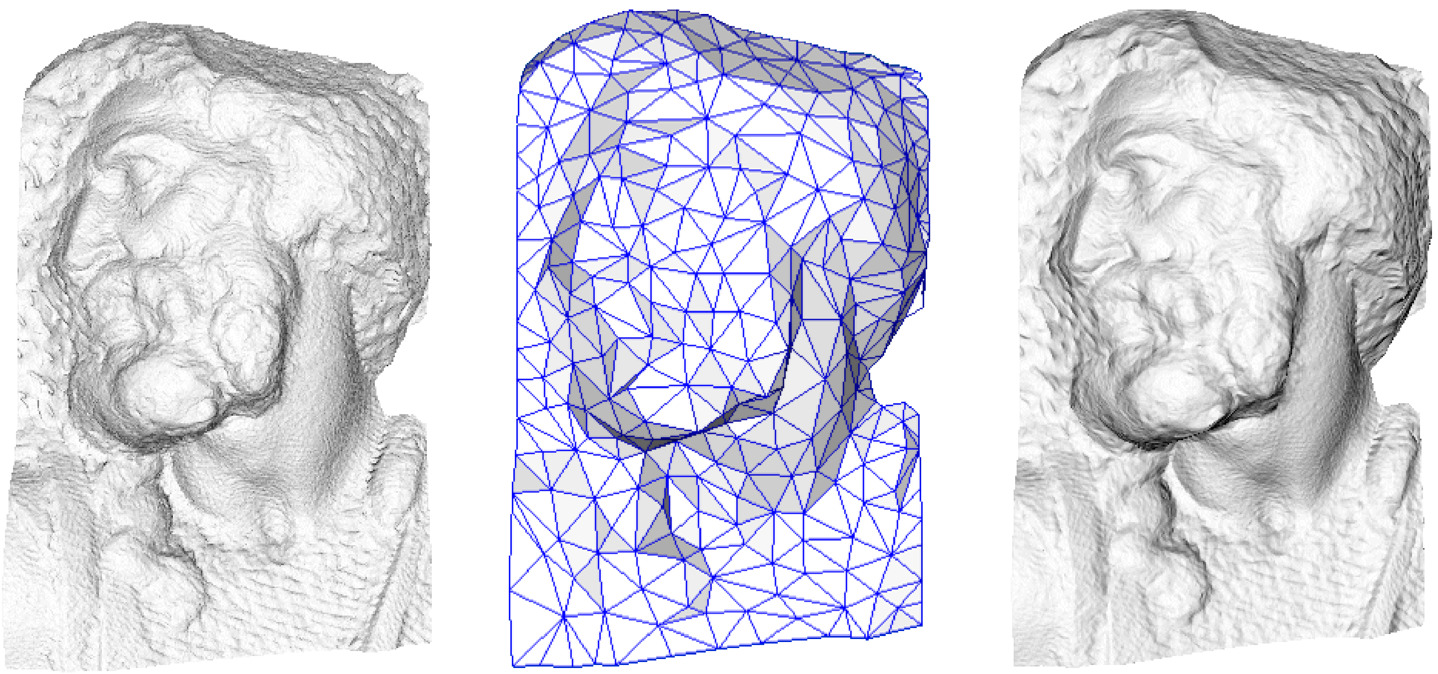
- Normal mapping
  - Background, algorithm
  - Tangent-space normal mapping
    - Calculating tangent basis
  - Which type of normal map to use
- Parallax occlusion mapping (POM)
  - Intermediate relief mapping method
    - Intro ray-tracing

Normal Mapping

- How many triangles?

Normal Mapping

- Normal and bump mapping:
- The illusion of high-resolution using textures



original mesh
4M triangles

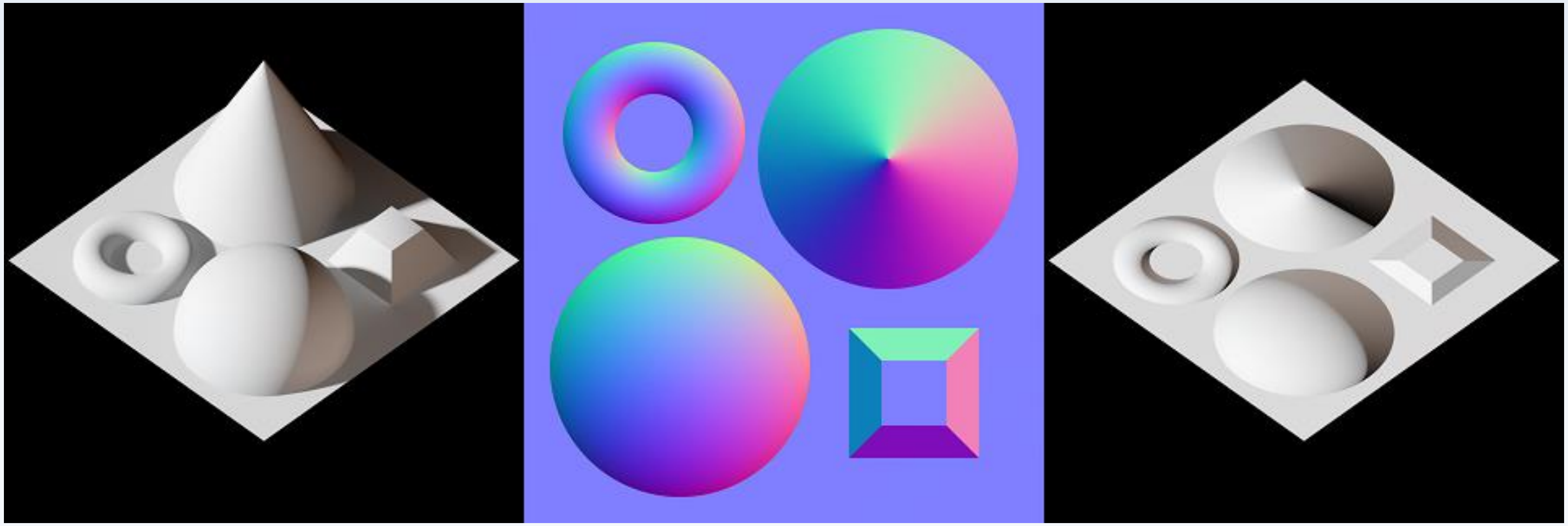
simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

From the Wikipedia page

Normal Mapping

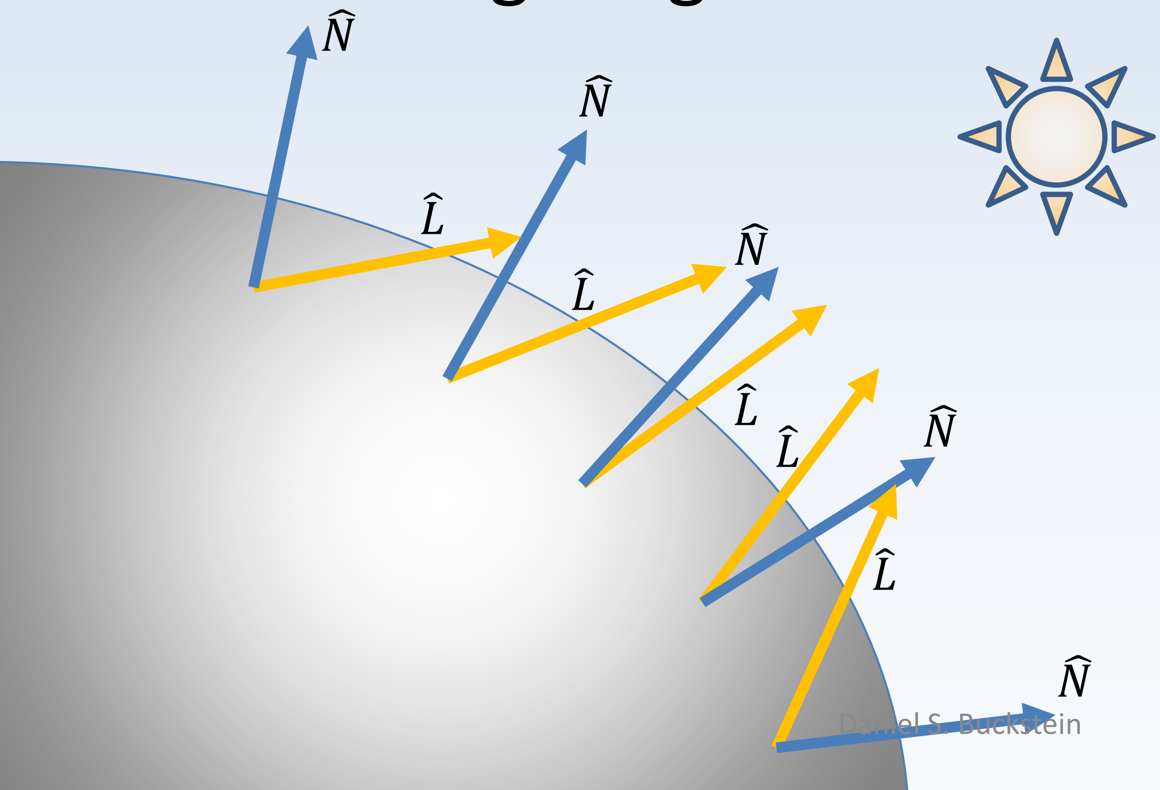
- Normal and bump mapping:
- Left: geometry used to produce normal map
- Middle: the normal map
- Right: Shading on plane using normal map



From the Wikipedia page

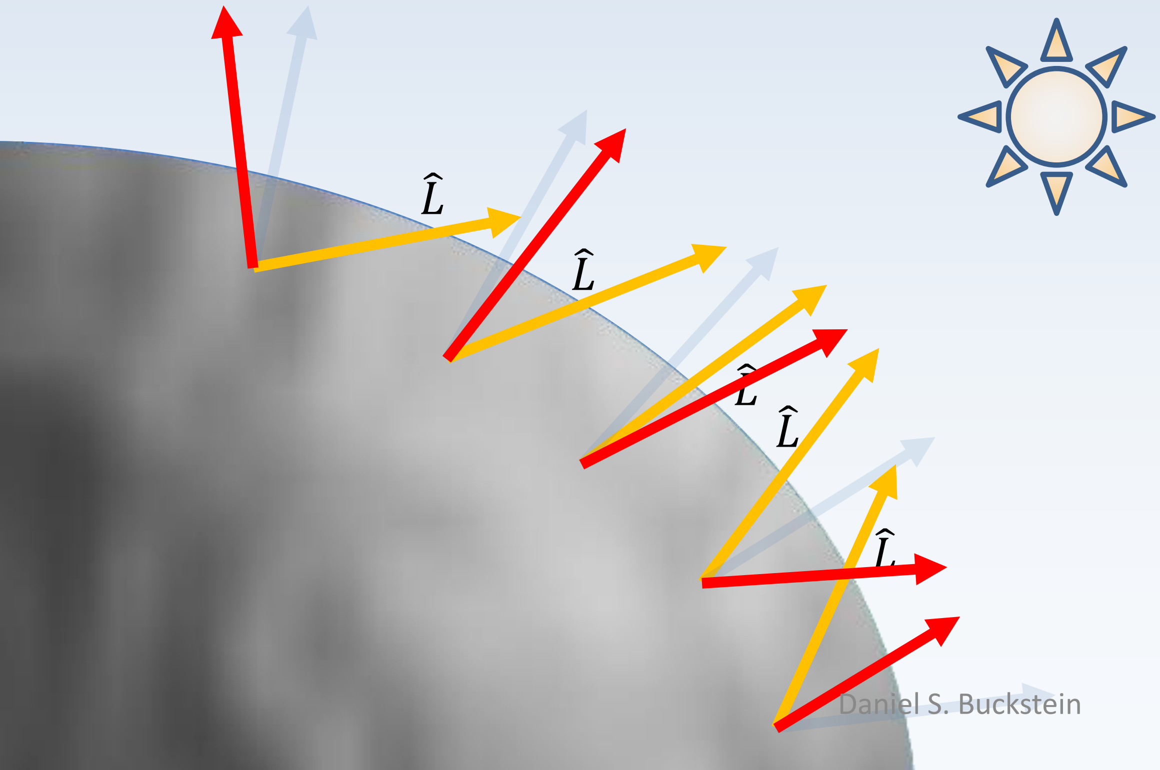
Normal Mapping

- Geometric normals interpolate across surface
    - "*Geometric normal*": the normal attribute ;)
- Diffuse lighting term = ???

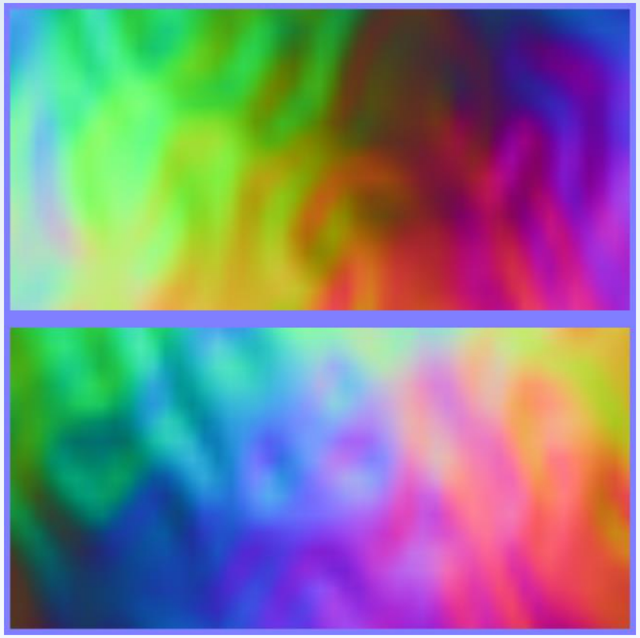


Normal Mapping

- Rugged surface means ***we cannot use linear-interpolated normals!!!***
- ***Enter <u>normal map</u>***



$\hat{N}$ is encoded as a texture!!!
→ Object-space normal map:



Normal Mapping

- ***Object-space normal mapping***:
- In lieu of geometric normal (which is in object-space), sample from texture…
- …then perform lighting as per usual!

- Shader example…

Normal Mapping

- ***Object-space normal mapping***:

```
// Obj. Space NM. VS (120)
attribute vec4 position;
attribute vec2 texcoord;
~~attribute vec3 normal;~~
// geometric normal not required!!!
// proceed with the rest of VS...
```

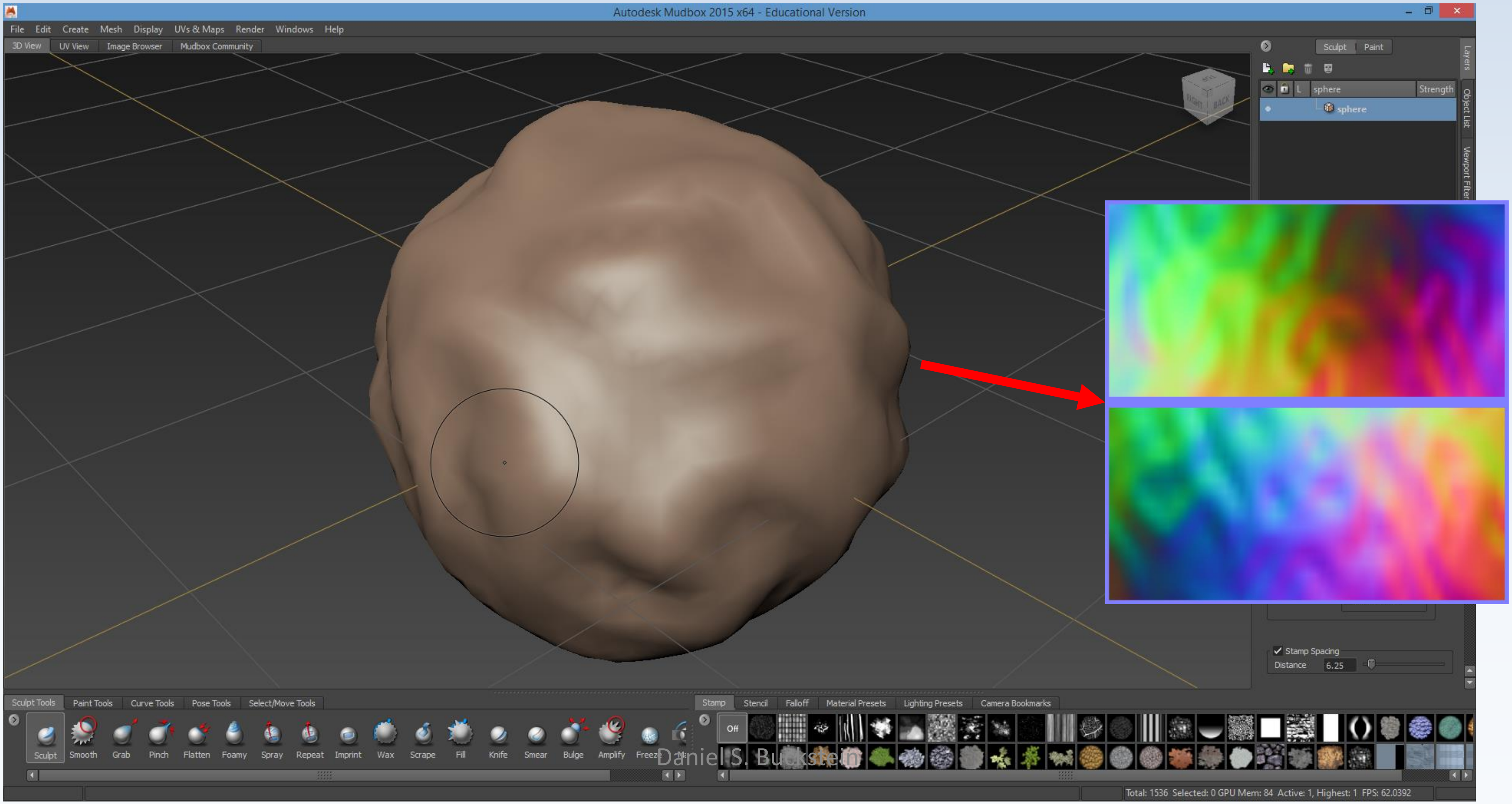
Normal Mapping

- ***Object-space normal mapping***:

```
// Obj. Space NM. FS (120)
varying vec3 normal_fromVS; // normal is not from attrib!
varying vec2 tc;            // texture coordinate from VS
uniform sampler2D normalMap; // the normal map!!!
... // (other variables...)
void main() {
    vec3 N = normalize (normal_fromVS);
    vec3 N = DESERIALIZE (texture2D (normalMap, tc).rgb);
... // (etc... Finally, do lighting calculations!!!)
    gl_FragColor = MY_SHADING_ALGO (N, L...);
```

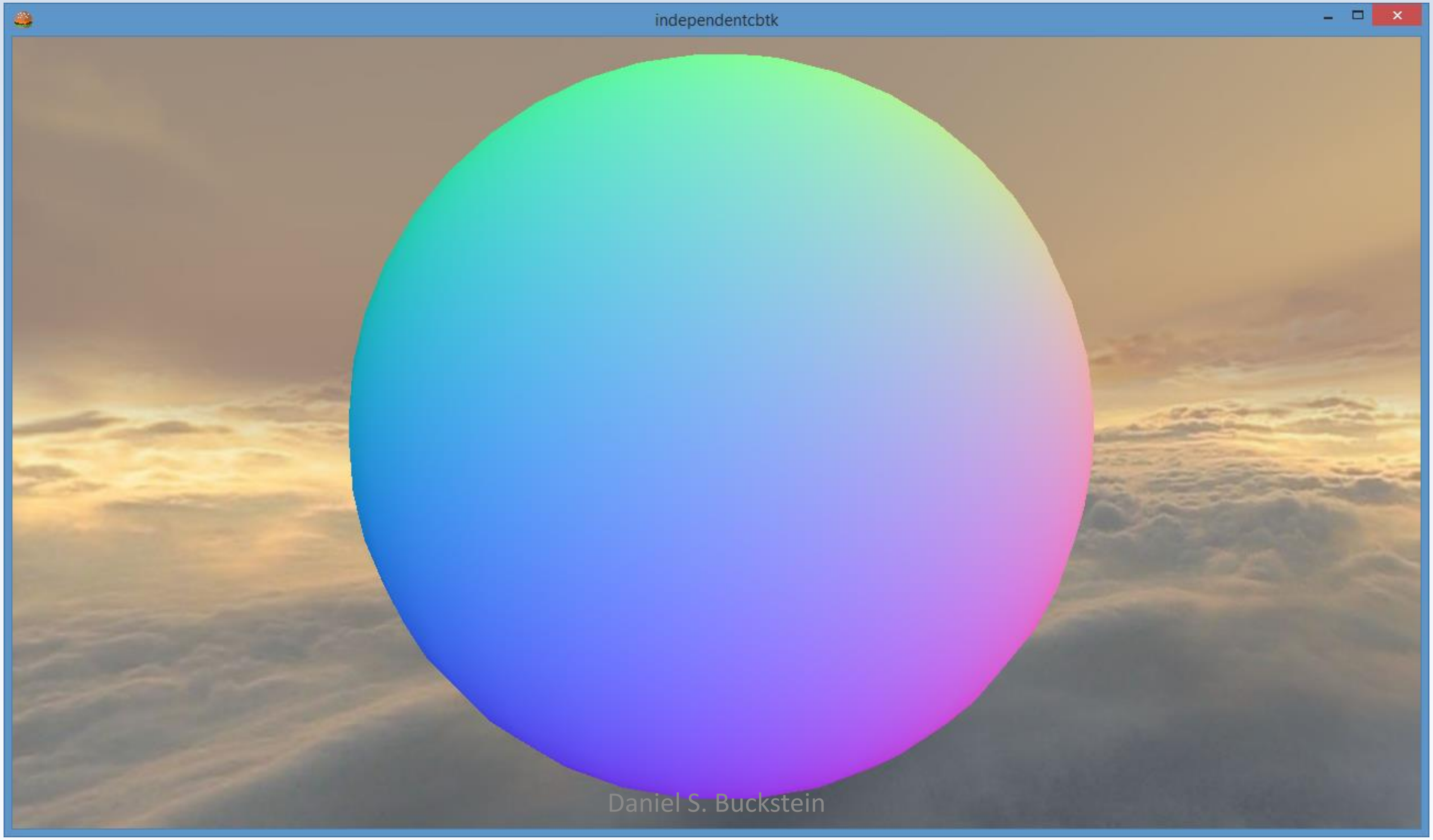
Normal Mapping

- Acquire normal map through sculpting:



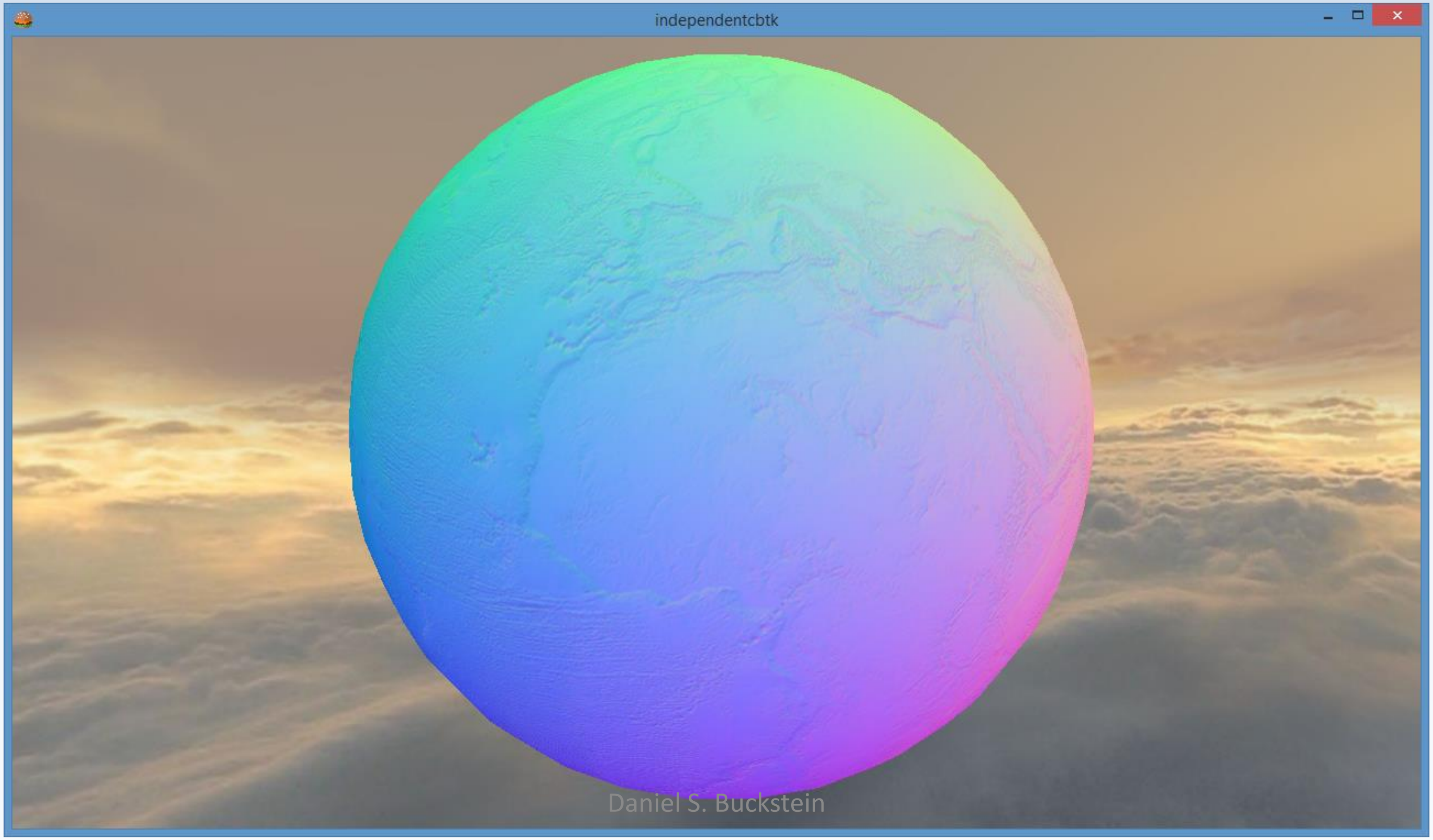
Normal Mapping

- Geometric normals (from attribute, obj. sp.):



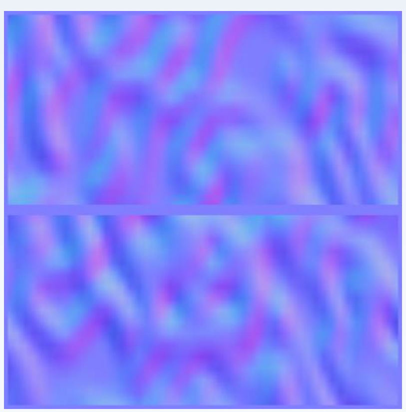
Normal Mapping

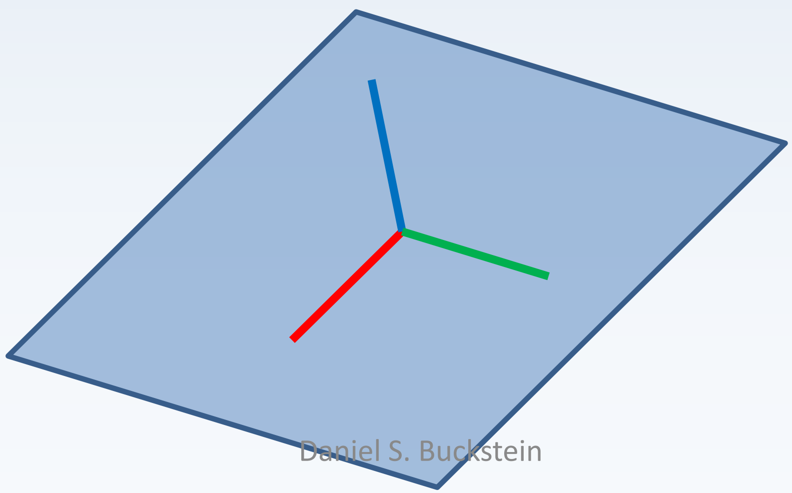
- Object space normal-map applied as color:

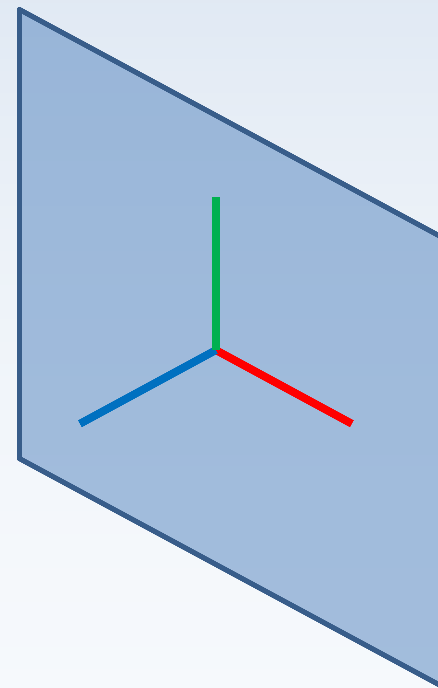


Normal Mapping

- "***Tangent space***": map is relative to ***surface***
- …it's a fancy way of saying "*texture space*"
- Makes sense because the texture is *tangential* to the surface!!!

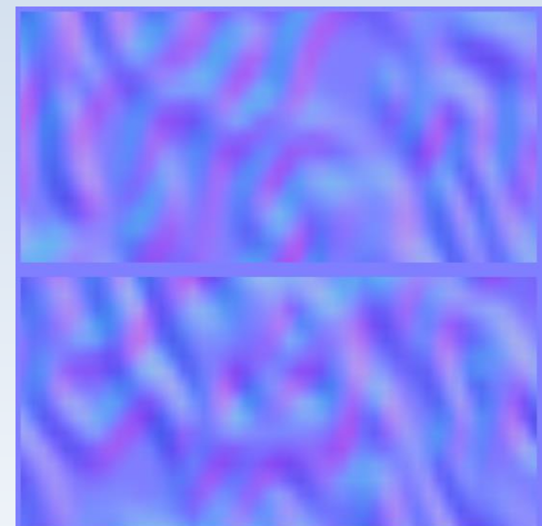






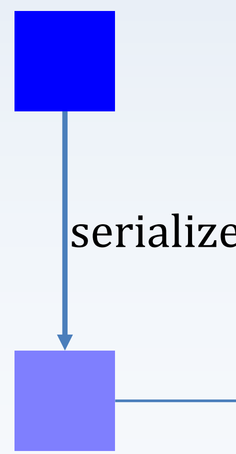
Normal Mapping

- So why do tangent space normal maps have that ubiquitous blue-purple color?
- Undisturbed normal represented as Z-axis (relative to surface):



$$\mathrm{range}_{\mathrm{vector}} = [-1, 1] \xrightarrow{\text{serialize}} \mathrm{range}_{\mathrm{color}} = [0, 1]$$

$$\hat{N}_{\mathrm{default}} = (0, 0, 1) \xrightarrow{\text{serialize}} \hat{N}'_{\mathrm{default}} = (0.5, 0.5, 1)$$



Normal Mapping

- Computing tangent basis for each vertex!!!
- What does a ***tangent*** represent in calculus???

- Exact same principle here… with one diff…

$$T = \frac{dp}{du} \quad , \qquad B = \frac{dp}{dv}$$

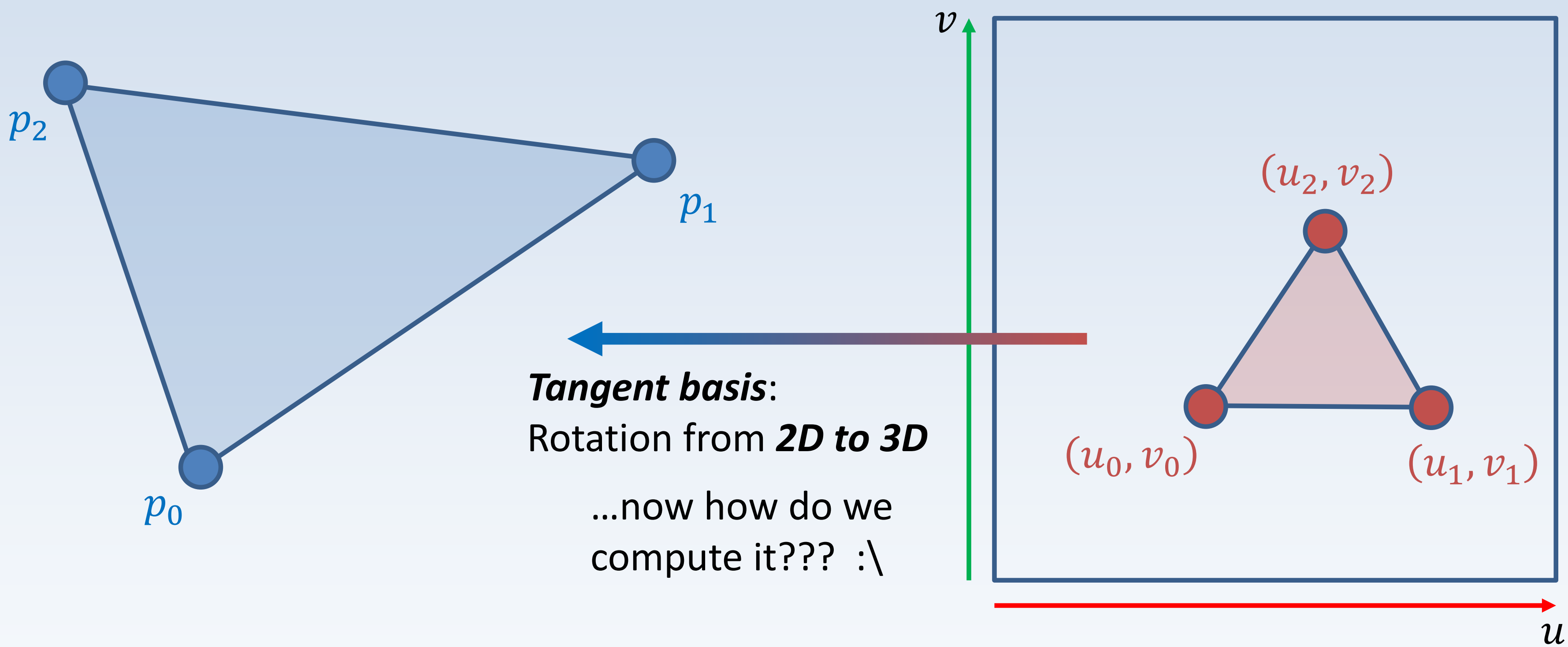
Continuous: the derivative of *3D position* with respect to *2D UV texture coordinate*

$$T = \frac{\Delta p}{\Delta u} \quad , \qquad B = \frac{\Delta p}{\Delta v}$$

Discrete: (in plain English) how does the *3D position* vary as we move across the *2D map*? (2D map of surface is texture space!)

Normal Mapping

- ***Tangent basis***: mapping from 2D to 3D

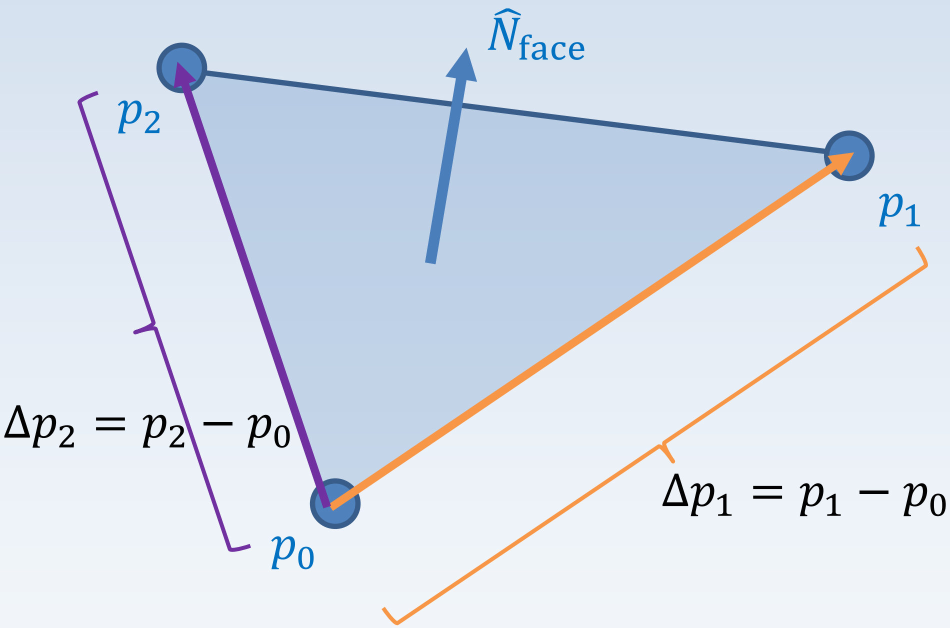


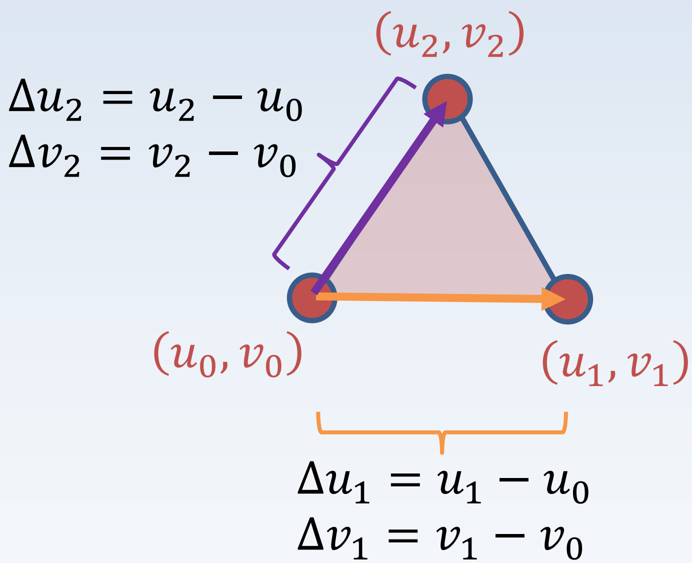
3D PHYSICAL SPACE: OBJECT
(spatial geometry lives here)

2D TEXTURE SPACE: TANGENT
(texture coordinates live here)

Normal Mapping

- Discrete deltas: differences between points!!!





$$\hat{\boldsymbol{N}}_{\textbf{face}} = \textbf{normalize}(\boldsymbol{\Delta p_1} \times \boldsymbol{\Delta p_2})$$

Normal Mapping

- Now we have the values from the formula:

$$T = \frac{\Delta p}{\Delta u} \quad , \qquad B = \frac{\Delta p}{\Delta v}$$

- We can solve *T* and *B* if we represent this as a system of equations (**integrate *T* and *B* into *p***):

$$\Delta p_1 = T\Delta u_1 + B\Delta v_1$$

$$\Delta p_2 = T\Delta u_2 + B\Delta v_2$$

Normal Mapping

- System of equations expressed with ***matrices***:

$$\Delta p_1 = T\Delta u_1 + B\Delta v_1$$

$$\Delta p_2 = T\Delta u_2 + B\Delta v_2$$

$$\begin{bmatrix} \Delta p_1 & \Delta p_2 \end{bmatrix} = \begin{bmatrix} T & B \end{bmatrix} \begin{bmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{bmatrix}$$

$$\begin{bmatrix} \Delta p_{1_x} & \Delta p_{2_x} \\ \Delta p_{1_y} & \Delta p_{2_y} \\ \Delta p_{1_z} & \Delta p_{2_z} \end{bmatrix} = \begin{bmatrix} T_x & B_x \\ T_y & B_y \\ T_z & B_z \end{bmatrix} \begin{bmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{bmatrix}$$

Normal Mapping

- But wait… we already have position…
- How do we isolate *T* and *B*???

$$\begin{bmatrix} \Delta p_{1_x} & \Delta p_{2_x} \\ \Delta p_{1_y} & \Delta p_{2_y} \\ \Delta p_{1_z} & \Delta p_{2_z} \end{bmatrix} \begin{bmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{bmatrix}^{-1} = \begin{bmatrix} T_x & B_x \\ T_y & B_y \\ T_z & B_z \end{bmatrix} \begin{bmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{bmatrix} \begin{bmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{bmatrix}^{-1}$$

- Interesting how things work out…

$$T = \frac{\Delta p}{\Delta u} \quad \rightarrow \quad \Delta p (\Delta u)^{-1}$$

#math #mathpower #hypeplane

Normal Mapping

- Solve the matrix product and you're golden!

$$\begin{bmatrix} T_x & B_x \\ T_y & B_y \\ T_z & B_z \end{bmatrix} = \begin{bmatrix} \Delta p_{1_x} & \Delta p_{2_x} \\ \Delta p_{1_y} & \Delta p_{2_y} \\ \Delta p_{1_z} & \Delta p_{2_z} \end{bmatrix} \begin{bmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{bmatrix}^{-1}$$

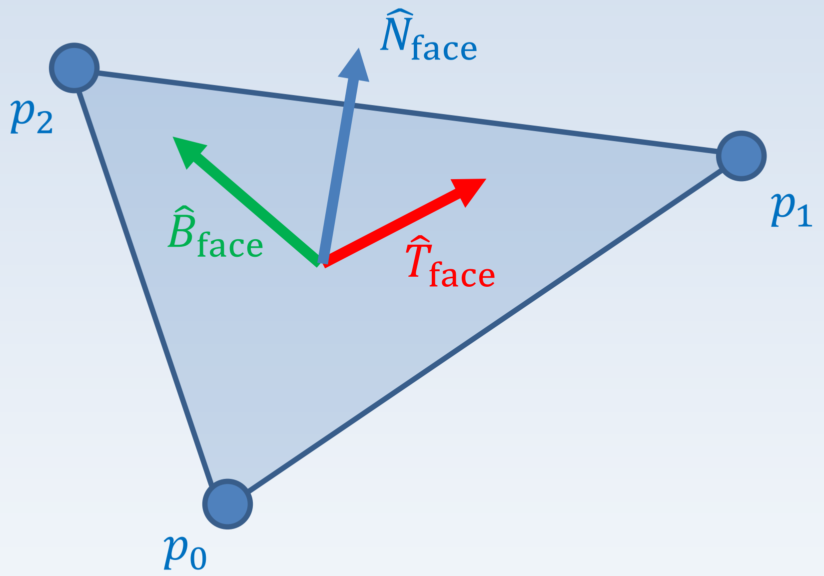
$$[T \quad B] = [\Delta p_1 \quad \Delta p_2] \underbrace{\begin{bmatrix} \Delta u_1 & \Delta u_2 \\ \Delta v_1 & \Delta v_2 \end{bmatrix}^{-1}}$$

2D matrix inverse!

$$[T \quad B] = [\Delta p_1 \quad \Delta p_2] \overbrace{\begin{bmatrix} \Delta v_2 & -\Delta u_2 \\ -\Delta v_1 & \Delta u_1 \end{bmatrix} \left(\frac{1}{\Delta u_1 \Delta v_2 - \Delta u_2 \Delta v_1} \right)}$$

Normal Mapping

- We now have the tangent basis for the ***face***



$$\hat{\boldsymbol{T}}_{\textbf{face}} = \textbf{normalize}(\boldsymbol{T})$$

$$\hat{\boldsymbol{B}}_{\textbf{face}} = \textbf{normalize}(\boldsymbol{B})$$

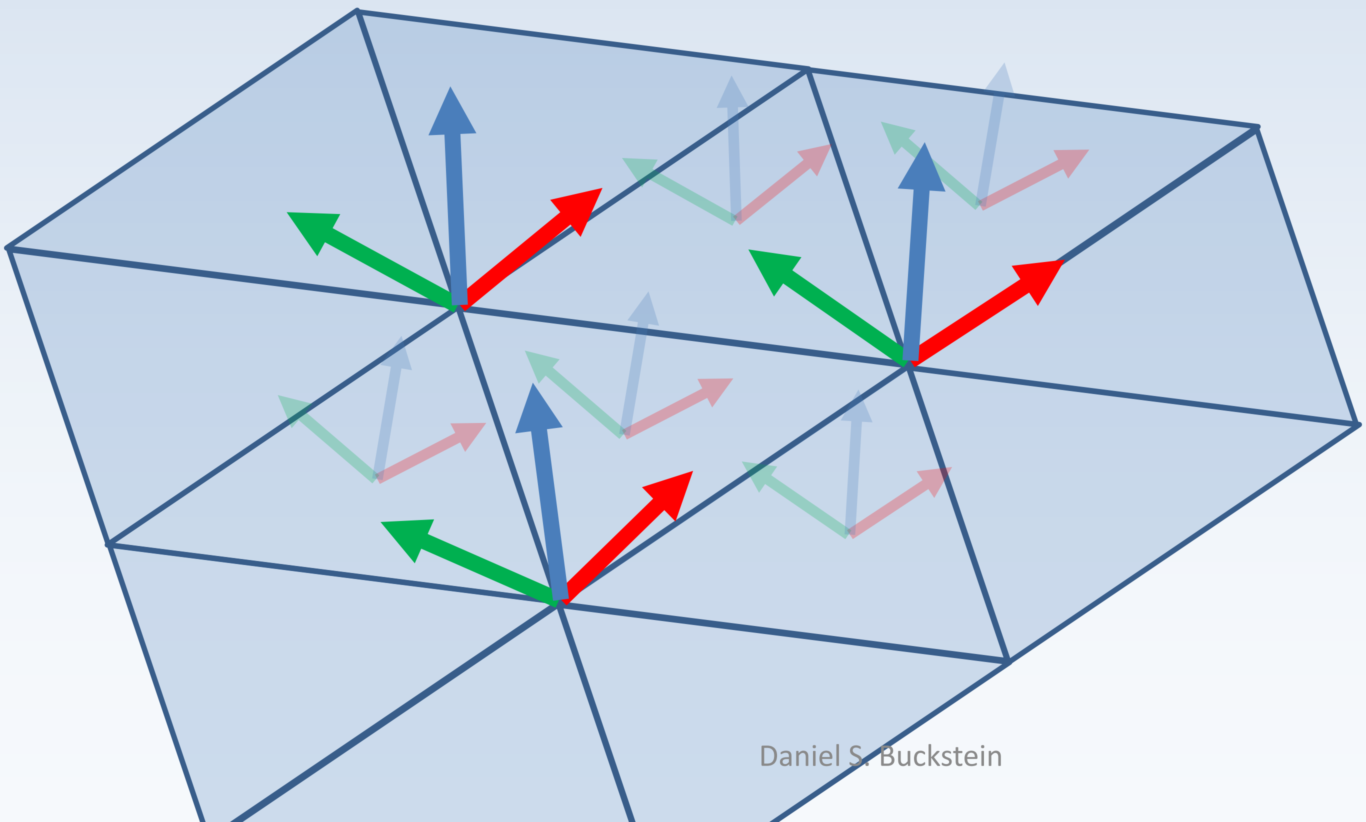
$$\hat{N}_{\text{face}} = \text{normalize}(\Delta p_1 \times \Delta p_2)$$

- Store these in a special rotation matrix:

$$\boxed{M_{TBN} = [\hat{T} \quad \hat{B} \quad \hat{N}]}$$

Normal Mapping

- You’re almost ready to go!
- How to get ***smooth vertex tangent bases***???



Normal Mapping

- ***Tangent-space normal mapping***:

```
// Tan. Space NM. VS (120)
attribute vec4 position;
attribute vec2 texcoord;
attribute vec3 tangent;
attribute vec3 bitangent;
attribute vec3 normal;
// use TBN matrix or 3 attributes:
//  *geometric* tangent, bitangent, and normal
```

Normal Mapping

- ***Tangent-space normal mapping***:

```
// Tan. Space NM. FS (120)
varying vec3 tangent_fromVS;
varying vec3 bitangent_fromVS;
varying vec3 normal_fromVS;

varying vec2 tc;
uniform sampler2D normalMap;
... // (other variables...)

// cont'd next slide
```

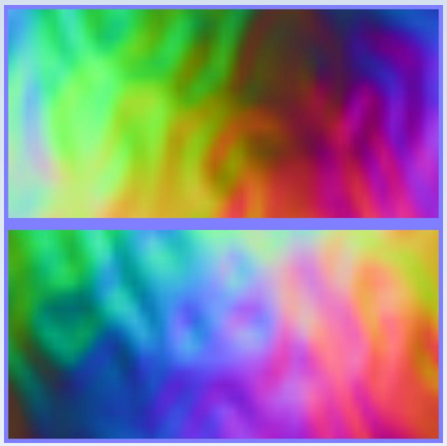
Normal Mapping

- ***Tangent-space normal mapping***:

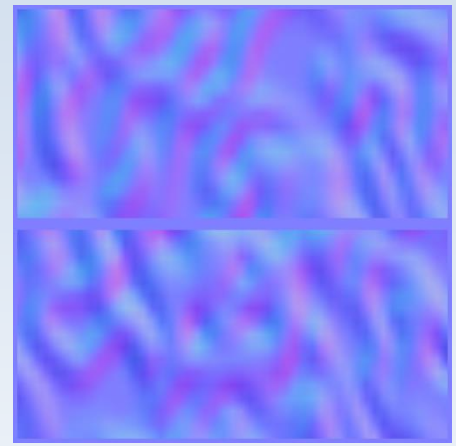
```
// Tan. Space NM. FS (120) (cont'd)
void main() {
    mat3 TBN = mat3 (normalize (tangent_fromVS),
                     normalize (bitangent_fromVS),
                     normalize (normal_fromVS));

    // sample normal map: TANGENT SPACE
    vec3 N_ts = DESERIALIZE (texture2D (normalMap, tc).rgb);
    // TBN matrix brings us from TAN to OBJ space!!!
    vec3 N_os = TBN * N_ts;
        // ...finally, proceed with shading!!!
```

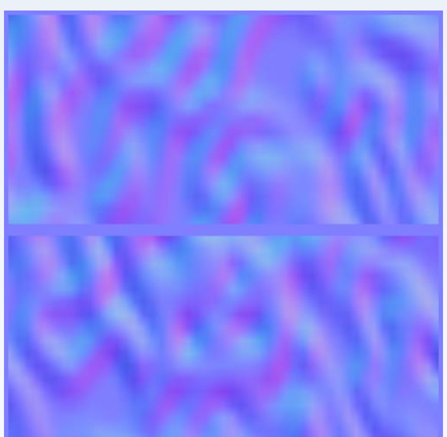
Normal Mapping

- Conversion between spaces:

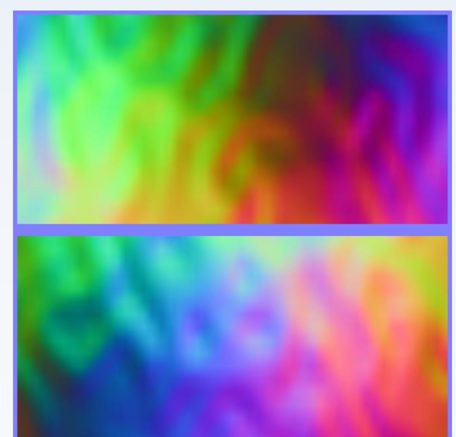


$$= M_{TBN} \times$$





$$= M_{TBN}^{-1} \times$$



Normal Mapping

- ***Pro tip 4 u:***
- Vertex tangent bases may not be ***orthonormal***
- Not orthonormal → not a rotation matrix ☹

- *Pro tip*: use the ***Gram-Schmidt process*** to *orthonormalize* your vectors!
  https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process
  – (use the ***normal*** as the fixed vector)

Normal Mapping

- ***Pro tip 4 u:***
- Which type of normal map should you use???

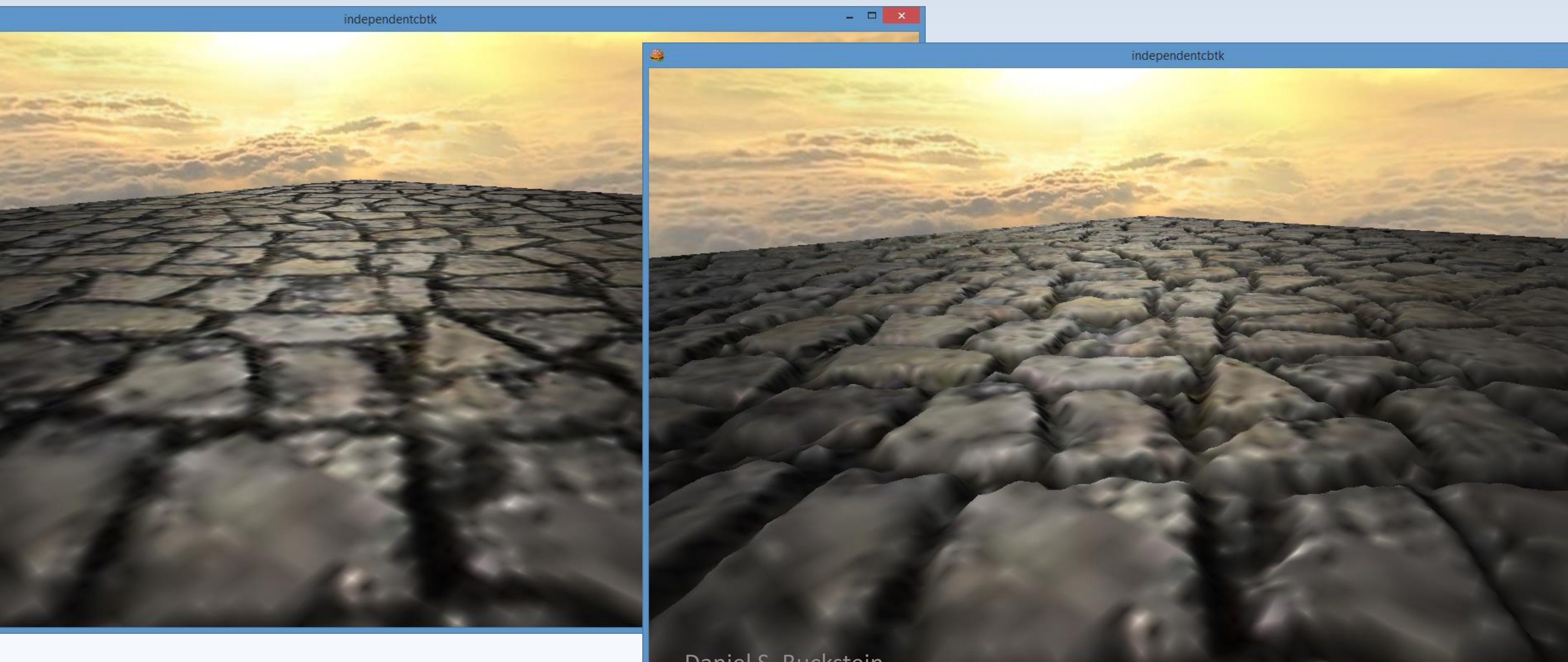
Is it safe to use a normal map in… →

…for an object that… →

	…tangent space…	…object space…	…world space…
…deforms?	**YES**	**NO**	**NO**
…rotates?	**YES**	**YES**	**NO**
…translates?	**YES**	**YES**	**YES**

Parallax Occlusion Mapping

- Normal mapping is nice… but it gets better ☺



Parallax Occlusion Mapping

- Normal mapping is just a ***shading technique***
- The illusion of detail due to ***light's*** interaction with surface
- Enter "***parallax occlusion mapping***" (POM)
- The illusion of ***physical detail***
- Based on the ***viewer's*** interaction with surface

Parallax Occlusion Mapping

- Overview of algorithm:
- ***Bump map*** / ***height map*** / ***displacement map*** describes depth into surface

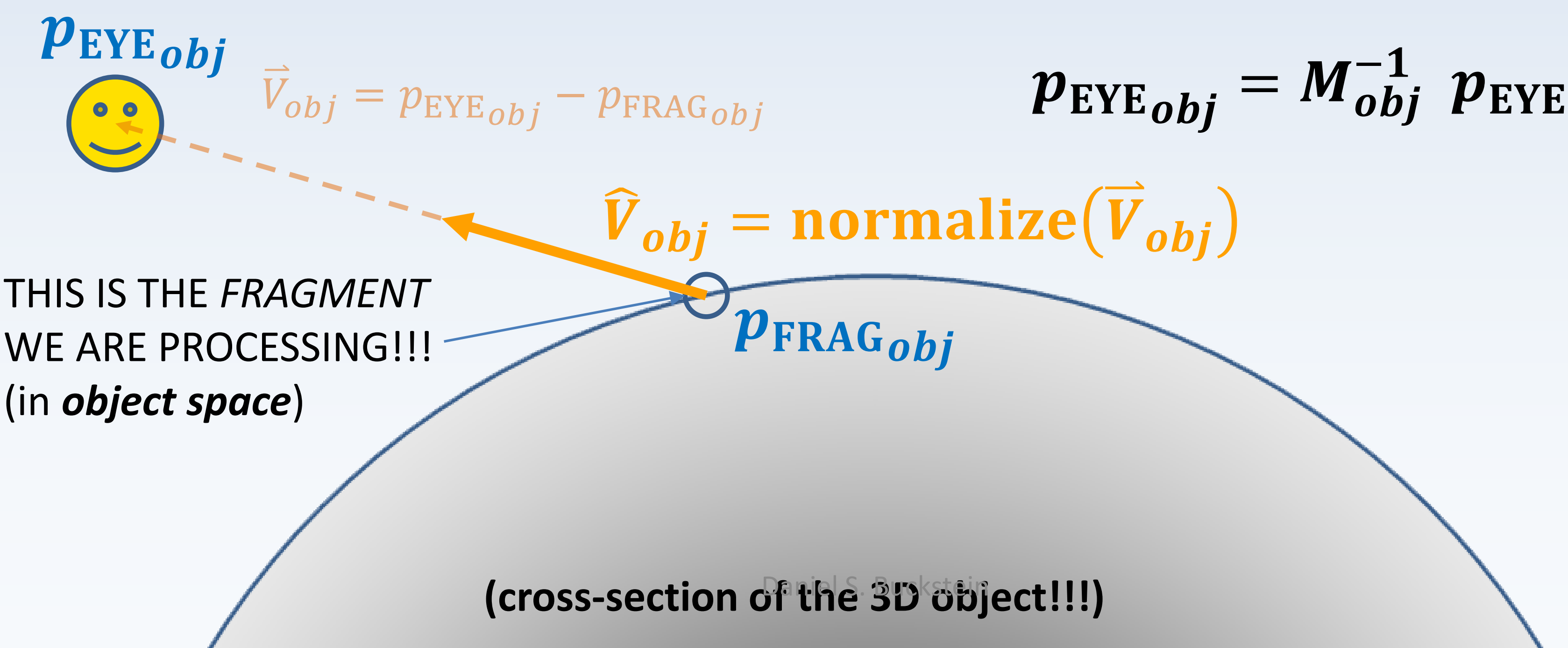
1) Determine viewer ray (like with Phong model)
2) Trace ray into surface, through bump map
3) Determine where ray intersects bump map
4) ***The goal***: compute *offset texture coordinate*

Parallax Occlusion Mapping

1) Calculate view ray: direction from surface to the eye (just like with Phong lighting!)

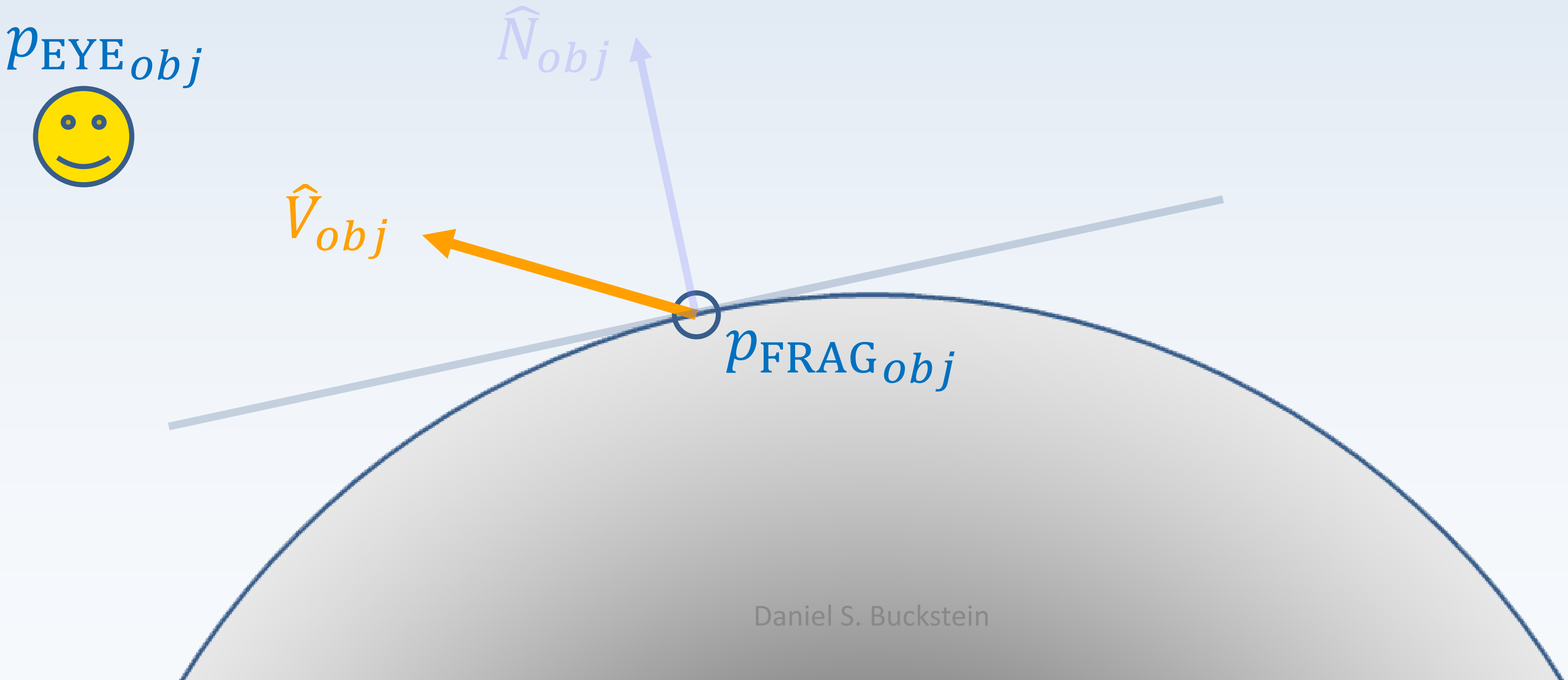
Convert viewer's position in *world space* to ***object space***:

$$\boldsymbol{p}_{\mathbf{EYE}_{obj}} = \boldsymbol{M}_{obj}^{-1}\ \boldsymbol{p}_{\mathbf{EYE}_{world}}$$



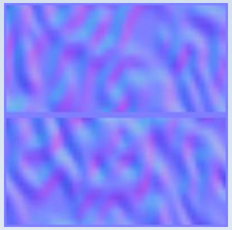
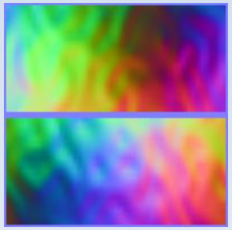
Parallax Occlusion Mapping

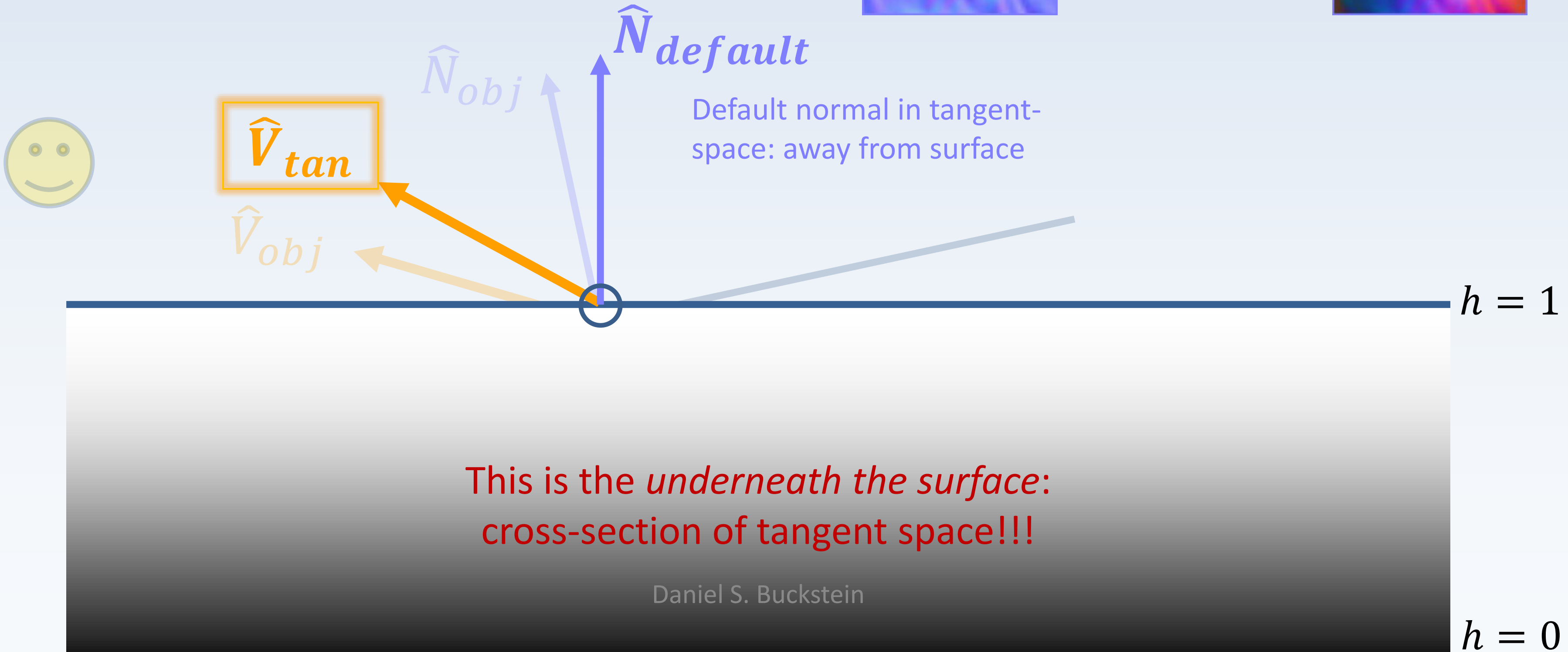
- We are mapping occlusion based on the "***shape of the surface***"
- Remember ***<u>tangent-space</u>***: ***relative to surface***



Parallax Occlusion Mapping

- Move into tangent space for the rest of the algorithm!!!

 $= M_{TBN}^{-1} \times$ 



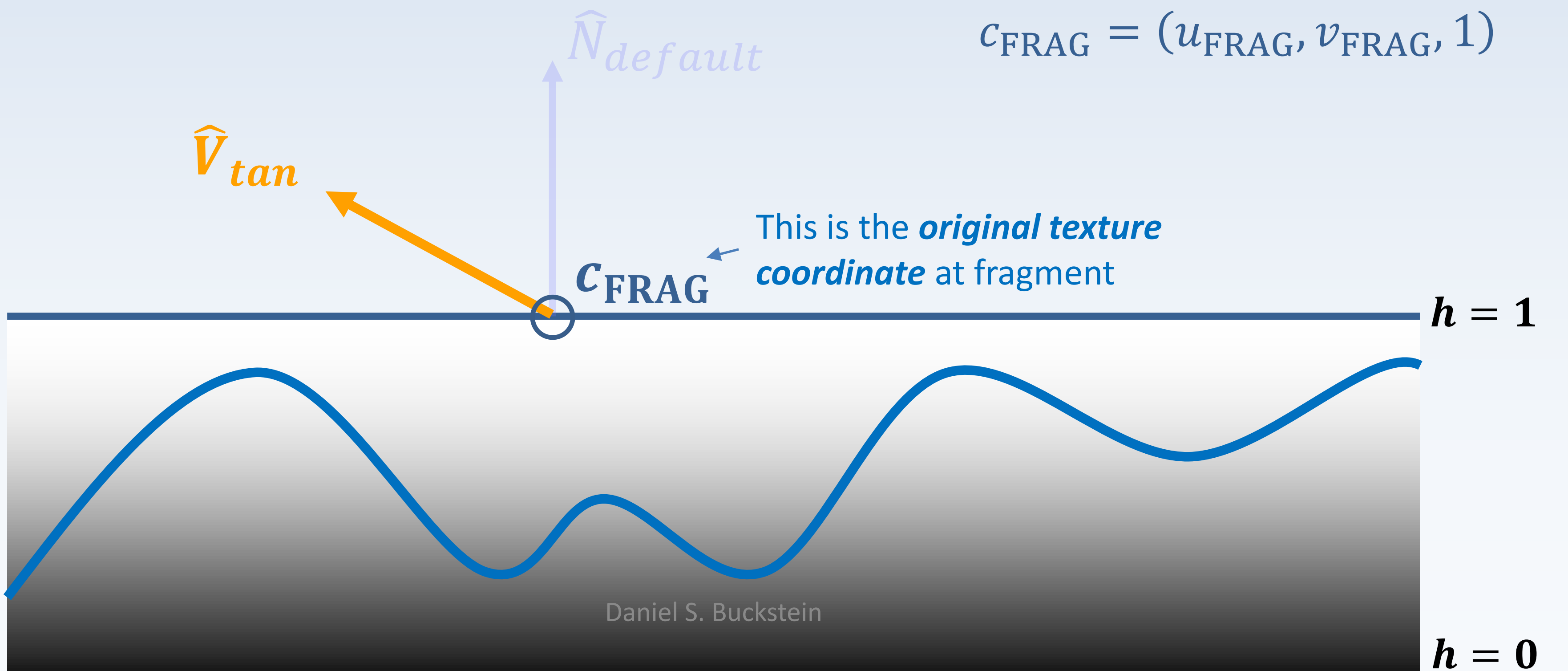
Parallax Occlusion Mapping

2) Trace ray into surface, through ***bump map***

- Greyscale ***height map*** used to determine actual shape of surface!!!

$c = (u, v, h)$

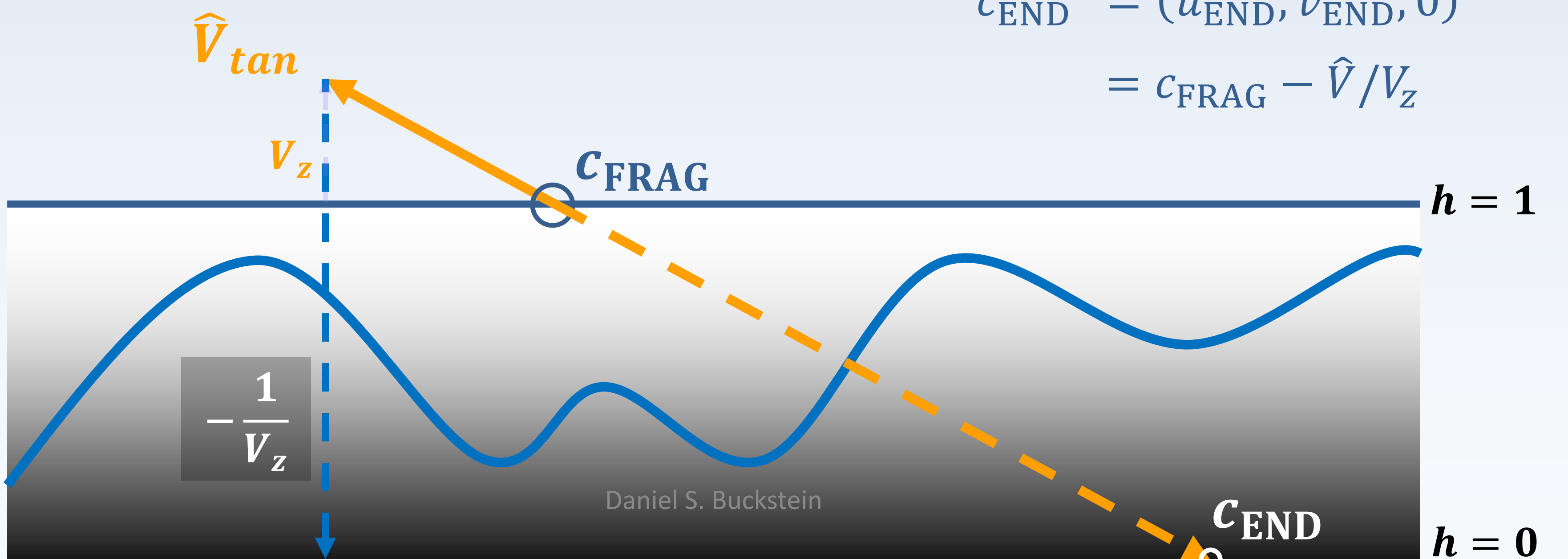
$c_{\text{FRAG}} = (u_{\text{FRAG}}, v_{\text{FRAG}}, 1)$



Parallax Occlusion Mapping

- We want to see where the view ray intersects the ***bumpy surface***

$$
\begin{aligned}
c &= (u, v, h) \\
c_{\mathrm{FRAG}} &= (u_{\mathrm{FRAG}}, v_{\mathrm{FRAG}}, 1) \\
c_{\mathrm{END}} &= (u_{\mathrm{END}}, v_{\mathrm{END}}, 0) \\
&= c_{\mathrm{FRAG}} - \hat{V}/V_z
\end{aligned}
$$

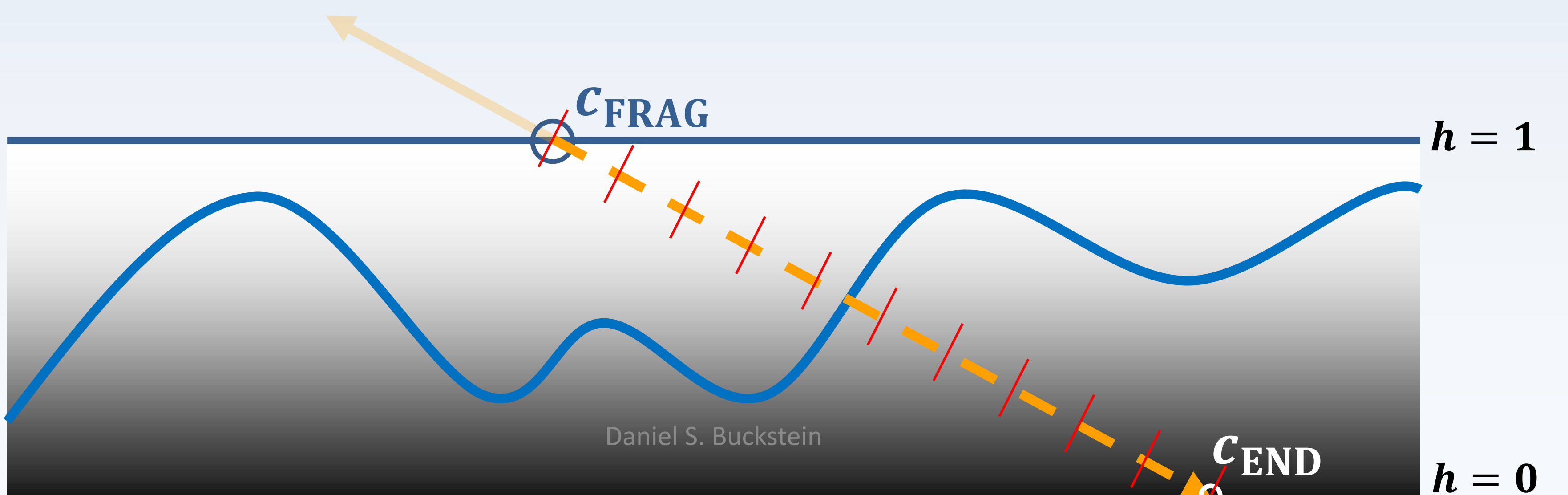


Parallax Occlusion Mapping

- ***How do we find the intersection???***
- ***<u>Ray-tracing</u>***: we have the ray's origin and farthest possible point…

$c = (u, v, h)$

$$\boxed{\begin{aligned} \boldsymbol{c_{FRAG}} &= (\boldsymbol{u}_{\mathbf{FRAG}}, \boldsymbol{v}_{\mathbf{FRAG}}, \mathbf{1}) \\ \boldsymbol{c}_{\mathbf{END}} &= (\boldsymbol{u}_{\mathbf{END}}, \boldsymbol{v}_{\mathbf{END}}, \mathbf{0}) \end{aligned}}$$



Parallax Occlusion Mapping

- Once again… ***linear interpolation for the win!***

$$n = \text{float(numSamples)}$$

$$dt = 1/n$$

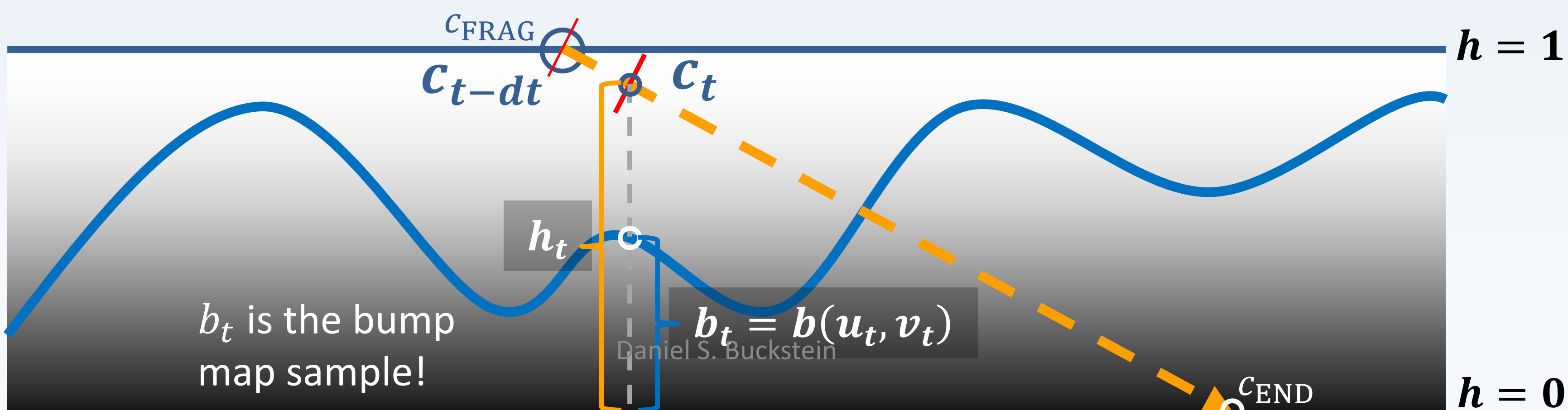
$$c_t = \text{lerp}(c_{\text{FRAG}}, c_{\text{END}}, t) = (u_t, v_t, h_t)$$

$$c = (u, v, h)$$

$$\boldsymbol{c_{FRAG}} = (\boldsymbol{u}_{\textbf{FRAG}}, \boldsymbol{v}_{\textbf{FRAG}}, \mathbf{1})$$

$$\boldsymbol{c}_{\textbf{END}} = (\boldsymbol{u}_{\textbf{END}}, \boldsymbol{v}_{\textbf{END}}, \mathbf{0})$$

- Iterate through samples…



Parallax Occlusion Mapping

- Checking if ***bump map height > ray height***...

$$n = \text{float}(\text{numSamples})$$

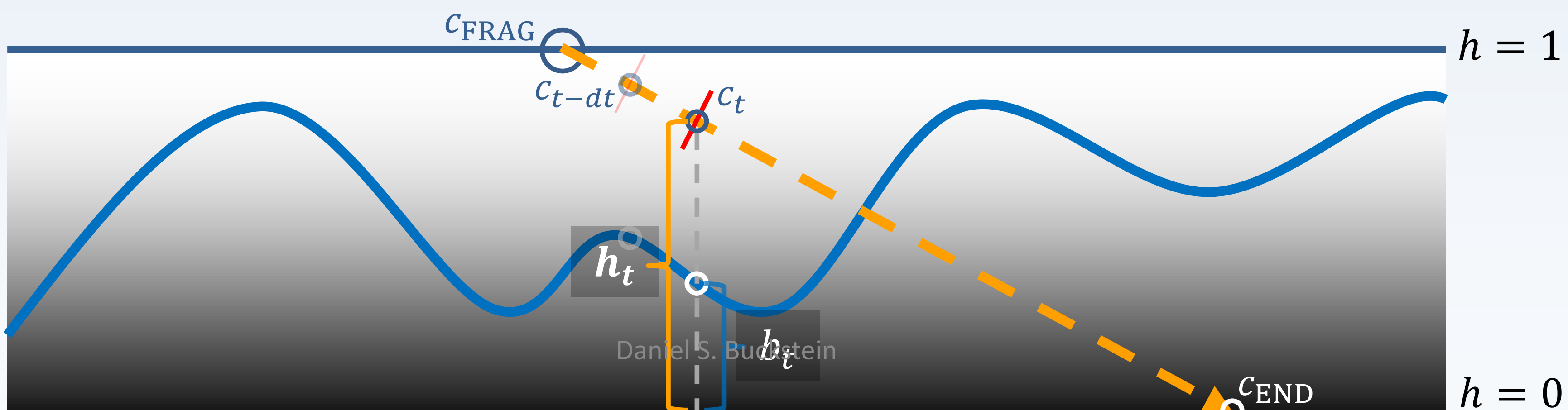
$$dt = 1/n$$

$$c_t = \text{lerp}(c_{\text{FRAG}}, c_{\text{END}}, t) = (u_t, v_t, h_t)$$

$$c = (u, v, h)$$

$$\boldsymbol{c_{FRAG}} = (\boldsymbol{u}_{\text{FRAG}}, \boldsymbol{v}_{\text{FRAG}}, \mathbf{1})$$

$$\boldsymbol{c}_{\text{END}} = (\boldsymbol{u}_{\text{END}}, \boldsymbol{v}_{\text{END}}, \mathbf{0})$$



Parallax Occlusion Mapping

- Checking if ***bump map height > ray height***…

$$n = \text{float(numSamples)}$$

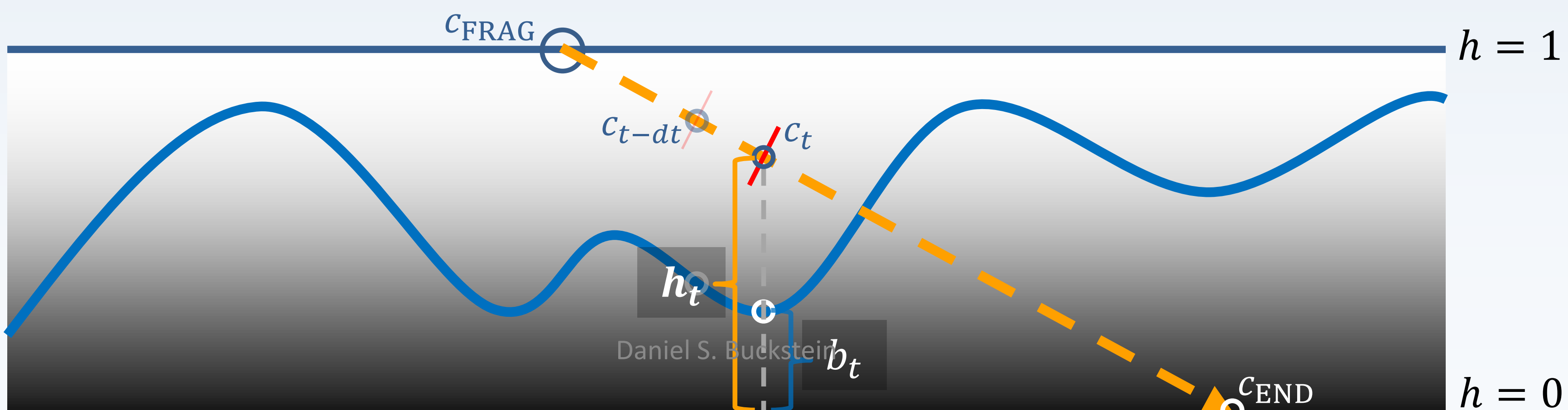
$$dt = 1/n$$

$$c_t = \text{lerp}(c_{\text{FRAG}}, c_{\text{END}}, t) = (u_t, v_t, h_t)$$

$$c = (u, v, h)$$

$$\boldsymbol{c_{FRAG}} = (\boldsymbol{u}_{\textbf{FRAG}}, \boldsymbol{v}_{\textbf{FRAG}}, \mathbf{1})$$

$$\boldsymbol{c}_{\textbf{END}} = (\boldsymbol{u}_{\textbf{END}}, \boldsymbol{v}_{\textbf{END}}, \mathbf{0})$$



Parallax Occlusion Mapping

- Checking if ***bump map height > ray height***…

$$n = \text{float}(\text{numSamples})$$

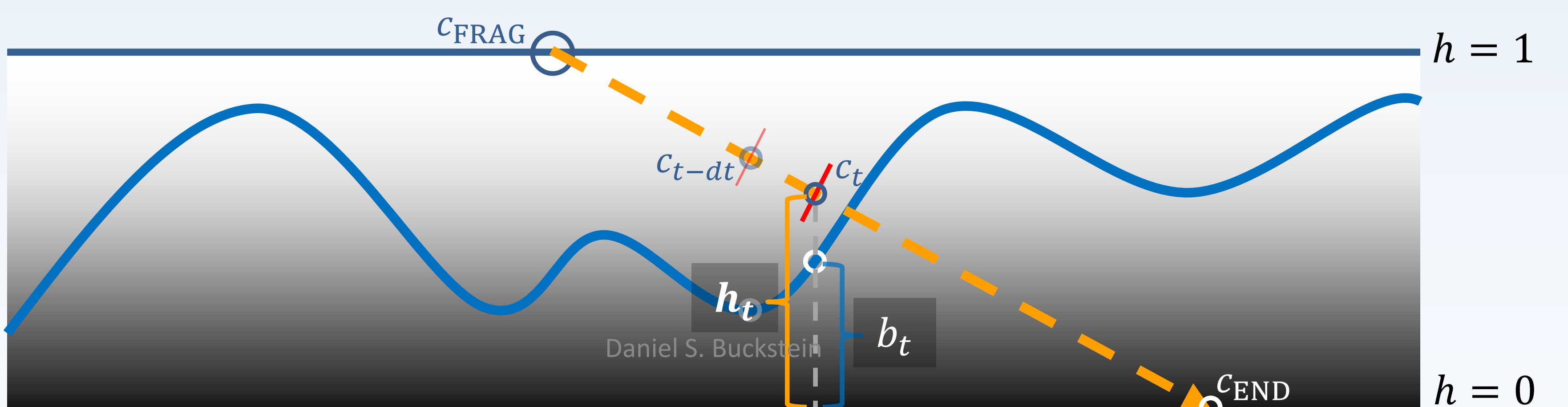
$$dt = 1/n$$

$$c_t = \text{lerp}(c_{\text{FRAG}}, c_{\text{END}}, t) = (u_t, v_t, h_t)$$

$$c = (u, v, h)$$

$$\boldsymbol{c_{FRAG}} = (\boldsymbol{u}_{\textbf{FRAG}}, \boldsymbol{v}_{\textbf{FRAG}}, \mathbf{1})$$

$$\boldsymbol{c}_{\textbf{END}} = (\boldsymbol{u}_{\textbf{END}}, \boldsymbol{v}_{\textbf{END}}, \mathbf{0})$$



Parallax Occlusion Mapping

- Checking if ***bump map height > ray height***…

$$n = \text{float}(\text{numSamples})$$

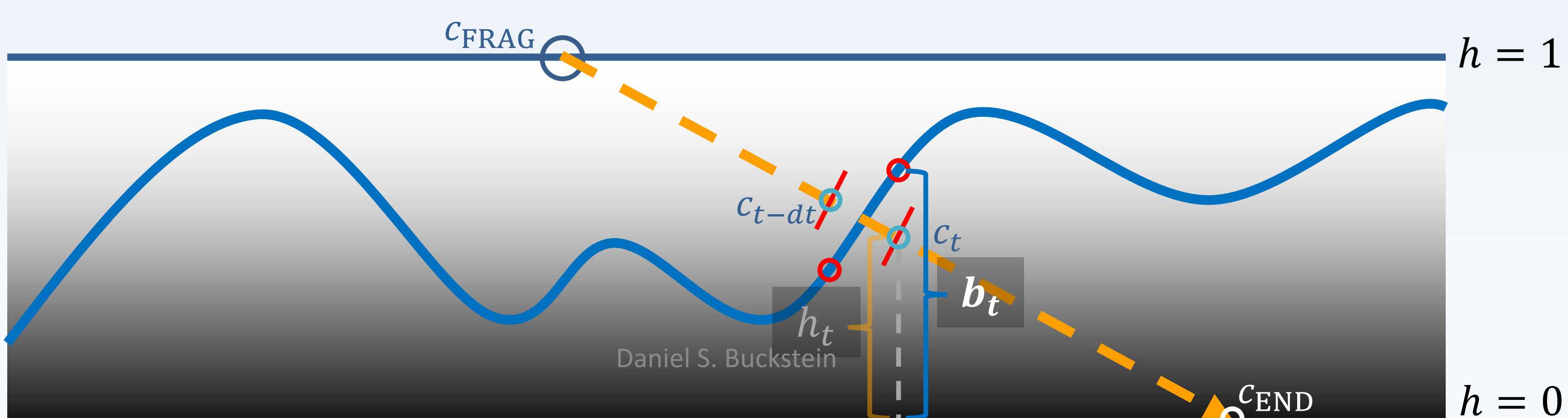
$$dt = 1/n$$

$$c_t = \text{lerp}(c_{\text{FRAG}}, c_{\text{END}}, t) = (u_t, v_t, h_t)$$

$$c = (u, v, h)$$

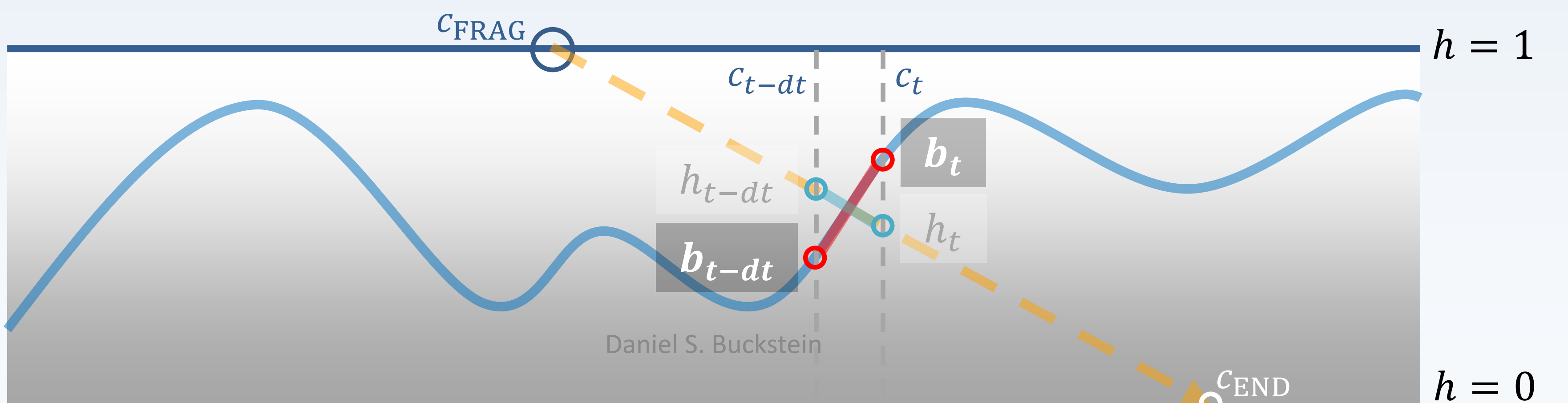
$$\boldsymbol{c_{FRAG}} = (\boldsymbol{u}_{\textbf{FRAG}}, \boldsymbol{v}_{\textbf{FRAG}}, \mathbf{1})$$

$$\boldsymbol{c}_{\textbf{END}} = (\boldsymbol{u}_{\textbf{END}}, \boldsymbol{v}_{\textbf{END}}, \mathbf{0})$$



Parallax Occlusion Mapping

- ***<u>STOP!!!</u>***
- This sample tells us where the ray intersects with the detailed surface ☺
- …but how do we know for sure???



Parallax Occlusion Mapping

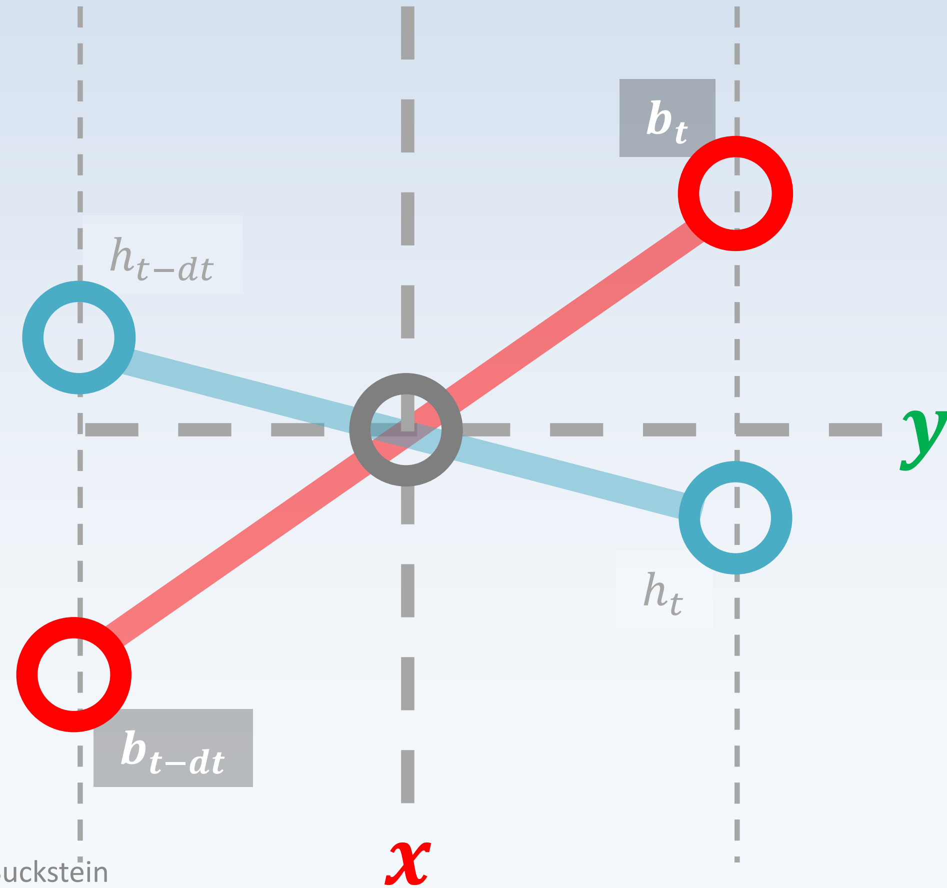
3) Find where ray intersects surface…

$$y = mx + b$$

- Same formula, different variables:

$$y = m_b x + b_{t-dt}$$
$$y = m_h x + h_{t-dt}$$

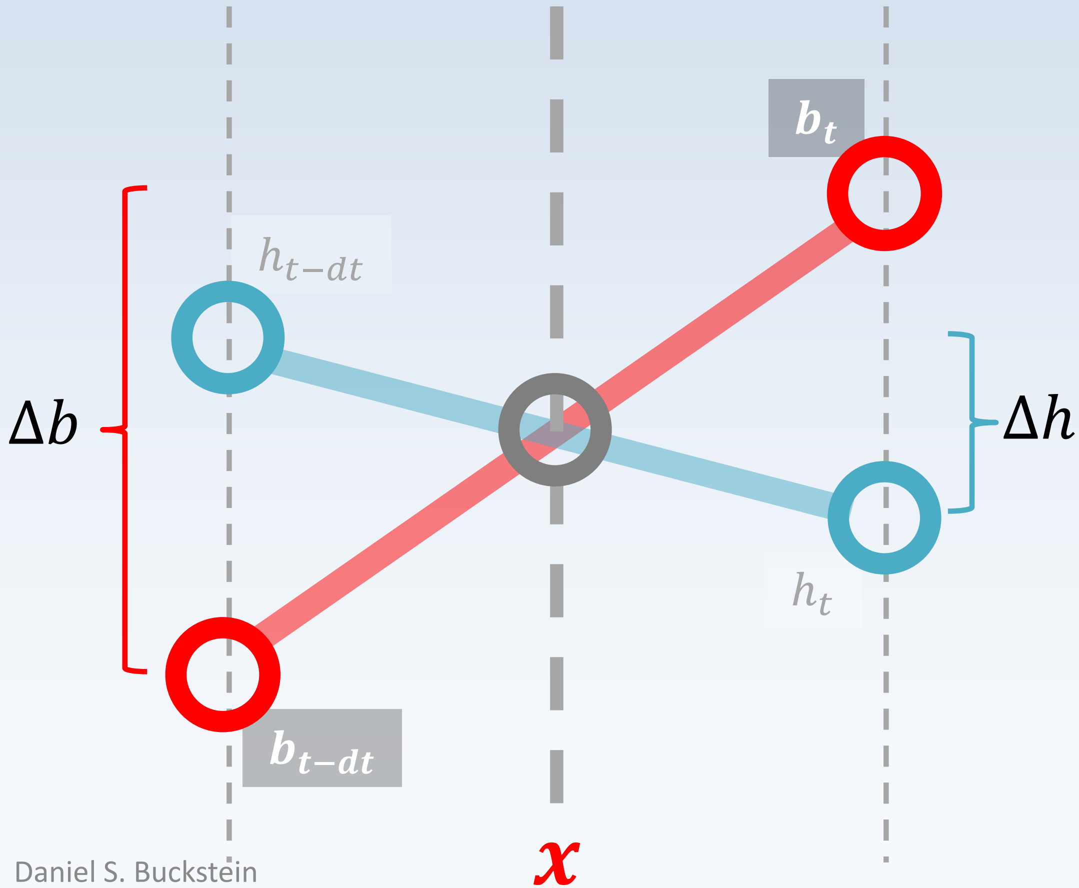
ISOLATE AND SOLVE FOR x



Parallax Occlusion Mapping

- Isolate and solve for $\boldsymbol{x}$

$$\boldsymbol{y} = \boldsymbol{y}$$
$$m_b\boldsymbol{x} + b_{t-dt} = m_h\boldsymbol{x} + h_{t-dt}$$
$$m = \frac{\Delta y}{\Delta x}$$
$$\frac{\Delta b}{\Delta x}\boldsymbol{x} + b_{t-dt} = \frac{\Delta h}{\Delta x}\boldsymbol{x} + h_{t-dt}$$
$$\Delta x = 1 \text{ (single step*)}$$
$$\Delta b\boldsymbol{x} + b_{t-dt} = \Delta h\boldsymbol{x} + h_{t-dt}$$
$$\Delta b\boldsymbol{x} - \Delta h\boldsymbol{x} = h_{t-dt} - b_{t-dt}$$
$$(\Delta b - \Delta h)\boldsymbol{x} = h_{t-dt} - b_{t-dt}$$



*ratio of step intervals for both lines; since they are the same here, ratio is 1

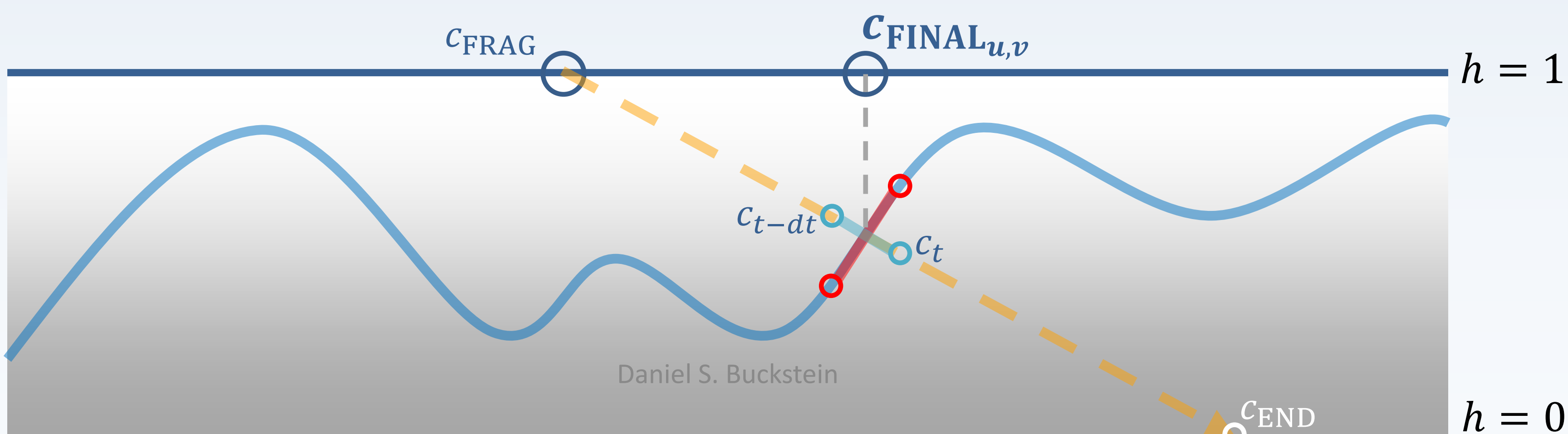
Parallax Occlusion Mapping

- We use $\boldsymbol{x}$ as an interpolation parameter:

$$\boldsymbol{x} = \frac{h_{t-dt} - b_{t-dt}}{\Delta b - \Delta h}$$

- ***FINAL TEXTURE COORDINATE:***

$$\boxed{\boldsymbol{c}_{\mathbf{FINAL}} = \mathbf{lerp}(\boldsymbol{c}_{\boldsymbol{t-dt}}, \boldsymbol{c}_{\boldsymbol{t}}, \boldsymbol{x})}$$



Parallax Occlusion Mapping

- Any *surface texture* in your shader should sample using the *final* texture coordinate
  - Diffuse map, specular map, normal map…
- Any *above-surface texture* uses the original
  - Clouds and other VFX
- The resulting illusion of "layers" is called "***parallax***"
- Next steps? Self-shadowing??? ;)

The end.

- Questions? Comments? Concerns?