Intermediate Graphics & Animation Programming

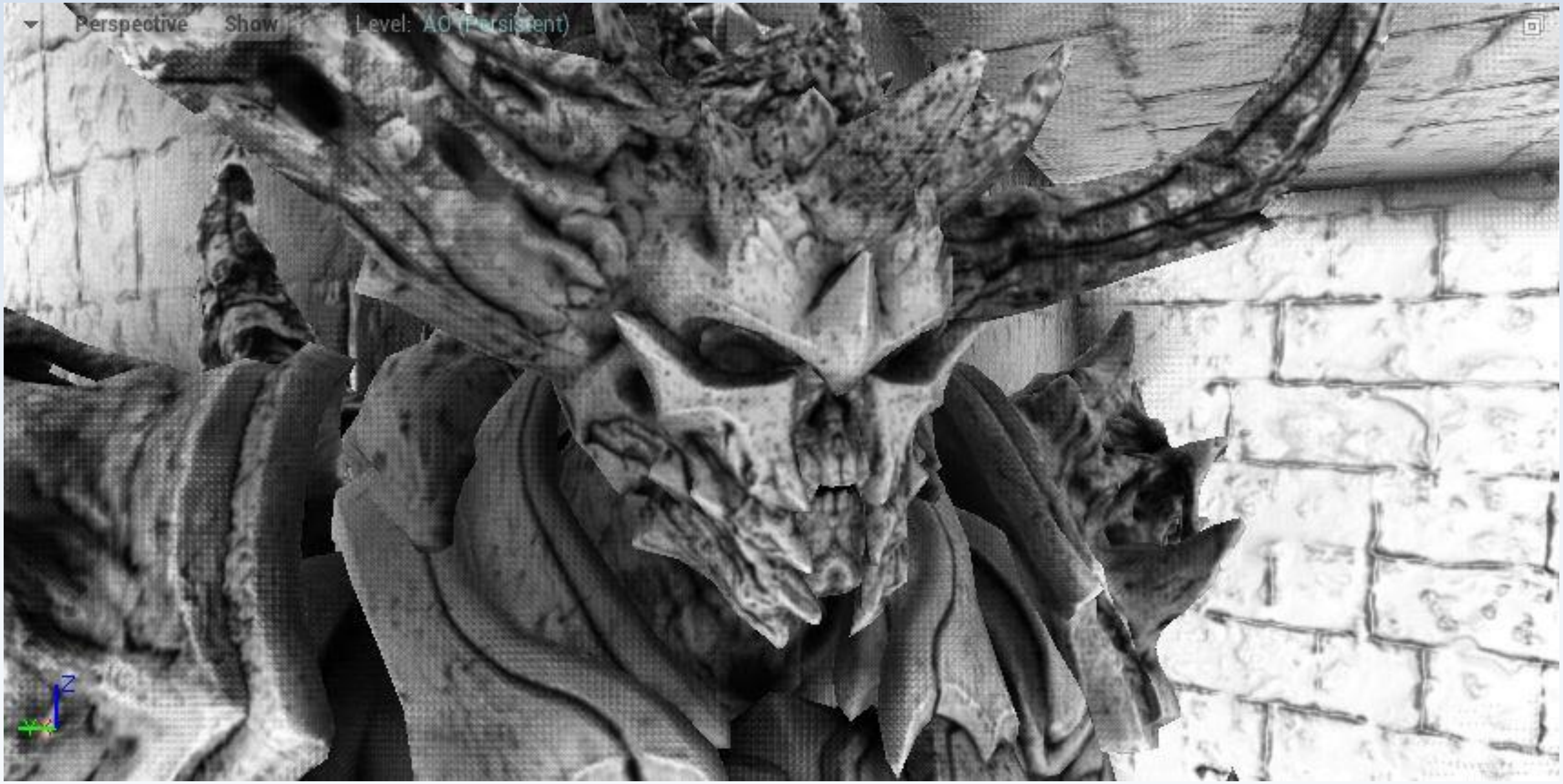
GPR-300
Daniel S. Buckstein

Global Illumination & Screen-Space Ambient Occlusion
Advanced Topics: Modern Techniques

License

Global Illumination

- Modern game engines:



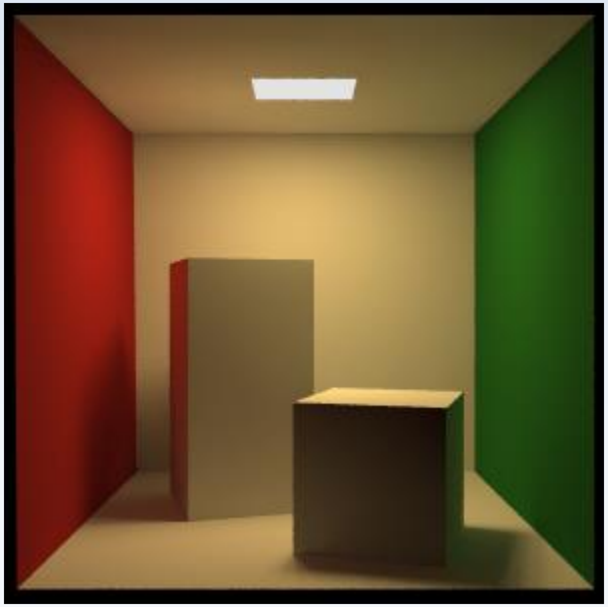
https://docs.unrealengine.com/latest/images/Engine/Rendering/LightingAndShadows/AmbientOcclusion/ao_1.jpg

Global Illumination

- Ubiquitous test for accurate global illumination:
- Utah teapot → generic geometry
- Stanford bunny → mesh reconstruction
- ***Cornell box → global illumination***
- ***Enter ray-tracing and realistic lighting models***

Global Illumination

- The Cornell Box:
- http://www.graphics.cornell.edu/online/box/



Global Illumination

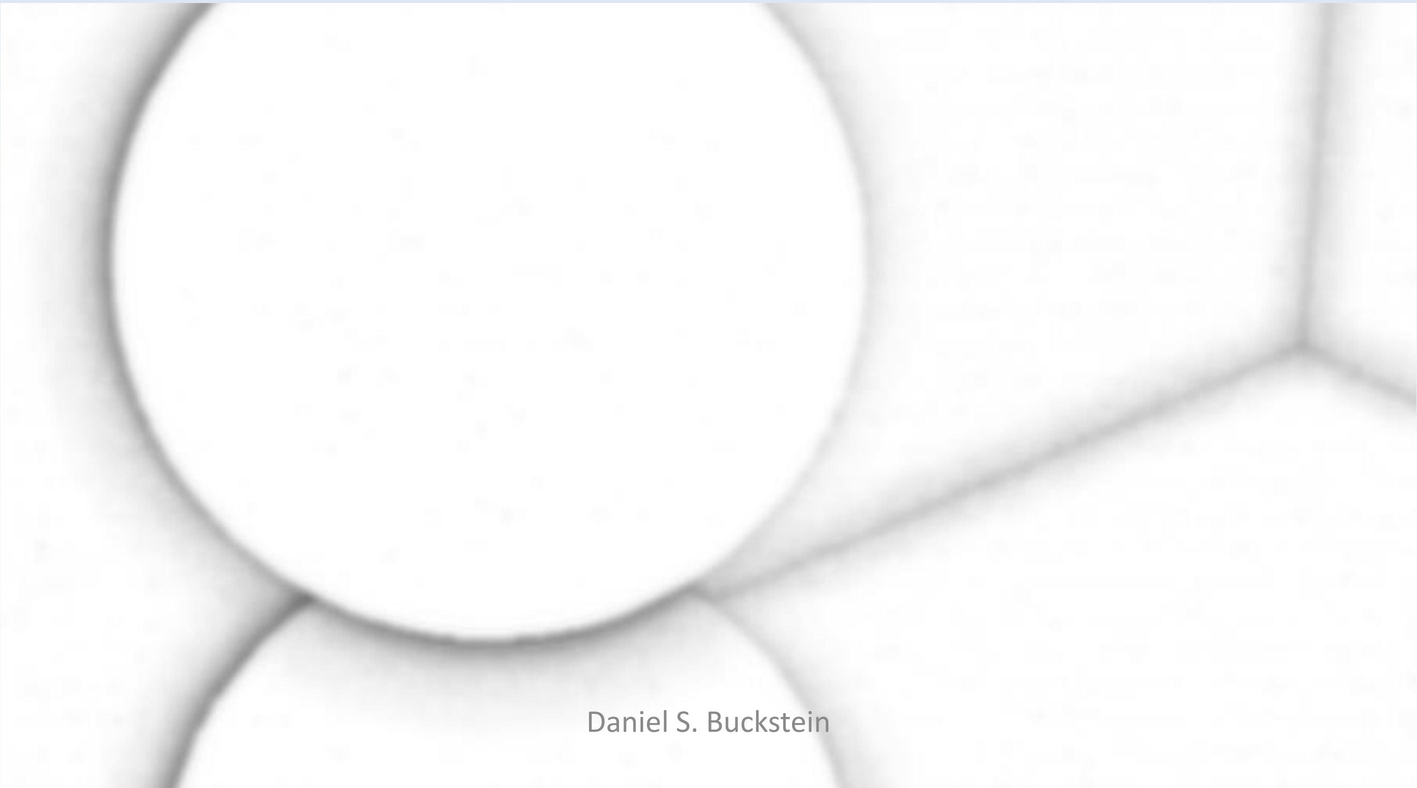
- Ray tracing summary:
- For each pixel, fire ray into scene
- Trace collisions with surfaces
- Each surface collision results in a ‘bounce’ and an accumulation of colour
- Repeat until ray expires or hits light source

Global Illumination

- PROBLEM with the Cornell box:
    - (and ray tracing methods in general)
- Realism over performance
- 200+ texture samples per fragment… ☹
- Modern renderers are getting more optimized for this ☺

SSAO

- ***Screen-Space Ambient Occlusion*** (SSAO):
- A reasonable alternative

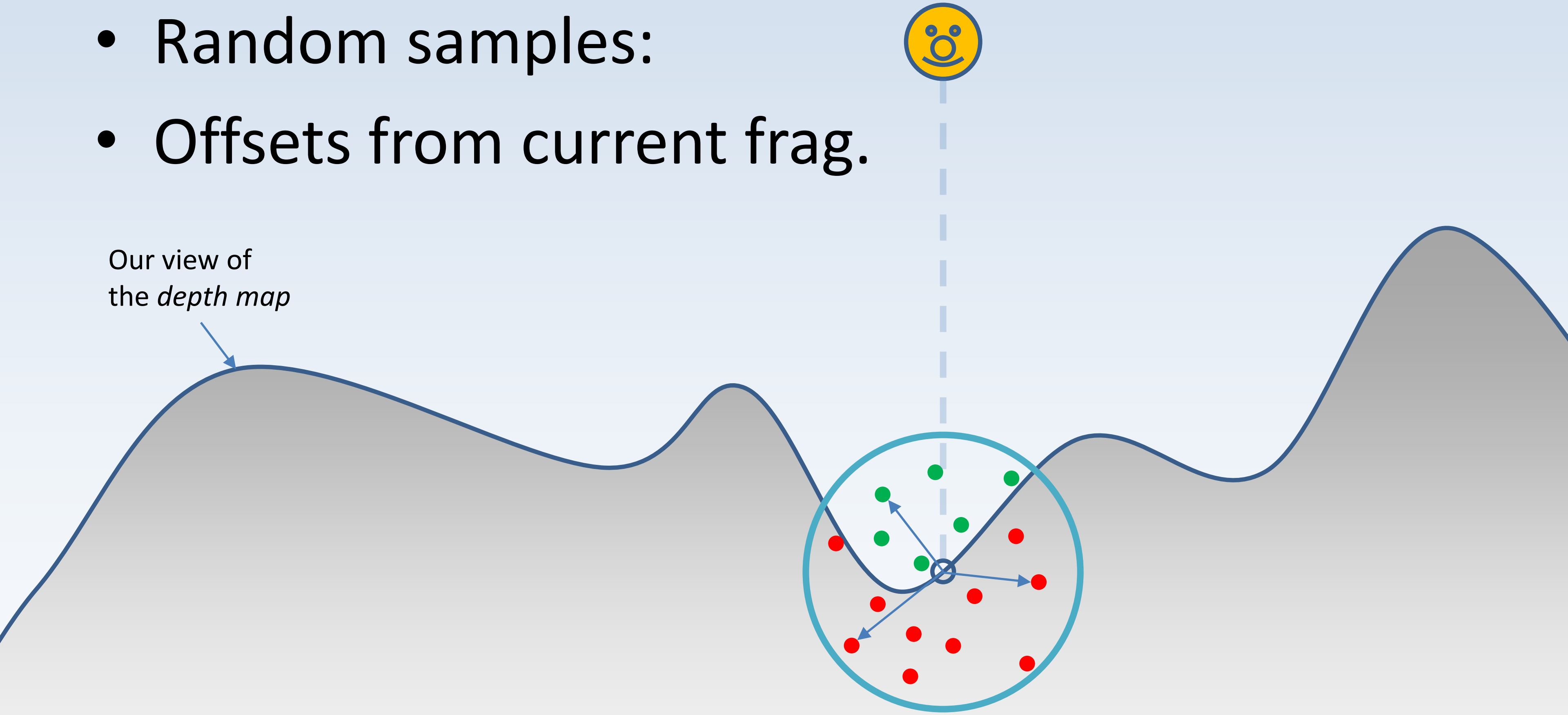


SSAO

- SSAO: *deferred* occlusion algorithm
- Makes heavy use of the depth map
- Many ways to do it:
- http://frederikaalund.com/a-comparative-study-of-screen-space-ambient-occlusion-methods/
- We'll talk about a good one to start with ☺

SSAO

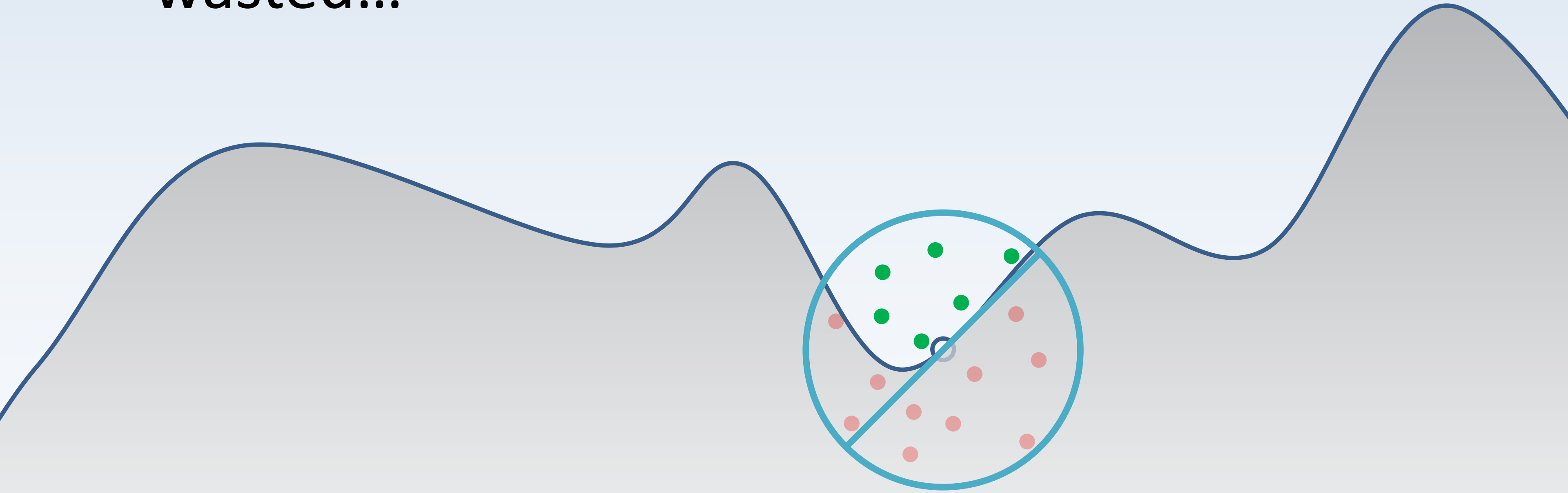
- Originally a spherical sampling algorithm:
- Random samples:
- Offsets from current frag.



Our view of
the *depth map*

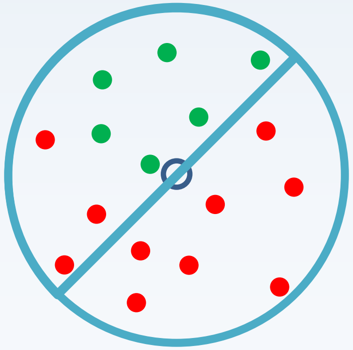
SSAO

- Good news: gets the job done with few samples
- Bad news: almost half of the samples are wasted…



Global Illumination

- Improved method: use a "***hemisphere sampling kernel***"
- 3D kernel of random samples that fit within a hemisphere around each point on the surface!



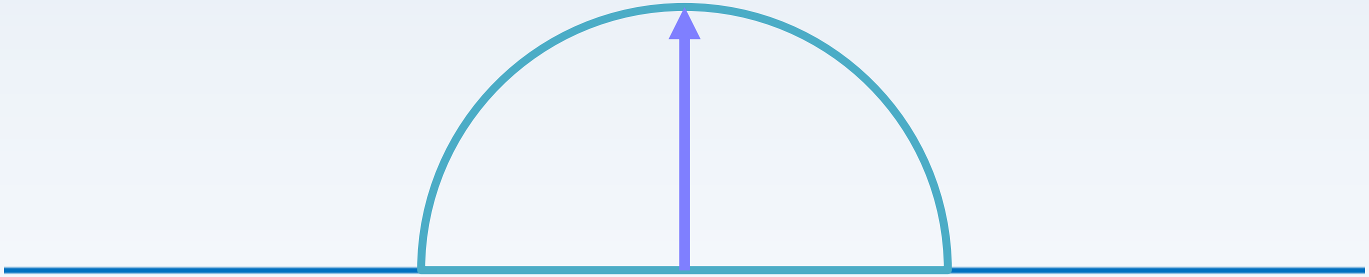
SSAO

- Constructing the hemisphere sampling kernel:
- ***This is done on the CPU side, one time (load)***
- Note: use of the word “kernel”: we are talking about 3D points, not a 2D convolution kernel!
- Pick how many samples you want and create an array of 3D vectors:

```
const int numSamples = 8;
vec3 hemiKernel[numSamples];
```

SSAO

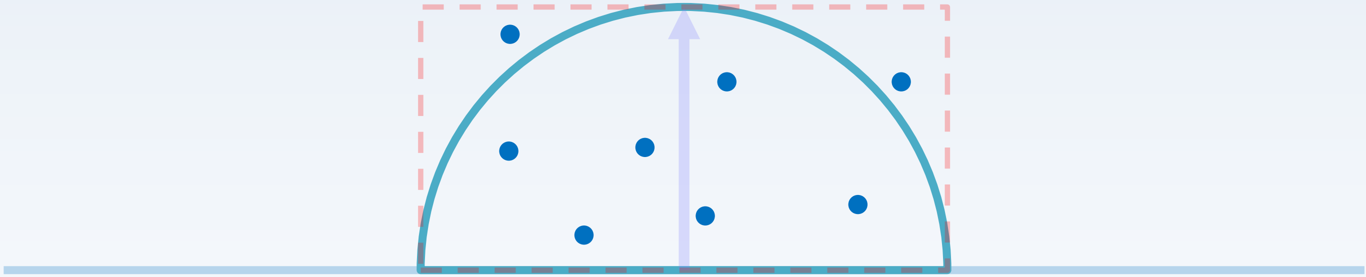
- Constructing the hemisphere sampling kernel:
- Need to imagine for a second that ***all fragments use the same kernel to start***
- Surface-relative: hemisphere oriented to ***default normal…???***



SSAO

- Constructing the hemisphere sampling kernel:
- ***Step 1***: for each vector ‘v_i’ in the sample list, pick a random vector in the hemisphere:

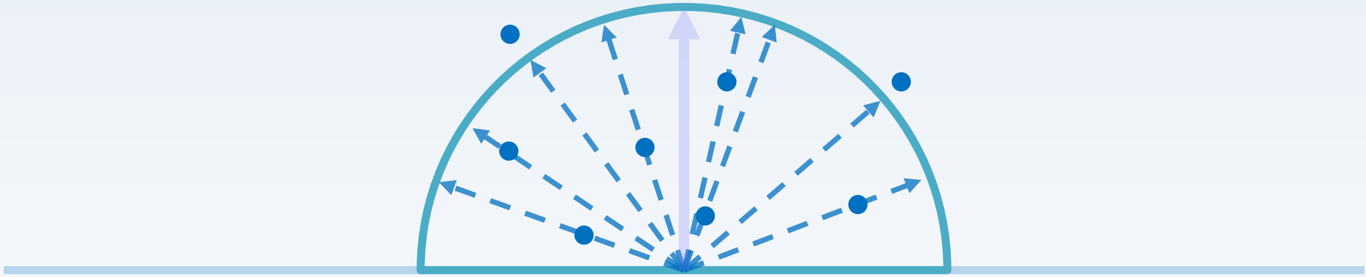
$$x_i = \text{random}(-1, +1)$$
$$y_i = \text{random}(-1, +1)$$
$$z_i = \text{random}(0, 1)$$



SSAO

- Constructing the hemisphere sampling kernel:
- ***Step 2***: normalize each vector to ensure it lies on hemispheric surface:

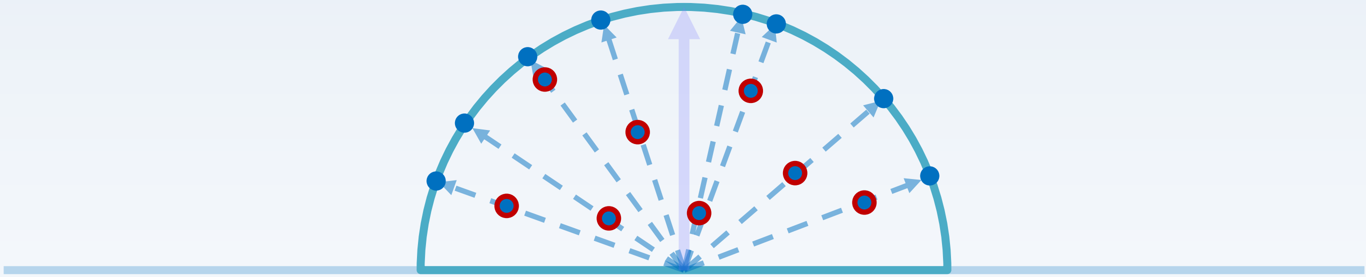
$$\hat{v}_i = \mathrm{normalize}(v_i)$$



SSAO

- Constructing the hemisphere sampling kernel:
- Result of normalization is random along the edge of the hemisphere… what about within?
- ***Step 3***: randomize the length of each vector:

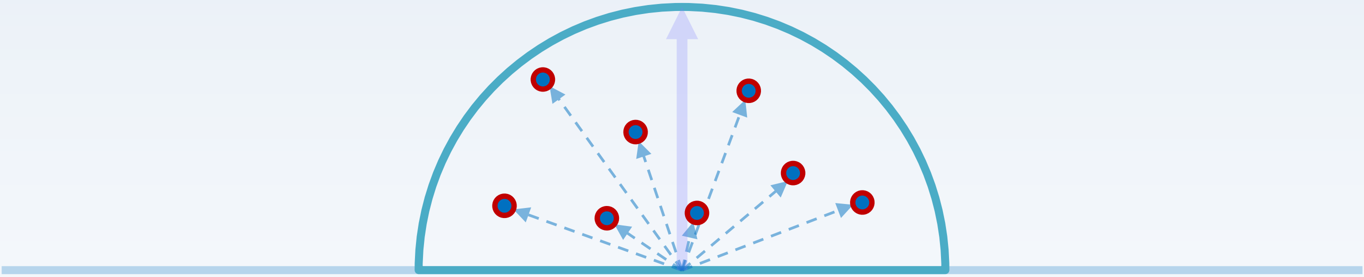
$$\boldsymbol{s_i} = \mathbf{random}(\mathbf{0}, \mathbf{1})\, \boldsymbol{\hat{v}_i}$$



SSAO

- Constructing the hemisphere sampling kernel:
- Optional step: bias length towards center
- …for now we'll just stick with this kernel:
- …how do we orient it to the surface???

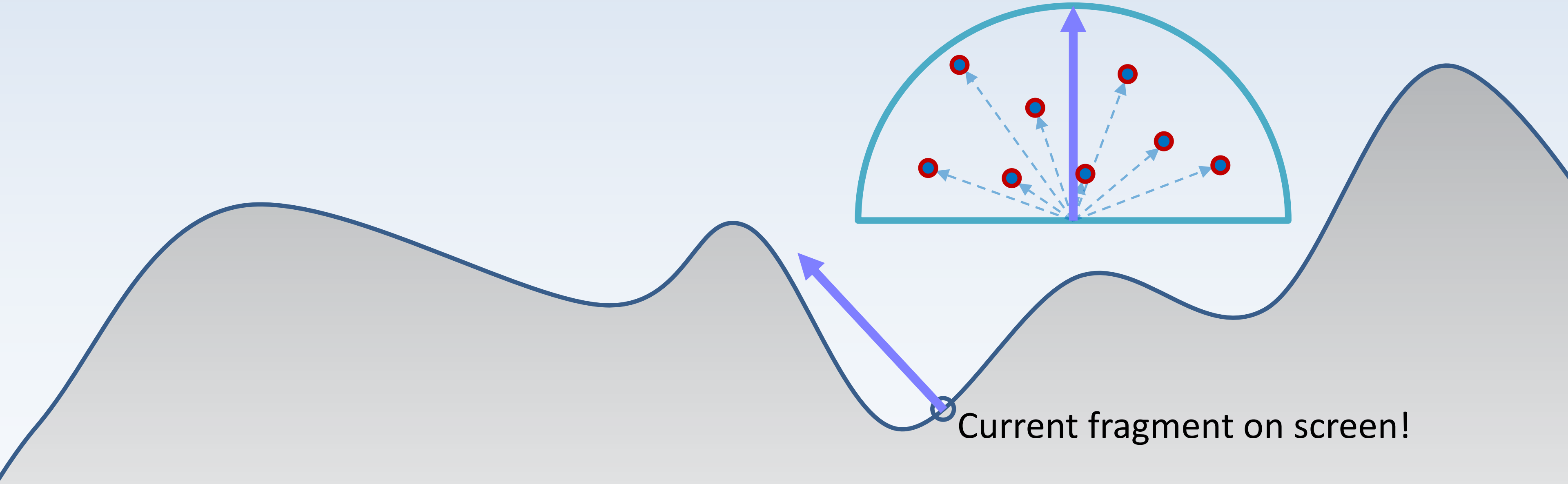
$$\boldsymbol{s_i} = \mathbf{random}(\mathbf{0}, \mathbf{1})\ \boldsymbol{\hat{v}_i}$$



SSAO

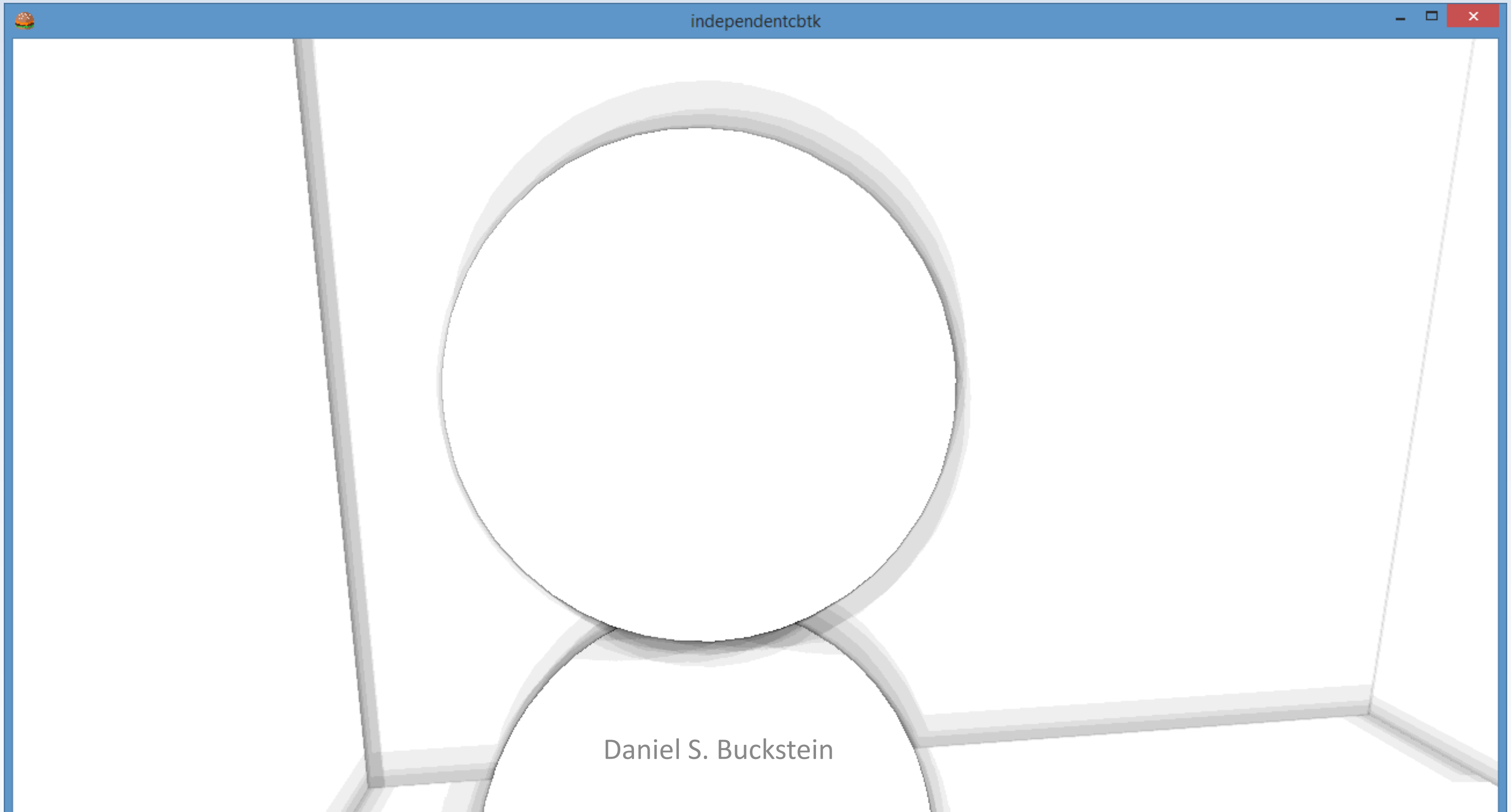
- Orientation is defined by the ***normal*** at each fragment…
- …and???

$$\boldsymbol{s_i} = \mathbf{random}(\mathbf{0}, \mathbf{1})\ \boldsymbol{\hat{v}_i}$$



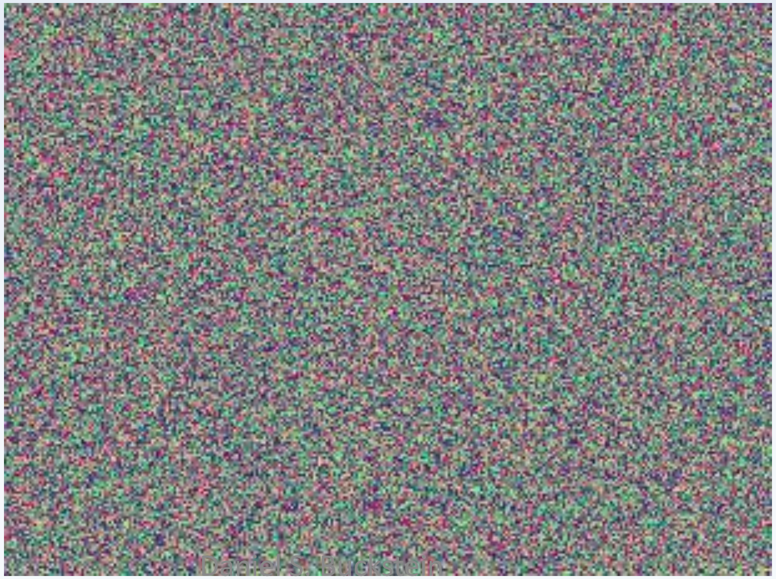
SSAO

- Need at least 2 vectors to define a rotation…
- …while avoiding strange artifacts…



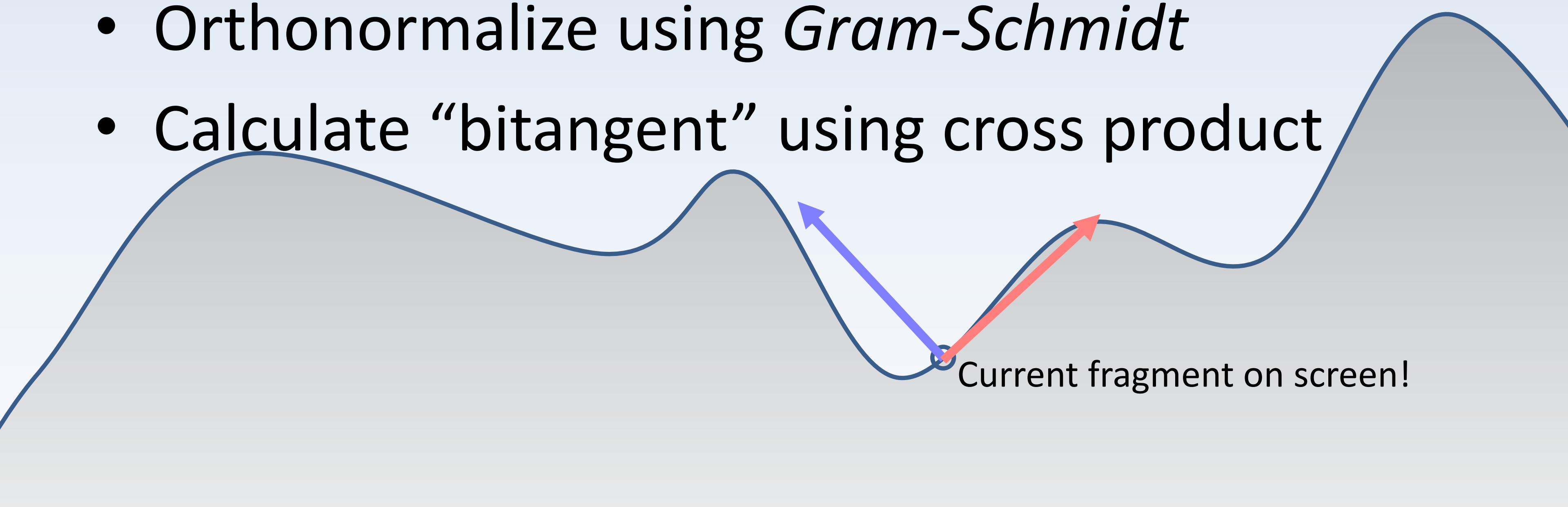
SSAO

- Also created on load: ***noise texture***
- Can define each pixel as a 2D vector
    - Default normal represents pure-Z, we just need an XY value to create a valid orientation



SSAO

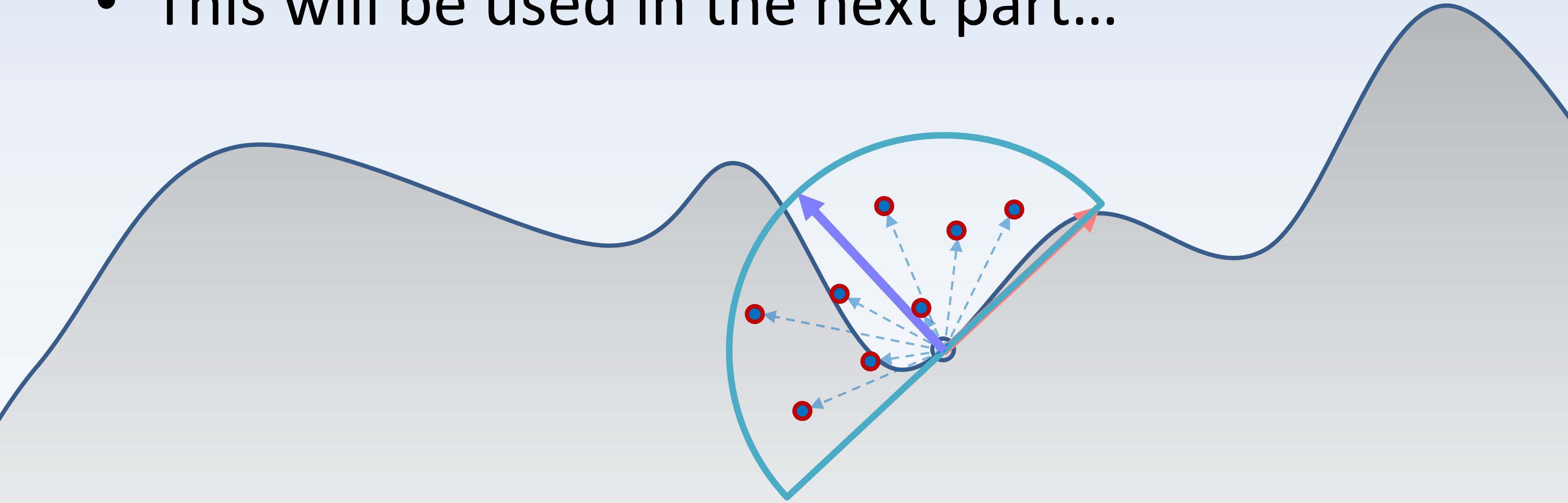
- Here we begin the ***SSAO algorithm***:
- The hemisphere kernel is passed in as uniform
- Sample from noise to get “tangent”
- Orthonormalize using *Gram-Schmidt*
- Calculate “bitangent” using cross product



https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process

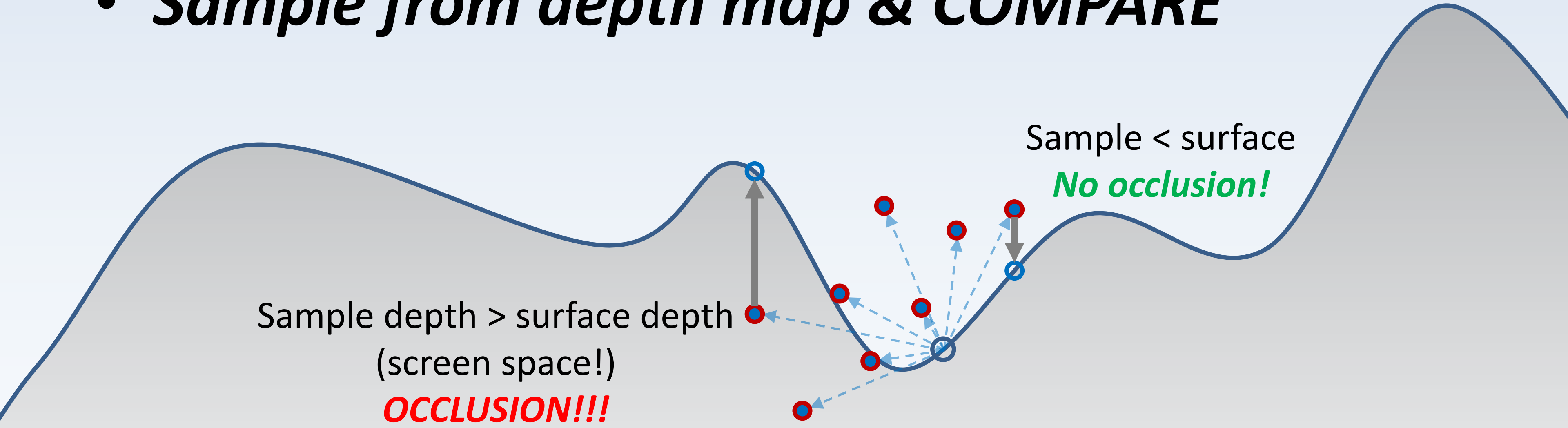
SSAO

- ***SSAO algorithm***:
- We now have a “random” rotation matrix for our 3D hemisphere sample kernel ☺
- This will be used in the next part…



SSAO

- ***SSAO algorithm***:
- ***Iterate through samples, offset from current fragment position on screen***
- ***Sample from depth map & COMPARE***



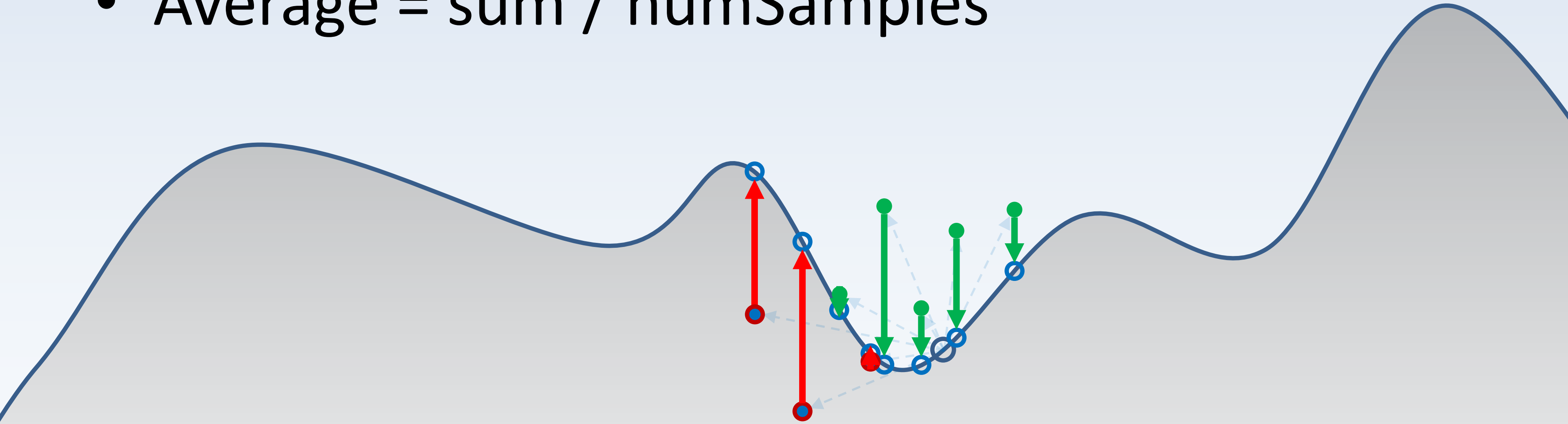
Sample < surface

No occlusion!

Sample depth > surface depth
(screen space!)
OCCLUSION!!!

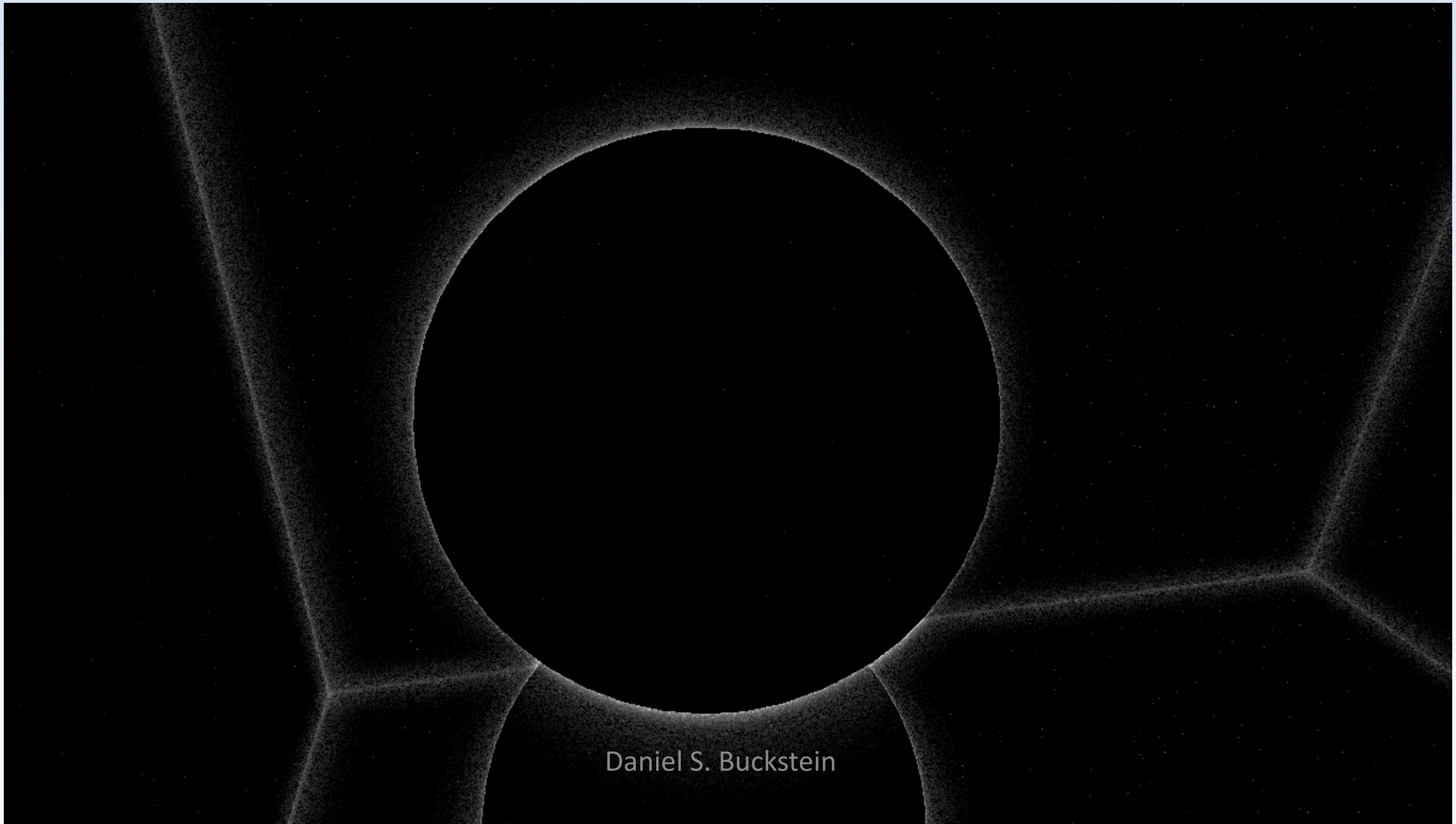
SSAO

- ***SSAO algorithm***:
- Accumulate all occlusions: add 1 if occluded, add 0 if not occluded
- Average = sum / numSamples



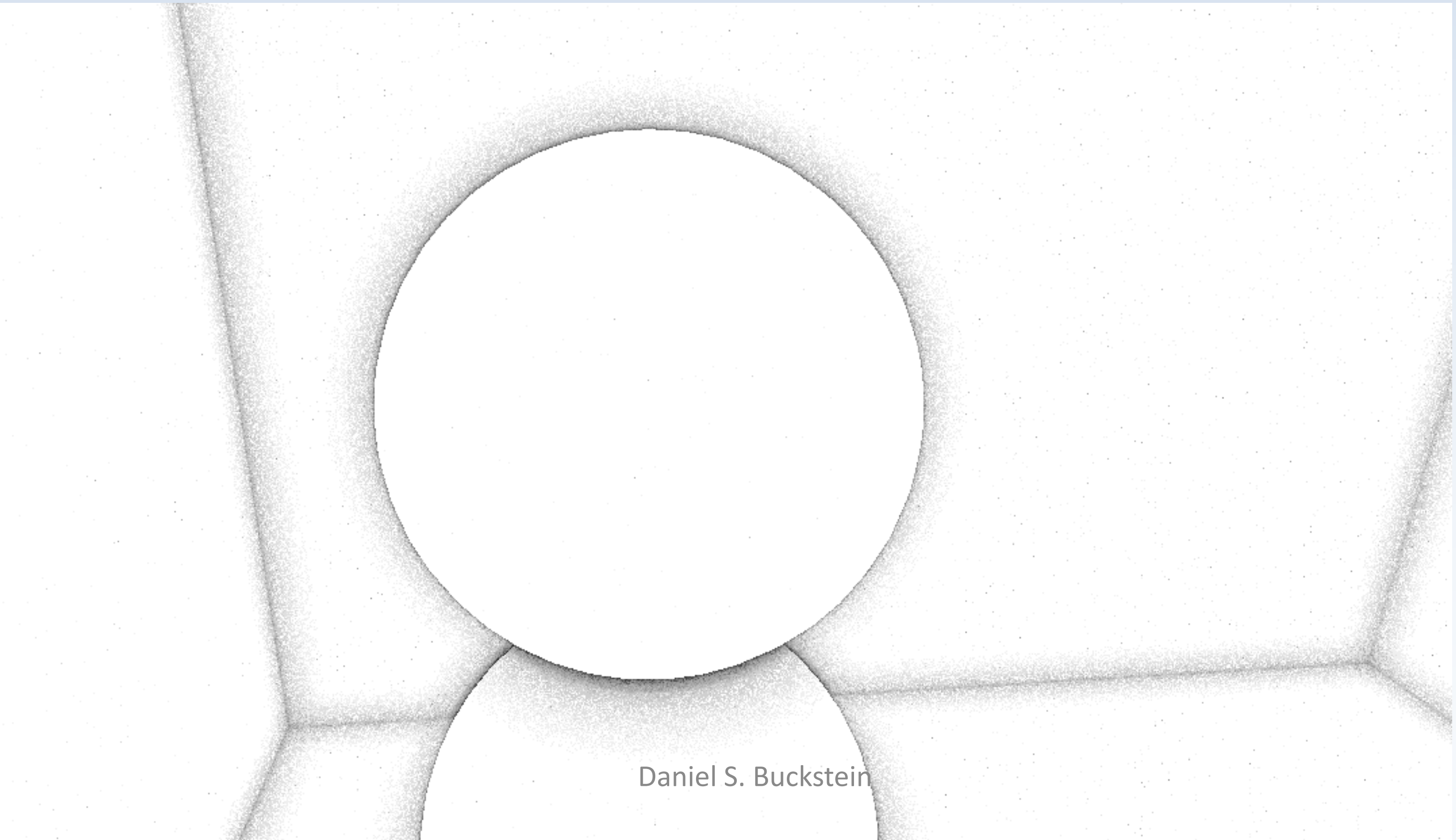
SSAO

- Result is a B/W image:
  - (you may see “stars”… it’s really pretty)



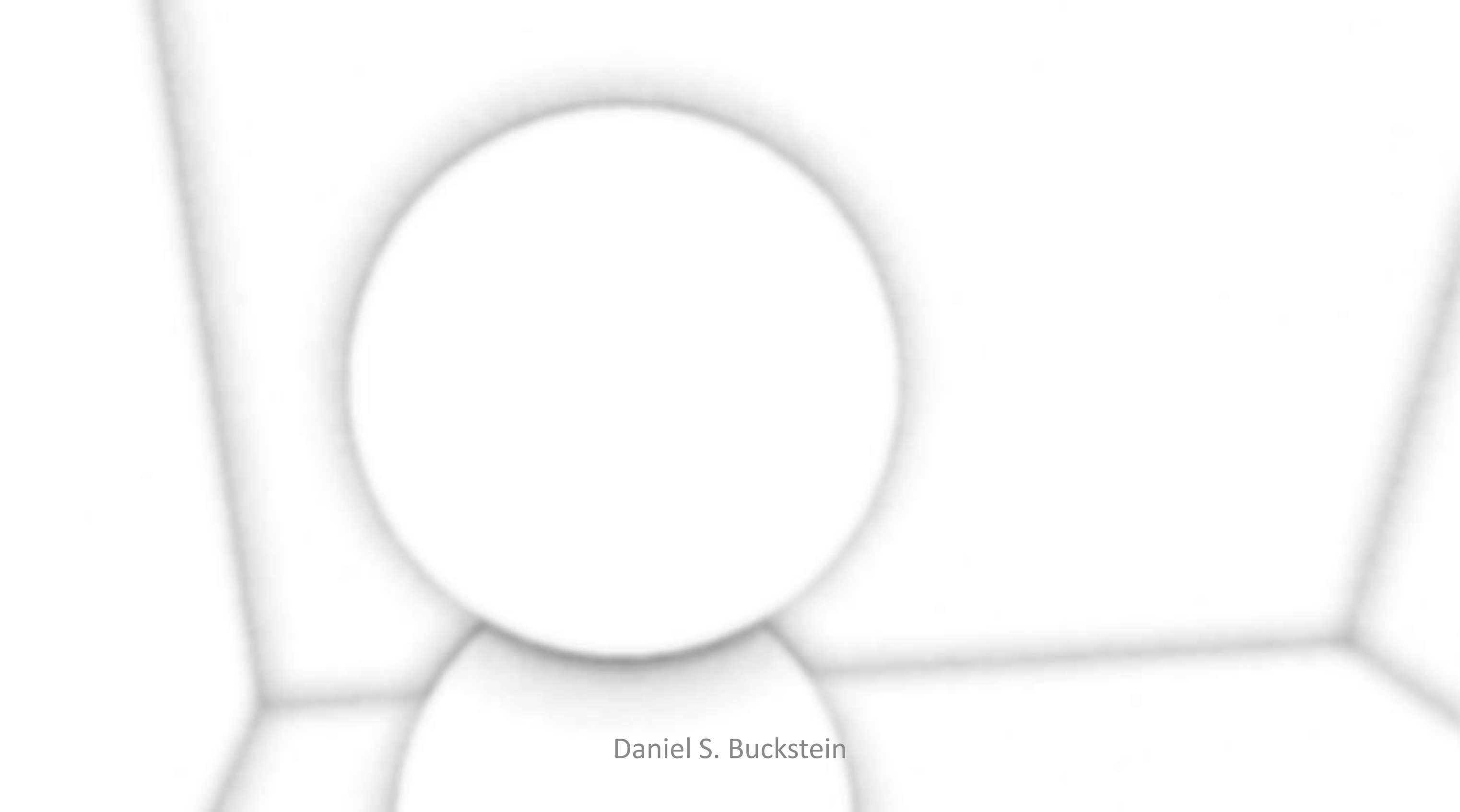
SSAO

- Invert average to get the classic look:



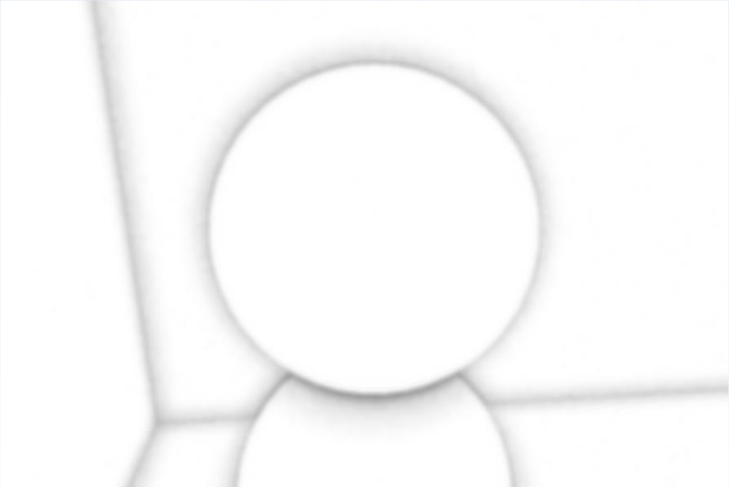
SSAO

- Problem: still looks a bit grainy…
- How do we make it look a bit smoother?

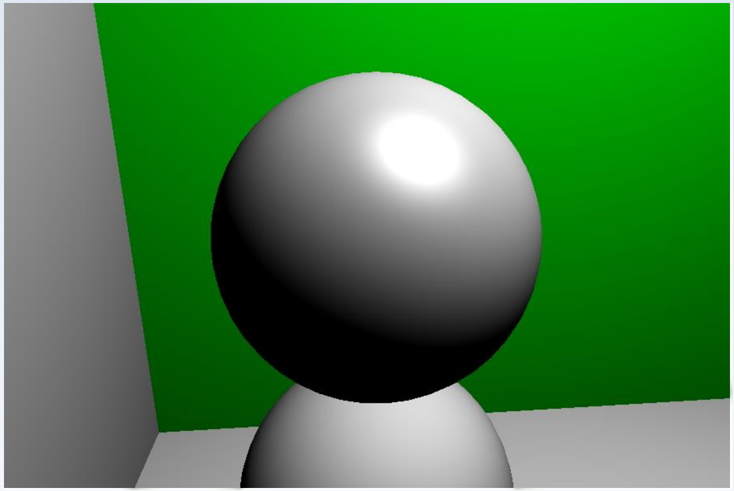


SSAO

- ***Final result***: multiply final SSAO map by final deferred shading/lighting result:



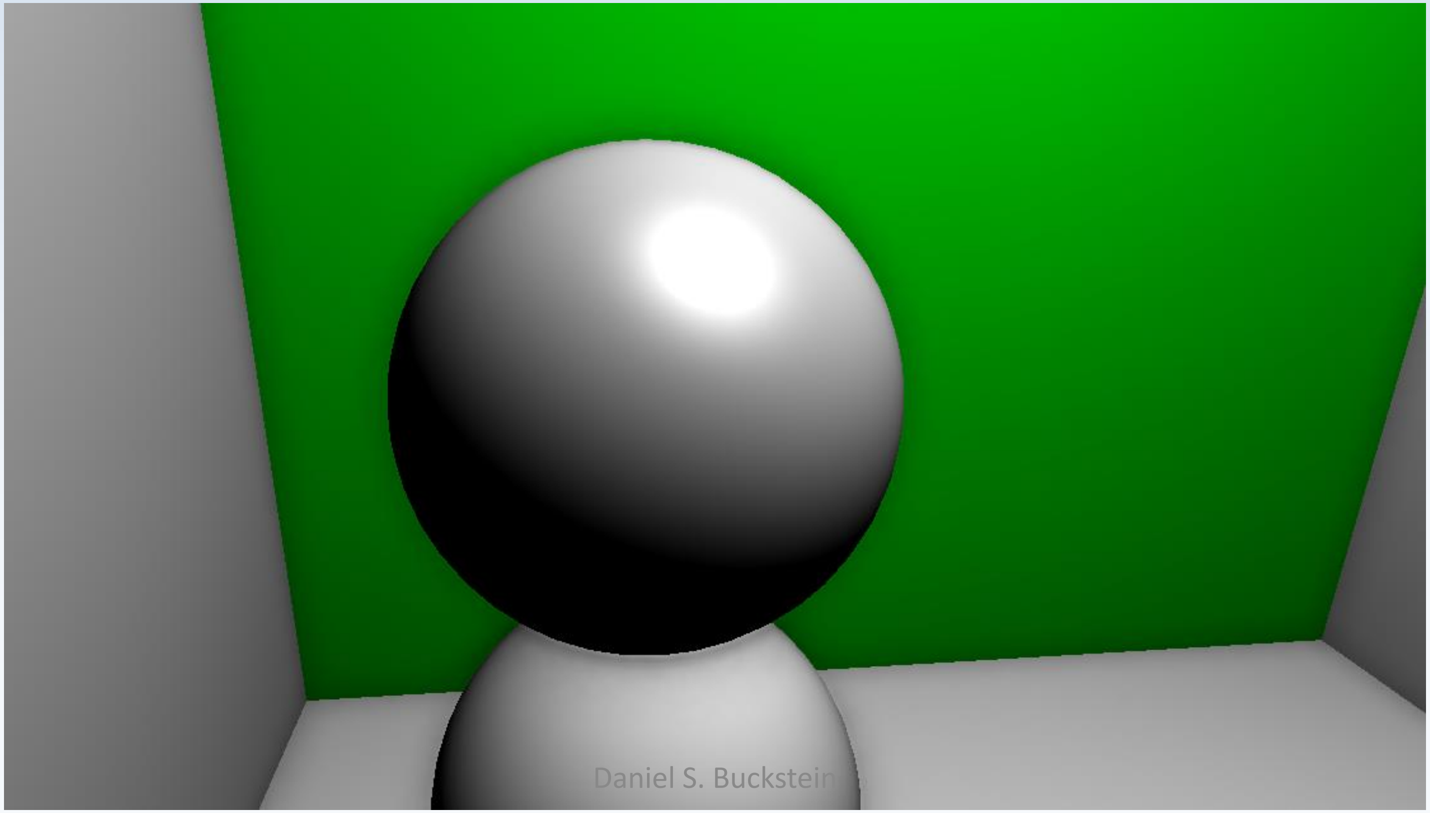
×



SSAO

- ***Final result***: multiply final SSAO map by final deferred shading/lighting result:

=



The end.

- Questions? Comments? Concerns?