Hybrid Stereoscopy in Ratchet & Clank: All 4 One

Jim Van Verth
Insomniac Games
www.insomniacgames.com
jim@essentialmath.com

Introductory Bits

- Talking about stereo solution for Ratchet & Clank: All 4 One
- Combination of standard stereographic and reprojection techniques
- No demo, sorry (stay tuned)

Brief History

- Wanted stereo in A4O
- Crytek announced reprojection
- Prototype Dec 2010
- Waffled between standard and reproj
- Shared work with R3
- Final solution May 2011

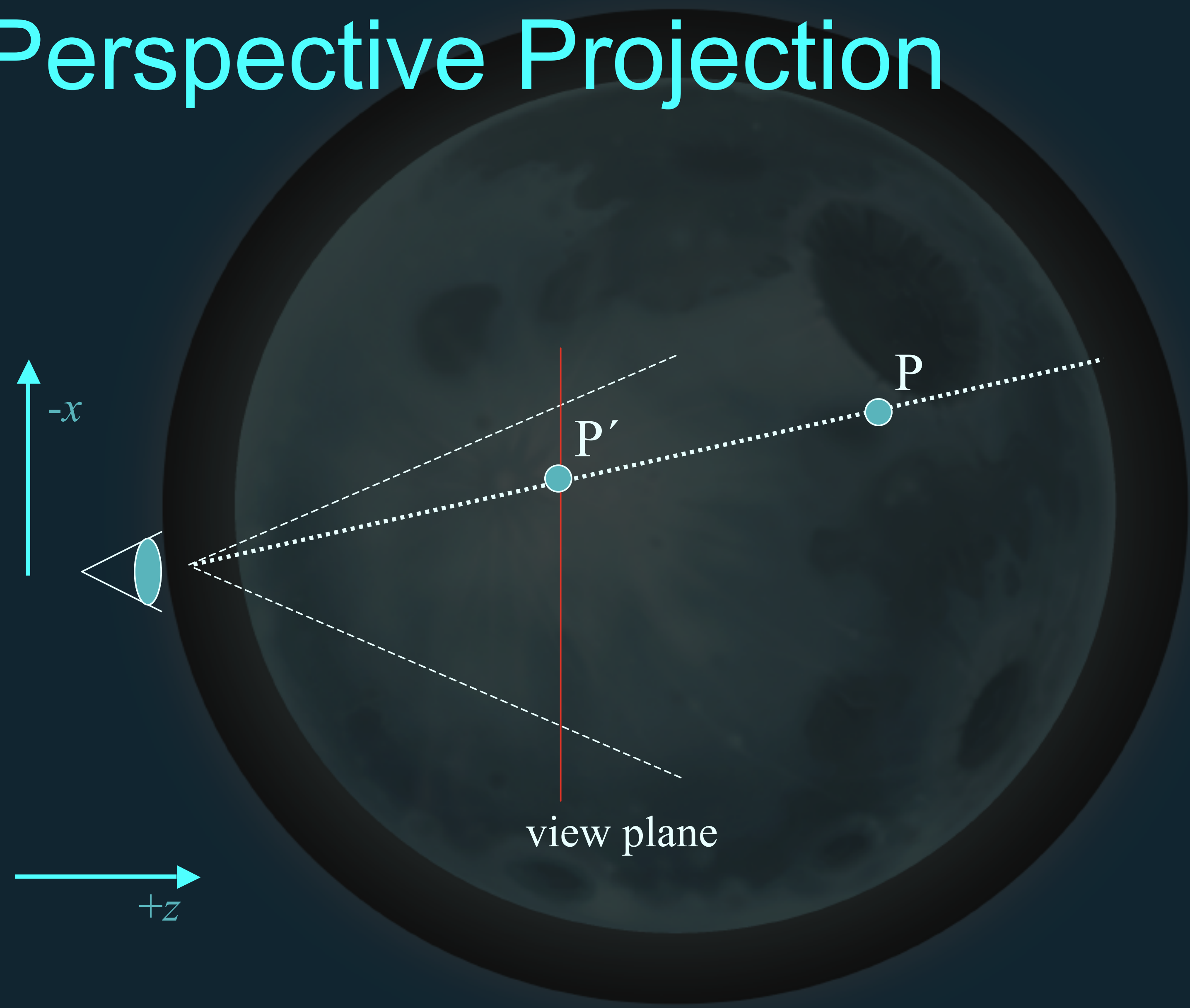
Outline

- Standard stereo projection
- Reprojection prototype
- Final solution
- Issues and future work

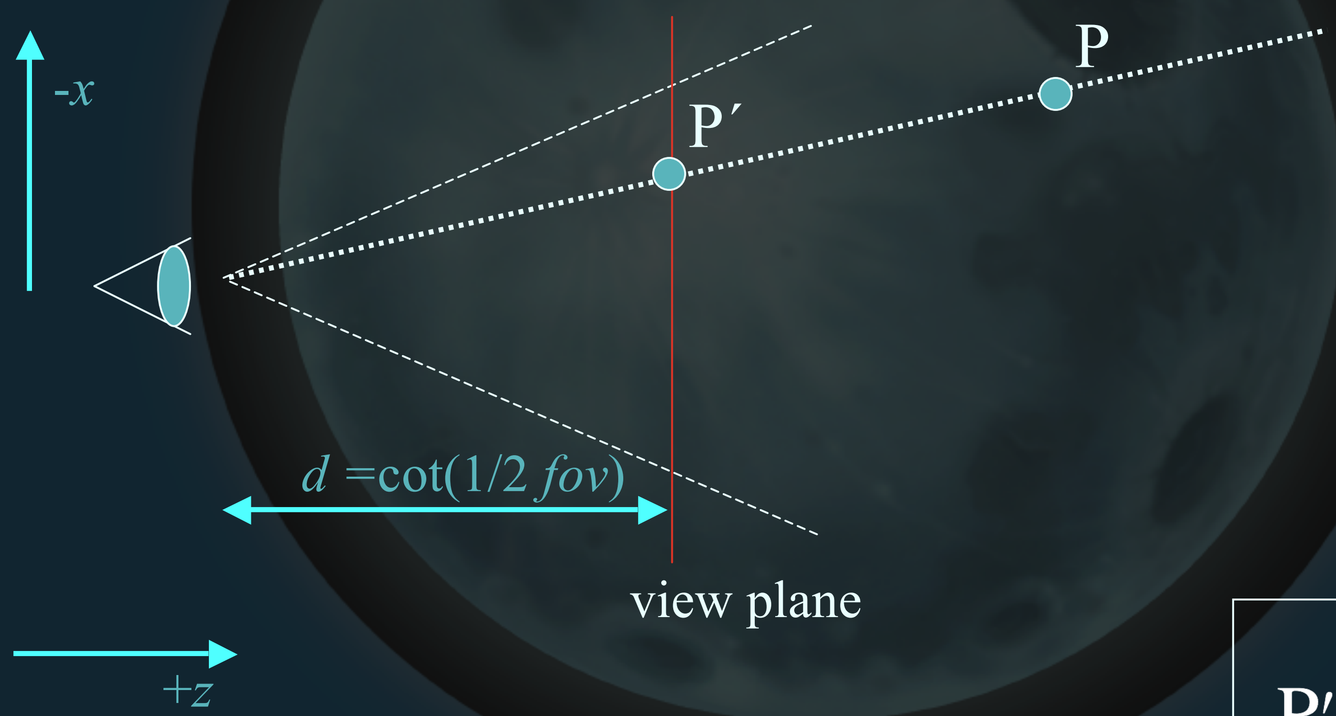
Perspective Projection

Perspective Projection

Perspective Projection



Perspective Projection



$$\boxed{P'_x = P_x \cdot \frac{d}{P_z}}$$

Perspective Projection

- In matrix form (Direct3D):

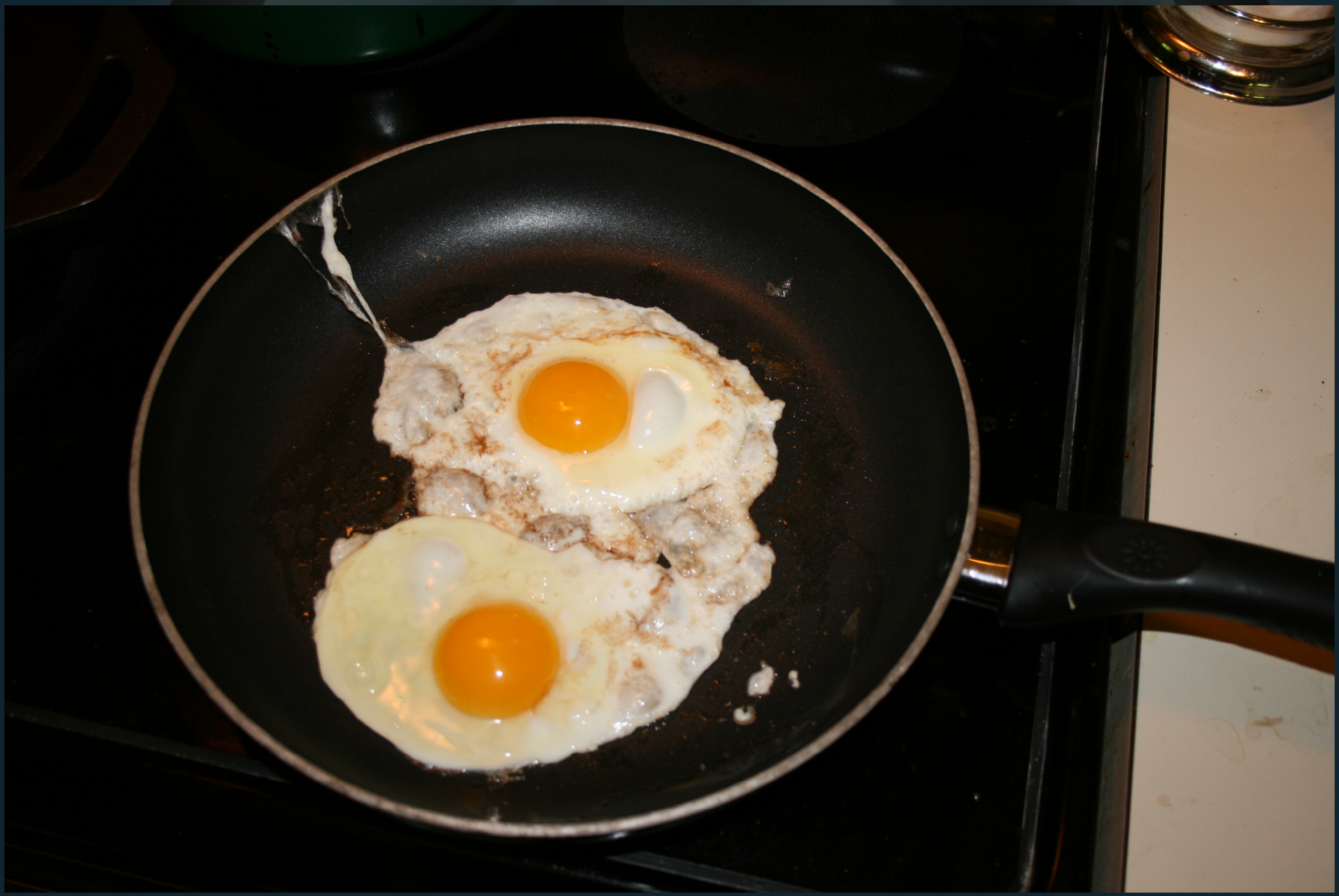
$$\begin{bmatrix} P_x & P_y & P_z & 1 \end{bmatrix} \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & da & 0 & 0 \\ 0 & 0 & Q & 1 \\ 0 & 0 & -Qn & 0 \end{bmatrix} = \begin{bmatrix} dP_x & daP_y & QP_z - Qn & P_z \end{bmatrix}$$

$$Q = \frac{f}{f - n}$$

- Reciprocal divide gives projected point

Stereographic Projection

- Two eyes, separated

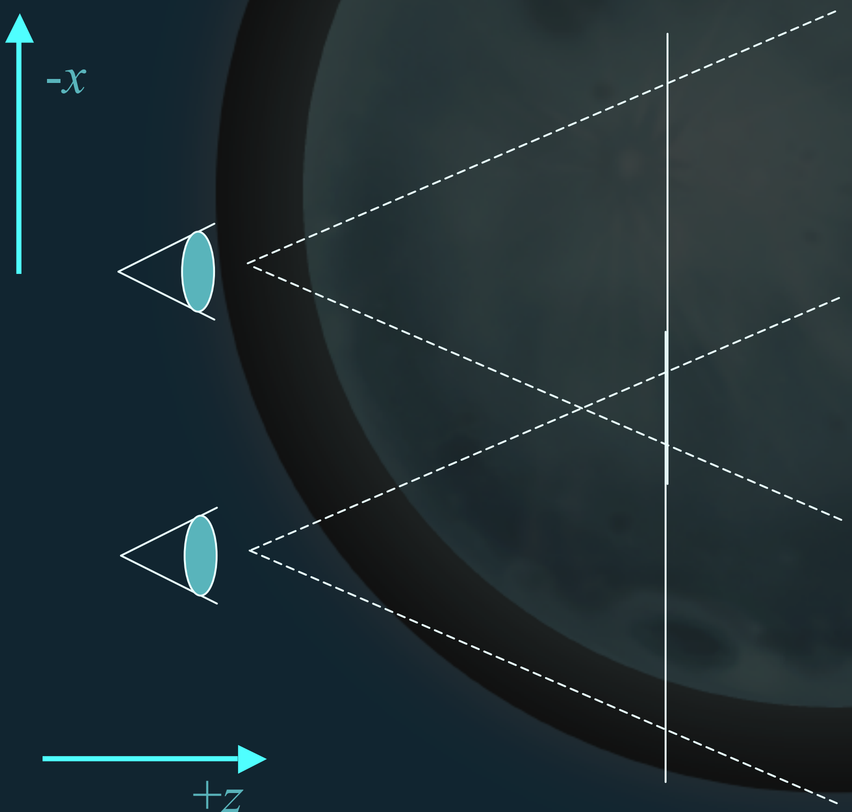


Stereographic Projection

- Two eyes, separated

Stereographic Projection

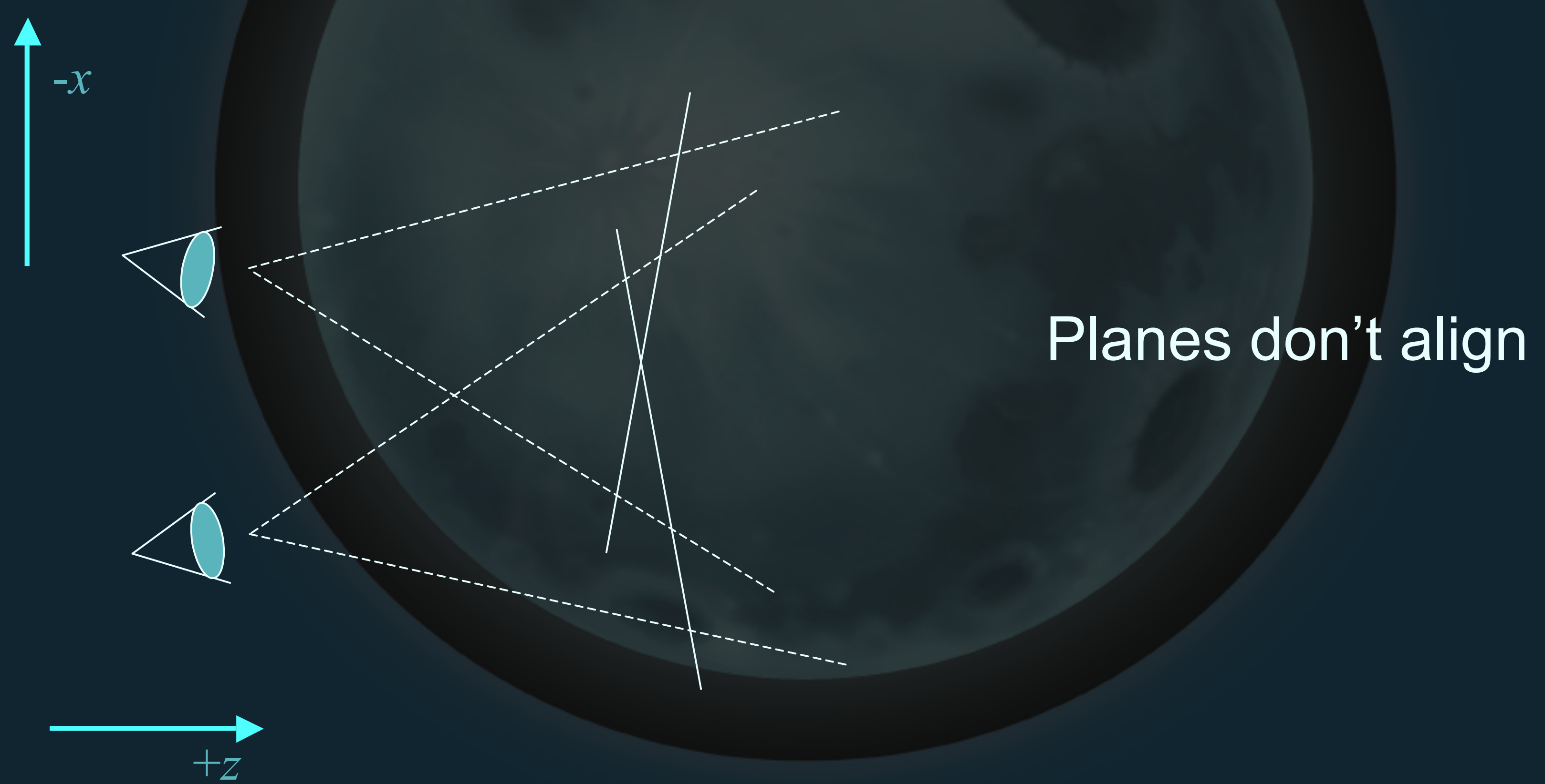
- Two eyes, separated



Bad convergence

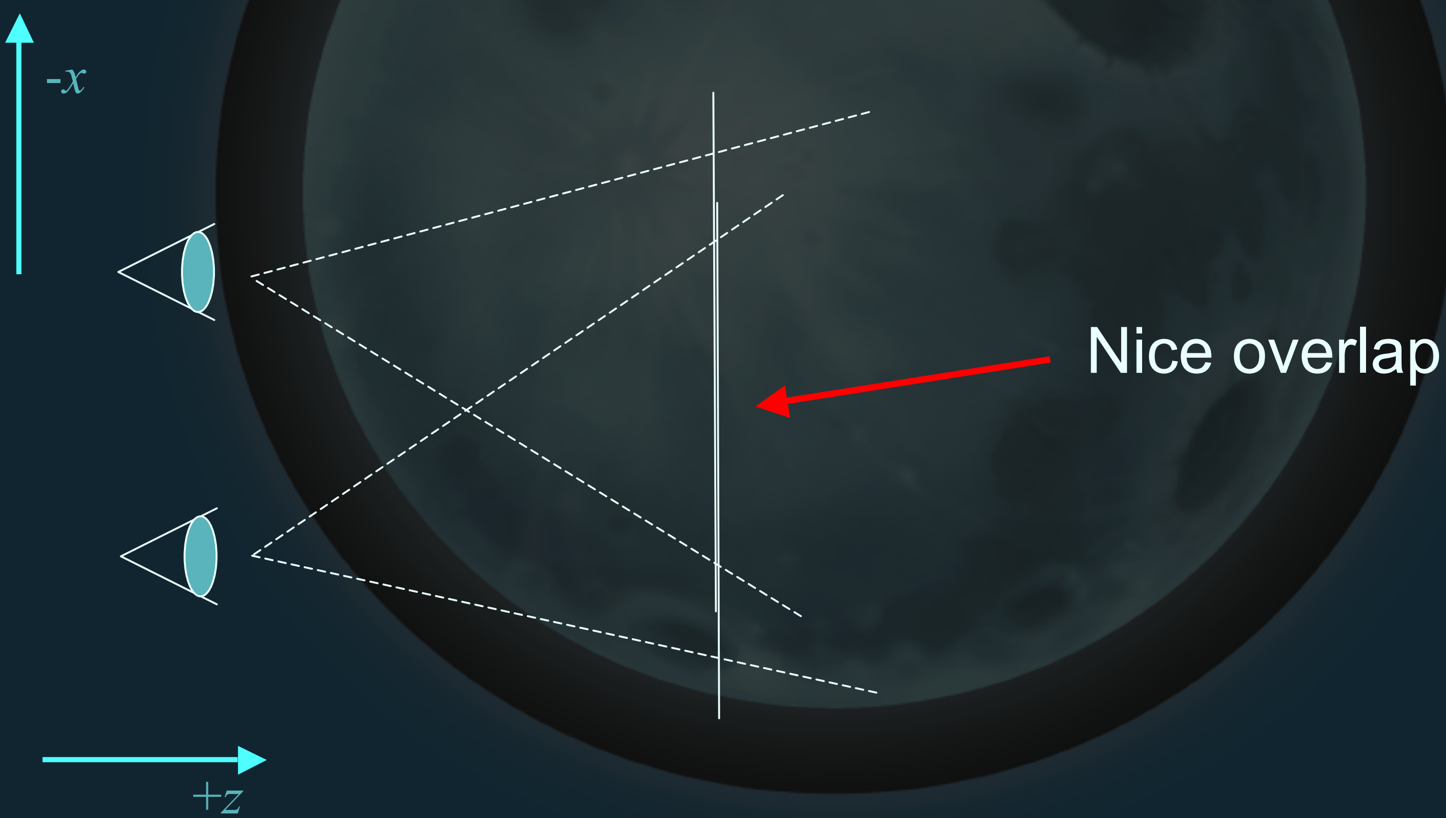
Stereographic Projection

- Two eyes, separated, toed in



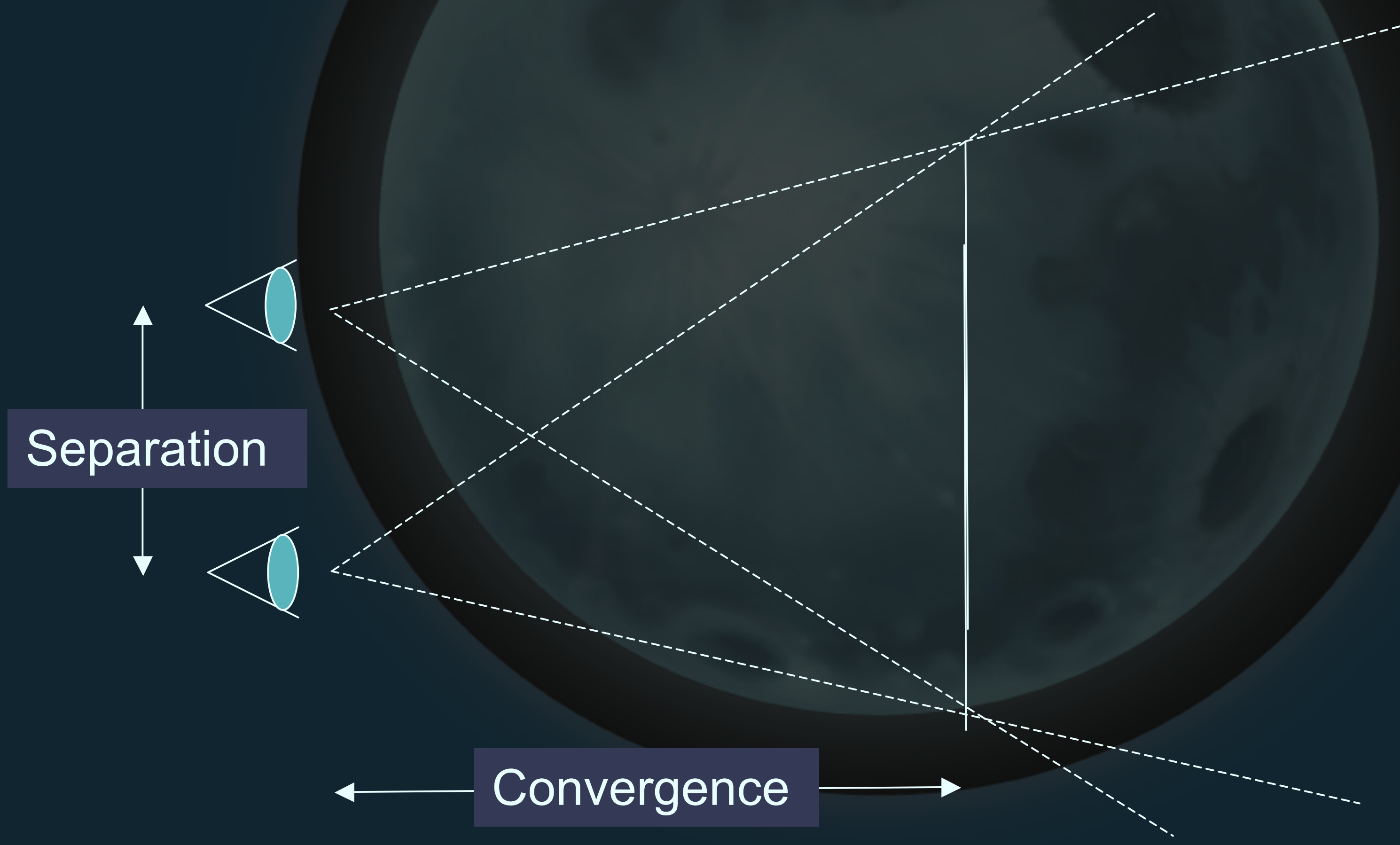
Stereographic Projection

- Two eyes, separated, sheared



Stereographic Projection

- Two eyes, separated, sheared



Stereographic Projection

- Separation
  - – Physical separation: interocular
  - – Virtual separation: interaxial
- Convergence distance
  - – Could be in view frame or NDC frame
  - – Know what frame you’re in!

Stereographic Projection

- Two pieces:
  - Add xz-shear to projection matrix
  - Translate cameras along x-axis

Stereographic Projection

- Shear

$$\begin{bmatrix} p_{11} & 0 & 0 & 0 \\ 0 & p_{22} & 0 & 0 \\ p_{31} \mp S & p_{32} & p_{33} & 1 \\ 0 & 0 & p_{34} & 0 \end{bmatrix}$$

(D3D)

- Camera translation

$$Trans(\pm T, 0, 0) \cdot \mathbf{M}_{\mathrm{view\_to\_world}}$$

- Left eye: -S, +T
- Right eye: +S, -T

Stereographic Projection

- Can incorporate camera xlate into projection

$$\begin{bmatrix} p_{11} & 0 & 0 & 0 \\ 0 & p_{22} & 0 & 0 \\ p_{31} \mp S & p_{32} & p_{33} & 1 \\ \pm T p_{11} & 0 & p_{34} & 0 \end{bmatrix}$$

Stereographic Projection

- Picking S & T

$$S = i/w$$

$$T = S \cdot c \cdot \tan(fov/2)$$

- Where

i = interocular (distance between pupils)

w = monitor width

c = convergence plane distance

fov = horizontal field of view

(From NVIDIA talk, amongst others)

Stereographic Projection

- Produces nice results
- But: must render everything twice
- Can cut framerate by half
- If running at 30fps, not so great

Stereographic Reprojection

- Idea:
  - Render view from one eye
  - Use depth and color information to generate other eye's view via pixel shader (or SPU)
  - In theory, much faster!
  - Only need worry about x (really u) direction
  - Note: need access to depth & color
    - So render to rendertarget or FBO, then copy to display

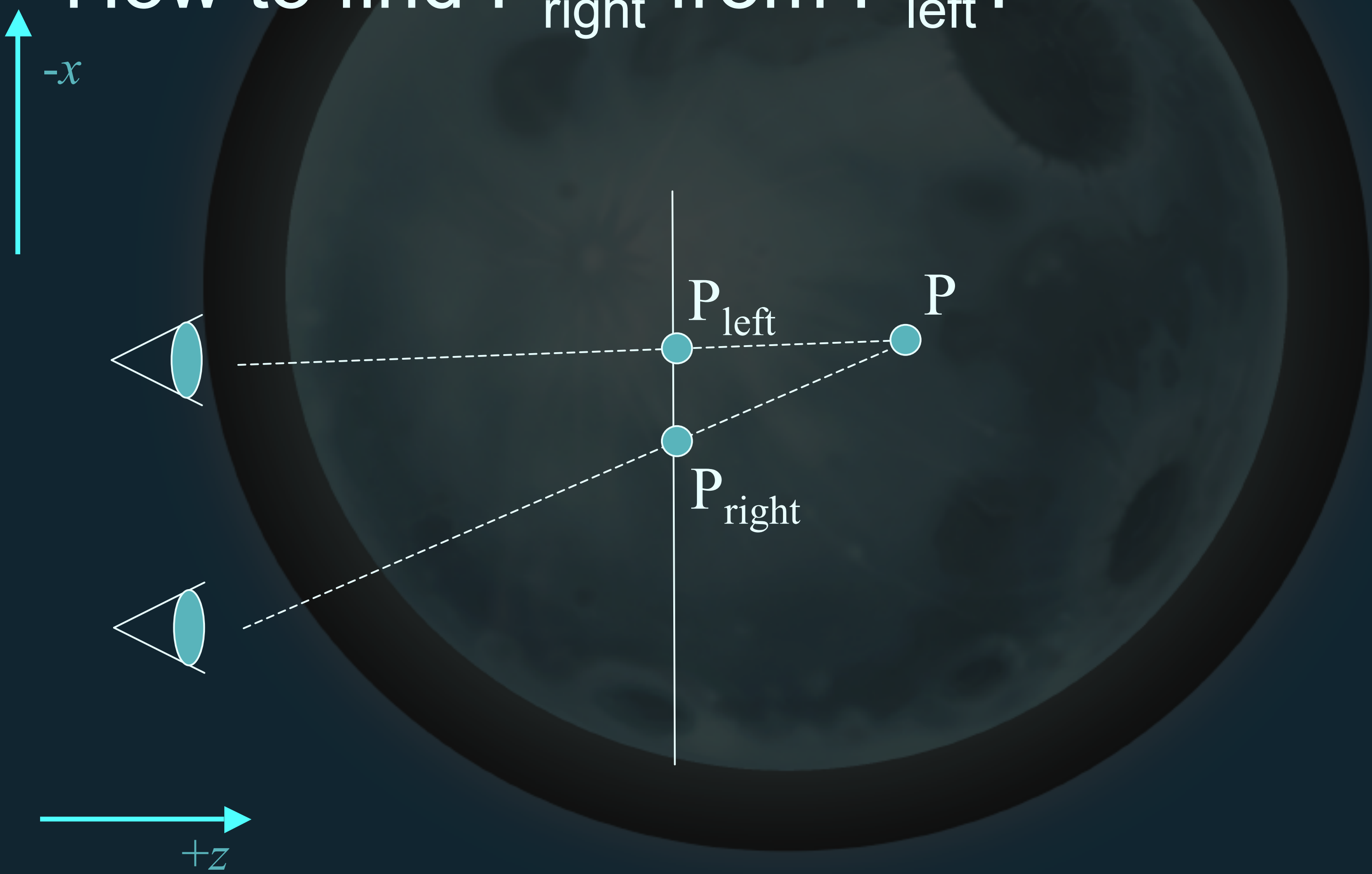
Stereographic Reprojection

Stereographic Reprojection

Stereographic Reprojection

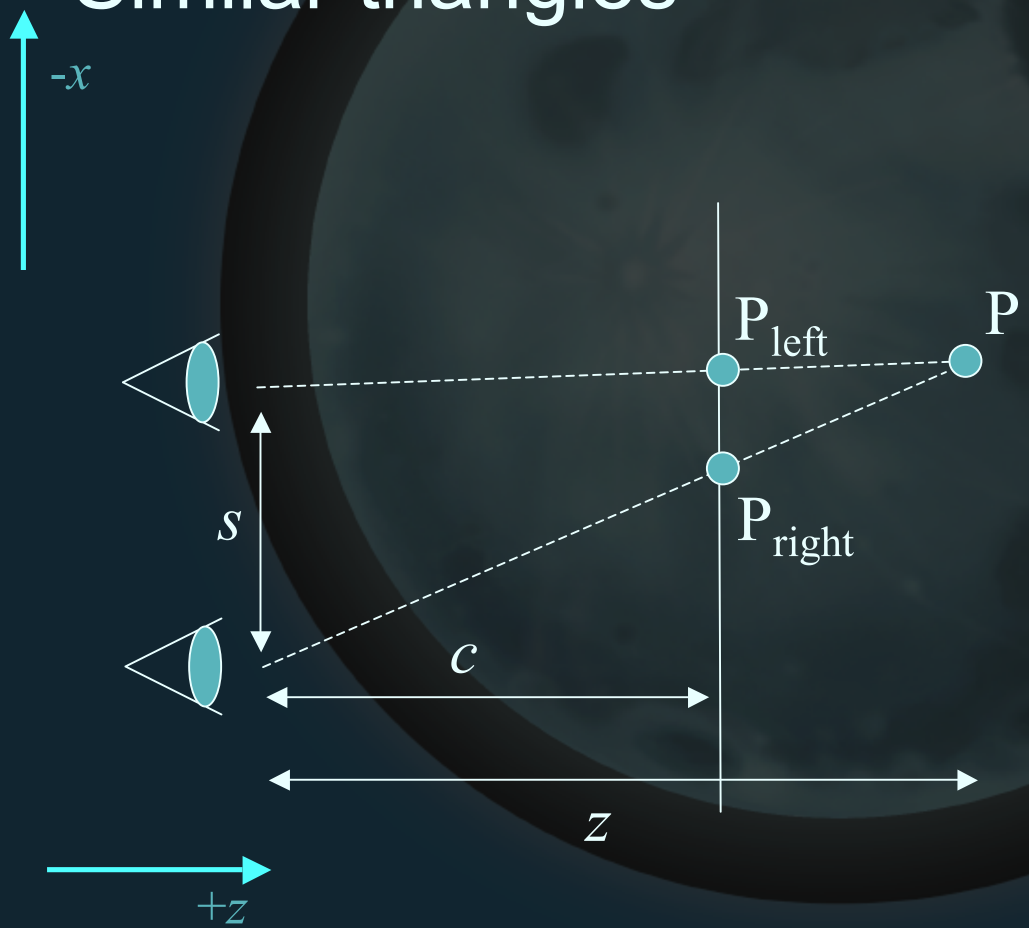
Stereographic Reprojection

- How to find P_{right} from P_{left}?



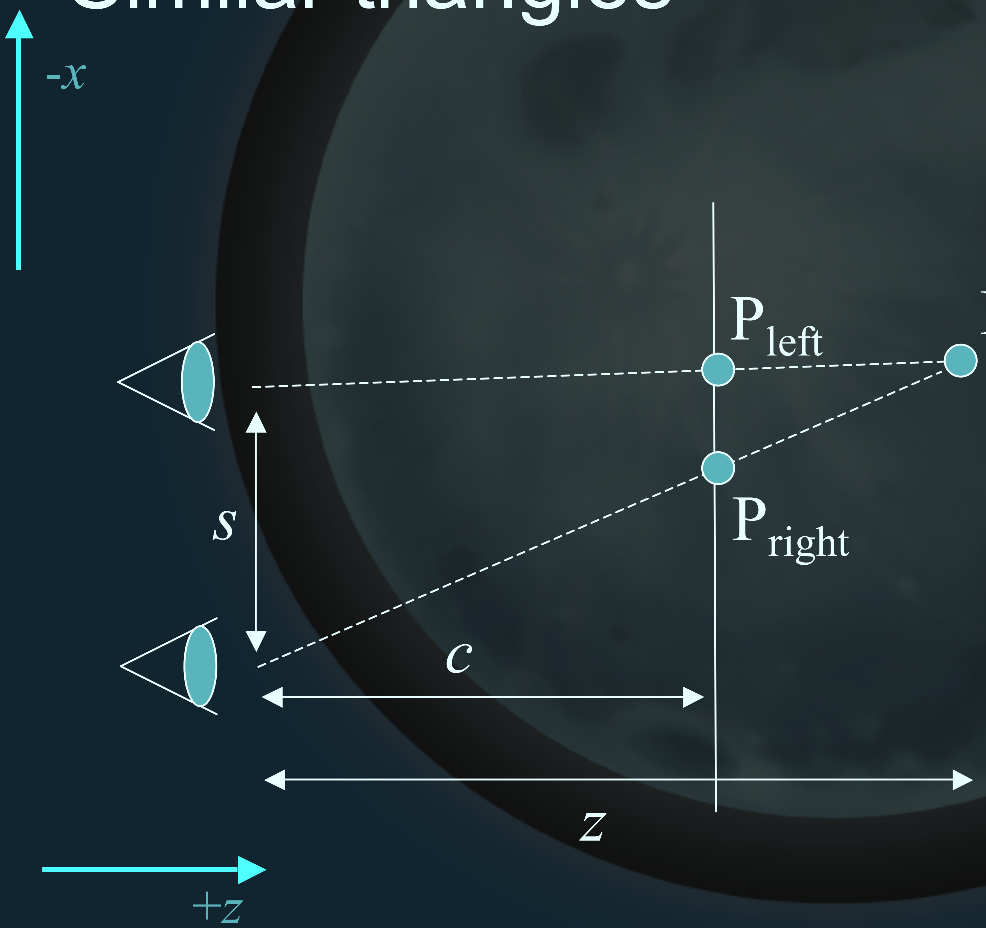
Stereographic Reprojection

- Similar triangles



Stereographic Reprojection

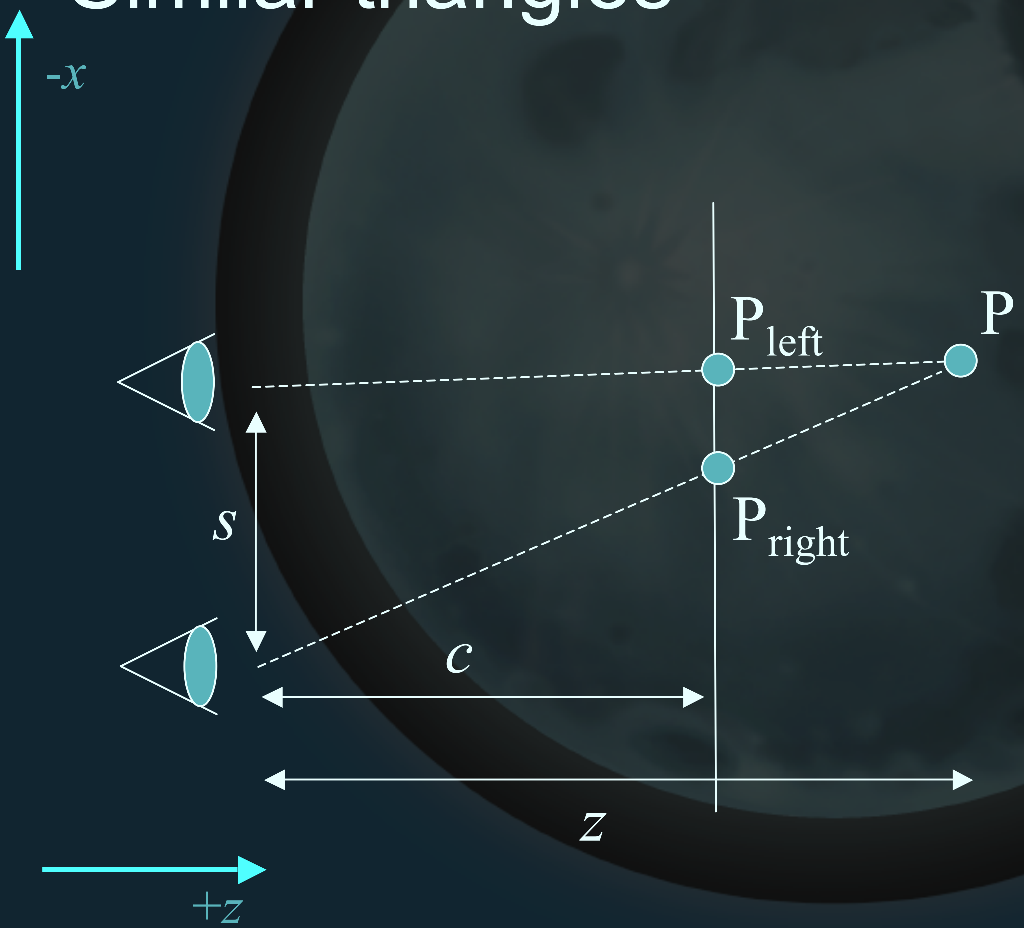
- Similar triangles



$$P_{right} = P_{left} + (s\frac{(z-c)}{z}, 0)$$

Stereographic Reprojection

- Similar triangles



$$P_{right} = P_{left} + (s - \frac{sc}{z}, 0)$$

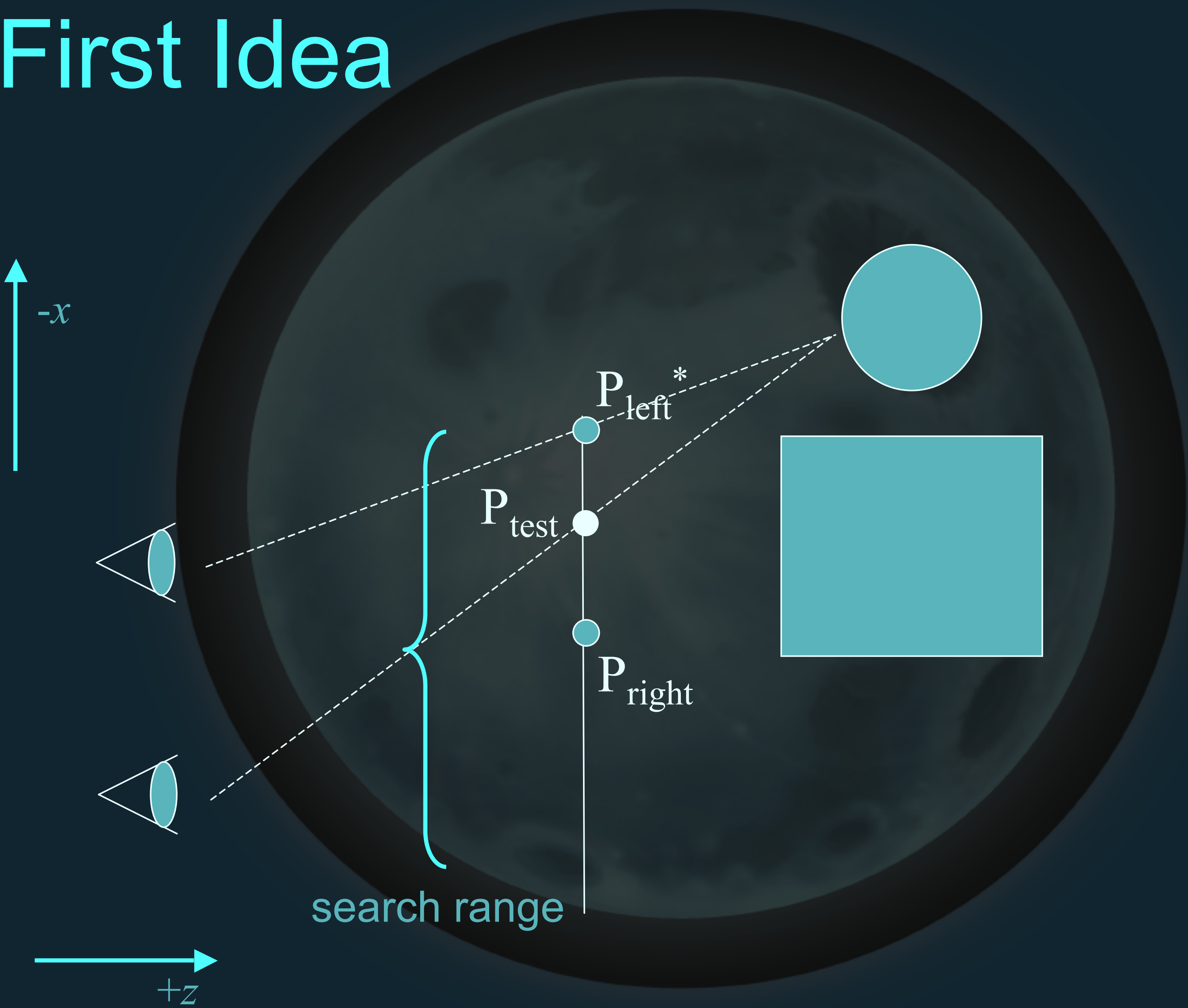
Stereographic Reprojection

- This works if we can scatter:
  - we know P_{left} and z
- But in pixel shader we can only gather
- I.e. we are rendering right image, so only know P_{right}, and not z
- Have to search
- Much like parallax occlusion mapping

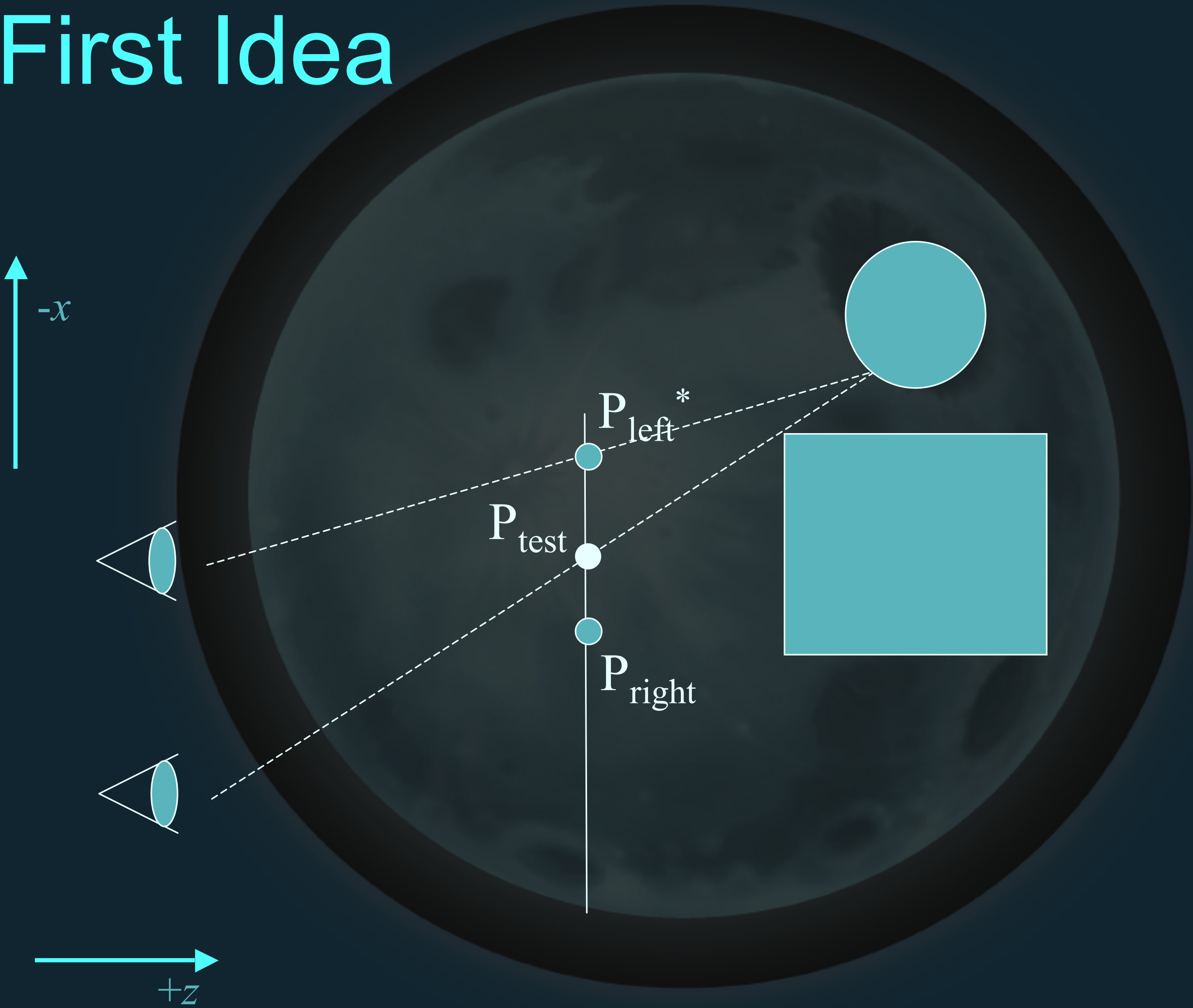
First Idea

- General concept:
  - Iterate along scanline in left-eye image around our texel location P_{right}
  - Reproject from left location P_{left}^* to get potential right location
  - If passes our texel location P_{right}, have hit
  - Use current and previous P_{left}^* texels to set our texel color

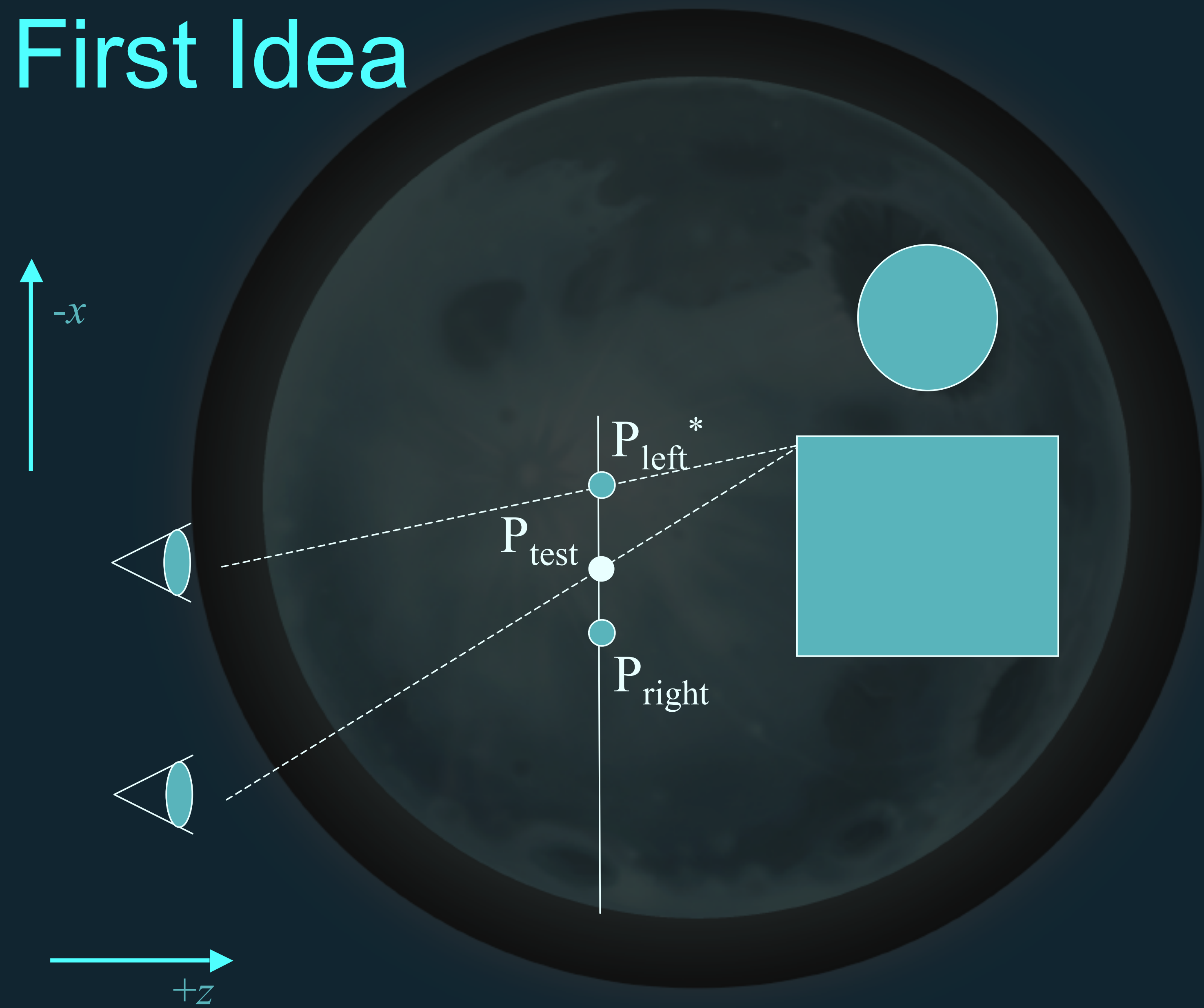
First Idea



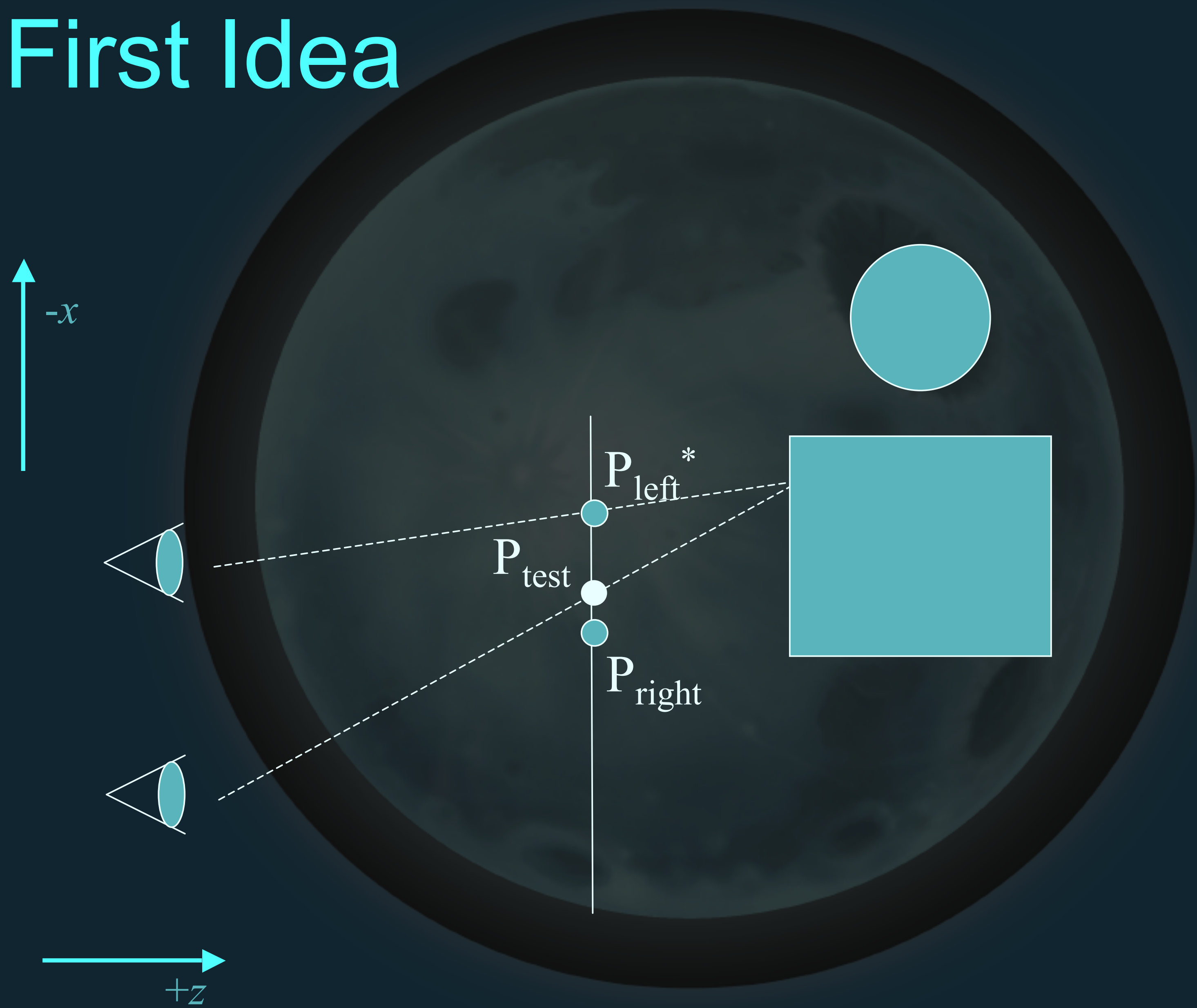
First Idea



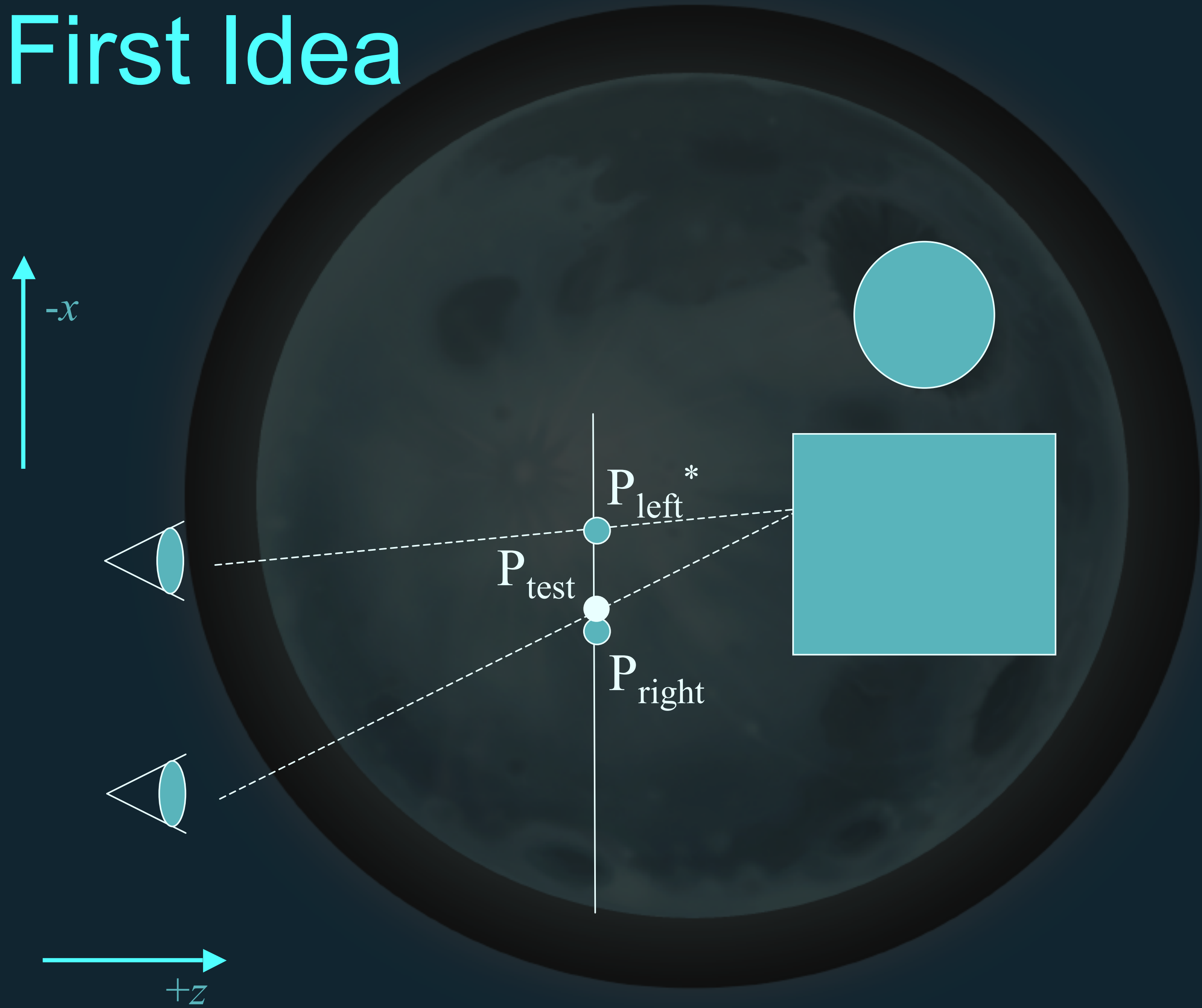
First Idea



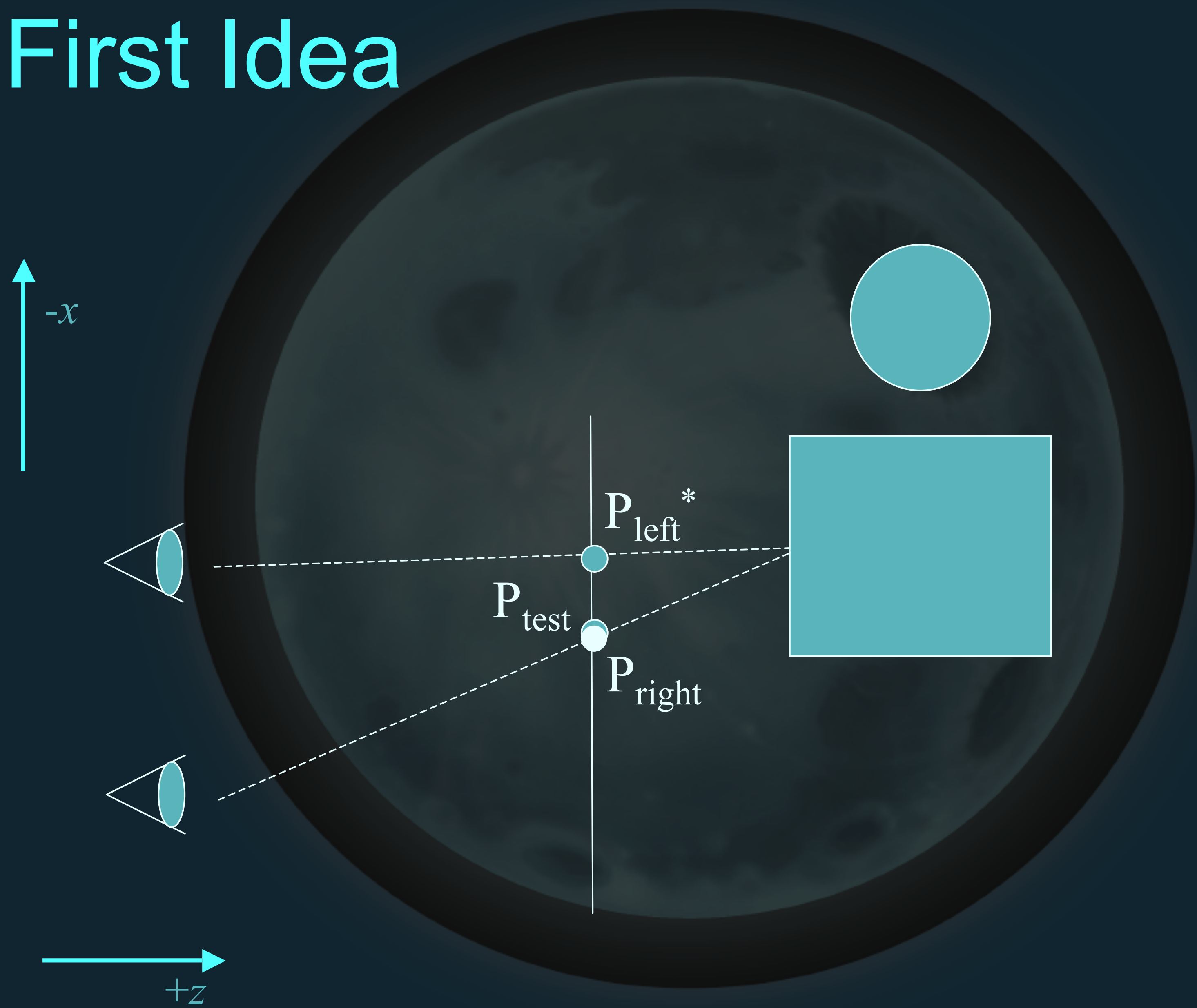
First Idea



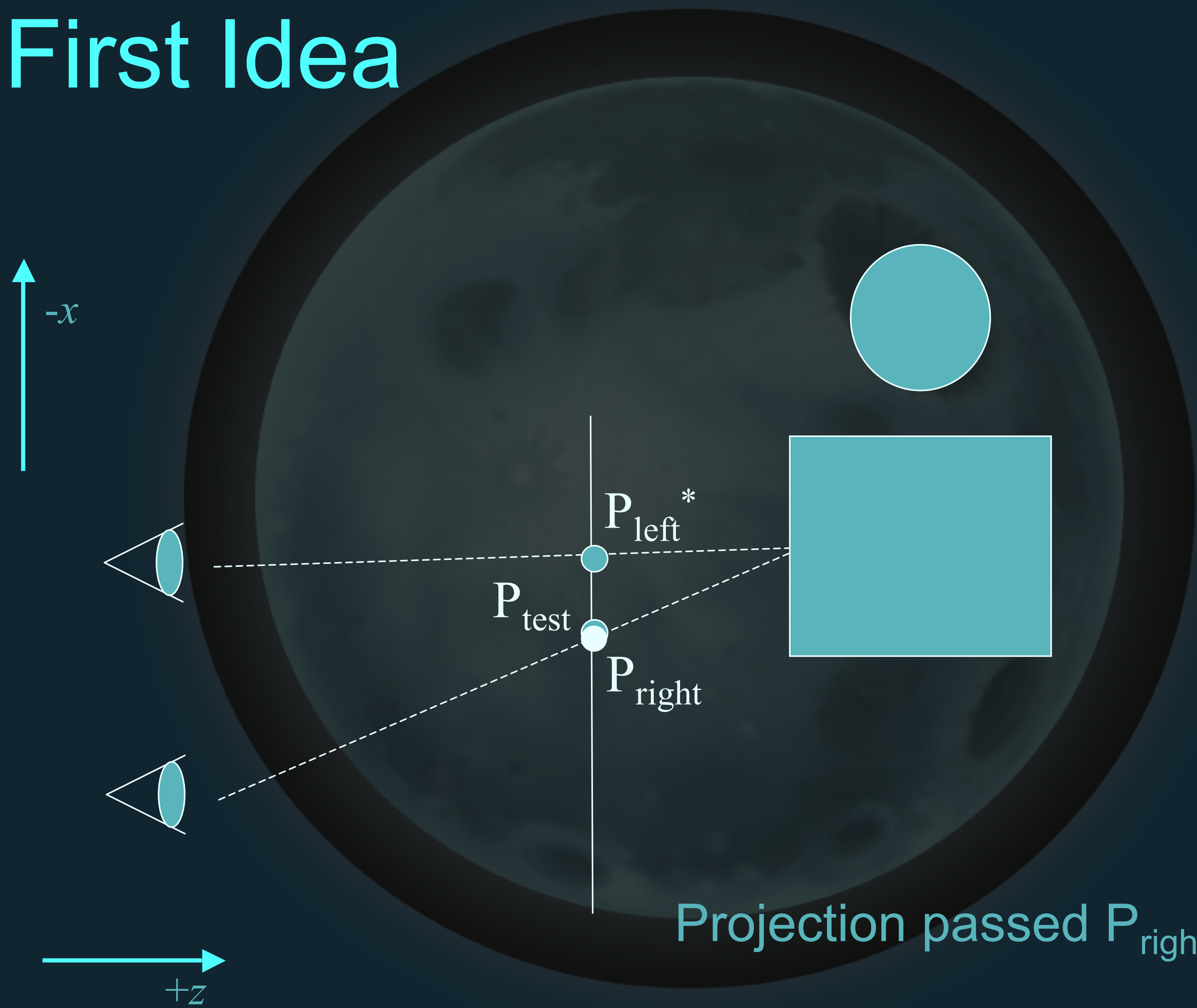
First Idea



First Idea



First Idea



Projection passed P_{right}, Stop

First Idea

- How far to search?
  - Range of texels around P_{right} from $-s$ to s
  - Could search farther before our current texel to capture additional negative parallax
- What color to choose?
  - Lerp by tex coords $test_{prev}$ & $test_{curr}$

$$t = \frac{P_{right} - test_{prev}}{test_{curr} - test_{prev}}$$

First Idea

- Note: need linear depth
- Depth buffer is non-linear
- Can get linear depth via

$$z_{linear} = \frac{1}{z_{ndc} \frac{n - f}{nf} + \frac{1}{n}}$$

- Careful of precision! Ideally use float buffer

First Idea

```
float orig_x = tex_coords.x;
float x_val = orig_x - abs(g_stereo_settings.x) - 1.0f;
float end_val = orig_x + abs(g_stereo_settings.x) + 1.0f;
float prev_x_val = x_val;
float prev_test = orig_x;
while (x_val <= end_val)
{
    tex_coords.x = x_val;
    float l = GetLinearDepth(g_depth_map, tex_coords.xy);
    float test = x_val + g_stereo_settings.x + g_stereo_settings.y / l;
    if (test > orig_x)
    {
          float t = (orig_x - prev_test)/(test-prev_test);
          tex_coords.x = (1-t)*prev_x_val + t*x_val;
          o.m_color = h4tex2D( g_tex2, tex_coords.xy).rgb;
          break;
    }
    else
    {
          prev_x_val = x_val;
          prev_test = test;
          x_val = x_val + 1.0f;
    }
}
return o;
```

First Idea

- Problems
  - As s increases, so does # of texels we search
  - Lerp blend makes for muddy edges, particularly against distant objects
  - Also, completely broken

First Idea - Da Bug

- Left eye standard

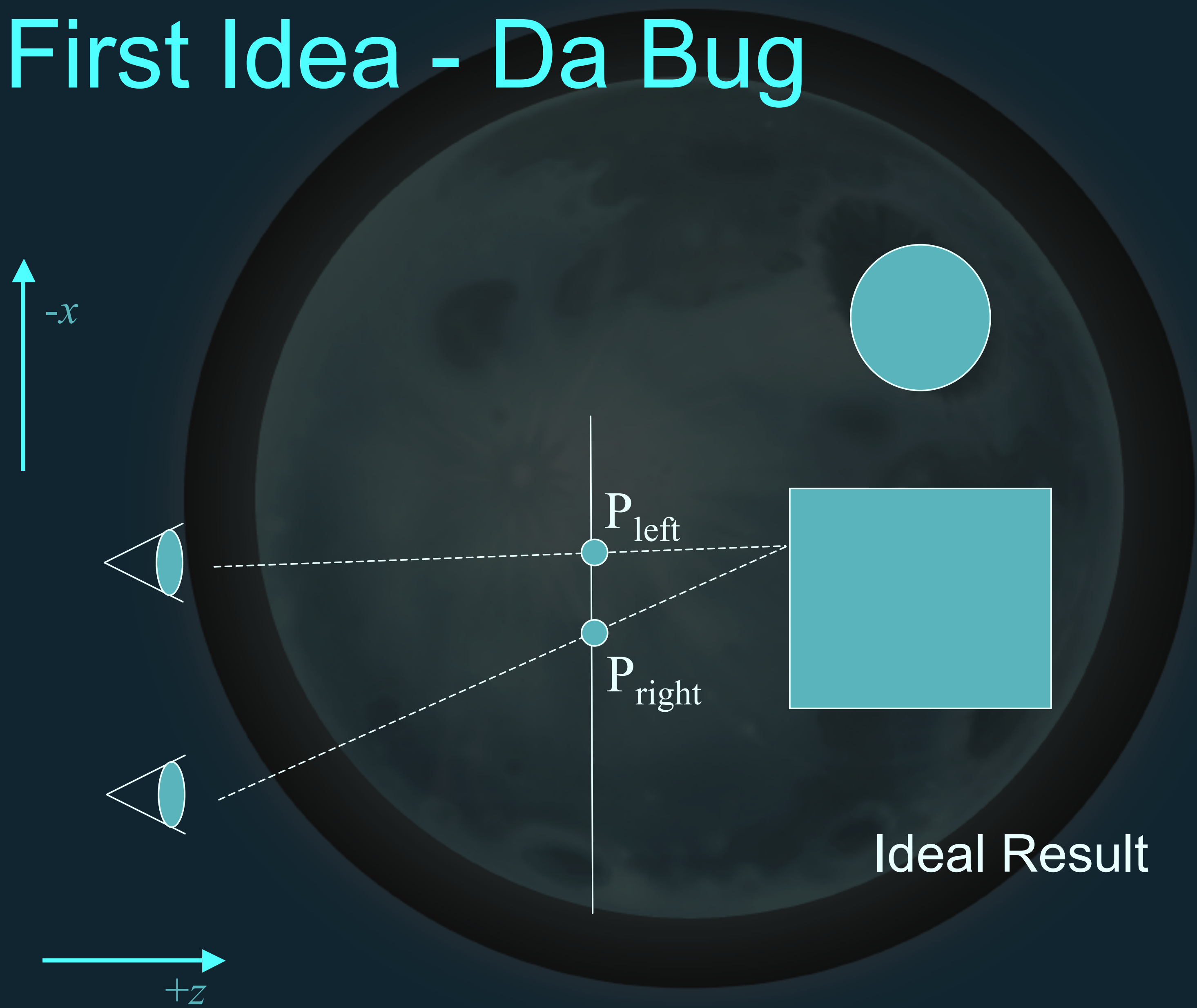
First Idea - Da Bug

- Right eye standard

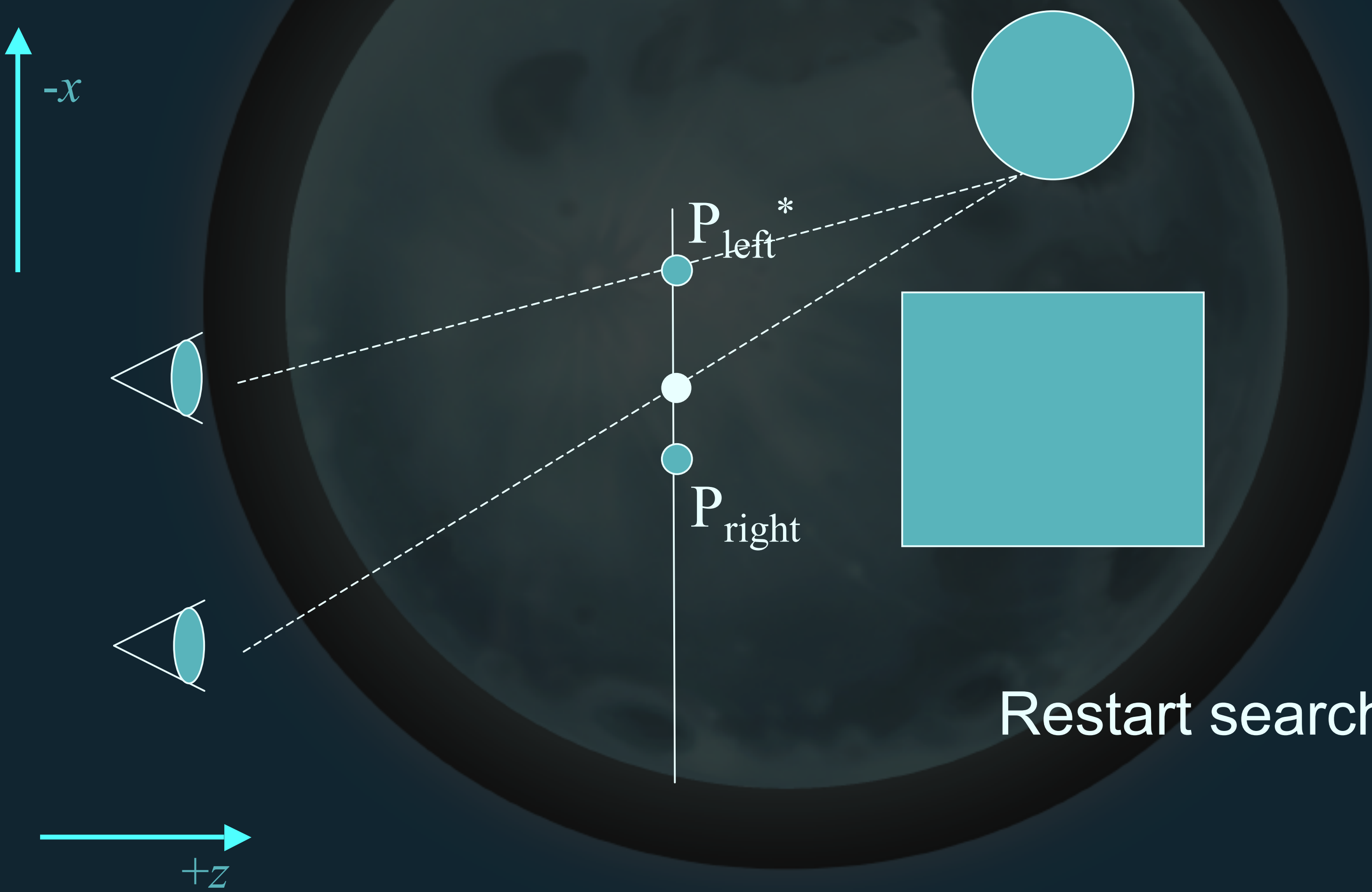
First Idea - Da Bug

- Right eye reprojected

First Idea - Da Bug

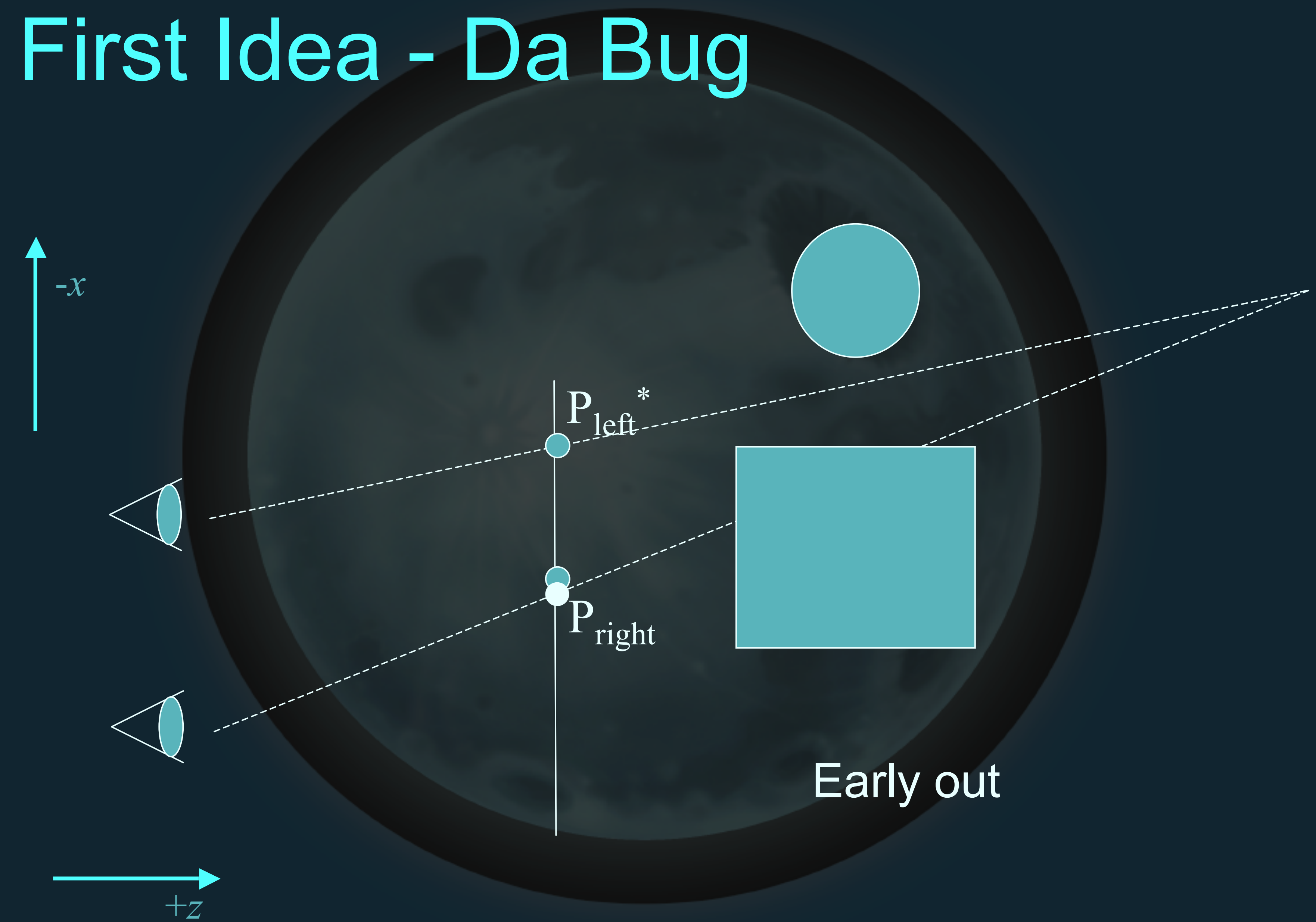


First Idea - Da Bug



Restart search

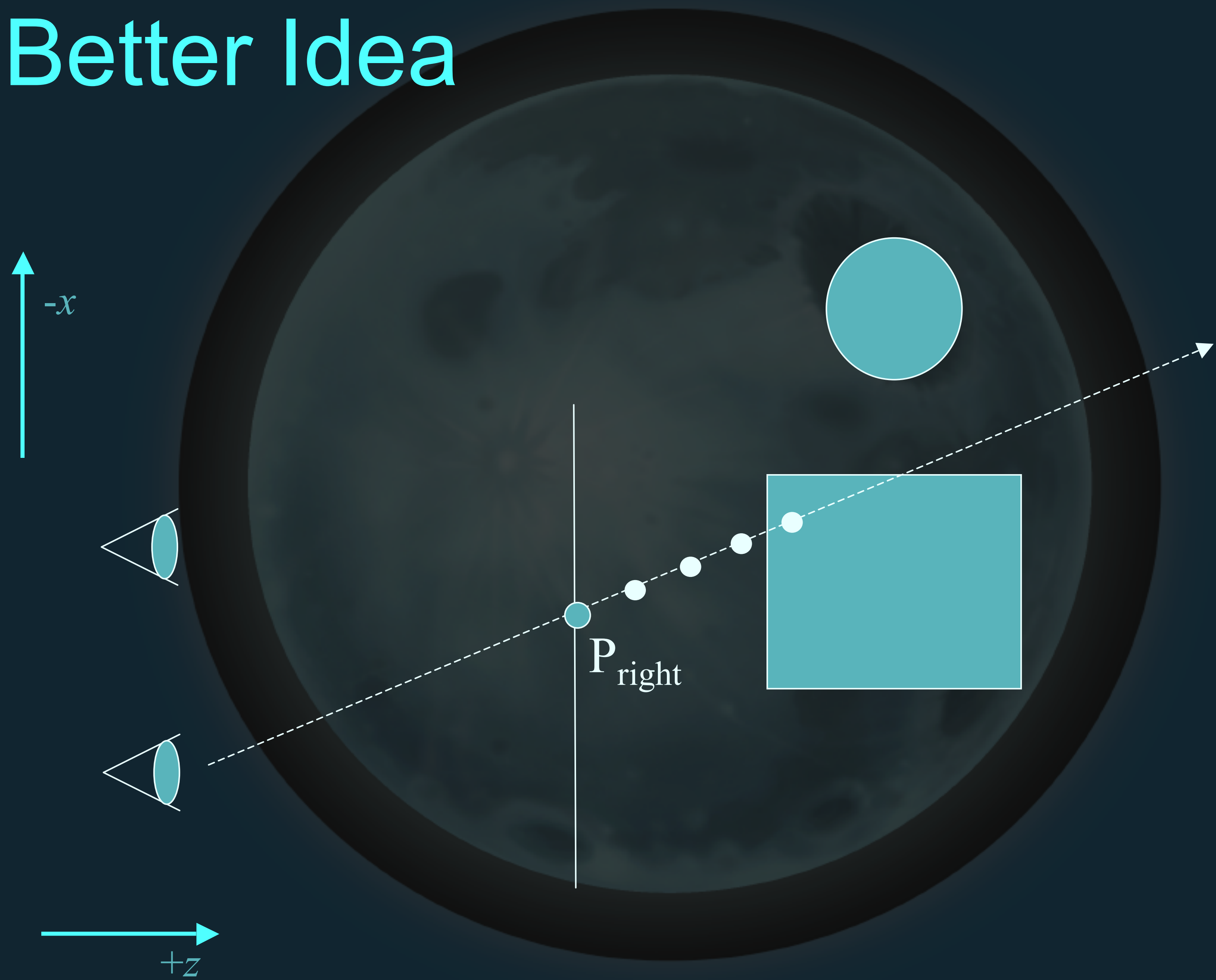
First Idea - Da Bug



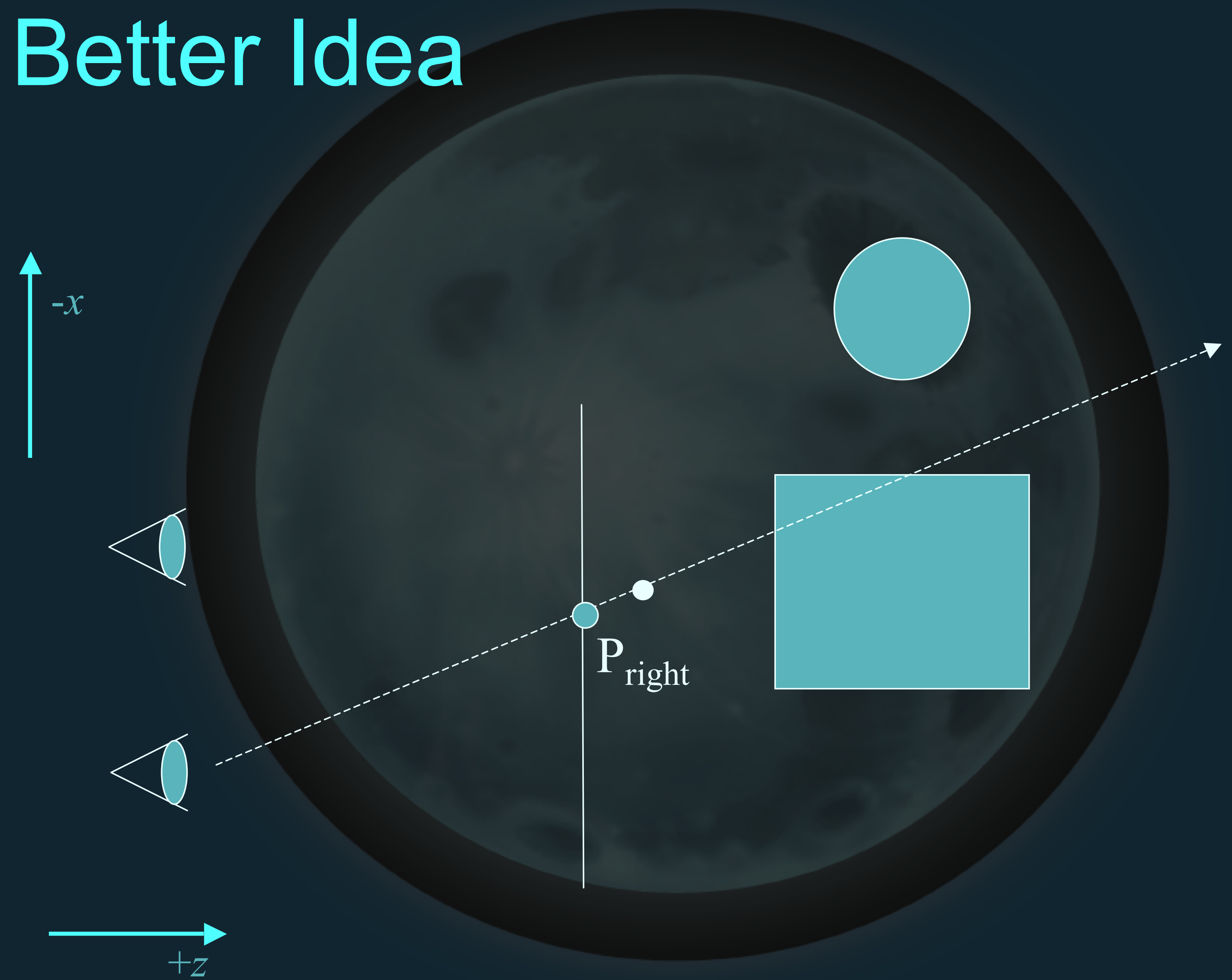
Despair and Salvation

- Could not see solution
  - – Stuck in pixel-search metaphor
- Then: paper by van de Hoef and Zalmstra
- The light dawns - go back to parallax mapping

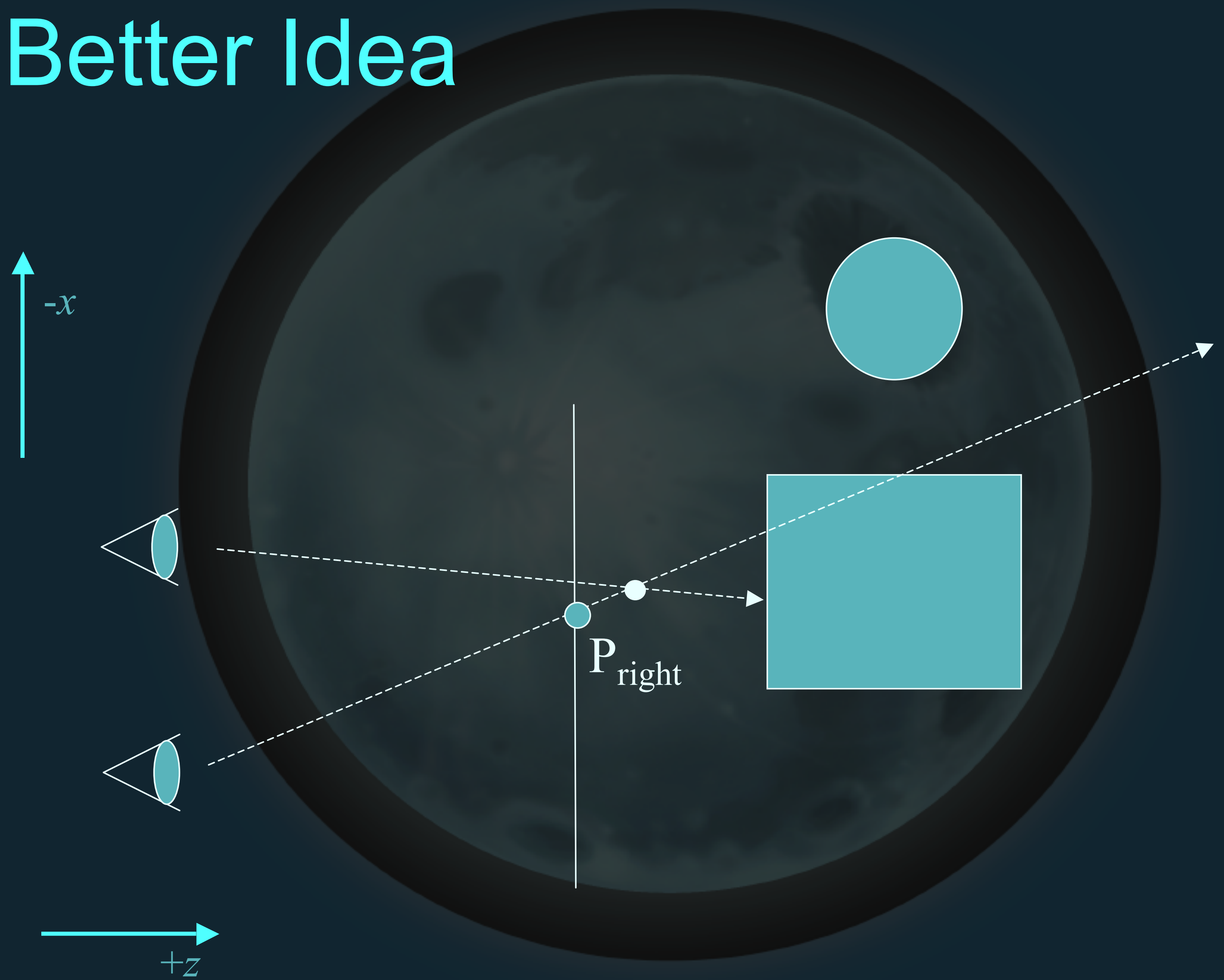
Better Idea



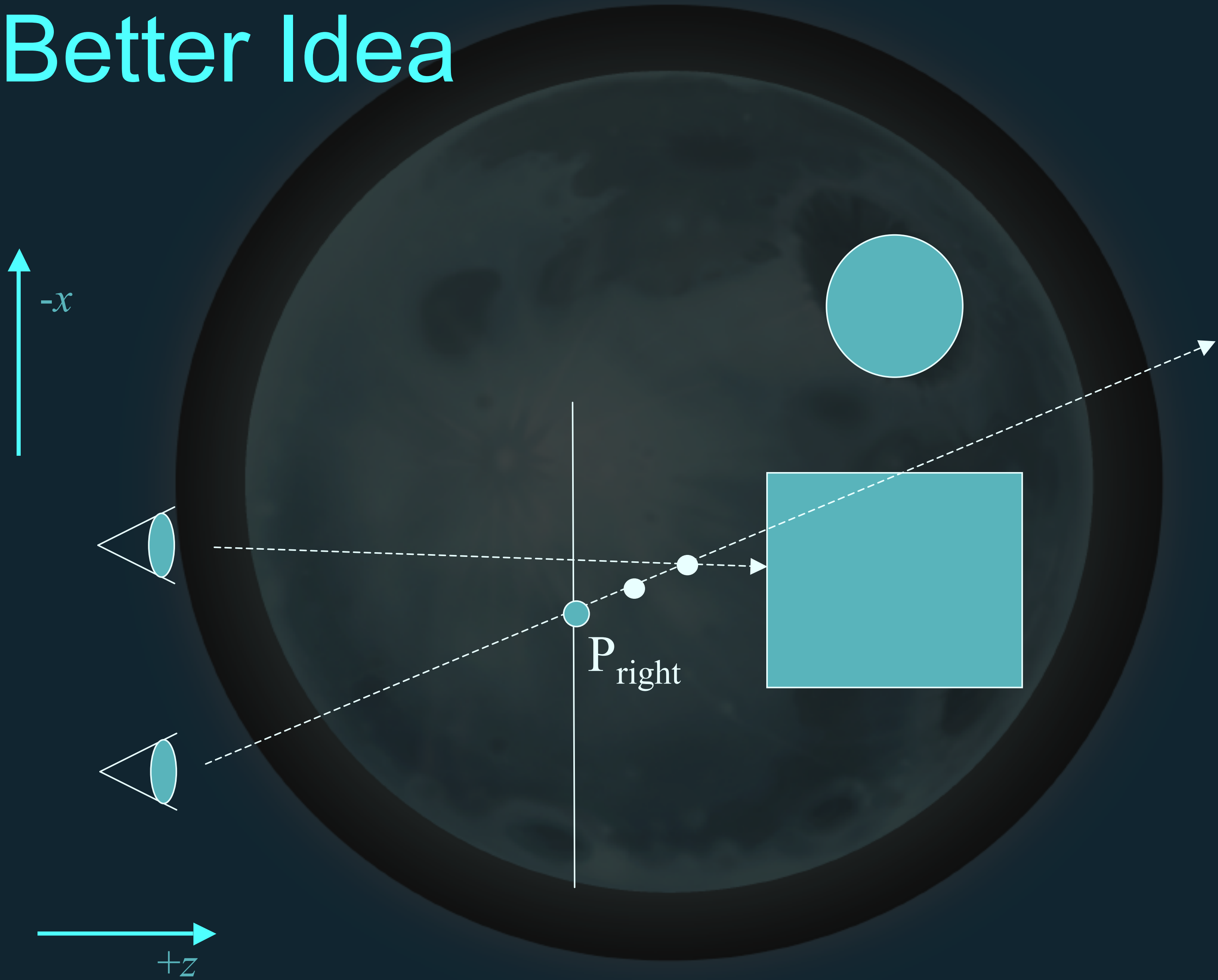
Better Idea



Better Idea

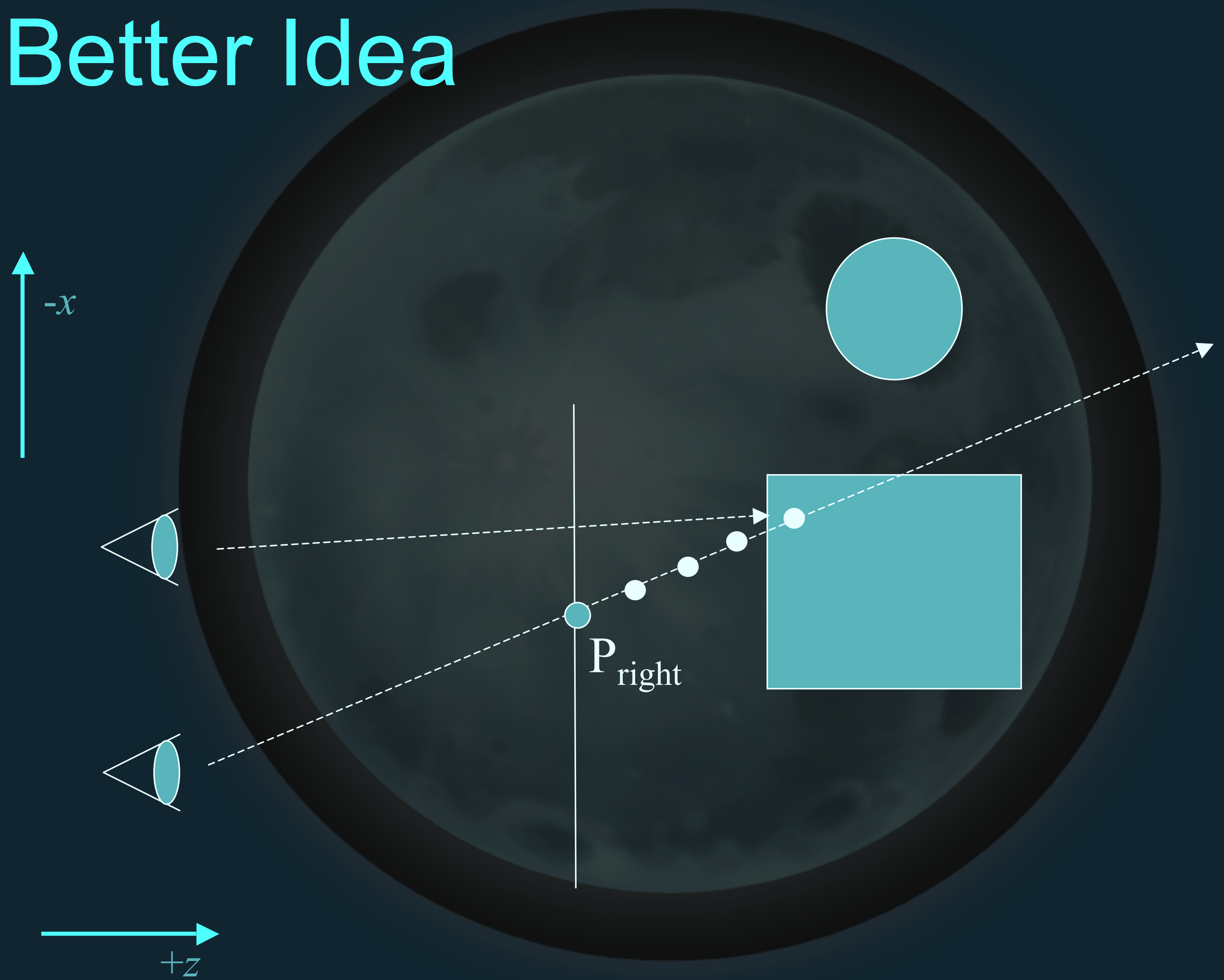


Better Idea

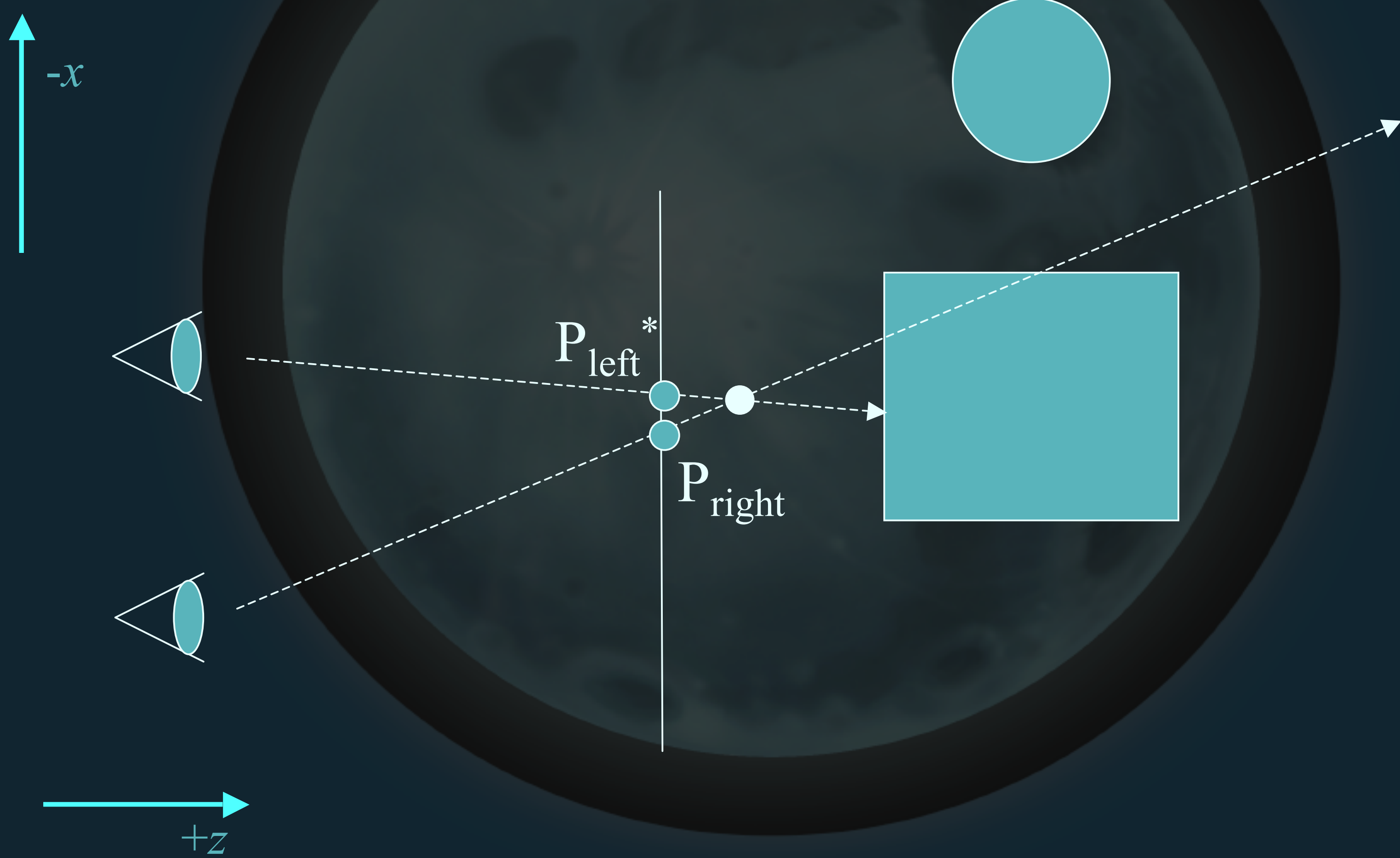


Better Idea

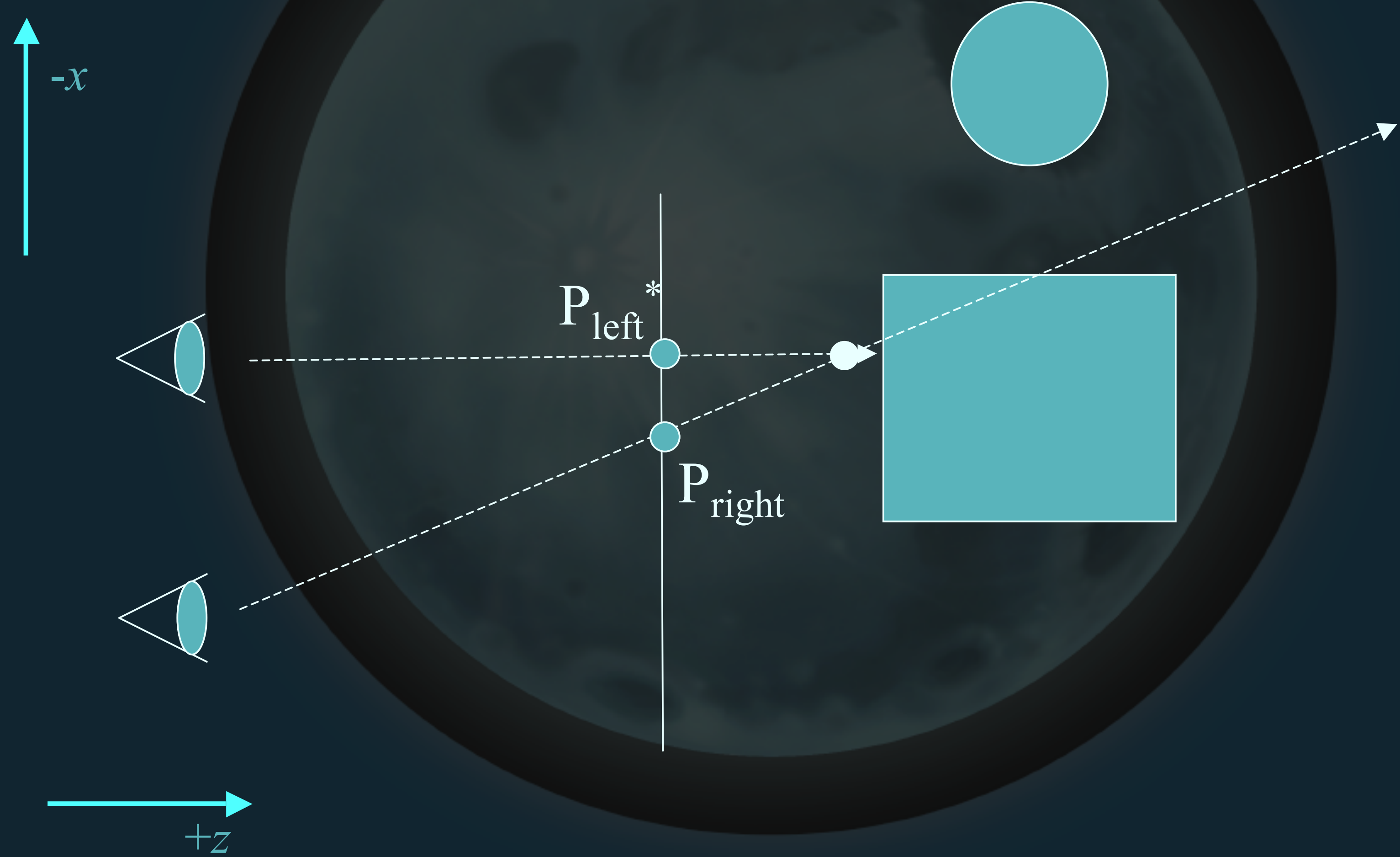
Better Idea



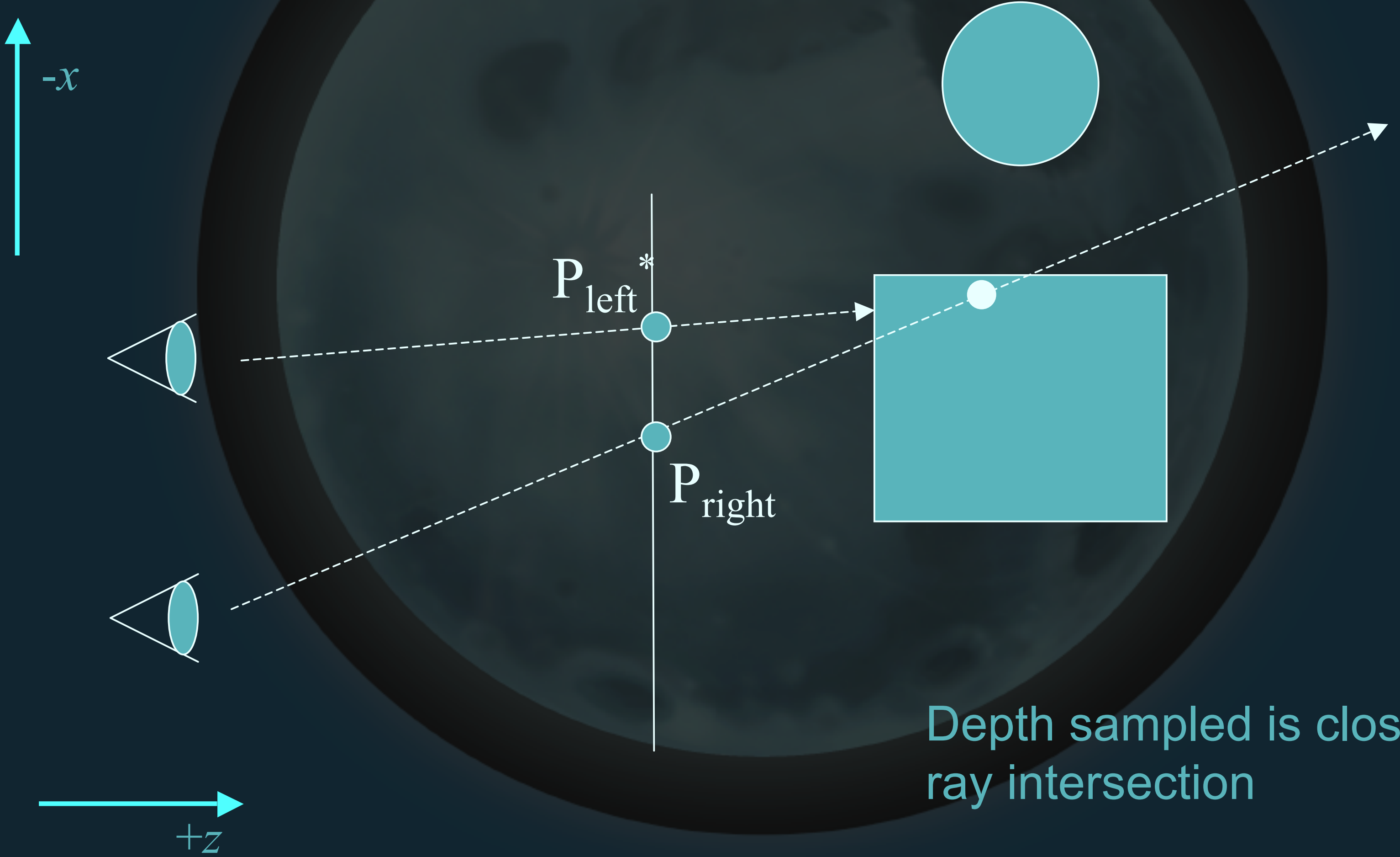
Another way to look at it



Another way to look at it

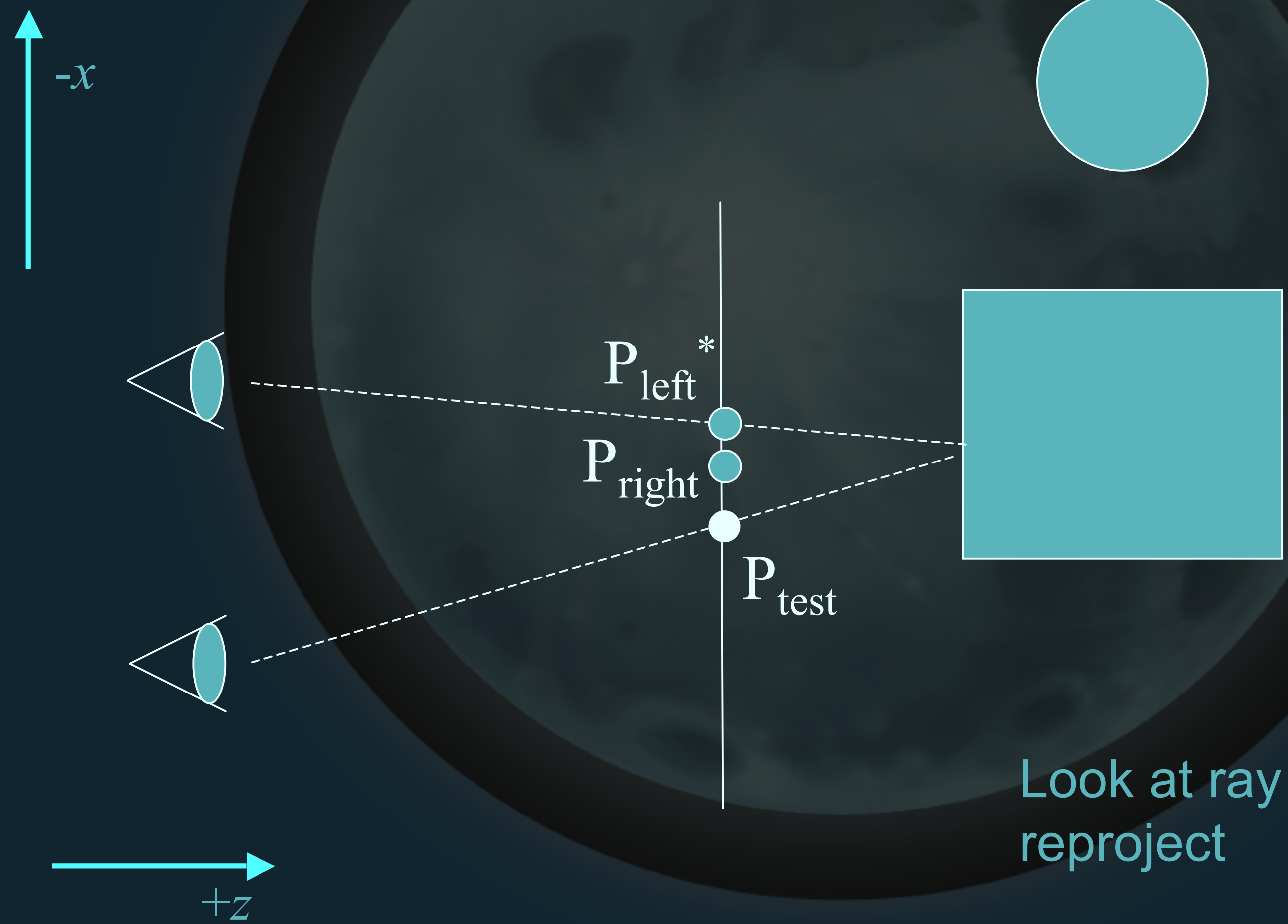


Another way to look at it



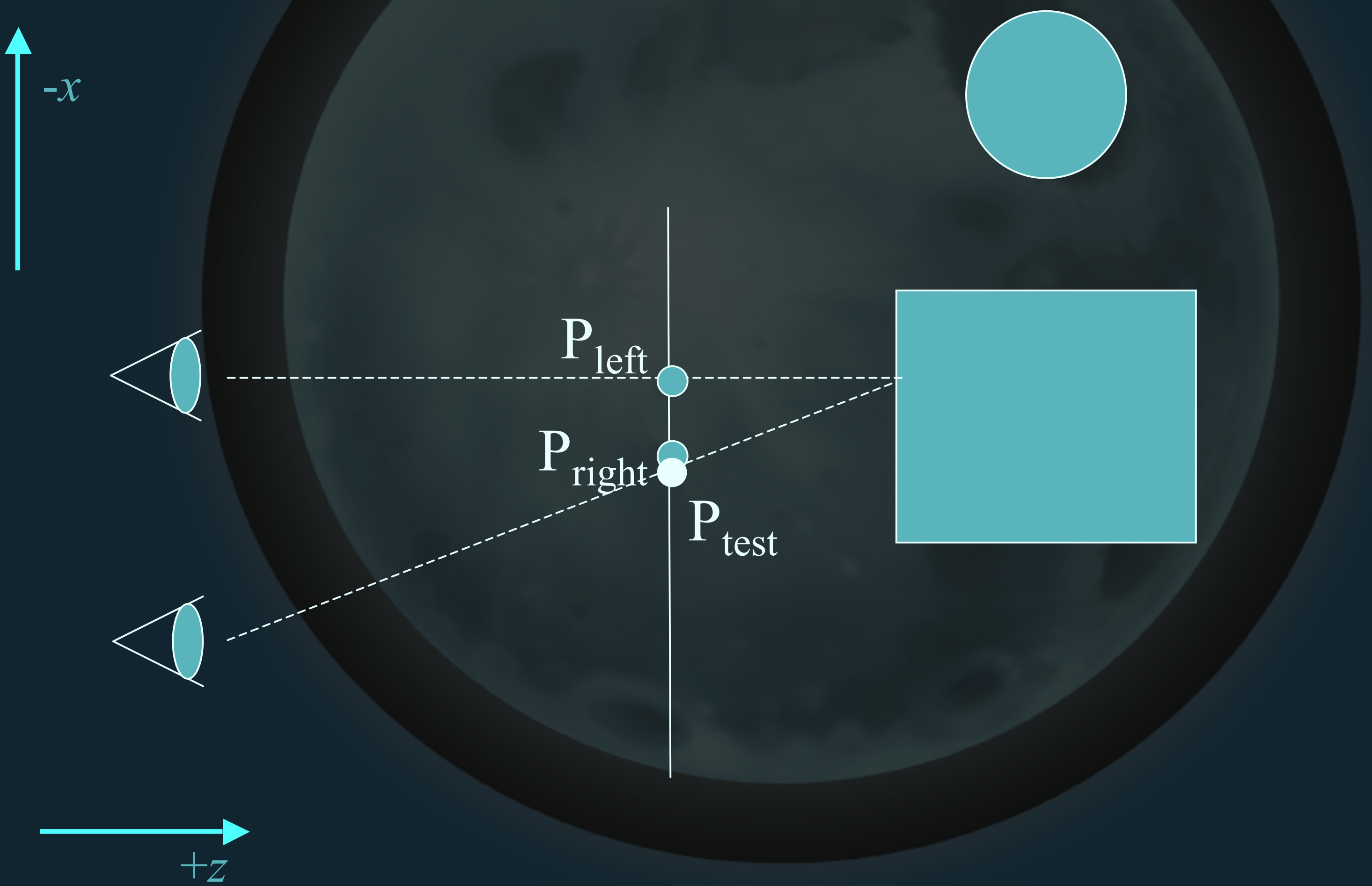
Depth sampled is closer than ray intersection

Yet another way to look at it



Look at ray through P_{left}^* and reproject

Yet another way to look at it



Yet another way to look at it

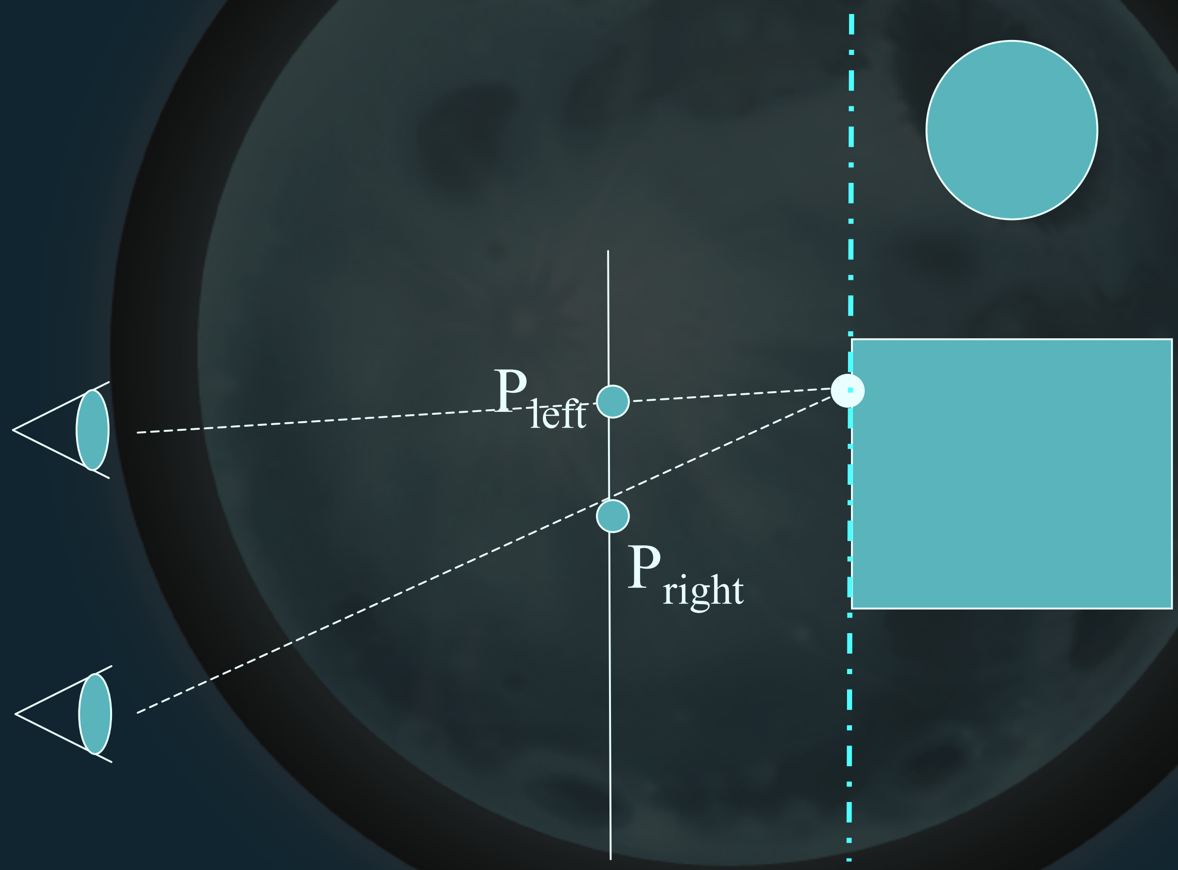
Yet another way to look at it

I.e. search in -x direction!

Better color sampling

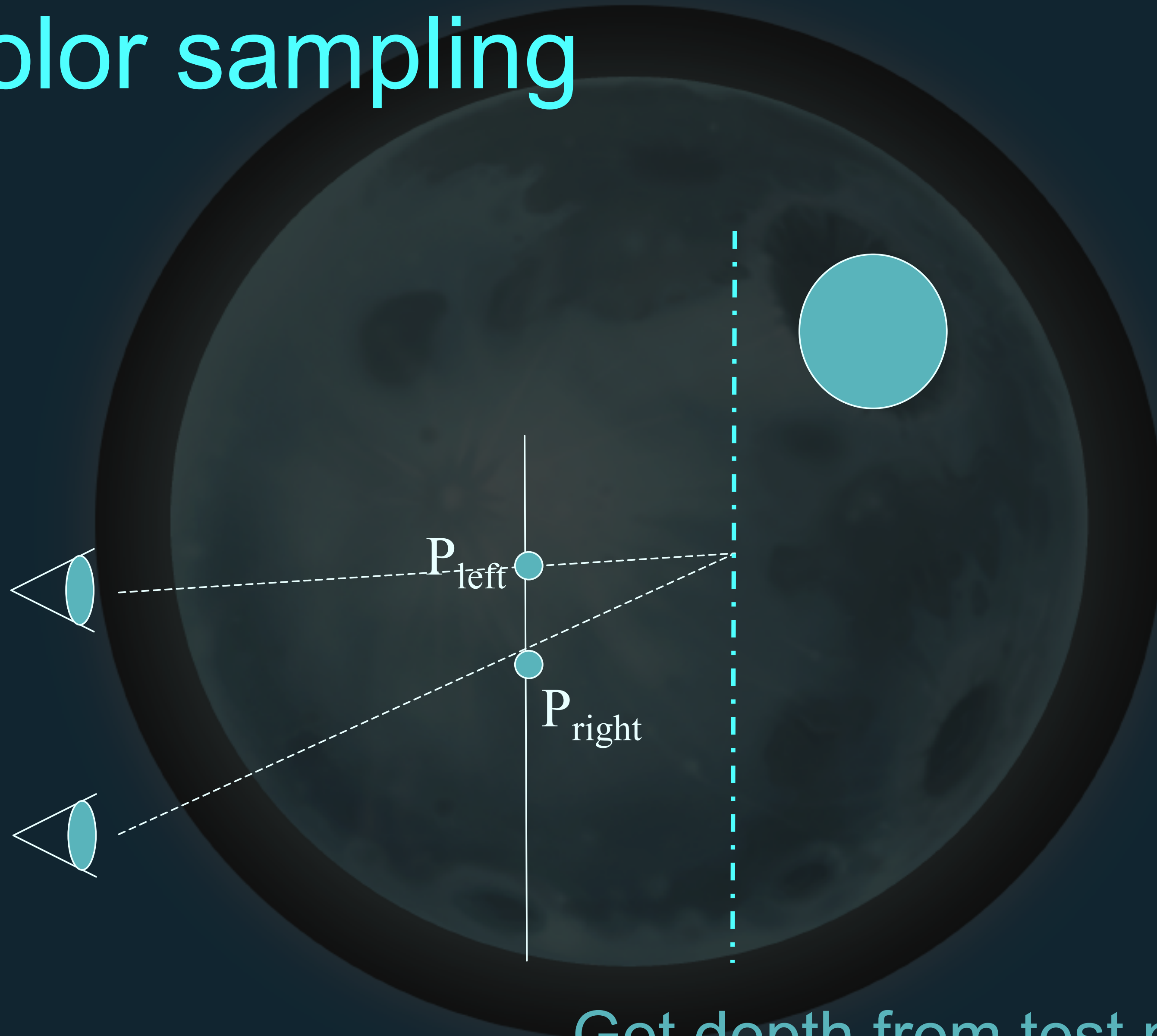
- From van de Hoef and Zalmstra
- Idea: use depth at left eye test point to reproject from right to left to get color

Color sampling



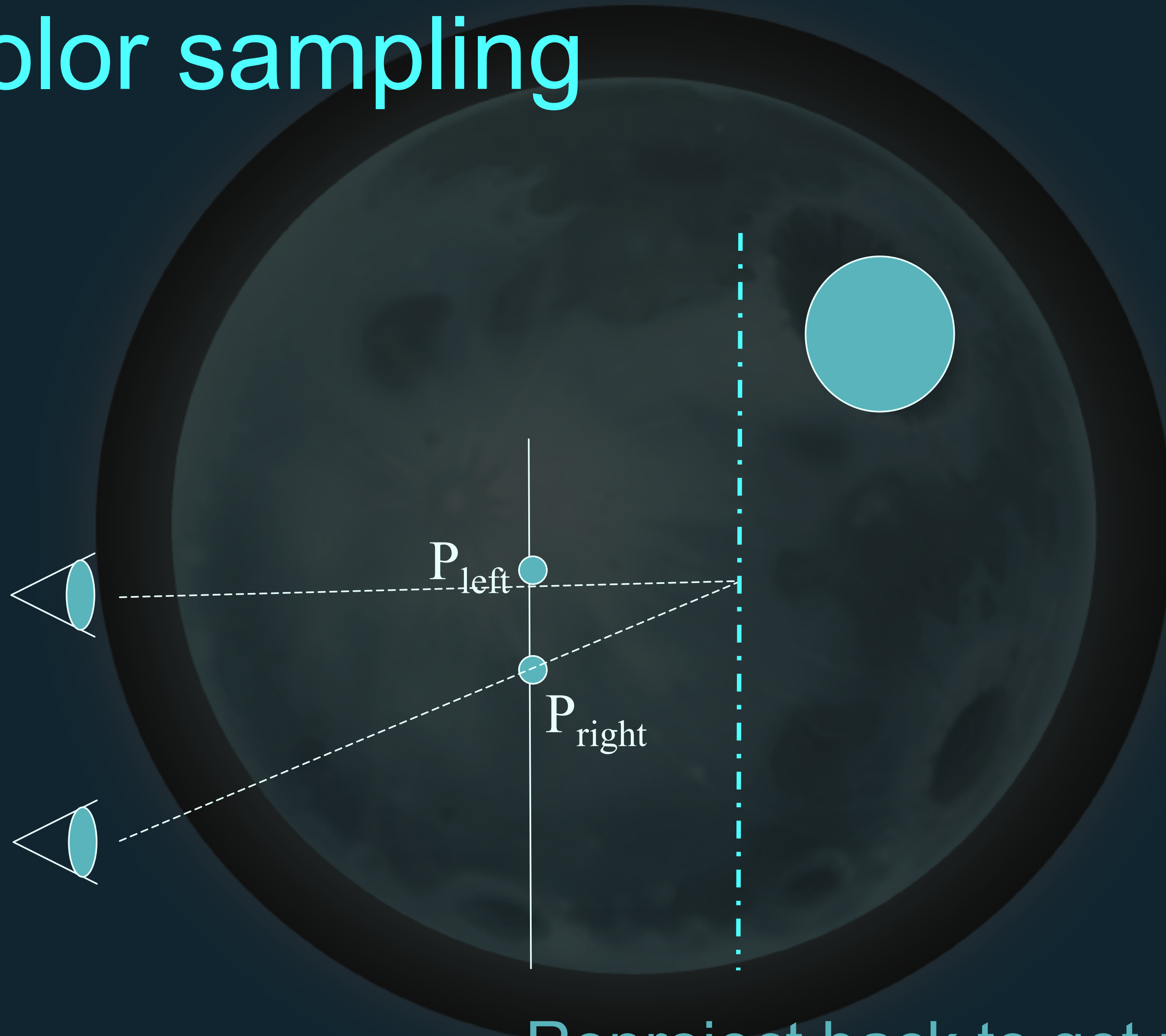
Get depth from test point

Color sampling



Get depth from test point

Color sampling



Reproject back to get new uv

Better Iterations

- Original method: dependant on *s*
- van de Hoef and Zalmstra: one sample!
- Unhappy with results
- Went back to parallax mapping idea:
  - fixed # samples at intervals

Reprojection, take 2

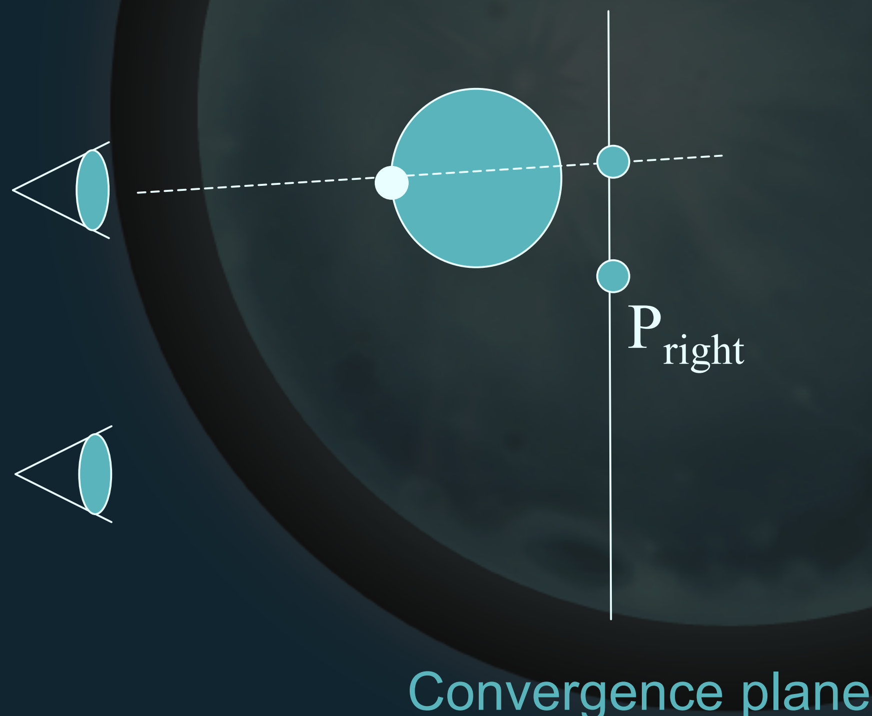
```
float s = sign(stereo_params.y);
float orig_x = tex_coords.x;
float shift = 0.5f* g_stereo_settings.x;
float end_shift = -g_stereo_settings.x;
float step = (end_shift - shift)*0.0625f;

int index = 0;
while (index <= 16)
{
    tex_coords.x = orig_x + s+ shift;
    float l = GetLinearDepth(g_depth_map, tex_coords.xy);
    float test = shift + g_stereo_settings.x + g_stereo_settings.y / l;
    if (s*test >= 0.0f)
    {
          out_tex_coords.x = orig_x - g_stereo_settings.x - g_stereo_settings.y / l;
          break;
    }
    shift = shift + step; index = index + 1;
}

o.m_color = h4tex2D( g_color_map, out_tex_coords.xy );
float lin = GetLinearDepth( g_depth_map, out_tex_coords.xy );
o.m_depth = g_misc_consts.x = g_misc_consts.y / l;
return o;
```

Reprojection, take 2

- Faster (at most 16 samples)
- More accurate
- Less blurry
- But still:
  - Artifacts
  - Alpha-blended geometry painted on

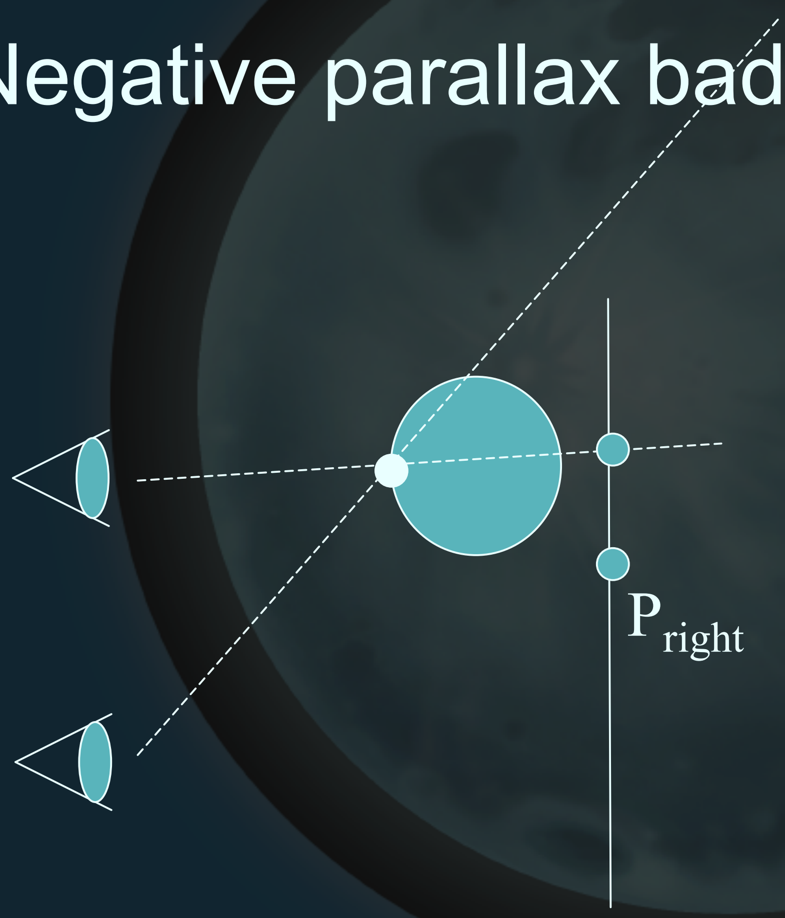
Artifacts

- Negative parallax bad



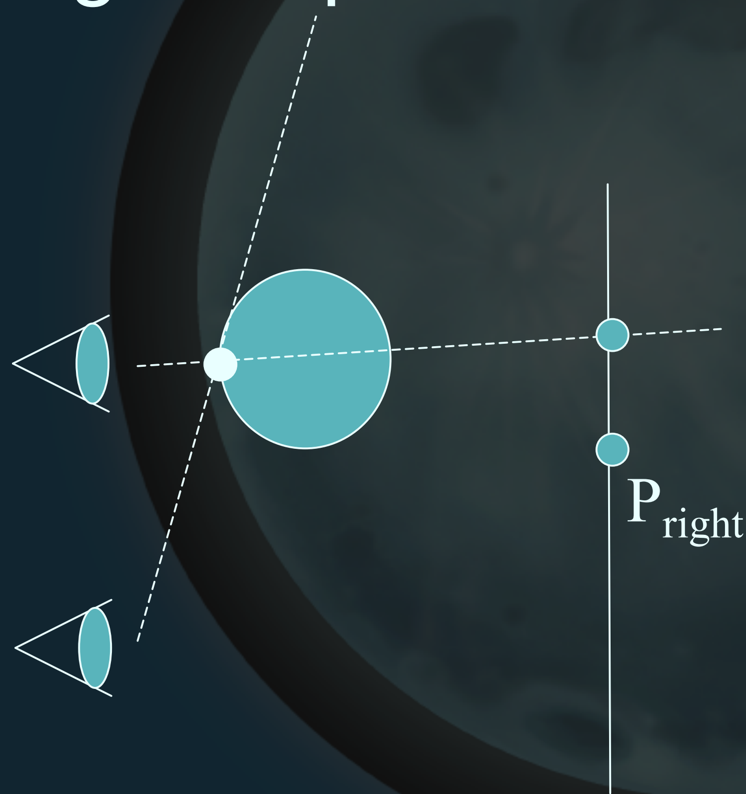
Artifacts

- Negative parallax bad



Artifacts

- Negative parallax bad



Artifacts: Parallax

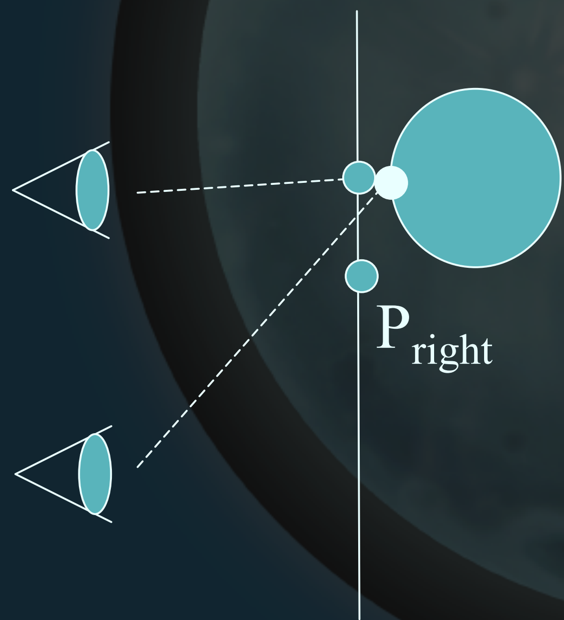
- Standard view

Artifacts

- Right-to-left reprojection

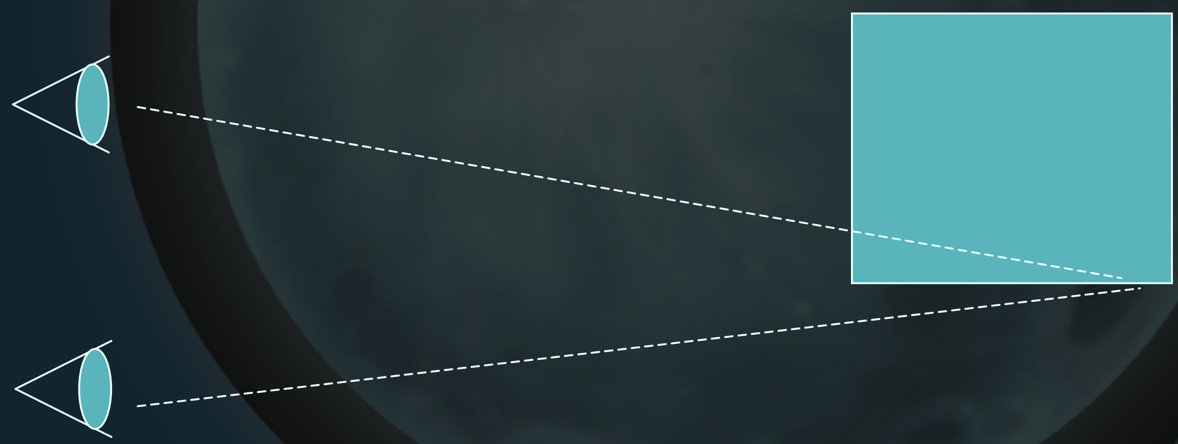
Artifacts

- Negative parallax
  - Avoid it, bring in convergence plane



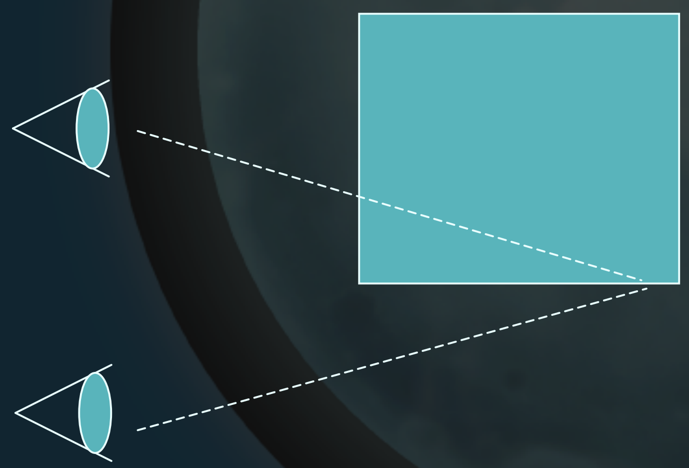
Artifacts

- Left eye can’t see side that right eye can
- No data in left image
- What to fill with? (Reconstruction)



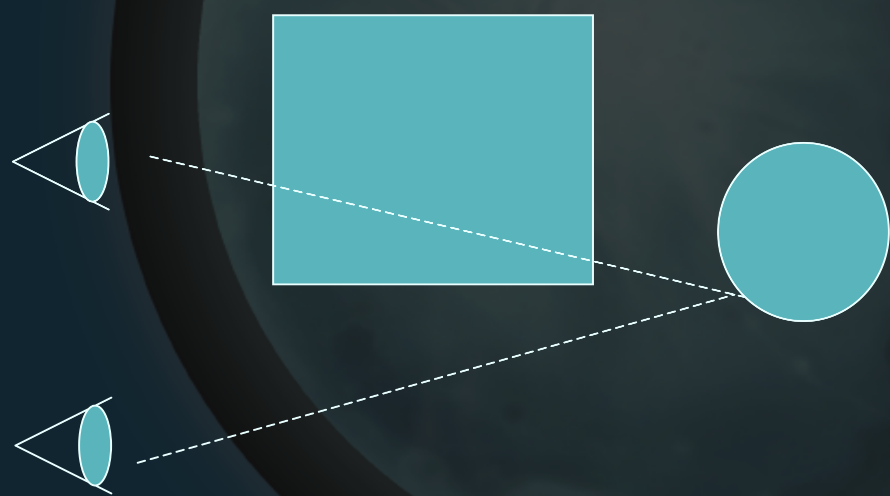
Artifacts

- Worse as object gets closer



Artifacts

- Similar issue when view is obscured



Artifacts

- Reconstruction issue
  - Repeat background or foreground
  - Can also bring in convergence plane
  - Use center reprojection to minimize it

Artifacts: Reconstruction

- Standard left view

Artifacts: Reconstruction

- Right to left reprojection

Artifacts: Reconstruction

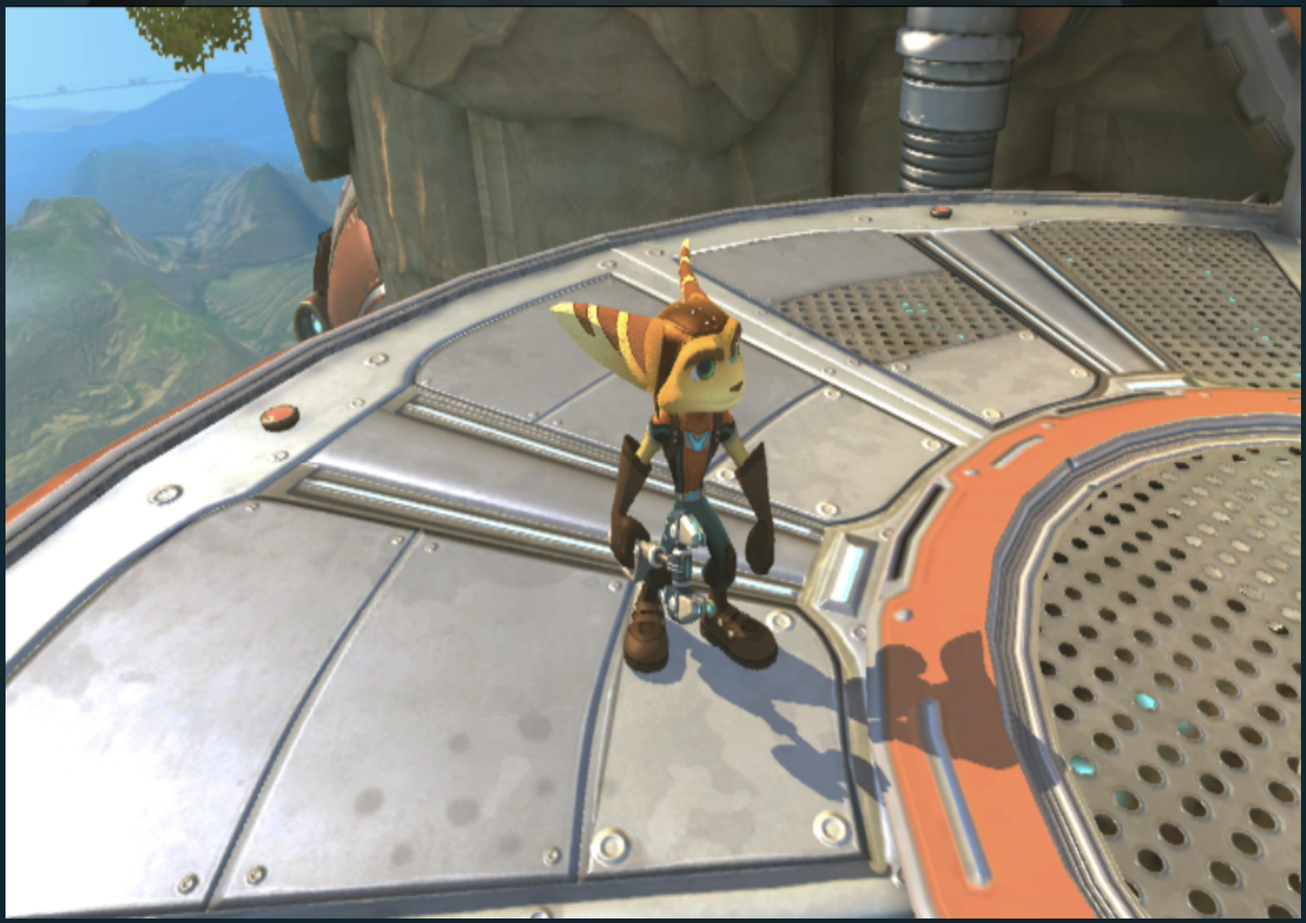
- Standard right view

Center Reprojection

- Render standard monoscopic view
- Reproject to both left and right
- Twice as long as left-to-right (or right-to-left) but better quality

Center Reprojection

- Standard projection

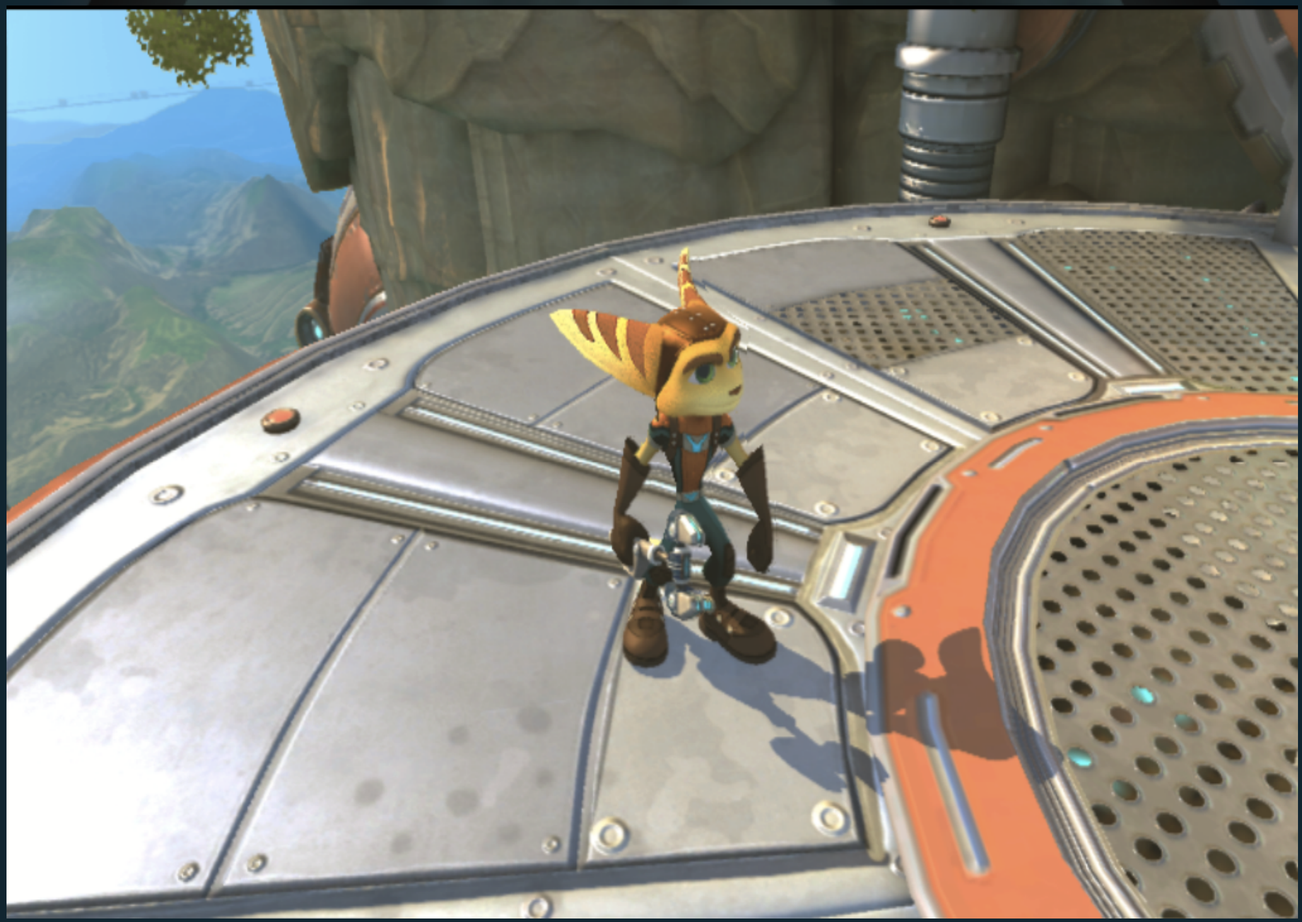


Center Reprojection

- Left-to-right reprojection

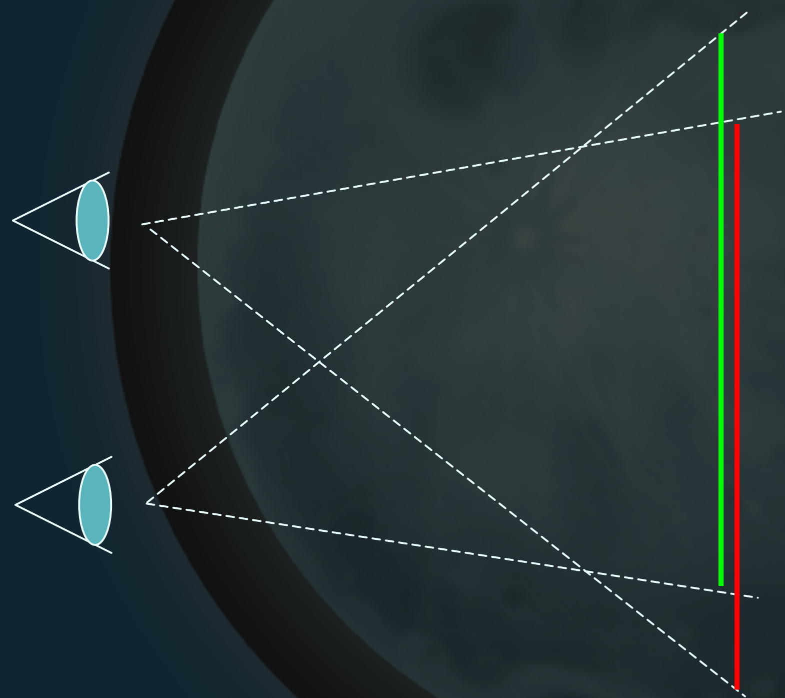
Center Reprojection

- Center reprojection



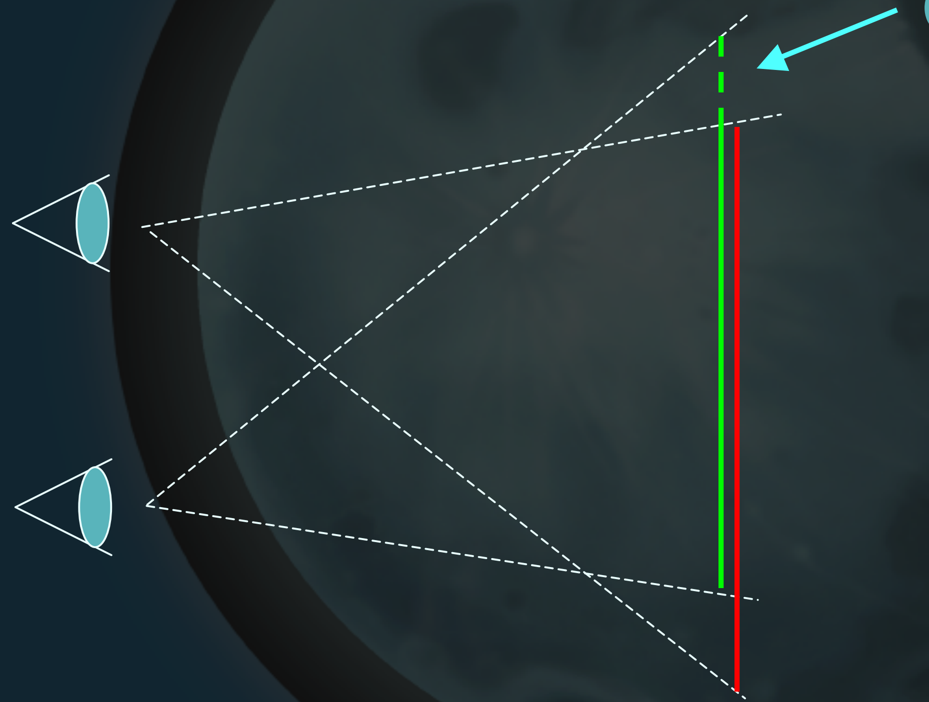
Artifacts

- Viewport edge



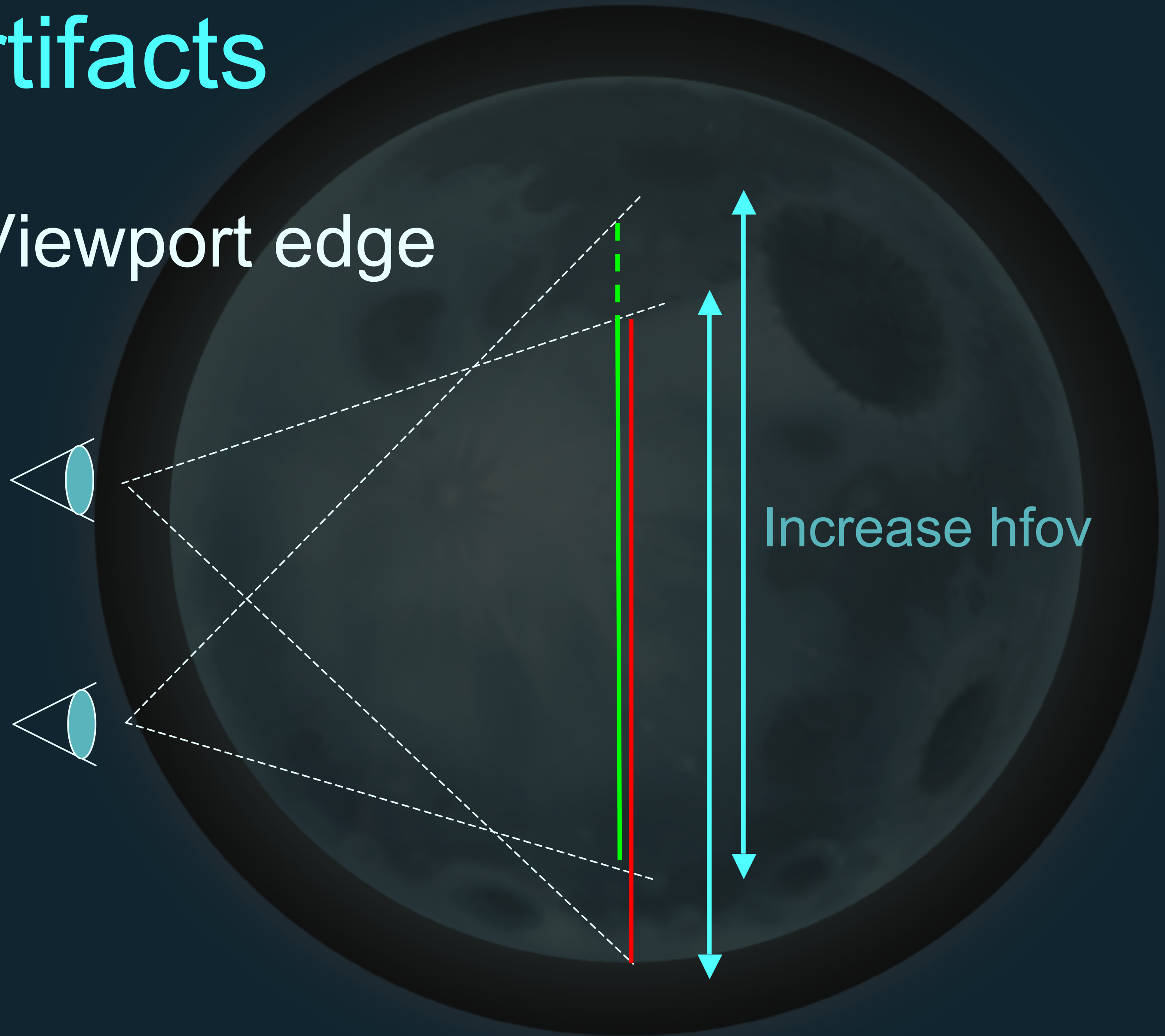
Artifacts

- Viewport edge



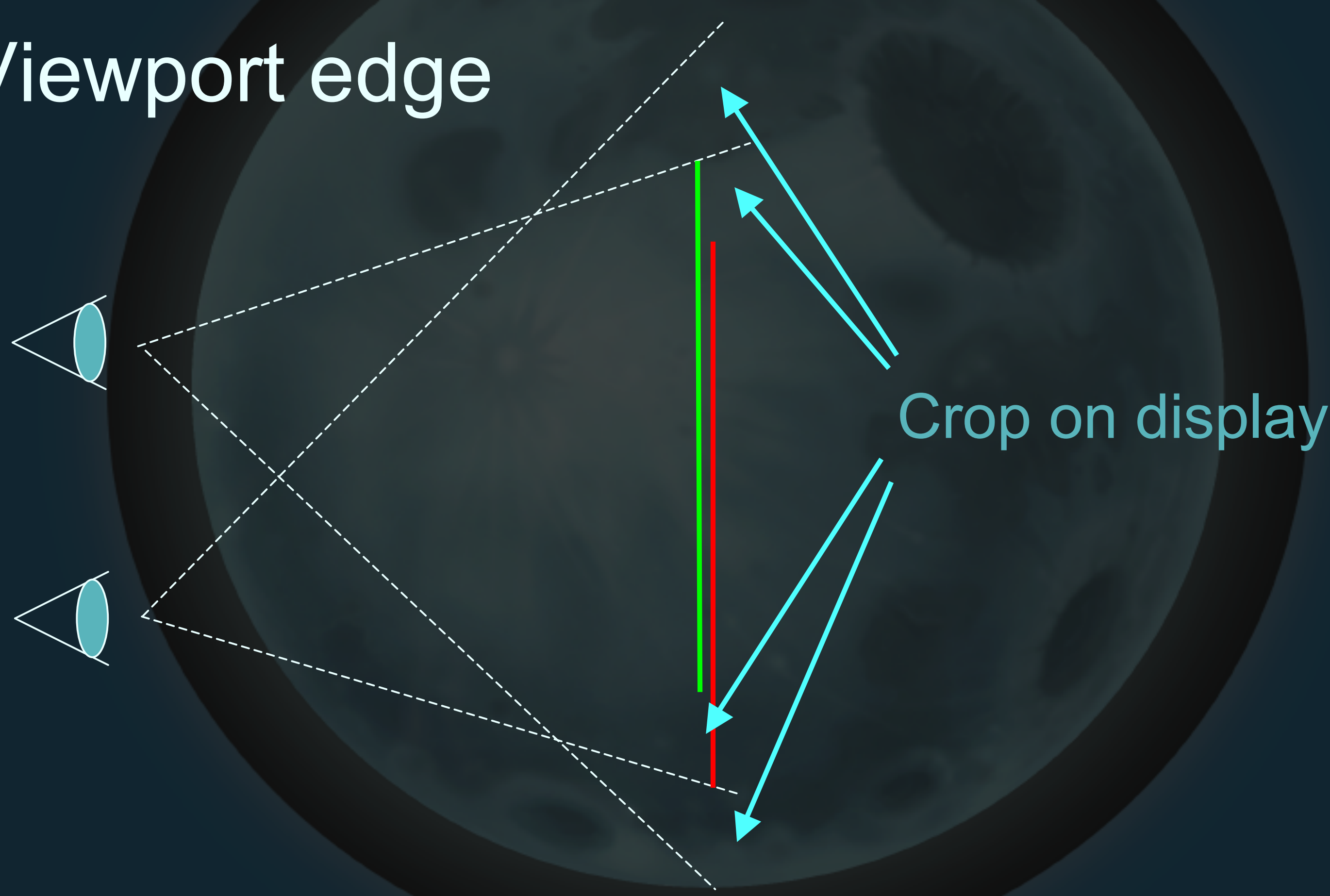
Artifacts

- Viewport edge



Artifacts

- Viewport edge



Artifacts

- Viewport edge

Alpha-Blended Objects

- No depth, looks painted on
- Solution:
  - Render opaque objects
  - Reproject to left and right images
  - Must write depth as well as color!
  - Render alpha into both images
- Tricky bit:
  - Aligning alpha with opaque

Alpha-Blended Objects

Standard Stereo

$$\begin{bmatrix} p_{11} & 0 & 0 & 0 \\ 0 & p_{22} & 0 & 0 \\ p_{31} \mp S & p_{32} & p_{33} & 1 \\ \pm T p_{11} & 0 & p_{34} & 0 \end{bmatrix}$$

Reprojection

$$\mathrm{P} = \mathrm{P}_{center} + (s - \frac{sc}{z}, 0)$$

where

$$S = i/w$$

$$T = S \cdot c \cdot \tan(fov/2)$$

Alpha-Blended Objects

Standard Stereo

$$\begin{bmatrix} p_{11} & 0 & 0 & 0 \\ 0 & p_{22} & 0 & 0 \\ p_{31} \mp S & p_{32} & p_{33} & 1 \\ \pm T p_{11} & 0 & p_{34} & 0 \end{bmatrix}$$

where

$$S = i/w$$

$$T = S \cdot c \cdot \tan\,(fov/2)$$

Reprojection

$$\mathrm{P} = \mathrm{P}_{center} + (s - \frac{sc}{z}, 0)$$

What is s?

Alpha Blended Objects

- Derive for left-to-right
  - Project point to left view

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & ad & 0 & 0 \\ -S & 0 & Q & 1 \\ dT & 0 & -Qn & 0 \end{bmatrix} = \begin{bmatrix} dx - Sz + dT & ady & Qz - Qn & z \end{bmatrix}$$

  - Consider only x after w divide:

$$\begin{aligned} x_{left} &= \frac{dx}{z} - S + \frac{dT}{z} \\ &= x_{center} - S + \frac{dT}{z} \end{aligned}$$

Alpha Blended Objects

- Derive for left-to-right
  - Can expand and simplify

$$x_{left} = x_{center} - S + \frac{dT}{z}$$

$$= x_{center} - S + \frac{dSc \cdot \tan(fov/2)}{z}$$

$$= x_{center} - S + \frac{Sc}{z} \qquad \text{(since } d = \cot(fov/2)\text{)}$$

  - Similarly

$$x_{right} = x_{center} + S - \frac{Sc}{z}$$

Alpha Blended Objects

- Derive for left-to-right
  - Subtracting to get shift

$$x_{right} - x_{left} = x_{center} + S - \frac{Sc}{z} - x_{center} + S - \frac{Sc}{z}$$

$$= 2S - \frac{2Sc}{z}$$

  - This is range [-1,1], halve for range [0,1]:

$$x_{right} - x_{left} = S - \frac{Sc}{z}$$

$$\boxed{x_{right} = x_{left} + S - \frac{Sc}{z}}$$

Alpha Blended Objects

- Derive for left-to-right
  - Compare to previous equation

$$x_{right} = x_{left} + S - \frac{Sc}{z}$$

$$x_{right} = x_{left} + s - \frac{sc}{z}$$

  - So s is just S!
  - For center reprojection, $S/2$

Alpha-Blended Objects

Standard Stereo

$$\begin{bmatrix} p_{11} & 0 & 0 & 0 \\ 0 & p_{22} & 0 & 0 \\ p_{31} \mp S & p_{32} & p_{33} & 1 \\ \pm T p_{11} & 0 & p_{34} & 0 \end{bmatrix}$$

Reprojection

$$\mathrm{P} = \mathrm{P}_{center} + \left(s - \frac{sc}{z}, 0\right)$$

where

$$S = i/w$$

$$T = S \cdot c \cdot \tan(1/2\, fov)$$

$$s = \mp 1/2 S$$

Alpha-Blended Objects

Standard Stereo

$$\begin{bmatrix} p_{11} & 0 & 0 & 0 \\ 0 & p_{22} & 0 & 0 \\ p_{31} \mp S & p_{32} & p_{33} & 1 \\ \pm T p_{11} & 0 & p_{34} & 0 \end{bmatrix}$$

Reprojection

$$\mathrm{P} = \mathrm{P}_{center} + (s - \frac{sc}{z}, 0)$$

where

$$S = i/w$$

$$T = S \cdot c \cdot \tan(1/2\, fov)$$

$$s = \mp 1/2S$$

(might scale by buffer width)

Alpha-Blended Objects

- Be careful of signs
- For left eye

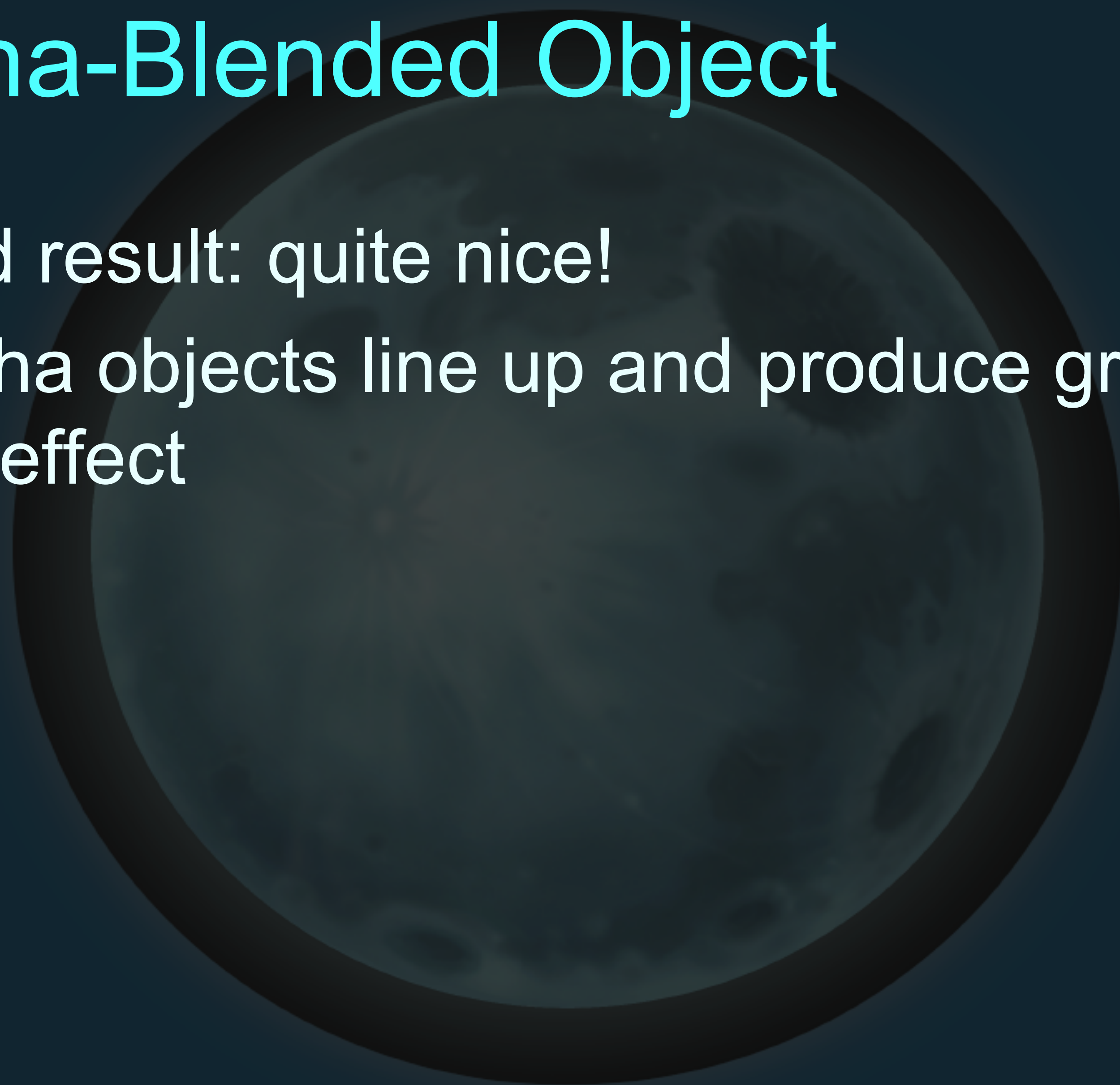
  $-S,\ +T,\ -s,\ +sc/z$

- For right eye

  $+S,\ -T,\ +s,\ -sc/z$

Alpha-Blended Object

- End result: quite nice!
- Alpha objects line up and produce great 3D effect

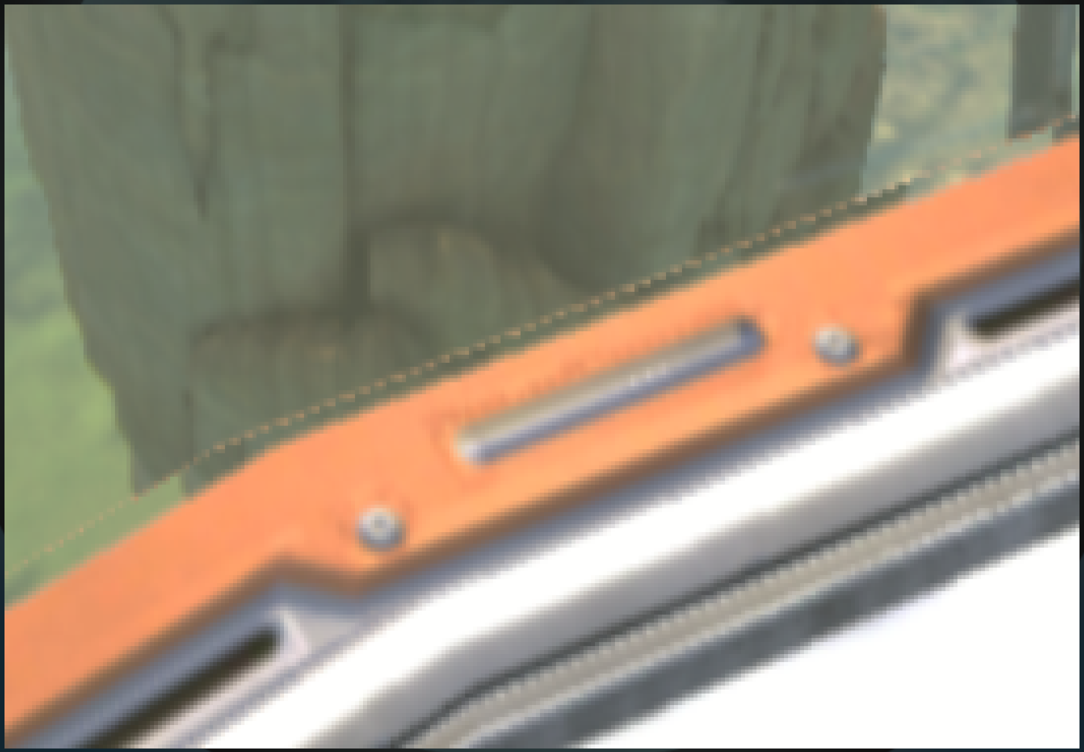


Future Work

- Alpha and opaque still not quite matching
  - Could be:
    - sampling off texel centers
    - reconstruction errors
    - float error
    - bad color buffer copy
    - fov tweaks
    - lighting in alpha pass

Future Work

- Artifacts at edge of reconstruction



Future Work

- Other color sampling possibilities
  - Try to find exact height field intersection
- Adaptive iterations
  - like parallax mapping

Summary

- Reprojection a success
  - Good frame rate
  - Reasonable results
  - Fun challenge

References

- Marries van de Hoef and Bas Zalmstra, “Fast Gather-based Construction of Stereoscopic Images Using Reprojection,” Utrecht University, 2011.
- Natalya Tatarchuk, “Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows,” ATI, 2006

References

- Samuel Gateau, “Stereoscopic 3D Demystified: From Theory to Implementation in Starcraft 2,” NVIDIA, GDC 2011
- Chris Hall, Rob Hall, and Dave Edwards, “Rendering in *Cars 2,*” Advances in Real-Time Rendering in Games, SIGGRAPH 2011

Questions?

Thank you for coming!

jim@essentialmath.com