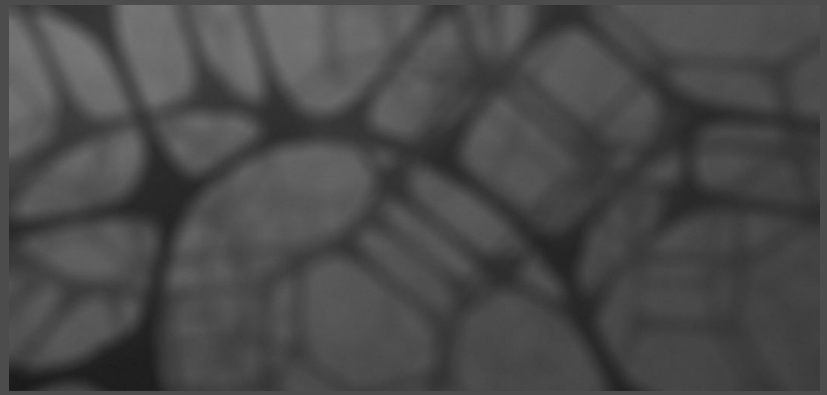
Conemarching in VR

Developing a Fractal experience at 90 FPS

Johannes Saam
Mariano Merchante

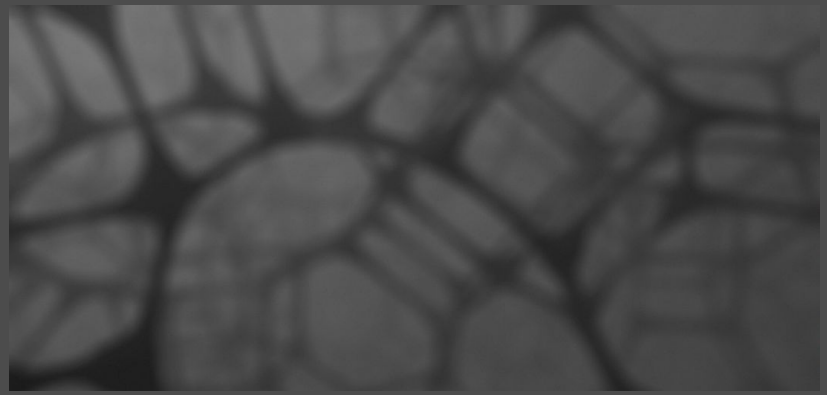
FRAMESTORE

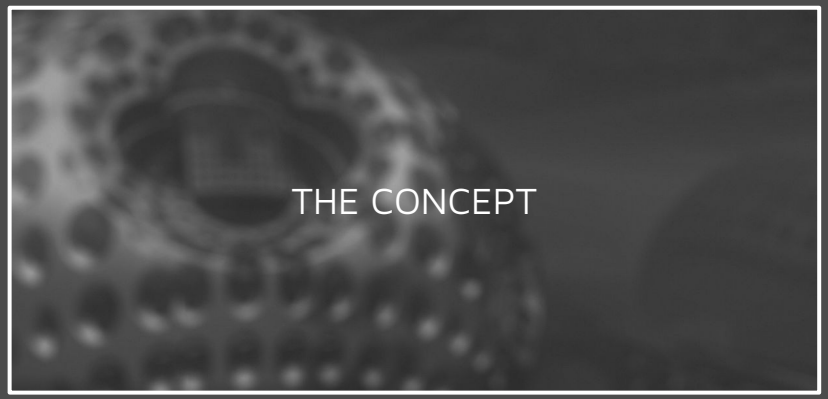
THE CONCEPT

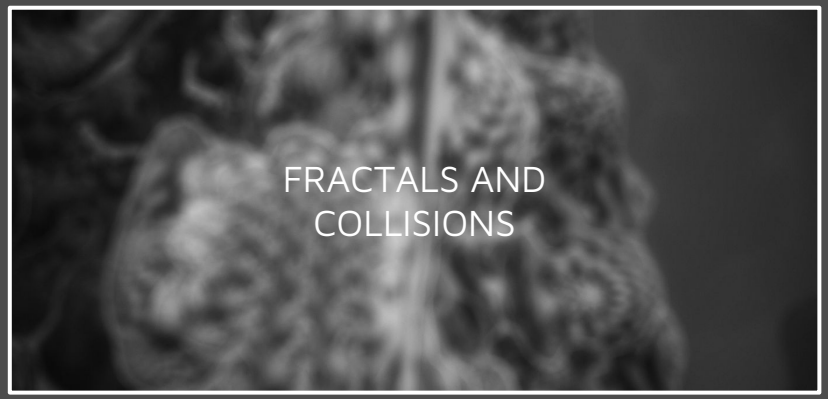


THE CONCEPT

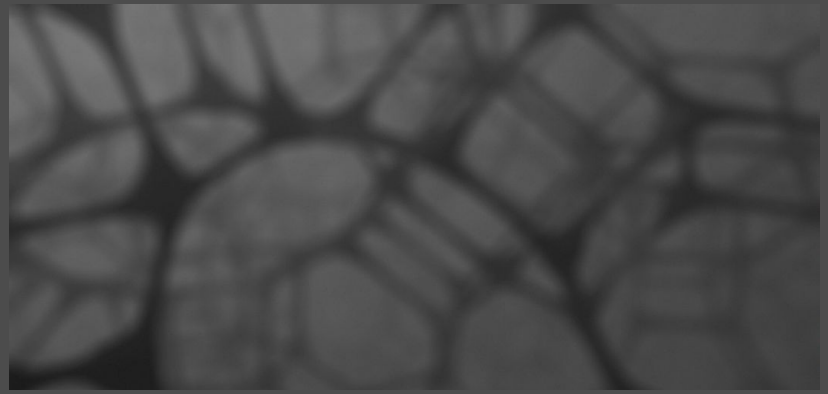
FRACTALS AND
COLLISIONS

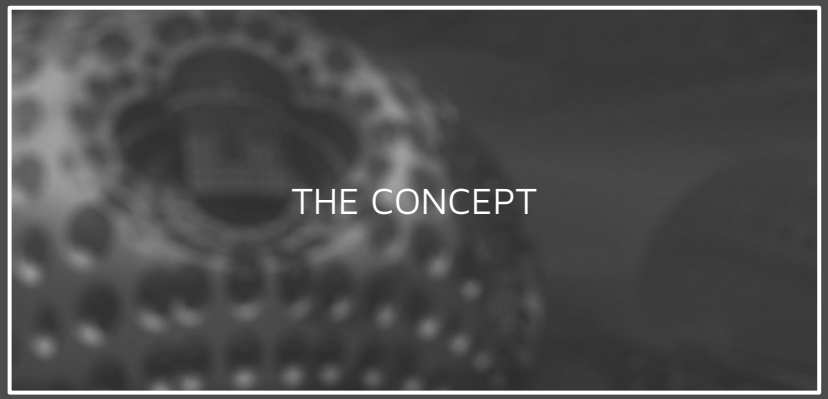


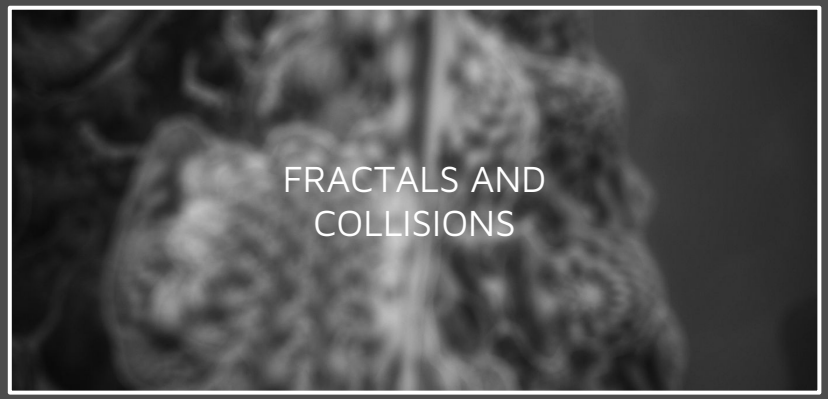

THE CONCEPT


FRACTALS AND
COLLISIONS


RAYMARCHING AND VR



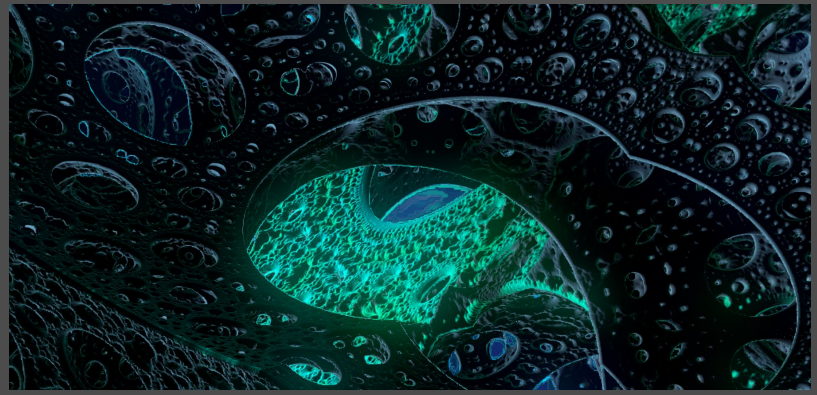

THE CONCEPT

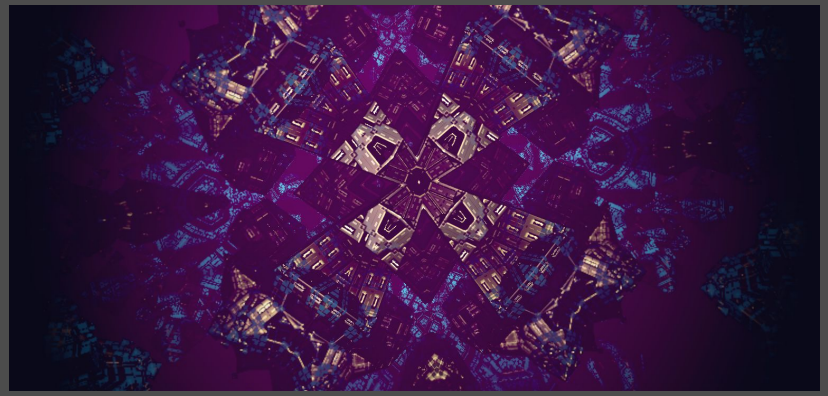

FRACTALS AND
COLLISIONS


RAYMARCHING AND VR

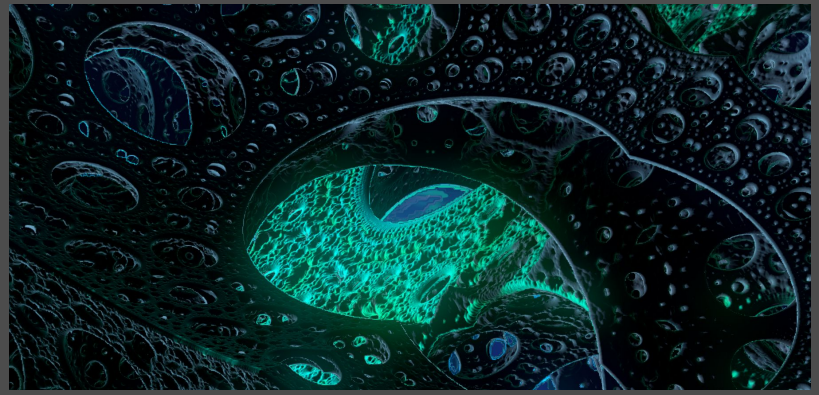

THE SHADING CHALLENGE

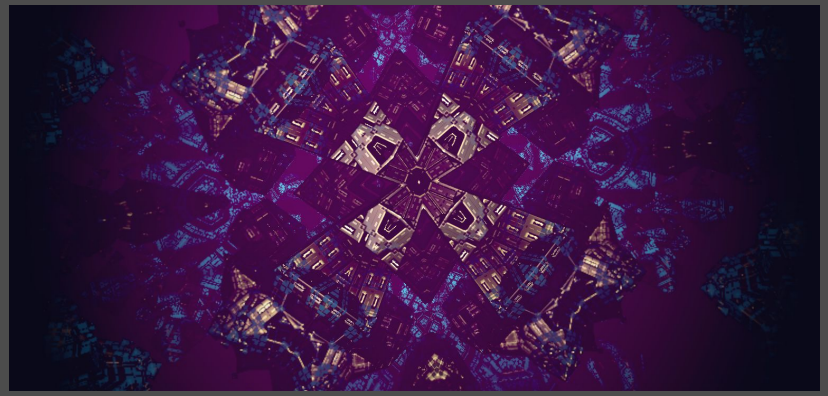
THE CONCEPT





THE CONCEPT





- Explore classic fractals in real-time VR
- Simple interaction model, sandbox feel
- Break a few rules!

HOW DO WE TRANSMIT SENSE OF SCALE?

HOW DO WE TRANSMIT SENSE OF SCALE?

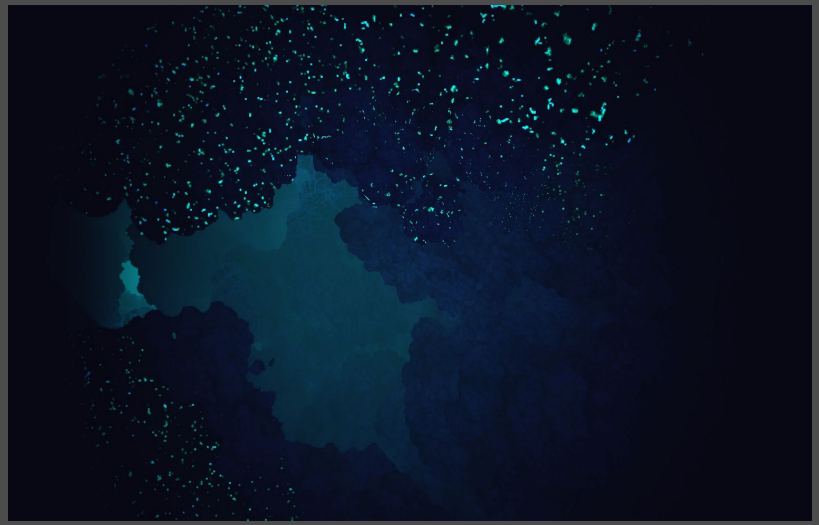
THE LOCOMOTION CHALLENGE

HOW DO WE TRANSMIT SENSE OF SCALE?

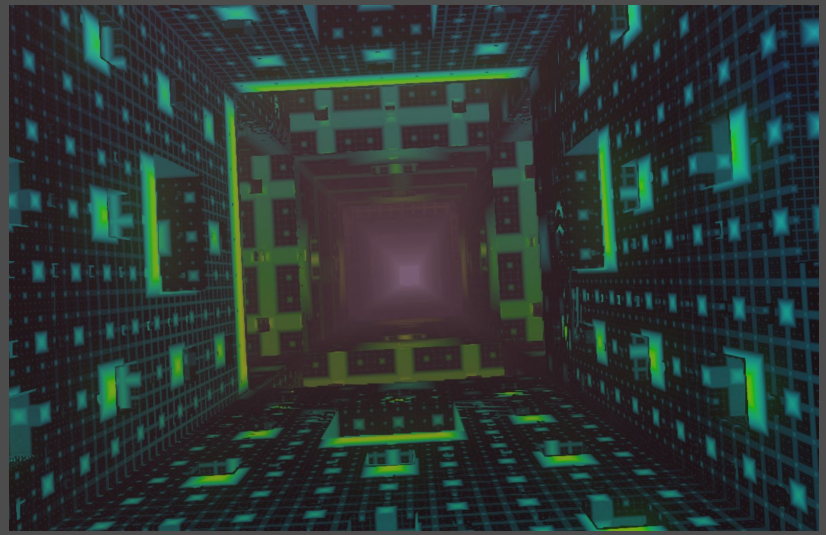
- Scale the player's eyes as they move around

HOW DO WE TRANSMIT SENSE OF SCALE?

- Scale the player's eyes as they move around
- Make sure we smooth scale to prevent discontinuities



- Scale the player's eyes as they move around
- Make sure we smooth scale to prevent discontinuities
- Design shading such that there is enough size contrast



- Scale the player's eyes as they move around
- Make sure we smooth scale to prevent discontinuities
- Design shading such that there is enough size contrast
- Loop each fractal to emphasize the sense of infinity

- Always move in the direction of your head…

THE LOCOMOTION CHALLENGE

- Always move in the direction of your head…
- … unless you prefer to strafe

THE LOCOMOTION CHALLENGE

- Always move in the direction of your head…
- … unless you prefer to strafe
- Velocity tied to scale and closest distance

THE LOCOMOTION CHALLENGE

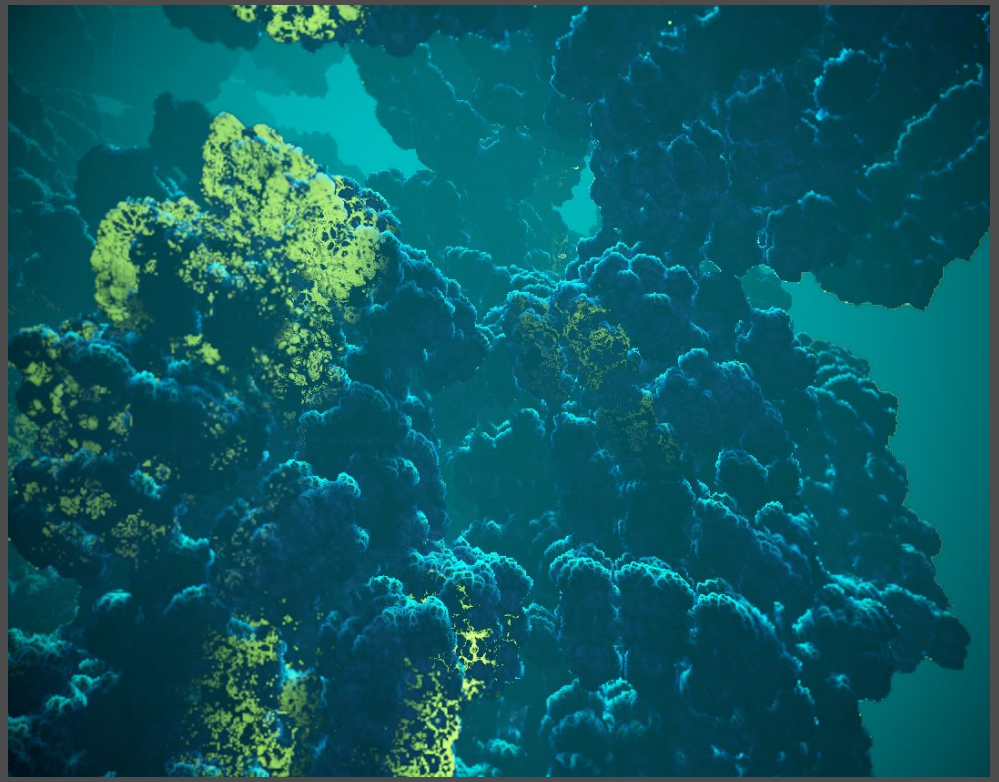
- Always move in the direction of your head…
- … unless you prefer to strafe
- Velocity tied to scale and closest distance
- Option to rotate 90 degrees with fade

THE LOCOMOTION CHALLENGE

- Always move in the direction of your head…
- … unless you prefer to strafe
- Velocity tied to scale and closest distance
- Option to rotate 90 degrees with fade
- Vignette intensity directly proportional to velocity and angular velocity.

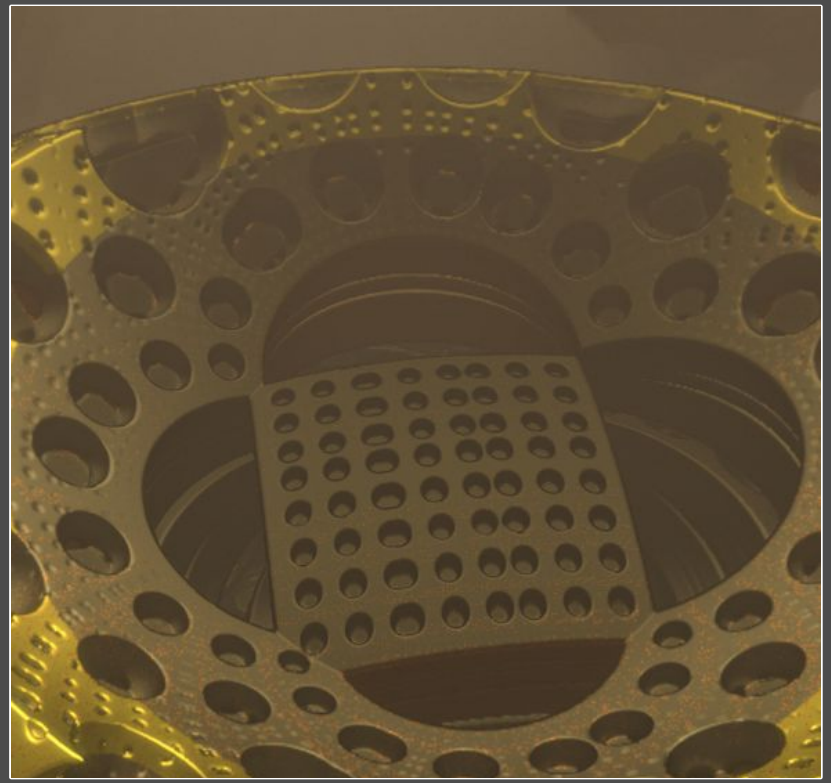
THE LOCOMOTION CHALLENGE

FRACTALS AND COLLISIONS

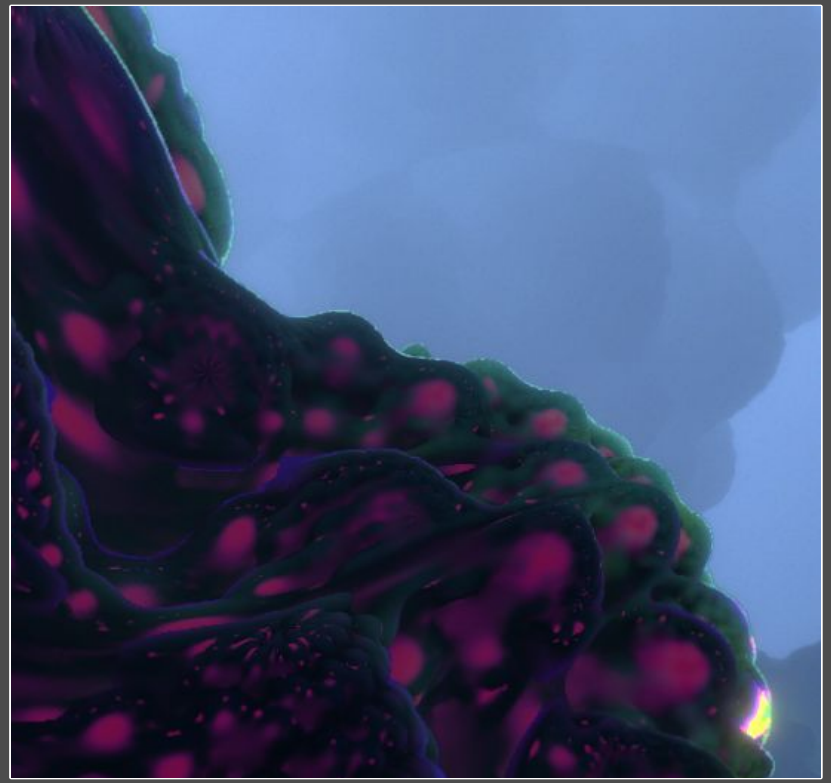


FRACTALS

- Require a distance field to be marched
- Generally very expensive!
  - Usually have multiple iterations for a single estimation



- Require a distance field to be marched
- Generally very expensive!
  - Usually have multiple iterations for a single estimation
- Scale detail as player approaches fractal
  - Rendering more expensive as we get smaller!



- Require a distance field to be marched
- Generally very expensive!
  - Usually have multiple iterations for a single estimation
- Scale detail as player approaches fractal
  - Rendering more expensive as we get smaller!
- Hard to predict and high frequency shapes
- Scaling leads to floating point precision problems!
  - Theoretically infinite detail

COLLISIONS

- We already use the fractal SDF for rendering

COLLISIONS

- We already use the fractal SDF for rendering
- We can use the same SDF for collisions!
  - R: distance, GBA: normal for bounce

COLLISIONS

- We already use the fractal SDF for rendering
- We can use the same SDF for collisions!
  - R: distance, GBA: normal for bounce
- Render a 1x1 texture and read it asynchronously (Thanks Unity!)
  - Prevent GPU stall due to VR context

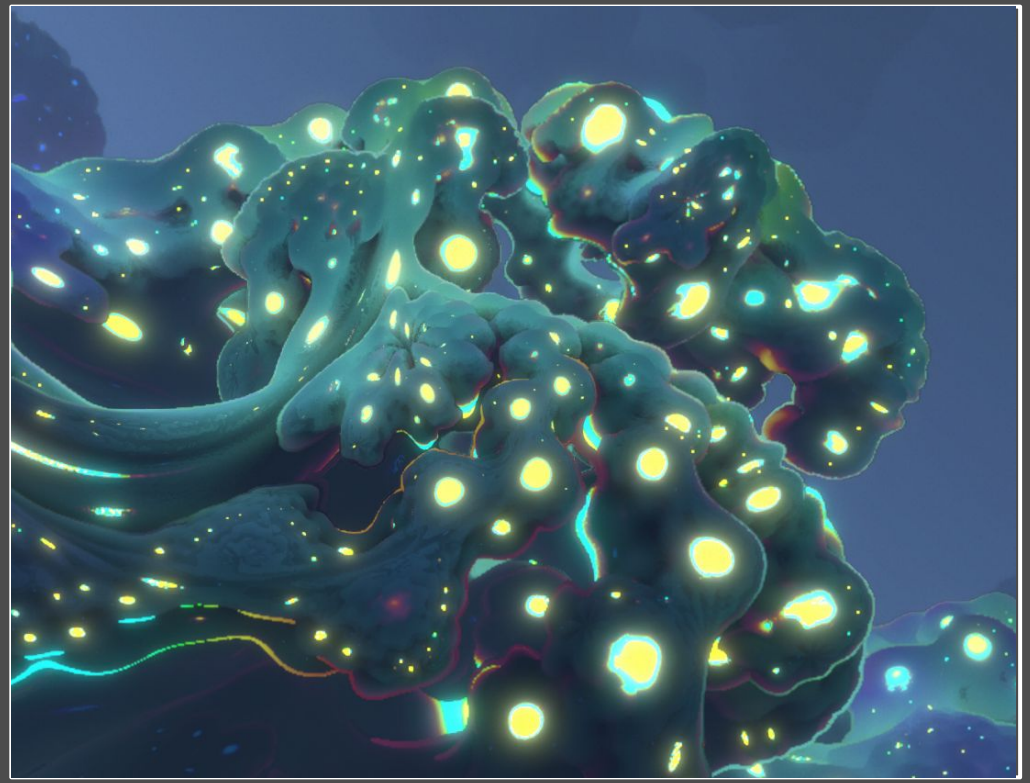
COLLISIONS

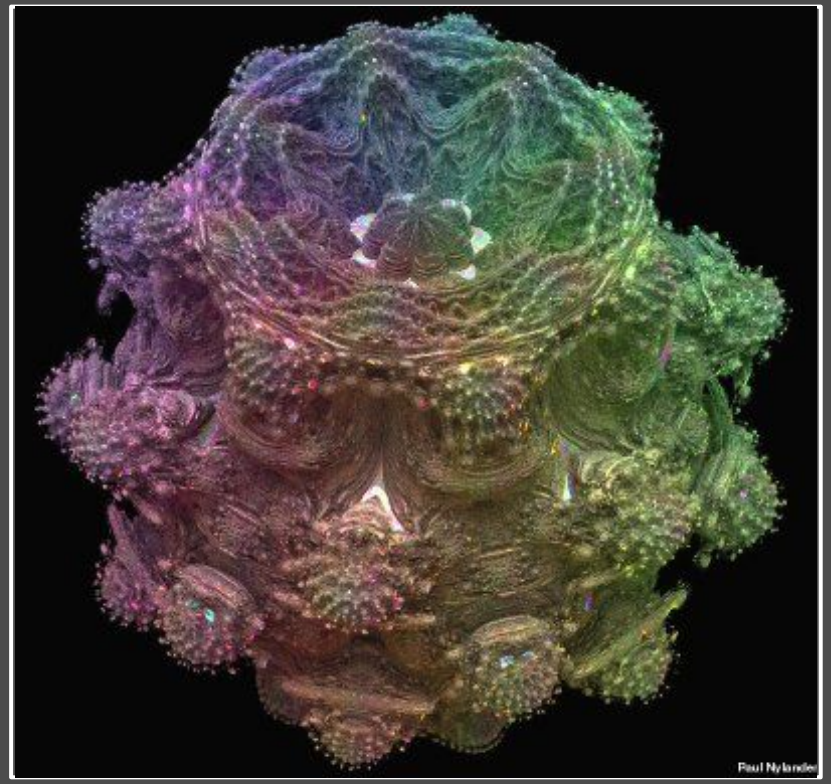
- We already use the fractal SDF for rendering
- We can use the same SDF for collisions!
  - R: distance, GBA: normal for bounce
- Render a 1x1 texture and read it asynchronously (Thanks Unity!)
  - Prevent GPU stall due to VR context
- Why not a compute shader?
  - Why not CPU?

COLLISIONS

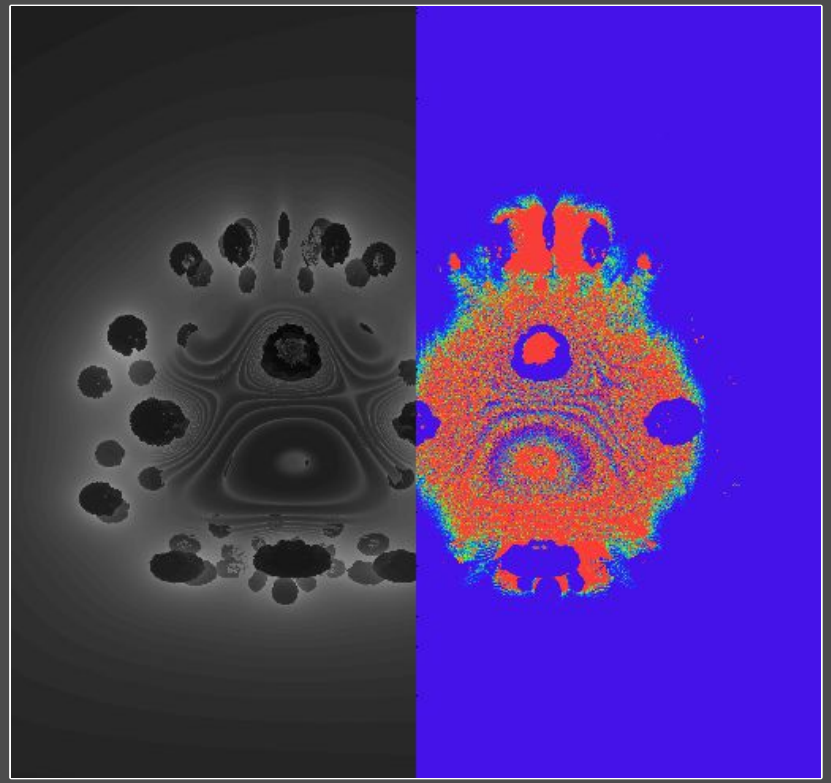
- We already use the fractal SDF for rendering
- We can use the same SDF for collisions!
  - R: distance, GBA: normal for bounce
- Render a 1x1 texture and read it asynchronously (Thanks Unity!)
  - Prevent GPU stall due to VR context
- Why not a compute shader?
  - Why not CPU?
- Predict a bit due to latency

FIRST STEPS: MANDELBULB

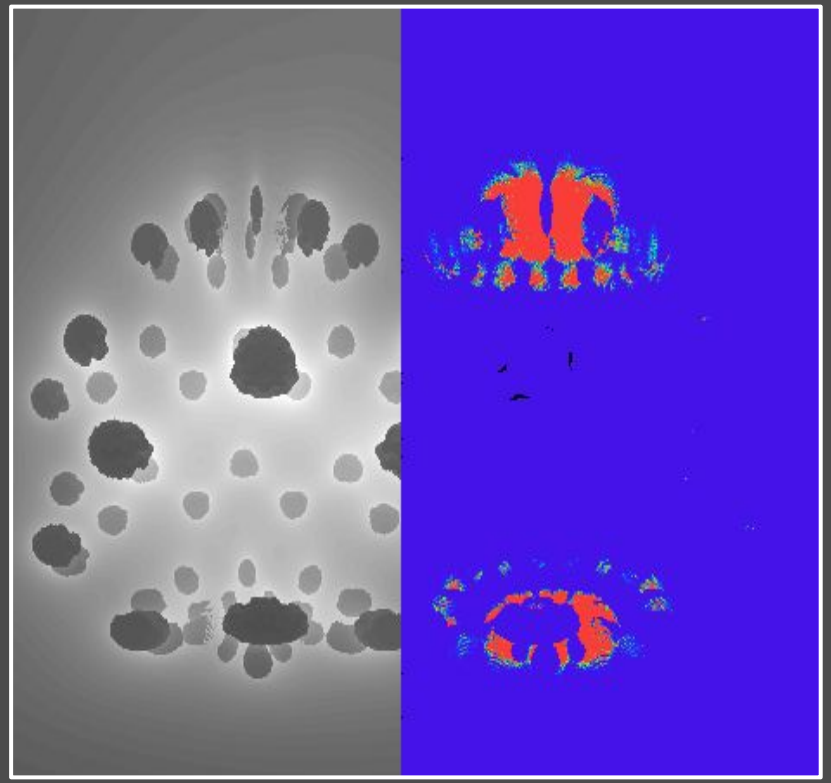




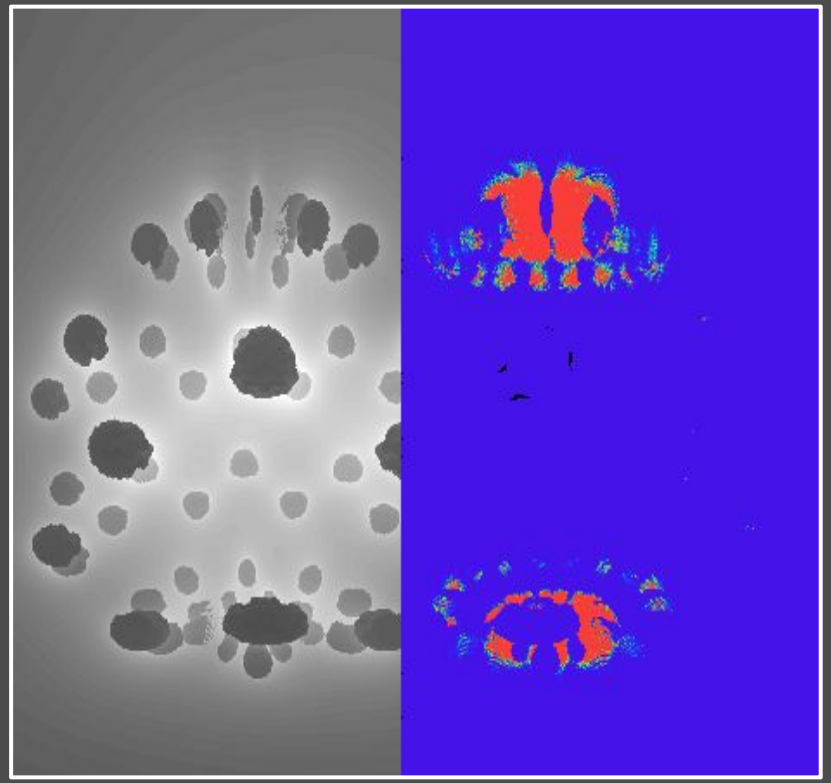
- Famous 3D fractal developed by Daniel White and Paul Nylander
  - The fractalforums thread is amazing!



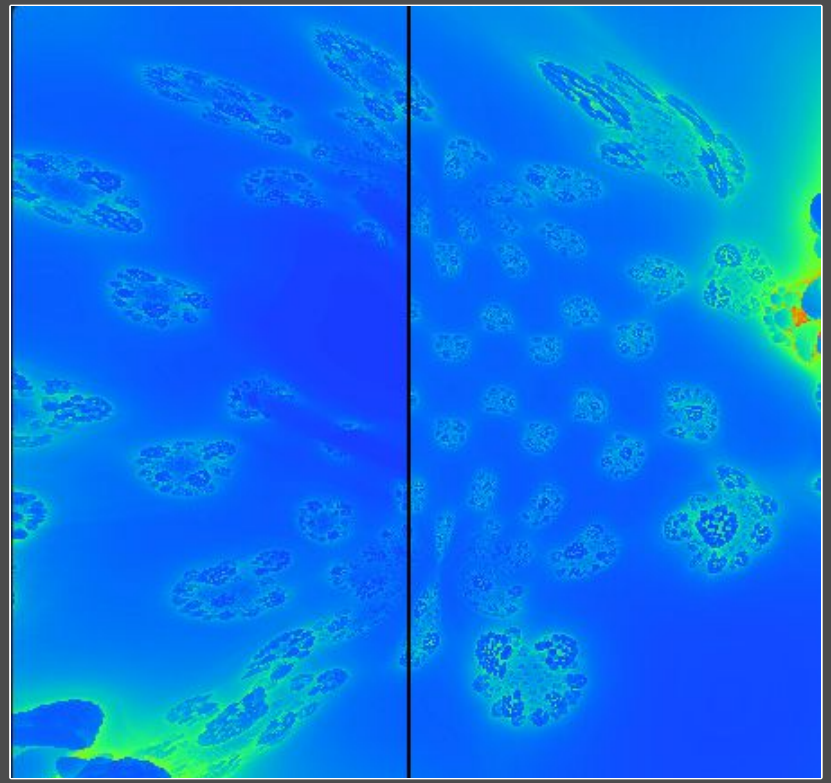
- Famous 3D fractal developed by Daniel White and Paul Nylander
  - The fractalforums thread is amazing!
- Decided to use Íñigo Quílez's formula, but found bubbles of overestimation for Julia offsets
  - We built a tool for finding these bubbles through stochastic brute force



- Famous 3D fractal developed by Daniel White and Paul Nylander
  - The fractalforums thread is amazing!
- Decided to use Íñigo Quílez's formula, but found bubbles of overestimation for Julia offsets
- Tweaked the formula to prevent these bubbles
  - Kept the maximum derivative as we iterate



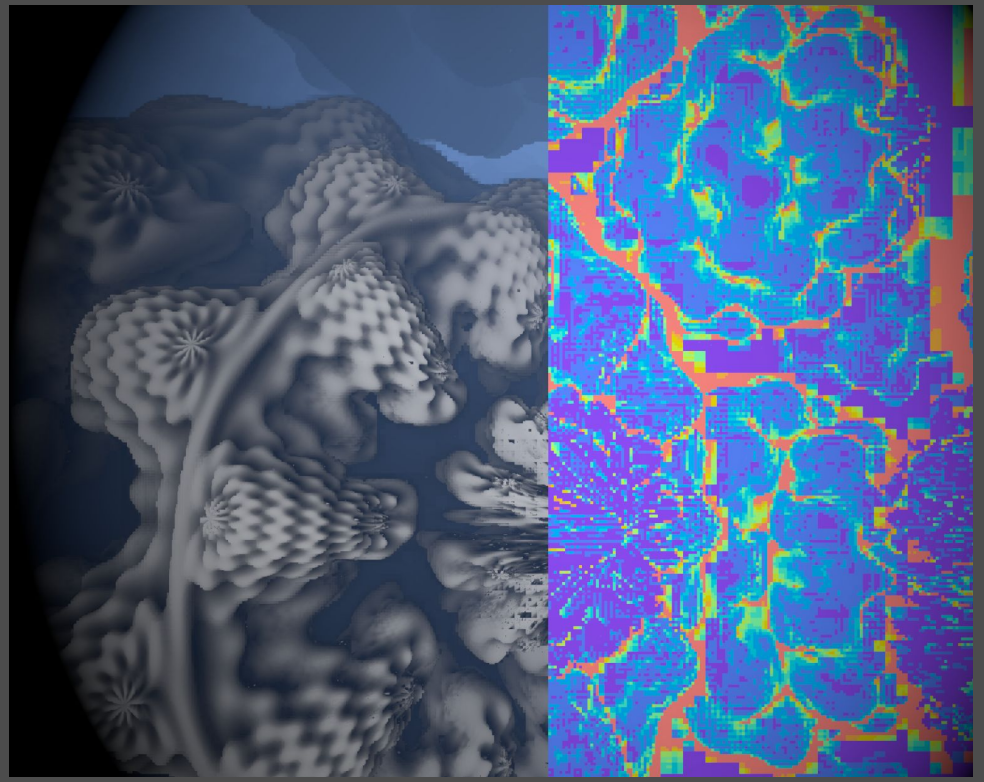
- Famous 3D fractal developed by Daniel White and Paul Nylander
  - The fractalforums thread is amazing!
- Decided to use Íñigo Quílez's formula, but found bubbles of overestimation for Julia offsets
- Tweaked the formula to prevent these bubbles
  - Kept the maximum derivative as we iterate
- More iterations needed to render, but overall better than reducing the epsilon



```
float evaluateMandelbulb(in vec3 p, in bool conservative)
{
    vec3 w = p;
    float m = dot(w,w);

    float dz = 1.0;
    vec3 J = vec3(.2);

    for( int i=0; i < 5; i++ )
    {
        if(conservative)
            dz = max(dz * DERIVATIVE_BIAS, 8.0*pow(m ,3.5)*dz + 1.0);
        else
            dz = 8.0*pow(m ,3.5)*dz + 1.0;

        float r = length(w);
        float b = 8.0*acos( clamp(w.y/r, -1.0, 1.0));
        float a = 8.0*atan( w.x, w.z );
        w = p + J + pow(r,8.0) * vec3( sin(b)*sin(a), cos(b), sin(b)*cos(a) );

        m = dot(w,w);

        if( m > 4.0 )
            break;
    }

    return 0.25*log(m)*sqrt(m)/dz;
}
```

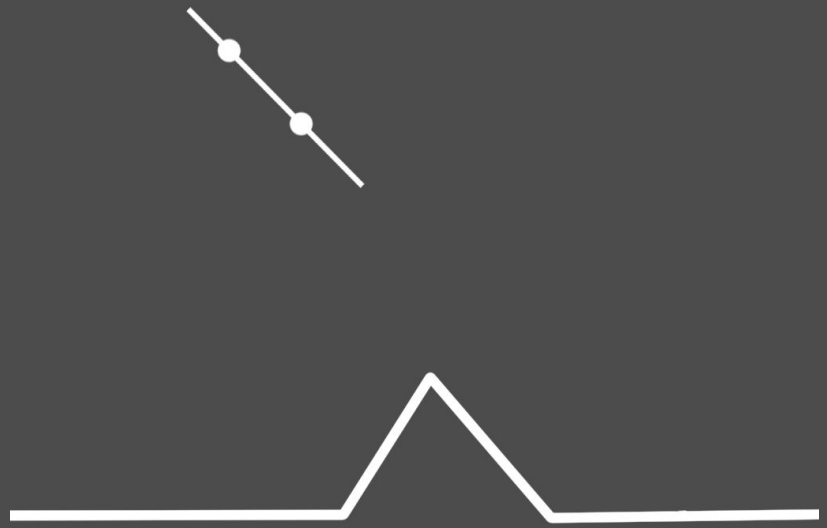
https://www.shadertoy.com/view/MdSBDR

RAYMARCHING AND VR

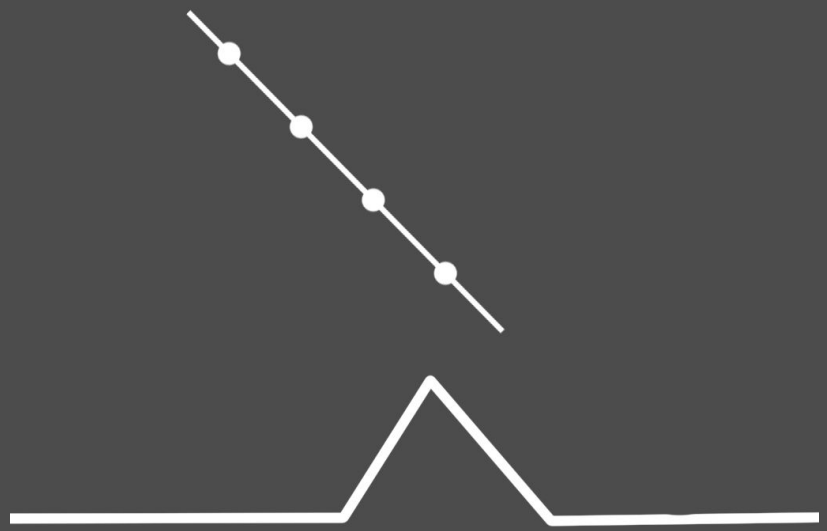


CLASSIC RAYMARCHING

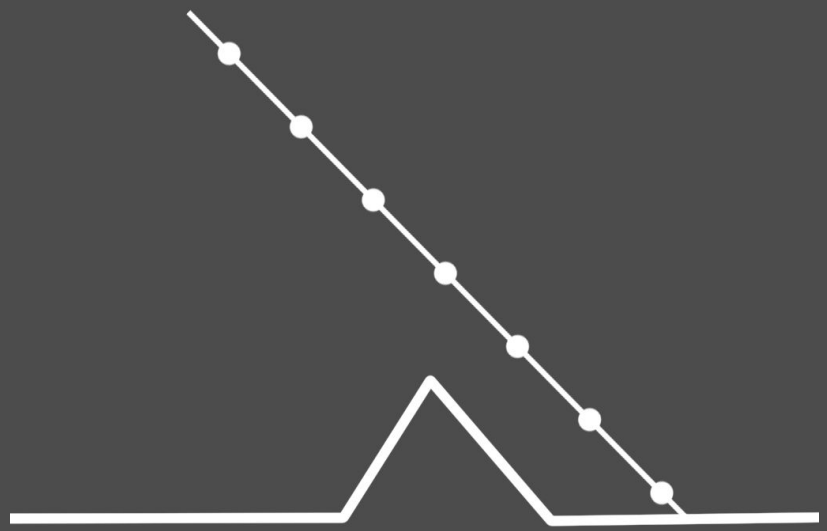
- Define a function that gives us the distance to a surface



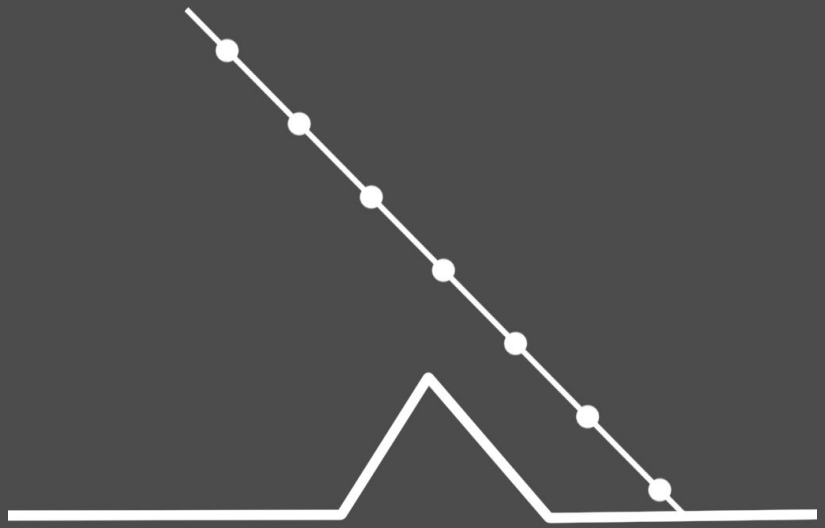
- Define a function that gives us the distance to a surface



- Define a function that gives us the distance to a surface
- Iterate through a ray with a fixed step



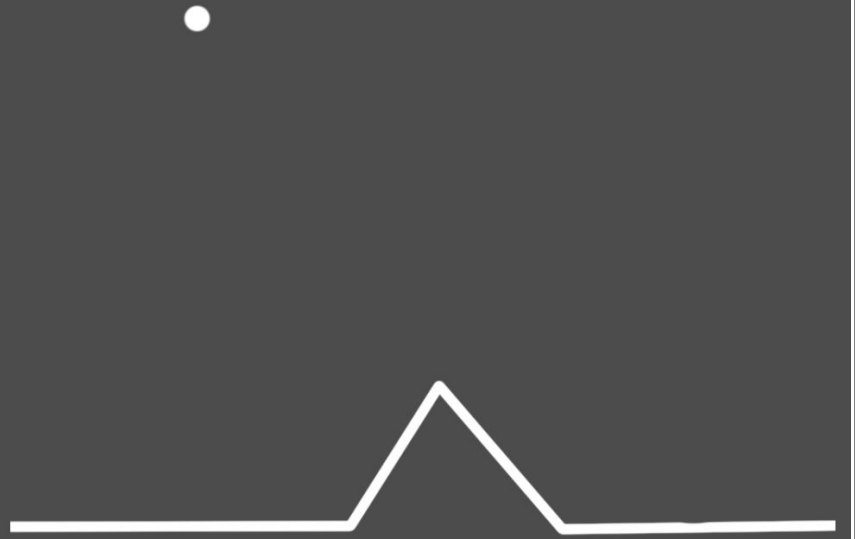
- Define a function that gives us the distance to a surface
- Iterate through a ray with a fixed step
- If the distance to the surface is less than a specified amount, we hit the surface



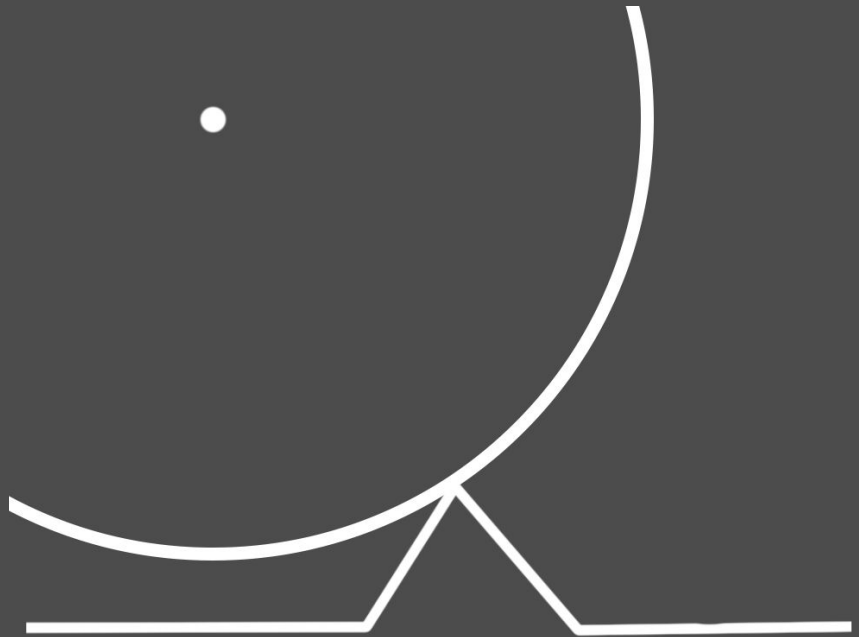
- Define a function that gives us the distance to a surface
- Iterate through a ray with a fixed step
- If the distance to the surface is less than a specified amount, we hit the surface
- Use finite differences for normal estimation and shading
- Shadows, AO, SSS, godrays very straightforward
- **Very** expensive!

SPHERE TRACING

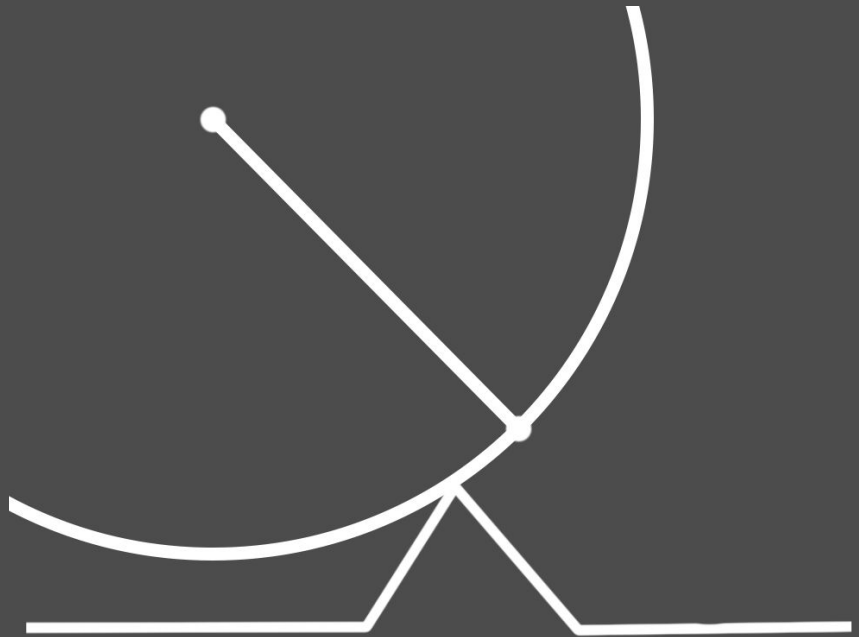
- Each step jump is proportional to the distance of the surface



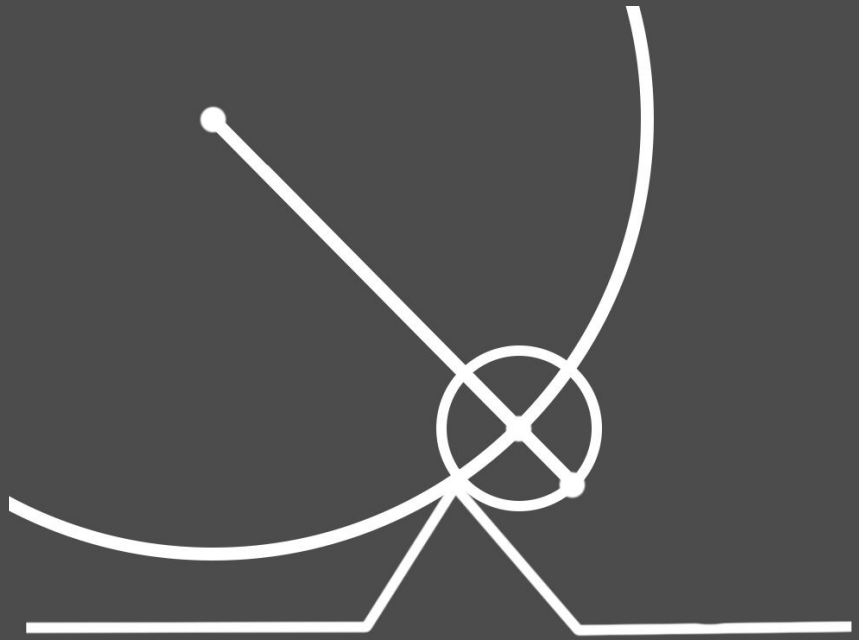
- Each step jump is proportional to the distance of the surface



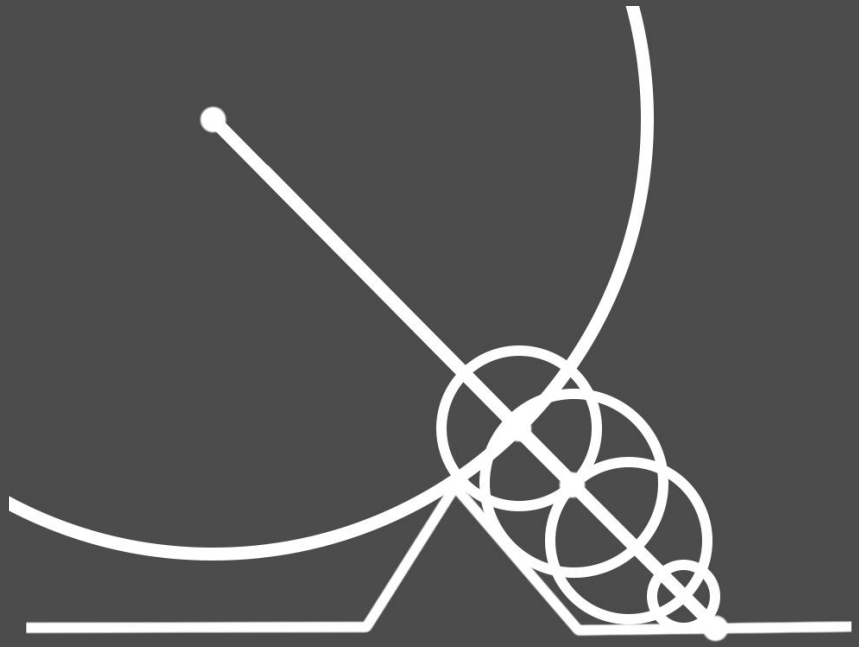
- Each step jump is proportional to the distance of the surface



- Each step jump is proportional to the distance of the surface



- Each step jump is proportional to the distance of the surface



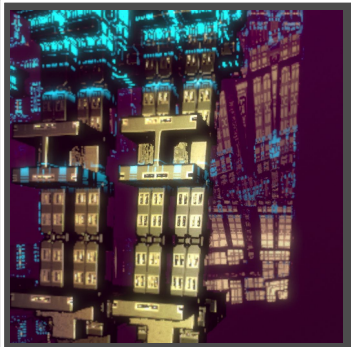
- Each step jump is proportional to the distance of the surface

SPHERE TRACING

- Each step jump is proportional to the distance of the surface
- Can lead to overestimation if the function is not well defined
  - Fractals are not ideal!
  - We bias each fractal SDF empirically

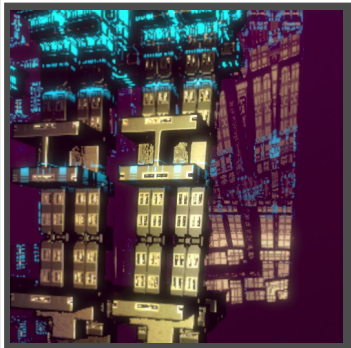
SPHERE TRACING

- Each step jump is proportional to the distance of the surface
- Can lead to overestimation if the function is not well defined
  - Fractals are not ideal!
  - We bias each fractal SDF empirically
- Preferred way to raymarch in modern approaches



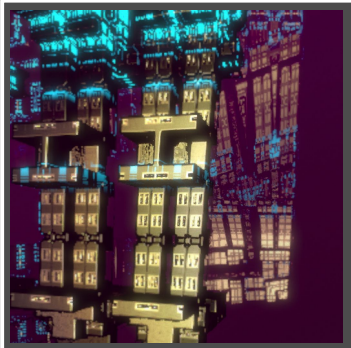
VIRTUAL REALITY CONSTRAINTS

- Our target is 90fps, so that gives us ~10ms

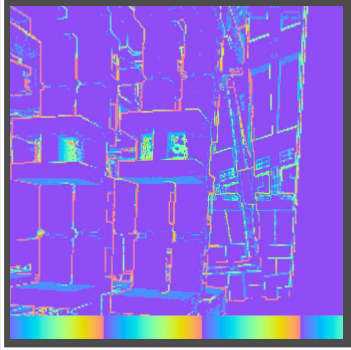


VIRTUAL REALITY CONSTRAINTS

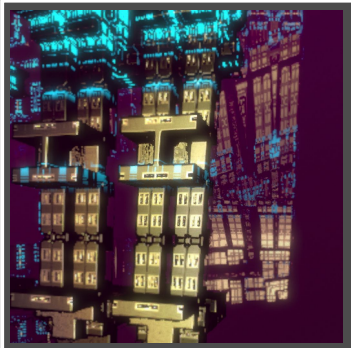
- Our target is 90fps, so that gives us ~10ms
- Any GPU stall will kill the experience



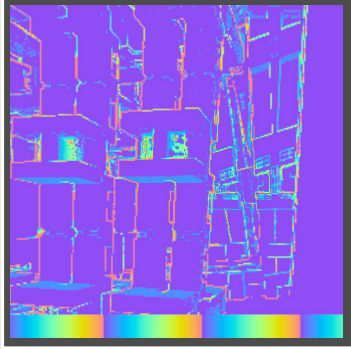
VIRTUAL REALITY CONSTRAINTS

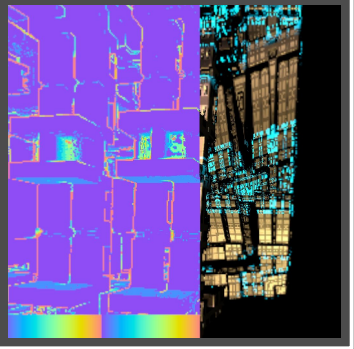


- Our target is 90fps, so that gives us ~10ms
- Any GPU stall will kill the experience
- If we can't hit the target framerate, time warping kicks in, halving fps.

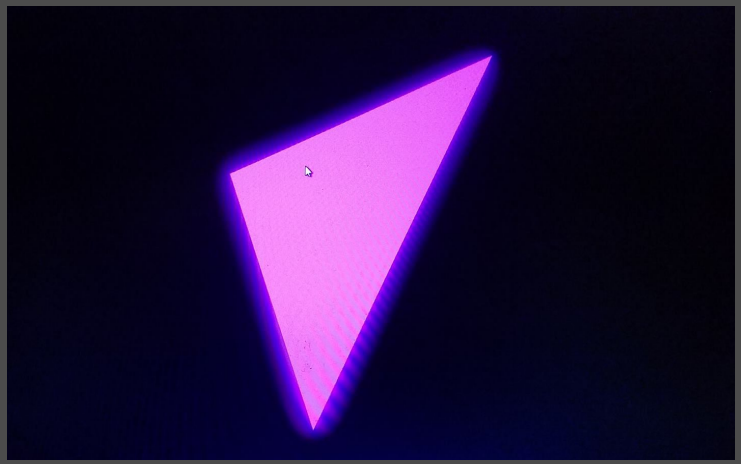


VIRTUAL REALITY CONSTRAINTS





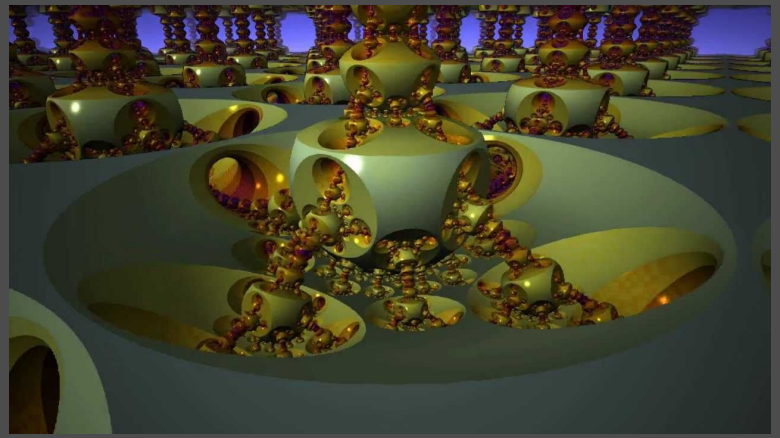
- Our target is 90fps, so that gives us ~10ms
- Any GPU stall will kill the experience
- If we can't hit the target framerate, time warping kicks in, halving fps.
- We have to render the same thing twice! :(

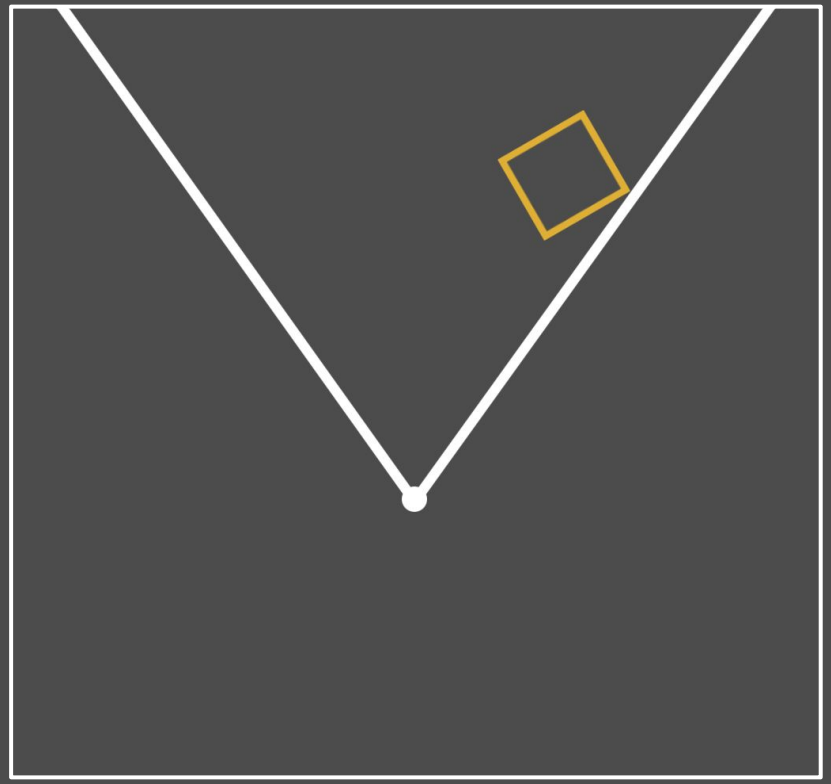


- Our target is 90fps, so that gives us ~10ms
- Any GPU stall will kill the experience
- If we can't hit the target framerate, time warping kicks in, halving fps.
- We have to render the same thing twice! :(
- Pipeline is not designed for raymarching, we have to hack Unity a bit
  - Big triangle where everything happens

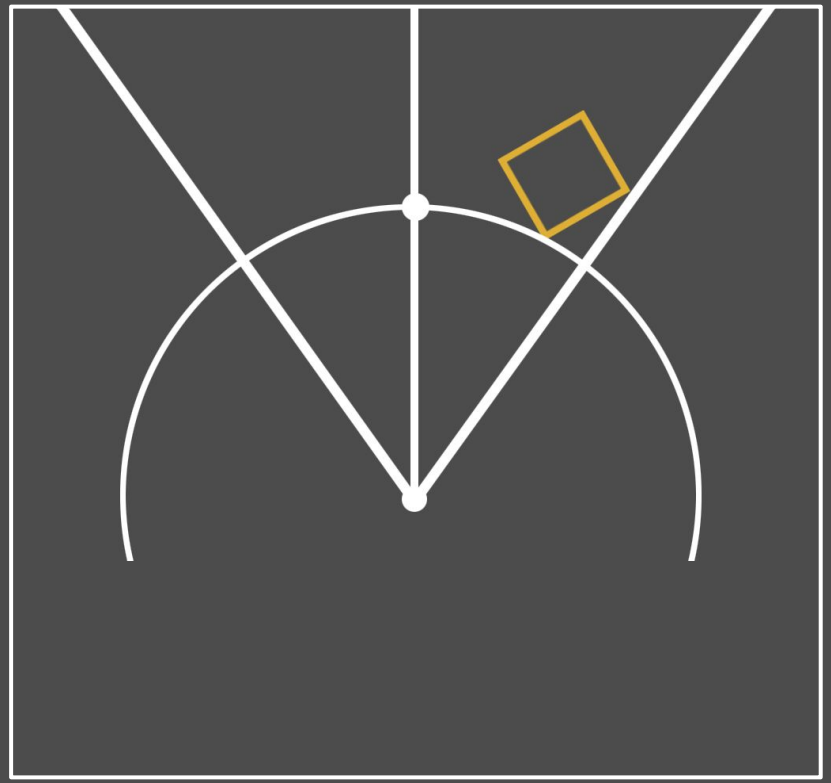
CONEMARCHING

- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012

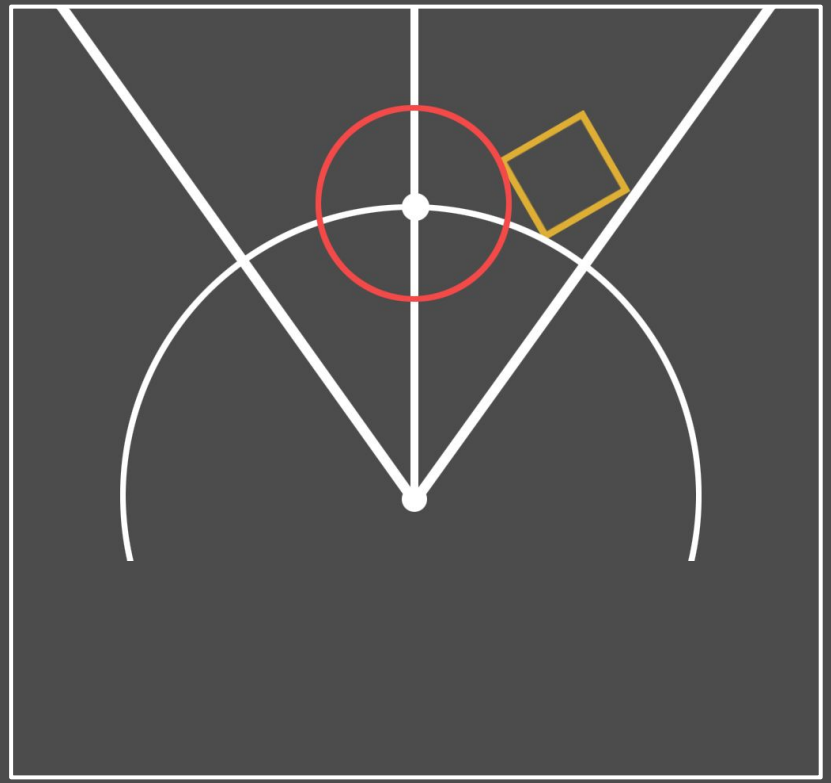




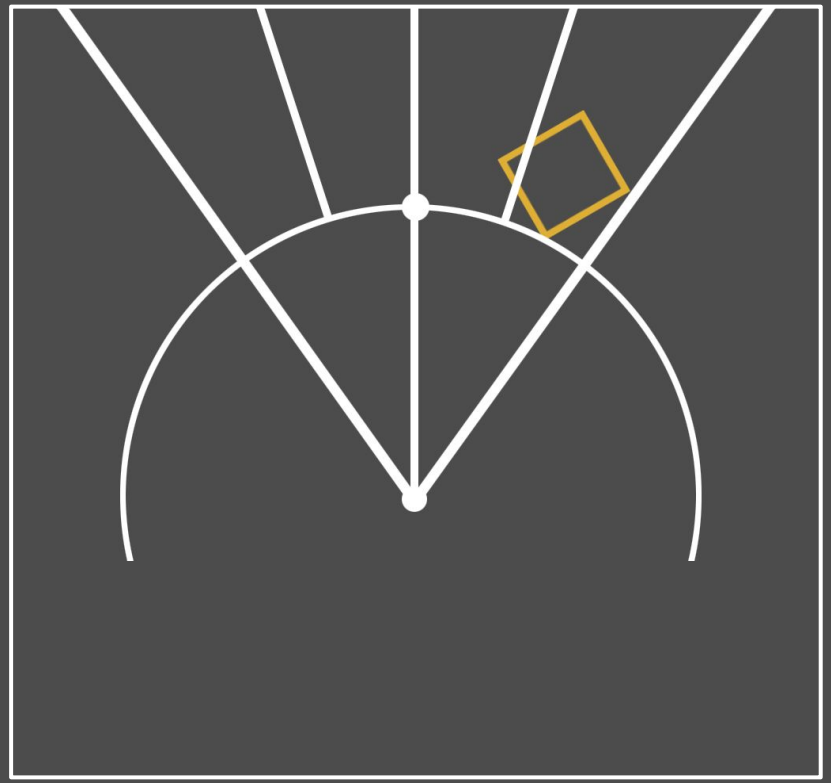
- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions



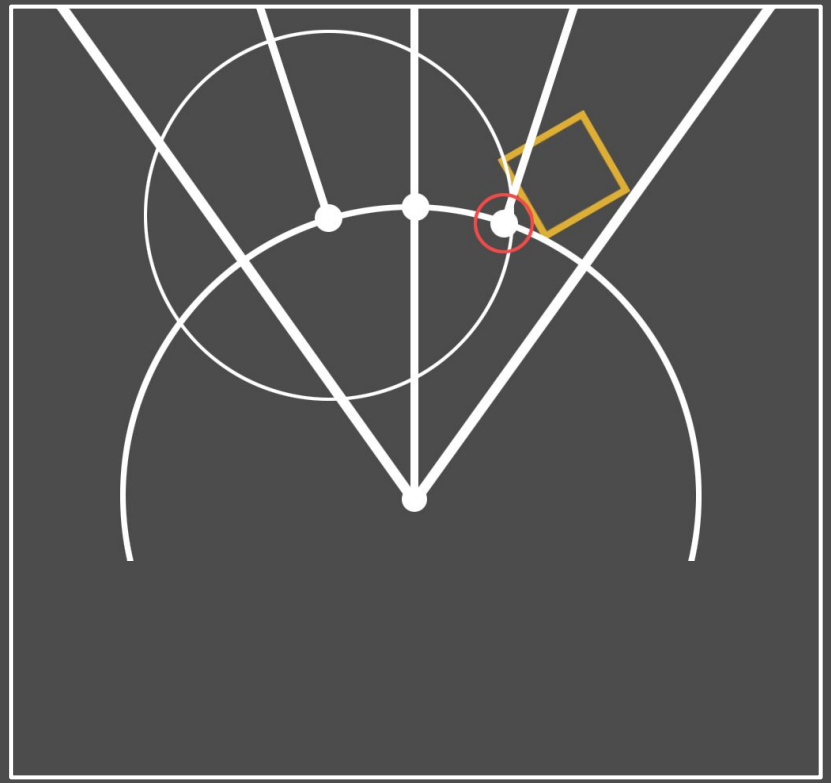
- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions
- On each pass, sphere trace until we can't guarantee there isn't an intersection



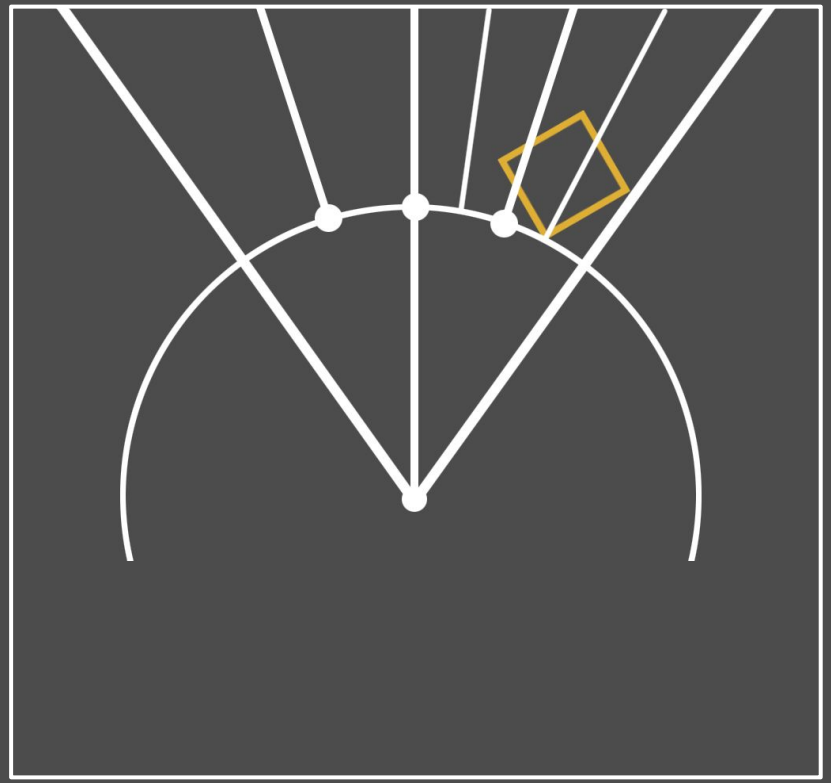
- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions
- On each pass, sphere trace until we can't guarantee there isn't an intersection



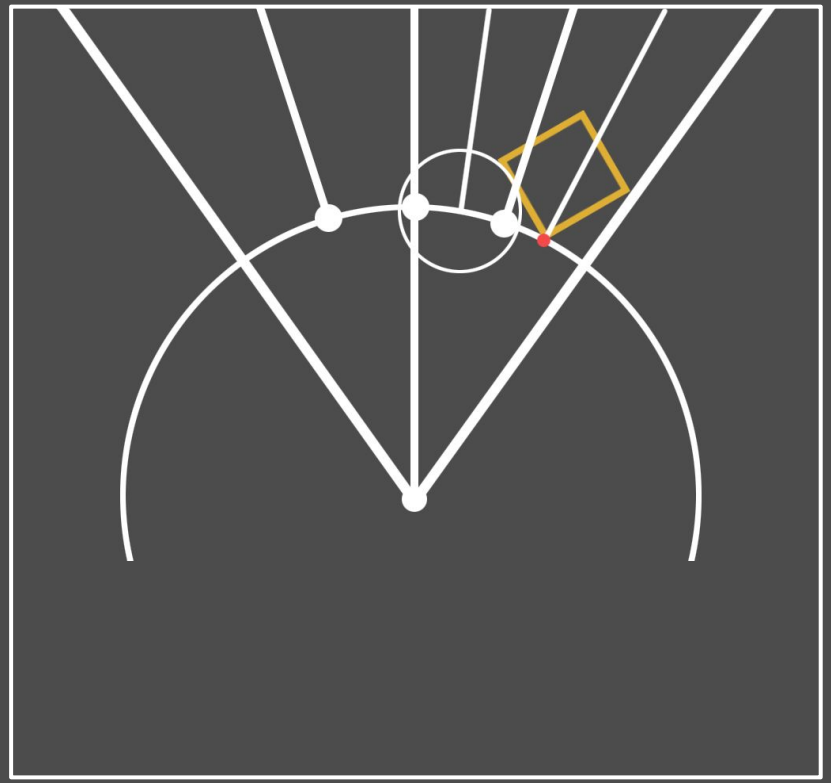
- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions
- On each pass, sphere trace until we can't guarantee there isn't an intersection
- Double the resolution and reuse the distance
  - Some bias is *usually* needed



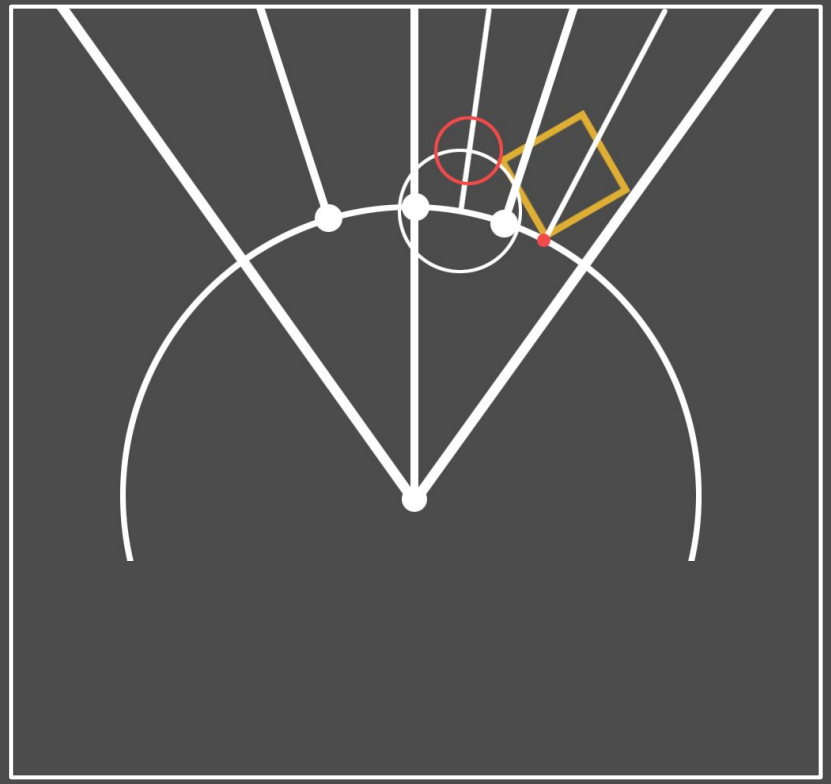
- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions
- On each pass, sphere trace until we can't guarantee there isn't an intersection
- Double the resolution and reuse the distance
  - Some bias is *usually* needed



- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions
- On each pass, sphere trace until we can't guarantee there isn't an intersection
- Double the resolution and reuse the distance
  - Some bias is *usually* needed



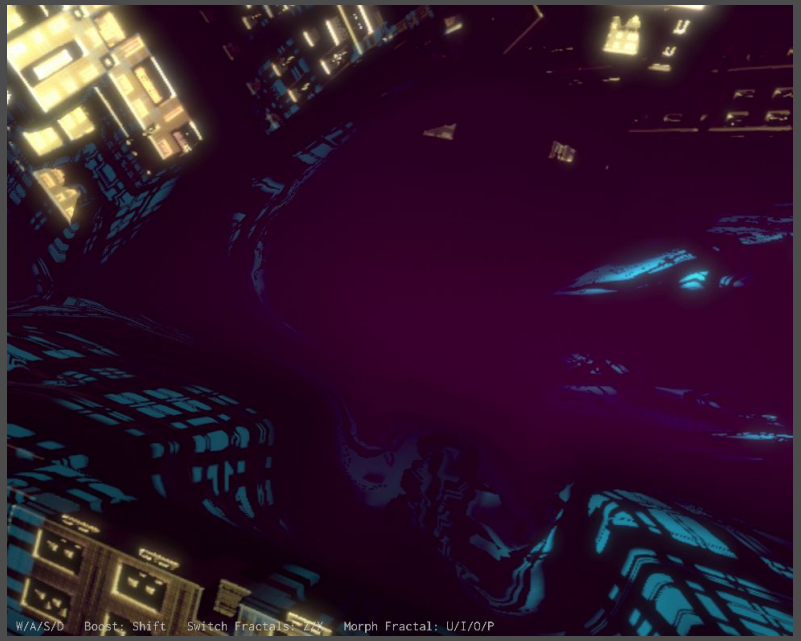
- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions
- On each pass, sphere trace until we can't guarantee there isn't an intersection
- Double the resolution and reuse the distance
  - Some bias is *usually* needed



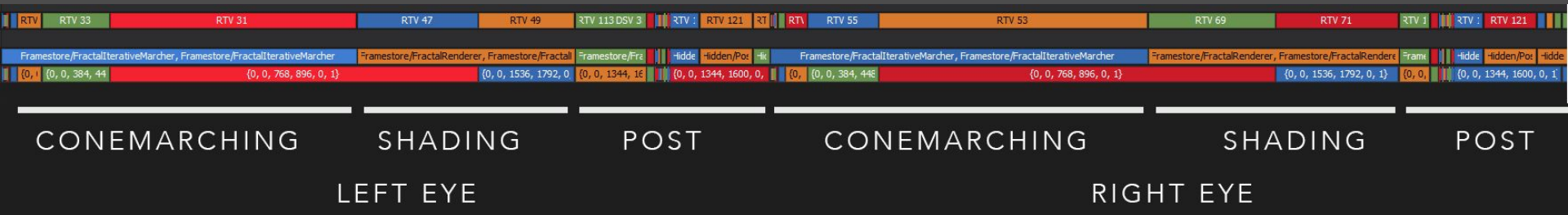
- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions
- On each pass, sphere trace until we can't guarantee there isn't an intersection
- Double the resolution and reuse the distance
  - Some bias is *usually* needed

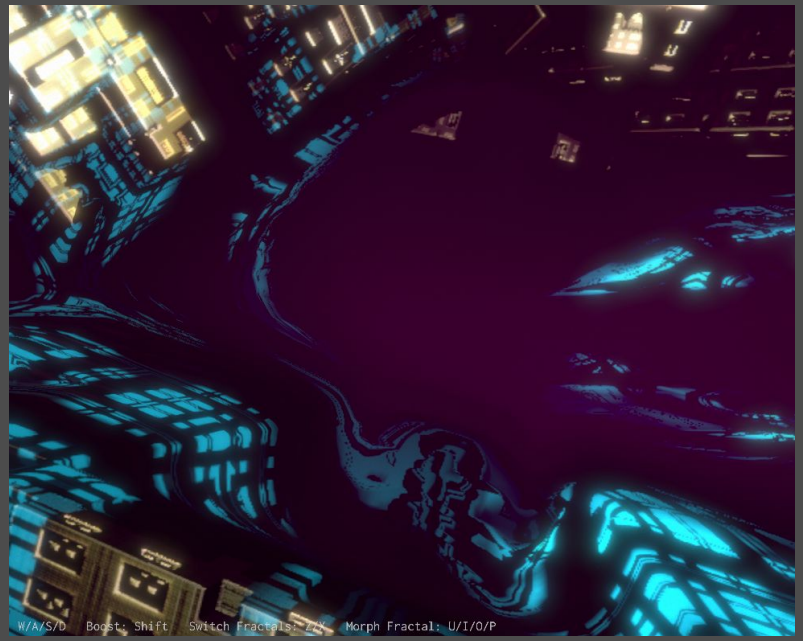
CONEMARCHING

- Originally from the demoscene
  - Fractus by Fulcrum, Revision 2012
- Idea is to progressively render the same scene at different resolutions
- On each pass, sphere trace until we can't guarantee there isn't an intersection
- Double the resolution and reuse the distance
  - Some bias is *usually* needed
- Optimizing the number of iterations and passes is not trivial. Some emergent behaviour can appear!

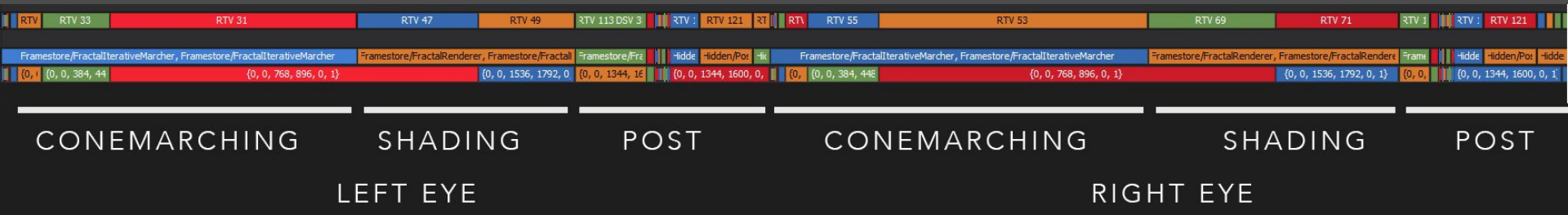


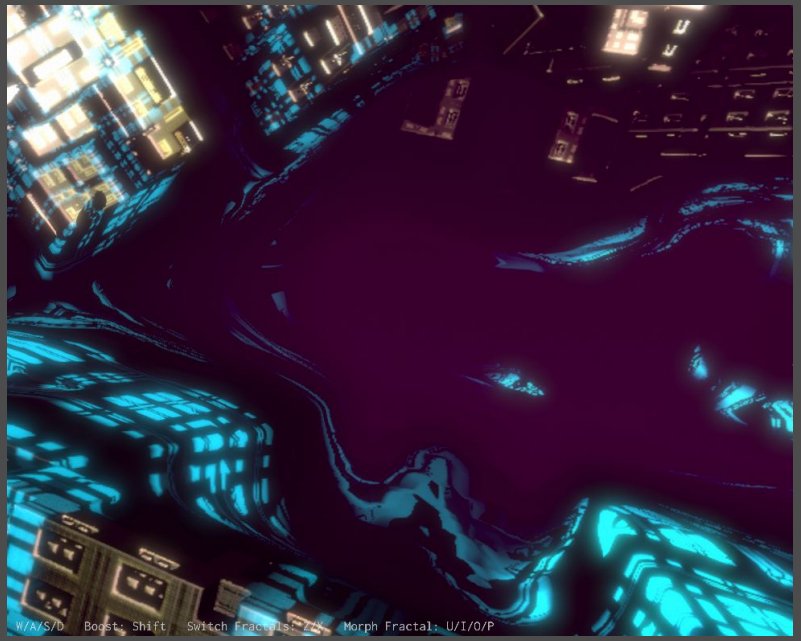
- 55-150 steps per pass
- 8 steps at highres



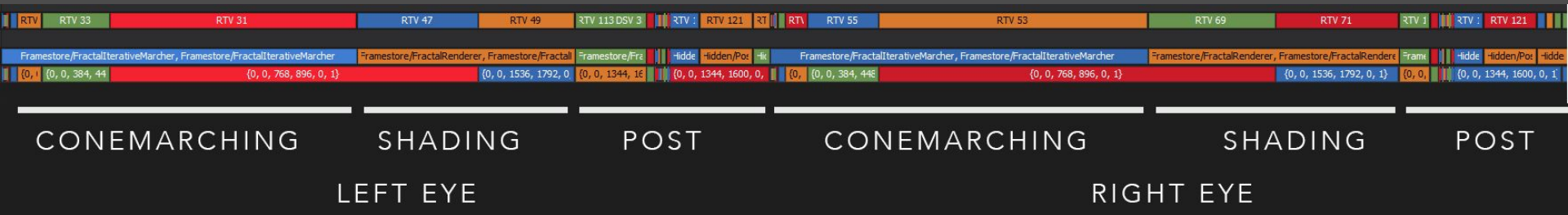


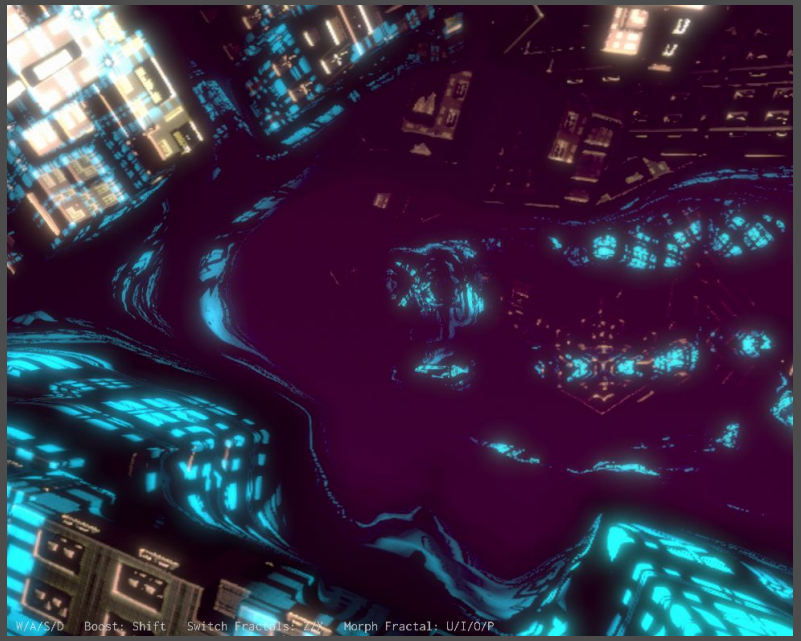
- 55-150 steps per pass
- 8 steps at highres



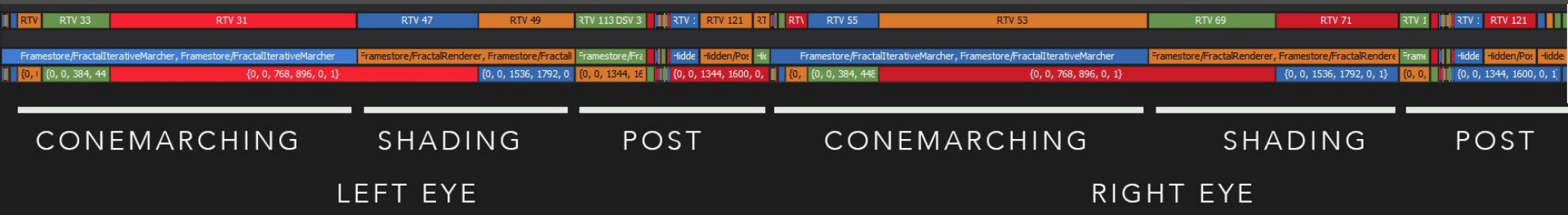


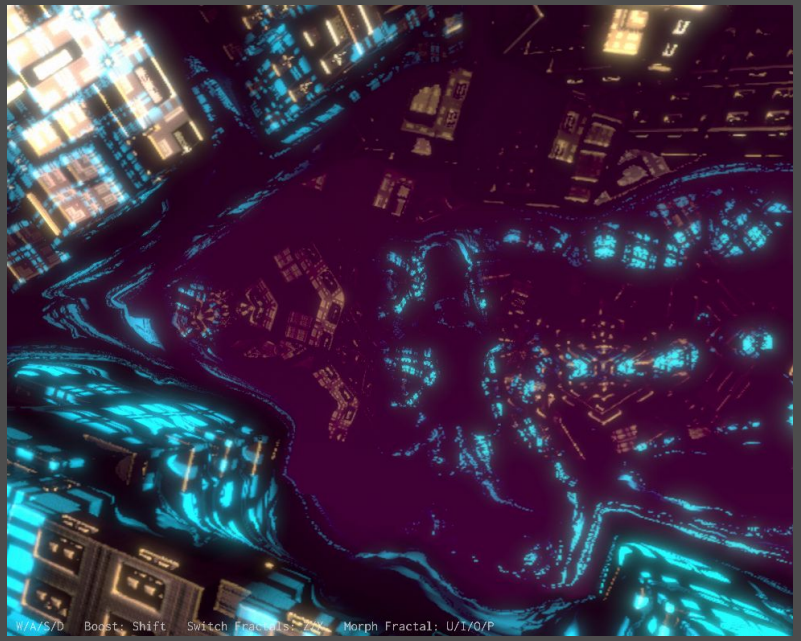
- 55-150 steps per pass
- 8 steps at highres



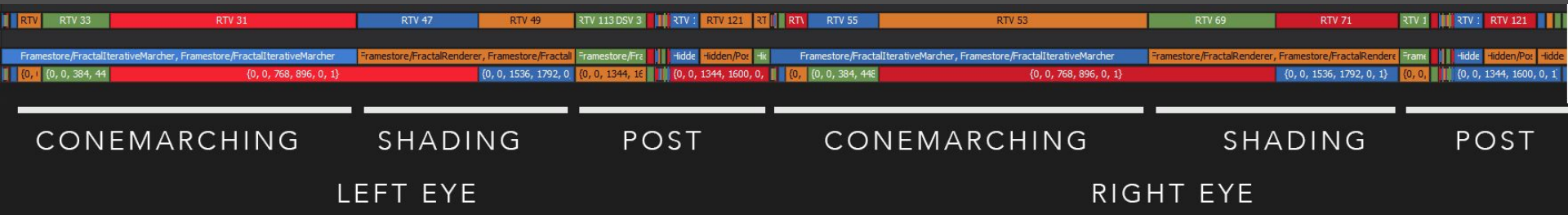


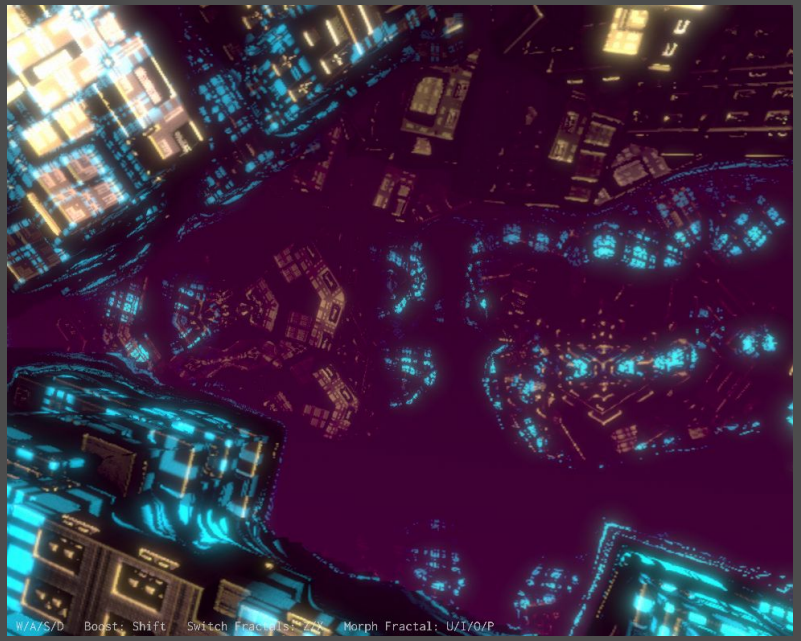
- 55-150 steps per pass
- 8 steps at highres



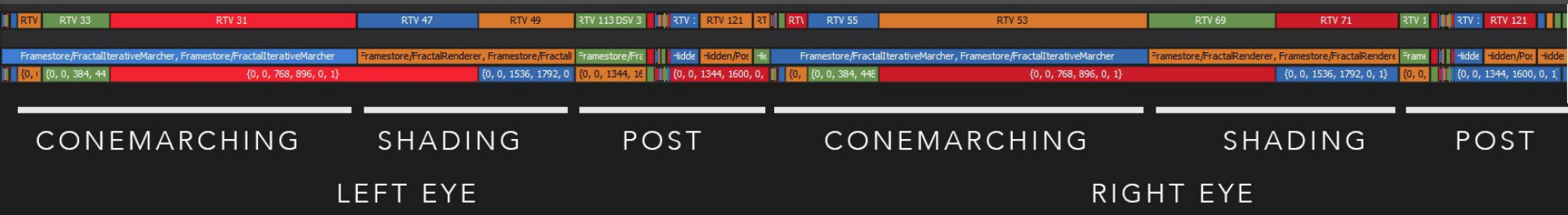


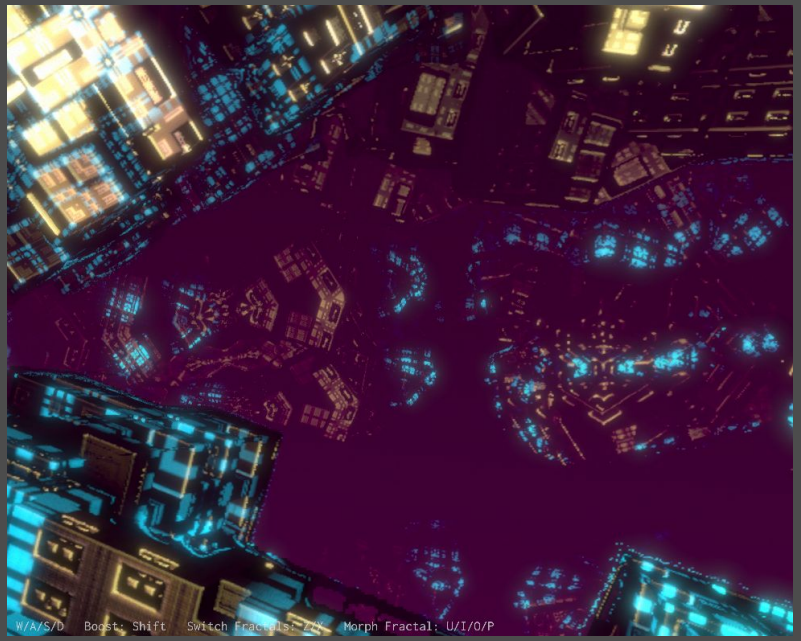
- 55-150 steps per pass
- 8 steps at highres



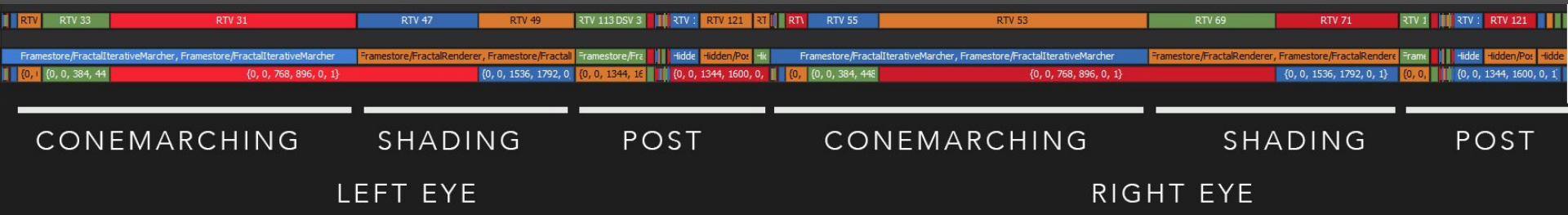


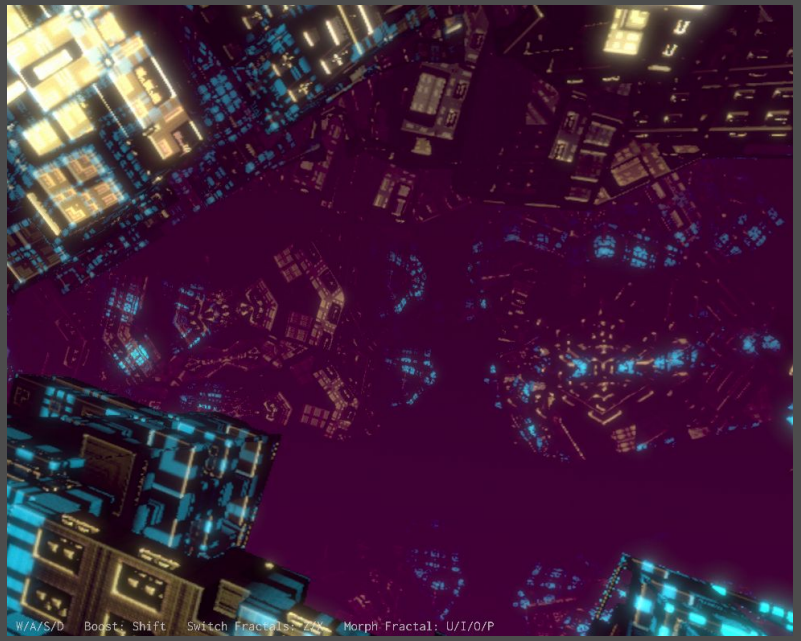
- 55-150 steps per pass
- 8 steps at highres



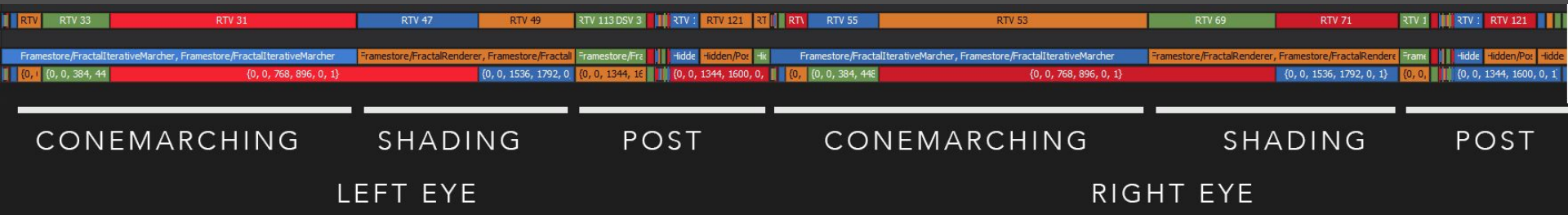


- 55-150 steps per pass
- 8 steps at highres





- 55-150 steps per pass
- 8 steps at highres

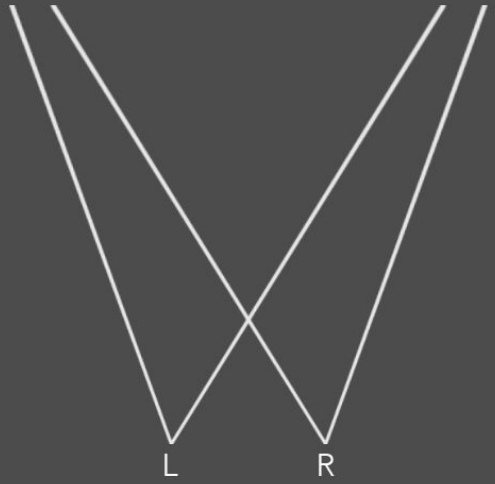


BUT…

- We’re still rendering everything twice!

REPROJECTION

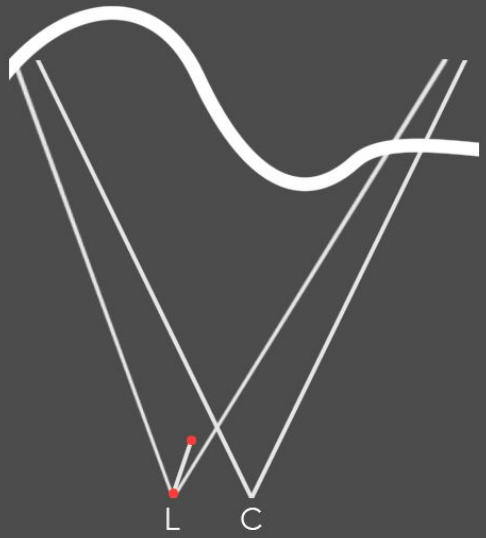
- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!



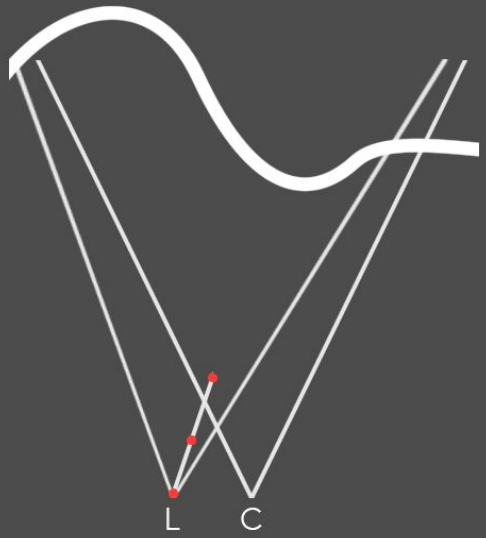
- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher

- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher

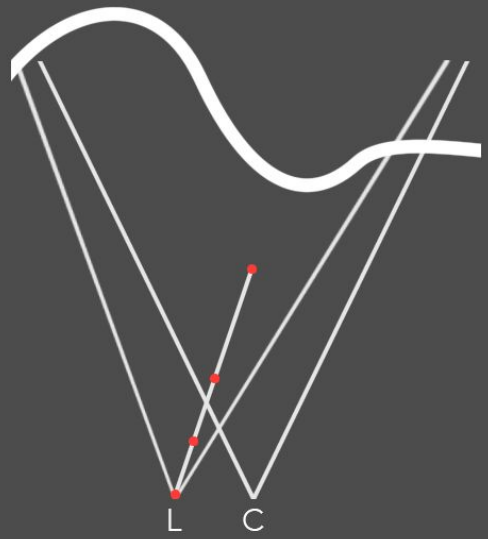
- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher
- Reproject to left and right eye



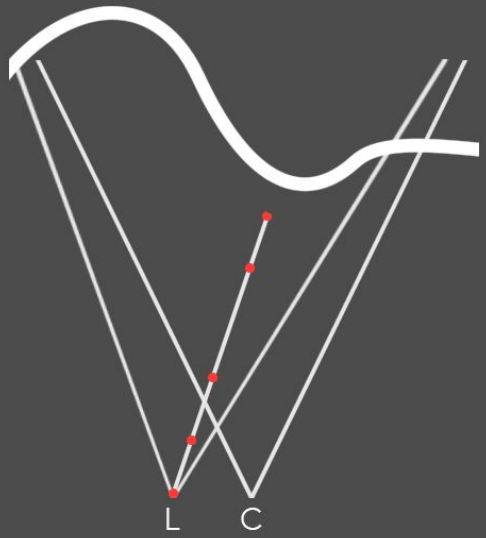
- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher
- Reproject to left and right eye
- Screen space ray marching with horizontal offset



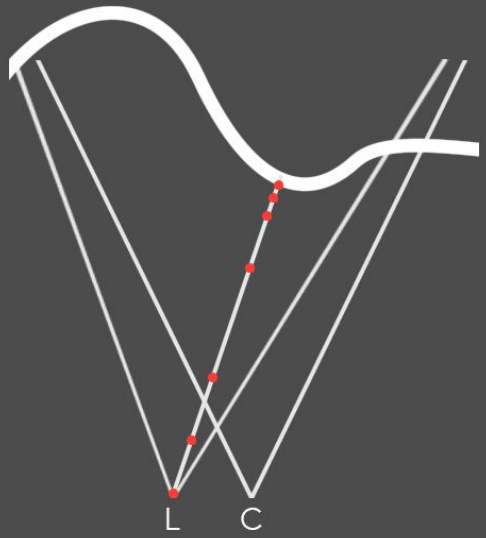
- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher
- Reproject to left and right eye
- Screen space ray marching with horizontal offset



- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher
- Reproject to left and right eye
- Screen space ray marching with horizontal offset



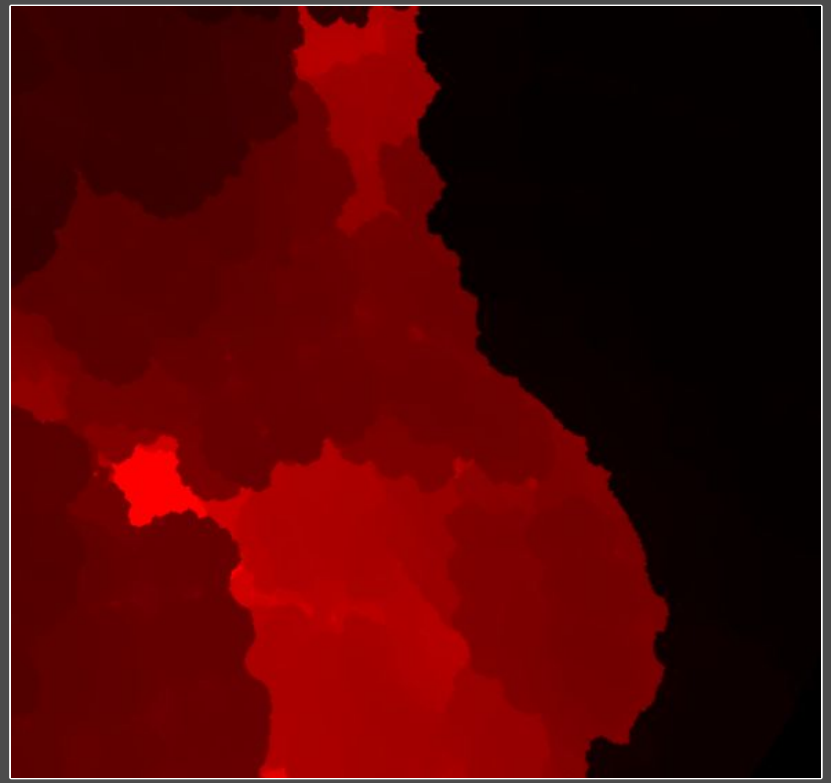
- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher
- Reproject to left and right eye
- Screen space ray marching with horizontal offset



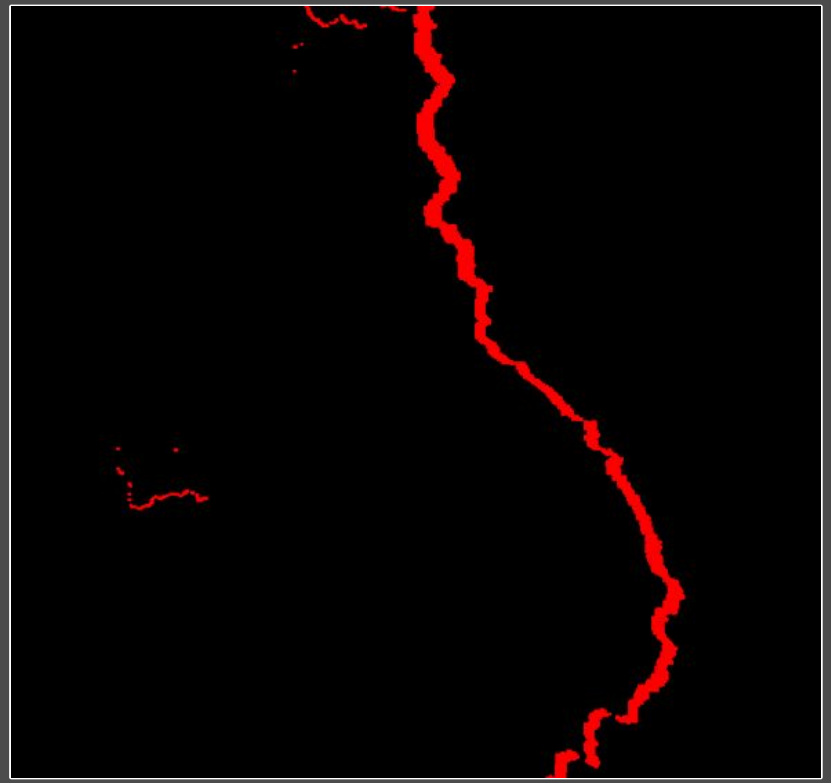
- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher
- Reproject to left and right eye
- Screen space ray marching with horizontal offset

```
// Iterate at different levels
t = GetReprojectedDistance(rayOrigin, rayDirection, t, _RaymarchingResultTex_6, 2, .2);
t = GetReprojectedDistance(rayOrigin, rayDirection, t, _RaymarchingResultTex_5, 4, .15);
t = GetReprojectedDistance(rayOrigin, rayDirection, t, _RaymarchingResultTex_3, 8, .075);
t = GetReprojectedDistance(rayOrigin, rayDirection, t, _RaymarchingResultTex_2, 4, .075);
t = GetReprojectedDistance(rayOrigin, rayDirection, t, _RaymarchingResultTex_1, 4, .075);
t = GetReprojectedDistance(rayOrigin, rayDirection, t, _RaymarchingResultTex_0, 16, 0.025);
```

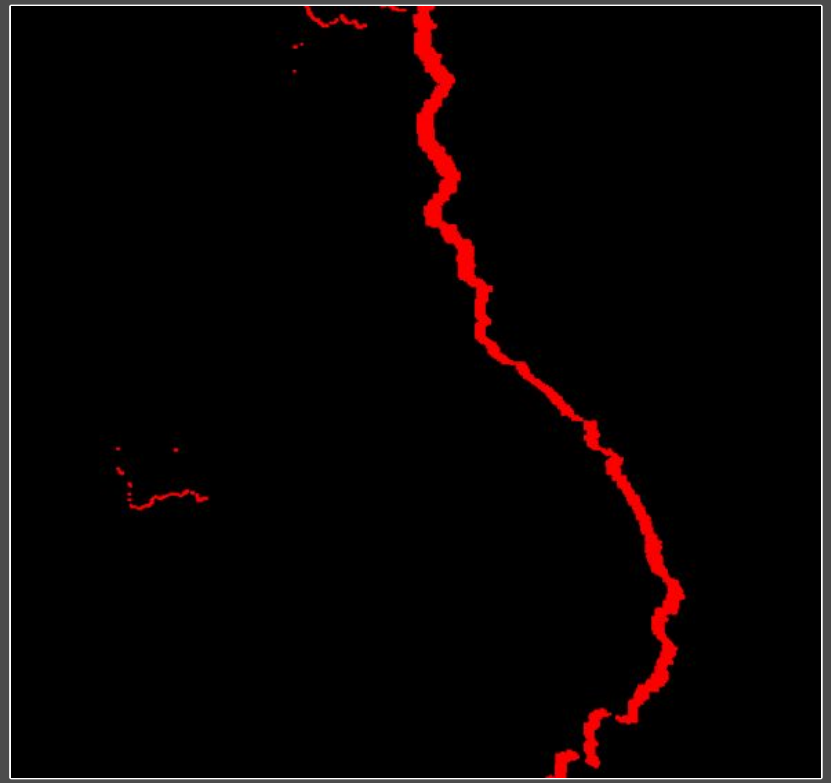
- We’re still rendering everything twice!
- Two problems: depth estimation and shading
  - Maybe we can we reproject depth? Yes!
- Render the center eye with the conemarcher
- Reproject to left and right eye
- Screen space ray marching with horizontal offset
- To get a better converging distance, we use the conemarching passes at lower res



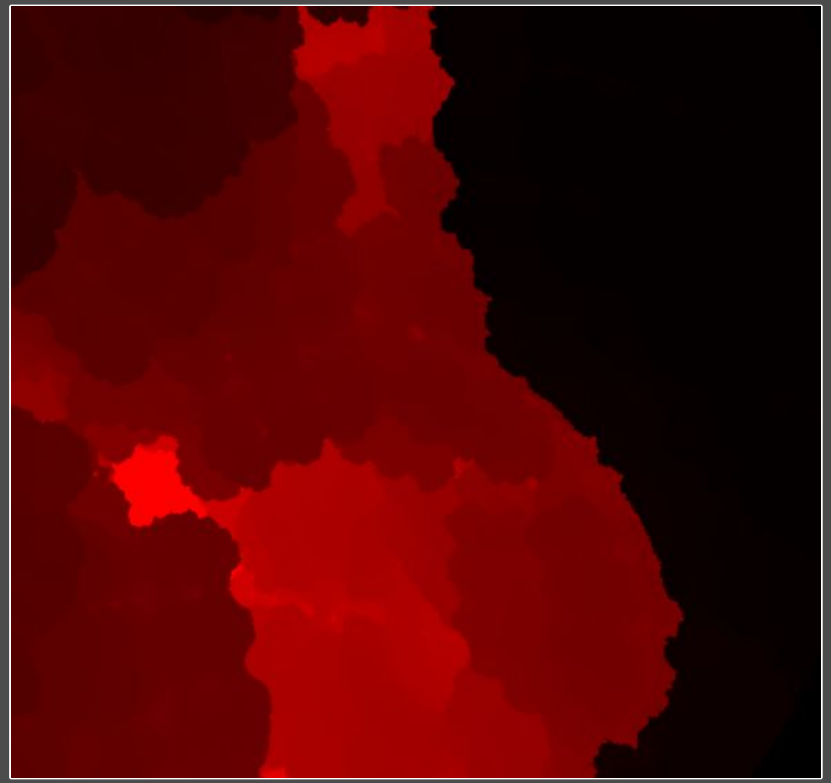
- Original paper deals with rasterization and couldn’t fix the discontinuities
  - **Fast Gather-based Construction of Stereoscopic Images Using Reprojection [van de Hoef, Zalmstra]**



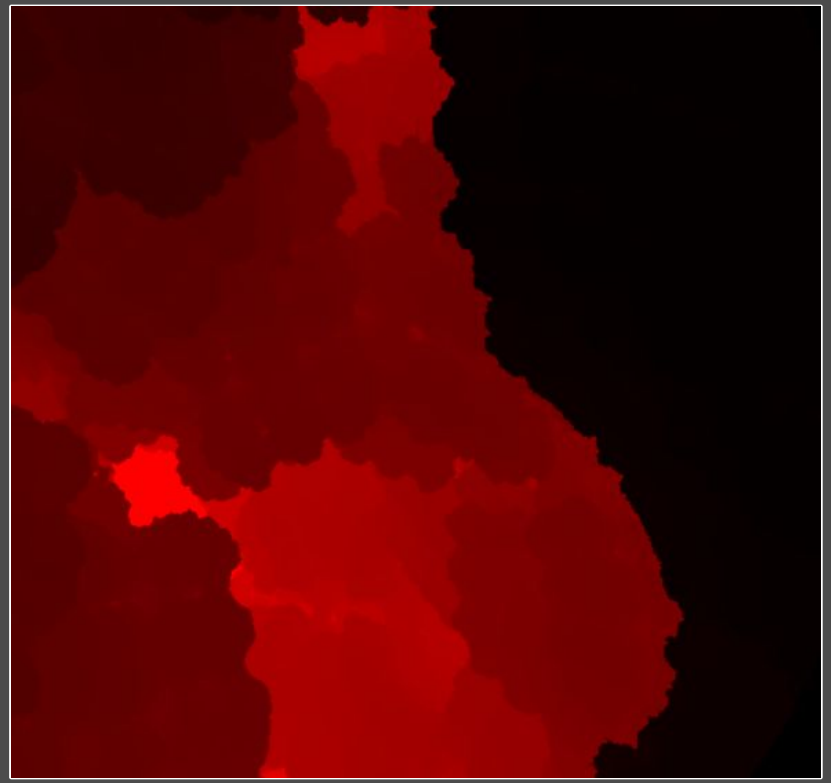
- Original paper deals with rasterization and couldn’t fix the discontinuities
- We can!
  - Just keep marching where there is a lot of stereo disparity



- Original paper deals with rasterization and couldn’t fix the discontinuities
- We can!
  - Just keep marching where there is a lot of stereo disparity
  - Also, shift samples horizontally

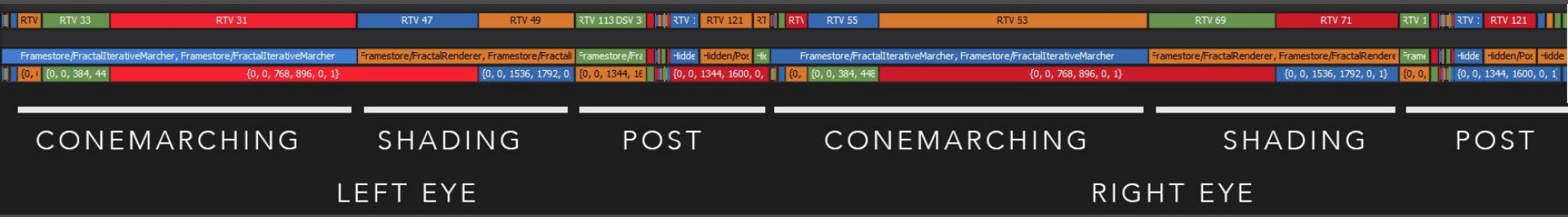


- Original paper deals with rasterization and couldn’t fix the discontinuities
- We can!
  - Just keep marching where there is a lot of stereo disparity
  - Also, shift samples horizontally
- Small artifacts, but can be masked



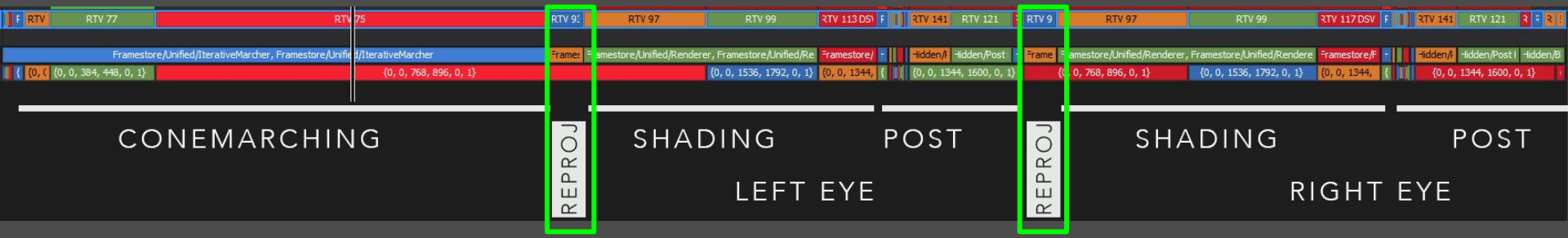
- Original paper deals with rasterization and couldn’t fix the discontinuities
- We can!
  - Just keep marching where there is a lot of stereo disparity
  - Also, shift samples horizontally
- Small artifacts, but can be masked
- Decreases quality with stereo separation
  - Fortunately, we already scale the fractal all the time, so that you’re always one unit away from the closest intersection

- With the conemarcher, we had to estimate depth twice

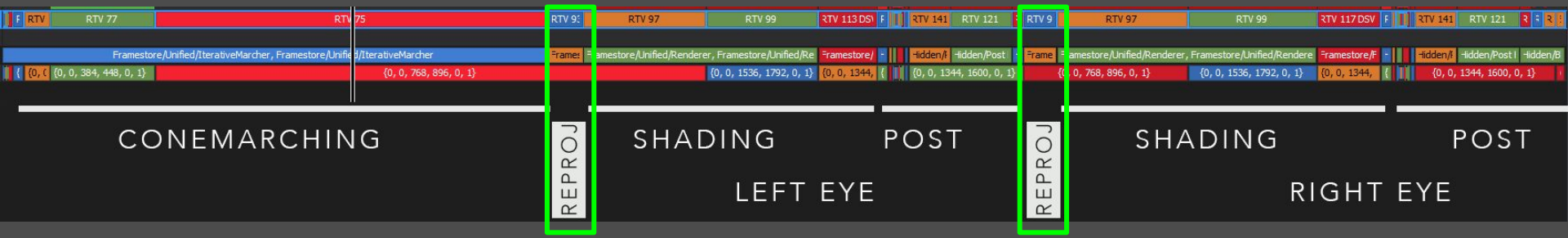


- With the conemarcher, we had to estimate depth twice
- Now we can directly cut a huge part of the pipeline

- With the conemarcher, we had to estimate depth twice
- Now we can directly cut a huge part of the pipeline
- Reprojection pass is usually fast
  - Performance improves proportionally to the conemarching pass



- With the conemarcher, we had to estimate depth twice
- Now we can directly cut a huge part of the pipeline
- Reprojection pass is usually fast
  - Performance improves proportionally to the conemarching pass
- Shading pass requires some tweaking, because some of the reprojection estimates are not perfect



FIXED FOVEATED RENDERING

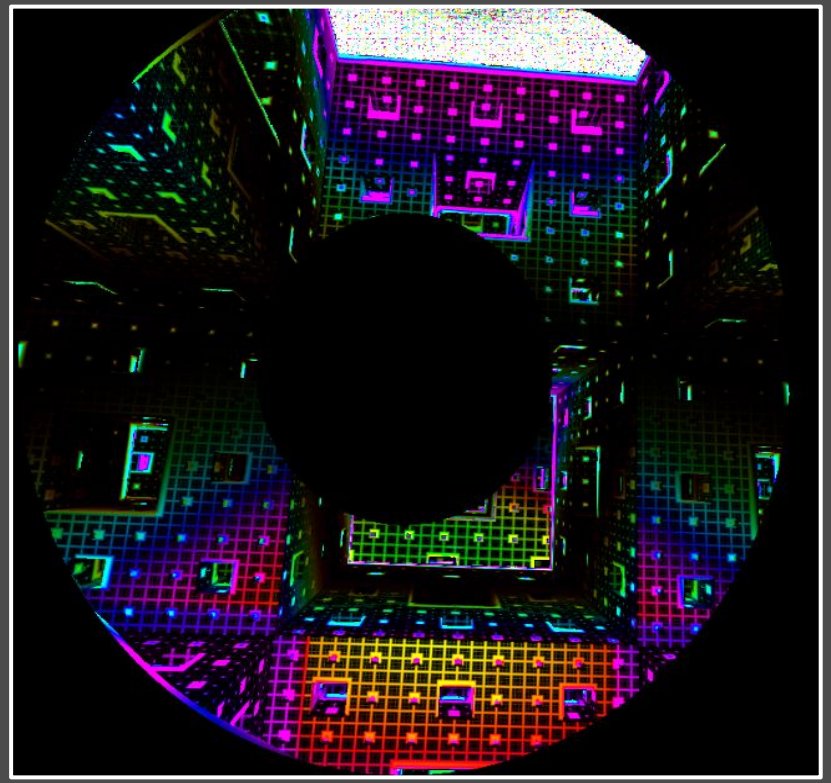
- Still expensive to render the shading

FIXED FOVEATED RENDERING

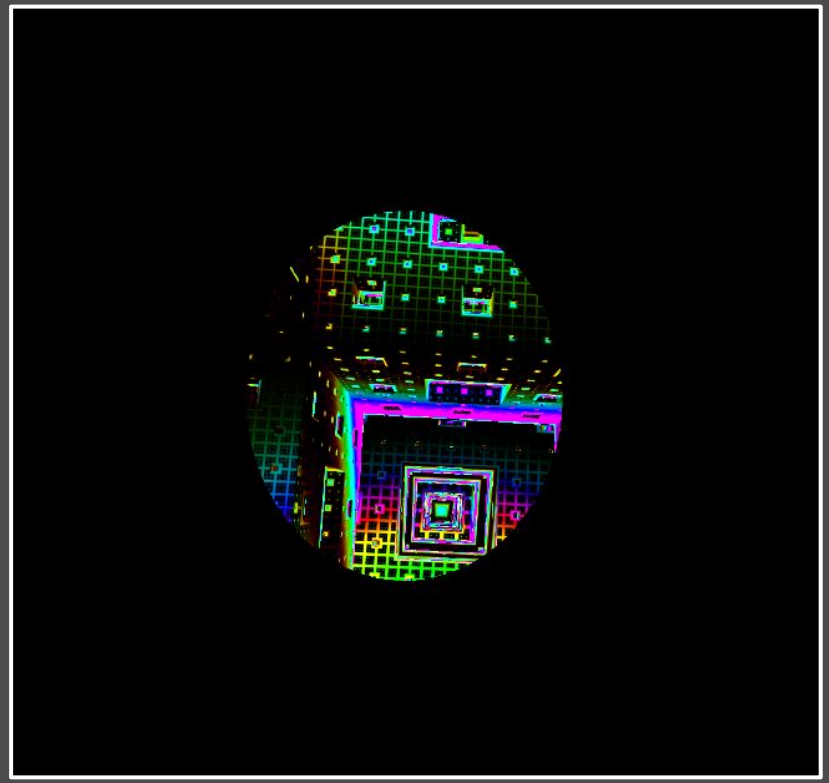
- Still expensive to render the shading
- We don’t need that much detail on the periphery

FIXED FOVEATED RENDERING

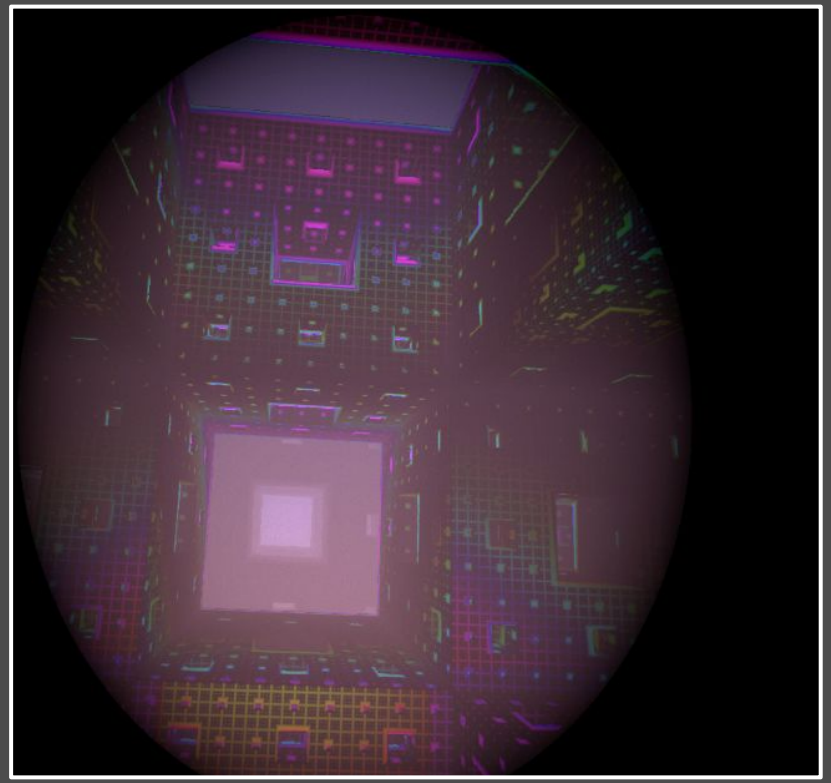
- Still expensive to render the shading
- We don’t need that much detail on the periphery
- We want something dynamic that can be scaled per fractal, and depending hardware



- Still expensive to render the shading
- We don’t need that much detail on the periphery
- We want something dynamic that can be scaled per fractal, and depending hardware
- Render at half res on the periphery, full resolution at the center and blend the edges

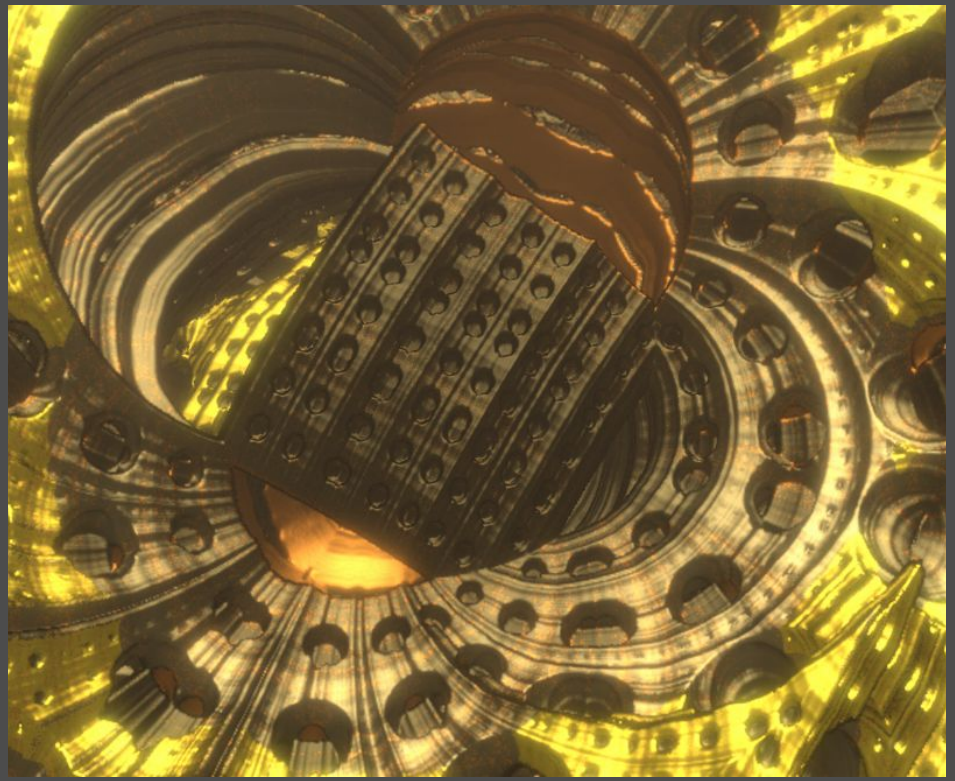


- Still expensive to render the shading
- We don’t need that much detail on the periphery
- We want something dynamic that can be scaled per fractal, and depending hardware
- Render at half res on the periphery, full resolution at the center and blend the edges

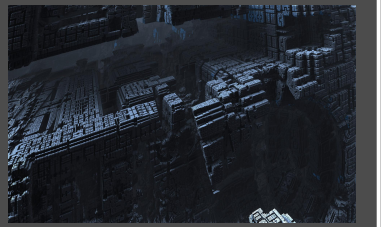


- Still expensive to render the shading
- We don’t need that much detail on the periphery
- We want something dynamic that can be scaled per fractal, and depending hardware
- Render at half res on the periphery, full resolution at the center and blend the edges
- Compose the result
  - Strong vignette saves us some computation time

THE SHADING CHALLENGE



HOW DO PEOPLE RENDER FRACTALS?



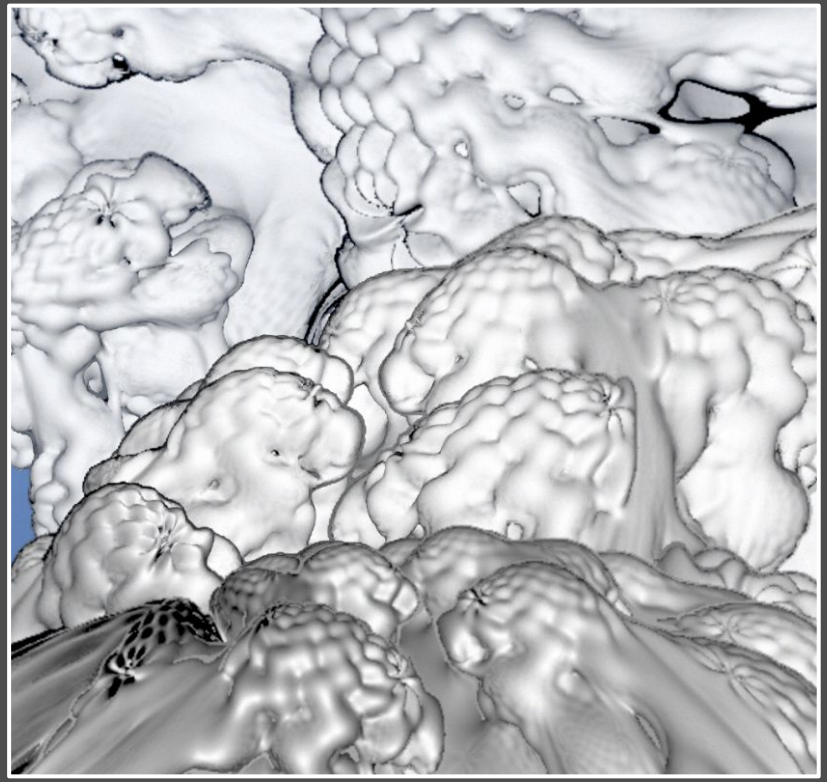
- How can we texture them?
  - We only have position and normal
- People generally use simple shading
  - Heavy use of shadows, occlusion and scattering to improve look
  - Heavy use of iteration glow
  - Orbit traps are not used cleverly
- We wanted to do something different

MORE CONSTRAINTS!

- Shadows are expensive!
- Normals are expensive!
- Occlusion is expensive!
- SSS is expensive!
- Our iteration glow is broken due to conemarching!

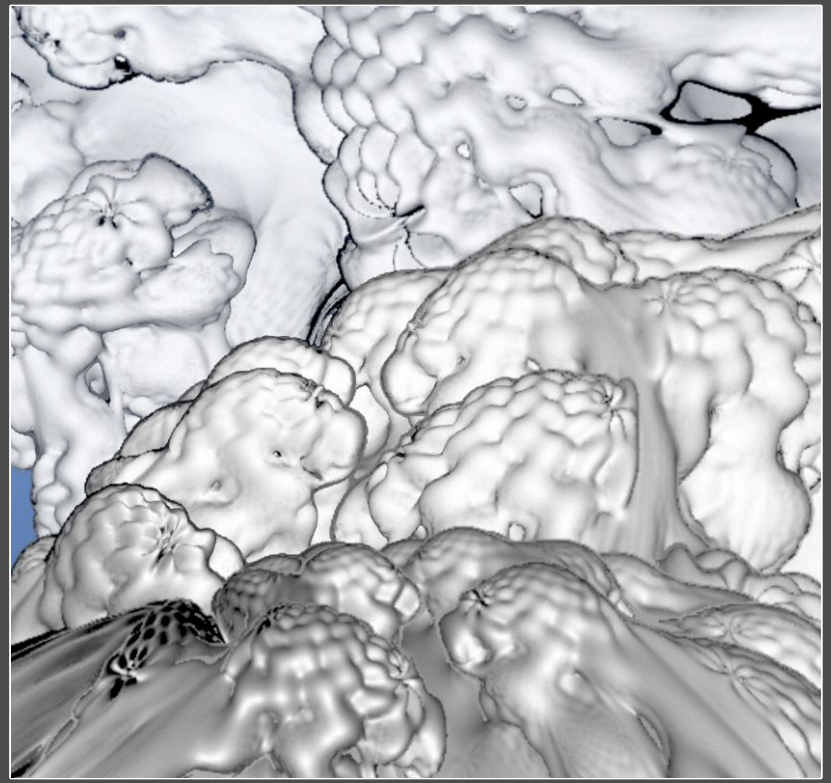
SHADING

- Experimented a lot with different ideas
  - Step the orbit traps to guide glowing elements
  - Warp the traps into themselves
  - Sample the fractal as the texture
    - Warp the fractal too!
  - Sample the derivative instead of the SDF
  - Use less iterations to fake SSS and AO
- Decided to design each fractal with a two color palette and heavy use of glowing sections.

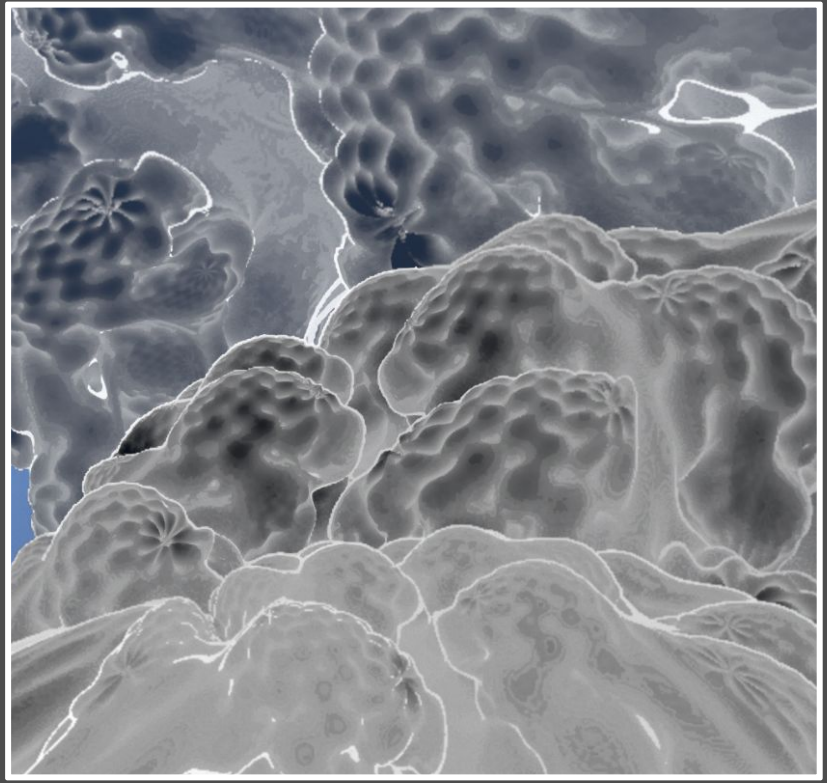


- Normal estimation generally uses finite differences

```
float3 normalEstimate(float3 p, float normalDistance)
{
    const float3 eps = float3(normalDistance, -normalDistance, 0.0);

    float dX = DEF(p + eps.xzz) - DEF(p + eps.yzz);
    float dY = DEF(p + eps.zxz) - DEF(p + eps.zyz);
    float dZ = DEF(p + eps.zzx) - DEF(p + eps.zzy);

    return normalize(float3(dX,dY,dZ));
}
```

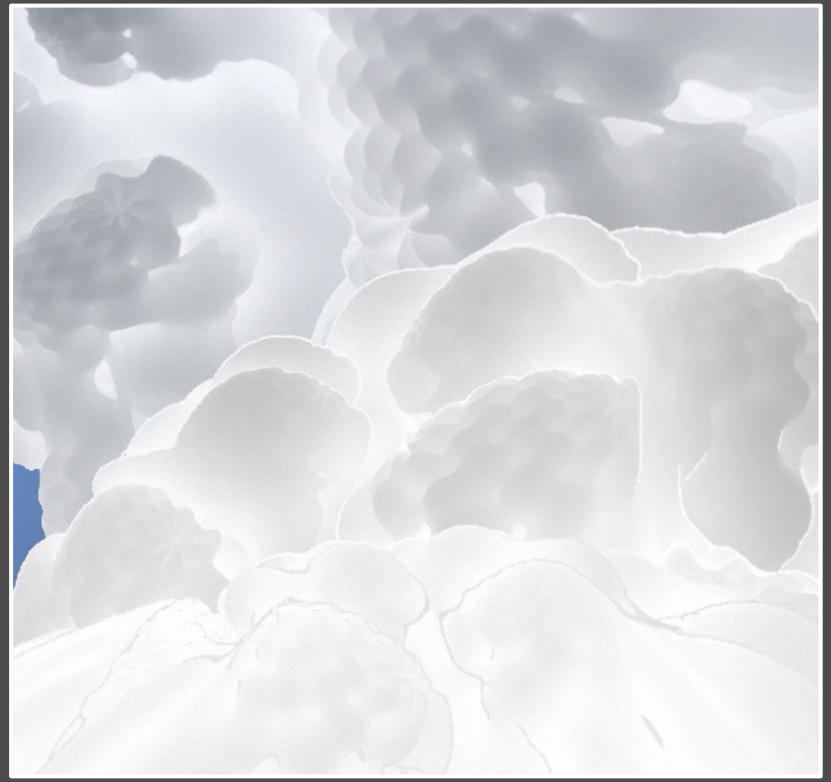


- Normal estimation generally uses finite differences
- Requires 6 samples of the sdf
  - Can be reduced to 3 by losing some quality

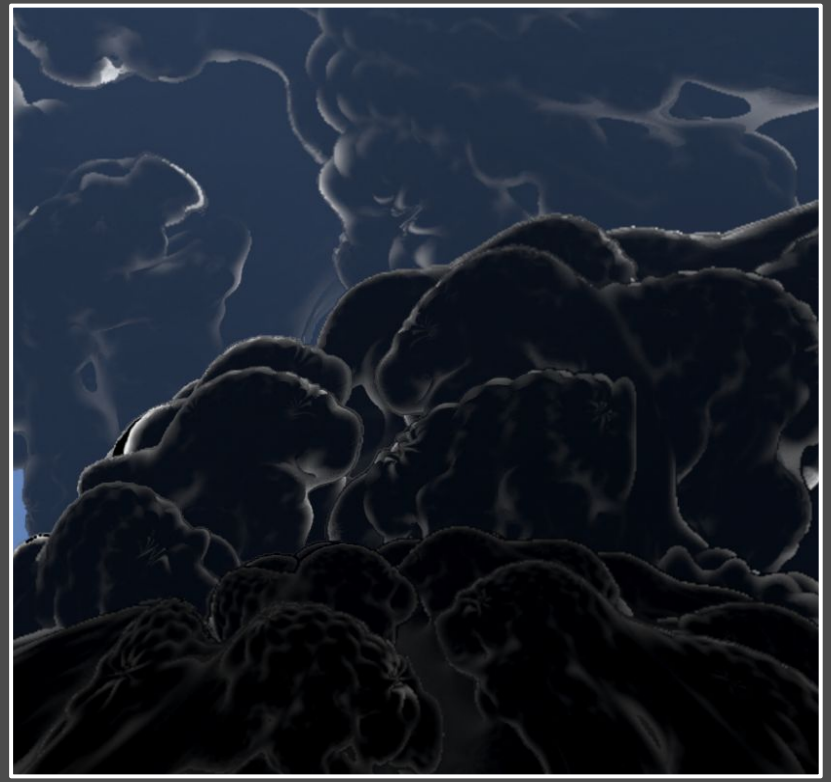


- Normal estimation generally uses finite differences
- Requires 6 samples of the sdf
  - Can be reduced to 3 by losing some quality
- Iq proposed on pouet.net a way to fake diffuse lighting with only one sample

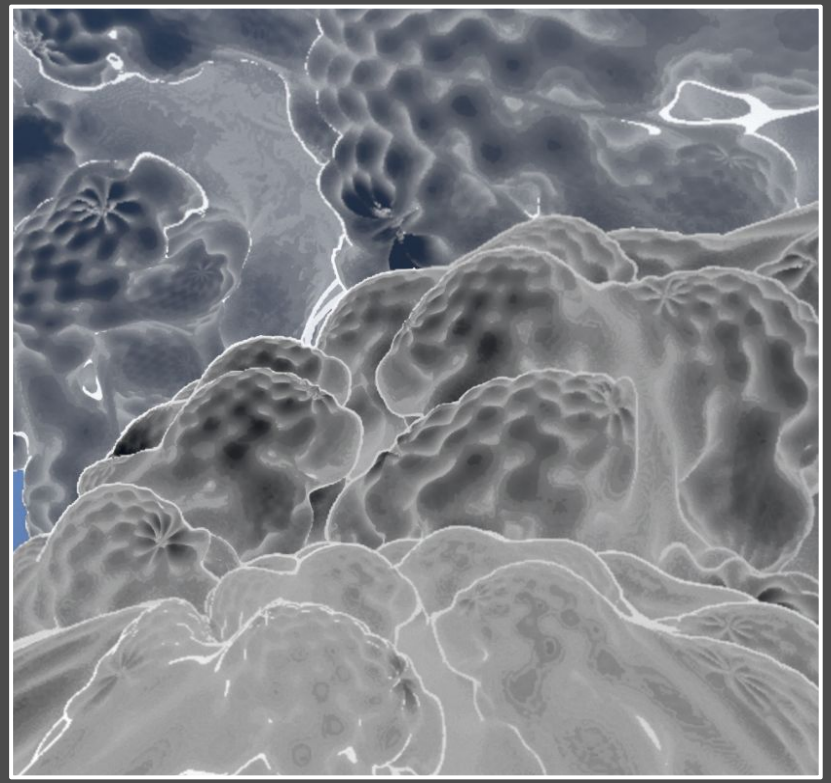
```
// Fast cosine estimation, reference: http://www.pouet.net/topic.php?which=7535&page=1
float estimateCosTheta(float3 pos, float3 lightDir, float eps)
{
    eps *= 2.25;
    float d = DEF(pos + lightDir * eps) / eps;
    return gain(d, .2);
}
```



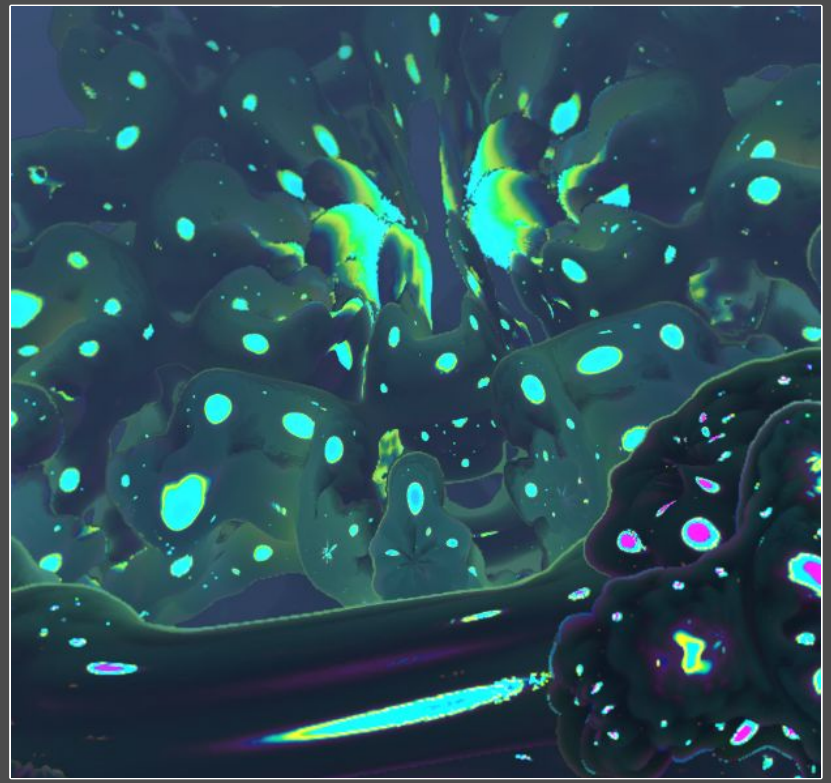
- Normal estimation generally uses finite differences
- Requires 6 samples of the sdf
  - Can be reduced to 3 by losing some quality
- Iq proposed on pouet.net a way to fake diffuse lighting with only one sample
- Apply this trick to everything!
  - Reasonable fake AO!
  - Reasonable fake SSS!
  - Rim lighting?



- Normal estimation generally uses finite differences
- Requires 6 samples of the sdf
  - Can be reduced to 3 by losing some quality
- Iq proposed on pouet.net a way to fake diffuse lighting with only one sample
- Apply this trick to everything!
  - Reasonable fake AO!
  - Reasonable fake SSS!
  - Rim lighting?

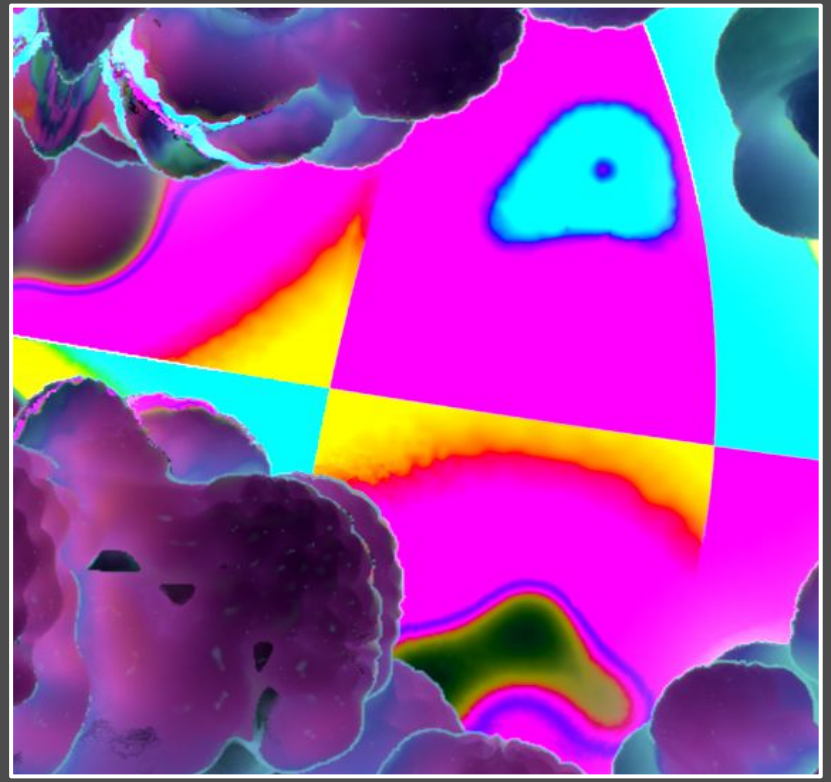


- Normal estimation generally uses finite differences
- Requires 6 samples of the sdf
  - Can be reduced to 3 by losing some quality
- Iq proposed on pouet.net a way to fake diffuse lighting with only one sample
- Apply this trick to everything!
  - Reasonable fake AO!
  - Reasonable fake SSS!
  - Rim lighting?
- Has some banding problems due to fractal epsilon
  - We still needed a proper normal for some fractals

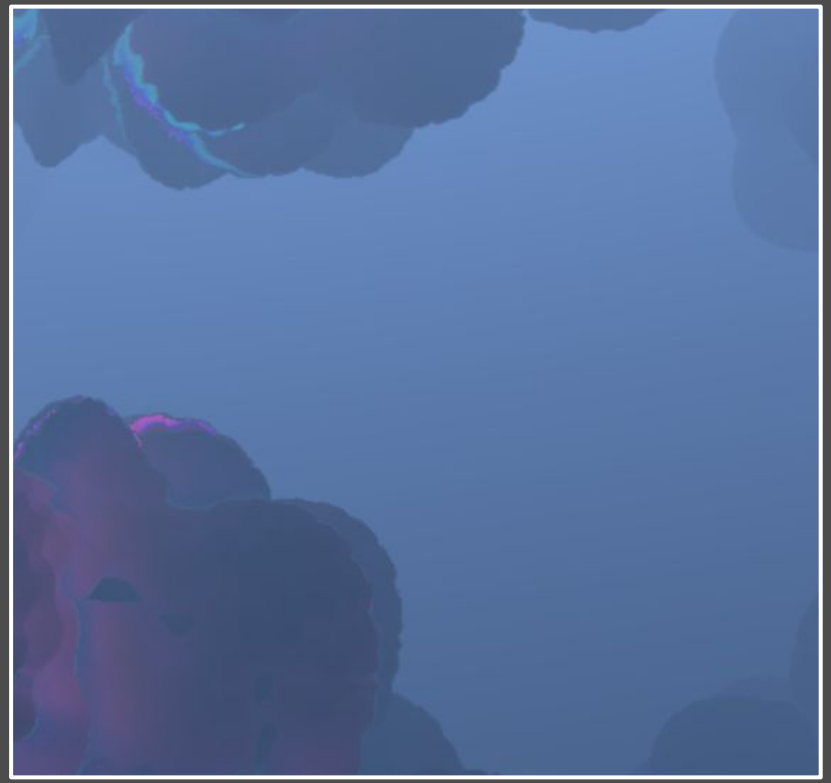


- Moving a little back to the camera instead generates a very interesting pattern
- Use an animated cosine palette that scrolls based on this function
- Heavy use of fake SSS
- Use of lower iteration Mandelbulb code for AO

```
75
76      float e = eps * 2.25;
77      float d = DEMandelbulb(pos + dir * e) / e;
78
```



- Don’t render very far

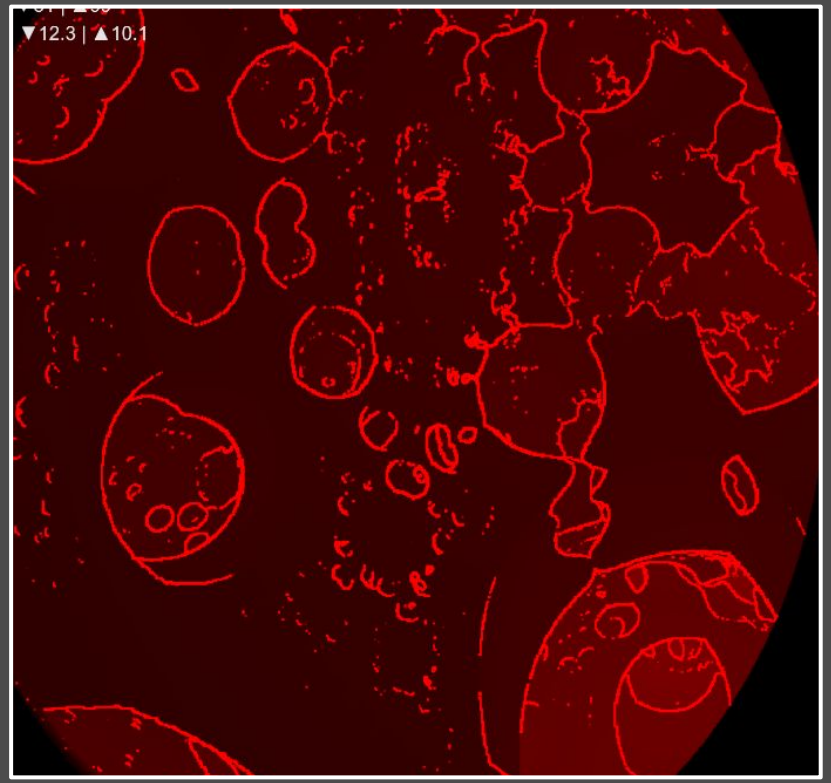


- Don’t render very far
- Use homogeneous scattering to hide it
  - This helps immersion, fortunately!

Acquiring Scattering Properties of Participating Media by Dilution
https://cseweb.ucsd.edu/~ravir/dilution.pdf

FUTURE WORK
(SHORT TERM)

- Temporal reprojection of depth estimation
- Reprojection of low frequency effects
- Use screen space normals to reduce shading complexity
  - Use lower quality shading on periphery
- Implement TAA + FXAA with the outlines provided by the stereo reprojection disparity



- Temporal reprojection of depth estimation
- Reprojection of low frequency effects
- Use screen space normals to reduce shading complexity
  - Use lower quality shading on periphery
- Implement TAA + FXAA with the outlines provided by the stereo reprojection disparity

FUTURE WORK
(LONG TERM)

- Implement stereo reprojection for shading (Oculus' approach)
- Optimize ray scheduling to maximize occupancy
  - **GPU Unchained (Timothy Lottes) from NVScene 2015**
- Experiment with low resolution voxel grid to accelerate depth estimation and other effects
- Alleviate strong aliasing artifacts

Thank you

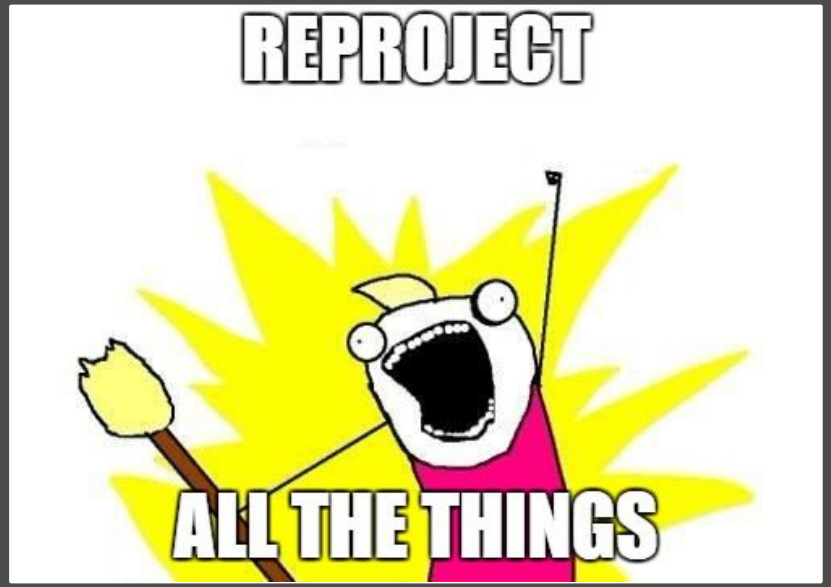
coral-vr.com

Johannes Saam
johannes.saam@framestore.com | @johannessaam

Mariano Merchante
mariano.merchante@framestore.com | @mmerchante

/ Framestore

BONUS!



- We liked the reprojection results
- Can we reproject low frequency image effects?
  - Glow
  - Volumetric scattering (godrays)
- Still working on it!

TEMPORAL REPROJECTION

- Can we reuse the previous frame's distance estimation?
  - TAA deals with similar ideas, but only for shading

TEMPORAL REPROJECTION

- Can we reuse the previous frame's distance estimation?
  - TAA deals with similar ideas, but only for shading
- Use motion vectors to predict positions

TEMPORAL REPROJECTION

- Can we reuse the previous frame's distance estimation?
  - TAA deals with similar ideas, but only for shading
- Use motion vectors to predict positions
- Can't use simple eye reprojection technique because delta is both positional and rotational
  - Scatter vs gather

TEMPORAL REPROJECTION

- Can we reuse the previous frame's distance estimation?
  - TAA deals with similar ideas, but only for shading
- Use motion vectors to predict positions
- Can't use simple eye reprojection technique because delta is both positional and rotational
  - Scatter vs gather
- The conemarching passes serve as lower bounds
  - So we can grab (dis)appearing objects!

TEMPORAL REPROJECTION

- Can we reuse the previous frame's distance estimation?
  - TAA deals with similar ideas, but only for shading
- Use motion vectors to predict positions
- Can't use simple eye reprojection technique because delta is both positional and rotational
  - Scatter vs gather
- The conemarching passes serve as lower bounds
  - So we can grab (dis)appearing objects!
- We're still working on this!