Advanced Material Rendering

Michał Drobot
Visual Technical Director
Reality Pump

Advanced Materials

- State of material rendering
  Several techniques from the 'old' toolbox
  - Diffuse + Specular + Normal + Phong
  - Parallax
  - Fur / Shell rendering
  - Alpha blending
  - Cube maps
    - IBL
    - Reflections / Refractions / Glossy Specular

Advanced Materials

- Material rendering stucked
    - Those techniques doesn't work right with current deferred rendering architectures
  - Deferred shading
    - Brings global light-material interaction shaders
      - Requires uniform BRDF across all materials during shading pass
    - Really fast
      - Requires one geometry pass
      - Fat G-Buffer might hurt the bandwidth
    - Lacks material variety
    - Adding different material support seriously hurts the speed
    - Alpha blending must be done in forward pass

Advanced Materials

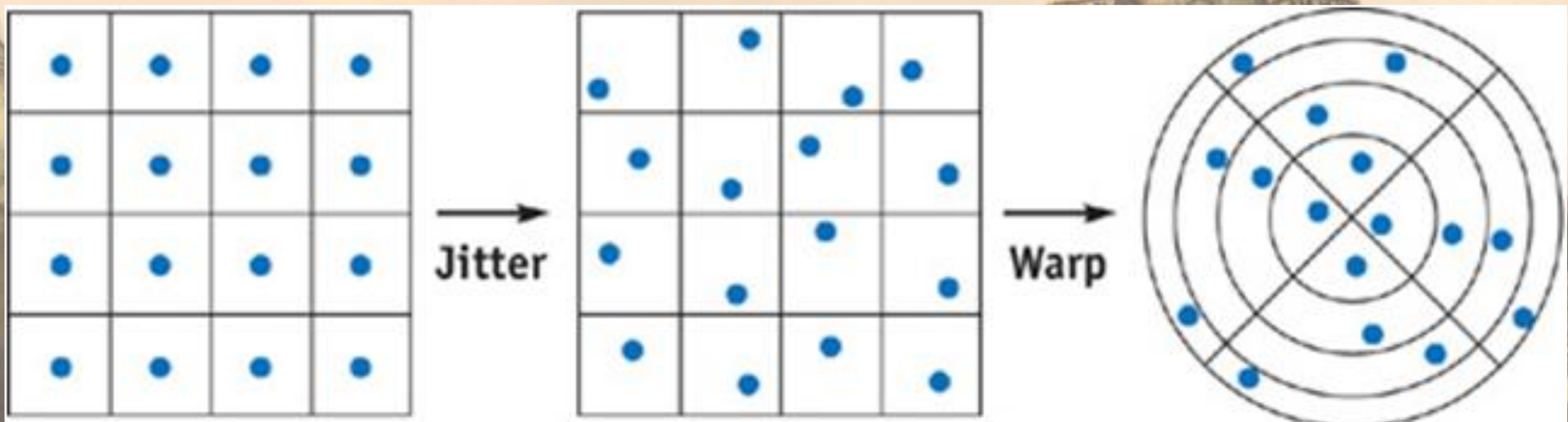
- Material rendering stucked
  - Light pre-pass
    - Requires double geometry pass
      - 'light' g-buffer
        - Normal + Z
      - Material pass
        - Renders invidual meshes with custom material shaders
        - Use light information gathered in light buffer, created from 'light' g-buffer
    - Allows usage of many different material shaders
      - Unified light interaction
    - Alpha blending must be done in forward pass

Advanced Materials

- We want a new toolbox
  - Compatible with deferred renderers
  - More advanced techniques

Jittering tricks

- Jittering
  - Sampling in a pattern to cover undersampling in more plausible noise
  - Normally done using ‘rotating disk’ of sample offset distribution
    - Uniform
    - Poisson



Jittering tricks

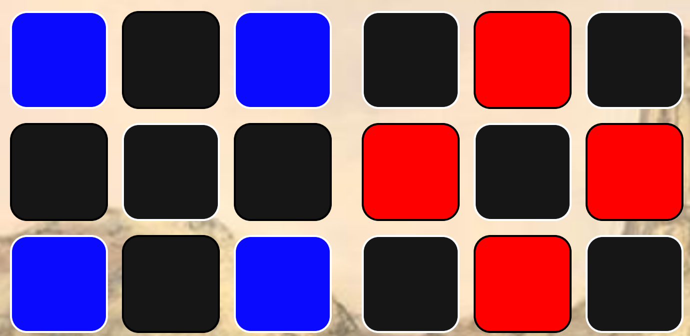
- Jittering using rotating disk
  - Precompute a good offset distribution table
    - N points in normalized space using disk distribution
  - For each shaded pixel
    - Get random normal vector N
    - For each sample
      - Rotate the point from the disk distribution by N
      - Sample using the point as the scaled offset
- Because of non-discrete sampling point, linear sampling is important

Jittering tricks

- Jittering using alternating pattern
  - What if we can't afford additional noise lookup, ALU per sample and linear filtering
  - We need carefull manual sampling pattern
  - We know the exact pixel position from VPOS
    - With that we can use dithering pattern
    - With different pixels we use different pattern
    - Used patterns cover different samples

Jittering tricks

- Jittering using alternating pattern
  - Example
    - Let's have 2 different sampling patterns
    - Together they cover the full sampling area with dither
    - We use different for even and odd pixels
      - Cover the whole region with 2 times less samples
      - Removes banding by adding controlable noise pattern



Jittering tricks

- Jittering using alternating pattern
- Shadowing example
  - Dual paraboloid soft shadows
  - 4 taps only
  - Minimal additional overhead
  - Plausible noise
  - Bigger softness requires more patterns

```
float4 tex2DSHDWPCF(sampler2D tex, float4 UV, float2 vP)
{
 const float4 gPCFJitter1[2] =  {
       float4(0.5, 0.0, -0.5,  0.0),
       float4(0.5, 0.5, -0.5, -0.5), };
 const float4 gPCFJitter2[2] =  {
       float4(0.0,  0.5,  0.0, -0.5),
       float4(0.5, -0.5, -0.5,  0.5), };

 float4 Samples;
 float  Index  = (vP.x + vP.y) % 2;
 float  JitDis = 0.003 * (1.0 + 2.0 * (frac(dot(UV.xy,
165697.0)) - Index * 0.5));

 float4 tC1 = gPCFJitter1[Index] * JitDis;
 float4 tC2 = gPCFJitter2[Index] * JitDis;

 tC1 += UV.xyxy;
 tC2 += UV.xyxy;

/…/
}
```

Transparency

- Transparency in deferred architecture is tricky
- Scenarios
  - Simple transparency (lit)
  - Fully transparent material
  - Semi-Transparent material (lit)
  - Translucent material (always lit)

Simple transparency

- Simple transparency
  - Think of simple fade in, fade out
    - Sometimes needed when objects get in our camera view (think leaves...)
    - Grass blend in/blend out
    - Objects popping in
  - Must be cheap and coherent with lighting

Simple transparency

- Simple transparency
  - Use screen door effect
    - Compute/lookup dithering patterns
    - Use them to 'kill' pixels
    - Alternate between patterns depending on transparency value
  - 4 level transparency easy to compute when bandwidth bound
    - Remember to check were the compiler is putting your 'kills' – should do it ASAP

Simple transparency

```
float jitteredTransparency(float alpha, float2 vP)
{
 const float jitterTable[4] =
 {
      float( 0.0 ),
      float( 0.26 ),
      float( 0.51 ),
      float( 0.76 ),
 };

 float jitNo = 0.0;
 int2  vPI   = 0;
         vPI.x = vP.x % 2;
         vPI.y = vP.y % 2;

 int jitterIndex = vPI.x + 2 * vPI.y;

 jitNo = jitterTable[jitterIndex];
 if (jitNo > alpha)
       return -1;

 return 1;
}
```

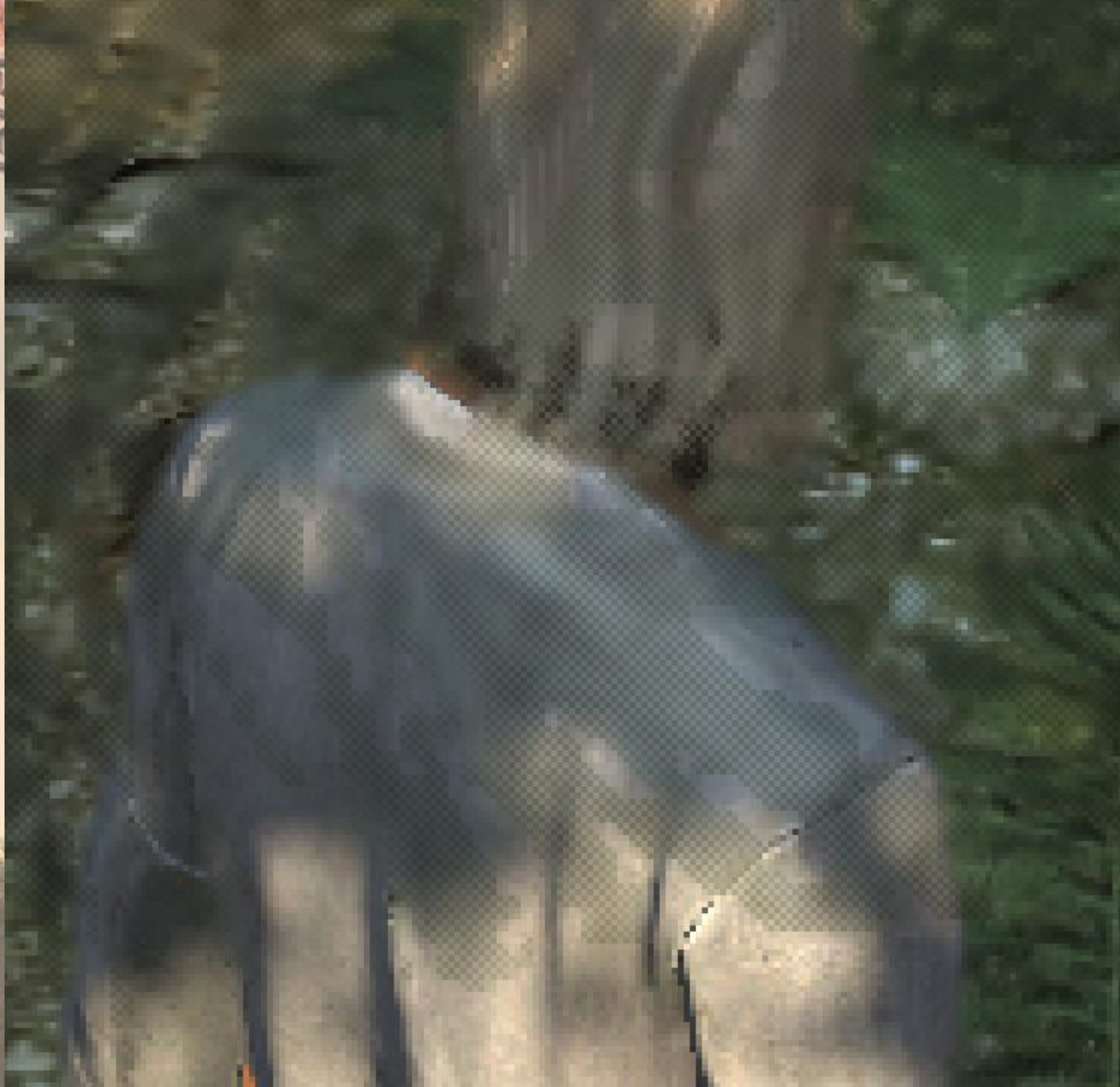

0%
25%
50%
75%
100%

Simple transparency

- Simple transparency
  - Dithered transparency looks bad in 720p
    - We would like to blur those nasty dithered pixels
    - Can't afford another pass that would detect them and blur
  - We are already doing it in Edge AA pass

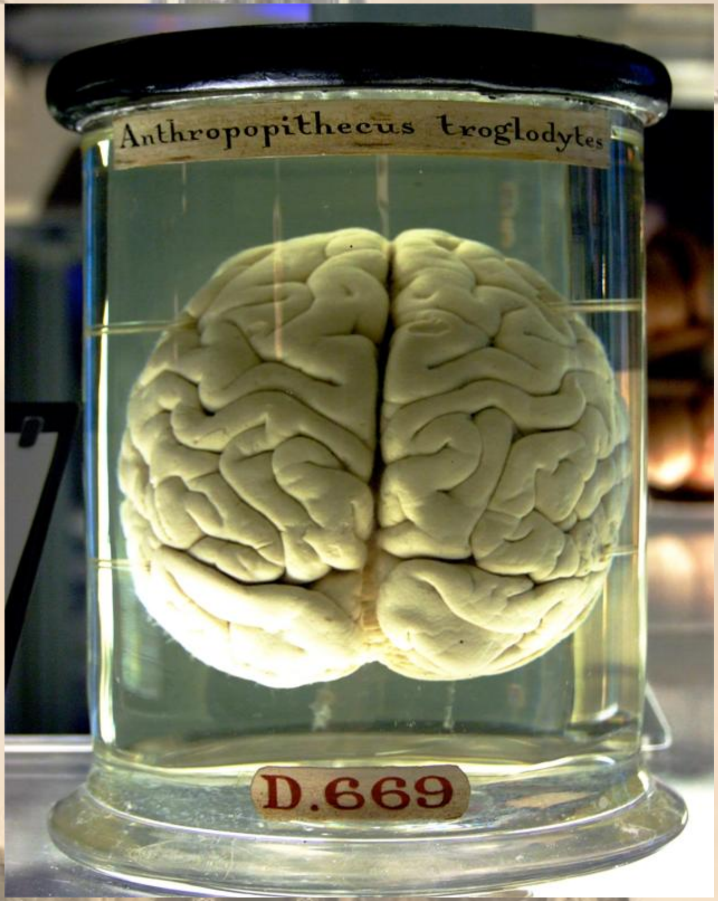
Simple transparency

- Custom Edge AA
  - Common technique in deferred renderers
  - Full screen pass
    - Find edges based on depth/normal data
    - Blur them
  - Can use it to our advantage
  - Just hint the Edge AA filter to find edges 'between' the killed pixels
    - You get nice blending for free
    - Could be done with a flag or more hacky by altering the source of edge detection (put discontinioutis in depth)



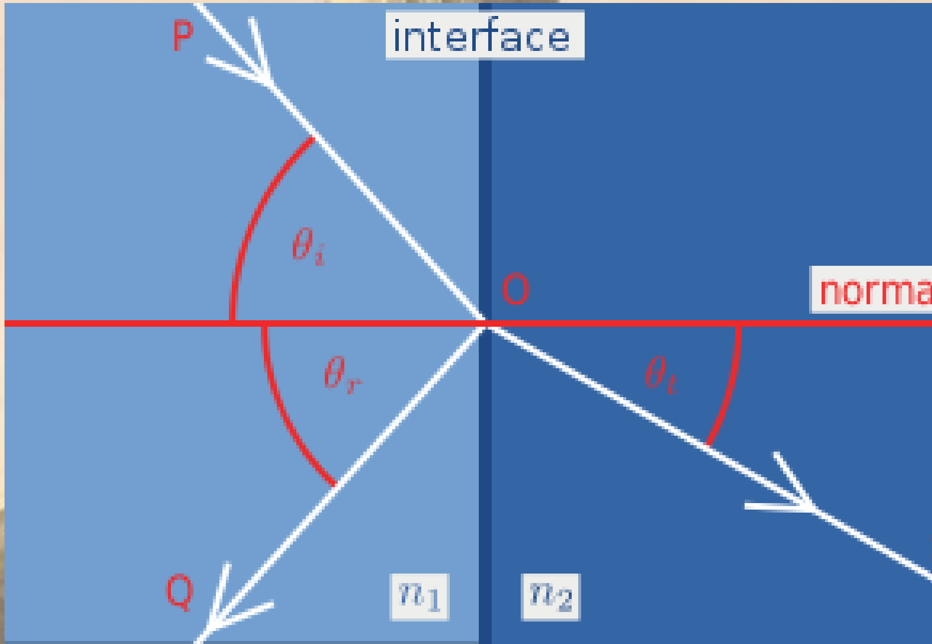
Fully transparent

- Fully transparent
  - Doesn’t need lighting
    - Just reflects / refracts light
  - Usefull for
    - Glass
    - Water
    - Distortion particles
  - Treated as post-effect
    - Requires backbuffer as a texture
    - Handy to have depth information in Alpha channel



Fully transparent

- Refraction
  - Use the eye vector
  - Refract it physically against surface normal
  - Project on backbuffer and read
  - Use refraction masking
    - Gpu Gems 2



Fully transparent

- Reflection
  - Treat the backbuffer as a spherical map
  - Reflect the eye vector against surface normal
  - Use spherical mapping for outgoing vector
    - We spherically map the backbuffer to fake RT reflection
  - Sample the backbuffer
    - Or some smaller – blured version for glossy relfeciton
  - Hacky
    - Looks quite convincing
  - Use dual-paraboloid enviromental map for quality

Advanced materials

- Glass
  - Fully transparent material
    - Rendered in post
  - Reflection - Refractions surface
    - Follows fresnel law
      - Mix reflection with refraction depending on angle beetwen eye vector and surface normal
    - Use fake real time reflection
    - Use backbuffer for refraction
      - Can use blurred backbuffer for glossiness and translucency approximation

Semi-Transparent material

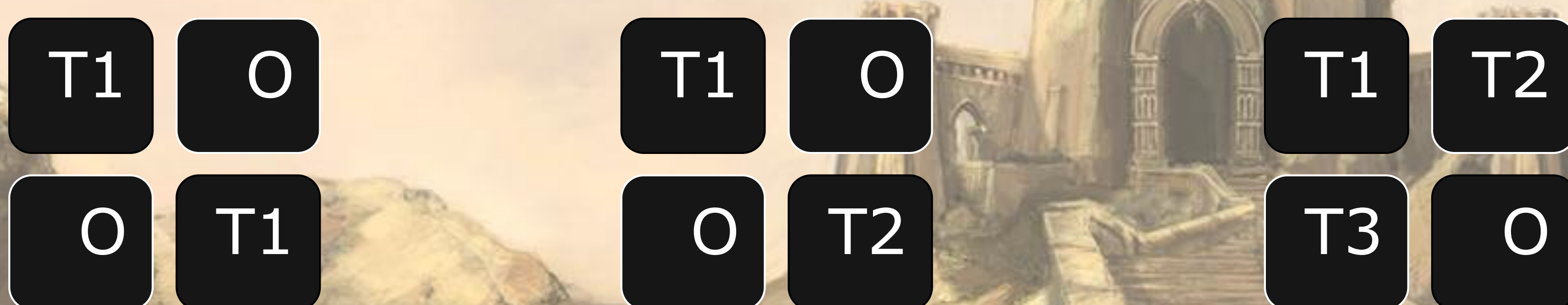
- Require lighting
  - Correct
  - Consistent with the whole scene
  - Shadowed
- Therefore we want it in deffered mode
  - Preferably with single lighting and shading cost
- Use dither patterns with sample reconstruction

Semi-Transparent material

- 2 pass rendering
  - 1 pass – semi-transparent materials are written into g-buffer using dithering patten
  - 2 pass – materials are fully rendered after light accumulation, using sample reconstruction to get correct lighting values. Sorting and alpha blending is required.
- Someone actually got the same idea :]
  - Inferred Rendering

Semi-Transparent material

- 1 pass
  - pattern covers the basic rendering quad (i.e. 2x2)
  - Pattern choice depends on number of transparent material layers beeing overlayed
    - One 2x2 quad can cover
      - 2 materials with 75:25 ; 50:50 quality ration
      - 3 materials with 50:25:25
      - 4 materials with 25:25:25:25 quality ratio
    - Each additional layer leads to quality loss of lighting



Semi-Transparent material

- 2 pass
  - Overlaping semi-transparent materials are sorted back to front (with solid beeing the first to be rendered)
  - For each overlaping material
    - Lightbuffer is sampled with correct pattern to acquire original lighting values
    - Material is rendered with full resolution textures and reconstructed lighting
    - Transparency is handled by alphablending with the backbuffer

Semi-Transparent material

- Lighting reconstruction
  - Taking one sample only leads to heavy aliasing
  - Must take multiple samples for reconstruction
    - Check if the pixel beeing shaded is the original one
      - If false, sample the neighbourhood for valid samples, weight them and average for sample reconstruction
      - If true, leave unaltered
    - Leads to less aliasing and more stability during movement
      - Using 2x2 quad for more than 2 materials=heavy texture cache trashing and aliasing

Semi-Transparent material

- Pros
  - Method suits light pre pass architecture
    - Same with hybrid deferred renderers
  - Flexible
  - Predictable, linear quality loss
- Cons
  - Taxing ROPs because of alpha blending
    - Especially frustrating when high precision blend operations are slow
  - Requires the second pass for solid and opaque geometry
    - Not a problem if doin light pre pass anyway
  - Sometimes problematic to flag the right objects to use dither
    - Mostly doing too much, thus losing quality and performance

Semi-Transparent material

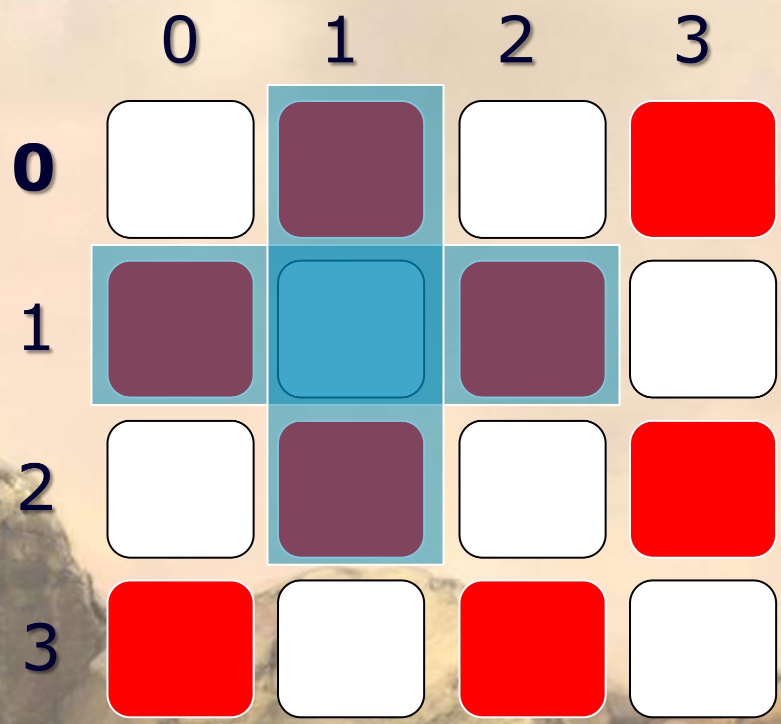
- We couldn't take the High Precision blending hit and additional geometry passes
  - Hybrid deferred renderer
- Settled with one layer transparency
  - Better performance, quality and stability
  - More flexible

Semi-Transparent material

- Deferred renderer with single transparency
  - Semi-transparent geometry is rendered to g-buffer with checkboard pattern
  - Albedo is set to 1
    - 1 – pass is feather weight – normals and specular only
  - After deferred shading
    - Acumulation buffer is containing alternating pixels of semi-transparent geometry lighting information and underlaying shaded geometry
    - 2 – pass is reconstructing both
      - Lighting data
      - Shaded background
    - Material is rendered with full quality
    - Alpha blending is done manually

Semi-Transparent material

- Deferred renderer with single transparency
  - Reconstruction
    - Sample a cross a pattern



For even pixel

Corners – light buffe

Middle – background

For odd pixels

Corners – backgroun

Middle – light buffer

Semi-Transparent material

- Deferred rendering with single transparency
  - Really fast
    - Only the semi-transparent geometry is using pixel 'kill'
    - Sample reconstruction is simple and coherent
    - No branching needed
  - High quality
    - Background and lighting data is ¼ resolution, bilaterally upscaled
    - Stable during movement

Translucent material

- Translucent materials
  - Only allows light to pass through diffusely
  - Transparent materials are clear, while translucent ones cannot be seen through clearly.
  - Because of light diffusion inside material volume
    - Material is lit additionally by Sub Surface Scaterring
    - Visible background is diffused (blurred) – refraction
  - SSS amount is dependant on material parameters and thickness
    - Thicks materials, requiring global SSS are unpractical for performance reasons
    - We can efficiently simulate local SSS (like in skin rendering)

Translucent material

- Translucent materials
  - For simplicity assume translucency with minimal local SSS
  - We need to simulate refracted light diffusion
    - Take the backbuffer
    - Perform hierarchical downscale with blurring
    - Sample original and blurred background
      - Lerp depending on translucency factor
    - Use for refracted light
      - Can use the same for fake real time glossy reflections

Skin rendering

- Skin rendering
  - Important for believable characters
  - Exhibits complex light interactions
    - Diffuse
    - Specular

Skin rendering

- Skin is multilayered
  - Oily layer
  - Epidermis
  - Dermis
- Know material
  - We see it everyday
- Therefore
  - Complex
  - Hard
    - Research
    - Tweaking

Skin rendering

- Oily layer
  - Responsible for specular reflectance
    - Fresnel reflectance
  - Dielectric
    - Reflects unaltered light
      - White light reflected as white light
  - Fine scale roughness
    - Requires advanced BRDF

Skin rendering

- Oily layer
  - Simulate using
    - Finescale detail normal map
    - Specular intensity and roughness maps
    - BRDF
      - Cook-Torrance
      - Shirmay-Kallos
        - Preferable for consoles due to easy factorization and performace optimizations

Skin rendering

- Oily layer
  - BRDF
    - Blinn-Phong with several lobes and fresnel reflectance
      - Optimal for consoles
      - We are using two lobes tweaked by artists

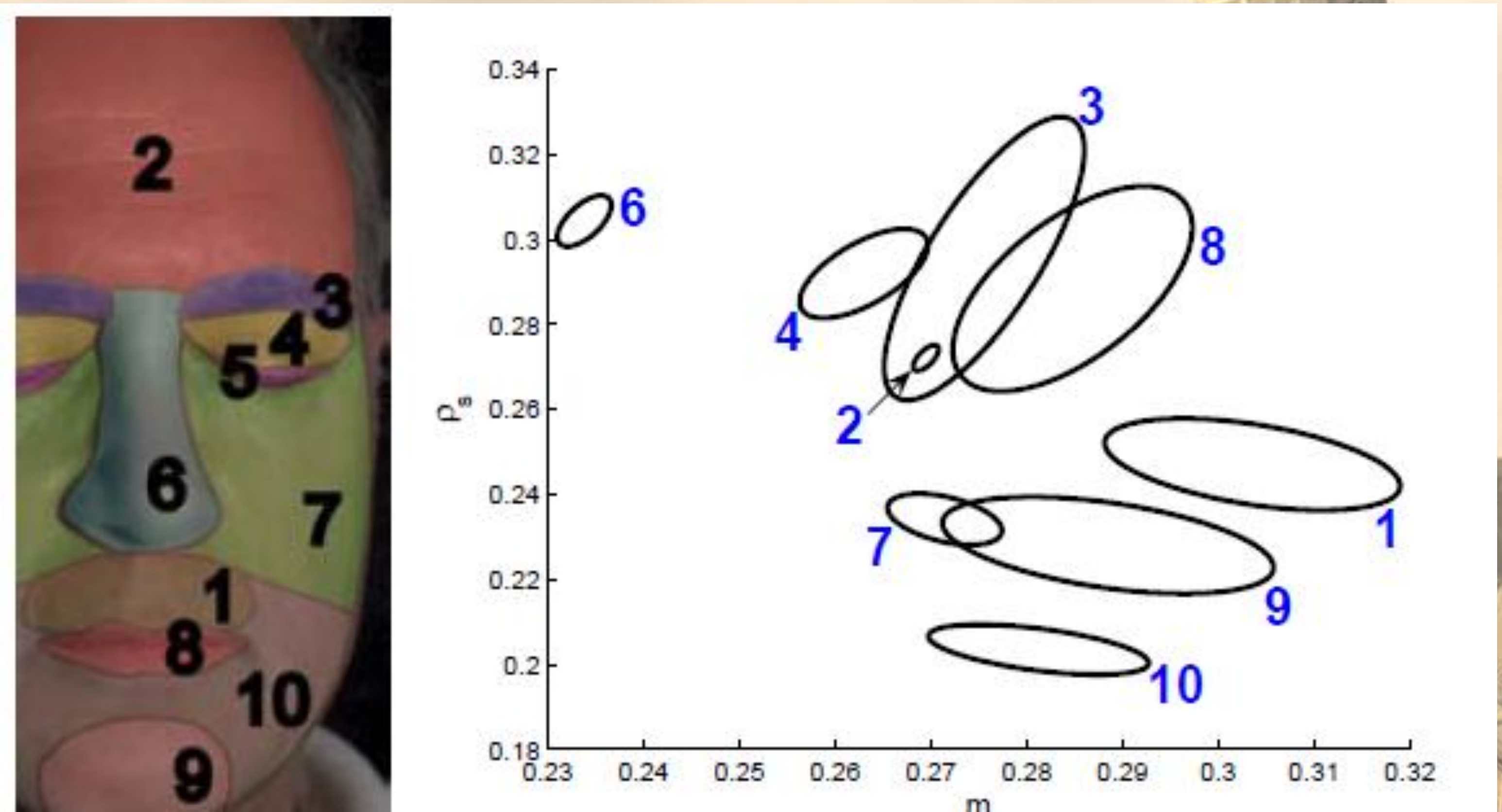
```
Specular = pow(dot(N,H),smallLobe)
Specular+= pow(dot(N,H),bigLobe)
```

Skin rendering

- Oily layer
  - Human face reflectance parameters varies depending on face region
    - **Acquisition of Human Faces Using A Measurement-Based Skin Reflectance Model. Weyrich 2006**
  - Several Cook-Torrance parameter maps exists based on empirical testing
  - Let your artists factor it into their specular maps

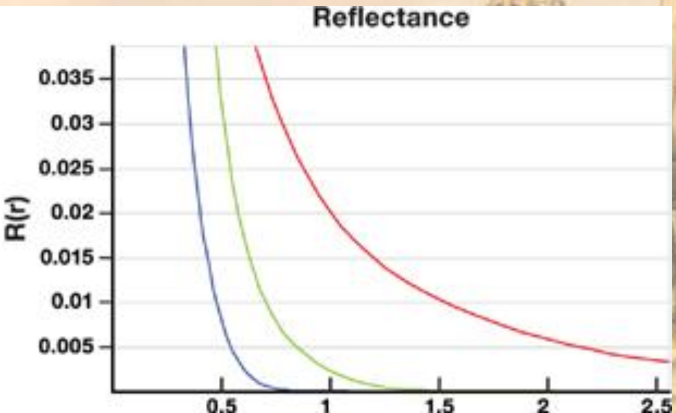
Skin rendering

- Ps – specular intensity
- M – specular roughness



Skin rendering

- Oily, Epidermis, Dermis
  - Responsible for diffuse light scattering
  - Light waves travel different distance because of scattering between layers
    - Aproximate with diffusion profile
      - Gpu Gems3 – Skin rendering
    - Measured empirically by light scattering study
      - Laser pointer in your: skin, wax, milk etc.



Skin rendering

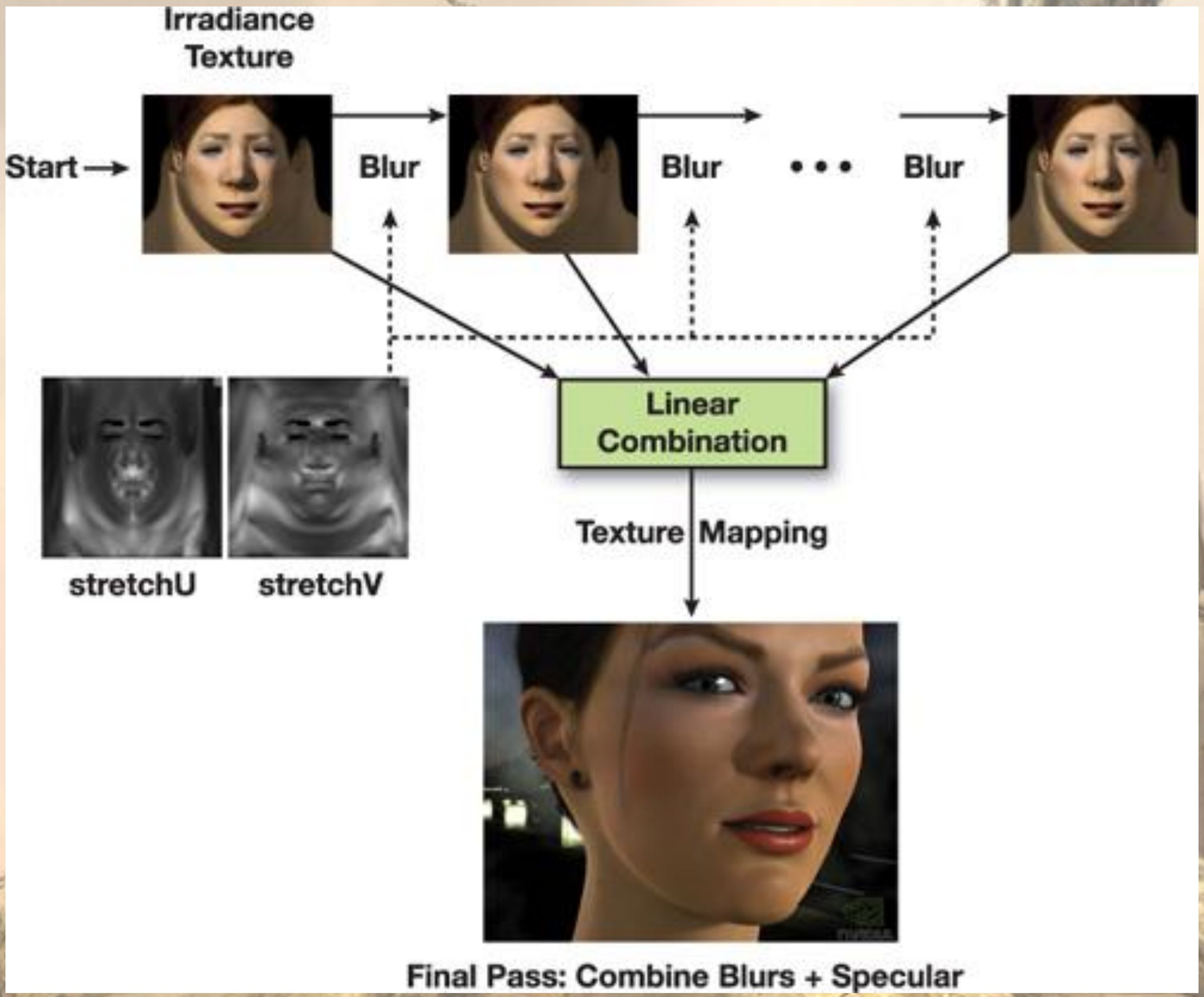
- Sub Surface Scattering
  - We can aproximate diffusion profiles by sum of weightened gaussians
  - Each material requires individual weight table
  - Example weights from Nvidia skin shader

	Variance (mm^2)	Blur Weights Red	Green	Blue
	0.0064	0.233	0.455	0.649
	0.0484	0.100	0.336	0.344
	0.187	0.118	0.198	0
	0.567	0.113	0.007	0.007
	1.99	0.358	0.004	0
	7.41	0.078	0	0

Skin rendering

- Sub Surface Scattering
  - Correct SSS lighting using texture space diffusion
    - Unwrap the object
    - Create object light buffer in texture space
    - Perform sum of gaussian convolutions over the unwraped boject light buffer
      - Take care for stretching
    - Wrap it back onto the model and use in shading

Skin rendering



Skin rendering

- SSS by texture space diffusion
  - Accurate
  - Costly
    - Unwraping
    - Additional memory
    - Relighting
  - In deferred architecture we have got everything we need in screen space light buffer

Skin rendering

- Screen Space Sub Subsurface Scattering
  - Use during material pass
  - Material shader samples the lightbuffer
    - Sample sum of gaussians
      - Take careful samples with diffusion profile weight table
  - Compute ddx and ddy for sampling radius control
  - Use masking to sample only from skin regions

Skin rendering

- Screen Space Sub Subsurface Scattering
  - Sampling
    - We take 9 taps with dynamic radius (good compromise for consoles)
      - Jittered sampling
      - Linear filtering (where possible and reasonable)
    - Weight table and distance tweaked manually, based on research papers
    - Sampling distance altered by current texel mip level
      - Prevents SSS stretching

Skin rendering

- Screen Space Sub Subsurface Scattering
  - Jittering
    - Use variable sampling pattern trick
      - Change sampling pattern depending on curent pixel VPOS
    - Cheap with great effect
  - Ignore samples from outside the object
    - Mask encoded in one bit (LSB) of light buffer

Skin rendering

- Screen Space Sub Subsurface Scatterin


Skin rendering

- Backside translucency
  - Operating in SS and in deferred mode
    - No light information regarding light transmission from behind
    - Important tranlucency effect
      - Red light through ears, hands (bone structure)

Skin rendering

- Backside translucency
  - Do in forward mode
    - Quick and dirty
      - Calculate backface lighting for n strongest lights
      - Attenuate by thickness map
        - Baked (xNormal) or done by artists
      - Works best for thin, non deformable, surfaces (leaves, ears)

Skin rendering

- Backside translucency
  - Accurate
    - For each light render the depth map (use the one from shadow mapping)
    - During shading, project the depth map and calculate the distance between the point beeing shaded and the point 'on the other side' along light vector
    - Calculate light value and attenuate it by calculated distance

Skin rendering

Hair rendering

- Hair
  - Use alpha tested quads with simple transparency
    - Based on pixel 'kill' – therefore no need for sorting
    - Jittering and blending takes care for plausible blending
  - For lively apperiance advanced anizotropic specular is required
    - Kajiya-Kai
    - Ward Anisotropic
  - Anizotropy direction easily controlable
    - Painted per vertex
    - Direction texture map
    - Or simply follow geometry tangent
      - Artists control the direction by Uvs rotation in texture space

Hair rendering

- Hair
  - Use polygon soup with simple transparency
    - Based on pixel 'kill' – therefore no need for sorting
    - Jittering and post smart blurring takes care for plausible blending

Hair rendering

- Hair
  - Advanced anizotropic specular is required for lively apperiance
    - Kajiya-Kai
    - Ward Anisotropic
  - Anizotropy direction easily controlable
    - Painted per vertex
    - Direction texture map
    - Or simply follow geometry tangent
      - Artists control the direction by Uvs rotation in texture space

Hair rendering

- Hair
  - 2 pass rendering
    - 1 – render the polygon soup
    - 2 – render after deferred shading
      - Backbuffer contains Blinn-Phong lit hair
      - Add ward anizotropic specular from 2 most influencial
      - Treat the camera as additional light
        - Photography trick
        - Hair look healthier and more alive

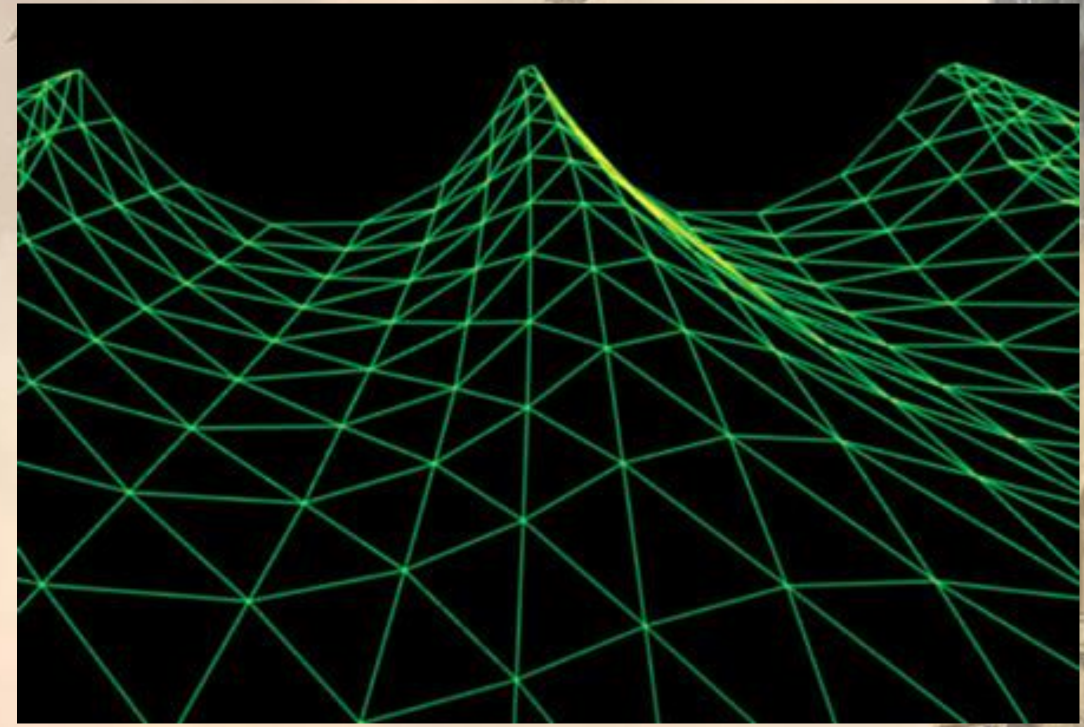
Water

- Water
  - Complex material
    - Geometry
      - Wave creation, propagation and interaction
    - Optics
      - Surface rendering
    - LODing scheme

Water

- Geometry
  - Render as tessaleted mesh
    - Adaptive Tesselation in screenspace
      - Nearer – more triangles
  - Use vertex shader for wave creation and propagation
    - Gerstner wave equation
      - Position and normal = fast computation
      - Can control choppiness
        - Verticies closer for wave crest
      - See Gpu Gems 1 : Effective Water Simulation from Physical Models
    - Generate several waves
      - Differ amplitude, frequency, direction, roughness

Water



$$\mathbf{P}(x,y,t)=\begin{pmatrix} x+\sum\left(Q_iA_i\times \mathbf{D}_i.x\times\cos\left(w_i\mathbf{D}_i\cdot(x,y)+\varphi_i t\right)\right),\\ y+\sum\left(Q_iA_i\times \mathbf{D}_i.y\times\cos\left(w_i\mathbf{D}_i\cdot(x,y)+\varphi_i t\right)\right),\\ \sum\left(A_i\sin\left(w_i\mathbf{D}_i\cdot(x,y)+\varphi_i t\right)\right)\end{pmatrix}.$$

Water

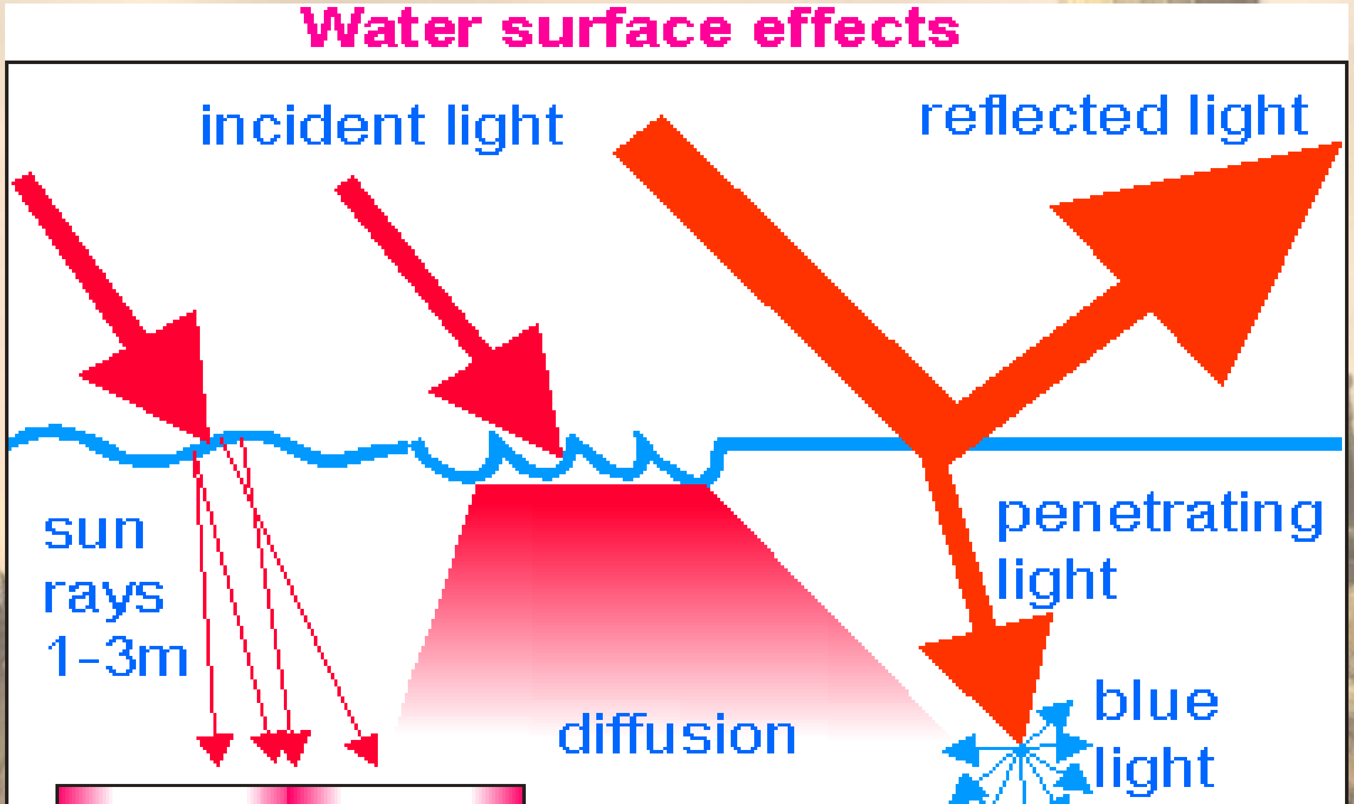
- Geometry
  - Wave amplitude is attenuated with vertex distance to sea bottow
    - Wave fadeout on beaches
  - Can generate foam particles on wave crest
    - We do it in pixel shader
      - Splash foam texture where needed
  - For physics
    - Evaluate the wave function per point when needed

Water

- Optics
  - Surface normal
  - Reflection
  - Refraction
  - Light scattering
  - Light extinction
  - Caustics
  - Solid surface decals
  - Specular

Water

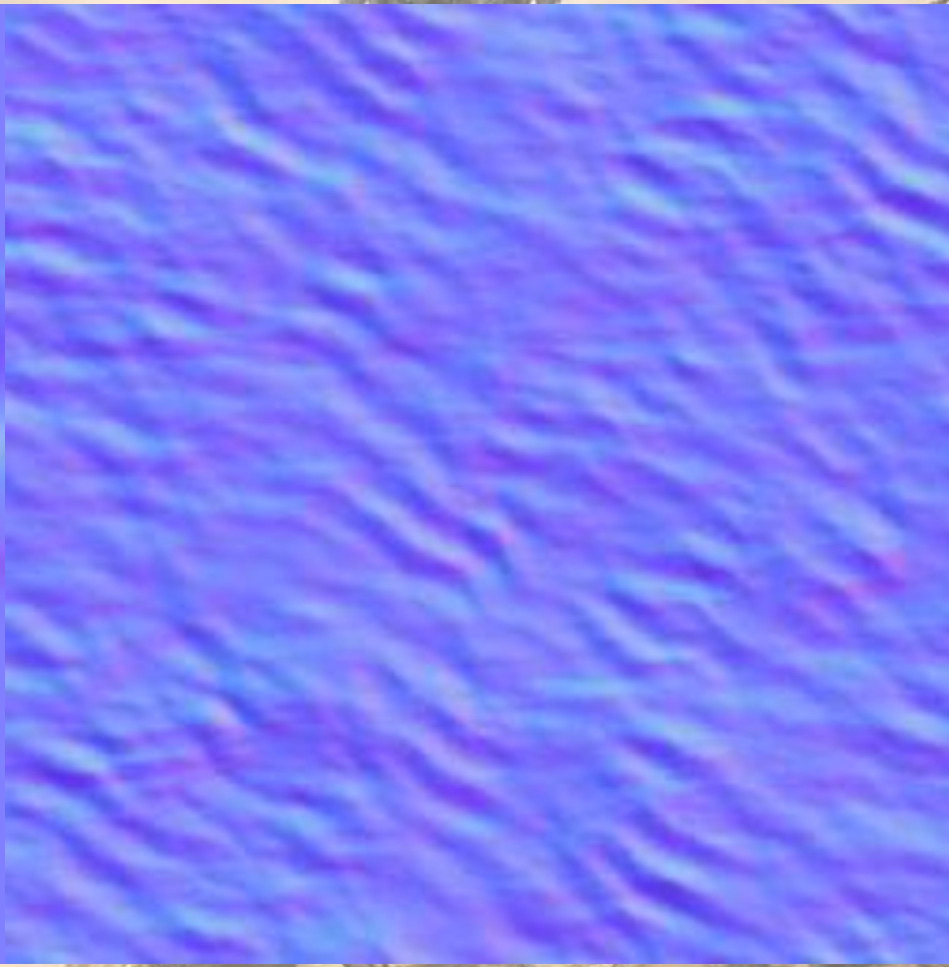
- Optics
  - Excellent references for underwater photography
    - http://www.seafriends.org.nz/phgraph/water.htm



Water

- Optics
  - Surface normal
    - Per vertex tangent basis from gerstner wave simulation
    - Per pixel normal blend
      - FFT
        - Computed real time
        - Blend of artist created, moving textures
      - Dynamic normal map using Navier Stokes
        - 256x256
        - Fluid splashes for each physical object
        - Centered at the camera position
        - Blends away from camera

Water



Water

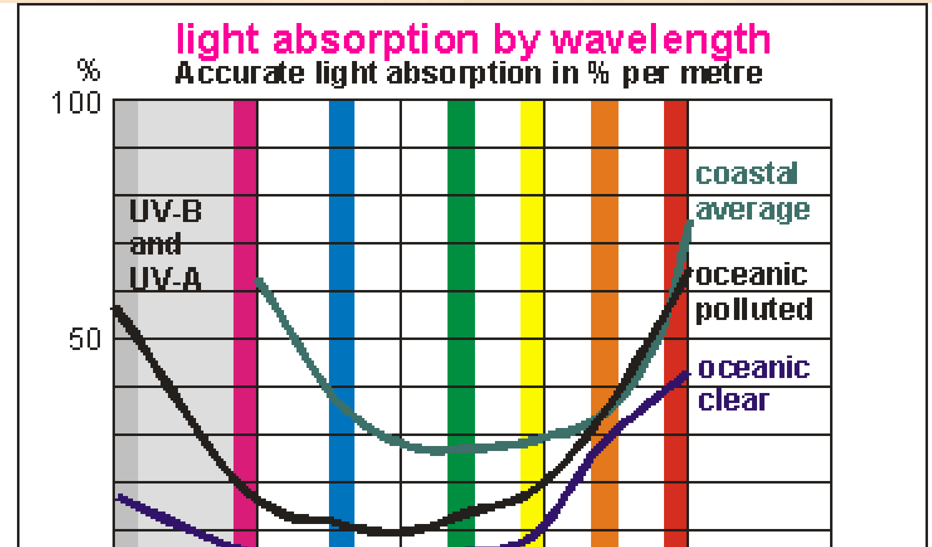
- Optics
  - Reflection
    - Render the reflection buffer
      - Use planar mirror matrix
      - Low res buffer (512x512)
      - LOD models, lights and shaders
      - Blur (stronger horizontal)
      - Must be HDR
        - RGBM8
    - Reflect the eye vector by surface normal
    - Project on reflection buffer and sample

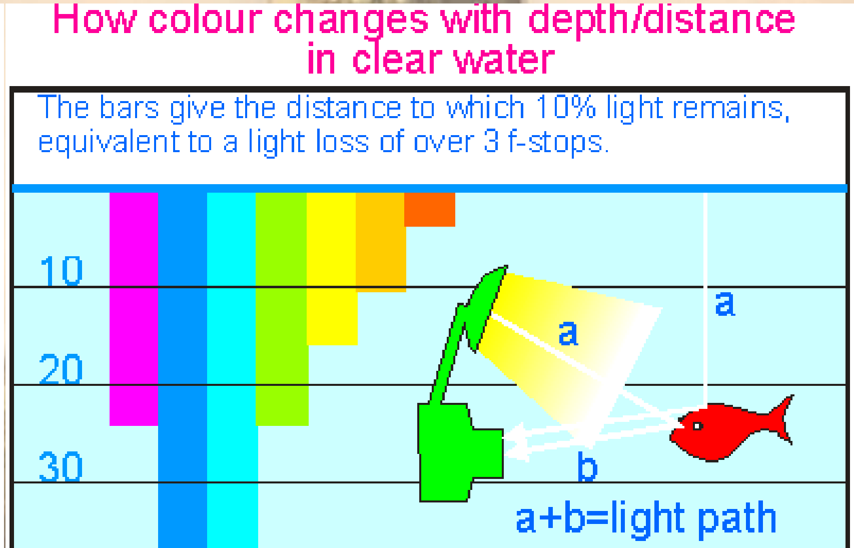
Water

- Optics
  - Refraction
    - Refract the eye vector by surface normal
    - Project on backbuffer
    - Sample the backbuffer
      - Can take 3 samples with offset – chromatic abberations
  - Sample = light
    - Scatter
    - Extinct

Water

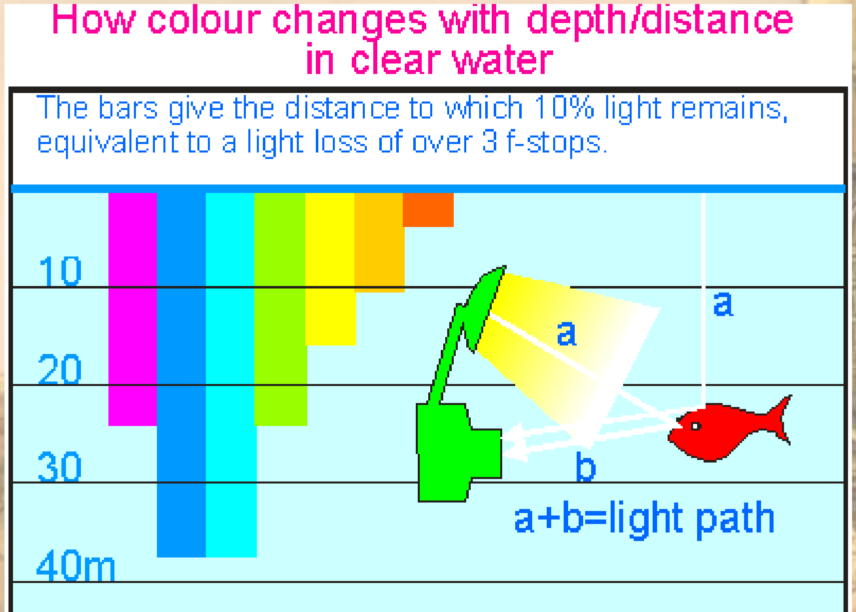
- Optics
  - Light extinction
    - Light coming from the sky is beeing attenuated by wavelength
      - Colour grading
    - Depends on D – ray length from surface to point beeing shaded
    - Must be attenuated per channel
      - Use research data





Water

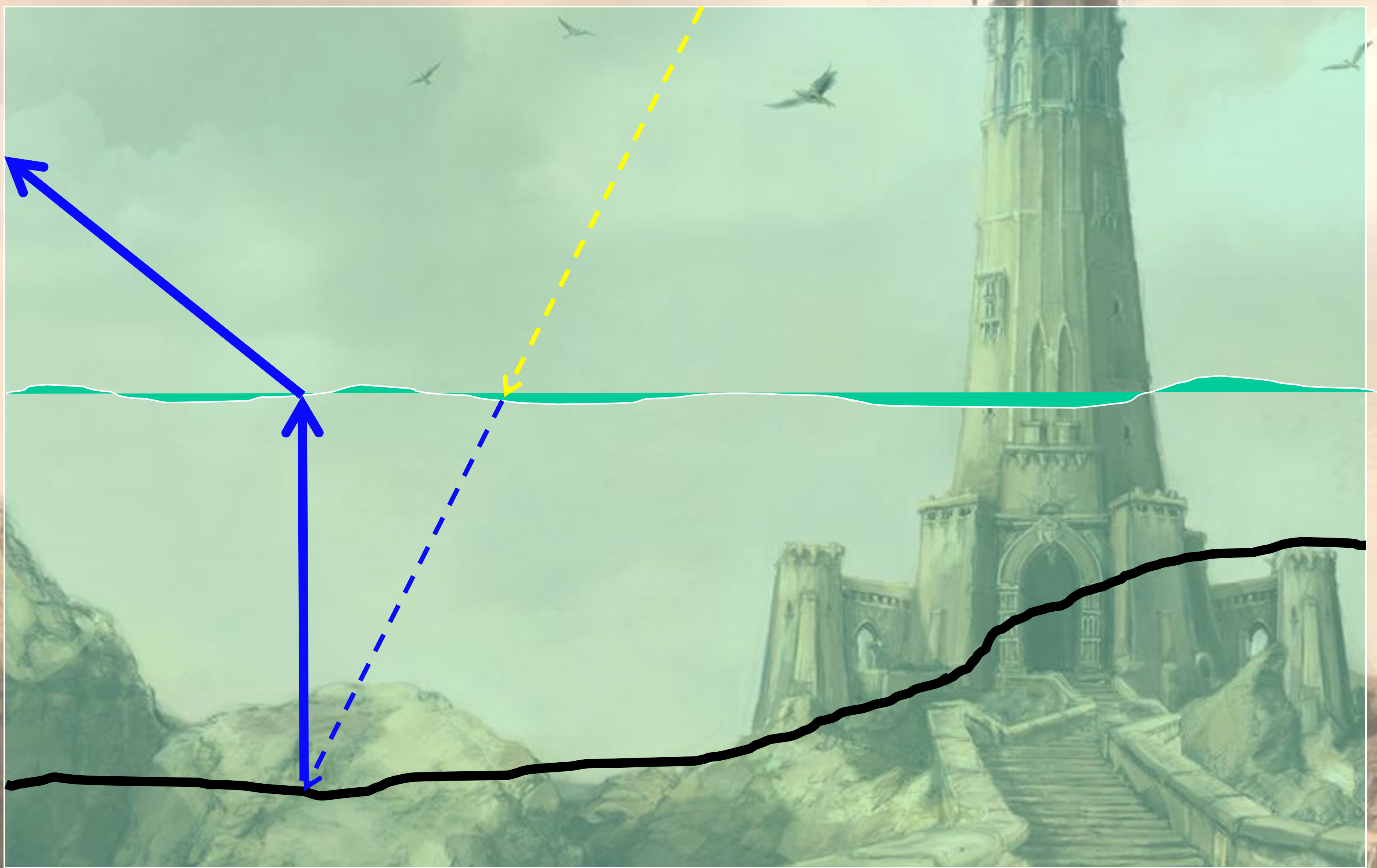
- Optics
  - Light scattering
    - Reflected light (incoming to camera) is scattered and diffused
      - Reyleigh – contrast loss
      - Tindall – bluring (can lerp between blured and original backbuffer)



Water

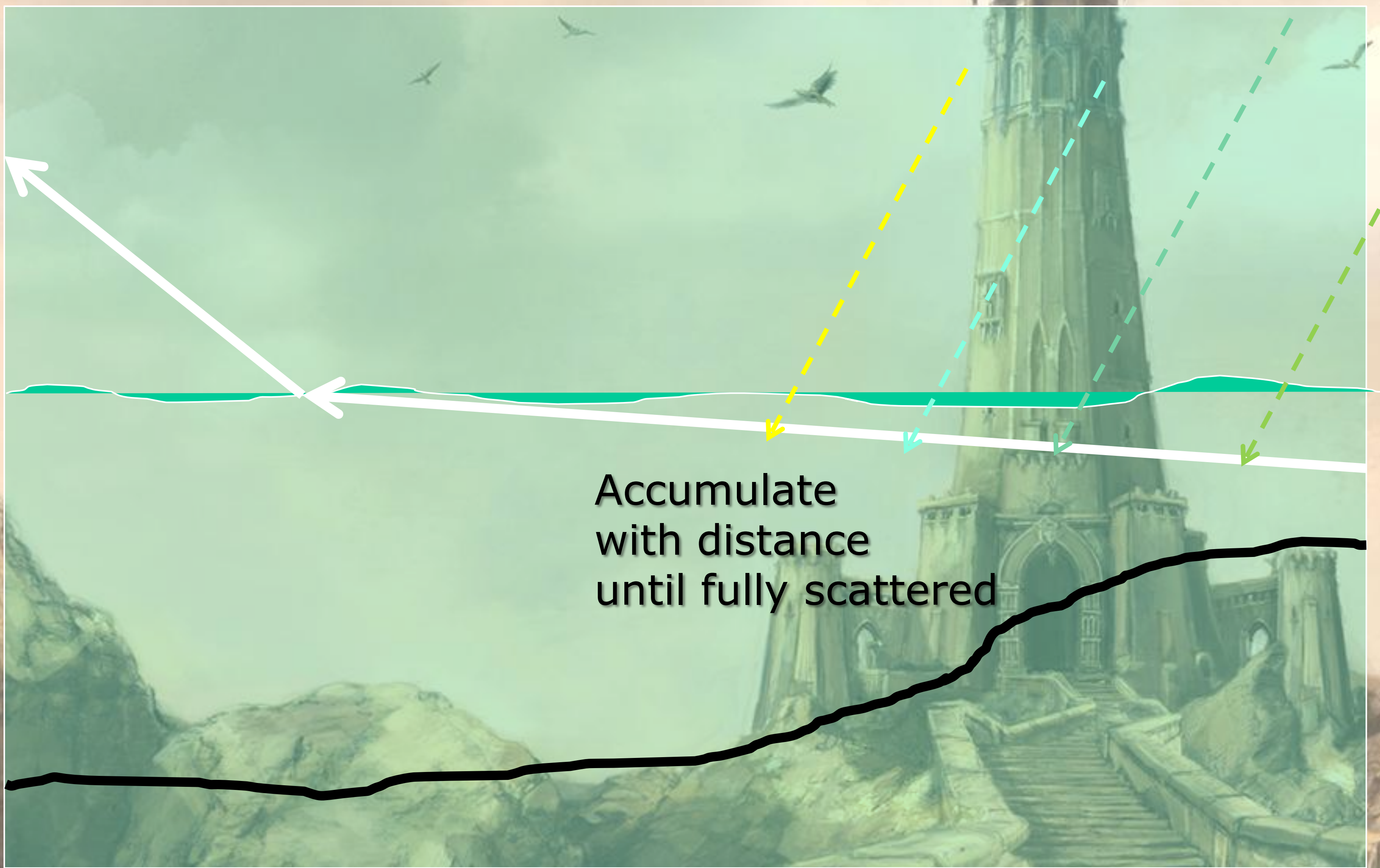
- Optics
  - Final light – simplified
    - Incoming light to camera
      - sL = extinct(L,distanceToSurface,waveLengthExtTable)
      - finalL = scatter(sL,distanceToCamera, attackAngle)
    - Proper evaluation requires
      - Precalcualted cube textures with calculated ray scattering and extinction
        - Must recalculate with water parameter change
    - Found a good aproximattion to given functions
      - Assume the camera is above water surface
    - Every distance easy to compute
      - Reconstruct Camera and World space position of point being shaded and point being sampled from backbuffer

Water



Water

Water



Water

- Approximate with a function
  - Dependant on
    - Attack angle
    - Distance from sampled point to surface
    - Distance from shaded point to sampled point
    - Water parameters (extinction table, tint)
  - See appendix
- Mix relfection and refraction using fresnel function

Water

- Causitcs
  - Project several caustic patterns on sea bottom
    - Project on backbuffer
      - Use reconstructed world position for Uvs and projection
    - Smartly animate
  - Attenuate using extinction

Water

- Surface decals
  - Textures blended with water
  - On top of water
  - Lit per-vertex

  - Foam
    - Foam texture
    - Blended where
      - Wave height > threshold
      - Distance from surface to bottom < threshold
      - Distance from surface to point sampled from backbuffer < threshold
        - Allows dynamic foam around objects – tricky to get right

Water

- Specular
  - Use true reflection vector
    - Better specular shape for sun
  - Average several lobes for area light specular
  - Take care for precise normals
    - Specular values are high
      - All precision artifacts will be visible

Water

- Soft edge
  - Get distance from point shaded to the point sampled from backbuffer
  - Use it to blend with backbuffer
  - Soft transition between water and shore (or objects)

Special Water Types

- Swamp water
  - Compute blurred backbuffer (BB)
    - 1/32 of original buffer
  - Refraction = lerp(original,blur,rayLengthFunction)
  - BB holds sun shadow mask in Alpha
    - Used for specular and light relfection attenuation
  - Using BB simulates volumetric lighting
  - Simplified scattering equation
    - No extinction (assumed too dense = solid color)
  - Different surface normals

Special Water Types

- Muddy water
  - Mix of ocean water and swampy water
  - Uses Navier Stokes velocity vectors to mix between original and blured backbuffer
    - Simulates water dusting due to movement
    - Can do the same using artist created textures
  - Use skyBox Cube for reflection
    - Speed up

Special Water Types

- River water
  - Mix of everything
  - Moving surface textures
    - Blending normals
  - Rivers layed down as paths (roads) of polygons
    - Direction
    - Speed
    - Foam amount
    - Curvature


Zombie


Zombie


Zombie

Special Water Types

- Presentation and code snippets available at
  - www.DROBOT.org
- Or mail me hello@drobot.org


WWW.DROBOT.ORG
?