MARCH 5-9
2007
SAN FRANCISCO

MOSCONE
CENTER

TAKE CONTROL

www.gdconf.com

CMP

GameDevelopers Conference 07
TAKE CONTROL
March 5-9, 2007 in San Francisco
CMP

Real-time Atmospheric Effects in Games Revisited

Carsten Wenzel

CRYTEK

The deal

- Follow up to a talk I gave at SIGGRAPH 2006
- Covers material presented at the time plus recent additions and improvements

Overview

- Introduction
- Scene depth based rendering
- Atmospheric effects breakdown
  - Sky light rendering
  - Fog approaches
  - Soft particles
  - Cloud rendering (updated/new)
  - Volumetric lightning approximation
  - River and Ocean rendering (updated/new)
- Scene depth based rendering and MSAA (new)
- Conclusions

Introduction

- Atmospheric effects are important cues of realism (especially outdoors)
- Why…
  - Create sense of depth
  - Increase level of immersion

Motivation

- Atmospheric effects are mathematically complex (so far usually coarsely approximated if any)
- Programmability and power of today's GPUs allow implementation of sophisticated models
- How to can these be mapped efficiently?

Related Work

- Deferred Shading (Hargreaves 2004)
- Atmospheric Scattering (Nishita et al 1993)
- Cloud Rendering (Wang 2003)
- Real-time Atmospheric Effects in Games (Wenzel 2006)

Scene Depth Based Rendering: Motivation

- Many atmospheric effects require accessing scene depth
- Similar to Deferred Shading [Hargreaves04]
- Mixes well with traditional style rendering
  - Deferred shading is **not** a must!
  - Think of it as writing a pixel shader with scene depth available
- Requires laying out scene depth first and making it available to following rendering passes

Scene Depth Based Rendering: Benefits

- Decouple rendering of opaque scene geometry and application of other effects
  - Atmospheric effects
  - Post-processing
  - More
- Apply complex models while keeping the shading cost moderate
  - Features are implemented in separate shaders
  - Helps avoiding hardware shader limits (can support older HW)

Scene Depth Based Rendering: Challenges

- Alpha-transparent objects
  - Only one color / depth value stored
  - However, per-pixel overdraw due to alpha transparent objects potentially unbound
  - Workaround for specific effects needed (will be mentioned later)

Scene Depth Based Rendering: API and Hardware Challenges

- Usually cannot directly bind Z-Buffer and reverse map
- Write linear eye-space depth to texture instead
- Float format vs. RGBA8
- Supporting Multi-Sample Anti-Aliasing is tricky (more on that later)

Recovering World Space Position from Depth

- Many deferred shading implementations transform a pixel’s homogenous clip space coordinate back into world space
  - 3 **`dp4`** or **`mul/mad`** instructions
- There’s often a simpler / cheaper way
  - For full screen effects have the distance from the camera’s position to its four corner points at the far clipping plane interpolated
  - Scale the pixel’s normalized linear eye space depth by the interpolated distance and add the camera position (one mad instruction)

Sky Light Rendering

- Mixed CPU / GPU implementation of [Nishita93]
- Goal: Best quality possible at reasonable runtime cost
  - Trading in flexibility of camera movement
- Assumptions and constraints:
  - Camera is always on the ground
  - Sky infinitely far away around camera
  - Win: Sky update is view-independent, update only over time

Sky Light Rendering: CPU

- Solve Mie / Rayleigh in-scattering integral
  - For 128x64 sample points on the sky hemisphere solve…

$$I_v(\lambda) = I_s(\lambda)K(\lambda)F(\theta, g)\int_{P_a}^{P_b}\left(e^{\frac{-h}{H_0}} e^{(-t(PP_c,\lambda)-t(PP_a,\lambda))}\right) \qquad (1)$$

  - Using the current time of day, sunlight direction, Mie / Rayleigh scattering coefficients
  - Store the result in a floating point texture
- Distribute computation over several frames
  - Each update takes several seconds to compute

Sky Light Rendering: GPU

- Map float texture onto sky dome
- Problem: low-res texture produces blocky results even when filtered
    - Solution: Move application of phase function to GPU (F(θ,g) in Eq.1)
    - High frequency details (sun spot) now computed per-pixel
- SM3.0/4.0 could solve Eq.1 via pixel shader and render to texture
    - Integral is a loop of ~200 asm instructions iterating 32 times
    - Final execution ~6400 instructions to compute in-scattering for each sample point on the sky hemisphere

Global Volumetric Fog

- Nishita's model still too expensive to model fog/aerial perspective
- Want to provide an atmosphere model
  - To apply its effects on arbitrary objects in the scene
- Developed a simpler method to compute height/distance based fog with exponential fall-off

Global Volumetric Fog

$$f\left((x, y, z)^T\right) = be^{-cz}$$

$$\vec{v}(t) = \vec{o} + t\vec{d}$$

$$\oint f(\vec{v}(t))dt = \int_0^1 f\left((o_x + td_x, o_y + td_y, o_z + td_z)^T\right)\|\vec{d}\|dt$$

$$= be^{-co_z}\sqrt{d_x^2 + d_y^2 + d_z^2}\left[\frac{1 - e^{-cd_z}}{cd_z}\right]$$

$$F(\vec{v}(t)) = e^{-\oint f(\vec{v}(t))dt} \qquad (2)$$

f – fog density distribution	b – global density
c – height fall-off	F – fog density along v

v – view ray from camera (o) to target pos (o+d), t=1

Global Volumetric Fog: Shader Implementation

Eq.2 translated into HLSL…

```
float ComputeVolumetricFog( in float3 cameraToWorldPos )
{
   // NOTE: cVolFogHeightDensityAtViewer = exp( -cHeightFalloff *
   cViewPos.z );

   float fogInt = length( cameraToWorldPos ) *
   cVolFogHeightDensityAtViewer;

   const float cSlopeThreshold = 0.01;
   if( abs( cameraToWorldPos.z ) > cSlopeThreshold )
   {
        float t = cHeightFalloff * cameraToWorldPos.z;
        fogInt *= ( 1.0 - exp( -t ) ) / t;
   }

   return exp( -cGlobalDensity * fogInt );
}
```

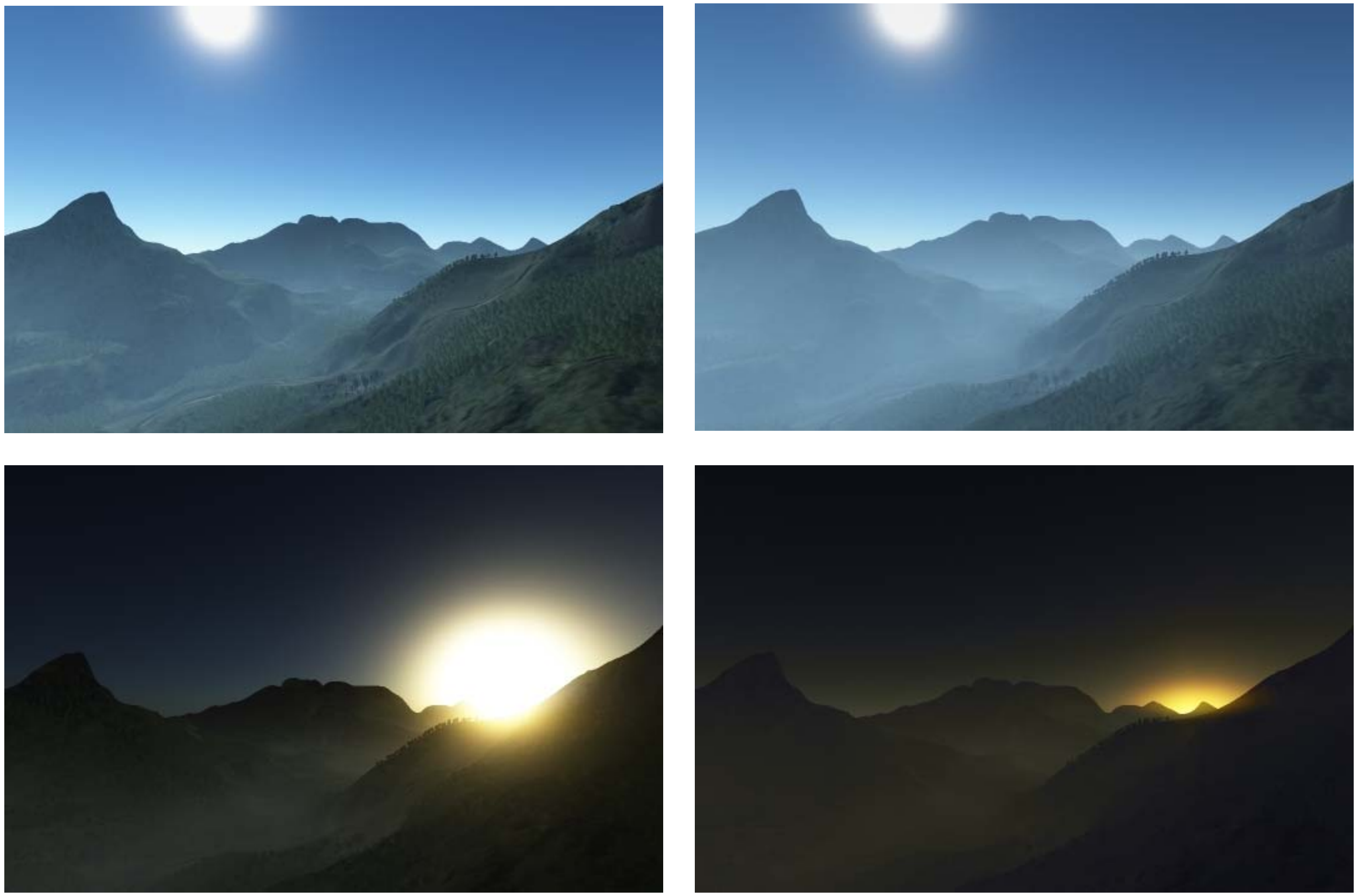
Combining Sky Light and Fog

- Sky is rendered along with scene geometry
- To apply fog…
  - Draw a full screen quad
  - Reconstruct each pixel's world space position
  - Pass position to volumetric fog formula to retrieve fog density along view ray
  - What about fog color?

Combining Sky Light and Fog

- Fog color
    - Average in-scattering samples along the horizon while building texture
    - Combine with per-pixel result of phase function to yield approximate fog color
- Use fog color and density to blend against back buffer

Combining Sky Light and Fog: Results



*

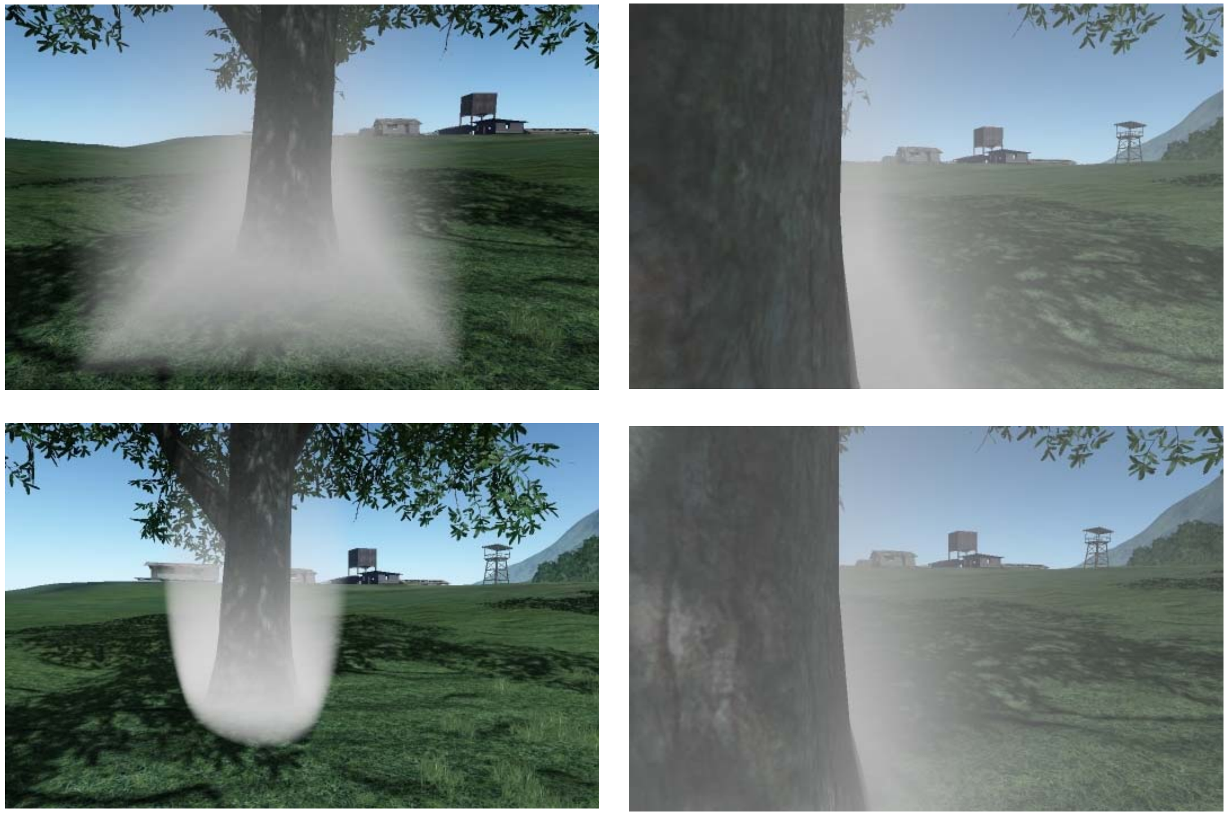
Fog Volumes

- Fog volumes via ray-tracing in the shader
- Currently two primitives supported: Box, Ellipsoid
- Generalized form of Global Volumetric Fog
  - Exhibits same properties (additionally, direction of height no longer restricted to world space up vector, gradient can be shifted along height dir)
- Ray-trace in object space: Unit box, unit sphere
- Transform results back to solve fog integral
- Render bounding hull geometry
  - Front faces if outside, otherwise back faces
- For each pixel…
  - Determine start and end point of view ray to plug into Eq.2

Fog Volumes

- Start point
  - Either camera pos (if viewer is inside) or ray’s entry point into fog volume (if viewer is outside)
- End point
  - Either ray’s exit point out of the fog volume or world space position of pixel depending which one of the two is closer to the camera
- Render fog volumes back to front
- Solve fog integral and blend with back buffer

Fog Volumes



Rendering of fog volumes: Box (top left/right), Ellipsoid (bottom left/right)

Fog and Alpha-Transparent Objects

- Shading of actual object and application of atmospheric effect can no longer be decoupled
  - Need to solve both and combine results in same pass
- Global Volumetric Fog
  - Approximate per vertex
  - Computation is purely math op based (no lookup textures required)
  - Maps well to older HW…
    - Shader Models 2.x
    - Shader Model 3.0 for performance reasons / due to lack of vertex texture fetch (IHV specific)

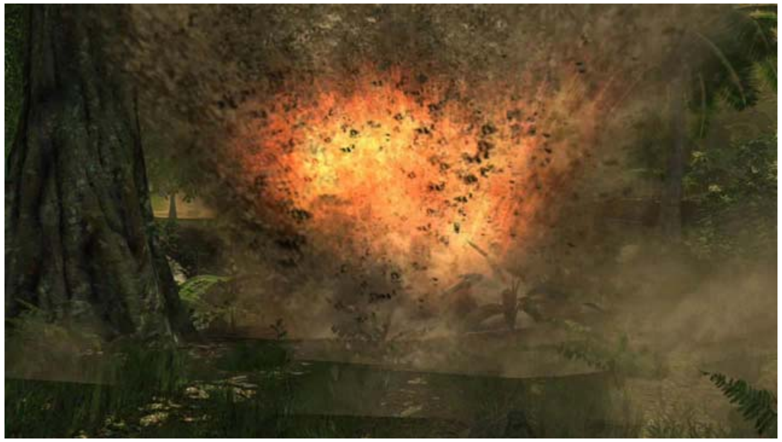
Fog and Alpha-Transparent Objects

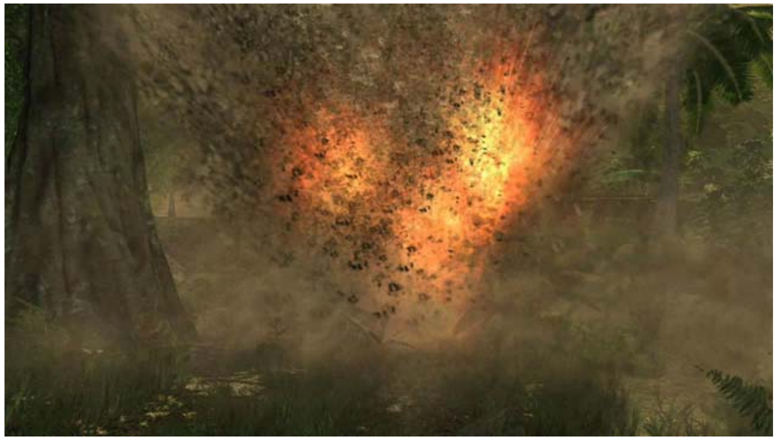
- Fog Volumes
  - Approximate per object, computed on CPU
  - Sounds awful but it’s possible when designers know limitation and how to work around it
    - Alpha-Transparent objects shouldn’t become too big, fog gradient should be rather soft
  - Compute weighted contribution by processing all affecting of fog volumes back to front w.r.t camera

Soft Particles

- Simple idea
  - Instead of rendering a particle as a regular billboard, treat it as a camera aligned volume
  - Use per-pixel depth to compute view ray's travel distance through volume and use the result to fade out the particle
  - Hides jaggies at intersections with other geometry
  - Some recent publications use a similar idea and treat particles as spherical volumes
    - We found a volume box to be sufficient (saves shader instructions; important as particles are fill-rate hungry)
  - GS can setup interpolators so point sprites are finally feasible

Soft Particles: Results

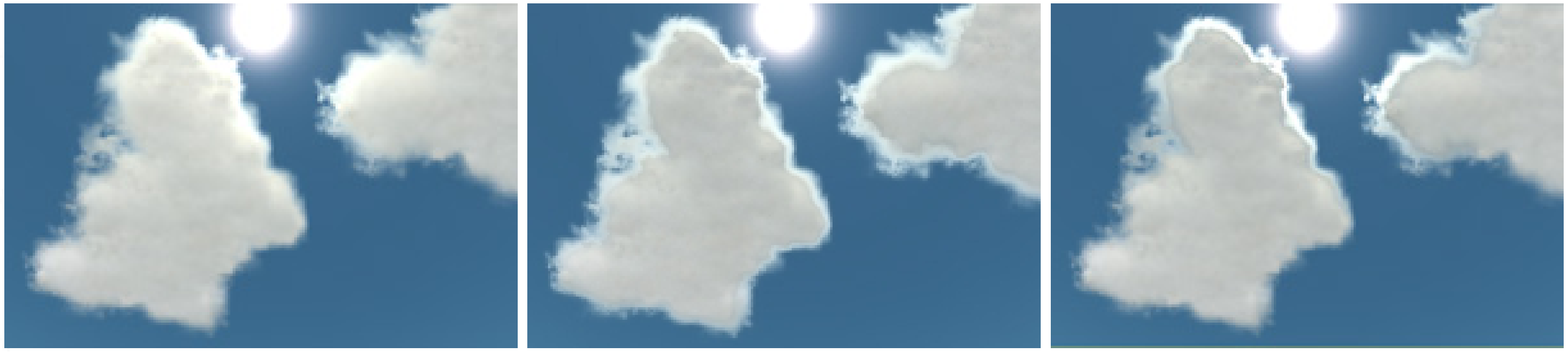




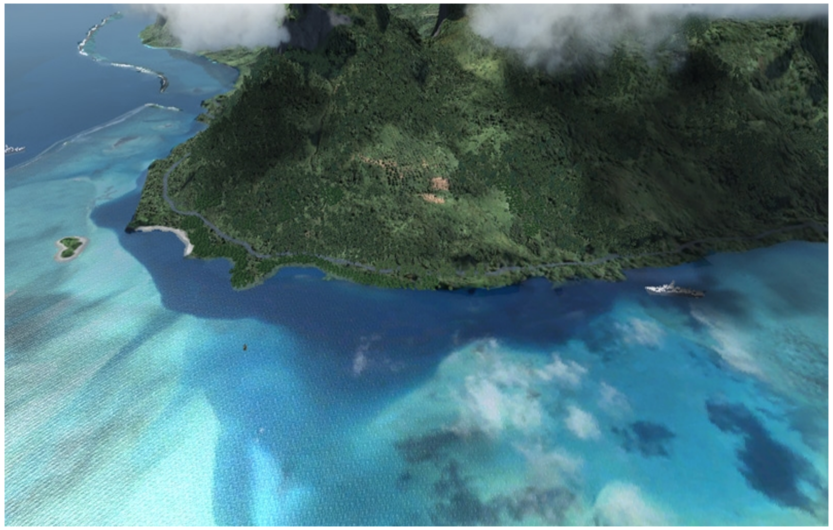
Comparisons shots of particle rendering with soft particles disabled (left) and enabled (right) *

Clouds Rendering Using Per-Pixel Depth

- Follow approach similar to [Wang03], Gradient-based lighting
- Use scene depth for soft clipping (e.g. rain clouds around mountains) – similar to Soft Particles
- Added rim lighting based on cloud density



Cloud Shadows



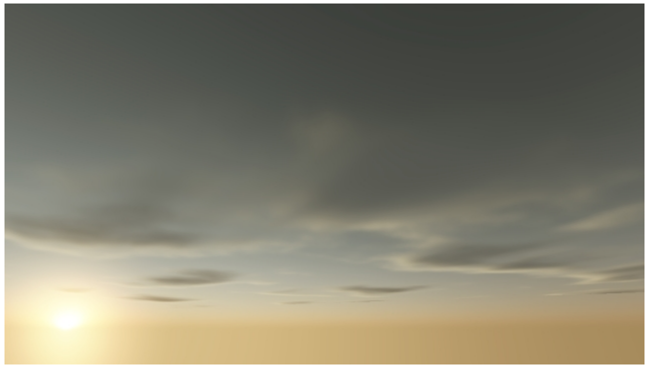
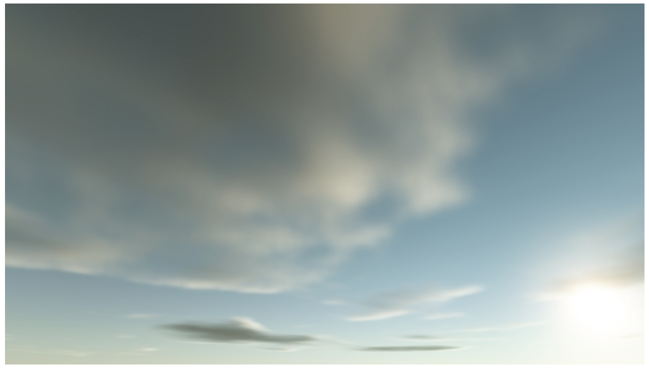
- Cloud shadows are cast in a single full screen pass
- Use depth to transform pixel position into shadow map space

Distance Clouds

- Dynamic sky and pre-baked sky box clouds don’t mix well
- Real 3D cloud imposters can be expensive and are often not needed
- Limited to 2D planes above the camera clouds can be rendered with volumetric properties
- Sample a 2D texture (cloud density) along the view dir
    - For each sample point sample along the direction to sun
- Adjust number of samples along both directions to fit into shader limits, save fill-rate, etc.

Distance Clouds

- Use the accumulated density to calc attenuation and factor in current sun / sky light

Distance Clouds at different times of day *

Volumetric Lightning Using Per-Pixel Depth

- Similar to Global Volumetric Fog
  - Light is emitted from a point falling off radially
- Need to carefully select attenuation function to be able to integrate it in a closed form
- Can apply this lighting model just like global volumetric fog
  - Render a full screen pass

Volumetric Lightning Model

$$f\left((x,y,z)^T\right)=\frac{i}{1+a\cdot\left\|\vec{l}-(x,y,z)^T\right\|^2}$$

$$\vec{v}(t)=\vec{o}+t\vec{d}$$

$$\oint f(\vec{v}(t))dt=\int_0^1 f\left((o_x+td_x,o_y+td_y,o_z+td_z)^T\right)\left\|\vec{d}\right\|dt$$

$$=2i\sqrt{{d_x}^2+{d_y}^2+{d_z}^2}\left[\frac{\arctan\left(\frac{v+2w}{\sqrt{4uw-v^2}}\right)-\arctan\left(\frac{v}{\sqrt{4uw-v^2}}\right)}{\sqrt{4uw-v^2}}\right]$$

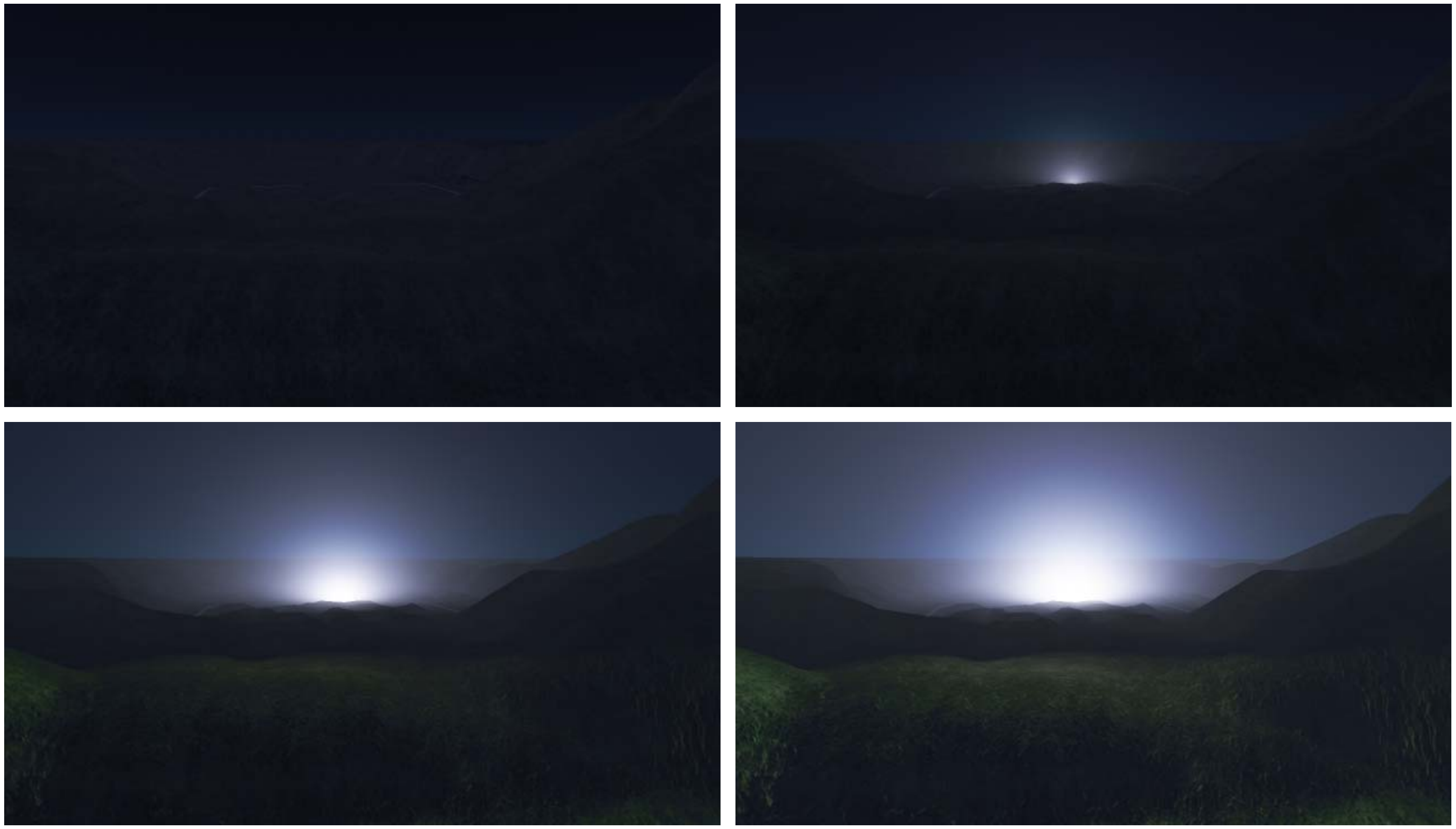
$$=F(\vec{v}(t)) \qquad (3)$$

f – light attenuation function	i – source light intensity
l – lightning source pos	a – global attenuation control value

v – view ray from camera (o) to target pos (o+d), t=1

F – amount of light gathered along v

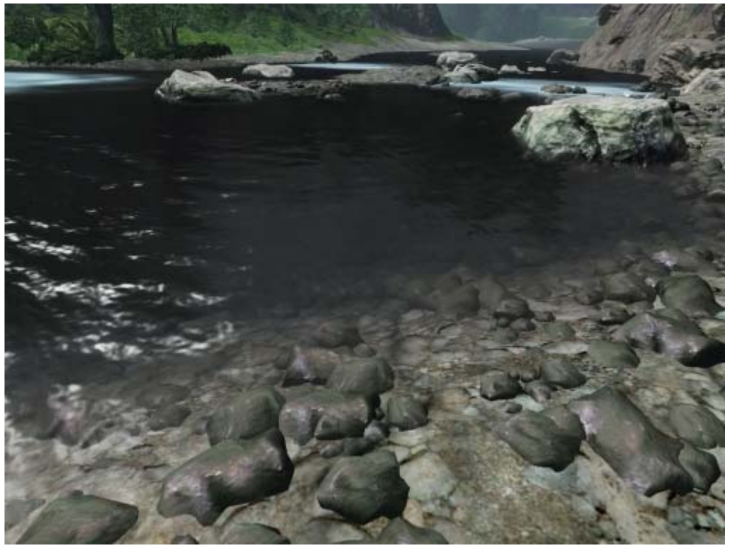
Volumetric Lightning Using Per-Pixel Depth: Results

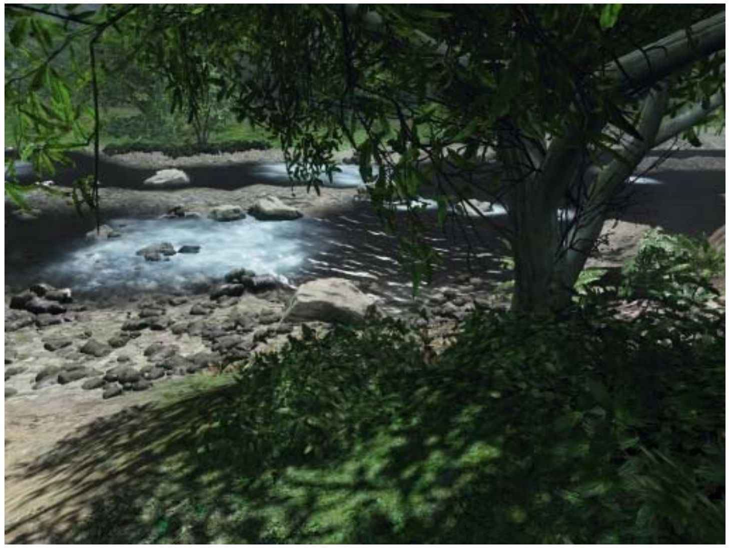


River shading

- Rivers (and water areas in general)
- Special fog volume type: Plane
- Under water fog rendered as described earlier (using a simpler uniform density fog model though)
- Shader for water surface enhanced to softly blend out at riverside (difference between pixel depth of water surface and previously stored scene depth)

River shading: Results





River shading –
Screens taken from a hidden section of the E3 2006 demo *

Ocean shading

- Very similar to river shading, however…
- Underwater part uses more complex model for light attenuation and in-scattering
- Assume horizontal water plane, uniform density distribution and light always falling in top down
- Can be described as follows…

Ocean shading

$$f\left((x,y,z)^T, t\right) = e^{-c(s-z)} e^{-ct} = e^{-c(s-z+t)}$$

$$\vec{v}(t) = \vec{o} + t\vec{d}$$

$$\vec{d} = \frac{\vec{p}-\vec{o}}{\left\|\vec{p}-\vec{o}\right\|}$$

$$l = \left\|\vec{p}-\vec{o}\right\|$$

$$attenuation = f(\vec{p}, l) = e^{-c(s-p_z+l)}$$

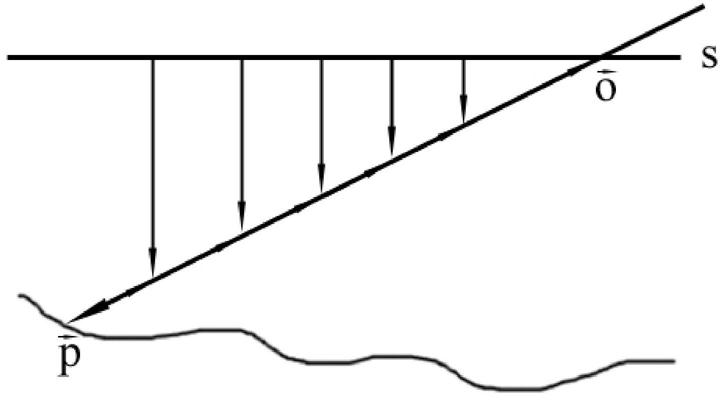
$$inscatter = \oint f\left(\vec{v}(t), t\right)dt = \int_0^l e^{-c(s-(o_z+td_z)+t)}\left\|\vec{d}\right\|dt$$

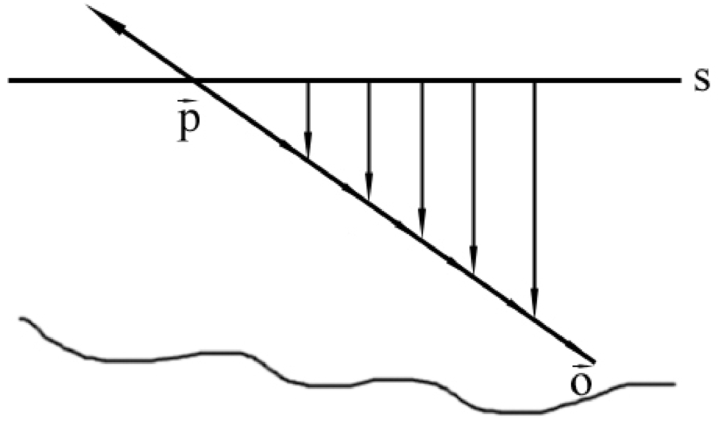
$$= e^{-c(s-o_z)}\int_0^l e^{-ct(1-d_z)}dt$$

$$= e^{-c(s-o_z)}\left[\frac{e^{(d_z-1)ct}}{(d_z-1)c}\right]_0^l$$

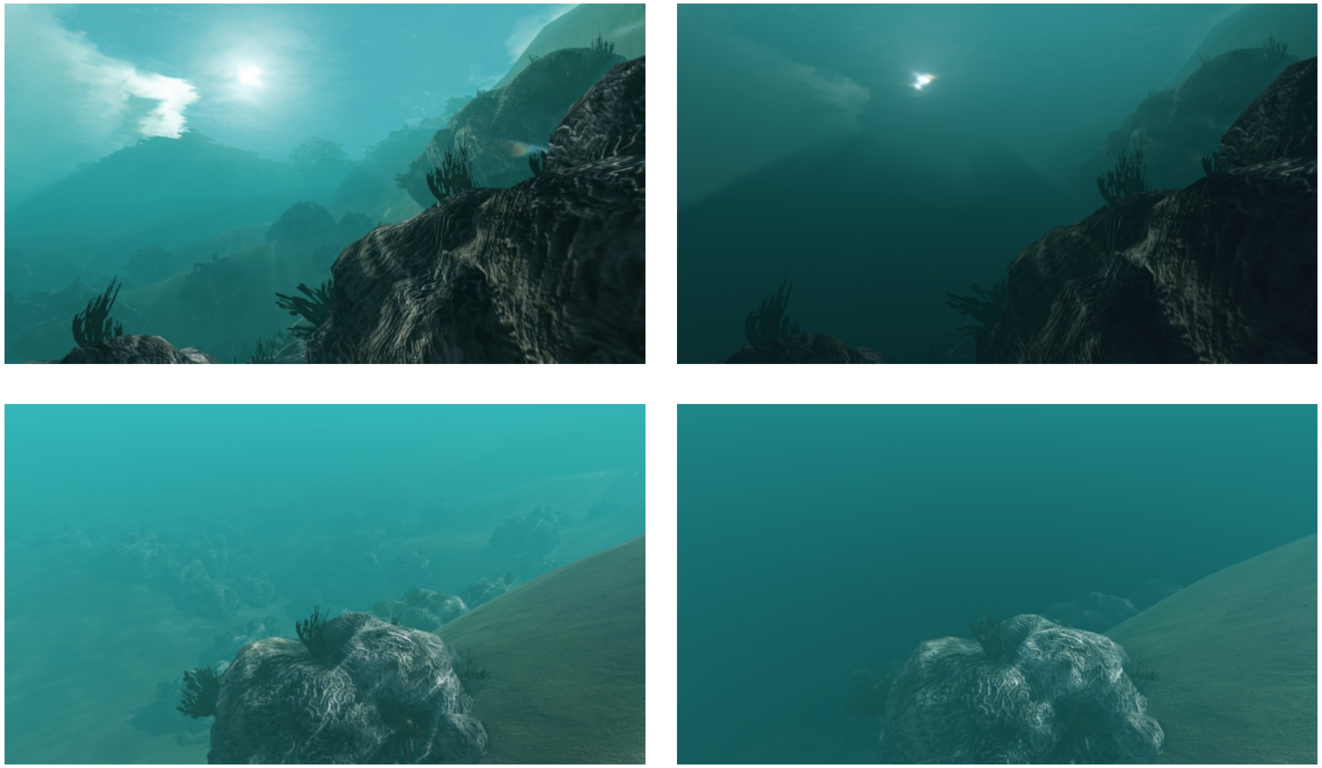
$$= e^{-c(s-o_z)}\left[\frac{e^{(d_z-1)cl}-1}{(d_z-1)c}\right]$$

$$finalCol = attenuation \cdot sceneCol + inscatter \cdot envlightCol \qquad (4)$$





Ocean shading: Results



Underwater view: from ground up (1st row), from underneath the surface down (2nd row). Same lighting settings apply. Higher density on the right column. *

Scene depth based rendering and MSAA

- Several problems
  - Cannot bind multi-sampled RT as texture
  - Shading per pixel and not per sample
- Need to resolve depth RT which produces wrong values at silhouettes ➔ potentially causes outlines in later shading steps
- Two problems we ran into
  - Fog
  - River / Ocean

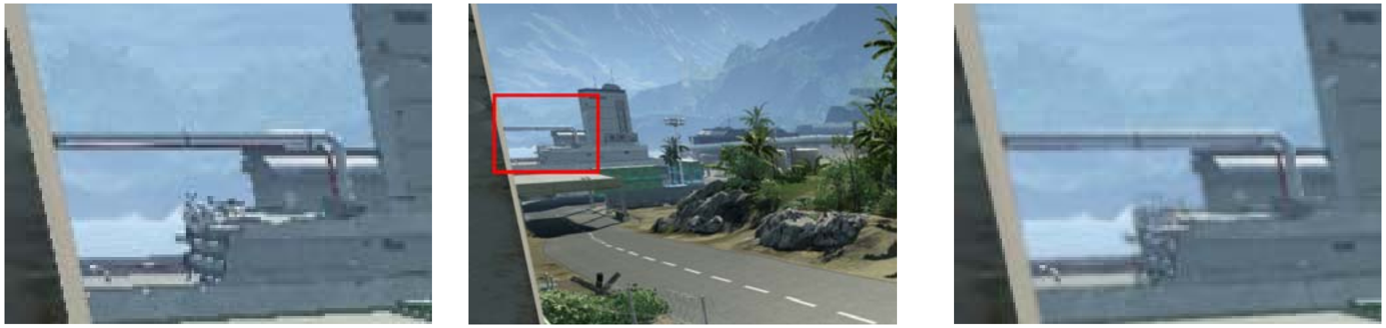
Scene depth based rendering and MSAA: Fog

- Fog color doesn't changed drastically for neighboring pixel while density does
- Have fog density computed while laying out depth (two channel RT)
- During volumetric fog full screen pass only compute fog color and read density from resolved RT
- Averaging density during resolve works reasonably well compared to depth

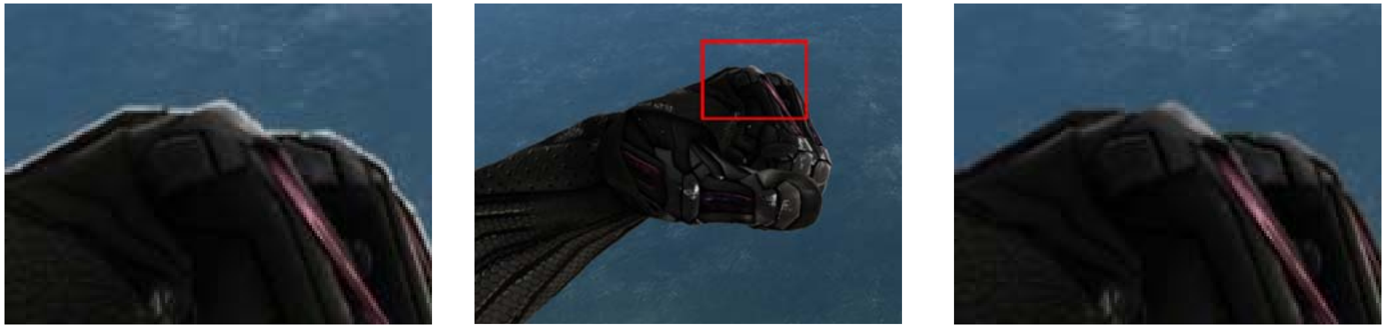
Scene depth based rendering and MSAA: River / Ocean

- Shader assumes dest depth > plane depth (otherwise pixel would have be rejected by z-test)
- With resolved depth RT this cannot be guaranteed (depends on pixel coverage of object silhouettes)
- Need to enforce original assumption by finding max depth of current pixel and all neighbors (direct neighbors suffice)

Scene depth based rendering and MSAA: Results



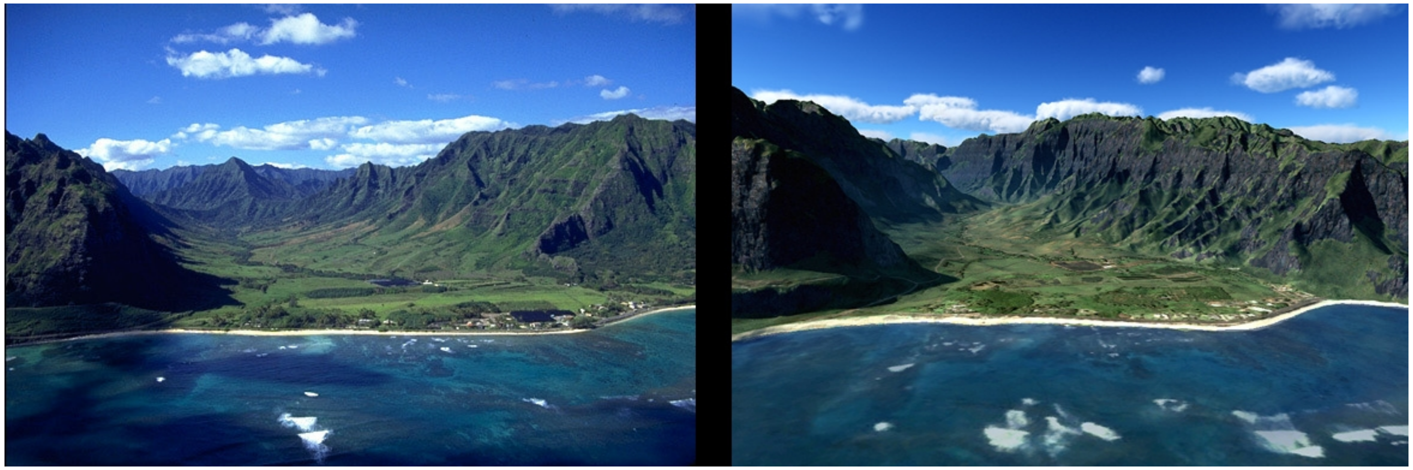
Fog full screen pass with MSAA disabled (left) / enabled (right)



River / Ocean shading artifact (left) and fix (right)

Conclusion

- Depth Based Rendering offers lot's of opportunities
- Demonstrated several ways of how it is used in *CryEngine2*
- Integration issues (alpha-transparent geometry, MSAA)



Kualoa Ranch on Hawaii –

Real world photo (left), internal replica rendered with *CryEngine2* (right)

References

- [Hargreaves04] Shawn Hargreaves, “Deferred Shading,” Game Developers Conference, D3D Tutorial Day, March, 2004.
- [Nishita93] Tomoyuki Nishita, et al., “Display of the Earth Taking into Account Atmospheric Scattering,” In Proceedings of SIGGRAPH 1993, pages 175-182.
- [Wang03] Niniane Wang, “Realistic and Fast Cloud Rendering in Computer Games,” In Proceedings of SIGGRAPH 2003.
- [Wenzel06] Carsten Wenzel, “Real-time Atmospheric Effects in Games,” SIGGRAPH 2006.

Acknowledgements

- Crytek R&D / Crysis dev team

Questions

???