SIGGRAPH2006

Real-time Atmospheric Effects in Games

Carsten Wenzel

CRYTEK

Overview

SIGGRAPH2006

- Introduction
- Scene depth based rendering
- Atmospheric effects breakdown
  - Sky light rendering
  - Fog approaches
  - Soft particles
  - Cloud rendering
  - Volumetric lightning approximation
  - Other interesting stuff
- Conclusions

Introduction

- Atmospheric effects are important cues of realism especially in outdoor scenes
- Create a sense of depth
- Help increase level of immersion

Motivation

SIGGRAPH2006

- Atmospheric effects have always been subject to coarse approximation due to their inherent mathematical complexity
- Increased power and flexibility of GPUs allows to implement more sophisticated models in real-time
- How to map them efficiently on HW?
- *CryEngine2* showcase

CryEngine2 Video

*
–

Related Work

- Deferred Shading (Hargreaves 2004)
- Atmospheric Scattering (Nishita et al 1993)
- Cloud Rendering (Wang 2003)

Scene Depth Based Rendering: Motivation

SIGGRAPH2006

- Many atmospheric effects require accessing scene depth
- Hybrid rendering approach akin to Deferred Shading [Hargreaves04]
- Can be used with variety of rendering approaches
  - Deferred Shading is *not* a requirement
  - *CryEngine2* uses traditional rendering style
  - Simply apply scene depth based rendering for specific effects
- Approach:
  - Lay out per-pixel scene depth first
  - Make it available to following rendering passes to be able to reconstruct world space position

Deferred shading general idea: No redundant shading cost by rendering geometry fist and shade later. While rendering geometry save out all necessary shading attributes to a fat frame buffer (position->depth, normal, diffuse/spec color, etc.). At a later stage apply shading using attributes stored in “fat” frame buffer.

Scene Depth Based Rendering: Benefits

SIGGRAPH2006

- Decouple rendering of opaque scene geometry and application of other effects
  - Atmospheric effects
  - Post-processing
  - More
- Can apply complex models while keeping the shading cost moderate
  - Features are implemented in separate shaders
  - Helps avoiding hardware shader limits
  - Allows broader use of these effects by mapping them to older hardware

Scene Depth Based Rendering: Concerns

SIGGRAPH2006

- Trouble child: Alpha-transparent objects
  - The problem: only one color / depth value stored; however, pixel overdraw caused by alpha transparent objects potentially unbound
  - Workaround for specific effects will be mentioned later

Scene Depth Based Rendering: API and Hardware Concerns

- 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

Recovering World Space Position from Depth

SIGGRAPH2006

- 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

SIGGRAPH2006

- 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

P_a – Start point of integration (in our context: viewer)
P_b – End point of integration (in our context: atmosphere top along view direction)
P_c – Sun
P – Point along path P_aP_b

I_v (λ) – in scattered light along path P_aP_b
I_s (λ) – sun intensity
I (λ) – scattering coefficient
F(θ, g) – Phase function

h – Height of P over ground
H_0 – Scale height

t(s, λ) – Optical depth function

Sky Light Rendering: GPU

- Map the float texture onto the sky dome
- Problem: low-res texture produces blocky results even when filtered
  - Solution: Move application of phase function to GPU ($F(\theta,g)$ in Eq.1)
  - High frequency details (sun spot) now computed per-pixel
- Next-Gen GPUs should be able to 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

SIGGRAPH2006

- 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

SIGGRAPH2006

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

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

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

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

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

(2)

- f – fog density distribution
- b – global density
- c – height fall-off
- v – view ray from camera (o) to target pos (o+d), t=1
- F – fog density along v

Global Volumetric Fog: Shader Implementation

SIGGRAPH2006

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

SIGGRAPH2006

- 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

SIGGRAPH2006

Fog Volumes

- Fog volumes via ray-tracing in the shader
- Currently two primitives supported: Box, Ellipsoid
- Generalized form of Global Volumetric Fog, exhibit 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), then 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

SIGGRAPH2006

- 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

SIGGRAPH2006

- 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 that for our purposes a volume box is sufficient {saving shader instructions; important as particles are fill-rate hungry}

Soft Particles: Results

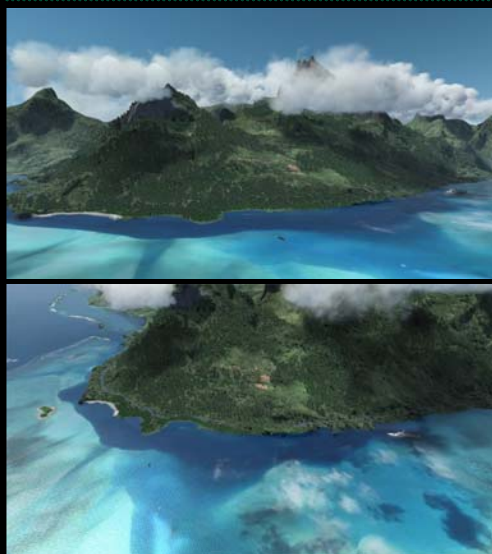
Comparisons shots of particle rendering with soft particles disabled (left) and enabled (right) *

Clouds Rendering Using Per-Pixel Depth

SIGGRAPH2006

- 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

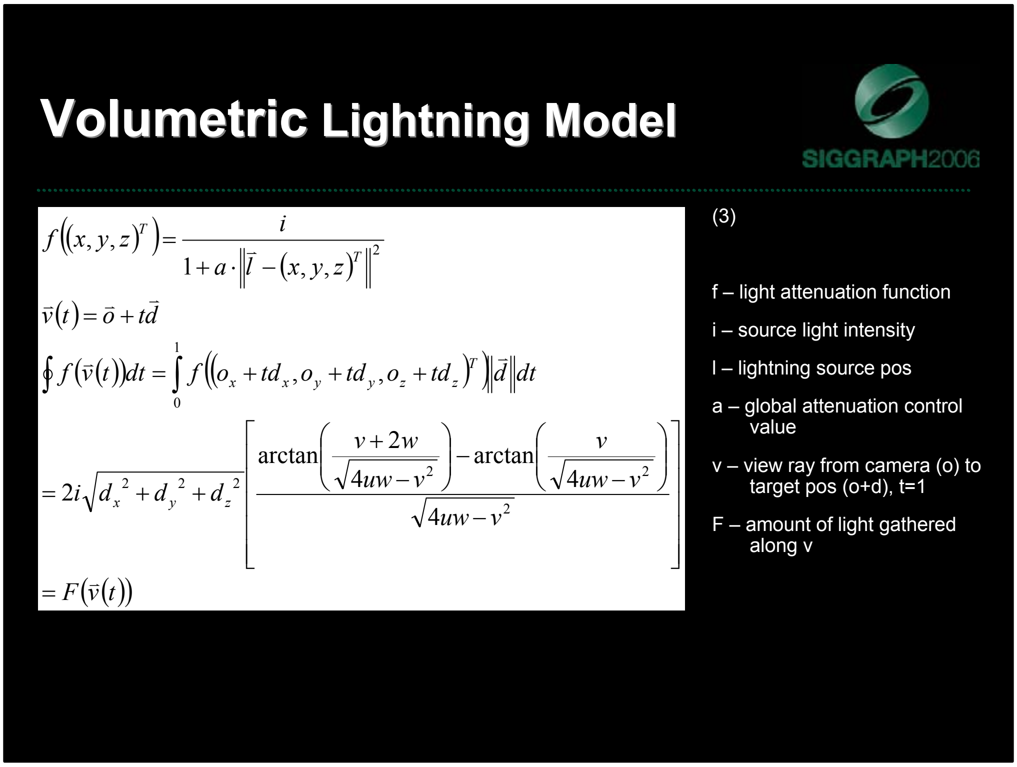
Cloud Shadows



- Cloud shadows are cast in a single full screen pass
- Use depth to recover world space pos, transform into shadow map space

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



Notice that HLSL’s arctan can compute up to four results in parallel. No need to call it twice!

Volumetric Lightning Using Per-Pixel Depth: Results

Other Effects using Per-Pixel Depth: Rivers

SIGGRAPH2006

- Rivers (and water areas in general)
- Special fog volume type: Plane
- Under water fog rendered as described earlier (using a simpler constant 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)

Other Effects using Per-Pixel Depth: River results

River shading –

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

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.

Questions

???

Acknowledgements

SIGGRAPH2006

Many thanks to…

Natalya Tatarchuk, ATI

Crytek R&D / *Crysis* dev team

P.S.

Interested in *CryEngine2* HDR footage?

Check out BrightSide's expo booth. It shows a fly through of *Crysis* level (Crytek's upcoming title) captured in HDR on their latest HDR HDTV displays.