

Crysis Next Gen Effects

Tiago Sousa Senior R&D Graphics Programmer



"Next Gen" Effects ?









CryEngine 2: Shading Overview

- Support shader model 2.0, up to 4.0
- Completely dynamic lighting and shadows
- Up to 4 point light sources per-pass
- Wide range of known and DIY shading models
- Some other fancy features
- Deferred mix with multi-pass rendering approach
- Average of 2K drawcalls per frame (~2M tris)



Water and Underwater Rendering

Intro water rendering video

Water and Underwater Rendering

- Rendering believable looking water
 - Underwater light-scattering [1]
 - Water surface animation & tessellation
 - **Reflections/Refraction**
 - Shore/Foam
 - **Caustics and God-rays**
 - Camera and objects interaction with water
 - Particles
- How to make all this efficiently in a very complex and open ended world in a game like Crysis ?

No more flat water !

- 3D waves
- Used statistical Tessendorf animation model [2]
- Computed on CPU for a 64x64 grid
- Upload results into a FP32 texture
- Vertex displacement on GPU
- Lower HW specs used sin waves sum
- Additionally 4 moving normals maps layers



Surface Tessellation

Screen Space Grid Projection Extreme detail nearby Problems Screen edges Aliasing at distance Dropped this approach in the end



Surface Tessellation

Camera aligned grid Keep detail nearby and in front of camera Problems Camera roll Balancing nearby VS far away detail Kept this approach in the end



Surface from a top perspective





Physics Interaction

- CPU animation is shared with Physics/Game
- For lowspec machines, did same "math" as in vertex shader on CPU
- Physics samples best water plane fitting object



Reflection

Per frame, we had an avg of 2K drawcalls (~ 2M tris)

- This really needed to be cheap and look good
- Problem: Reflections added about 1500 drawcalls
- Draw calls minimization
- Final average of 300 drawcalls for reflections
- Total internal reflection also simulated
- Half screen resolution RT



Update Dependencies

- Time
- Camera orientation/position difference from previous camera orientation/position
- Surface visibility ratio using HW occlusion queries
- Multi-GPU systems need extra care to avoid out of sync reflection texture



Anisotropic Reflection

- Blur final reflection texture vertically
- Also helps minimizing reflection aliasing





Refraction

- No need to render scene again [3]
- Use current back-buffer as input texture
- Mask out everything above water surface
- Water depth > World depth = leaking
 Don't allow refraction texture offset for this case
- Chromatic dispersion approx. for interesting look
 Scaled offset for R, G and B differently



Refraction Masking







Chromatic Dispersion

Chromatic dispersion Exagerated for picture

Procedural Caustics

- Extra rendering pass, can handle opaque/transparent
- Based on real sun direction projection
- Procedural composed using 3 layers
- Chromatic dispersion approximation
- Darken slightly to simulate wet surface



Procedural Caustics



Shore and Foam

- Soft water intersection
- Shore blended based on surface depth distance to world depth
- Waves foam blended based on current height distance to maximum height
- Foam is composed of 2 moving layers with offsets perturbation by water bump
- Acceptable quality



Shore and Foam



Underwater God-Rays [4]

- Essentially the same procedural caustics shader
- Based on real sun direction projection
- Projected into multiple planes in front of camera
- Rendered into a 4x smaller than screen RT
- Finally add to frame-buffer





Underwater God-Rays

Light Scattering



+ God-Rays (exaggerated for picture)





- How to handle case where camera intersects an animated water surface ?
- Water droplets effect when coming out of water
- When inside water used a subtle distortion
- Water particles similar to soft-particles



Things for the Future

- Rolling waves didn't made into final game
 Special animated water decal geometry
- Water splashes
- Surface interaction with shoreline
- Dynamic surface interaction
- Maybe in nearby future project ? ^(C) ^(C) ^(C)



Frozen Surfaces

Intro frozen surfaces video

Frozen Surfaces

- Huge Headache
- Haven't found previous research on the subject
- Unique Alien Frozen World: How to make it ?
 Should it look realistic ?
 Or an "artistic" flash frozen world ?
 Make everything Frozen ?
 Dynamically ?
 Custom frozen assets ?
 Reuse assets ?
- Took us 4 iterations until final result

1st iteration: Custom frozen assets

2nd iteration: Procedural frozen shader

3rd iteration: Mixing both approaches

Final iteration: Procedural Frozen Shader

30 60

3 017 100

SE

Lessons learned

Final iteration

- Main focus was to make it visually interesting
- Used real world images as reference this time
- Minimize artist amount of work as much as possible
- Impossible to make every single kind of object look realistically frozen (*and good*) with a single unified approach ⁽³⁾
- 1 week before hitting feature lock/alpha (gulp)

Frozen water dropplets accumulated on sides

> Subtle Reflection Glittering

Accumulated snow on top

Constitute A sugar

Putting all together

- Accumulated snow on top
 - Blend in snow depending on WS/OS normal z
- Frozen water droplets accumulated on side
 - 2 layers using different uv and offset bump scales to give impression of volume
- 3D Perlin noise used for blending variation
 - 3 octaves and offsets warping to avoid repetitive patterns
- Glittering
 - Used a 2D texture with random noise vectors
 - Pow(saturate(dot(noise, viewVector), big value)
 - If result > threshold, multiply by big value for hdr to kick in














Blend variation and final result

Procedural Frozen

- Reused assets
- Dynamic freezing possibility and with nice transition
- Didn't gave artists more control than required
 Artists just press button to enable frozen
- Relatively cheap, rendering cost wise
- Visually interesting results
- Only looks believable under good lighting conditions



Post-Effects

Intro post-effects video



Post-Effects Overview

Post Effects Mantra:

- Final rendering "make-up"
- Minimal aliasing (for very-hi specs)
- Never sacrifice quality over speed Unless you're doing really crazy expensive stuff !
- Make it as subtle as possible

But not less - or else average gamer will not notice it



Camera Motion Blur (CMB)

LDR No bright streaks Washed out details

HDR

Screen-space velocities

- Render a sphere around camera
- Use previous/current camera transformation to compute delta vector
 - Lerp between previous/current transformation by a shutter speed ratio (n / frame delta), to get correct previous camera matrix

From previous/current positions compute velocity vector

- Can already accumulate N samples along velocity direction
 - But will get constant blurring everywhere



Velocity Mask

- Used depth to generate velocity mask
- We let camera rotation override this mask
- Depth is used to mask out nearby geometry If current pixel depth < nearby threshold write 0 Value used for blending out blur from first person arms/weapons
- Velocity mask is used later as a scale for motion blurring velocity offsets
 - Blurring amount scales at distance now

CMB Vertex Shader Sample

vPos.xyz += vWorldViewPos.xyz;

```
float4 vNewPos = mul(mViewProj, vPos);
float4 vPrevPos = mul(mViewProjPrev, vPos);
```

```
OUT.HPosition = vNewPos;
OUT.vCurr = HPosToScreenTC( vNewPos );
OUT.vPrev = HPosToScreenTC( vPrevPos );
```



CMB Pixel Shader Sample

```
half4 cMidCurr = tex2Dproj(screenMap, IN.vCurr);
half fDepth = tex2Dproj(depthMap,IN.vCurr).x*NearFarClipDist.y;
```

```
float2 vStart = IN.vCurr.xy/IN.vCurr.w;
float2 vPrev = (IN.vPrev.xy/IN.vVPrev.w) * fScale;
float2 vCurr = vStart * fScale;
```

```
float2 vStep = vPrev - vCurr;
float4 accum = 0;
```

```
[unrol1]
for(float s = -1.0; s < 1.0; s += fWeightStep ) {</pre>
```

```
float2 tcFinal = vCurr.xy - vStep.xy * s;
// Apply depth scaling/masking
half fDepthMask = tex2D(screenMap, tcFinal).w;
tcFinal += vStep.xy * (s - s * fDepthMask);
```

```
accum += tex2D(screenMap, tcFinal);
```

```
accum *= fWeight;
```

}

// Remove remaining scene bleeding from 1st player hands
OUT.Color = lerp(cMidCurr, accum, saturate(fDepth-1.0));

Improving quality

- Iterative sampling approach
- First pass uses 8 samples
- Ping-pong results
- Second pass uses blurred results, this results in 8
 * 8 samples (virtually 64)
- 3rd = 512 samples, 4th = 4096, etc
- High quality at relatively low cost



Iterative quality improve



Optimization strategies

- If no movement skip camera motion blur entirely
 Compare previous camera transformation with current
- Estimate required sample count based on camera translation/orientation velocity
 - If sample count below certain threshold, skip

Adjust pass/sample count accordingly

This gave a nice performance boost
 Average case at 1280x1024 runs at ~ 1 ms on a G80



Object Motion Blur (OMB)

LDR Bright streaks gone Washed out details



DX9 HW Skinning limitation

- 256 vertex constant registers limit
- Our characters have an average of 70 bones per drawcall
- Each bone uses 2 registers = 140 registers
- For motion blur we need previous frame bones transformations
- $2 \times 140 = 280$ registers, bummer...
- Decided for DX10 only solution

Step 1: Velocity pass

- Render screen space velocity, surface depth and instance ID into a FP16 RT
- If no movement, skip rigid/skinned geometry
 Compare previous transformation matrix with current
 Compare previous bone transformations with current
- If per-pixel velocity below certain threshold write 0 to RT

Can use this data for optimizing further



Why dilation needed ?





- Used a 8x smaller render target
- Apply Gaussian blur to velocity length
- Result is a reasonable fast estimation of screen space needed for dilation and motion blurring
- Mask is used to efficiently skip pixels during dilation passes



Step 3: Velocity Dilation

- Edge dilation
- Done using separated vertical and horizontal offsets
- 4 passes total (2 for horizontal, 2 for vertical)
- If center offset has velocity or velocity mask is 0 skip processing entirely
- Dilate if world depth > surface depth

Dilation shader sample

```
float4 vCenterVZID = tex2D(tex0, tc.xy);
float fCenterDepth = GetResampledDepth(tex1, tc.xy);
float fOffsetScale = tex2D(tex2, tc.xy).x;
```

```
if( fOffsetScale == 0 || dot(vCenterVZID.xy, vCenterVZID.xy) ){
   OUT.Color = float4(vCenterVZID.xyzw) ;
   return OUT;
```

```
}
```

}

```
[unrol1]
for( int n = 0; n < nOffsets; n++ ) {
  float2 tcLookup = tc.xy + vOffsets[n].xy *vScrSizeRecip;
  float4 vCurrVZID = tex2Dlod(tex0, float4(tcLookup , 0, 0));</pre>
```

```
float fDepthCmp = saturate( fCenterDepth- vCurrVZID.z );
fDepthCmp *= dot(vCurrVZID.xy, vCurrVZID.xy);
fDepthCmp *= Dilated.z == 0;
```

```
if( fDepthCmp)
Dilated = vCurrVZID;
```



Velocity Dilation



Final Step: Motion Blurring

- Accumulate N samples along velocity direction
- Can use surface ID to avoid leaking Extra lookup..
- Clamp velocity to acceptable range
 Very important to avoid ugly results, especially with jerky animations
- Divide accumulated results by sample count and lerp between blurred/non blurred
 - Using alpha channel blend mask

OMB Pixel Shader Sample

```
float4 cScreen = tex2Dlod(tex0, float4(tc.xy, 0, 0));
float4 cVelocity = tex2Dlod(tex1, float4(tc.xy, 0, 0));
OUT.Color = cScreen;
if( dot(cVelocity.xy, cVelocity.xy) < fThreshold )
   return OUT;</pre>
```

```
float4 cAccum = 0;
float fLen = length(cVelocity.xy);
if( fLen ) cVelocity.xy /= fLen;
cVelocity.xy *= min(fLen, vMaxRange) //Clamp velocity to MaxRange
```

```
[unroll]
for(float i = 0; i < nSamples; i++) {
  float2 tcMB = cVelocity * ((i * fRecipSamples)-0.5) + tc;
  float4 cCurr = tex2Dlod(tex0, float4(tcMB, 0, 0));
  cAccum += float4(cCurr.xyz, saturate(10000 * cCurr.w));
}</pre>
```

```
if( cAccum.w ) { // Blend with scene
    cAccum *= fRecipSamples;
    OUT.Color = lerp(cScreen, cAccum, saturate( cAccum.w * 2) );
```

}



Blend Mask





Blend with scene





Object Motion Blur

- Object motion blur with per-pixel dilation
 Not perfect, but good quality results for most cases
 Geometry independent
 Problematic with alpha blending
- Future improvements / challenges
 Self-motion blurring
 Multiple overlapping geometry + motion blurring
 Could use adaptive sample count for blurring



aka Crepuscular Rays/God Rays/Sun beams/Ropes of Maui...



www.crytek.com

=Ń

Screen Space Sun Shafts

- Generate depth mask
 - Mask = 1 normalized scene depth
- Perform local radial blur on depth mask
 - Compute sun position in screen space
 - Blur vector = sun position current pixel position
- Iterative quality improvement
 Used 3 passes (virtually = 512 samples)
 RGB = sun-shafts, A = vol. fog shadow aprox
- Compose with scene
 - Sun-shafts = additive blending
 - Fog-shadows = soft-blend [5]



Depth Mask Radial Blurring

Depth Mask











Sun Shafts: Results







Color Grading

- Artist control for final image touches Depending on "time of day" in game Night mood != Day mood, etc
- Usual saturation, contrast, brightness controls and color levels like in Far Cry [6]
- Image sharpening through extrapolation [9]
- Selective color correction
 - Limited to a single color correction
- Photo Filter
- Grain filter



Image Sharpening







Selective Color Correction

- Color range based on Euclidian distance
 ColorRange = saturate(1 length(src col.xyz));
- Color correction done in CMYK color space [8]
 c = lerp(c, clamp(c + dst_c, -1, 1), ColorRange);
- Finally blend between original and correct color Orig =lerp(Orig, CMYKtoRGB(c), ColorRange);





Photo Filter

- Blend entire image into a different color mood
- Artist defines "mood color"
 - cMood = lerp(0, cMood, saturate(fLum * 2.0));
 - cMood = lerp(cMood, 1, saturate(fLum 0.5) * 2.0);
- Final color is a blend between mood color and backbuffer based on luminance and user ratio final= lerp(cScreen, cMood, saturate(fLum * fRatio));





14.81

SE

NE

N



120

Final artist tweaks

SE

N




Final artist tweaks



Conclusion

- Awesome time to be working in real time computer graphics
- Hollywood movie image quality still far away
- Some challenges before we can reach it
 Need much more GPU power (Crysis is GPU bound)
 Order independent alpha blending for real world cases
 Good reflection/refraction
 We still need to get rid of aliasing Everywhere



Special Thanks

To entire Crytek team

All work presented here wouldn't have been possible without your support [©]



- [1] Wenzel, "Real time atmospheric effects", GDC 2007
- [2] Tessendorf, "Simulating Ocean Water", 1999
- [3] Sousa, "Generic Refraction Simulation", Gpu Gems 2, 2004
- [4] Jensen + et al, "Deep Water Animation and Rendering", 2001
- [5] Nishita + et al, "A Fast Rendering Method for Shafts of Light in Outdoor Scene", 2006
- [6] Gruschel, "Blend Modes", 2006
- [7] Bjorke, "Color Controls", GPU Gems 1
- [8] Green, "OpenGL Shader Tricks", 2003
- [9] Ford + et al, "Colour Space Conversions", 1998
- [10] Haerberli + et al, *"Image processing by Interpolation and Extrapolation"*, 1994



Questions?

Tiago@Crytek.de

www.crytek.com



Crytek is hiring !

http://www.crytek.com/jobs/

www.crytek.com