SIGGRAPH2011

Secrets of CryENGINE 3 Graphics Technology

Contents:

- Rendering Pipeline
- Position Reconstruction
- Coverage Buffer
- Deferred Lighting
- Shadows
- Screen Space Techniques
- Deferred Techniques
- Batched HDR Post Processing
- Stereo Rendering

Secrets of CryENGINE 3 Graphics Technology

Nickolay Kasyan — Senior Graphics Engineer
Nicolas Schulz — Senior Graphics Engineer
Tiago Sousa — Principal Graphics Engineer

Crytek

SIGGRAPH2011

Physically Based Rendering

- Linear Correct HDR Rendering [Gritz 2008] [Reinhard 2010]
- Minimal G-Buffer: Depth and Normals
- Opaque, decals, def. decals,terrain layers
- Deferred Lighting
- Ambient, env. lighting probes
- GI, SSDO, RLR, Lights
- Physically based shading
- Opaque / Transparent passes
- HDR / LDR Posts

G-Buffer/L-Buffers/Scene Targets

- Slim G-Buffer
  - A8B8G8R8 World Space BF Normals + Glossiness
  - Readback D24S8 Depth + Stencil bits for tagging indoor surfaces
- 2x A2B10G10R10F_EDRAM for Diffuse and Specular buffers
  - X360: Resolved to A2B10G10R10 [Cook 2008]
  - PS3: 2x A8B8G8R8 encoded in RGBM
    - Encoding <=> No HW blending possible. Workaround: r/w from dst RT
  - PC: 2x A16B16G16R16F (most general format)
- A2B10G10R10F_EDRAM for scene target
  - PS3: A8B8G8R8 RGBM encoded for opaque, FP16 for transparents
  - PC: A16B16G16R16F

Z-Buffer Depth Caveats

- Hyperbolic distribution
  - Needs conversion to linear space before using in shaders

```
//Constants
g_ProjRatio.xy = float2( zfar / (zfar-znear), znear / (znear-zfar) );
//HLSL function
float GetLinearDepth(float fDevDepth)
{ return g_ProjRatio.y/(fDevDepth-g_ProjRatio.x);}
```

Problem: First person view objects

- Depth buffer is used to prevent overlapping FPV objects with the rest of the scene
- Different FOV and near/far plane (art specific choice)
- Different depth range to prevent actual overlapping
- => Deferred techniques don't work 100% for such objects

Addressing First Person View Objects

- Our final solution:
  - Modifying depth reconstruction function
  - Convert hardware depth to linear one
  - Different depth scale for first person view object
  - Selecting based on depth

```
float GetLinearDepth(float fDevDepth) {
    float bNearDepth = step(fDevDepth, g_PS_DepthRangeThreshold);
    float2 ProjRatio.xy = lerp(g_PS_ProjRatio.xy, g_PS_NearestScaled.xy, bNearDepth);
    return  ProjRatio.y/(fDevDepth-ProjRatio.x);
}
```

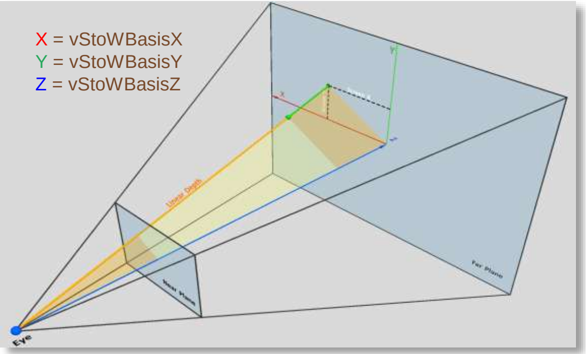
Reconstructing Position from Depth

SIGGRAPH2011

- Idea: linearly transform VPOS from screen space S directly to target homogeneous space W (shadow space or world space)

```
float4 HPos = (vStoWBasisZ + (vStoWBasisX*VPos.x)+(vStoWBasisY*VPos.y) ) *
    fSceneDepth;
HPos += vCamPos.xyzw;
```

- Direct transformation from screen clip space to homogeneous matrix
- VPOS is simplest way to render deferred light volumes
- Separate adjustment for s3D



Geometrical Meaning of Reconstruction

Coverage Buffer

- Mostly outdoors for Crysis2
  - Approximately 70%
  - Portals or PVS not efficient there
- Coverage Buffer as main occlusion culling system
  - Essentially low resolution depth buffer
  - Coarse CPU rasterization of object AABBs/OBBs for visibility z-tests
- Too slow to prepare fully detailed C-Buffer on CPU
  - Huge computation cost
  - Full rendering pipeline has to be duplicated in software to obtain all details for C-Buffer

Coverage Buffer

- Read-back of previous frame's GPU depth buffer on CPU
  - Downscaling ZBuffer on GPU (max filter) after G-Buffer pass
  - Culling done by rasterizing BBoxes in a separate CPU thread

Original:	1920x1080
Downscaled:	256x128

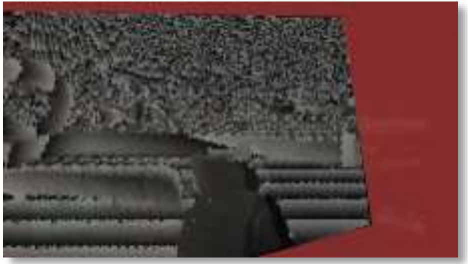
Coverage Buffer

- Used on X360/PS3 and DX11 HW
  - Consoles perfect for this (1 frame latency)
  - Read-back latency on PC is higher but still acceptable (up to 4 frames)
  - C-Buffer size limited to 256x128 in Crysis 2 on consoles
  - Problem: Mismatch between previous/current frame cameras
  - Results in wrong visibility tests

Coverage Buffer Reprojection

SIGGRAPH2011

- Ended up using C-Buffer CPU reprojection from prev. frame camera
  - Point splatting of reprojected fragments
  - Camera info is encoded into the c-buffer data
- CPU readback and reprojection in separate thread
  - ~2ms on SPU, ~3-4ms on Xbox 360 with vectorized code
- Stitching of holes inside C-Buffer after reprojection
  - 3x3 dilation pass
  - Remaining C-Buffer holes: we assume objects are visible
- Reprojection improved culling efficiency greatly
  - Solved all kind of occlusion test artifacts, detected invalid areas
  - Works more efficiently with higher framerate



Remaining C-Buffer holes after reprojection

Deferred Lighting: Ambient

- Outdoor / Indoor
  - Stencil tagged regions in G-Buffer passes
  - FS Quad and indoor volume BBox
  - Additive blended

Deferred Lighting: Environment Probes

- Accurate diffuse lighting and specular lighting
  - Artist pick important sampling locations
  - HDR encoded (RGBM)
  - Linear blending for diffuse and specular cube map
  - Spherical light volume
- G-Buffer material glossiness used for picking mip level
  - Consistent specular behaviour between IBL and regular lights
- Alpha blended passes? Pick nearest probe

Deferred Lighting: Environment Probes

SIGGRAPH2011

Deferred Lighting: GI, SSDO, RLR, Lights

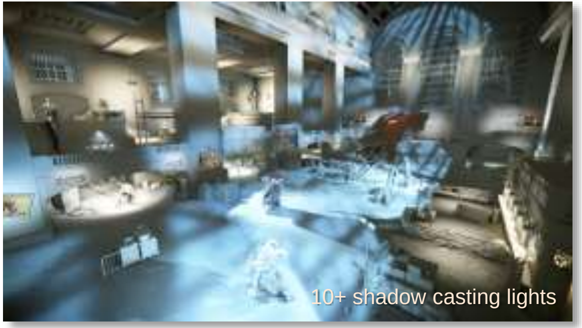
- Add GI [Kaplanyan 2010] (add blend)
- Apply SSDO (mul blend) and RLR (alpha blend)
- Add light sources
  - Lights rendering depends on screen coverage heuristics
    - FS quads with stencil volumes pre-passes, convex light volumes or 2D quads
  - Normalized Blinn-Phong [Hoffman 2010]
  - For C2, only projectors and point lights
  - Limited shadow casters count on consoles - go bananas on PC

Shadows

Deferred Shadows

- Shadow mask for sun
  - Special render target to accumulate shadow occlusion
  - Shadow mask combines multiple shadowing technique on top on each other before using with actual shading
- Point light shadows rendered directly to the light buffer

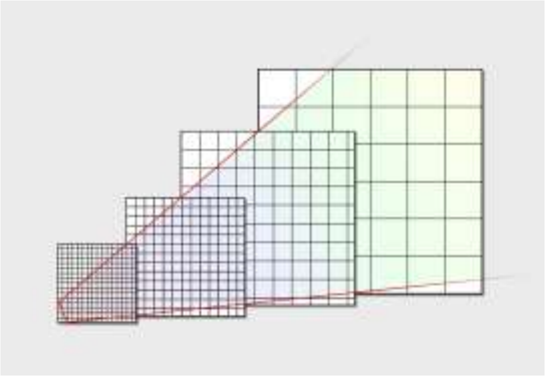
Sun shadow mask



10+ shadow casting lights

Cascaded Shadows Maps

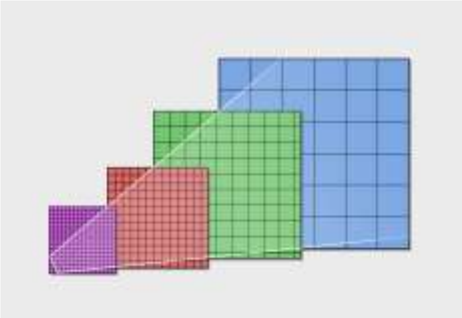
- Cascaded shadow maps used since Crysis 1
- Cascades Splitting Scheme
  - Approximate Logarithmic texel density distribution
  - Shadow frustums adjusted to cover camera view frustum conservatively
  - Orientation for shadow frustums is fixed in world space

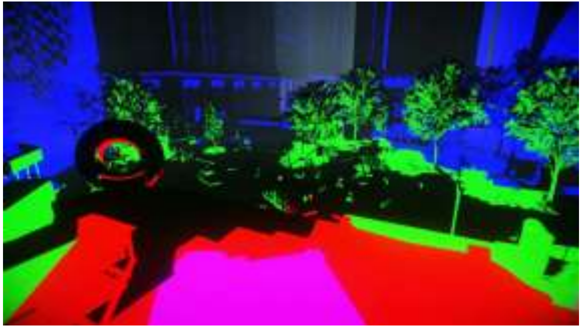


- More cascades allow
  - Improved texel density, reduced acne and improved self-shadowing for wider shadow range due to better approximation of the logarithmical distribution
- For each cascade snap the shadow frustum to the SM's texel grid

Deferred Shadow Passes

- Shadow passes for cascades/point lights rendered in deferred way
- Potential shadow receiving areas tagged in stencil buffer by rendering frustum volumes
  - Allows to have more sophisticated splitting into cascade
  - Pick a cascade with the highest resolution in overlapping regions
  - Avoids wasting of shadow map space





Shadow Cascades Caching

- Not all the cascades are updated during a single frame
  - Update cost distributed across several frames
  - Performance reasons (PS3 particularly)
- Allows us to have more cascades – better shadow map density distribution
- Cached Shadow Maps use cached Shadow Matrices
- Distant cascades are updated less frequently
- Last cascade uses VSM and blends additively with the shadow mask
  - Allows to have large penumbras from huge distant objects

Point Light Shadows

SIGGRAPH2011

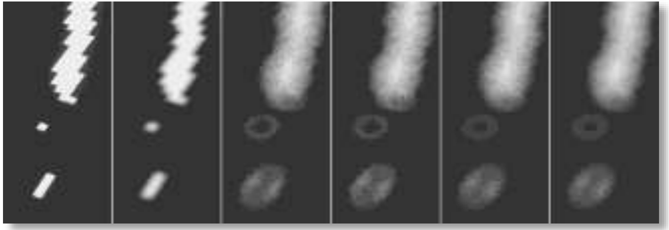
- We always split omni-directional lights into six independent projectors
- Shadow map for each projector is scaled separately
  - Based on the shadow projection coverage
  - Final scale is a result of logarithmic shadow map density distribution function, which uses the coverage as a parameter
- Big texture atlas to pack all shadow maps each frame after scaling
  - Texture atlas is allocated permanently to avoid memory fragmentation
  - Size on consoles: 1024x1024 (4 MB)
- Receiving area tagged by stencil

Shadow atlas

Soft Shadows Approximation

Soft Shadows Approximation

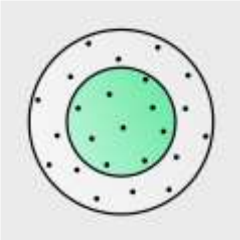
- We use Poisson PCF taps with randomized rotations in shadow space



- Adjusting the kernel size at runtime allows to have a good approximation of soft shadows with variable penumbra
- Soft shadows idea: Estimate average distance ratio to shadow casters
- Similar to Percentage-Closer Soft Shadows [Randima2005]

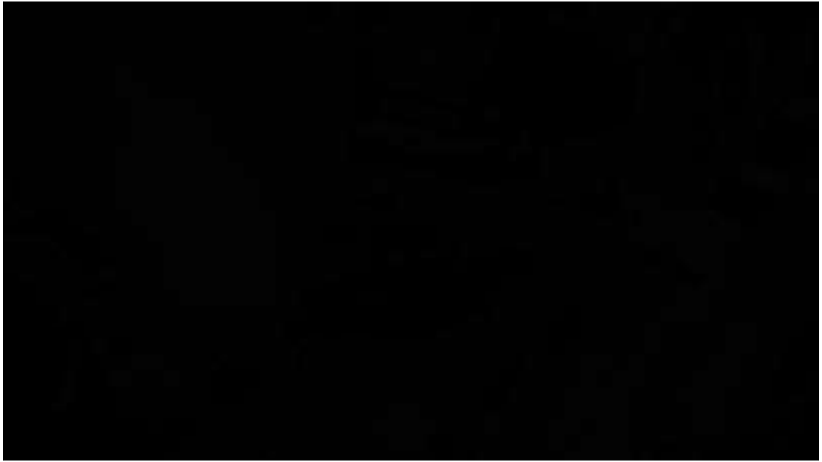
Soft Shadows Approximation

- Basic Algorithm:
  - Poisson distributed taps are presorted by distance from the kernel center
  - Initial kernel radius set to match maximum range (= biggest/longest penumbra)
  - Use this kernel to estimate the average distance ratio
  - The number of samples is reduced proportionally to the avg. distance ratio
    - This affects the radius of the kernel since the taps are sorted
  - Use only the reduced number of samples for final shadow computation
- Cascade shadow maps need custom kernel scale adjustment to handle transitions between cascades
- Compute Shader option: fetch all taps to CS shared memory and reuse them for both distance estimation and shadow computation



Shadows & Transparency

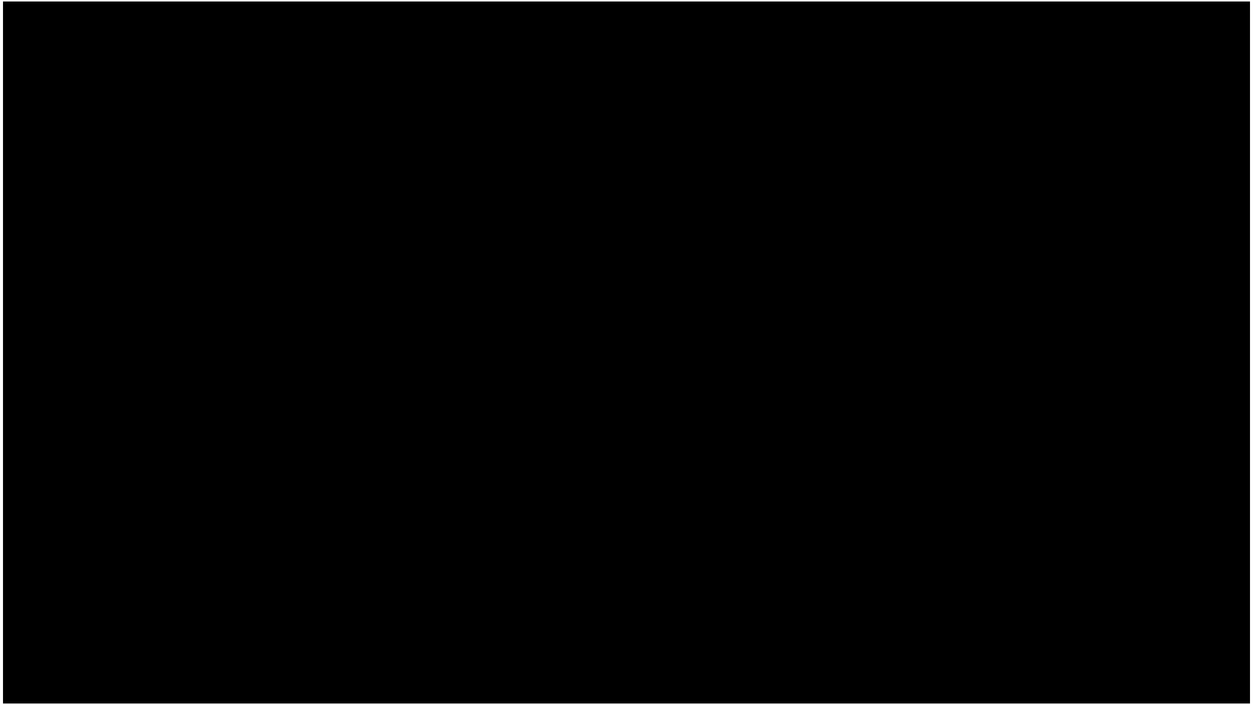
- For alpha blended shadow receivers
  - Forward passes to apply shadows
- For transparent shadow casters we accumulate the alpha values of the casters
  - Stored in 8-bit render target



Shadows & Transparency

- Translucency map generation:
  - Depth testing using depth buffer from regular opaque shadow map to avoid back projection/leaking
  - Alpha blended shadow generation pass to accumulate translucency alpha (sorted back to front)
  - In case of cascaded shadow maps, generate translucency map for each cascade
  - Shadow terms from shadow map and translucency map are both combined during deferred shadow passes with max() operation

Shadows Video



Real Time Local Reflections

SIGGRAPH2011

Real Time Local Reflections

- Reflections are expensive with rasterization
  - Usually planar reflections or cubemaps
  - Require re-rendering of scene
- Standard reflections limited
  - Either planar surface
  - Tiny area for cube maps
  - Usually no curved surfaces
- Reflections straightforward with raytracing
  - Raytracing in screen space to approximate local reflections
  - First iteration was demonstrated at GDC 09

Real Time Local Reflections (RLR)

- Basic Algorithm
  - Compute reflection vector for each pixel
    - Uses deferred normal and depth targets
  - Raymarch along reflection vector
  - Sample depth and check if ray depth is within threshold to scene depth
  - If hit, reproject into framebuffer of previous frame and sample color
- Results
  - Relatively cheap
  - Local reflections everywhere (even on complex surfaces)
  - Very cool where it works ☺
  - Plenty of problematic cases due to limited data in screen space

Real Time Local Reflections (RLR)

SIGGRAPH2011

- Implementation Tips
  - Very limited screen space data
    - Rather no reflections than broken looking ones
    - Smoothly fade out if reflection vector faces viewer as no data is available in that case
    - Smoothly fade out reflection samples at screen edges
  - Add jittering to step size to hide noticeable step artifacts
  - HDR color target sampled in Crysis 2
  - Jitter or blur based on surface glossiness

Contact Shadows

- Core idea same as SSDO [Ritschel 2010]
  - Modulate lighting with computed screen space occlusion
  - Produces soft contact shadows
  - Can also hide shadow bias issues
  - Considerable quality gain over just SSAO
- However, directional occlusion info accessible in a deferred way
  - Fits better into existing lighting pipeline
  - Can be applied efficiently to every light source

Contact Shadows

Contact Shadows

- Occlusion info generation
  - Compute and store bent normal N' during SSAO pass
    - Bent normal is average unoccluded direction
  - Requires clean SSAO without any self-occlusion and relatively wide radius

- For each light
  - Compute N dot L as usual
  - Compute N' dot L
  - Attenuate lighting by occlusion amount multiplied with clamped difference between the two dot products

Deferred Skin Shading

- Main Idea: Reuse diffuse lighting accumulation
  - Proof of concept since beginning of project (early 2008)
- Subsurface scattering in screen space
  - Gather lighting taps during geometry pass
  - Used a Poisson distribution
- Best bits
  - No additional memory requirements for atlas
  - No redundant work
  - Concept expandable to UV space [Borshukov 2001]
  - Cost proportional to screen area coverage

Head model courtesy of Infinite-Realities

Screen Space Self-Shadowing

- Couldn't afford per-character shadow map (memory)
  - How to workaround lack of memory on consoles ?
- Simple trick/approximation
  - Ray march along screen space light vector
  - Macro self-shadowing details for all characters, even on consoles




Soft Alpha-Test

- Efficient hair rendering is a pain on this hardware generation
- Robust solutions:
  - Alpha test looks very 1999. >= 4x SSAA ? Not yet for consumer HW
  - ATOC doesn't look so good and requires hardware MSAA
  - Stippling + filtering ? Kind of works, but half res'ish look
- Idea: Post process for anti-aliasing the alpha test look
  - Another oldie from beginning of project (early 2008)
  - Directional blur (8 taps) along Tangent vector in Screen Space
  - Visual hint for transparency and smooth hair look
  - Additional Hair geometry pass
- Fur rendering? Directional blur along Normal vector in SS

Soft Alpha-Test

Batched HDR Post Processing

Camera & Object Motion Blur

- Re-projection for static + velocity buffer for dynamic obj.
  - Full resolution means at least ~3 ms spent (consoles)
- Half resolution and RGBM encoded
  - Total 16 x less BW versus full res and fat formats
- Composition mask for blending with full resolution
- Object velocity buffer dilation [Sousa 2008] on the fly
- All done in linear space before tone mapping
  - Bright streaks are kept and propagated [Debevec 1998]
  - Consoles: 9 taps; PCs higher specs: 24 taps

Camera & Object Motion Blur

Half-Res linear input (RGBM)

RT0: Blurred Target (RGBM)

Camera & Objects V

RT1:Composite Mask

Bokeh DOF: Another Kernel and Weights

Half-Res linear input (RGBM)

RT0: Blurred Target (RGBM)

CoC

RT1:Composite Mask

Maximum Batching

- It's just a blur we really want
  - How to reuse all taps/bw, share all gpu work ?
- Idea: Offsets morphing based on blur type
  - Camera/Objects moving? Directional versus disk kernel
  - More uses: masked blur, radial blur, etc
- Final composite done directly in tone mapping pass
- 1 ms on consoles. Super fast on PC at 1080p
  - Almost 10x faster than doing all posts individually (at fullres)

Ultra Specs: Motion Blur

- Single pass at full screen resolution
  - 12 taps + clamp maximum range to limit undersampling
  - Alpha channel stores objects ID
- Blur masking scheme based on velocity and ID
  - ||V|| > threshold ? Allow bleeding, else reject tap
  - Early out if ||V|| < threshold

Ultra Specs: Bokeh DOF

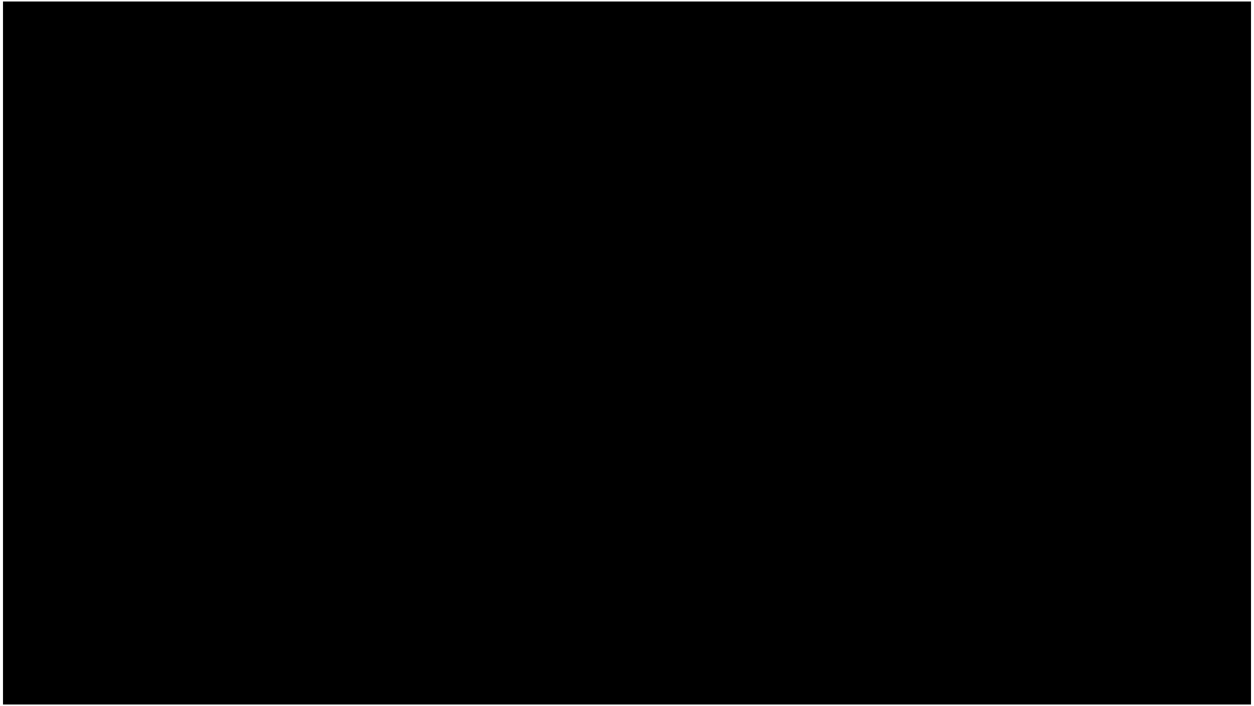
- Go Bananas: Render a Quad/Sprite per pixel [Cyril 2005]
  - GS to scale quads by CoC factor
- Accumulate results into Foreground/Background targets
  - Masking by sorting quads per layers
  - Alpha channel stores quad count
  - Dithering to minimize precision loss
- Composite with final scene
  - Divide layers result by layer alpha
- Great Quality ! (but very slow)

Ultra Specs: Bokeh DOF

SIGGRAPH2011

- Making it fast
  - Render Half Res
  - Reject Quads in interleaved pattern
  - Early out for VS/PS
  - Spherical aperture using ALU
  - Future: avoid geometry shader usage
- Composite with final scene
  - Back layer composited using full resolution CoC
  - Front layer is ok to bleed anyway

Ultra Specs Post Processes Video



What if…

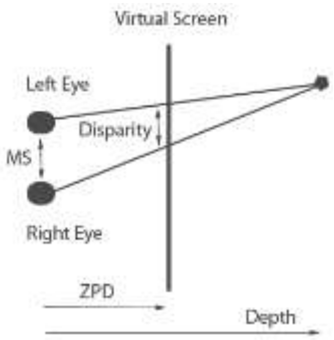
S3D Image Generation

S3D Image Generation

- Standard approach: Render scene two times
  - Great quality, pretty straightforward
  - Problematic for GPU heavy games
    - Often means half the framerate
  - Would require heavy quality reduction for C2
    - Resolution, LODs, view distance, effect quality/amount
- Image space approach: Reprojection
  - Reconstruct view/world space position of pixel and project into space of left/right eye cameras
  - Basically point splatting
  - Scattering not efficiently possible on D3D9/10 GPUs
  - Requires second hole filling pass

S3D Image Generation

- Image offsetting in Pixel Shader
  - Gather based approach
  - For each pixel, compute disparity from depth buffer
  - Sample backbuffer with positive/negative disparity as offset to generate warped image (using bilinear filtering)



Disparity computed using Thales Theorem

Disparity: distance between projected positions of point in left/right views

MS: maximum separation (e.g. 3% of screen)

ZPD: zero parallax plane distance

$$\text{Disparity} = \text{MS} * (1 - \text{ZPD} / \text{Depth})$$

S3D Image Generation

- Algorithm samples backbuffer with offset
  - Issues due to missing data in main view
- Problem: Occlusion
  - Background should be sampled but foreground object is in image
  - Easy to find by checking for closer depth, replicate background instead
  - Our solution: Find minimum depth and use it for computing disparity
    - Small depth means small offset as well
    - Adds some randomization to the replication

S3D Image Generation: Shader Sample

SIGGRAPH2011

```
const float samples[3] = { 0.5, 0.66, 1 };
float minDepthL = 1.0, minDepthR = 1.0;
float2 uv = 0;

for( int i = 0; i < 3; ++i ) {
    uv.x = samples[i] * MaxSeparation;
    minDepthL = min( minDepthL, GetDepth( baseUV + uv ) );
    minDepthR = min( minDepthR, GetDepth( baseUV - uv ) );
}

float parallaxL = MaxSeparation * (1 – ZPD / minDepthL );
float parallaxR = MaxSeparation * (1 – ZPD / minDepthR );

left = tex2D( backBuf, baseUV + float2( parallaxL, 0 ) );
right = tex2D( backBuf, baseUV - float2( parallaxR, 0 ) );
```

S3D Image Generation

- Simple and very efficient, about 1 ms on consoles
  - Works with standard stereo parameters, negative and positive parallax
  - Overall results very similar to rendering two times
- Does not handle disocclusion
  - Works well enough for scenes with sparse silhouettes (e.g. city scene with huge buildings) and carefully chosen stereo parameters
  - More problematic for very close objects like FP weapon
    - Huger area with missing information
    - See slight halos around objects (replicated background)
- Does not work for transparent objects (not in depth buffer)
  - They get separation of background (somehow acceptable if overall scene depth is limited)
- Left/right screen edges need special consideration
  - For example cropping image

S3D Image Generation

- Conclusion
  - Technique is far from being perfect
    - Many potential issues
    - Good amount of tweaking required
  - Very well received for Crysis 2, especially on consoles
    - No loss of visual fidelity in s3D
  - Can be an option if you can live with some artifacts and limited depth range

S3D Image Generation

SIGGRAPH2011

- Respecting some rules is essential to create a comfortable experience
  - Everything goes into the screen in Crysis 2
    - No window violation possible
    - No extreme refocusing required
  - Avoiding depth conflicts
    - Perceived 3D depth does not match rendering
    - E.g. crosshair is deeper than wall but rendered in front of it
    - Very annoying, causes unpleasant side effects for viewers

S3D Image Generation

- 3D HUD
  - Core elements placed carefully in 3D space to avoid intersections with world
- Crosshair
  - Pushed into the world to be in front of player
  - Can often cause depth conflicts
  - Cast ray into world, check for intersection
  - Adapt position smoothly in case of intersection

S3D Image Generation

- Certain ZPD required to get acceptable parallax distribution
  - Problematic for FPS games
  - Weapon would come out of screen
- Solution
  - Weapon pushed into screen during stereo image generation
  - However, can cause depth conflicts when close to wall
  - To avoid this, check for close objects and reduce ZPD to fade out stereo effect smoothly

Conclusion

SIGGRAPH2011

- 3.5 years ⇔ immense amount of work
  - We only covered here a very small sub-set ☺
- Whats next for CryENGINE ?
  - Bigger, better, faster. Avatar Quality in Realtime ?
    - Not that crazy, with properly done assets for realtime
  - We need faster consoles for a BIG “next gen” step:
    - Big = huge visual step, as in Far Cry to Crysis 1
    - Insane amount of GPU Power (4x NV 590) ☺
    - Insane amount of memory (>= 8 GB)

Special Thanks

- Vaclav Kyba, Michael Kopietz, Carsten Wenzel, Vladimir Kajalin, Andrey Konich, Anton Knyazyev, Ivo Zoltan Frey, Ivo Herzeg
- Marco Corbetta, Christopher Evans, Chris Auty
- Magnus Larbrant, Pierre-Ives Donzallaz, Chris North
- Natalya Tatarchuk

- And to the entire Crytek Team

References

- Gritz, L. “The Importance of Being Linear”, 2008
- Debevec, P. “Recovering High Dynamic Range Radiance Maps from Photographs”, 1998
- Tchou, C. “HDR The Bungie Way”, 2008
- Vlachos, A. “Post Processing in The Orange Box”, 2008
- Cook, D. “Xbox Textures – Formats, Conversion, Fetching and Performance”, 2008
- Valient, M. “The Rendering Technology of Killzone 2“, 2008
- Kaplanyan, A. “Real-time Diffuse Global Illumination in CryENGINE 3”, 2009
- Sousa, T. “Crysis Next Gen Effects”, 2008
- Reinhard, E. et al. “High Dynamic Range Imaging” , 2010
- Hoffman, N. et al. “Physically-Based Shading Models in Film and Game Production”, 2010
- Sousa, T. “CryENGINE 3 Rendering Techniques”, 2011
- Grosch T. and Ritschel T. “Screen-Space Directional Occlusion”. GPU Pro, 2010
- Randima, F “Percentage Closer Soft-Shadows”, 2005
- Krassnigg, J “A Deferred Decal Rendering Technique”, Game Engine Gems 1, 2011
- Cyril, P. et al.”Photographic Depth of Field Rendering”, 2005
- Borshukov, G. and Lewis, J.P “Realistic Human Face Rendering for “The Matrix Reloaded””, 2001

Questions

SIGGRAPH2011

Tiago@crytek.com / Twitter: Crytek_Tiago

Nicolas@crytek.com

Nick@crytek.com

Bonus Slides

HDR & Linear Correctness

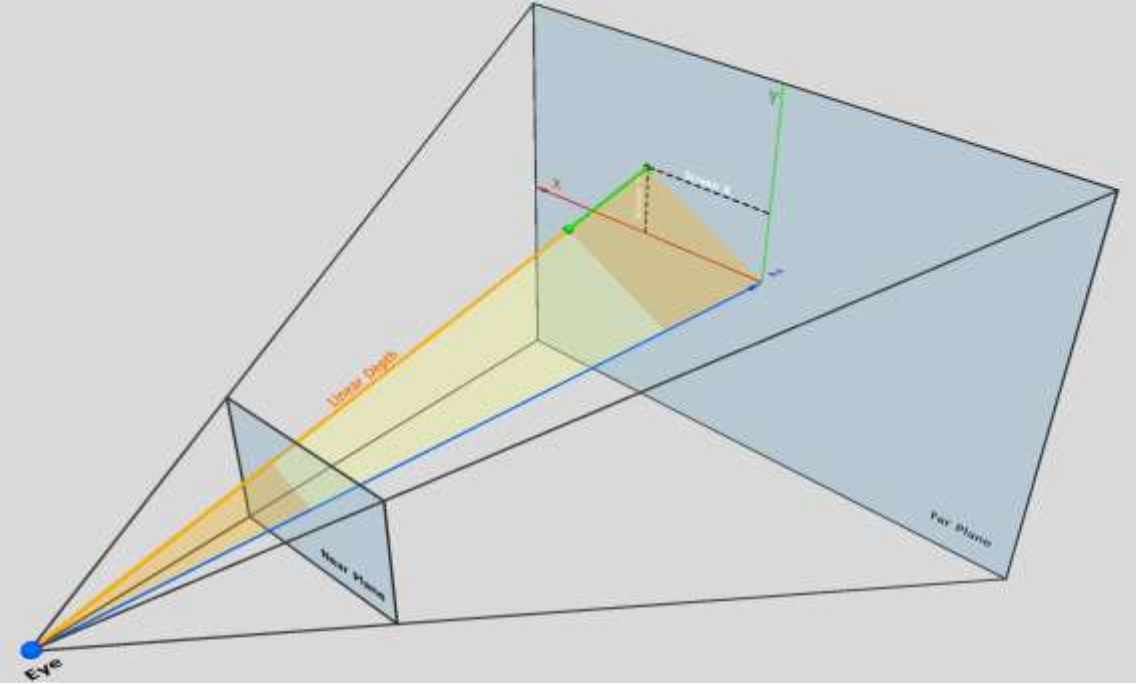
- HDR [Reinhard 2010]
  - Precision, range
  - Physically based post processing
- Linear Correctness [Gritz 2008]
  - All computations in exact same space

Computing VPOS warp basis

SIGGRAPH2011

```
float fProjectionRatio = fViewWidth/fViewHeight;    // projection ratio
//all values are in camera space
float fFar = cam.GetFarPlane();
float fNear = cam.GetNearPlane();
float fWorldHeightDiv2 = fNear * cry_tanf(cam.GetFov()*0.5f);
float fWorldWidthDiv2 = fWorldHeightDiv2 * fProjectionRatio;
float k = fFar/fNear;
Vec3 vStereoShift = camMatrix.GetColumn0().GetNormalized() * cam.GetAsymL();
//apply matrix orientation
Vec3 vZ = (camMatrix.GetColumn1() * fNear + vStereoShift)* k; // size of vZ is the distance from camera pos
     to near plane
Vec3 vX = camMatrix.GetColumn0() * fWorldWidthDiv2 * k;
Vec3 vY = camMatrix.GetColumn2() * fWorldHeightDiv2 * k;
vZ = vZ - vX;
vX *= (2.0f/fViewWidth);
vZ = vZ + vY;
vY *= -(2.0f/fViewHeight);
//Transform basis to any local space (shadow space here)
vWBasisX = mShadowTexGen * Vec4 (vX, 0.0f);
vWBasisY = mShadowTexGen * Vec4 (vY, 0.0f);
vWBasisZ = mShadowTexGen * Vec4 (vZ, 0.0f);
vCamPos =  mShadowTexGen * Vec4 (cam.GetPosition(), 1.0f);
```

Geometrical Meaning of Reconstruction



Shadow Acne

- Two main reasons
  - Low shadow map texel density
  - Precision of depth buffers
- Different scenarios to overcome acne
  - Sun shadows: front faces rendered with slope-scaled depth bias
  - Point light shadows: back face rendering, works better for indoors scenes
  - Variance shadows for distant LODs - render both faces to shadow maps
- Constant depth bias during deferred shadow passes to overcome depth buffer precision

Minimizing Light Bleeding

- Can't afford each light casting shadows on consoles
- Clip Boxes / Volumes
  - Tool for lighting artists to eliminate light bleeding using stencil culling
  - Each deferred light doesn't need to have fully generated shadow maps
  - Artist can create arbitrary convex volumes or use default primitives

Deferred Decals

- Forward Decals have quite some issues
  - Additional memory, mesh re-allocations causing memory fragmentation, CPU time for dynamic mesh creation, etc

- Deferred decals to the rescue! [14]
  - Rendered as a decal box volume, very similar to deferred lights
  - Separate diffuse layer + normal map buffer blending
  - Fetching diffuse texture and computing attenuation for shading
  - No lighting/shading computed here
  - Applied to static geometry only

Deferred Decals

Initial scene

Decals diffuse layer

Road Decals

Final scene

Deferred Decals

SIGGRAPH2011

- Problems
  - In case decals have normal maps results in tangent space mismatch with the regular decals
  - Leaking of deferred decals

Deferred Decals

SIGGRAPH2011

- Solutions for leaking
  - Adjustable decal volume and
    decal attenuation function
  - Dot product between Scene Normals and Decal Tangent Space Normal
    - Problematic if decal uses normal maps itself
    - X360: render to EDRAM and always have separate copy in the resolved Scene Normals texture
    - PS3 : render target read/write during convex volume’s rasterization