Rendering Techniques in Gears of War 2

Niklas Smedberg
Senior Engine Programmer
Epic Games, Inc.

Daniel Wright
Engine Programmer
Epic Games, Inc.
About Epic Games

• 19th year of business
• 110 employees in NC and still hiring
• Creators of Unreal Engine 3
• Shipped Gears of War 2
• Shipped Unreal Tournament 3
  — Xbox 360, PC and PS3
• External studios in Poland, Shanghai, and Utah
About Gears of War 2

- 93% Metacritic rating
- Over 4 million copies sold
- 117 award wins and nominations
- Shipped in two years by Gears 1 team
About Niklas Smedberg

• A.k.a. “Smedis”
• C64 Swedish demo scene
• Went pro 1997
• Moved to the US in 2001
• Shipped titles for PS1, PC, Xbox, Xbox 360, PS3
• Joined Epic Games 2005
Topics
Niklas Smedberg

• Gore techniques
• Blood techniques
• Character lighting
• Screen Space Ambient Occlusion optimizations
THE FOLLOWING PRESENTATION HAS NOT BEEN RATED FOR ALL AUDIENCES

VIEWER DISCRETION IS ADVISED
Gore Goals

- Goals:
  - No seams between breakable pieces (Gears of War 1 used rigid 1-bone skinning)
  - Dismember skinned meshes (e.g. ragdolls)
  - Hand-modeled internals
  - Minimal impact on undamaged meshes
Gore Mesh

- Artists make two versions of the skinned mesh
- Undamaged mesh with no overhead
- Gore mesh with additional information:
  - Completely independent of the undamaged mesh
  - Full freedom to hand-model cuts and guts
  - Can use different/additional textures and shaders
  - Pre-cut with all gore pieces separated
Gore Mesh

- Skeleton with breakable joint constraints
  - Broken by script code - not using physics engine (as in Gears 1)
  - More control of when/how to break the constraints
- Info per gore piece:
  - Hit points
  - Type of special effect
  - Dependency of other gore pieces
  - Etc
- An extra set of bone weights for all vertices:
  - Hand-made 4-bone weights
  - Weighted to properly separate all gore pieces along the cuts
Tearing off a limb

• Switch to the gore mesh
• Determine which constraint to break
• Get the pair of broken bones
• Create a separate vertexbuffer for vertex weights
  – Unique per gore mesh instance
  – Replaces all weights in the original vertexbuffer
  – Used as a separate vertex stream
• For each vertex:
  – If influenced by a broken bone, copy 2nd set of weights
  – Otherwise, copy 1st set of weights
• Add physics impulse to the torn-off limb
  – Let physics move the limb away
Data-driven Gore

- Set up in the Unreal Editor
- Only used for really big enemies (e.g. Brumak)
- Used as visual “damage states”
  - Not necessarily to break off whole limbs
- Gore pieces stacked on top of each other
  - Break “armor pieces”, damage flesh, etc.
- Hit-points, impact radius, dependency links (leaf first), etc.
Data-driven Gore, 1 of 3
Data-driven Gore, 2 of 3
Data-driven Gore, 3 of 3
Scripted Gore

• Set up in gameplay code (Unreal Script)
• Only dismember dead people
  – No animation/gameplay consequences
• Different damage types break constraints differently
  – Complete obliteration (e.g. grenade-tag)
  – Partial obliteration (e.g. shotguns)
• Headshots spawn additional effects (gibs)
  – Hand-made gib meshes can be specified per Pawn
• Chainsaw breaks specific constraints
• Ragdolls break constraints based on hit location
Next Example: Meatshield

- Meatshields break constraints in a pre-determined sequence
  - Based on overall “health” for the entire meatshield
  - Used as gory visual indicator instead of a boring “health bar”
Data-driven gore #3
• Screenshot: Meatshield 4
Data-driven gore #6
Next Example: Chainsaw

- Chainsaw kill:
  - Gameplay code choosing a specific constraint to break
Scripted gore

• Movie: Chainsaw
Topics
Niklas Smedberg

- Gore techniques
- Blood techniques
- Character lighting
- Screen Space Ambient Occlusion optimizations
Blood Techniques

• Many different techniques used in combination:
  – Projected blood decals
  – Screen-space blood splatter
  – World-space particle effects
  – Surface-space shader effects
  – Fluid simulations
  – Morphing geometry
Blood Techniques

• Movie: Riftworm heart chamber, chainsawing through arteries
Decal Features

- Decal system much improved over Gears 1
- Much easier to spawn decals on anything
- Supports any artist-created material (shader), lighting, blend mode, normal maps, animated, render-to-texture, etc
- Three decal pools:
  - Blood decals, explosion decals, bullet decals
  - Blood splatter unaffected by the number of bullet hit-marks
- Heuristic to replace oldest decal (when necessary):
  - Oldest not visible decal (LastRenderTime)
  - Oldest visible decal (Lifetime)
- Proximity check to avoid multiple decals on top of each other
Decal Optimizations

- AABB tree to find triangles within the decal frustum
  - These triangles are not clipped
- Decals have their own index buffers
  - Re-use whole triangles from the base mesh vertexbuffers
- Frustum-clipping handled by GPU:
  - clip(), early-out branch, scissor test
- Decal visibility tests:
  - Re-use base mesh visibility test
  - Decal-specific frustum and distance check
- Decals on fractured meshes:
  - Re-use the fragment-visibility check
  - Upon fracture, re-calculate affected decals only
Decal Optimizations

• One drawcall per decal, but:
  – This allows for scissoring
  – Each decal is fully featured

• Statically lit decals re-use lightmaps from underlying surface

• Dynamically lit decals use additive multi-pass lighting

• Decals on dynamic objects:
  – Projections are in local space
  – Re-use matrix to follow the mesh

• Decals on static objects:
  – Placed in the various static drawlists (“fire and forget”)

• Static decals (manually placed in editor):
  – Pre-clipped geometry (no clipping on GPU)
Example: Blood Smears

- Project decals on ground and on cover walls when you’re hurt
- Project a new decal every 0.5s while moving
- Using fading “paintbrush” effect along movement direction
  - Overall fade + “paintbrush” fade
  - Shader parameter curves exposed to artists
- Mix with standard blood decals to break up pattern
Blood Smears, 1 of 4
Blood Smears, 2 of 4
Blood Smears, 3 of 4
Blood Smears, 4 of 4
Screen-space Blood Effects

• Screen-space blood splatter:
  – Spawned by level design (Kismet), animation (AnimNotify nodes) or script code
  – 3D particle effects that use camera-facing particles
  – Game code re-positions these camera-effects each frame
  – Allows for more dynamic “screen-space” effects
    • Full-featured 3D particle effects
    • Artist-made materials (shaders)
    • Like any material, can take shader parameters from gameplay script code
    • Meshes, sprites, movie textures, normal-mapping, etc
World-space Blood Effects

- World-space 3D particle effects
- Use world-space coordinates for fake lighting on sprites and gibs
  - Can make up-facing pixels brighter, etc
  - Normal-mapped sprites helps
- 4x6 “movie frames” in a sub-UV texture to get realistic motion
  - 2048x1024 color map + 1024x512 normal map
  - Re-used for most liquid-type effects
- Mix meshes and sprites to achieve a better 3-D feeling
- Mix additive and translucent blending to work well in both dark and bright areas
Surface-space Blood Effects
Example: Surface-space

- Inside the Riftworm, characters use a blood shader
- One scalar shader parameter (0-1) to indicate amount of blood
- Texture masks for bloody regions and pattern
- Scroll textures downwards in texture-space
  - Gradient pattern & normalmap
  - (Roughly correlates to down in world-space)
- Masks and shader parameter multiply
  - Used as lerp factor between a constant blood color and un-bloodied color
- Blood color have negative Blue and Green components
  - Produces slight discoloration in the transition areas

Fluid Simulation

• Using our Unreal Engine 3 fluid simulation feature
  – Allows the player to wade through large pools of blood

• Ability to raise / lower the entire fluid
  – Controlled by a triggered cinematic

• Simulated on separate thread

• Height-field tessellated on GPU
  – No vertex buffer to update, all vertices are created on-the-fly on GPU
  – The height-field is used as a dynamic vertex texture
  – (Not all platforms)
Example: Riftworm Heart Chamber

• Many different techniques in combination:
  – Gore mesh
  – Screen-space blood splatter
  – World-space blood particle effects
  – Surface-space shader effect on the main character
  – Blood fluid simulation

• Note: Cutting through an artery is not done with gore mesh
  – Skeleton animation
  – Throbbing is morphing geometry
Blood Techniques
Blood

• Movie: Riftworm heart chamber,
chainsawing through arteries
Morphing Blood Geometry

• Morphing:
  – Animate individual vertices, not a skeleton
  – Blend between “morph targets” (or “blend shapes”)

• Usage examples:
  – Throbbing guts inside the Riftworm
  – Some cinematics use morph targets
  – Meatshield, to prevent it from intersecting the holder

• Final example: Exit from the Riftworm
  – Notice blood pouring out (morphed geometry)
In Conclusion

• Achieving the right look for blood and gore:
  – Mix many different techniques
  – Use highly featured and flexible tools
About Daniel Wright

- Joined Epic Games in 2006
- Huge graphics nerd
- Optimized graphics for Xbox 360, PS3 and PC
Topics
Daniel Wright

- Gore techniques
- Blood techniques
- Character lighting
- Screen Space Ambient Occlusion optimizations
Topics
Daniel Wright

- Gore techniques
- Blood techniques
- Character lighting
  - Game lighting
  - Shadows
  - Cinematic lighting
- Screen Space Ambient Occlusion optimizations
Game Character Lighting Goals

- Integrate closely with environment
- Minimal artist effort, should ‘just work’
- Upper bound on shading cost
Background

• Hundreds of static local light sources in Gears of War 2 levels
Background

- Hundreds of static local light sources in Gears of War 2 levels
Background

- Some of these are dominant light sources
Background

- The rest are fill (bounce) lights
Background

• Hundreds of static local light sources in Gears of War 2 levels
• Too many to light the character dynamically with all of them!
Introducing the Light Environment

- A Light Environment consists of:
  - The incident lighting for the object it is responsible for lighting
  - Update and apply logic
Incident Lighting Representation

- Used a Spherical Harmonic basis
- 9 coefficients

Incident lighting stored in a spherical harmonic basis for two lights
Creating the Light Environment

- Dynamically composite relevant lights into the spherical harmonic basis unique for each character
- Light visibility determined with ray traces
- Separate static and dynamic environments
Creating the Light Environment

• Static lights go into the static environment
• For the static environment:
  – Update cost is spread across multiple frames
  – Update rate is determined by visibility, distance to the viewer, and object importance
Creating the Light Environment

Every frame

- Incident lighting from dynamic lights calculated
- Static environment interpolated
- Final Representation

Every static update

- Incident lighting from static lights calculated
Applying the Light Environment

• Directional light for dominant direction
Applying the Light Environment

- Dual-Hemisphere light for remaining environment
- Medium quality
Applying the Light Environment

- Directly evaluate remaining SH environment per-pixel
- Gives highest quality
Applying the Light Environment

Fill light

Dominant light
Applying the Light Environment

- Additional dynamic lights can be applied
  - Useful for quickly changing gameplay lights like muzzle flashes
  - Artist’s discretion
Character Shadowing

- Shadow direction
  - Determined by dominant light direction
  - Fill lights are excluded
Character Shadowing

Dominant light

Dominant light and shadow

Dominant light, shadow, and unshadowed secondary light
Cinematic Character Lighting

- Mix of light environments and multiple shadowed dynamic lights
  - Whatever looks best for each cinematic
In Conclusion

- Combine relevant lights into one efficient representation
Topics
Daniel Wright

- Gore techniques
- Blood techniques
- Character lighting
- Screen Space Ambient Occlusion optimizations
Topics
Daniel Wright

• Gore techniques
• Blood techniques
• Character lighting
• Screen Space Ambient Occlusion optimizations
  – Review
  – Motivation
  – Down sampling
  – Temporal filtering
  – Computation masking
Screen Space Ambient Occlusion Review

- Sample neighbor depths
- Compare depth differences
Screen Space Ambient Occlusion Review

• Noisy output
Screen Space Ambient Occlusion Review

- Spatial filter passes
  - Removes high frequency noise

Edges preserved
Screen Space Ambient Occlusion Review

- PC games have shipped with SSAO as a high end feature
- Gears 2 first console game (that we’re aware of)
  - Many more coming soon!
- Lots of available information on making it look good
- Only talking about optimizing here
Baseline

- 720p, 16 occlusion samples, full res, 20 filter samples

Ambient Occlusion only
Baseline

- 720p, 16 occlusion samples, full res, 20 filter samples
Baseline

- 720p, 16 occlusion samples, full res, 20 filter samples
Screen Space Ambient Occlusion Optimizations

• Motivation
  – Shortage of GPU resources
  – Targeting console hardware
• Trade off quality for performance
Occlusion Pass

- Reduce neighbor samples
  - Multi-frequency occlusion is lost

8 occlusion samples, 12 filter samples
Occlusion Pass

- Reduce neighbor samples
  - Multi-frequency occlusion is lost

16 occlusion samples, 20 filter samples  
8 occlusion samples, 12 filter samples
Occlusion Pass

- Texture cache thrashing
  - Due to random per-pixel offsets
- Clamp offset radius in screen space
Down sampling

- Downsized render target
  - AO is mostly low frequency

Down sampled by a factor of two
Down sampling

- Downsized render target
  - AO is mostly low frequency

Full resolution

Down sampled by a factor of two
Down sampling

- Downsized render target
  - Use furthest of source depths
    - Effectively shrinks silhouettes of nearby objects

Using one source depth

Using furthest of source depths
Down sampling

• Downsized render target
  – Store depth in same render target as occlusion
    • Occlusion and filtering passes can reuse
    • Later passes can get occlusion and depth with one texture lookup
    • Using RG16F
  – Ended up using a down sample factor of two
  – Reduces Xbox 360 resolve cost by a factor of four
Results
Temporal Filter

• Motivation
  – Large spatial filters needed to hide low occlusion samples
  – Detail is lost with large filters
  – Lots of variance between frames
Temporal Filter

• Filter the occlusion factor over time
  – Reverse Reprojection Caching
    • Use as a filter instead of an optimization
Temporal Filter

• Reverse Reprojection Caching

- Lookup this frame and last frame’s occlusion values
- If last frame’s value is invalid, throw it away
- Otherwise
  - New = lerp(Current, Old, Rate)
- Update the history with new occlusion
Temporal Filter

- Reverse Reprojection Caching

1. If last frame’s value is invalid, throw it away.
2. Update the history with new occlusion.
3. Otherwise, New = lerp(Current, Old, Rate).
4. Lookup this frame and last frame’s occlusion values.
Temporal Filter

- Reverse Reprojection Caching

- Lookup this frame and last frame’s occlusion values

- If last frame’s value is invalid, throw it away

- Otherwise
  New = lerp(Current, Old, Rate)

- Update the history with new occlusion
Temporal Filter

• Reverse Reprojection Caching

- Lookup this frame and last frame’s occlusion values
- If last frame’s value is invalid, throw it away
- Otherwise:
  - New = lerp(Current, Old, Rate)
  - Update the history with new occlusion
Temporal Filter

- Dynamic objects

- **Source Position**
  - **Last Frame**
    - Animation
    - World space position
    - Screen space position
  - **This Frame**
    - Animation
    - World space position
    - Screen space position
Temporal Filter

• Static objects
Temporal Filter

- Update history of all pixels as if they were static
- Overwrite with correct value for dynamic objects
  - Depth testing
Temporal Filter

- Newly Unoccluded pixels

New Pixels

Halo regions

Character movement direction
Temporal Filter

- Additional convergence control

New pixels from camera rotation for one frame

Accelerating Convergence over multiple frames

Pixels with an existing history

Constant Convergence Rate
Without Temporal Filter
With Temporal Filter
Temporal Filter

- Doesn’t help with static camera/objects
Temporal Filter

- Almost all flickering gone with moving camera
- Can get away with less occlusion samples, less filter samples
Temporal Filter Gotchas

- Accurate last frame position
- Bilinear filtering
- Double buffer
- World space precision
Temporal Filter Limitations

- Skeletal meshes
- Not detecting cache misses for every sample

Streaks where moving objects meet
Computation Masking

- Pixels with minimal occlusion
Computation Masking

- First person weapon
Computation Masking

- Distant pixels
Computation Masking

- Cull these pixels with Hi-Stencil
  - Requires use of down sampled depth/stencil buffer
  - Special fast case for distance

Computation mask
In Conclusion

- Use Reverse Reprojection Caching as a temporal filter
- Use Computation Masking to avoid doing work where it won’t be noticed
Questions?

- niklas.smedberg@epicgames.com
- daniel.wright@epicgames.com