Visual Effects in Star Citizen

Alistair Brown
Director of Graphics Engineering,
Cloud Imperium Games / Foundry 42
Introduction to Star Citizen

- Ambitious futuristic space-sim
  - First person perspective
  - Space combat, exploration, trading, mining
  - ‘Instanced’ MMO
  - Full single player campaign – Squadron 42
  - Crowd funded project
CryEngine

- Mature FPS and Multi-platform Code Base
- AAA standard technology and effects
- Physically based rendering pipeline
- Deferred/Tiled-Deferred/Forward+
- 10 different modes of Anti-Aliasing
Visual Quality

Video...
Visual Quality

- Extremely high-end visuals
  - Long term focus on quality
  - High system specs
    - Current high-spec PC will be mid-spec by release
  - Currently DX11 only
Ship Complexity

- Extremely high poly ships
  - 60%-40% split between texture and geometry memory rather than more typical 80-20
- Far more assets than can fit into typical GPU memory
- Heavy use of streaming
Ship Complexity

- Initially we allocated more geometry than we had space for on the GPU
- Mostly fine because only a small % on screen at once
  - Different LODs
  - Mutually exclusive assets (damage states)
Ship Complexity

- However memory paging to GPU eventually reared its ugly head
  - Difficult to predict and avoid
  - ‘GPU View’ tool can be useful in tracking this
  - As can GPU hardware vendors' assistance 😊
  - DX12 would help avoid/diagnose these from the application
Ship Complexity

- Ideally predict meshes required in advance
  - Easy for LODs
  - Impossible to predict when damage will occur and therefore require damage meshes
- Need to stay within GPU memory budget as much as possible
  - Avoid rarely-used / mutually-exclusive assets
Ship Complexity
Ship Complexity

- Discrete damage models for each ship part
  - 0%, 25%, 50%, 75%, 100%
- Switch independently
- All need LODs
- 10+ parts, 5 damage states, 4 LODs
- 200+ meshes for one ship!
Ship Damage – Goals

- Identified need to improve system
- Key goals were:
  - Less art intensive
  - Better use of modern hardware & DX11
  - Lower memory usage
  - More accurate and location specific damage
  - Maintain or improve on visual fidelity
Ship Damage
Ship Damage
Ship Damage – Overview

● Key idea was to ditch decals and 25% / 50% / 75% damage states as these are primarily just surface damage
● Store data about any damage on the GPU and feed this into the pixel shader to visualize the damage
● Use DX11 & DirectCompute to enable more complex damage model and improved visuals
● Keep 100% damage state for major silhouette changes
Ship Damage – Overview

- Mesh Data
  - Rasterize
  - Graphics pipeline
  - Rasterize
- Position Map
- Damage Map
  - Scatter
- Impact Data
  - DirectCompute
- Update
- Screen
Ship Damage – Damage Model

- We chose to model some more physical properties that just ‘damage’ to achieve more complex and dynamic results
  - Deformation
  - Thickness
  - Temperature
  - Burn
Ship Damage – Damage Model

- Burns
- Dent
- Paint
- Metal
- Burnt Paint
- Burnt Metal
- Hole
- Exposed Metal
Ship Damage – Damage Model

- Need to decide which space to store this data
  - Mostly just surface damage
  - Needs consistent resolution
- Opted for 2D textures as opposed to 3D textures or vertices
- Considered more complex structures such as octree’s but overhead was considered too high
Ship Damage - Damage Model

- Need geometric representation of ship on GPU in order to paint into this Damage Map
- Solution is to use object space texture
- However ship parts animate
  - Store bone ID in alpha channel
Ship Damage – Damage Model
Ship Damage – Adding Damage

- Arbitrary artist-defined UV layout
- Read of position map and skinning limit performance
- However most impacts only effect <5% of damage map
- ~95% of the GPU work required to determine this
Ship Damage – Adding Damage

- Potentially multiple hits on multiple ships in a single frame
  - Especially for larger ships
  - In heavy combat could be a major performance hit
- Need to find a way to direct GPU work to the desired parts of the damage map
Ship Damage – Adding Damage

- Texture is 99% spatially coherent
- Logical to split into smaller tiles to avoid work
  - Calculate bounds of each tile
  - Use like Hi-Z buffer to early out
- However texels aren’t static but are skinned!
  - Moving flaps / wings / turrets
Ship Damage – Adding Damage

- Hard to avoid wasted work with pixel shader
  - Every pixel still needs to read some memory, do some calculations then bail out – usually bandwidth limited
- Compute shaders open up many different ways to optimize
  - Thread group shared memory
  - DispatchIndirect
Ship Damage – Adding Damage

- Calculate N bounding spheres per tile
  - N isn’t fixed, but in practice has an upper limit of about 4
- Each thread tests one bounding sphere against the impact location
- Only shade pixels if at least one passes
- Distributes the cost – *much* quicker rejection
- But most threads still idle during this stage
Ship Damage – Adding Damage

- Most impacts last multiple frames, and larger ships will likely receive many hits at once
- Use idle threads to calculate multiple hits
- $\text{threadGroupSize} = \text{maxBonesPerTile} \times \text{maxHits}$
- More hits would require a loop per thread
Ship Damage – Adding Damage

● Alternate approach is to perform one tile-bone-impact calculation per thread and store results in a buffer and accumulate number of tiles visible
● DispatchIndirect can be used to trigger a 2\textsuperscript{nd} compute on just the required tiles
● Theoretically less wastage, but overhead of intermediate buffer and 2\textsuperscript{nd} dispatch are significant
Ship Damage – Extras

- Parallax occlusion mapping for internals
  - Perfect use case as silhouette is hidden and poly count needs to be low for memory & performance

- Screen space height-map
  - Use the differential to calculate surface gradient and perturb normal [Mikkelson 2010]
  - ddx/ddy on bilinear filtering looks bad under magnification
  - Take 2 extra samples and manually calculate forward difference
Ship Damage – Extras

- Heat dissipation in compute shader
- Hole cutting
  - clip()
  - Needs including in depth/shadow pass when close-up 😞
- Investigating the possibility of identifying bones that have holes in the compute shader and using DispatchIndirect to limit the number of polys using clip()
no damage
+burn albedo
+ dent normals
+ burn normals
+ hole cutting
+ cut height/normals
Ship Damage – Results

Video...
Ship Damage – Extensions

● But we’re just scratching the surface!
  ● Pun intended 😊

● Geometry Morphing
  ● Create second version of ship which panels shrunken inwards
  ● Export offsets in compacted 32bit RGBE format
  ● Push vertices towards offset as they are dented
  ● Could potentially use tessellation for more accuracy
Ship Damage – Extensions

● GPU particles
  ● Spawn in compute shader when thickness is modified
  ● Use position map for location/orientation
  ● Use thickness, temperature & diffuse map for color
  ● Complete GPU solution for VFX 😊
  ● But need to efficiently manage variable particle count
Ship Damage – Extensions

● Many GPU particle implementations don’t handle arbitrary particle counts
● Can use append/consume buffer as free-list for spare slots in a fixed sized array – but hard avoid cost of empty slots in particle update & render
● Instead make the upper bound of the number of particles that can be spawned from each hit deterministic
● Treat particle array as a ring-buffer
Ship Damage – Extensions

- Keep track of start/end points
- Dispatch updates just for the particles we need
- Skip unused particles in compute shader
- Different ring buffers for different particle life-times
Ship Damage – Extensions

● Sparse / Tiled memory
  ● Most tiles are empty most of the time
    ● Especially on larger ships
  ● Ideally allocate on demand
  ● Tiled resource in DX11.2
  ● Our current min-spec is DX11.1 😞
  ● Revisit later in development
Scale Issues
Scale Issues

- Enormous scale
  - No ‘faked’ UI or FPS arms/body
  - Everything is ‘in the world’
  - UI is ~3cm from camera
  - Ships up to a mile long with ~100 rooms
  - Planets are hundreds/thousands of miles wide
  - Inter-planetary travel
Scale Issues

- Quickly reached 32bit precision issues
- Upgraded CryEngine to 64bit transforms
- Renderer stays 32bit camera-relative
- Depth buffer changed to inverted 32f
  - No performance hit on modern hardware
Scale Issues

Depth buffer precision comparison, range 0.01 .. 10,000,000 (9 decades)

Brano Kemen - http://outerra.blogspot.co.uk/2012/11/maximizing-depth-buffer-range-and.html
Environments

- Due to the MMO part of the game we require a LOT of environments and so opted for a modular approach
- Modular ‘kits’ are built that easily snap together
- Simplifies art pipeline (e.g. out-sourcing)
- Very flexible for level designers
Environments

- We immediately hit many performance issues
  - Poly count due to desired fidelity
  - Texel density too high for baking textures
  - Tiling textures means many draw calls per mesh
  - Lots of meshes to build a room
  - Even more meshes for a space-station!
  - LOTS of draw calls
Environments

- Texture arrays are a potential solution
  - Resolution limitation means streaming is difficult
  - Instead use low-resolution texture arrays just for LODs
  - No need to stream individual textures – entire texture array at 256x256 for a level is < 15Mb

- Can now render LODs with a single draw call! 😊
  - Vertex buffers sorted by material ID so can still use high-res textures if required
Environments

- Mesh merging solution similar to KillZone
- Build LODs for each individual modular asset
- Iterative heuristic algorithm to combine LODs to build a hierarchy with min draw calls and memory
- Relies on aggressive LODs but can drastically reduce draw calls with no manual artist work
Future

- That’s just a tiny sub-set of what we’re doing, there’s lots more to come...
  - Gas-clouds
  - Asteroid fields
  - Stars
  - Planets
  - Worm-holes
Thanks

• Thanks for listening!

• Special thanks to our awesome team at CIG, especially these lot:

  Nicolas Thibieroz, Chris Roberts, Erin Roberts, Neil McKnight, Bjorn Seinstra, Nathan Dearsley, Okka Kyaw, Geoff Birch, Muhammad Ahmad, Matt Intrieri, Mark Abent
Obligatory “We’re Hiring” Slide

https://cloudimperiumgames.com/jobs
Manchester (UK) + Frankfurt + Santa Monica
Questions?

alistair.brown@cloudimperiumgames.com
References

- Rendering Wounds in Left 4 Dead 2

- Making of Killzone 3
  - https://www.youtube.com/watch?v=QfvBIHFex9Y

- Maximizing Depth Buffer Range – Brano Kemen
  - http://outerra.blogspot.co.uk/2012/11/maximizing-depth-buffer-range-and.html

- Bump Mapping Unparametrized Surfaces on the GPU - Morten S. Mikkelsen
Unofficial Trailer

Fan Trailer...

Watch on YouTube