FORZA MOTORSPORT

Streaming Massive Environments

From Zero to 200MPH

Chris Tector (Software Architect Turn 10 Studios)

Turn 10

- Internal studio at Microsoft Game Studios - we make Forza Motorsport
- Around 70 full time staff

Why am I here?

- Our goals at Turn 10
- The massive model visualization hierarchy
- Our pipeline, from preprocessing to the runtime

Why are you here?

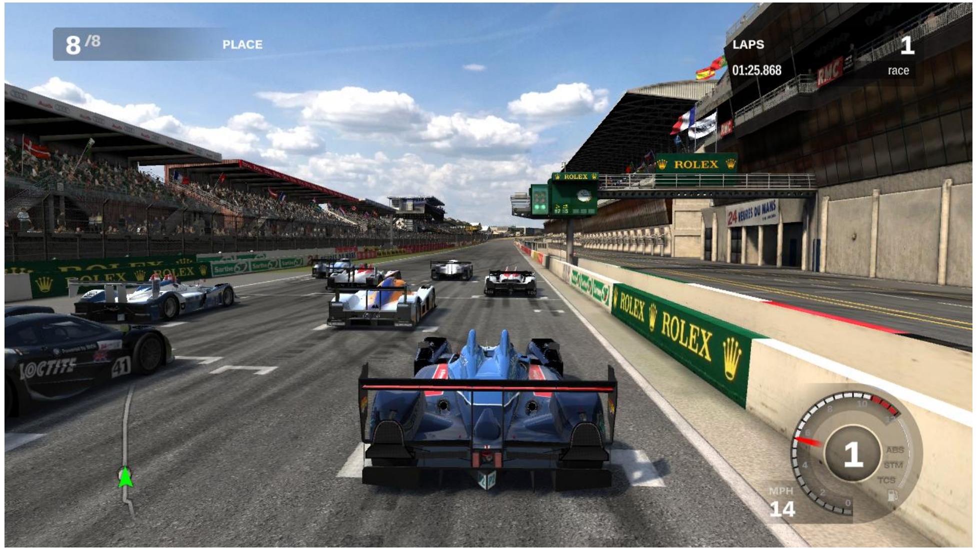
- **Learn about streaming**
  - Typical features in a system capable of streaming massive environments
  - Understand the importance of optimization in processing streaming content
  - Practical takeaways for your game
- **Primarily presented as a general system**
  - But there are some 360 specific features which are pointed out as they are encountered

At Turn 10

GOALS

Streaming

- Rendering at 60fps
- Track, 8 cars and UI
- Post processing, reflections, shadows
- Particles, skids, crowds
- Split-screen, replays

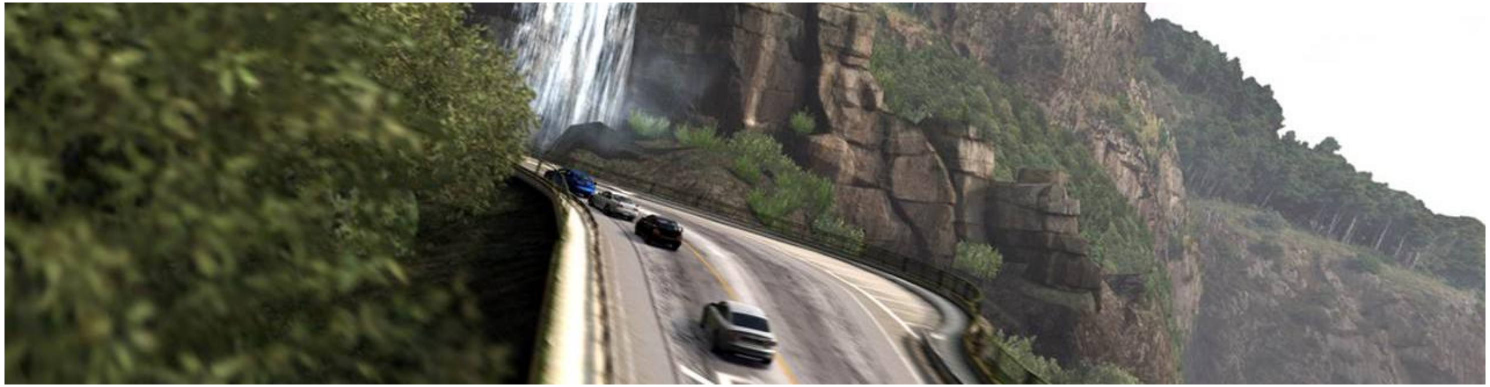


Massive Environments

- Over 100 tracks, some up to 13 miles long
- Over 47000 models and over 60000 textures

Zero

- Looks great when standing still
- All detail in there when in game or photo mode
- Especially the track since it is the majority of the screen



200

- Looks great at high speeds
- All detail is there when in game or replay mode, UGC video
- Again, especially the track

Running Example

- Le Mans is an 8.4 mile long track
- It has roughly 6000 models and 3000 textures
- As this talk goes on we can track how much data is streamed

  - Data streamed :
    - 13.3 Miles driven
    - 1.6 Laps
    - 0.98 GB Loaded
      - 0.14 GB Mesh
      - 0.84 GB Texture

Factors to Optimize for

- **Minimize**
  - Size on disk (especially when shipping large amounts of content)
  - Size in memory
- **Maximize**
  - Disk to memory rate
  - Memory to processor rate
- **All while maximizing quality**

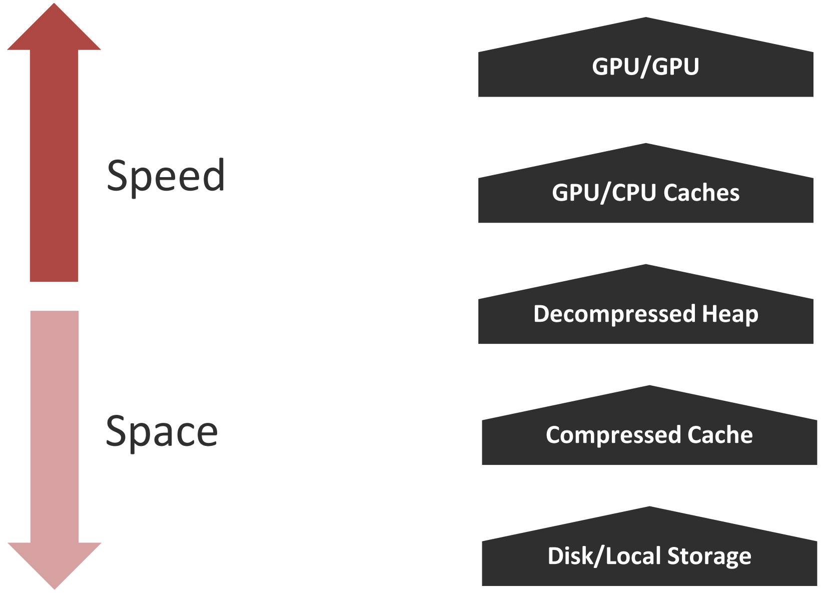
The Hierarchy

MASSIVE MODEL VISUALIZATION

Massive Model Visualization in Research

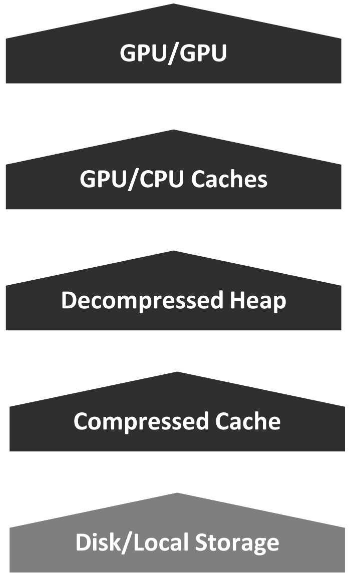
- Most relevant area to search
- Good course notes from Siggraph 2007
    - http://www.siggraph.org/s2007/attendees/courses/4.html
- But a lot of "real time" options in the literature aren't game real time

Typical massive model visualization hierarchy



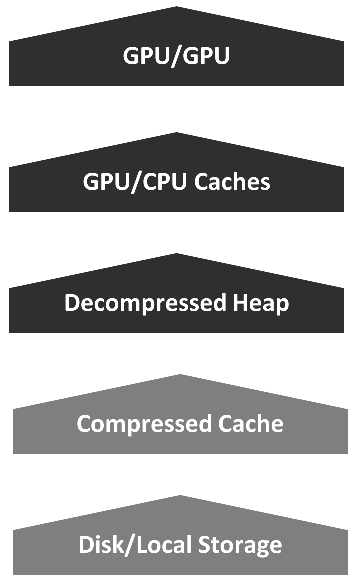
Disk

- **Stored on zip disk in packages**
  - We store some extra data in zip format, but honor base format so standard browsing tools all still work (explorer, WinZip, etc.)
  - Stored in LZX format inside the archive
- **90-300MB per track**



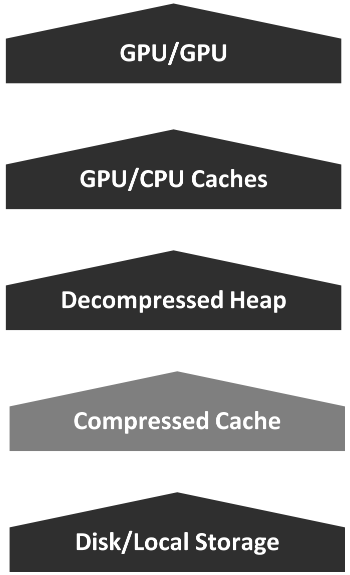
Disk to Compressed Cache

- **Fast IO in cache block sizes**
  - Block is a group of files within the zip
  - Total up size of files until block size is reached
  - Retrieve that file group with a single read
- **Compressed cache reduces seeks**
  - 15MB/s peak
  - 10MB/s average
  - But 100ms seeks



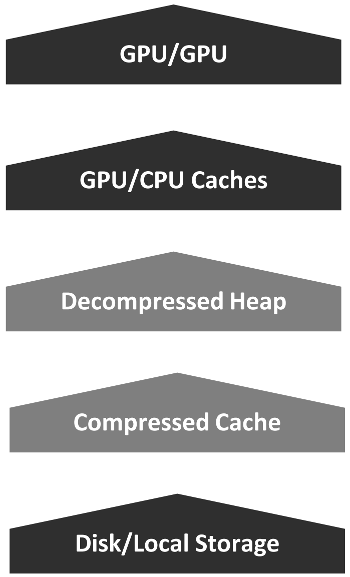
Compressed Cache

- LZX format in-memory storage
- Cache blocks streamed in on demand and out LRU
- 56 MB
- Block sizes tuned per track, but typically 1 MB



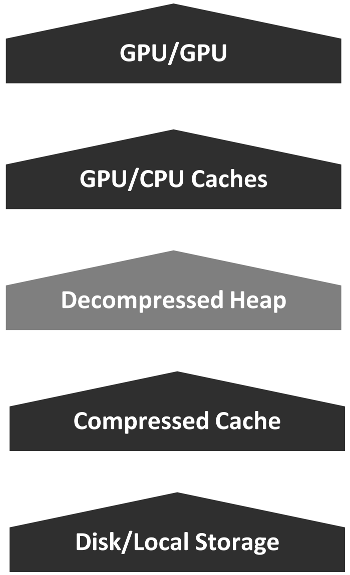
Compressed Cache to Heaps

- **Fast platform specific decompression**
  - 20 MB/s average
- **Heap implementation**
  - Optimized for speed of alloc and free operations
  - Good fragmentation characteristics using address ordered first-fit



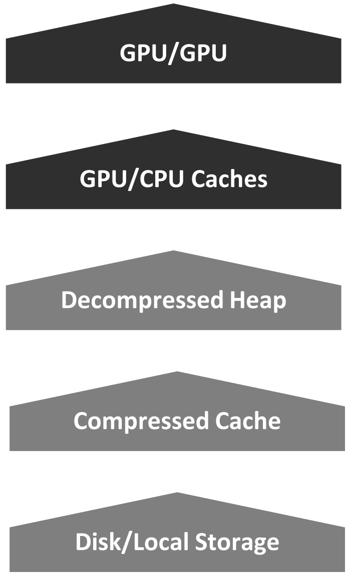
Decompressed Heap

- Ready for GPU or CPU to consume
- Contiguous and aligned per allocation
- 194MB



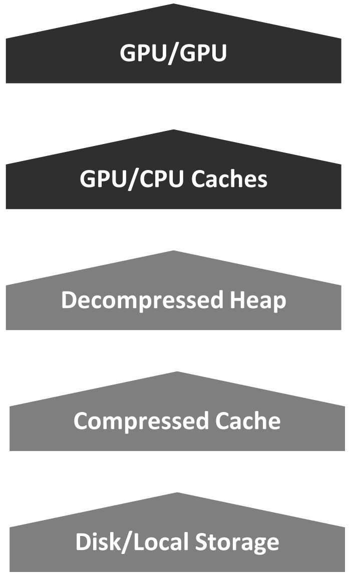
Multiple Levels of Texture Storage

- **Three views of each texture**
  - Top Mip: Mip 0, the full resolution texture
  - Mip–Chain: Mip 1 down to 1x1
  - Small Texture: 32x32 down to 1x1
- **Platform specific support here to not require relocating textures as top mip is streamed in**



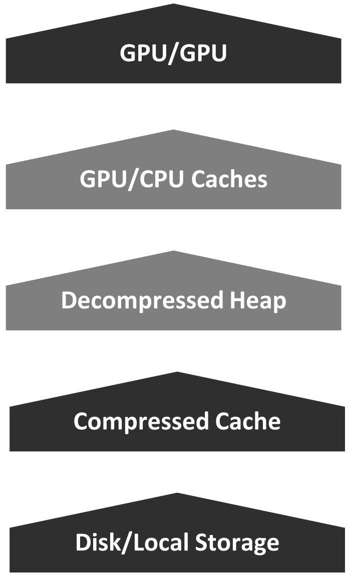
Multiple Levels of Geometry Storage

- **LOD**
  - We consider different LODs as different objects to allow streaming to dump higher LODs when they wouldn't contribute
- **Instances**
  - Models are instanced with per instance transform and shader data



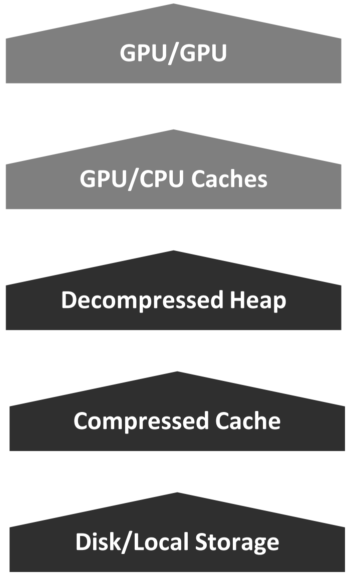
Memory to GPU/CPU Cache

- **CPU specific optimizations for cache friendly rendering**
  - High frequency operations have flat, cache line sized structures
  - L1/L2 Caches for CPU
- **Heavy use of command buffers to avoid touching unnecessary render data**



GPU/CPU Caches

- **Right sizing of formats relative to shader needs**
- **Vertex/texture fetch caches for GPU**
  - Vertex formats, stream counts
  - Texture formats, sizes, mip usage
- **Use of platform specific render controls to reduce mip access, etc.**



Running Example

- **Data streamed :**
  - 66.8 Miles driven
  - 7.9 Laps
  - 4.9 GB Loaded
    - 0.7 GB Mesh
    - 4.2 GB Texture

The Pipeline

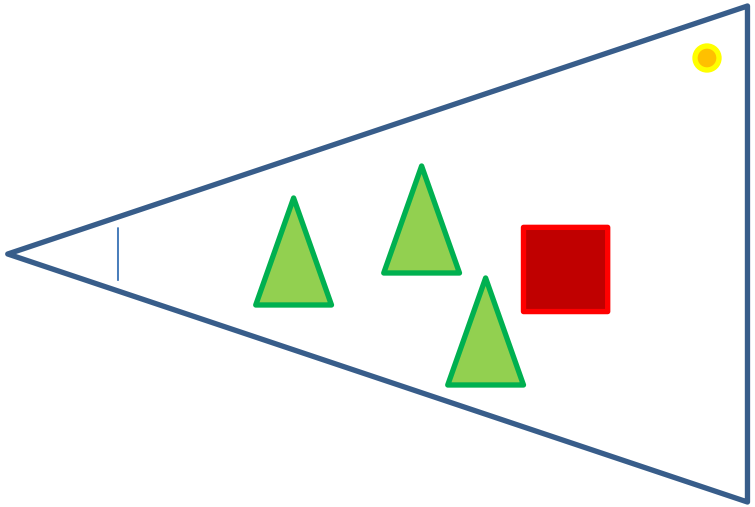
BREAK IT DOWN

Pre-Computed Visibility

- **Standard Solution**
  - Given a scene what is actually visible at a given location
  - Many implementations use conservative occlusion
- **Our Variant Includes**
  - Occlusion (depth buffer rejection)
  - LOD selection
  - Contribution Rejection (Don’t draw model if less than n pixels)

Culling – Given this View

- Occlusion culled (square)
    - Other objects block this in the view
- Contribution culled (circle)
    - This object does not contribute enough to the view



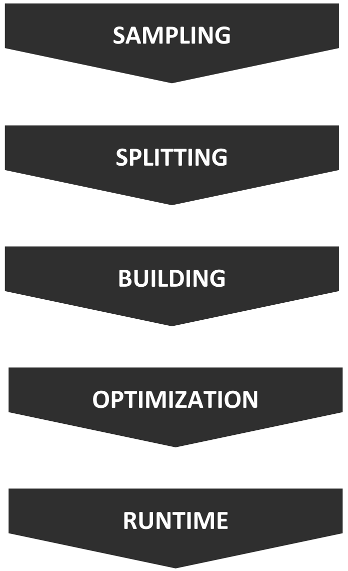
Could do it at Runtime

- **LOD and contribution are easy, occlusion can be implemented**
- **Most importantly would have to optimize in runtime**
  - Or not do it at all, but that means streaming and rendering too much
- **Visibility information is typically a large amount of data**
  - Which means touching a large amount of data
  - Which is bad for cache performance
- **Our solution: don’t spend CPU/GPU on an essentially offline process**

Pipeline

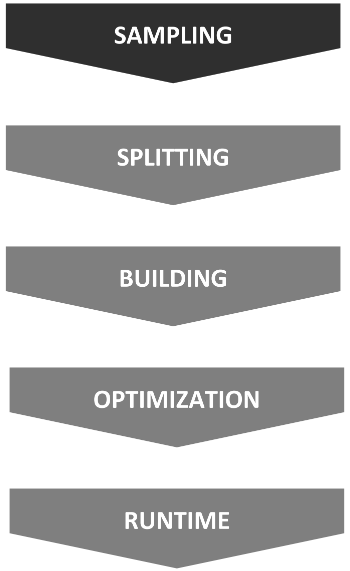
- **Our track processing pipeline is broken into 5 major steps**
  - Sampling
  - Splitting
  - Building
  - Optimization
  - Runtime

- **All of this is fully automated**
  - Art checks in source scenes
  - Pipeline produces optimized game ready tracks



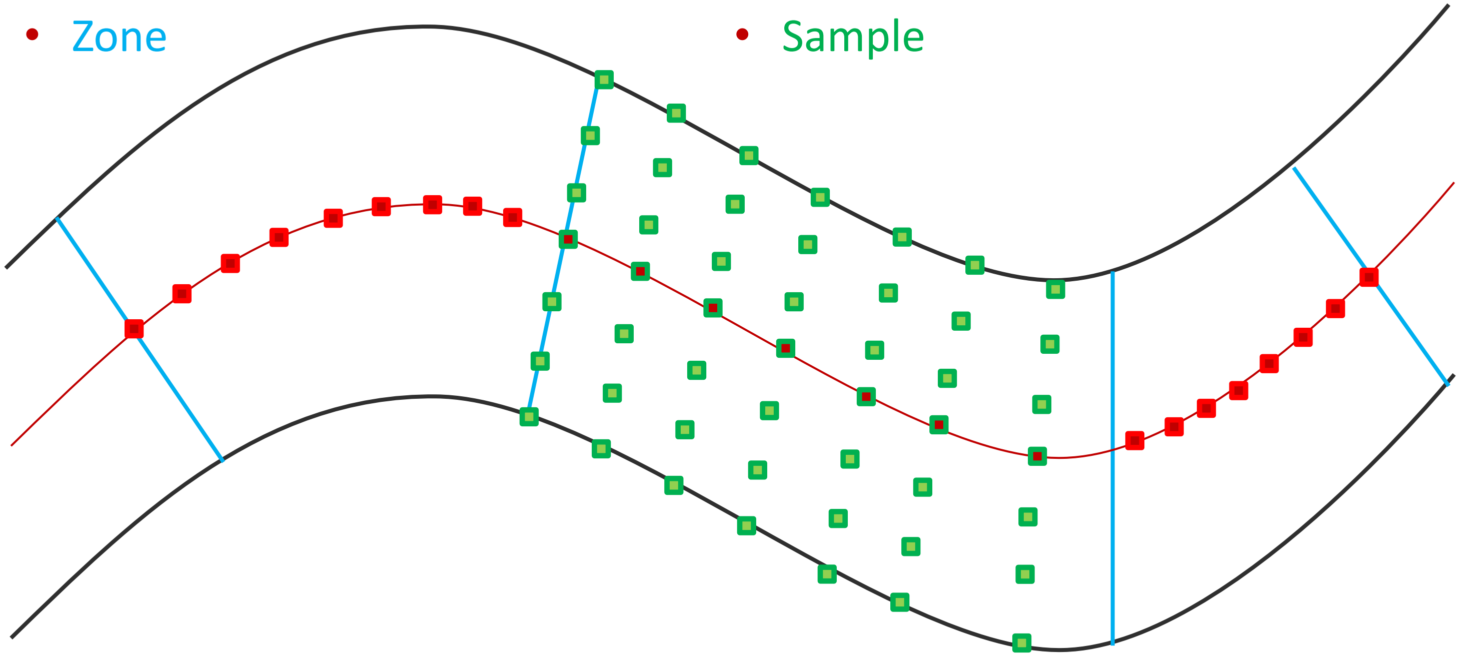
Linearize the Space

- **Track is broken up into zones using AI linear view of track**
  - Art generates inner and outer splines for track
  - Tools fit a central spline and normalize the space
  - Waypoints are generated at regular intervals along the central spline
  - Zone boundaries are set every n waypoints
  - Runtime Sample points are evenly distributed within the zones



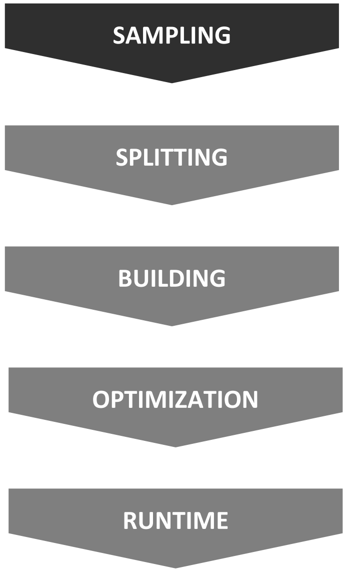
Track Space

- Track
- Zone
- Waypoint
- Sample



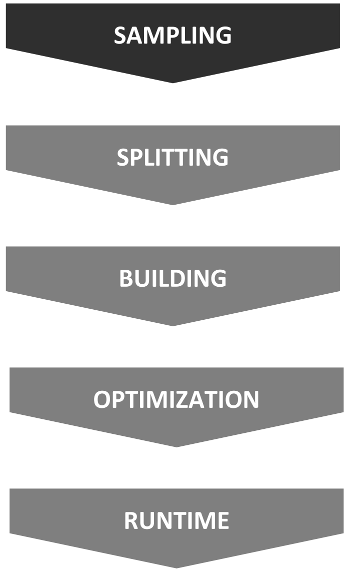
How do we Sample

- **Environment is sampled along track surface only and at a limited height**
- **Track is rendered from four views at each sample point**
  - Oriented to local track space
- **Sampled values stored at each sample point**
  - Also stored at neighboring sample points
  - This is to reduce visibility pops when moving between samples



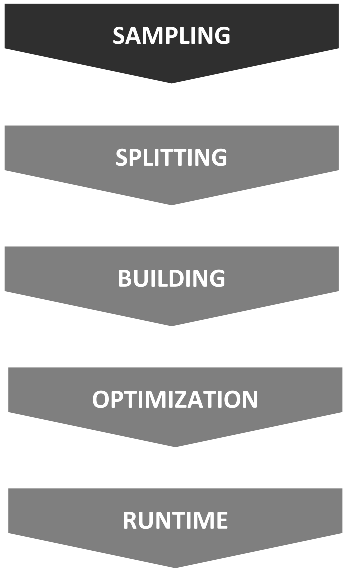
Sampling

- **Render all models to depth**
  - Run using position only mesh version of each model on entire track
- **Render each individual model inside a D3D occlusion query and store**
  - Object ID
  - Location of the camera during rendering
  - Pixel count
- **This includes LOD, occlusion and contribution culling**



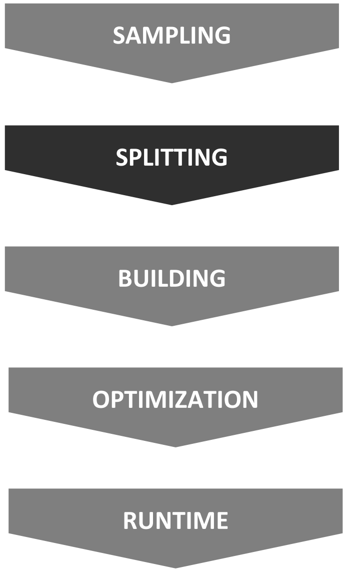
Size Reduction

- **Sample data is enormous**
  - Contains visibility of every model at every sample point
- **Combine all samples to reduce data required for further processing**
- **We condense it down to a list of visibility of models for each zone**
- **Keep track of the per model maximum pixel counts, not just binary visibility**
  - The pixel counts are the real value!
- **Most data is used during pre-processing and then thrown out or drastically reduced for the runtime**



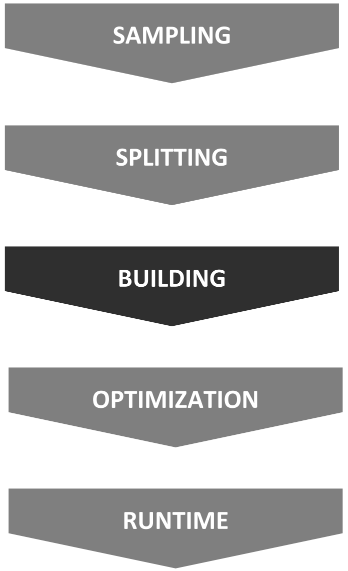
Splitting

- **Breaks large artist meshes down to object level**
  - Example: an entire corner can be modeled and instanced into the track
  - Break model down against a world grid
- **Clusters objects seen together into single models**
- **This stage represents workflow balance:**
  - The further we move towards procedural data providing greater opportunities for instancing, the less this step does since we don't split instanced objects
  - But it provides workflow savings when modeling large amounts unique geometry



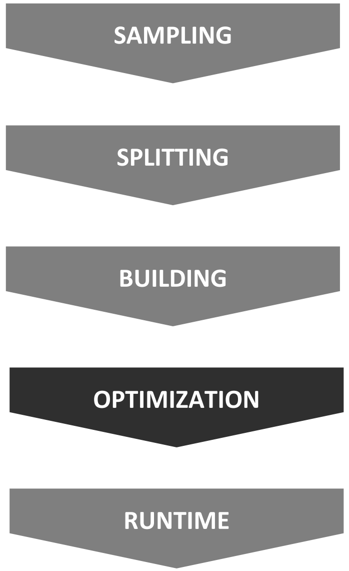
Building

- **Geometry**
  - Collect common geometry per model to reduce draws
  - Create texture and shader usage
- **Textures**
  - Removal of duplicates at multiple levels
    - Similar source
    - Cross texture comparisons
    - Cross mip comparisons
    - Compression settings
- **Small Texture Set**
  - Holds the small texture for all textures used in the environment (mip chain from 32x32 down to 1x1)
  - Only example of preloaded models or textures
  - 20-60MB

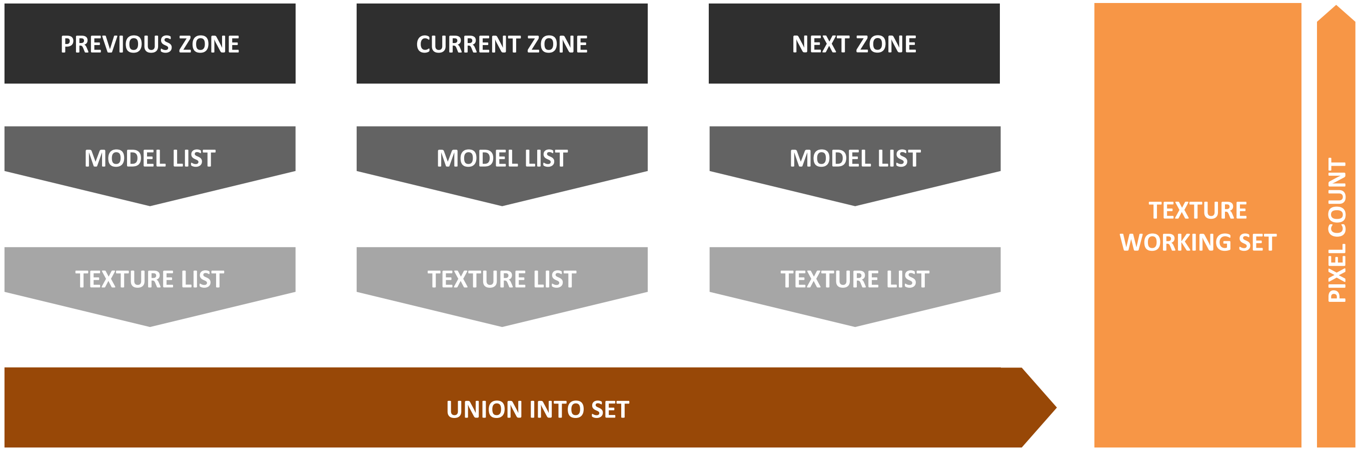


Optimization

- **Accumulate the working set**
  - For the three zones centered on the camera
  - Accumulate the list of models that are visible
  - Based on the set of visible models, generate a visible texture set using the per model texture usage data from the build phase
  - Order the texture list by maximum pixel count of all models which use a particular texture



Working Set



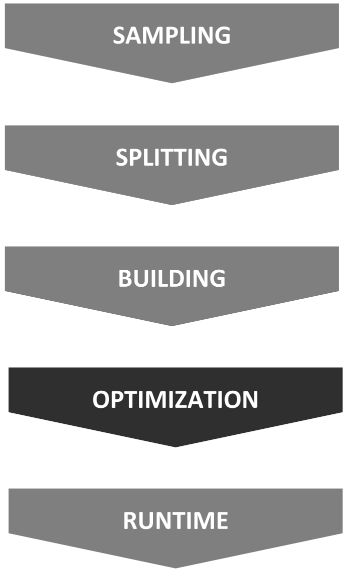
- **Texture working set holds textures in pixel count order using maximum pixel count from all models which use a texture**

Optimization Mechanism

- **Removal of textures only**
  - Geometry removed by sampling

- **Remove textures at two levels**
  - Drop top mip – means the texture rendered will only come from mip 1 and lower
  - Drop mip chain – means the texture rendered will only come from the small texture set

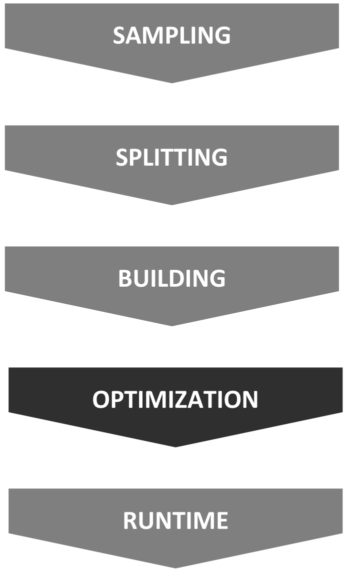
- **Texture level is removed from texture lists in zones contributing to the working set**



Optimization Criteria

Multiple Reduction Passes

- **Trivial Reduction Based on mip Size**
  - Object pixel count vs. total pixels in the small texture
    - I.e. Is object pixel count < 32*32?
- **Total Working Set Memory Size**
  - Sum of model and texture sizes vs. decompressed heap size
  - Remove top mips or mip-chains in increasing pixel count order
- **Total Streaming Bandwidth**
  - Compute the difference of the working set of zone n and the working set of zone n+1
  - Sum of model and texture sizes in working set delta vs. streaming bandwidth (Assuming zone physical size and maximum racing speed you can calculate the time allowed to stream the set)



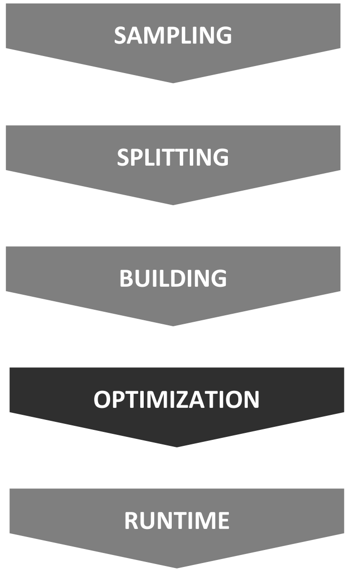
Running Example

- **A single zone transition can vary widely due to occlusion behavior**
  - 0.33/2.22 MB Mesh avg/max
  - 1.87/17.9 MB Texture avg/max

- **Data streamed :**
  - 120 Miles driven
  - 14 Laps
  - 8.8 GB Loaded
    - 1.3 GB Mesh
    - 7.5 GB Texture

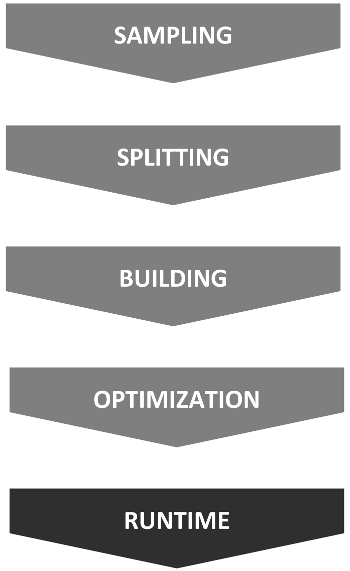
Optimization

- **Create a cache efficient order for the package**
  - To reduce seek distance and increase cache hit rate
  - We use a “first seen” metric
  - Walk over the zones and track which zone is the first to use a model or texture
  - Group all models together and order by first zone, same with textures



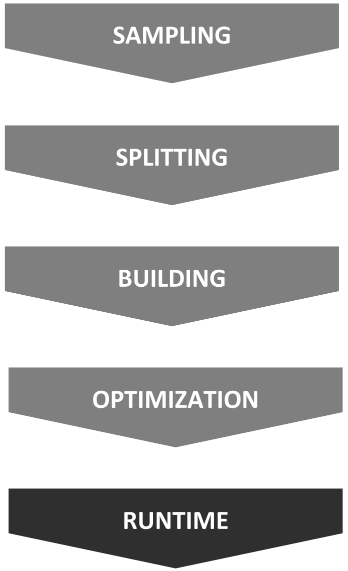
Runtime

- **Create delta of zones**
  - Decide where camera is in visibility space
  - Map camera position to zones to load
  - Difference of currently loaded zones and zones to load
- **Create delta of resources based on zone deltas**
  - Basically reference counting
  - Consolidate the work to ensure free first ordering (this is to help with fragmentation)
- **Stream out (free) data in trailing zones**
- **Stream in (allocation, IO and decompress) data in leading zones**



Runtime Considerations

- **Key Areas**
  - Work ordering
  - Heap efficiency
  - Decompression efficiency
  - Disk efficiency
- **For many problems any solution is better than doing nothing**
  - Make sure all levels of the hierarchy have been addressed



Flythrough Demo

Errors

- **Popping**
  - Limited to two Classes
    - **Late Arrival** (Tuned by limiting the amount needed per zone to stay within the system throughput)
    - **Visibility Errors** (Tuned by further clustering objects or biasing the sampling results)
  - These tunings conflict though
  - We provide manual overrides
    - **Geometry Bias** (Affects sampling results)
    - **Texture Bias** (Affects position in texture working sets during optimization)
- **No amount of automation can compete with unrealistic expectations**
  - Example: all models are visible in a single zone means there won't be space for any textures

Future Directions

- **Non-linear streaming**
  - Integration in sampling, optimization and runtime
- **Domain specific decompression**
  - Procedural generation
  - Texture transcoding
- **Streaming over the wire**
  - Missing piece of the massive model visualization hierarchy

Finally

- **Data streamed:**
  - 147 Miles driven
  - 17.5 Laps
  - 10.8 GB Loaded
    - 1.5 GB Mesh
    - 9.3 GB Texture

Questions?

FORZA MOTORSPORT

Streaming Massive Environments

From Zero to 200MPH

Chris Tector (Software Architect Turn 10 Studios)