Game Developers Conference® | March 9-13, 2010 | Moscone Center | San Francisco, CA

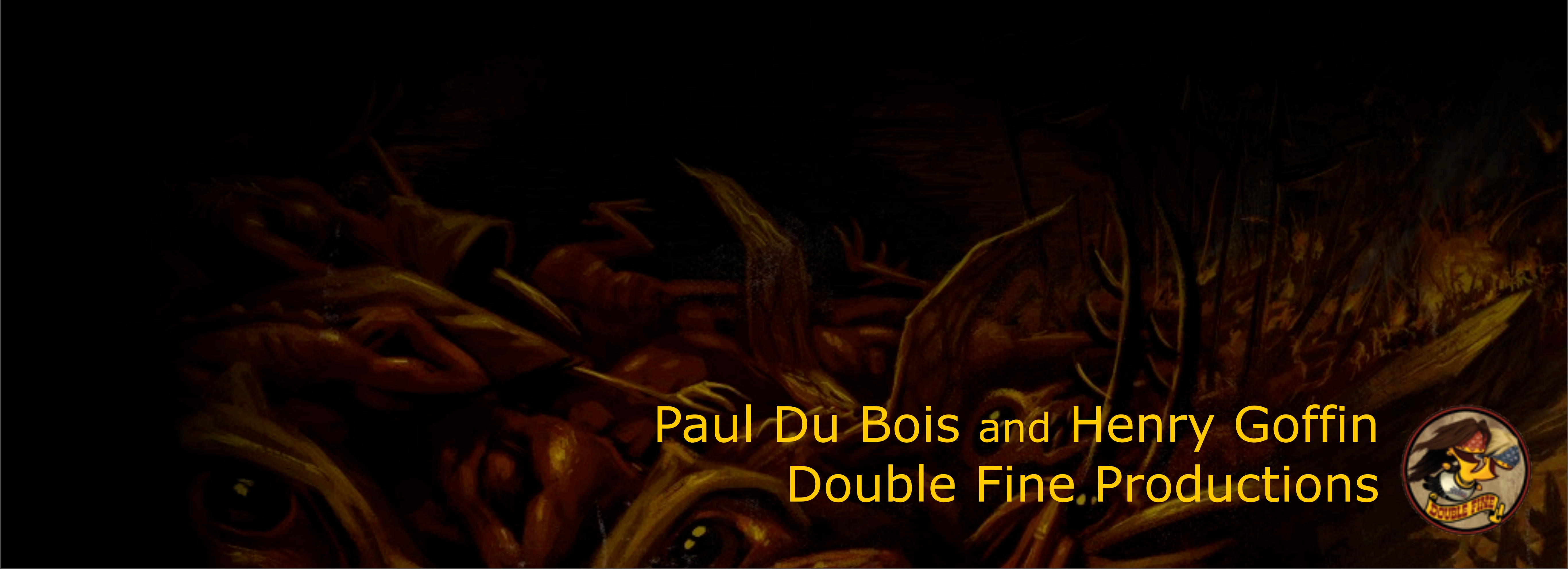
THINK SERVICES
A DIVISION OF UNITED BUSINESS MEDIA LLC

GDC 10

Learn. Network. Inspire.

www.GDConf.com

Data is a four-letter word



Paul Du Bois and Henry Goffin
Double Fine Productions

Background

- In 2005, what does Double Fine need?
  - Massively scalable next-gen content pipeline
  - Streaming open-world engine from scratch
  - Low-maintenance, low-impact solutions
- Not a lot of public information available
  - Data management is not sexy

Talk Overview

- Brief glance over our generic asset pipeline
- Runtime techniques for asset management
- Scheduling I/O requests for optical media
- Key concept: static asset reference graph

About Us

- Henry Goffin – the naïve young upstart
- Paul Du Bois – the battle-scarred veteran

About Brütal Legend

- Started from scratch in 2005
- Shipped on October 13, 2009
- Blend of action, strategy, and open-world gameplay elements with a heavy-metal theme

DOUBLE FINE

The Basics

One Hundred Thousand Assets

AIDifficulties
AchievementRequirements
AlbumCover
AmbMeshDefinition
AmbTileData
AnimMap
AnimResource
ArtBrowserAssets
AttachmentPointTable
AudioEnvironment
AudioProgrammerReport
AudioWavbankMarkers
AudioWavebankData
Blob
BuddhaGlobalAssets
Buff
BuffEffectTable
BuffEffectsData
CameraPath
CameraSettings
Climate
CollisionShape
CombatManeuver
ComboAnim
ComboPose
ControllerConfig
Cutscene
CutsceneClump
DUIMovie
DamageResponseTable
DialogReactionSets
DialogSets
DifficultySet
Effect
EffectTable
EncounterTable
FlashConfig
FurData
GameMapRegions
GameUnlocks
GibData
HUDSkin
Heightfield
InputAliases
InputTextures
InstanceVertexData
JournalEntries
LevelData
LevelList
Material
Mesh
MeshMunger
MeshSet
MissionData
MusicNameTable
MusicSet
NavMeshData
NavigationSystemGraph
ObjectData
OceanData
Outfit
ParticleSystemData
PathTileData
PhysicalSurfaceMap
PhysicsRigidBody
PlaylistResource
PrototypeResource
QuadTileData
Ragdoll
ResourceBuildStamp
RichPresenceInfo
Rig
RigidBodyEventData
RndTileData
RockSolo
SaveData
SimulationData
SoloSetup
Stance
StatLimits
Story
StrategicResponses
StringTable
TechTree
TerrainMaterial
Texture
TileData
TutorialCardSet
UnitInfos
UpgradeCategory
UpgradeSet
VehicleKeyframeData
VidSubtitles
VisualTypeDefinitions
VoiceSettings
WangTileset
WaterEffectTable
Weather
Ycombinator
Zymurgy

General Categories

- Audio and video
- Meshes and textures
- “Trivial” assets
- Everything else

Audio and Video

- Largely outside the scope of our engine
- Re-route file I/O through our own systems
- *We'll get back to this later...*

Meshes and Textures

- Managed exclusively by the rendering system
- *We'll talk about this later, too.*

Trivial Assets

- Simple data structs, mostly under 200 bytes
- Not really worth thinking about
- Loaded on startup and always resident
- Final game contains 10,000+ trivial assets
  - Still trivial “enough”

Everything Else

- Animations, collision data, navigation info, etc.
- Simplest possible implementation
  - Load only on explicit request
  - Reference count, unload on zero
- Game will stall if asset is not already loaded

That Sounds Horrible

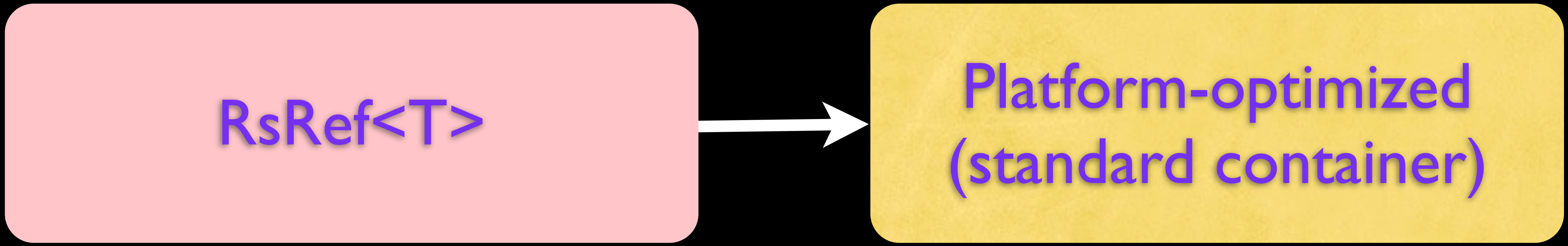
- Don’t want to lay burden on game code
  - Manual refcounting and preloading sucks
- Don’t want to deal with individual assets
  - What does it mean to preload an asset?
  - Examining unloaded assets to find dependencies is impossible
- What do we want?

Asset pipe

Assets

- Type-agnostic asset system
- Single system for creating optimized assets
- One file per asset during development

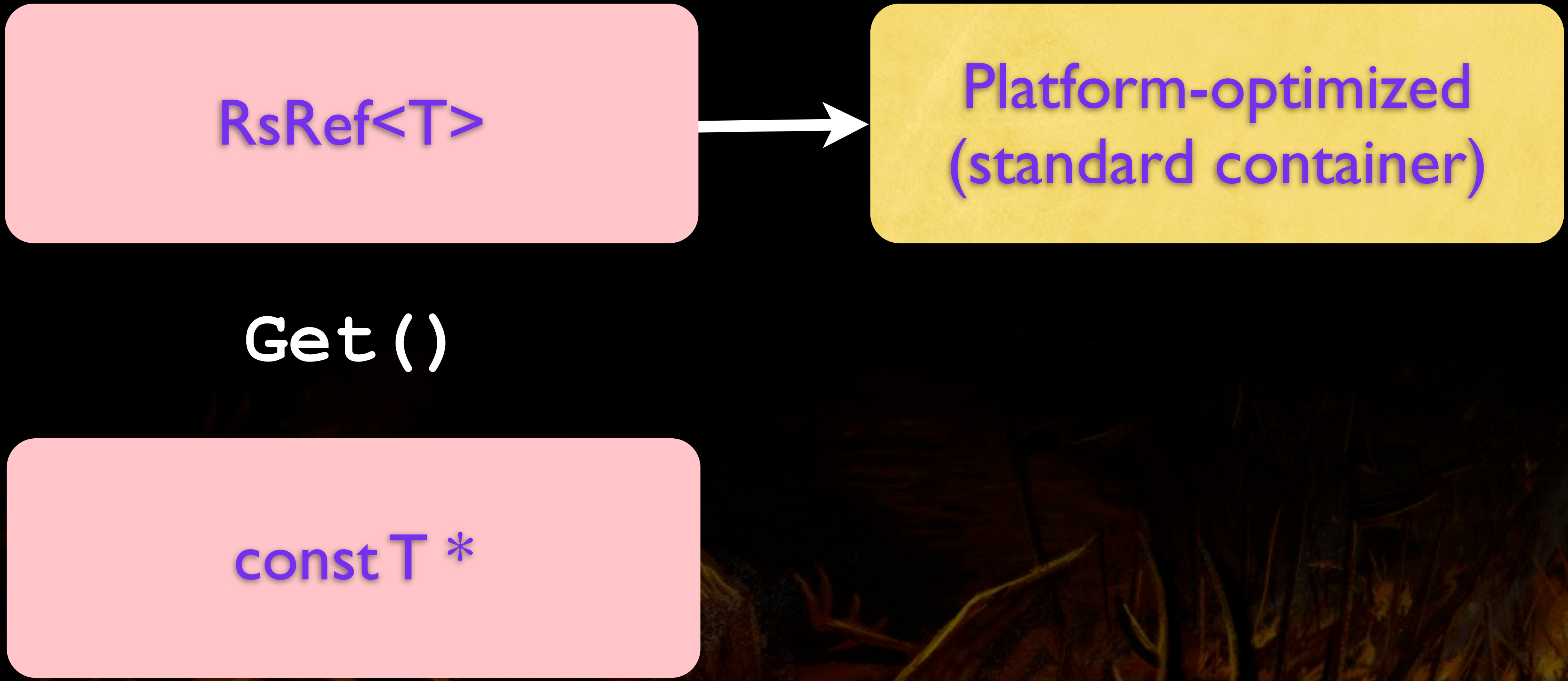
Assets



RsRef<T>

Platform-optimized (standard container)

Assets



RsRef<T>

Platform-optimized (standard container)

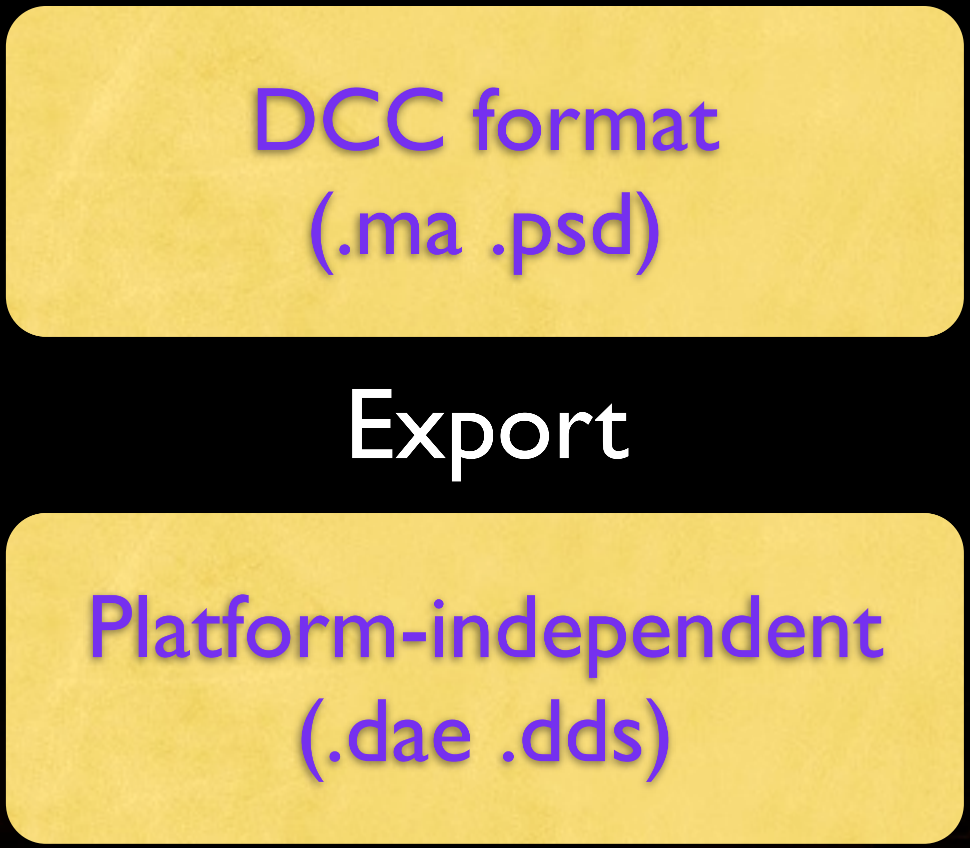
`Get()`

const T *

Asset Pipe

DCC format
(.ma .psd)

Asset Pipe

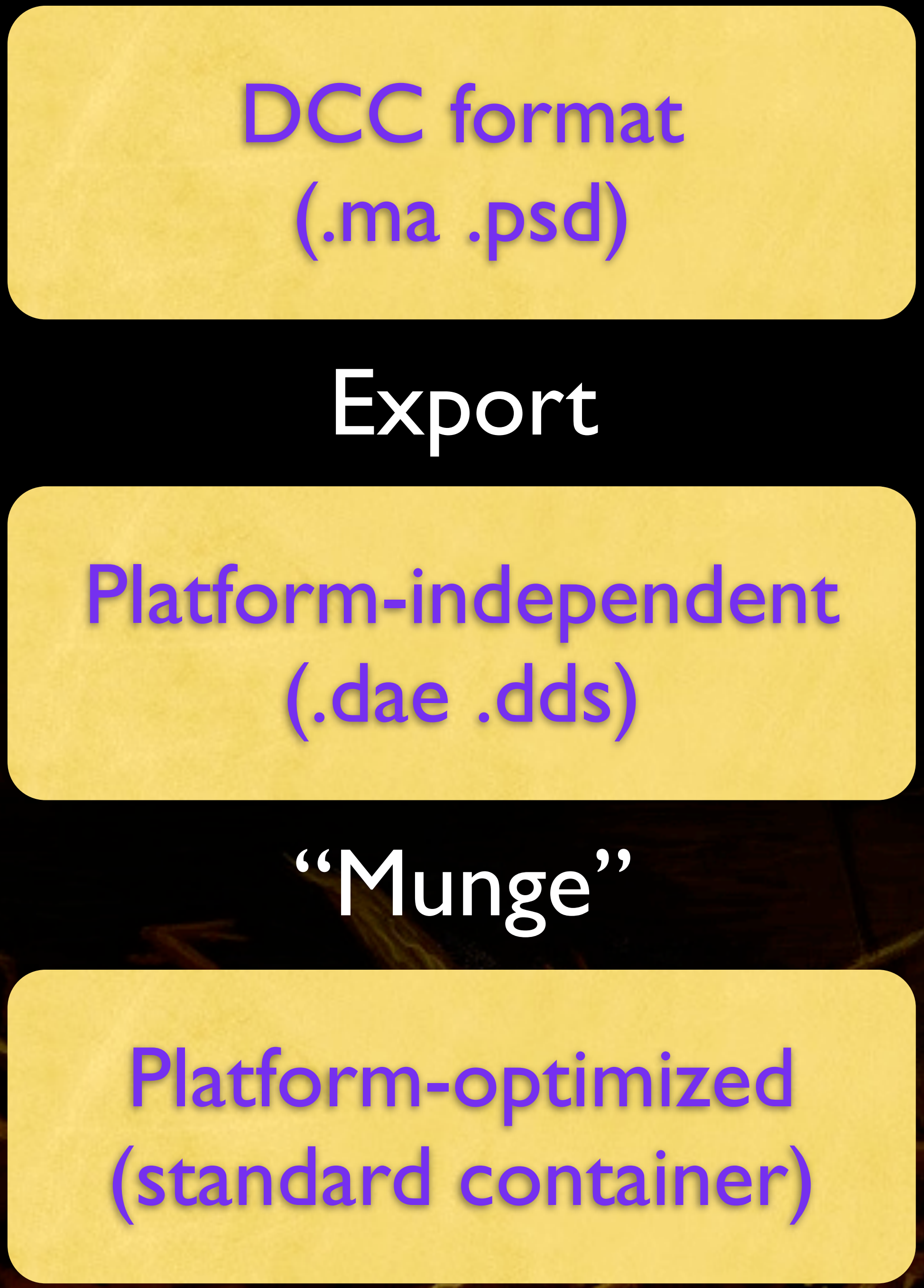


DCC format
(.ma .psd)

Export

Platform-independent
(.dae .dds)

Asset Pipe



DCC format
(.ma .psd)

Export

Platform-independent
(.dae .dds)

“Munge”

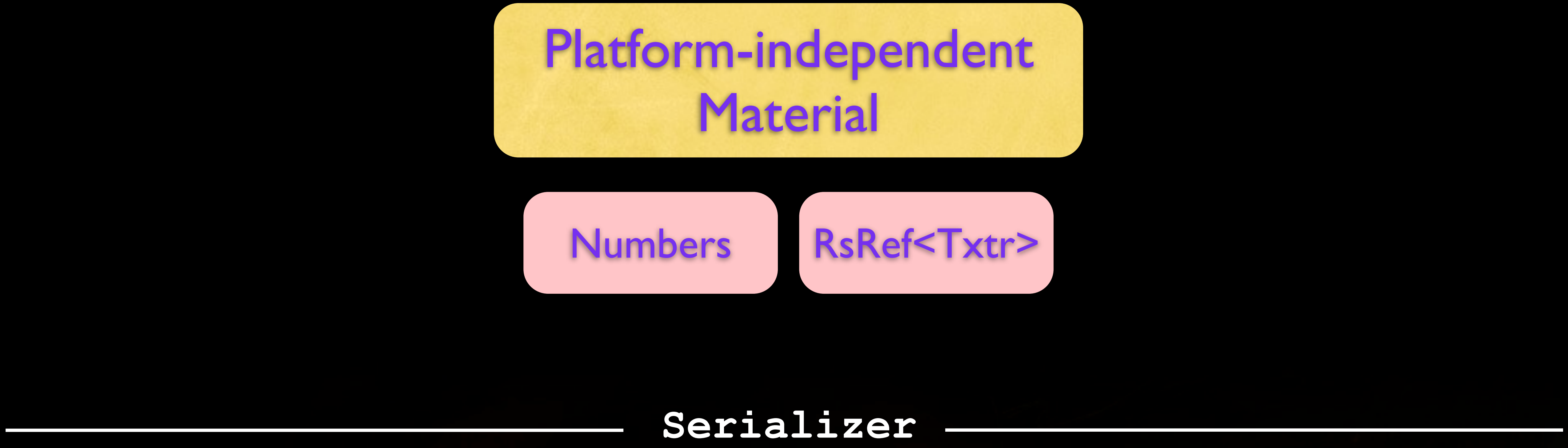
Platform-optimized
(standard container)

Metadata (references)

Platform-independent Material

Serializer

Metadata (references)



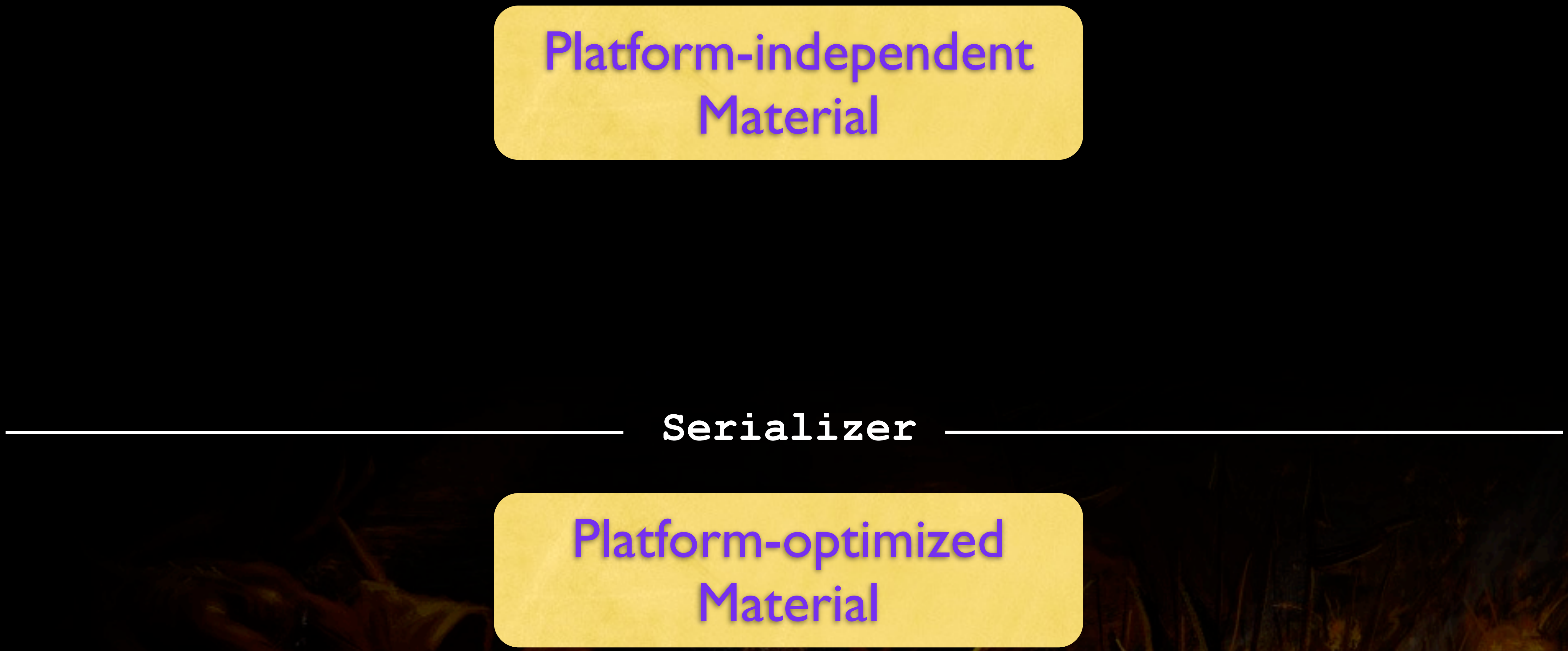
Platform-independent Material

Numbers

RsRef<Txtr>

Serializer

Metadata (references)



Platform-independent Material

Serializer

Platform-optimized Material

Metadata

```
buried within SeaOfBlackTears.dae:

  <library_effects>
    <effect id="SeaOfBlackTears-fx">
      <profile_CG>
        <include sid="include" url=
           "Environments/Materials/B/SeaOfBlackTears.Mtrl"/>
      </profile_CG>
    </effect>
  </library_effects>
```

Metadata

```
SeaOfBlackTears.Mtrl:

Material
{
  NormalTexture =
    @Environments/Textures/Special/OceanWaves_Norm;

  DiffuseTexture =
    @Particles/Textures/WaterFoam_Foam;

  EnvironmentMapTexture =
    @Particles/Textures/SparkleCube/BlackTears_Env;
}
```

Metadata

- Other sources of references

Metadata

- Other sources of references
  - C++: game config, level list, special cases

```
RsRef<LevelList> rList =
    RsBind<LevelList>("Gameplay/Levels/Levels");
const LevelList* list = rList.Load();
```

Metadata

- Other sources of references
  - C++: game config, level list, special cases
  - Prototypes: meshes, animation lists, rigid bodies

```
Prototype Coal : GameObject {
   Add CoRenderMesh {
      MeshSet=@Characters/Props/Rig/Coal;
      ShadowCaster=true;
   }
};
```

Metadata

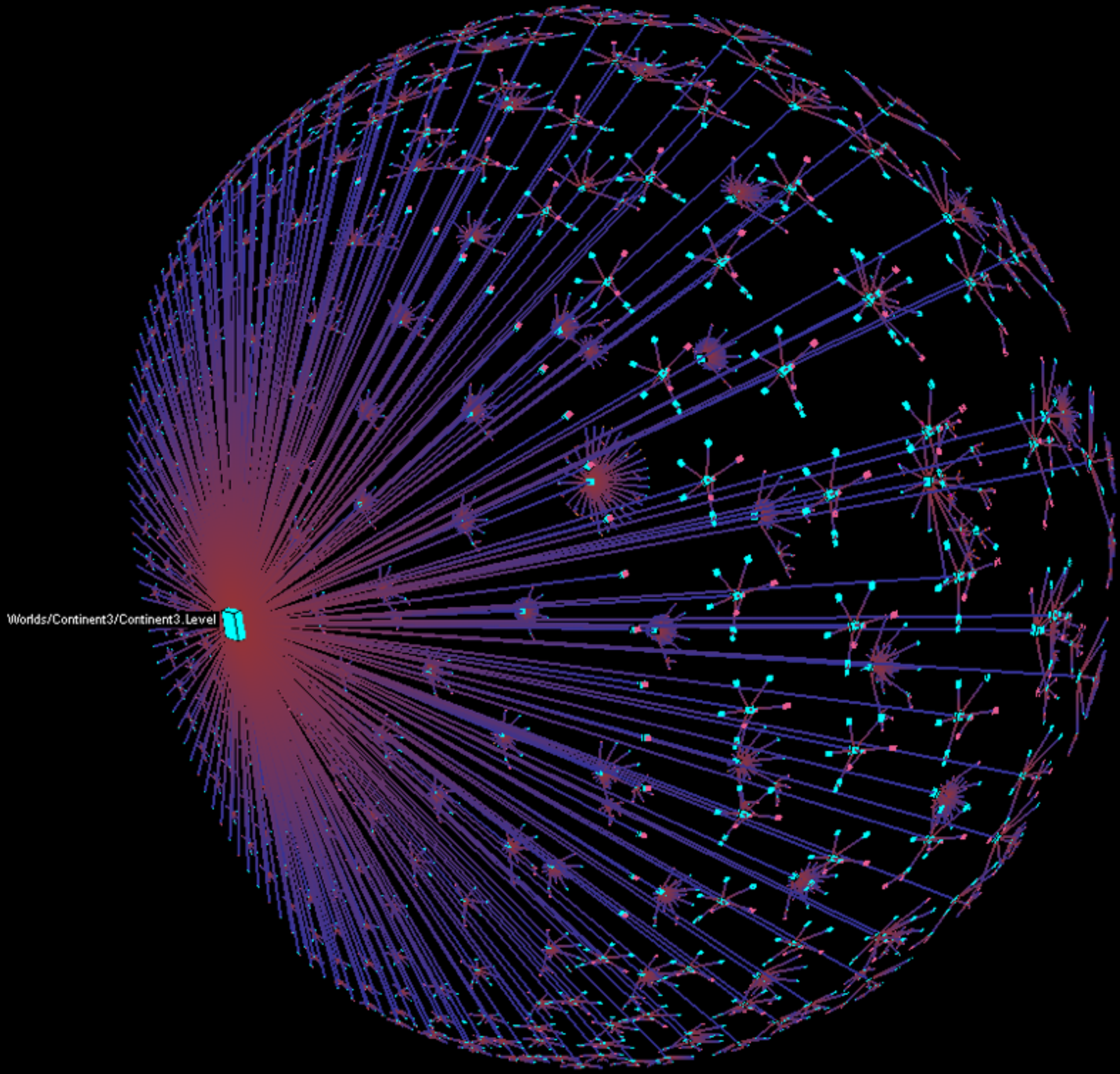
- Other sources of references
  - C++: game config, level list, special cases
  - Prototypes: meshes, animation lists, rigid bodies
  - Script: cutscenes, prototypes

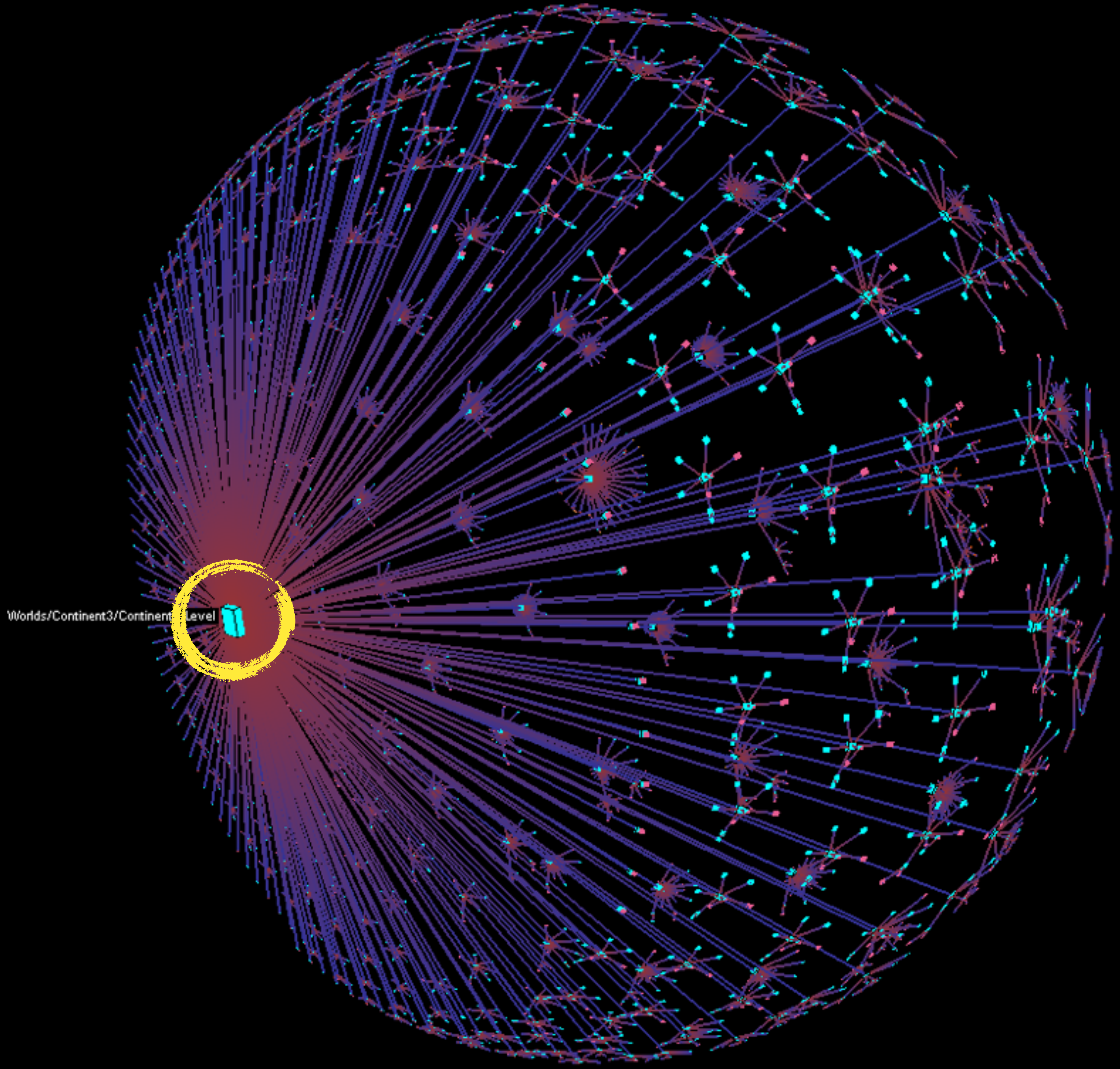
```
game.Spawn( PROTO('Coal'), ... )

cs.LoadAndPause(RESOURCE('Cutscenes/SMLO/SMLO.Ctsn'))
```

Refgraph

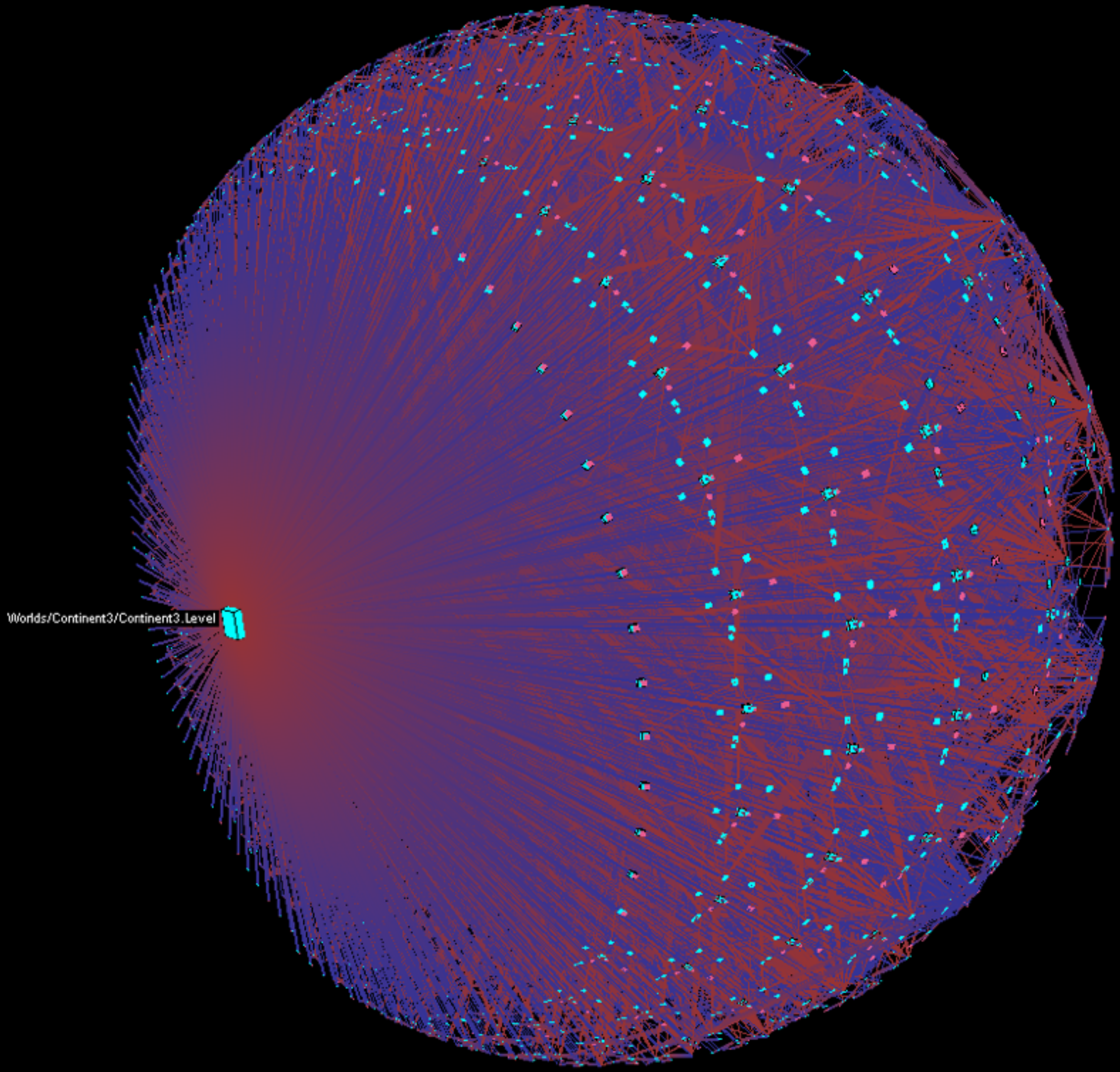
Refgraph

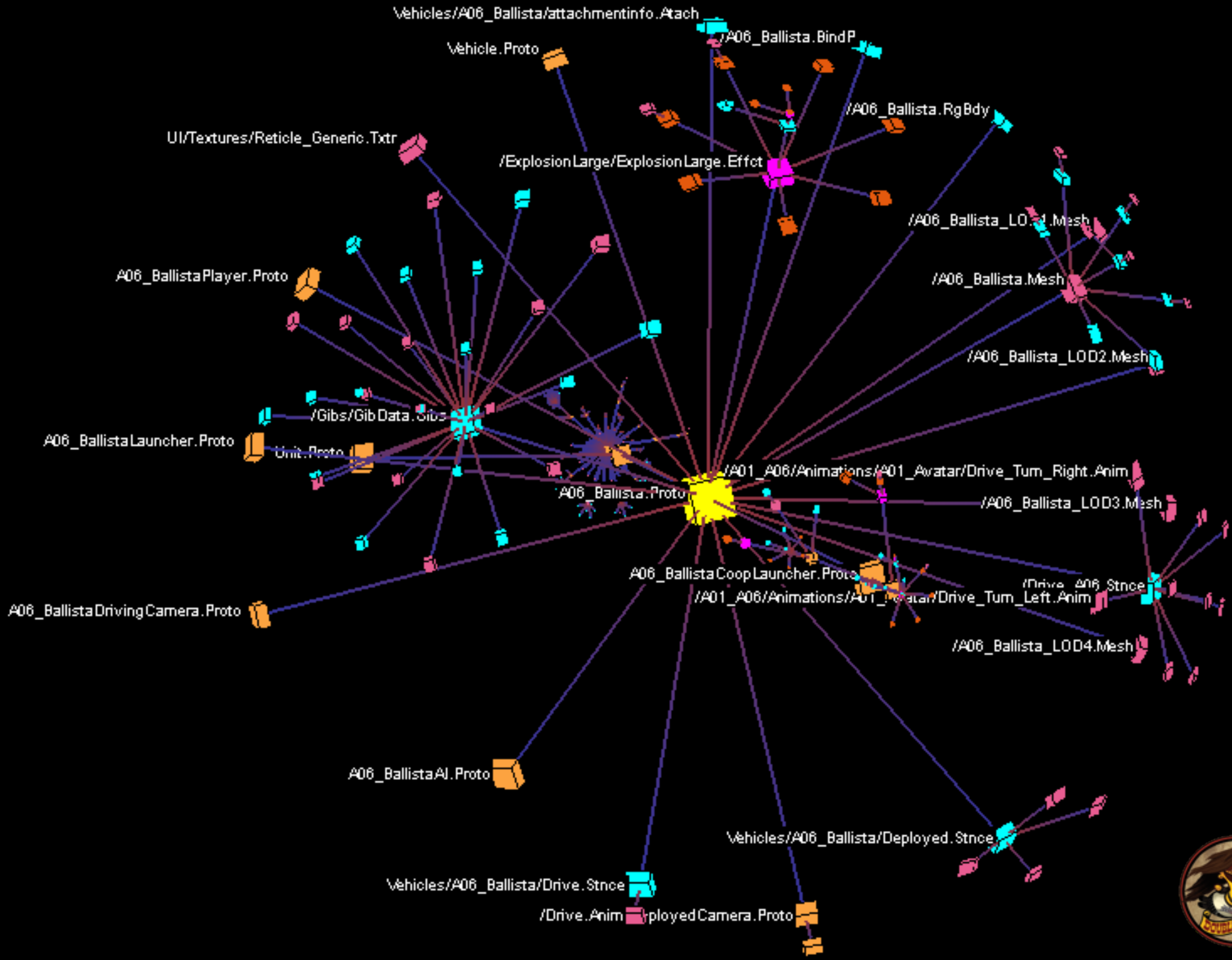
- Directed graph that contains **all** assets
- Nodes are assets, cpp, scripts, prototypes
- Edges are references

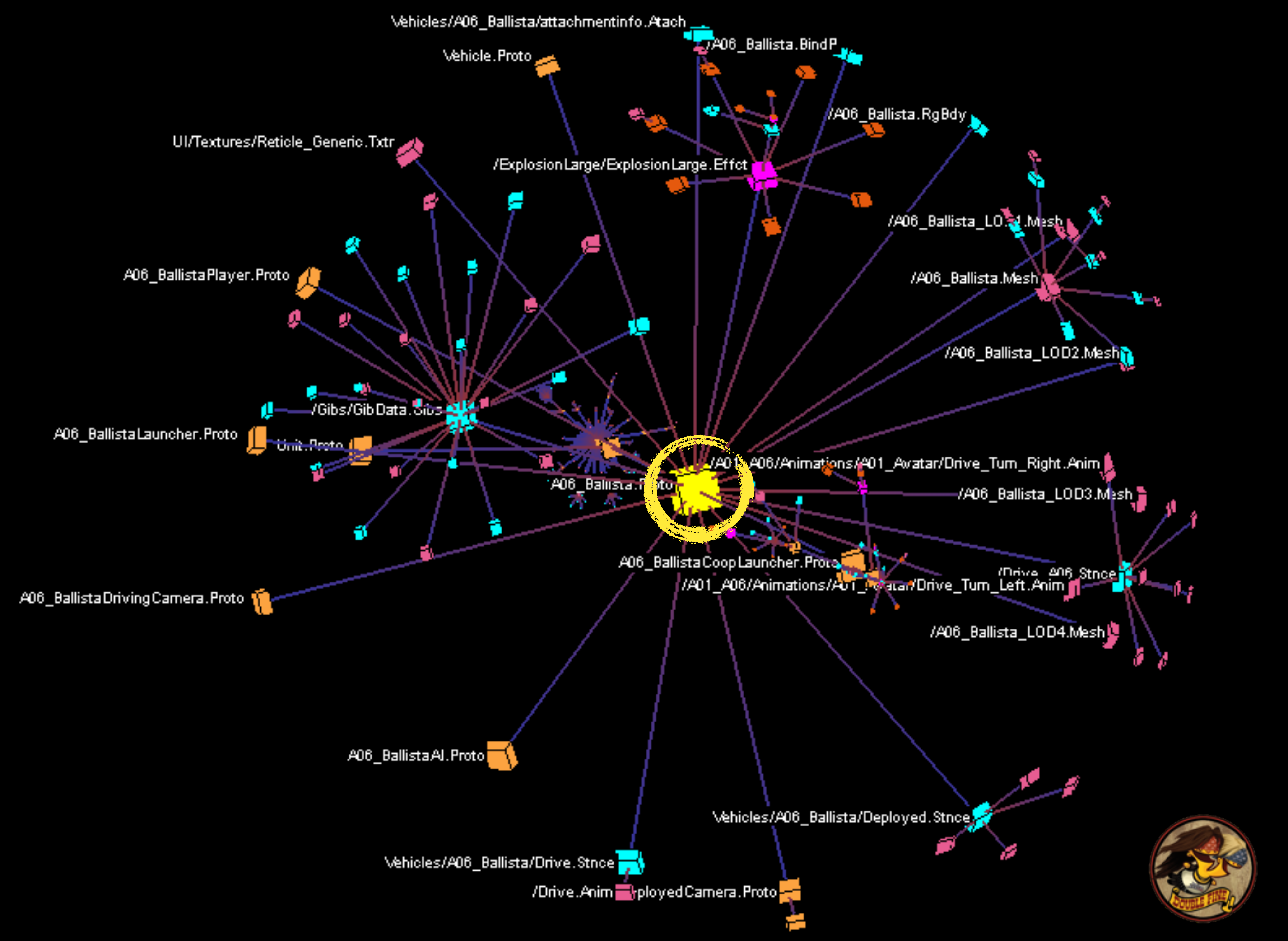

Worlds/Continent3/Continent3.Level

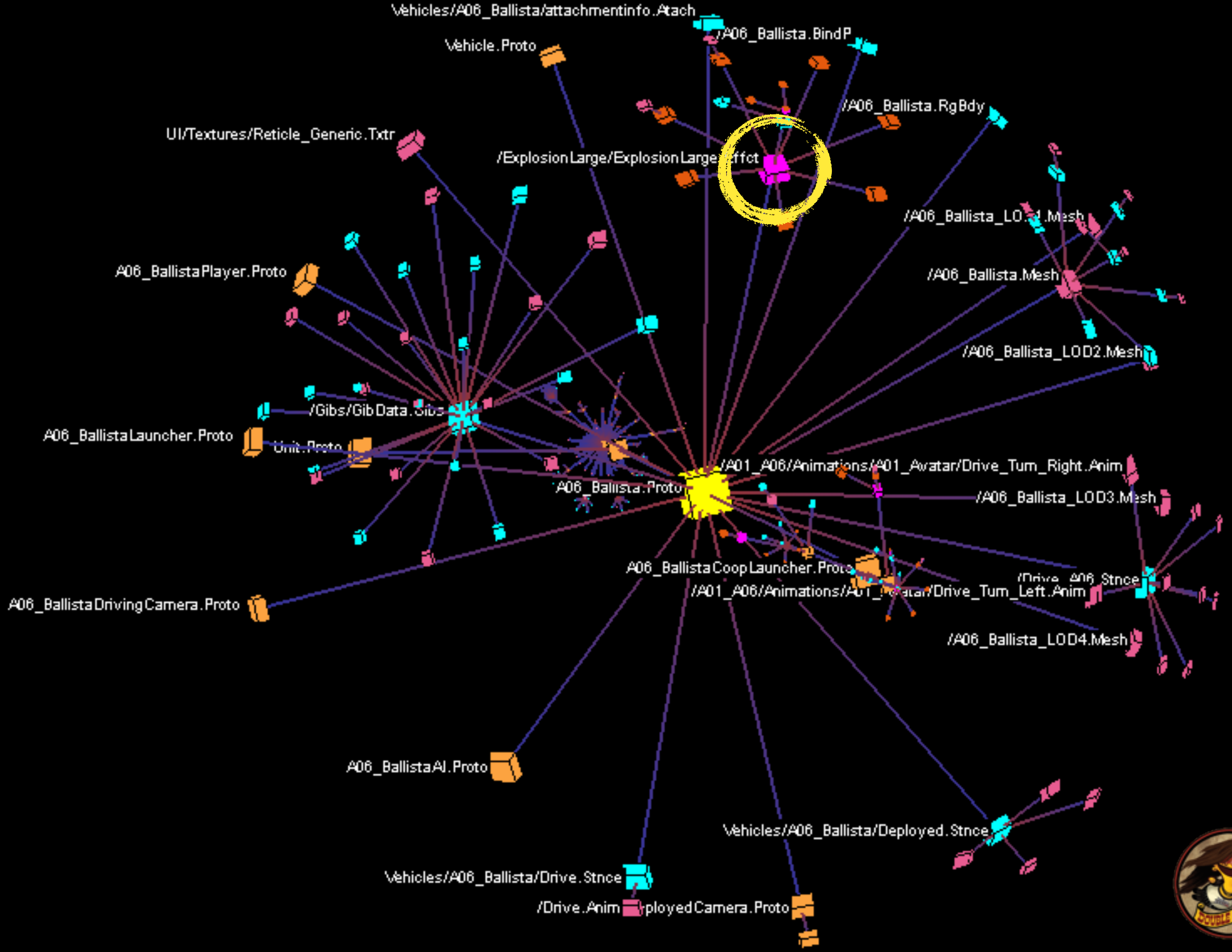


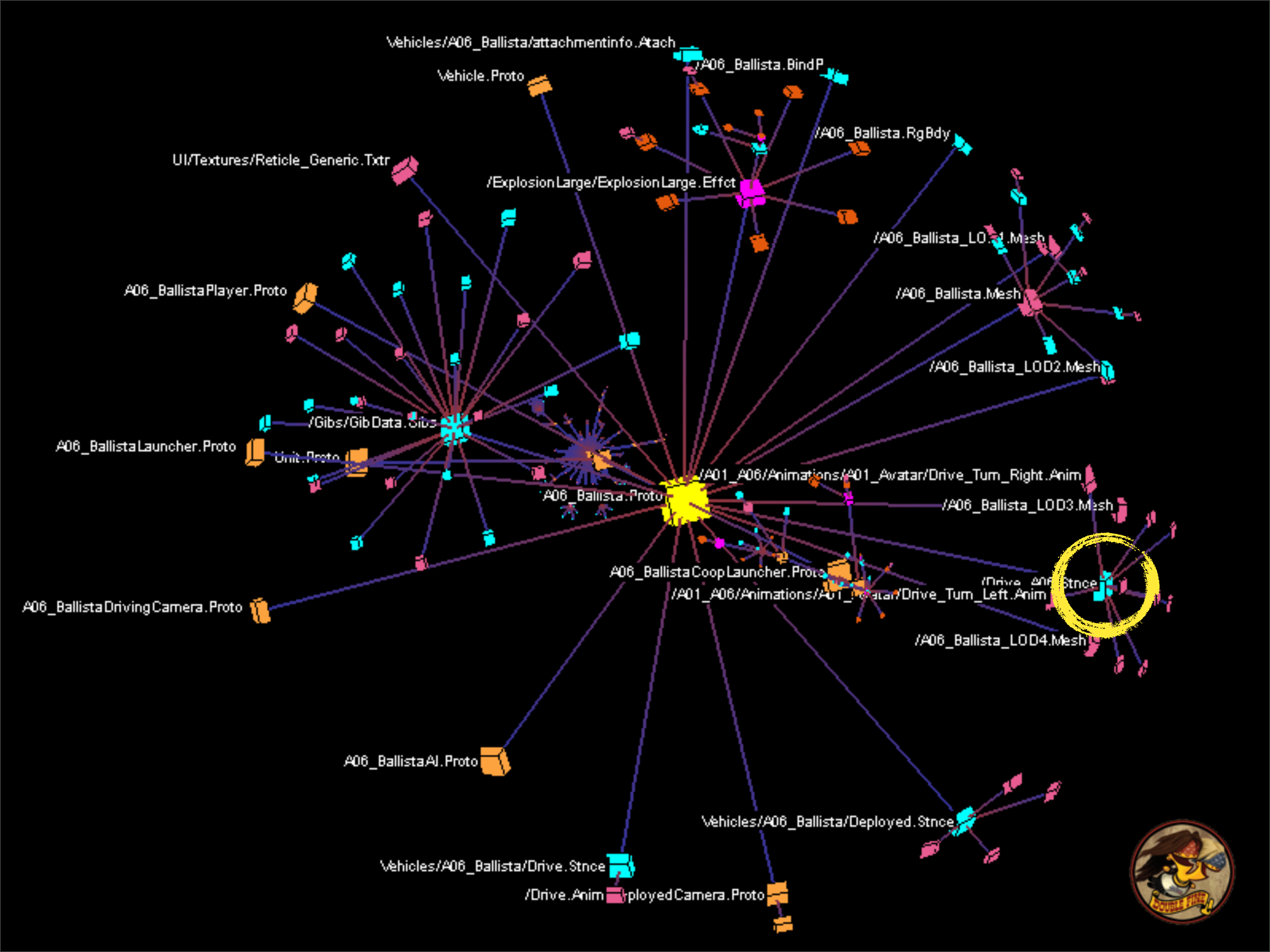
Worlds/Continent3/Continent Level

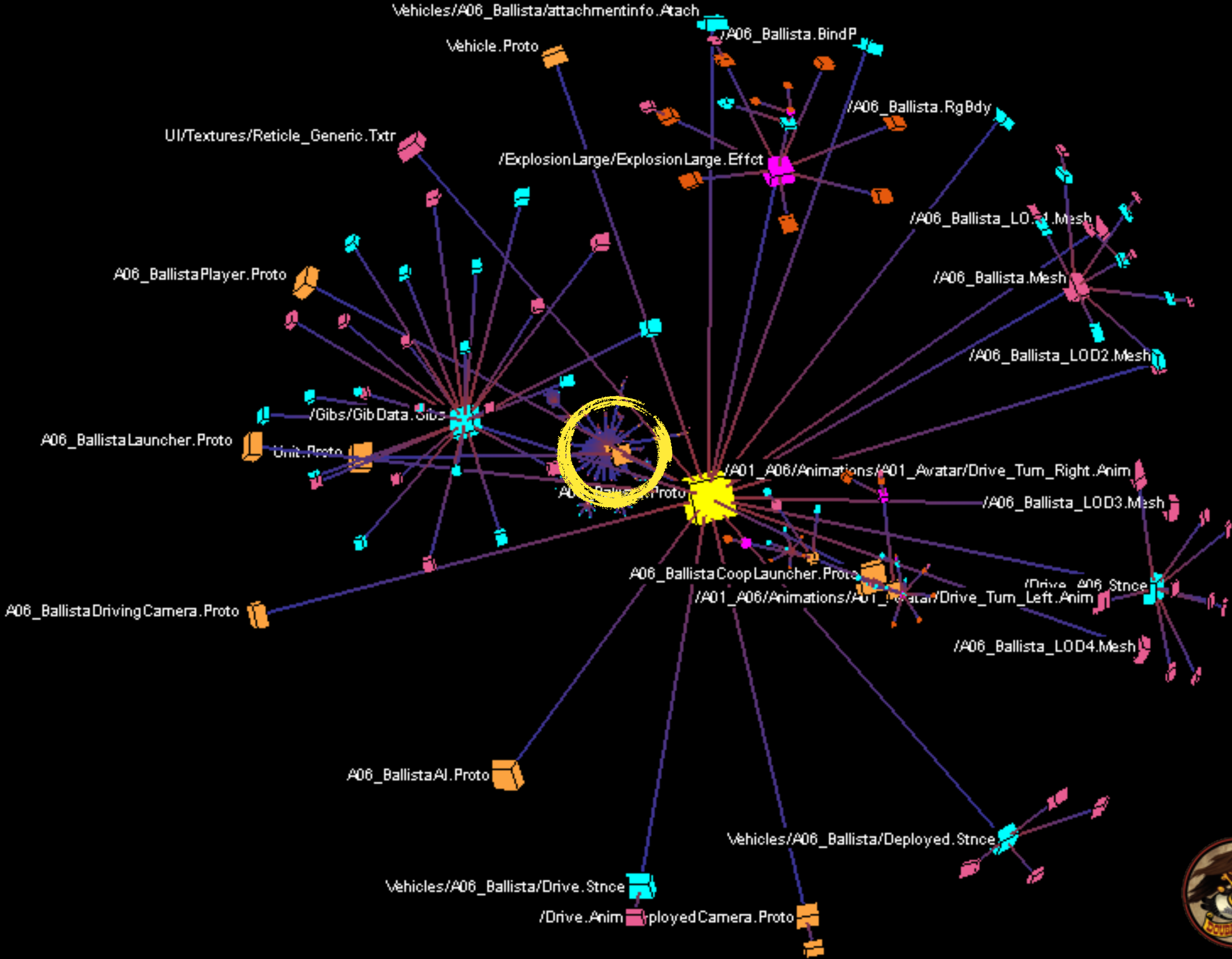

Worlds/Continent3/Continent3.Level


Vehicles/A06_Ballista/attachmentinfo.Atach
/A06_Ballista.BindP
Vehicle.Proto
/A06_Ballista.RgBdy
UI/Textures/Reticle_Generic.Txtr
/ExplosionLarge/ExplosionLarge.Effct
/A06_Ballista_LOD1.Mesh
A06_BallistaPlayer.Proto
/A06_Ballista.Mesh
/A06_Ballista_LOD2.Mesh
/Gibs/GibData.Gibs
A06_BallistaLauncher.Proto
Unit.Proto
/A01_A06/Animations/A01_Avatar/Drive_Turn_Right.Anim
A06_Ballista.Proto
/A06_Ballista_LOD3.Mesh
A06_BallistaCoopLauncher.Proto
/Drive_A06_Stnce
/A01_A06/Animations/A01_Avatar/Drive_Turn_Left.Anim
A06_BallistaDrivingCamera.Proto
/A06_Ballista_LOD4.Mesh
A06_BallistaAI.Proto
Vehicles/A06_Ballista/Deployed.Stnce
Vehicles/A06_Ballista/Drive.Stnce
/Drive.Anim
ployedCamera.Proto


Vehicles/A06_Ballista/attachmentinfo.Atach
/A06_Ballista.BindP
Vehicle.Proto
/A06_Ballista.RgBdy
UI/Textures/Reticle_Generic.Txtr
/ExplosionLarge/ExplosionLarge.Effct
/A06_Ballista_LO 1.Mesh
A06_BallistaPlayer.Proto
/A06_Ballista.Mesh
/A06_Ballista_LOD2.Mesh
/Gibs/GibData.Gibs
A06_BallistaLauncher.Proto
Unit.Proto
/A01_A06/Animations/A01_Avatar/Drive_Turn_Right.Anim
A06_Ballista.Proto
/A06_Ballista_LOD3.Mesh
A06_BallistaCoopLauncher.Proto
/Drive_A06_Stnce
/A01_A06/Animations/A01_Avatar/Drive_Turn_Left.Anim
A06_BallistaDrivingCamera.Proto
/A06_Ballista_LOD4.Mesh
A06_BallistaAI.Proto
Vehicles/A06_Ballista/Deployed.Stnce
Vehicles/A06_Ballista/Drive.Stnce
/Drive.Anim
ployedCamera.Proto


Vehicles/A06_Ballista/attachmentinfo.Atach
/A06_Ballista.BindP
Vehicle.Proto
/A06_Ballista.RgBdy
UI/Textures/Reticle_Generic.Txtr
/ExplosionLarge/ExplosionLarge ffct
/A06_Ballista_LO 1.Mesh
A06_BallistaPlayer.Proto
/A06_Ballista.Mesh
/A06_Ballista_LOD2.Mesh
/Gibs/GibData.Gibs
A06_BallistaLauncher.Proto
Unit.Proto
/A01_A06/Animations/A01_Avatar/Drive_Turn_Right.Anim
A06_Ballista.Proto
/A06_Ballista_LOD3.Mesh
A06_BallistaCoopLauncher.Proto
/Drive_A06_Stnce
/A01_A06/Animations/A01_Avatar/Drive_Turn_Left.Anim
A06_BallistaDrivingCamera.Proto
/A06_Ballista_LOD4.Mesh
A06_BallistaAI.Proto
Vehicles/A06_Ballista/Deployed.Stnce
Vehicles/A06_Ballista/Drive.Stnce
/Drive.Anim
ployedCamera.Proto


Vehicles/A06_Ballista/attachmentinfo.Atach
/A06_Ballista.BindP
Vehicle.Proto
/A06_Ballista.RgBdy
UI/Textures/Reticle_Generic.Txtr
/ExplosionLarge/ExplosionLarge.Effct
/A06_Ballista_LOD1.Mesh
/A06_Ballista.Mesh
A06_BallistaPlayer.Proto
/A06_Ballista_LOD2.Mesh
/Gibs/GibData.Gibs
A06_BallistaLauncher.Proto
Unit.Proto
/A01_A06/Animations/A01_Avatar/Drive_Turn_Right.Anim
A06_Ballista.Proto
/A06_Ballista_LOD3.Mesh
A06_BallistaCoopLauncher.Proto
/Drive_A06 Stnce
/A01_A06/Animations/A01_Avatar/Drive_Turn_Left.Anim
A06_BallistaDrivingCamera.Proto
/A06_Ballista_LOD4.Mesh
A06_BallistaAI.Proto
Vehicles/A06_Ballista/Deployed.Stnce
Vehicles/A06_Ballista/Drive.Stnce
/Drive.Anim
ployedCamera.Proto
DOUBLE FINE


Vehicles/A06_Ballista/attachmentinfo.Atach
/A06_Ballista.BindP
Vehicle.Proto
/A06_Ballista.RgBdy
UI/Textures/Reticle_Generic.Txtr
/ExplosionLarge/ExplosionLarge.Effct
/A06_Ballista_LOD1.Mesh
A06_BallistaPlayer.Proto
/A06_Ballista.Mesh
/A06_Ballista_LOD2.Mesh
/Gibs/GibData.Gibs
A06_BallistaLauncher.Proto
Unit.Proto
/A01_A06/Animations/A01_Avatar/Drive_Turn_Right.Anim
A06_Ballista.Proto
/A06_Ballista_LOD3.Mesh
A06_BallistaCoopLauncher.Proto
/Drive_A06.Stnce
/A01_A06/Animations/A01_Avatar/Drive_Turn_Left.Anim
A06_BallistaDrivingCamera.Proto
/A06_Ballista_LOD4.Mesh
A06_BallistaAI.Proto
Vehicles/A06_Ballista/Deployed.Stnce
Vehicles/A06_Ballista/Drive.Stnce
/Drive.Anim
ployedCamera.Proto

Interactive Visualization of Large Graphs and Networks

graphics.stanford.edu/papers/munzner_thesis

Refgraph uses

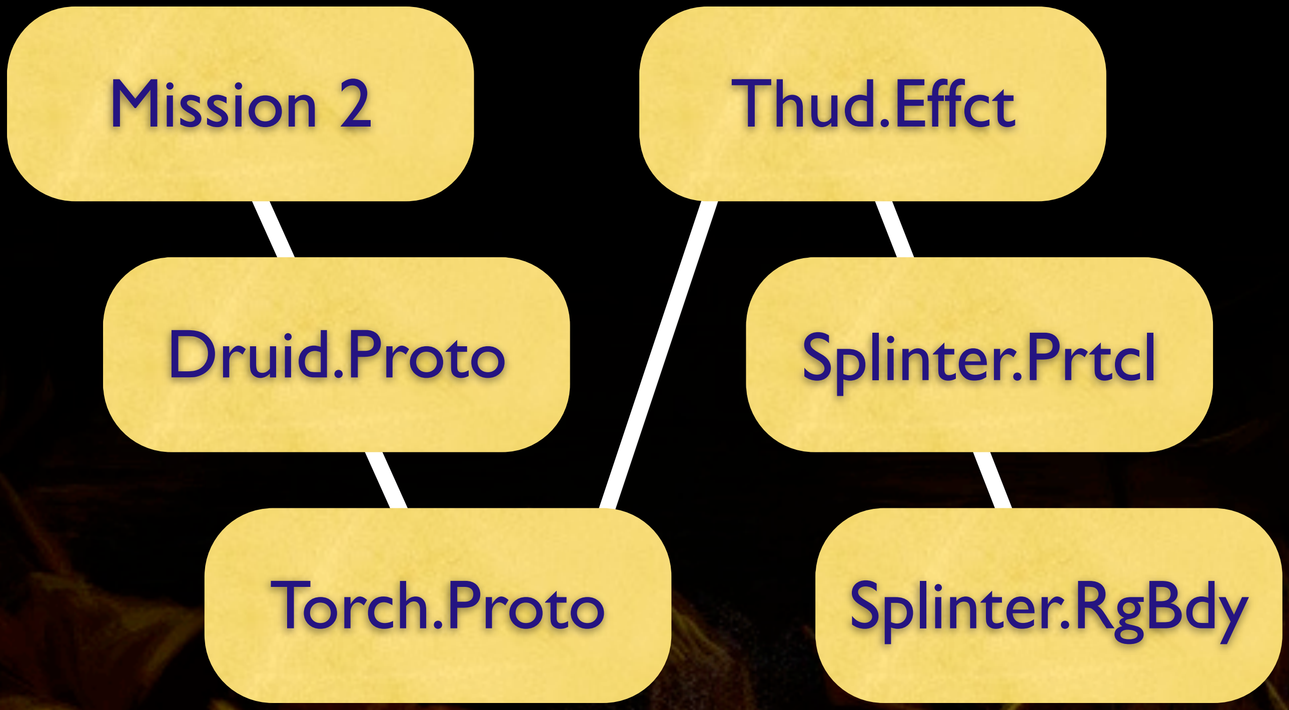
- “Where is a particular asset used?”
  - Find all graph nodes which link to the asset
- “Which assets do we ship?”
  - Choose some “seed” assets
  - Find all assets reachable from seeds

Refgraph uses

- “Which assets do we preload?”
- We want runtime access to this data!
- Refgraph raw form is unsuitable

Refgraph uses

- We don’t care about the topology



Mission 2

Thud.Effct

Druid.Proto

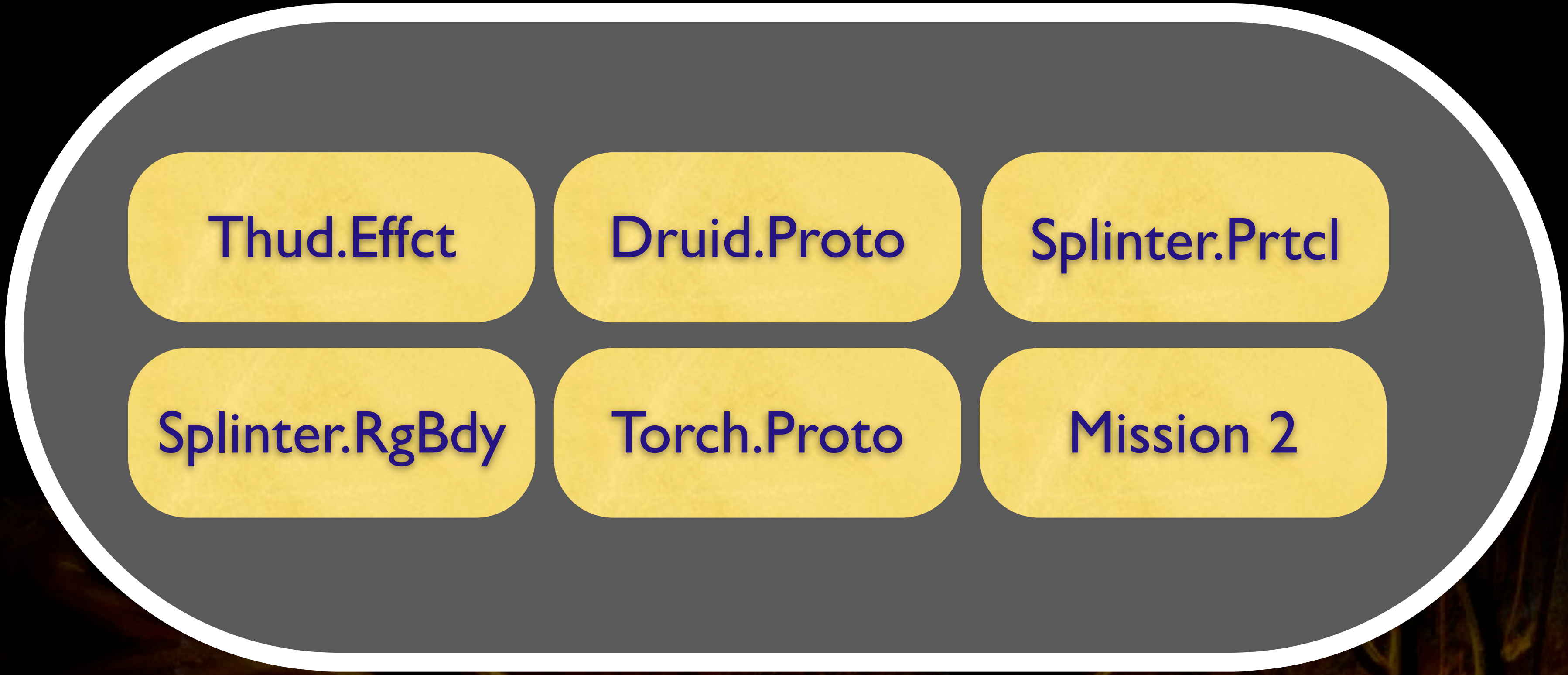
Splinter.Prtcl

Torch.Proto

Splinter.RgBdy

Refgraph uses

- We don’t care about the topology
- ...or the traversal order



Thud.Effct

Druid.Proto

Splinter.Prtcl

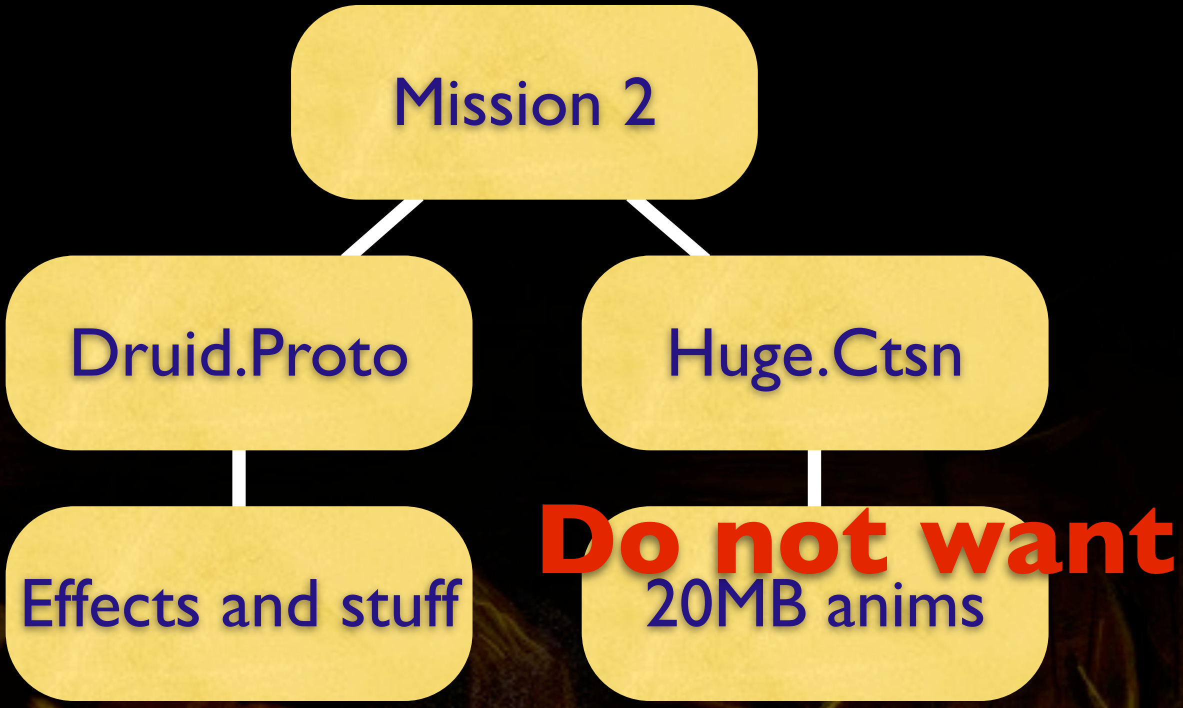
Splinter.RgBdy

Torch.Proto

Mission 2

Refgraph uses

- We don’t want to follow every edge

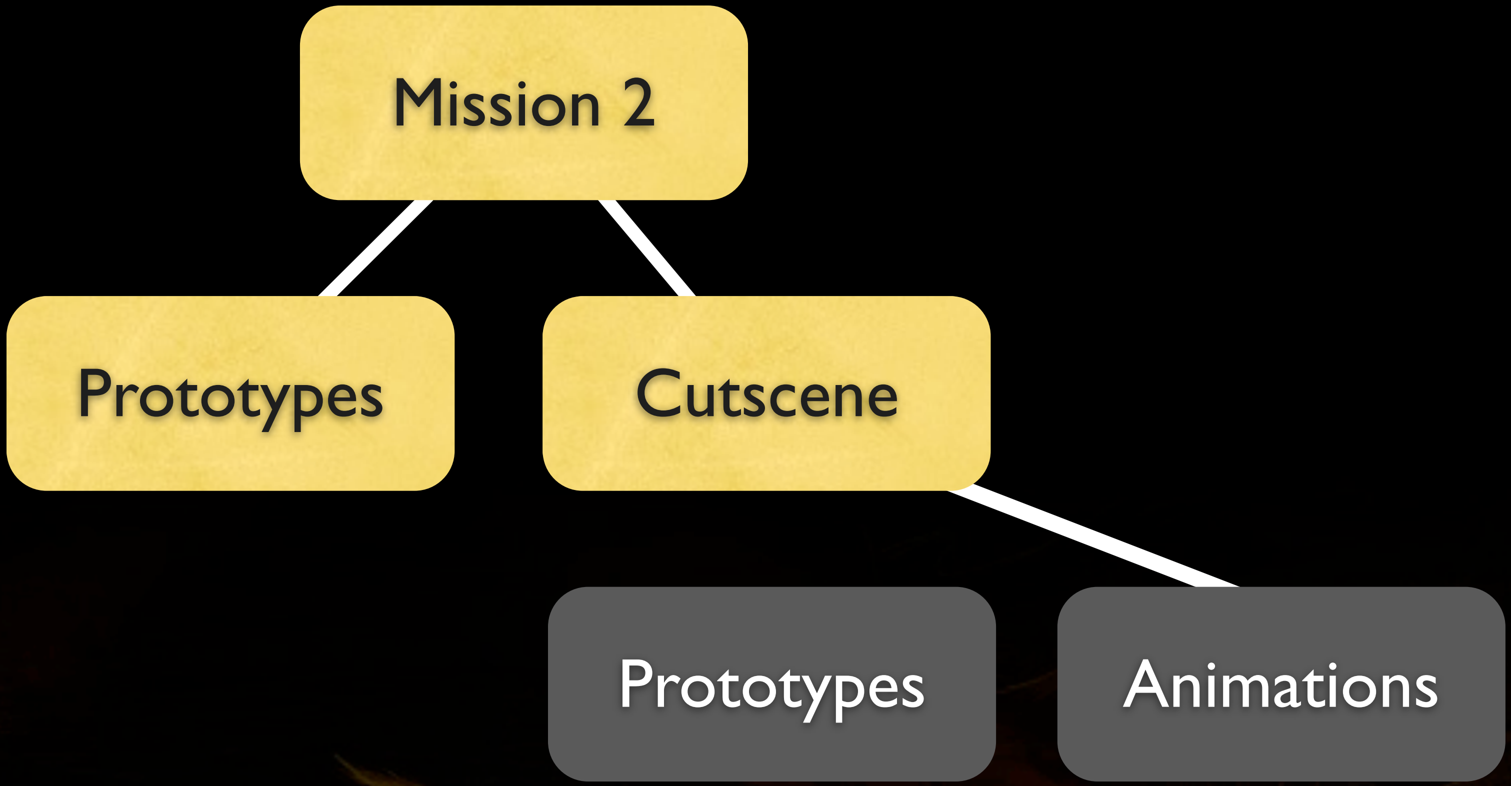


Clumps

Clumps

- Pre-traverse refgraph offline
- Apply domain-specific traversal rules
- Flatten resulting sub-graph into a set
- Call it "clump" instead of Set<RsRef>
  - Also, representation *much* more compact than a generic set

Traversal rules



Mission 2

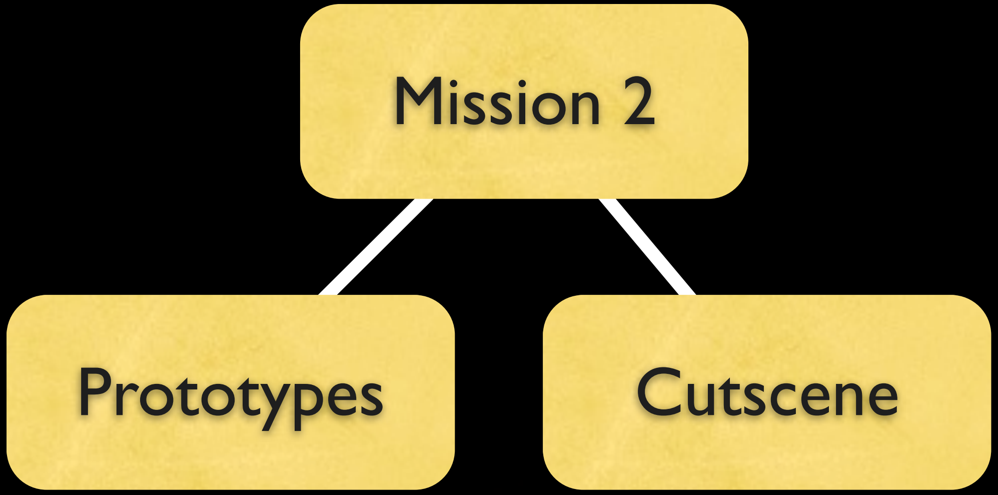
Prototypes

Cutscene

Prototypes

Animations

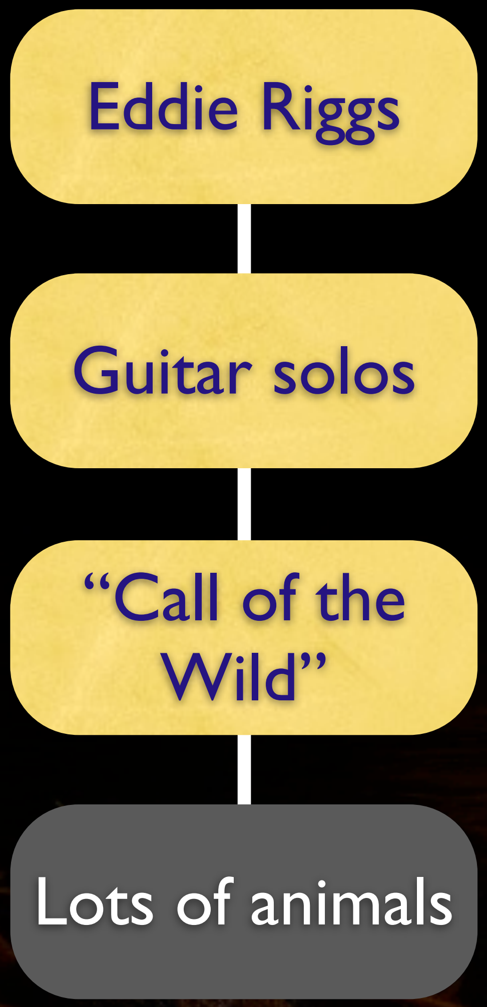
Traversal rules


Mission 2

Prototypes

Cutscene

Traversal rules



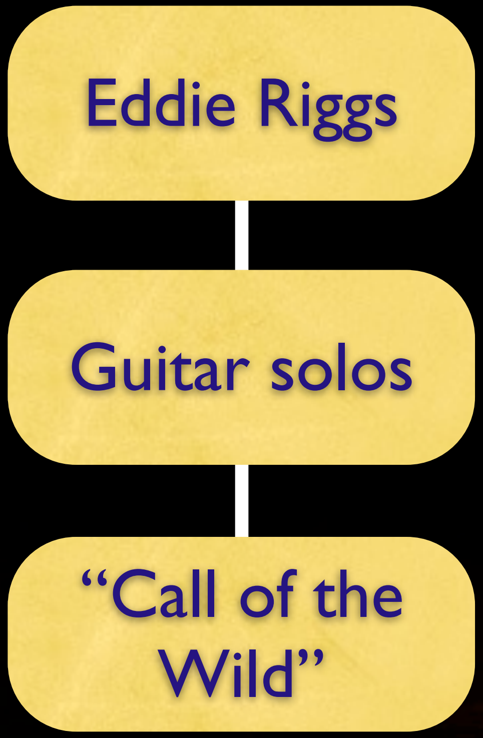
Eddie Riggs

Guitar solos

"Call of the Wild"

Lots of animals

Traversal rules



Eddie Riggs

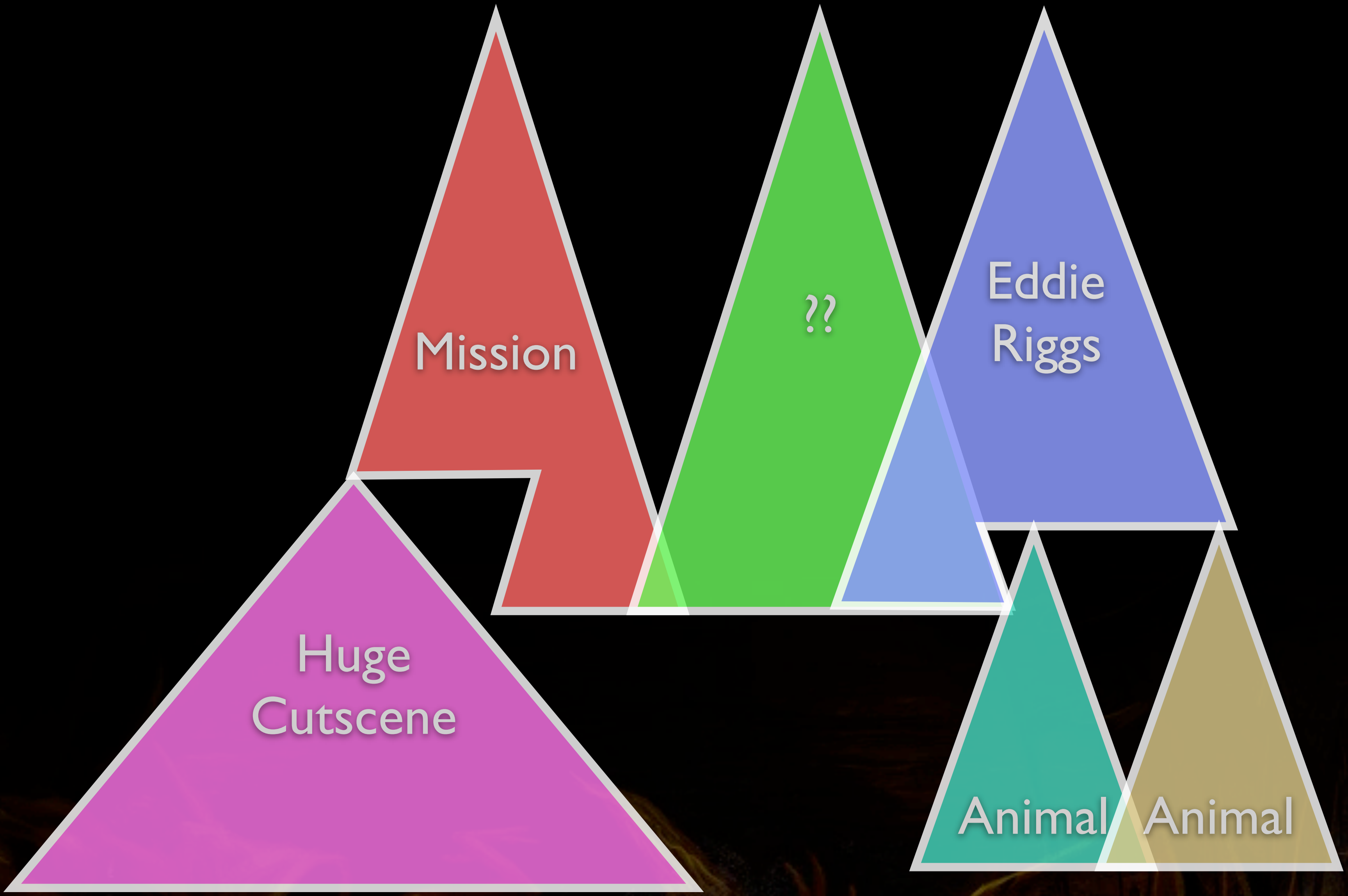
Guitar solos

“Call of the Wild”

Clumps

- Unordered partial traversal of the refgraph
- Not a partition; they can overlap
- **Start** at **nodes** corresponding to runtime decisions
- **Stop** at **edges** corresponding to runtime decisions

Clumps in my head



Using Clumps

Clumps for Everything

- *“What do we want?”*
- Clumps are lists of reachable asset RsRefs
- We want to load everything reachable!
- Not completely unlike level-based games

Clumps for Everything

- Clumps are split up by “runtime decisions”
- Decisions made by a few core systems
  - Terrain tile manager: tile clump = all pre-placed objects
  - Game mission system: mission clump = all scripted objects
  - Cutscene manager, VFX manager, a few other major systems
  - (Plus one or two special cases)

Clumps for Everything

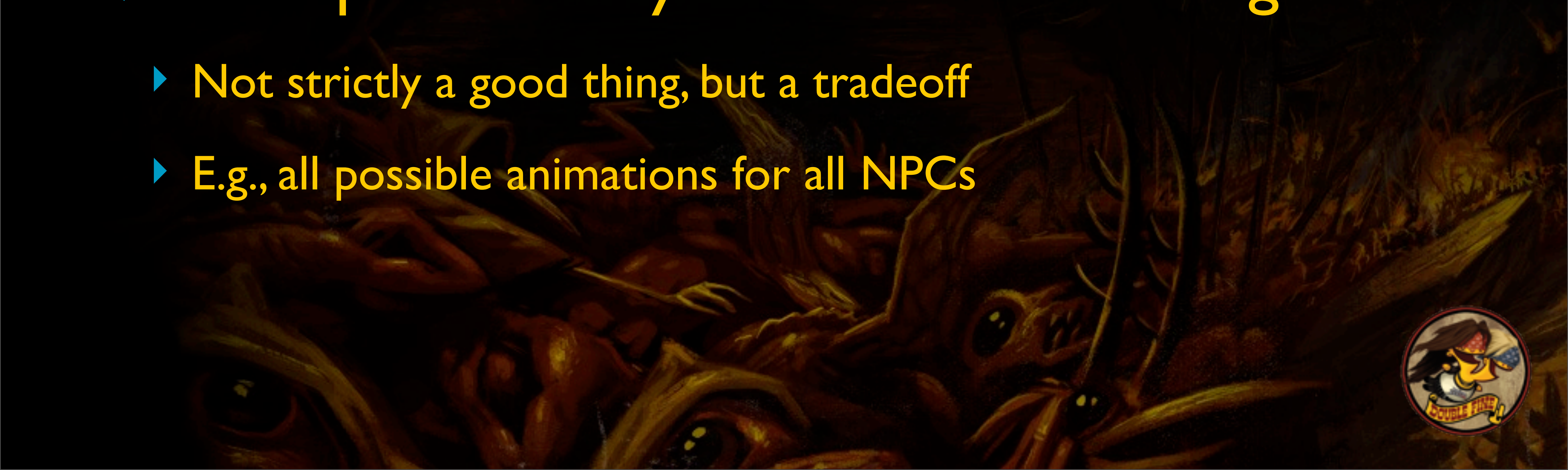
- Simple clump API for handling bulk loads:

```
clump = RsAssetSet::LoadClump(key)
clump->AddReference()
clump->Preload(PRIORITY_Normal)

if (clump->AllLoaded())
   ConstructGameObjects()
```

- Clump data itself compresses very well

Clumps for Everything

- No work required for most clients
  - Few core systems provide 100% reachability coverage
  - Clients can be ignorant of "loaded" state
- Clumps are always worst-case coverage
  - Not strictly a good thing, but a tradeoff
  - E.g., all possible animations for all NPCs



Recap

The Story So Far

The Story So Far

- Asset relationships automatically extracted

The Story So Far

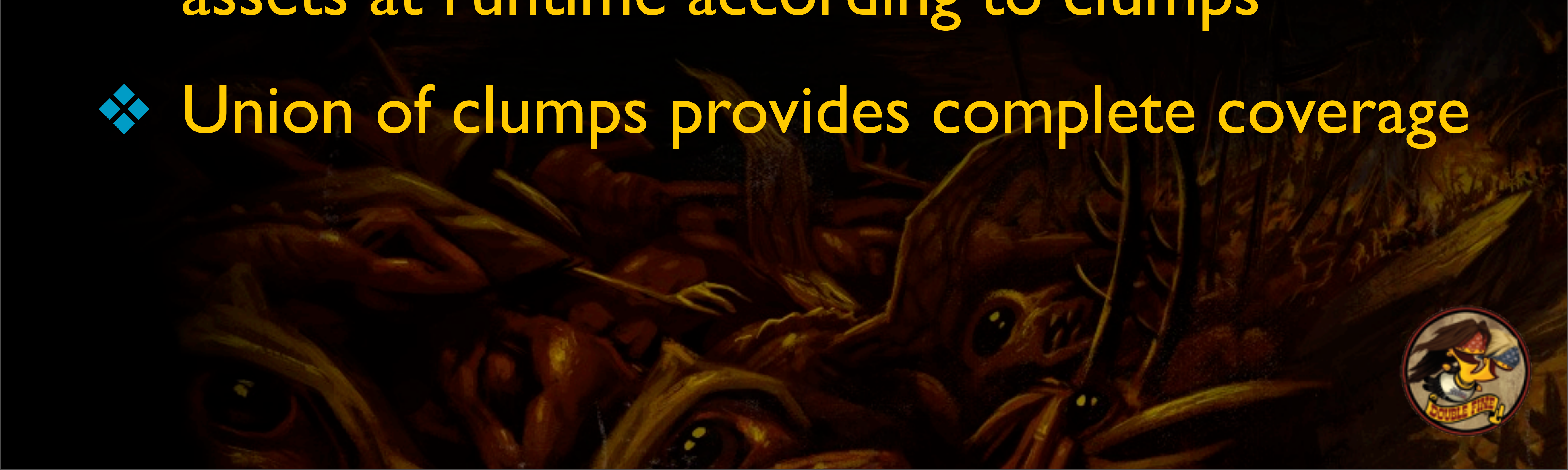
- Asset relationships automatically extracted
- Transformed into clump (set of related refs)

The Story So Far

- Asset relationships automatically extracted
- Transformed into clump (set of related refs)
- Top-level systems bulk-load and unload assets at runtime according to clumps

The Story So Far

- Asset relationships automatically extracted
- Transformed into clump (set of related refs)
- Top-level systems bulk-load and unload assets at runtime according to clumps
- Union of clumps provides complete coverage



The Story So Far

- Asset relationships automatically extracted
- Transformed into clump (set of related refs)
- Top-level systems bulk-load and unload assets at runtime according to clumps
- Union of clumps provides complete coverage
  - With some exceptions.

Render Assets

Meshes and Textures

- Render assets can be extremely large
  - Actual on-screen working set is reasonable
  - Worst-case “reachable” set is not reasonable
- Clump-based ops ignore meshes/textures
- Only the renderer decides when to load

Meshes and Textures

- On-demand loading with some margin
  - Fade in or delay LOD switch if necessary
- Stored in GPU-accessible memory pool
- Treat like LRU cache, evict oldest when full
  - Pool size is much larger than the average working set
  - Hit rate is great thanks to shared materials, asset re-use

Minimizing Pop-In

- LRU cache does a really good job
- World position is continuous, mostly
  - Pop-in at the furthest LOD range is not noticeable
- Teleportation is strictly controlled
  - Pop-in hidden by fade-to-black or loading screen
  - Generally OK with players due to intentional initiation

Side-note: Fragmentation

Side-note: Fragmentation

- Pool fragmentation is a real problem
  - Render assets are large and have unbounded lifetimes

Side-note: Fragmentation

- Pool fragmentation is a real problem
  - Render assets are large and have unbounded lifetimes
- Repack data towards one end of the pool

Side-note: Fragmentation

- Pool fragmentation is a real problem
  - Render assets are large and have unbounded lifetimes
- Repack data towards one end of the pool
- CPU-side implementation is difficult

Side-note: Fragmentation

- Pool fragmentation is a real problem
  - Render assets are large and have unbounded lifetimes
- Repack data towards one end of the pool
- CPU-side implementation is difficult
- Use console-specific GPU memory writes

Side-note: Fragmentation

- Pool fragmentation is a real problem
  - Render assets are large and have unbounded lifetimes
- Repack data towards one end of the pool
- CPU-side implementation is difficult
- Use console-specific GPU memory writes
- Beware platform issues

Side-note: Panic Mode

Side-note: Panic Mode

- Last-minute addition to the engine

Side-note: Panic Mode

- Last-minute addition to the engine
- Artists worked with OSD “budget meter”

Side-note: Panic Mode

- Last-minute addition to the engine
- Artists worked with OSD “budget meter”
- Didn’t account for worst-case player actions

Side-note: Panic Mode

- Last-minute addition to the engine
- Artists worked with OSD “budget meter”
- Didn’t account for worst-case player actions
- Detect critical low free memory, rate-limit new loads while furiously cutting mip levels

Side-note: Panic Mode

- Last-minute addition to the engine
- Artists worked with OSD “budget meter”
- Didn’t account for worst-case player actions
- Detect critical low free memory, rate-limit new loads while furiously cutting mip levels
- Reload textures when danger has passed

Audio/Video

Audio and Video

- Licensed external solutions
- Re-route all file I/O through our scheduler
- Way more painful than anticipated
  - Need a lower-level view to understand the problems
- *We'll get back to this later, again.*

Scheduler

I/O Scheduler

- Queue of pending I/O requests
- One queue/scheduler/thread per device
  - Simplify I/O loop, avoid complex async notifications
- Optical drive is the most important device
  - Might be the **only** device
  - Everything that applies here also helps the hard drive

Know Your Devices

- Some of our preconceptions were wrong
- Optical drives are not slow
  - Max throughput on par with, or better than, hard drive
- Seeking is the enemy of throughput
  - Impact of seeking cannot be overstated
  - Even short seeks can turn 12 MB/sec into 1 MB/sec

I/O Scheduler

I/O Scheduler

- Goal: minimize throughput lost to seeking
  - Use forward-scan “one-way elevator” pattern
  - Sort incoming requests according to disc position
  - Pick the closest forward-seeking request, ideally contiguous

I/O Scheduler

- Goal: minimize throughput lost to seeking
  - Use forward-scan "one-way elevator" pattern
  - Sort incoming requests according to disc position
  - Pick the closest forward-seeking request, ideally contiguous
- Goal: service high-priority requests quickly
  - Higher priority trumps all seek considerations

I/O Scheduler

- Perfect place to use a tree-heap (treap)
  - Combination of binary tree and priority queue
  - Tree key is the physical location of the asset
  - Heap node priority is the request's priority class
  - Look at the root node to get the max priority of all requests
  - Walk the tree to find the closest key at this priority

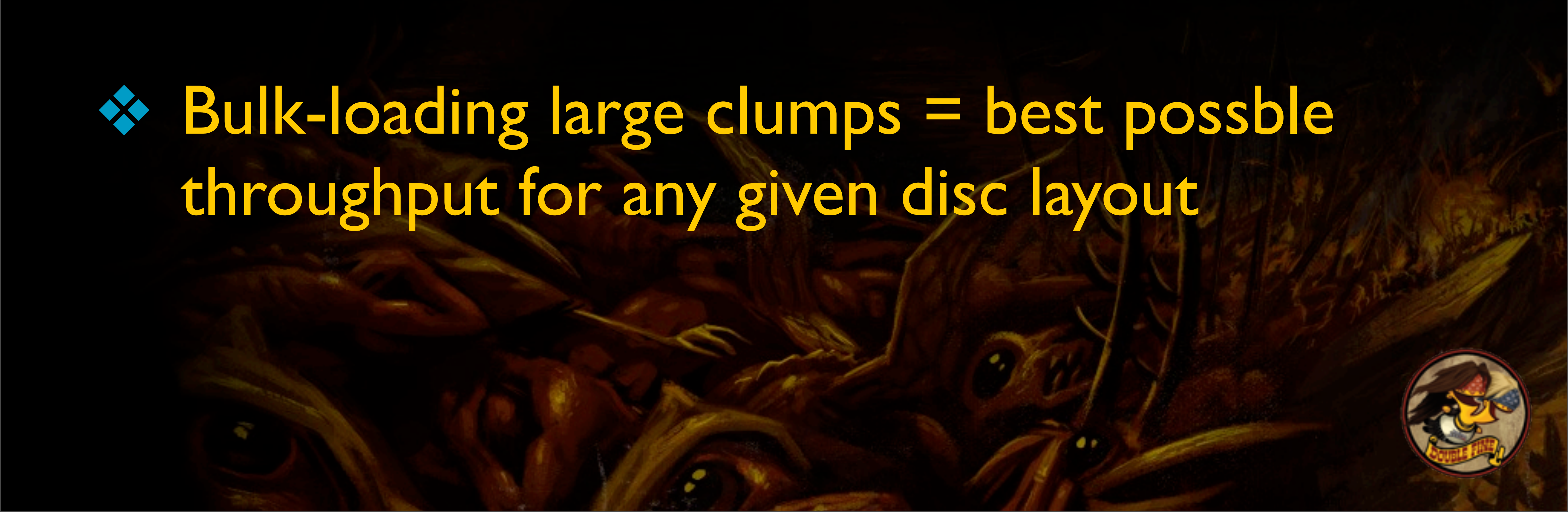
I/O Scheduler (simplified)

```
lastKey = 0
loop:
  topPrio = treap.root.prio
  request = treap.FindNext(lastKey, topPrio)
  if request is null:
    request = treap.FindNext(0, topPrio)
  ProcessRequest(current)
  lastKey = current.key
  treap.Remove(current)
```

I/O Scheduler

- Larger queue means better scheduling
  - Request one asset at a time: scheduler is impotent
  - Request everything at once: scheduler is optimal

- Bulk-loading large clumps = best possble throughput for any given disc layout



I/O Scheduler

- Priority changes have a high cost
  - Data at different priority levels is rarely related
  - (Almost) every priority switch is a large seek
  - Throughput lost to seeks outweighs benefits
- Lesson learned: *don't mess with priority*
  - Game mostly uses "normal" priority
  - Time-critical clumps (tiles) use "high" priority

Audio and Video

- Licensed external solutions
- Re-route all file I/O through our scheduler
- Initial implementation is not shippable
  - Sounds play extremely late
  - Videos and music suffer from stalls and skips
- Examination reveals lots of bad behavior

Audio and Video

- Stuttering causes poor throughput for caller
  - External system interleaves reads with small amounts of work
  - Amplified version of the "priority change" problem
  - Read-ahead helps, but not a 100% fix (eg, small nearby seeks)
- *Hack:* after processing a read from an external system, expect another read
  - Better to wait for 8ms timeout than to perform two large seeks

Audio and Video

- Frequent requests lock out everybody else
  - External system issues one small, unrelated request every 100ms
  - Takes almost 50ms to seek each way
  - Hardly any lower-priority requests get through
- *Hack:* after dropping to a lower priority level, go to location-only mode and ignore priority
  - Guarantees some minimum amount of forward progress

Side-note: Third-Party Libraries

- No standard API for batch requests
  - Generic API encourages blocking on one read at a time
- Not always designed to play with others
  - Often developed and tested without real-world I/O load
- Improving with time and feedback
  - Rising awareness of data streaming issues; FIOS, etc

Audio and Video

- These two hacks solved most of our issues
- In retrospect, maybe not "hacks" at all
- Tricky to formalize, but easy to understand
  - Keep the core concepts as simple as possible
  - Watch out for abusive client behavior
  - Basically, just try to minimize seeks

Disc Layout

Disc Layout

Problem

- Find all used data
- Concatenate it into one or more files
- Minimize seeking

Disc Layout

Observations

- Clumps are used to load batches of data
- Assets in the same clump should be nearby
- Clumps are generated by traversing refgraph

Disc Layout

Concatenate all clumps?

- 1:many mapping for assets and locations adds significant complexity
- Even if it fits on disc, drown in data
- Other reasons; ask if you're curious

Disc Layout

Solution

- Choose a reasonable* ordering of clumps
-
```
foreach clump in clumps:
  foreach asset in clump:
    if not already_packed(asset):
      packfile.append(asset)
```
- Caveat: Sharing breaks contiguity

Disc Layout

Sharing breaks contiguity

Lionwhyte.Mission
Headbanger.Proto
Lionwhyte.Proto

Bladehenge.Mission
Headbanger.Proto
Lars.Proto

Disc Layout

Sharing breaks contiguity

Lionwhyte.Mission

Headbanger.Proto

Lionwhyte.Proto

Bladehenge.Mission

Headbanger.Proto

Lars.Proto

Disc Layout

Sharing breaks contiguity

Lionwhyte.Mission

Headbanger.Proto

Lionwhyte.Proto

Bladehenge.Mission

Headbanger.Proto

Lars.Proto

Disc Layout

Sharing breaks contiguity

Lionwhyte.Mission
Headbanger.Proto
Lionwhyte.Proto

Bladehenge.Mission
Headbanger.Proto
Lars.Proto

Disc Layout

Sharing breaks contiguity

Lionwhyte.Mission

Headbanger.Proto

Lionwhyte.Proto

Bladehenge.Mission

Headbanger.Proto

Lars.Proto

Disc Layout

Sharing breaks contiguity

Lionwhyte.Mission

Headbanger.Proto

Lionwhyte.Proto

Bladehenge.Mission

Lars.Proto

Disc Layout

Sharing breaks contiguity

Lionwhyte.Mission

Headbanger.Proto

Lionwhyte.Proto

Bladehenge.Mission

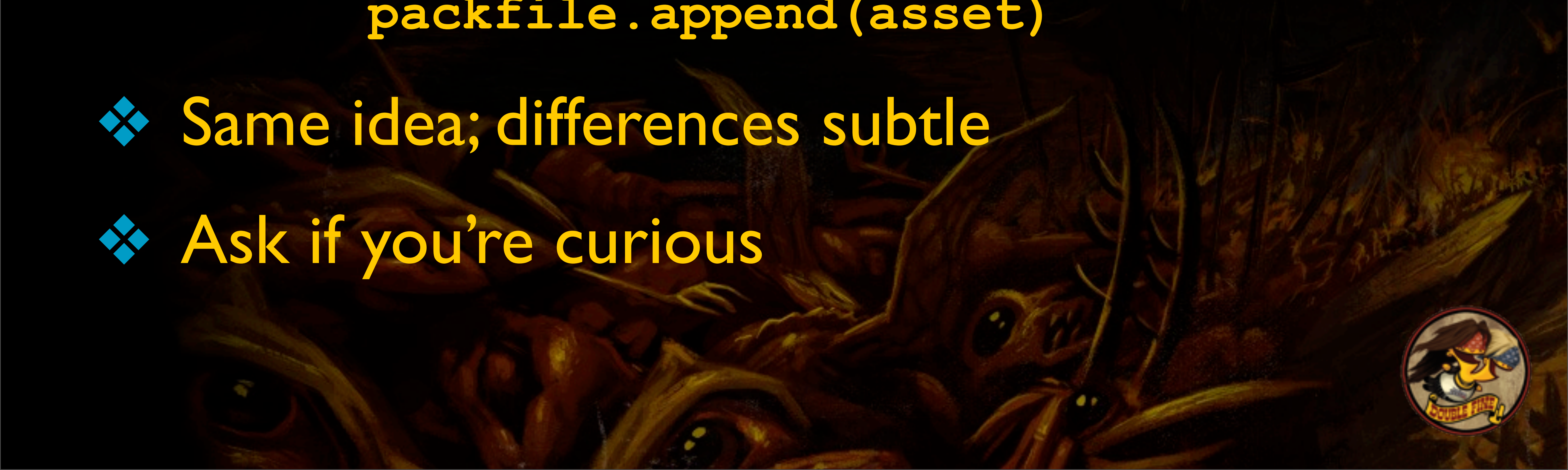
Lars.Proto

Disc Layout

Solution (actual)

- Choose a reasonable ordering of seeds

-
  ```
  foreach seed in seeds:
    foreach asset in traverse(graph, seed):
      if not already_packed(asset):
        packfile.append(asset)
  ```
- Same idea; differences subtle
- Ask if you’re curious



Disc Layout

Why does this work?

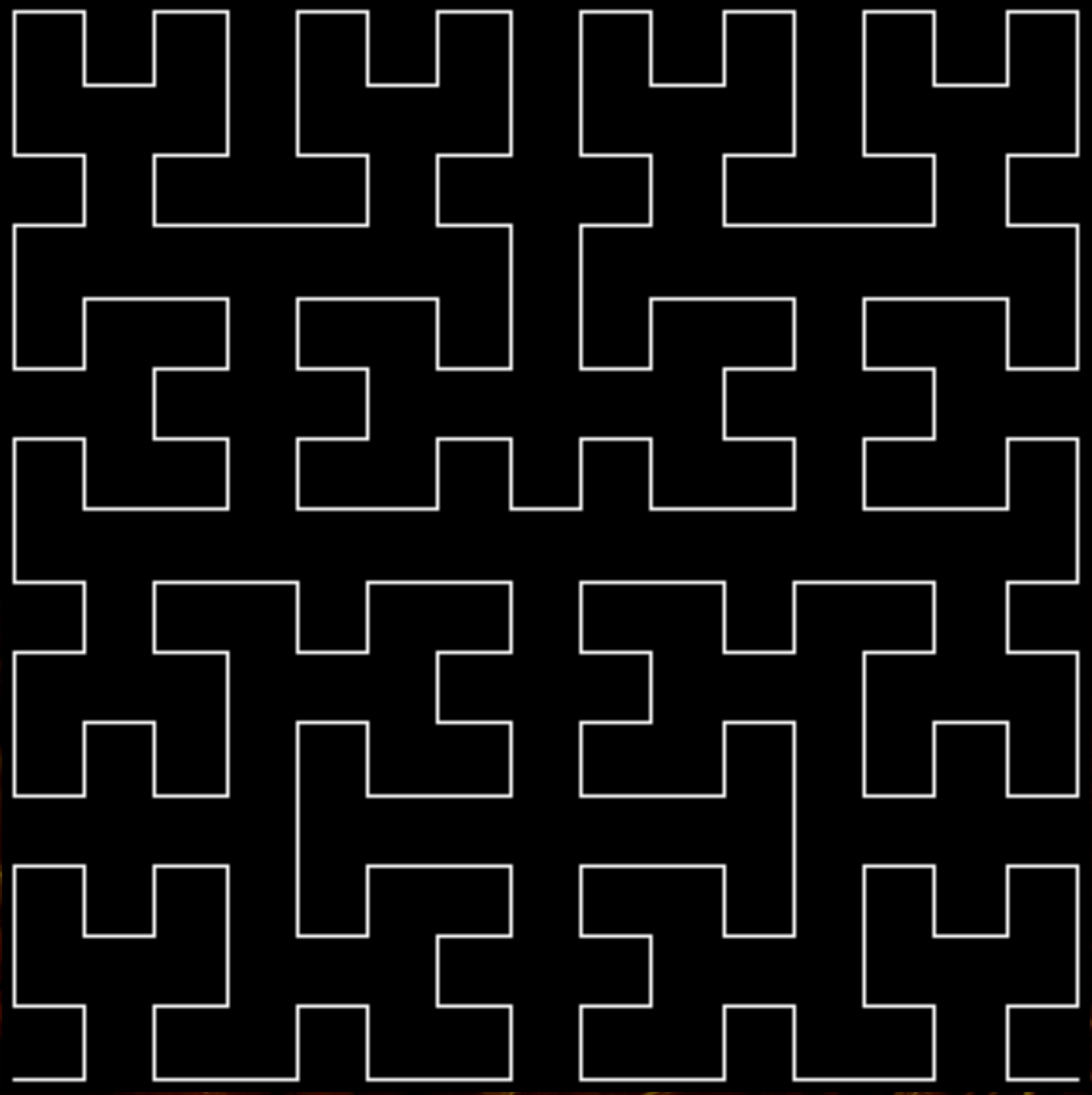
- Reading contiguous assets is very fast
  - Because I/O manager sorts them optimally
- We’re making the clumps contiguous

Disc Layout

“Didn’t you say sharing breaks contiguity?”

- Yes, but data is well correlated
- Heuristics applied aren’t terribly clever
  - Space-filling curve for the continent
  - Curve based on player path probably better
  - Open question: if they were, how much better would they work?
- *The data sorts itself*

Disc Layout



Disc Layout

The meaning of ‘reasonable’

```
traverse_nodes('player_[abc].Proto')
traverse_nodes('.TTree')     # tech tree
output_pack('Char')

traverse_nodes('.Ctsn')
output_pack('Ctsn')

traverse_nodes('levels/', by_spacefill)
output_pack('Level')

traverse_nodes('.Missn', by_mission)
output_pack('Mission')
```

Wrap-up

Wrapping it up

- Our asset system, metadata extraction
- How to knit metadata into a ‘refgraph’
- Pervasive use of ‘clumps’ for asset loading, preloading, unloading, data packing
- Special-case systems
- Optimization of inner I/O system

Q̈uestions?

dubois@doublefine.com
henry@doublefine.com

Fill out forms!