# Data Orientation

POD

No classes, locks, pointers

Entity Component System

Table based

cache efficient

* Table is in principle read only for event handlers
* compressed table can be traversed from top to bottom cache efficient, no gaps, use a dictionary as one level of indirection for writeable tables OR with gaps after delete OR read only table w/o dictionary
* Queue OR locked inserts, updates and deletes
* Queued changes will be done lock free in a special table thread after event handling

Concurrent read/writes

1. Either have a prev. read-only state, and a future state with only one writer thread (does this work for all cases??). This can be done when including user callbacks.
2. Or ensure that certain tables are currently read-only or have only one writer, like when updating the world matrices, this is done by the engine!
3. Techniques: atomic\_shared\_ptr, hazard pointers, read-copy-update (rcu) done for deleting, not updating!

First copy state for all writable tables, then burn through event handlers and physics.

Thread Pool threads must have the following FIFO queues:

1. Local queue: everything with fixed thread assignment goes through this LIFO queue, the thread takes out items, create calls are served immediately, update/polling calls are handed to the update queue.
2. Update queue: local, FIFO, fixed thread assignment, is processed only if the local queue is empty.
3. Work stealing queue: LIFO queue, can be appended directly

OR create a transaction log for each table that is then executed

OR use **thread pool dependencies** to first create, then update -> record a create and update chain

OR use locks to lock a set of tables

Create and delete entities:

1. First create entity in entity table, get handle, mark as in\_construction
2. Either schedule other create functions with promise/future, they poll until the handle is ready
3. OR schedule a dependency waiting for the creation and then create the other entries
4. Finally set entity state to created
5. If deleting set entity state to deleted, finally remove deleted entities
6. Make table delete relations, so that deletes are propagated

Table directory can also be just a std::unordered\_map mapping the primary key to the table index

For sorting use either a tuple, or an int-array, or struct inheriting from tuple <https://www.youtube.com/watch?v=9-rliekgoAk>

Single thread vs multithread versions

TPCHILD( MyFunction(a,b,c,d) );

* Single thread: simply define addChild to call the function
* Multithreaded: MyFunction returns a std::bind function object, and maybe also a thread ID

Makros: TPCHILD, TPSPAWN, TPDEPEND

Events through tables, direct copy of GPU,

Data Streaming on tables or queries, and use filters, transforms, generators

Synchronization by thread association – can be stored in the handle, an association list, or calculate through Counter, ObjectIdx, etc

Delay deletion by marking objects as deleted

Handles: {Counter, PoolIdx, union {ObjectIdx16, OBI16, OI32}} or {Counter, Pointer} or Counter

Skip lists, tiled with pointers, tiled compressed

Hierarchical LOD

Components in tables, specialization

Use asserts, do not use exceptions

Thread pool: 3 queues: LIFO work stealing, LIFO local for associated tasks, FIFO local for polling

Search index for component table: handle is an int, hash into a first table, provides index of first hash item in another table. There the items with the same hash are in a linked list.

Simple descriptor binds

for each view {

bind global resourcees // set 0

for each shader {

bind shader pipeline

for each material {

bind material resources // sets 2,3

}

}

}

#typedef HANDLE uint32\_t

std::array<int32\_t> hashed\_index[512]; //use hash(HANDLE) to get index of index list, if -1 then empty

struct component\_t {

HANDLE handle; //handle of the entity

int32\_t prev\_hdl, next\_hdl; //if -1 then no predecessor or successor

component data…

};

std::array<component\_t> component\_table[512];

Descriptors: Bindless, frequency based, one set per shader stage, slot based (one set per descriptor type like texture, sampler, buffers, …), object type based (one set per object type like light, camera, shadow maps, object maps, …)

# Features

Shadows: ray traced, cube map, Dual Paraboloid Shadow Mapping (DPSM)

Multiple Shadow Maps with Geometric Shader, static shadow and light maps for static light/geometry

Billboards, particle effects through geometric shader, fire, rain, snow

Tesselation of terrain and objects

Cube maps

Light probes

glTF 2.0, PBR modular, unified shader

Bone animation

Videos as textures, stream game graphics as videos, screenshots

Frustum culling, Occlusion culling, Light culling (Bounding box, or BB with scissor test/depth test)

Earth athmosphere, clouds

Vulkan queries, visibility queries

Post processing

* Lens flare, depth of field, vintage, HUD, Blood Indicator

GUI: Nuklear

Measure events, timing, threads, show in diagrams

Data driven rendering

Updates of trees with dirty bits

Mirror, Stencil Buffer

Water, normal map blending and animation

Transparent Glass

Grass, Trees, Wind, Debris

Parallax, horizon, occlusion mapping

# Renderers

Simple Forward

* Must run on old Macs, Intel integrated HW
* Frequency based descriptor sets
* Sequential shadow maps

Optimal Forward

* Differential recording or Full parallel recording
* DrawIndexedIndirect
* shader parameters
* Bindless descriptor sets through arrays/buckets, use Push descriptor sets for updating buckets
* Multiple shadow maps with geometry shader
* Ray traced shadows?

Deferred

Ray Traced – RTX

Ray Traced – Compute Shader

References

C++ map: <http://jguegant.github.io/blogs/tech/performing-try-emplace.html>

ECS: <https://gamedev.stackexchange.com/questions/31473/what-is-the-role-of-systems-in-a-component-based-entity-architecture/31491>

Edya Ladan-MozesNir ShavitNir Shavit, An optimistic approach to lock-free FIFO queues, February 2008Distributed Computing 20(5):323-341, DOI: 10.1007/s00446-007-0050-0, <https://www.researchgate.net/publication/225566366_An_optimistic_approach_to_lock-free_FIFO_queues>

<http://kylehalladay.com/blog/tutorial/vulkan/2018/01/28/Textue-Arrays-Vulkan.html>

Data-Oriented Design Richard Fabian October 8, 2018 , Fabian, Richard. Data-oriented design: software engineering for limited resources and short schedules (S.1). Richard Fabian. Kindle-Version.

* Tables, existential design

Writing an Efficient Vulkan Renderer Arseny Kapoulkine , in Engel, Wolfgang. GPU Zen 2: Advanced Rendering Techniques (S.227). Black Cat Publishing. Kindle-Version.

Slot based descriptor sets, Frequency based descriptor sets, Bindless descriptor sets, limits on uniform / storage buffers, object based descriptor sets

Full screen quad without buffers <https://www.saschawillems.de/blog/2016/08/13/vulkan-tutorial-on-rendering-a-fullscreen-quad-without-buffers/>

Halladay Blog

<http://kylehalladay.com/blog/tutorial/2017/11/27/Vulkan-Material-System.html>

<http://kylehalladay.com/blog/tutorial/vulkan/2018/02/05/Bind-Once-Uniform-Data-Vulkan.html>

INTEL TBB concurrent map

<https://www.threadingbuildingblocks.org/docs/help/reference/containers_overview/concurrent_unordered_map_cls.html>

Performance map vs unordered\_map : <http://supercomputingblog.com/windows/ordered-map-vs-unordered-map-a-performance-study/>