OGRE

Pittfals & Design proposal for Ogre 2.0

Matías Nazareth Goldberg, 2012

Ogre forum user: dark_sylinc

Pitfalls

- Too many cache misses :(
- Inefficient Scene traversal & processing
- Fat, unflexible, vertex format
- Fixed functions vs programmable shaders
  - “setFog”, etc

- Too many cache misses :(
- Inefficient Scene traversal & processing
- Fat, unflexible, vertex format
- Fixed functions vs programmable shaders
  - “setFog”, etc

Cache misses

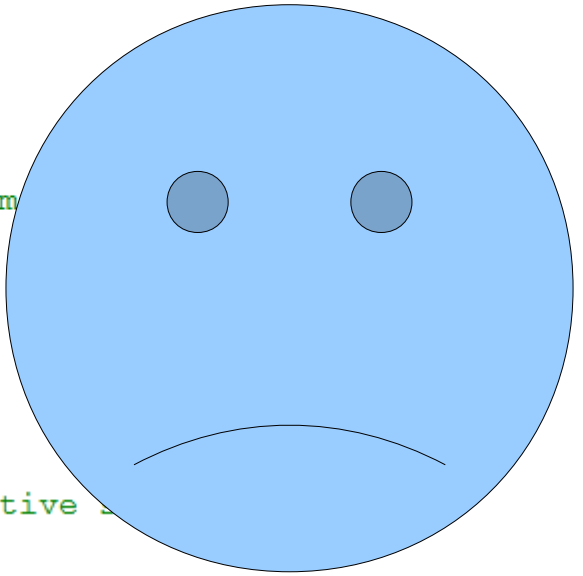
- Caches matter. A LOT.
  - Sergey Solyanik (from Microsoft):

“Linux was routing packets at ~30Mbps [wired], and wireless at ~20. Windows CE was crawling at barely 12Mbps wired and 6Mbps wireless. ...

We found out Windows CE had a LOT more instruction cache misses than Linux. ...

After we changed the routing algorithm to be more cache-local, we started doing 35MBps [wired], and 25MBps wireless - 20% better than Linux.” [1]

```
bool Frustum::isVisible(const AxisAlignedBox& bound, FrustumPlane* culledBy) const
{
    // Null boxes always invisible
    if (bound.isNull()) return false;

    // Infinite boxes always visible
    if (bound.isInfinite()) return true;

    // Make any pending updates to the calculated frustum
    updateFrustumPlanes();

    // Get centre of the box
    Vector3 centre = bound.getCenter();
    // Get the half-size of the box
    Vector3 halfSize = bound.getHalfSize();

    // For each plane, see if all points are on the negative
    // If so, object is not visible
    for (int plane = 0; plane < 6; ++plane)
    {
        // Skip far plane if infinite view frustum
        if (plane == FRUSTUM_PLANE_FAR && mFarDist == 0)
            continue;

        Plane::Side side = mFrustumPlanes[plane].getSide(centre, halfSize);
        if (side == Plane::NEGATIVE_SIDE)
        {
            // ALL corners on negative side therefore out of view
            if (culledBy)
                *culledBy = (FrustumPlane)plane;
            return false;
        }

    }

    return true;
}
```



Arghhh!!!!

MY EYES!!!

```
bool Frustum::isVisible(const AxisAlignedBox& bound, FrustumPlane* culledBy) const
{
    // Null boxes always invisible
    if (bound.isNull()) return false;

    // Infinite boxes always visible
    if (bound.isInfinite()) return true;

    // Make any pending updates to the calculated frustum planes
    updateFrustumPlanes();

    // Get centre of the box
    Vector3 centre = bound.getCenter();
    // Get the half-size of the box
    Vector3 halfSize = bound.getHalfSize();

    // For each plane, see if all points are on the negative side
    // If so, object is not visible
    for (int plane = 0; plane < 6; ++plane)
    {
        // Skip far plane if infinite view frustum
        if (plane == FRUSTUM_PLANE_FAR && mFarDist == 0)
            continue;

        Plane::Side side = mFrustumPlanes[plane].getSide(centre, halfSize);
        if (side == Plane::NEGATIVE_SIDE)
        {
            // ALL corners on negative side therefore out of view
            if (culledBy)
                *culledBy = (FrustumPlane)plane;
            return false;
        }

    }

    return true;
}
```

CACHE MISS / LHS

CACHE MISS / LHS

X6 CACHE MISS / LHS

X6 CACHE MISS / LHS

See “Typical C++ Bullshit” by @mike_acton [2]

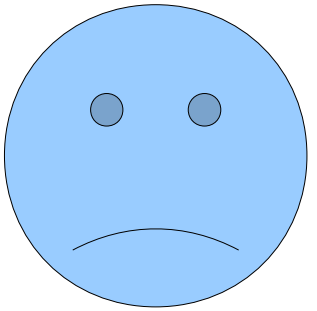
See “Culling the Battlefield” by Daniel Colling (DICE)[3]

Frostbyte* 2 uses SoA (structure of arrays), SIMD and conditional moves to optimize this routine

Cache misses, cache misses everywhere...

```
const Vector3 & Node::_getDerivedPosition(void) const
{
    if (mNeedParentUpdate)
    {
        _updateFromParent();
    }
    return mDerivedPosition;
}
```

This all over Ogre code.



- Made sense in 2000, where CPUs were ALU bounds.
  - But ALU growth is much higher than memory latency & bandwidth!

- See "Pitfalls of Object Oriented Programming" [4], by SCEE
  - ✔ x86 & x64 CPUs have branch predictors to alleviate the problem
    - ✗ But Ogre is expanding to mobile android & iPhone devices! :(
    - ✗ And consoles too! :(

- Good

```
const Vector3 & Node::_getDerivedPosition(void) const
{
        _updateFromParent();
        return mDerivedPosition;
}
```

✔ Better

```
const Vector3 & Node::_getDerivedPosition(void) const
{
#ifdef _DEBUG
        assert( !m_needParentUpdate ); //m_needParentUpdate is only present in debug
#endif
        return mDerivedPosition;
}

//Function for newbies
const Vector3 & Node::_getDerivedPositionUpdated(void) const
{
        _updateFromParent();
        return mDerivedPosition;
}
```

- Good

```
const Vector3 & Node::_getDerivedPosition(void) const
{
        _updateFromParent();
        return mDerivedPosition;
}
```

✔ Better

```
const Vector3 & Node::_getDerivedPosition(void) const
{
#ifdef _DEBUG
        assert( !m_needParentUpdate ); //m_needParentUpdate is only present in debug
#endif
        return mDerivedPosition;
}

//Function for newbies
const Vector3 & Node::_getDerivedPositionUpdated(void) const
{
        _updateFromParent();
        return mDerivedPosition;
}
```

A good game and/or render engine design would leave updating the derived position to a centralized position, and should be done **once per frame** only.

- There are ocassions when updating more than once per frame is unavoidable → call _updateFromParent manually or _getDerivedPositionUpdated instead.
  - If you really need to do this, a good programmer should be aware of what's happening internally in it's engine anyway.

```
void Node::updateFromParentImpl(void) const
{
    if (mParent)
    {
        // Update orientation
        const Quaternion& parentOrientation = mParent->_getDerivedOrientation();
        if (mInheritOrientation)
        {
            // Combine orientation with that of parent
            mDerivedOrientation = parentOrientation * mOrientation;
        }
        else
        {
            // No inheritance
            mDerivedOrientation = mOrientation;
        }

        // Update scale
        const Vector3& parentScale = mParent->_getDerivedScale();
        if (mInheritScale)
        {
            // Scale own position by parent scale, NB just combine
            // as equivalent axes, no shearing
            mDerivedScale = parentScale * mScale;
        }
        else
        {
            // No inheritance
            mDerivedScale = mScale;
        }

        // Change position vector based on parent's orientation & scale
        mDerivedPosition = parentOrientation * (parentScale * mPosition);

        // Add altered position vector to parents
        mDerivedPosition += mParent->_getDerivedPosition();
    }
    else
    {
        // Root node, no parent
```

Cache miss

Pipeline stall

Pipeline stall

Pipeline stalls

- Floating point operations & branches don't go together.
- Use branch-less conditional moves.
  - ***fsel*** in PPC
  - ***fcmov*** in x87 FPU
  - Conditional move just introduced in SSE5. → But can be performed with compare & mask instructions!
- See “Down With fcmp: Conditional Moves For Branchless Math”, by Elan Rusky (Valve) [5]

Pipeline stalls – conditional moves

Now, this is more like it:

```
void Node::updateFromParentImpl(void) const
{
    // Update orientation
    const Quaternion& parentOrientation = mParent->_getDerivedOrientation();
    parentOrientation = fsel( mInheritOrientation, parentOrientation * mOrientation, mOrientation );
    // Update scale
    const Vector3& parentScale = mParent->_getDerivedScale();
    // Scale own position by parent scale, NB just combine as equivalent axes, no shearing
    mDerivedScale = fsel( mInheritScale, parentScale * mScale, mScale );
    mDerivedPosition = parentOrientation * (parentScale * mPosition);
    mDerivedPosition += mParent->_getDerivedPosition();
}
```

“fsel” function uses fsel/fcmov/sse. Depending on the architecture. Ensure by looking at assembly it inlines as desired.

Conditional moves with SSE2

- Naive approach:
  - Create a mask → mask = cmp(condition1, condition2);
  - AND & NOT both args → t1 = arg1 & mask;
    → t2 = arg2 & ~mask;
  - OR both results → r = t1 | t2

Total: 4 instructions! :)

Conditional moves with SSE2

- Naive approach:
  - Create a mask → mask = cmp(condition1, condition2);
  - AND & NOT both args → t1 = arg1 & mask;
    → t2 = arg2 & ~mask;
  - OR both results → r = t1 | t2

Total: 4 instructions! :)

Internally, some SSE architectures **flag** xmm registers as containing integer or floating point data. Using integer operations (bitwise logic) on floating point xmm registers will incur a performance penalty (flagging the register as integer, then flagging it back as float when used again).

That's why MOVAPS (floats) appears to do the same as MOVDQA (ints).

Conditional moves with SSE2

✔ Smart approach:

- Create a mask → mask = cmp(condition1, condition2);
- Sub arg1 & arg2 → t = arg2 - arg1
- AND temporary 't' → t = t & mask;
- Add masked t to arg1 → r = arg1 + t;

Total: 4 instructions! ;)

Conditional moves with SSE2

✔ Smart approach:

- Create a mask → mask = cmp(condition1, condition2);
- Sub arg1 & arg2 → t = arg2 - arg1
- AND temporary 't' → t = t & mask;
- Add masked t to arg1 → r = arg1 + t;

Total: 4 instructions! ;)

Addition & substraction are trivial operations. The instruction count is the same (cycle count might vary though) but in this approach, **only one** register is flagged from float to int then back to float; **as opposed to the naive approach, which flagged both registers.**

Won't work if arg1 has nan or infs! (use assert & keep naive approach too)

“Premature optimization is root of all evil”

“Premature optimization is root of all evil”

BULLSHIT!

I think of these guys everytime I see a loding screen.

“Premature optimization is root of all evil”

BULLSHIT!

I think of these guys everytime I see a loding screen.

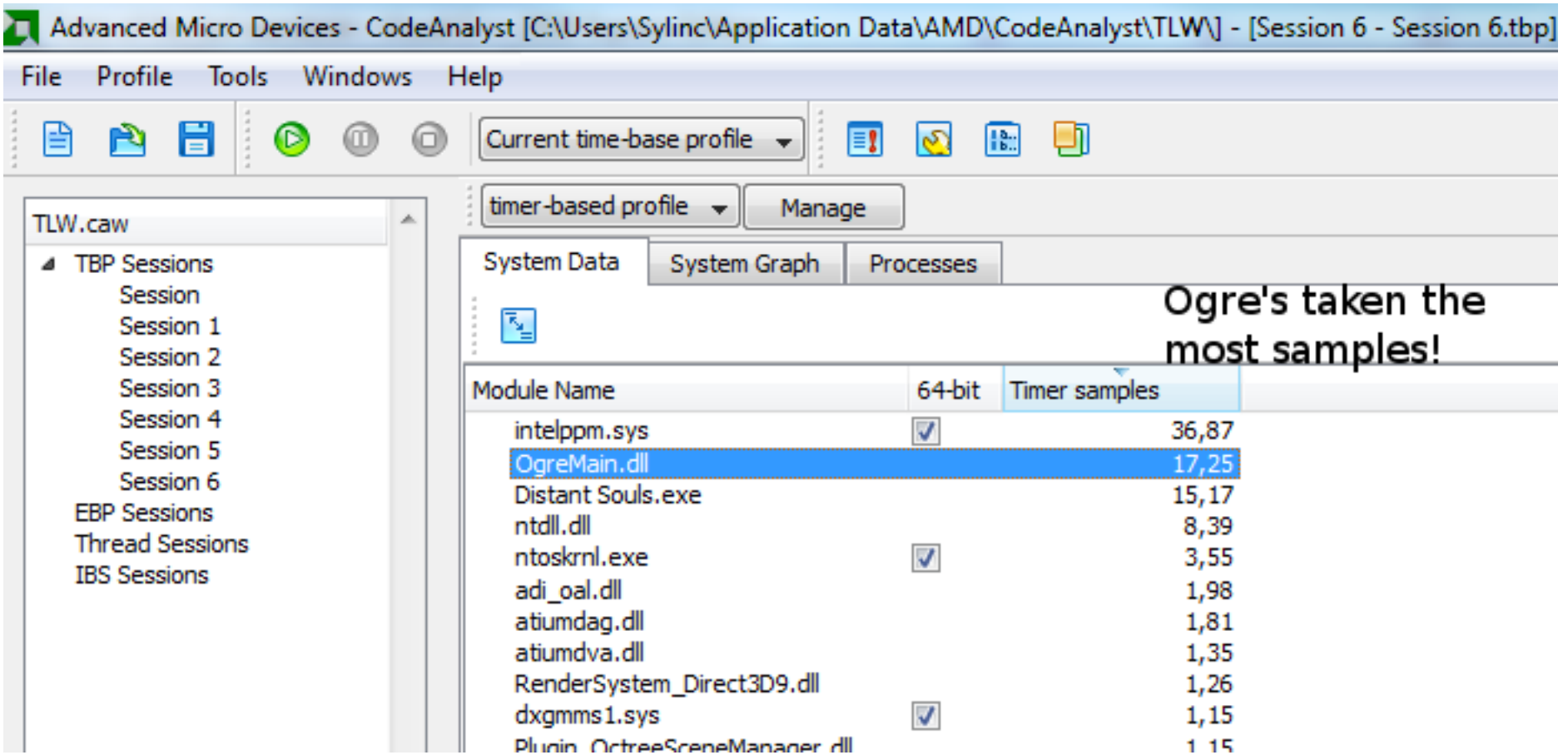
THESE ARE NOT PREMATURE NOR MICRO-OPTIMIZATIONS!!!

This is a focus on hotspots based on profiled runs of real world applications from +10-year-old code.

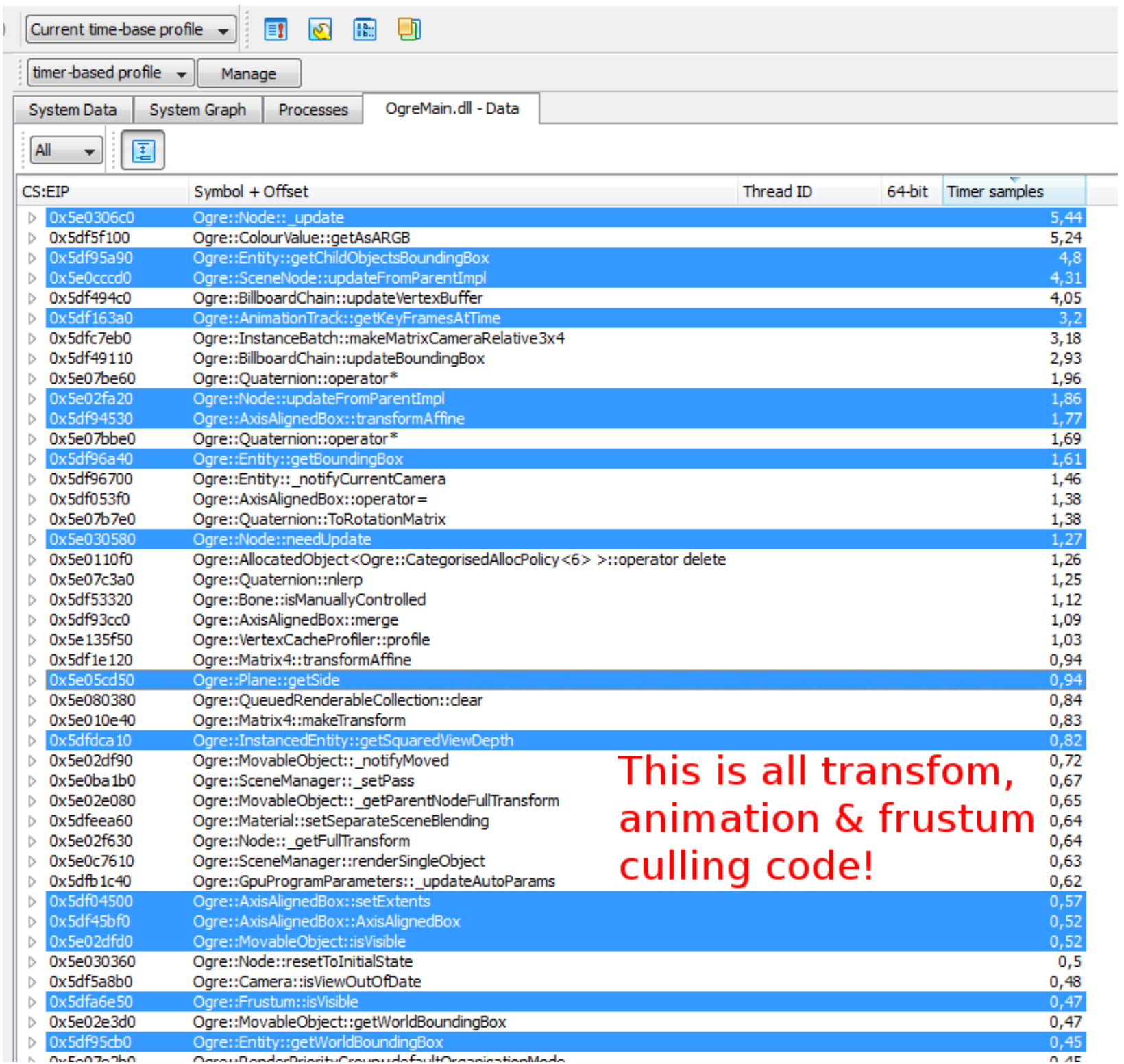
SIMD, parallel, cache-friendly algorithms are the **industry standard** today (CryEngine* 2, Frostbyte* 2 engine, Uncharted* engine)

Performance gains from applying these techniques are very real and worthwhile.

Not convinced?



By the way, inside Distant Souls.exe, 40 samples are spent inside a wait function, which is the **logic thread waiting** until the 16ms of it's fixed frame rate is over (live spin lock). The render thread runs at variable framerate.


Current time-base profile
timer-based profile
Manage
System Data
System Graph
Processes
OgreMain.dll - Data
All
CS:EIP
Symbol + Offset
Thread ID
64-bit
Timer samples
0x5e0306c0 Ogre::Node::_update 5,44
0x5df5f100 Ogre::ColourValue::getAsARGB 5,24
0x5df95a90 Ogre::Entity::getChildObjectsBoundingBox 4,8
0x5e0cccd0 Ogre::SceneNode::updateFromParentImpl 4,31
0x5df494c0 Ogre::BillboardChain::updateVertexBuffer 4,05
0x5df163a0 Ogre::AnimationTrack::getKeyFramesAtTime 3,2
0x5dfc7eb0 Ogre::InstanceBatch::makeMatrixCameraRelative3x4 3,18
0x5df49110 Ogre::BillboardChain::updateBoundingBox 2,93
0x5e07be60 Ogre::Quaternion::operator* 1,96
0x5e02fa20 Ogre::Node::updateFromParentImpl 1,86
0x5df94530 Ogre::AxisAlignedBox::transformAffine 1,77
0x5e07bbe0 Ogre::Quaternion::operator* 1,69
0x5df96a40 Ogre::Entity::getBoundingBox 1,61
0x5df96700 Ogre::Entity::_notifyCurrentCamera 1,46
0x5df053f0 Ogre::AxisAlignedBox::operator= 1,38
0x5e07b7e0 Ogre::Quaternion::ToRotationMatrix 1,38
0x5e030580 Ogre::Node::needUpdate 1,27
0x5e0110f0 Ogre::AllocatedObject<Ogre::CategorisedAllocPolicy<6> >::operator delete 1,26
0x5e07c3a0 Ogre::Quaternion::nlerp 1,25
0x5df53320 Ogre::Bone::isManuallyControlled 1,12
0x5df93cc0 Ogre::AxisAlignedBox::merge 1,09
0x5e135f50 Ogre::VertexCacheProfiler::profile 1,03
0x5df1e120 Ogre::Matrix4::transformAffine 0,94
0x5e05cd50 Ogre::Plane::getSide 0,94
0x5e080380 Ogre::QueuedRenderableCollection::clear 0,84
0x5e010e40 Ogre::Matrix4::makeTransform 0,83
0x5dfdca10 Ogre::InstancedEntity::getSquaredViewDepth 0,82
0x5e02df90 Ogre::MovableObject::_notifyMoved 0,72
0x5e0ba1b0 Ogre::SceneManager::_setPass 0,67
0x5e02e080 Ogre::MovableObject::_getParentNodeFullTransform 0,65
0x5dfeea60 Ogre::Material::setSeparateSceneBlending 0,64
0x5e02f630 Ogre::Node::_getFullTransform 0,64
0x5e0c7610 Ogre::SceneManager::renderSingleObject 0,63
0x5dfb1c40 Ogre::GpuProgramParameters::_updateAutoParams 0,62
0x5df04500 Ogre::AxisAlignedBox::setExtents 0,57
0x5df45bf0 Ogre::AxisAlignedBox::AxisAlignedBox 0,52
0x5e02dfd0 Ogre::MovableObject::isVisible 0,52
0x5e030360 Ogre::Node::resetToInitialState 0,5
0x5df5a8b0 Ogre::Camera::isViewOutOfDate 0,48
0x5dfa6e50 Ogre::Frustum::isVisible 0,47
0x5e02e3d0 Ogre::MovableObject::getWorldBoundingBox 0,47
0x5df95cb0 Ogre::Entity::getWorldBoundingBox 0,45
This is all transfom, animation & frustum culling code!

Still not convinced?

Distant Souls (Ogre)

Taken on Intel Core 2 Quad Extreme X9650 @3.0Ghz
AMD Radeon HD 7770 1 GB RAM
4 GB RAM
1280x720, No MSAA, Max Quality

Distant Souls runs on 2 threads. One thread is **exclusively for Ogre** while the other one handles the logic & physics.

Distant Souls (Ogre)

Taken on Intel Core 2 Quad Extreme X9650 @3.0Ghz
AMD Radeon HD 7770 1 GB RAM
4 GB RAM
1280x720, No MSAA, Max Quality

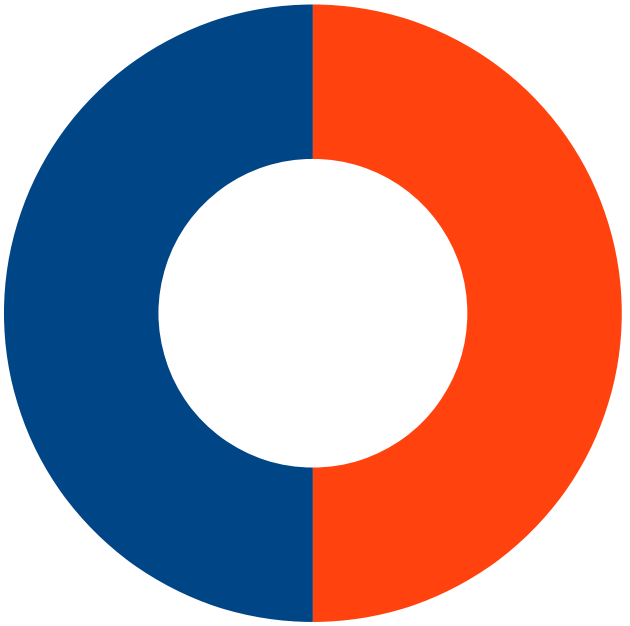
Distant Souls runs on 2 threads. One thread is **exclusively for Ogre** while the other one handles the logic & physics.

Assassin's Creed* 2 (Ubisoft*)

Taken on Intel Core 2 Quad Extreme X9650 @3.0Ghz
AMD Radeon HD 7770 1 GB RAM
4 GB RAM
1280x720. No MSAA, Max Quality

Assassin's Creed* 2 (Ubisoft*)

Taken on Intel Core 2 Quad Extreme X9650 @3.0Ghz
AMD Radeon HD 7770 1 GB RAM
4 GB RAM
1280x720. No MSAA, Max Quality



Twice performance with 3x objects....

(we're clearly doing something wrong)

Distant Souls' render thread is taking 43ms. The GPU spends a lot of idle time.

Logic & physics thread runs at steady 16ms (unlocking it yields 8ms)

Render Thread only updates all visible objects position from other thread and calls renderOneFrame

Main reasons are cache misses and complex compositor passes (next slides will talk about it's innefficiency).

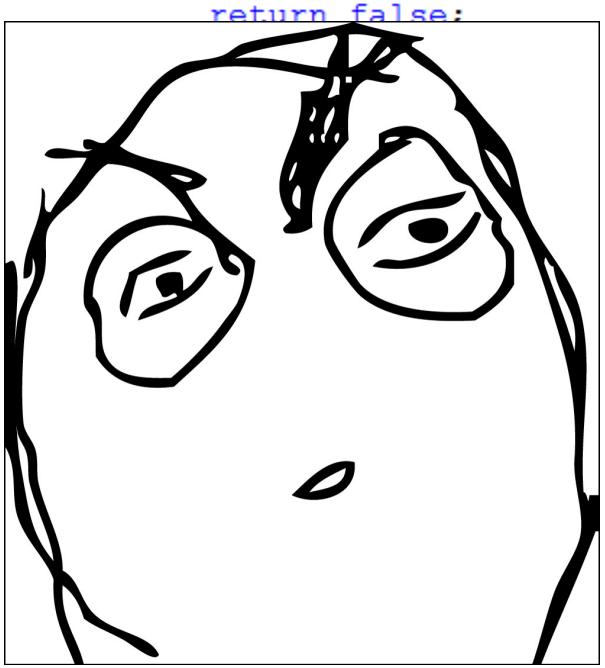
- ~~Too many cache misses :(~~
- Inefficient Scene traversal & processing
- Fat, unflexible, vertex format
- Fixed functions vs programmable shaders
  - “setFog”, etc

Poor data updates

```
bool Entity::cacheBoneMatrices(void)
{
    Root& root = Root::getSingleton();
    unsigned long currentFrameNumber = root.getNextFrameNumber();
    if ((*mFrameBonesLastUpdated != currentFrameNumber) ||
        (hasSkeleton() && getSkeleton()->getManualBonesDirty()))
    {
        if ((!mSkipAnimStateUpdates) && (*mFrameBonesLastUpdated != currentFrameNumber))
            mSkeletonInstance->setAnimationState(*mAnimationState);
        mSkeletonInstance->_getBoneMatrices(mBoneMatrices);
        *mFrameBonesLastUpdated  = currentFrameNumber;

        return true;
    }
    return false;
}
```

Poor data updates

```
bool Entity::cacheBoneMatrices(void)
{
    Root& root = Root::getSingleton();
    unsigned long currentFrameNumber = root.getNextFrameNumber();
    if ((*mFrameBonesLastUpdated != currentFrameNumber) ||
        (hasSkeleton() && getSkeleton()->getManualBonesDirty()))
    {
        if ((!mSkipAnimStateUpdates) && (*mFrameBonesLastUpdated != currentFrameNumber))
            mSkeletonInstance->setAnimationState(*mAnimationState);
        mSkeletonInstance->_getBoneMatrices(mBoneMatrices);
        *mFrameBonesLastUpdated  = currentFrameNumber;

        return true;
    }
    return false;
```



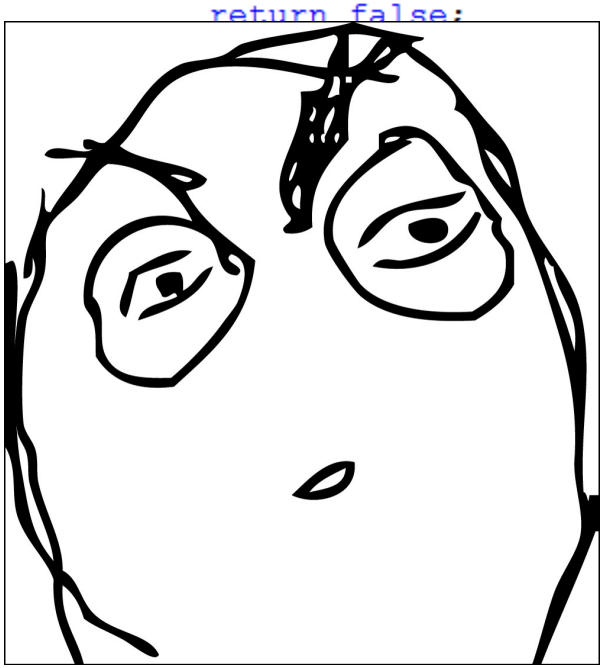
When this function returns true, Ogre will then proceed to update every bone of the skeleton.

It uses *mFrameBonesLastUpdated* to keep track of “dirty” updates according comparing against current global frame #.

This function may get called between 3 to 6 times per frame depending on effects complexity.

Poor data updates

```
bool Entity::cacheBoneMatrices(void)
{
    Root& root = Root::getSingleton();
    unsigned long currentFrameNumber = root.getNextFrameNumber();
    if ((*mFrameBonesLastUpdated != currentFrameNumber) ||
        (hasSkeleton() && getSkeleton()->getManualBonesDirty()))
    {
        if ((!mSkipAnimStateUpdates) && (*mFrameBonesLastUpdated != currentFrameNumber))
            mSkeletonInstance->setAnimationState(*mAnimationState);
        mSkeletonInstance->_getBoneMatrices(mBoneMatrices);
        *mFrameBonesLastUpdated  = currentFrameNumber;

        return true;
    }
    return false;
```



When this function returns true, Ogre will then proceed to update every bone of the skeleton.

It uses *mFrameBonesLastUpdated* to keep track of "dirty" updates according comparing against current global frame #.

THIS IS WRONG IN SO MANY LEVELS

This function may get called between 3 to 6 times per frame depending on effects complexity.

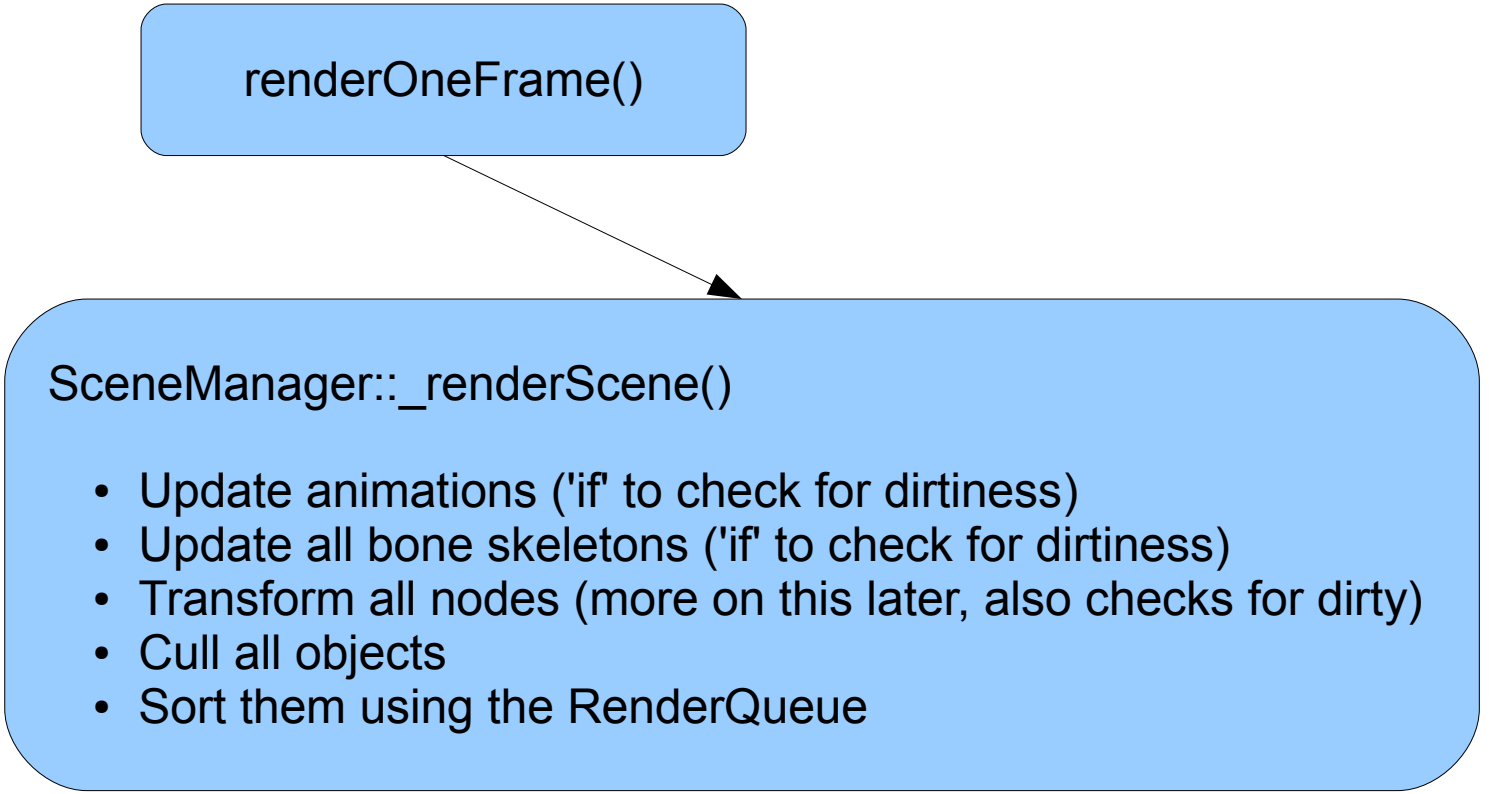
This pattern repeats itself with
mFrameAnimationLastUpdated & *mDirtyFrameNumber*

WHAT IS GOING ON???

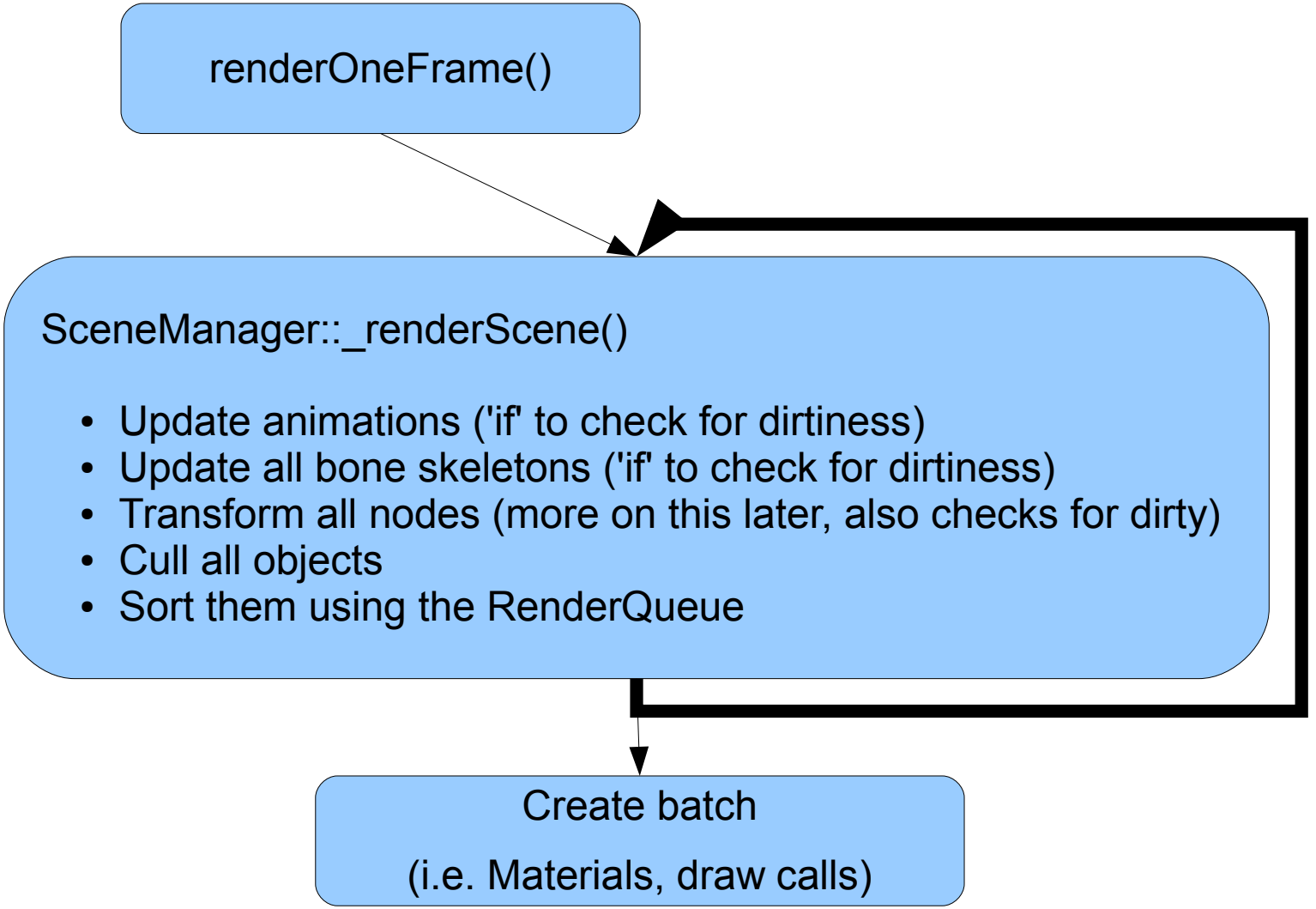
Let's take a look at the big picture:

Typical Ogre render flow

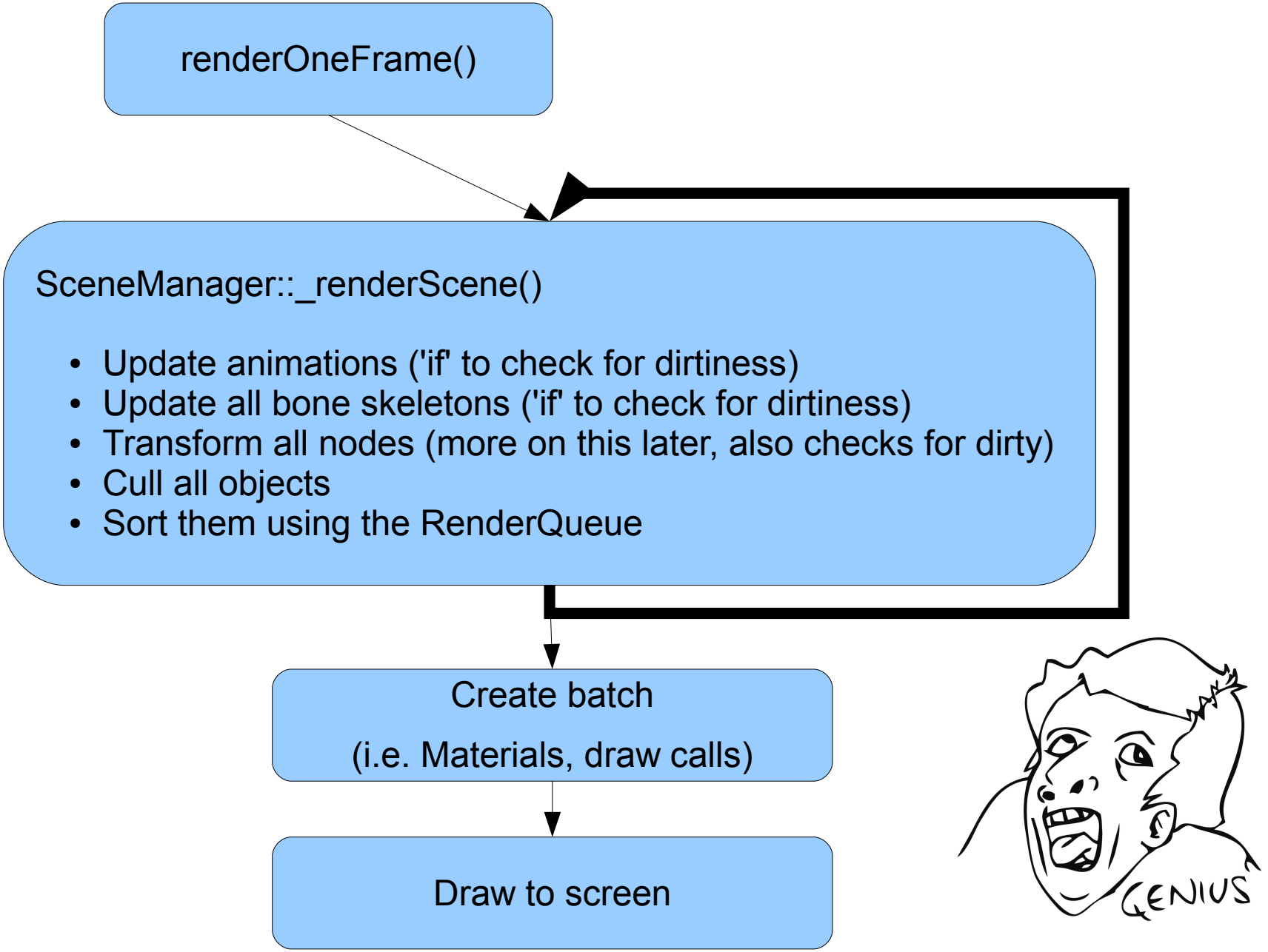
Typical Ogre render flow



Typical Ogre render flow



Typical Ogre render flow



Typical Ogre render flow

- SceneManager::_renderScene() calls itself many times.
  - Once for every shadow map
  - Once for every compositor “render_scene” pass.
- PSSM/CSM with 3 splits + a custom render_scene pass = 6 calls to itself :@

Typical Ogre render flow

- SceneManager::_renderScene() calls itself many times.
  - Once for every shadow map
  - Once for every compositor “render_scene” pass.
- PSSM/CSM with 3 splits + a custom render_scene pass = 6 calls to itself :@

✗ No reuse of culling data.

Typical Ogre render flow

- SceneManager::_renderScene() calls itself many times.
  - Once for every shadow map
  - Once for every compositor “render_scene” pass.
- PSSM/CSM with 3 splits + a custom render_scene pass = 6 calls to itself :@

✗ No reuse of culling data.

✗ Lots of unnecessary variables for tracking “dirty” states → Cache pollution & memory explosion.

Typical Ogre render flow

- SceneManager::_renderScene() calls itself many times.
  - Once for every shadow map
  - Once for every compositor “render_scene” pass.
- PSSM/CSM with 3 splits + a custom render_scene pass = 6 calls to itself :@

✗ No reuse of culling data.

✗ Lots of unnecessary variables for tracking “dirty” states → Cache pollution & memory explosion.

✗ Lots of unnecessary 'if' (cache misses, pipeline stalls)

Whats wrong with this script?

```
compositor Post/DeferredShading
{
	technique
	{
		texture_ref GBuffer DeferredShading GBuffer

		target_output
		{
			pass render_scene
			{
				first_render_queue	10
				last_render_queue	90
			}

			pass render_quad
			{
				material	SomePostProcessMaterial
				input		0	GBuffer	0
			}

			pass render_scene
			{
				first_render_queue	91
				last_render_queue	95
			}
		}
	}
}
```

Whats wrong with this script?

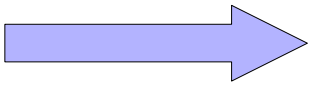
```
compositor Post/DeferredShading
{
    technique
    {
        texture_ref GBuffer DeferredShading GBuffer

        target_output
        {
            pass render_scene
            {
                first_render_queue   10
                last_render_queue    90
            }

            pass render_quad
            {
                material   SomePostProcessMaterial
                input      0     GBuffer    0
            }

            pass render_scene
            {
                first_render_queue   91
                last_render_queue    95
            }
        }
    }
}
```

The cull stage will traverse objects from 10 to 95; then before sorting in the RenderQueue, all objects between 91 & 95 will be discarded

Whats wrong with this script?

```
compositor Post/DeferredShading
{
    technique
    {
        texture_ref GBuffer DeferredShading GBuffer

        target_output
        {
            pass render_scene
            {
                first_render_queue   10
                last_render_queue    90
            }

            pass render_quad
            {
                material    SomePostProcessMaterial
                input       0     GBuffer    0
            }

            pass render_scene
            {
                first_render_queue   91
                last_render_queue    95
            }
        }
    }
}
```

The cull stage will traverse objects from 10 to 95; then before sorting in the RenderQueue, all objects between 91 & 95 will be discarded



The cull stage will traverse objects from 10 to 95 (again) and discard objects 10 to 90

Whats wrong with this script?

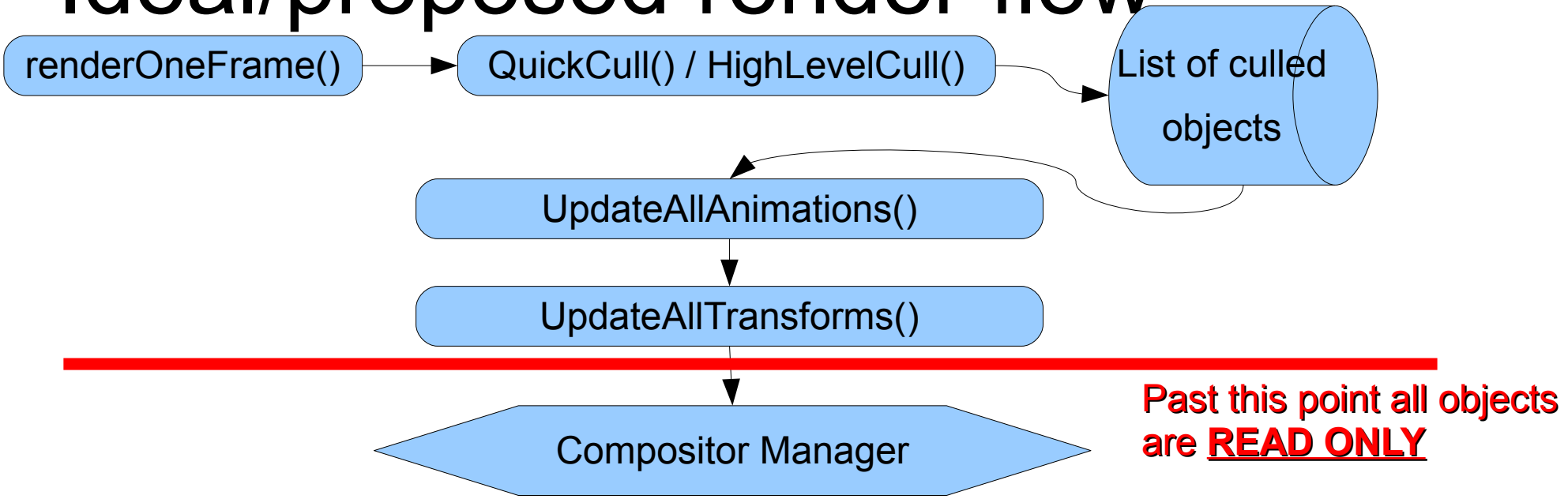
```
compositor Post/DeferredShading
{
    technique
    {
        texture_ref GBuffer DeferredShading GBuffer

        target_output
        {
            pass render_scene
            {
                first_render_queue   10
                last_render_queue    90
            }

            pass render_quad
            {
                material    SomePostProcessM
                input       0     GBuffer    0
            }

            pass render_scene
            {
                first_render_queue   91
                last_render_queue    95
            }
        }
    }
}
```

The cull stage will traverse objects from 10 to 95; then before sorting in the RenderQueue, all objects between 91 & 95 will be discarded

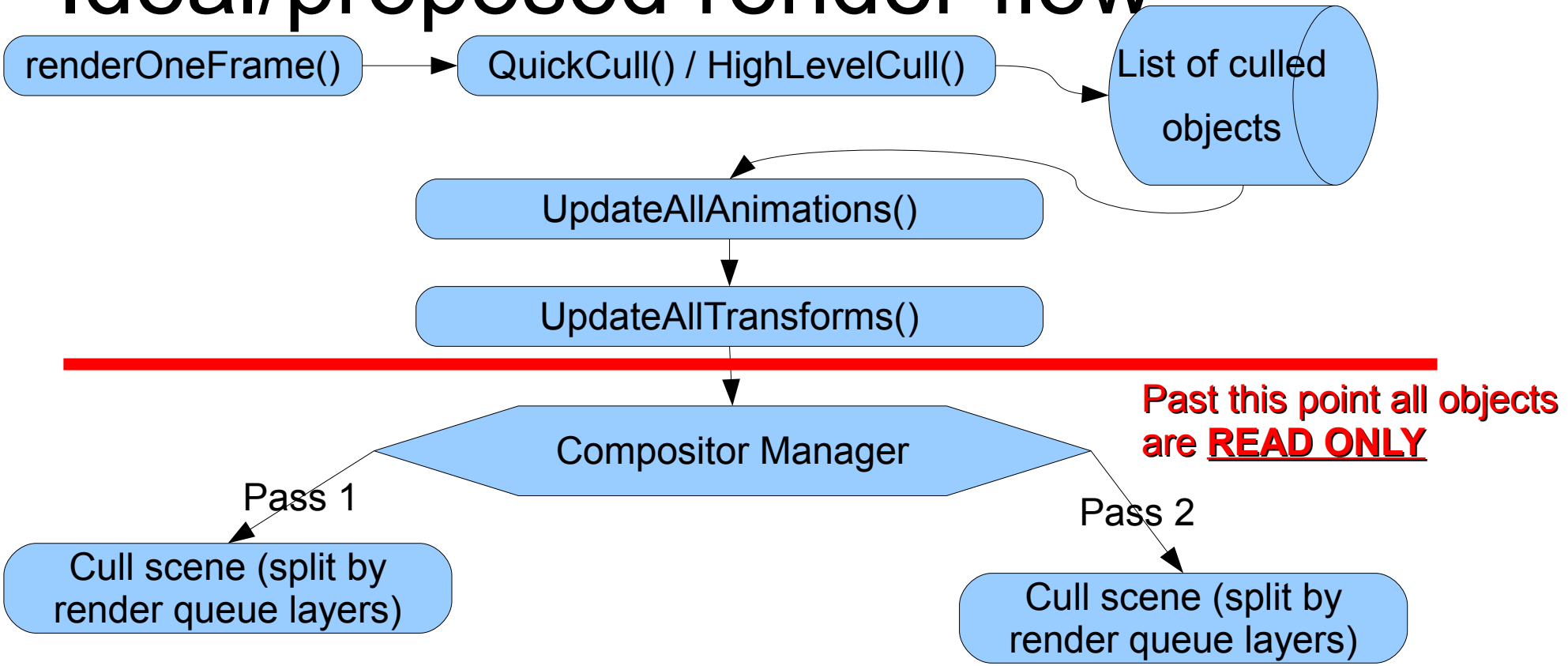
THE RENDER QUEUE IS SMART. BUT THE CULLING STAGE IS COMPLETELY INEFFICIENT

The cull stage will traverse objects from 10 to 95 (again) and discard objects 10 to 90

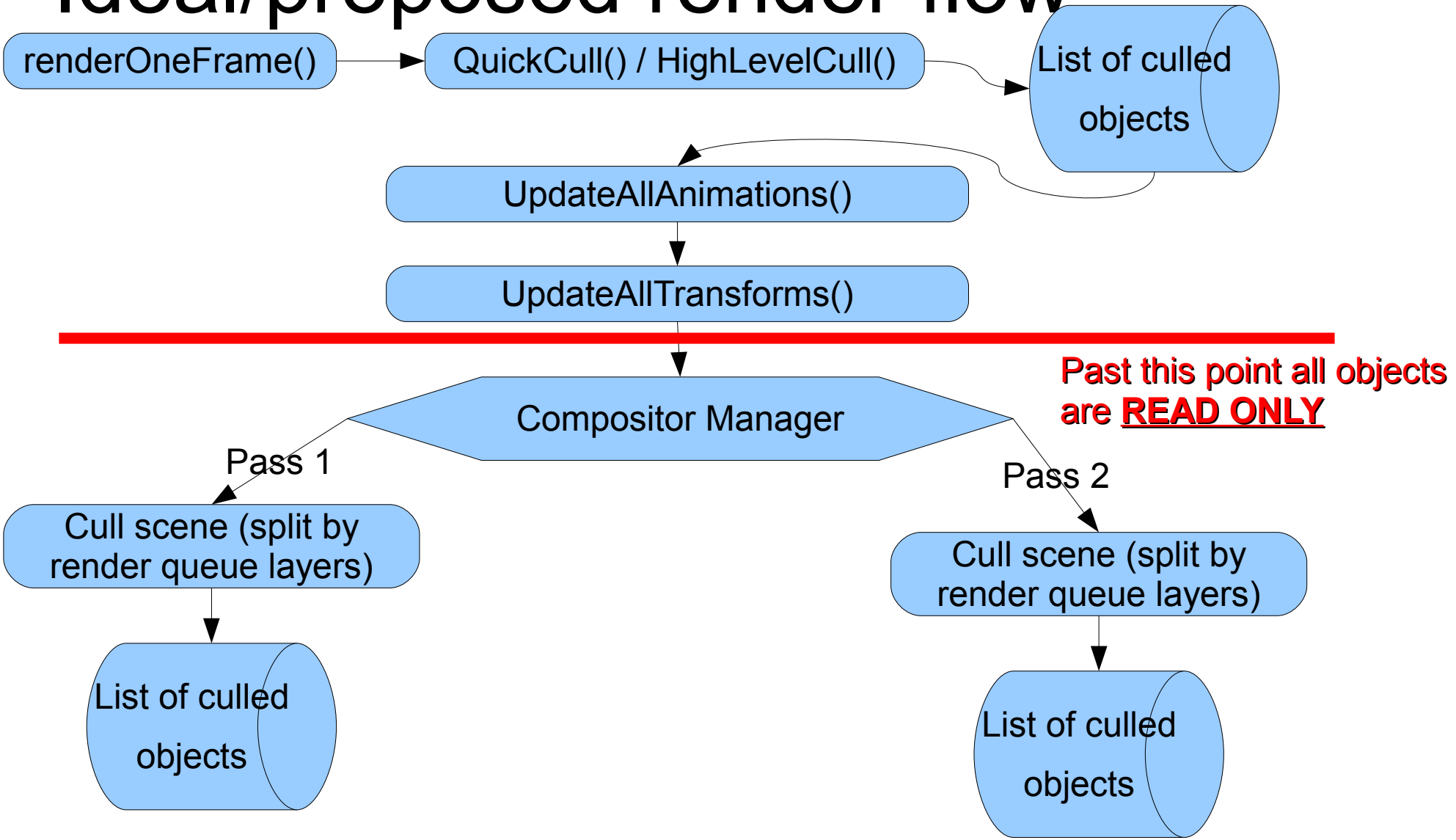
Ideal/proposed render flow



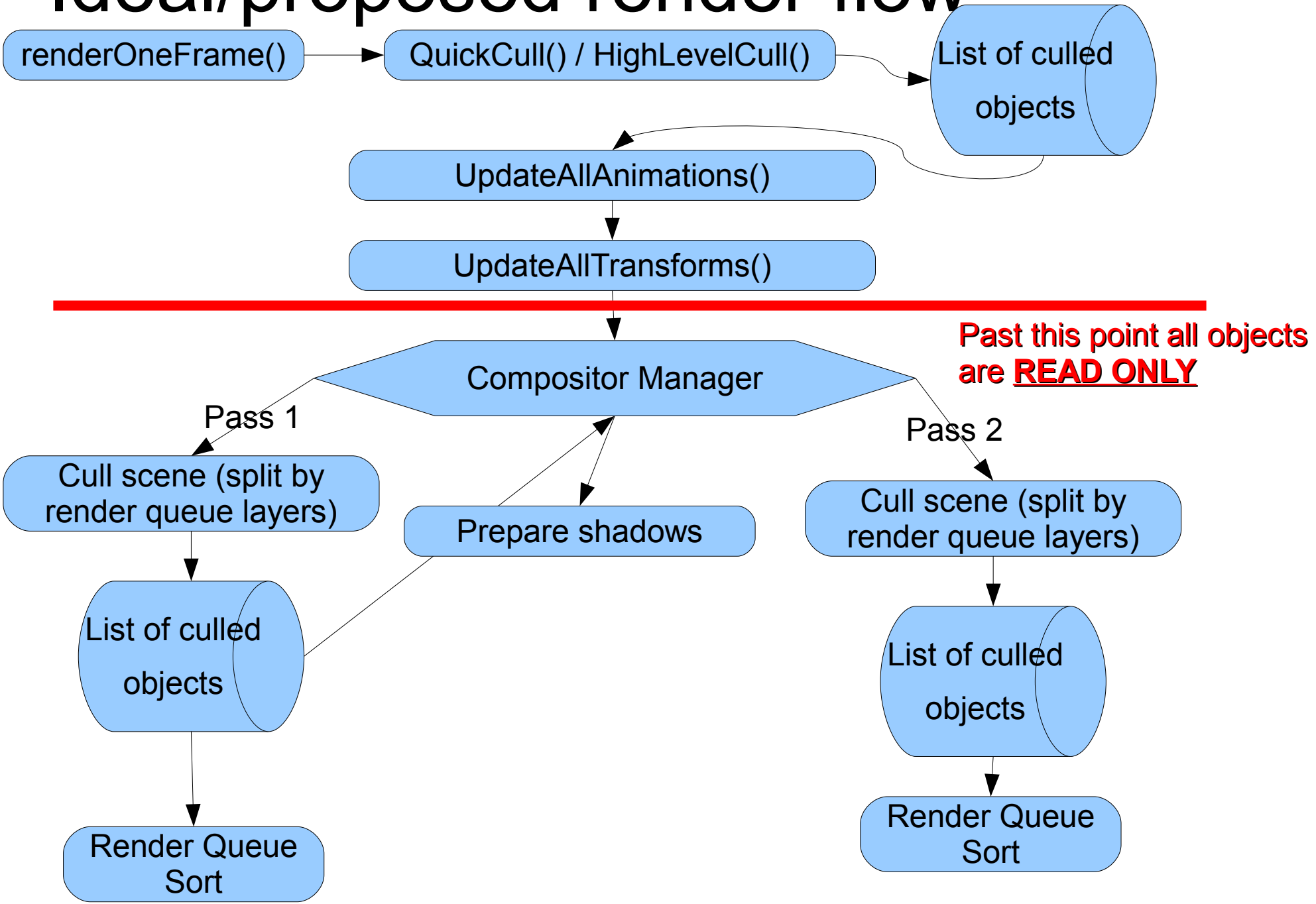
Ideal/proposed render flow



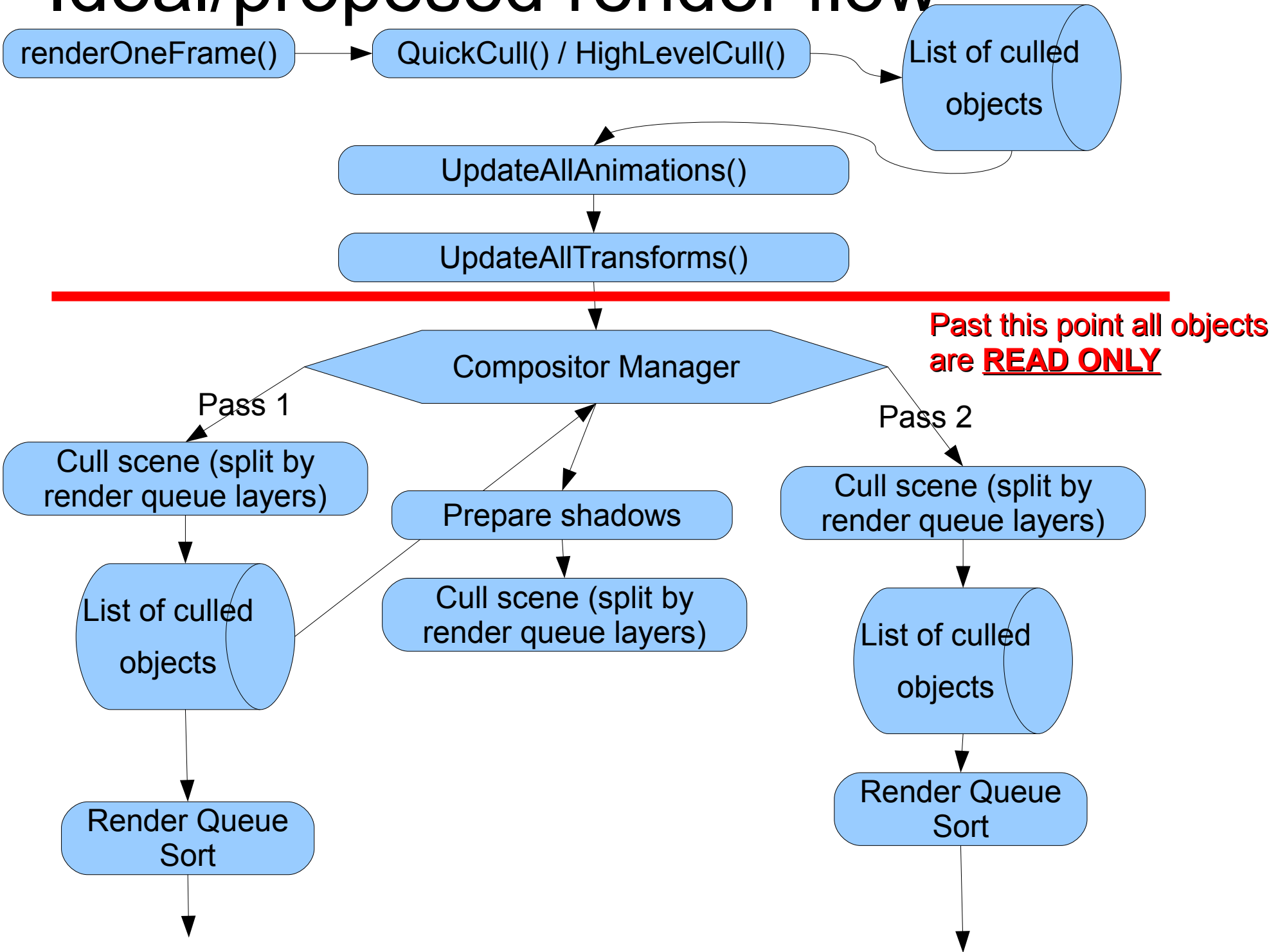
Ideal/proposed render flow



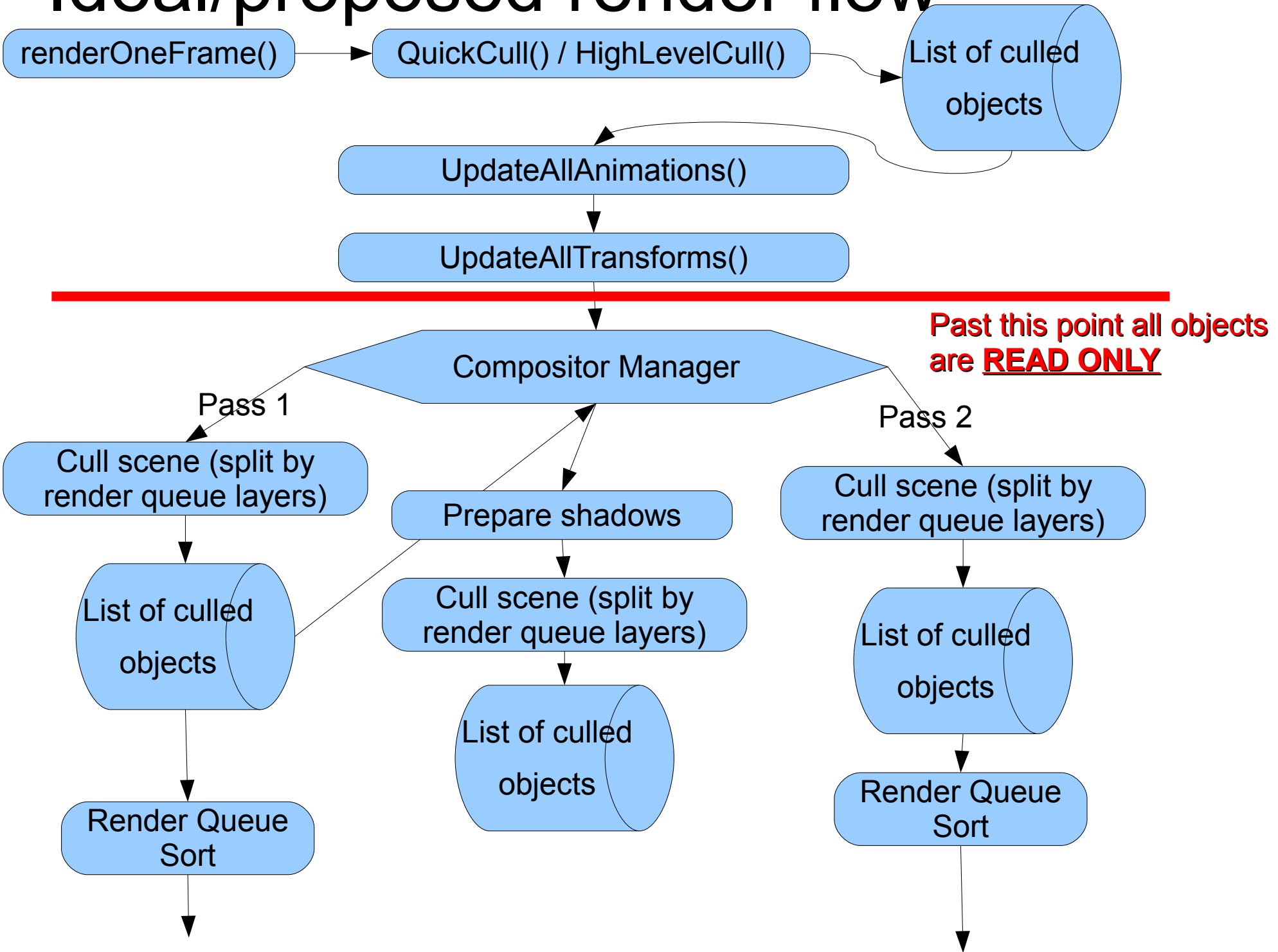
Ideal/proposed render flow



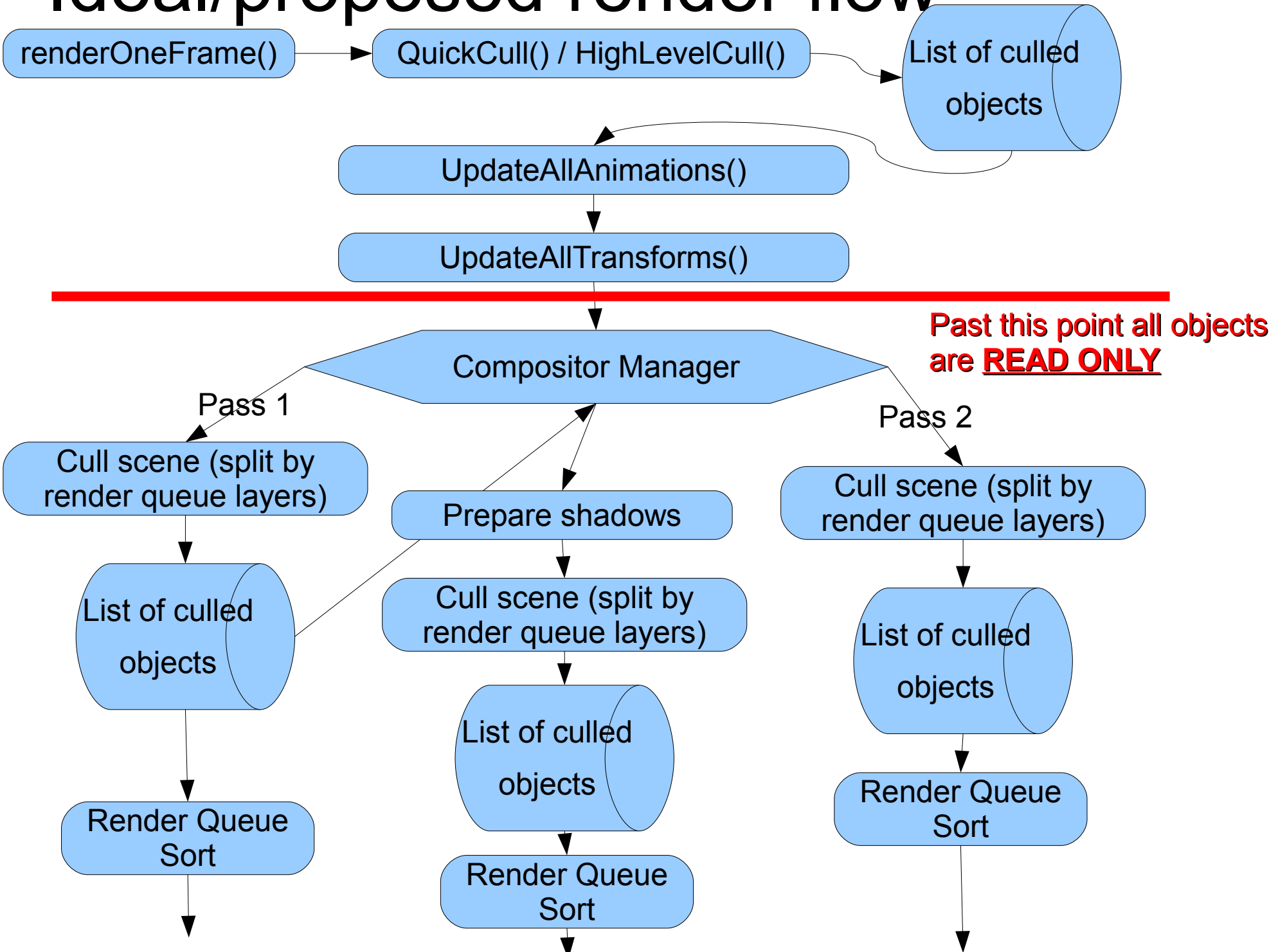
Ideal/proposed render flow



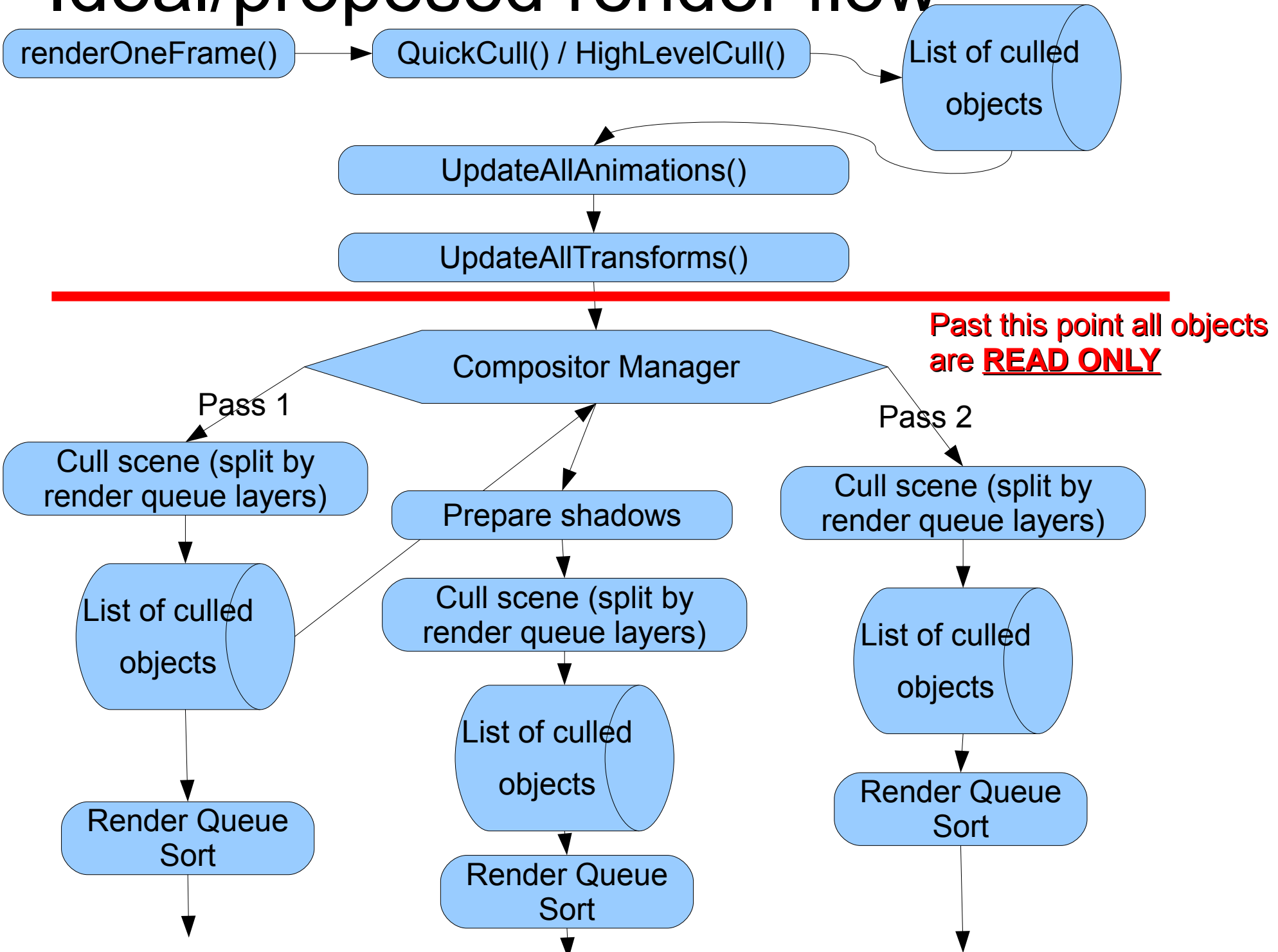
Ideal/proposed render flow

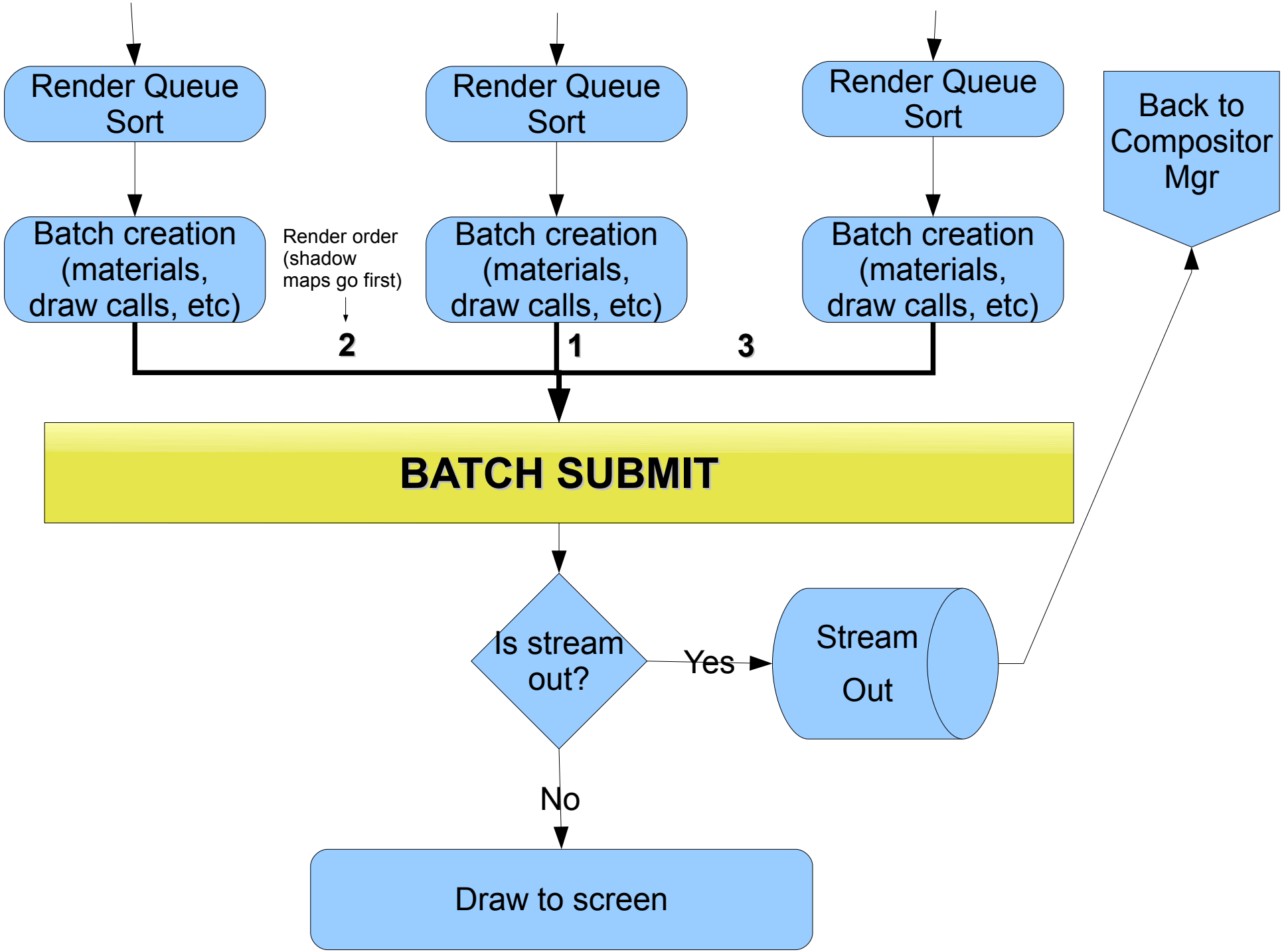


Ideal/proposed render flow



Ideal/proposed render flow




Render Queue Sort
Render Queue Sort
Render Queue Sort
Back to Compositor Mgr
Batch creation (materials, draw calls, etc)
Render order (shadow maps go first)
Batch creation (materials, draw calls, etc)
Batch creation (materials, draw calls, etc)
2
1
3
BATCH SUBMIT
Is stream out?
Yes
Stream Out
No
Draw to screen

Proposed render flow

- ✔ Update-once philosophy. Cull lists can be reused for multiple passes.
- ✔ Clear, modular roles for every component
- ✔ Multiple SIMD opportunities
- ✔ Highly threadable. “Read only” minimizes false cache sharing.
  - ✔ Processing multiple passes concurrently on the CPU → Great opportunity for environment mapping.
  - ✔ DX11 batch-threading ready
  - ✗ Needs batch creation serialization for non-threadable render systems (i.e. DX9)
- Compositor Manager is no longer an optional component. It's a core one.
  - Default compositor for those who don't need it
  - It's responsible for deciding which passes depend on other passes (race conditions, impossible scenarios i.e. Stream Out)
- ✔ HighLevelCull implementations, UpdateAllAnimations, UpdateAllTransforms can have threading of their own.

Proposed render flow

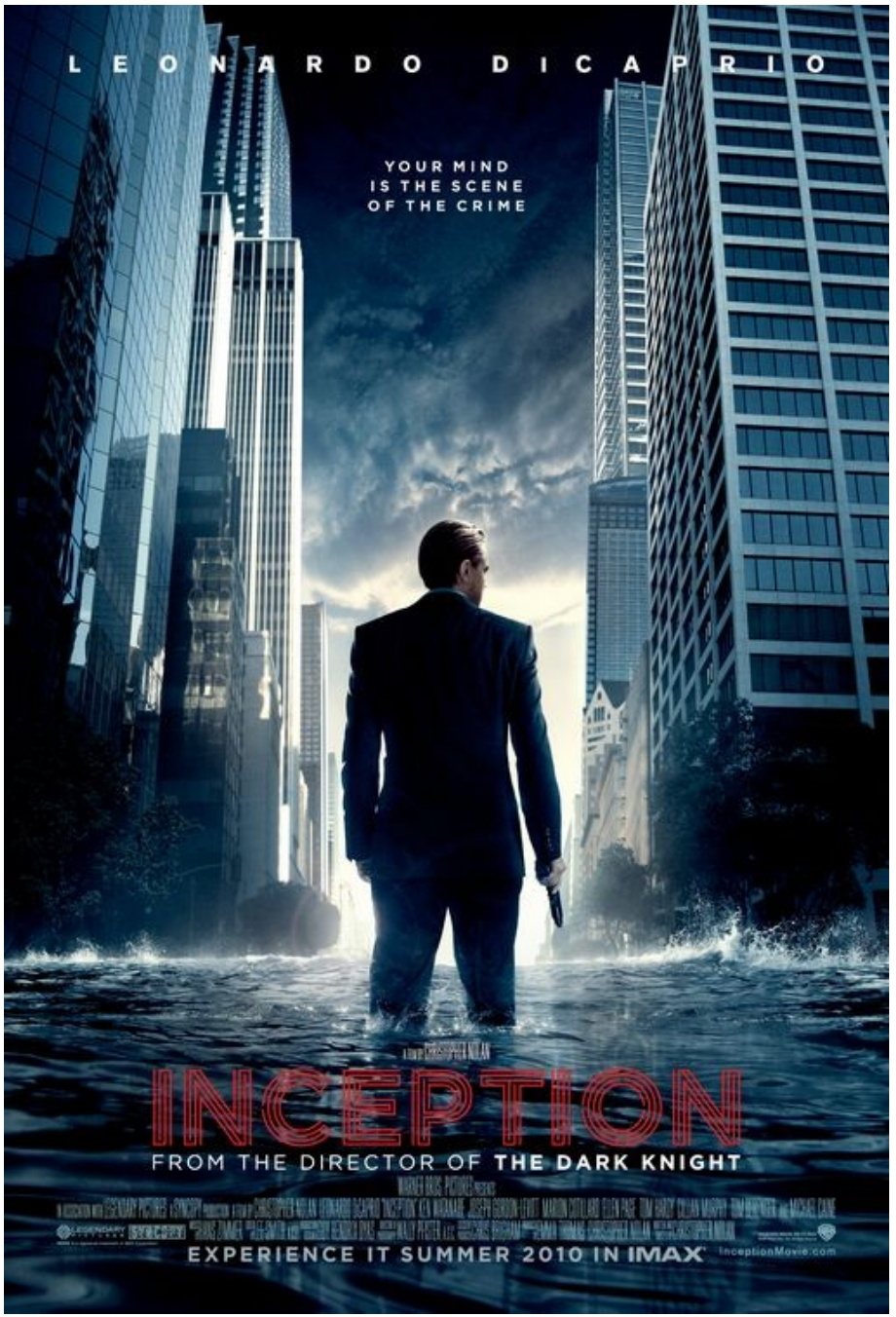
- Note: Shadows don't cull further based on the cull list from the original pass; they have to build their own cull list from scratch like any other pass. The original pass' list is used for shadow's camera frustum calculations, etc.
- Note 2: There's an optional tradeoff between processing rendering two+ passes in parallel, and reusing the cull list from a previous pass.

Proposed: HighLevelCull()

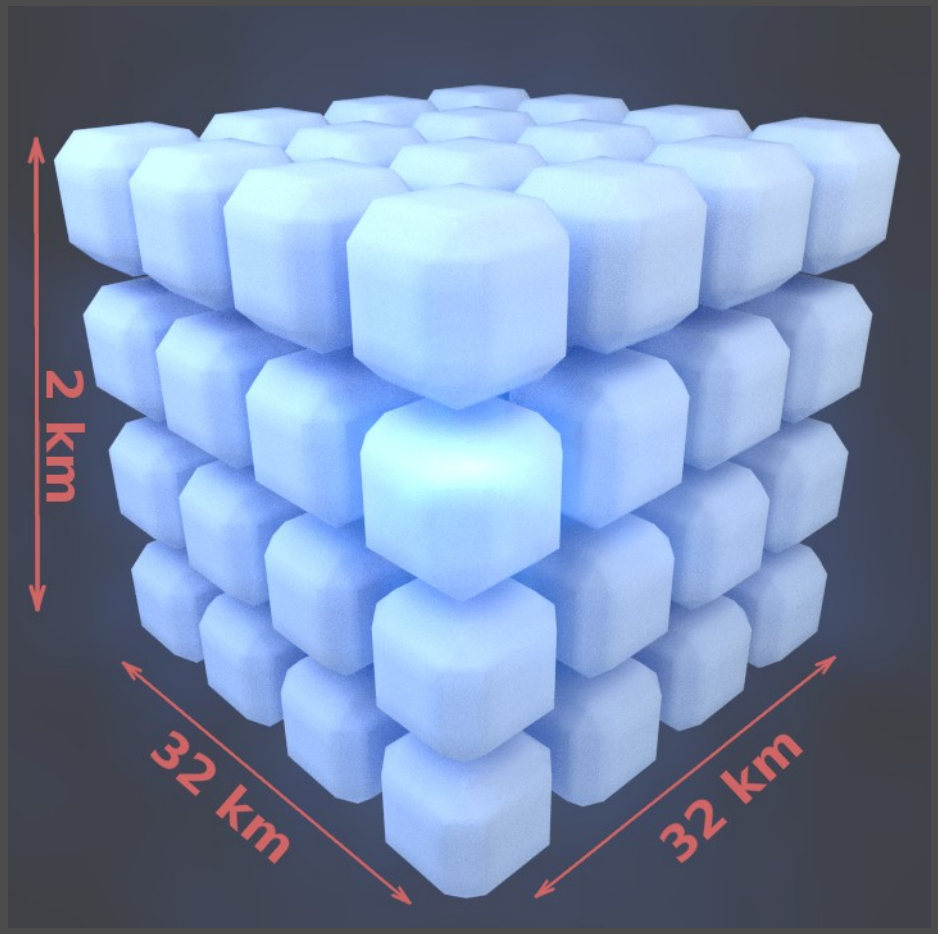
- What we currently know as “OctreeSceneManager”, “BSPSceneManager”, etc; but lighter.
- Binary trees for Graphics don't tend to scale well in multithreading due to their heterogeneous nature (sparse trees)
  - Top trend in 2000. Still useful in Physics engines & Scientific research.
  - Have horrendous cache locality unless using a custom memory allocator. See “Second Generation of Behavior Trees” by Alex J. Champandard [6] for optimal locality in tree nodes.
- Voxel grid easier to parallelize using simple pointer arithmetic (**a grid within a grid**). → Make it right or *beware of false cache sharing!*

Proposed: HighLevelCull()

- Grids within grids and Trees **with high depth have a lot of book-keeping**. Specially if we try to keep the Entities in each cell SIMD-ready and in the same memory block every time they move to another cell. A depth higher than 2 shouldn't usually be needed.
- The main purpose of HighLevelCull is to prevent updating ALL animations & transformations, while limiting the input for the next culling stage when having **vast scenes**.
  - Think Just Cause* 2 size.
  - We can skip updating objects 10-20km away.
  - ✗ Because currently, updating skeletons can be a bottleneck (depending on bone count & number of entities)



Grids within Grids



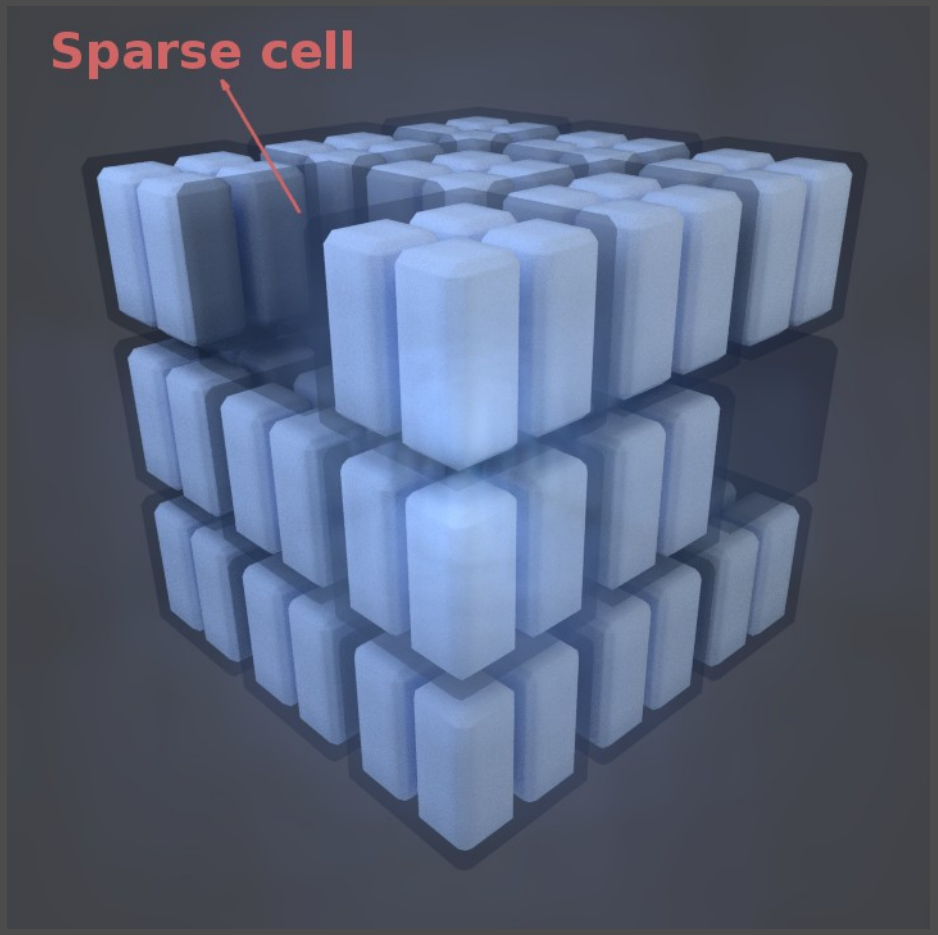
At Depth = 0 → Simple 3D Grid

Depth 0: 4x4x4

Total nodes = 64

Very fast for random access. Needs to cull 64 blocks/nodes.

32x32x2 km paremeter given at startup → Physics engines require this anyway (broadphase size)



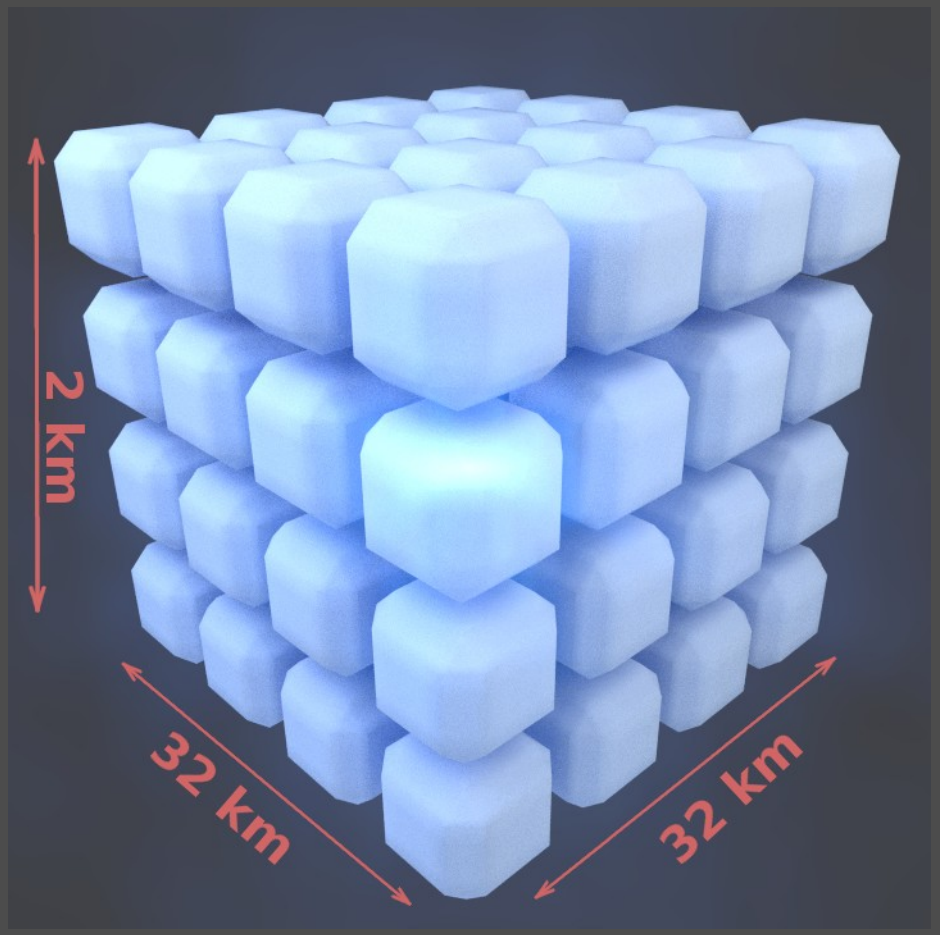
At Depth >= 1 → Grid within grid

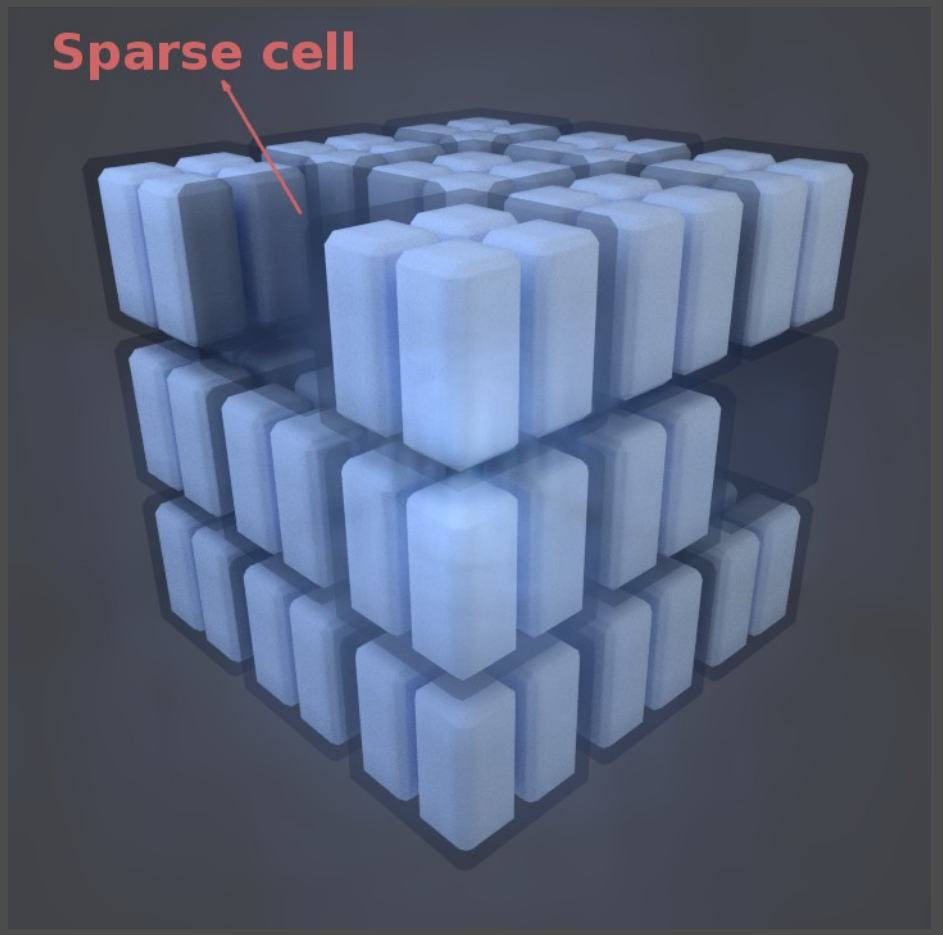
Depth 0: 3x3x3

Depth 1: 2x2x1

Total nodes = 108 - (On screenshot: 100)

Good for large empty regions. Hierarchy occlusion culling (first cull Depth Lvl. 0) may gain / lose performance – Dependes on scenario.



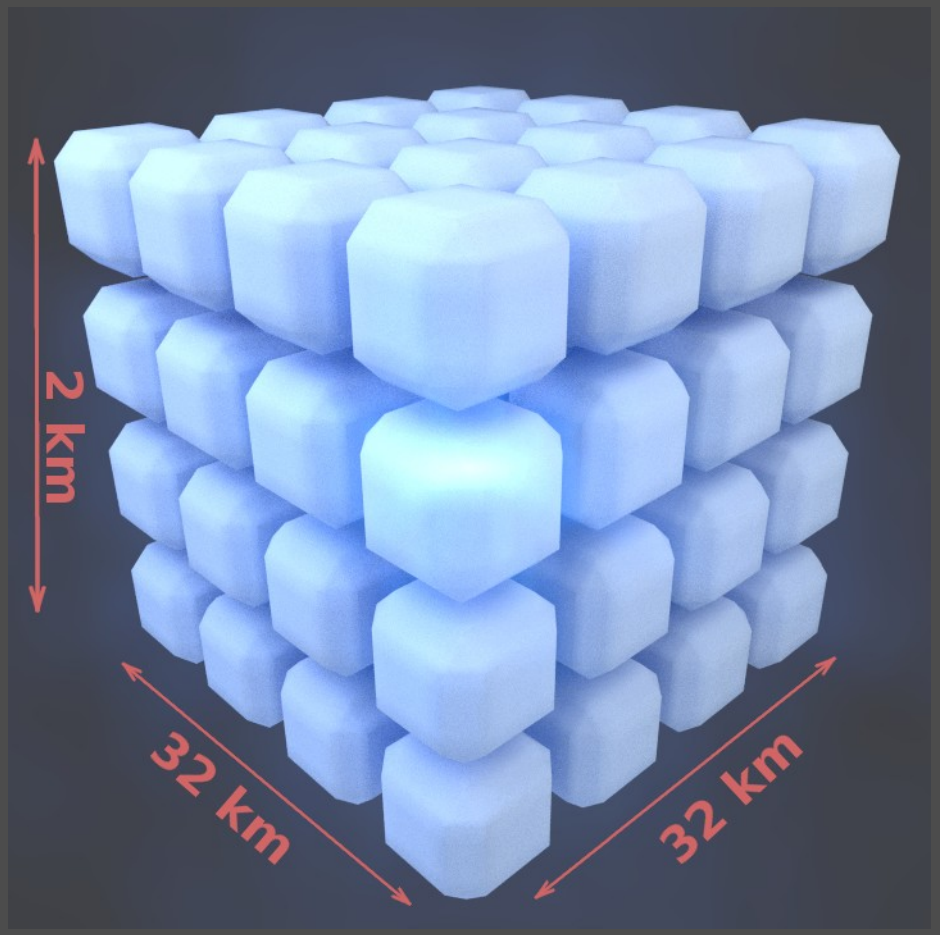


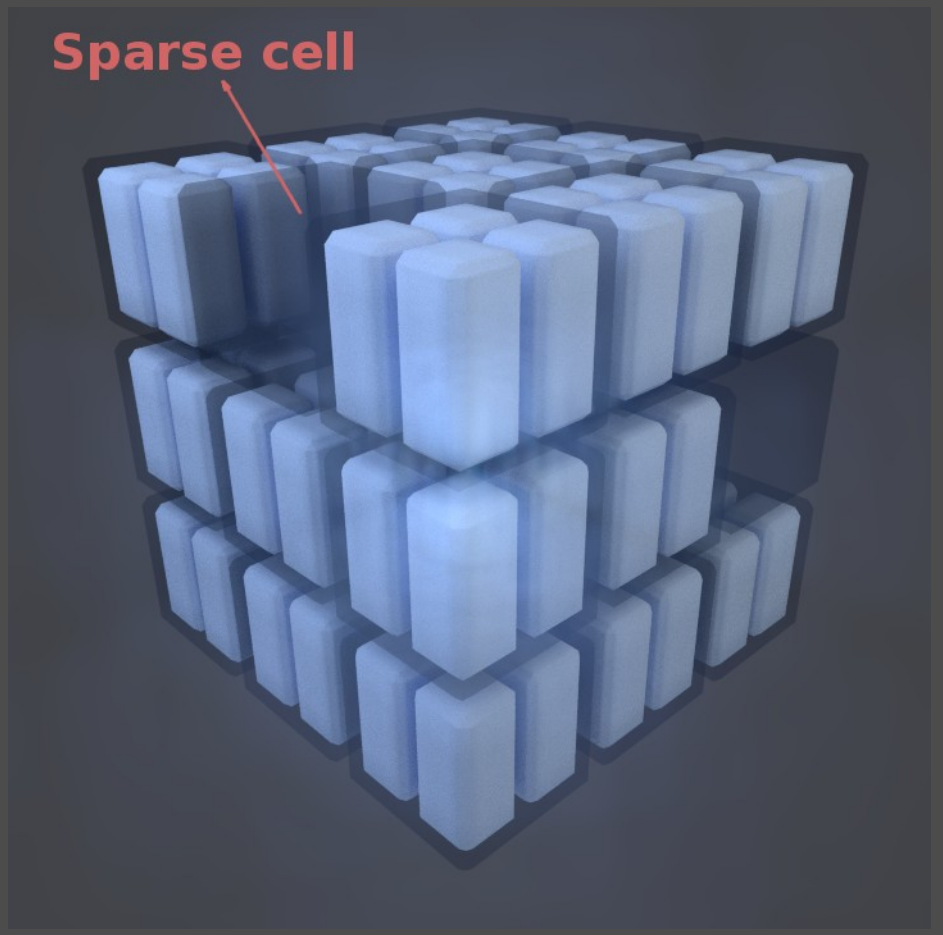
Each cell holds a contiguous array(s) of SceneNodes & SoA data (Vector, matrix, etc)

- Ideally, already grouped by render queue ID.

This is a HighLevelCull suggestion. Each implementation can do it's own, as long as it's output is an array (or various arrays) of contiguous culled entities, filtered by render queue ID.

Grid's resolution (#cells), size (in Ogre units) and depth can be to adjusted by user.





Can grids grow if the world becomes bigger? → Implementation defined.

- May increase the 3D grid's resolution, ignore, etc.

What happens if the obj. falls out of grid? → Implementation defined.

- May hide the object, leave it in the closest edge cell, move to orphan list, grow the grid's size, etc.

Resolution always 2x2x2? → That's an octree.

Many depths? → Almost always bad for performance.

Summary of alternatives: “Sparse-world storage formats” by Tom Forsyth [16]

- Hash maps based on XYZ coord. may be a good alternative

Main purpose of HighLevelCull is to do efficient hierarchy culling.

A null HighLevelCull wouldn't cull, but still needs to make sure it's output complies with specifications (filter by requested render queues, keep chunks in SoA & SIMD capable).

Proposed: UpdateAllAnimations

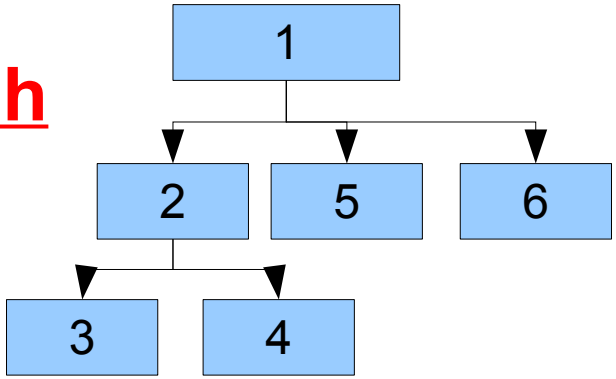
Not much to say...

- There's no reason we can't split entities into multiple threads.
    - Current implementation doesn't enforce read-only at the time the bones are updated :|
    - Chaotic skeleton update order makes it worse to fix it.
- There's no reason we can't use SoA to arrange the same entities in a SIMD fashion. (See next slides)
- Must be done before updateAllTransforms()
    - Tag points would be out of sync.

Proposed: UpdateAllTransforms()

Currently Ogre uses a **depth-first** traversal of the scene.

SceneNode is using **an unordered/hash map** to store it's children!

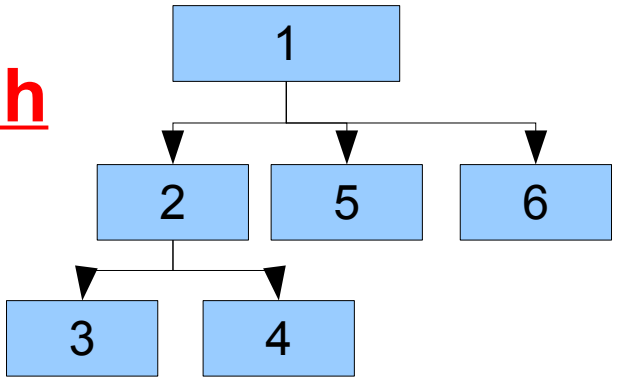


- But insertions & deletions are rare
- And iteration is the most important!

Proposed: UpdateAllTransforms()

Currently Ogre uses a **depth-first** traversal of the scene.

SceneNode is using **an unordered/hash map** to store it's children!



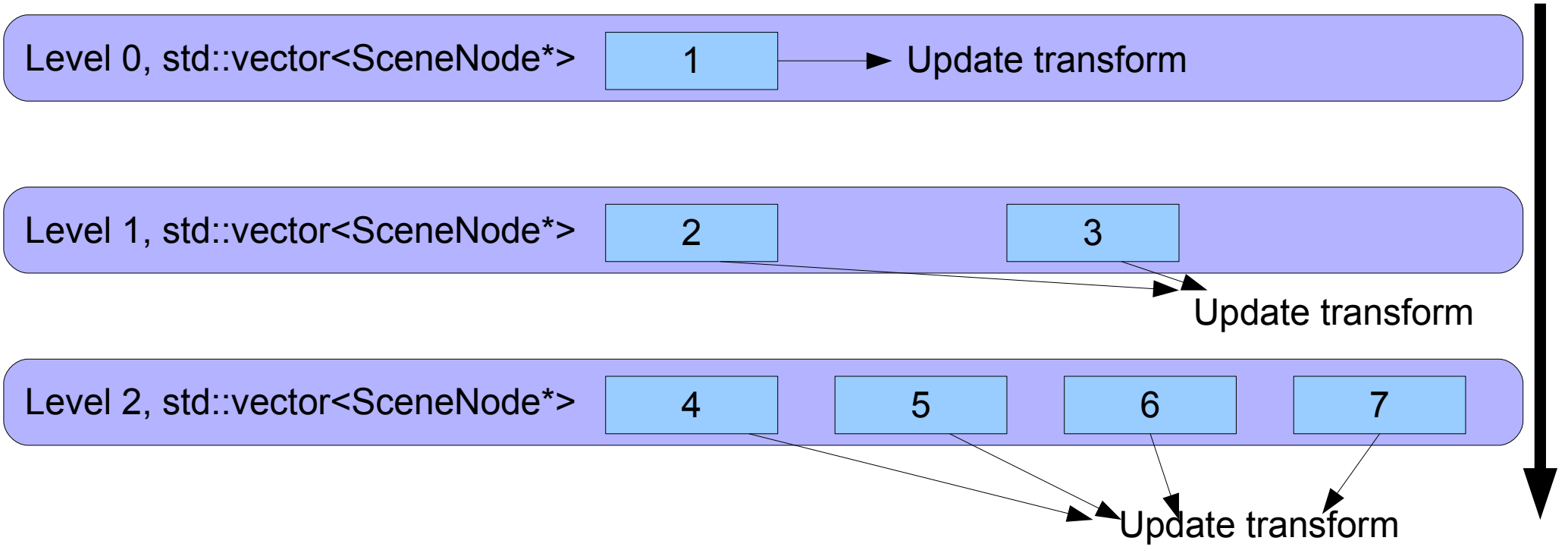
- But insertions & deletions are rare
- And iteration is the most important!

Use Pittfals of Obj. Oriented Programming [4] approach:

- **Breath-first**
- All objects in same hierarchy level are memory contiguous (re-parenting is very rare)

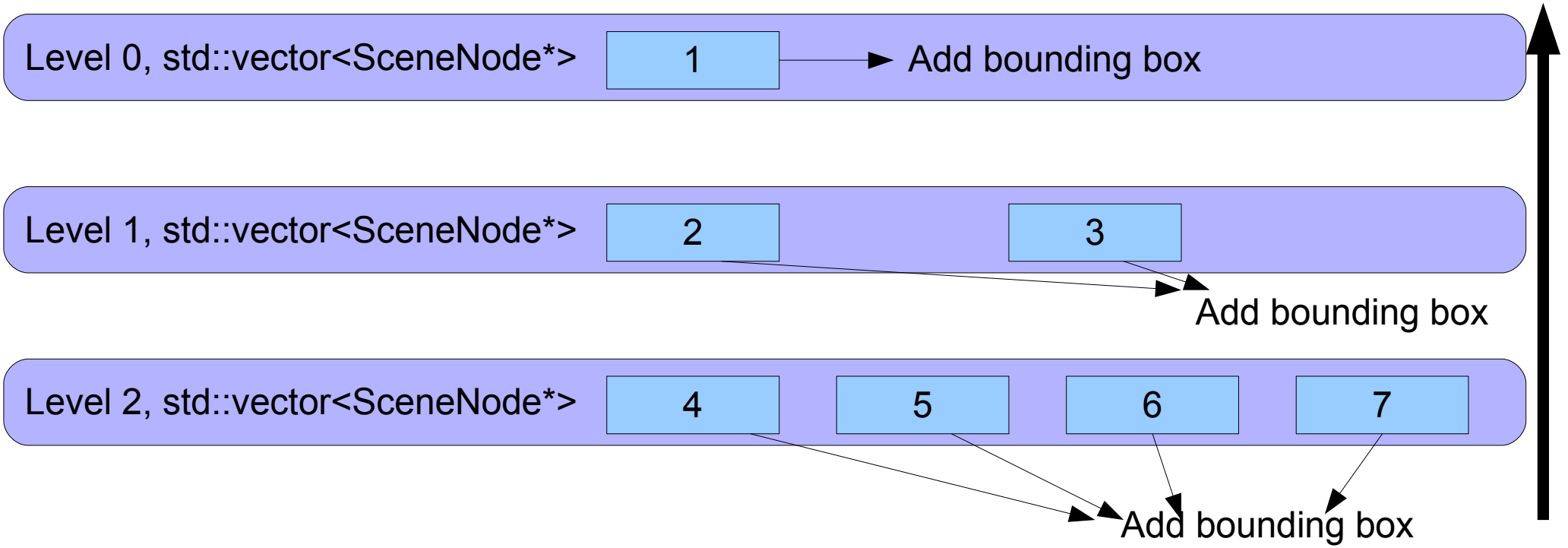
Proposed: UpdateAllTransforms()

Cache-friendly breath first:



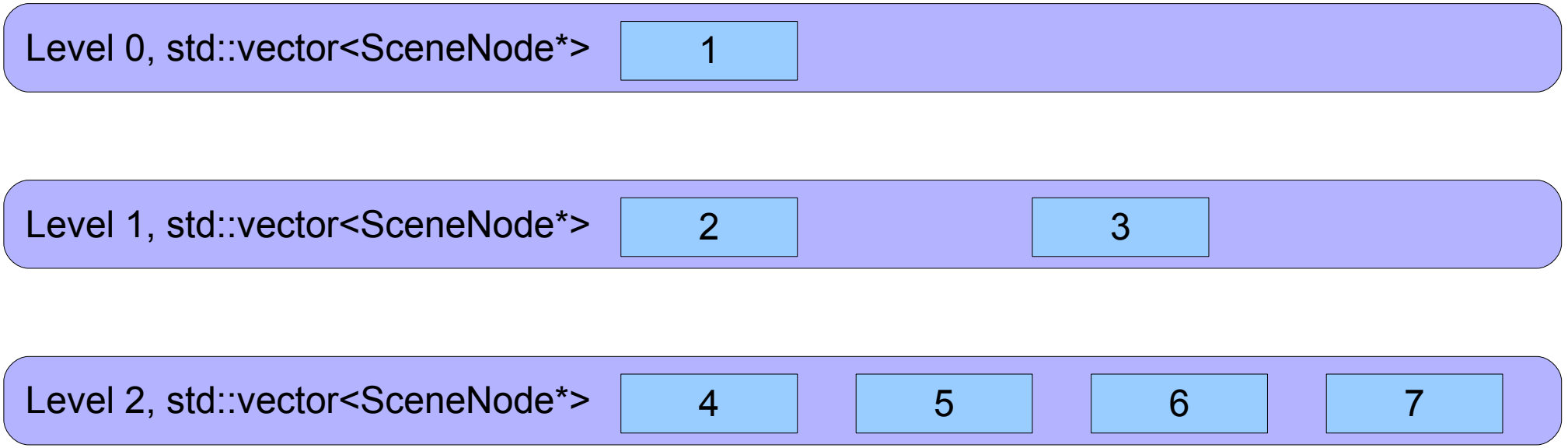
Proposed: UpdateAllTransforms()

Cache-friendly breath first:



Proposed: UpdateAllTransforms()

Cache-friendly breath first:



All SceneNodes* are actually contiguous in memory.

Nodes 2 & 3 can be processed in parallel.; after Level 0 finished (MT barrier sync)

Nodes 4-7 can be processed in parallel; after Level 1 finished (MT barrier sync)

Don't process them interleaved in each thread, or else false cache sharing will occur

Use SoA for best cache locality, but also for SIMD capabilities

Proposed: UpdateAllTransforms()

Node listeners?

Calling an indirect function, like a node listener, after updating each scene node would cause a pipeline stall (LHS: Load hit store)

But transform update is now centralized and done only once[*]

→ let's take advantage of that!

[*] The user ought to be able to manually call UpdateAllTransform outside renderOneFrame for tight control. Also don't forget “_getDerivedPositionUpdated” may be possible

Proposed: UpdateAllTransforms()

Node listeners?

Calling an indirect function, like a node listener, after updating each scene node would cause a pipeline stall (LHS: Load hit store)

But transform update is now centralized and done only once[*]

→ let's take advantage of that!

Create an array of registered listeners. Each SceneNode will still keep track of it's attached listener.

After all nodes have been updated, iterate through the array calling the listeners

- The idea? → don't do a thousand checks in a thousand SceneNodes when only one SceneNode has a listener...

[*] The user ought to be able to manually call UpdateAllTransform outside renderOneFrame for tight control. Also don't forget “_getDerivedPositionUpdated” may be possible

SoA in SceneNodes

```
class SoA_Vector3
{
        float           *m_chunkBase;   //Large preallocated memory
        unsigned char   m_index;        //Value between [0; 4) for SSE

        Vector3 getAsVector3() const
        {
                return Vector3( m_chunkBase[0 + m_index]        //X
                                m_chunkBase[4 + m_index],       //Y
                                m_chunkBase[8 + m_index] );     //Z
        }
};
```

This is how to convert between SoA (for packed SSE arithmetic) and AoS (for scalar arithmetic)

SoA in SceneNodes

```
class SoA_Vector3
{
        float           *m_chunkBase;   //Large preallocated memory
        unsigned char   m_index;        //Value between [0; 4) for SSE

        Vector3 getAsVector3() const
        {
                return Vector3( m_chunkBase[0 + m_index]        //X
                                m_chunkBase[4 + m_index],       //Y
                                m_chunkBase[8 + m_index] );     //Z
        }
};
```

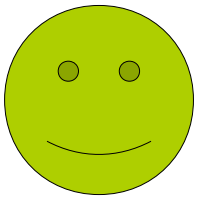
WAIT! That's not cache friendly!

SoA in SceneNodes

```
class SoA_Vector3
{
        float           *m_chunkBase;   //Large preallocated memory
        unsigned char   m_index;        //Value between [0; 4) for SSE

        Vector3 getAsVector3() const
        {
                return Vector3( m_chunkBase[0 + m_index]        //X
                                m_chunkBase[4 + m_index],       //Y
                                m_chunkBase[8 + m_index] );     //Z
        }
};
```

WAIT! That's not cache friendly!

YES IT IS



SoA data layout

Let's take an inside look:

Grid / Octree subdivision (2D slice, for sake of simplifaction)

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

SoA data layout

Let's take an inside look:

Grid / Octree subdivision (2D slice, for sake of simplifaction)

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

Each octant has it's own chunk

✗ Must copy the SceneNode's SoA data when moving to another octant.

✔ Low depths & small grid sizes to keep book-keeping to a minimum

Must be large enough to hold all scene nodes in it's area

→ User has to hint the implementation

Can reallocate itself (grow) at the expense of an fps spike (ouch!)

→ Might use an array of m_chunks to overcome this problem?

SoA data layout

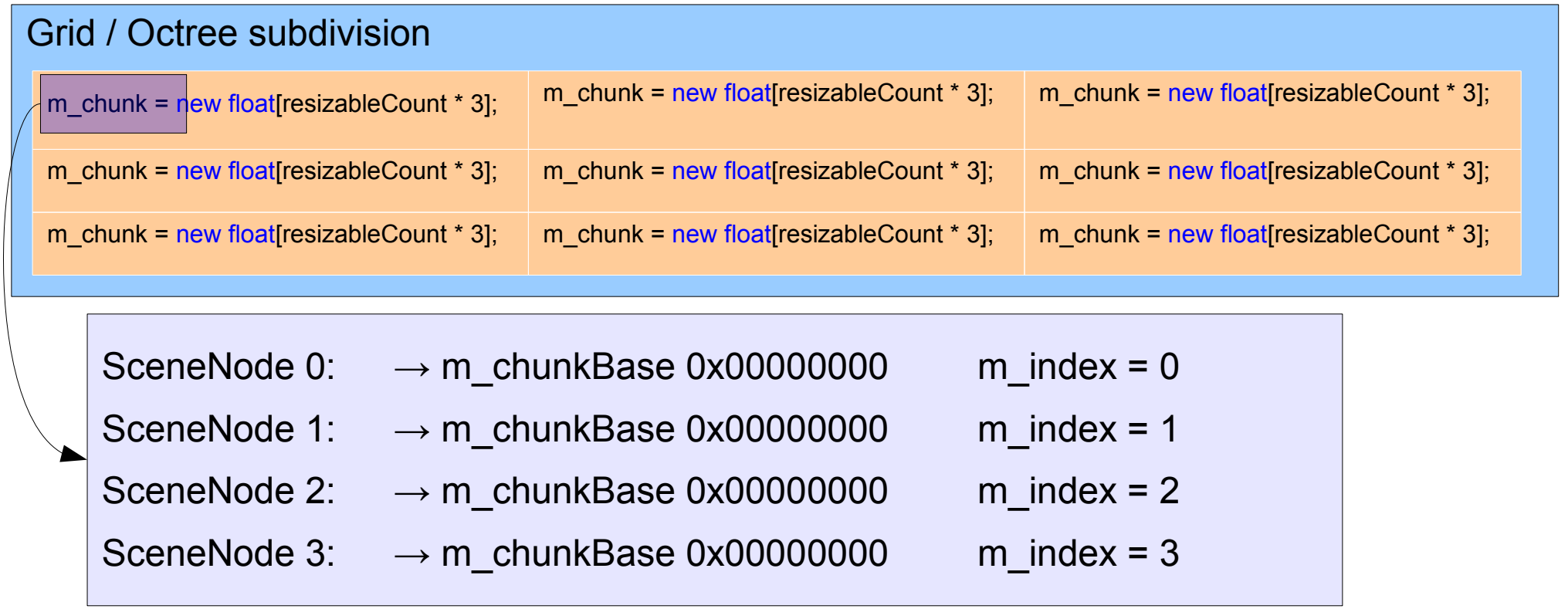
Let's asume m_chunk starts at 0x00000000:

Grid / Octree subdivision

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

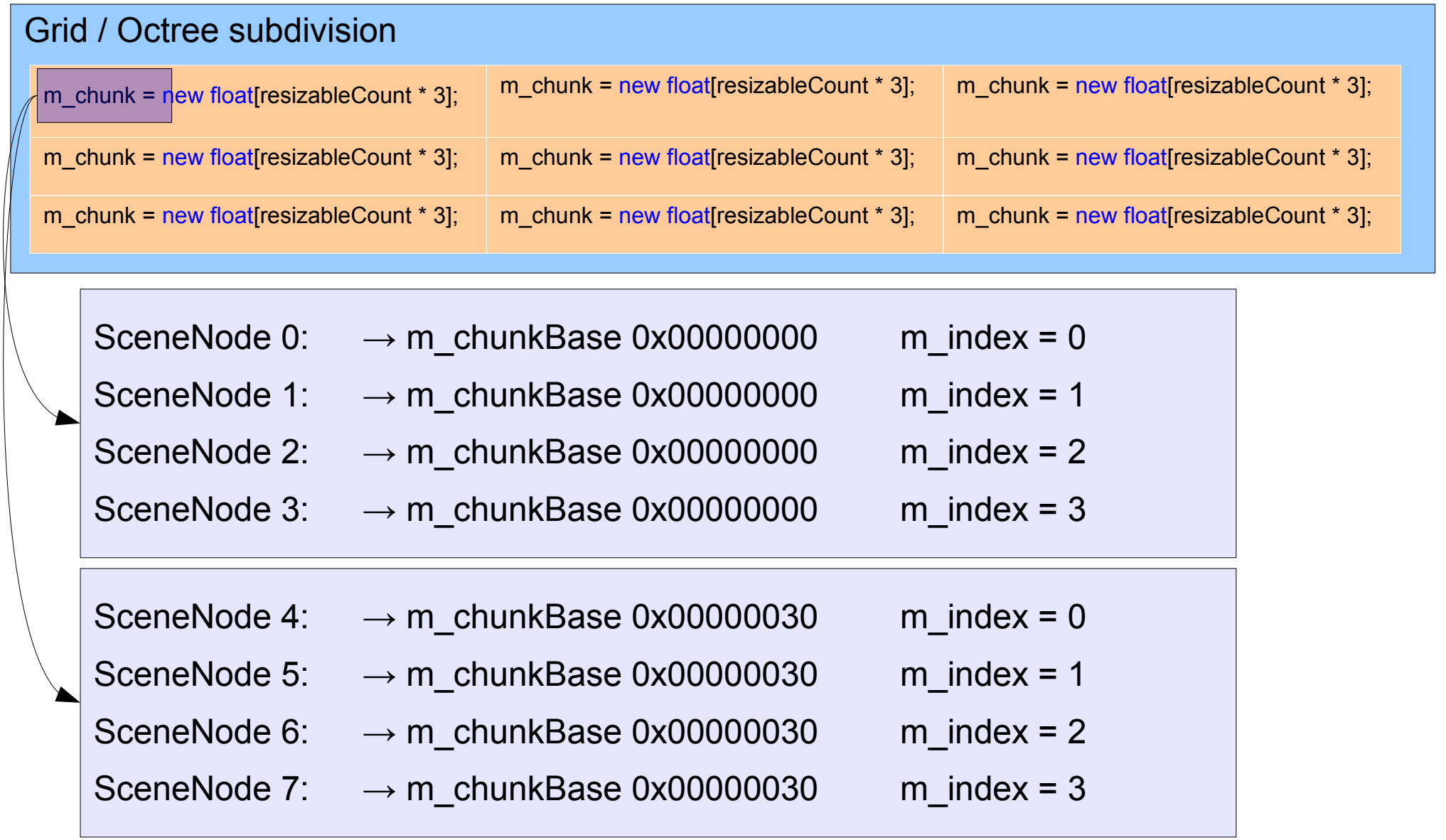
SoA data layout

Let's asume m_chunk starts at 0x00000000:



SoA data layout

Let's asume m_chunk starts at 0x00000000:



SoA data layout

Let's asume m_chunk starts at 0x00000000:

Grid / Octree subdivision

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

m_chunk =

X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z	X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z

SoA data layout

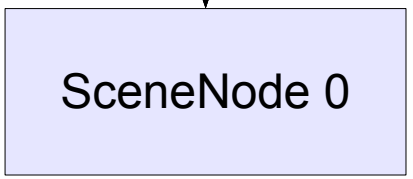
Let's asume m_chunk starts at 0x00000000:

Grid / Octree subdivision

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

m_chunk =

00	04	08	0c	10	14	18	1c	20	24	28	2c	30	34	38	3c	40	44	48	4c	50	54	58	60
X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z	X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z



SoA data layout

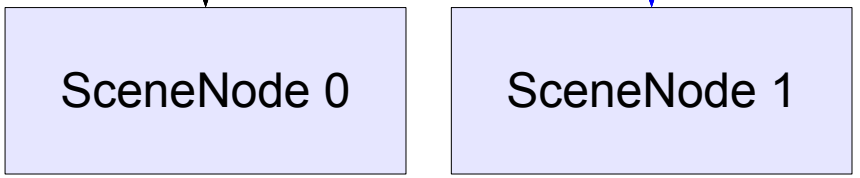
Let's asume m_chunk starts at 0x00000000:

Grid / Octree subdivision

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

m_chunk =

00	04	08	0c	10	14	18	1c	20	24	28	2c	30	34	38	3c	40	44	48	4c	50	54	58	60
X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z	X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z



SoA data layout

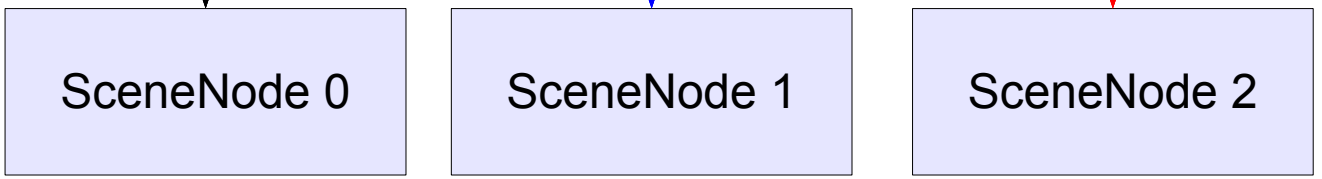
Let's asume m_chunk starts at 0x00000000:

Grid / Octree subdivision

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

m_chunk =

00	04	08	0c	10	14	18	1c	20	24	28	2c	30	34	38	3c	40	44	48	4c	50	54	58	60
X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z	X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z



SoA data layout

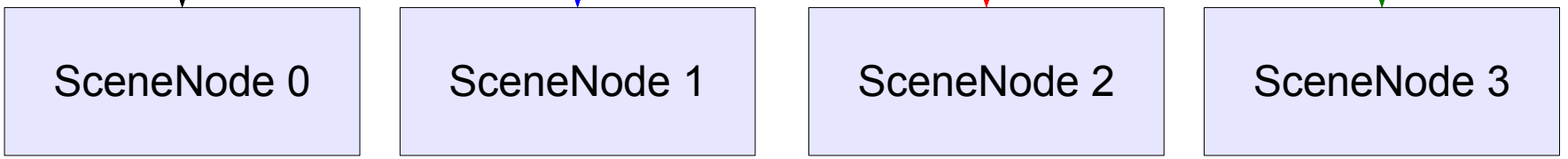
Let's asume m_chunk starts at 0x00000000:

Grid / Octree subdivision

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

m_chunk =

00	04	08	0c	10	14	18	1c	20	24	28	2c	30	34	38	3c	40	44	48	4c	50	54	58	60
X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z	X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z



SoA data layout

Let's asume m_chunk starts at 0x00000000:

Grid / Octree subdivision

m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];
m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];	m_chunk = new float[resizableCount * 3];

m_chunk =

00	04	08	0c	10	14	18	1c	20	24	28	2c	30	34	38	3c	40	44	48	4c	50	54	58	60
X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z	X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z

SceneNode 0: → m_chunkBase 0x00000000 m_index = 0
SceneNode 1: → m_chunkBase 0x00000000 m_index = 1
SceneNode 2: → m_chunkBase 0x00000000 m_index = 2
SceneNode 3: → m_chunkBase 0x00000000 m_index = 3

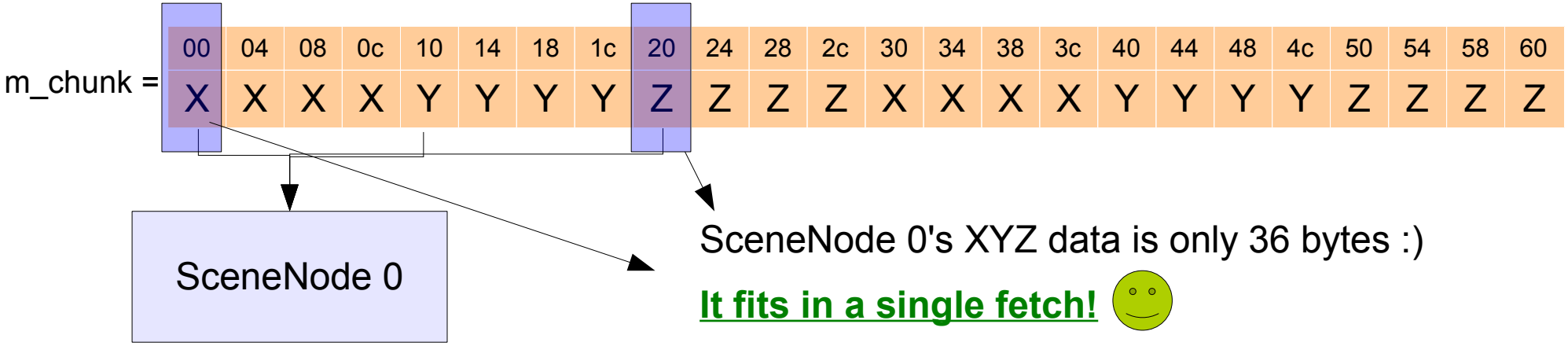
SceneNode 4: → m_chunkBase 0x00000030 m_index = 0
SceneNode 5: → m_chunkBase 0x00000030 m_index = 1
SceneNode 6: → m_chunkBase 0x00000030 m_index = 2
SceneNode 7: → m_chunkBase 0x00000030 m_index = 3

SoA data layout

How is this cache friendly then?

How caches work:

- Fetches are done in blocks. If a single byte in a block was modified, the whole block needs to be flushed again.
- Most (if not all) x86/x64 CPUs fetch 64-byte blocks



SceneNode 0's XYZ data is only 36 bytes :)

It fits in a single fetch!

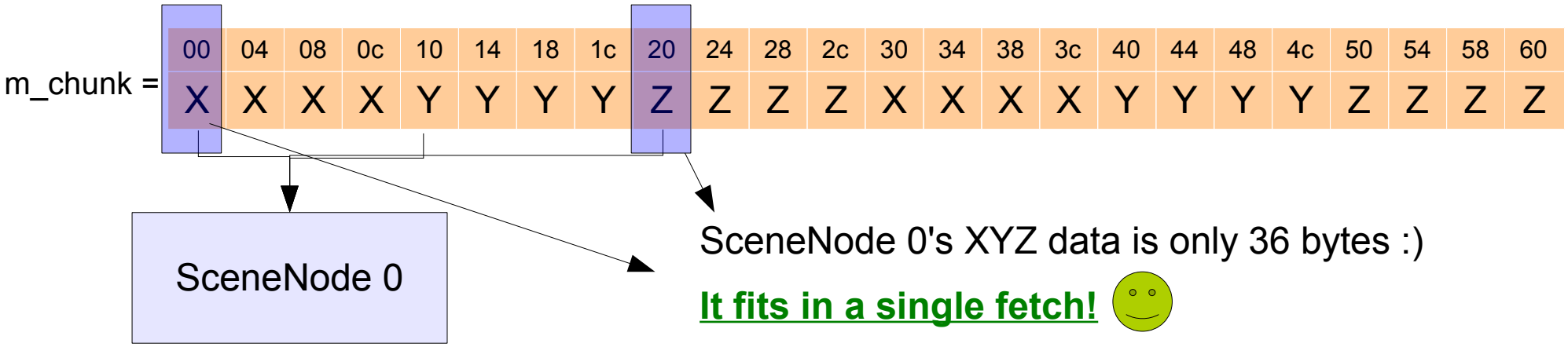
And when accessing SceneNode 1's data, it will also be already in the cache!

SoA data layout

How is this cache friendly then?

How caches work:

- Fetches are done in blocks. If a single byte in a block was modified, the whole block needs to be flushed again.
- Most (if not all) x86/x64 CPUs fetch 64-byte blocks



SceneNode 0's XYZ data is only 36 bytes :)

It fits in a single fetch!

And when accessing SceneNode 1's data, it will also be already in the cache!

✗ Many mobile PPC & ARM devices fetch 32-byte blocks :(

- Must be able to adjust (reduce or increase) at compile-time number of floats that can be packed together; with option to leave blank unused areas (depends on arch.)

SoA data layout

Extremely cache friendly.

Must allow reducing/increasing packing (Mobile: 1 to 3 floats. VMX: 8 floats)

Cull list returned by HighLevelCull() **must** follow these rules!

- It will be also responsible for keeping locality of the SceneNodes & entities.

Store Quaternions, Scale & 4x4 Matrices in the same way.

- Transform, normalize & cull **4 objects at once, in multiple threads**

  → Taking DICE* [3] idea to a whole new level!
- Final matrix cache result must be stored AoS → 16 SoA floats won't fit in a single fetch and will polute the cache. UpdateAllAnimation performs packed operations, but must store (cache) all matrices in scalar form for the RenderQueue to prepare the batch. Scalar mat. may have to be thread-local.

  → **Use non-temporal stores to write the scalar matrix result!**

  → **After low level culling, the order can't be guaranteed and operations must go scalar.**

SoA data layout

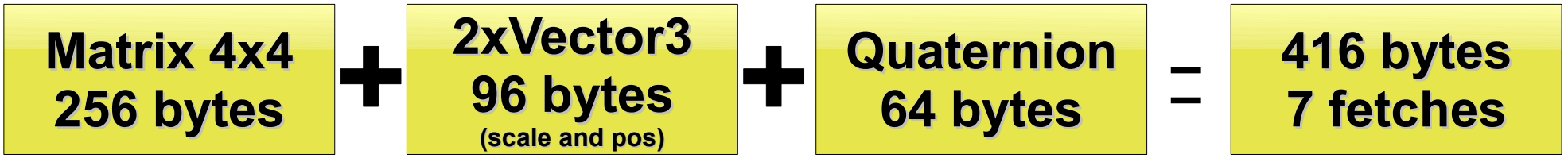
Matrix 4x4 layout: 4 * 4 * 4 * 4 = 256 bytes = 4 fetches

00	04	08	0c	10	14	18	1c	20	24	28	2c	30
a11	b11	c11	d11	a12	b12	c12	d12	a13	b13	c13	d13	...

Quaternion layout: 4 * 4 * 4 = 64 bytes = 1 fetch

00	04	08	0c	10	14	18	1c	20	24	28	2c	30
X	X	X	X	Y	Y	Y	Y	Z	Z	Z	Z	W..

UpdateAllTransforms()



UpdateAllTransforms()

Matrix 4x4
256 bytes

+

2xVector3
96 bytes
(scale and pos)

+

Quaternion
64 bytes

=

416 bytes
7 fetches

UpdateAllAnimations() - Alternative 1

2xMatrix 4x3
384 bytes
lerp(time0_mat, time1_mat)

=

384 bytes
6 fetches

UpdateAllAnimations() - Alt. 2

2x(2xVector3
+ Quat.)
lerp(pos, rot, scale)

=

320 bytes
6 fetches
(not 5! Have to sum
Individual fetches)

UpdateAllTransforms()

Matrix 4x4
256 bytes

+

2xVector3
96 bytes
(scale and pos)

+

Quaternion
64 bytes

=

416 bytes
7 fetches

UpdateAllAnimations() - Alternative 1

2xMatrix 4x3
384 bytes
lerp(time0_mat, time1_mat)

=

384 bytes
6 fetches

UpdateAllAnimations() - Alt. 2

2x(2xVector3
+ Quat.)
lerp(pos, rot, scale)

=

320 bytes
6 fetches
(not 5! Have to sum Individual fetches)

During cull stage

3xMatrix 4x4
768 bytes
world, view, proj

=

768 bytes
12 fetches

UpdateAllTransforms()

Matrix 4x4
256 bytes

+

2xVector3
96 bytes
(scale and pos)

+

Quaternion
64 bytes

=

416 bytes
7 fetches

UpdateAllAnimations() - Alternative 1

2xMatrix 4x3
384 bytes
lerp(time0_mat, time1_mat)

=

384 bytes
6 fetches

UpdateAllAnimations() - Alt. 2

2x(2xVector3
+ Quat.)
lerp(pos, rot, scale)

=

320 bytes
6 fetches
(not 5! Have to sum Individual fetches)

During cull stage

3xMatrix 4x4
768 bytes
world, view, proj

=

768 bytes
12 fetches

Most of this data is

NON-TEMPORAL

Have fun with PREFETCH*h*!

MUST- READ! Chapter 7 from Intel® 64 and IA-32 Architectures Optimization Reference Manual [7]

UpdateAllTransforms()

Matrix 4x4
256 bytes

+

2xVector3
96 bytes
(scale and pos)

+

Quaternion
64 bytes

=

416 bytes
7 fetches

UpdateAllAnimations() - Alternative 1

2xMatrix 4x3
384 bytes
lerp(time0_mat, time1_mat)

=

384 bytes
6 fetches

UpdateAllAnimations() - Alt. 2

2x(2xVector3
+ Quat.)
lerp(pos, rot, scale)

=

320 bytes
6 fetches
(not 5! Have to sum Individual fetches)

During cull stage

3xMatrix 4x4
768 bytes
world, view, proj

=

768 bytes
12 fetches

Most of this data is

NON-TEMPORAL

Have fun with PREFETCH*h*!

MUST- READ! Chapter 7 from Intel® 64 and IA-32 Architectures Optimization Reference Manual [7]

What about shared data? (i.e. AABB, bounding radius)

Can't guarantee SIMD-friendly order (cost larger than benefit)

Use shiftps and other packing/unpacking instructions.

Data address being loaded depends on data [14], can't be calculated

(needs to fetch entity ptr to load the associated mesh next)

Vector4
64 bytes
(halfExtent or min, radius in W)

+

Vector3
48 bytes
(halfExtent's center, or max)

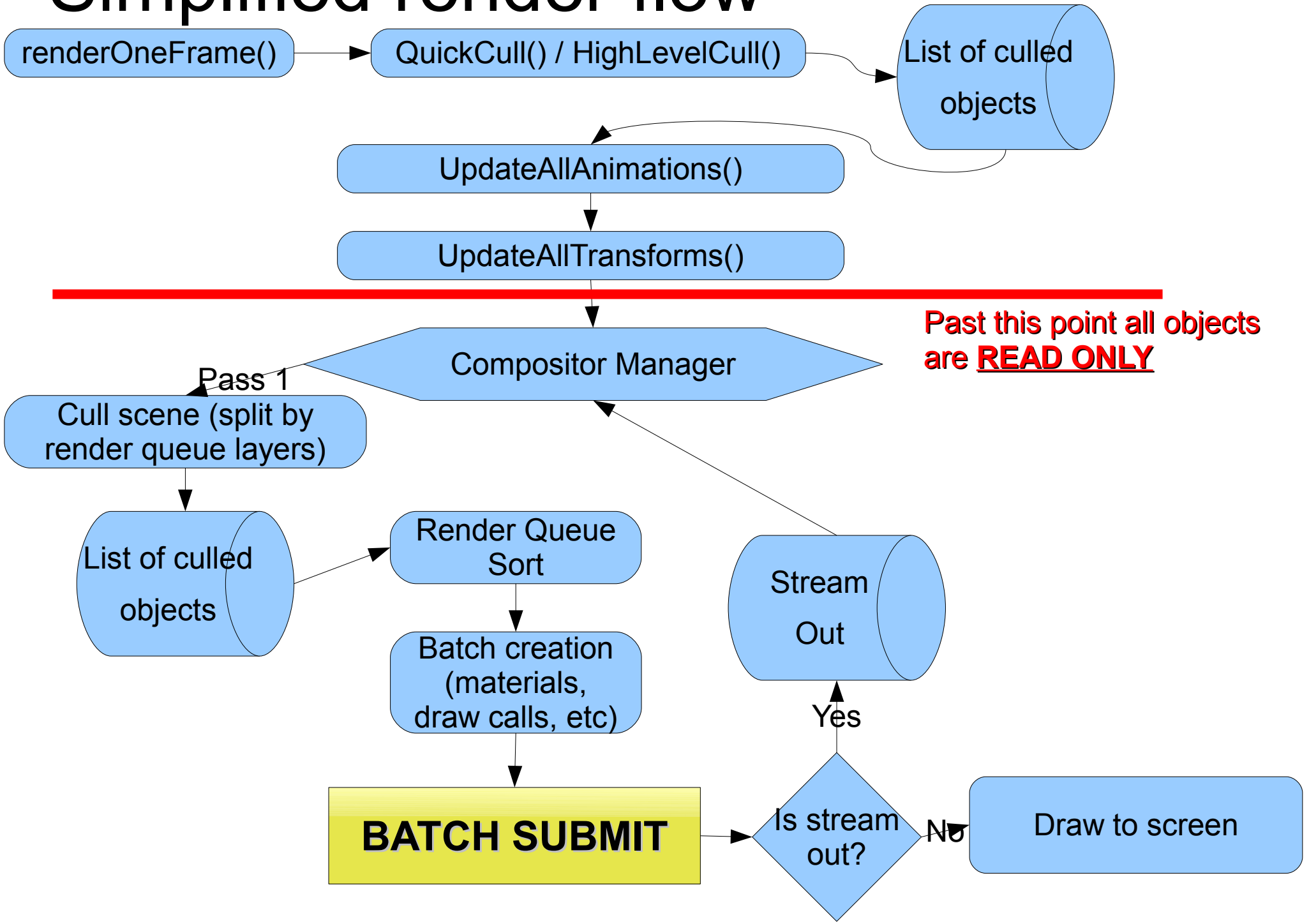
=

112 bytes
2 fetches

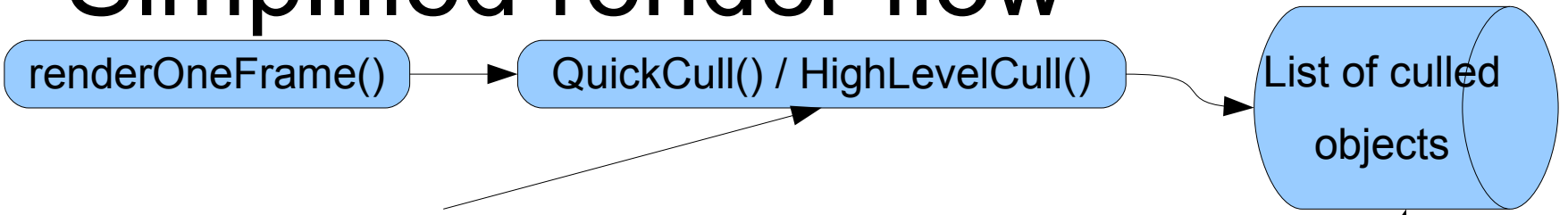
Due to it's reusage on each entity, this data is probably TEMPORAL

Keep AABB & radius SoA. Even if there's no SIMD. Cache locality is crucial to avoid "data dependency on data" problem. Probably all meshes' data fits in L2 cache!

Simplified render flow



Simplified render flow



Scalability depends on implementation & technique.
In charge of book keeping SceneNode & Entities SoA's cache locality.

Needs to keep objects separated by RenderQueue ID for fast selective rendering

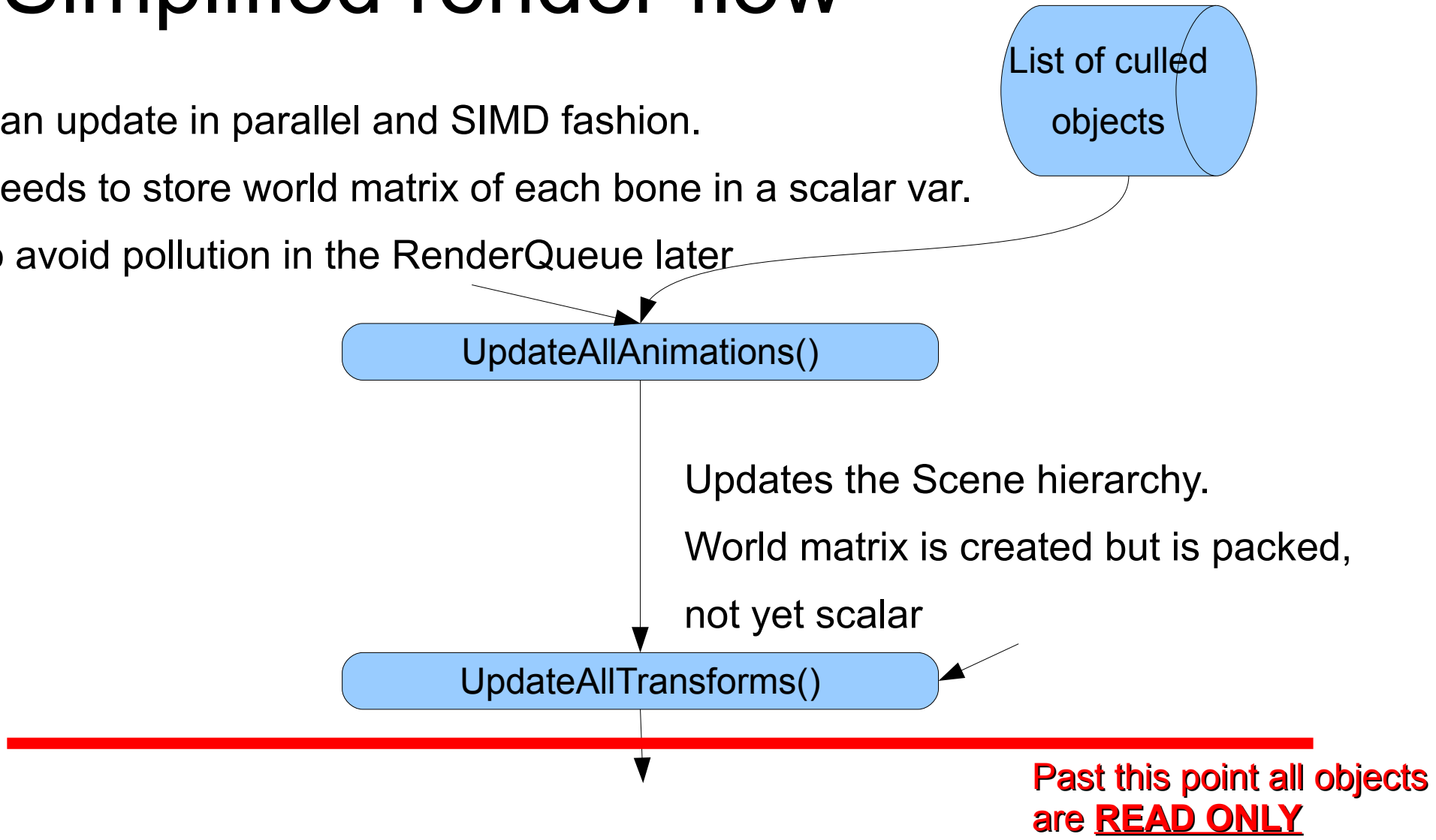
List doesn't require any order; except that all SceneNodes & entities are contiguous in memory.
Culling may be very inexact or even non-existant.
Entities sharing the same skeleton need to be adjacent.

Simplified render flow

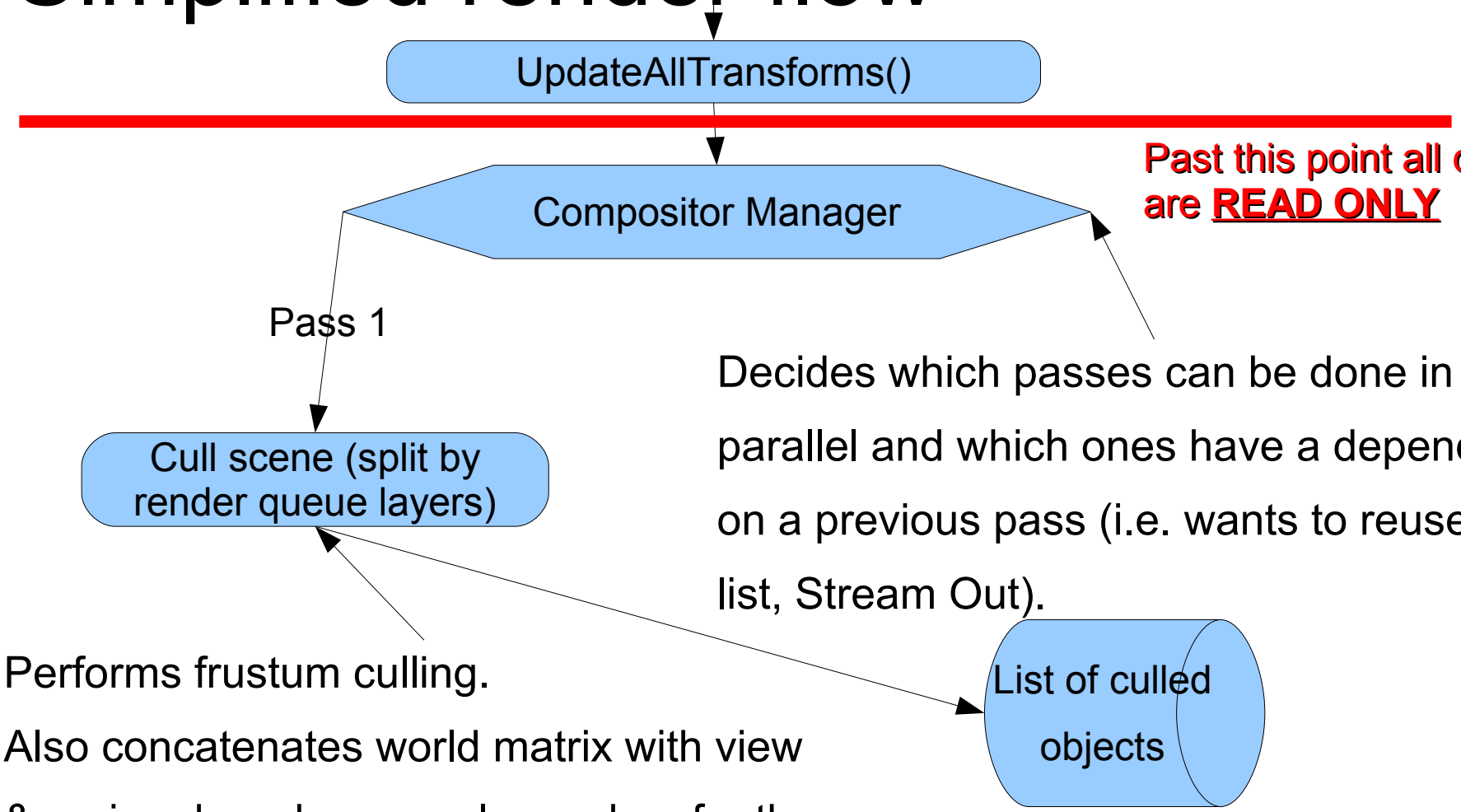
Can update in parallel and SIMD fashion.

Needs to store world matrix of each bone in a scalar var.

to avoid pollution in the RenderQueue later



Simplified render flow

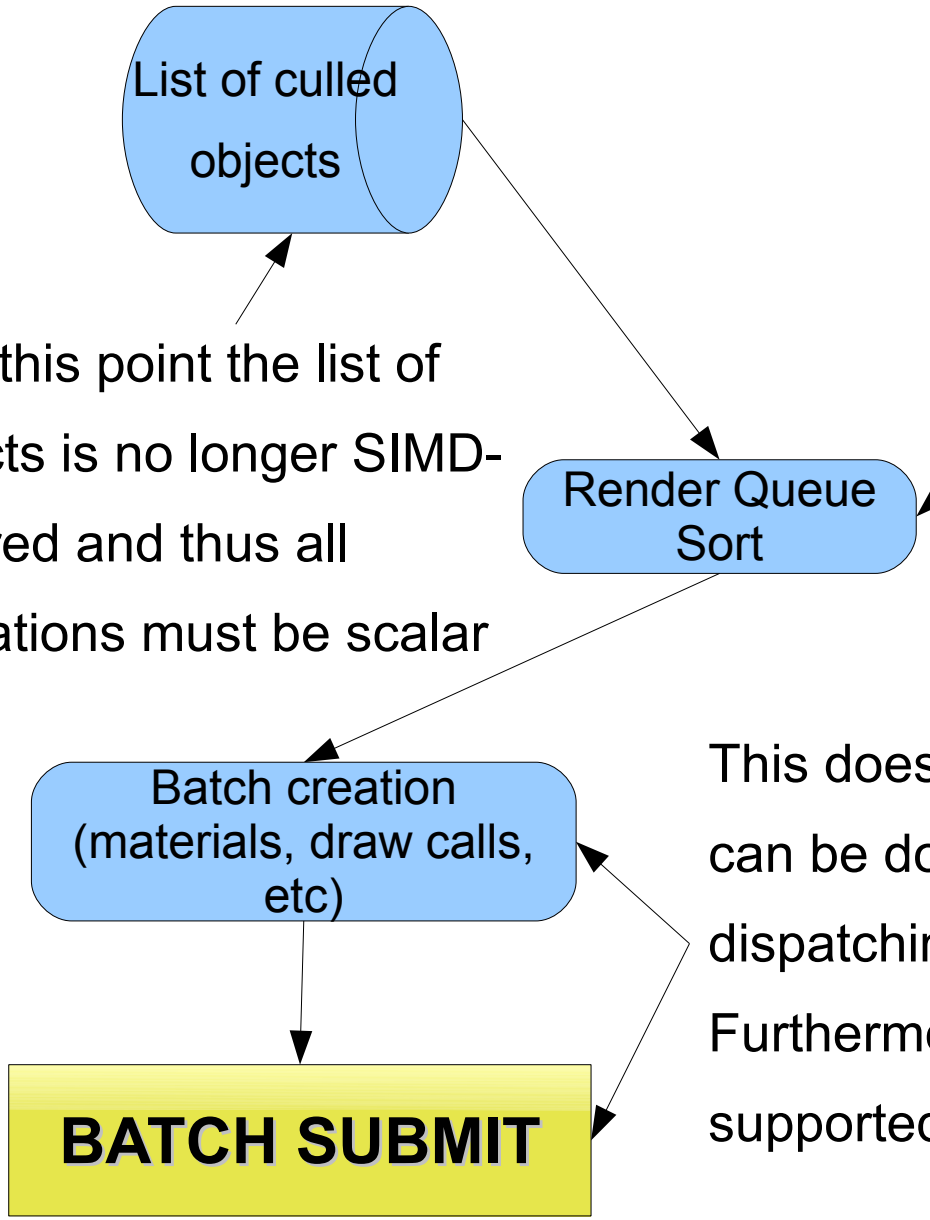


Performs frustum culling.

Also concatenates world matrix with view & proj and produces scalar caches for the RenderQueue.

The cache is obviously local to the thread

Simplified render flow



Past this point all objects are **READ ONLY**

Past this point the list of objects is no longer SIMD-ordered and thus all operations must be scalar

The render queue has been filtered in HighLevelCull. However this RQ may still want to reorder Renderables (i.e. Early Z & Transparency)

This doesn't change much. Except now it can be done in parallel as well as dispatching draw call commands. Furthermore, Stream Out isn't currently supported in Ogre

What seems “wasteful” or inefficient

World view proj is concatenated *before* culling, causing unnecessary muls.

- Doing it after culling involves many 'if's or **an additional level of indirection**
- Culling aids the GPU, not the CPU (argueable).
- Worst-case scenario all or most objects pass the cull stage

  → Better stable framerate than jumpy one
- Tradeoff for SIMD, cache locality & Threading to overcome this.

  → ALU growth is the highest. An additional level of indirection would hurt us in the future

Bone's scalar world Matrix is cached too early (again before culling).

- Same arguments above.
- This one may actually matter, due to the high number of bones multiplied by instances, and this is a hotspot. Matrix4 scalar variable count may explode.
- The problem is that after culling, it is no longer SIMD.
- Play with the HighLevelCull settings.

Low-level cull methods

- None → Return list “as is” (i.e. CPU bound)

Low-level cull methods

- None → Return list “as is” (i.e. CPU bound)
- Simple camera frustum check (balance)

Low-level cull methods

- None → Return list “as is” (i.e. CPU bound)
- Simple camera frustum check (balance)
- Software occlusion culling (GPU bound)
  - Very popular these days. Highly scalable & SIMD
  - Basic idea: A software rasterizer that only performs Depth-check and renders low-poly bounding geometries to a low-res image.
  - Objects that are totally occluded are rejected.
  - Killzone* 3 [8], Frostbite* 2 [3]
  - Coverage buffer variation in Cryengine* 3 [9]
- Others...

Software occlusion culling

Highly developed literature ([3], [8], [9])

Relatively easy & tempting to do.

Hard for artists to model good occlusions [8]

- It would be great to have tool-asisted modelling.

Mesh format needs upgrade to support custom meta-tags

- Occlusion geometry would be stored there.
- Custom meta-tags are very demanded by artists anyway

Other tips

Use unique ids instead of string names for reference and as key look up.

- See InstancedEntity::mId

Use static string more often, instead of bloated std::string (possibly UTF by the way...)

- std::string & wstring are fine. The problem is that we use it ***everywhere***.
- “A sprintf that isn't as ugly” by Tom Forsyth [15]
  home.comcast.net/~tom_forsyth/blog.wiki.html

Other tips

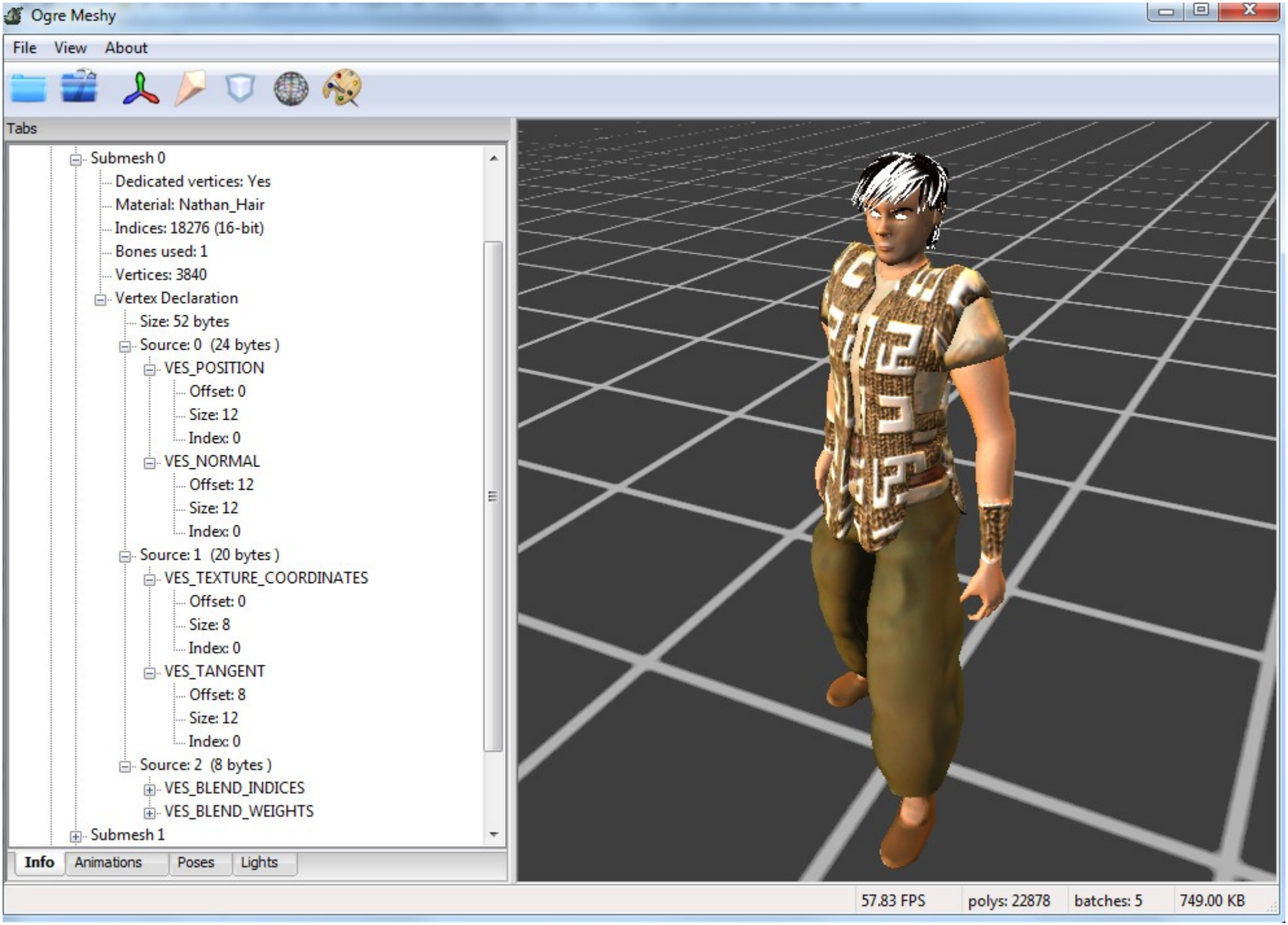
Use unique ids instead of string names for reference and as key look up.

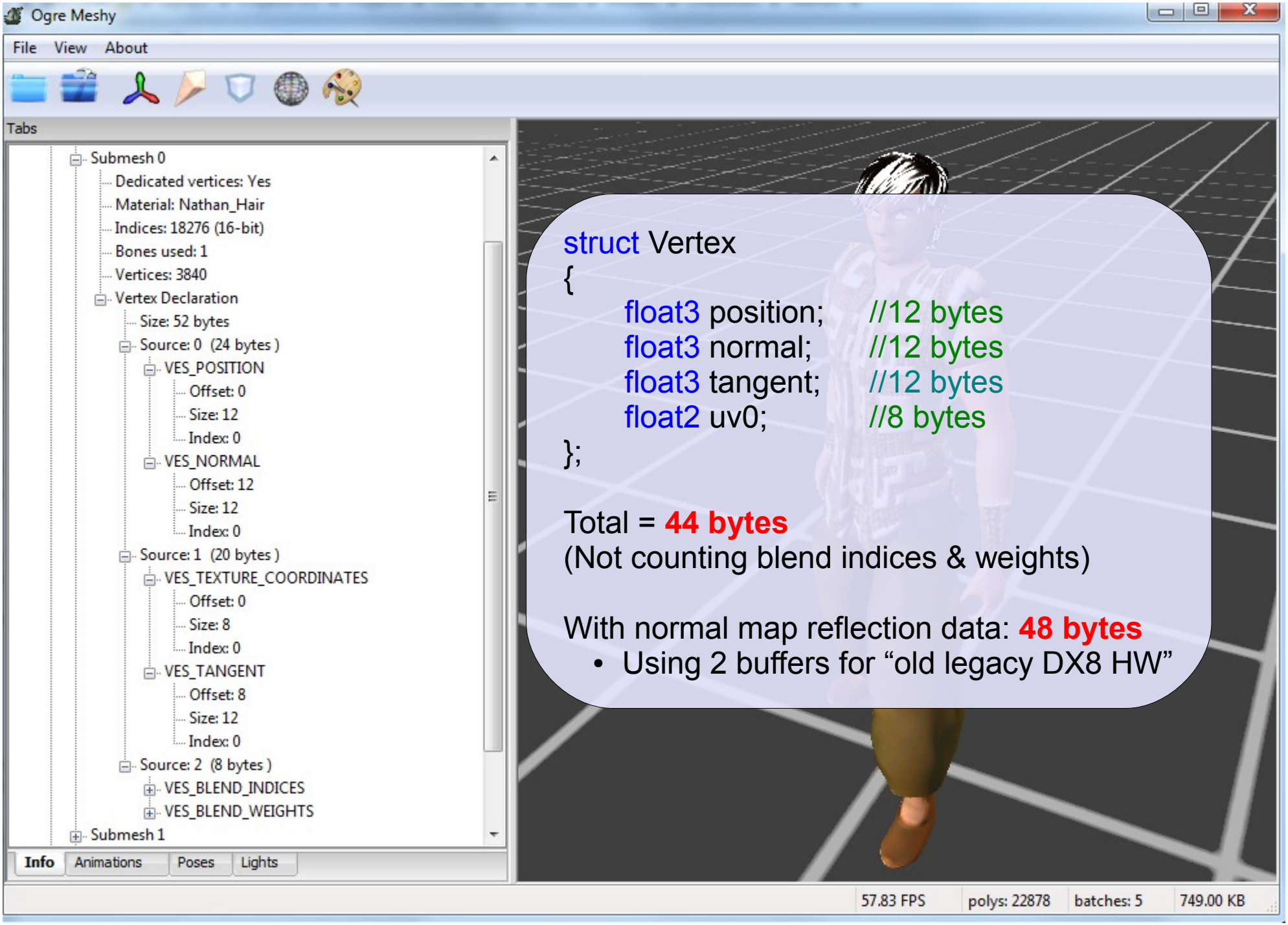
- See InstancedEntity::mId

Use static string more often, instead of bloated std::string (possibly UTF by the way...)

- std::string & wstring are fine. The problem is that we use it ***everywhere***.
- “A sprintf that isn't as ugly” by Tom Forsyth [15]
  home.comcast.net/~tom_forsyth/blog.wiki.html
  - The idea: Encapsulate sprintf into a class and use it like std::string
  - No dynamic allocation (optional → grab from preallocated space)
  - std::string like interface (i.e. Easy to read string concatenation)
  - Type-safe
  - Overflow-safe (i.e. it will refuse to scribble, and will assert in debug mode)

- ~~Too many cache misses :(~~
- ~~Inefficient Scene traversal & processing~~
- Fat, unflexible, vertex format
- Fixed functions vs programmable shaders
  - “setFog”, etc


Ogre Meshy
File
View
About
Tabs
Submesh 0
Dedicated vertices: Yes
Material: Nathan_Hair
Indices: 18276 (16-bit)
Bones used: 1
Vertices: 3840
Vertex Declaration
Size: 52 bytes
Source: 0 (24 bytes)
VES_POSITION
Offset: 0
Size: 12
Index: 0
VES_NORMAL
Offset: 12
Size: 12
Index: 0
Source: 1 (20 bytes)
VES_TEXTURE_COORDINATES
Offset: 0
Size: 8
Index: 0
VES_TANGENT
Offset: 8
Size: 12
Index: 0
Source: 2 (8 bytes)
VES_BLEND_INDICES
VES_BLEND_WEIGHTS
Submesh 1
Info
Animations
Poses
Lights
57.83 FPS
polys: 22878
batches: 5
749.00 KB


Ogre Meshy
File View About
Tabs
Submesh 0
Dedicated vertices: Yes
Material: Nathan_Hair
Indices: 18276 (16-bit)
Bones used: 1
Vertices: 3840
Vertex Declaration
Size: 52 bytes
Source: 0 (24 bytes)
VES_POSITION
Offset: 0
Size: 12
Index: 0
VES_NORMAL
Offset: 12
Size: 12
Index: 0
Source: 1 (20 bytes)
VES_TEXTURE_COORDINATES
Offset: 0
Size: 8
Index: 0
VES_TANGENT
Offset: 8
Size: 12
Index: 0
Source: 2 (8 bytes)
VES_BLEND_INDICES
VES_BLEND_WEIGHTS
Submesh 1
Info
Animations
Poses
Lights
struct Vertex
{
float3 position; //12 bytes
float3 normal; //12 bytes
float3 tangent; //12 bytes
float2 uv0; //8 bytes
};
Total = 44 bytes
(Not counting blend indices & weights)
With normal map reflection data: 48 bytes
• Using 2 buffers for “old legacy DX8 HW”
57.83 FPS
polys: 22878
batches: 5
749.00 KB

Let's take a look at other engines...

CryENGINE* 3 [10]

```
struct Vertex
{
      float16 position;   //8 bytes, 2 unused
      float16 uv0;        //4 bytes
      short4 Tangent;     //8 bytes
};
```

Total = **20 bytes!!!**

- Use 1 UV (vs 2 in Just Cause)
- Encodes normal, tang. & binorm. using QTangents in just 8 bytes (higher precision than Just Cause)
- Still 2 bytes unused...

Just Cause* 2 [11]

```
struct Vertex
{
      short4 position;    //8 bytes, 2 unused
      short4 uv0_uv1;     //8 bytes
      ubyte4 Tangents;    //4 bytes
      ubyte4 Color;       //8 bytes
};
```

Total = **24 bytes!!!**

- Uses 2 UVs!!
- Encodes normal, tangent & binormal data in just 4 bytes
- Has additional colour data for forest randomization
- Still 2 bytes unused...

CryENGINE* 3 [10]

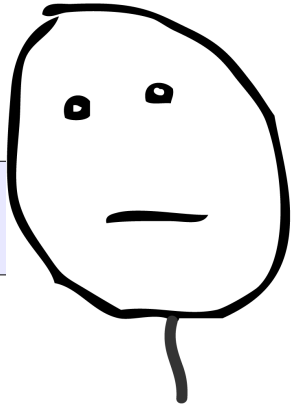
```
struct Vertex
{
      float16 position;   //8 bytes, 2 unused
      float16 uv0;        //4 bytes
      short4 Tangent;     //8 bytes
};
```

Total = **20 bytes!!!**

- Use 1 UV (vs 2 in Just Cause)
- Encodes normal, tang. & binorm. using QTangents in just 8 bytes (higher precision than Just Cause)
- Still 2 bytes unused...

OGRE

Total = **48 bytes**

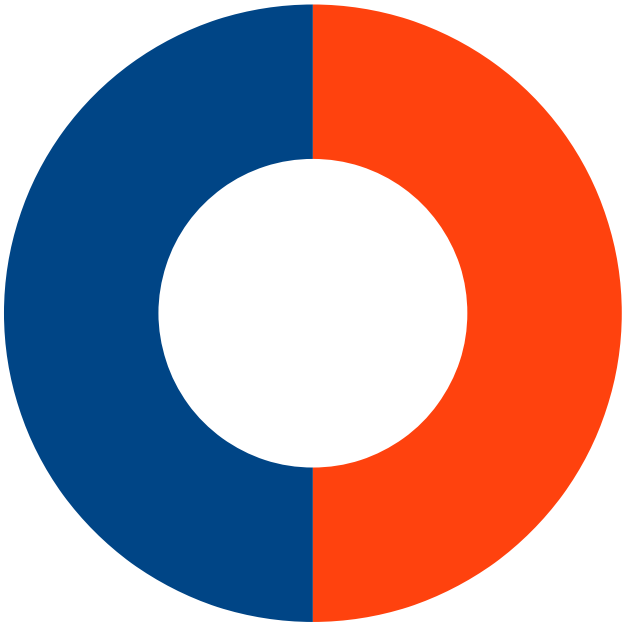


Just Cause* 2 [11]

```
struct Vertex
{
      short4 position;    //8 bytes, 2 unused
      short4 uv0_uv1;     //8 bytes
      ubyte4 Tangents;    //4 bytes
      ubyte4 Color;       //8 bytes
};
```

Total = **24 bytes!!!**

- Uses 2 UVs!!
- Encodes normal, tangent & binormal data in just 4 bytes
- Has additional colour data for forest randomization
- Still 2 bytes unused...



More data for half the size....

“Fat” vertices

Position: 16-bit is enough for local pos. Not so much for absolute/world pos.

- Unless it's a very big mesh.

UVs: 16-bit is almost always good enough.

Normal encoding is relatively new

- QTangents (Crytek) are awesome.
- Angle-based (Avalanche) may be lossy, but it's small.

Blend weights: ubyte4 is often enough.

✗ Hard to escape from the automatic “DX8” re-layout.

✗ No way to tell the cmd tools the Vertex layout in a flexible way

Flexible data layout: Ideal

```
C:\> ogrexmlconverter file.mesh.xml out.mesh -l vertex.vlayout
```

Vertex.vlayout

```
struct Vertex : buffer[0]
{
    POSITION        : half4;
    TEXCOORD3       : ubyte4;
    TEXCOORD201     : short1, short2, short1;
    NORMALS         : qtangent|angle|normalxyz, half3;
};

struct Vertex : buffer[1]
{
    TANGENT         : float4;
    BINORMAL        : half3;
};
```

Vertex.vlayout would be a simple text file

Flexible data layout: Ideal

```
C:\> ogrexmlconverter file.mesh.xml out.mesh -l vertex.vlayout
```

Vertex.vlayout

```
struct Vertex : buffer[0]
{
    POSITION        : half4;
    TEXCOORD3       : ubyte4;
    TEXCOORD201     : short1, short2, short1;
    NORMALS         : qtangent|angle|normalxyz, half3;
};

struct Vertex : buffer[1]
{
    TANGENT         : float4;
    BINORMAL        : half3;
};
```

Create a position semantic on **source buffer 0**, 4 components. 16-byte float

Declaration order IS important

Flexible data layout: Ideal

```
C:\> ogrexmlconverter file.mesh.xml out.mesh -vl vertex.vlayout
```

Vertex.vlayout

```
struct Vertex : buffer[0]
{
    POSITION        : half4;
    TEXCOORD3       : ubyte4;
    TEXCOORD201     : short1, short2, short1;
    NORMALS         : qtangent|angle|normalxyz, half3;
};

struct Vertex : buffer[1]
{
    TANGENT         : float4;
    BINORMAL        : half3;
};
```

Take TEXCOORD3 of the xml file, convert it to four ubyte

It will now become texcoord #0 in the output because it's the first one to be declared

If texcoord had less than 4 components, warn and fill with zeroes. Warn also about possible truncation.

Flexible data layout: Ideal

```
C:\> ogrexmlconverter file.mesh.xml out.mesh -vl vertex.vlayout
```

Vertex.vlayout

```
struct Vertex : buffer[0]
{
    POSITION        : half4;
    TEXCOORD3       : ubyte4;
    TEXCOORD201     : short1, short2, short1;
    NORMALS         : qtangent|angle|normalxyz, half3;
};

struct Vertex : buffer[1]
{
    TANGENT         : float4;
    BINORMAL        : half3;
};
```

Take first value of TexCoord 2 and fill into X; the first two values of T.C. 0 to fill into YZ, & the first value of T.C. 1 to put into W.

The merge will output TexCoord 1 xyzw, short4.

Error if TC 0 had less than 2 outputs

The output must contain all the same base type (obvious)

Merging is very useful for freeing up vertex semantics. Makes handling shader permutations easier. Some GPUs aren't scalar and might boost perf.

Flexible data layout: Ideal

```
C:\> ogrexmlconverter file.mesh.xml out.mesh -vl vertex.vlayout
```

Vertex.vlayout

```
struct Vertex : buffer[0]
{
    POSITION        : half4;
    TEXCOORD3       : ubyte4;
    TEXCOORD201     : short1, short2, short1;
    NORMALS         : qtangent|angle|normalxyz, half;
};

struct Vertex : buffer[1]
{
    TANGENT         : float4;
    BINORMAL        : half3;
};
```

Store normal data, and choose compression scheme.

Last parameter (half) can be ignored for compressed schemes.

Can only specify base type, Element count is ignored i.e. 'float' & 'short' are the same as 'float3' or 'short4' respectively.

Flexible data layout: Ideal

```
C:\> ogrexmlconverter file.mesh.xml out.mesh -vl vertex.vlayout
```

Vertex.vlayout

```
struct Vertex : buffer[0]
{
    POSITION        : half4;
    TEXCOORD3       : ubyte4;
    TEXCOORD201     : short1, short2, short1;
    NORMALS         : qtangent|angle|normalxyz, half;
};

struct Vertex : buffer[1]
{
    TANGENT         : float4;
    BINORMAL        : half3;
};
```

Source buffer 1

If uses 4 components (i.e. float4) stores binormal parity in W. Else always assume 3 components (warn if not 3 or 4)

Always set to 3 components.

Warn if not so (and set to 3)

Flexible data layout

- Allows very thorough optimization
- Provide a group of presets for each platform
  - Shorts are faster than halfs in consoles
- No need for artist intervention
  - Write layout once, use it for every asset

Rendering huge environments

(so far 3 titles I've worked on)

What's wrong with this shader? (1)

```
uniform float3x4 viewProj;
uniform float3x4 worldMat3x4[NUM_MAT3X4]; //world_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( worldMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( viewProj, vPos );
//...
```

What's wrong with this shader? (1)

```
uniform float3x4 viewProj;
uniform float3x4 worldMat3x4[NUM_MAT3X4]; //world_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( worldMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( viewProj, vPos );
//...
```

Suppose the camera is at (6000, 6000, 6000), object is visible and close to the camera

What's wrong with this shader? (1)

```
uniform float3x4 viewProj;
uniform float3x4 worldMat3x4[NUM_MAT3X4]; //world_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( worldMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( viewProj, vPos );
//...
```

Suppose the camera is at (6000, 6000, 6000), object is visible and close to the camera

.41 = 6010 .42 = 6005 .43 = 6005

X = 6018 Y = 6015 Z = 6004

What's wrong with this shader? (1)

```
uniform float3x4 viewProj;
uniform float3x4 worldMat3x4[NUM_MAT3X4]; //world_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( worldMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( viewProj, vPos );
//...
```

Suppose the camera is at (6000, 6000, 6000), object is visible and close to the camera

.41 = 6010 .42 = 6005 .43 = 6005

X = 6018 Y = 6015 Z = 6004

These are all very large values:

Precision <u>SUCKS</u>.

✗Jittering

✗Shaking

Hard to see in a still frame. See it in motion. Polygons shake like in Playstation* 1 titles (which used fixed point numbers). It's 1996 again

My eye is popping out!

It's much worse in motion

What's wrong with this shader? (1)

- ✔ SceneManager::setCameraRelativeRendering is a **very** good paliative
  - ✗ Adds an 'if' on every rendered object → luckily predicts very well.
  - ✗ Some artifacts still present when far away.

What's wrong with this shader? (1)

SOLUTION:

```
uniform float3x4 worldViewProj;
uniform float3x4 localMat3x4[NUM_MAT3X4]; //local_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( localMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( worldViewProj, vPos );
//...
```

What's wrong with this shader? (1)

SOLUTION:

```
uniform float3x4 worldViewProj;
uniform float3x4 localMat3x4[NUM_MAT3X4]; //local_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( localMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( worldViewProj, vPos );
//...
```

- Idea is to work the animation in local space, then transform by wvp
- ✔ Performance: Saves concatenating the world matrix against each bone
- ✔ Numbers are always small. Precision win!
- Doesn't work for lighting calculations?

What's wrong with this shader? (1)

SOLUTION:

```
uniform float3x4 worldViewProj;
uniform float3x4 localMat3x4[NUM_MAT3X4]; //local_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( localMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( worldViewProj, vPos );
//...
```

•Idea is to work the animation in local space, then transform by wvp

✔Performance: Saves concatenating the world matrix against each bone

✔Numbers are always small. Precision win!

•Doesn't work for lighting calculations?

- Work in view space!
- Pass camera_matrix_array_3x4 → Animation in view space.
- Then multiply by Projection matrix instead of wvp.
- Doesn't save performance, but precision stays accurate :)

See Creating Vast Game Worlds (Emil Persson) [11] for more precision tips

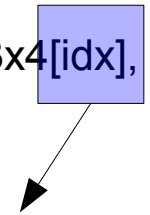
- Always construct inverse wvp matrices by doing the opposite calculations

What's wrong with this shader? (2)

```
uniform float3x4 viewProj;
uniform float3x4 worldMat3x4[NUM_MAT3X4]; //world_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( worldMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( viewProj, vPos );
//...
```

What's wrong with this shader? (2)

```
uniform float3x4 viewProj;
uniform float3x4 worldMat3x4[NUM_MAT3X4]; //world_matrix_array_3x4
//Skeletally animated object
int idx = input.blendIdx;
float4 vPos = float4( mul( worldMat3x4[idx], input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( viewProj, vPos );
//...
```



Shader constant waterfalling!!! :(

The higher the vertex count & indexing divergences of adjacent vertices, the slower the shader will run.[12]

When NUM_MAT3x4 is very high (>25), it can be a real bottleneck

Parallel operations running on the same shader unit with deverging indices must be serialized. It's fine if they all access the same entries (i.e. during lighting calculations in forward rendering)

HW Skinning through constant registers was the only way for VS 1.1 & 2.0

- DX10/11: Texture buffers to the rescue.
- DX9: Vertex Texture Fetch works like a charm on DX10 capable devices.
  - Current Ogre's HW VTF Instancing implementation proves it! We already do this! (guess who wrote it...)

HW Skinning through constant registers was the only way for VS 1.1 & 2.0

- DX10/11: Texture buffers to the rescue.
- DX9: Vertex Texture Fetch works like a charm on DX10 capable devices.
  - Current Ogre's HW VTF Instancing implementation proves it! We already do this! (guess who wrote it...)

No more 85-bone limit! → More bones can be crazy for games, but enables non-gaming applications (i.e. Cinematic tools).

Use similar system (VTF/Texture buffers) for **morph targets**!

✔Would make it straightforward to integrate to shader.

- Maximum of 4-24 active morph targets/blend shapes is stone age!!! (24 uses insane amount of bandwidth)
- Almost nobody ends up using it.

Medusa demo

Yeah..... 4 blend shapes....

Thanks to envydream.blogspot.com for taking the screenshot

What's wrong with this shader? (2)

SOLUTION:

```
uniform float3x4 worldViewProj;
uniform texture2D localMat3x4Tex; //Can be 1D & save 1 interpolator
//Skeletally animated object
int idx = input.blendIdx;
float3x4 mat3x4;
mat3x4[0] = tex2Dlod( localMat3x4Tex, (input.m03.xy).xyyy );
mat3x4[1] = tex2Dlod( localMat3x4Tex, (input.m03.xy + float2( 1.0f / texWidth, 0 ) ).xyyy );
mat3x4[2] = tex2Dlod( localMat3x4Tex, (input.m03.xy + float2( 2.0f / texWidth, 0 ) ).xyyy );
float4 vPos = float4( mul( mat3x4, input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( worldViewProj, vPos );
//...
```

What's wrong with this shader? (2)

SOLUTION:

```
uniform float3x4 worldViewProj;
uniform texture2D localMat3x4Tex; //Can be 1D & save 1 interpolator
//Skeletally animated object
int idx = input.blendIdx;
float3x4 mat3x4;
mat3x4[0] = tex2Dlod( localMat3x4Tex, (input.m03.xy).xyyy );
mat3x4[1] = tex2Dlod( localMat3x4Tex, (input.m03.xy + float2( 1.0f / texWidth, 0 ) ).xyyy );
mat3x4[2] = tex2Dlod( localMat3x4Tex, (input.m03.xy + float2( 2.0f / texWidth, 0 ) ).xyyy );
float4 vPos = float4( mul( mat3x4, input.pos ).xyz, 1.0f ) * blendWght;
outPos = mul( worldViewProj, vPos );
//...
```

On +G80 & HD 2000 HW texture cache works great

DX10/11 & GL would use a different code path (no need for texWidth/invTexWidth with tbuffers)

Any other SM 3.0 GPU that could run this would be G60 & G70 but... SLOOOW

- ~~Too many cache misses :(~~
- ~~Inefficient Scene traversal & processing~~
- ~~Fat, unflexible, vertex format~~
- Fixed functions vs programmable shaders
  - “setFog”, etc

Fixed function

Begginers are often confused: Using “setFog & setSpecular” works. Then they decide to used shaders. And everything stops working all of a sudden.

Fixed function

Begginers are often confused: Using “setFog & setSpecular” works. Then they decide to used shaders. And everything stops working all of a sudden.

- Move all functionality FF to “states” → Similar to D3D10 state concepts.
  - The idea → move functionality out of the SceneManager (& possibly from some materials)

Fixed function

Begginers are often confused: Using “setFog & setSpecular” works. Then they decide to used shaders. And everything stops working all of a sudden.

- Move all functionality FF to “states” → Similar to D3D10 state concepts.
  - The idea → move functionality out of the SceneManager (& possibly from some materials)
  - The developer would create a couple of states, and apply them
  - Easier to manage redundant state changes
  - Allows FF emulation & cache through RRTS or similar
  - Works on platform where FF is the norm
  - Completely optional

- RTSS could evolve into something more node-like. Artists love tweaking UI nodes (specially UDK*'s node system).

Managing shaders

RTSS: Works out of the box FF emulation. That's it's original purpose.

New possible goals:

- Manage shader permutations → Assign chunks of custom shader code as nodes.
  - Each node has named inputs and outputs to be passed as function parameters
  - Example: Keep vertex transform & shadow code, but swap BDRFs.
  - Example: Keep everything, but use instancing's transformation code (one weight).
  - Very similar to how it already works, but allowing custom shader code.
- Visual editor to set up the nodes (WYSIWYG)

Embracing DirectX 11 & OGL 4.3

“Everything that can be done with D3D11 can be done in D3D9”

(This is usually believing D3D11 is like 9 with tesselation)

Tesselation?

New compression formats?

HDR in compute shaders?

MSAA custom resolves?

MRT with MSAA?

More MRT targets?

Depth textures?

Texture buffers?

Tesselation?

New compression formats?

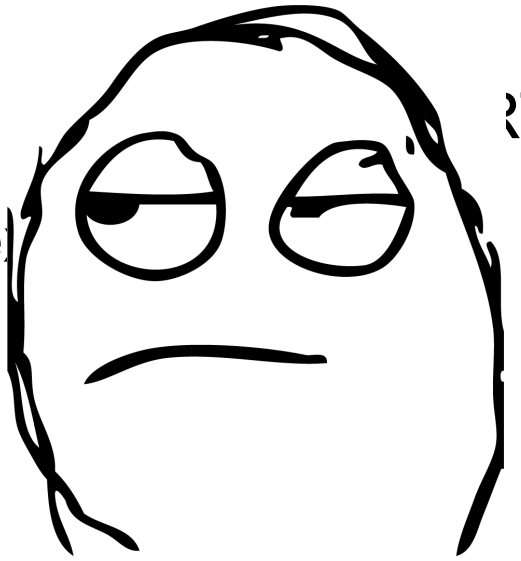
HDR in compute shaders?

MSAA custom resolves?

MRT with MSAA?

RT targets?

Depth te



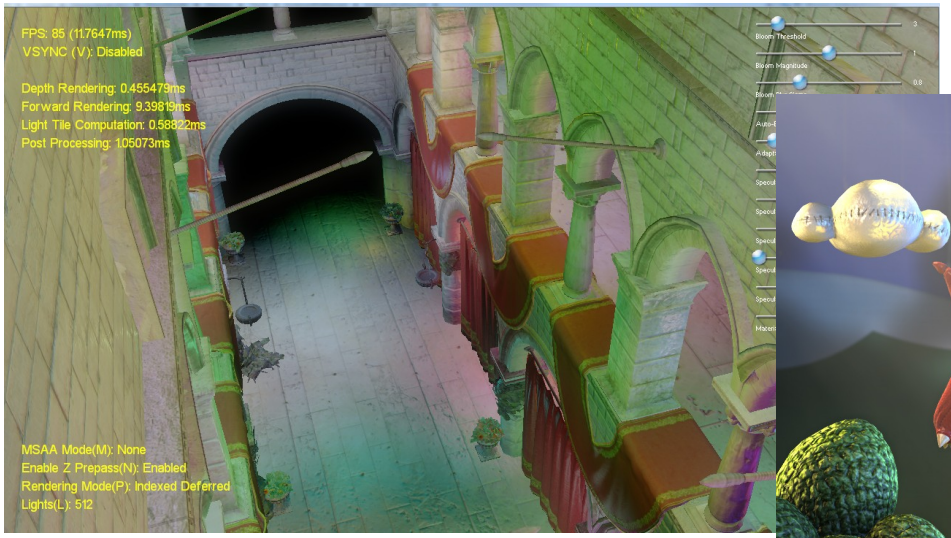
Texture buffers?

Naahh.....

What's hot in D3D11? (aka. You can't do these in D3D9)

Light indexed deferred & Cluster forward shading [19] → Needs atomics and UAVs

- AMD's Leo Demo.
- MJP's [17] Demo
- See also Deferred Rendering for Current and Future Rendering Pipelines [18]



MJP's demo

AMD's Leo demo

What's hot in D3D11? (aka. You can't do these in D3D9)

Voxel based Real time Global Illumination → Needs compute shaders

- CryEngine 3 [20] & UDK 4 [21] already do this
  - See also Cascaded Light Propagation Volumes for Indirect Illumination [22]
- NVIDIA's Voxel cone tracing & Sparse Voxel Octree [23]. → Not fully real time. Needs a preprocessing step. But has it's niche and it's multi-bounce.

NVIDIA's Voxel cone tracing demo

UDK 4's Elemental demo

What's hot in D3D11? (& GL 4.3)

Light indexed deferred & Cluster forward shading

- It's actually forward rendering, but works like deferred.
- Moving the tile creation to the CPU, it is doable on D3D10 level hardware
  - ✔ Many lights
  - ✔ MSAA compatible
  - ✔ Alpha blending compatible
  - ✔ Faster than deferred shading
  - ✔ Multiple materials/BDRFs

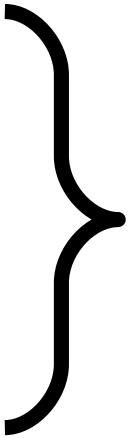
Real time Global Illumination

- **Do I need to say more?**
- UDK & CryEngine's approach requires artist's tweaking of which objects are occluders for optimal results. Small objects are removed from the calculations.

What's hot in D3D11? (& GL 4.3)

Light indexed deferred & Cluster forward shading

- It's actually forward rendering, but works like deferred.
- Moving the tile creation to the CPU, it is doable on D3D10 level hardware
  - ✔ Many lights
  - ✔ MSAA compatible
  - ✔ Alpha blending compatible
  - ✔ Faster than deferred shading
  - ✔ Multiple materials/BDRFs



I'm no fortune teller but this looks like the future in 2 years

Real time Global Illumination

- **Do I need to say more?**
- UDK & CryEngine's approach requires artist's tweaking of which objects are occluders for optimal results. Small objects are removed from the calculations.

What's hot in D3D11? (& GL 4.3)

UAVs & atomic counters are awesome.

Ultra fast Bokeh depth of field (4th technique being described) [24] is an example of other possible uses. Sprite-based compositing (i.e. Lens flares & Bokeh DoF) is now possible.

All the other features (depth textures, custom AA resolves, texture arrays, etc) are nice bonus, but nothing that can't really be done in D3D9 or that brings a lot of quality difference.

BONUS SLIDES!

Data Oriented Design (DOD) vs Object Oriented Design (OOD)

OOD is well known. DOD is about coding around data, and cache efficiency

Most common example is **virtual**

- Wide spread OOD pattern all over OGRE → Good. But overused

Data Oriented Design (DOD) vs Object Oriented Design (OOD)

OOD is well known. DOD is about coding around data, and cache efficiency

Most common example is **virtual**

- Wide spread OOD pattern all over OGRE → Good. But overused

*Do we **really** need these to be virtual?*

```
virtual const Vector3& SceneNode::getPosition();
virtual SceneNode::setPosition( const Vector3 &newPos );
```

OOD vs DOD

But it's flexible! Allowing derived types to modify their behavior!

- Yeah, and really slow.
- But there's a lot of possible usages. This is open source!
- What do other (licensable) engines do???
  - Pay for source code → **Modify** source code
  - No virtual in commonly called functions.

OOD vs DOD

But it's flexible! Allowing derived types to modify their behavior!

- Yeah, and really slow.
- But there's a lot of possible usages. This is open source!
- What do other (licensable) engines do???
  - Pay for source code → **Modify** source code
  - No virtual in commonly called functions.

Open Source is the same, except for the pay for source code part[*] :)

Flexibility through modification, not virtual overloading

→ **But can this get annoying?**

[*] Ok, Stallman won't agree. Yes, yes, we know. Open source is about freedom

OOD vs DOD

YES! It can get annoying!

SOLUTION: Flexibility levels!

```
#if OGRE_FLEXIBILITY_LEVEL >= 0
    #define virtual_l0 virtual
#else
    #define virtual_l0
#endif
#if OGRE_FLEXIBILITY_LEVEL >= 1
    #define virtual_l1 virtual
#endif
//(...) Up to level 2 probably

virtual_l2 const Vector3& SceneNode::getPosition(); //Less likely to be overloaded
virtual_l1 SceneNode::setPosition( const Vector3& ); //More likely to be overloaded
```

OOD vs DOD: Flexibility levels

- Get best of both worlds: Let the users decide *how* they modify the code.
  - If they want to overload, let them overload
- Higher level for less overloaded functions or highly called ones.

- Breaks ABI compatibility though.
  - OgreMain.dll built with different flexibility levels can't be mixed.
  - We can't mix builds w/ & w/out Boost anyway....
- Default shipping DLL: OGRE_FLEXIBILITY_LEVEL = -1 (disabled)

- Still use 'virtual' for functions that are internally (by design) virtual
  - Examples: InstanceBatch techniques. Particle systems.
  - Try to use DOD for those when possible (see next slide)

OOD vs DOD

OOD approach:

```
virtual FooClass::foo() { /* work on FooClass 'this' */ }
for( size_t x=0; i<count; ++i )
    object[i]->foo();
```

OOD vs DOD

OOD approach:

```
virtual FooClass::foo() { /* work on FooClass 'this' */ }
for( size_t x=0; i<count; ++i )
        object[i]->foo();
```

DOD approach:

```
//Static like behavior
virtual FooClass::foo( FooClass **fooInstancesInOut, size_t count ) const
{
        for( size_t x=0; i<count; ++i )
                /* work on 'fooInstancesInOut[i]' */
}

objectMgr->foo( objects, count );
```

OOD vs DOD

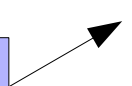
OOD approach:

```
virtual FooClass::foo() { /* work on FooClass 'this' */ }
for( size_t x=0; i<count; ++i )
        object[i]->foo();
```

DOD approach:

```
//Static like behavior
virtual FooClass::foo( FooClass **fooInstancesInOut, size_t count ) const
{
        for( size_t x=0; i<count; ++i )
                /* work on 'fooInstancesInOut[i]' */
}
```

```
objectMgr->foo( objects, count );
```



Virtual vtable is evaluated once!

OOD vs DOD

- OOD virtuals in x86/x64 are cheap (thanks to branch predictors & OoOE)
  - Not so much in all the other target architectures
  - Branch predictors are expensive & consume a lot of power.
    - Don't hope they'll appear in mobile anytime soon.
  - Same with Out of Order Execution (OoOE)

OOD & DOD... 'vs'?

DOD is about **data layout & access patterns.**

OOD is about **coding style & relationship between constructs** called 'objects'.

In theory DOD & OOD aren't contradictory at all. OOD is just easier for humans to understand & visualize.

But unfortunately C++ mixes both concepts together.

- There's no way to specify a different data layout other than variable's declaration order.
- There's no way to *let the compiler know or hint* we want our OOD-looking code full of virtuals to translate into DOD-friendly assembly code with little virtuals

....may be in some distant future

64-bit readiness

It's about time we start thinking in default 64-bit builds

- Extra 8 xmm registers may come in handy

Portability issues? Pointer truncation bugs?

- Bruce Dawson [13] (Valve) to the rescue!
- http://randomascii.wordpress.com/2012/02/14/64-bit-made-easy/

64-bit readiness

It's about time we start thinking in default 64-bit builds

- Extra 8 xmm registers may come in handy

Portability issues? Pointer truncation bugs?

- Bruce Dawson [13] (Valve) to the rescue!
- http://randomascii.wordpress.com/2012/02/14/64-bit-made-easy/

Basic idea

- At start-up allocate large space of virtual addreses, but small chunks of memory
  - Overhead is very low
- Exhaust the first 4 GB address-range. All in-engine allocations must now use 64-bit
- Pointer truncation and similar problems will cause crashes in no time!
- Beware of the side effects (i.e. App Verifier) described in the site.
- Even more 64-bit info:
  http://software.intel.com/en-us/blogs/2011/07/07/all-about-64-bit-programming-in-one-place/

```
void ReserveBottomMemory()
{
#ifdef _WIN64
    static bool s_initialized = false;
    if ( s_initialized )
        return;
    s_initialized = true;

    // Start by reserving large blocks of address space, and then
    // gradually reduce the size in order to capture all of the
    // fragments. Technically we should continue down to 64 KB but
    // stopping at 1 MB is sufficient to keep most allocators out.

    const size_t LOW_MEM_LINE = 0x100000000LL;
    size_t totalReservation = 0;
    size_t numVAllocs = 0;
    size_t numHeapAllocs = 0;
    size_t oneMB = 1024 * 1024;
    for (size_t size = 256 * oneMB; size >= oneMB; size /= 2)
    {
        for (;;)
        {
            void* p = VirtualAlloc(0, size, MEM_RESERVE, PAGE_NOACCESS);
            if (!p)
                break;

            if ((size_t)p >= LOW_MEM_LINE)
            {
                // We don't need this memory, so release it completely.
                VirtualFree(p, 0, MEM_RELEASE);
                break;
            }

            totalReservation += size;
            ++numVAllocs;
        }
    }

    // Now repeat the same process but making heap allocations, to use up
    // the already reserved heap blocks that are below the 4 GB line.
```

```
HANDLE heap = GetProcessHeap();
for (size_t blockSize = 64 * 1024; blockSize >= 16; blockSize /= 2)
{
    for (;;)
    {
        void* p = HeapAlloc(heap, 0, blockSize);
        if (!p)
            break;

        if ((size_t)p >= LOW_MEM_LINE)
        {
            // We don't need this memory, so release it completely.
            HeapFree(heap, 0, p);
            break;
        }

        totalReservation += blockSize;
        ++numHeapAllocs;
    }
}

// Perversely enough the CRT doesn't use the process heap. Suck up
// the memory the CRT heap has already reserved.
for (size_t blockSize = 64 * 1024; blockSize >= 16; blockSize /= 2)
{
    for (;;)
    {
        void* p = malloc(blockSize);
        if (!p)
            break;

        if ((size_t)p >= LOW_MEM_LINE)
        {
            // We don't need this memory, so release it completely.
            free(p);
            break;
        }

        totalReservation += blockSize;
        ++numHeapAllocs;
```

```
        }
    }

    // Print diagnostics showing how many allocations we had to make in
    // order to reserve all of low memory, typically less than 200.
    char buffer[1000];
    sprintf_s(buffer, "Reserved %1.3f MB (%d vallocs,"
                      "%d heap allocs) of low-memory.\n",
            totalReservation / (1024 * 1024.0),
            (int)numVAllocs, (int)numHeapAllocs);
    OutputDebugStringA(buffer);
#endif
}
```

Unmodified code. **Thanks to Bruce Dawson!**, reproduced with permission

- http://randomascii.wordpress.com/2012/02/14/64-bit-made-easy/

Call ReserveBottomMemory right at start up of the process!

Legal

Matías Nazareth Goldberg
@matiasgoldberg

References

[1] CPU Caches and Why You Care, Scott Meyers, Ph.D.; ACCU 2011 Conference.
http://www.aristeia.com/TalkNotes/ACCU2011_CPUCaches.pdf

[2] Typical C++ Bullshit, Mike Acton.
http://macton.posterous.com/roundup-recent-sketches-on-concurrency-data-d#

[3] Culling the Battlefield, Daniel Colling (DICE), GDC 2011.
http://publications.dice.se/attachments/CullingTheBattlefield.pdf

[4] Pitfalls of Object Oriented Programming, Tony Albrecht (SCEE), GCAP 09
http://research.scee.net/files/presentations/gcapaustralia09/Pitfalls_of_Object_Oriented_Programming_GCAP_

[5] “Down With fcmp: Conditional Moves For Branchless Math”, Elan Rusky (Valve)
http://assemblyrequired.crashworks.org/2009/01/04/fcmp-conditional-moves-for-branchless-math/

[6] Understanding the Second-Generation of Behavior Trees, Alex J. Champandard, 2012, #AltDevConf http://aigamedev.com/insider/tutorial/second-generation-bt/

[7] Intel® 64 and IA-32 Architectures Optimization Reference Manual
http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manua

[8] Practical Occlusion Culling in Killzone 3, Michael Valient, SIGGRAPH 2011
http://www.guerrilla-games.com/presentations/Siggraph2011_MichalValient_OcclusionInKillzone3.pptx

[9] Secrets of CryENGINE 3 Graphics Technology, Kasyan – Schulz – Sousa, SIGGRAPH 2011
http://www.crytek.com/cryengine/presentations/secrets-of-cryengine-3-graphics-technology

References

[10] Spherical Skinning with Dual-Quaternions and QTangents, Ivo Zoltan Frey, SIGRAPH 2011 Vancounver. http://www.crytek.com/download/izfrey_siggraph2011.ppt

[11] Creating Vast Game Worlds, Emil Persson, SIGGRAPH 2012 http://www.humus.name/Articles/Persson_CreatingVastGameWorlds.pptx

[12] Seven Ways to Skin a Mesh: Character Skinning Revisited for Modern GPUs, Matt Lee, Gamefest Unplugged (Europe) 2007

[13] 64-Bit made easy, Bruce Dawson, 2012 http://randomascii.wordpress.com/2012/02/14/64-bit-made-easy/

[14] Multi-core is Here! But How Do You Resolve Data Bottlenecks in PC Games?, Michael Wall (AMD), GDC 2008 http://developer.amd.com/wordpress/media/2012/10/AMD_GDC_2008_MW.pdf

[15] A sprintf that isn't as ugly, by Tom Forsyth, 2011, home.comcast.net/~tom_forsyth/blog.wiki.html

[16] Sparse-world storage formats, by Tom Forsyth, 2012, home.comcast.net/~tom_forsyth/blog.wiki.html

[17] Light Indexed Deferred Rendering, Matt Pettineo, 2012 http://mynameismjp.wordpress.com/2012/03/31/light-indexed-deferred-rendering/

[18] Deferred Rendering for Current and Future Rendering Pipelines, Andrew Lauritzen, SIGGRAPH 2010 http://software.intel.com/en-us/articles/deferred-rendering-for-current-and-future-rendering-pipelines/

References

[19] Clustered Deferred and Forward Shading, Olsson, Billeter, Assarsson, 2012
http://www.cse.chalmers.se/~olaolss/main_frame.php?contents=publication&id=clustered_shading

[20] Real-time Diffuse Global Illumination in CryENGINE 3, Kaplanyan. SIGGRAPH 2010
http://www.crytek.com/cryengine/presentations/real-time-diffuse-global-illumination-in-cryengine-3

[21] The Technology Behind the “Unreal Engine 4 Elemental demo”, Martin Mittring, SIGGRAPH 2012
http://www.unrealengine.com/files/misc/The_Technology_Behind_the_Elemental_Demo_16x9_(2).pdf

[22] Cascaded Light Propagation Volumes for Indirect Illumination, Kaplanyan & Dachsbacher, SIGGRAPH 2010
http://www.crytek.com/cryengine/cryengine3/presentations/cascaded-light-propagation-volumes-for-real-time-in

[23] Voxel Cone Tracing and Sparse Voxel Octree for Real-time Global Illumination, Cyril Crassin, 2012
http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/SB134-Voxel-Cone-Tracing-Octree

[24] How To Fake Bokeh (And Make It Look Pretty Good), Matt Pettineo, 2012,
http://mynameismjp.wordpress.com/2011/02/28/bokeh/