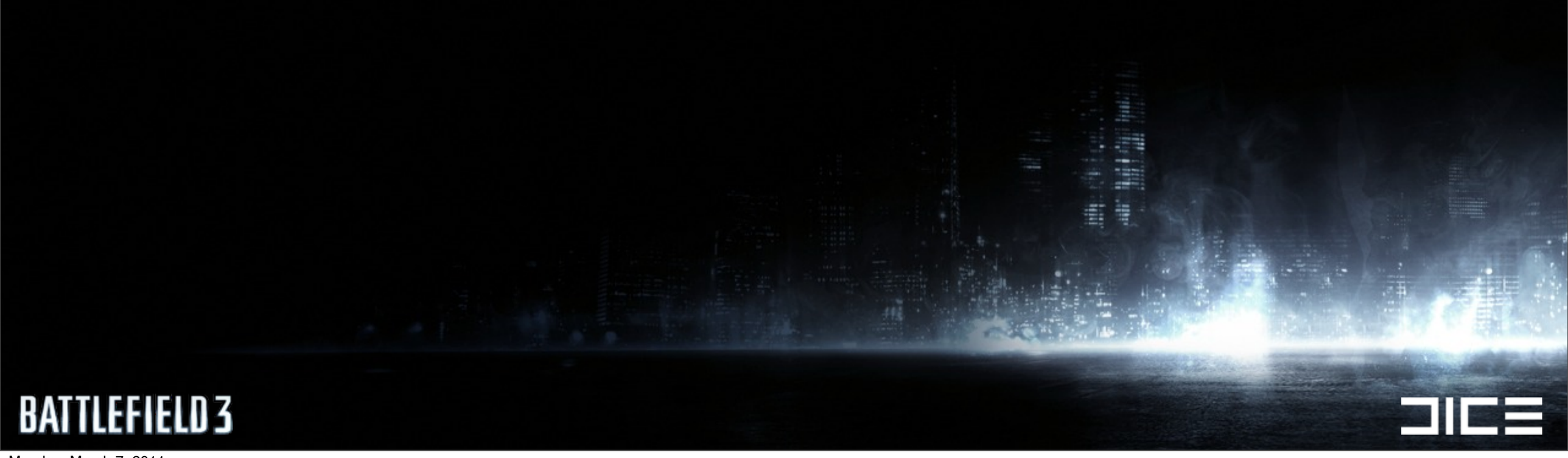
BATTLEFIELD 3

Culling the Battlefield

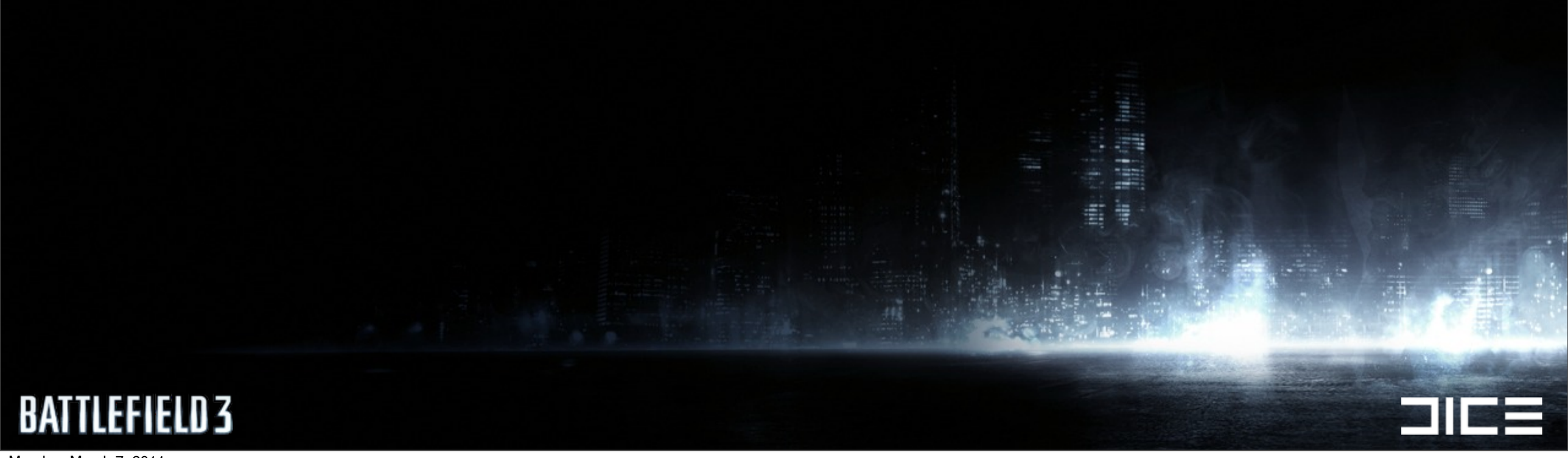
Daniel Collin (DICE)

Overview



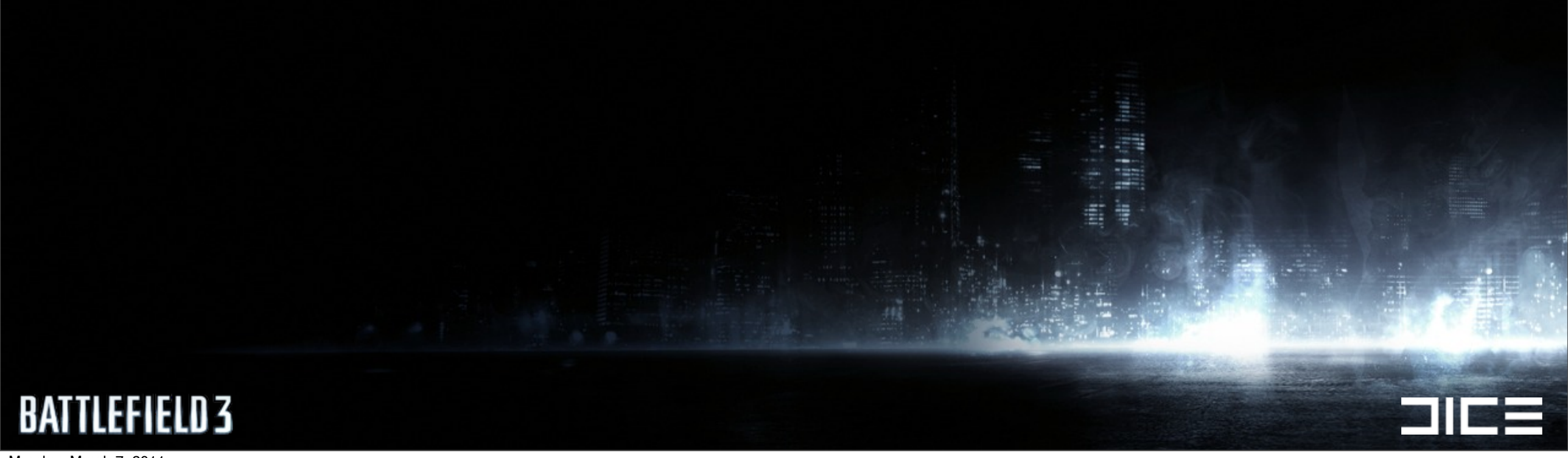
Overview

- Background



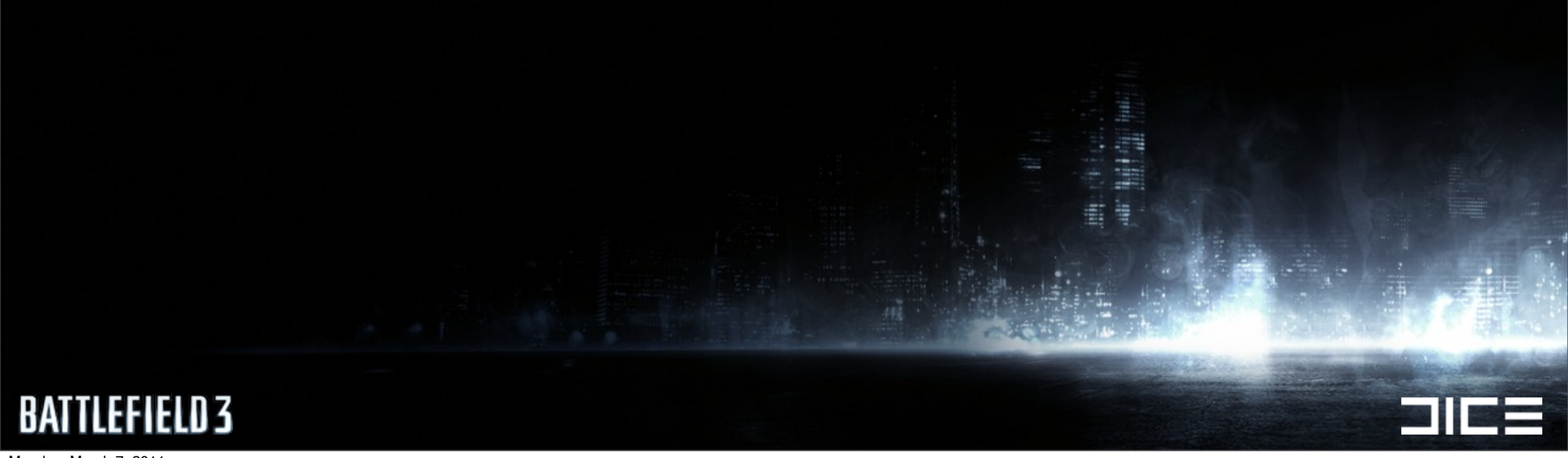
Overview

- Background
- Why rewrite it?



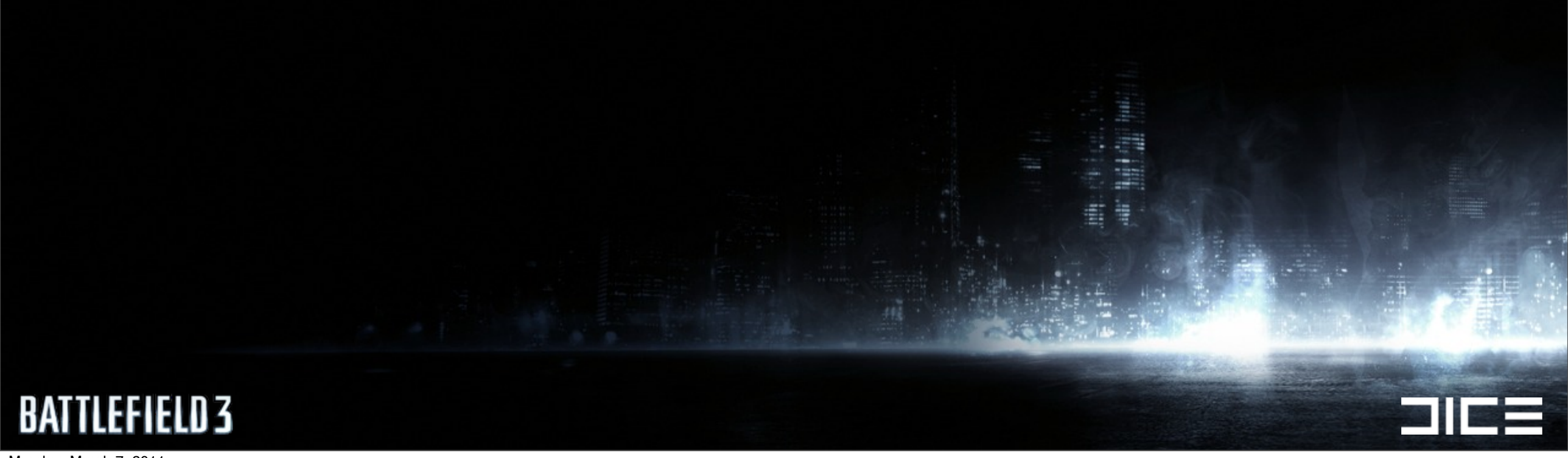
Overview

- Background
- Why rewrite it?
- Requirements



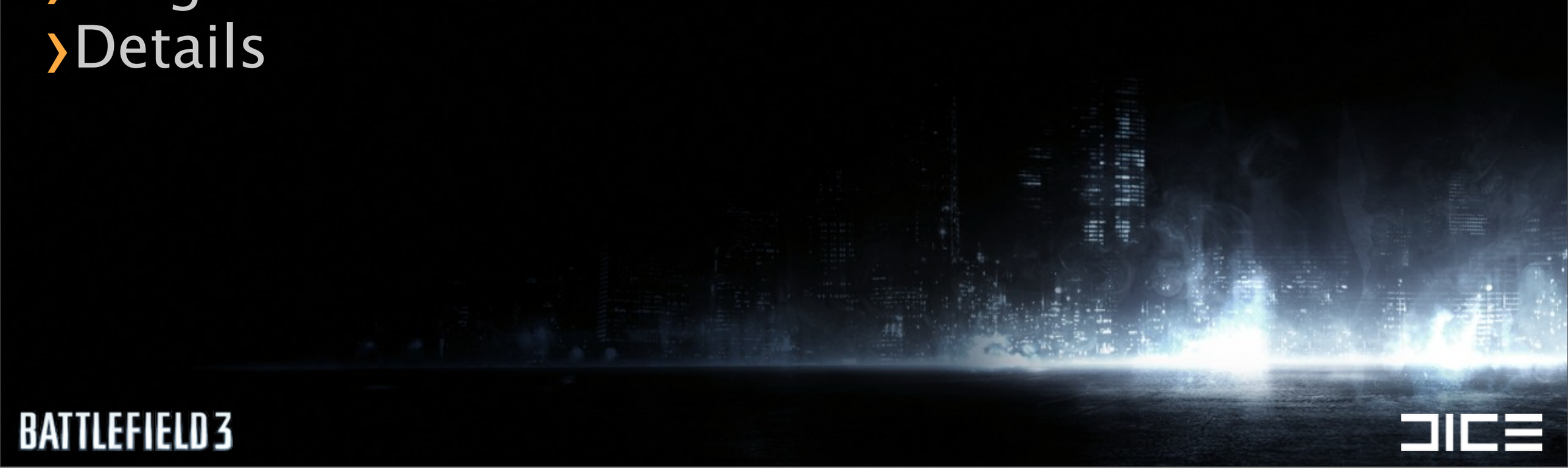
Overview

- Background
- Why rewrite it?
- Requirements
- Target hardware



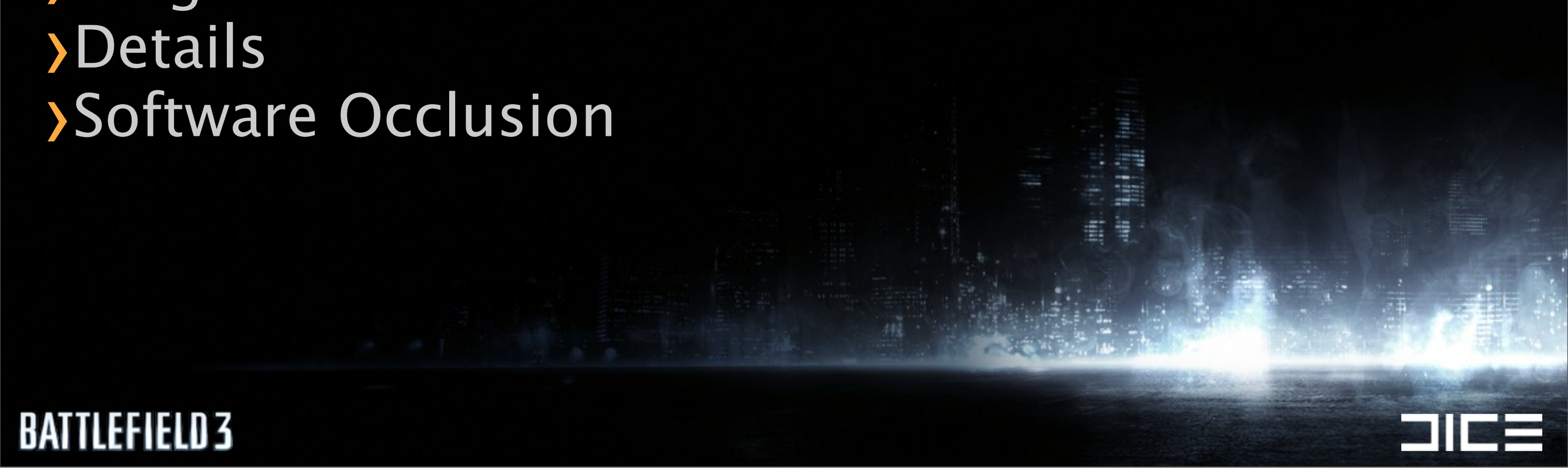
Overview

- Background
- Why rewrite it?
- Requirements
- Target hardware
- Details



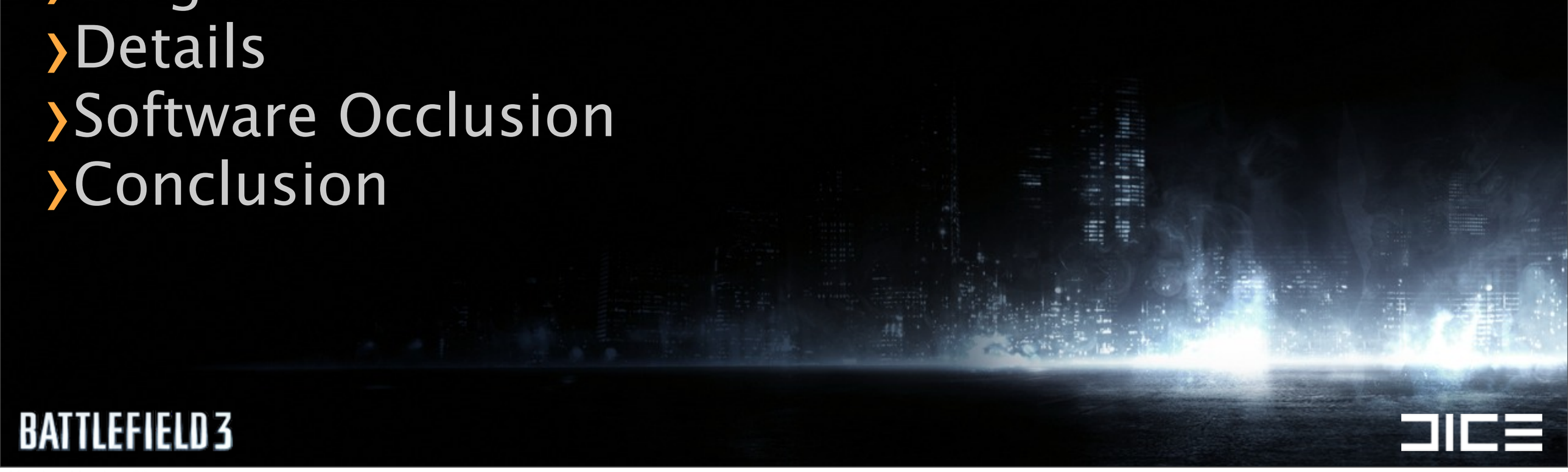
Overview

- Background
- Why rewrite it?
- Requirements
- Target hardware
- Details
- Software Occlusion

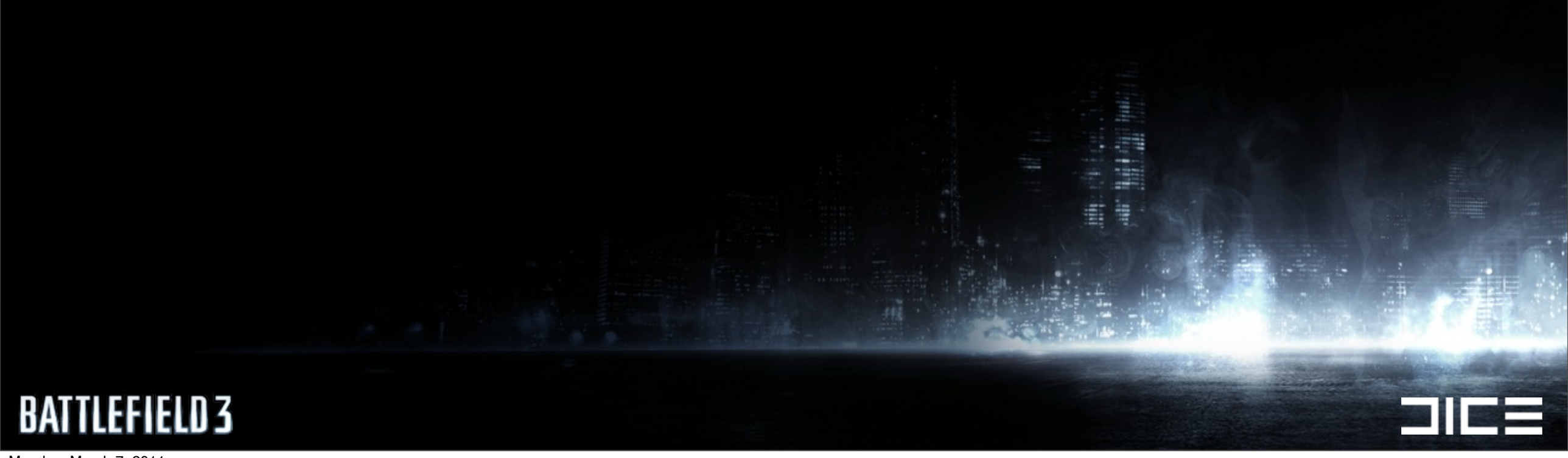


Overview

- Background
- Why rewrite it?
- Requirements
- Target hardware
- Details
- Software Occlusion
- Conclusion

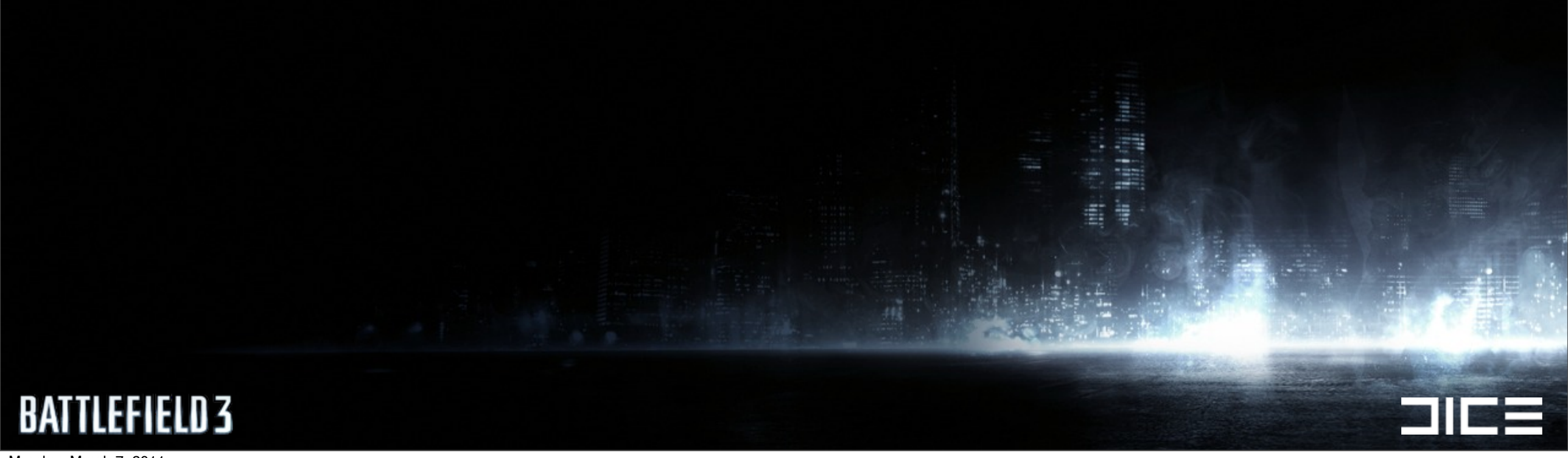


Background of the old culling



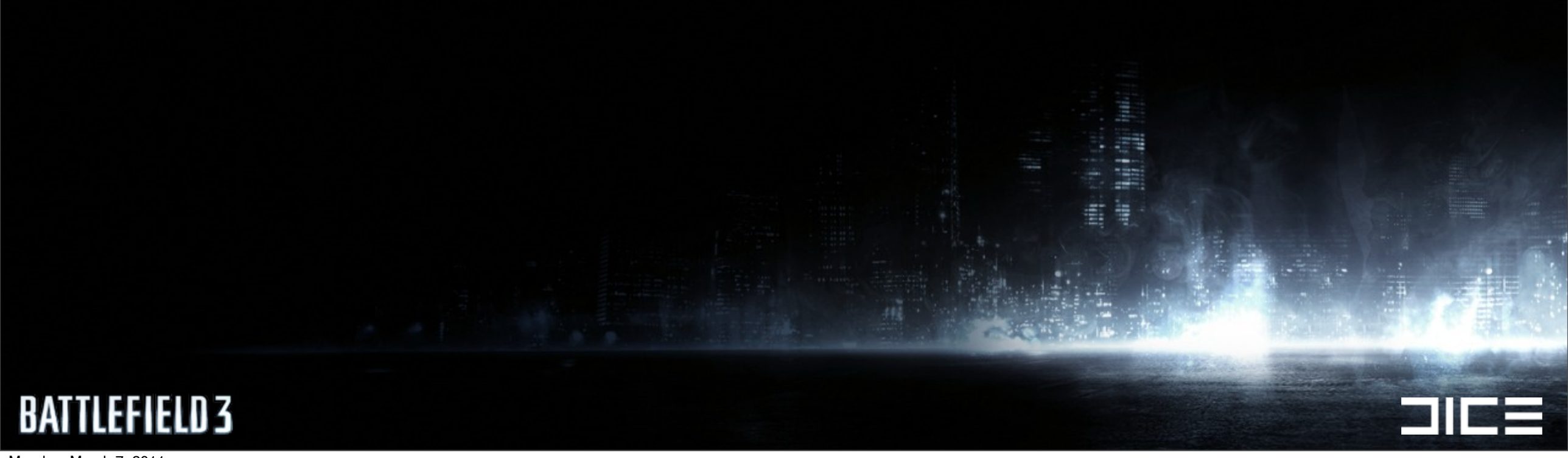
Background of the old culling

- Hierarchical Sphere Trees



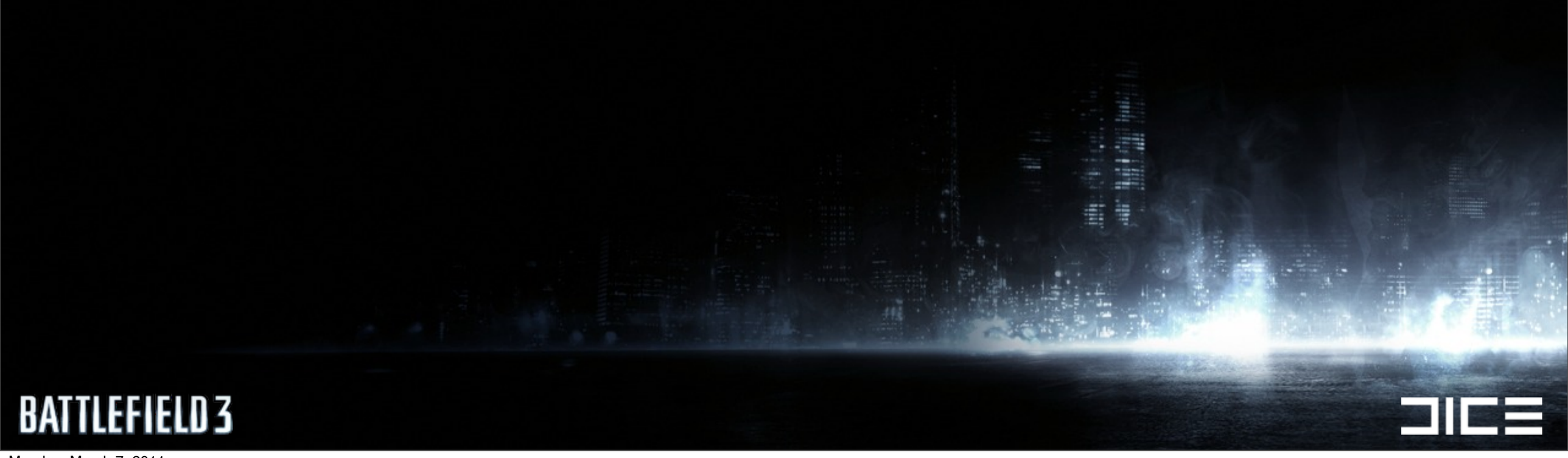
Background of the old culling

- Hierarchical Sphere Trees
- StaticCullTree

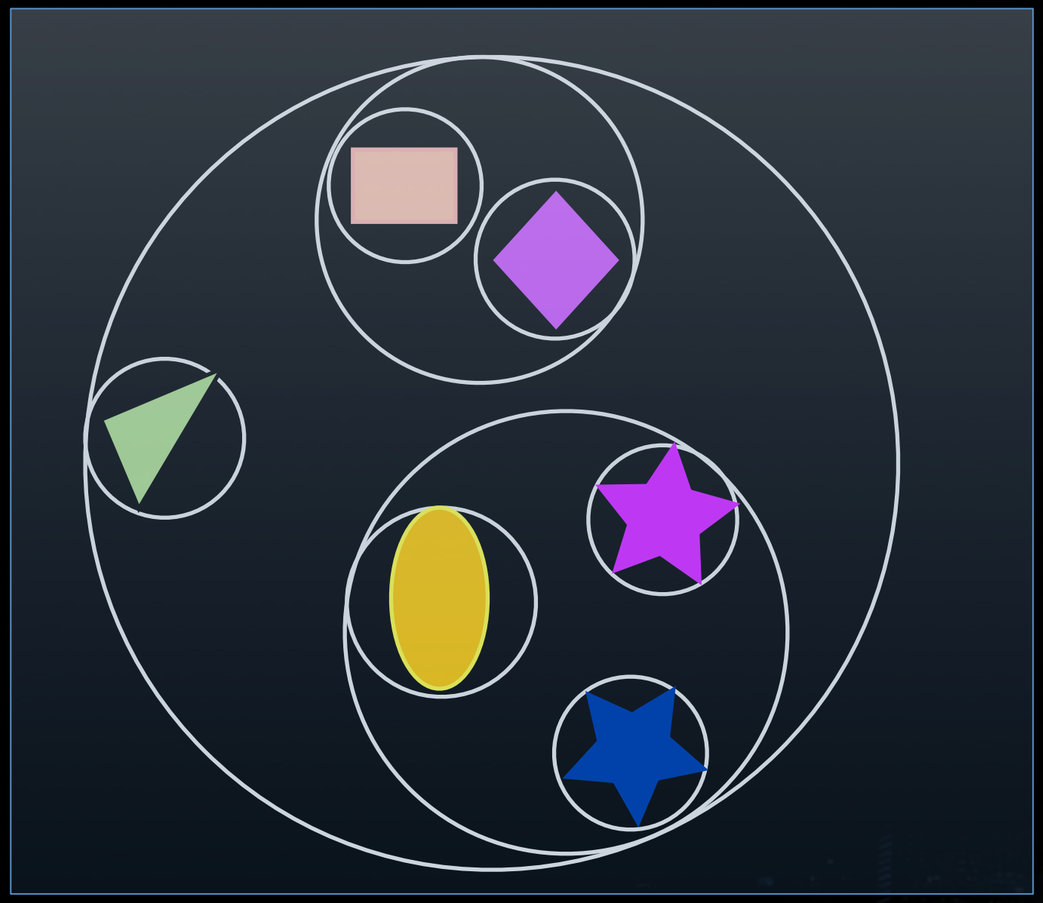


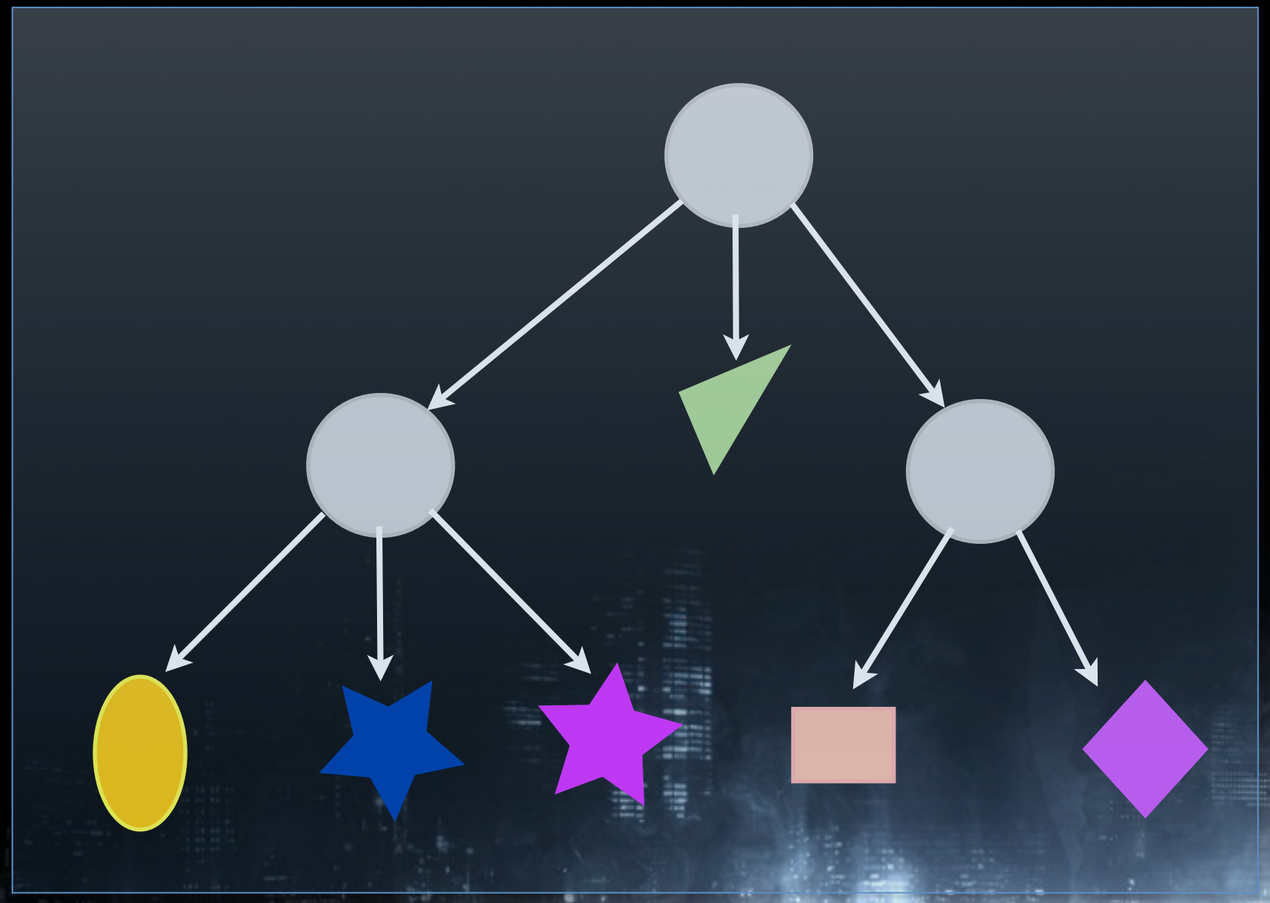
Background of the old culling

- Hierarchical Sphere Trees
- StaticCullTree
- DynamicCullTree

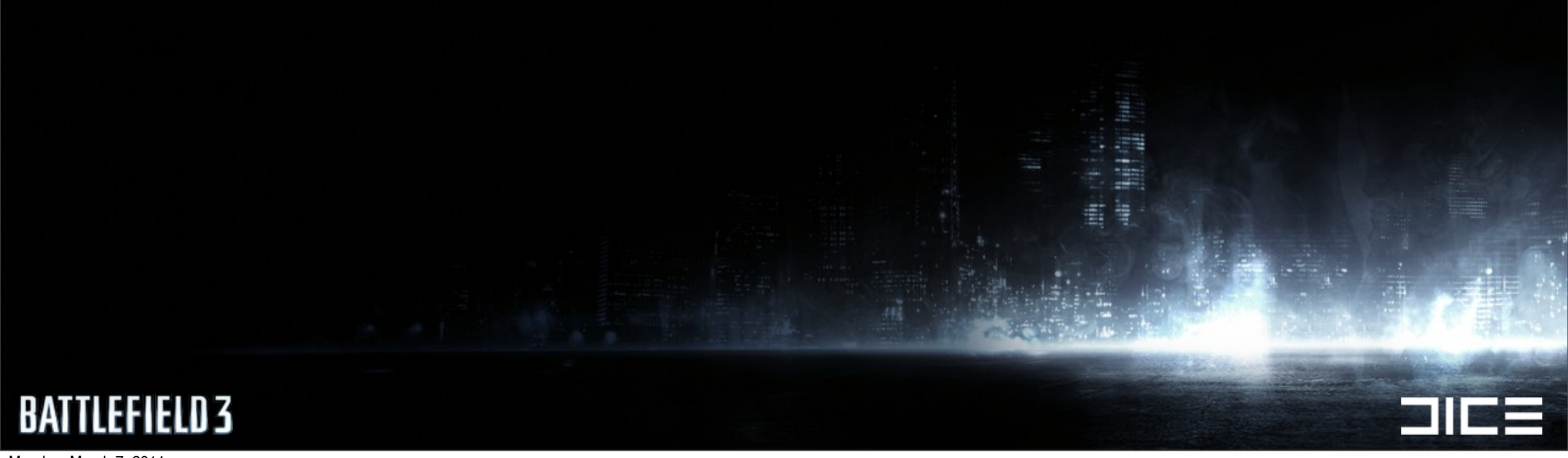


Background of the old culling



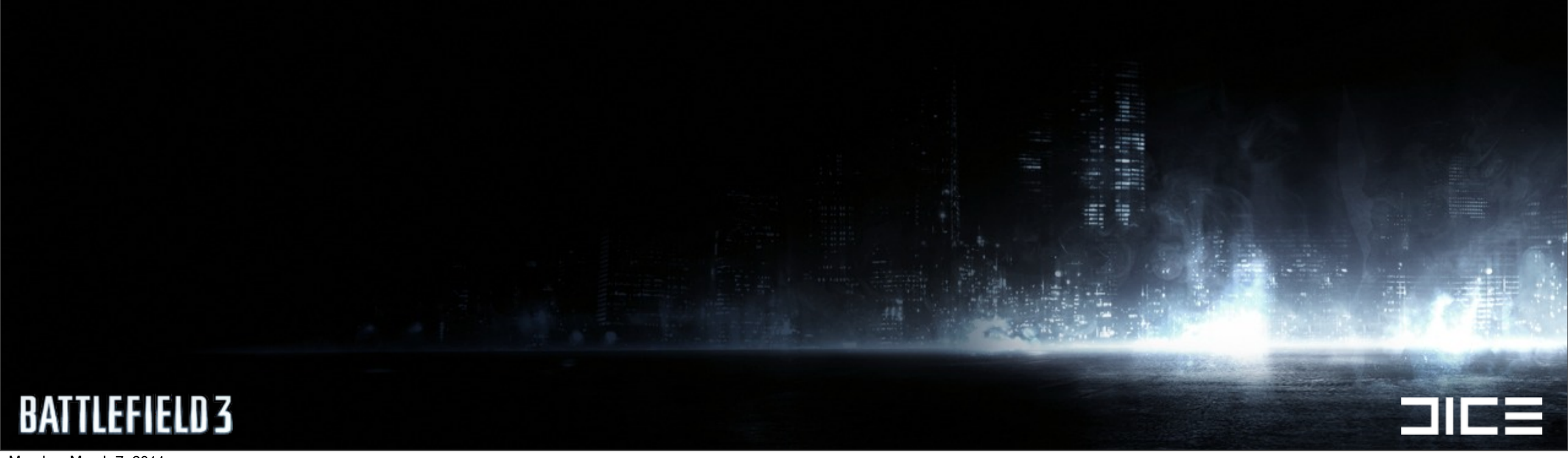


Why rewrite it?



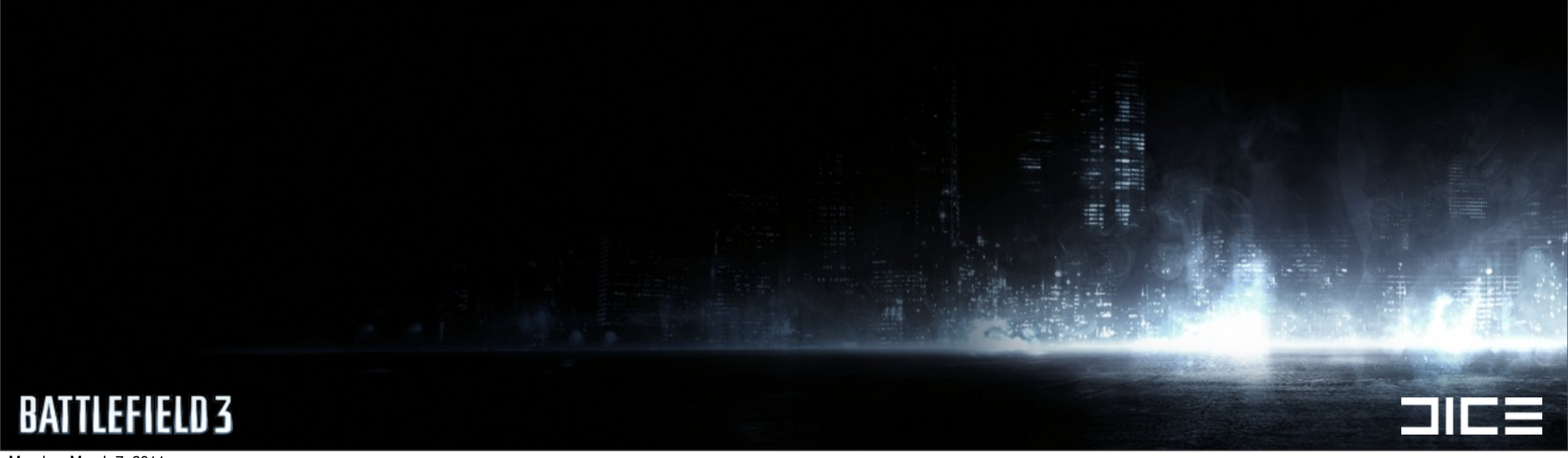
Why rewrite it?

- DynamicCullTree scaling



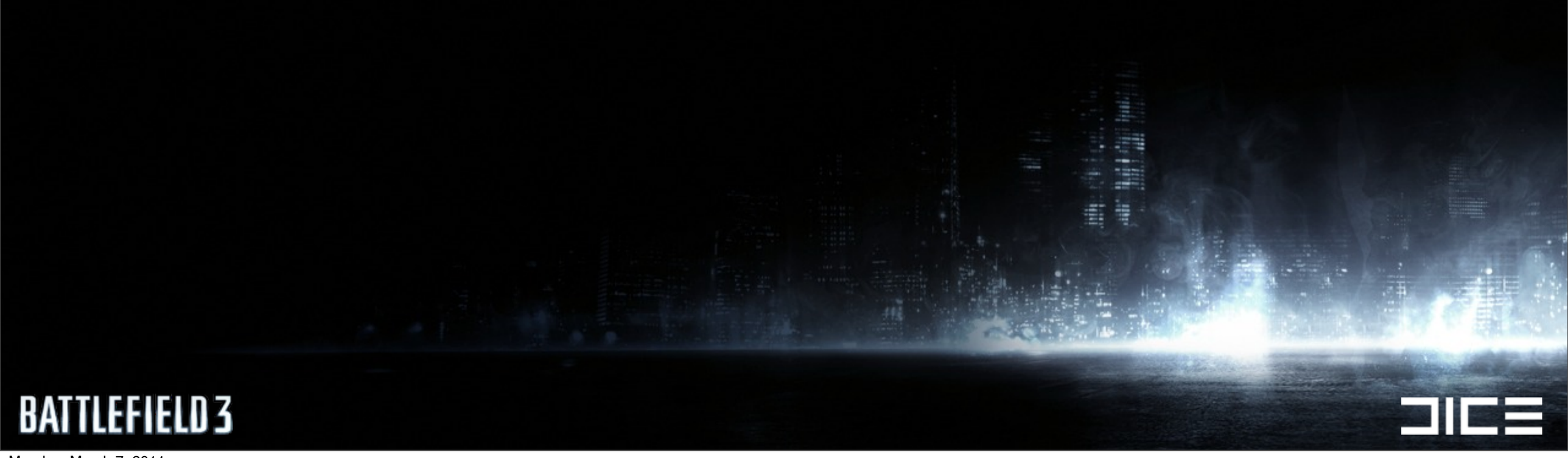
Why rewrite it?

- DynamicCullTree scaling
- Sub-levels



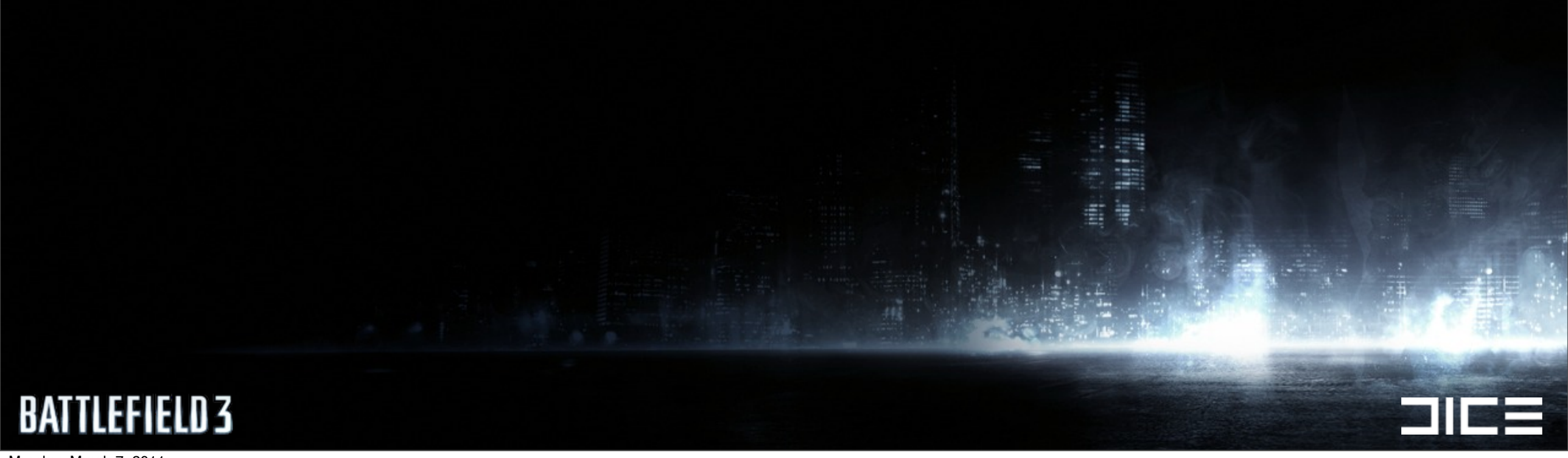
Why rewrite it?

- DynamicCullTree scaling
- Sub-levels
- Pipeline dependencies



Why rewrite it?

- DynamicCullTree scaling
- Sub-levels
- Pipeline dependencies
- Hard to scale

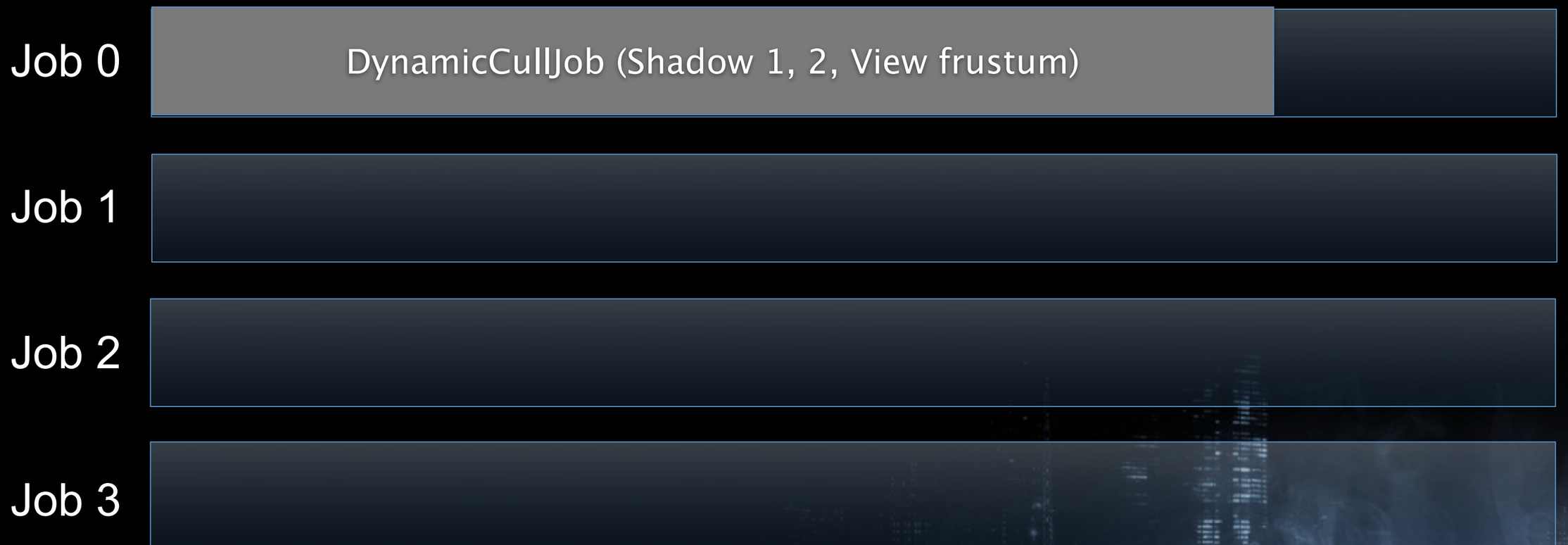


Why rewrite it?

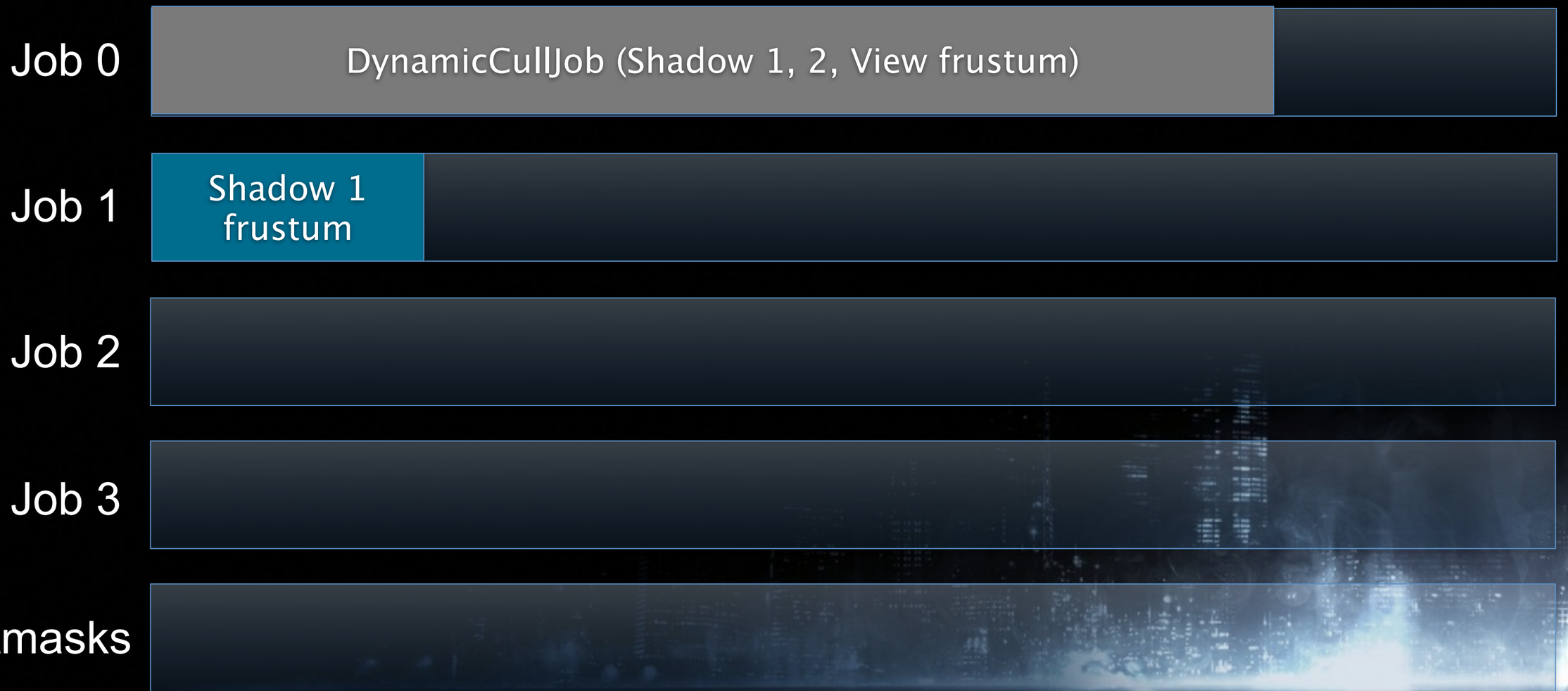
- DynamicCullTree scaling
- Sub-levels
- Pipeline dependencies
- Hard to scale
- One job per frustum

Job graph (Old Culling)

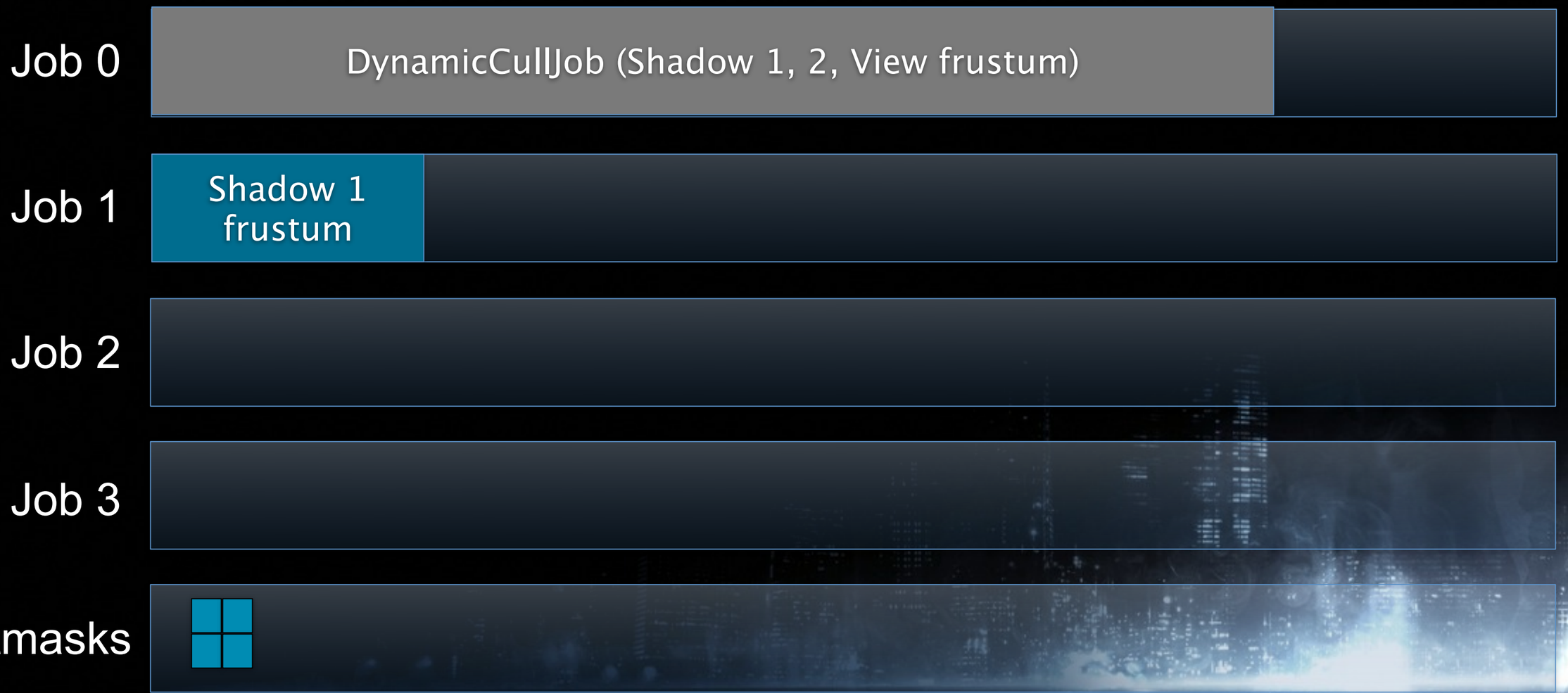
Job graph (Old Culling)



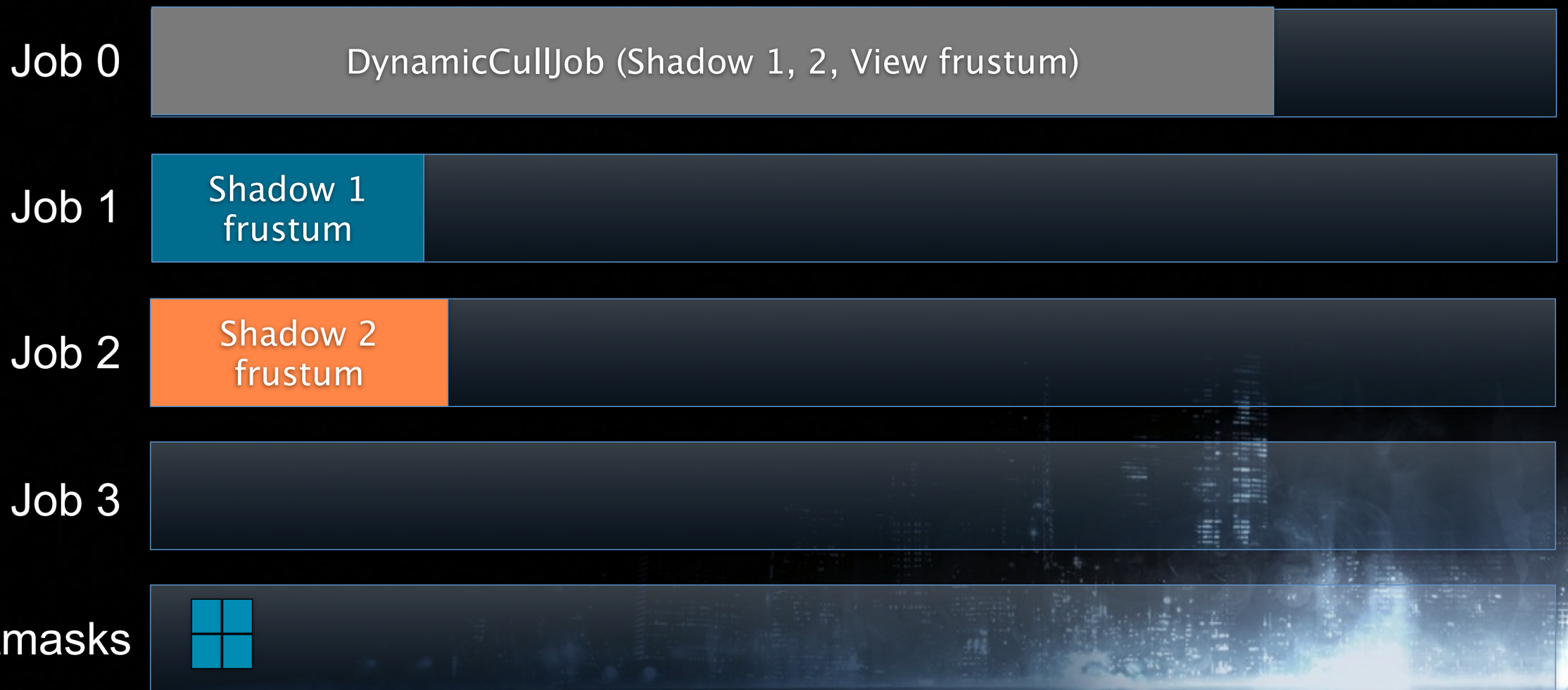
Job graph (Old Culling)



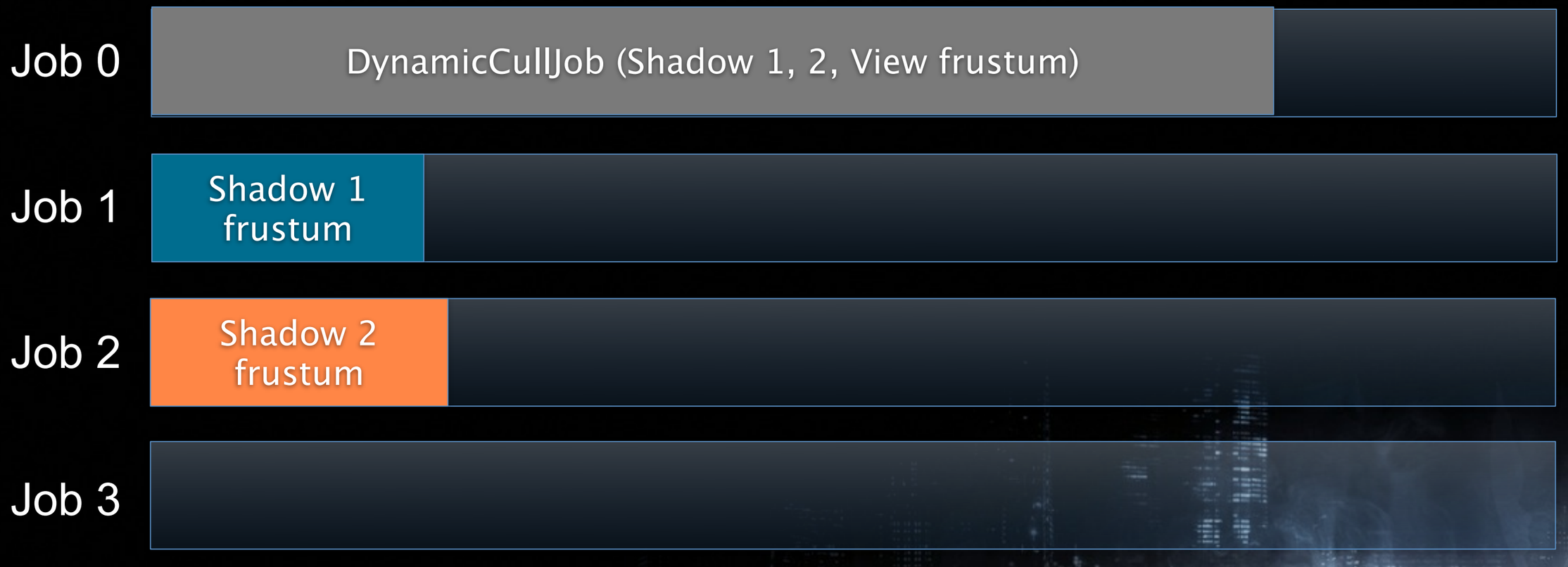
Job graph (Old Culling)



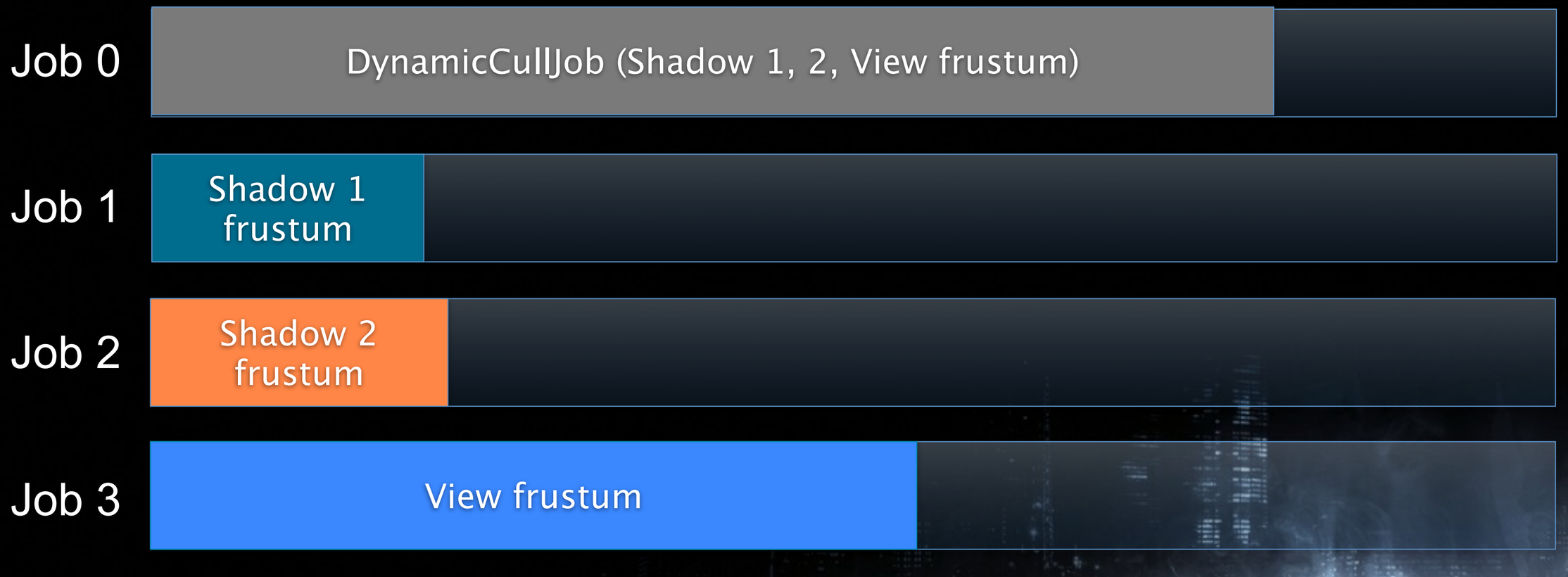
Job graph (Old Culling)



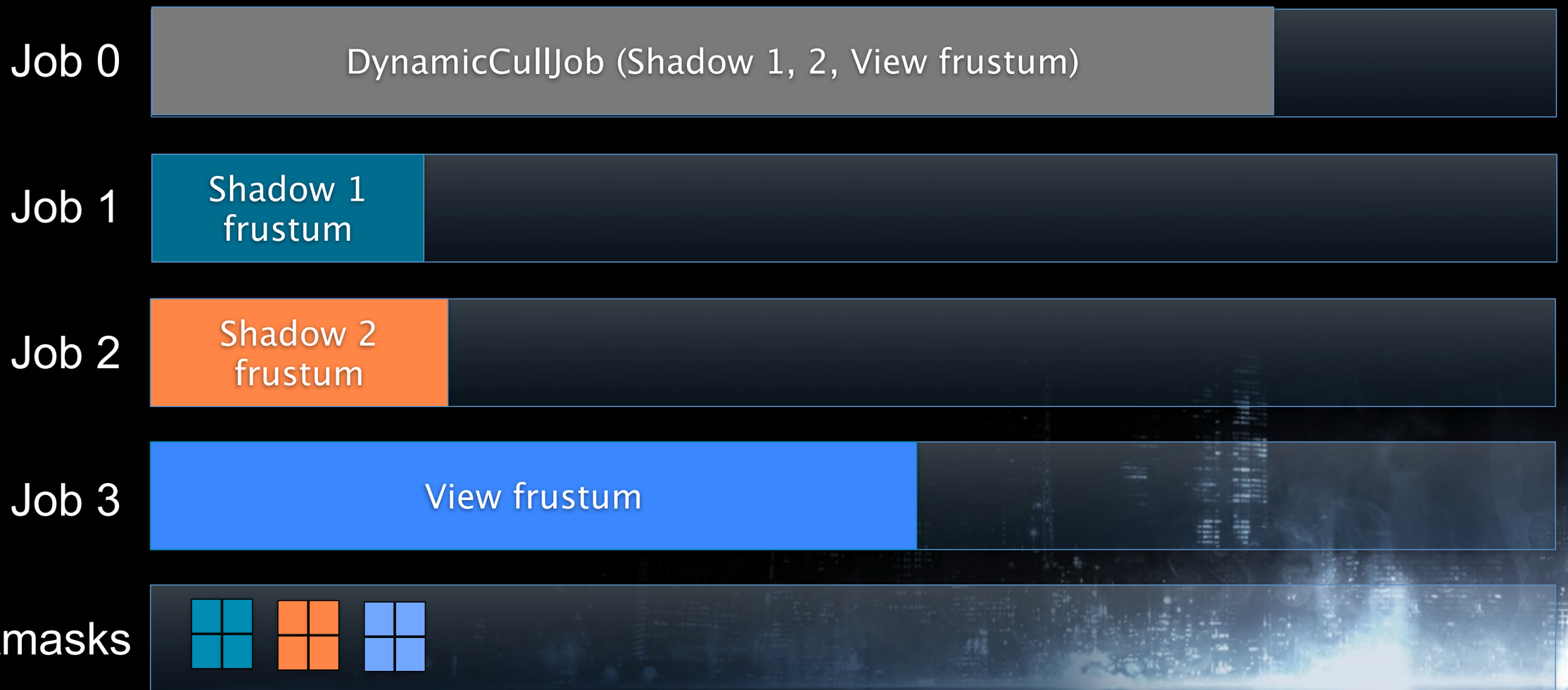
Job graph (Old Culling)



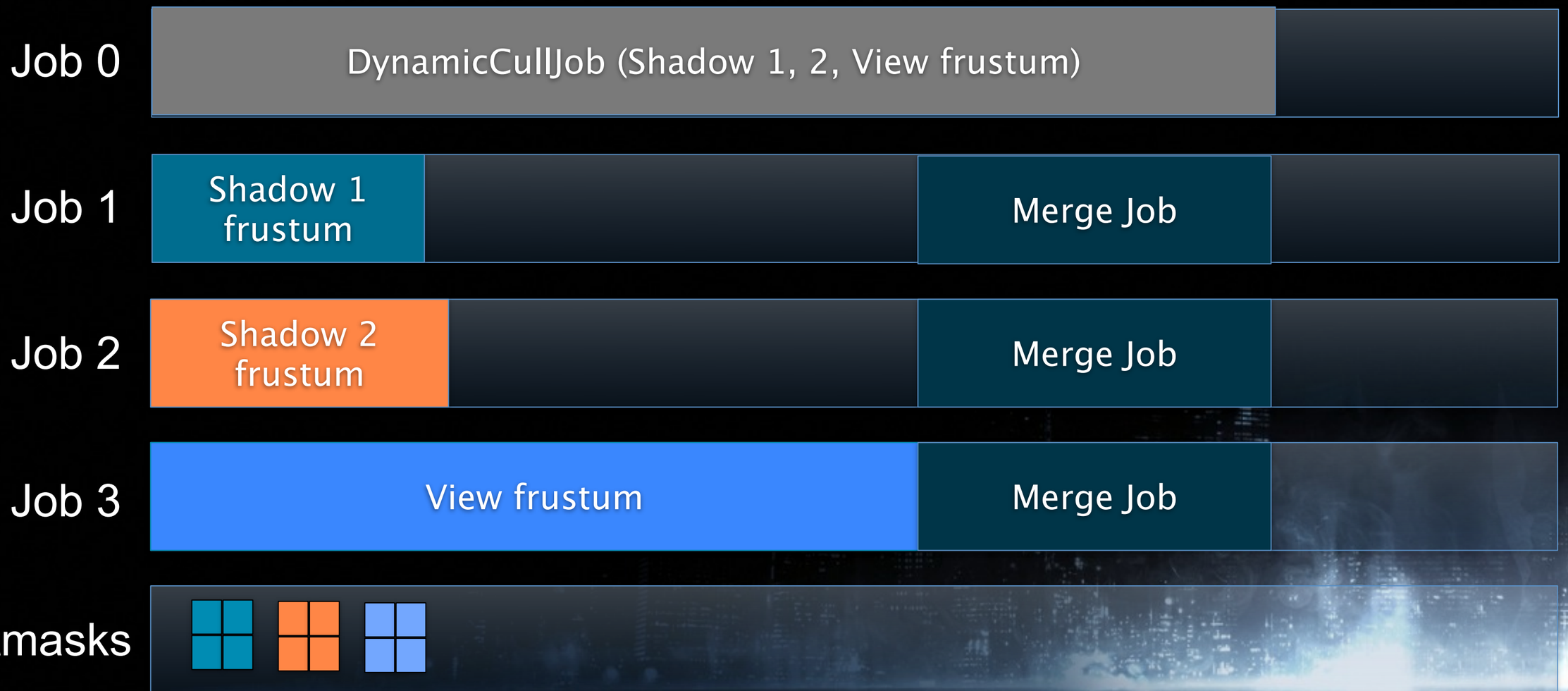
Job graph (Old Culling)



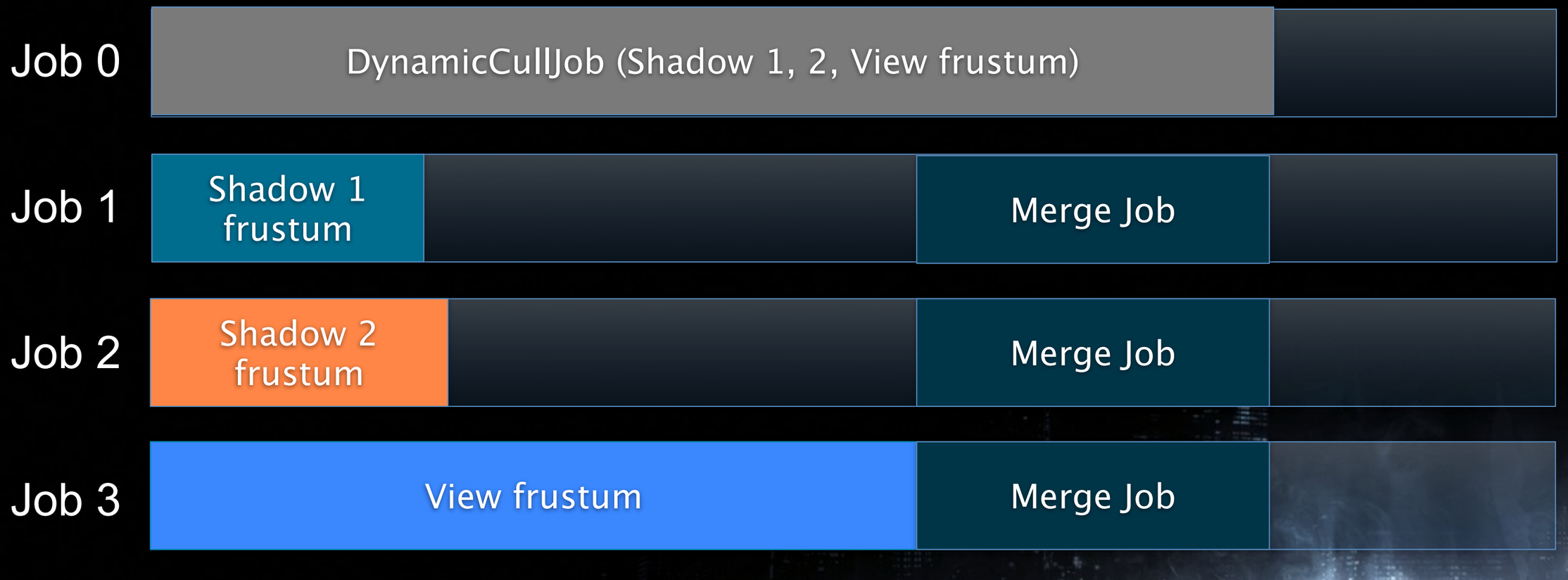
Job graph (Old Culling)



Job graph (Old Culling)



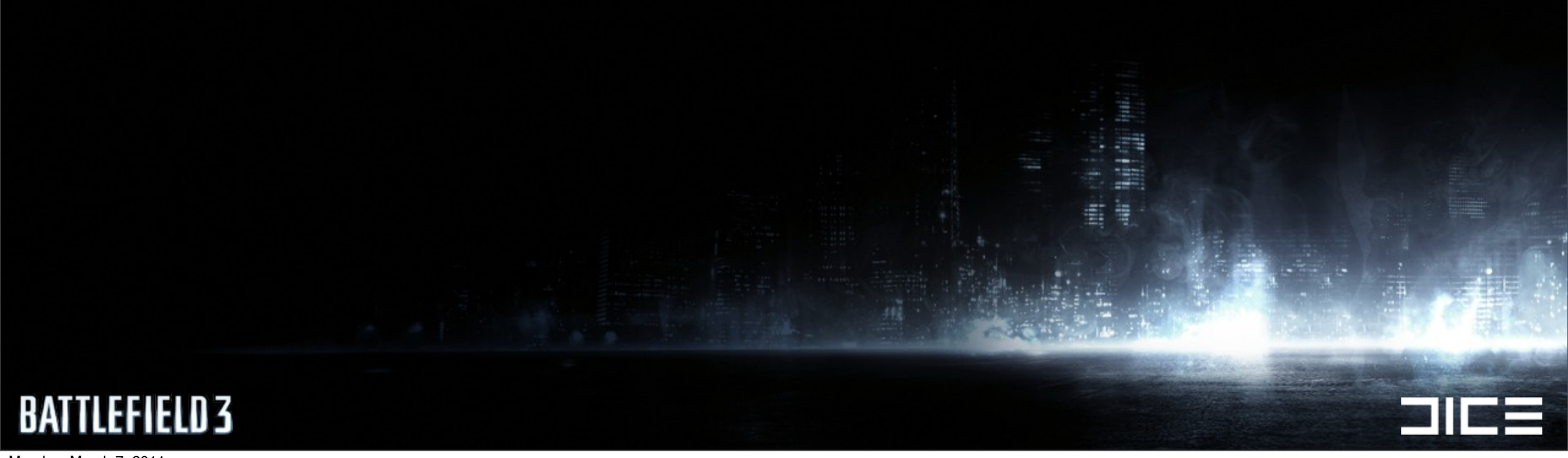
Job graph (Old Culling)



Requirements for new system

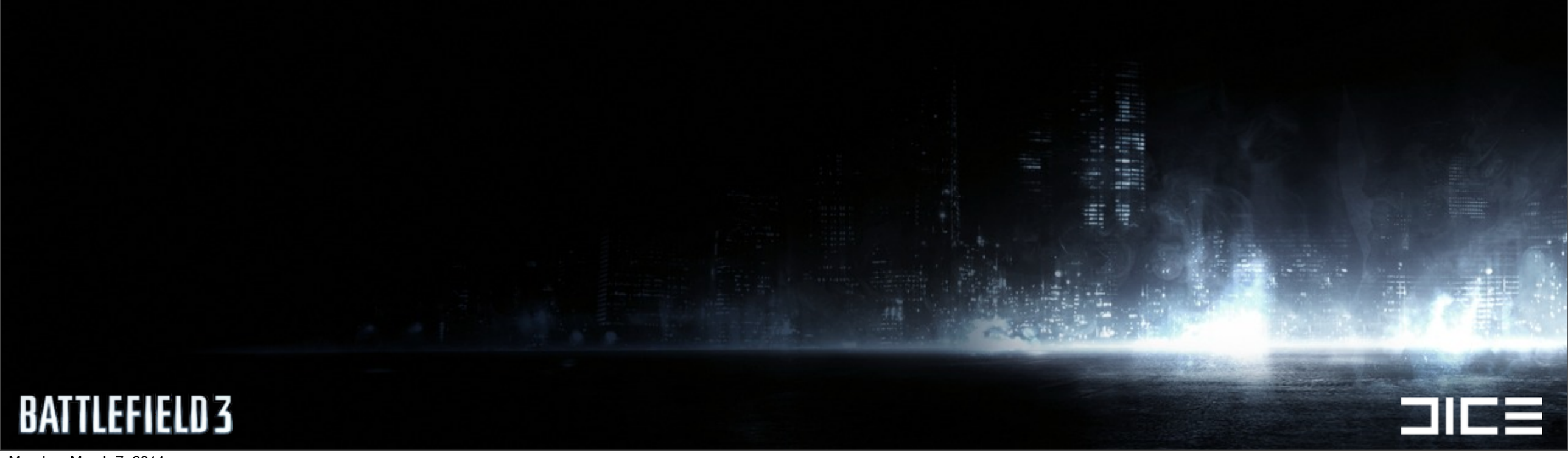
Requirements for new system

- Better scaling



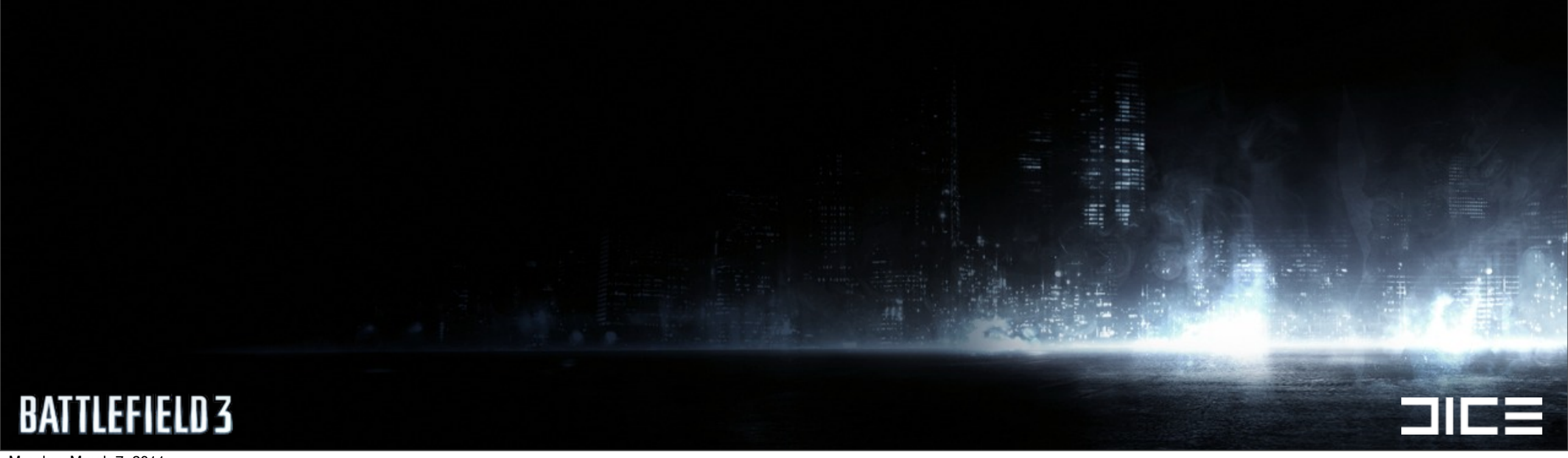
Requirements for new system

- Better scaling
- Destruction



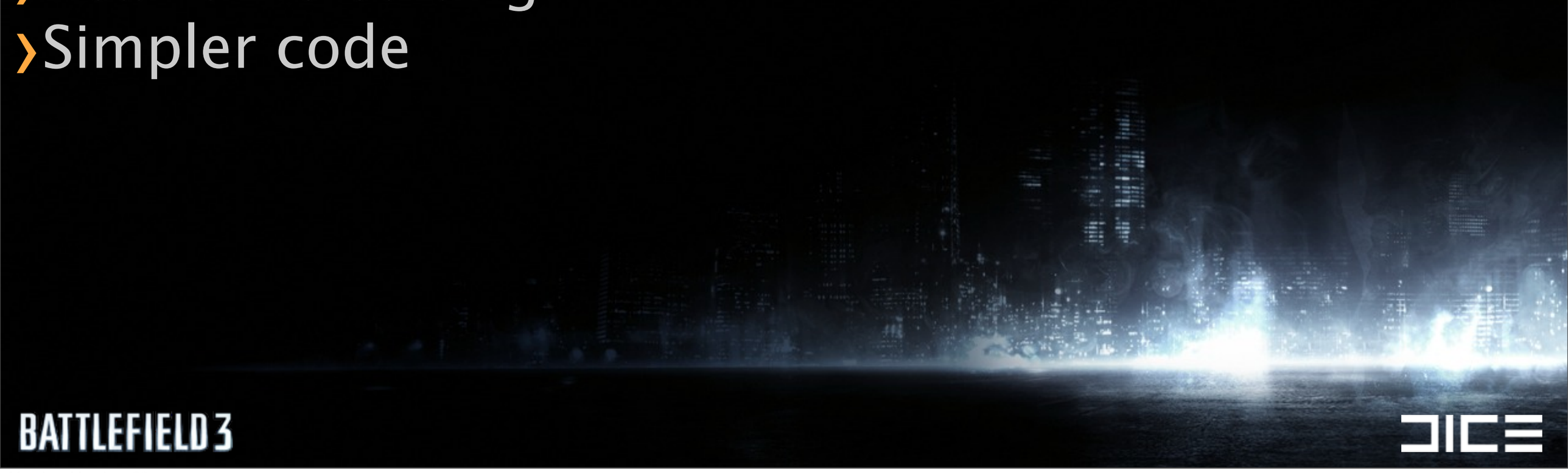
Requirements for new system

- Better scaling
- Destruction
- Real-time editing



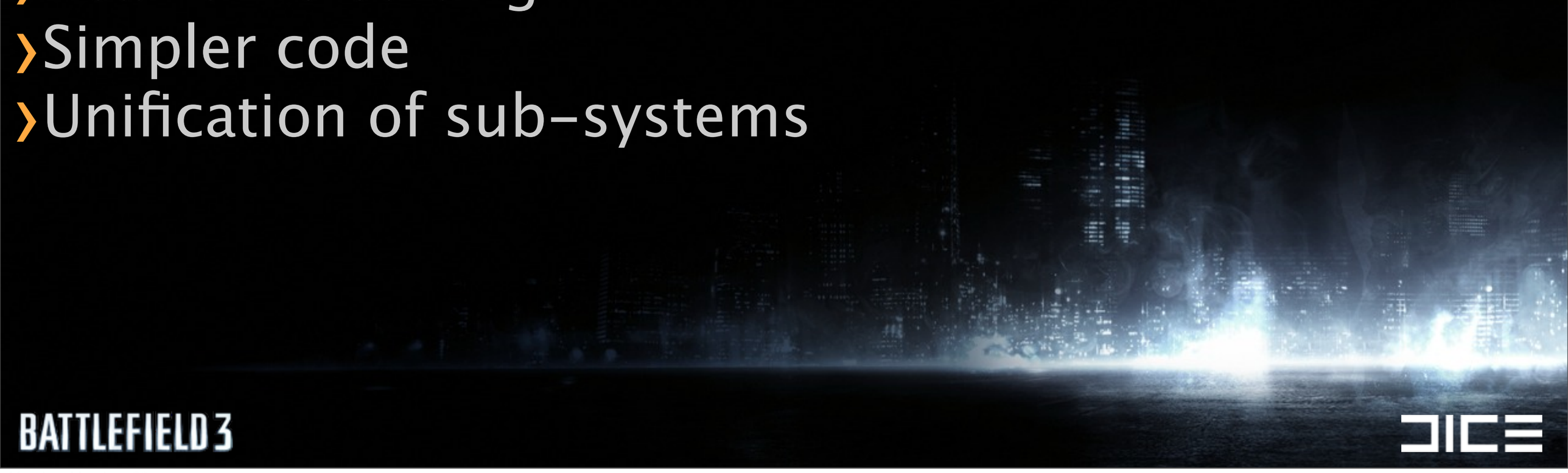
Requirements for new system

- Better scaling
- Destruction
- Real-time editing
- Simpler code



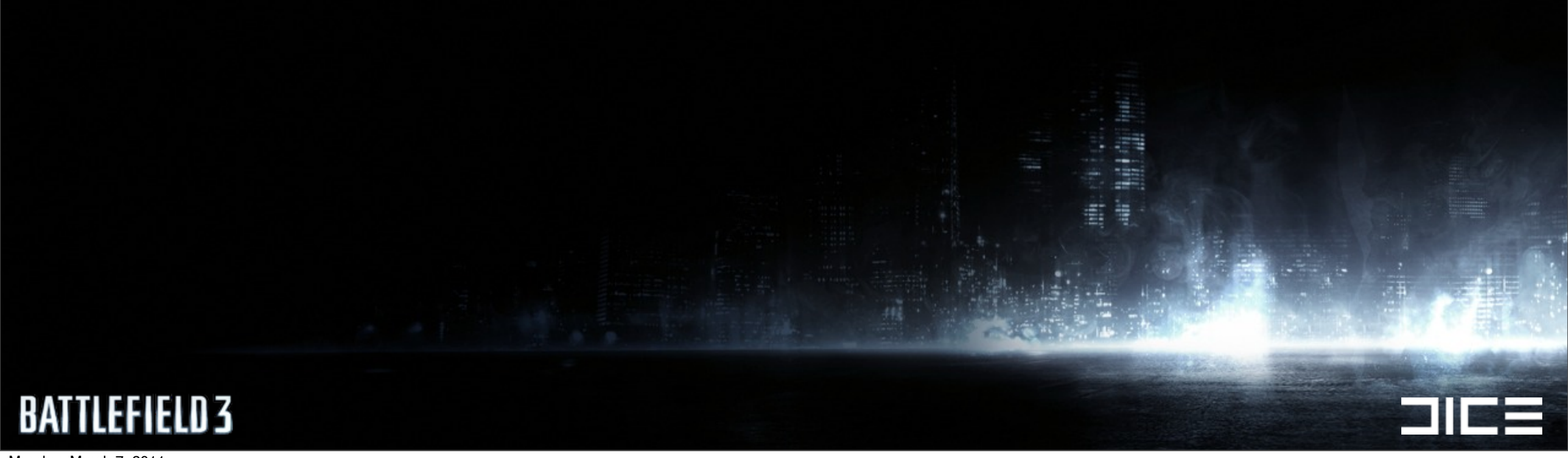
Requirements for new system

- Better scaling
- Destruction
- Real-time editing
- Simpler code
- Unification of sub-systems



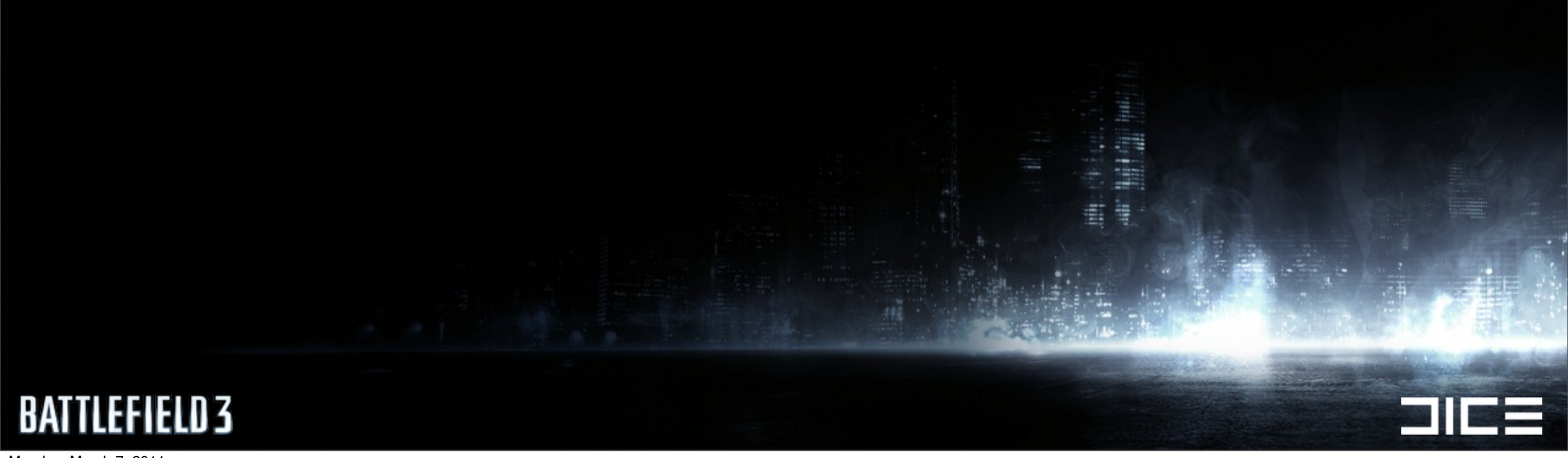
Target hardware

What **doesn't** work well on these systems?



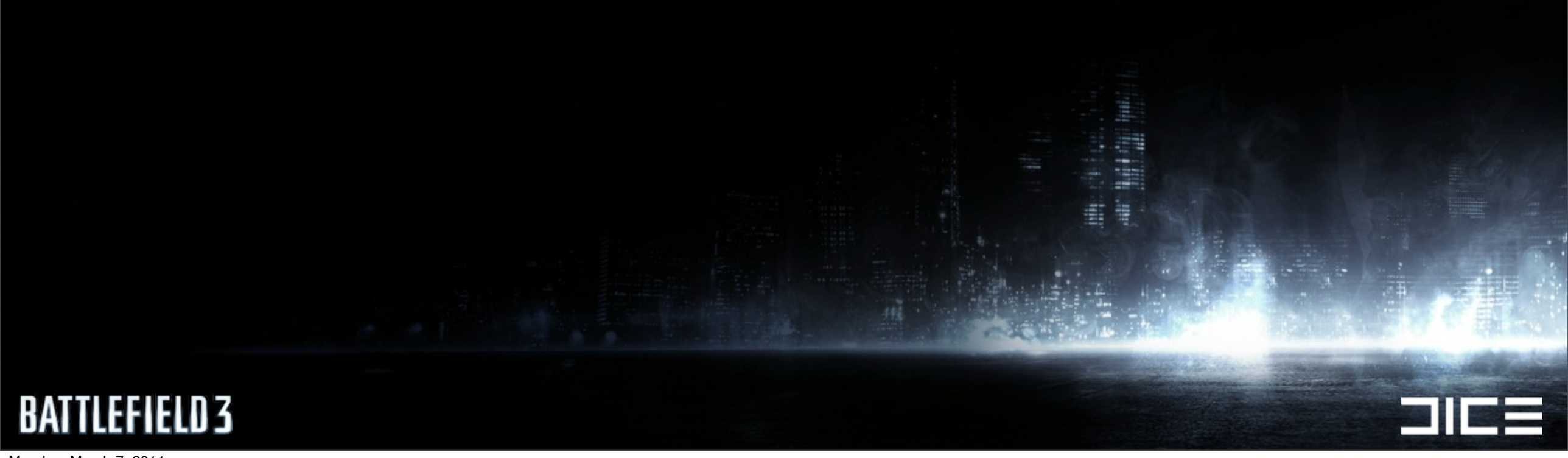
What **doesn't** work well on these systems?

- Non-local data



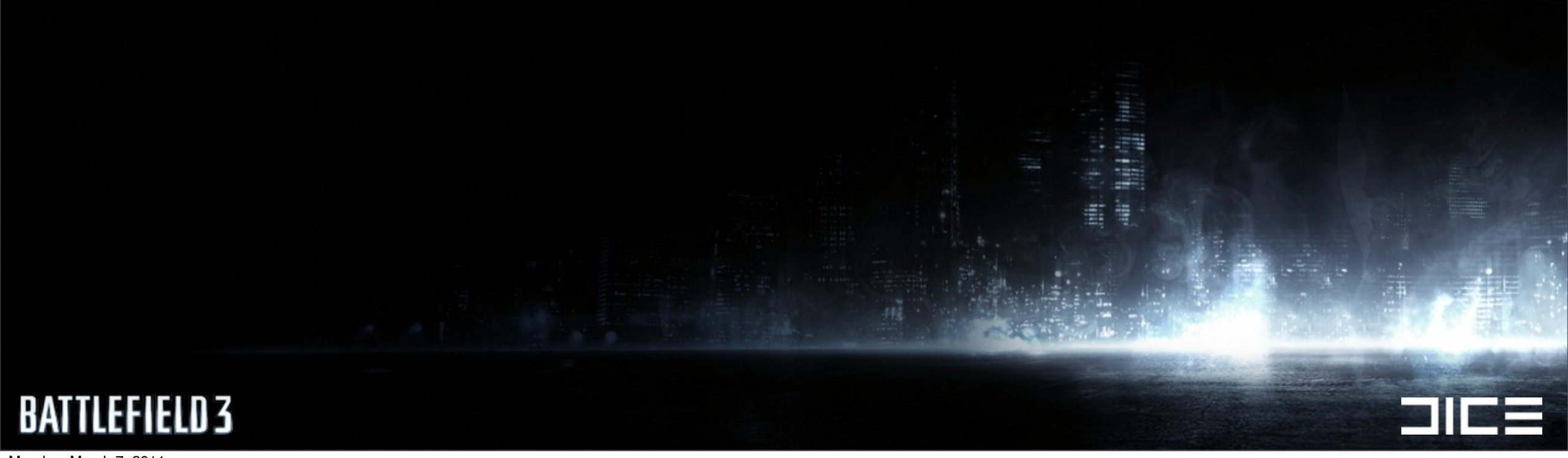
What **doesn't** work well on these systems?

- Non-local data
- Branches



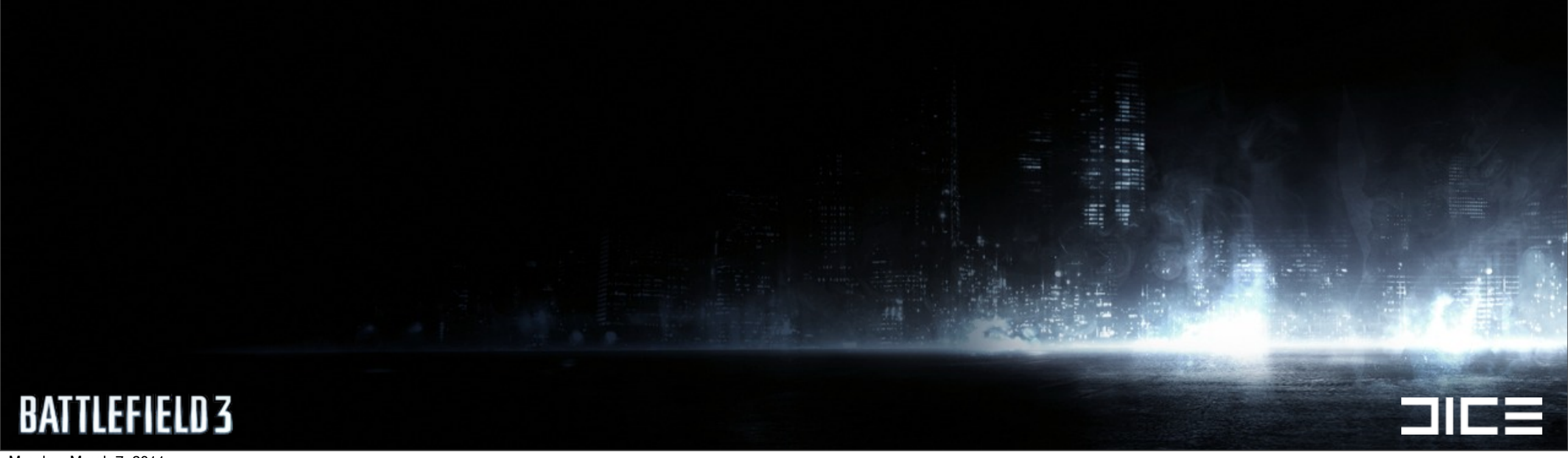
What **doesn't** work well on these systems?

- Non-local data
- Branches
- Switching between register types (LHS)



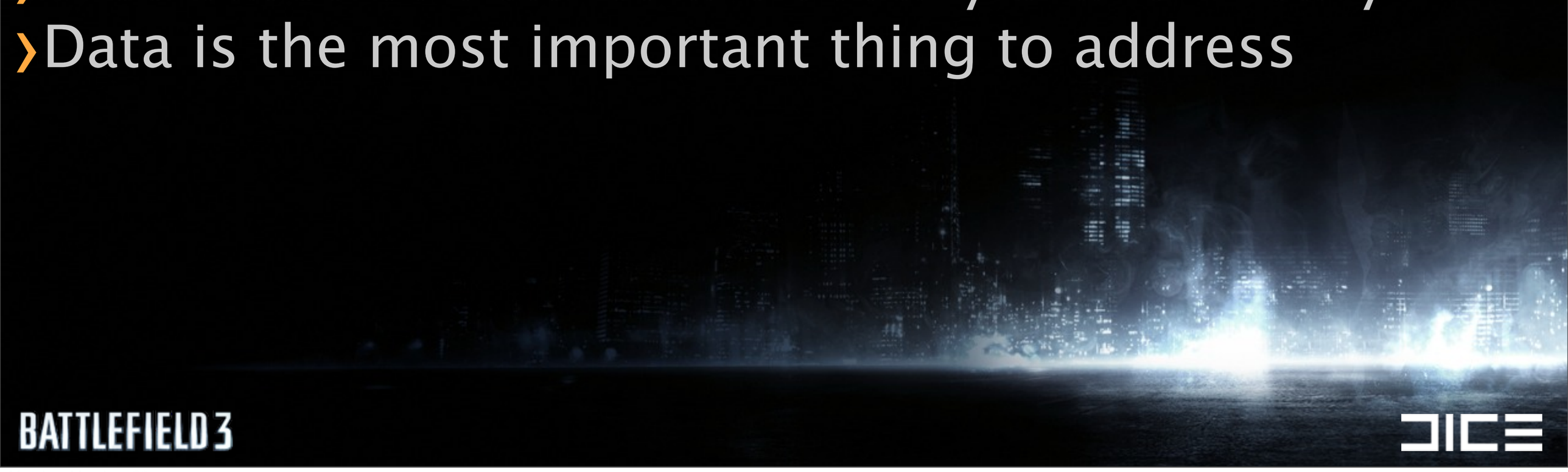
What **doesn't** work well on these systems?

- Non-local data
- Branches
- Switching between register types (LHS)
- Tree based structures are usually branch heavy

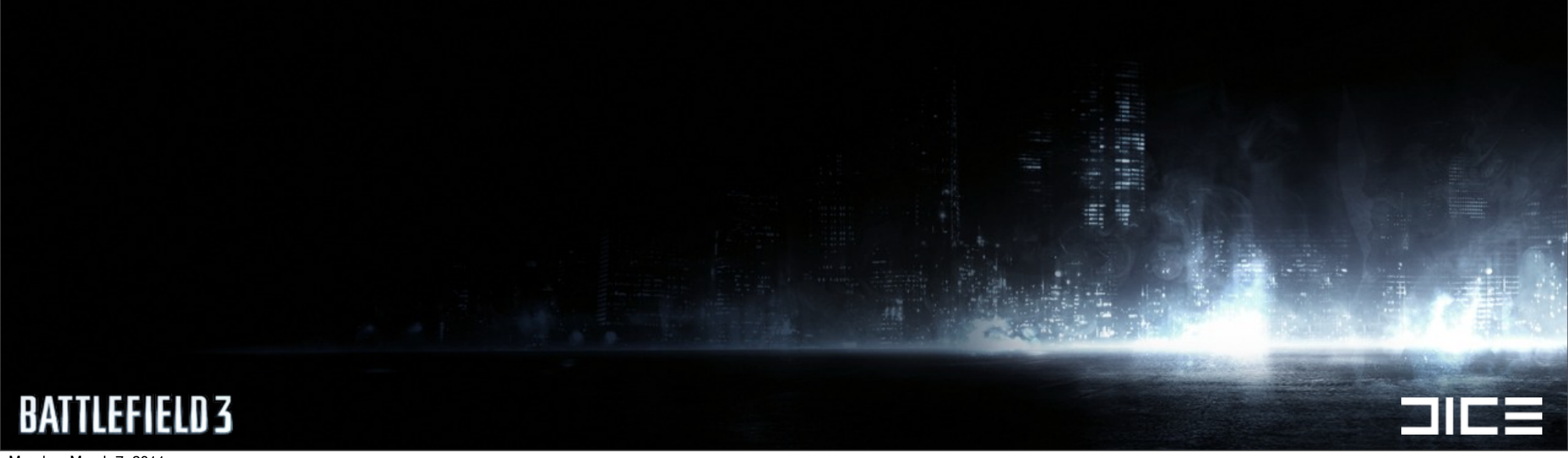


What **doesn't** work well on these systems?

- Non-local data
- Branches
- Switching between register types (LHS)
- Tree based structures are usually branch heavy
- Data is the most important thing to address

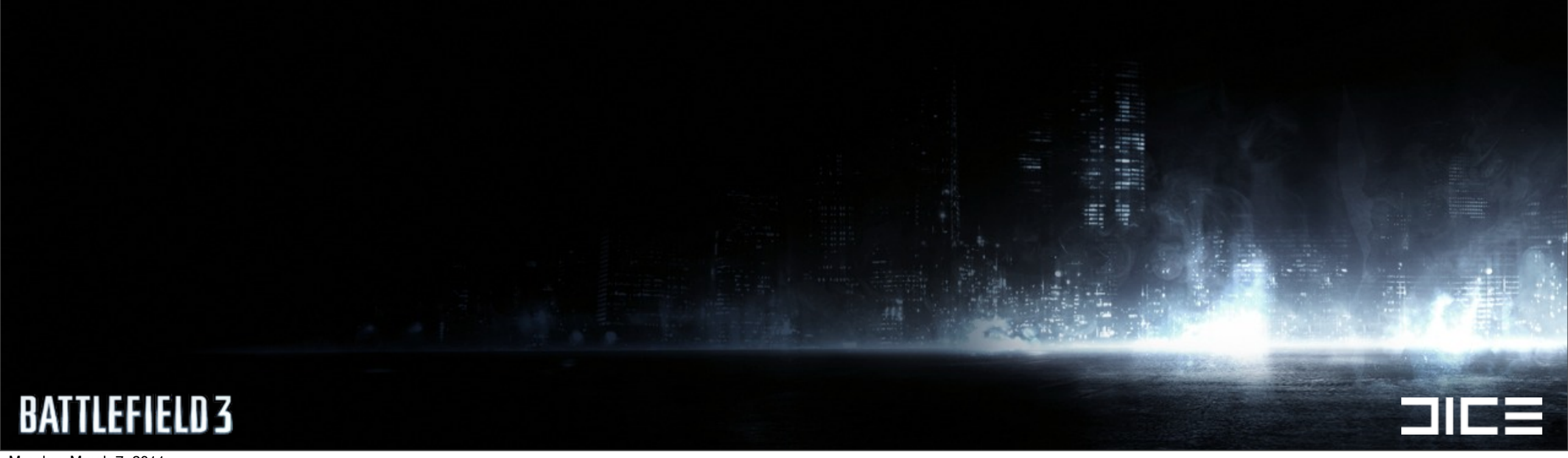


What does work **well** on these systems?



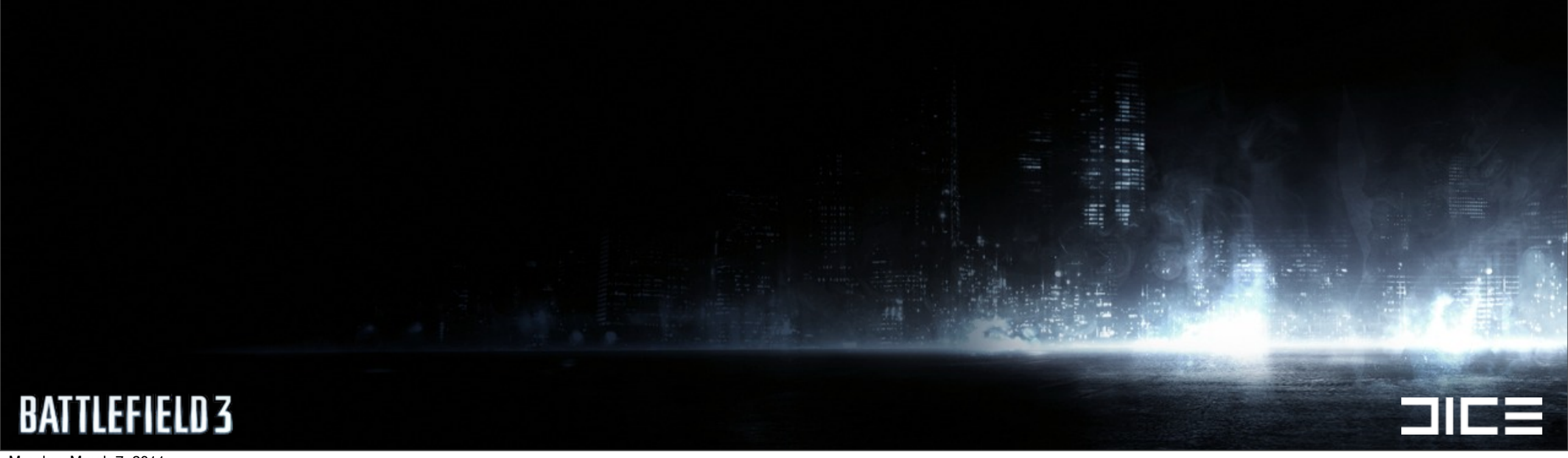
What does work **well** on these systems?

- Local data



What does work **well** on these systems?

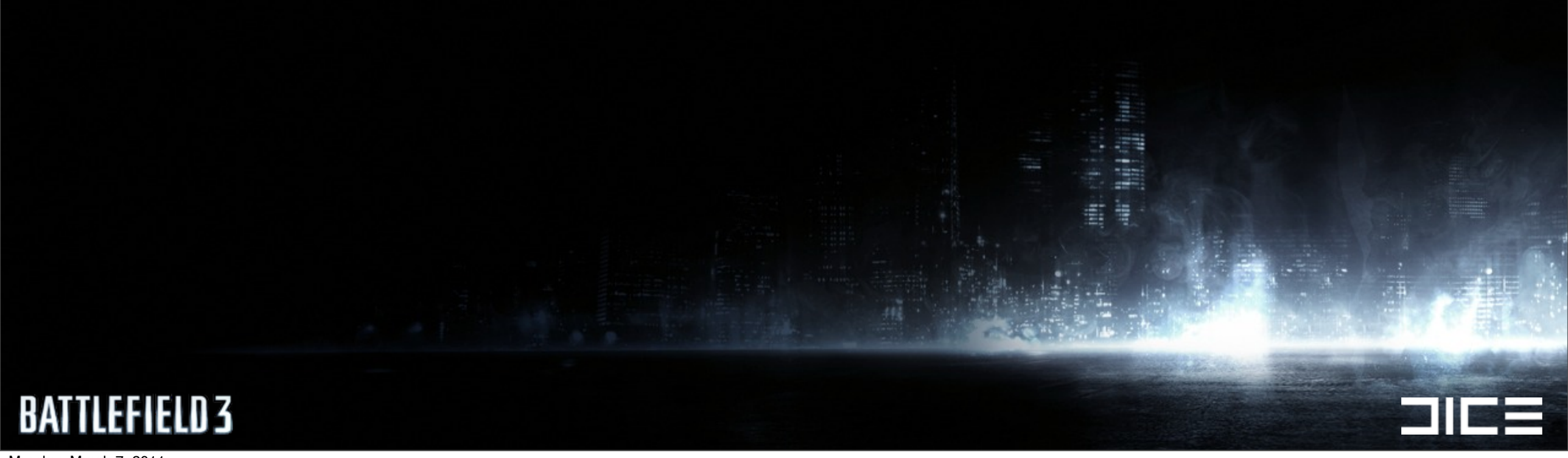
- Local data
- (SIMD) Computing power



What does work **well** on these systems?

- Local data
- (SIMD) Computing power
- Parallelism

The new culling

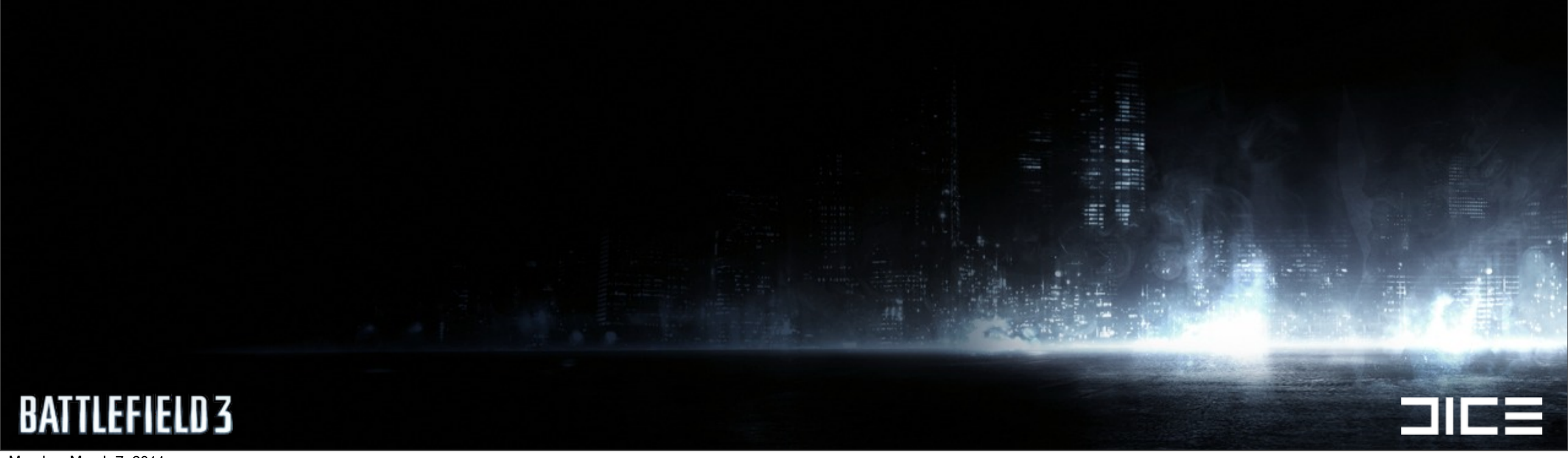


The new culling

- Our worlds usually has max ~15000 objects

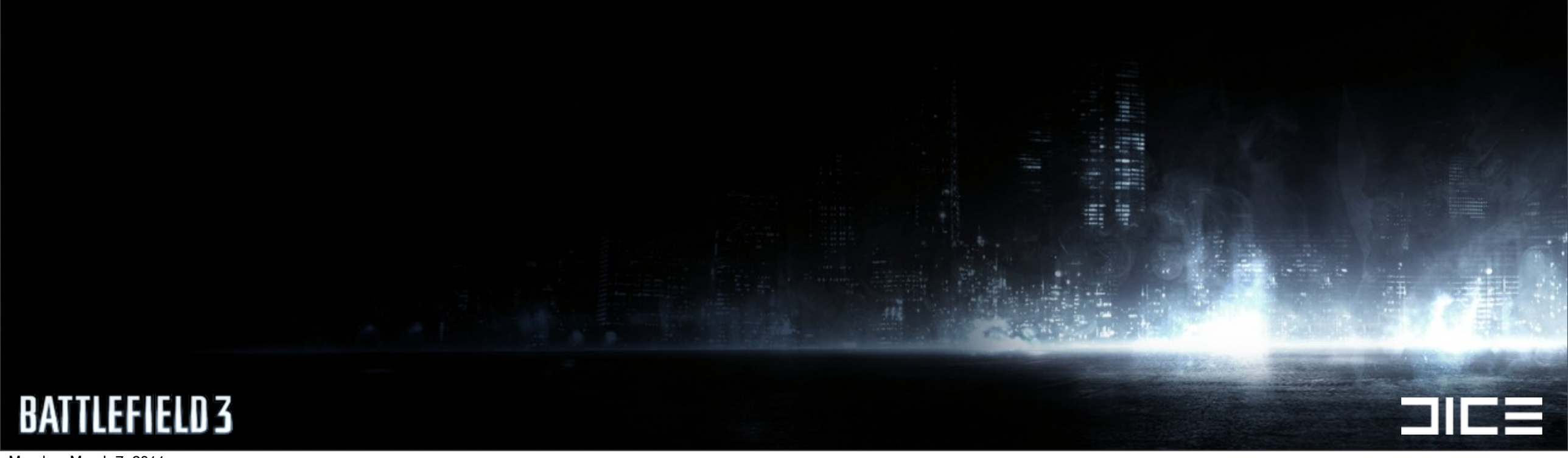
The new culling

- Our worlds usually has max ~15000 objects
- First try was to just use parallel brute force



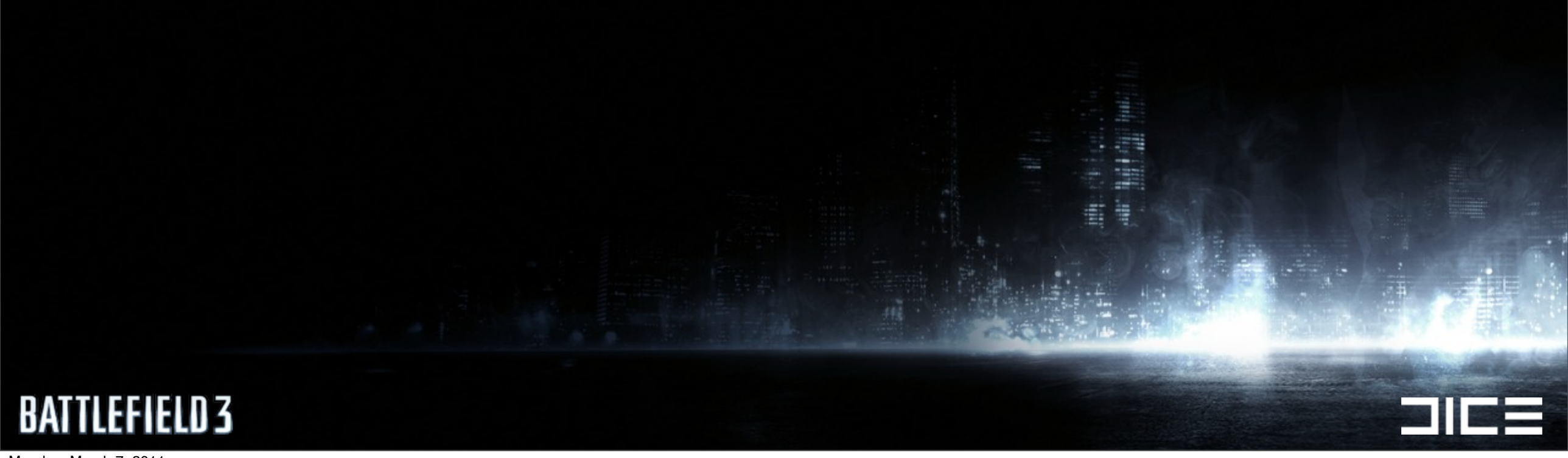
The new culling

- Our worlds usually has max ~15000 objects
- First try was to just use parallel brute force
- 3x times faster than the old culling



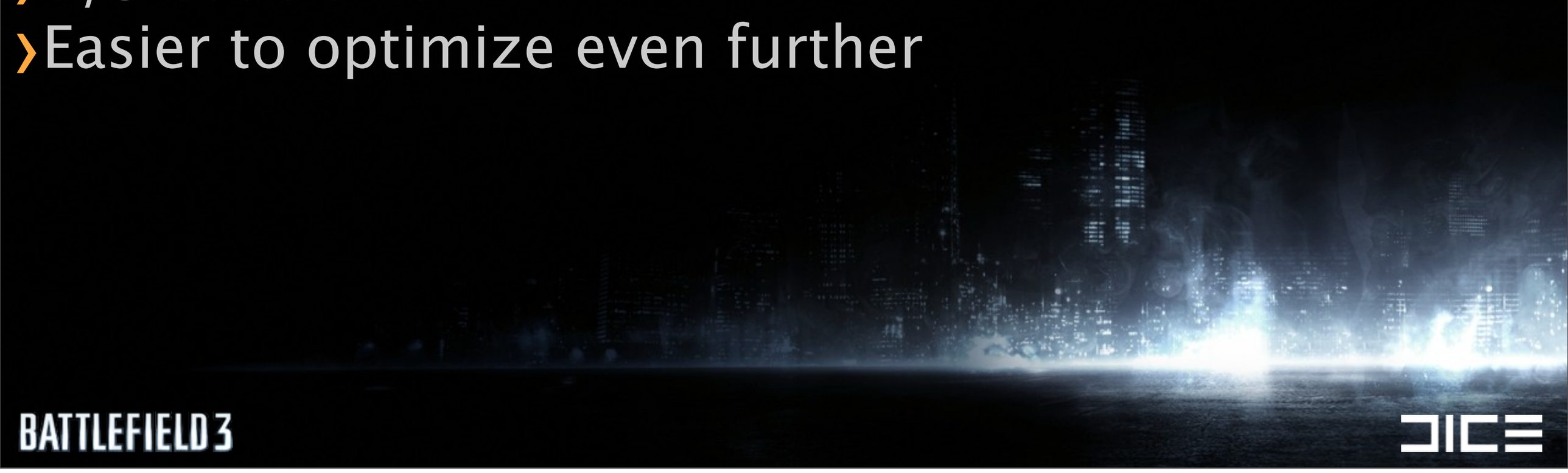
The new culling

- Our worlds usually has max ~15000 objects
- First try was to just use parallel brute force
- 3x times faster than the old culling
- 1/5 code size

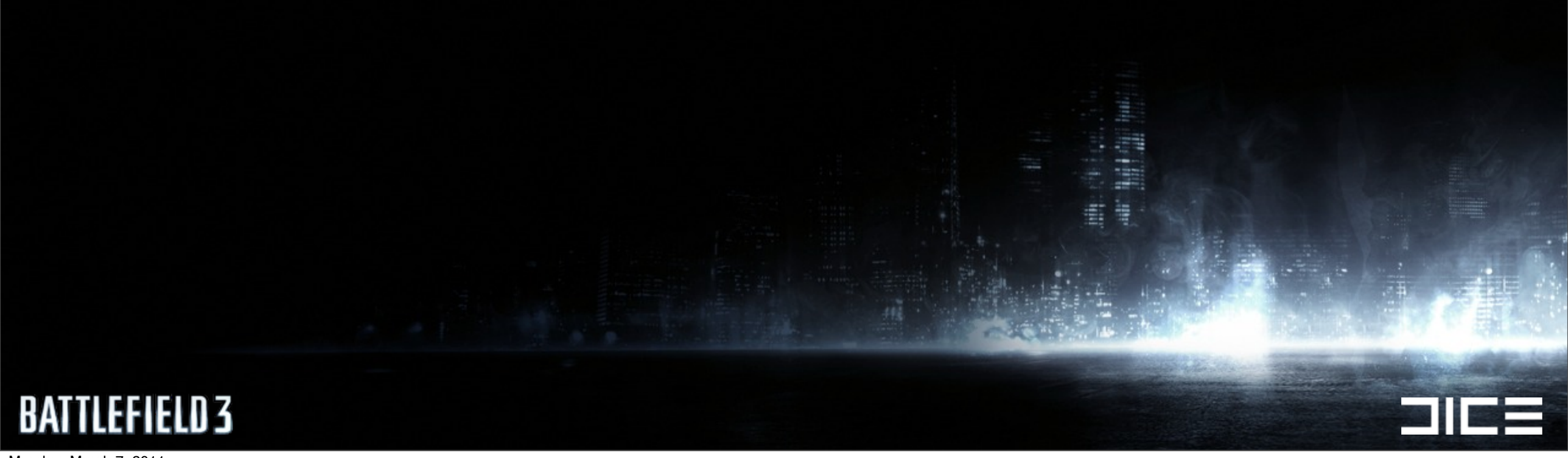


The new culling

- Our worlds usually has max ~15000 objects
- First try was to just use parallel brute force
- 3x times faster than the old culling
- 1/5 code size
- Easier to optimize even further

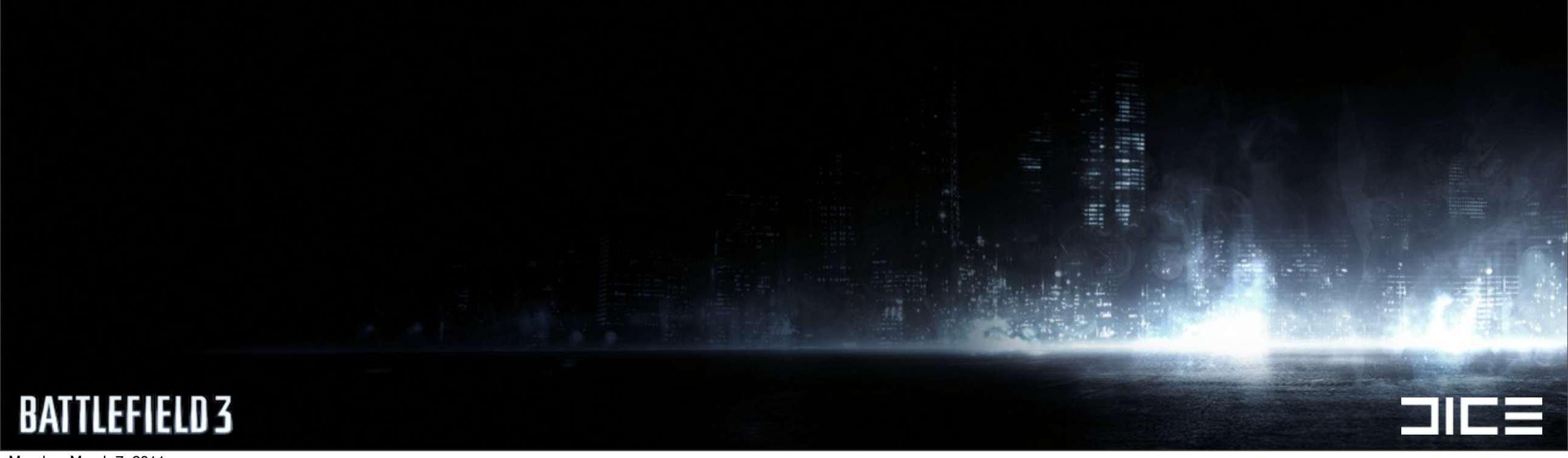


The new culling



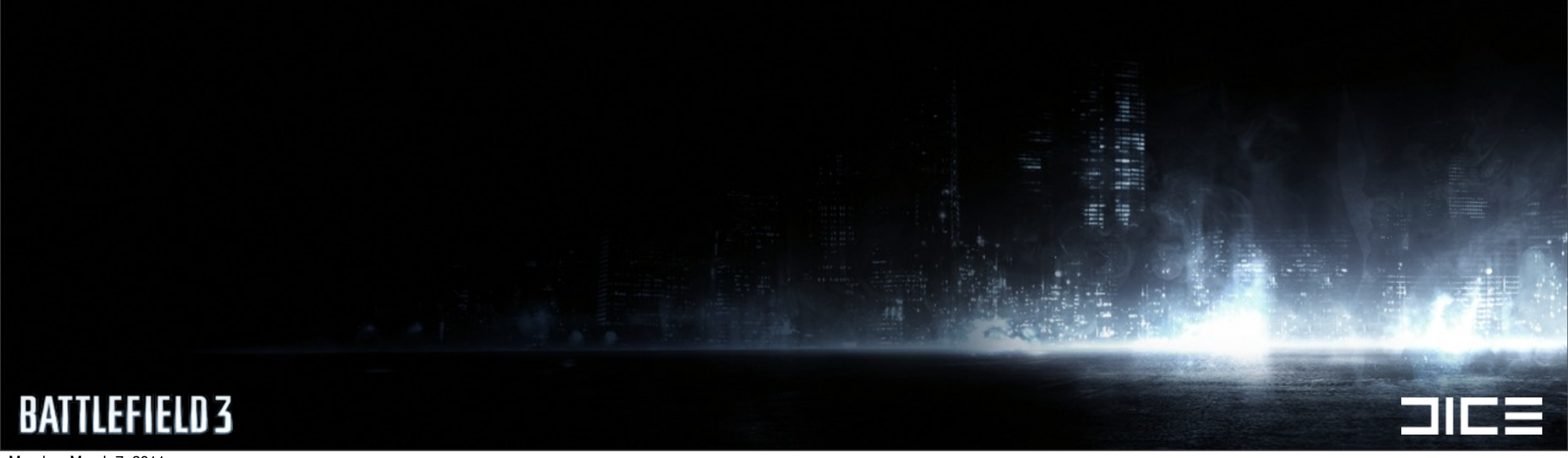
The new culling

- Linear arrays scale great



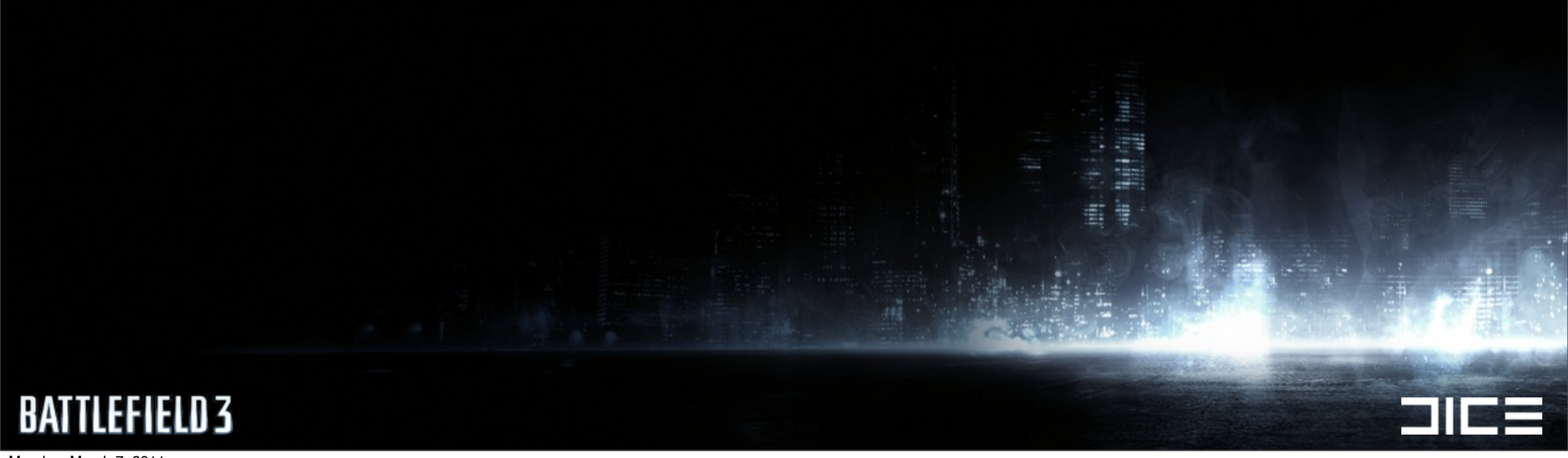
The new culling

- Linear arrays scale great
- Predictable data



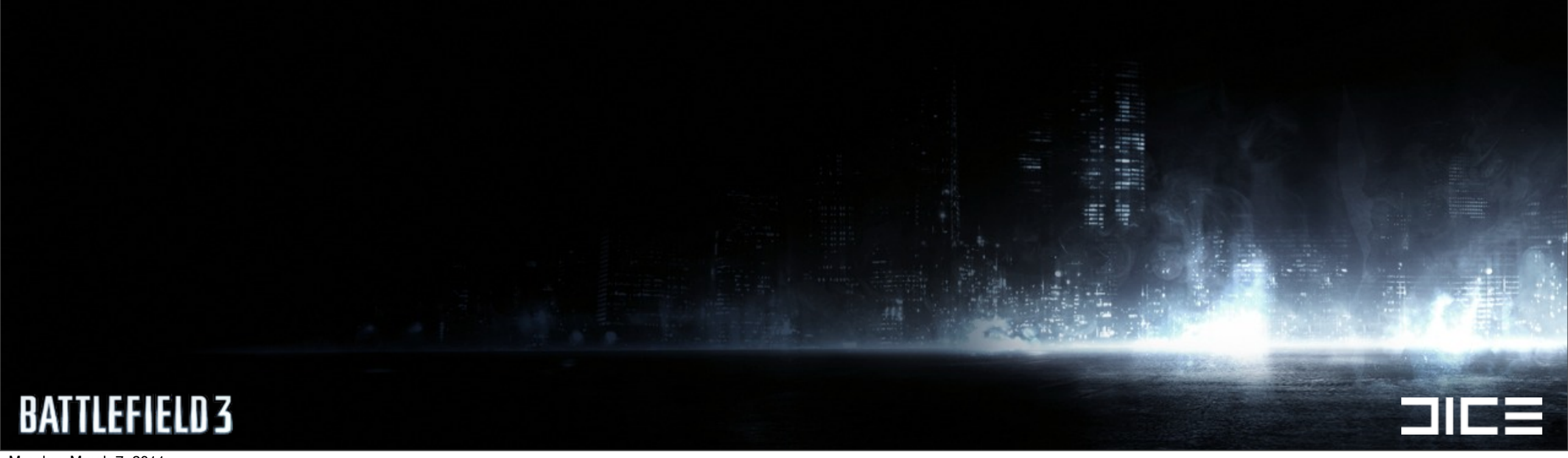
The new culling

- Linear arrays scale great
- Predictable data
- Few branches



The new culling

- Linear arrays scale great
- Predictable data
- Few branches
- Uses the computing power

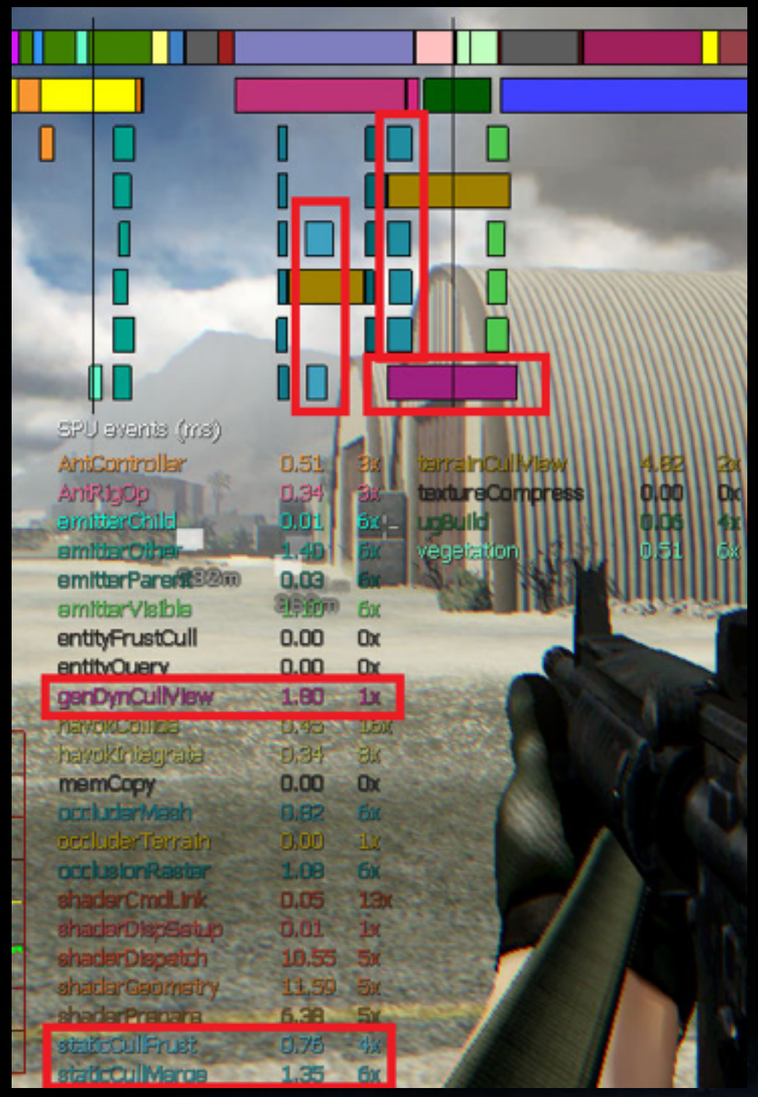


The new culling

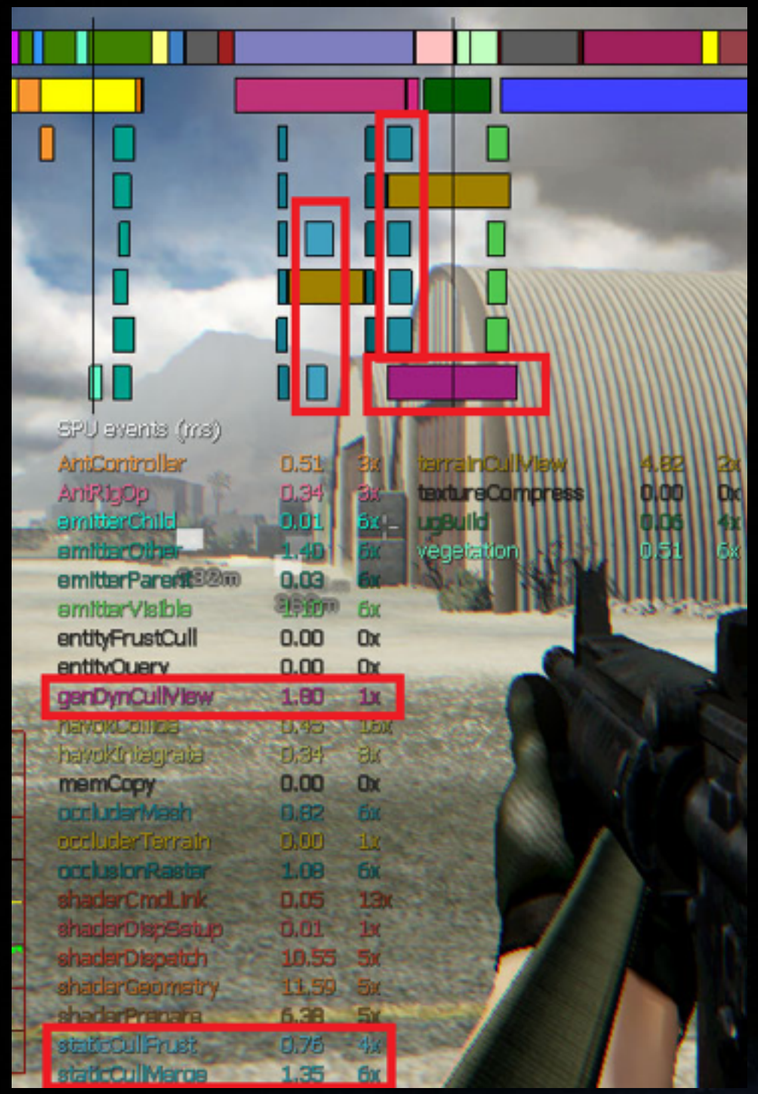
BATTLEFIELD 3

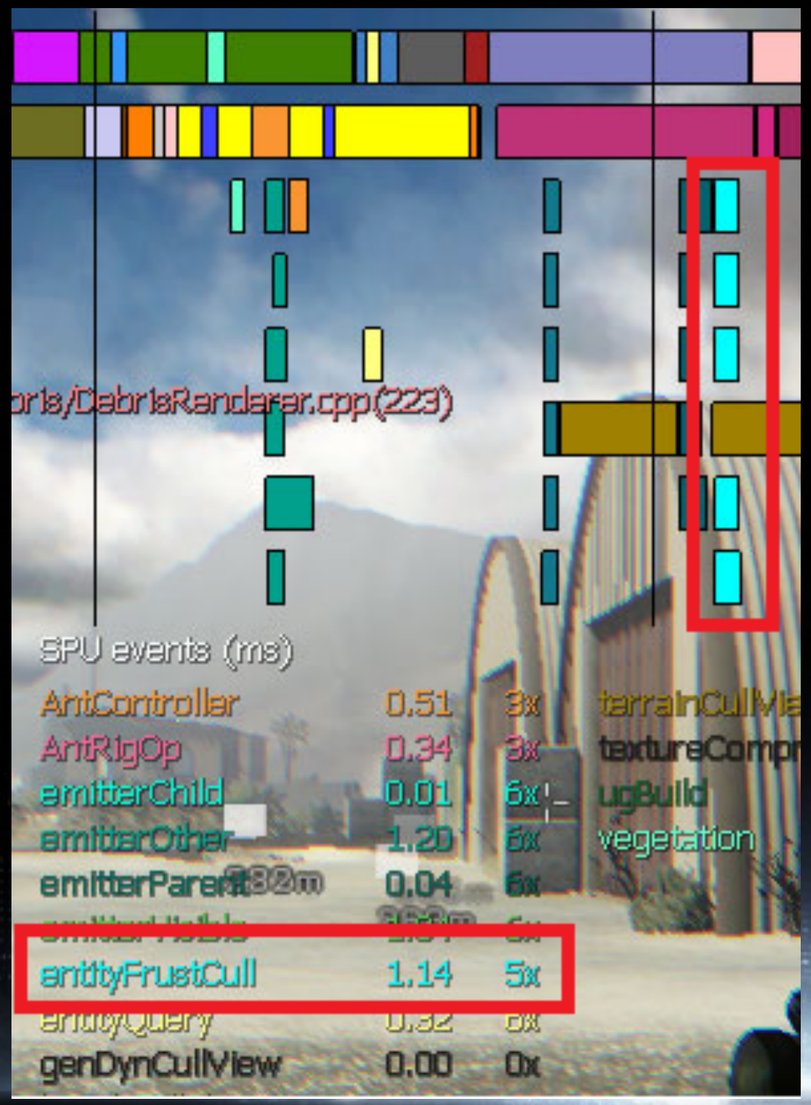
DICE

The new culling



The new culling

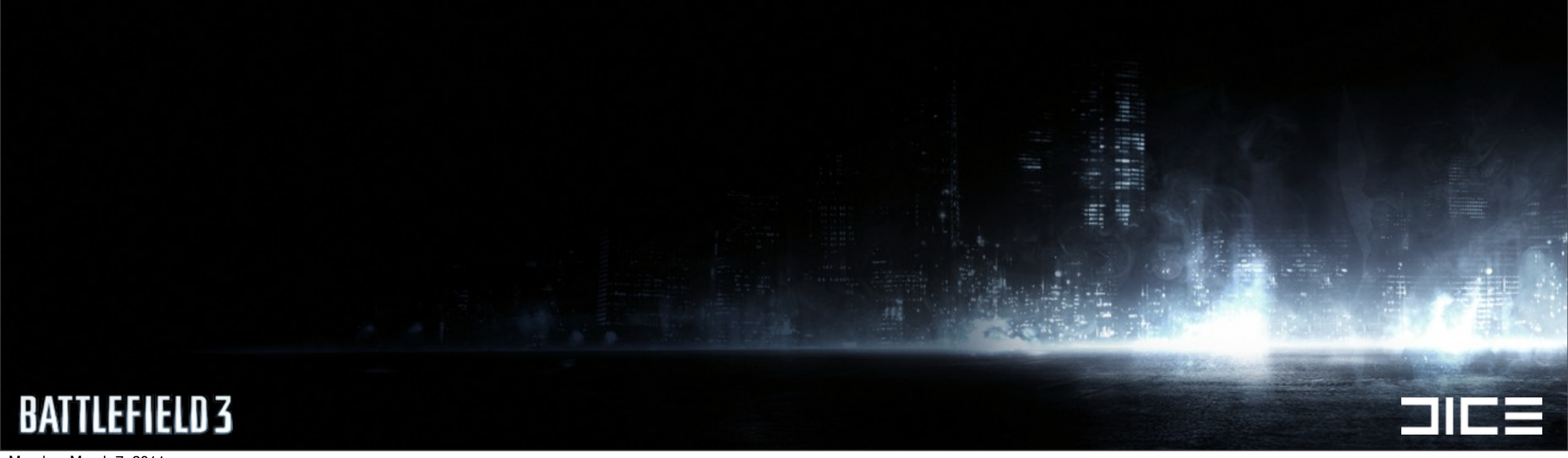




BATTLEFIELD 3

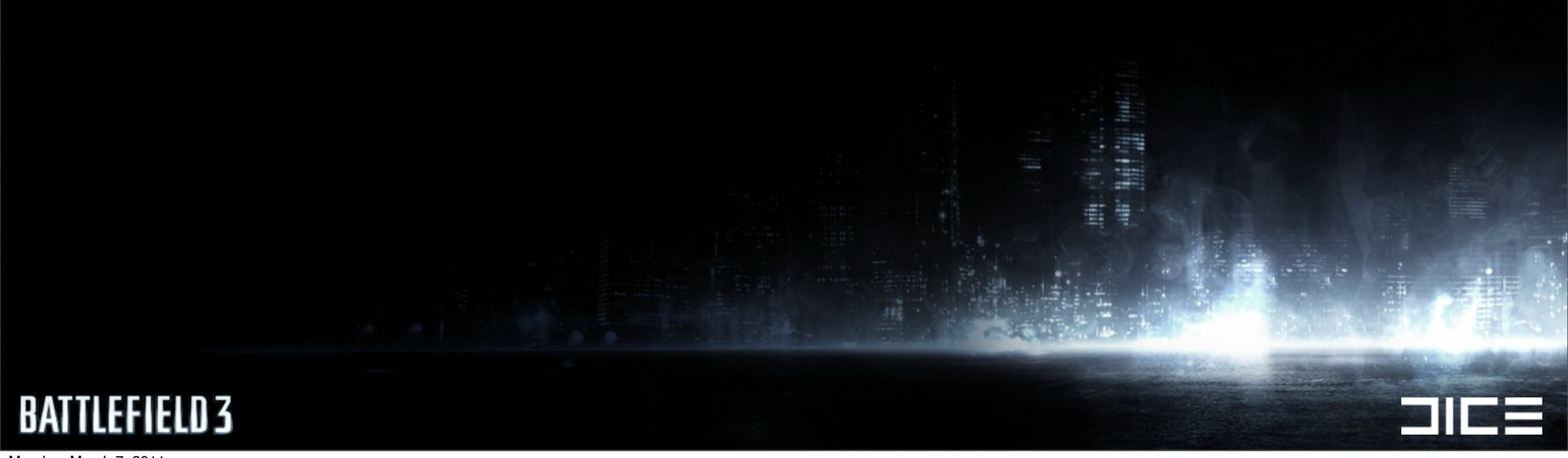
DICE

Performance numbers (no occlusion)
15000 Spheres



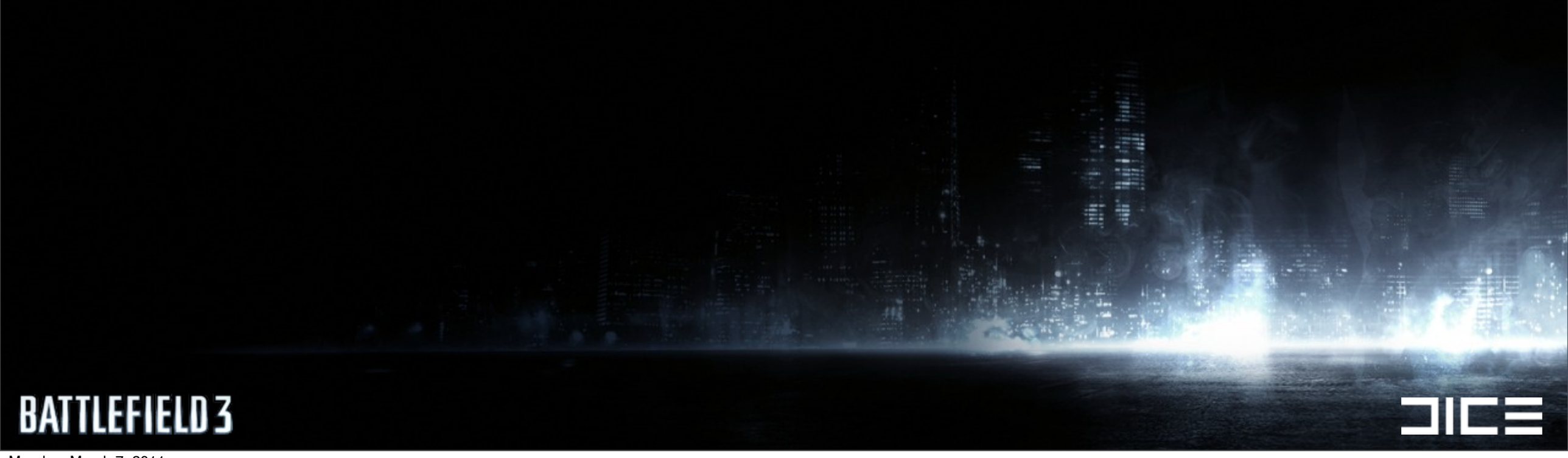
Performance numbers (no occlusion)
15000 Spheres

Platform	1 Job	4 Jobs



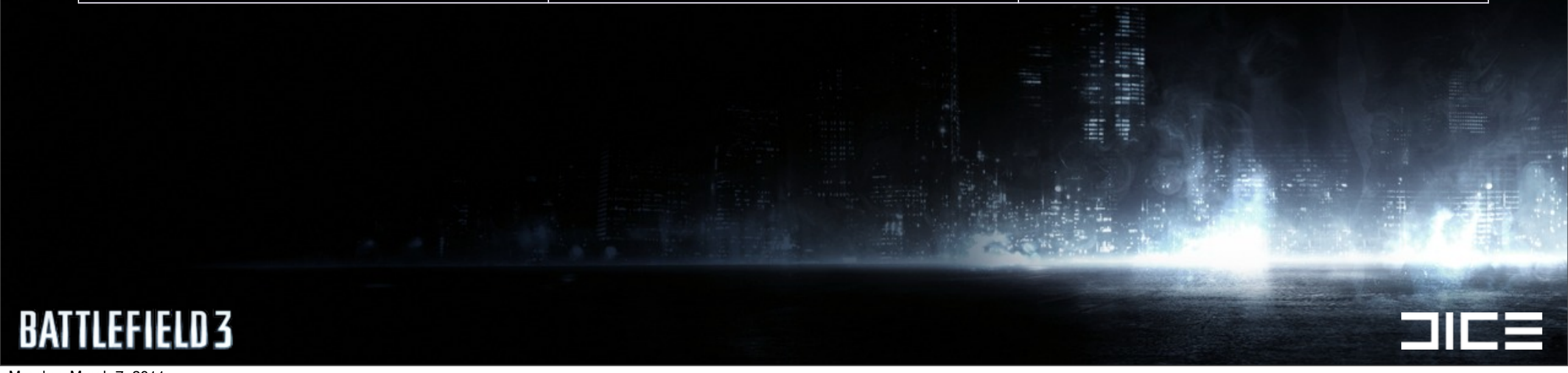
Performance numbers (no occlusion)
15000 Spheres

Platform	1 Job	4 Jobs
Xbox 360	1.55 ms	(2.10 ms / 4) = 0.52



Performance numbers (no occlusion)
15000 Spheres

Platform	1 Job	4 Jobs
Xbox 360	1.55 ms	(2.10 ms / 4) = 0.52
x86 (Core i7, 2.66 GHz)	1.0 ms	(1.30 ms / 4) = 0.32



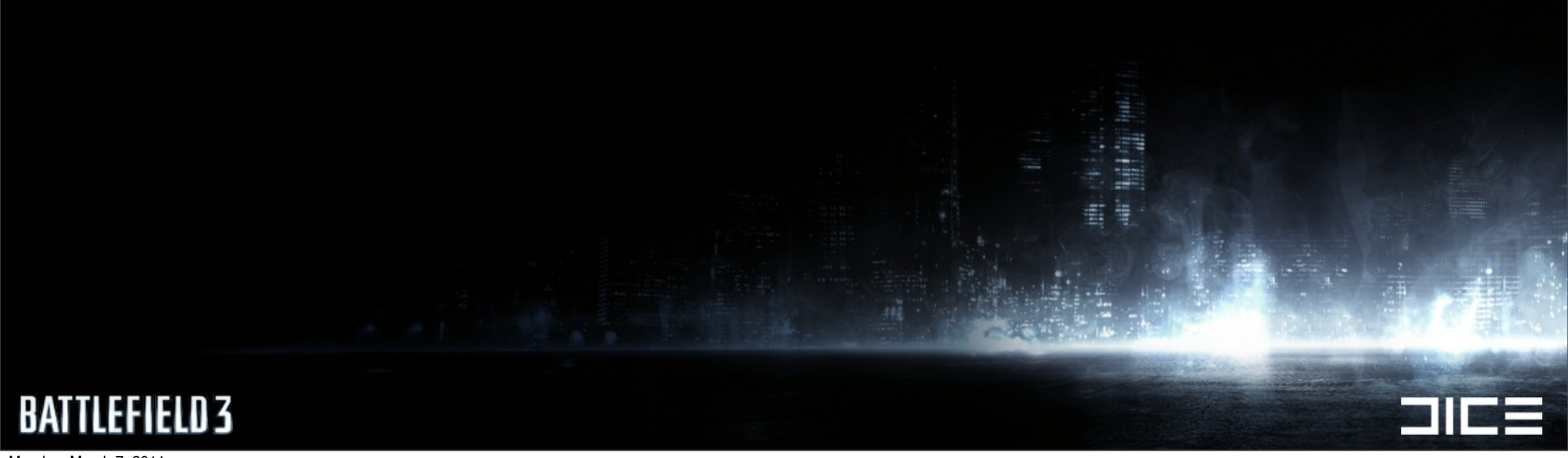
Performance numbers (no occlusion)
15000 Spheres

Platform	1 Job	4 Jobs
Xbox 360	1.55 ms	(2.10 ms / 4) = 0.52
x86 (Core i7, 2.66 GHz)	1.0 ms	(1.30 ms / 4) = 0.32
Playstation 3	0.85 ms	(0.95 ms / 4) = 0.23

Performance numbers (no occlusion)
15000 Spheres

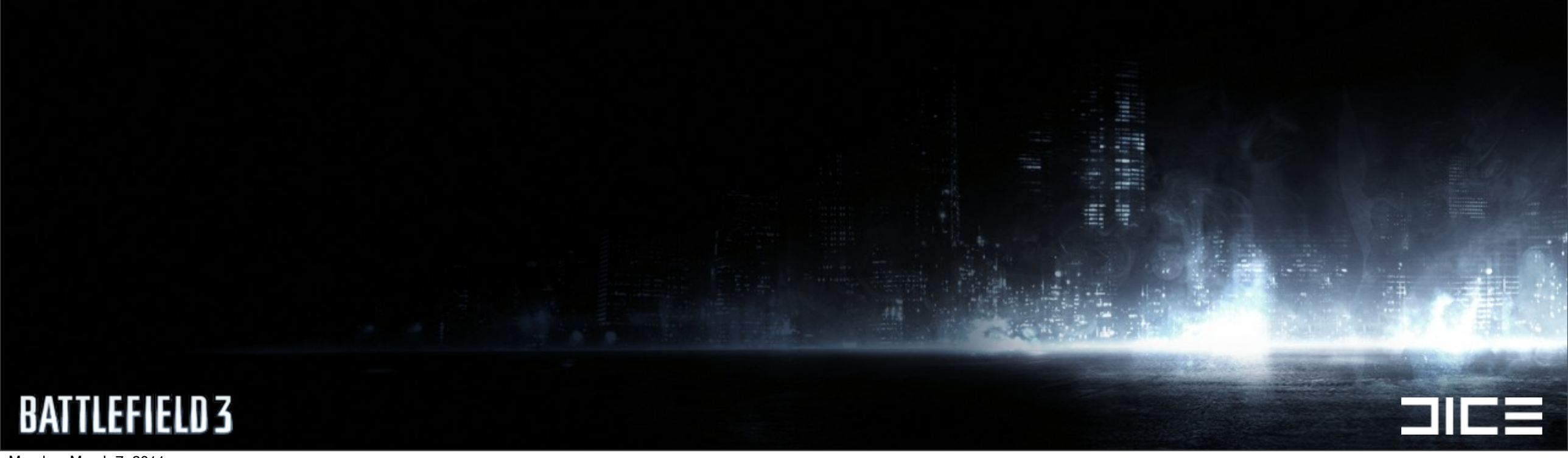
Platform	1 Job	4 Jobs
Xbox 360	1.55 ms	(2.10 ms / 4) = 0.52
x86 (Core i7, 2.66 GHz)	1.0 ms	(1.30 ms / 4) = 0.32
Playstation 3	0.85 ms	(0.95 ms / 4) = 0.23
Playstation 3 (SPA)	0.63 ms	(0.75 ms / 4) = 0.18

Details of the new culling



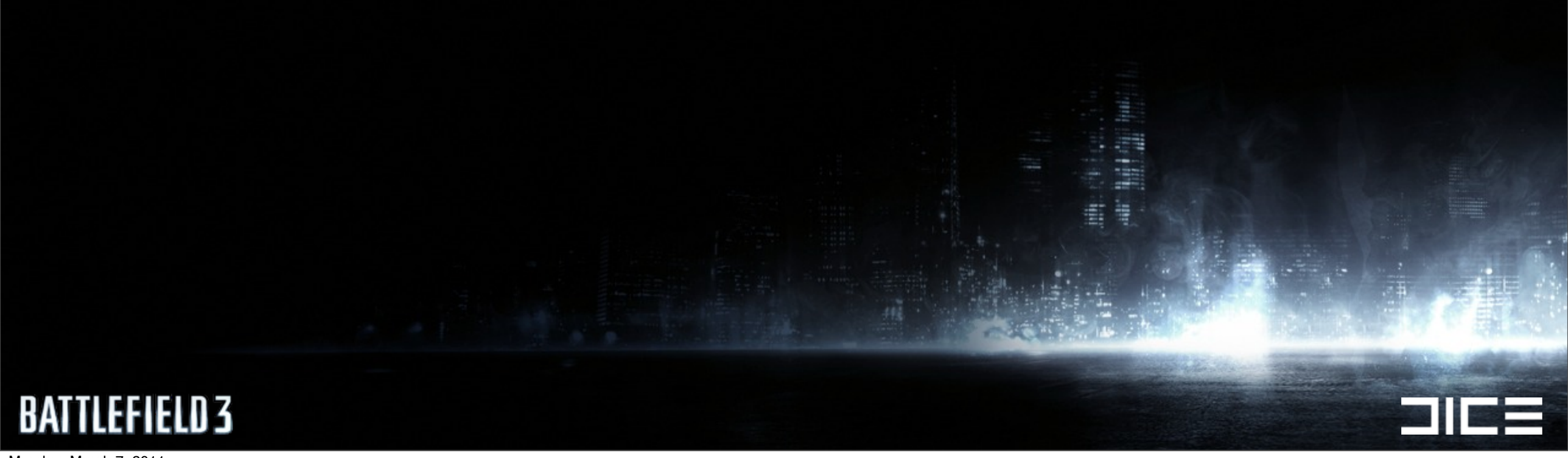
Details of the new culling

- Improve performance with a simple grid



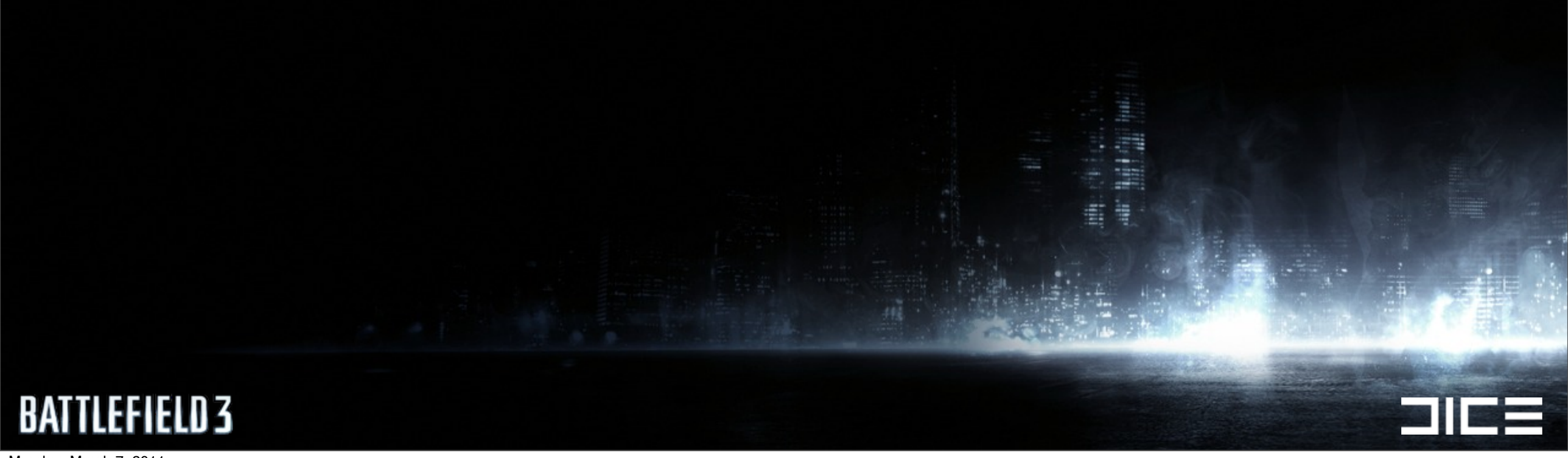
Details of the new culling

- Improve performance with a simple grid
- Really an AABB assigned to a “cell” with spheres



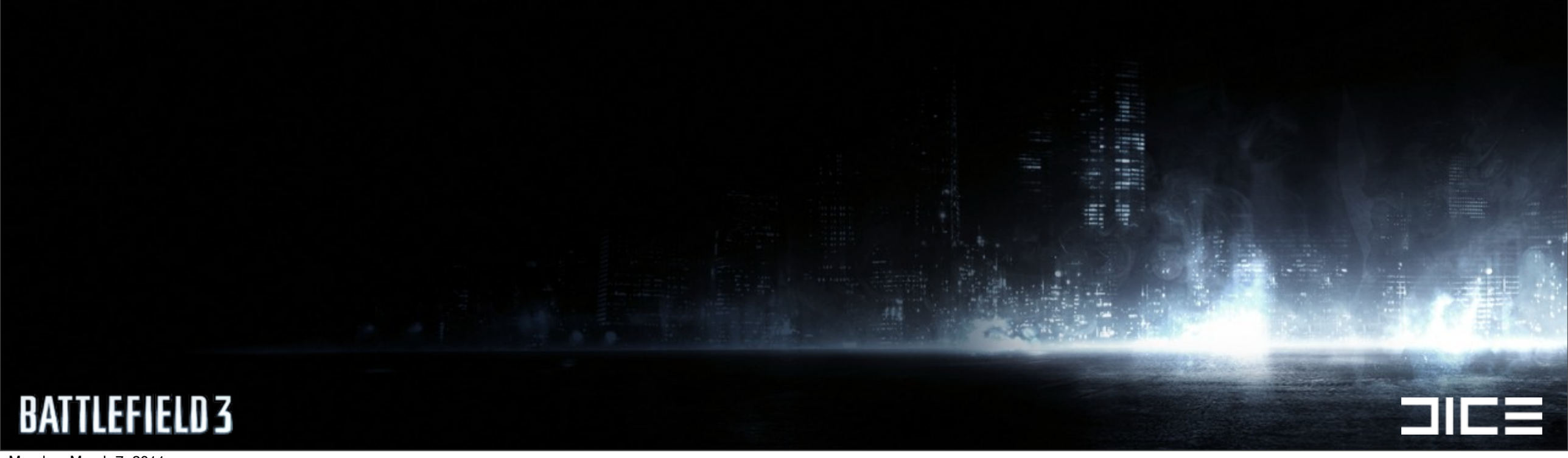
Details of the new culling

- Improve performance with a simple grid
- Really an AABB assigned to a “cell” with spheres
- Separate grids for



Details of the new culling

- Improve performance with a simple grid
- Really an AABB assigned to a “cell” with spheres
- Separate grids for
-



Details of the new culling

- Improve performance with a simple grid
- Really an AABB assigned to a “cell” with spheres
- Separate grids for
-
  - Rendering: Static

Details of the new culling

- Improve performance with a simple grid
- Really an AABB assigned to a “cell” with spheres
- Separate grids for
  - Rendering: Static
  - Rendering: Dynamic

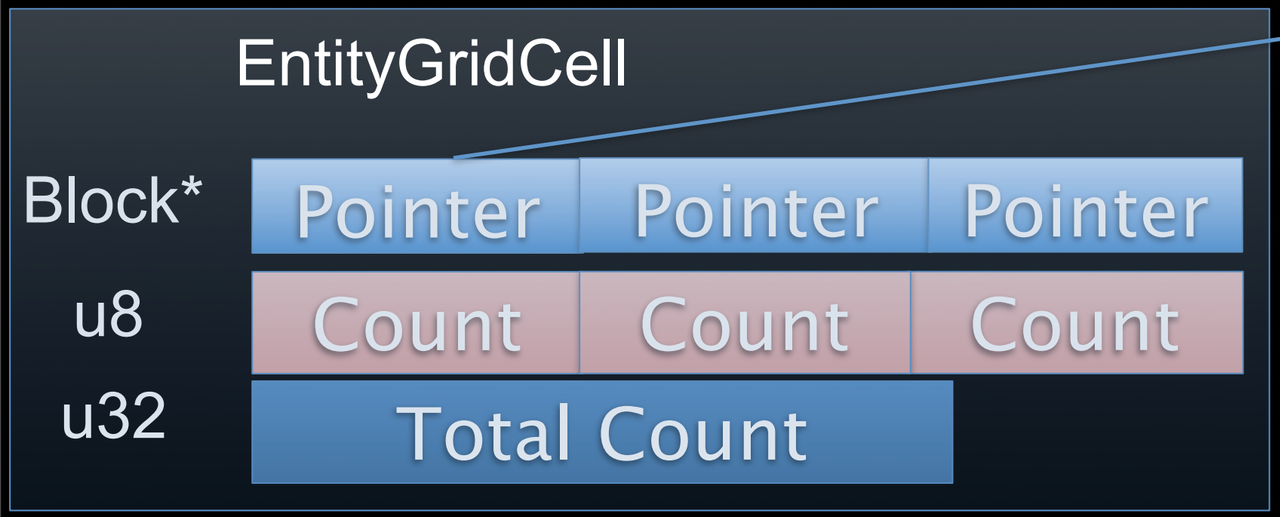
Details of the new culling

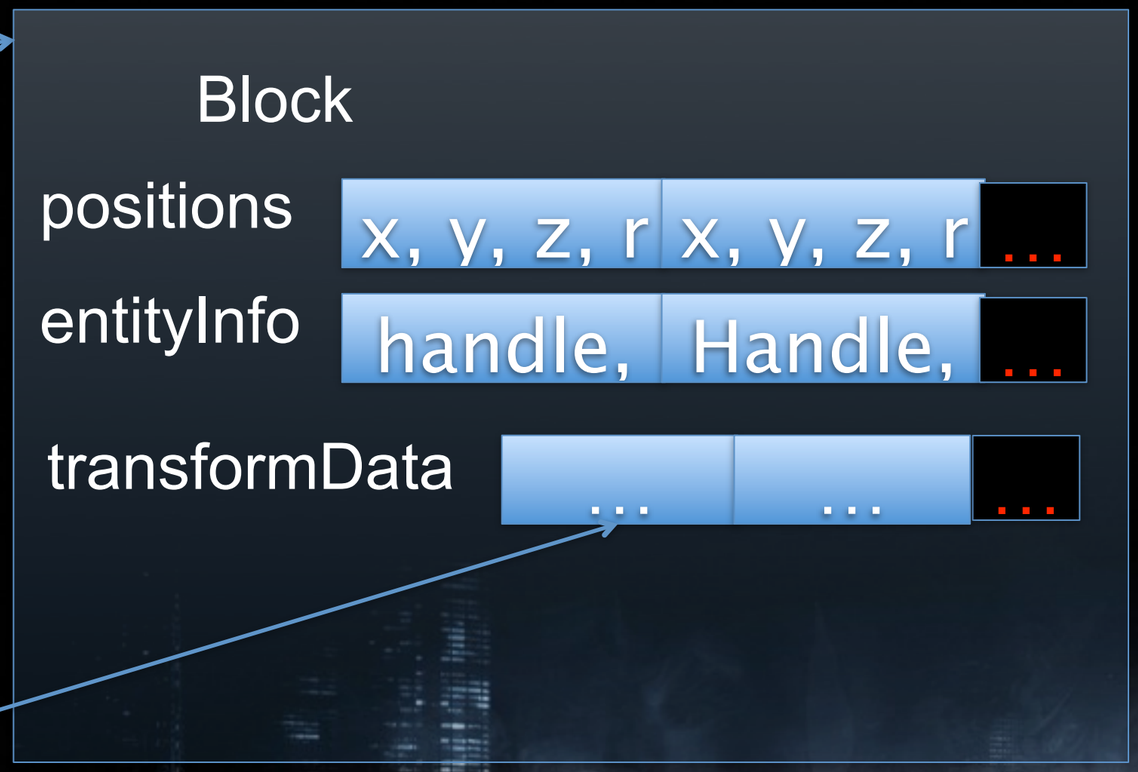
- Improve performance with a simple grid
- Really an AABB assigned to a “cell” with spheres
- Separate grids for
  - Rendering: Static
  - Rendering: Dynamic
  - Physics: Static

Details of the new culling

- Improve performance with a simple grid
- Really an AABB assigned to a “cell” with spheres
- Separate grids for
  - Rendering: Static
  - Rendering: Dynamic
  - Physics: Static
  - Physics: Dynamic

Data layout

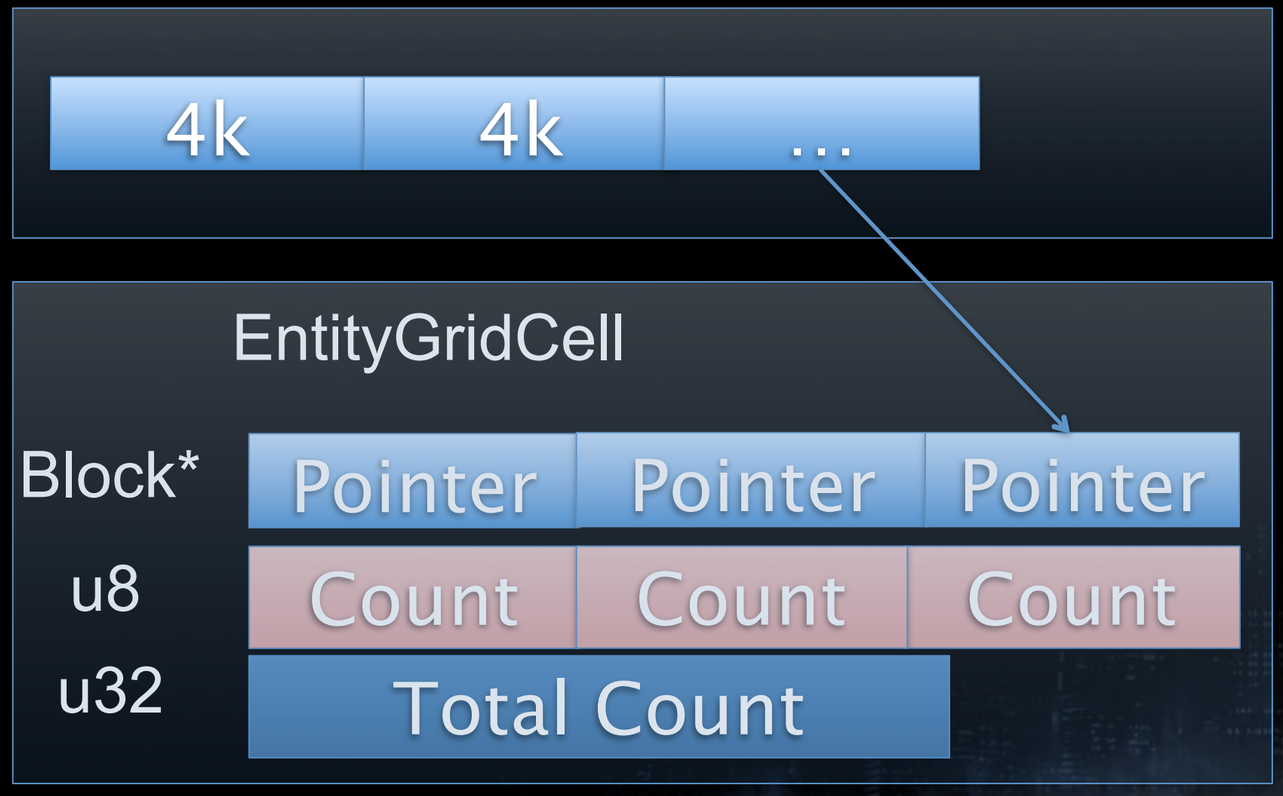





```
struct TransformData
{
        half rotation[4];
        half minAabb[3];
        half pad[1];
        half maxAabb[3];
        half scale[3];
};
```

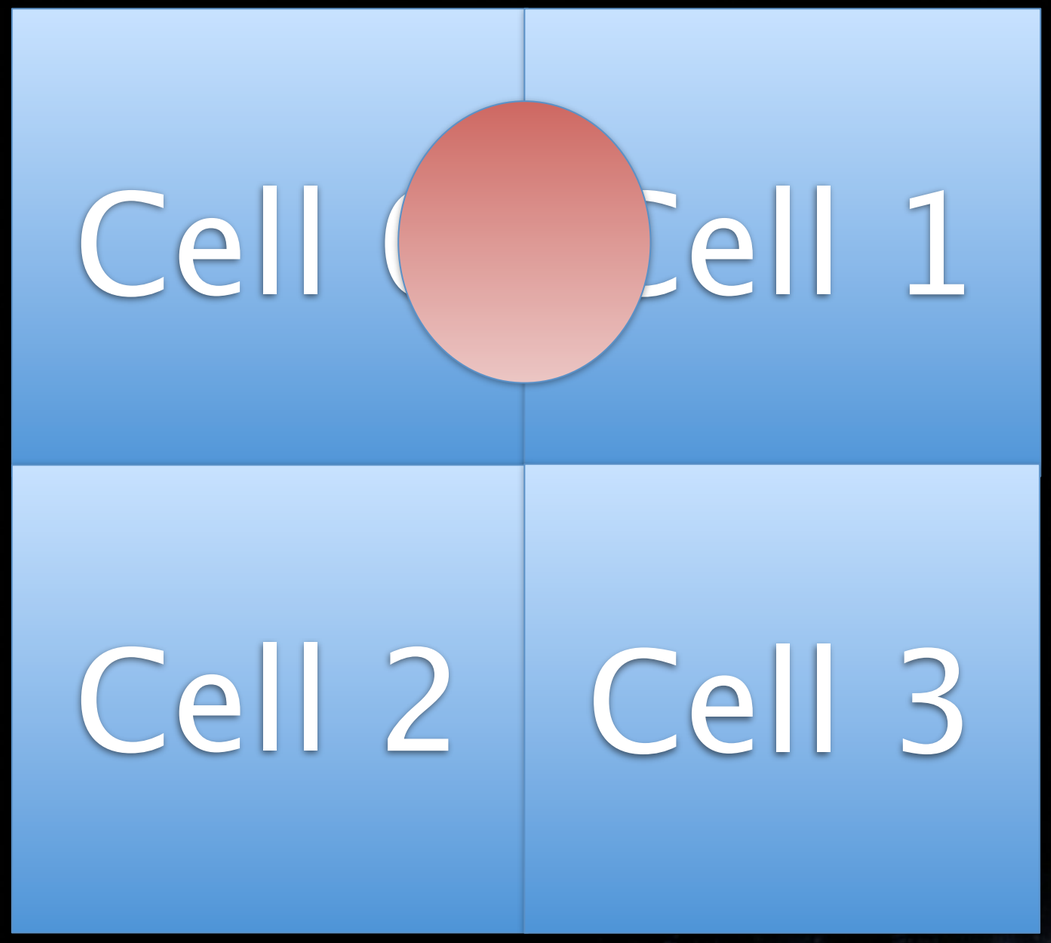
Adding objects

- Pre-allocated array that we can grab data from

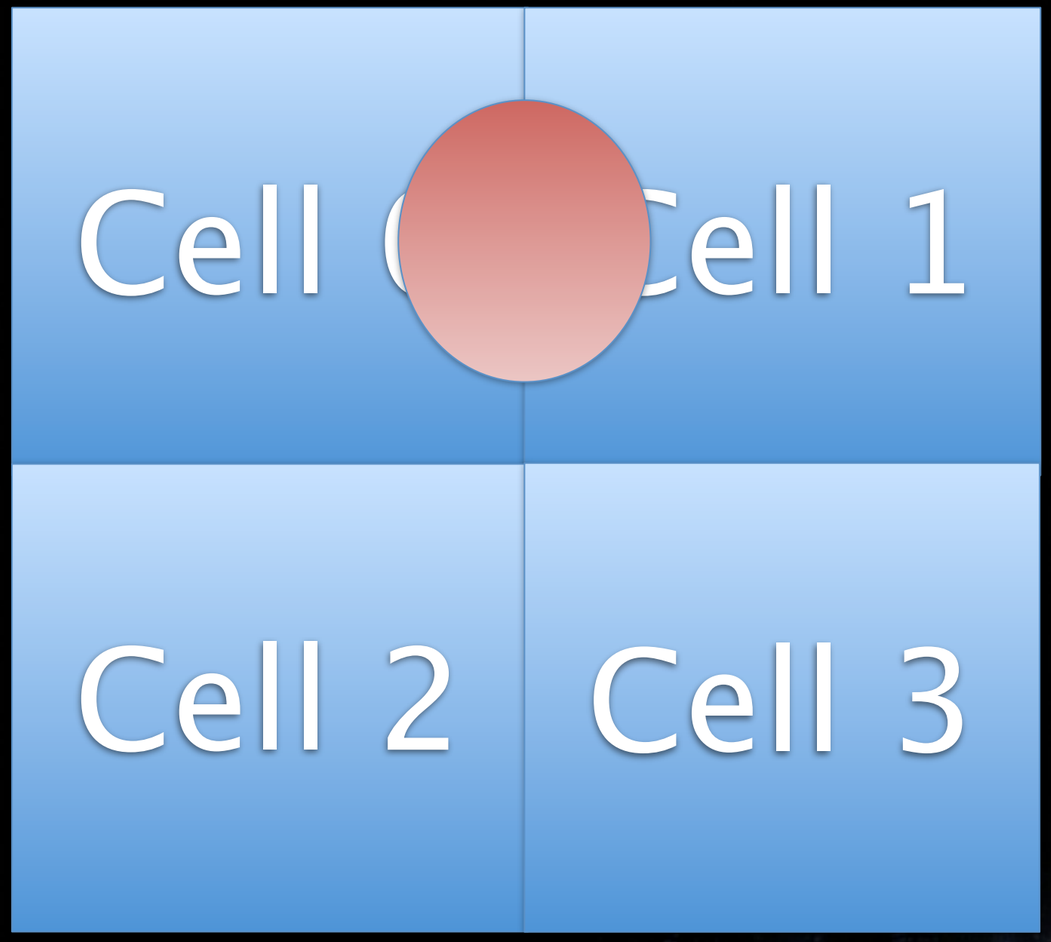


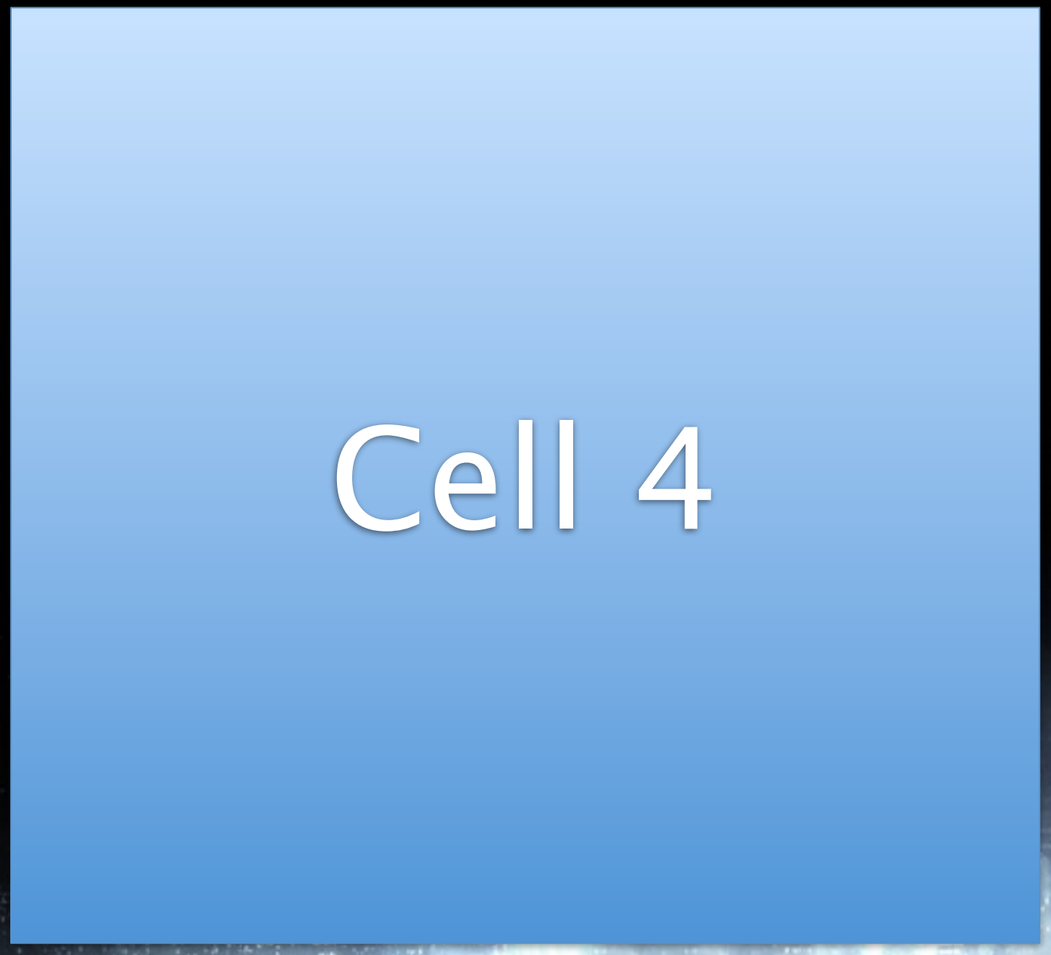
AtomicAdd(…) to “alloc” new block

Adding objects

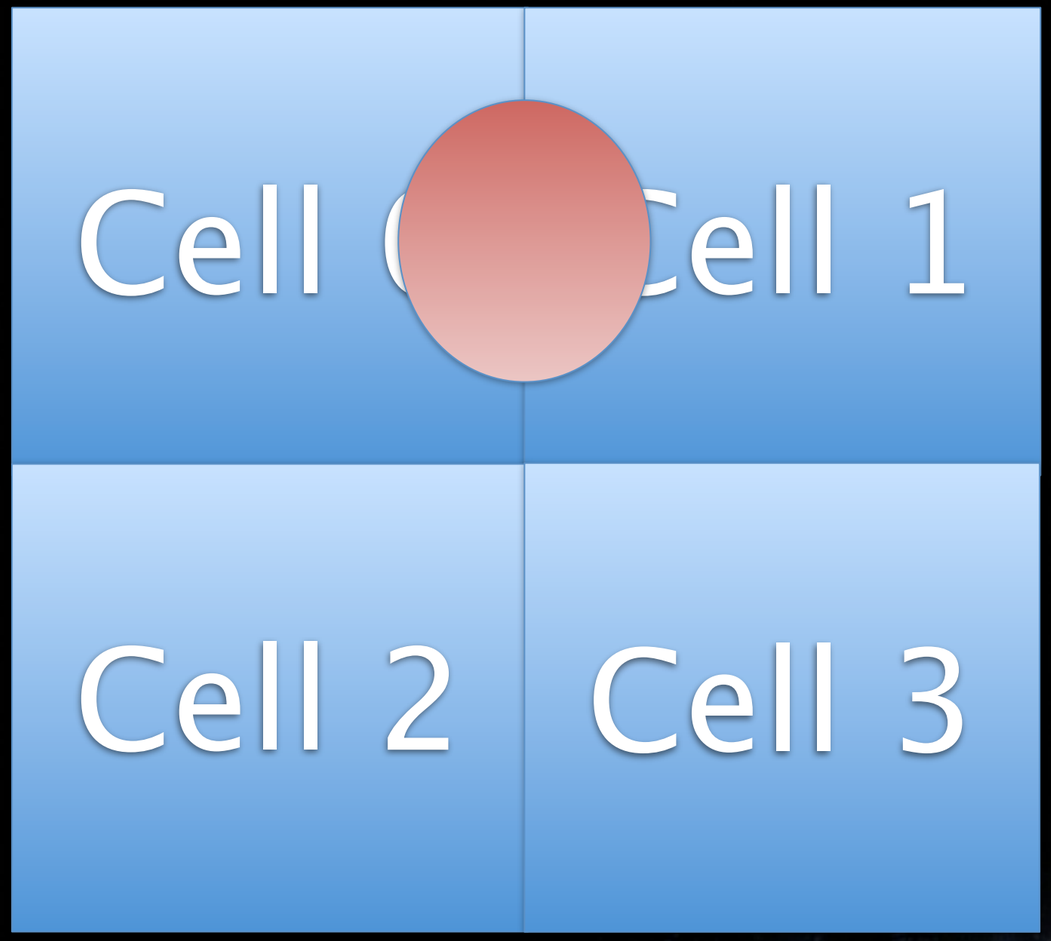


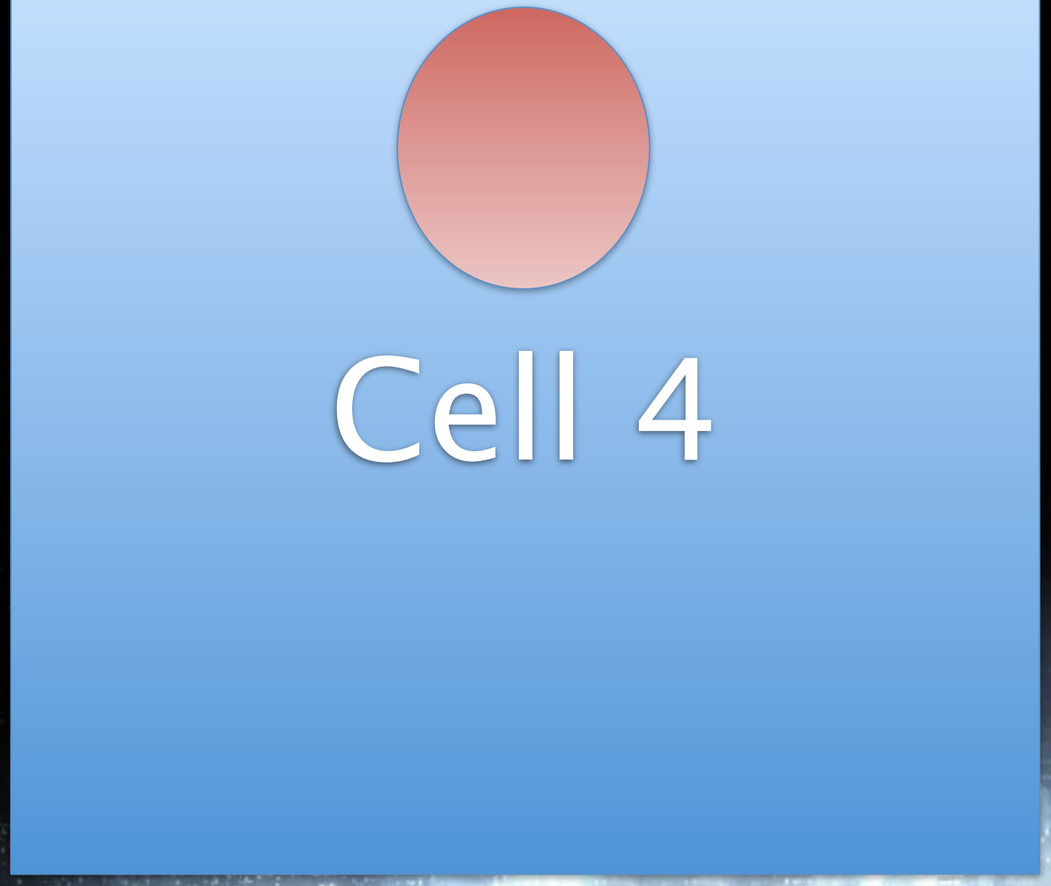
Adding objects



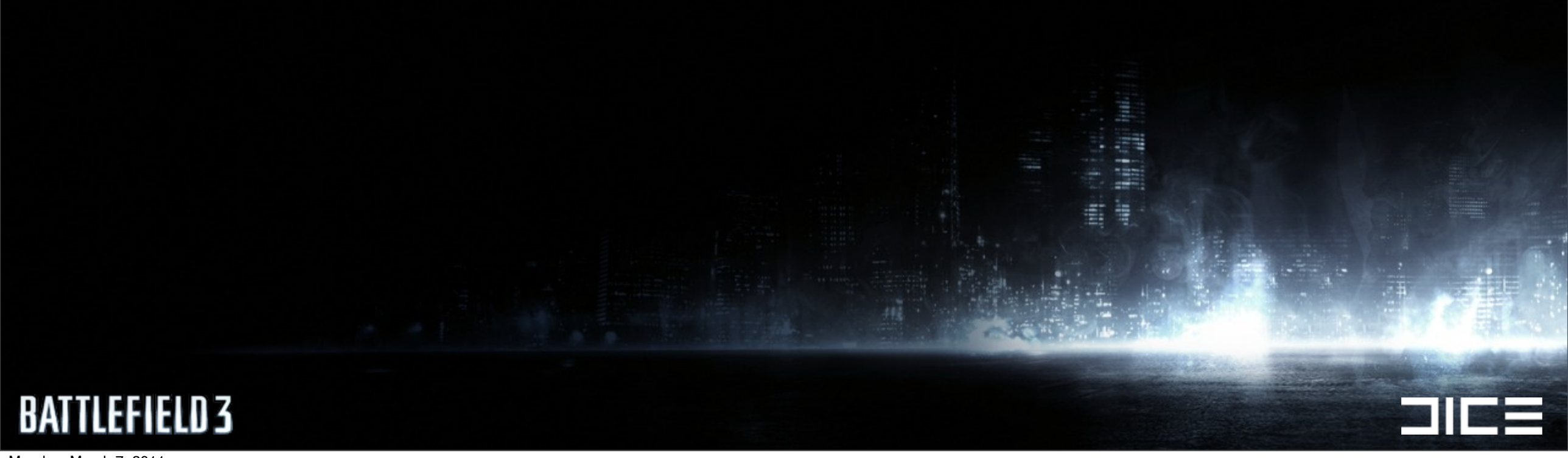


Adding objects



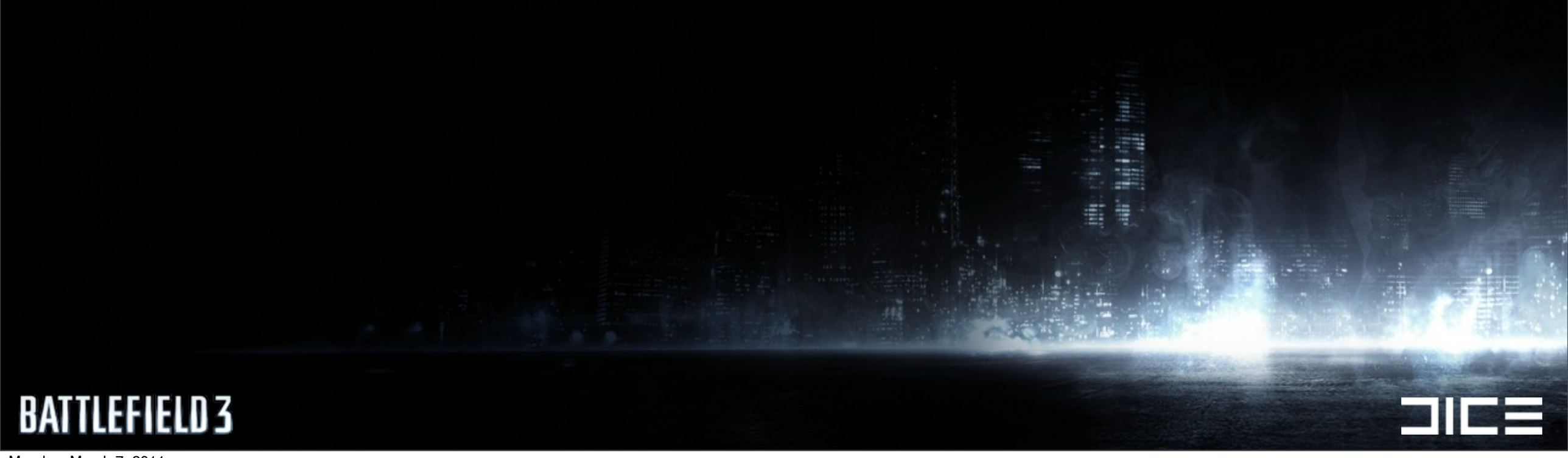


Removing objects



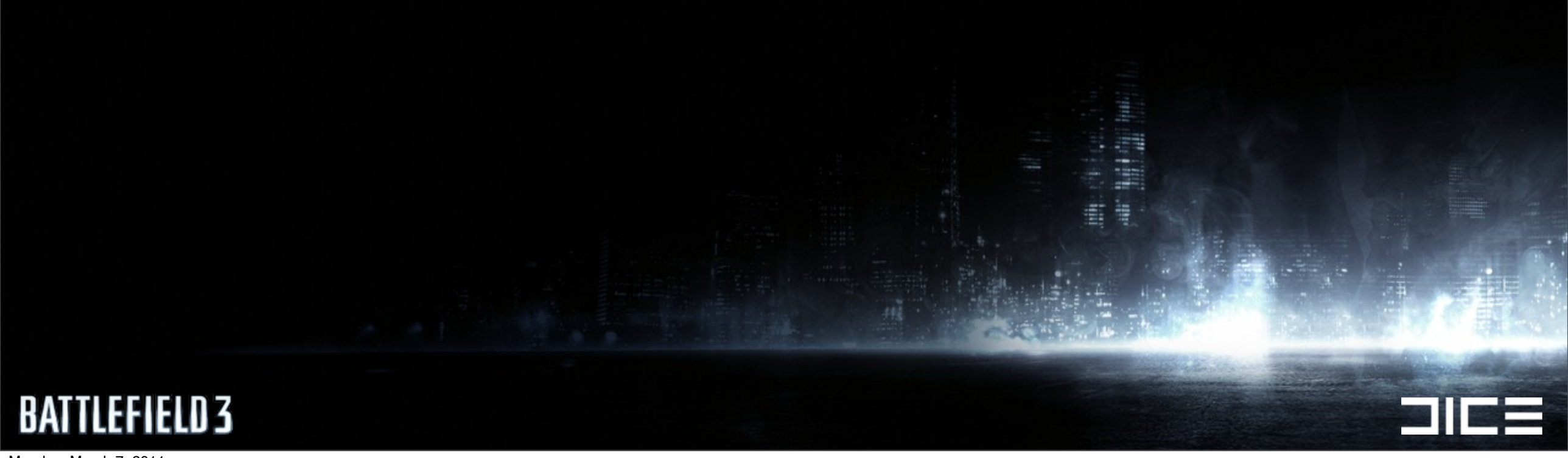
Removing objects

- Use the “swap trick”



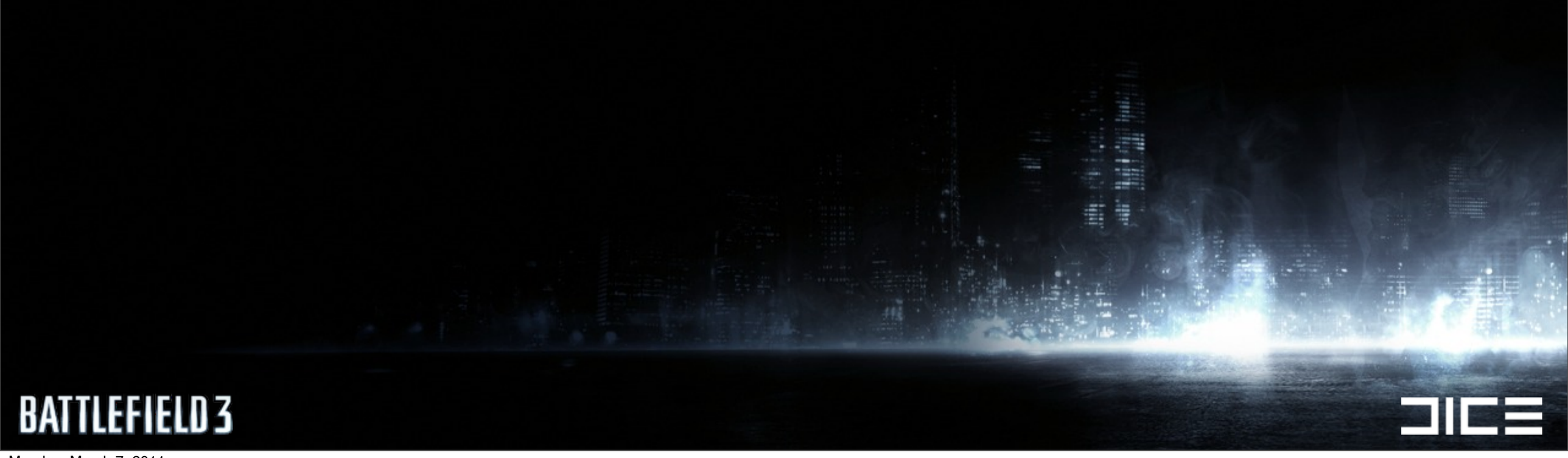
Removing objects

- Use the “swap trick”
- Data doesn’t need to be sorted



Removing objects

- Use the “swap trick”
- Data doesn’t need to be sorted
- Just swap with the last entry and decrease the count



Rendering culling

- Let’s look at what the rendering expects

```
struct EntityRenderCullInfo
{
    Handle entity;    // handle to the entity
    u16 visibleViews; // bits of which frustums that was visible
    u16 classId;      // type of mesh
    float screenArea; // at which screen area entity should be culled
};
```

Culling code

```
while (1)
{
	uint blockIter = interlockedIncrement(currentBlockIndex) - 1;

	if (blockIter >= blockCount) break;

	u32 masks[EntityGridCell::Block::MaxCount] = {}, frustumMask = 1;
	block = gridCell->blocks[blockIter];

	foreach (frustum in frustums, frustumMask <<= 1)
	{
		for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
		{
			u32 inside = intersect(frustum, block->postition[i]);
			masks[i] |= frustumMask & inside;
		}
	}

	for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
	{
		// filter list here (if masks[i] is zero it should be skipped)
		// ...
	}
}
```

Culling code

```
while (1)
{
        uint blockIter = interlockedIncrement(currentBlockIndex) - 1;

        if (blockIter >= blockCount) break;

        u32 masks[EntityGridCell::Block::MaxCount] = {}, frustumMask = 1;
        block = gridCell->blocks[blockIter];

        foreach (frustum in frustums, frustumMask <<= 1)
        {
                for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
                {
                        u32 inside = intersect(frustum, block->postition[i]);
                        masks[i] |= frustumMask & inside;
                }
        }

        for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
        {
                // filter list here (if masks[i] is zero it should be skipped)
                // ...
        }
}
```

Culling code

```
while (1)
{
        uint blockIter = interlockedIncrement(currentBlockIndex) - 1;

        if (blockIter >= blockCount) break;

        u32 masks[EntityGridCell::Block::MaxCount] = {}, frustumMask = 1;
        block = gridCell->blocks[blockIter];

        foreach (frustum in frustums, frustumMask <<= 1)
        {
                for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
                {
                        u32 inside = intersect(frustum, block->postition[i]);
                        masks[i] |= frustumMask & inside;
                }
        }

        for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
        {
                // filter list here (if masks[i] is zero it should be skipped)
                // ...
        }
}
```

Culling code

```
while (1)
{
        uint blockIter = interlockedIncrement(currentBlockIndex) - 1;

        if (blockIter >= blockCount) break;

        u32 masks[EntityGridCell::Block::MaxCount] = {}, frustumMask = 1;
        block = gridCell->blocks[blockIter];

        foreach (frustum in frustums, frustumMask <<= 1)
        {
                for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
                {
                        u32 inside = intersect(frustum, block->postition[i]);
                        masks[i] |= frustumMask & inside;
                }
        }

        for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
        {
                // filter list here (if masks[i] is zero it should be skipped)
                // ...
        }
}
```

Culling code

```
while (1)
{
        uint blockIter = interlockedIncrement(currentBlockIndex) - 1;

        if (blockIter >= blockCount) break;

        u32 masks[EntityGridCell::Block::MaxCount] = {}, frustumMask = 1;
        block = gridCell->blocks[blockIter];

        foreach (frustum in frustums, frustumMask <<= 1)
        {
                for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
                {
                        u32 inside = intersect(frustum, block->postition[i]);
                        masks[i] |= frustumMask & inside;
                }
        }

        for (i = 0; i < gridCell->blockCounts[blockIter]; ++i)
        {
                // filter list here (if masks[i] is zero it should be skipped)
                // ...
        }
}
```

Intersection Code

```
bool intersect(const Plane* frustumPlanes, Vec4 pos)
{
	float radius = pos.w;
	if (distance(frustumPlanes[Frustum::Far], pos) > radius)
		return false;
	if (distance(frustumPlanes[Frustum::Near], pos) > radius)
		return false;
	if (distance(frustumPlanes[Frustum::Right], pos) > radius)
		return false;
	if (distance(frustumPlanes[Frustum::Left], pos) > radius)
		return false;
	if (distance(frustumPlanes[Frustum::Upper], pos) > radius)
		return false;
	if (distance(frustumPlanes[Frustum::Lower], pos) > radius)
		return false;

	return true;
}
```


ARGH! My eyes!

See “Typical C++ Bullshit” by @mike_acton

Intersection Code

```
bool intersect(const Plane* frustumPlanes, Vec4 pos)
{
    float radius = pos.w;
    if (distance(frustumPlanes[Frustum::Far], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Near], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Right], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Left], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Upper], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Lower], pos) > radius)
        return false;

    return true;
}
```

Intersection Code

LHS!

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    radius = pos.w;
if (distance(frustumPlanes[Frustum::Far], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Near], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Right], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Left], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Upper], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Lower], pos) > radius)
    return false;

return true;
}
```

Intersection Code

LHS!

LHS!

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    radius = pos.w;
if (distance(frustumPlanes[Frustum::Far], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Near], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Right], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Left], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Upper], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Lower], pos) > radius)
    return false;

return true;
}
```

Intersection Code

LHS!

LHS!

LHS!

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    float radius = pos.w;
    if (distance(frustumPlanes[Frustum::Far], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Near], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Right], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Left], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Upper], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Lower], pos) > radius)
        return false;

    return true;
}
```

Intersection Code

LHS!

LHS!

LHS!

Float branch!

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    float radius = pos.w;
    if (distance(frustumPlanes[Frustum::Far], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Near], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Right], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Left], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Upper], pos) > radius)
        return false;
    if (distance(frustumPlanes[Frustum::Lower], pos) > radius)
        return false;

    return true;
}
```

Intersection Code

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    radius = pos.w;
if (distance(frustumPlanes[Frustum::Far], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Near], pos) >
    return false;
if (distance(frustumPlanes[Frustum::Right], pos)
    return false;
if (distance(frustumPlanes[Frustum::Left], pos) >
    return false;
if (distance(frustumPlanes[Frustum::Upper], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Lower], pos) > radius)
    return false;

return true;
}
```

LHS!

LHS!

LHS!

Float branch!

Float branch!

Intersection Code

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    radius = pos.w;
if (distance(frustumPlanes[Frustum::Far], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Near], pos) >
    return false;
if (distance(frustumPlanes[Frustum::Right], pos)
    return false;
if (distance(frustumPlanes[Frustum::Left], pos) >
    return false;
if (distance(frustumPlanes[Frustum::Upper],
    return false;
if (distance(frustumPlanes[Frustum::Lower], pos)
    return false;

return true;
}
```

LHS!

LHS!

LHS!

Float branch!

Float branch!

Float branch!

Intersection Code

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    radius = pos.w;
if (distance(frustumPlanes[Frustum::Far], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Near], pos) >
    return false;
if (distance(frustumPlanes[Frustum::Right], pos)
    return false;
if (distance(frustumPlanes[Frustum::Left], pos) >
    return false;
if (distance(frustumPlanes[Frustum::Upper],
    return false;
if (distance(frustumPlanes[Frustum::Lower], pos)
    return false;

    true;
```

LHS!

Float branch!

LHS!

Float branch!

LHS!

Float branch!

LHS!

Intersection Code

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    radius = pos.w;
if (distance(frustumPlanes[Frustum::Far], pos) > radius)
    return false;
if (distance(frustumPlanes[Frustum::Near], pos) >
    return false;
if (distance(frustumPlanes[Frustum::Right], pos)
    return false;
if (distance(frustumPlanes[Frustum::Left], pos) >
    return false;
if (distance(frustumPlanes[Frustum::Upper],
    return false;
if (distance(frustumPlanes[Frustum::Lower], pos)
    return false;

    true;
```

LHS!

Float branch!

LHS!

Float branch!

LHS!

Float branch!

LHS!

Float branch!

Intersection Code

```
intersect(const Plane* frustumPlanes, Vec4 pos)

    radius
if (distan                                  um::Far], pos) > radius)
    retur
    (dista                                  m::Near], pos) >
    retur
if (dista                                   m::Right], pos)
    retu
if (dista                                   m::Left], pos) >
    retur
    (distanc                                :Upper],
    return false;
if (distance(frustumPlanes[Frustum::Lower], pos)
    return false;

    rue;
```

LHS!

LHS!

LHS!

LHS!

So what do the consoles think?

:(

Float branch!

Float branch!

Float branch!

Float branch!

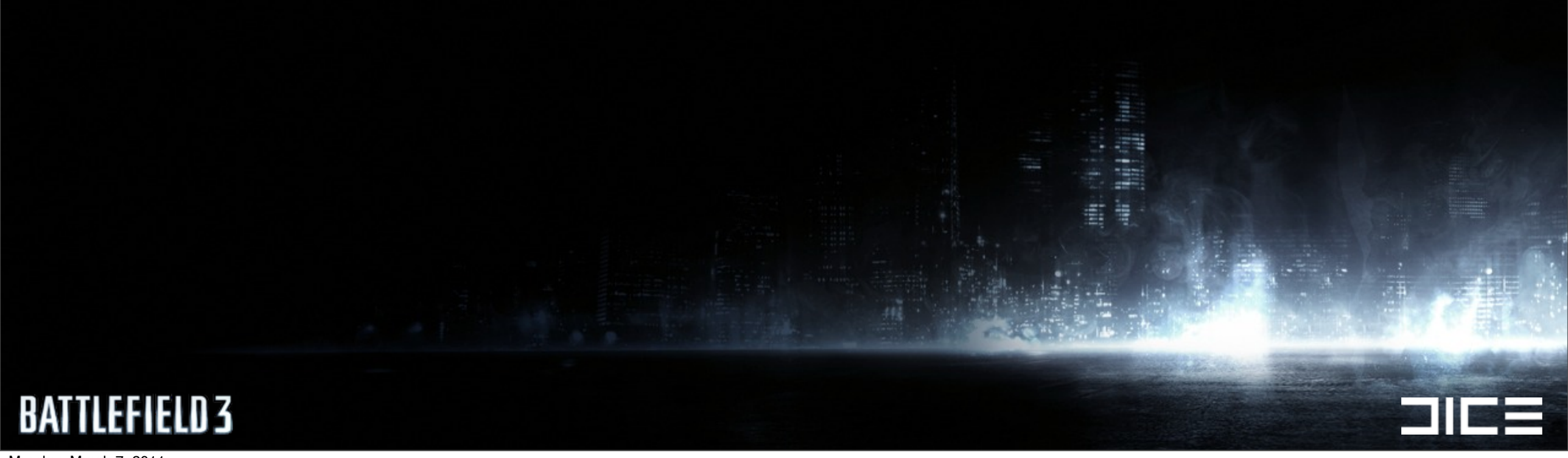
Inter Code

```
bool intersect                                          pos)
{
     float
     if (d           lanes[Frustum::Fa                    radius)

     if              (f          [Frustum::Near],           radius)
                    false
     if             e(frustu          stum::Right],          radius)
                    false;
     if             e(frustumPla               :Left],        adius)
                    false;
     if              (frustumPlanes[            er]           radius)
                     lse;
     if (d            ustumPlanes[Frust                    > radius)
          re

     return tr
}
```

BATTLEFIELD 3

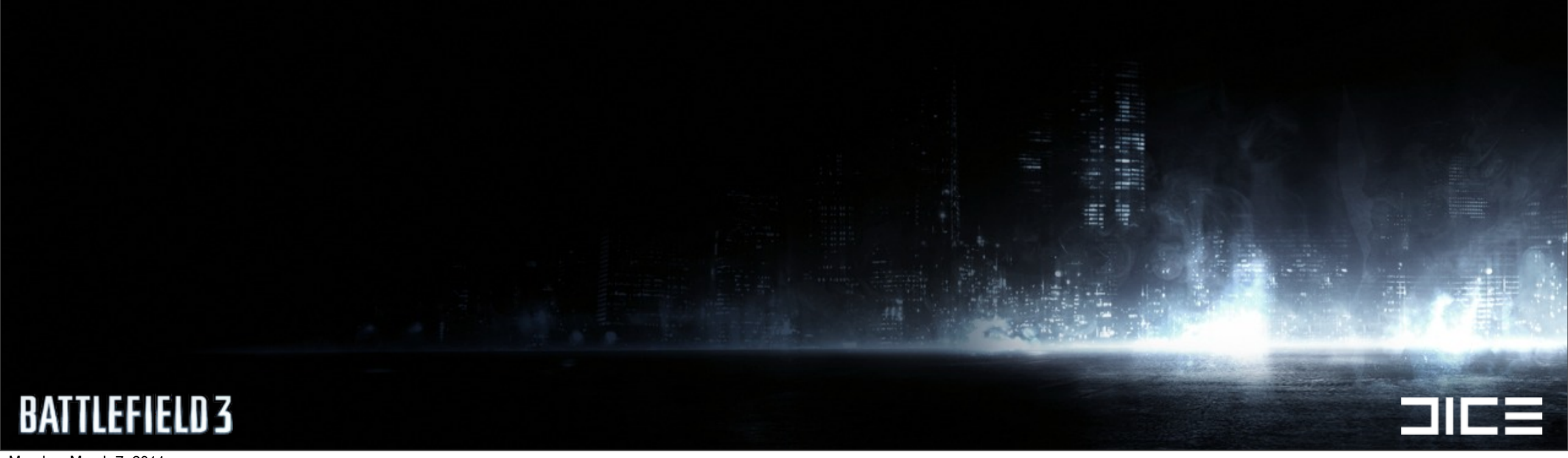
DICE

Intersection Code



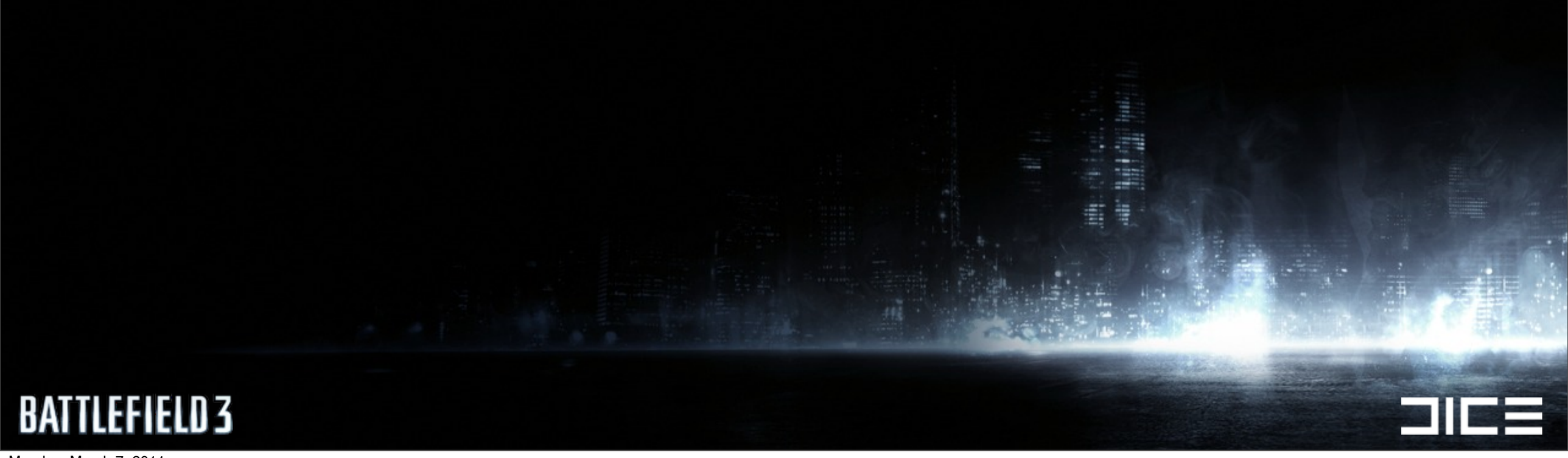
Intersection Code

- How can we improve this?



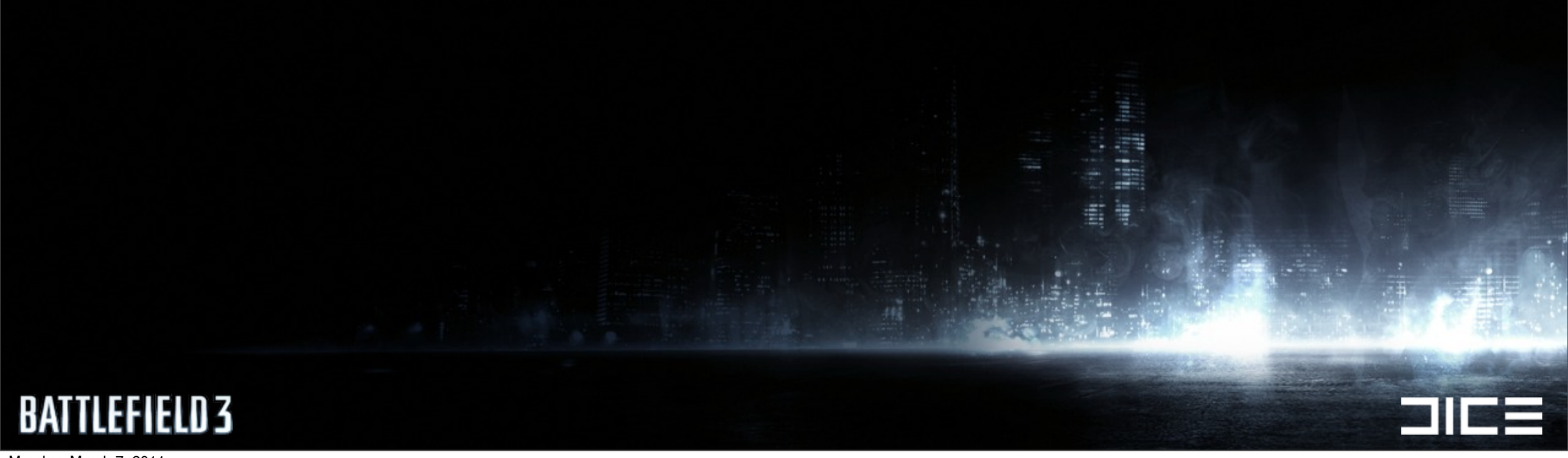
Intersection Code

- How can we improve this?
- Dot products are not very SIMD friendly



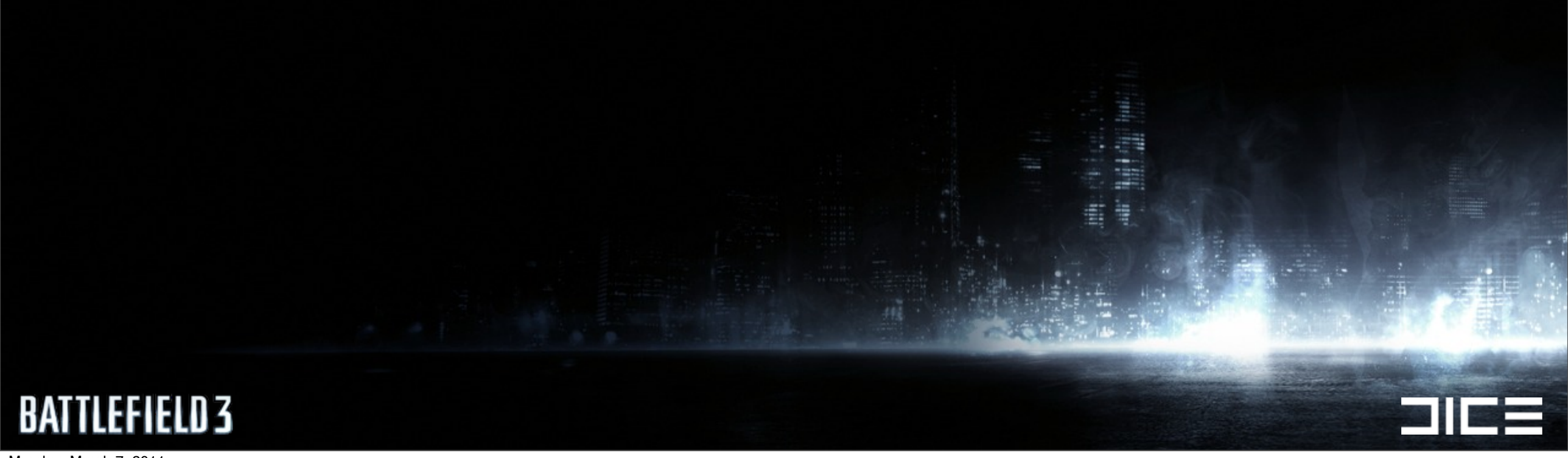
Intersection Code

- How can we improve this?
- Dot products are not very SIMD friendly
- Usually need to shuffle data around to get result



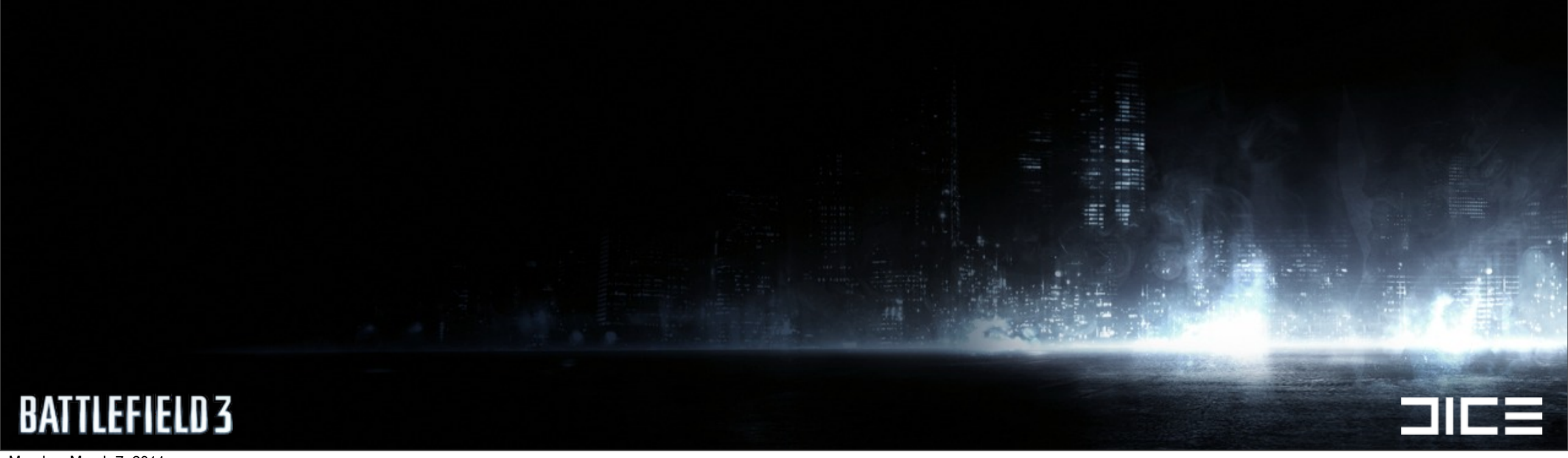
Intersection Code

- How can we improve this?
- Dot products are not very SIMD friendly
- Usually need to shuffle data around to get result
- (x0 * x1 + y0 * y1 + z0 * z1 + w0 * w1)

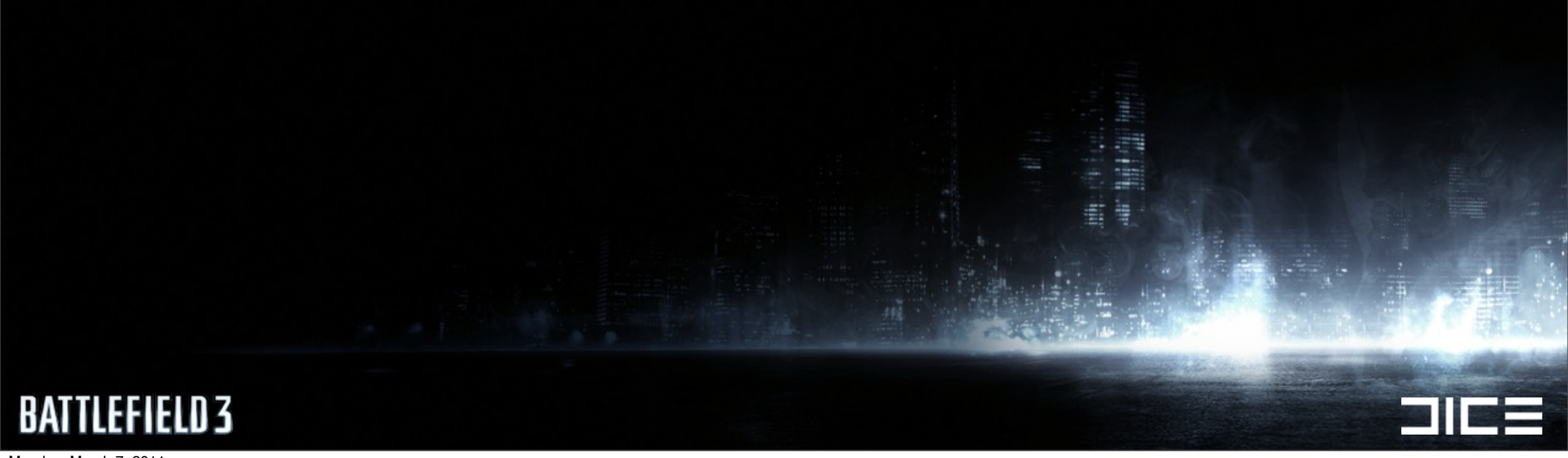


Intersection Code

- How can we improve this?
- Dot products are not very SIMD friendly
- Usually need to shuffle data around to get result
- (x0 * x1 + y0 * y1 + z0 * z1 + w0 * w1)

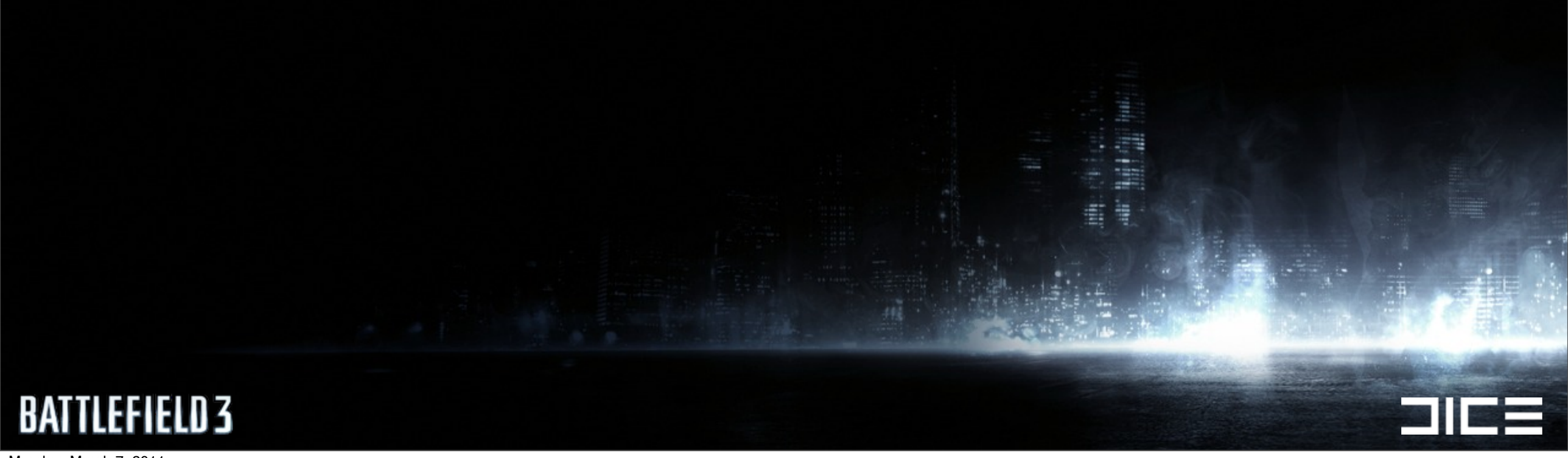


Intersection Code



Intersection Code

- Rearrange the data from AoS to SoA



Intersection Code

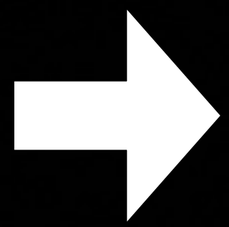
- Rearrange the data from AoS to SoA

```
Vec 0 X0 Y0 Z0 W0
Vec 1 X1 Y1 Z1 W1
Vec 2 X2 Y2 Z2 W2
Vec 3 X3 Y3 Z3 W3
```

Intersection Code

- Rearrange the data from AoS to SoA

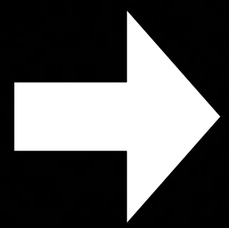
```
Vec 0 X0 Y0 Z0 W0
Vec 1 X1 Y1 Z1 W1
Vec 2 X2 Y2 Z2 W2
Vec 3 X3 Y3 Z3 W3
```



Intersection Code

- Rearrange the data from AoS to SoA

```
Vec 0 X0 Y0 Z0 W0
Vec 1 X1 Y1 Z1 W1
Vec 2 X2 Y2 Z2 W2
Vec 3 X3 Y3 Z3 W3
```

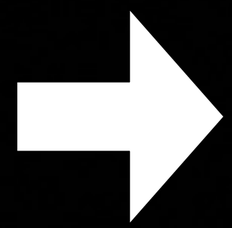


```
VecX X0 X1 X2 X3
VecY Y0 Y1 Y2 Y3
VecZ Z0 Z1 Z2 Z3
VecW W0 W1 W2 W3
```

Intersection Code

- Rearrange the data from AoS to SoA

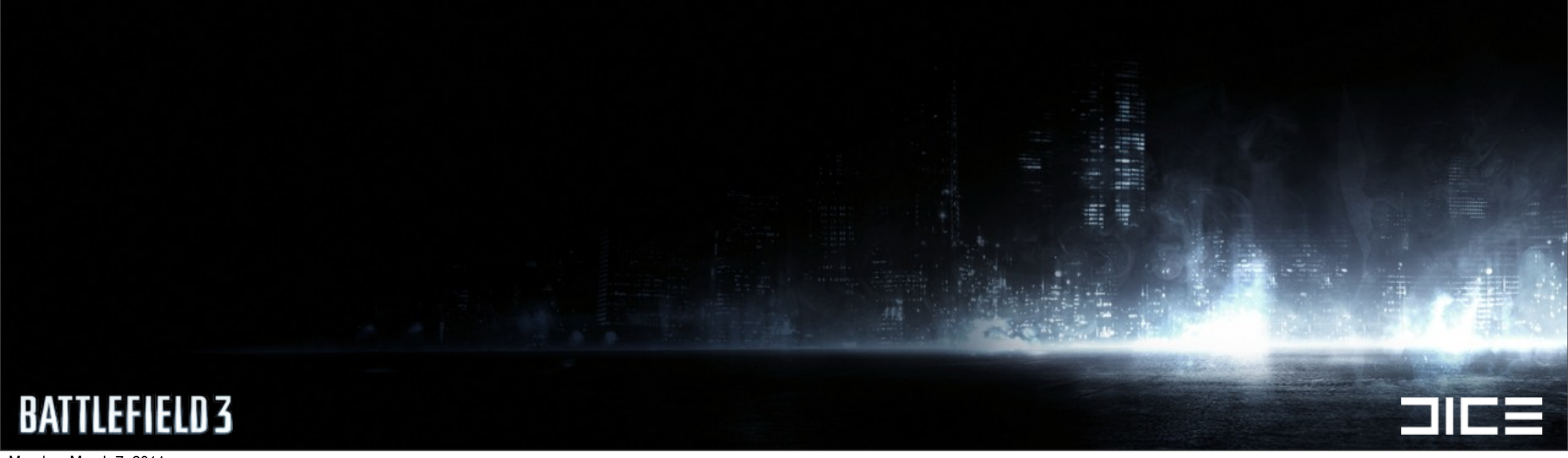
```
Vec 0 X0 Y0 Z0 W0
Vec 1 X1 Y1 Z1 W1
Vec 2 X2 Y2 Z2 W2
Vec 3 X3 Y3 Z3 W3
```



```
VecX X0 X1 X2 X3
VecY Y0 Y1 Y2 Y3
VecZ Z0 Z1 Z2 Z3
VecW W0 W1 W2 W3
```

- Now we only need 3 instructions for 4 dots!

Rearrange the frustum planes

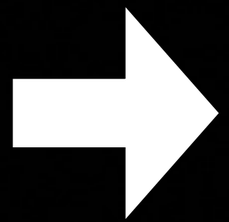


Rearrange the frustum planes

```
Plane 0 X0 Y0 Z0 W0
Plane 1 X1 Y1 Z1 W1
Plane 2 X2 Y2 Z2 W2
Plane 3 X3 Y3 Z3 W3
Plane 4 X4 Y4 Z4 W4
Plane 5 X5 Y5 Z5 W5
```

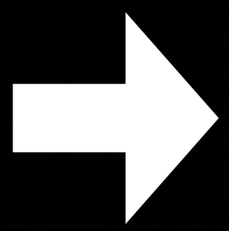
Rearrange the frustum planes

```
Plane 0 X0 Y0 Z0 W0
Plane 1 X1 Y1 Z1 W1
Plane 2 X2 Y2 Z2 W2
Plane 3 X3 Y3 Z3 W3
Plane 4 X4 Y4 Z4 W4
Plane 5 X5 Y5 Z5 W5
```



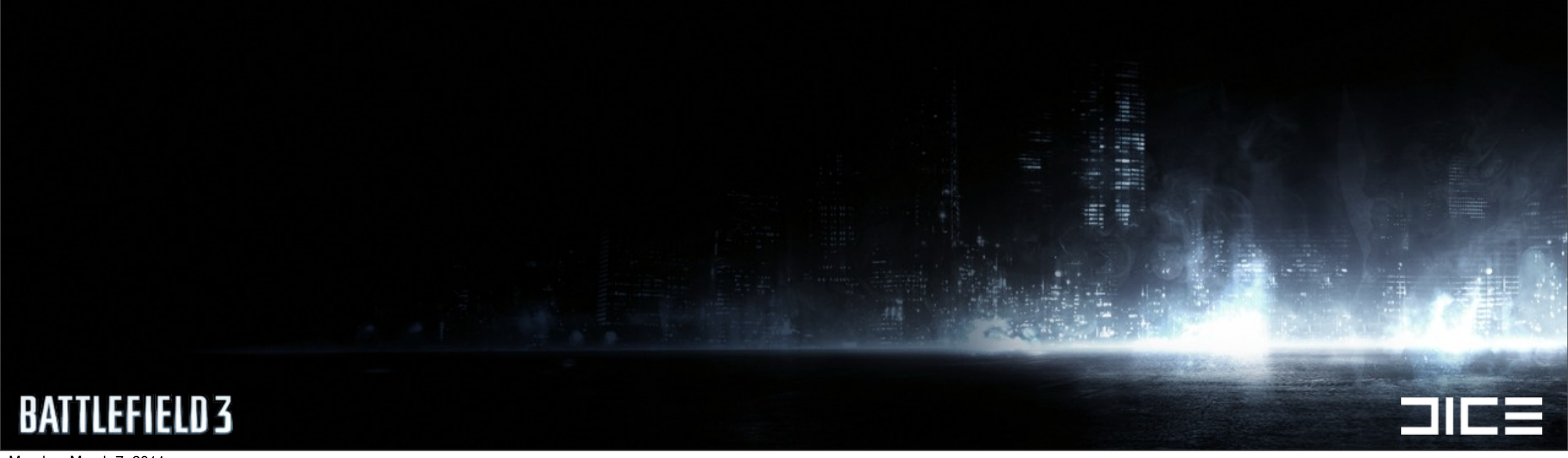
Rearrange the frustum planes

```
Plane 0 X0 Y0 Z0 W0
Plane 1 X1 Y1 Z1 W1
Plane 2 X2 Y2 Z2 W2
Plane 3 X3 Y3 Z3 W3
Plane 4 X4 Y4 Z4 W4
Plane 5 X5 Y5 Z5 W5
```



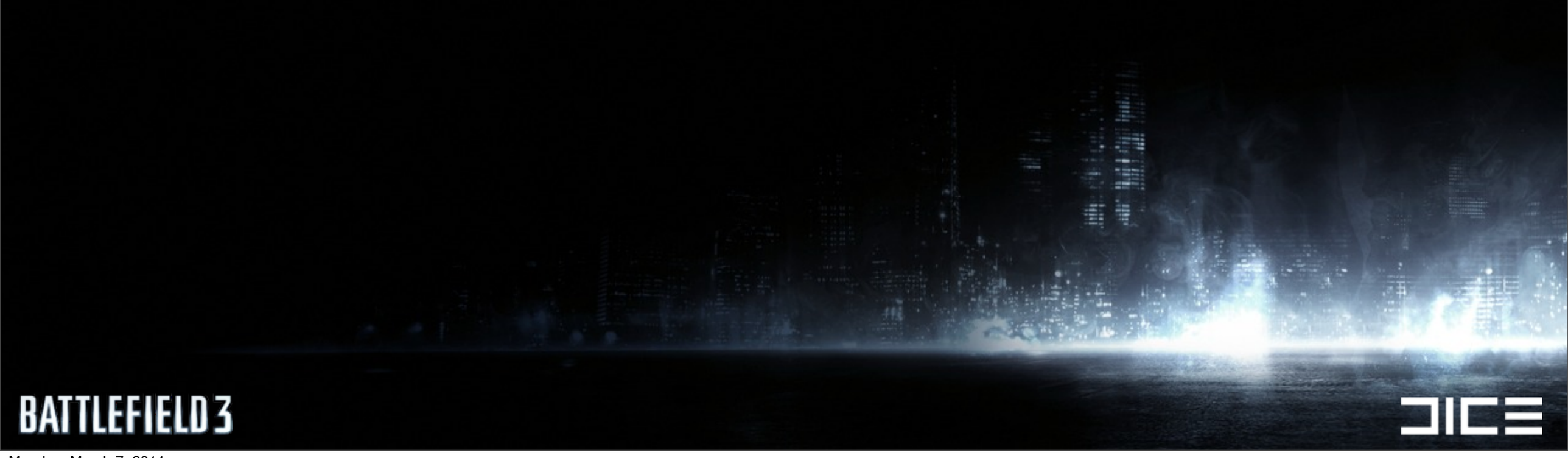
```
X0 X1 X2 X3
Y0 Y1 Y2 Y3
Z0 Z1 Z2 Z3
W0 W1 W2 W3

X4 X5 X4 X5
Y4 Y5 Y4 Y5
Z4 Z5 Z4 Z5
W4 W5 W4 W5
```

New intersection code



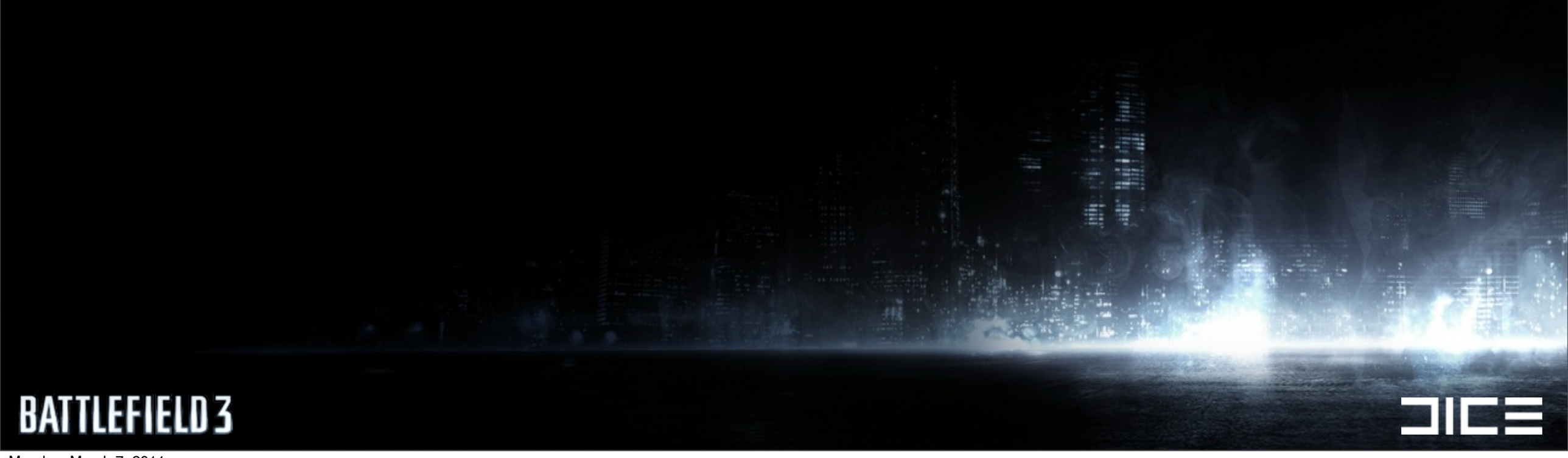
New intersection code

- Two frustum vs Sphere intersections per loop



New intersection code

- Two frustum vs Sphere intersections per loop
- 4 * 3 dot products with 9 instructions



New intersection code

- Two frustum vs Sphere intersections per loop
- 4 * 3 dot products with 9 instructions
- Loop over all frustums and merge the result

New intersection code

- Two frustum vs Sphere intersections per loop
- 4 * 3 dot products with 9 instructions
- Loop over all frustums and merge the result

New intersection code (1/4)

```
Vec posA_xxxx = vecShuffle<VecMask::_xxxx>(posA);
Vec posA_yyyy = vecShuffle<VecMask::_yyyy>(posA);
Vec posA_zzzz = vecShuffle<VecMask::_zzzz>(posA);
Vec posA_rrrr = vecShuffle<VecMask::_wwww>(posA);

// 4 dot products

dotA_0123 = vecMulAdd(posA_zzzz, pl_z0z1z2z3, pl_w0w1w2w3);
dotA_0123 = vecMulAdd(posA_yyyy, pl_y0y1y2y3, dotA_0123);
dotA_0123 = vecMulAdd(posA_xxxx, pl_x0x1x2x3, dotA_0123);
```

New intersection code (1/4)

```
Vec posA_xxxx = vecShuffle<VecMask::_xxxx>(posA);
Vec posA_yyyy = vecShuffle<VecMask::_yyyy>(posA);
Vec posA_zzzz = vecShuffle<VecMask::_zzzz>(posA);
Vec posA_rrrr = vecShuffle<VecMask::_wwww>(posA);

// 4 dot products

dotA_0123 = vecMulAdd(posA_zzzz, pl_z0z1z2z3, pl_w0w1w2w3);
dotA_0123 = vecMulAdd(posA_yyyy, pl_y0y1y2y3, dotA_0123);
dotA_0123 = vecMulAdd(posA_xxxx, pl_x0x1x2x3, dotA_0123);
```

New intersection code (1/4)

```
Vec posA_xxxx = vecShuffle<VecMask::_xxxx>(posA);
Vec posA_yyyy = vecShuffle<VecMask::_yyyy>(posA);
Vec posA_zzzz = vecShuffle<VecMask::_zzzz>(posA);
Vec posA_rrrr = vecShuffle<VecMask::_wwww>(posA);
```

```
// 4 dot products

dotA_0123 = vecMulAdd(posA_zzzz, pl_z0z1z2z3, pl_w0w1w2w3);
dotA_0123 = vecMulAdd(posA_yyyy, pl_y0y1y2y3, dotA_0123);
dotA_0123 = vecMulAdd(posA_xxxx, pl_x0x1x2x3, dotA_0123);
```

New intersection code (2/4)

```
Vec posB_xxxx = vecShuffle<VecMask::_xxxx>(posB);
Vec posB_yyyy = vecShuffle<VecMask::_yyyy>(posB);
Vec posB_zzzz = vecShuffle<VecMask::_zzzz>(posB);
Vec posB_rrrr = vecShuffle<VecMask::_wwww>(posB);

// 4 dot products

dotB_0123 = vecMulAdd(posB_zzzz, pl_z0z1z2z3, pl_w0w1w2w3);
dotB_0123 = vecMulAdd(posB_yyyy, pl_y0y1y2y3, dotB_0123);
dotB_0123 = vecMulAdd(posB_xxxx, pl_x0x1x2x3, dotB_0123
```

New intersection code (2/4)

```
Vec posB_xxxx = vecShuffle<VecMask::_xxxx>(posB);
Vec posB_yyyy = vecShuffle<VecMask::_yyyy>(posB);
Vec posB_zzzz = vecShuffle<VecMask::_zzzz>(posB);
Vec posB_rrrr = vecShuffle<VecMask::_wwww>(posB);

// 4 dot products

dotB_0123 = vecMulAdd(posB_zzzz, pl_z0z1z2z3, pl_w0w1w2w3);
dotB_0123 = vecMulAdd(posB_yyyy, pl_y0y1y2y3, dotB_0123);
dotB_0123 = vecMulAdd(posB_xxxx, pl_x0x1x2x3, dotB_0123
```

New intersection code (2/4)

```
Vec posB_xxxx = vecShuffle<VecMask::_xxxx>(posB);
Vec posB_yyyy = vecShuffle<VecMask::_yyyy>(posB);
Vec posB_zzzz = vecShuffle<VecMask::_zzzz>(posB);
Vec posB_rrrr = vecShuffle<VecMask::_wwww>(posB);
```

```
// 4 dot products

dotB_0123 = vecMulAdd(posB_zzzz, pl_z0z1z2z3, pl_w0w1w2w3);
dotB_0123 = vecMulAdd(posB_yyyy, pl_y0y1y2y3, dotB_0123);
dotB_0123 = vecMulAdd(posB_xxxx, pl_x0x1x2x3, dotB_0123
```

New intersection code (3/4)

```
Vec posAB_xxxx = vecInsert<VecMask::_0011>(posA_xxxx, posB_xxxx);
Vec posAB_yyyy = vecInsert<VecMask::_0011>(posA_yyyy, posB_yyyy);
Vec posAB_zzzz = vecInsert<VecMask::_0011>(posA_zzzz, posB_zzzz);
Vec posAB_rrrr = vecInsert<VecMask::_0011>(posA_rrrr, posB_rrrr);

// 4 dot products

dotA45B45 = vecMulAdd(posAB_zzzz, pl_z4z5z4z5, pl_w4w5w4w5);
dotA45B45 = vecMulAdd(posAB_yyyy, pl_y4y5y4y5, dotA45B45);
dotA45B45 = vecMulAdd(posAB_xxxx, pl_x4x5x4x5, dotA45B45);
```

New intersection code (3/4)

```
Vec posAB_xxxx = vecInsert<VecMask::_0011>(posA_xxxx, posB_xxxx);
Vec posAB_yyyy = vecInsert<VecMask::_0011>(posA_yyyy, posB_yyyy);
Vec posAB_zzzz = vecInsert<VecMask::_0011>(posA_zzzz, posB_zzzz);
Vec posAB_rrrr = vecInsert<VecMask::_0011>(posA_rrrr, posB_rrrr);

// 4 dot products

dotA45B45 = vecMulAdd(posAB_zzzz, pl_z4z5z4z5, pl_w4w5w4w5);
dotA45B45 = vecMulAdd(posAB_yyyy, pl_y4y5y4y5, dotA45B45);
dotA45B45 = vecMulAdd(posAB_xxxx, pl_x4x5x4x5, dotA45B45);
```

New intersection code (3/4)

```
Vec posAB_xxxx = vecInsert<VecMask::_0011>(posA_xxxx, posB_xxxx);
Vec posAB_yyyy = vecInsert<VecMask::_0011>(posA_yyyy, posB_yyyy);
Vec posAB_zzzz = vecInsert<VecMask::_0011>(posA_zzzz, posB_zzzz);
Vec posAB_rrrr = vecInsert<VecMask::_0011>(posA_rrrr, posB_rrrr);
```

```
// 4 dot products

dotA45B45 = vecMulAdd(posAB_zzzz, pl_z4z5z4z5, pl_w4w5w4w5);
dotA45B45 = vecMulAdd(posAB_yyyy, pl_y4y5y4y5, dotA45B45);
dotA45B45 = vecMulAdd(posAB_xxxx, pl_x4x5x4x5, dotA45B45);
```

New intersection code (4/4)

```
// Compare against radius

dotA_0123 = vecCmpGTMask(dotA_0123, posA_rrrr);
dotB_0123 = vecCmpGTMask(dotB_0123, posB_rrrr);
dotA45B45 = vecCmpGTMask(dotA45B45, posAB_rrrr);

Vec dotA45 = vecInsert<VecMask::_0011>(dotA45B45, zero);
Vec dotB45 = vecInsert<VecMask::_0011>(zero, dotA45B45);

// collect the results

Vec resA = vecOrx(dotA_0123);
Vec resB = vecOrx(dotB_0123);

resA = vecOr(resA, vecOrx(dotA45));
resB = vecOr(resB, vecOrx(dotB45));

// resA = inside or outside of frustum for point A, resB for point B

Vec rA = vecNotMask(resA);
Vec rB = vecNotMask(resB);

masksCurrent[0] |= frustumMask & rA;
masksCurrent[1] |= frustumMask & rB;
```

New intersection code (4/4)

```
// Compare against radius

dotA_0123 = vecCmpGTMask(dotA_0123, posA_rrrr);
dotB_0123 = vecCmpGTMask(dotB_0123, posB_rrrr);
dotA45B45 = vecCmpGTMask(dotA45B45, posAB_rrrr);

Vec dotA45 = vecInsert<VecMask::_0011>(dotA45B45, zero);
Vec dotB45 = vecInsert<VecMask::_0011>(zero, dotA45B45);

// collect the results

Vec resA = vecOrx(dotA_0123);
Vec resB = vecOrx(dotB_0123);

resA = vecOr(resA, vecOrx(dotA45));
resB = vecOr(resB, vecOrx(dotB45));

// resA = inside or outside of frustum for point A, resB for point B

Vec rA = vecNotMask(resA);
Vec rB = vecNotMask(resB);

masksCurrent[0] |= frustumMask & rA;
masksCurrent[1] |= frustumMask & rB;
```

New intersection code (4/4)

```
// Compare against radius

dotA_0123 = vecCmpGTMask(dotA_0123, posA_rrrr);
dotB_0123 = vecCmpGTMask(dotB_0123, posB_rrrr);
dotA45B45 = vecCmpGTMask(dotA45B45, posAB_rrrr);
```

```
Vec dotA45 = vecInsert<VecMask::_0011>(dotA45B45, zero);
Vec dotB45 = vecInsert<VecMask::_0011>(zero, dotA45B45);

// collect the results

Vec resA = vecOrx(dotA_0123);
Vec resB = vecOrx(dotB_0123);

resA = vecOr(resA, vecOrx(dotA45));
resB = vecOr(resB, vecOrx(dotB45));
```

```
// resA = inside or outside of frustum for point A, resB for point B

Vec rA = vecNotMask(resA);
Vec rB = vecNotMask(resB);

masksCurrent[0] |= frustumMask & rA;
masksCurrent[1] |= frustumMask & rB;
```

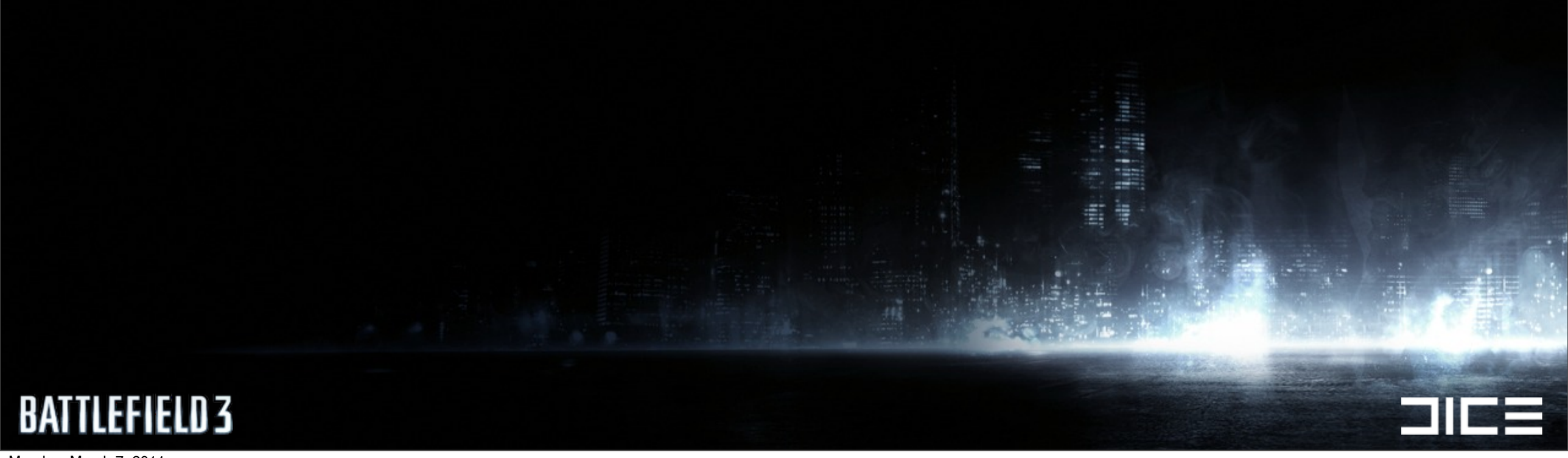
New intersection code (4/4)

```
// Compare against radius

dotA_0123 = vecCmpGTMask(dotA_0123, posA_rrrr);
dotB_0123 = vecCmpGTMask(dotB_0123, posB_rrrr);
dotA45B45 = vecCmpGTMask(dotA45B45, posAB_rrrr);
```

```
Vec dotA45 = vecInsert<VecMask::_0011>(dotA45B45, zero);
Vec dotB45 = vecInsert<VecMask::_0011>(zero, dotA45B45);

// collect the results

Vec resA = vecOrx(dotA_0123);
Vec resB = vecOrx(dotB_0123);

resA = vecOr(resA, vecOrx(dotA45));
resB = vecOr(resB, vecOrx(dotB45));
```

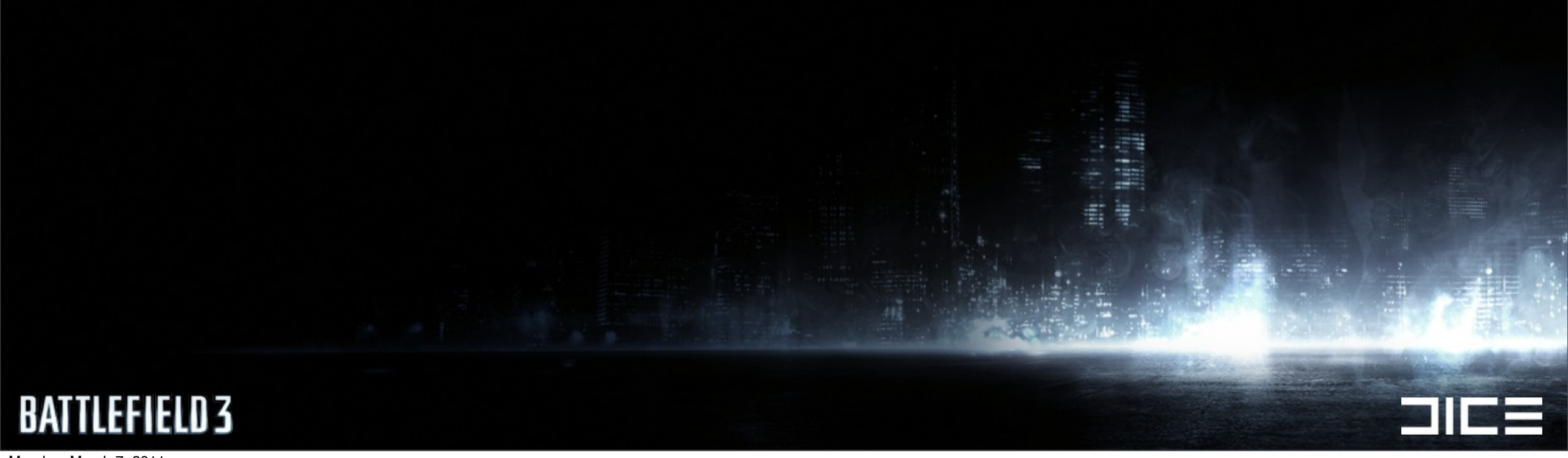
```
// resA = inside or outside of frustum for point A, resB for point B

Vec rA = vecNotMask(resA);
Vec rB = vecNotMask(resB);

masksCurrent[0] |= frustumMask & rA;
masksCurrent[1] |= frustumMask & rB;
```

SPU Pipelining Assembler (SPA)



SPU Pipelining Assembler (SPA)

- Like VCL (for PS2) but for PS3 SPU

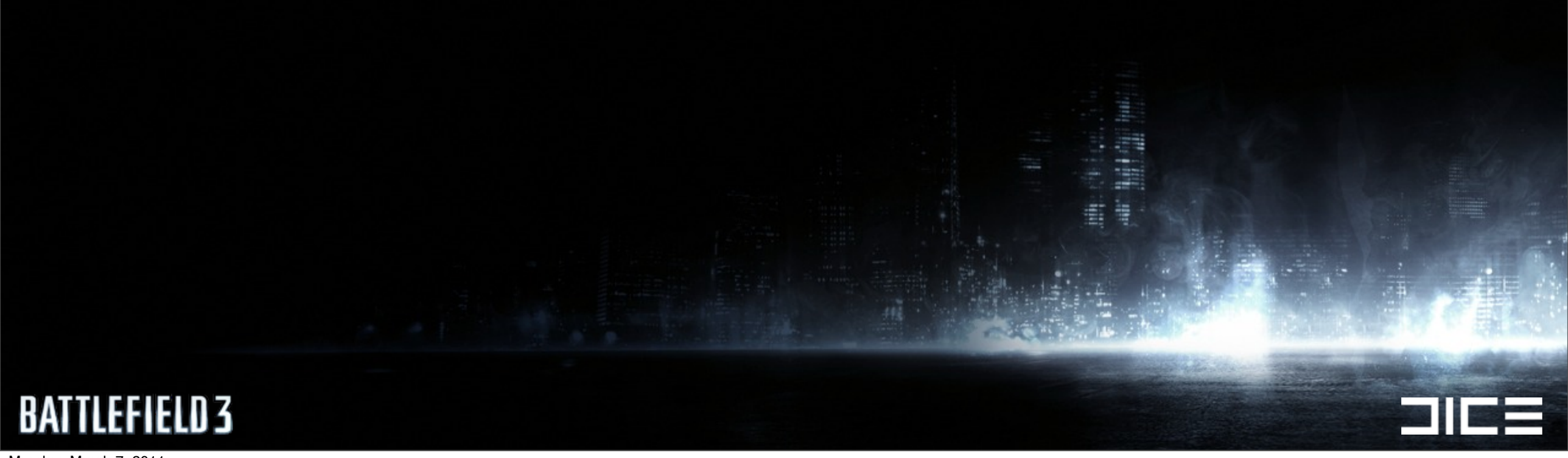


SPU Pipelining Assembler (SPA)

- Like VCL (for PS2) but for PS3 SPU
- Can give you that extra boost if needed

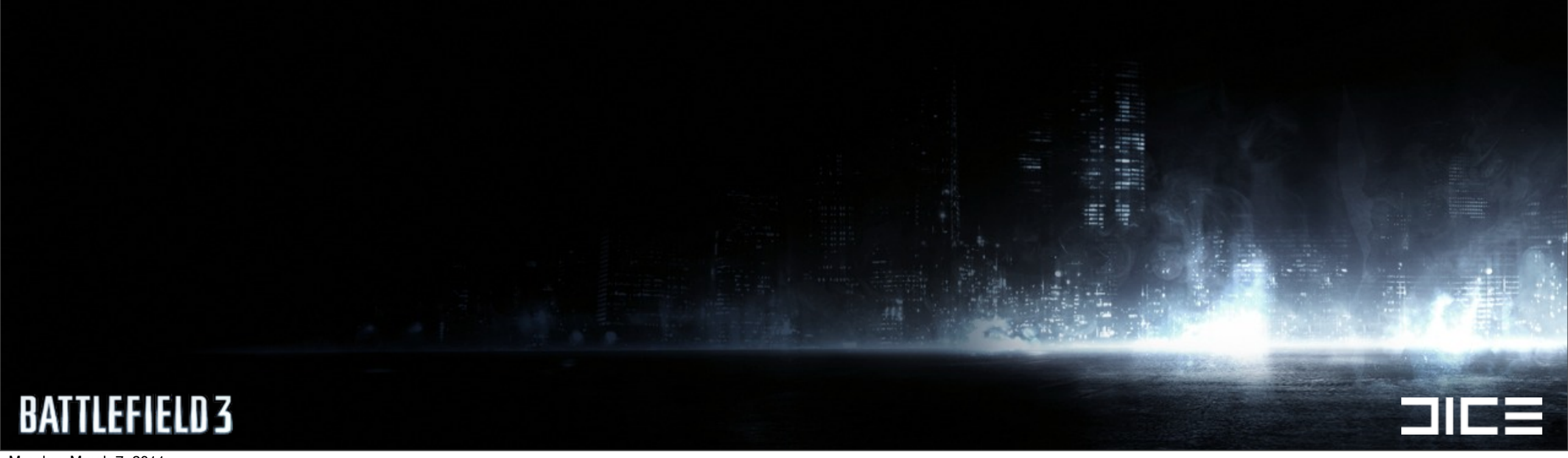
SPU Pipelining Assembler (SPA)

- Like VCL (for PS2) but for PS3 SPU
- Can give you that extra boost if needed
- Does software pipelining for you



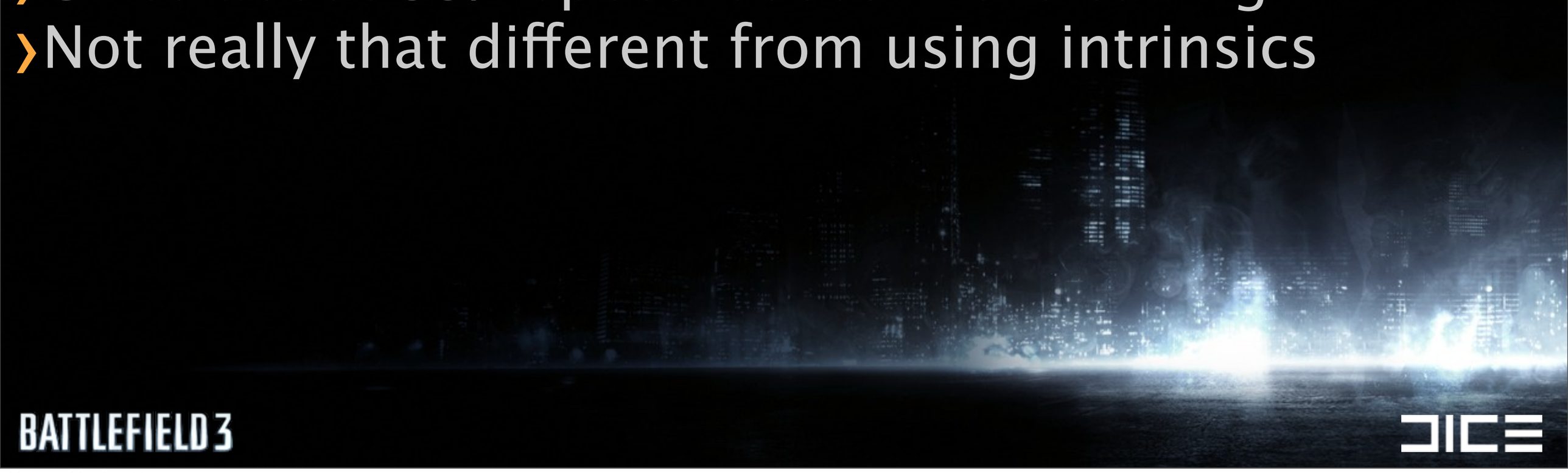
SPU Pipelining Assembler (SPA)

- Like VCL (for PS2) but for PS3 SPU
- Can give you that extra boost if needed
- Does software pipelining for you
- Gives about 35% speed boost in the culling



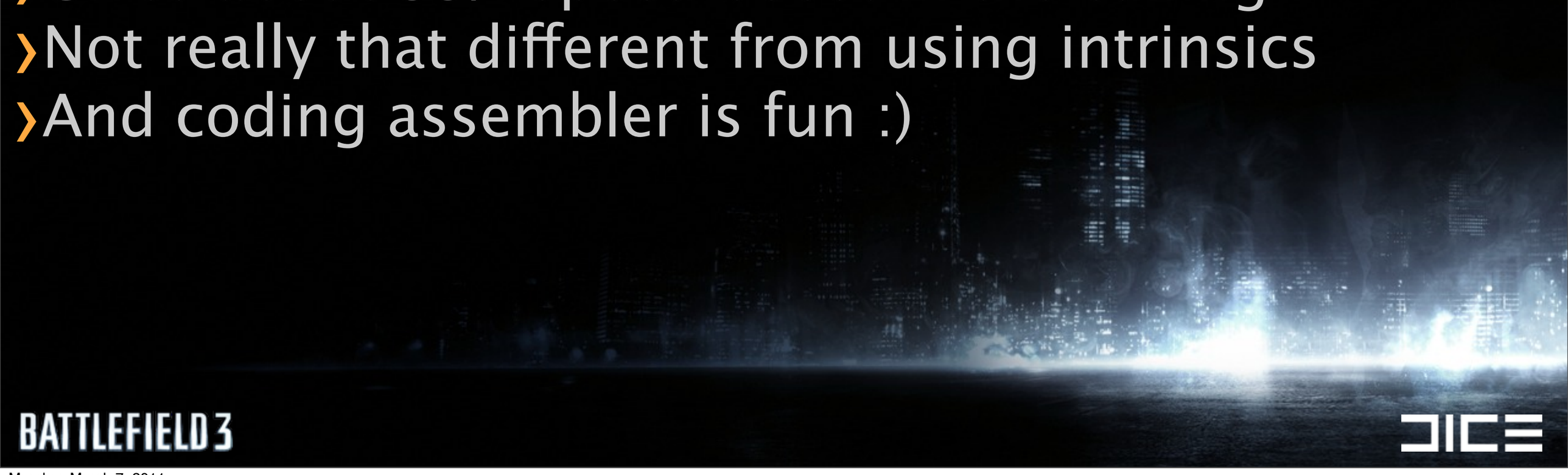
SPU Pipelining Assembler (SPA)

- Like VCL (for PS2) but for PS3 SPU
- Can give you that extra boost if needed
- Does software pipelining for you
- Gives about 35% speed boost in the culling
- Not really that different from using intrinsics



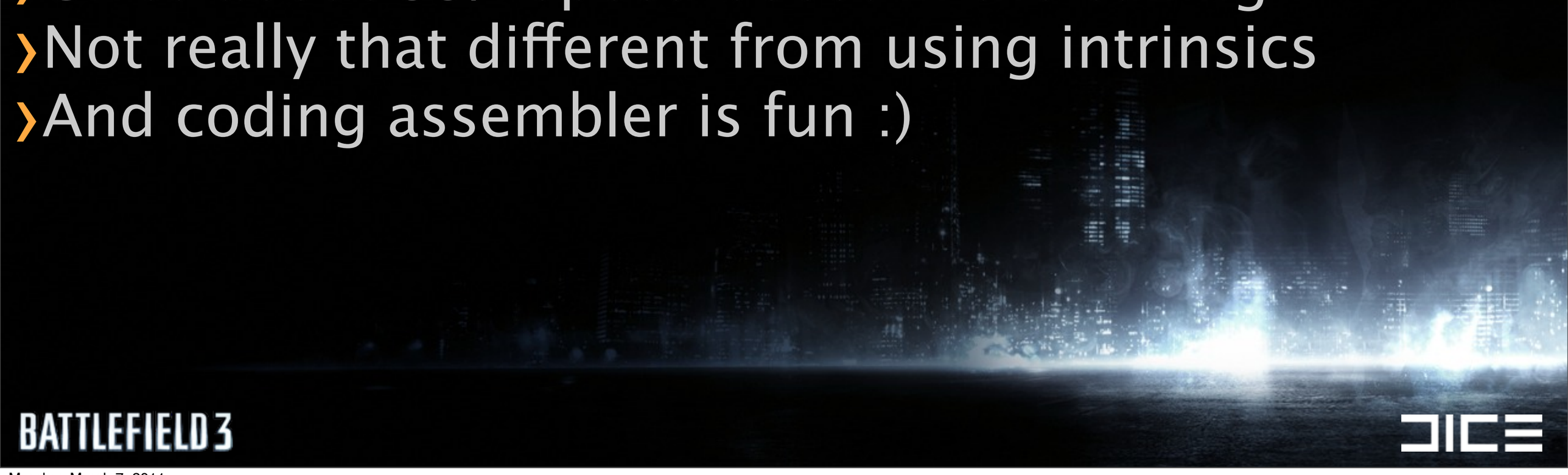
SPU Pipelining Assembler (SPA)

- Like VCL (for PS2) but for PS3 SPU
- Can give you that extra boost if needed
- Does software pipelining for you
- Gives about 35% speed boost in the culling
- Not really that different from using intrinsics
- And coding assembler is fun :)



SPU Pipelining Assembler (SPA)

- Like VCL (for PS2) but for PS3 SPU
- Can give you that extra boost if needed
- Does software pipelining for you
- Gives about 35% speed boost in the culling
- Not really that different from using intrinsics
- And coding assembler is fun :)



SPA Inner loop (partly)

```
lqd    posA, -0x20(currentPos)
lqd    posB, -0x10(currentPos)

shufb  posA_xxxx, posA, posA, Mask_xxxx
shufb  posA_yyyy, posA, posA, Mask_yyyy
shufb  posA_zzzz, posA, posA, Mask_zzzz
shufb  posA_rrrr, posA, posA, Mask_wwww

// 4 dot products

fma    dotA_0123, posA_zzzz, pl_z0z1z2z3, pl_w0w1w2w3
fma    dotA_0123, posA_yyyy, pl_y0y1y2y3, dotA_0123
fma    dotA_0123, posA_xxxx, pl_x0x1x2x3, dotA_0123

shufb  posB_xxxx, posB, posB, Mask_xxxx
shufb  posB_yyyy, posB, posB, Mask_yyyy
shufb  posB_zzzz, posB, posB, Mask_zzzz
shufb  posB_rrrr, posB, posB, Mask_wwww

// 4 dot products

fma    dotB_0123, posB_zzzz, pl_z0z1z2z3, pl_w0w1w2w3
fma    dotB_0123, posB_yyyy, pl_y0y1y2y3, dotB_0123
fma    dotB_0123, posB_xxxx, pl_x0x1x2x3, dotB_0123
```

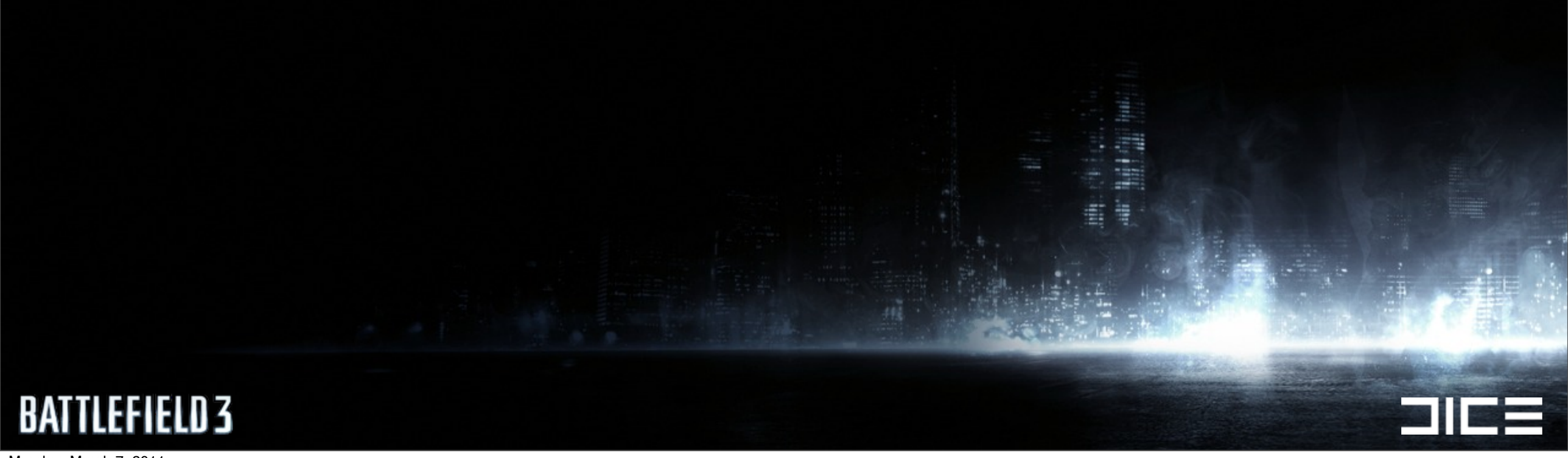
SPA Inner loop

```
# Loop stats - frustumCull::loop
# (ims enabled, sms disabled, optimisation level 2)
#       resmii : 24 (*)          (resource constrained)
#       recmii : 2               (recurrence constrained)
# resource usage:
#       even pipe : 24 inst. (100% use) (*)
#                     FX[15] SP[9]
#       odd pipe  : 24 inst. (100% use) (*)
#                     SH[17] LS[6] BR[1]
# misc:
#       linear schedule = 57 cycles (for information only)
# software pipelining:
#       best pipelined schedule = 24 cycles (pipelined, 3 iterations in parallel)
# software pipelining adjustments:
#       not generating non-pipelined loop since trip count >=3 (3)
# estimated loop performance:
#       =24*n+59 cycles
```

SPA Inner loop

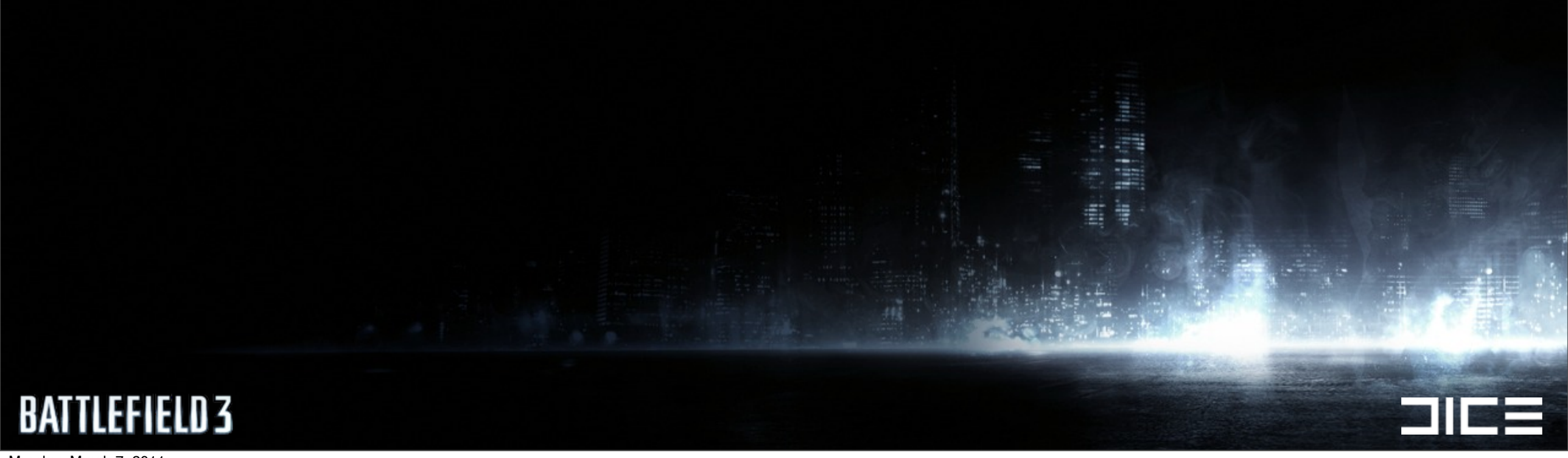
```
_local_c0de000000000002:
    fma         $46,$42,$30,$29;        /* +1 */    shufb       $47,$44,$37,$33         /* +2 */
    fcgt        $57,$20,$24;            /* +2 */    orx         $48,$15                 /* +2 */
    selb        $55,$37,$44,$33;        /* +2 */    shufb       $56,$21,$16,$33         /* +1 */
    fma         $52,$16,$28,$41;        /* +1 */    orx         $49,$57                 /* +2 */
    ai          $4,$4,32;                           orx         $54,$47                 /* +2 */
    fma         $51,$19,$26,$45;        /* +1 */    orx         $53,$55                 /* +2 */
    fma         $50,$56,$27,$46;        /* +1 */    shufb       $24,$23,$23,$34         /* +1 */
    ai          $2,$2,32;               /* +2 */    lqd         $13,-32($4)
    or          $69,$48,$54;            /* +2 */    lqd         $23,-16($4)
    fma         $20,$18,$26,$52;        /* +1 */    lqd         $12,-32($2)             /* +2 */
    nor         $60,$69,$69;            /* +2 */    lqd         $43,-16($2)             /* +2 */
    or          $62,$49,$53;            /* +2 */    shufb       $59,$22,$17,$33         /* +1 */
    and         $39,$60,$35;            /* +2 */    shufb       $11,$14,$24,$33         /* +1 */
    nor         $61,$62,$60;            /* +2 */    shufb       $22,$13,$13,$36
    fcgt        $15,$51,$14;            /* +1 */    shufb       $17,$23,$23,$36
    and         $58,$61,$35;            /* +2 */    shufb       $19,$13,$13,$3
    fma         $10,$59,$25,$50;        /* +1 */    shufb       $18,$23,$23,$3
    fma         $9,$22,$32,$31;                     shufb       $16,$23,$23,$38
    or          $8,$58,$43;             /* +2 */    shufb       $21,$13,$13,$38
    or          $40,$39,$12;            /* +2 */    shufb       $14,$13,$13,$34
    ai          $7,$7,-1;               /* +2 */    shufb       $42,$19,$18,$33
    fma         $41,$17,$32,$31;                    stqd        $8,-16($2)              /* +2 */
    fcgt        $44,$10,$11;            /* +1 */    stqd        $40,-32($2)             /* +2 */
    fma         $45,$21,$28,$9;                     brnz        $7,_local_c0de000000000002 /* +2 */
    nop         ;                                   hbrr
```

Additional culling



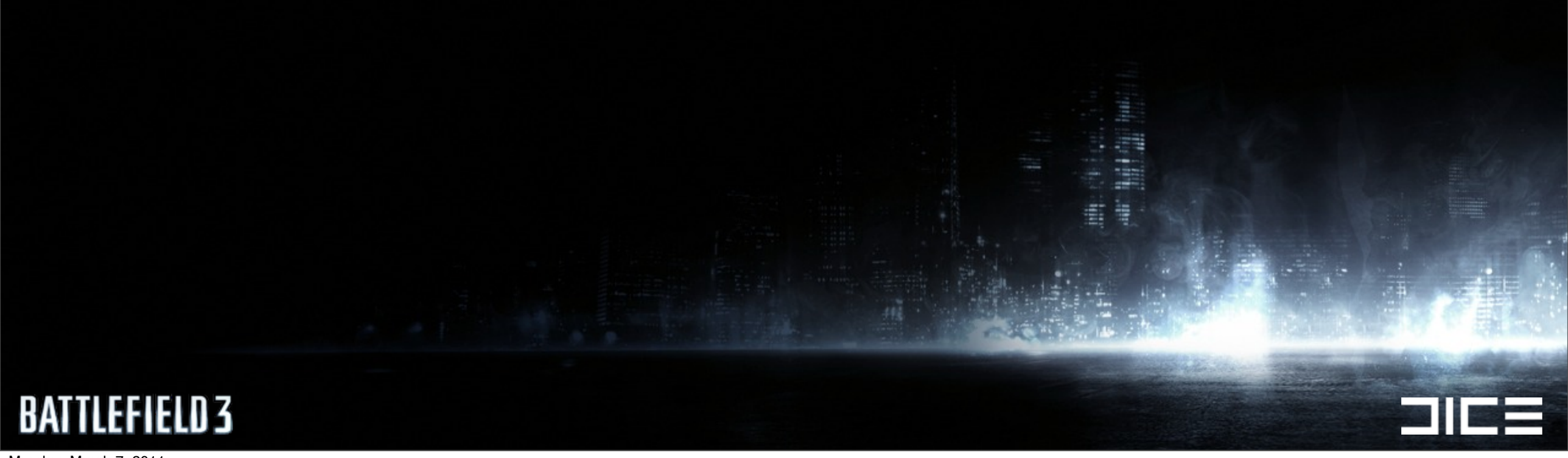
Additional culling

- Frustum vs AABB



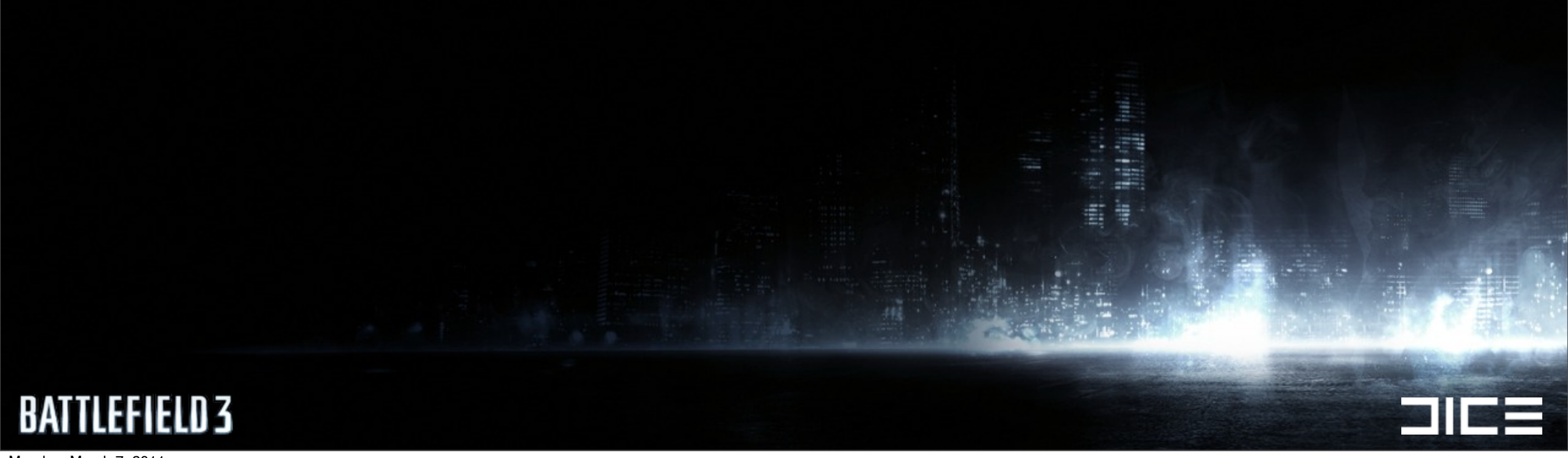
Additional culling

- Frustum vs AABB
- Project AABB to screen space



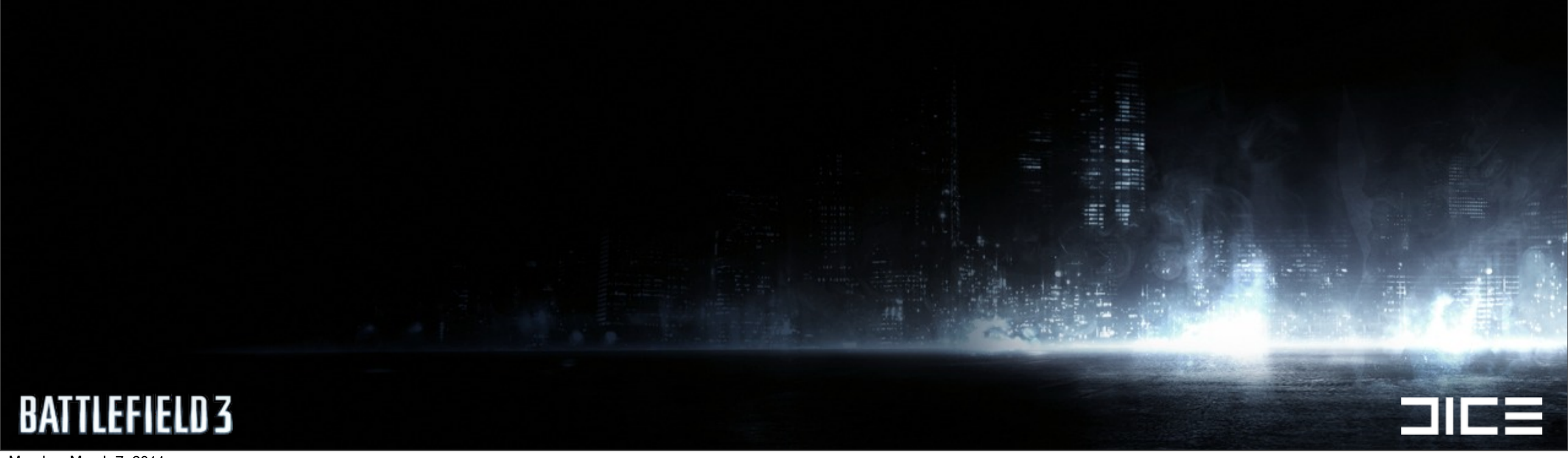
Additional culling

- Frustum vs AABB
- Project AABB to screen space
- Software Occlusion

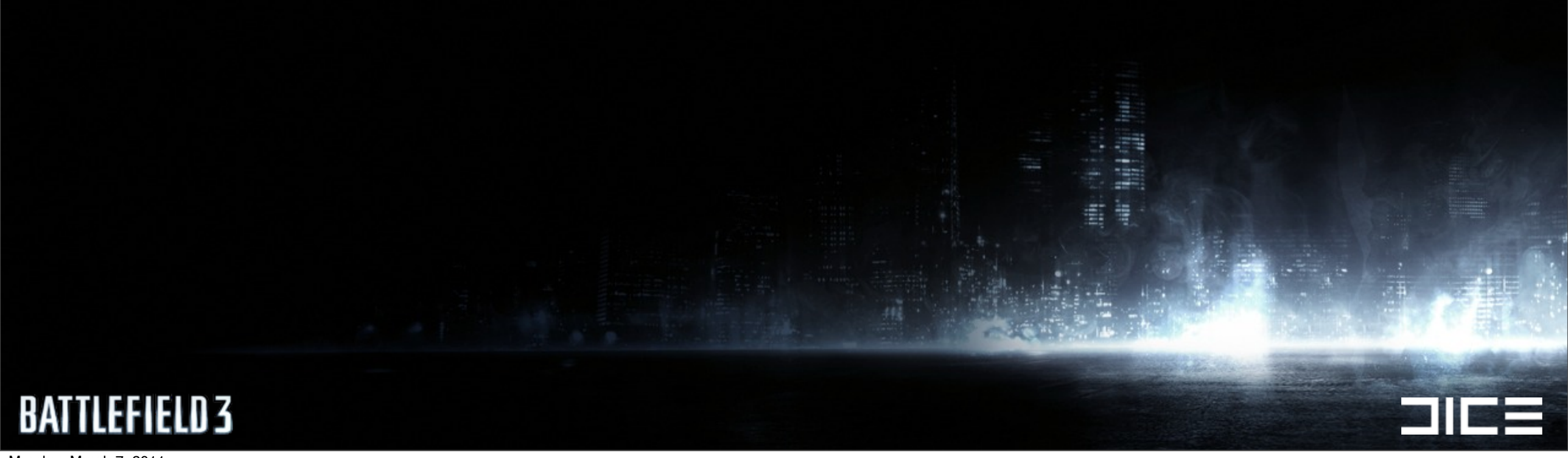


Additional culling

- Frustum vs AABB
- Project AABB to screen space
- Software Occlusion

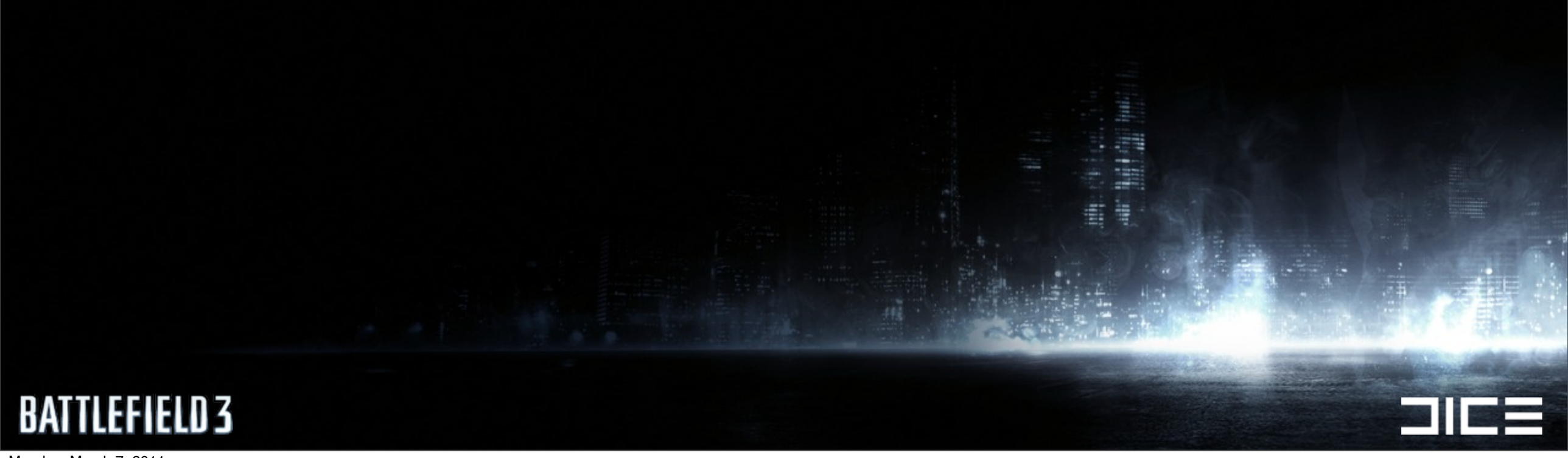


Project AABB to screen space



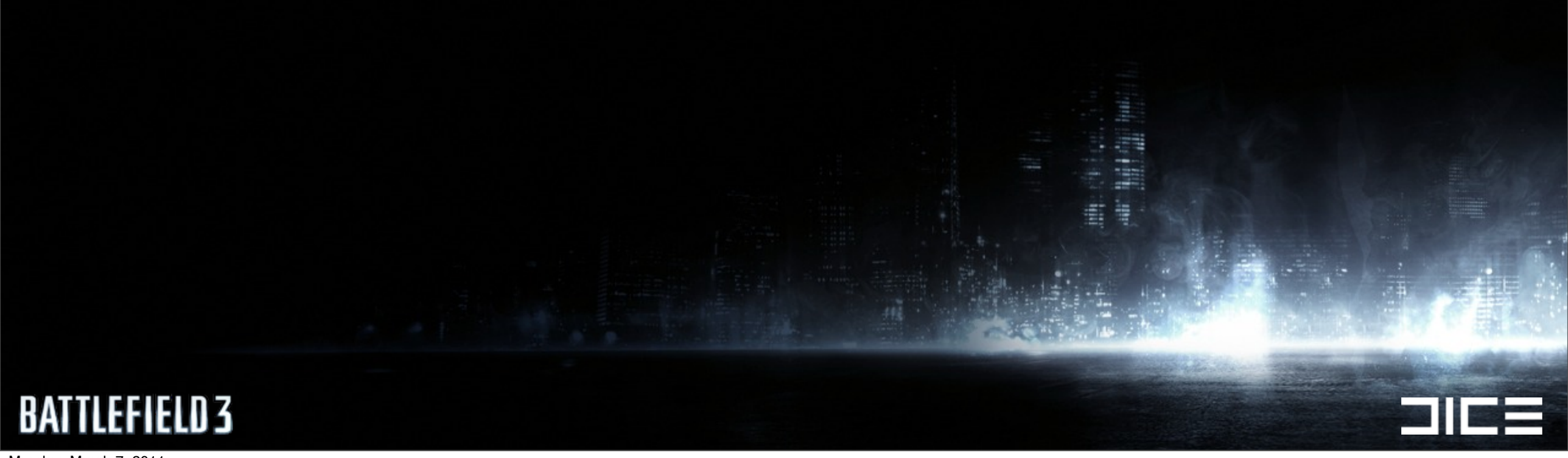
Project AABB to screen space

- Calculate the area of the AABB in screen space



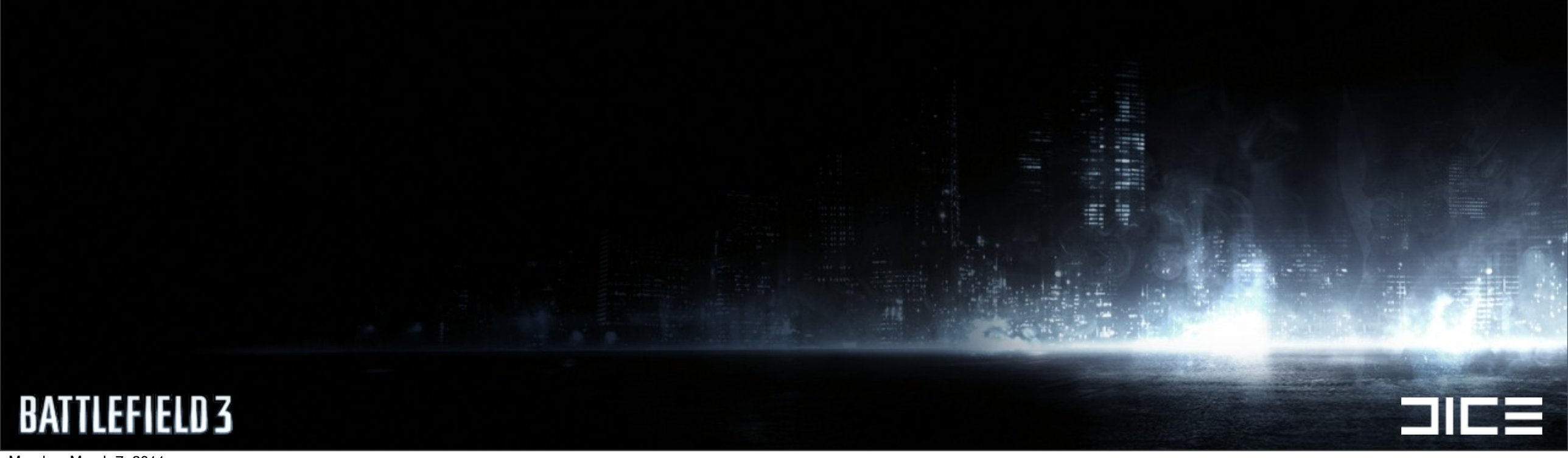
Project AABB to screen space

- Calculate the area of the AABB in screen space
- If area is smaller than setting just skip it



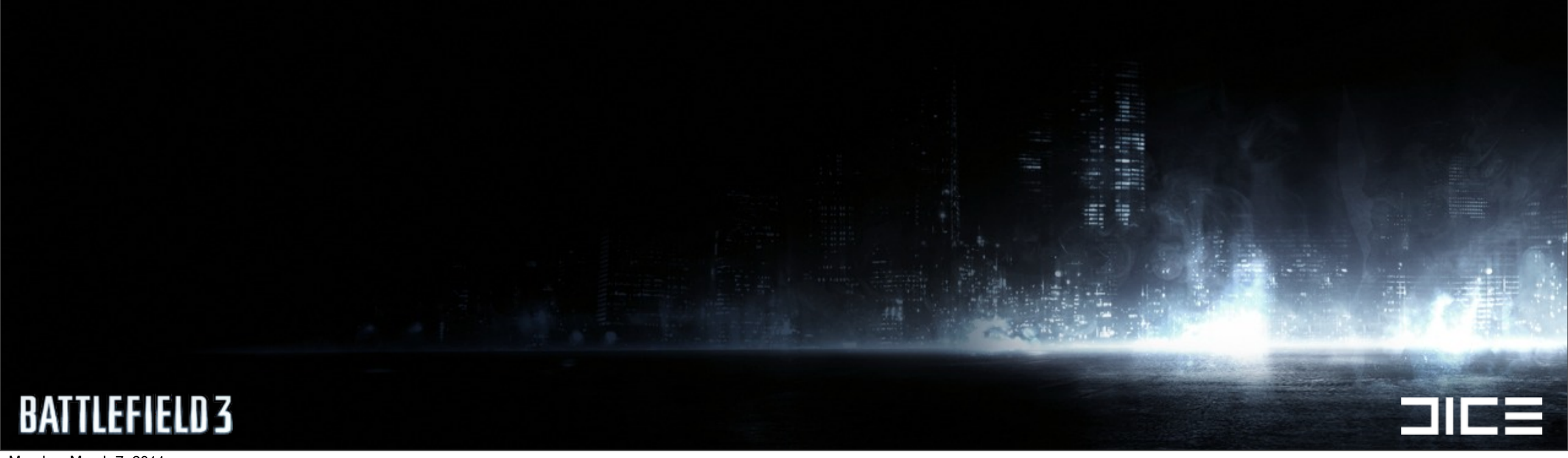
Project AABB to screen space

- Calculate the area of the AABB in screen space
- If area is smaller than setting just skip it
- Due to FOV taking distance doesn't work

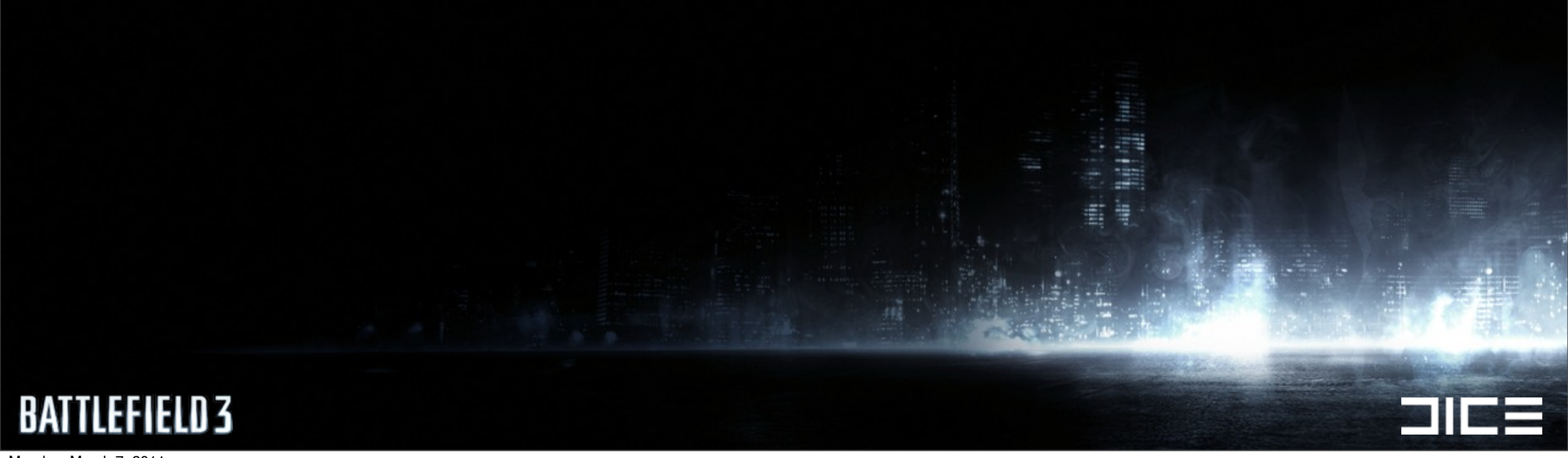


Project AABB to screen space

- Calculate the area of the AABB in screen space
- If area is smaller than setting just skip it
- Due to FOV taking distance doesn't work

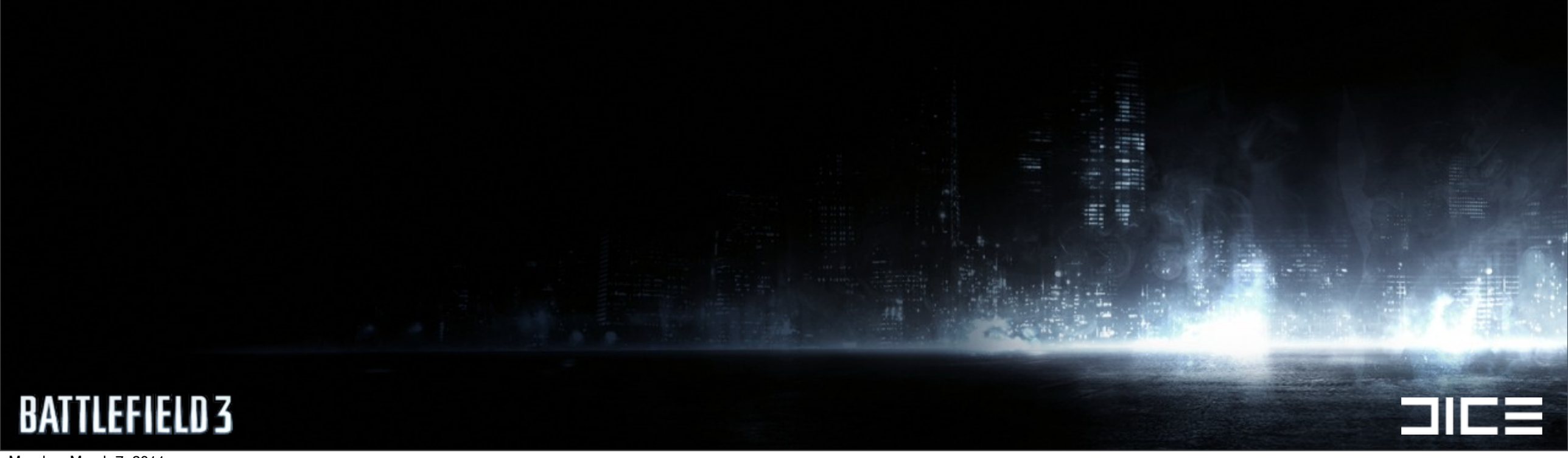


Software Occlusion



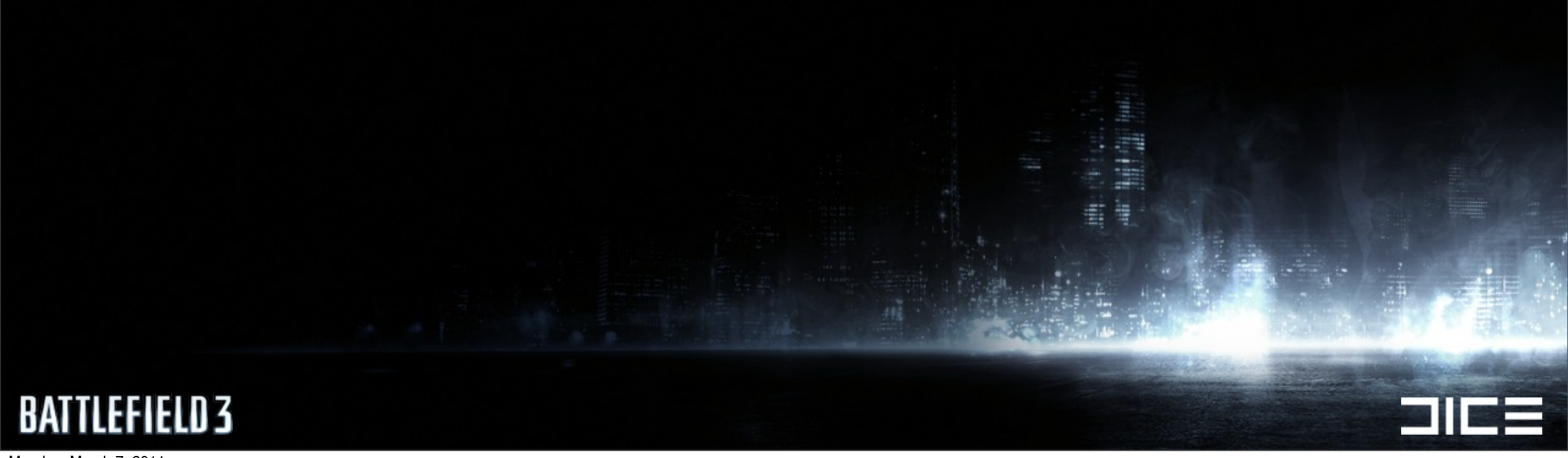
Software Occlusion

- Used in Frostbite for 3 years



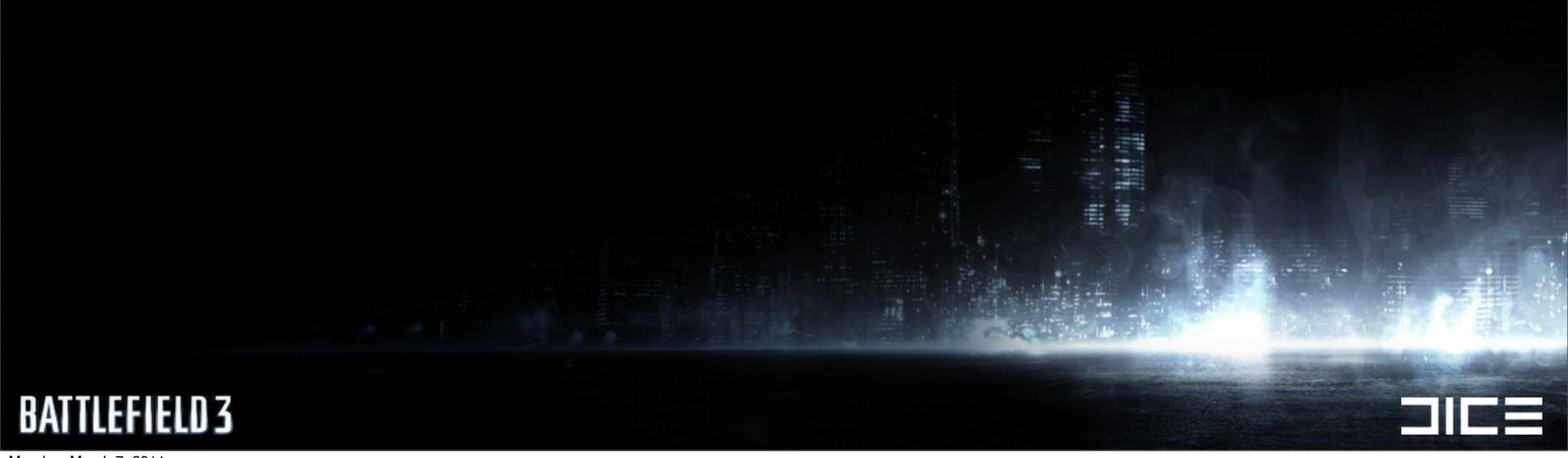
Software Occlusion

- Used in Frostbite for 3 years
- Cross platform



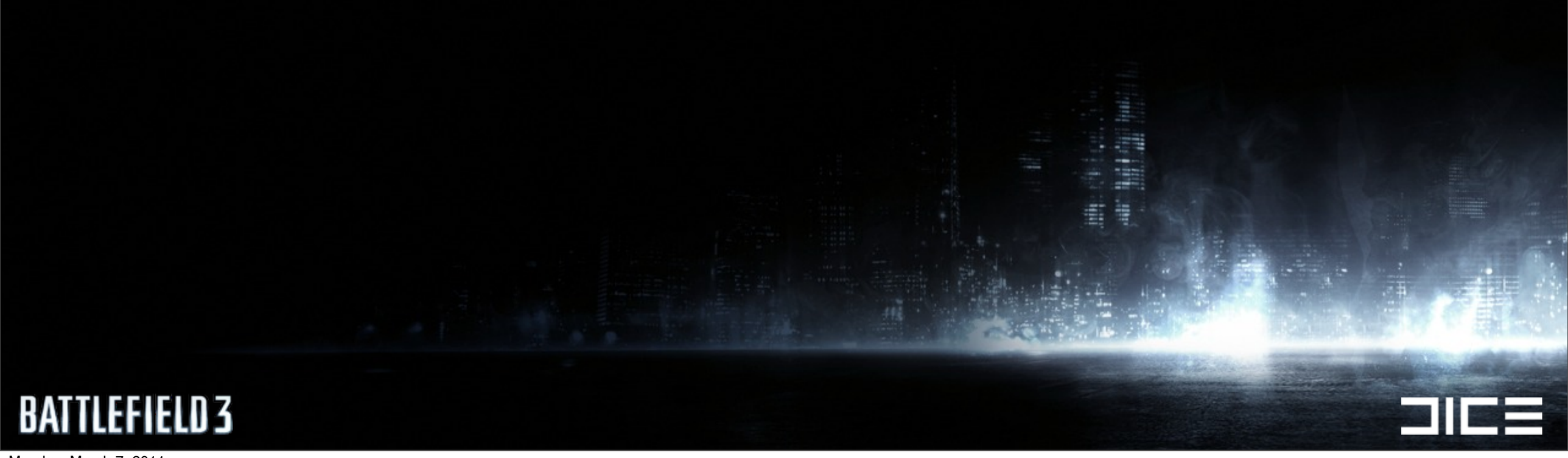
Software Occlusion

- Used in Frostbite for 3 years
- Cross platform
- Artist made occluders



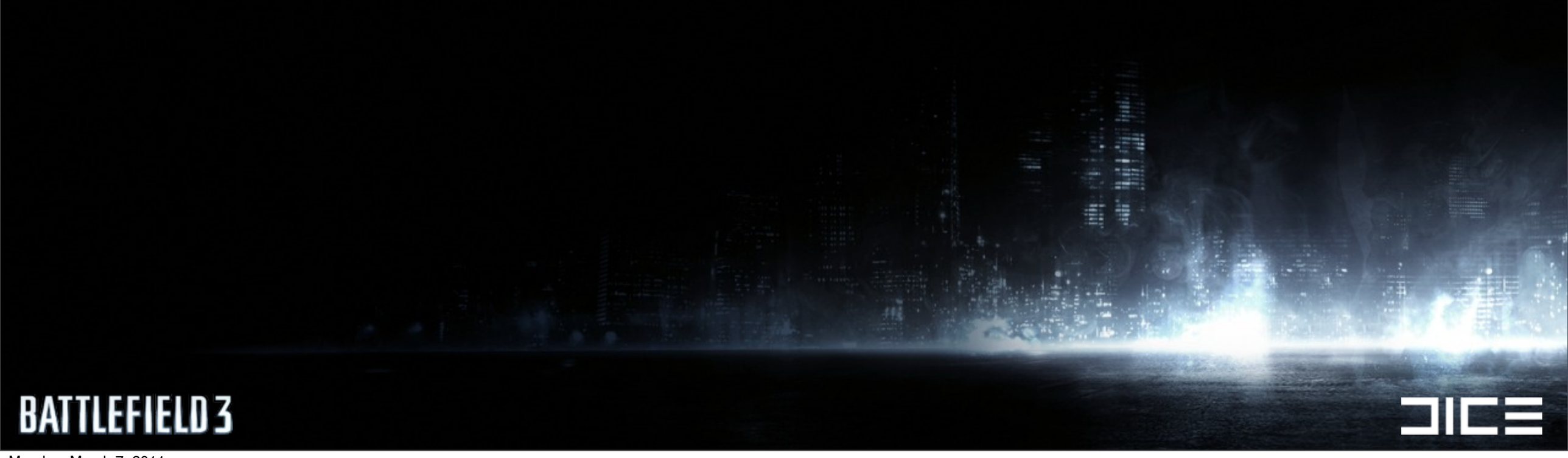
Software Occlusion

- Used in Frostbite for 3 years
- Cross platform
- Artist made occluders
- Terrain

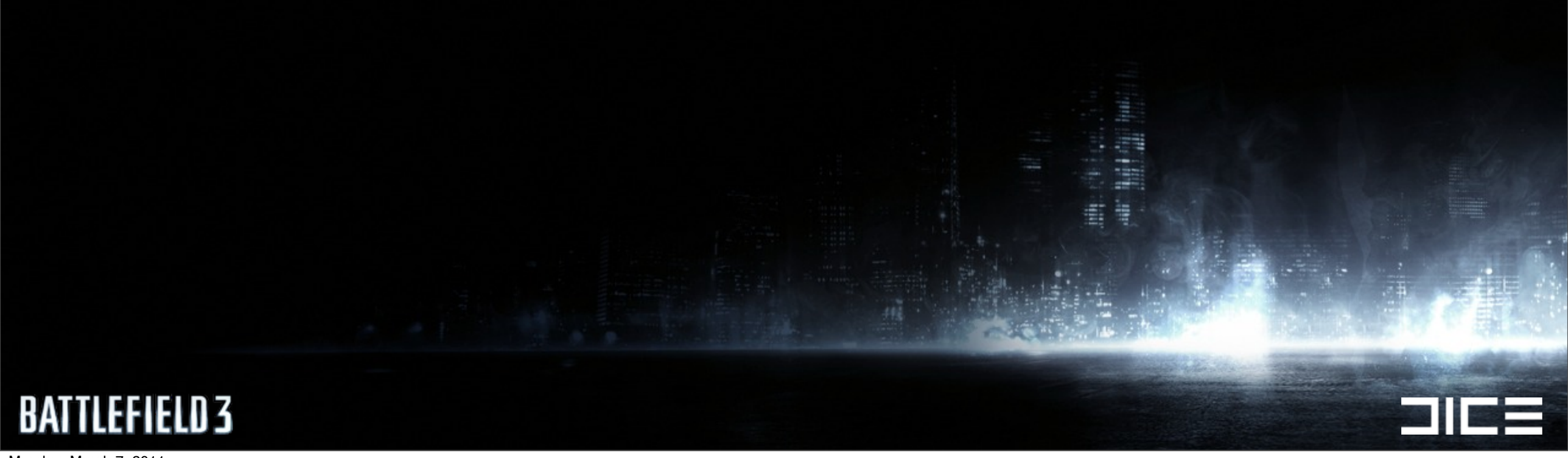


Software Occlusion

- Used in Frostbite for 3 years
- Cross platform
- Artist made occluders
- Terrain

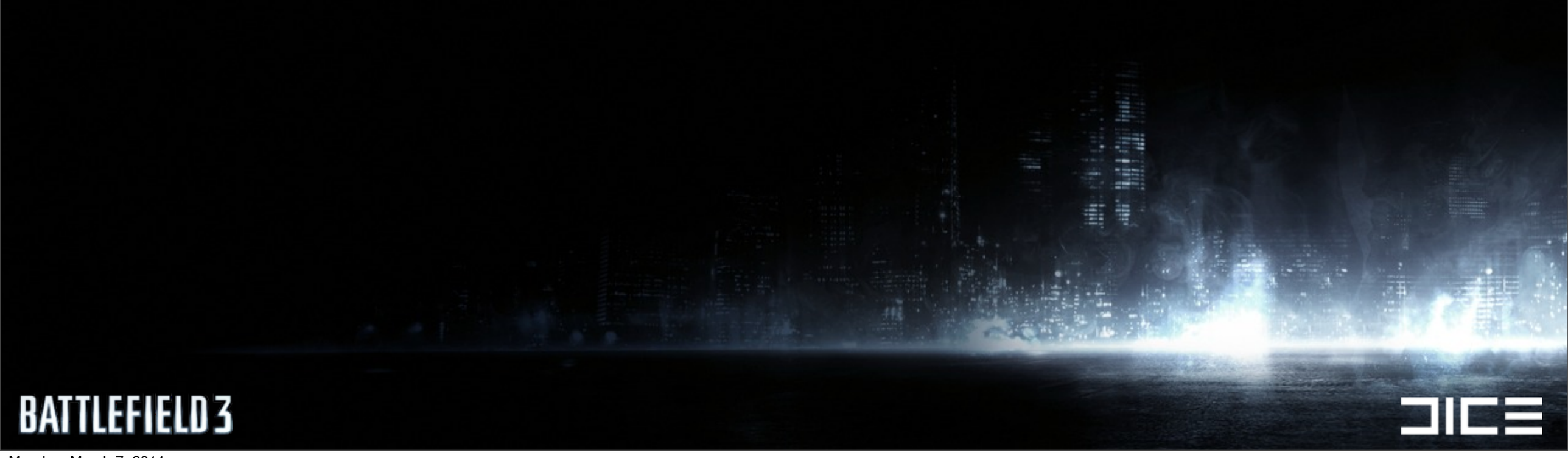


Why Software Occlusion?



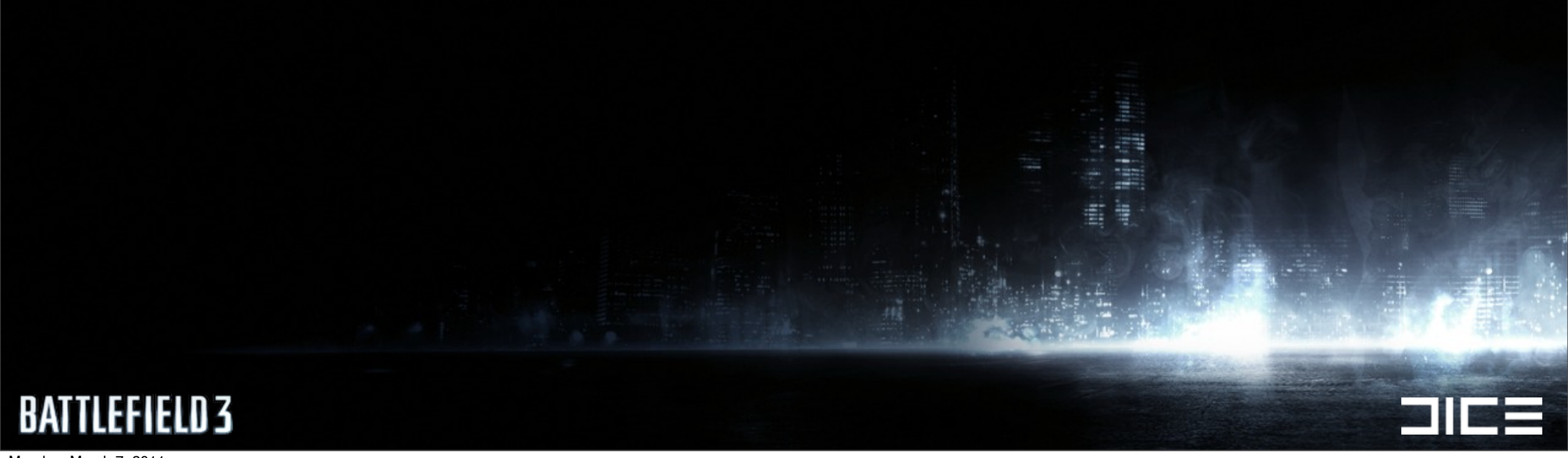
Why Software Occlusion?

- Want to remove CPU time not just GPU



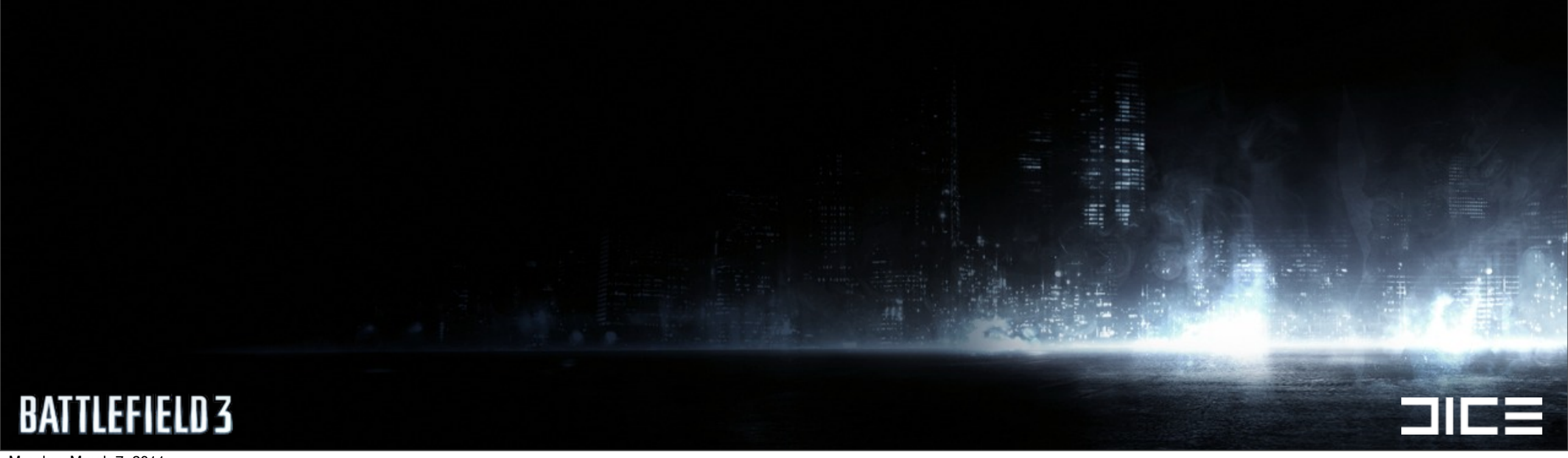
Why Software Occlusion?

- Want to remove CPU time not just GPU
- Cull as early as possible



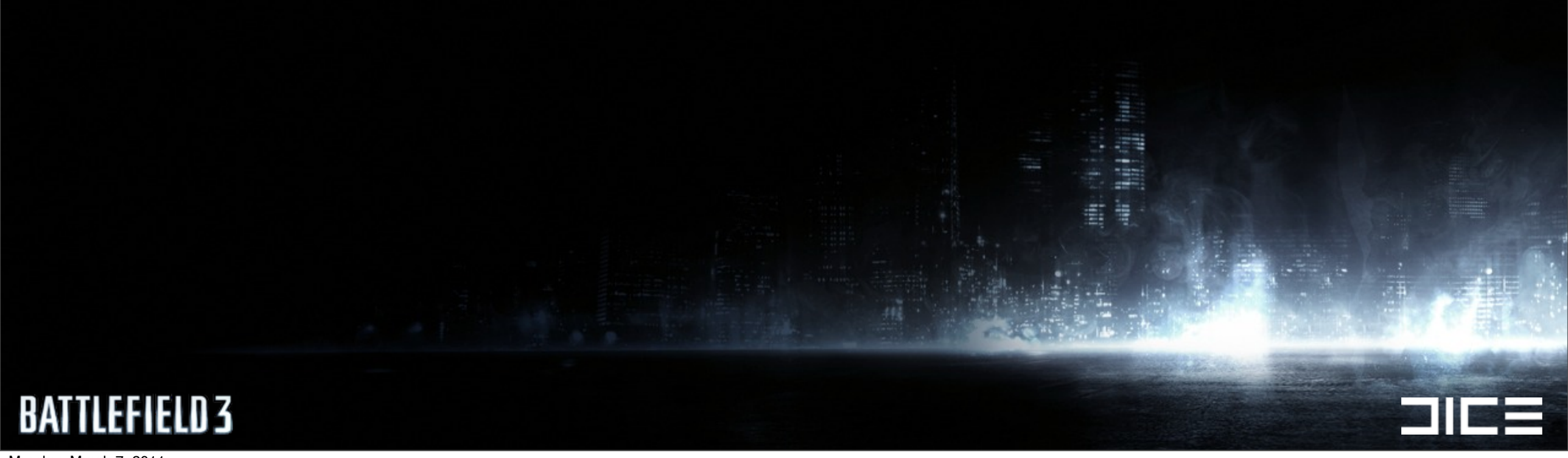
Why Software Occlusion?

- Want to remove CPU time not just GPU
- Cull as early as possible
- GPU queries troublesome as lagging behind CPU



Why Software Occlusion?

- Want to remove CPU time not just GPU
- Cull as early as possible
- GPU queries troublesome as lagging behind CPU
- Must support destruction



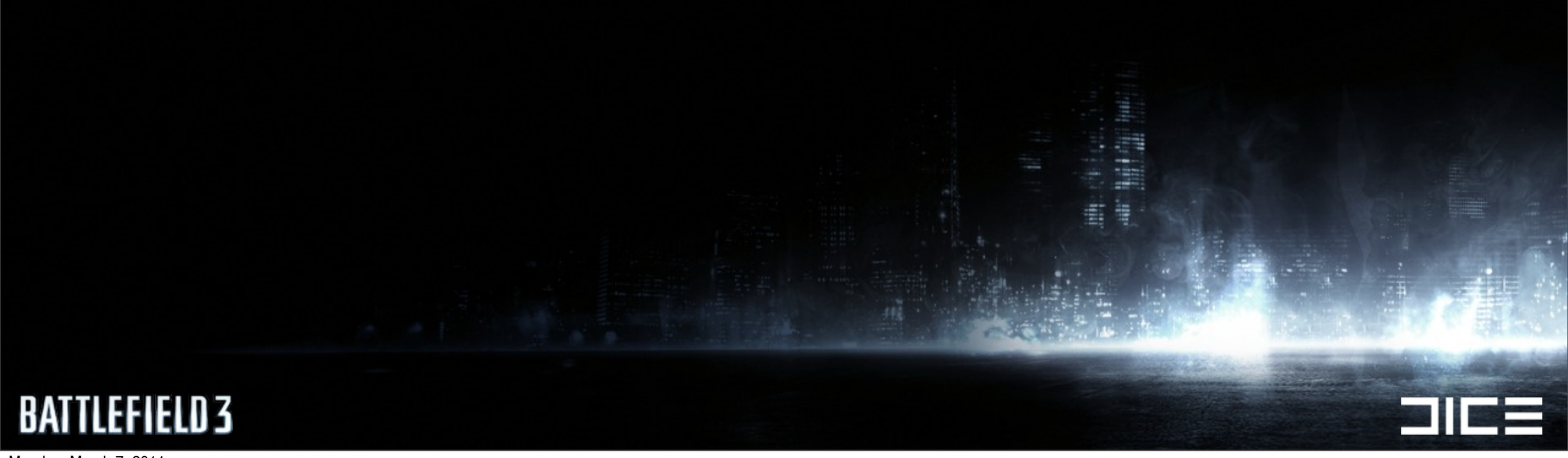
Why Software Occlusion?

- Want to remove CPU time not just GPU
- Cull as early as possible
- GPU queries troublesome as lagging behind CPU
- Must support destruction
- Easy for artists to control

Why Software Occlusion?

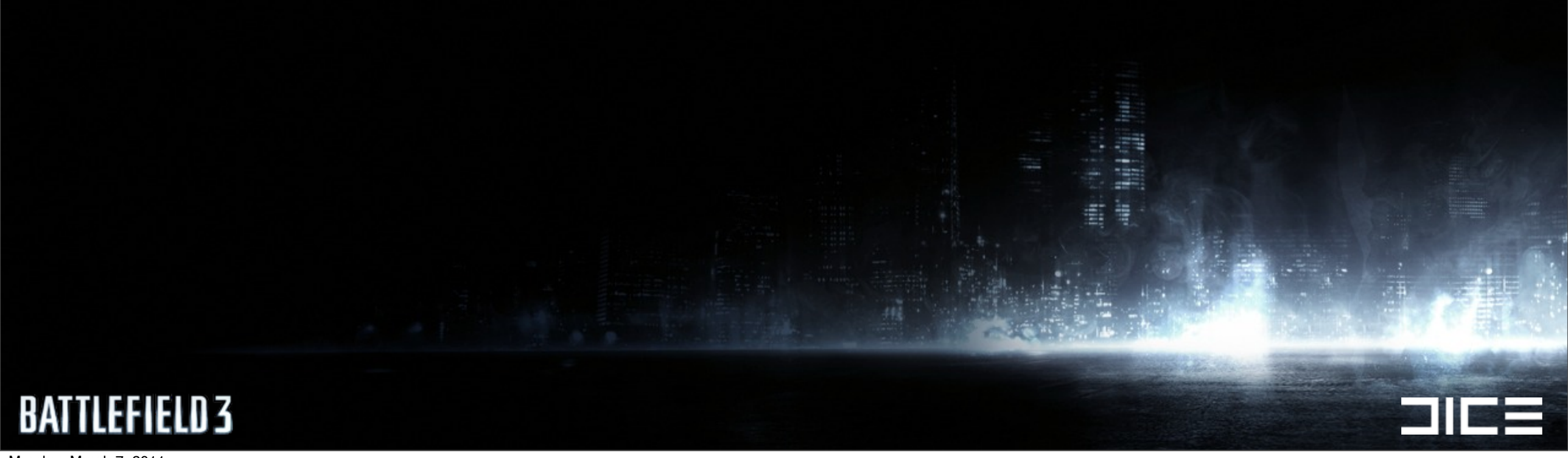
- Want to remove CPU time not just GPU
- Cull as early as possible
- GPU queries troublesome as lagging behind CPU
- Must support destruction
- Easy for artists to control

Software Occlusion



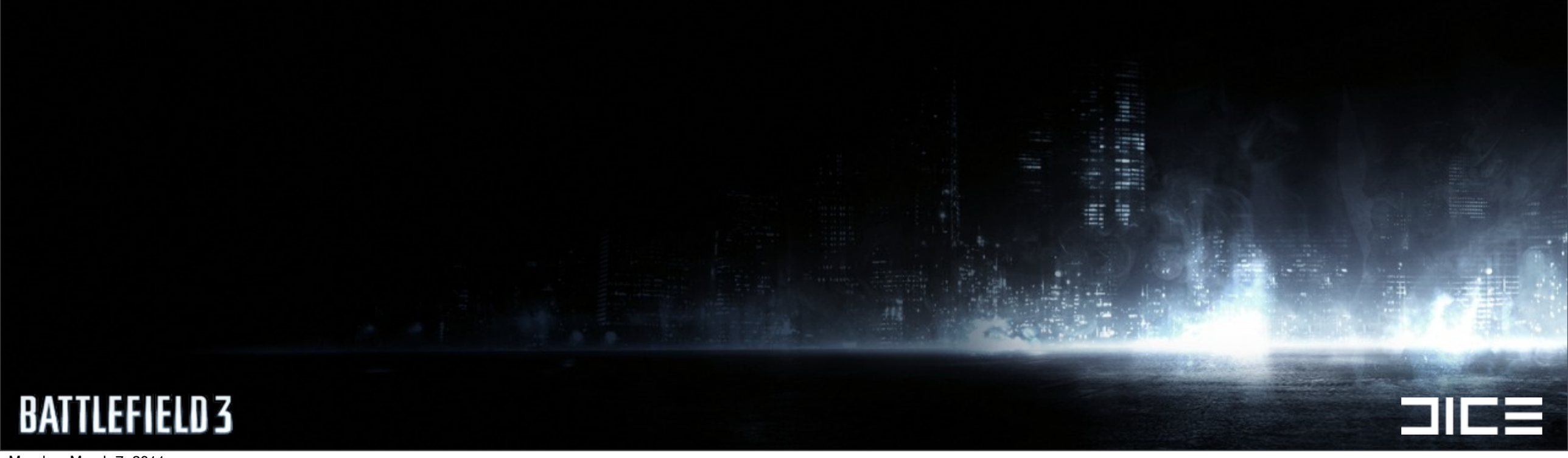
Software Occlusion

› So how does it work?



Software Occlusion

- So how does it work?
- Render PS1 style geometry to a zbuffer using software rendering



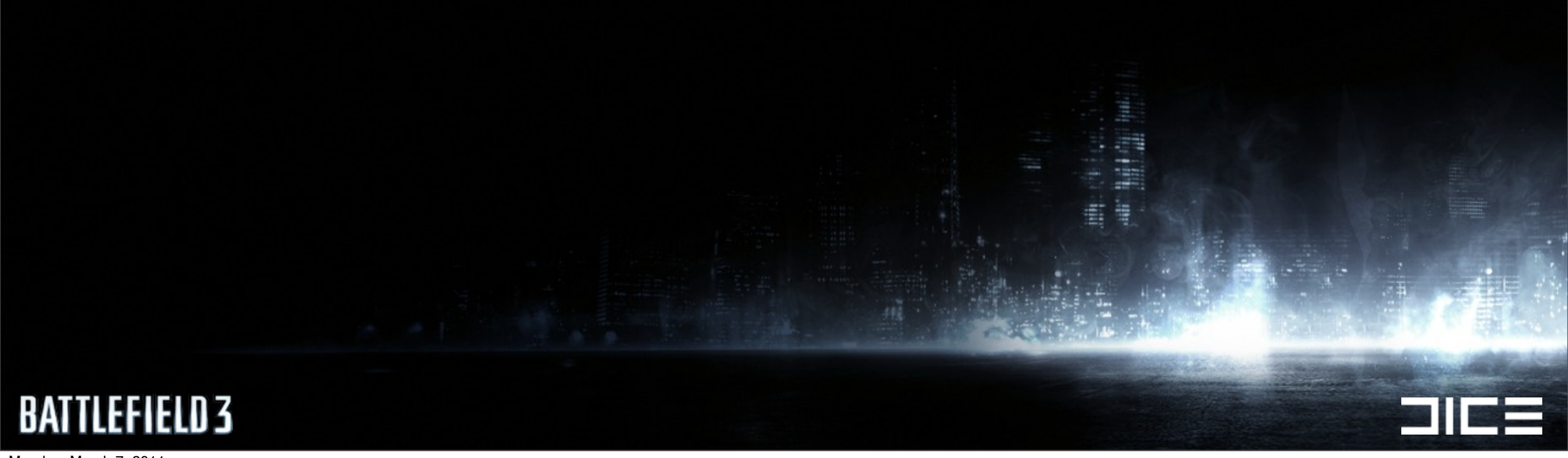
Software Occlusion

- So how does it work?
- Render PS1 style geometry to a zbuffer using software rendering
- The zbuffer is 256 x 114 float

Software Occlusion

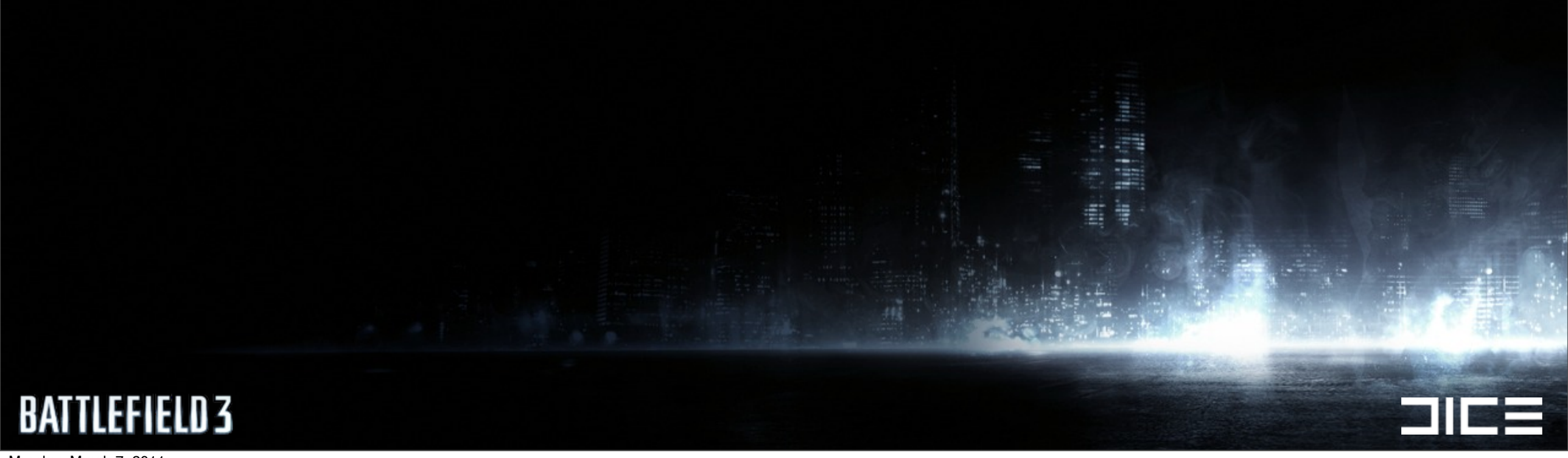
- So how does it work?
- Render PS1 style geometry to a zbuffer using software rendering
- The zbuffer is 256 x 114 float

Software Occlusion



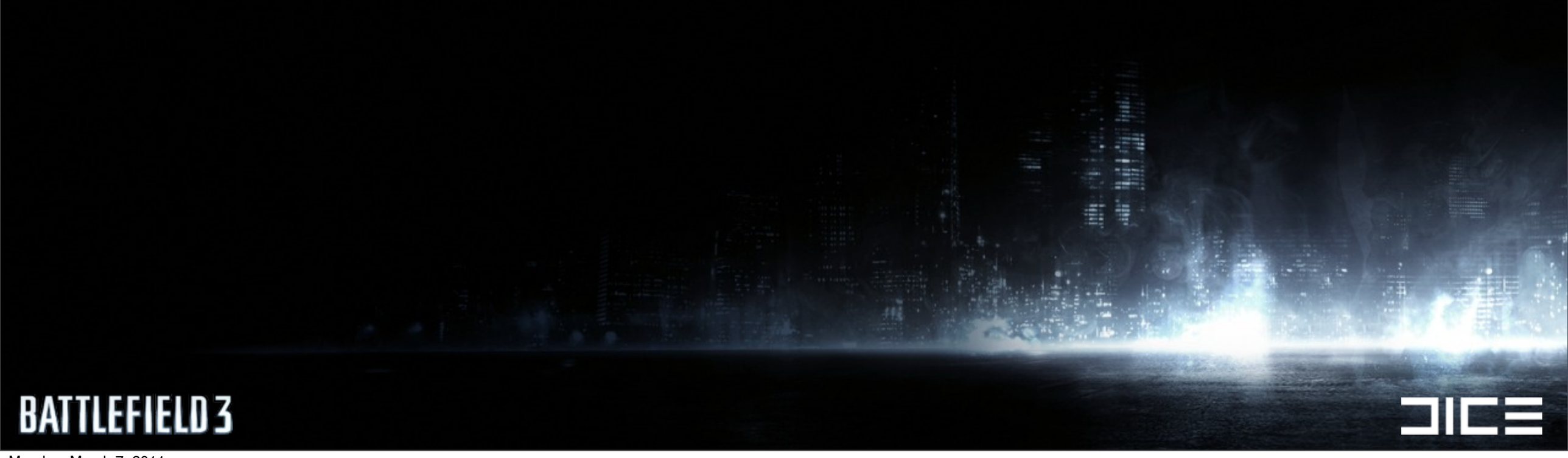
Software Occlusion

- Occluder triangle setup



Software Occlusion

- Occluder triangle setup
- Terrain triangle setup

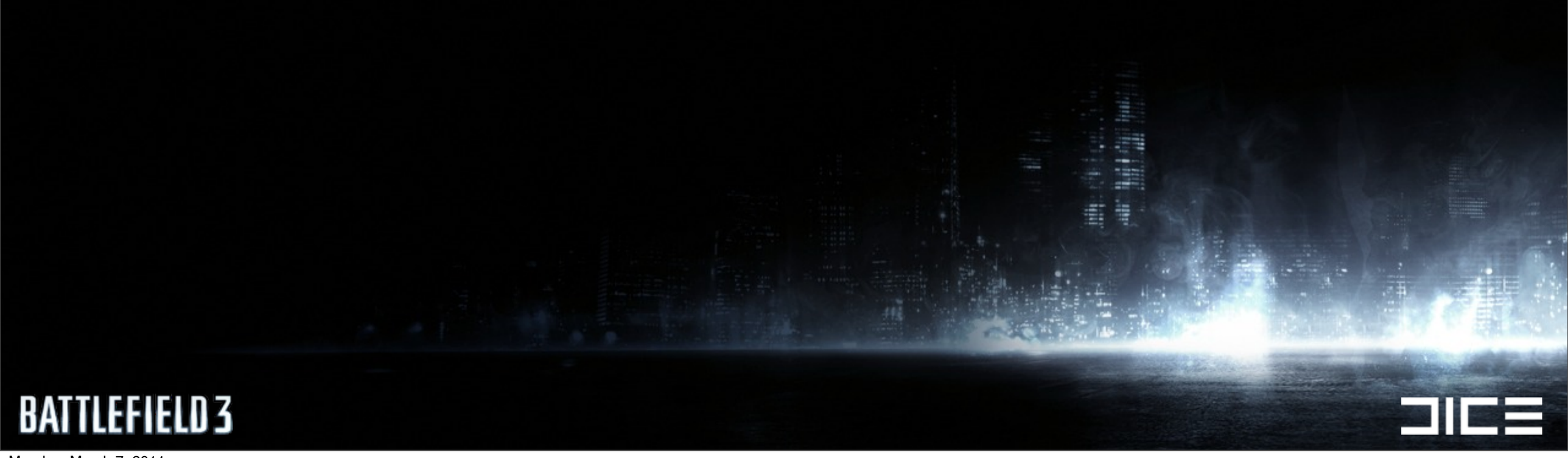


Software Occlusion

- Occluder triangle setup
- Terrain triangle setup
- Rasterize triangles

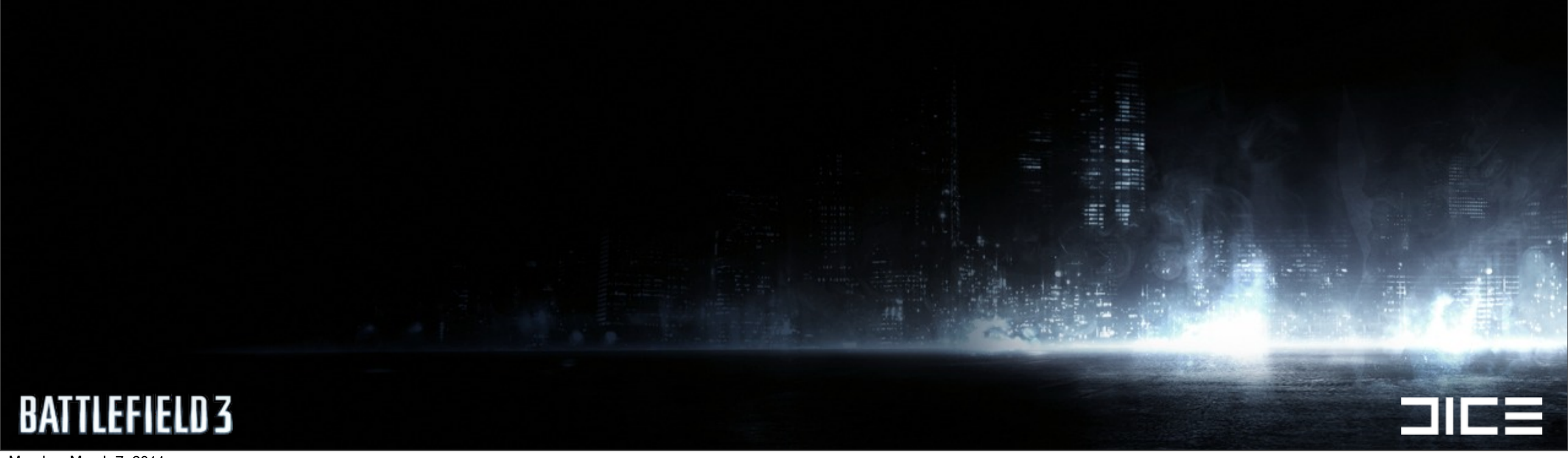
Software Occlusion

- Occluder triangle setup
- Terrain triangle setup
- Rasterize triangles
- Culling

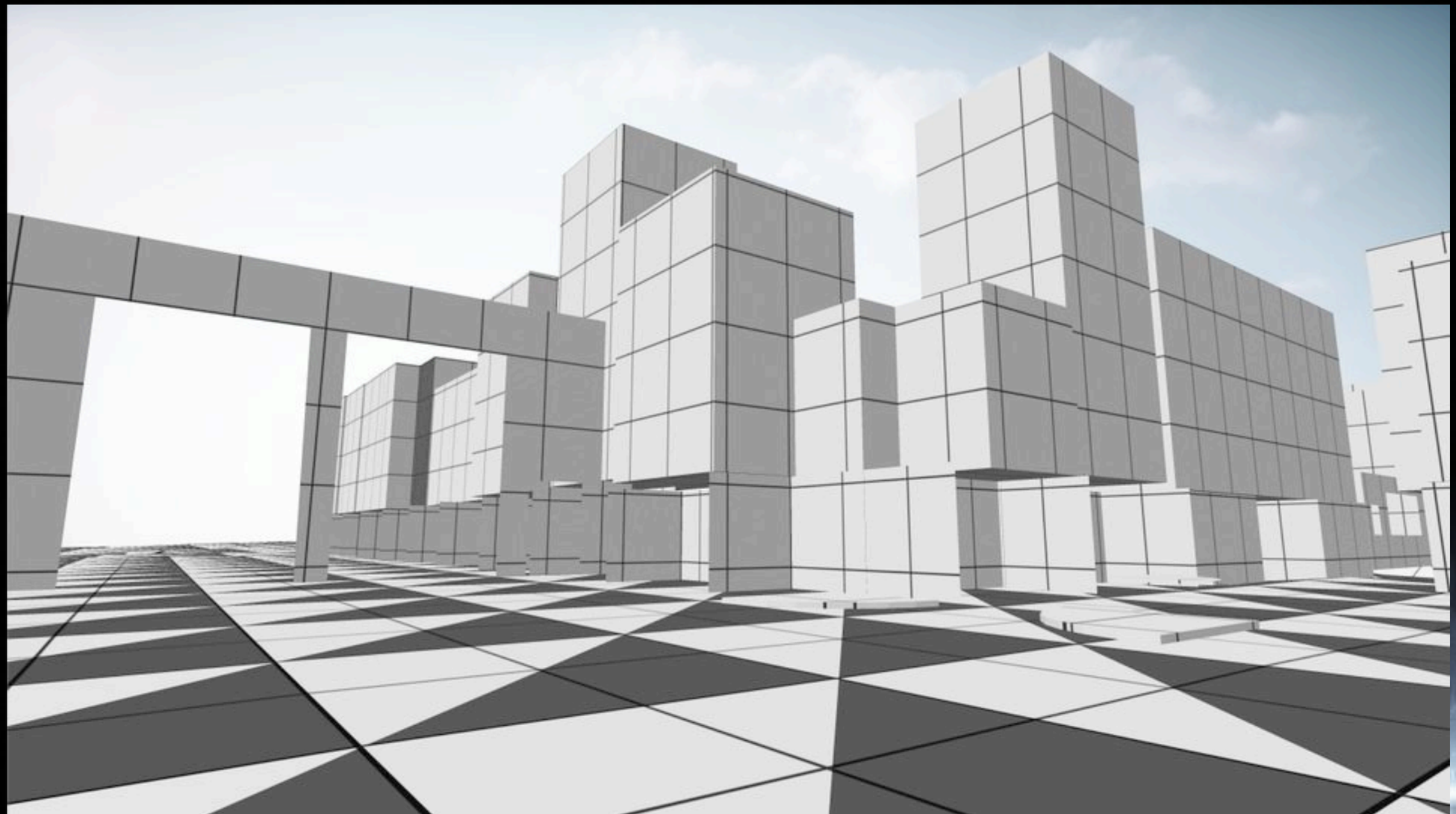


Software Occlusion

- Occluder triangle setup
- Terrain triangle setup
- Rasterize triangles
- Culling



Software Occlusion (Occluders)



BATTLEFIELD 3

DICE

Software Occlusion (In-game)

BATTLEFIELD 3

DICE

Software Occlusion (In-game)

BATTLEFIELD 3

DICE

Software Occlusion (In-game)

BATTLEFIELD 3

DICE

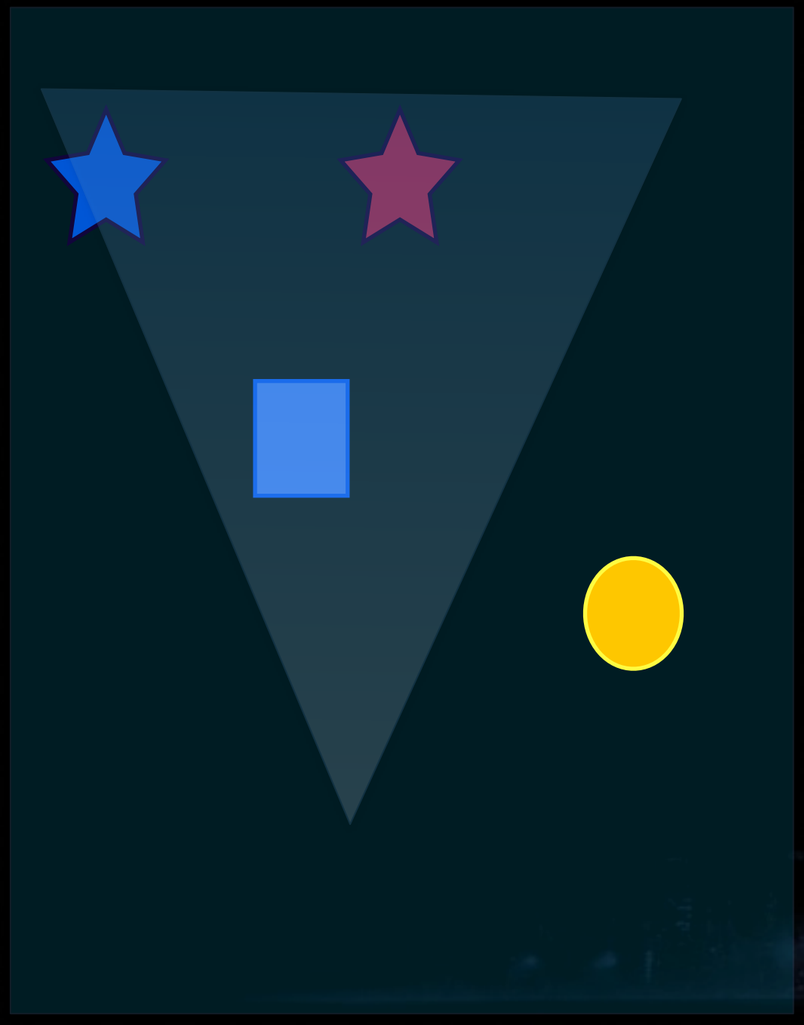
Culling jobs

Culling jobs

Culling jobs

Culling jobs

Occluder triangles



Job 0

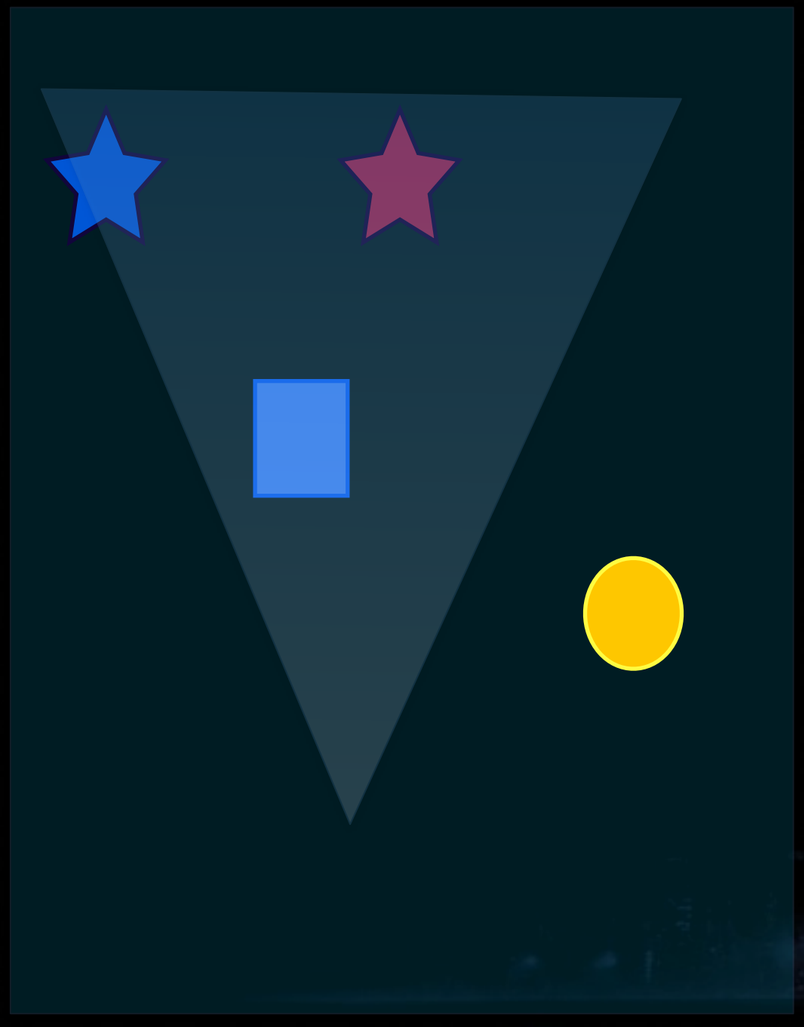
Job 1

Job 2

Job 3

Output

Occluder triangles



Job 0

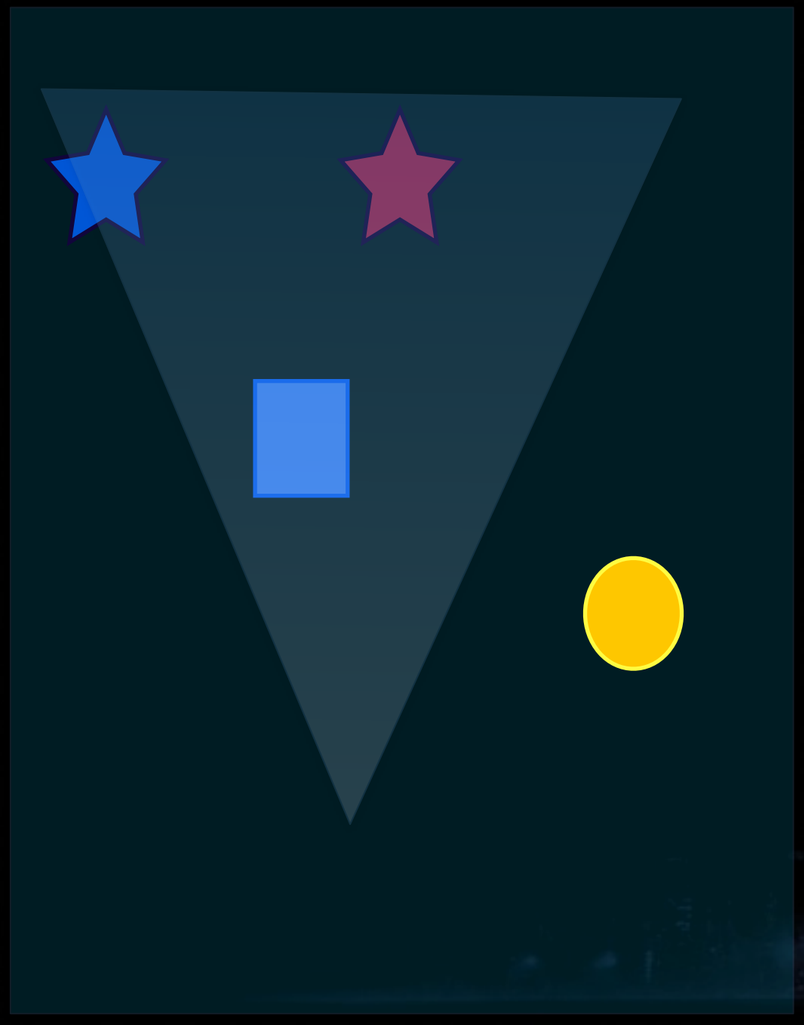
Job 1

Job 2

Job 3

Output

Occluder triangles



Job 0

Inside, Project triangles

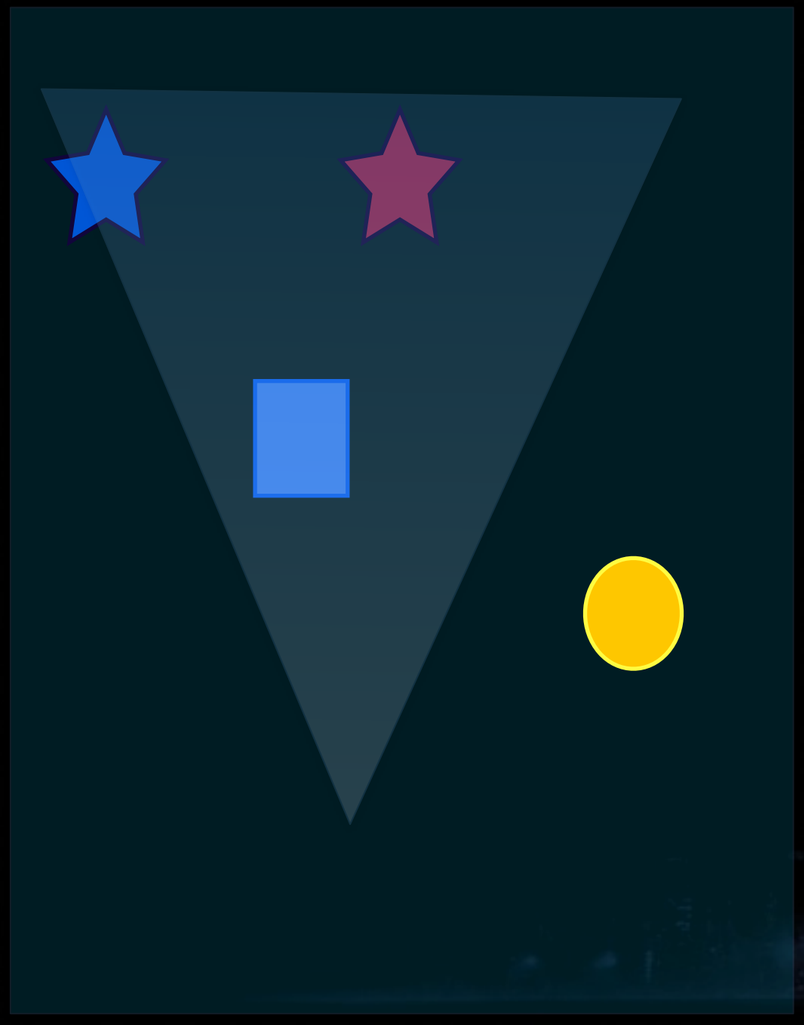
Job 1

Job 2

Job 3

Output

Occluder triangles



Job 0

Inside, Project triangles

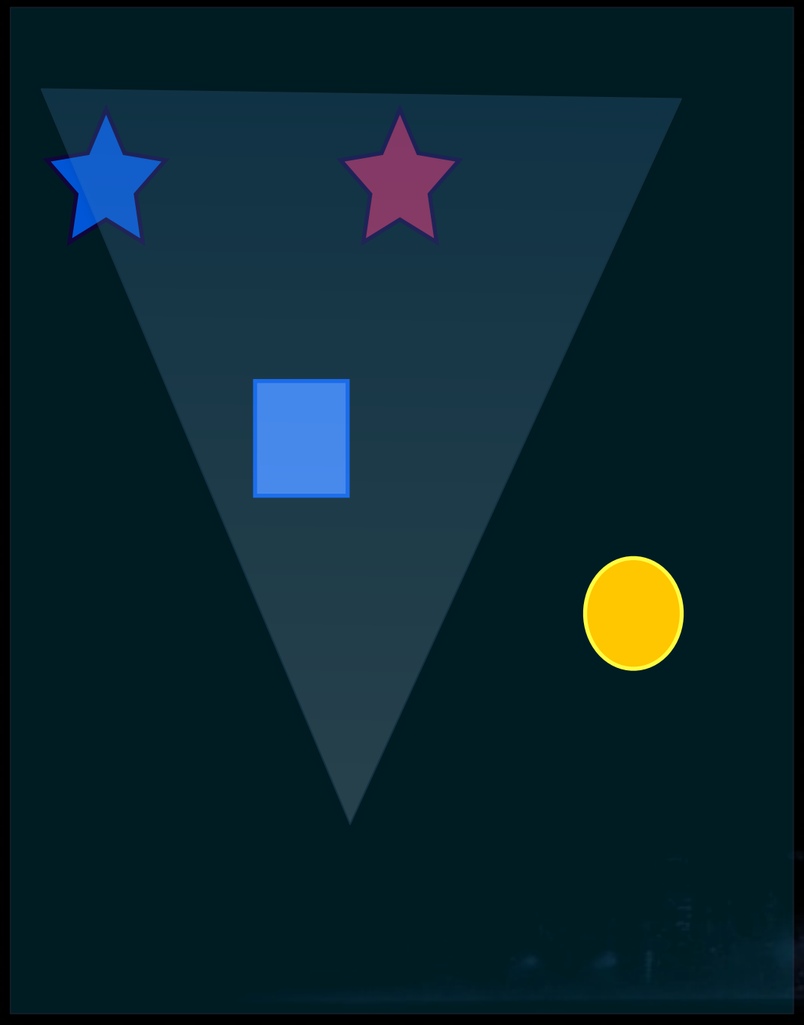
Job 1

Job 2

Job 3

Output

Occluder triangles



Job 0

Inside, Project triangles

Job 1

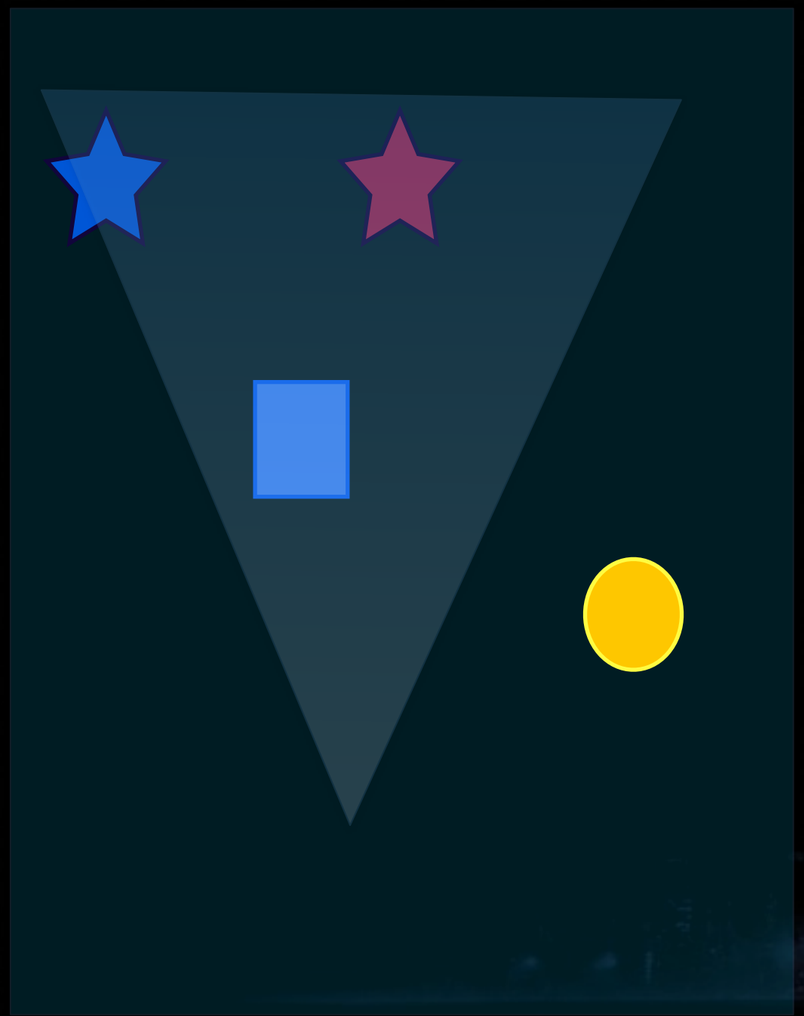
Inside, Project triangles

Job 2

Job 3

Output

Occluder triangles



Job 0

Inside, Project triangles

Job 1

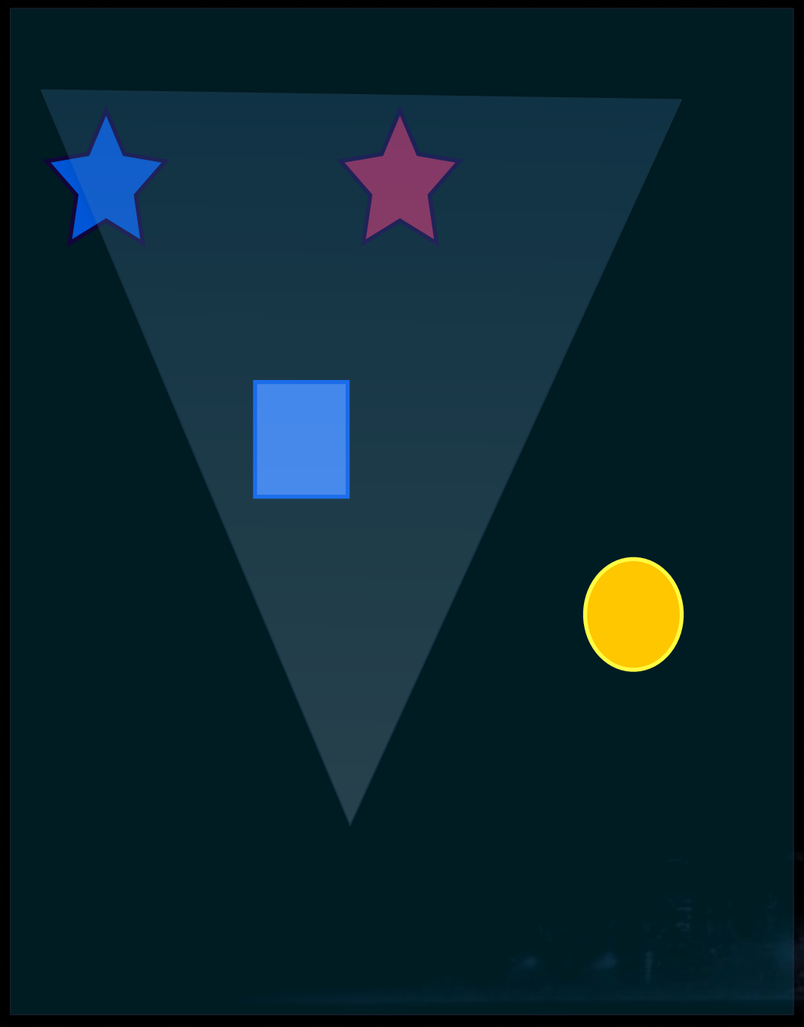
Inside, Project triangles

Job 2

Job 3

Output

Occluder triangles



Job 0

Inside, Project triangles

Job 1

Inside, Project triangles

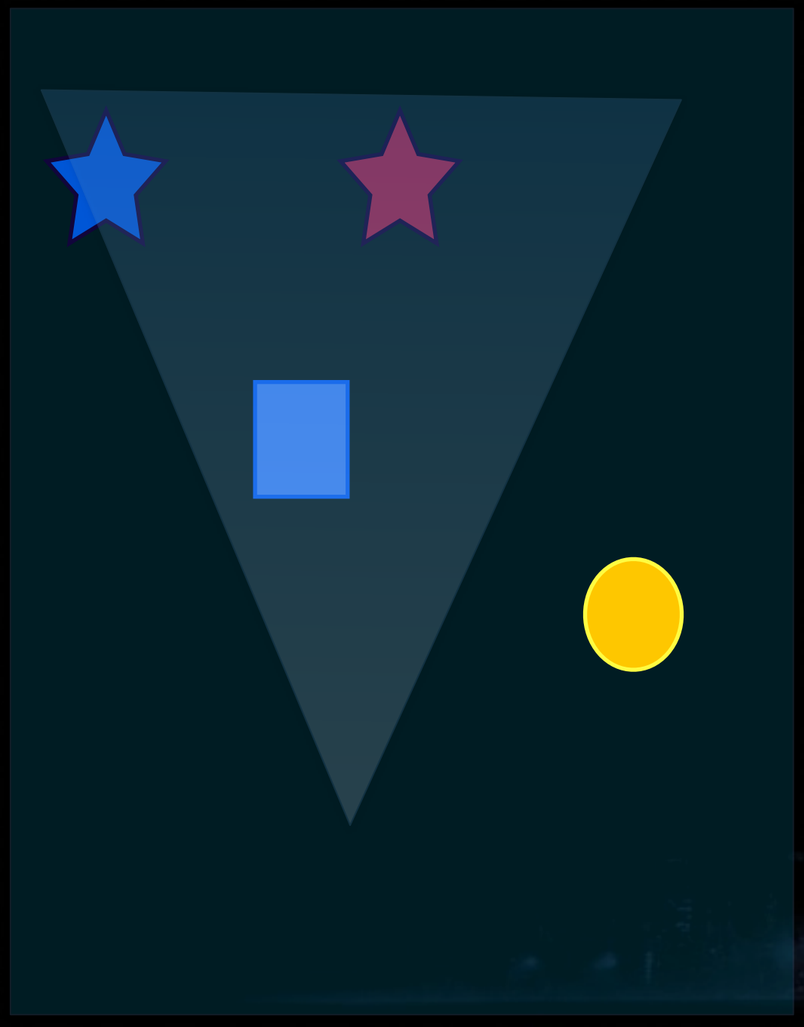
Job 2

Intersecting, Clip, Project

Job 3

Output

Occluder triangles



Job 0

Inside, Project triangles

Job 1

Inside, Project triangles

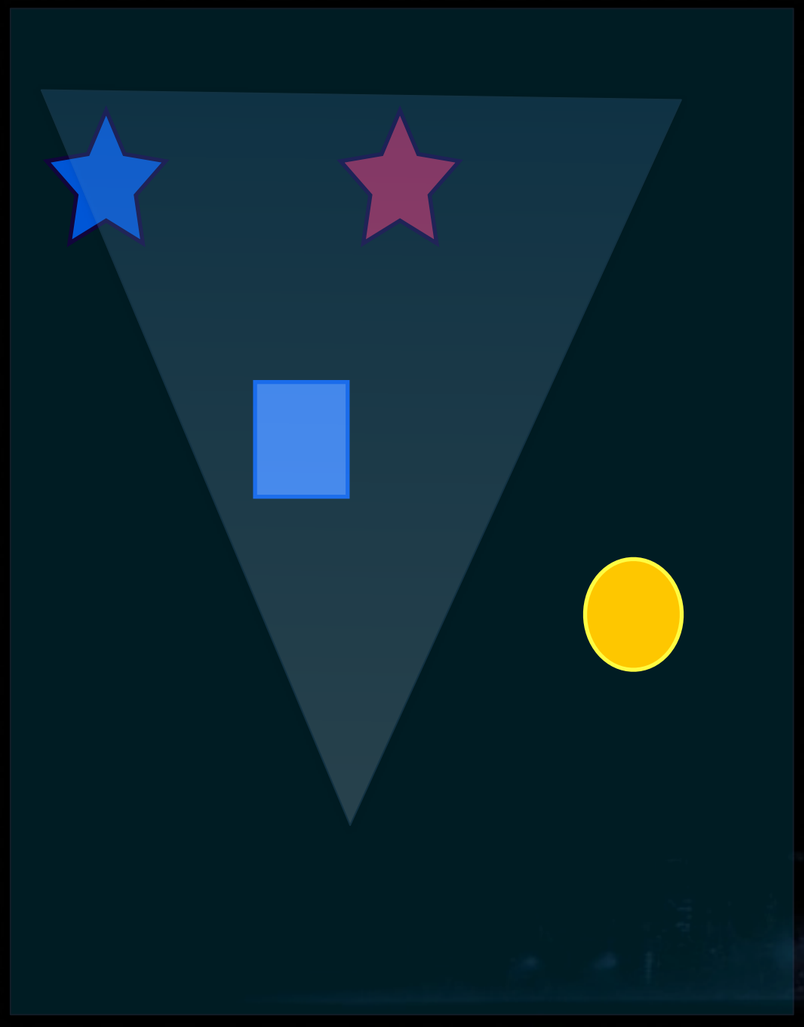
Job 2

Intersecting, Clip, Project

Job 3

Output

Occluder triangles



Job 0 — Inside, Project triangles

Job 1 — Inside, Project triangles

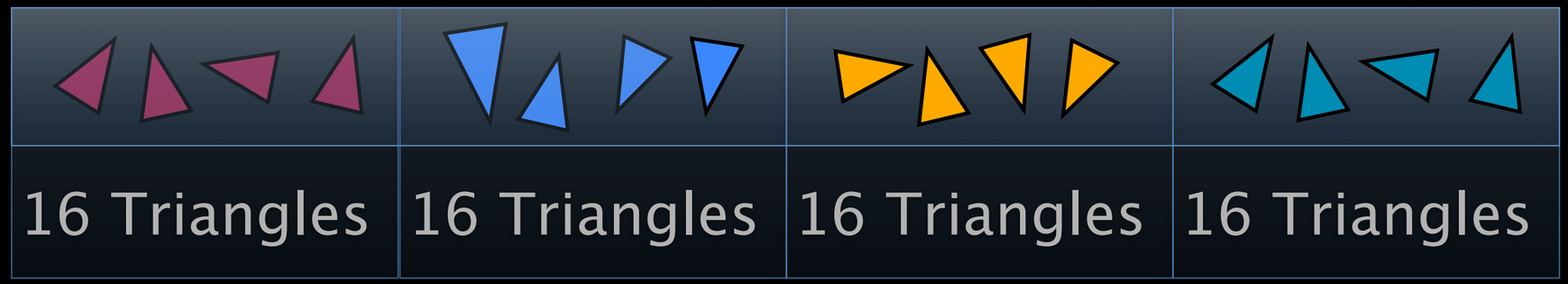
Job 2 — Intersecting, Clip, Project

Job 3 — Outside, Skip

Output

Occluder triangles

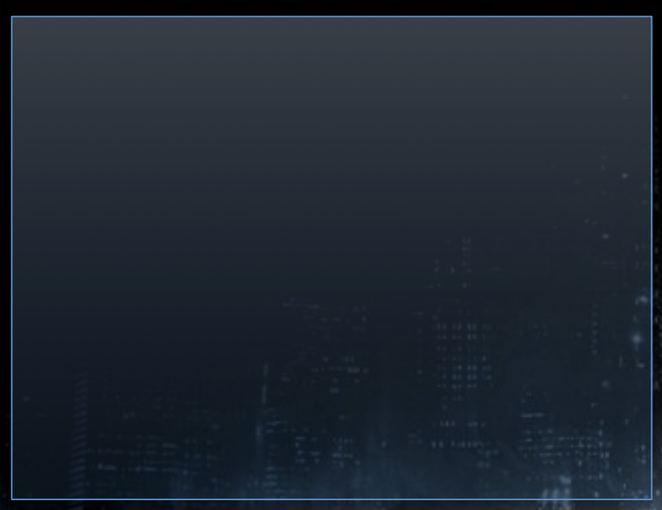
Input

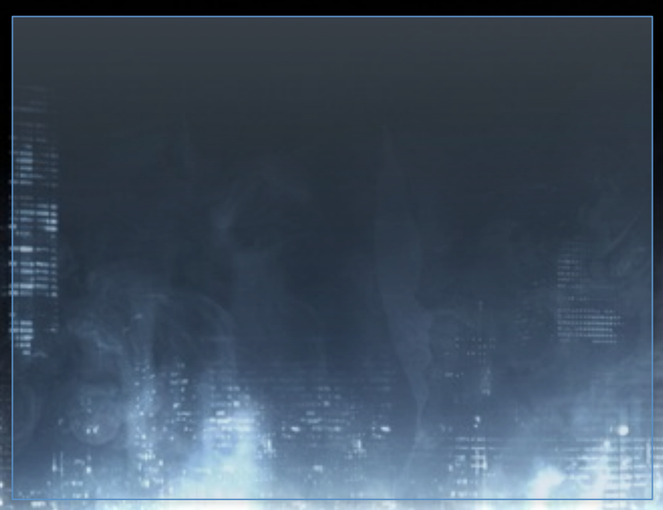


256 x 114 zbuffer

256 x 114 zbuffer

256 x 114 zbuffer





Job 0

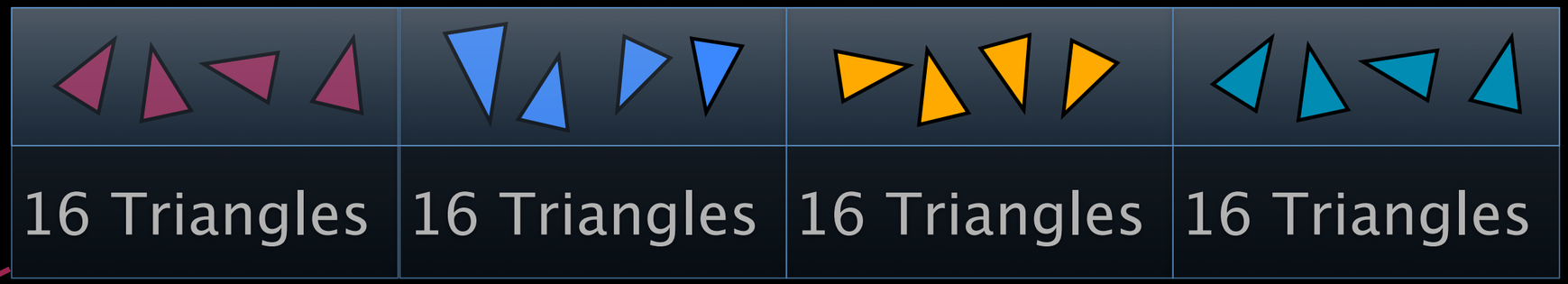
Job 1

Merge step

Colors>

Occluder triangles

Input



256 x 114 zbuffer

256 x 114 zbuffer

256 x 114 zbuffer

Job 0

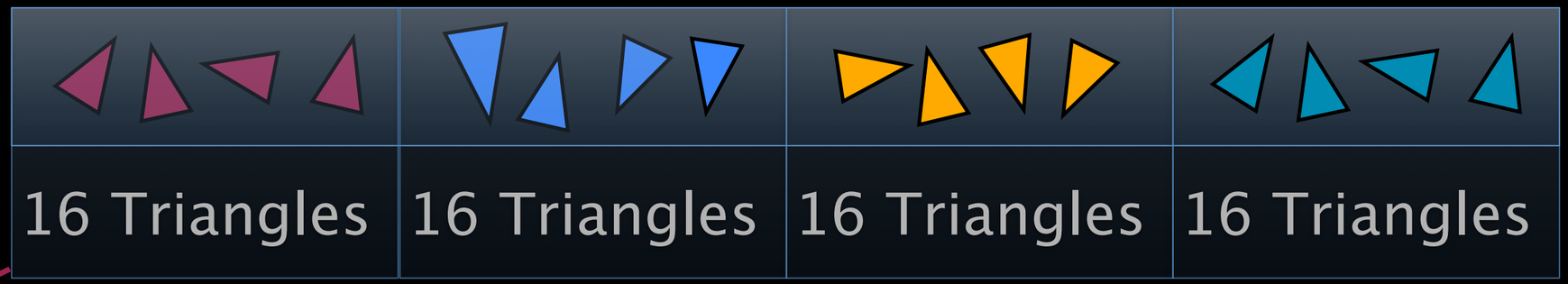
Job 1

Merge step

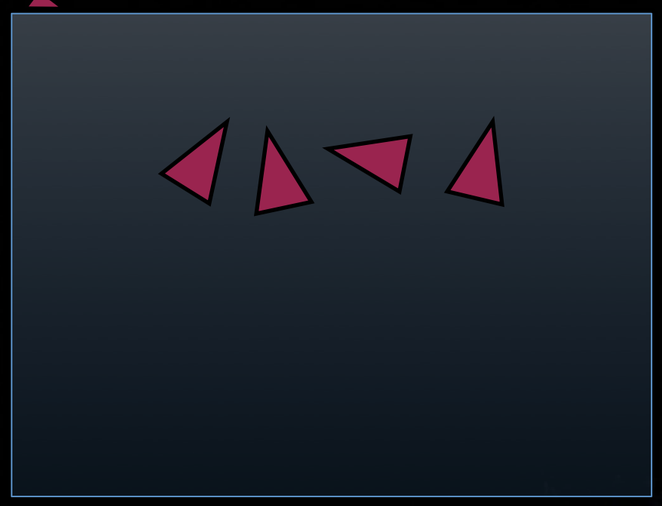
Colors>

Occluder triangles

Input

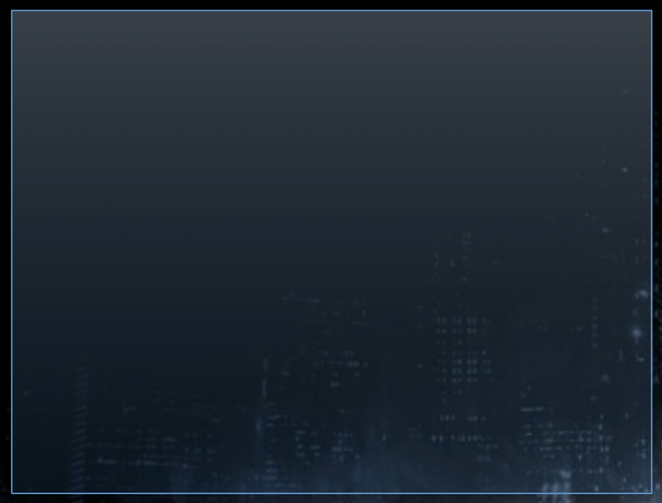


256 x 114 zbuffer



Job 0

256 x 114 zbuffer



Job 1

256 x 114 zbuffer

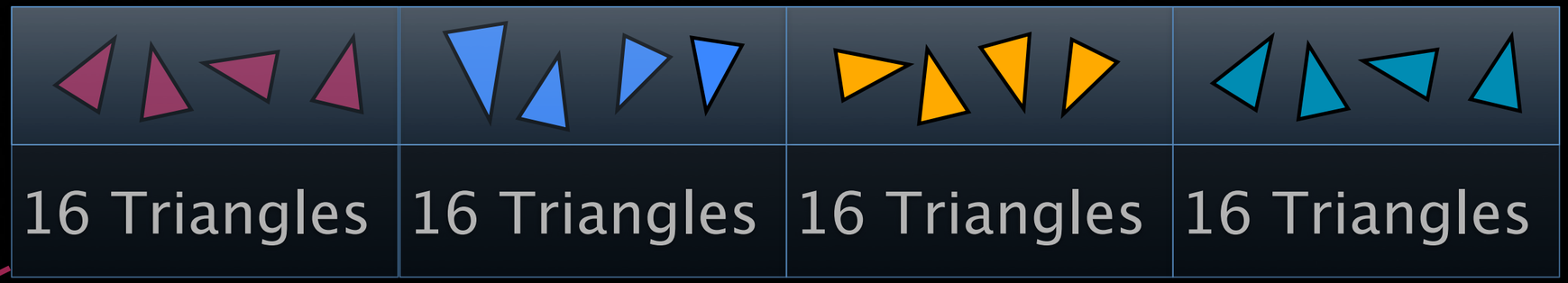
Merge step

Colors>

BATTLEFIELD 3

Occluder triangles

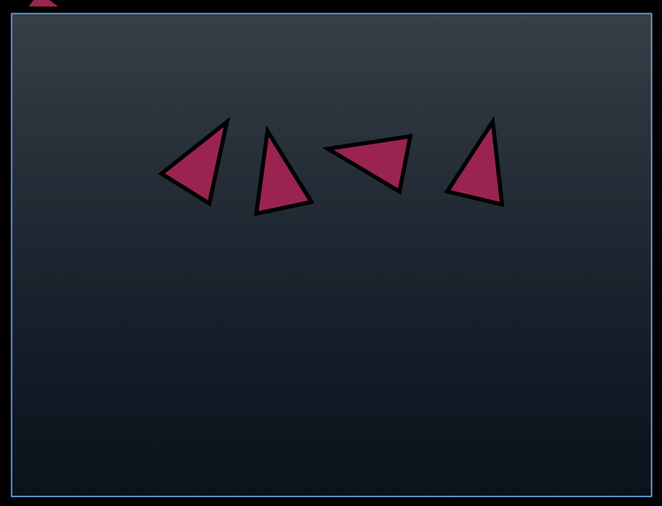
Input



256 x 114 zbuffer

256 x 114 zbuffer

256 x 114 zbuffer



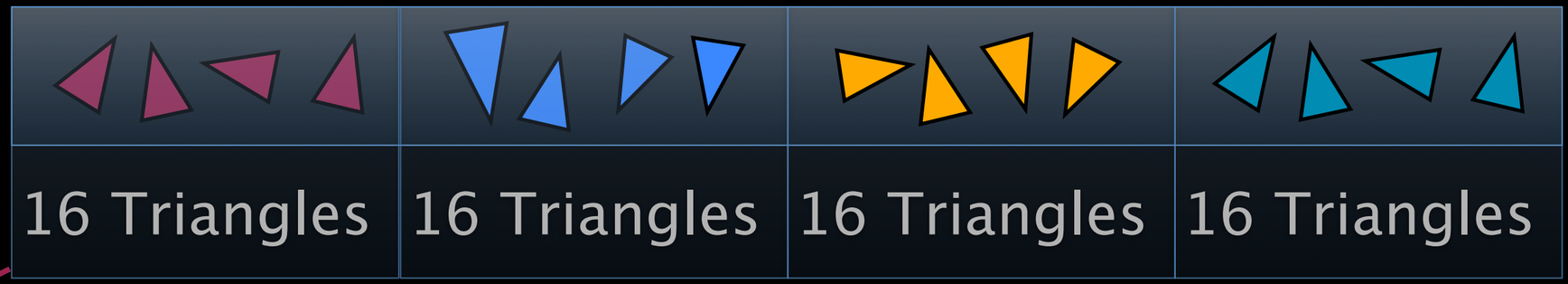
Job 0

Job 1

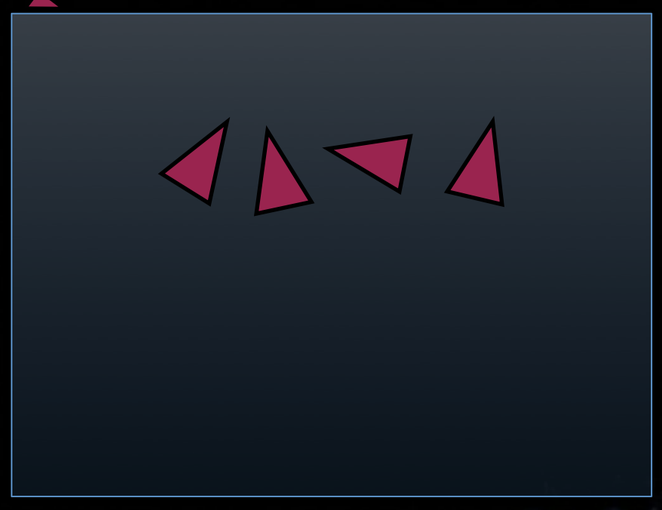
Merge step

Occluder triangles

Input



256 x 114 zbuffer



Job 0

256 x 114 zbuffer

Job 1

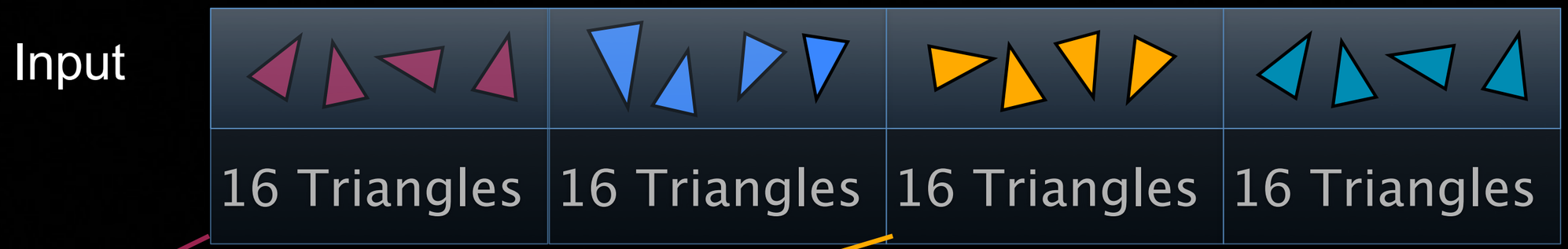
256 x 114 zbuffer

Merge step

BATTLEFIELD 3

DICE

Occluder triangles



256 x 114 zbuffer

256 x 114 zbuffer

256 x 114 zbuffer

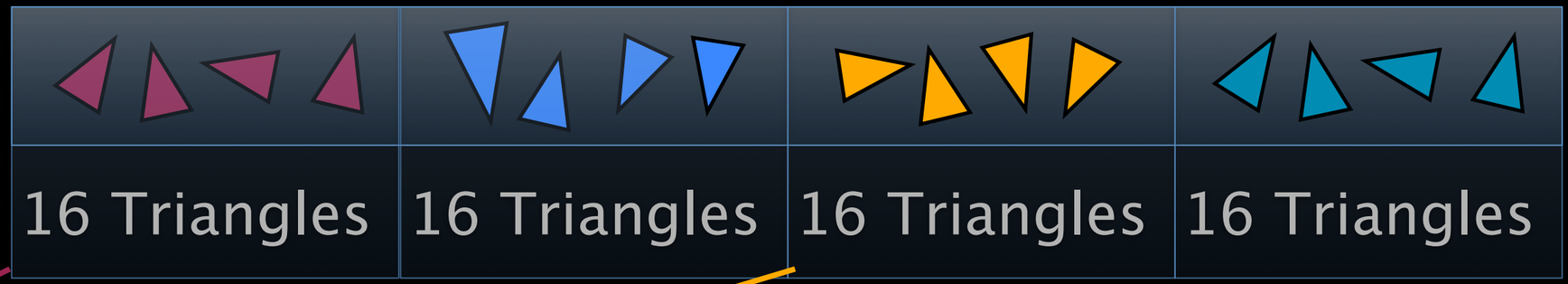
Job 0

Job 1

Merge step

Occluder triangles

Input



256 x 114 zbuffer

256 x 114 zbuffer

256 x 114 zbuffer

Job 0

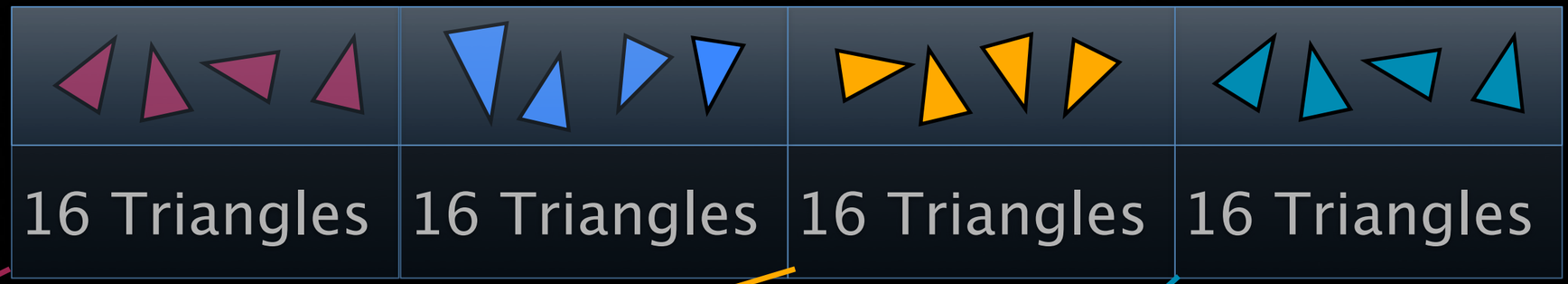
Job 1

Merge step

BATTLEFIELD 3

Occluder triangles

Input



256 x 114 zbuffer

256 x 114 zbuffer

256 x 114 zbuffer

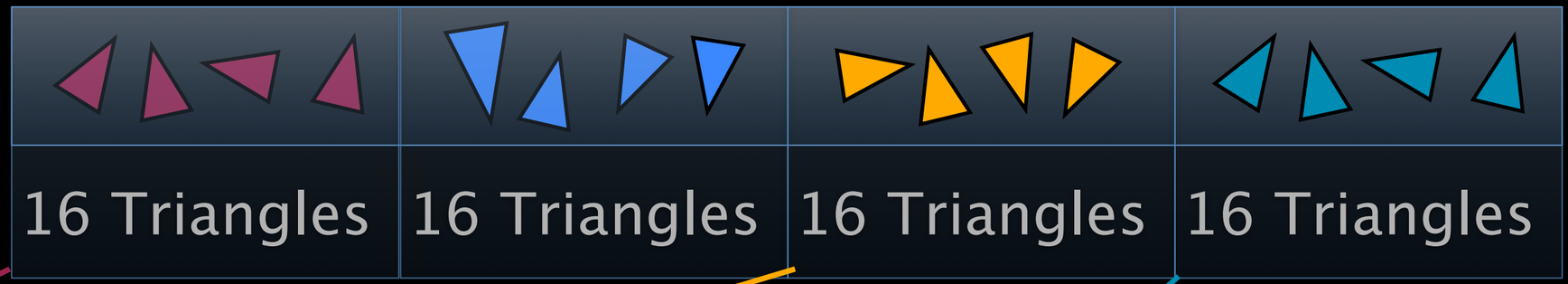
Job 0

Job 1

Merge step

Occluder triangles

Input



256 x 114 zbuffer

256 x 114 zbuffer

256 x 114 zbuffer

Job 0

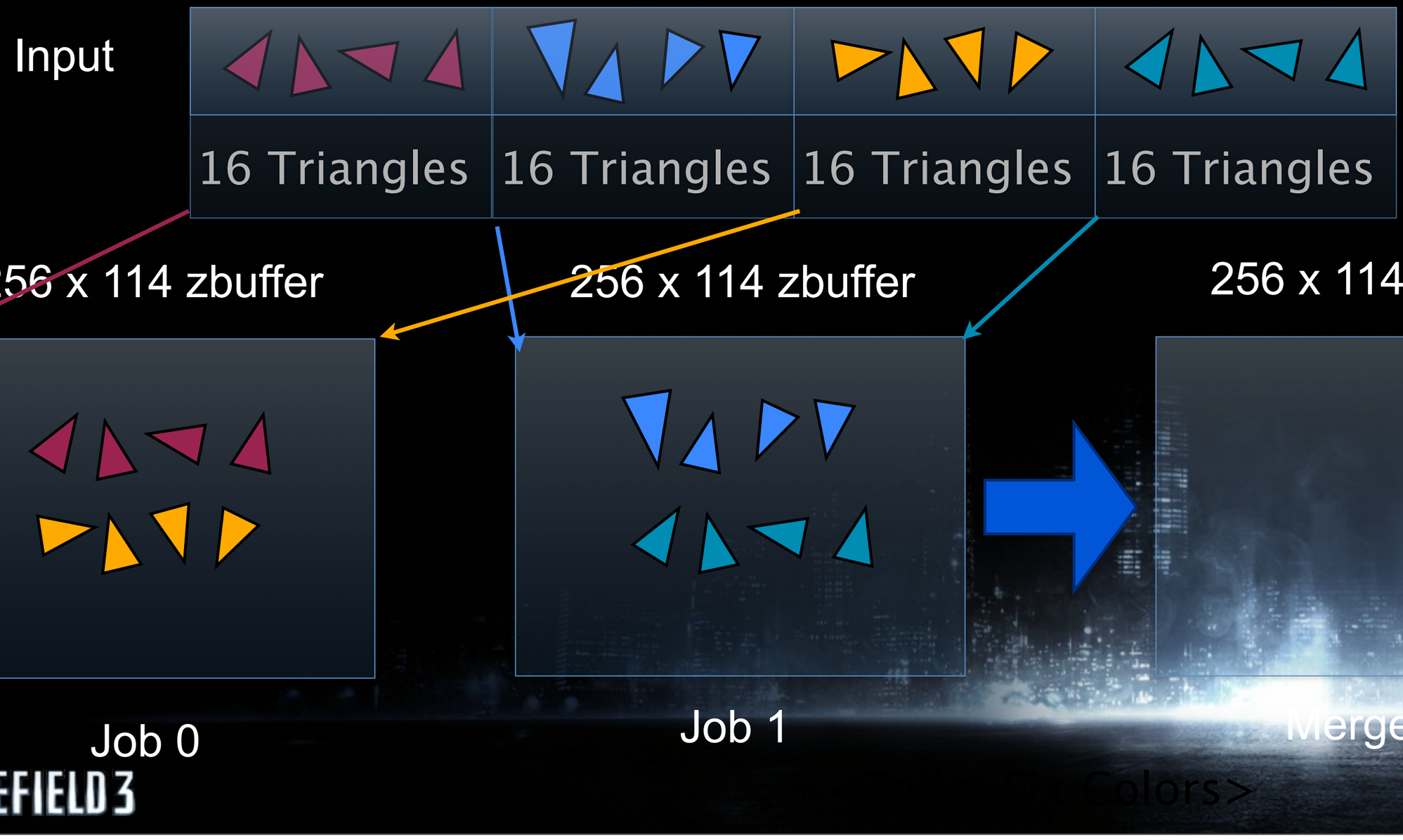
Job 1

Merge step

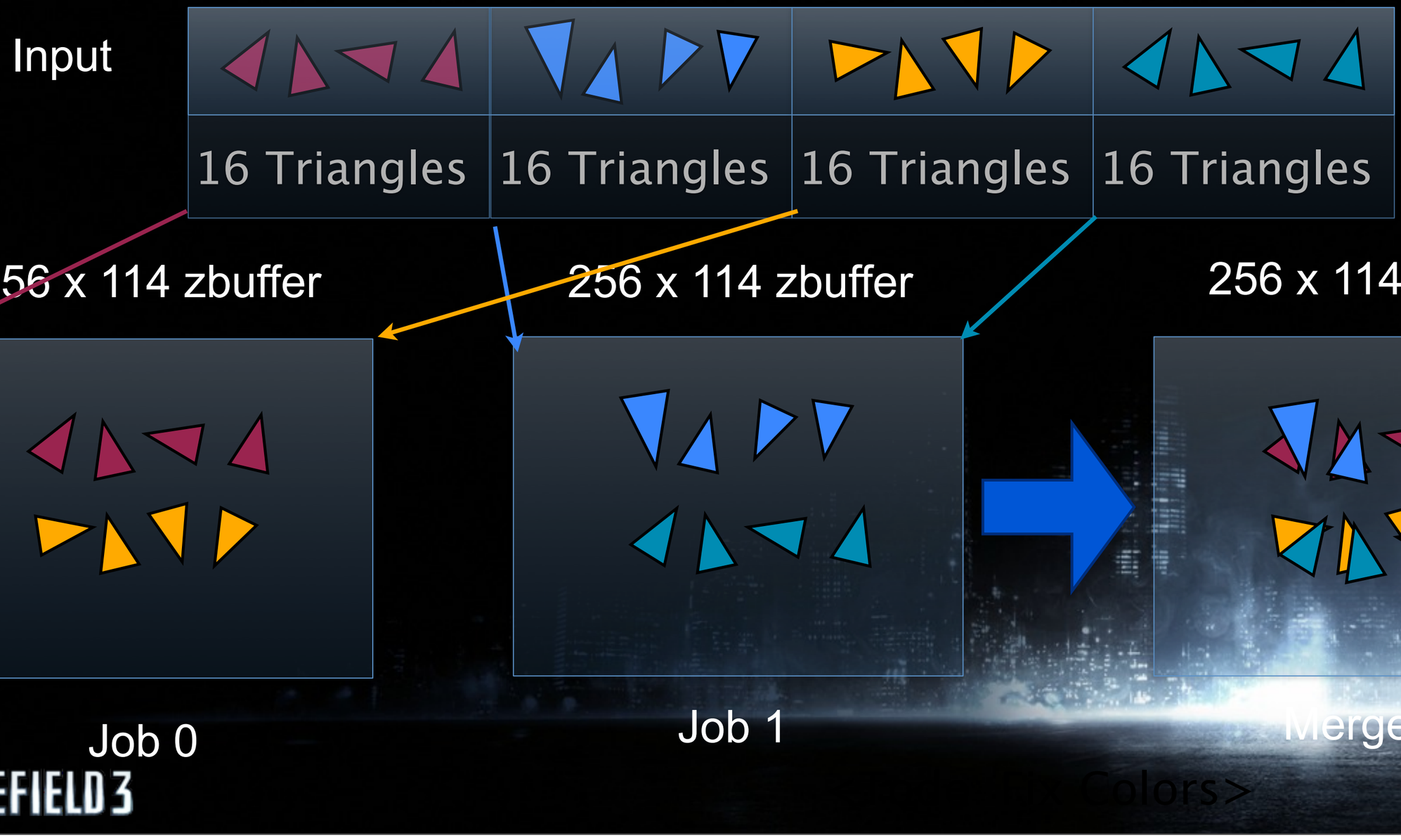
Colors>

BATTLEFIELD 3

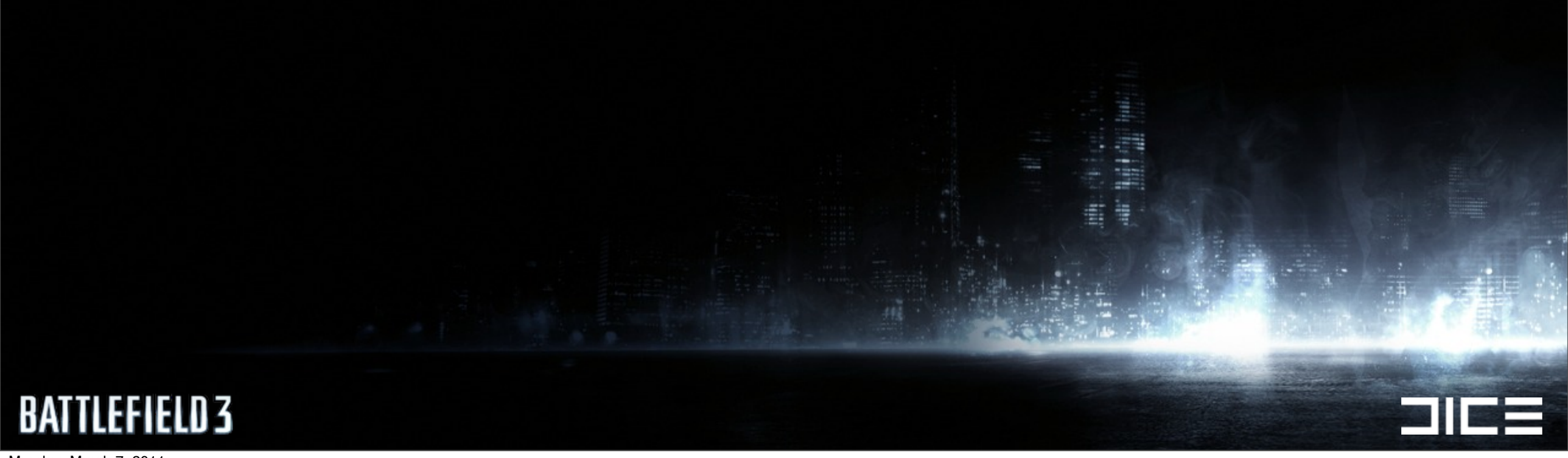
Occluder triangles



Occluder triangles



z-buffer testing

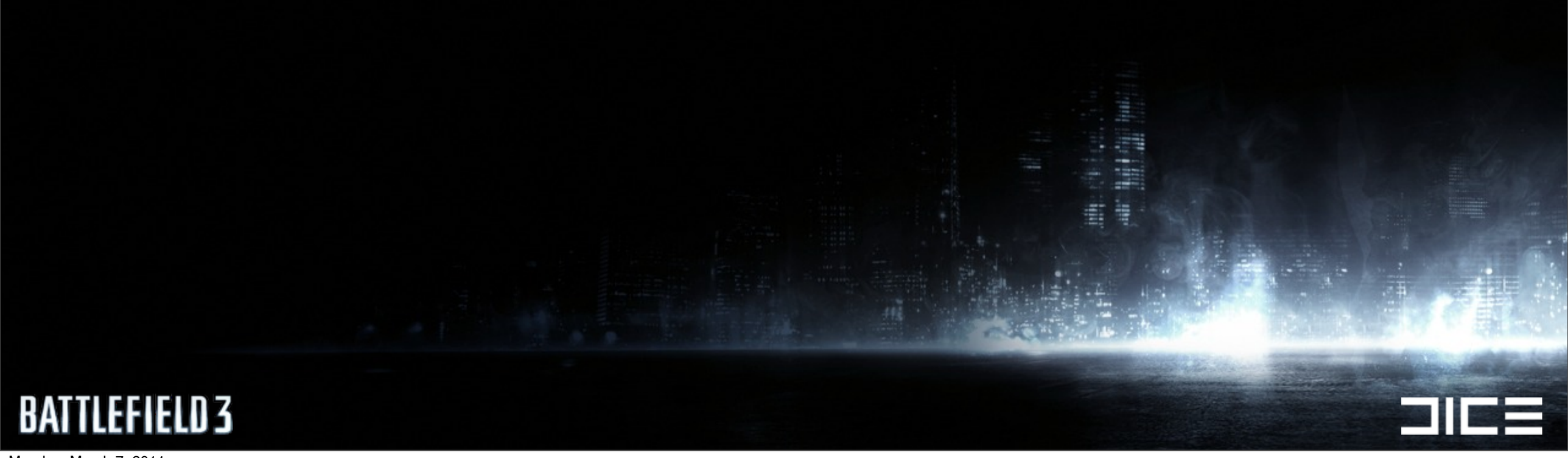


z-buffer testing

- Calculate screen space AABB for object

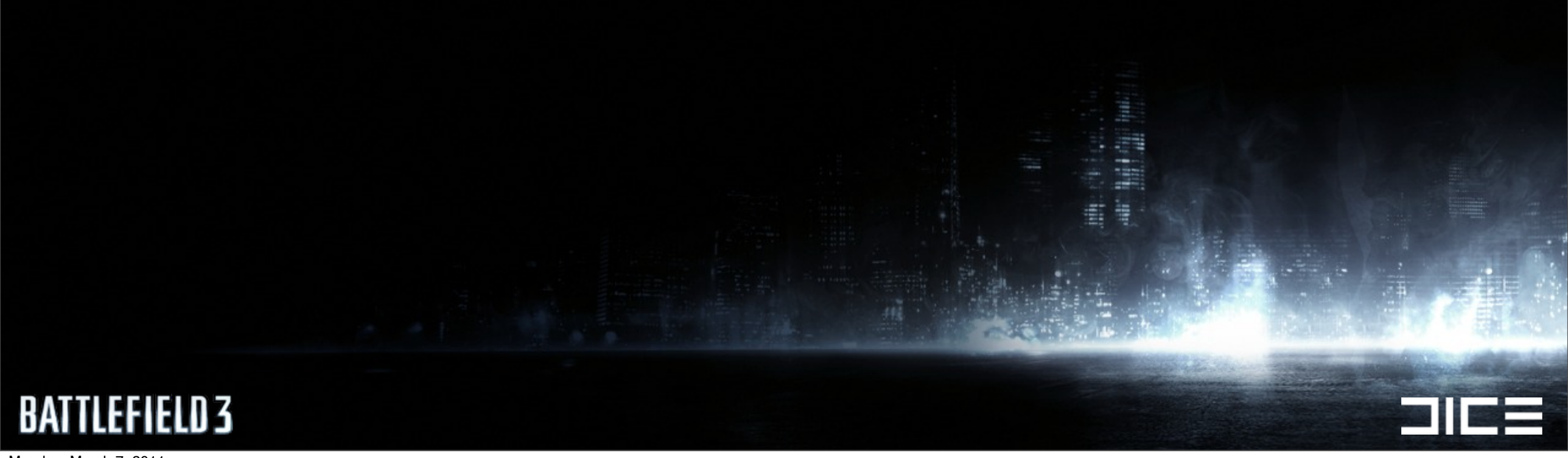
z-buffer testing

- Calculate screen space AABB for object
- Get single distance value



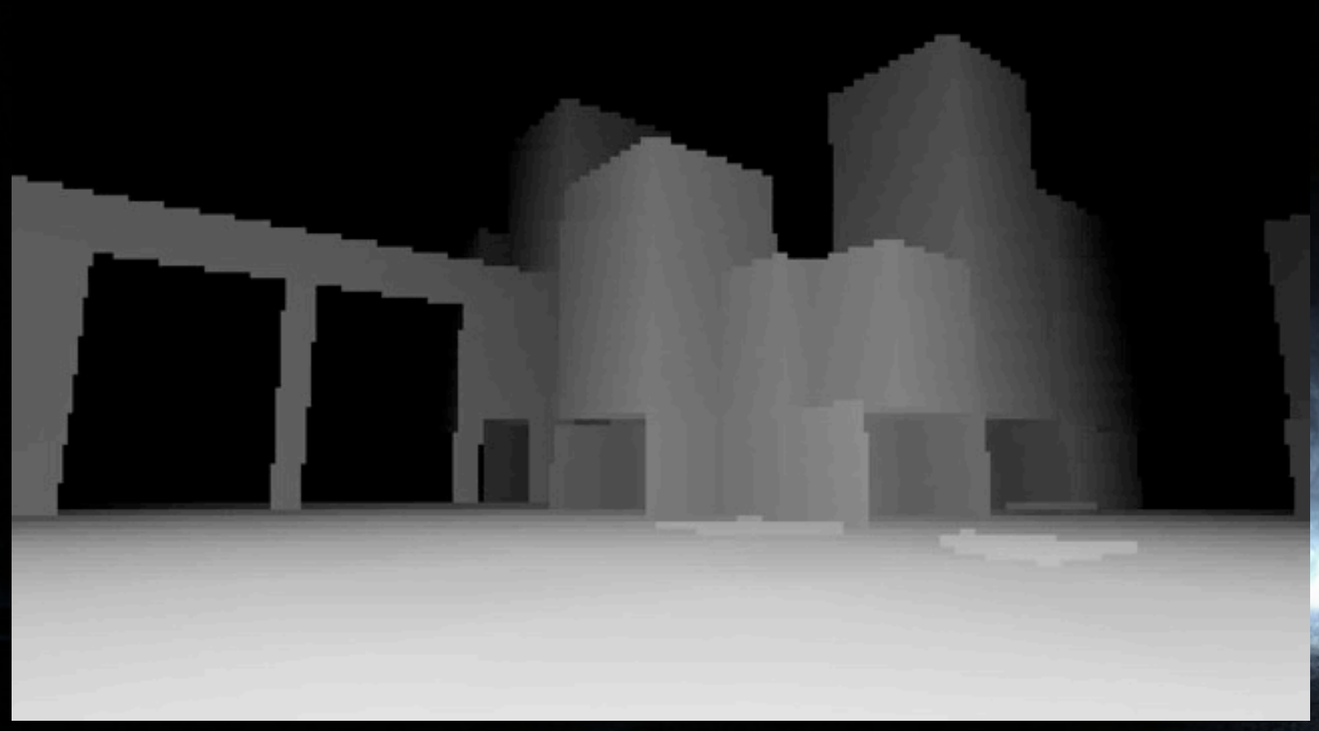
z-buffer testing

- Calculate screen space AABB for object
- Get single distance value
- Test the square against the z-buffer



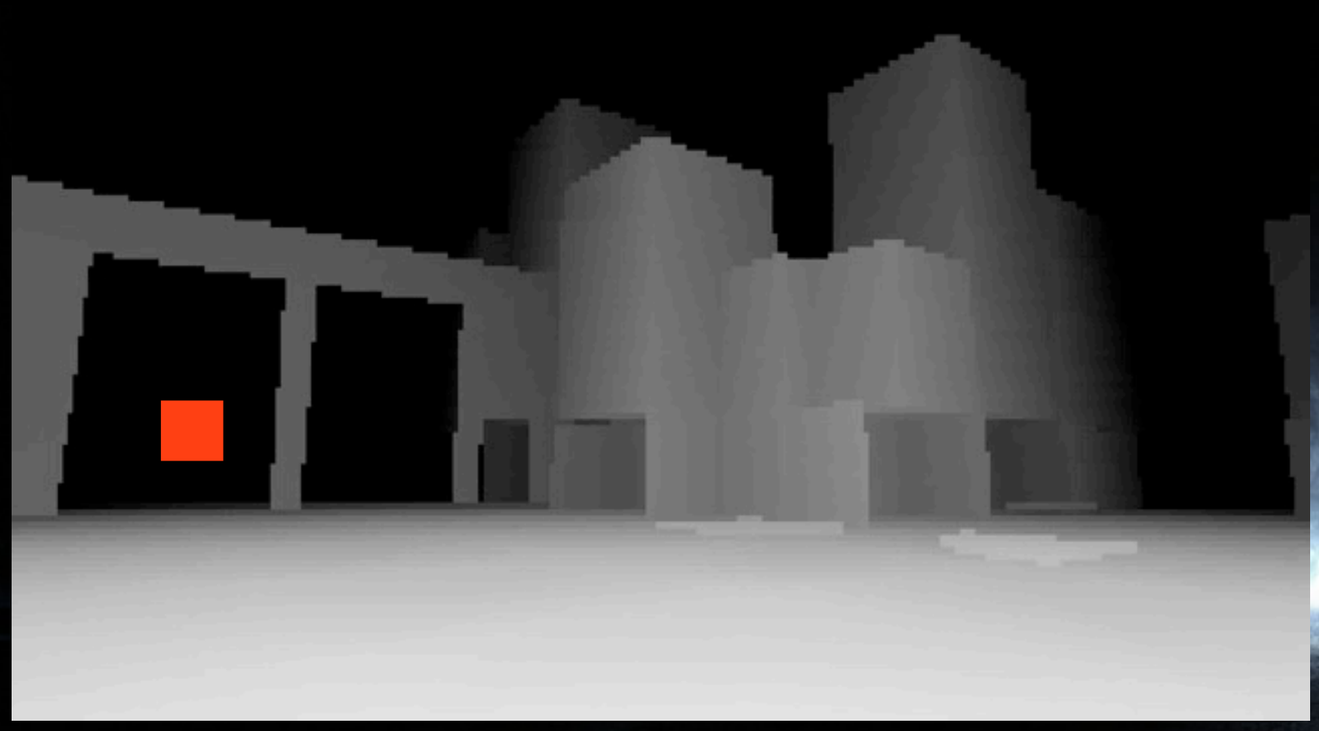
z-buffer testing

- Calculate screen space AABB for object
- Get single distance value
- Test the square against the z-buffer



z-buffer testing

- Calculate screen space AABB for object
- Get single distance value
- Test the square against the z-buffer

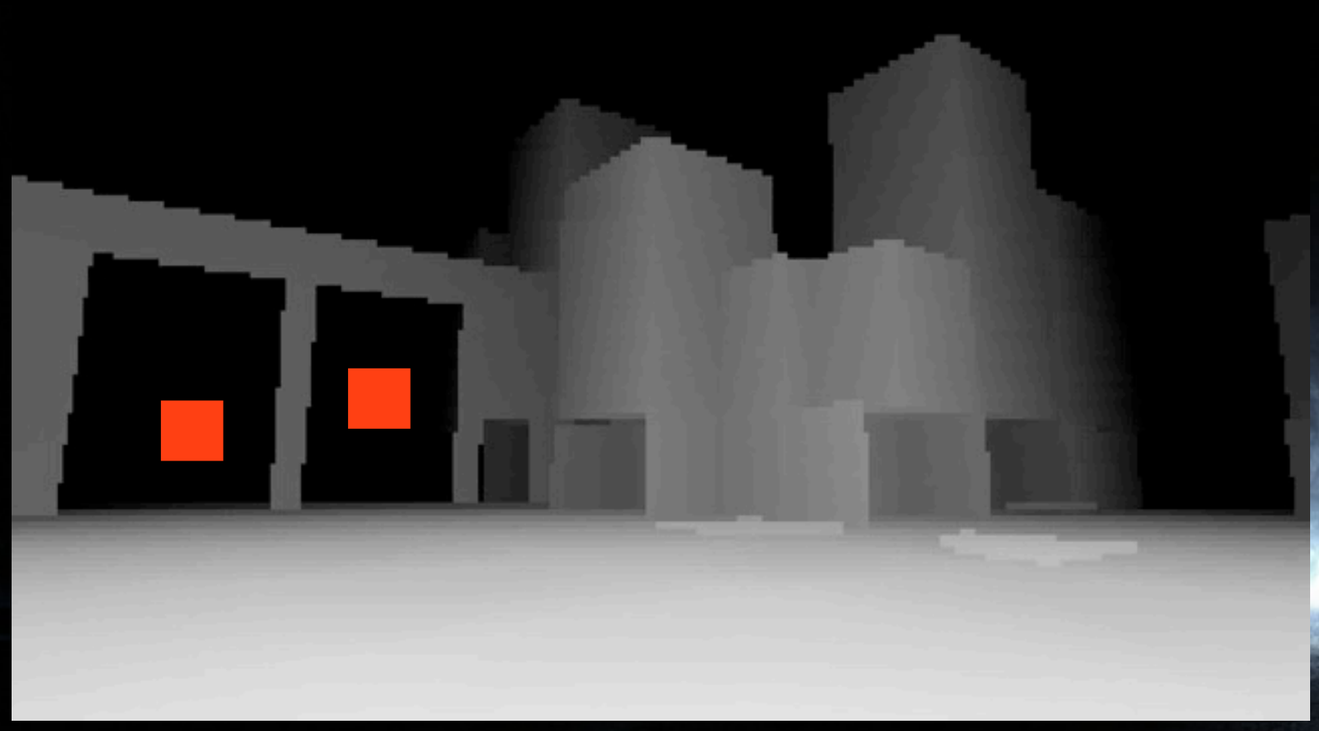


BATTLEFIELD 3

DICE

z-buffer testing

- Calculate screen space AABB for object
- Get single distance value
- Test the square against the z-buffer

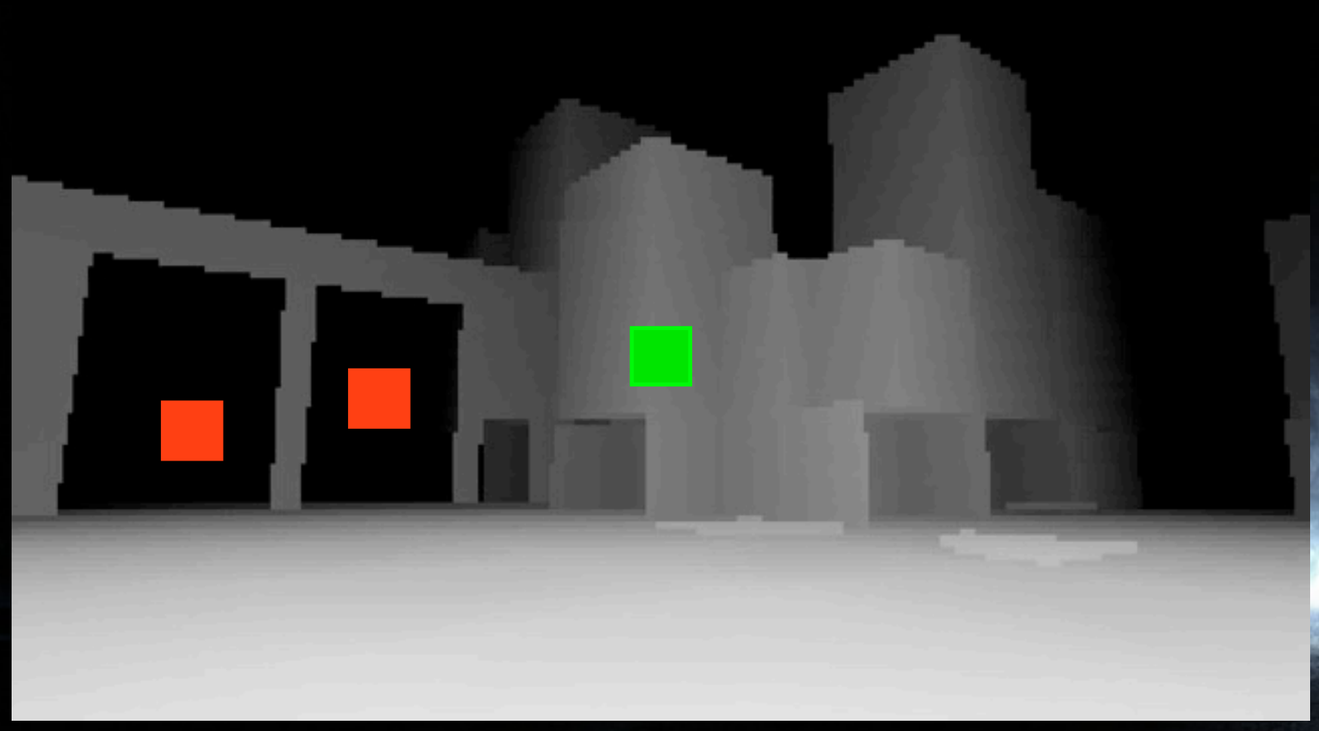


BATTLEFIELD 3

DICE

z-buffer testing

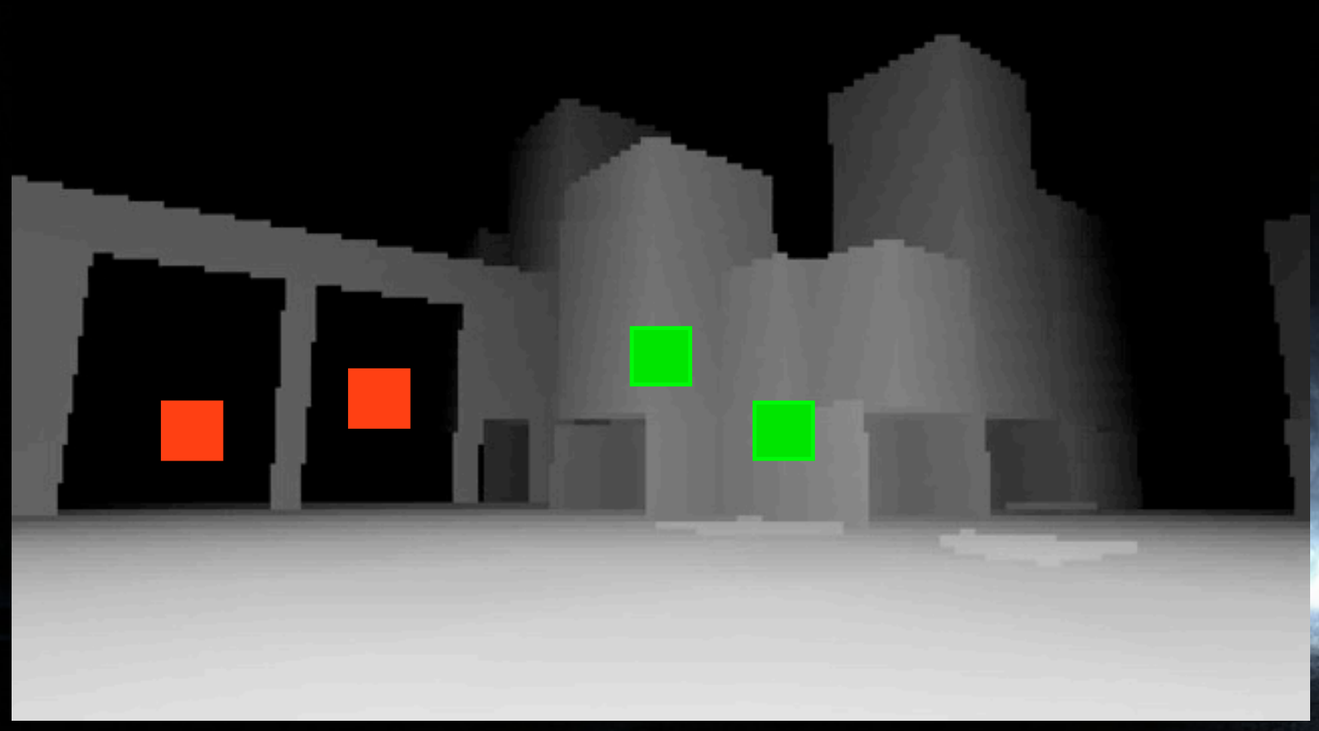
- Calculate screen space AABB for object
- Get single distance value
- Test the square against the z-buffer



BATTLEFIELD 3

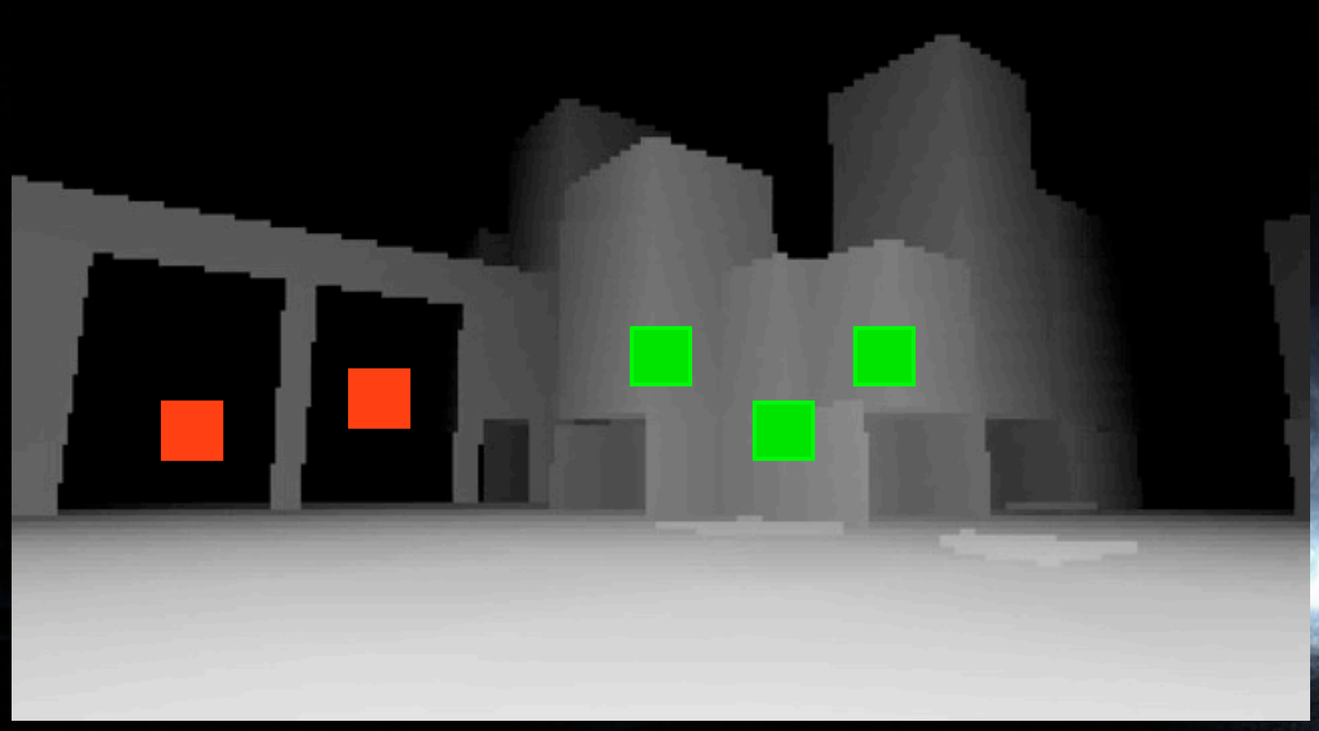
z-buffer testing

- Calculate screen space AABB for object
- Get single distance value
- Test the square against the z-buffer



z-buffer testing

- Calculate screen space AABB for object
- Get single distance value
- Test the square against the z-buffer



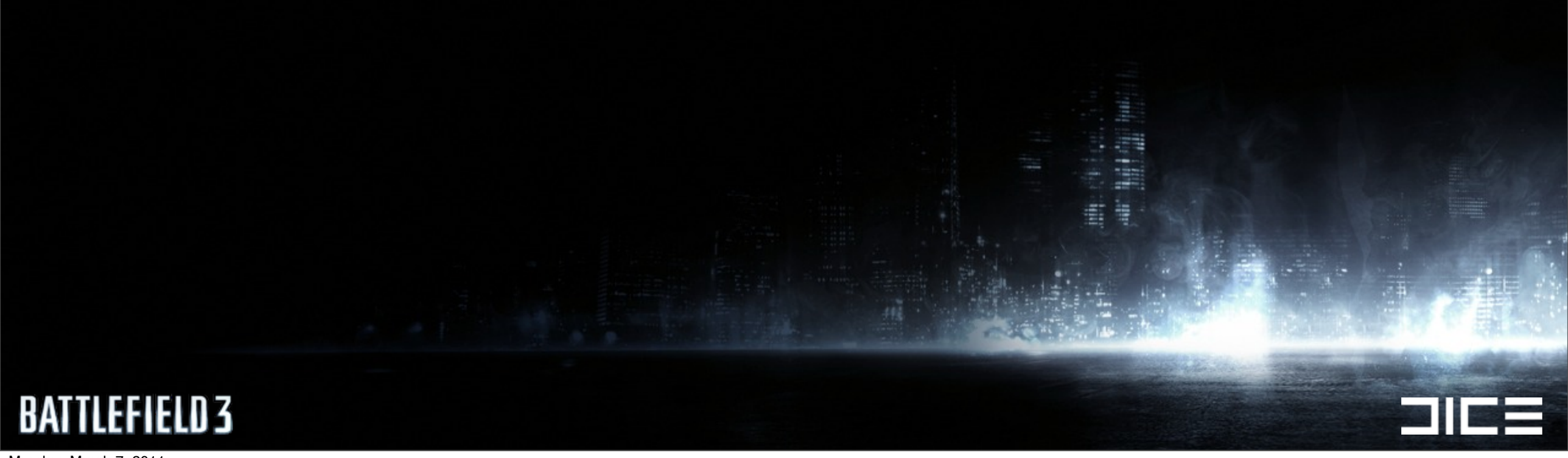
BATTLEFIELD 3

DICE

Conclusion

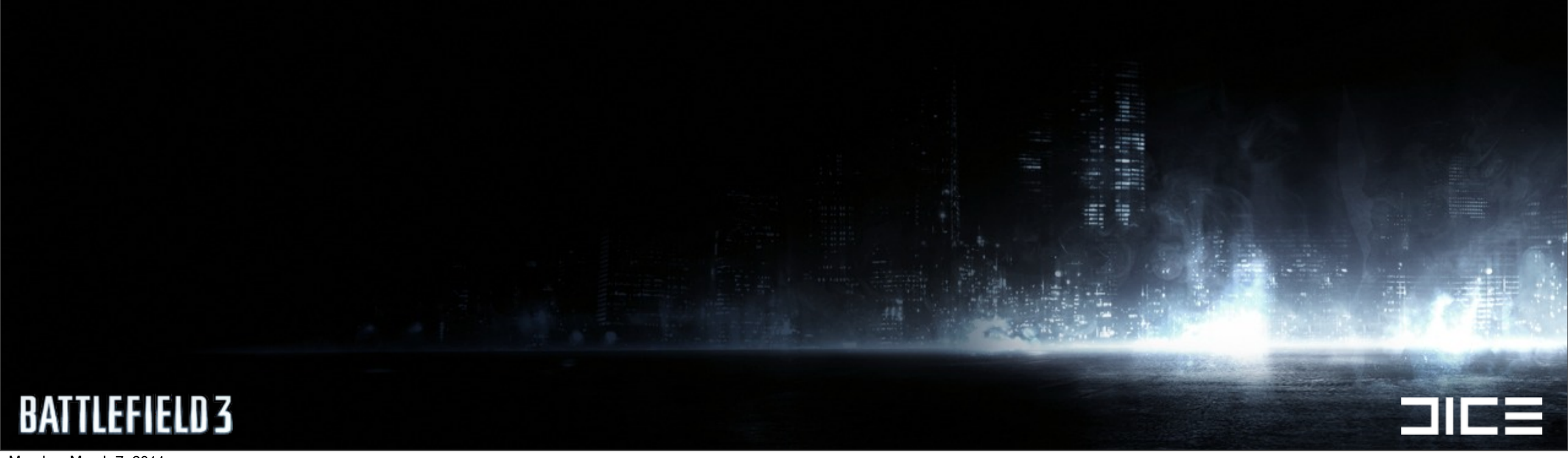
Conclusion

- Accurate and high performance culling is essential



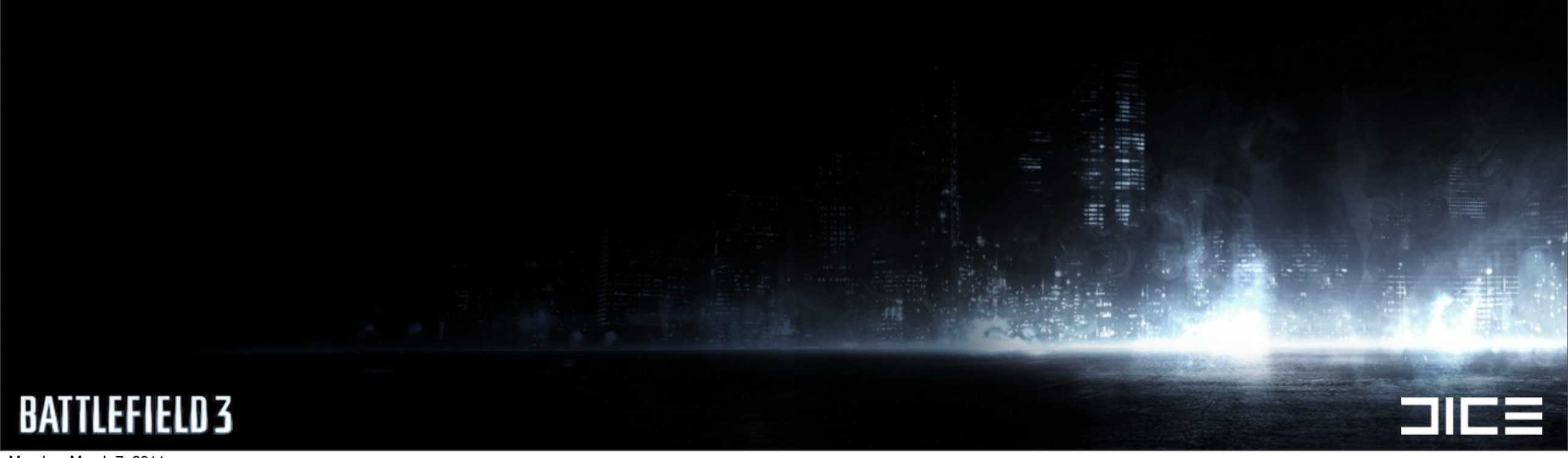
Conclusion

- Accurate and high performance culling is essential
- Reduces pressure on low-level systems/rendering



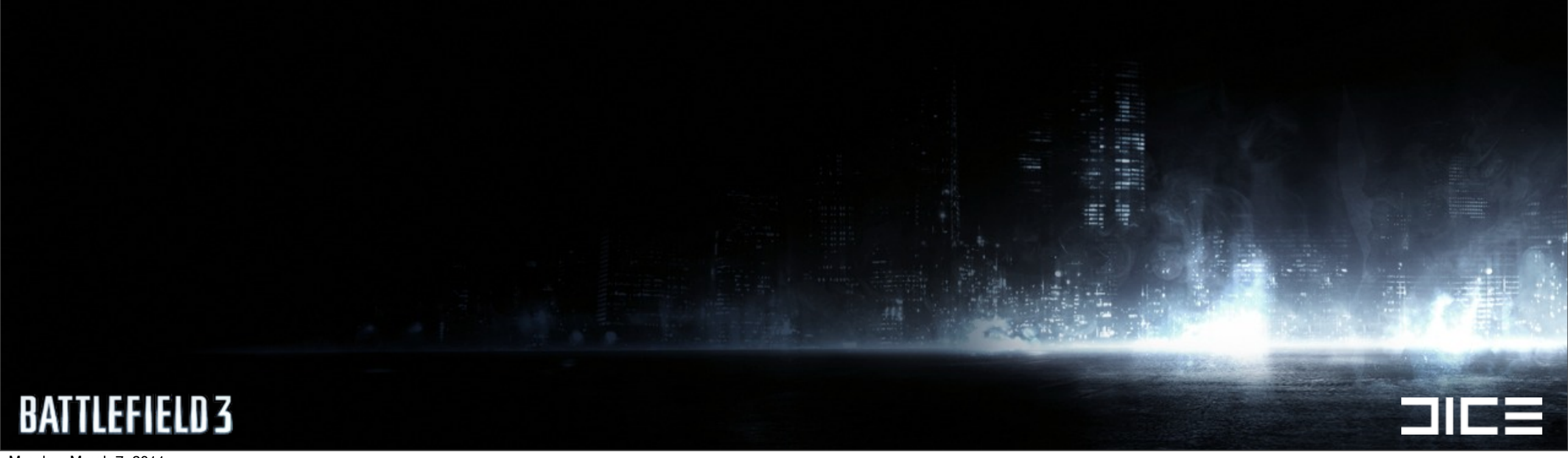
Conclusion

- Accurate and high performance culling is essential
- Reduces pressure on low-level systems/rendering
- It's all about data



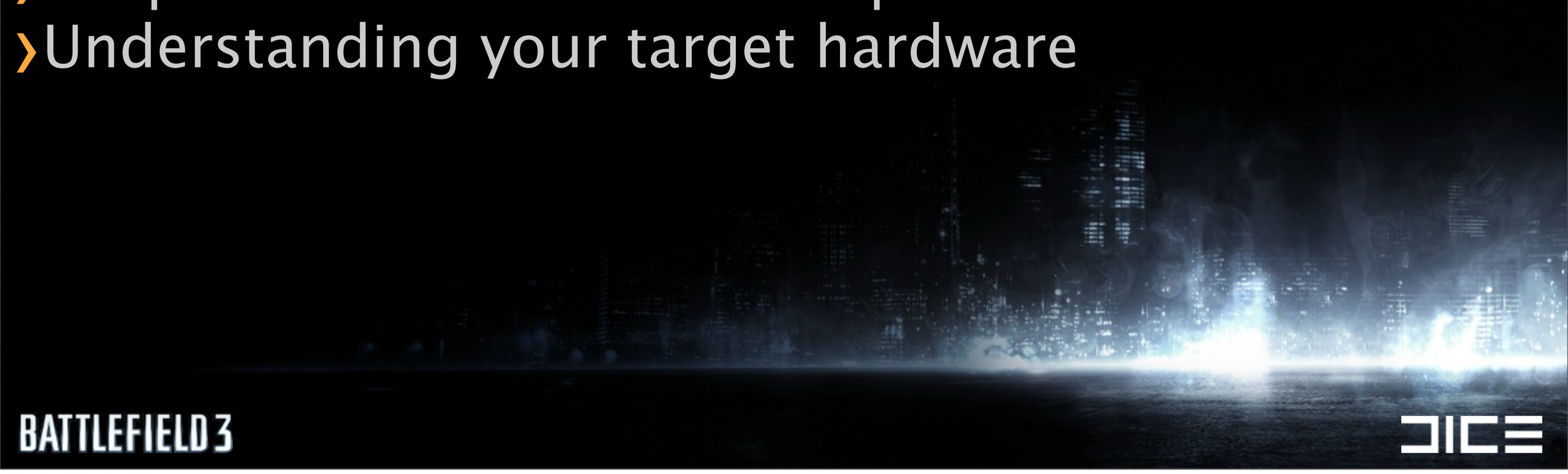
Conclusion

- Accurate and high performance culling is essential
- Reduces pressure on low-level systems/rendering
- It's all about data
- Simple data often means simple code



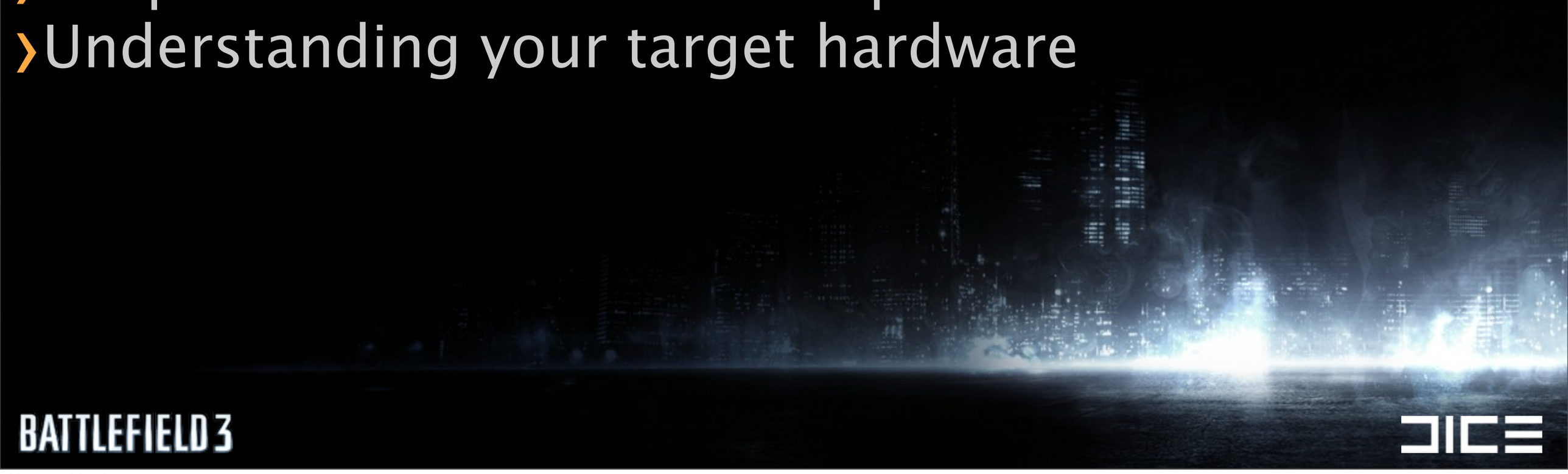
Conclusion

- Accurate and high performance culling is essential
- Reduces pressure on low-level systems/rendering
- It's all about data
- Simple data often means simple code
- Understanding your target hardware



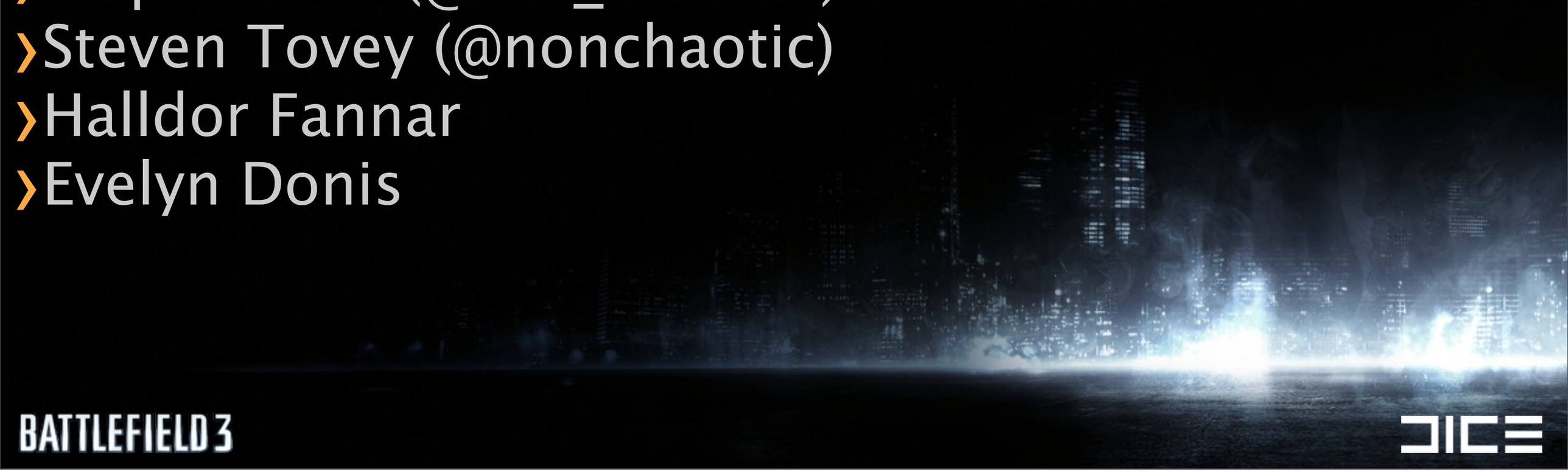
Conclusion

- Accurate and high performance culling is essential
- Reduces pressure on low-level systems/rendering
- It's all about data
- Simple data often means simple code
- Understanding your target hardware



Thanks to

- Andreas Fredriksson (@deplinenoise)
- Christina Coffin (@christinacoffin)
- Johan Andersson (@repi)
- Stephen Hill (@self_shadow)
- Steven Tovey (@nonchaotic)
- Halldor Fannar
- Evelyn Donis



Questions?

Email: daniel@collin.com
Blog: zenic.org
Twitter: @daniel_collin

Battlefield 3 & Frostbite 2 talks at GDC'11:

Mon 1:45	*DX11 Rendering in Battlefield 3*	Johan Andersson
Wed 10:30	*SPU-based Deferred Shading in Battlefield 3 for PlayStation 3*	Christina Coffin
Wed 3:00	*Culling the Battlefield: Data Oriented Design in Practice*	Daniel Collin
Thu 1:30	*Lighting You Up in Battlefield 3*	Kenny Magnusson
Fri 4:05	*Approximating Translucency for a Fast, Cheap & Convincing Subsurface Scattering Look*	Colin Barré-Brisebois

GDC

For more DICE talks: http://publications.dice.se

DICE