GDC 09

learn
network
inspire

www.GDConf.com

Game Developers Conference®

March 23-27, 2009 | Moscone Center, San Francisco

Shadows & Decals: D3D10 techniques from Frostbite

Johan Andersson
Daniel Johansson

Single-pass Stable Cascaded Bounding Box Shadow Maps

(SSCBBSM?!)

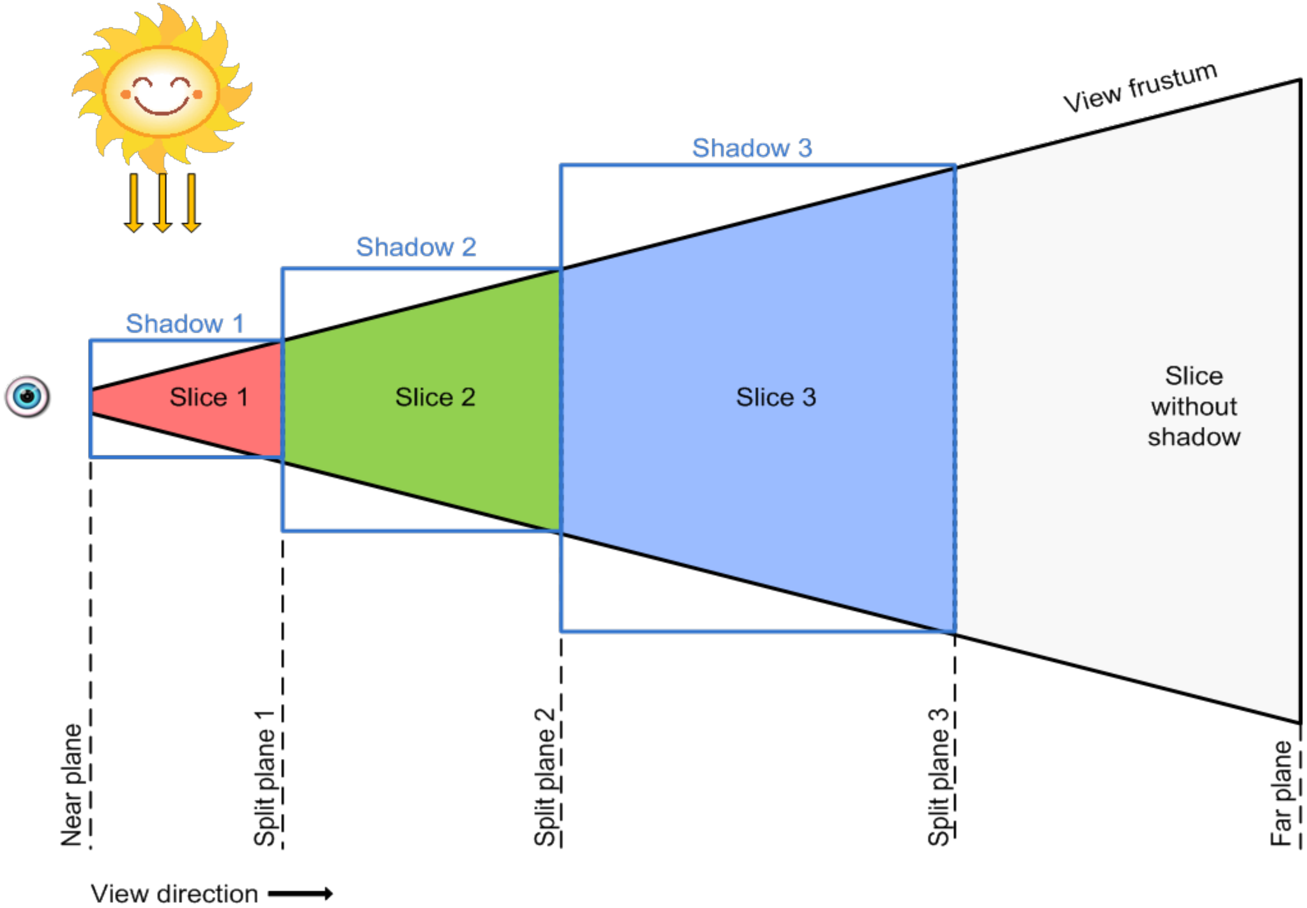
Johan Andersson

Overview

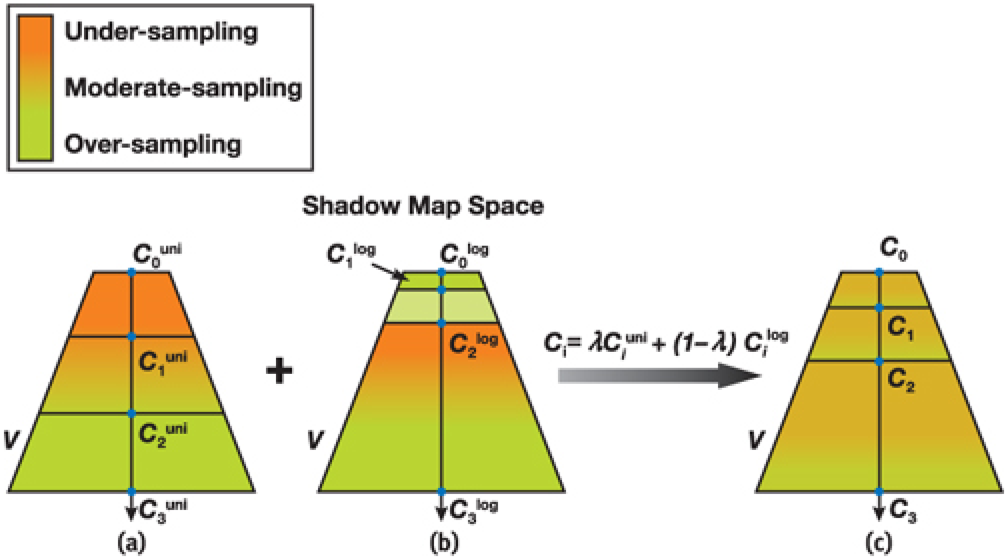
- Basics
- Shadowmap rendering
- Stable shadows
- Scene rendering
- Conclusions

- (Q&A after 2nd part)

Cascaded Shadow Maps



Practical Split Scheme

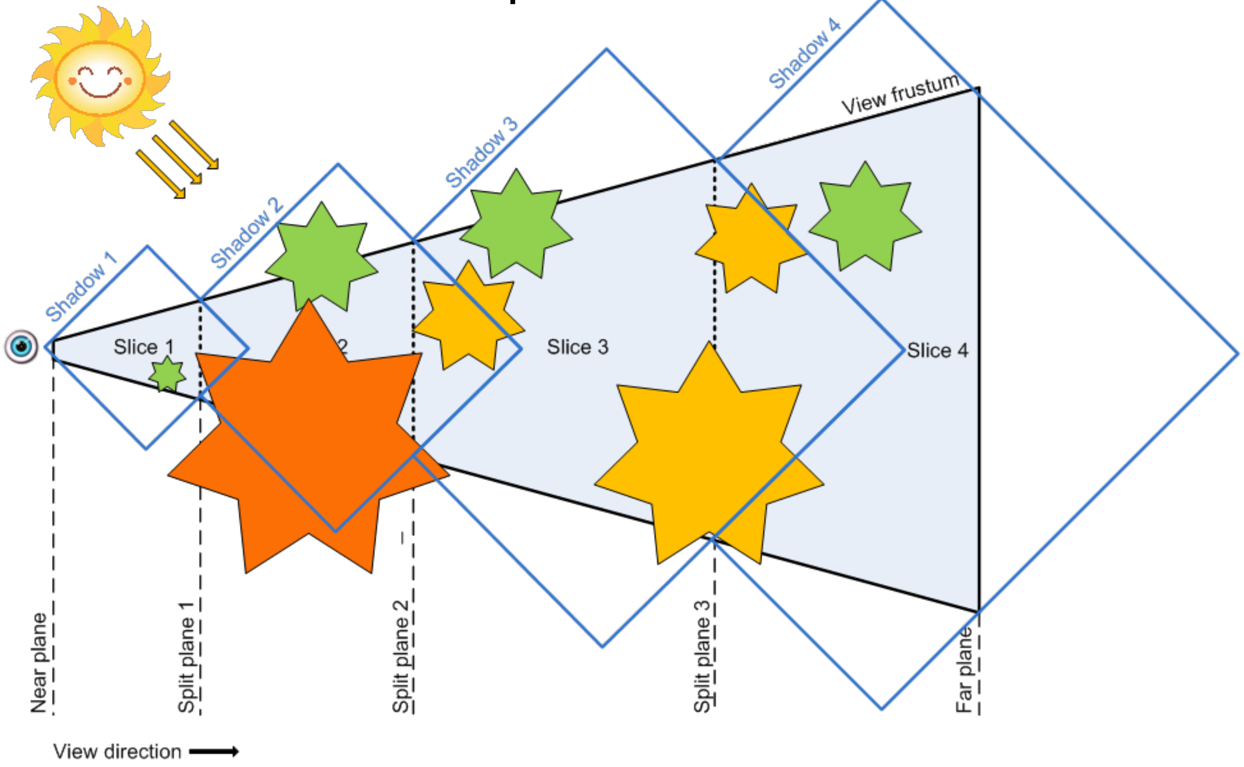


From: Parallel-Split Shadow Maps on Programmable GPUs [1]

```
for (uint sliceIt = 0; sliceIt < sliceCount; sliceIt++)
{
    float f = float(sliceIt+1)/sliceCount;
    float logDistance = nearPlane * pow(shadowDistance/nearPlane, f);
    float uniformDistance = nearPlane + (shadowDistance - nearPlane) * f;
    splitDistances[sliceIt] = lerp(uniformDistance, logDistance, weight);
}
```

Traditional Shadowmap Rendering

- Render world *n* times to *n* shadowmaps
  - Objects interesecting multiple slices are rendered multiple times



Traditional Shadowmap Rendering

- More/larger objects or more slices = more overhead
- Both a CPU & GPU issue
  - CPU: draw call / state overhead
  - GPU: primarily extra vertices & primitives
- Want to reduce CPU overhead
  - More objects
  - More slices = higher resolution
  - Longer shadow view distance

DX10 Single-pass Shadowmap Rendering

- Single draw call outputs to multiple slices
  - Shadowmap is a texture array
  - Depth stencil array view with multiple slices
  - Geometry shader selects output slice with `SV_RenderTargetArrayIndex`
- No CPU overhead
  - With many objects intersecting multiple frustums
- Multiple implementations possible

Shadowmap texture array view

- Creation:

```
D3D10_DEPTH_STENCIL_VIEW_DESC viewDesc;
viewDesc.Format = DXGI_FORMAT_D24_UNORM_S8_UINT;
viewDesc.ViewDimension = D3DALL_DSV_DIMENSION_TEXTURE2DARRAY;
viewDesc.Texture2DArray.FirstArraySlice = 0;
viewDesc.Texture2DArray.ArraySize = sliceCount;
viewDesc.Texture2DArray.MipSlice = 0;
device->CreateDepthStencilView(shadowmapTexture, &viewDesc, &view);
```

- SampleCmp only supported on 10.1 for texture arrays
  - 10.0 fallback: Manual PCF-filtering
  - Or vendor-specific APIs, ask your IHV rep.

SV_RenderTargetArrayIndex

- Geometry shader output value

- Selects which texture slice each primitive should be rendered to

- Available from D3D 10.0

Geometry shader cloning

```
#define SLICE_COUNT 4
float4x4 sliceViewProjMatrices[SLICE_COUNT];

struct GsInput
{
    float4 worldPos : SV_POSITION;
    float2 texCoord : TEXCOORD0;
};
struct PsInput
{
    float4 hPos : SV_POSITION;
    float2 texCoord : TEXCOORD0;
    uint sliceIndex : SV_RenderTargetArrayIndex;
};

[maxvertexcount(SLICE_COUNT*3)]
void main(triangle GsInput input[3],
          inout TriangleStream<PsInput> stream)
{
    for (int sliceIt = firstSlice; sliceIt != lastSlice; sliceIt++)
    {
        PsInput output;
        output.sliceIndex = sliceIt;
        for( int v = 0; v < 3; v++ )
        {
            output.hPos = mul(input[v].worldPos, sliceViewProjMatrices[sliceIt]);
            output.texCoord = input[v].texCoord;
            stream.Append(output);
        }
        stream.RestartStrip();
    }
}
```

Geometry shader cloning

- Benefits
  - Single shadowmap draw call per object even if object intersects multiple slices
- Drawbacks
  - GS data amplification can be expensive
  - Not compatible with instancing
  - Multiple GS permutations for # of slices
  - Fixed max number of slices in shader

Instancing GS method

- Render multiple instances for objects that intersects multiple slices
  - Combine with ordinary instancing that you were already doing
- Store slice index per object instance
  - In vertex buffer, cbuffer or **tbuffer**
  - Together with the rest of the per-instance values (world transform, colors, etc)
- Geometry shader only used for selecting output slice

Instancing geometry shader

```
struct GsInput
{
    float4 hPos : SV_POSITION;
    float2 texCoord : TEXCOORD0;
    uint sliceIndex : TEXCOORD1;  // from VS vbuffer or tbuffer (tbuffer faster)
};

struct PsInput
{
    float4 hPos : SV_POSITION;
    float2 texCoord : TEXCOORD0;
    uint sliceIndex : SV_RenderTargetArrayIndex;
};

[maxvertexcount(3)]
void main(triangle GsInput input[3],
          inout TriangleStream<PsInput> stream)
{
    PsInput output;
    output.sliceIndex = input[v].sliceIndex;
    output.hPos = input[v].hPos;
    output.texCoord = input[v].texCoord;
    stream.Append(output);
}
```

Instancing geometry shader

- Benefits
  - Works together with ordinary instancing
  - Single draw call per shadow object *type!*
  - Arbitrary number of slices
  - Fixed CPU cost for shadowmap rendering
- Drawbacks
  - Increased shadowmap GPU time
    - Radeon 4870x2: ~1% (0.7–1.3%)
    - Geforce 280: ~5% (1.9–18%)
  - Have to write/generate GS permutation for every VS output combination

Shadow Flickering

- Causes
  - Lack of high-quality filtering (>2x pcf)
  - Moving light source
  - Moving player view
  - Rotating player view
  - Changing field-of-view
- With a few limitations, we can fix these for static geometry

Flickering movies

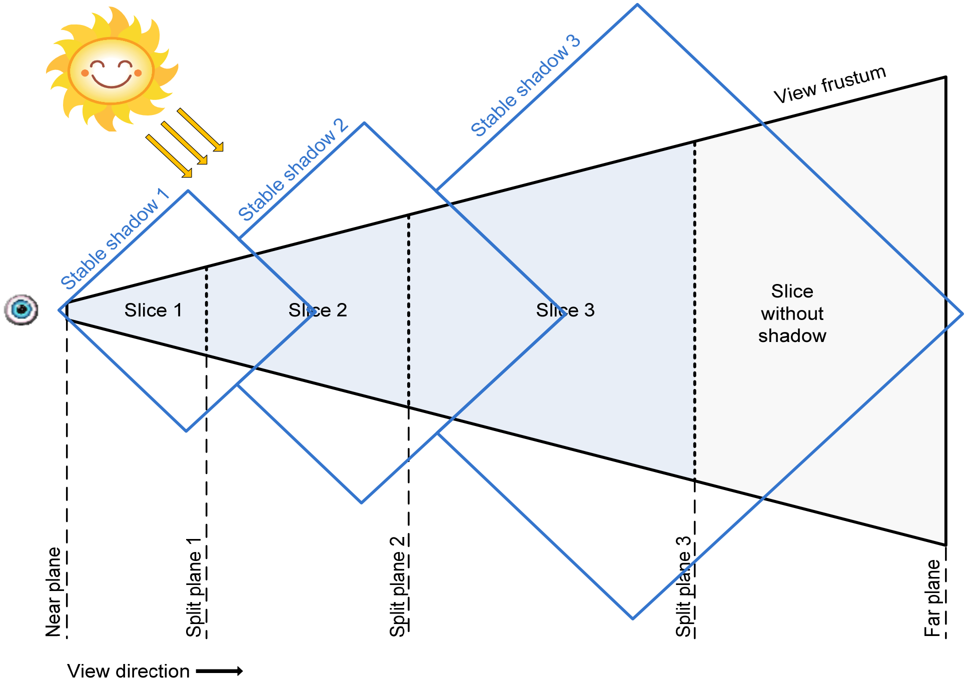
<show>

</show>

Stabilization (1/2)

» Orthographic views

- Scene-independent
- Make rotationally invariant = Fixed size



Stabilization (2/2)

- Round light-space translation to even texel increments

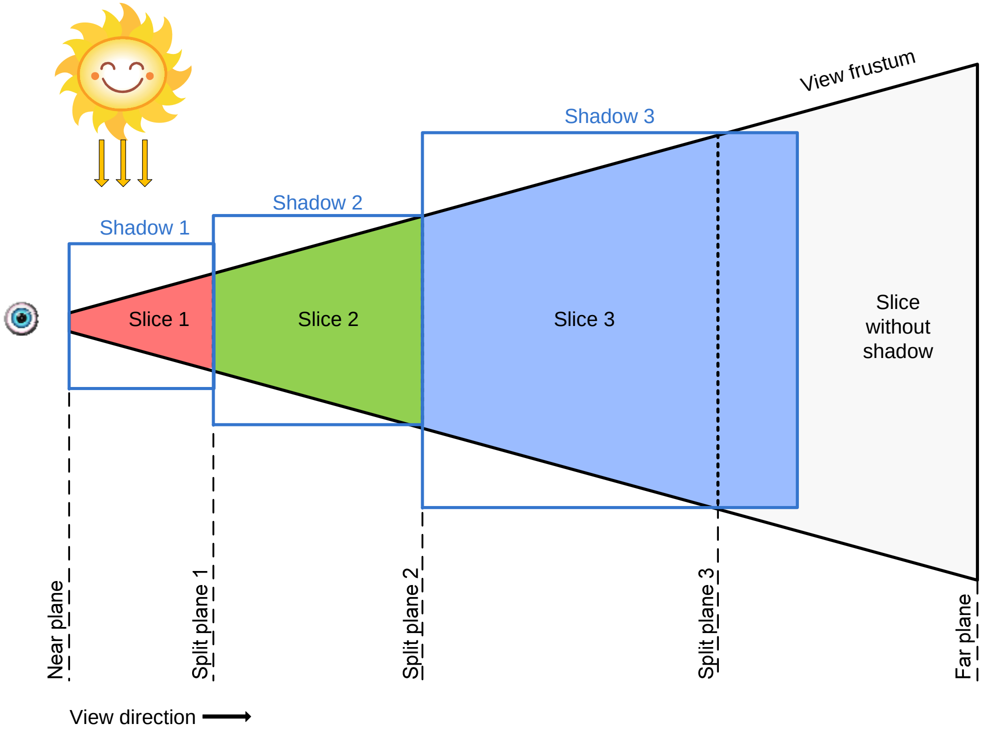
```
float f = viewSize / (float)shadowmapSize;
translation.x = round(translation.x/f) * f;
translation.y = round(translation.y/f) * f;
```

- Still flickers on FOV changes & light rotation
    - So don’t change them ☺

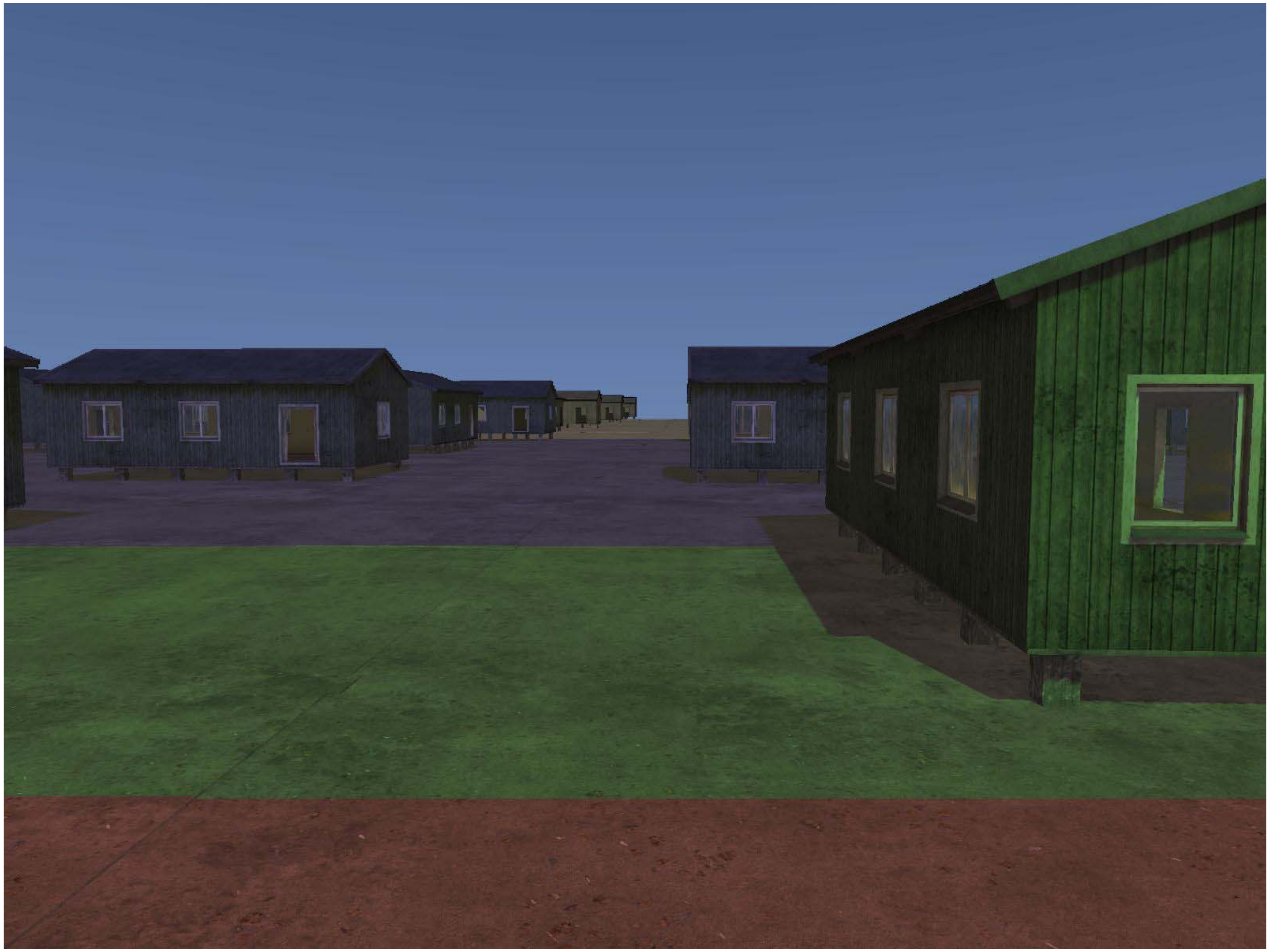
Scene rendering

» Slice selection methods

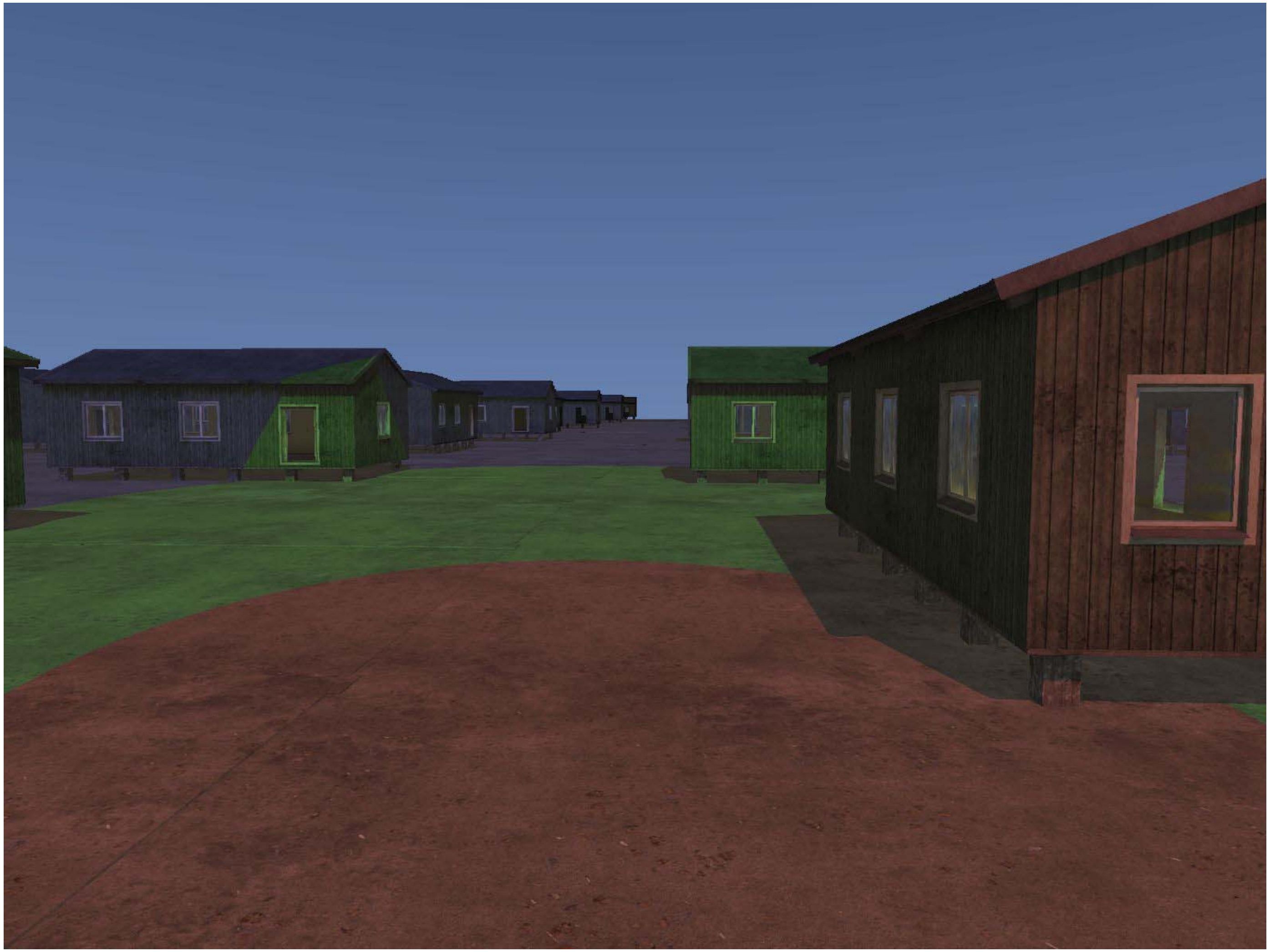
- Slice plane (viewport depth)
- Bounding sphere (Killzone 2 [2])
- Bounding box (BFBC / Frostbite)



Slice plane selection



Bounding sphere selection



Bounding box selection

Shadowmap texture array sampling shader

```
float sampleShadowmapCascadedBox3Pcf2x2(
    SamplerComparisonState s,
    Texture2DArray tex,
    float4 t0,    // t0.xyz = [-0.5,+0.5]  t0.w == 0
    float4 t1,    // t1.xyz = [-0.5,+0.5]  t1.w == 1
    float4 t2)    // t2.xyz = [-0.5,+0.5]  t2.w == 2
{
    bool b0 = all(abs(t0.xyz) < 0.5f);
    bool b1 = all(abs(t1.xyz) < 0.5f);
    bool b2 = all(abs(t2.xy) < 0.5f);

    float4 t;
    t = b2 ? t2 : 0;
    t = b1 ? t1 : t;
    t = b0 ? t0 : t;
    t.xyz += 0.5f;

    float r = tex.SampleCmpLevelZero(s, t.xyw, t.z).r;
    r = (t.z < 1) ? r : 1.0;
    return r;
}
```

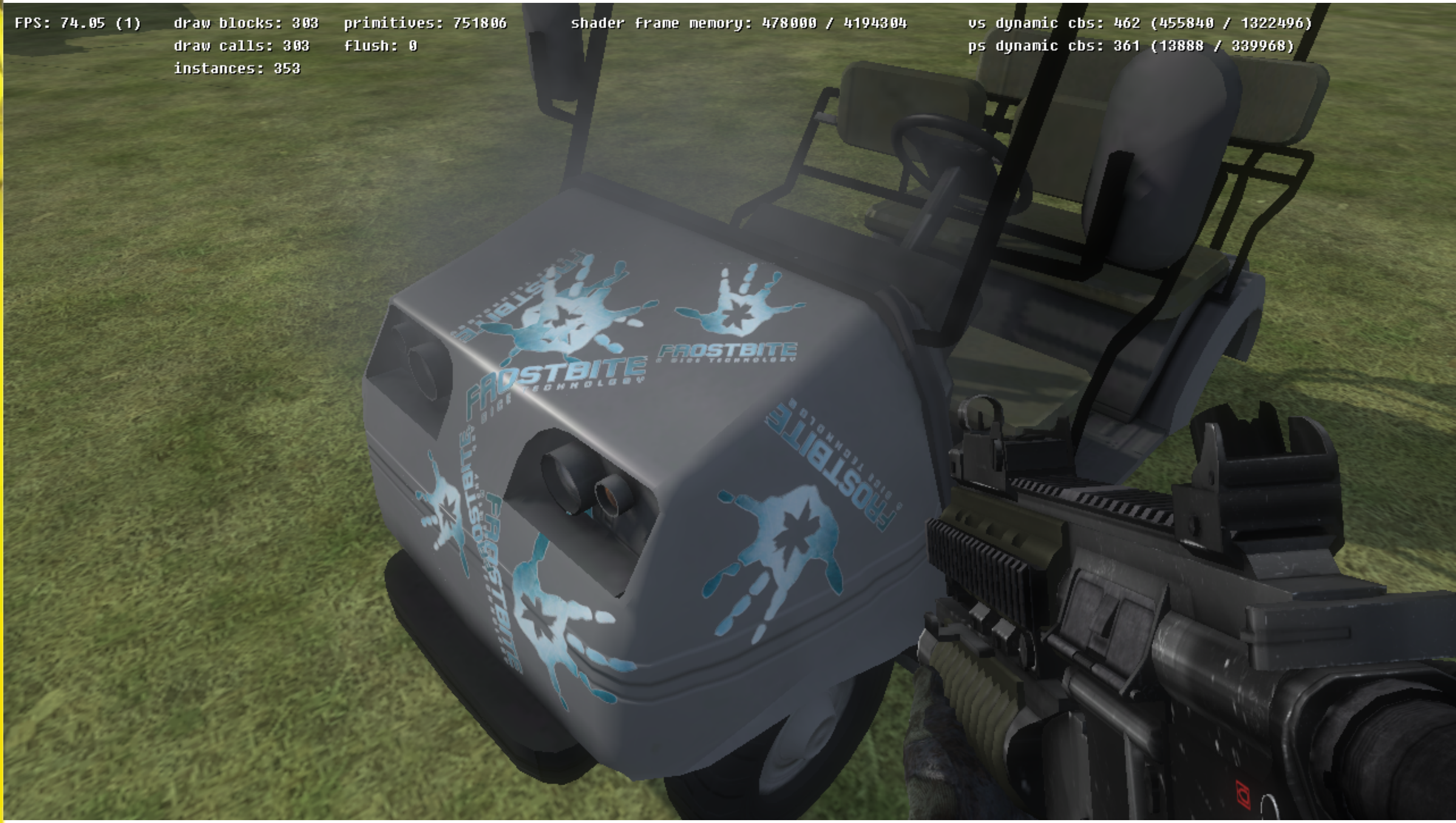
Conclusions

- Stabilization reduces flicker
  - With certain limitations
- Bounding box slice selection maximizes shadowmap utilization
  - Higher *effective* resolution
  - Longer *effective* shadow view distance
  - Good fit with stabilization
- Fewer draw calls by rendering to texture array with instancing
  - Constant CPU rendering cost regardless of number of shadow casting objecs & slices
  - At a small GPU cost

Decal generation using the Geometry Shader and Stream Out

Daniel Johansson

What is a Decal?



```
FPS: 74.05 (1)    draw blocks: 303   primitives: 751806       shader frame memory: 478000 / 4194304       vs dynamic cbs: 462 (455840 / 1322496)
                  draw calls: 303    flush: 0                                                             ps dynamic cbs: 361 (13888 / 339968)
                  instances: 353
```

Overview

- Problem description
- Solution
- Implementation
- Results
- Future work

- Q & A for both parts

Problem description

- Decals were using physics collision meshes
  - Caused major visual artifacts
  - We need to use the actual visual meshes
- Minimize delay between impact and visual feedback
  - Important in fast paced FPS games

Problem description

- Already solved on consoles using shared memory (Xbox360) and SPU jobs (PS3)
- No good solution existed for PC as of yet
  - Duplicating meshes in CPU memory
  - Copying to CPU via staging resource

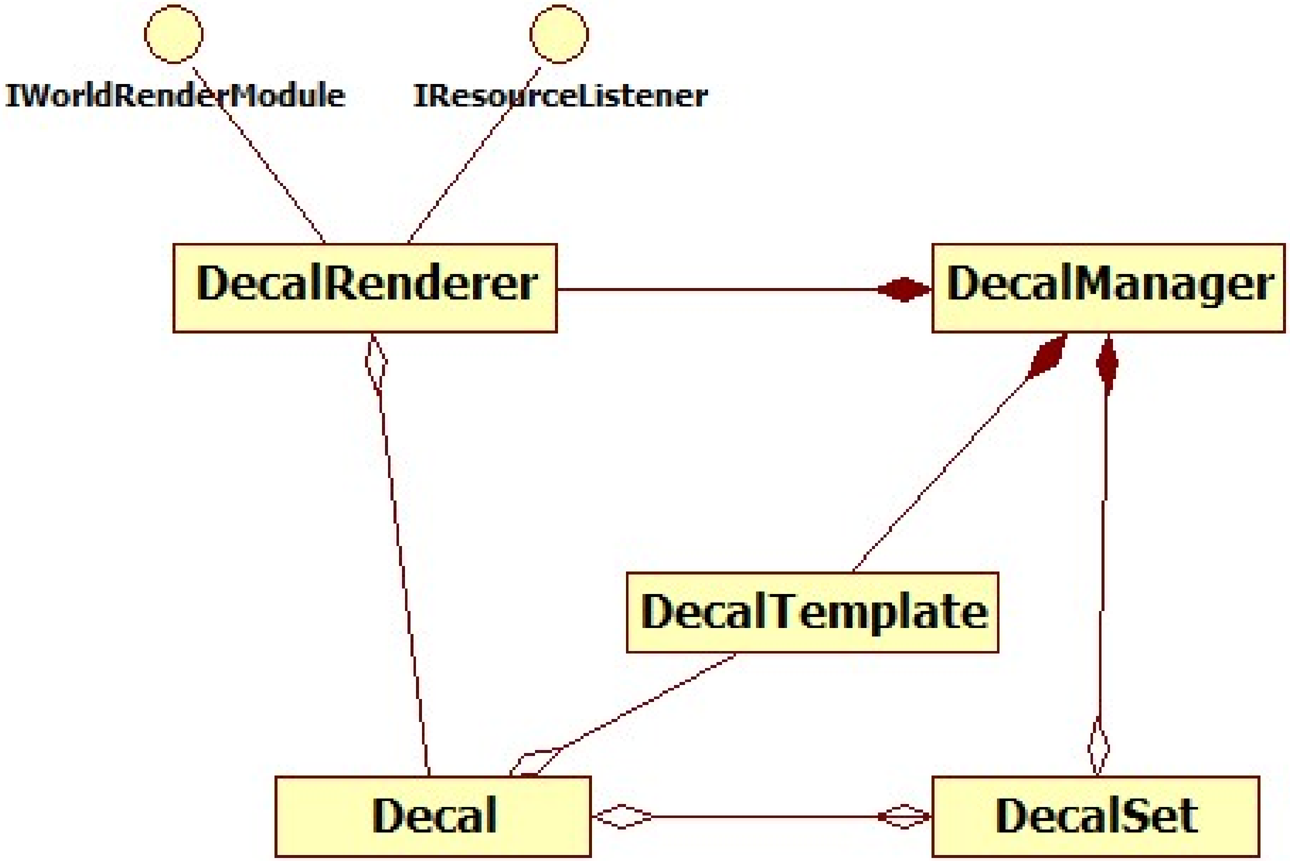
Solution

- Use the Geometry shader to **cull** and extract decal geometry
    - From mesh vertex buffers in GPU RAM
- Stream out the decal geometry to a vertex ring buffer
- Use clip planes to **clip** the decals when drawing

Solution

- Allows us to transfer UV-sets from the source mesh to the decal
- Takes less vertex buffer memory than older method
  - Due to use of clipplanes instead of manual clipping

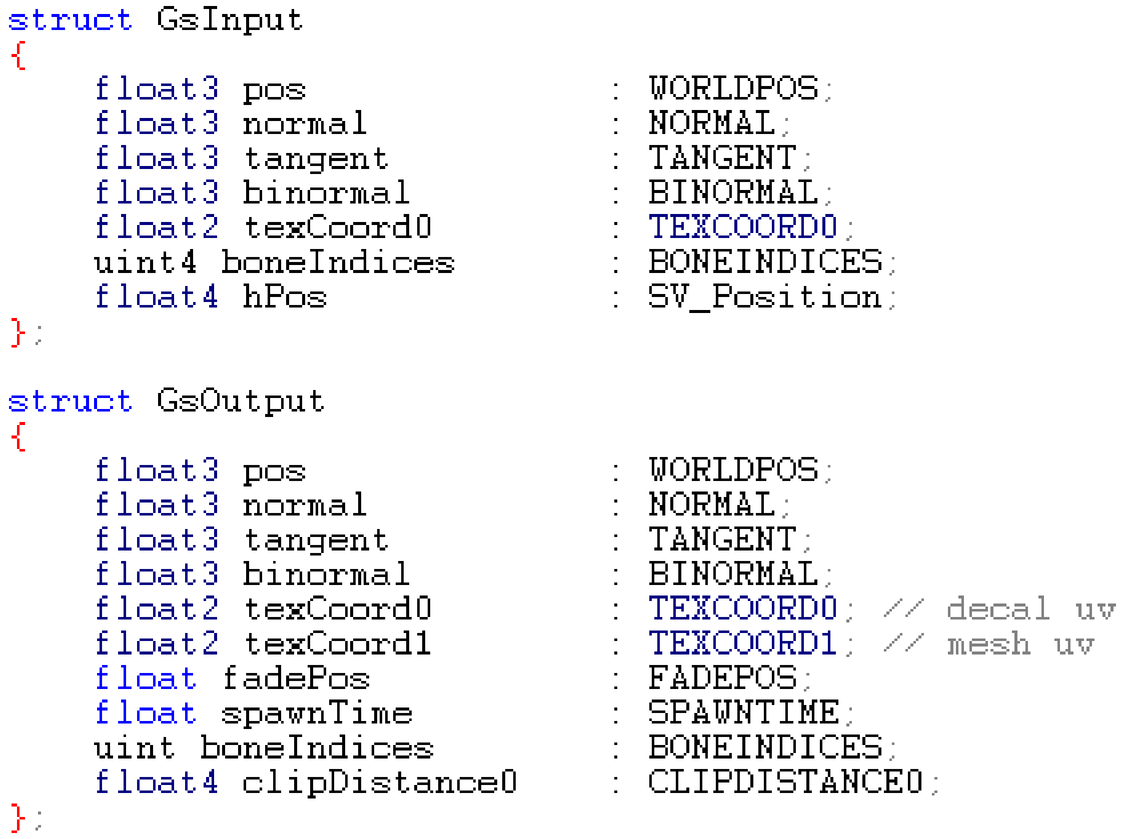
Implementation – UML



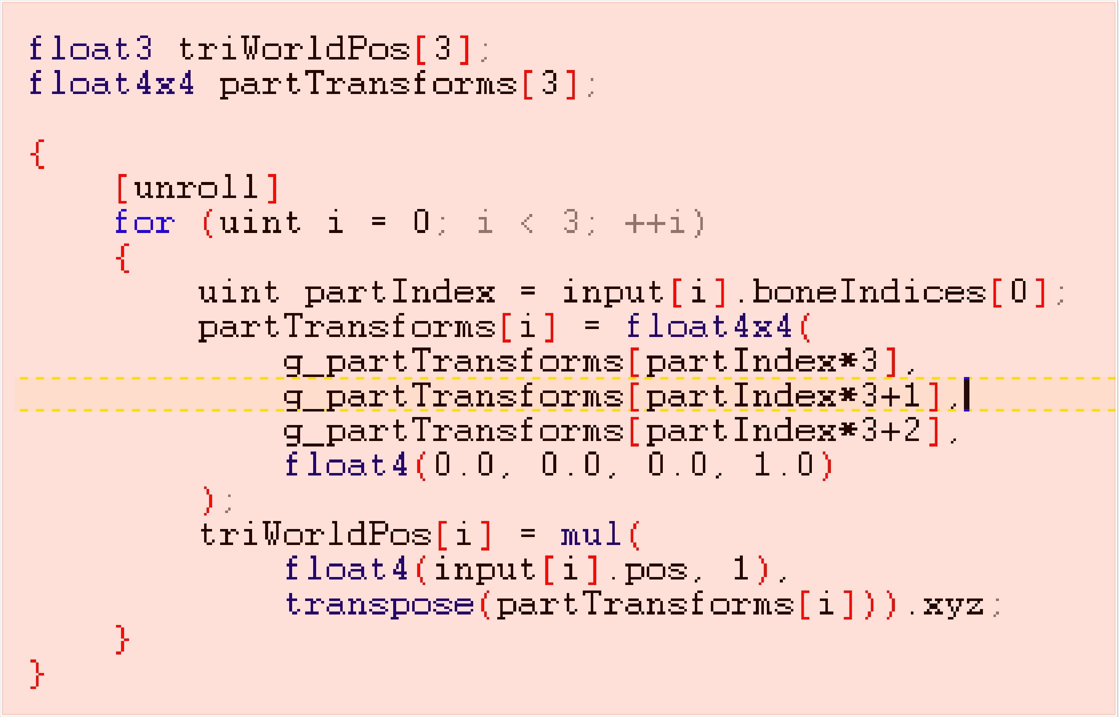
Implementation – Geometry Shader

- GS pass ”filters” out intersecting geometry from the input mesh
  - Also performs a number of data transforms
- GS pass parameters
  - Decal transform, spawn time, position in vertex buffer etc
- Let’s take a closer look at the GS code!

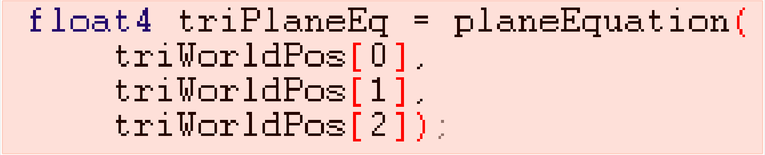
Geometry Shader – in/output



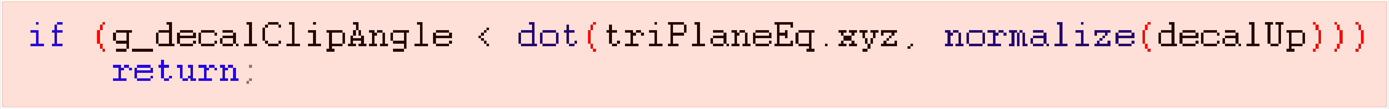
```
struct GsInput
{
    float3 pos                  : WORLDPOS;
    float3 normal               : NORMAL;
    float3 tangent              : TANGENT;
    float3 binormal             : BINORMAL;
    float2 texCoord0            : TEXCOORD0;
    uint4 boneIndices           : BONEINDICES;
    float4 hPos                 : SV_Position;
};

struct GsOutput
{
    float3 pos                  : WORLDPOS;
    float3 normal               : NORMAL;
    float3 tangent              : TANGENT;
    float3 binormal             : BINORMAL;
    float2 texCoord0            : TEXCOORD0; // decal uv
    float2 texCoord1            : TEXCOORD1; // mesh uv
    float fadePos               : FADEPOS;
    float spawnTime             : SPAWNTIME;
    uint boneIndices            : BONEINDICES;
    float4 clipDistance0        : CLIPDISTANCE0;
};
```



```
float3 triWorldPos[3];
float4x4 partTransforms[3];

{
    [unroll]
    for (uint i = 0; i < 3; ++i)
    {
        uint partIndex = input[i].boneIndices[0];
        partTransforms[i] = float4x4(
            g_partTransforms[partIndex*3],
            g_partTransforms[partIndex*3+1],
            g_partTransforms[partIndex*3+2],
            float4(0.0, 0.0, 0.0, 1.0)
        );
        triWorldPos[i] = mul(
            float4(input[i].pos, 1),
            transpose(partTransforms[i])).xyz;
    }
}
```



```
float4 triPlaneEq = planeEquation(
    triWorldPos[0],
    triWorldPos[1],
    triWorldPos[2]);
```



```
if (g_decalClipAngle < dot(triPlaneEq.xyz, normalize(decalUp)))
    return;
```

Transform geometry to world space
Setup plane equation
Discard if angle too big

```
float3 triDecalLocalBoxMin = float3(FLT_MAX, FLT_MAX, FLT_MAX);
float3 triDecalLocalBoxMax = float3(-FLT_MAX, -FLT_MAX, -FLT_MAX);
{
    [unroll]
    for (uint i = 0; i < 3; ++i)
    {
        float3 triDecalLocalPos;
        triDecalLocalPos = triWorldPos[i];
        triDecalLocalPos = mul(
            float4(triDecalLocalPos, 1),
            g_decalObjectToWorldInv).xyz;
        triDecalLocalPos = mul(
            float4(triDecalLocalPos, 1),
            g_decalTransformInv).xyz;

        triDecalLocalBoxMin = min(
            triDecalLocalBoxMin,
            triDecalLocalPos);
        triDecalLocalBoxMax = max(
            triDecalLocalBoxMax,
            triDecalLocalPos);
    }
}

float size = g_decalSize;
float radius = sqrt(2*size*size);

if (!intersectSphereAABB(
        c_zero,
        radius,
        triDecalLocalBoxMin,
        triDecalLocalBoxMax))
    return;
```

Transform triangle to decal object space

Calculate triangle bbox

Do a sphere/box test

Code break

- __asm { int 3; }

```
float3 decalWorldPos[4];
{
    [unroll]
    for (uint i = 0; i < 4; ++i)
    {
        decalWorldPos[i] = c_quadVertices[i];
        decalWorldPos[i] = mul(
            float4(decalWorldPos[i], 1),
            (float4x3)g_decalTransform);
        decalWorldPos[i] = mul(
            float4(decalWorldPos[i], 1),
            (float4x3)g_decalObjectToWorld);
    }
}
```

Setup clip planes from decal quad edges (cookie cutter)

Calculate tangent and binormal

```
float4 clipPlanes[4];
{
    [unroll]
    for (uint i = 3, j = 0; j < 4; i = j++)
    {
        float3 decalEdge = decalWorldPos[j] - decalWorldPos[i];
        float3 clipNormal = normalize(cross(decalEdge, decalUp));
        clipPlanes[i] = planeEquation(-clipNormal, decalWorldPos[i]);
    }
}
```

```
float4 decalAxisX;
float4 decalAxisY;
decalAxisX.xyz = decalWorldPos[3] - decalWorldPos[0];
decalAxisY.xyz = decalWorldPos[1] - decalWorldPos[0];
decalAxisX.w = dot(decalAxisX.xyz, decalAxisX.xyz);
decalAxisY.w = dot(decalAxisY.xyz, decalAxisY.xyz);

float3 decalTangent = -normalize(decalAxisX.xyz);
float3 decalBinormal = -normalize(decalAxisY.xyz);

if (dot(cross(triPlaneEq.xyz, decalTangent), decalBinormal) < 0.0f)
    decalTangent = -decalTangent;
```

```
[unroll]
for (uint i = 0; i < 3; ++i)
{
    GsOutput decalPolygon;

    decalPolygon.pos = input[i].pos;
    decalPolygon.normal = input[i].normal;
    decalPolygon.tangent = mul(
        float4(decalTangent, 1),
        partTransforms[i]).xyz;
    decalPolygon.binormal = mul(
        float4(decalBinormal, 1),
        partTransforms[i]).xyz;
    decalPolygon.fadePos = g_decalFadePos;
    decalPolygon.spawnTime = g_decalSpawnTime;
    decalPolygon.texCoord0 = projectPlanar(
        triWorldPos[i],
        decalWorldPos[0],
        decalAxisX,
        decalAxisY);
    decalPolygon.texCoord1 = input[i].texCoord0;
    decalPolygon.boneIndices =
        input[i].boneIndices[3] << 24 |
        input[i].boneIndices[2] << 16 |
        input[i].boneIndices[1] << 8 |
        input[i].boneIndices[0];
    decalPolygon.clipDistance0 = float4(
        distancePlane(clipPlanes[0], triWorldPos[i]),
        distancePlane(clipPlanes[1], triWorldPos[i]),
        distancePlane(clipPlanes[2], triWorldPos[i]),
        distancePlane(clipPlanes[3], triWorldPos[i]));

    TriStream.Append(decalPolygon);
}

TriStream.RestartStrip();
```

Copy mesh texcoords

Calculate clip distances

Append triangle to stream

Apply planar projection

Transform tangents/ binormals to world (object space)

Geometry Shader Performance

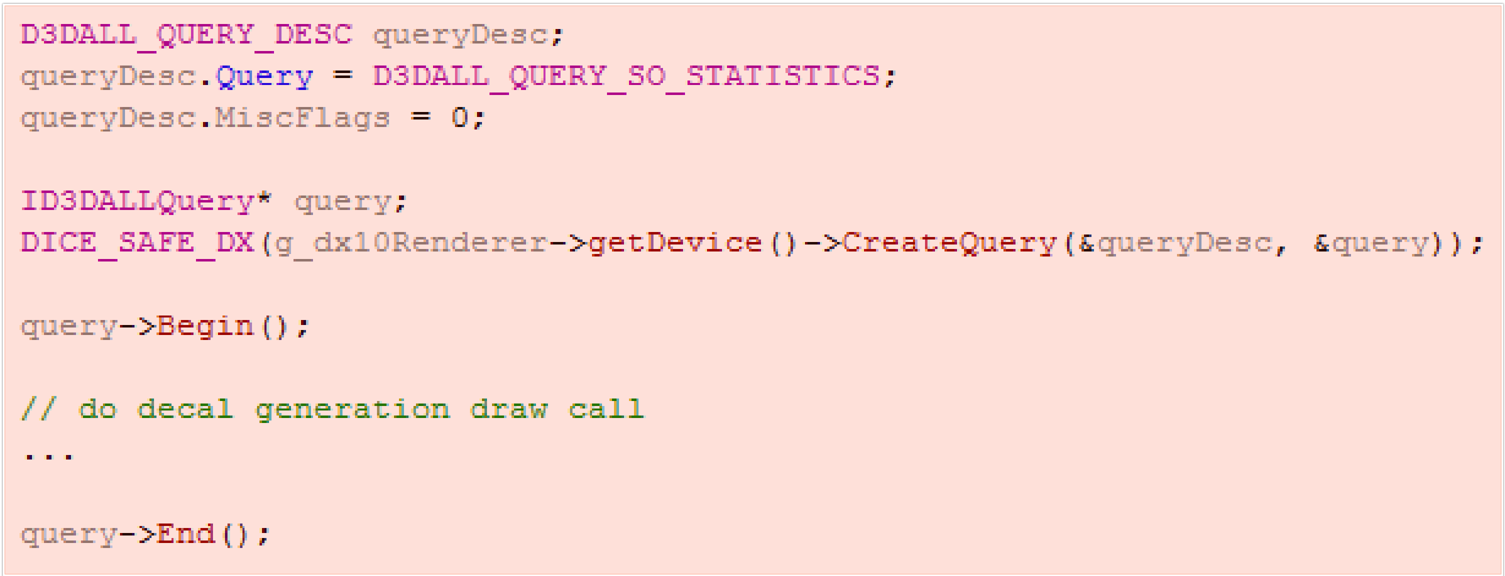
- Complex GS shader - ~260 instructions
  - Room for optimization
- GS draw calls usually around 0.05-0.5 ms
  - Depending on hardware of course
- Per frame capping/buffering used to avoid framerate drops

Implementation – Buffer usage

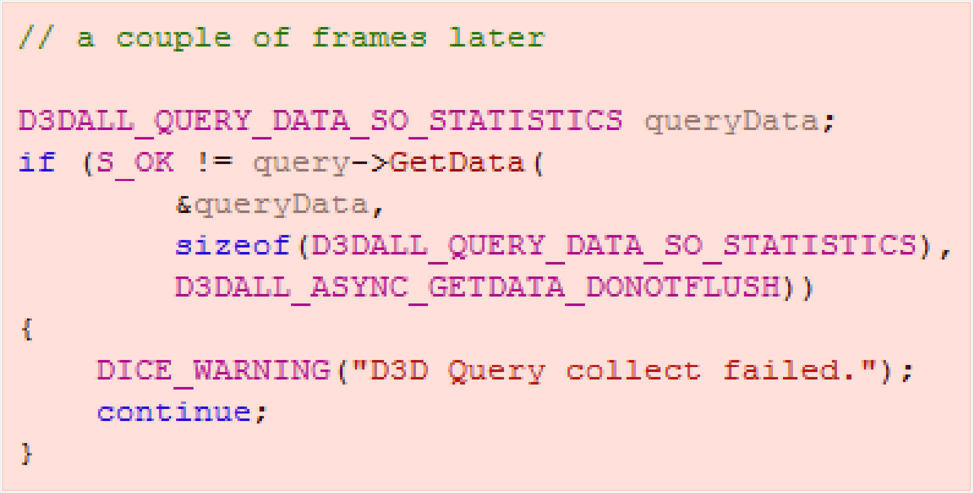
- One decal vertex buffer used as a ring buffer
- One index buffer – dynamically updated each frame
- Decal transforms stored on the CPU (for proximity queries)

Implementation – Queries

- Grouped together with each decal generation draw call
- Result is used to ”commit” decals into their decal sets or discard them if no triangles were written



```
D3DALL_QUERY_DESC queryDesc;
queryDesc.Query = D3DALL_QUERY_SO_STATISTICS;
queryDesc.MiscFlags = 0;

ID3DALLQuery* query;
DICE_SAFE_DX(g_dx10Renderer->getDevice()->CreateQuery(&queryDesc, &query));

query->Begin();

// do decal generation draw call
...

query->End();
```



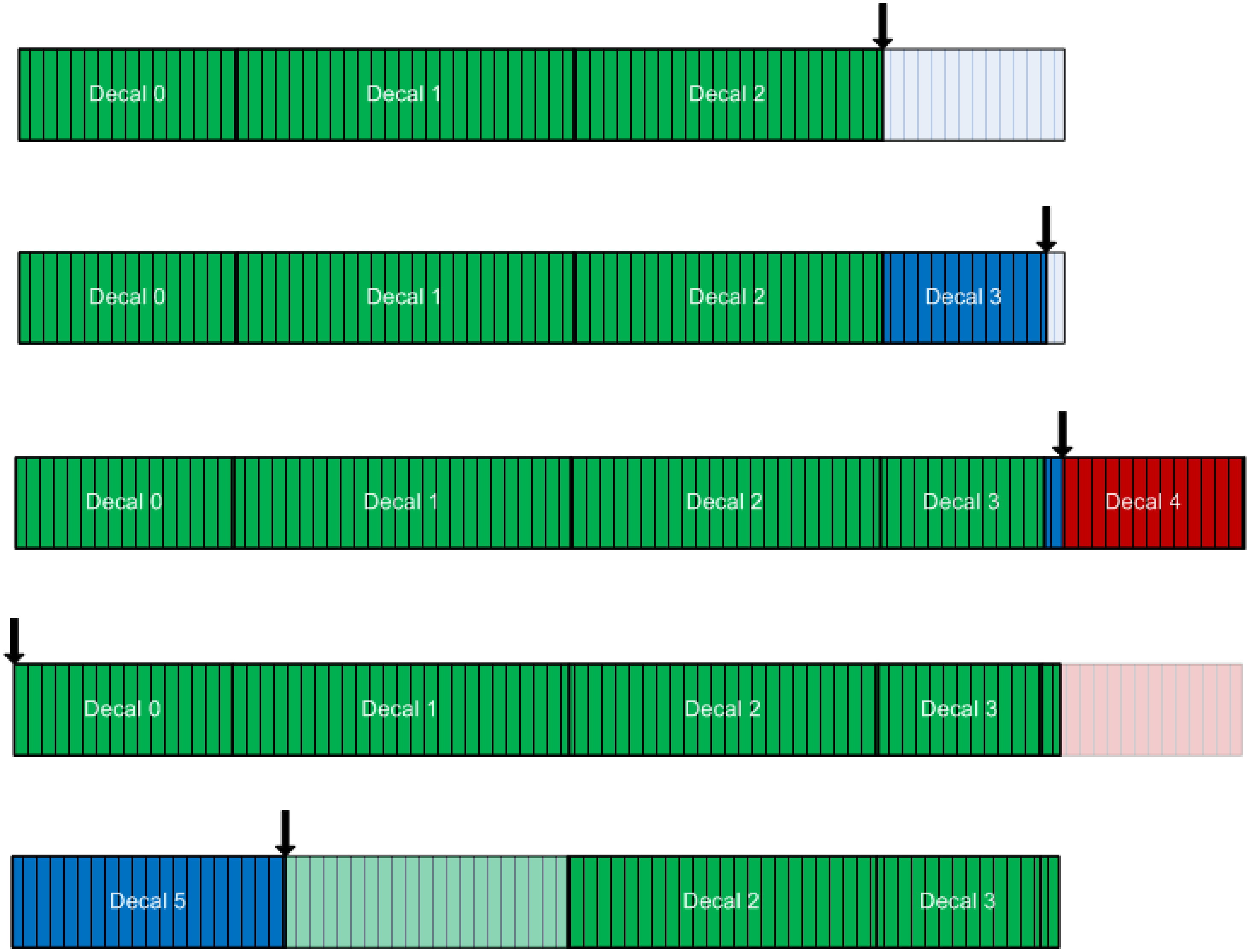
```
// a couple of frames later

D3DALL_QUERY_DATA_SO_STATISTICS queryData;
if (S_OK != query->GetData(
        &queryData,
        sizeof(D3DALL_QUERY_DATA_SO_STATISTICS),
        D3DALL_ASYNC_GETDATA_DONOTFLUSH))
{
    DICE_WARNING("D3D Query collect failed.");
    continue;
}
```

Implementation – Queries

- Issues
  - Buffer overflows
  - Syncronization
- No way of knowing **where** in the buffer vertices were written
  - Only have NumPrimitivesWritten and PrimitiveStorageNeeded

Implementation – Queries

- Solution: When an overflow is detected the buffer is wrapped around.
    - If any decals are partially written they are committed, otherwise discarded.


Decal 0
Decal 1
Decal 2
Decal 0
Decal 1
Decal 2
Decal 3
Decal 0
Decal 1
Decal 2
Decal 3
Decal 4
Decal 0
Decal 1
Decal 2
Decal 3
Decal 5
Decal 2
Decal 3

Results

Future Work

- Rewrite to make use of DrawAuto()
- Experiment more with material masking possibilites
- Port to DX11 Compute Shader
- Implement GPU-based ray/mesh intersection tests
- SLI/Crossfire

Questions?

Contact:
johan.andersson@dice.se
daniel.johansson@dice.se

igetyourfail.com

References

- [1] Zhang et al. ”Parallel-Split Shadow Maps on Programmable GPUs". GPU Gems 3.
- [2] Valient, Michael. "Stable Rendering of Cascaded Shadow Maps". ShaderX6