
SIGGRAPH2010
The People Behind the Pixels

Destruction Masking in Frostbite 2 using Volume Distance Fields

Robert Kihl
DICE

What is destruction masking?

In the frostbite game engine one important feature is destructable environments. We have developed several systems that, when put together, lets us do belivable destructable environments. One of these systems is destruction masking.

Here is an example to illustrate this.

When destroying a part of this house, we begin by removing a piece of the geometry.

We then add detail meshes around the destroyed section.

And the final step is to add the destruction mask.

We can go on and let the player destroy other parts of the house and the detail meshes and mask will be added for those parts as well. We also add particle effects and mesh debris for a more dramatic look.

Agenda

- History – Frostbite 1
- Volume Distance Fields
- Optimisation Techniques
- Results

1. How we did destruction masking in the first frostbite engine.
2. How we use volume distance fields.
3. Two techniques for drawing the destruction mask efficiently.
4. Result screenshots and performance numbers.

Destruction Masking in Frostbite 1.0

- Used in BFBC, BF1943 and BFBC2
- Good visual quality
- Time consuming workflow: main issue to address
  - Requires UV mapping mask for each destructable part

Destruction masking
In BFBC2

The frostbite 1 engine has been around for a few years now and we have shiped three titels on it. The first was Battlefield Bad Company, followed by Battlefield 1943 and the latest was Battlefield Bad Company 2.

All these titles used a system for destruction masking. This system had good visual quality and decent performance. The main issue we wanted to address was the time consuming workflow which involved creating UV maps for each destructable part. Although giving the artist full control of the mask, we felt that speeding up the workflow, to allow our artists to create more content, was more important.

Frostbite 2.0

- Focus on workflows
- Frostbite 2 uses deferred rendering
- Target platforms PS3, Xbox 360 and DirectX10/11

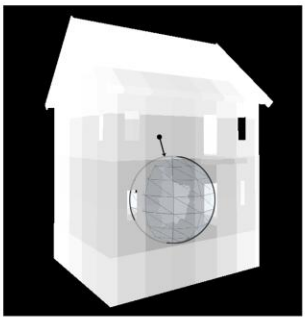
In frostbite 2 our main focus has been speeding up workflows and iteration times.

We have also switch to using deferred rendering, which opens up for some rendering techniques we couldn't do before.

The target platforms for frostbite 2 are PS3, xbox 360 and directx 10 and 11

Signed Volume Distance Field

- Spheres are placed on the geometry
  - Marks out location for the destruction mask
- Distance field computed from the group of spheres
  - Stored in a volume texture
  - Low resolution: ~2 meters/pixel
- One texture per masked geometry



How do we use signed volume distance fields as one step in creating the destruction mask.

We start by marking out the areas where we want the mask to appear on the target geometry. We do this by placing spheres on the target locations.

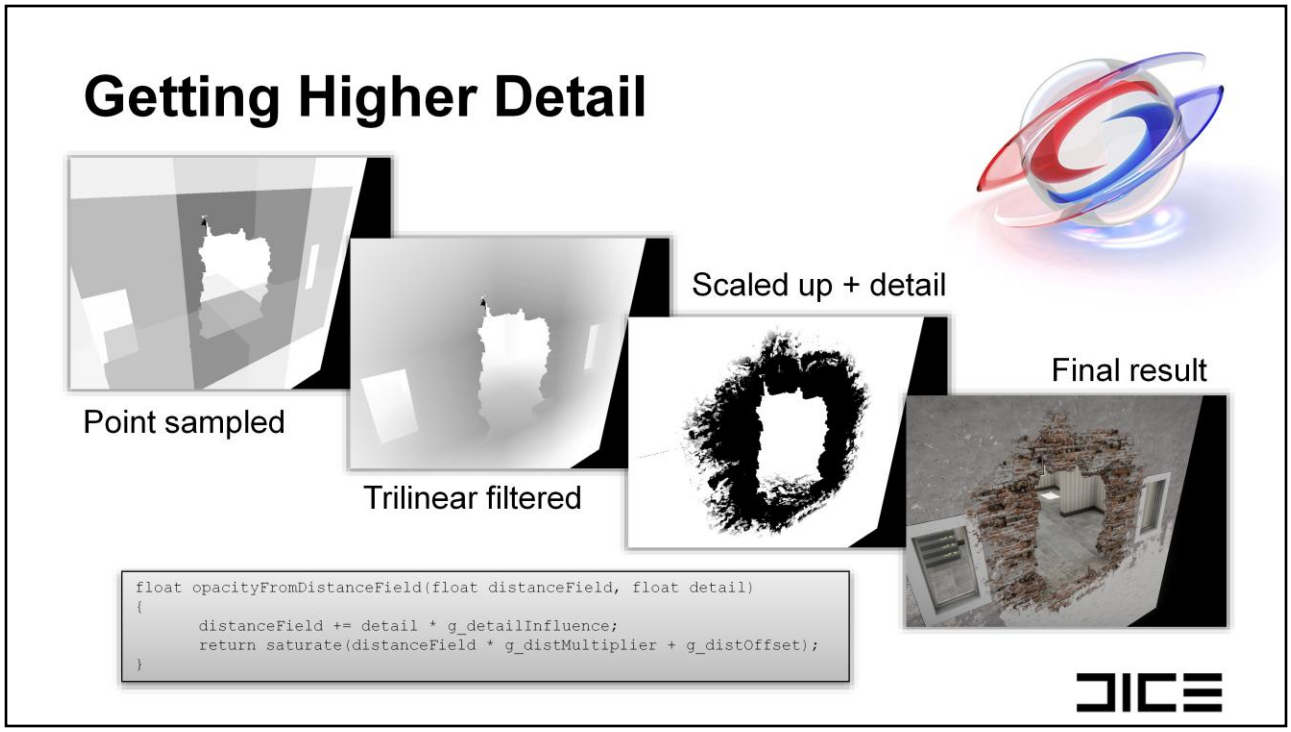
Next we take this group of spheres and compute a signed volume distance field. We store the distance field in a volume texture so that each texel contains the shortest distance to any sphere.

Using spheres makes computing the distance field a fast operation.

We can also get away with a rather low resolution distance field of approximatly 2 meters per pixel.

The volume texture is local to the target geometry and placed to tightly encapsulate it. So we have one volume textures for each destruction masked geometry in the scene.

The spheres are authored by the artist by first auto generating a set of spheres so that each destructable part has one. The artist can then go on and move/scale the spheres as well as add new ones. For example, to cover a hole that is not close to square the artist might want to put several smaller spheres to get a better fit.



The next step is to use the low resolution distance field to get a detailed destruction mask.

1. Here we can see a destroyed part of a building wall and the distance field around it drawn with point sampling.

2. Turning on trilinear filtering gives a smooth gradient, the surface of the destruction mask is at gradient value 0.5.

3. By multiplying and adding an offset we can find the surface of the destruction mask. Additionally we project a detail texture onto the geomery and use it to offset the distance field. This breaks up the low frequency shape of the mask and creates a more interesting look.

4. Final result with texturing.

Volume Texture Considerations

- Xbox 360 volume texture requirement:
  - Dimension multiple of $32 \times 32 \times 4$
- Many small textures will waste memory
- Use texture atlases
  - One atlas per dimension simplifies packing
  - Need padding to prevent border leakage

One thing to consider when working with small textures on Xbox 360 is that texture dimensions need to be a multiple of the tile size. For 3D textures the tile size is 32x32x4.

Our distance field textures are typically 8x8x8 in dimension which will be a considerable waste of memory if we allocate them one by one and have a lot of them.

Our solution to this is to use texture atlases.

To simplify packing textures into the atlas we keep one atlas per texture dimension.

As always when working with texture atlases we need to add padding between the textures in the atlas to prevent leakage across texture borders. Since we don't need to generate mipmaps for these textures (they are so low resolution that we LOD them before min filtering would be needed) padding only needs to be one pixel wide.

Drawing the Destruction Mask

- Draw together with geometry
  - Optimize with `[branch]` in pixel shader
- Dynamic branching on RSX not efficient
  - 6 cycles hit for branching
  - Coarse segment size (800-1600 pixels)
- Draw mask as Deferred Decal

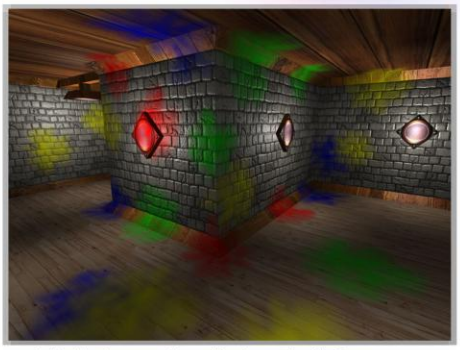
When we draw geometry with destruction masking we first sample the distance field, if we then see that the pixel is outside the mask volume we can use dynamic branching to early out from the rest of the mask computations. This saves us a lot of computation time for pixels not affected by the mask.

However, when we tried this on PS3 we saw that dynamic branching was not very efficient. The RSX has a fixed cost of 6 cycles when introducing a branch in the shader, and for the branch to be taken a group of up to 1600 pixels all need to take the same path.

This led us to to try out a different approach, drawing the mask as a deferred decal.

Deferred Decals

1. Draw scene geometry
2. Draw convex volume around the decal area
3. Fetch depth buffer and convert to local position within volume
4. Use position to look up opacity from e.g. a volume texture
5. Blend on top of geometry



[1] *Volume decals*, by Emil Persson

Deffered decals works simular to how we apply light sources in a deferred renderer.

1. Begin by drawing the main geometry of the scene to fill in the g-buffer.

2. Next draw a convex volume covering the decal area.

3. When drawing the volume fetch depth from the depth buffer and convert to local position within volume.

4. Knowing the position within the volume we can look up the decal opacity by for example sample it from a volume texture.

5. Finally we alpha blend the result to the g-buffer.

One advantage of this method is that the convex volume we draw covers only the decal area, and pixels outside of it are uneffected. So if we can make a good fit for the convex volume for the decal area we can save performance. Many optimization techniques has been developed for applying light sources in a deferred renderer (stenciling, tile classification) and we could potentially use those here as well.

Projecting Detail Textures

- For projected detail/normal texture, tangent vectors are needed
- G-buffer normals are not suitable
  - Contains data from normal maps
- Limited amount of tangent vectors needed
  - Write out index to g-buffer with main geometry
  - Use index to sample texture containing tangent vectors lookup table

The destruction mask replaces the underlying diffuse albedo and normals by projecting detail textures onto the masked geometry.

In order to achive that we need tangent vectors for the projected surface - both for converting world space positions into texture coordinates, and to convert tangent space normals from the detail normal maps to world space normals.

It is appealing to construct tangent vectors from g-buffer normals, but those normals are typically based on normal maps, while we want the normals from the actual geometry.

We took a look at the geometry our artists are creating and saw that there are only a few tangent vectors required for a typical object. With that in mind, we created a look up table with the required tangent vectors and stored them in a texture. Then, when drawing the scene geometry we write out an index to the g-buffer. We sample this index when drawing the deferred volume and use it as an offset into the lookup table texture.

Alpha Blending and g-buffer

- Fixed function blend causes problem when alpha blending to g-buffer
  - Output alpha used both for blending and destination alpha
- Limits us in output alpha for the decals
- Limits us in g-buffer layout
- Good use case for programmable blending! [2]

What about alpha blending to the g-buffer?

One thing we found was that the fixed function blend in current GPU hardware was limiting when alpha blending to the g-buffer.

The problem comes from that we use the g-buffer alpha channel to store data, but from the shader we can't output one alpha to use in the blending equation and a seperate one to store in the render target.

This limits us in what we can output to the alpha channel when drawing the decals.

It also limits us in g-buffer layout. We need to arrange g-buffer data in an order that supports blending of the components we are interested in.

We see a good use case here for programmable blending.

Getting Correct Mipmap Selection

- Artifacts around borders due to quad mipmap selection
- Calculate mip level in shader, sample with tex2Dlod

$$lod = \log_2 \frac{pixelsPerMeter \times \tan(fov) \times distToNearPlane}{screenRes \times \mathbf{v} \cdot \mathbf{n}}$$

Another thing to consider is getting correct mipmap selection for our textures.

When we have discontinuities in the texture coordinates within a quad, the mipmap selection for that quad is going to be wrong.

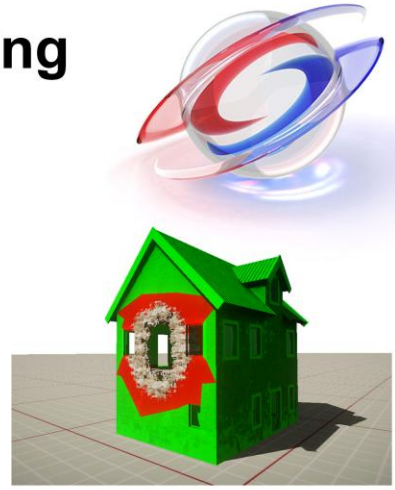
In the zoomed in picture to the right we can see an example of this. Here the texture coordinates for the upper part of the quad samples from the brickwall texture and the bottom part from the floor texture. These texture coordinates are at different locations within one texture atlas and the GPU is going to select the lowest mipmap for this group of pixels.

The way we solved this problem was to manually calculate the correct mip level for each individual pixel and explicitly sample with tex2Dlod.

It is possible to optimize the computation by creating a 2D lookup table texture with the inputs v.n and distToNearPlane.

Distance Field Triangle Culling

- Branch per triangle
  - Test each triangle against distance field
  - Output two index buffers
  - Issue two draw calls
- Uses PLAYSTATION®Edge Geometry Library
- Cache result of culling
  - Update when distance field changes



Red triangles drawn with mask+house shader

Another PS3 specific method we have explored to optimize drawing the destruction mask is distance field triangle culling.

Instead of branching in the fragment shader, which we saw is not very efficient on the PS3, we have moved the branch from the GPU to the SPUs and do the branch per triangle.

The way it works is that for each triangle in the mesh we test it against the distance field to see if it inside or outside the mask volume. We output the triangles into two seperate index buffers and issue two draw calls. The draw call for triangles inside the distance field volume is drawn with a shader that computes the mask while outside triangles are drawn as normal.

To help us do this in an efficient way on the SPUs, we take advantage of the playstation edge geometry library and run it a step of that geometry pipeline.

We only need to do the actual culling against the distance field when the distance field changes, inbetween we can cache the result.

GPU Performance

Technique	ms[1]
Forward rendered	0.77
Forward rendered, with [branch]	0.45
Deferred volume[2]	0.31
Forward rendered, triangle culling	0.20

1) Performance metrics from typical game scenario on PS3
2) Deferred volume drawn as box around masked area

This table shows the GPU cost for drawing the destruction volume in a typical scenario running on PS3. Numbers differ from different scenarios, but they indicate that the method of culling triangles is the best method for GPU performance on PS3.

Deferred volumes didn't give as good performance as we had hoped for. However, we still have optimization methods for this technique left to explore.

Summary

- Volume distance field
  - Defines destruction mask shape
- Triangle culling gives best GPU performance
  - But PS3 specific
- Deferred decals is an interesting technique
  - Could replace traditional decal solutions

We have seen how volume distance fields can be used to define the shape of the destruction mask.

We have explored two different methods for drawing the mask in an efficient way.

Although the method of triangle culling initially has shown us the best GPU performance it's not clear that it's the best solution for other reasons. Because it is a PS3 specific solution it requires us to maintain a seperate code path and the whole solution is more difficult to integrate into the game engine in an elegant way.

Before settling with one solution we need to do more testing and explore further optimization techniques for deferred decals.

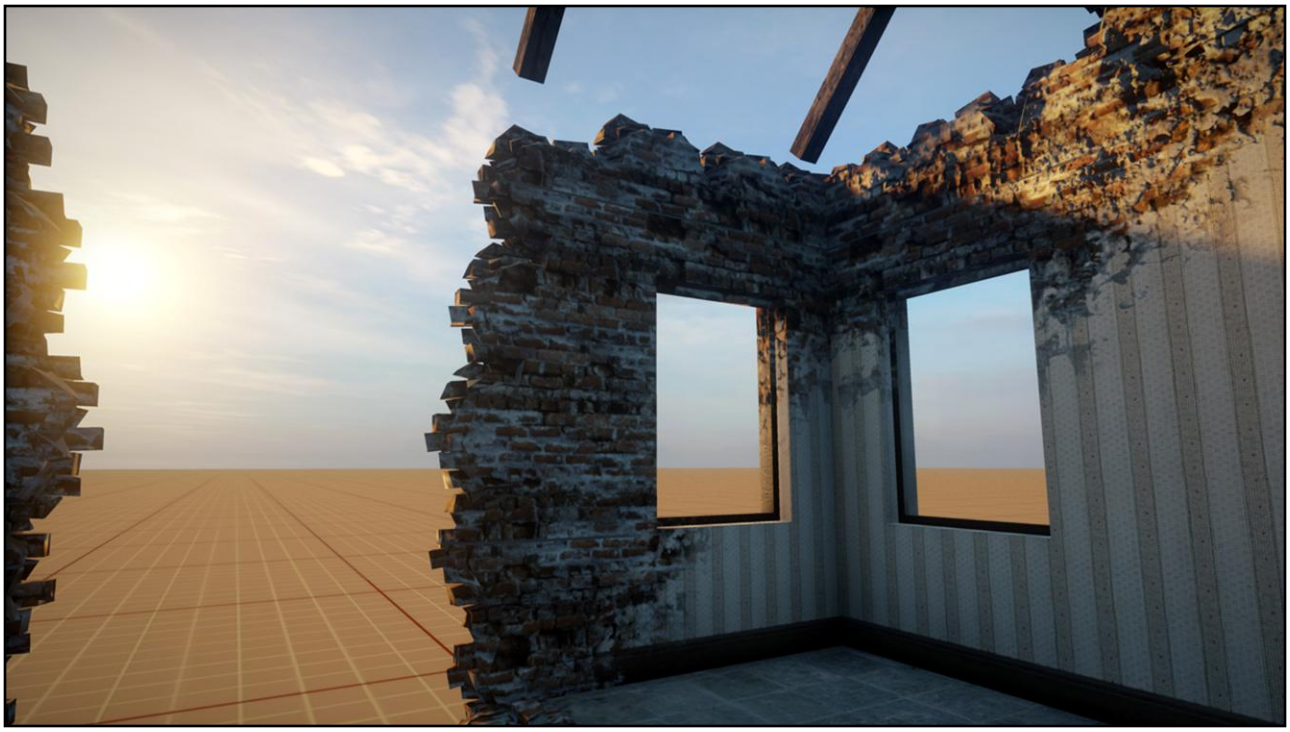
Deferred decals is an interesting technique which potentially has other use cases than destruction masking. For example, traditional decal solutions are often complex platform dependent implementations and suffer from rendering artifacts such as z-fighting. Deferred decals could be a possible replacement.

Results

Here we can see the interior of the house. In the ceiling we use a second detail normal map that we blend in around the edges to make them pop out more.

This next screenshot shows a zoomed out view with more destruction applied to the house.

This is more or less our worse case scenario, where most of the house is covered by the destruction mask.



One more screenshot of the interior of the house.

And here is an outside view.

We can see that we get a lot of detail in the mask from the projected detail textures.

Questions?

Thank you for your attention

robert.kihl@dice.se

More DICE talks: http://publications.dice.se

References

- [1] Emil Persson, *Volume Decals*, http://www.humus.name/index.php?page=3D&ID=83
- [2] Johan Andersson, *Bending the Graphics Pipeline*, Beyond Programmable Shading Course – Siggraph 2010