Paper

GPU Friendly Laplacian Texture Blending by Bartlomiej Wronski

Journal of Computer Graphics Techniques Vol. 14, No. 1, 2025

https://jcgt.org/published/0014/01/02/

https://research.nvidia.com/labs/rtr/publication/wronski2025laplacian/

Problem

Given 3 textures:

- A and B of the same type (albedo, normals, whatever)
- M is the blending mask

How do you blend A and B according to M in a natural way?

Insights

Each mask blur level works best with a certain image feature level

A sharp mask works best for nicely blending high-frequency details while a blurrier mask work best for nicely blending low-frequency details. Maybe we should focus on splitting up frequencies into bands that are each blended in separately with a mask that's been blurred appropriately?

The difference between successive mip levels of a texture are different frequency bands

If we only use 4 mip levels, we get 3 frequency bands:

- `M(0) - M(1)` gives us a high-frequency signal.
- `M(1) - M(2)` gives us a medium-frequency signal.
- `M(2) - M(3)` gives us a low-frequency signal.

High/medium/low is very relative. What does matter is that the frequencies are nicely split.

Collapsing a Laplacian pyramid of a given image gives us the original image back

If we follow these steps for a given image:

- Build a Gaussian pyramid of an image:
  - `G(0)` is the orignal image
  - `G(i + 1)` is more blurry than `G(i)`.
- Build a Laplacian pyramid based on that: `L(i) = G(i) - G(i + 1)`.

- Collapse the Laplacian against a Gaussian base.

Then we have the original image back.

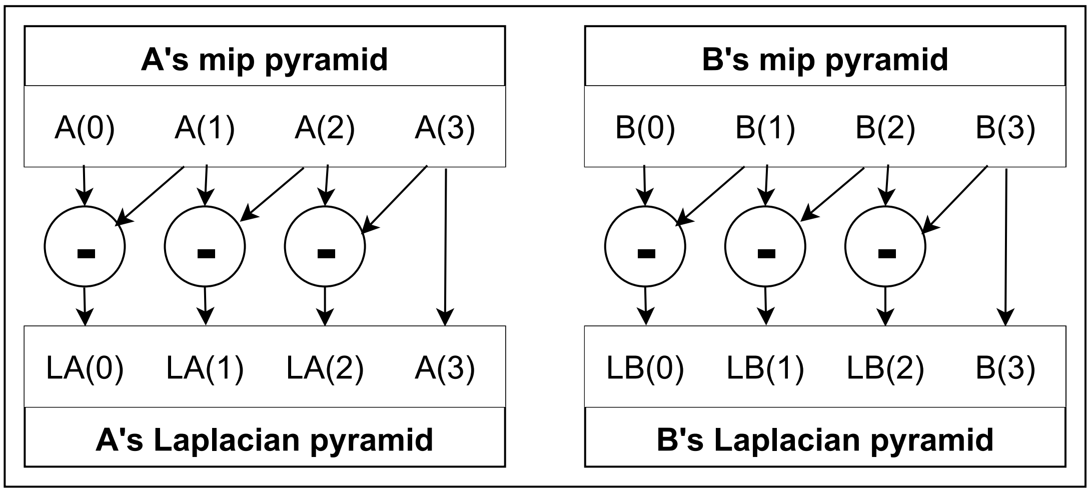
Let's say we use 4 Laplacian levels, here's what happens during collapse:

```
LC4 = G(4) + L(3) + L(2) + L(1) + L(0)
    = G(4) + (G(3) - G(4)) + (G(2) - G(3)) + (G(1) - G(2)) + (G(0) - G(1))
    = G(0)
```

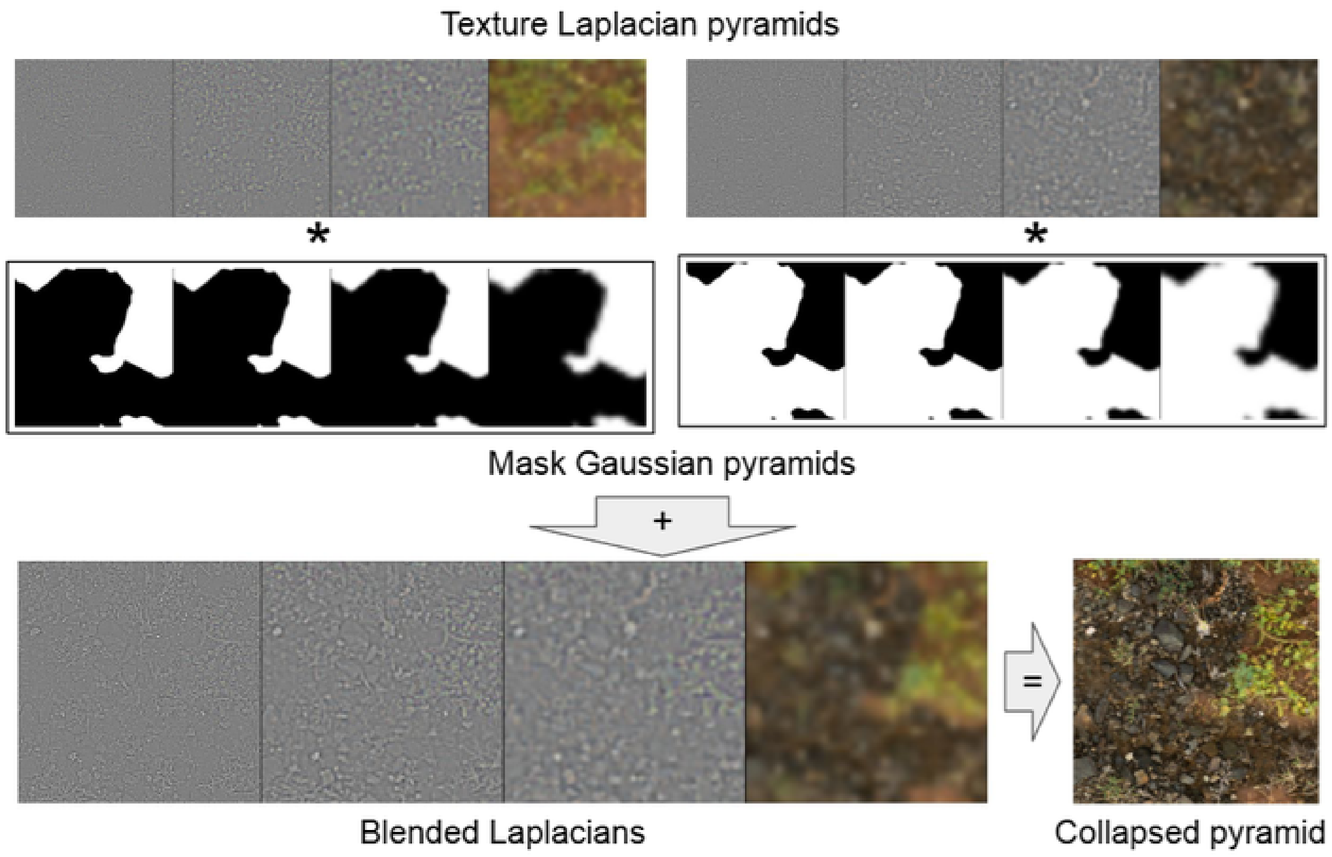
It cancels out. Now of course in practice we don't have infinite precision but mathematically it works out.

Method overview

Generate the Laplacian pyramids of both textures:



Compute the final blended image based on both Laplacian pyramids:



This second image is from the *GPU Friendly Laplacian Texture Blending* paper by Bartlomiej Wronski.

The mask-weighted per-frequency-band contributions of both images are computed separately and then combined into a single Laplacian pyramid, which then gets collapsed to produce the final blended image.

Algorithm

Prerequisites:

- Pick a number of bands called N, let's say 4.
- Generate standard mip pyramids for A, B and M. At least `N + 1` levels are needed.

```
// mask values: 1 picks A, 0 picks B
float4 Blend(Texture2D A, Texture2D B, Texture2D M, SamplerState sampler, float2
tc)
{
    const uint N = 4; // number of Laplacian levels

    float4 mipsA[N + 1];
    float4 mipsB[N + 1];
    float4 mipsM[N + 1];
    for(uint i = 0; i < N + 1; i++)
    {
        float mipLevel = float(i);
        mipsA[i] = A.SampleLevel(sampler, tc, mipLevel);
        mipsB[i] = B.SampleLevel(sampler, tc, mipLevel);
        mipsM[i] = M.SampleLevel(sampler, tc, mipLevel);
    }

    float4 blended = lerp(mipsA[N], mipsB[N], mipsM[N]);
    for(uint i = 0; i < N; i++)
    {
        float4 laplacianA = mipsA[i] - mipsA[i + 1];
        float4 laplacianB = mipsB[i] - mipsB[i + 1];
        blended += lerp(laplacianA, laplacianB, mipsM[i]);
    }

    return blended;
}
```