Seamless Compressed Textures

Andrea Maggiordomo[1] and Marco Tarini[1]

[1]Università degli Studi di Milano "La Statale"

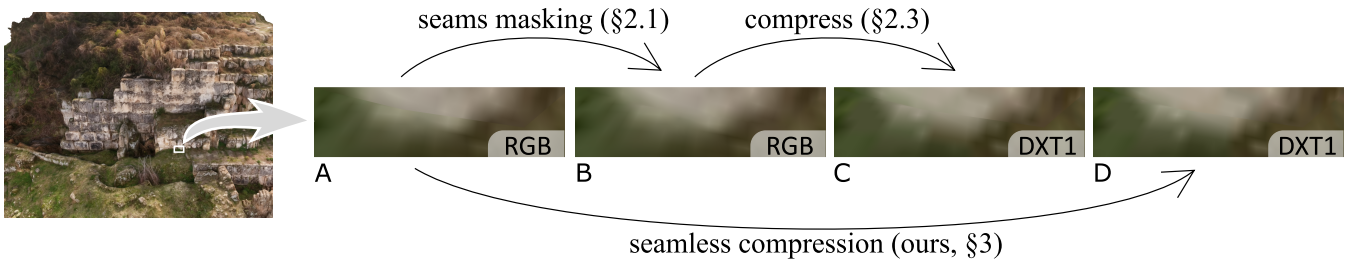

Figure 1: *Texture seams produce continuity artifacts on extreme close-ups, due to bilinear interpolation (A); they can be alleviated by automatically tweaking texture values (B), but the effect is partly lost when the texture is compressed for GPU (C). Our new method integrates seam-masking with texture compression, reducing seam artifacts on compressed textures (D). Textured model from [MPCT20] data-set.*

Abstract

We present an algorithm to hide discontinuity artifacts at seams in GPU compressed textures.
Texture mapping requires UV-maps, and UV-maps (in general) require texture seams; texture seams (in general) cause small visual artifacts in rendering; these can be prevented by careful, slight modifications a few texels around the seam. Unfortunately, GPU-based texture compression schemes are lossy and introduce their own slight modifications of texture values, nullifying that effort. The result is that texture compression may reintroduce the visual artefacts at seams. We modify a standard texture compression algorithm to make it aware of texture seams, resulting in compressed textures that still prevent the seam artefacts.

CCS Concepts

• ***Computing methodologies*** → ***Texturing; Image compression;***

1. Introduction

Texture mapping requires UV-maps, which require texture seams.

Texture seams. A texture seam, or texture cut, is an internal mesh edge which is mapped over two distinct twin edges in texture space, each on one of the two sides of the texture. Seams are a necessary evil, and are necessary (barring rare exceptions) in order to map the surface on the texture image without incurring in excessive distortions. In addition to other complications of various nature [YLT19], texture seams imply a minor but noticeable visual artifact deriving from the mismatching bilinear interpolated values occurring at the two sides of the cuts. In simple terms, in magnification filters, sampling the texture image with bilinear interpolation produces a discontinuity in screen space of the sampled values, breaking the C0 continuity that the signal reconstruction exhibits anywhere else.

Masking of texture seams. The problem has been identified [YLT19] and countered using several approaches. The most obvious countermeasure is to limit the number of seams, and carefully place them in areas where the artifacts will be less evident or less disturbing. This difficult to formalize objective is implicitly sought by many digital artists constructing the UV-map, and, incidentally, is one of the characteristics making automatic UV-map construction differ from the ones authored by digital artists. Automatic approaches have been proposed. [RNLL10] constructs UV-maps by restricting texture seams to the axis-aligned case, which is artifact free; [LFJG17] tweaks both an existing UV-map and an existing texture image to minimize artifacts. In our work, we consider the technique [Seb15] (Sec. 2.1), which is a simple approach that achieves similar resultsby only adjusting the texel values. Unfortunately, all these approaches rely on careful selection of texel values, which are impacted by texture compression.

Texture compression. Texture images consume GPU-RAM, which is a scarce, critical resource in many contexts, making texture compression essential. Compression schemes for textures dif-

DOI: 10.2312/egp.20221009

fer from other image compression schemes in that individual texel values must be accessible (e.g. during per *fragment processing*) in constant time, while keeping the image compressed in GPU-RAM (in contrast, standard image compression schemes require decompressing the entire image before pixels are accessed). As a consequence, texture compression schemes are lossy, with a fixed compression ratio, and a comparably unfavourable trade-off between quality loss and space occupancy. Examples include simple schemes such as palette (or color-map) compression, or quantization of RGB channels, and more sophisticated schemes such as 3Dc, A8L8 (often employed for normal maps), and DXT schemas (S3TC). In this work, we consider the case of the DXT1 schema, which is an S3TC schema designed for RGB images, popular in games and other applications, featuring a 1:6 compression ratio (Sec. 2.2). Other schemes could be adapted similarly.

Objectives. In this work, we show how to achieve seam masking on textures compressed with one popular format.

2. Prerequisites

2.1. Least Squared Seam Masking

Closely adapting [Seb15], a simple method to mask seams works is as follow. First, all seam edges are subdivided in small segments (having width of approx 0.1 texel side), and for each segment i we record its center p_i and its length. Then, a linear least square system is solved, with variables for all pixel values affecting any texture value used to interpolated values at any p_i. The system minimizes a quadratic energy function defined as $E_c + \alpha E_s$, where the color conservation term E_c is the squared discrepancy between original pixel values and new pixel values, and E_s is the seam energy consisting in the length-weighted squared discrepancy between colors at the two sides of the seam segment, computed at its center p_i; α is a weight used to balance between adherence to the original, and seam masking, which we default at 2.0.

2.2. The DXT1 encoding schema

DXT1 compression schema [INH99] works by splitting the image into 4×4 *blocks*. Within each block, two quantized colors c_0 and c_1 are stored (with 16 bits per colors using the R5:G6:B5 RGB encoding), and then a pair of bits is stored for each pixel, encoding a number t in $\{0, 1/3, 2/3, 1\}$. The final color is for that pixel is then: $c_0 \cdot (1-t) + c_1 \cdot t$. This schema totals 64 bits for the block, against a 384 bits of an uncompressed, 8-bits per channel, 3 channeled block.

2.3. Compression algorithm for DXT1

One natural algorithm to construct this structure for a given block consists of three phases: (1) the principal axis of a is found for the 16 pixels, using PCA; (2) the 16 pixels are projected on a, and then t values are found by quantizing the scalar parametric positions value between their max and the min values; (3) once t values are known for all pixels, a least squared minimization is employed to solve for c_0 and c_1, minimizing the (squared) discrepancies between the encoded and original pixel values, summed for the 16 pixels; (4) quantize $c_{0,1}$. Different methods can be employed (e.g. [Bro07]), but they invariably include something similar to phase (3).

3. Integrating Seam Masking with Texture compression

In our method, we combine phase (3) of the above algorithm with the Least Squares Seam masking (Sec. 2.1).

We start by applying Least Square Seam Masking, then apply a modification of the DXT compression algorithm to it. In the modified version, step (3) is replaced by a global Least Squares system: the variables are the values of c_0 and c_1 of *all* blocks involved by any seam, and the energy being minimized is $E_c + \alpha E_s$. Specifically, E_c penalizes the discrepancies between original color values at integer pixel locations and their compressed counterparts, so it serves to limit the compression loss. At the same time, E_s measures the discrepancy, integrated along all seam edges, of the bilinearly interpolated values of two sides of the seams.

Because interpolated color values are a linear function of the variables (remember that per-texel values t are fixed, in this stage), the global system is still a Least Squares minimization, actually involving fewer variables than in Least Squares Seam Masking.

4. Implementation and Results

We provide a reference implementation of our method as a publicly available Open Source application. This application produces a *seamless compressed texture*, given given an original texture and a mesh, which must be provided in order to identify the seams. We show a visual comparison in Fig. 1, and we plot a quantitative comparison in the poster. In conclusion, our method manages to hide seam artifacts in GPU-compressed textures.

References

[Bro07] BROWN S.: Dxt compression techniques (squish). blog post - https://www.sjbrown.co.uk/posts/dxt-compression-techniques/, Jan 2007. 2

[INH99] IOURCHA K. I., NAYAK K. S., HONG Z.: System and method for fixed-rate block-based image compression with inferred pixel values, Sept. 21 1999. US Patent 5,956,431. 2

[LFJG17] LIU S., FERGUSON Z., JACOBSON A., GINGOLD Y.: Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Transactions on Graphics (TOG) 36*, 6 (Nov. 2017), 216:1–216:15. doi:10.1145/3130800.3130897. 1

[MPCT20] MAGGIORDOMO A., PONCHIO F., CIGNONI P., TARINI M.: Real-world textured things: A repository of textured models generated with modern photo-reconstruction tools. *Computer Aided Geometric Design 83* (2020), 101943. doi:https://doi.org/10.1016/j.cagd.2020.101943. 1

[RNLL10] RAY N., NIVOLIERS V., LEFEBVRE S., LEVY B.: Invisible seams. In *EUROGRAPHICS Symposium on Rendering Conference Proceedings* (2010), Lawrence J., Stamminger M., (Eds.), Eurographics, Eurographics Association. 1

[Seb15] SEBASTIAN SYLVAN: Fixing texture seam with linear least-square. blog post - https://www.sebastiansylvan.com/post/LeastSquaresTextureSeams/, 2015. 1, 2

[YLT19] YUKSEL C., LEFEBVRE S., TARINI M.: Rethinking texture mapping. *Comput. Graph. Forum 38*, 2 (2019), 535–551. URL: https://doi.org/10.1111/cgf.13656, doi:10.1111/cgf.13656. 1