Volume-encoded UV-maps
Additional Materials:

Combining Volume-encoded UV-maps with Tangent-space Normal-maps

Marco Tarini*
Università dell'Insubria, Varese *and* ISTI-CNR, Pisa

Abstract

In this document we show how Tangent-space Normal-maps can be applied over a volume-encoded UV-mapped surface. The key is that we can define, and efficiently recover, appropriate tangent directions over the rendered surface.

1 Background

A Normal-map is a texture where each texels stores normal describing a local surface orientation, used for dynamic relighting of detailed-looking surfaces.

Tangent-space Normal-maps (from now on, TSNM) are a type of Normal-maps [Peercy et al. 1997] where normals are expressed in the local tangent space, i.e. the space defined by the three unit vectors: tangent $\vec{t}$, "bi-tangent" $\vec{b}$ and geometric normal $\vec{n}$, with $\vec{t}$, $\vec{b}$ approximating the local directions of the gradients $\nabla u, \nabla v$ of the u, v texture coordinates over the surface. A TSNM texture can be understood as describing modifications to be applied to the geometric "smooth" normal $\vec{n}$.

TSNM are recognized to offer many advantages over straightforward, Object-space Normal-maps: the most important one is that the Normal-map is defined independently from the orientation of the surface it is applied to. This means that the same Normal-map can be shared by different models, or by different parts of the same models (e.g. with tiling or for symmetry exploitation). Also, Normal-maps can be authored without any assumption on the surface, making them as versatile as plain color textures. Consequently, TSNM are the most common kind of Normal-maps in, e.g. Computer Games.

The drawback is that not only the original normal $\vec{n}$ but also a pair of additional 3D tangent directions, $\vec{t}$ and $\vec{b}$, must be available to the fragment shader.

2 Storing and accessing tangent directions

For traditional per-vertex UV-maps the standard solution is to precompute $\vec{t}$ and $\vec{b}$ from the UV-map, store them as per-vertex attributes, and linearly interpolate them during rendering.

In our case of volume-encoded UV-maps, our solution is to store them as texels in a second volumetric texture, formatted as P, and interpolate tri-linearly between these values, within a second volumetric texture-fetch operation.

Note that, in both cases, tangent directions are stored and accessed, in the same way uv texture coordinates are. This choice is forced, because the location of the discontinuities of uv positions coincide with the locations of discontinuities of ∇u and ∇v (and their approximations $\vec{t}$ and $\vec{b}$).

3 Pre-computing tangent directions

In both the traditional and the novel scenario, $\vec{t}$ and $\vec{b}$ directions can only be approximations of the real gradients ∇u and ∇v. Not only it would be impractical, within a GPU architecture, to recover ∇u and ∇v, but also undesirable, as they are are discontinuous, even away from cuts (because piecewise linear, or tri-linear, interpolation is only C^0); the resulting lighting would therefore reveal the underlying discretization.

In particular, in the traditional per-vertex case, ∇u and ∇v are constant on triangles (and discontinuous across triangles). Per-vertex $\vec{t}$, $\vec{b}$ vectors are defined as the (re-normalized) average of these per-face constant quantities, and are stored as per-vertex attributes. The final per-fragment $\vec{t}$, $\vec{b}$ vectors, inside faces, are then interpolated between these attributes.

In our case, the situation is conceptually analogous. Gradients ∇u, ∇v (see eq. 6 and 7 of main paper) have one constant coordinate on all *edges* of the regular grid. Specifically, the X coordinate of ∇u (and ∇v) is constant on edges aligned to direction X: its value is simply the the signed difference of the two u (or v) values stored at the two endpoint of that edge (and likewise for Y and Z). We define the per-texel value of $\vec{t}$, $\vec{b}$ as the average these constant values at the 6 incident edges of that texel (excluding the edges falling inside the "gap" voxels separating zones). Analogously to the standard solution, at rendering time we (tri-linearly) interpolate, inside voxels, between these averaged values.

Construction boils down to a simple algorithm: the coordinate X of $\vec{t}$ (of $\vec{b}$) for a texel i is precomputed as the halved difference of the u values (of the v values) associated to the previous and the next 3D texels in the X direction, and likewise for Y and Z. After this, per-texel $\vec{t}$ and $\vec{b}$ vectors are renormalized. Care must be taken that the u and v differences are always computed between texels belonging to the same *zone* (refer to the source code).

4 Optimizations and assumptions

A common optimization for the per-vertex scenario, which we inherit, consists in storing only $\vec{t}$, and replace $\vec{b}$ with its approximation $\vec{b}'$, computed on the fly:

$$\vec{b}' = \vec{n} \times \vec{t}$$

thus saving both on memory and on bandwidth.

This requires the assumption that $\vec{t}, \vec{b}, \vec{n}$ are all (approximately) reciprocally orthogonal. Orthogonality between $\vec{t}$ and $\vec{b}$ stems from

*e-mail:marco.tarini@isti.cnr.it

the *conformality* of the UV-map. In the traditional scenario, orthogonality between $\vec{t}$ (or $\vec{b}$), and $\vec{n}$, holds by construction (barring approximations). In our case, it stems from the *orthogonality* of f instead.

5 Results

See Fig. 15 of the original paper, and the attached demo, for examples of dynamically relighted Normal-mapped objects, where the tangent-space normal texture is mapped with a volume-encoded UV-map. Just as with standard TSNM, the results look convincing, whenever the displacements represented in the Normal-maps are small.

6 Discussion

At rendering time, the only additional overhead consist in a trilinearly interpolated access to the 3D texture storing $\vec{t}$.

With respect to the traditional per-vertex alternative, the tangent directions pre-computation is in our case more straightforward, less expensive (requiring fewer operations), and trivial to parallelize (because the algorithm works on regular grids). This makes on-the-fly precomputation a more viable option (for example, after importing a model in a Game Engine).

As for memory consumption, values are stored per-texel, instead of per-vertex, leading to the same exact tradeoff of the uv coordinates (see second last column of table 1 of original paper). Again, this tradeoff can be, but is not always, very convenient.

With respect to many other alternative ways to represent UV-maps (see Sec. 2.1 of original paper), the ability to define and rapidly evaluate smoothed tangent directions is one additional key benefit of our representation. For example, it is not clear how they could be even defined in purely volumetric representations like Brickmaps [Christensen and Batali 2004], spatial hashing [Lefebvre and Hoppe 2006; García et al. 2011] or octree based ones [Benson and Davis 2002], or how they could be recovered or smoothed with [Tarini et al. 2004], [Lefebvre and Dachsbacher 2007] or [Yuksel et al. 2010].

References

BENSON, D., AND DAVIS, J. 2002. Octree textures. *ACM Trans. Graph. 21*, 3, 785–790.

CHRISTENSEN, P. H., AND BATALI, D. 2004. An irradiance atlas for global illumination in complex production scenes. In *Proc. of EG Conf. on Rendering Techniques*, EGSR'04, 133–141.

GARCÍA, I., LEFEBVRE, S., HORNUS, S., AND LASRAM, A. 2011. Coherent parallel hashing. *ACM Trans. Graph. 30*, 6.

LEFEBVRE, S., AND DACHSBACHER, C. 2007. Tiletrees. In *Proc. of the Symp. on Interact. 3D Graph. and Games*, ACM, 25–31.

LEFEBVRE, S., AND HOPPE, H. 2006. Perfect spatial hashing. In *ACM Trans. Graph.*, vol. 25, ACM, 579–588.

PEERCY, M., AIREY, J., AND CABRAL, B. 1997. Efficient bump mapping hardware. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '97, 303–306.

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. Polycube-maps. *ACM Trans. Graph. 23*, 853–860.

YUKSEL, C., KEYSER, J., AND HOUSE, D. H. 2010. Mesh colors. *ACM Trans. Grap. 29*, 2, 15:1–15:11.