Octree Textures on Graphics Hardware

Joe Kniss
University of Utah

Aaron Lefohn
University of California, Davis

Robert Strzodka
Caesar Research Institute, Bonn

Shubhabrata Sengupta
University of California, Davis

John D. Owens
University of California, Davis

We implement an interactive 3D painting application that stores paint in an octree-like GPU-based adaptive data structure. Interactive painting of complex or unparameterized surfaces is an important problem in the digital film community. Many models used in production environments are either difficult to parameterize or are unparameterized implicit surfaces. We address this problem with a system that allows interactive 3D painting of complex, unparameterized models. The included movie demonstrates interactive painting of a 817k polygon model (as shown in Figure 1) with effective paint resolutions varying between 64^3 to 2048^3. Our implementation differs from previous work [Benson and Davis 2002; Carr and Hart 2004; DeBry et al. 2002; Lefebvre et al. 2004] in two important ways: first, it uses an adaptive data structure implemented entirely on the GPU, and second, it enables interactive performance with high quality by supporting quadlinear (mipmapped) filtering and fast, constant-time data accesses.

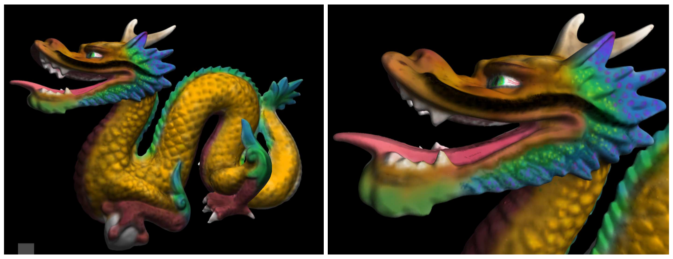
Figure 1: Our interactive 3D paint application stores paint in an octree-like structure with an effective resolution of 2048^3 (using 15 MB of GPU memory). Our system paints and renders this 817k polygon model with an octree texture at 15–20 fps and supports quadlinear filtering.

Implementation Detail

Our implementation is inspired by the CPU-based octree texture techniques of DeBry et al. [2002] and Benson and Davis [2002], including mipmap support. Our GPU data structure is implemented in the Glift library [Lefohn et al. 2005]. The structure uses a 3D virtual and physical address space (a small 3D physical memory buffer) with an address translator that utilizes a mipmap hierarchy of page tables. The 3D physical memory format enables the GPU to perform native trilinear filtering. We use the normalized 3D vertex coordinates of the rest pose of the model as texture coordinates.

Using the octree data structure in a Cg shader is similar to a conventional 3D texture access. The Glift library inserts the shader code required to perform the data access at compile time. Below is an example shader:

```
float4 main( uniform VMem3D octreePaint,
                     float3 objCoord  ) : COLOR {
   return octreePaint.vTex3D( objCoord );
}
```

The bulk of development time was spent designing brushes, painting techniques, and developing appropriate mipmap creation filters for our node-centered data representation. Our application locates brush-model intersections using the GPU to rasterize the model's texture coordinates. While brush profiles are described in screen space, painting on a surface is dependent on surface orientation and visibility (see Figure 2L). Thus texels are first projected to screen space and brush weights are applied to texture samples based on the 2D brush profile. Texture resolution is selected dynamically based on the brush size, requiring refinement or coarsening as necessary.

We unify memory usage for adaptive paint resolutions and mipmapping. The page table hierarchy allows us to share physical tiles between mip levels when the adaptive resolution is coarser than or equal to the mip level's resolution. This is non-trivial for two reasons. First, since we are using a "node-centered" scheme, as opposed to traditional "cell-centered" textures, the filter support is different (see Figure 2R). Second, since we are allocating texels in blocks, many texture samples do not intersect the surface and thus will never be assigned colors. It is important that these samples do not contribute to the filtering used for coarsening. Thus we identify "valid" texels by recording the resolution level at which each texel has been written into the alpha channel. This allows us to refine from low resolutions to higher resolutions while still identifying which samples were actually painted at a specific resolution.

Results and Future Work

The frame rates for viewing textured models in our 3D paint application are unaffected by the use of a single octree texture map. The frame rates are thus determined entirely by the complexity of geometry and varied between 15 and 80 fps with models ranging in complexity from 50k to 1M polygons. Frame rates while painting depend on the size of the current brush, and we maintain highly interactive rates during painting.

Future work includes adding support for 1- or 2-level page tables to provide a balance between access time and memory consumption. We are also working on support for the normal-keyed indexing described in previous octree texture implementations [Benson and Davis 2002; DeBry et al. 2002].

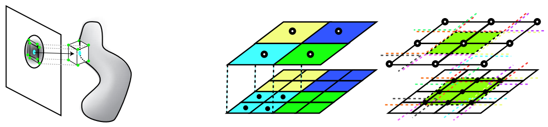
Figure 2: Left: Applying a 2D brush profile to 3D texels, by projecting to screen space. Right: Cell-centered versus node-centered filtering. Note the difference in filter support; cell-centered has a 2×2 support, where node-centered has 3×3 with some fine nodes shared by multiple coarse nodes.

References

Benson, D., and Davis, J. 2002. Octree textures. *ACM Transactions on Graphics 21*, 3 (July), 785–790.

Carr, N. A., and Hart, J. C. 2004. Painting detail. *ACM Transactions on Graphics 23*, 3 (Aug.), 845–852.

DeBry, D., Gibbs, J., Petty, D. D., and Robins, N. 2002. Painting and rendering textures on unparameterized models. *ACM Transactions on Graphics 21*, 3 (July), 763–768.

Lefebvre, S., Hornus, S., and Neyret, F. 2004. All-purpose texture sprites. Tech. Rep. 5209, INRIA, May.

Lefohn, A. E., Kniss, J., Strzodka, R., Sengupta, S., and Owens, J. D. 2005. Glift: Generic, efficient, random-access GPU data structures. *ACM Transactions on Graphics*. To appear.