3D Paint Baking Proposal

Robert L. Cook
Pixar Technical Memo 07-16

Synopsis

3d paint has long been one of the most expensive parts of rendering at Pixar. This proposal is for a new baking technique that would greatly reduce the run-time cost of 3d paint and require no changes to the existing workflow. The implementation makes heavy use of existing code, which reduces the amount of risk in the project.

Background

Our early shorts used a projection paint technique. Textures were painted over a rendered reference pose, then converted to a uv-parameter space and stored in a mip-map. This conversion made 3d paint relatively inexpensive at run time. Later, we changed the geometric primitives in our character models from bicubic patches to subdivision surfaces (*subdivs*). Because subdivs do not have a natural uv-parameterization, we started doing the 3d projection paint calculations at run time. These operations became even more costly because we started painting on multiple views of each object and added features like paint occlusion, feathering, etc.

Current technique

A review of our 3d-paint technique shows why it is so expensive. For each 3d-paint variable, for every micropolygon (*mp*), for each of a number of reference *poses*, the mp's position in that pose in is projected onto a number (usually at least 6) of *views*. For each pose-view pair, the position and a derivative are used to do a texture-map lookup. Also, for each pose, the position is used to do a shadow-map lookup, and this shadow value and the depth of the surface in the reference pose are used to block the paint if the surface is not the front-most surface in the reference pose. A feathering calculation is applied to gradually taper off the paint at silhouette edges. Then the paint from the different views and poses is composited.

The advantage of the current technique is that it is very convenient for painters because the can render different views of the reference pose and then paint directly on these views. The disadvantage is that these calculations are expensive and repeated for every rendering of every frame, even though the identical answer is obtained every time.

Proposed technique

Because the projected paint changes only when the texure maps or model geometry change, we can bake the results of these calculations. The lack of a uv-parameterization means that baking into normal mip-maps won't work, and a volumetric approach like brick maps is slow and takes a lot of storage. The new technique proposed here is fast and does not require a global uv-parameterization.

In this approach, paint is baked in accordance with the subdivision hierarchy. When a subdiv is rendered, it is divided into a number of *faces* (triangles, quadrilaterals, or pentagons), each of which is divided into one or more quadrilaterals (*quads*). These quads are then diced along the uv direction into mps. The texture information for each quad can therefore be baked into a standard mipmap. The baking for a subdiv consists of a collection of mipmaps and the information about their topology within the subdiv.

The technique consists of a baking process and a sampling process. In the baking process, we create a texture mipmap for each quad of each subdiv surface. (The texture mipmaps for multiple quads can be combined into a single texture file using a method described below, but for now think there being one texture file per quad.) This consists of the following steps:

- From the paint files and the reference poses of the subdiv, determine the dicing rate that corresponds to the resolution of the paint. Round this resolution up to the next power of two. This is our highest baking resolution.
- Split the subdiv into its faces and split each face into its quads.
- For each quad
    - For every power-of-two from 1x1 to the highest baking resolution:
        - dice the quad at that resolution into a grid of mps
        - invoke the 3d paint code to determine the paint value for each mp
    - combine the baked paint values at these resolutions into a mipmap
    - store the mipmap as a standard texture file

In the sampling process, the 3d-paint shading library gets the baked paint values for each grid from the texture files. This process consists of the following steps:

- construct the file name from the subdiv name, the face id, and the quad id.
- invoke the texture system to read and interpolate the paint values.

Implementation notes

Fortunately, the face and uv information is either already available in the renderer or can be made available easily. Each subdiv surface already has a unique identifier that is currently available to the shader. A unique number identifying each face in the subdiv and each quad in the face can be made available to the shader as uniform variables. (I've spoken to Julian about this, who said he could make this change for us without much work.) The location within each diced quad is already available as the varying parameters u and v, and the dicing rate is available as du and dv.

The code the renderer uses to split and dice is already available in a c library. The 3d paint projection code is also available in a c library. The code to create texture files from the mipmap data is simple and can borrow heavily from existing code.

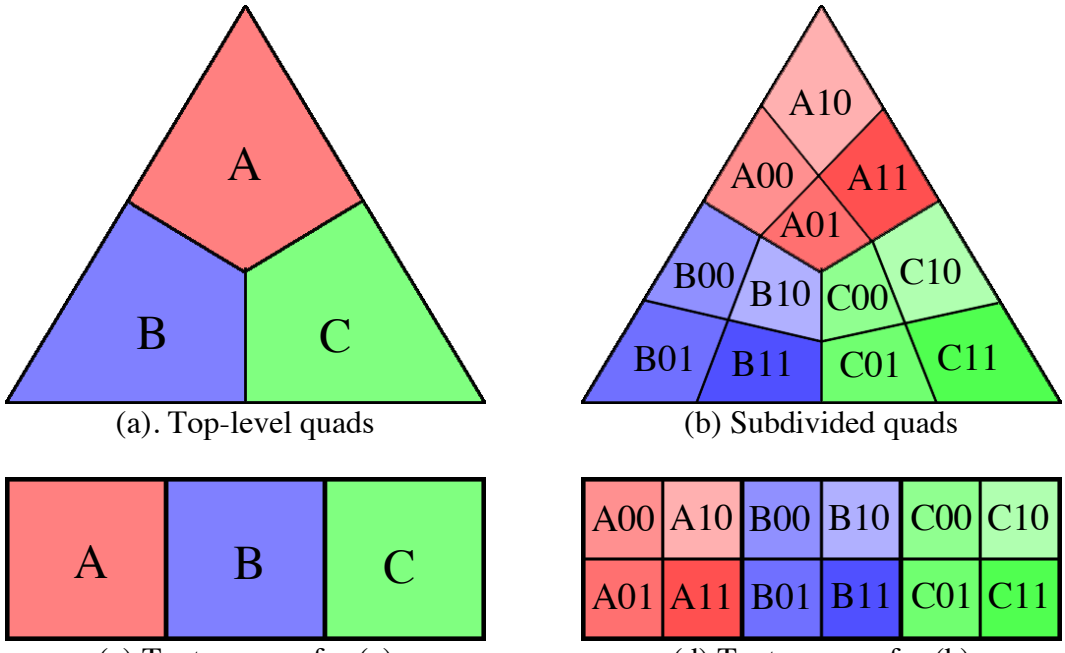
Because the projected paint changes only when the texure maps or model geometry. The 3d paint shading library will use the existing interface but will be rewritten to read the

paint from the texture file. The existing texture routines can be used to do the interpolation.

Multi-mipmap files

With one texture file per quad, there may be an untenable number of texture files in a scene. In order to avoid this problem, we can store all of the texture mipmaps for all of the quads in a character in a single texture and thus in a single texture file.

The top level of the mipmap would contain one pixel per quad; that pixel contains the value of the texture if the quad was a micropolygon. (Alternatively, if shading is done at vertices, it could contain one pixel per vertex of the quad.) Lower levels of the mipmap contain the higher resolution texture information of their parent pixel (i.e, quad). For purposes of this discussion, we treat the top level as an n by n image, where n^2 is the number of quads, but the image can be any convenient 2d shape.



(a). Top-level quads (b) Subdivided quads

(c) Texture map for (a) (d) Texture map for (b)

Figure 1.

The file format is illustrated in Figure 1. Figure 1a shows the top-level quads, and Figure 1b shows one level of subdivision. Figure 1c shows the pixels in the top level of the texture mipmap file that corresponds to 1a, and Figure 1d shows the pixels in the mipmap level that corresponds to 1b.

If in an image quad A is not subdivided when it is rendered, then the value A in texture map 1c is used. If in an image quad A is diced into a 2x2 micropolygon grid when it is rendered, then the values A00, A01, A10, and A11 in texture map 1d are used for the micropolygons. No texture map interpolation is needed in this case, or in any power-of-2 dicing case.

If non-power-of-2 dicing is used, then it's necessary to interpolate. Note, however, that the uv values will be such that we never interpolate between texture values for different quads. For example, the u mapping for Figure 1d is shown in Figure 2. As u_A varies from 0 to 1 we interpolate between A00 and A10, and as u_B varies from 0 to 1 we interpolate between B00 and B10, but we never interpolate between A10 and B00 because there is no u that maps to this region.

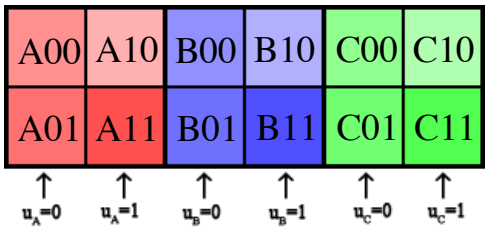


Figure 2.

We may want store the paint for a character in a few files instead of in one file. Different parts of the character may have paint at different resolutions and thus may have different numbers of mipmap levels. If we were to put all of the paint for a character in one file, some parts of the mipmap could be empty. Because of this, we probably want to create one texture file for each painting resolution – there would be one file for all subdiv faces that have 5 levels of mipmap, one for faces with 6 levels, etc.