High Performance Interactive Painting On The GPU

M. Großer[1] and M. Wacker[1]

[1]Dresden University of Applied Sciences, Dresden, Germany

Abstract

This paper addresses the problem of mapping between two different texture coordinate setups for one single mesh. We exploit the advantages of the render pipeline on the GPU to obtain interactive texture painting on an already textured mesh. The approach combines projection mapping with a render-to-texture technique in order to bake additional layers of texture or detail into a given texture. The benefit of our approach is high performance and complete independence of complexity of the object and the parametrization.

1. Introduction

Interactive texture painting has become a widely used tool not only to apply paint or texture with different styles onto an object (e.g. [SH10]) but also to add geometric detail [Pix11] or to simulate physical processes [LF04]. Most existing approaches do not refer to an existing mapping and automatically create a new one for the newly added detail (as in [Iga01]). Schärfig and Hormann [SH10] present an approach where the basic idea is similar to ours. However, their implementation is different. They need a brush geometry to guarantee the brush texture being copied correctly into the texture atlas. Furthermore, their approach depends on the window size and the camera position. The presented approach does not have these restrictions. Two simple shader programs provide a fast texture baking process for interactive texture painting. Moreover, we apply our method to visualize the process of concrete spraying in an industrial application [Put11].

2. Projection Mapping

For an object $O \subseteq \mathbb{R}^3$ we have a predefined mapping $\theta : O \to \mathfrak{T}$ for the basic texture. Projection mapping provides us with a map $\varphi : O \to \mathfrak{B}$ from the object O to the brush texture $\mathfrak{B}$. The projection is similar to the standard transformation from the 3D world space to the screen coordinate system. Therefore, the transformation matrix T looks familiar:

$$T = N \cdot P \cdot V \ , \tag{1}$$

where V is the projector view transformation matrix, P is the projection matrix and N is a normalization step. So far, we have two different mappings for O, the θ, which was given, and φ from the projection mapping. Remember, the aim is to realize the mapping $\omega : \mathfrak{B} \to \mathfrak{T}$ in an efficient way.

3. Render to Texture

The render-to-texture method is commonly used in practice for image processing on a pre-rendered image. OpenGL extensions support render-to-texture, i.e. a frame buffer object or a pixel buffer. Nowadays, render-to-texture is a well known tool. The basic idea is to change the render target from the screen to a texture. We use this method to generate a new texture which combines the color information of the basic texture and the brush texture. Replacing the basic texture implies that the texture atlas of the new texture and the basic texture are the same.

4. Implementation

Combining the above described ideas in our implementation, only a simple vertex and fragment shader program is necessary to bake the brush texture into the basic texture (Shader Model 2.0, here we use GLSL). In the conventional pipeline, the vertex shader transforms each vertex from the local coordinate system to the screen coordinate system. Additionally, it can influence the vertex attributes like texture coordinates or vertex color. In contrast, our approach renders the result of the predefined parametrization θ into the texture buffer (line 14). Hence the pixels appear as in the texture space.

The input vertex is only necessary to provide the reference to the projection mapping (line 7).

```
1 uniform mat4 T;
2 uniform mat4 local2World;
3
4 void main(void)
5 {
6   gl_TexCoord[0] = gl_MultiTexCoord0;
7   gl_TexCoord[1] = T * local2World * gl_Vertex;
8
9   mat4 norm = mat4( 2, 0, 0, 0,
10                    0, 2, 0, 0,
11                    0, 0, 0, 0,
12                   -1,-1, 0, 1 );
13
14  gl_Position = norm * gl_MultiTexCoord0;
15 }
```

The fragment shader program implements the color calculation for each pixel and we also calculate the new (blended) color of the current pixel. First, we obtain the color from the basic texture (line 16 and 21, 22). The brush color is added only if the corresponding texture coordinates $(u,v) \in [0,1]^2$ (line 32,33). Ultimately, a blending function is used to get the final color for the current pixel in the new texture. An example is given in line 35.

```
16 uniform sampler2D baseTex;
17 uniform sampler2D brushTex;
18
19 void main(void)
20 {
21    vec4 basic = texture2D(baseTex,
22                 gl_TexCoord[0].st);
23
24    vec4 brush = vec4(0,0,0,0);
25
26    float s = gl_TexCoord[1].s /
27              gl_TexCoord[1].q;
28
29    float t = gl_TexCoord[1].t /
30              gl_TexCoord[1].q;
31
32    if(s <= 1 && t <= 1 && s >= 0 && t >= 0)
33         brush = texture2D(brushTex,vec2(s,t));
34
35    gl_FragColor = color1 + color2;
36 }
```

5. Conclusions and Results

With these shader programs, we have introduced a high performance technique for geometry painting and texture baking on the GPU. We can handle every given parametrization and do not have to generate a new one. The following table presents our results of different test scenarios (Intel Core i7 860@2.8GHz, GeForce GTX470). We used a simple cube and a high resolution model of the Blender Monkey Head with an automatically generated (complex) texture atlas (figure 1).

Object	Vertices	Texture Size	FPS
Simple cube	28	1024 x 1024	3300
Monkey head	125.952	2048 x 2048	1600

Figure 1: *Monkey head (left) and the texture atlas (right)*

Additionally, we implemented our approach in a simulation application [Put11] to realize the concrete spraying mechanism of a shotcrete machine (figure 2). With the described approach we visualized the spraying process from the nozzle to the tunnel wall in an highly interactive simulation.

Figure 2: *Putzmeister SPM500 shotcrete machine*

6. Future Work

We presented the basic method for a hardware accelerated texture baking, texture painting or bump mapping algorithm. It enables extensions such as integration of occlusion or multilayered textures which will be considered in future work.

References

[Iga01] IGARASHI T.: Adaptive unwrapping for interactive texture painting. In *ACM Symposium on Interactive 3D Graphics* (2001), ACM, pp. 209–216. 1

[LF04] LAWRENCE J., FUNKHOUSER T.: A painting interface for interactive surface deformations. *Graph. Models 66* (November 2004), 418–438. 1

[Pix11] PIXOLOGIC: Zbrush. http://www.pixologic.com/zbrush/, March 2011. 1

[Put11] PUTZMEISTER: Putzmeister develops a new sprayed concrete simulation system. http://shotcrete.putzmeister.es/main.php?act=dnew&n=61&l=e, March 2011. 1, 2

[SH10] SCHÄRFIG R., HORMANN K.: Hardware accelerated 3D mesh painting. In *Proceedings of Vision, Modeling, and Visualization 2010* (November 2010), Eurographics Association, pp. 211–218. 1