Hardware-Assisted Relief Texture Mapping

Masahiro Fujita and Takashi Kanai

Keio University Shonan-Fujisawa Campus, Fujisawa, Kanagawa, Japan

Abstract

Image-Based Rendering by Warping (IBRW) creates three-dimensional scene by deforming several 2D images with depth information. Image-based rendering has an advantage compared to traditional polygon rendering that the rendering time is still constant even if a scene becomes complex. Relief texture mapping decomposes IBRW into a very simple 1D image operation and a traditional texture mapping, which allows more effective computations. In this paper, we try to apply some hi-quality shading effects such as reflection mapping for relief texture mapping. Their effects can be realized by per-pixel shading technology of today's graphics hardware. Our method for relief texture mapping allows fast and almost the same shading as well as traditional polygon rendering.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, Shading, Shadowing, and Texture

1. Introduction

Relief texture mapping[8] establishes 3D rendering of image-based objects by warping several 2D images with depth information. As the rendering speed does not depend on the size of 3D model, it has various advantages for rendering large 3D models compared to polygon rendering.

However, in general, image-based rendering approaches including relief texture mapping have poor shading qualities than polygon rendering. In the field of polygon rendering, there have been various improvements of shading techniques using 2D images. The most popular technique is texture mapping. By encoding normal or lighting information as texture image and by blending them, it is also possible to merge bumpy or lighting effects on polygonal surfaces.

In this paper we discuss a method for combing additional shading effects in relief texture mapping. Our method adopts a hybrid approach of multi-texture mapping and image-based rendering, which allows hi-quality shading effects with texture-based images. Because traditional graphics API is designed for rendering per-vertex shading as a basis, it was difficult to apply image-based techniques which need per-pixel calculations.

Fortunately, recent graphics hardware such as GeForce3 by has programmable per-pixel shader functions. We use such functions to combine them with various texture mapping techniques.

2. Related work

Rendering Techniques Using Images

Two popular techniques for adding three-dimensional effects on surfaces by using images are *bump mapping*[1] and *displacement mapping*[3]. Bump mapping represents a bumpy surface by perturbing surface's normal vector. However, it does not actually exhibits geometric properties of a surface. For example, bumpy shapes disappear when a view direction becomes close to a surface.

On the other hand, displacement mapping can actually transform geometric property of surface using images which represents displacement information. Displacement mapping can represent more natural bumpy shapes than bump mapping, but it dramatically consumes rendering computations because an original surface must be subdivided.

Image-Based Rendering

In recent years, many techniques have been proposed for image-based rendering (IBR). The movie-map system by Lippmap[6] is the most early technique to reconstruct scene environment by images, which view directions are limited in a fixed region. Chen et al. [2] develop QuichTime VR system which constructs scene environment from cylindrical images.

McMillan[7] proposes the 3D image warping method which

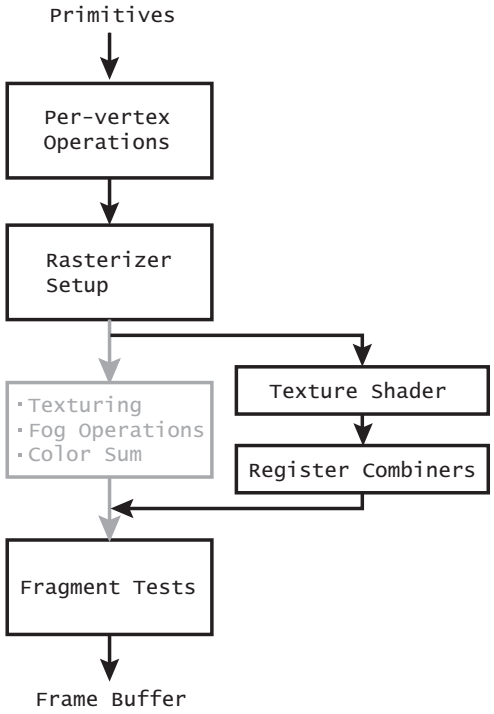

Figure 1: *Per-pixel shading pipeline*

constructs 3D scenes only from several images with its corresponding depth information. Relief texture mapping by Oliveira[8] decomposes this 3D image warping approach into 1D image operations (called *pre-warping*) and texture mapping, enabling 3D visual effects only by using images.

Per-Pixel Shading

By per-pixel shader functions originally developed NVIDIA Corp.[4], users can operate some rendering pipeline processes programmable, especially for texel fetch stage and a part of fragment processing stages.

In graphics API such as OpenGL, these per-pixel shading functions replace texture-fetch, fog operation and color sum states to texture shader and register combiner stages (Figure 1).

In the texture shader stage, there are 21 texture-fetch programs. Per-fragment operations such as dot product, offsetting (perturbing texture coordinates), and clipping tests can be done. In register combiner state, users can compute fragment colors programmable in GPU by treating fragment data such as texture color, vertex color, and fog color as like CPU registers, It also supports per-fragment operations such as dot product, multiplication, comparison and addition.

Using these functions, it is possible to apply some visual effects such as reflection mapping which need per-pixel

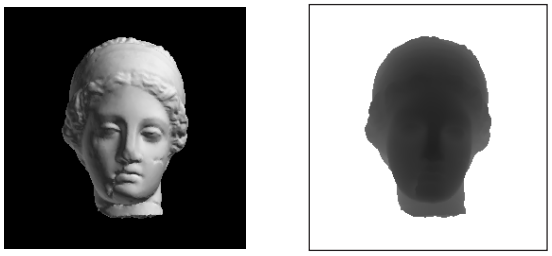

Figure 2: *Relief texture image*

dot computation by graphics hardware. DirectX 8.0, one of widely spread graphics API has already supported these shader functions. OpenGL, the other major graphics API , will support soon (Version 2.0). Thus, we believe this per-pixel shading functions to become commonly available.

3. Relief Texture Mapping

In this section, we describe the detail of relief texture mapping. Relief texture mapping is based on the three-dimensional image warping equation by MacMillan[7] which calculates per-pixel, perspective-corrected image of any view, using an original image, called *source image*, corresponded with depth value. The image which is reconstructed at any view position is called *target image*.

Oriveira[8] decomposes three-dimensional warping equation into *pre-warping equation* and texture mapping. Relief texture mapping is achieved by generating a pre-warped image on a source image plane using pre-warping equation, and then projecting it to a target image plane using texture mapping.

A pre-warped image can be calculated by resolving each moving position (u_i, v_i) on a source image plane. From the relationship between (u_i, v_i) and a point (u_s, v_s) on an original image, we define a pre-warping equation as follows:

$$u_i = \frac{u_s + k_1 displ(u_s, v_s)}{1 + k_3 displ(u_s, v_s)} \quad (1)$$

$$v_i = \frac{v_s + k_2 displ(u_s, v_s)}{1 + k_3 displ(u_s, v_s)} \quad (2)$$

$$k_1 = \frac{\vec{f} \cdot (\vec{b} \times \vec{c})}{\vec{a} \cdot (\vec{b} \times \vec{c})}, k_2 = \frac{\vec{f} \cdot (\vec{c} \times \vec{a})}{\vec{b} \cdot (\vec{c} \times \vec{a})}, k_3 = \frac{\vec{f} \cdot (\vec{a} \times \vec{b})}{\vec{c} \cdot (\vec{a} \times \vec{b})}$$

where $\vec{a}, \vec{b}, \vec{c}, \vec{f}$ denote coordinate axis vectors of both the source and the target images, *displ* denotes the orthogonal displacement. As the pre-processing, we calculate $(k_1 displ(u_s, v_s), k_2 displ(u_s, v_s), 1/1 + k_3 displ(u_s, vs))$ and store constant tables to use depth values as indices. These tables are used to calculate (u_i, v_i) effectively at the rendering process.

Textures of relief texture mapping consist of images of color pixels and their associated depth values. (Figure 2). It

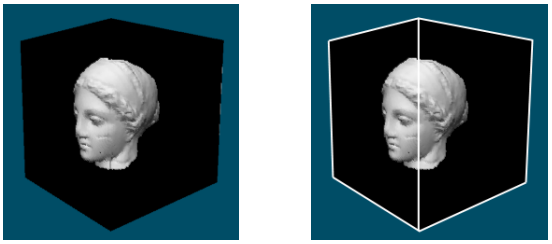

Figure 3: *Object represented by relief texture mapping*

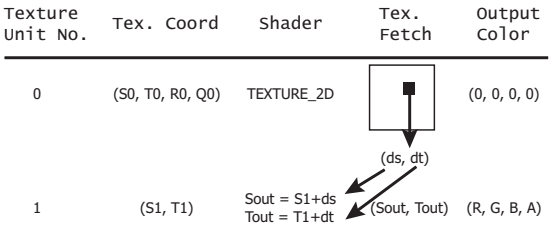

Figure 4: *Per-texel normal perturblation using offset-texture-2d*

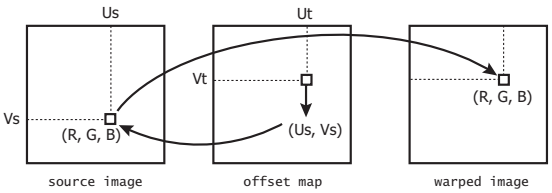

Figure 5: *warping image generation using offset-texture-2d*

is possible to represent an object by organizing textures as cube (Figure 3).

We use *forward warping* to calculate a final image of a target image plane from each pixel of a source image plane. A pre-warping equation described above follows this forward warping. In forward warping, sampling is needed because one pixel of source image plane can be possibly mapped one or more pixels of a target image plane.

4. Hardware-Assisted Relief Texture Mapping

In this section, we describe our method for replacing a part of operations of relief texture mapping to per-pixel shader functions. We also apply some shading effects to relief textures, normal mapping, reflection mapping and light mapping using these functions.

4.1. Replacing Relief Texture Mapping Procedure

An original relief texture mapping is done by the following procedure:

1. Set view point and view direction.
2. Pre-caluclate constant table.
3. Generate pre-warped image from a source image.
   a. Calculate pre-warping position (u_t, v_t) using pre-warping equation (1), (2).
   b. Copy color pixels of a source image to pre-warped positions.
   c. If the pixel covers one or more pixels of a target image, then interpolate several sampling points.
4. Render quad polygons by applying pre-warped images as texture.

With our method, relief texture mapping using per-pixel shading functions replaces the original procedure described above to the following procedure:

1. Set view point and view direction.
2. Pre-caluclate constant table.
3. For each pixel of source image (u_s, v_s), generate **offset map** using the following procedure.
   a. Calculate pre-warped position (u_t, v_t) using pre-warping equation (1), (2).
   b. Write positions of source image (u_s, v_s) as a **RGB value** to pre-warped position.
   c. If the pixel covers one or more pixel of target image, then interpolate several sampling points.
4. Render quad polygons with texture shader to `OFFSET_TEXTURE_2D`, offset map as source, source image as target.

4.2. Generating Offset Maps

Offset Texture 2D (`OFFSET_TEXTURE_2D`) function is one of texture shader functions (Figure 4). This can perturb texture coordinates in texel level. A pre-warped image of relief texture mapping can be represented as offset values using offset texture (Figure 5). In the pre-warped image calculation phase, we generate them a pre-warped image texture by writing positon information as rgb value, called *offset map*, not by generating a pre-warped image texture dynamically copying source image pixel. In the texture rendering phase, calculating a warped image and applying multiple texture mapping can be done at a time using `OFFSET_TEXTURE_2D` function (Figure 6). Each pixel value of this offset map represents a texel position of a source image. Once an offset map is calculated, it is possible to apply it to other reference images such as normal map described below (Figure 7).

4.3. Normal Mapping

Because relief texture only stores surface colors, diffuse shading is possible by using only relief texture. More complex shading such as specular hilights, lighting and reflection mapping requires surface normal information. However, as a

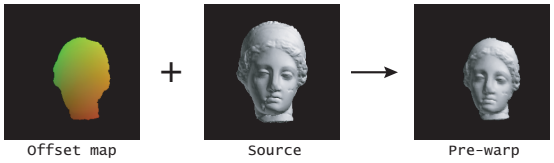

Figure 6: *Pre-warp image generation from source image using offset map.*

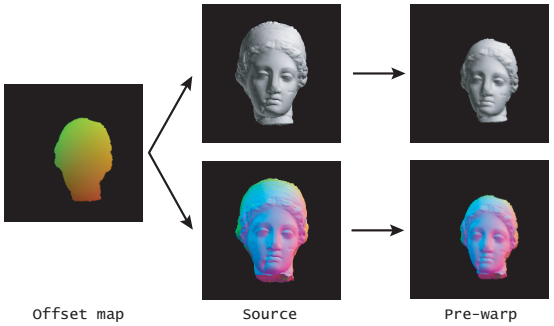

Figure 7: *Applying offset map to different source images.*

relief texture is an "image", it cannot compute surface normal in rendering time. Then, it is possible to calculate shading computations which require surface normal information by providing normal information as images in advance. *Normal map* is the texture which represents x, y, z coordinates of a normal vector as a RGB value. By combining this normal map and relief texture mapping, shading operations such as per-pixel reflection mapping described below, is possible. We create normal map using BMRT rendering tool[5] with custom shader which converts a normal vector into a RGB value.

4.4. Reflection Mapping

By using `DOT_PRODUCT_CONSTANT_EYE_REFLECT_CUBE_MAP`, one of texture shader functions, it is possible to apply per-pixel reflection mapping (Figure 9). dot-product-constant-eye-reflection-mapping is the function of reflection mapping using cube map texture with constant view (eye vector is constant). Cube map texture is the texture representing reflected environment as 6 images of cube (Figure 10).

Per-pixel reflection mapping using texture shader can be done with the following 2-pass rendering procedure:

1. Set normal map for source image.
2. **The first pass**: Render pre-warp image of normal map using offset map.
3. Read back frame buffer and converts a rendered pre-warp image into texture.
4. **The second pass**: Set texture shader to `DOT_PRODUCT_CONSTANT_EYE_TEXTURE_CUBE_MAP`, texture unit 0 to pre-warp texture of normal map rendered in pass 1, and texture unit 3 to cube map texture, then render.

Figure 8: *normal map*

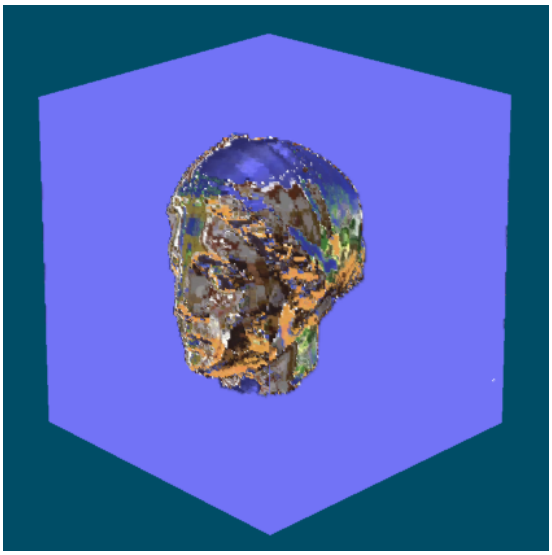

Figure 9: *Per-pixel constant eye reflection mapping.*

The reason why our reflection mapping is done in 2-pass is that GeForce3 video card we target in this paper is up to 4 texture units in each pass. Our method uses 2 units for warped image generation, 4 units for dot-product-constant-eye-cube-map texture shader, summing 6 texture units. If a future video chip supports over 6 texture units, it can be rendered in a single pass.

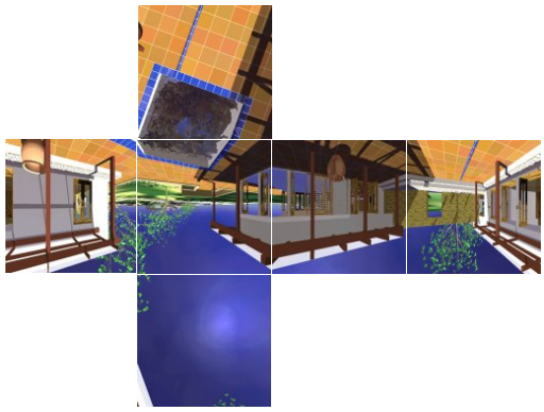

Figure 10: *Cube map texture (courtesy of NVIDIA Corp.)*

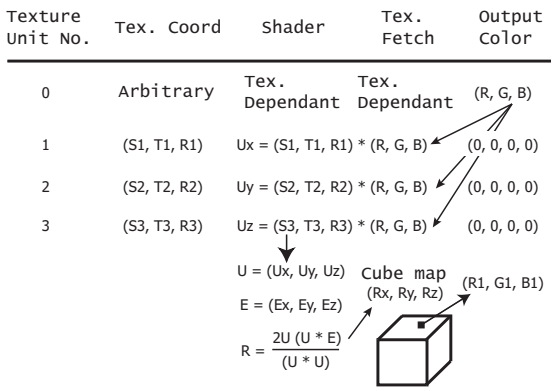

Figure 11: *Texture shader settings for constant eye reflection cube map*

4.5. Light mapping

If the lighting has only a diffuse component and is static (such as lighting of radiosity), it is possible to add lighting information to relief texture image using pre-computation. If the lighting is view-dependent, a specular component should be included, or light may be moved dynamically, the *light map* can be used (Figure 12). Light map is the reflection cube map which value is light intensity, not reflected environment. So we can apply this light mapping by a similar procedure of reflection mapping. By using light mapping, it is possible to include complex lighting into relief texture mapping.

5. Results and Discussion

The comparison of rendering performance between our proposed hardware-assisted method and software implemented method is shown in Table 1.

Each value is the frame-rate of rendering two relief texture with 256x256 pixel size on PentiumIII 1GHz PC with Geforce3 graphics card. In gouraud shading, there are no differences. In reflection mapping rendering, our method is 25% faster than software rendering.

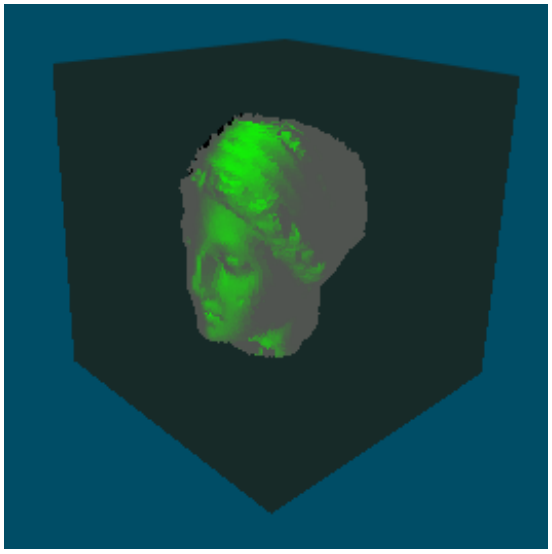

Figure 12: *Light mapping which have green hilights*

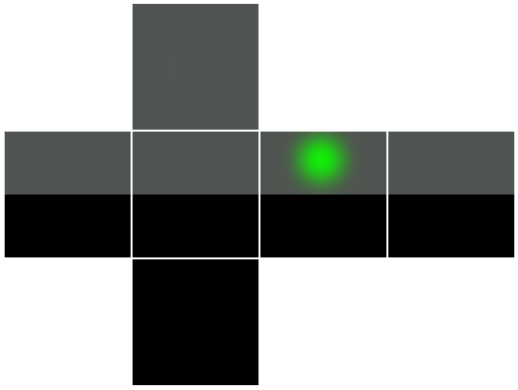

Figure 13: *Light map image*

Relief texture mapping is finally rendered as texture mapped planar polygon, so depth buffer records the depth of planar polygon. Thus, The inconsistency occurs when mixing it with traditional polygon rendering. To represent correct depth of relief texture in screen space, it requires per-pixel depth modification. Texture shader's `DOT_PRODUCT_DEPTH_OFFSET` displaces depth value based on per fragment dot calculation. Using this functions, it may be possible to represent correct depth in screen space.

Reflection mapping presented here is organized with reading back the frame buffer and with converting into textures in the first pass. Usually, frame buffer access is very costly op-

Shading	Method	Frame/sec
Gouraud	Software	14.0
Gouraud	Hardware-assisted	13.0
Reflection mapping	Software	8.3
Reflection mapping	Hardware-assisted	10.4

Table 1: *rendering performance(rendering of two 256x256 relief texture)*

eration. Recent graphics hardware supports off-screen rendering called P-buffer (Pixel buffer) extension[9]. This enables faster memory copy from frame buffer into texture. By using P-buffer extension, it is expected that the process which needs 2 or more pass rendering such as reflection mapping shown as our proposed method can be significantly faster.

6. Conclusion and Feature Work

We presented the combination of relief texture mapping and hardware-assisted various image mapping method and have shown that these enables hi-quality shading images. We have also shown that it is possible to compute texture-based shading efficiently once offset maps are calculated. By using multi-pass rendering, complex and accurate shading can be added to relief texture mapping. We expected that the appearance of per-pixel shading functions enables the ability of real-time rendering more higher.

Because our implementation of the reflection mapping is eye constant, shading quality is poor. The texture shader has a dot-product-cube-map-reflection function which uses texture coordinates as view position. However current texture shader only supports per-vertex texture mapping input, not per-fragment.

In the gouraud shading, there is no difference in rendering speed between software implementation and our hardware-assisted implementation. If we implement pre-warping equation and re-sampling process with per-pixel shading functions, more speedup will be expected. Currently, we try to implement pre-warping equation by using texture shader and register combiner, but we suffer from lack of floating point accuracy, then our trivial implementation cannot get correct results.

References

1. J. F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (Proc. SIGGRAPH 78)*, pages 286–292. ACM Press, New York, 1978.
2. S. E. Chen. Quicktime VR — an image-based approach to virtual environment navigation. In *Computer Graphics (Proc. SIGGRAPH 95)*, pages 29–38. ACM Press, New York, 1995.
3. R. L. Cook. Shade trees. In *Computer Graphics (Proc. SIGGRAPH 84)*, pages 223–231. ACM Press, New York, 1984.
4. S. Dominé and J. Spitzer. Texture shaders. Game Developers Conference (NVIDIA Corp. presentation slide), 2001. http://developer.nvidia.com/view.asp?IO=texture_shaders.
5. E. Inc. BMRT: Blue moon rendering tools. http://www.exluna.com/.
6. A. Lippman. Movie-Maps: An application of the optical videodisc to computer graphics. In *Computer Graphics (Proc. SIGGRAPH 80)*, pages 32–42. ACM Press, New York, 1980.
7. L. McMillan. An image-based approach to three-dimensional computer graphics. Technical Report TR97-013, University of North Carolina at Chapel Hill, Computer Sceience Department, Apr. 1997.
8. M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *Computer Graphics (Proc. SIGGRAPH 2000)*, pages 359–368. ACM Press, New York, 2000.
9. C. Wynn. Using P-buffers for off-screen rendering in OpenGL. NVIDIA Corp. white paper, 2001. http://developer.nvidia.com/view.asp?IO=PBuffers_for_OffScreen.