ICAT 2001
December 5-7, Tokyo, JAPAN

Detailed Shape Representation with Parallax Mapping

Tomomichi KANEKO[1], Toshiyuki TAKAHEI[2], Masahiko INAMI[1],
Naoki KAWAKAMI[1],Yasuyuki YANAGIDA[3], Taro MAEDA[1] and Susumu TACHI[1]

[1]Information Physics and Computing,
School of Information Science and Technology, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656 Japan
{kaneko,minami,kawakami,maeda,tachi}@star.t.u-tokyo.ac.jp
[2] The Institute of Physical and Chemical Research (Riken)
2-1 Hirosawa, Wako-shi, Saitama, 351-0198 Japan
takahei@atras.riken.go.jp
[3]ATR Media Information Science Laboratories
2-2 Hikaridai, Seika-cho, Souraku-gun, Kyoto 619-0288, Japan
yanagida@atr.co.jp

Abstract

Although texture mapping is a common technique for adding apparent surface detail to 3-D objects, it lacks the capability to represent the motion parallax effect. So the mesh to which it is applied limits its realism. In this paper, we propose Parallax Mapping, a simple method to motion parallax effects on a polygon. This method has very fast per-pixel shape representation and can be performed in real-time with current hardware.

Key words: Displacement Mapping, Image-Based Rendering, Texture Mapping, Real-time Rendering, 3D Computer Graphics

1. Introduction

Among the domain of research in real-time computer graphics, there are two main approaches to render images of a scene: geometry-based rendering and image-based rendering. The former can generate images from different viewpoints with 3-D data sets, such as polygon meshes and their surface properties. However a complex scene may involve large data sets that can be prohibitive to process in real-time. So the developer must take care to maintain manageable data sets. On the other hand, for the latter approach, rendering time does not depend on the scene's complexity. It renders images from different viewpoints by interpolating between photographic images, so it is easy to generate photo-realistic output. However this approach tends to require larger data sets, and it sometimes restricts the viewpoint and scene structure. Further, its calculations tend to be slow because of the lack of graphics hardware acceleration.

Texture mapping [1][2] is a nice combination of the two approaches. You can easily represent the apparent detail of surfaces with simple image data, without increasing geometric primitives. It is processed extremely rapidly on today's graphics hardware, and its rendering cost is independent of the complexity of its pattern.

Although it is suitable for many cases, there are inadequacies. The quality of its output image depends on the geometric primitives it is applied upon. Of particular note, a texture image mapped onto curved or bumpy surfaces does not have curvature within a single polygon. This problem is magnified when inspecting a texture-mapped surface with stereoscopic vision, because the lack of binocular disparity between each mapped polygon is a depth cue that the surface it represents is flat.

There are two main reasons for this problem. First, texture mapping lacks the capability to represent a motion parallax effect – an apparent effect of relative movement of objects due to a viewpoint. So you cannot represent view-dependent unevenness or change of silhouettes. Second, while texture mapping works as a per-pixel table lookup, mapping coordinates are defined per-vertex from the geometry it is applied on. So even if you apply a texture image to a curved surface, the result looks flat from the linear interpolation of texture coordinates on each polygon.

To solve this problem, we proposed a Parallax Mapping method[3][4]. This method represents the motion parallax effect on a single polygon surface using per-pixel texture coordinate addressing. So you can get per-pixel level representation of view-dependent surface unevenness and silhouette change for each polygon. It uses a per-pixel image distortion process like image-based rendering, but it has a capacity to be accelerated by graphics hardware. So it is fast enough to be used with many polygons in geometry-based rendering.

2. Related Work

Texture Mapping is the addition of separately defined image data to a surface. Texture mapping hardware is

common today, and it works extremely fast even on affordable consumer graphics cards. However, traditional texture mapping only affects the color of surfaces, like painting on the wall. Silhouettes still depend entirely upon the geometry of the object as shown in Fig. 1.

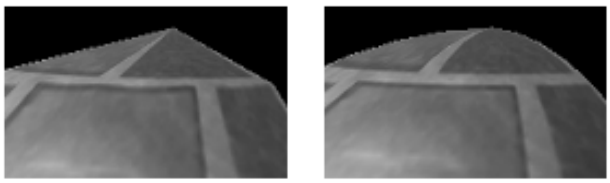

Fig.1 Per-vertex shape (left) and per-pixel shape (right)

Today, geometry-based rendering hardware is becoming cheaper and faster than ever, and we can render complex polygonal objects with texture mapping in real-time. However, from nearby viewpoints, it is evident from the rough, per-vertex-computed silhouette that the textures are only paintings on flat surfaces. To improve this appearance, there are many techniques to represent additional per-pixel details.

In real-time geometry-based rendering, Gouraud shading [5] is usually used for shading the surfaces, which computes lighting at each vertex, and interpolates vertex results across pixels. Recently there are some researches on real-time Phong shading [6], which computes the lighting at every pixel on the surfaces. These real-time techniques represent smoothness of the surfaces by hardware accelerated per-pixel lighting computations.

Bump Mapping [7] adds per-pixel relief shading to represent small unevenness of surfaces by perturbing the surface normal directions. It is useful to represent rough or etched surfaces, and today's some graphics hardware can compute it in real-time [8]. This technique, however, is only a shading effect, and the bumps are illusory. So it can't describe large unevenness, and the silhouettes of the applied polygons are still flat.

Displacement Mapping [9] actually displaces the surface in 3-D space. Although it can modify the silhouette and realize parallax effects, it needs many polygons to generate a transformation through height map data. It is a kind of modeling technique that is unsuitable for real-time rendering.

3. Parallax Mapping

In this paper we present Parallax Mapping, a real-time parallax distortion to represent detail shape on a single polygon. It uses per-pixel texture coordinate addressing to enhance its rendering quality and to enable the execution of per-pixel computations with graphics hardware. With this method, motion parallax is realized by mapping a texture that is distorted dynamically to correspond to the destination represented shape. Although the dynamic distortion of texture has been investigated in the domain of Image Based Rendering[10], techniques for real-time rendering have gone unreported. We realize this texture generation process not by distorting the actual texture, but by shifting texture coordinates of each drawn pixels as the texture is mapped to the polygon online. This process is accelerated by utilizing a graphics API supported by commonly used graphic hardware. Hereafter, the arguments for implementation are described.

3.1 Calculation of Texture Coordinate Shift

In our approach, used in geometry-based rendering, a scene consists of many polygons, and Parallax Mapping adds to them some additional details such as the displacement mapping in real-time. So the entire perspective distortions are processed in normal geometrical fashion, and the perspective correctness in the parallax is relatively less important. Especially for surfaces relatively small in a rendered image, or on billboard-type objects that always face the viewer, he can't distinguish between a parallax effect with perspective projection and one with orthogonal projection. So we introduce orthogonal projection to calculate the texture coordinate shift corresponding to motion parallax.

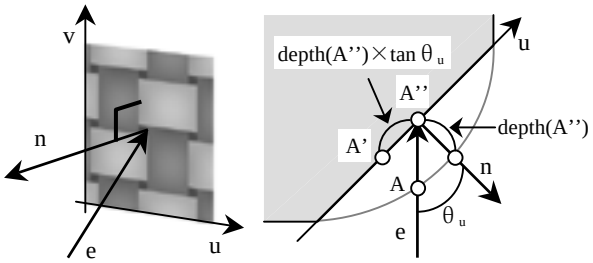

Fig.2 Geometry of texture coordinate shift

The left side of Fig.2 is the schematic of a texture coordinate system of an oblong polygon defined by the u, v axes. In this figure, n and e are the vectors of observation orientation, and the surface normal of the polygon, respectively. This polygon is supposed as a consistent of a part of curvature, and the curvature floated from polygon with depth named depth(u,v). The right figure is the image of projected left figure to the plane defined by u,n axes. Now, to express point A as on expected curvature, it is necessary that the texture coordinate of point A' is put on point A''. We set the angel θ_u between the texture axis u and the visual axis e, the translation of the shifted texture coordinates are described as followed.

$u' = u + \tan\theta_u \times depth(u,v)$

u : source texture coordinate of u axis

u': resulting texture coordinate of u axis

The angle which texture axis v and visual axis make is set θ_v, it is the same also about v axis.

$v' = v + \tan\theta_v \times depth(u,v)$
v : source texture coordinate of v axis
v': resulting texture coordinate of v axis

3.2 Depth Approximation

By the way, in the stage of rasterization, texture coordinate of point A'' is used to render point A. At this time, it need the depth value of point A' to calculate the texture coordinate of point A' , but we could get the depth value of A''. So, we suppose the expressed curvature is smooth and the angle which the normal of polygon and visual axis make is small, followed approximation is introduced.

$depth(u,v) = depth(u',v')$

3.3 Per-pixel texture Coordinate Addressing

In our Parallax Mapping method, we address texture coordinates at each pixel. For this implementation, we use Environment Mapped Bump Mapping (EMBM) supported on Microsoft® Direct3D® API [11]. The texture coordinate address of each pixel is computed by following equation:

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix} \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

where u, v are the source texture coordinates, and u', v' are the resulting texture coordinate address. M is a user-defined 2x2 matrix for scaling and rotating the texture coordinates, and Δu, Δv are delta values of texture coordinates defined in a second texture map (referred to here as a distortion map), applied on the same surface, to distort the source texture coordinates. EMBM was originally used to represent bump mapping effects by perturbing specular or diffuse environment maps. Some types of consumer graphics cards can accelerate this feature significantly, just like standard texture mapping.

The simplest implementation using per-pixel texture coordinate addressing does not create a new texture image by painting pixels in it, but sets the source texture coordinates in its distortion map. For example,

M : identity matrix
$\Delta u = u - u'$, $\Delta v = v - v'$.

It needs some extra processing, but it can enhance quality of resulting image. Because the distortion map is treated as a kind of texture image, when the distortion map is magnified in the rasterization process, it can be filtered with bilinear or trilinear interpolation. This interpolation filtering can be hardware accelerated, and processed at each pixel in the resulting image.

3.4 Implementation

From the argument so far, we implemented Parallax Mapping by introducing the following values into the equation of EMBM.

$M_{00} = \tan\theta_u$
$M_{01}, M_{10} = 0$
$M_{11} = \tan\theta_v$
$\Delta u = depth(u, v)$
$\Delta v = depth(u, v)$.

4.Estimating approximation distortion

Suppose that we make a texture mapped cylinder object with a rough prism using Parallax Mapping. Fig.3 shows the top view of this object. In this figure, the doted curved line is the side of the supposed cylinder, the gray blocks are polygonal prisms which surrounded by polygons, and the bold curves are the results distorted by our approximation. Here you can see that the rougher the polygonal object base, and the greater angle between the distorted texture mapped surfaces and the viewer, the more distortion occur.

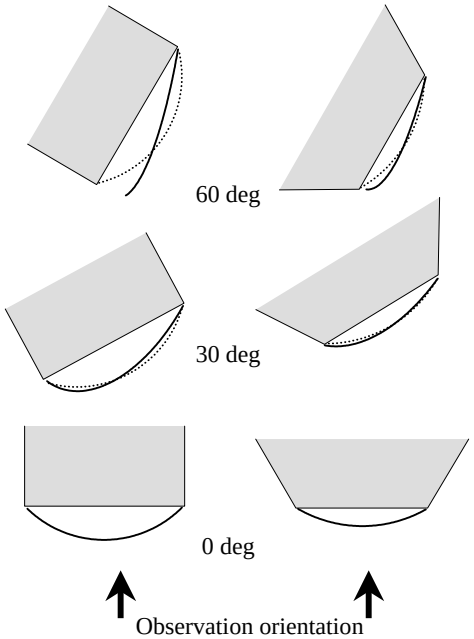

Fig.3 Distortion caused by depth approximation

5.Results

We used a desktop PC with Pentium® III 500MHz and ATI RADEON™ DDR 32MB graphics card. The resolution of the texture image was 128 x 128, 24-bit color depth. The corresponding depth data was 128 x 128, 8-bit depth. The output size was 640 x 480, 32-bit color depth. Fig.4 shows the rendering results of a Parallax Mapped single polygon compared to one with standard texture mapped. The result of Parallax Mapping shows the modification corresponding to motion parallax, and the expression of the smooth solid shape of the bricks. Furthermore, the drawing speed (434.50fps) is more than 80% of that of texture mapping alone(530.52fps).

An example that applied Parallax Mapping to the facial expression using a color picture and actual depth

information acquired with three-dimensional scanner is shown in Fig.5. The opposing pictures on the right and left are the results of drawing from the different observation angles, respectively. The upper set is the result of ordinary texture mapping and the lower set is result of Parallax Mapping. With an uncross-eyed stereo match to these pictures, a smooth form of a nose is perceivable from the results of Parallax Mapping. So it was shown that Parallax Mapping can express the per-pixel binocular disparity correctly.

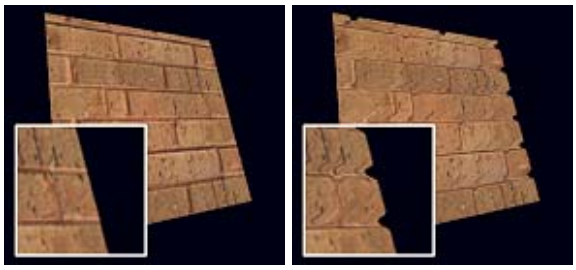

Fig.4 Results of Texture Mapping(left)
and Parallax Mapping(right).

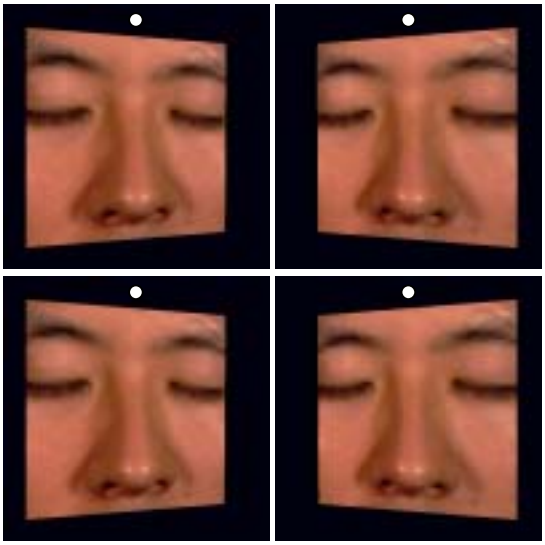

Fig.5 Examples for stereo matching
Upper: Texture Mapping, Lower: Parallax Mapping

Fig.6 is an example of applying parallax mapping to solid model using a cylindrical expression with 8-sided prism model. For comparison, the top part of each cylinder shows the result of texture mapping with a 32-sided prism model. This example shows the result for the case that the polygon is set to the interior of the cylinder. The left image is the result of ordinary texture mapping and the right image is the result of Parallax Mapping. There is mismatching of the texture pattern at the boundary of top and bottom parts of the left image. In contrast, the texture pattern is matched for a wide range on the right side. So by using Parallax mapping, it is shown that the shape of cylinder is represented correctly by per-pixel texture coordinate shift corresponding to motion parallax.

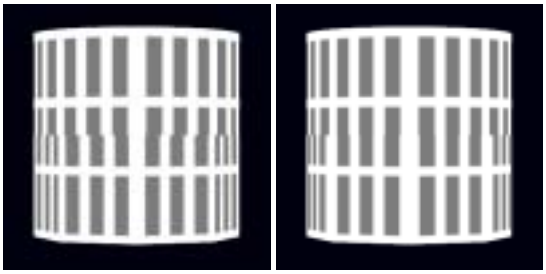

Fig.6 Examples of solid model
Texture Mapping(left) and Parallax Mapping(right)

6.Conclusion

We present Parallax Mapping to represent motion parallax on a single polygon in real-time. It simplifies the image-based approach in order to be fast enough to use with complex polygonal objects. It also enhances the quality of the rendering result by using per-pixel texture coordinate addressing. Attention must be given to the undesirable distortion from its approximation, but for the general case it performs extremely rapidly with today's graphics hardware acceleration.

References

[1]Ed Catmull: "A Subdivision Algorithm for Computer Display of Curved Surfaces", Ph.D. thesis, University of Utah, 1947.

[2]Paul S. Heckbert: "Survey of texture mapping", IEEE Computer Graphics and Applications, 6(11), 56-67, November 1986.

[3]T. Takahei, M. Inami, Y, Yanagida, T. Maeda and S. Tachi: "Real-time texture mapping method for stereo viewing", Proc. of VRSJ 5th Annual Conference, 189-192, September 2000 (in Japanese).

[4]T. Takahei, T. Kaneko, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda and S. Tachi: "Real-time 3D rendering method by using texture maping", Proc. of VRSJ 6th Annual Conference, 315-318, September 2001 (in Japanese).

[5]Gouraud, H.: "Continuous shading of curved surfaces", IEEE Transactions on Computers, 1971, 20, 623-628.

[6]Sim Dietrich: "Dot Product Lighting", Technical report, NVIDIA Corporation, www.nvidia.com, November 2000.

[7]James F. Blinn: "Simulation of wrinkled surfaces", Computer Graphics (Proc. of SIGGRAPH'78), volume 12, 286-292.

[8]Mark J, Kilgard: "A Practical and Robust Bump-Mapping Technique for Today's GPUs", Technical report, NVIDIA Corporation, www.nvidia.com, February 2000.

[9]Cook, R.: "Shade trees", Computer Graphics (Proc. of SIGGRAPH'84), 13(3), 223-231.

[10]Oliveira, M., Bishop, G. and McAllister, D.: "Relief Texture Mapping", ACM SIGGRAPH2000, 359-368.

[11] Microsoft DirectX , www.microsoft.com, 2000.