# Anisotropic Cone Mapping

Yu-Jen Chen and Yung-Yu Chuang
National Taiwan University
E-mail: {colac,cyy}@cmlab.csie.ntu.edu.tw

***Abstract*—This paper presents a more efficient and accurate method for inverse displacement mapping, methods which attempt to render effects offered by displacement mapping, motion parallax, self-occlusion, self-shadowing and silhouette, without actually perturbing the surface geometry. We first propose an extension to cone step mapping. The extension, Anisotropic Cone Mapping (ACM), provides a more efficient and accurate way for calculating intersections. Anisotropic cones are used as the bounding volume for the empty space above texels, allowing faster convergence and rendering speed. Experiments show that ACM has better rendering quality and convergence speed than previous methods. In addition, we propose a hierarchical adaptive preprocessing method that is around 10 times faster than conventional methods, allowing our space leaping approach to be used for rendering dynamic displacement maps. The proposed method is suited for many real-time 3D applications because of their low memory cost and good performance in both rendering quality and speed.**

## I. Introduction

Over the past few years, we have seen an impressive improvement in the capabilities of graphics processing units (GPUs). Among many revolutionary features, the most fascinating achievement is GPU's programmability for real-time realistic rendering. This capability allows us to synthesize many compelling effects in real time. To create appealing appearance, texture and bump mapping are often used to significantly improve the realism of rendered images by adding spatially-varying appearance due to surface details. These mapping techniques are widely adopted because they are suitable for GPU implementation. *Bump mapping* simulates surface details of uneven surface's interaction with light by adjusting normals to provide more realistic look [2]. However, bump mapping has its limitations. It only captures the shading caused by normal variation, but not the visually important effects such as motion parallax, self-occlusion, self-shadowing and silhouette.

*Displacement mapping* [4] provides a more effective representation for surface details with a displacement map instead of many small triangles. Points on a surface are moved along their normals by the amount specified by the displacement map. All the features that bump mapping fails to capture are naturally supported by displacement mapping. However, it requires altering geometry during shading. Even with today's graphics hardware, this is still a very expensive operation.

*Inverse displacement mapping* [9] is a class of mapping methods which attempt to render those effects offered by displacement mapping, such as *motion parallax*, *self-occlusion*, *self-shadowing* and *silhouette*, without actually perturbing the geometry of the surface being mapped. These methods have several advantages: they require lower amount of memory; they do not need to change the original geometry; and most importantly, they are performed in image space and can be efficiently implemented using fragment shaders on current GPUs. Recent progresses of inverse displacement mapping take advantage of the programmability and parallel nature of modern graphics hardware to render surface details in real time [11], [7], [3], [12], [1], [8], [5], [6]. One key question these methods attempt to answer is how to effectively search for the closest intersection to the surface defined by the displacement map along a given viewing ray. A compromise between convergence speed and rendering quality is often made by these methods.

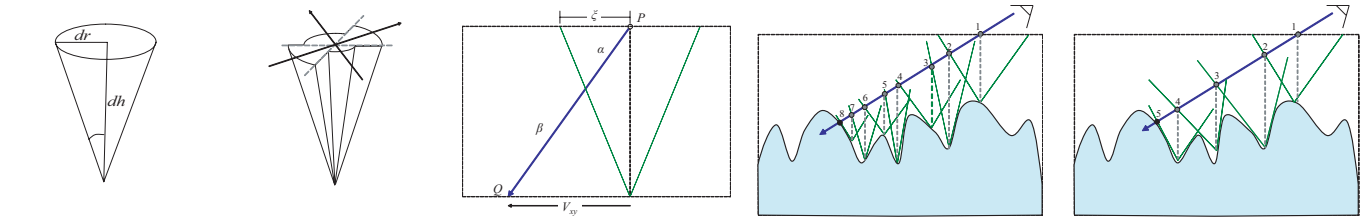
This paper proposes *anisotropic cone mapping* (ACM), which extends existing cone step mapping and provides a better tradeoff between rendering speed and guaranteed accuracy. Furthermore, we propose a fast preprocessing algorithm, which is around 10 times faster than previous methods. Thus, even requiring preprocessing, our space leaping method can be used for rendering scenes with dynamic displacement maps at a frame rate of 50 fps. The main contributions of this paper include: (1) an improved technique to provide better compromise between accuracy and rendering speed, and (2) a real-time adaptive hierarchical processing method, which allows rendering scenes with dynamic displacement maps in real time.

## II. Anisotropic Cone Mapping

Our method is built upon a previous method, cone step mapping (CSM), proposed by Dummer [6] on the game development forum. Section II-A reviews the main concept of CSM. Section II-B explains our extension to speed up CSM by using anisotropic cones. Next, we present our adaptive hierarchical preprocessing algorithm so that ACM can handle dynamic displacement maps (Section II-C). Finally, Section II-D discusses implementation issues.

### A. Cone step mapping

CSM belongs to the category of space leaping. Its bounding volume is a symmetric cone. For each texel, a cone is computed associated with it. It describes the largest empty space above that texel which can be bounded by a cone. This cone's tip is the texel itself. Hence, such a cone is described by a single parameter, cone ratio. The cone ratio $\xi$ is defined as the ratio between the radius and the height of the cone, i.e. $\xi = dr/dh$, measuring the opening angle of the cone as shown in Figure 1(a).




(a) Symmetric cone (b) Asymmetric cone (c) Ray-line intersection (d) Cone step mapping (e) Anisotropic cone mapping

**Fig. 1** Configuration of symmetric (a) and asymmetric cones (b). (c) illustrates our ray-line intersection algorithm. (d) and (e) show how CSM and ACM work.

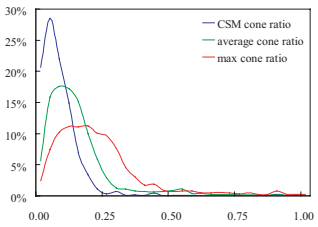



**Fig. 2** Histogram of cone ratio.

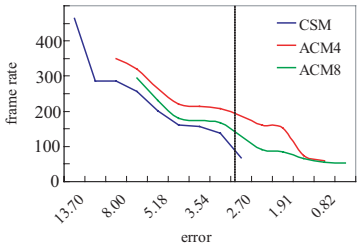



**Fig. 3** Comparisons of error and frame rate among CSM, ACM4 (4-sector ACM) and ACM8 (8-sector ACM).

The cone ratio is within the range of $[0, 1]$. It is partly because we can obtain better precision by mapping 256 levels of texture to a smaller range. More importantly, in practice, cone ratio is unlikely larger than one. For example, the blue curve in Figure 2 is the histogram of cone ratios for the brick texture. Most cone ratios are clustered below 0.25.

As shown in Figure 1(d), once we have the cone ratio for each texel, the exact intersection of the viewing ray and the displacement map can be found by repeatedly performing 2D ray-line intersections. Dummer used a more expensive procedure for finding 2D ray-line intersection. Here, we present a simpler and faster method as shown in Figure 1(c). Because of similar triangles, we know that $\alpha : \beta = \xi : |V_{xy}|$, where $V_{xy}$ is the vector $PQ$'s projection on the $x - y$ plane of the tangent space, $P$ and $Q$ are entry and exit points. Hence, the ratio $\gamma = \alpha/(\alpha + \beta)$ equals $\xi/(\xi + |V_{xy}|)$. Therefore, we should march $\gamma V_{xy}$ in the texture space. The procedure halts when either the solution converges or loop count exceeds a maximum limit.

### *B. Anisotropic cone mapping*

We propose a method to improve CSM by removing cone symmetry [1]. Many displacement maps have strong anisotropic property. It is too conservative to use the minimal cone ratio as the representative because other directions with larger empty space are often be compromised. Dummer suggested that ellipses probably could be for anisotropic displacement maps. However, it seems to be computationally more expensive. Hence, we propose to use asymmetric cones as the bounding volumes. An asymmetric cone has $n$ sectors and each has its own cone ratio. Figure 1(b) shows the configuration for a 4-sector asymmetric cone. The intersection finding procedure is similar to CSM, except that the cone ratio is now indexed by the viewing direction. Figure 1(e) shows such a procedure of ray marching.

One choice we have to make for ACM is the number of sectors. The use of more sectors obviously increase the cone ratio, thereby, offering faster convergence. Figure 2 displays the histogram of average and maximum cone ratios for 4-sector ACM. To compare the performance of CSM, 4-sector ACM and 8-sector ACM we performed an experiment to analyze the relationship between accuracy and rendering speed. We first generate a ground truth image using displacement mapping. Then, for each algorithm, we run it with 10, 20, 30, 40 and 50 steps and recorded the corresponding frame rate and its $L_2$ error from the ground truth. The results are shown in Figure 3. It is obvious that for a fixed accuracy, ACM4 is the fastest and CSM is the slowest. We observed that there is almost no visually noticeable difference when the error is below 3.0. The frame rates when error equals to 3.0 for CSM, ACM4 and ACM8 are roughly 100, 200 and 150fps respectively. Therefore, ACM4 is roughly two times faster than CSM. Though with better cone ratios, ACM8 is slower than ACM4 because of more texture accesses. In the following experiments, we will use ACM4 to represent ACM. Figure 4 compares the progression of convergence for CSM and ACM. Note that, on the discontinuity in the middle, CSM at 20 steps appears too flat. However, ACM at 20 steps is already very close to the final solution. It is obvious that ACM converges faster than CSM.

[1]Note that Policarpo and Oliveira have a similar extension concurrent to our work. They called it quad-directional cone step mapping [10].

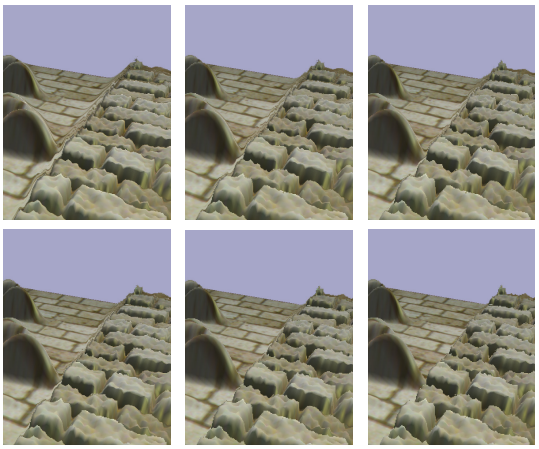



**Fig. 4** Comparisons of convergence between CSM and ACM. The top row shows convergence of CSM and the bottom row is for ACM. Note that, on the discontinuity border in the middle, CSM at 20 steps appears too flat. However, ACM at 20 steps is already very close to the final solution.

### C. Adaptive hierarchical preprocessing

The preprocessing of ACM is pretty straightforward. For each texel, we expand the search radius incrementally for each sector. For each expansion, several new texels are considered. The cone ratio is updated if the new texel offers a lower cone ratio. The current minimal cone ratio also provides the short circuit condition to terminate the search. This process takes on average 0.03 to 0.2 seconds for a $128\times128$ displacement map depending on its content. To apply ACM to dynamic displacement maps, we propose an adaptive hierarchical preprocessing algorithm (Algorithm 1) to speed up the process. We first build a pyramid for the displacement map using the max operator. That is, the height of the parent equals the maximal height of its four children. This is for maintaining the property that parents have more conservative cone ratios than their children. This property guarantees that, for a time-critical application, children can simply copy parent's cone ratio for conservative bounding volumes. The convergence could be a little slower with conservative ratios, but it is always correct. The ACM for the top level is calculated as usual. When calculating the cone ratios from top to bottom, we first compare whether a texel's height is similar to its parent's height. If so, then it simply copies its parent's cone ratio. Otherwise, it performs regular cone ratio search, but with its parent's cone ratio as the initial short circuit condition. This often terminates the search very quickly since we already have a very close estimation.

In practice, we found that it is often sufficient to set a high threshold for similarity in the above algorithm. It is because the resulted ACM is just more conservative but won't incur errors. Thus, we found the adaptive method seldom reaches the finest level. Table I shows the preprocessing time for three different displacement maps. The adaptive method offers more than 10x speedup by using three levels of hierarchy. The processing time is now below 0.02 seconds, making it possible to apply ACM to dynamic displacement maps.

| | original | adaptive | speedup |
|---|---|---|---|
| brick | 0.195 | 0.017 | 11.5 |
| rockwall | 0.186 | 0.014 | 13.3 |
| stone | 0.080 | 0.013 | 6.2 |

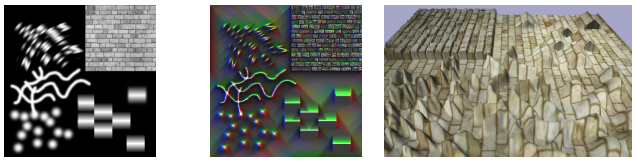
**TABLE I** Comparisons between the original preprocessing method and adaptive hierarchical preprocessing method.

**Algorithm 1 Adaptive ACM Preprocessing.** *Given a displacement map $H$, this procedure generates its ACM, $M_0$.*

**procedure** ADAPTIVEACM(DisplacementMap $H$)
  $H_0$=$H$; ▷ build a pyramid using the max operator
  **for** ($i=0$; $i\le l-1$; $i=i+1$) **do**
    $H_{i+1}$=CreateMap($\frac{1}{2}H_i$.width, $\frac{1}{2}H_i$.height);
    **for each** texture coordinate $(u,v)$ of $H_{i+1}$
      $H_{i+1}(u,v)=\max_{\Delta_u,\Delta_v\in\{0,1\}} H_i(2u+\Delta_u, 2v+\Delta_v)$;
    **end for**
  **end for**
  $M_l$=CreateACM($H_l$); ▷ regular cone ratio search for $H_l$
  **for each** ($i=l-1$; $i\ge0$; $i=i-1$) ▷ from top to bottom
    **for each** texture coordinate $(u,v)$ of $M_i$
      **if** Similar$\big(H_i(u,v), H_{i+1}(\frac{u}{2},\frac{v}{2})\big)$ **then**
        $M_i(u,v) = M_{i+1}(u,v)$;
      **else** ▷ regular cone ratio search at $(u,v)$ on $H_i$
        $M_i(u,v)$=ACM($u$,$v$,$H_i$,$M_{i+1}(\frac{u}{2},\frac{v}{2})$);
      **end if**
    **end for**
  **end for**
**end procedure**

### D. Implementation

We use an RGB$\alpha$ texture to store 4-sector ACM texture. Figure 5 shows the visualization for the ACM texture for the brick displacement map. For 8 sectors, we put two ACM textures side by side into a single texture. A fast GPU implementation for ACM needs care. Earlier, we implemented ACM using branch instructions to select proper sector according to the viewing direction and its performance was even worse than CSM. Later, we optimized the shader code by encoding almost all branch conditions into dot product computations as described by nVidia's shader tricks. It boosted the performance a lot. However, the most important performance gain of ACM comes from its larger cone ratios, allowing us to converge in less steps. This has big impact on performance. For current GPUs, there is a performance bound for texture accesses. When a shader has more texture accesses than this bound, the shader performance drops dramatically. Because ACM can converge with less steps, namely less texture accesses, it has better chance to reach the target accuracy before exceeding this bound than other methods.



(a) Displacement map (b) ACM texture (c) Rendering

**Fig. 5** ACM texture. (a) The brick displacement map. (b) The computed 4-sector ACM texture. (c) Rendering of mapping brick over a plane using ACM.

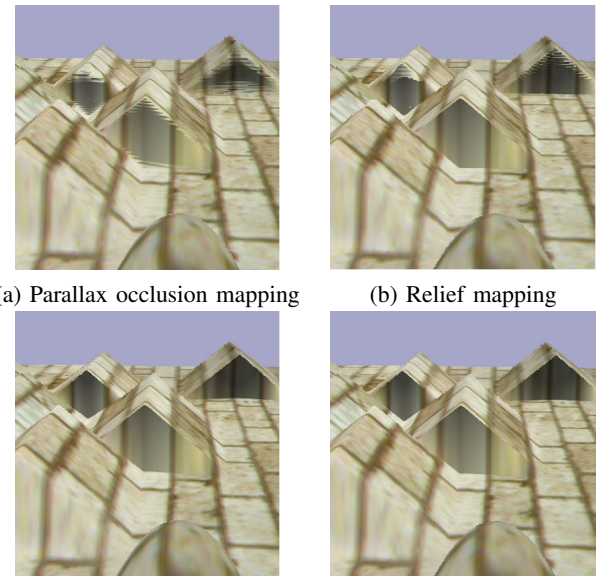
## III. Results

We performed experiments on a Pentium 4 3.0GHz machine with an NVIDIA GeForce 7900 GTX GPU with 512 MB memory. We used several displacement maps, exhibiting both high-frequency and low-frequency contents. All scenes were rendered at a resolution of $1024 \times 768$.

Figure 6 shows comparisons of *parallax occlusion mapping* (POM) [3], *Relief mapping* (RM) [11], CSM and ACM. POM and RM have an aliasing problem with discontinuity of height field. Hence, even with 200 steps, there are still stair-stepping artifacts. Because many steps are used, their frame rates drop to around 25 fps. On the other hand, CSM and ACM have less artifacts. Note that CSM-like approaches are not artifacts free but usually have less artifacts than root finding ones. As pointed out by Policarpo and Oliveira, CSM-like approaches could still have artifacts if number of steps is enough [10]. The frame rates of CSM and ACM are around 100 fps and 200 fps respectively. Figure 7 shows an example of applying dynamic displacement on a simple water simulation. We generate displacement maps at each time instance using simple sinusoids. The synthetic displacement map is then processed by our adaptive algorithm to obtain its ACM texture. These processes are performed on CPU. The displacement map and its ACM texture are then fed to the rendering system on GPU. The combined frame rate is around 50 fps including synthesizing height field, preprocessing of height field, texture transfer from CPU to GPU and rendering on GPU.

## IV. Conclusions

In this paper, we present an extension towards more accurate intersection for real-time inverse displacement mapping. The proposed anisotropic cone mapping provides better cone ratio than cone step mapping because of the use of asymmetric cones. Therefore, it usually requires less steps to converge. As a result, it is faster than CSM to reach a target quality. As relaxed cone stepping, combination of ACM with binary search could further reduce artifacts with a faster speed. In addition, our adaptive hierarchical preprocessing algorithm allows us to render scenes with dynamic displacement maps in real time. This approach can be extended to handle relaxed cone step mapping as well, greatly reducing the overhead of preprocessing.



(a) Parallax occlusion mapping (b) Relief mapping

(c) Cone step mapping (d) Anisotropic cone mapping

**Fig. 6** Comparisons of inverse displacement mapping methods.

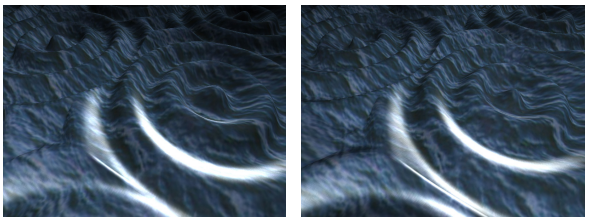


**Fig. 7** Real-time rendering with dynamic displacement maps.

## References


[1] L. Baboud and X. Décoret. Rendering geometry with relief textures. In *Proceedings of Graphics Interface 2006*, 2006.

[2] J. F. Blinn. Simulation of wrinkled surfaces. In *Proceedings of ACM SIGGRAPH*, pages 286–292, 1978.

[3] Z. Brawley and N. Tatarchuk. Parallax occlusion mapping: Self-shadowing, perspective-correct bump mapping using reverse height map tracing. In *ShaderX$^3$: Advanced Rendering with DirectX and OpenGL, W. Engel Ed.*, pages 135–154. Charles River Media, 2005.

[4] R. L. Cook. Shade trees. In *Proceedings of ACM SIGGRAPH 1984*, pages 223–231, 1984.

[5] W. Donnelly. Per-pixel displacement mapping with distance functions. In *GPU Gems 2, Matt Pharr Ed.*, pages 123–136. Addison-Wesley, 2005.

[6] J. Dummer. Cone step mapping. http://www.lonesock.net/files/ConeStepMapping.pdf, 2006.

[7] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, and S. Tachi. Detailed shape representation with parallax mapping. In *Proceedings of the ICAT 2001*, pages 205–208, 2001.

[8] A. Kolb and C. Rezk-Salama. Efficient empty space skipping for per-pixel displacement mapping. In *Proceedings of Vision, Modeling and Visualization*, 2005.

[9] J. W. Patterson, S. G. Higgar, and J. R. Logie. Inverse displacement mapping. *Computer Graphics Forum*, 10:129–139, 1991.

[10] F. Policarpo and M. M. Oliveira. Relaxed cone stepping for relief mapping. In *GPU Gems III*, pages 409–428. Addison-Wesley, 2007.

[11] F. Policarpo, M. M. Oliveira, and J. L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of Symposium on Interactive 3D Graphics and Games 2005*, pages 359–368, 2005.

[12] E. A. Risser, M. A. Shah, and S. Pattanaik. Interval mapping. Technical Report, School of Engineering and Computer Science, University of Central Florida, 2006.