Real-time Soft Shadowing of Dynamic Height Map Using a Shadow Height Map

Sung-Ho Lee[O] Chang-Hun Kim
Korea University
(pocorall, chkim)@korea.ac.kr

그림자 높이 맵을 이용한 실시간 그림자

이성호[O] 김창헌
고려대학교 컴퓨터학과

Abstract

This paper introduces a novel real-time soft shadowing method applicable for height maps. As well as supporting self-shadowing of the height map, our method allows shadows to be caught on other objects. The method is very suitable for dynamically changing height maps because it requires no precomputation. A shadow height map (SHM) is a new structure which represents the height of the shadow at each discretized coordinate of a height map. Constructing the SHM is O(n), where n is the number of texels in the SHM. Shadow can be computed from this map quickly and simply, using a pixel shader. Examples demonstrate good real-time performance and plausible visual quality.

요 약

본 논문에서는 지형 맵에 실시간으로 부드러운 그림자를 드리우는 방법에 대해 소개한다. 이 방법은 셀프 셰도우 뿐만 아니라, 다른 물체에 그림자를 드리우는 것도 가능하다. 또한 이 방법은 지형에 대해 미리 계산해 두는 과정이 없기 때문에 지형이 변화하는 경우에도 적합하다. Shadow height map(SHM)이라는 새로운 자료구조는 높이 맵의 각 좌표에 해당하는 그림자의 높이를 기록한다. n 이 SHM 안에 있는 텍셀 수라고 할 때, SHM 을 계산하는 것은 O(n)의 알고리즘 복잡도를 가진다. 그림자는 이 맵을 이용해 단순하고 빠르게 계산할 수 있다. 예제들은 실시간으로 표현할 수 있는 좋은 성능을 보이며, 적절한 시각적 품질을 보인다.

키워드 : 실시간 그림자; 부드러운 그림자
Keywords : *Real-time shadow; soft shadow*

1. Introduction

Shadow provides important visual cues about a scene. Many methods of computing shadows at interactive rate have been proposed. However, shadows are sparingly used in practice because they impose significant performance degradation on the application. Shadowing in real time is still a challenging problem.

We focus on shadowing for height map structure. Height maps are widely used to render terrain [3], bump mapping [1] or relief mapping [7][12]. A height map is tessellated into huge number of triangles. Therefore shadowing methods designed for arbitrary triangles such as shadow volume [2] and shadow mapping [13] are too slow. For this structure, horizon mapping [4][9] and ambient aperture lighting [6] can be considered. But these methods require precomputation, and do not allow casting shadows on other objects. We propose a new method with the following features:

- Cast shadows on other objects.
- No precomputation is required.
- Fast computation.
- Can generate hard or approximated soft shadows.

Because our method does not require precomputation, the height map can be changed dynamically, allowing to be applied to dynamically changing height map surface such as water and bombarded terrain or incorporated in interactive modeling tools.

In most practical applications, other objects are placed on the height map. It is indispensable to cast shadows to the objects near the surface. While other methods [6][9] do not allow this, our method allows it by simple implementation.

We introduce shadow height map (SHM), which represents the height of the shadow at each coordinate on a grid. The SHM is calculated in the CPU, and transferred to the GPU as a texture. Shadow testing can be performed in a pixel shader. There is no difference between testing the visibility of the height map itself and that of other objects. The same shader program and SHM texture are used in each case.

Casting shadows in parallax occlusion mapping [12] is usually performed by ray marching with a given sampling step, and then testing whether the incremented ray intersects with the height map. This needs to be performed for every pixel on the screen individually. Although this testing loop can be implemented in the GPU, that would make it a major performance bottleneck. There is a chance to optimize this hot spot because several different pixels invoke the intersection testing subroutine for the same location during the ray marching.

In computing the SHM, rays march from the light, not from the fragment. Calculating shadow height requires no subloop for a location because it utilizes ray marched result of previous step. The process is similar to dynamic programming.

Although a SHM does not deal with area light source, soft shadows can be approximated by specifying penumbra depth. The shadow has constant depth of penumbra. Although this is not a physically accurate model, the result is visually plausible.

2. Related works

Our method calculates height of shadowed area as a map. It is similar to shadow maps [13] which are aligned to the direction of the light, but the SHM is aligned to accompanying height map. Shadow mapping requires additional rendering pass to construct the map, but calculating the SHM only depends on the height map, and is done by much simple arithmetic than that.

A horizon mapping [4][9] precalculates horizon of sampled directions for each texel. Usually, the region is sampled in 8 directions, and horizons in other directions are approximated by a circular basis function [9]. Sharp features, such as the shadow from a thin tall tower, can be missed by this procedure.

Precomputed radiance transfer (PRT) [10] allows interesting effects, such as soft shadows, subsurface scattering, glossy materials, and interreflection to be achieved in real time. In recent works [8] shadows can be casted onto other objects within a dynamic scene. However, these methods are not feasible for a large scale height maps because the memory requirements are large or some properties (light, view, or geometry) must be assumed to be static. Moreover, PRT is not feasible to cast sharp shadow.

It is possible for objects to cast shadows onto other objects by storing the transferred radiance for ambient space [5]. But the resulting shadows are too smooth and a lot of memory is required to sample ambient radiance in 3D.

Ambient aperture lighting [6] is a less computationally intensive approach than PRT. A spherical cap is precalculated for each location, and a visibility test is performed in real time using only simple arithmetic. It is a fast way to calculate soft self-shadows, but it still requires precomputation.

3. Shadow height map

We define a shadow height map $y = S(x,z) \in \mathbb{R}$ that stores height information of shadow at location $(x,z) \in \mathbb{Z}^2$. A SHM can be discretized as a texture that has the same resolution and transformation with an accompanying height map (Figure 1). A separate SHM is required for each light source and a SHM must be recalculated when either the direction of its light source or the height map is changed. If the value of the height map $H(x,z)$ is smaller than $S(x,z)$ at some location, than that location is shadowed.

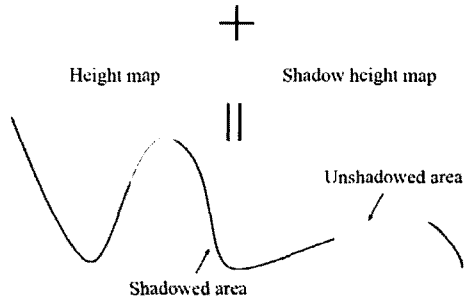


Figure 1: Shadow height map stores the height values in which we can start to see the light source for each location on a height map.

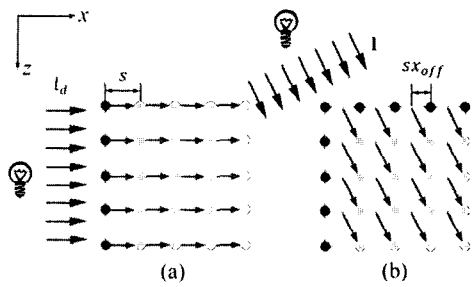


Figure 2: Calculate the values of the closest locations from light source (red dots) first, and propagate them. There are two types of SHM. (a) Type 1: axis aligned directional light. (b) Type 2: a directional light.

The method of calculating a SHM depends on the light source. We will consider two types of light, looking first at the simplest case.

3.1. Calculating SHM

A type 1 shadow is cast by a directional light aligned with the axis of the SHM. The SHM function $S_1(x,z,l_d,l_a)$ has four parameters: two integers for the coordinate (x,z), the direction of the light l_d and the tangent of altitude of the light l_a. Because the calculation is the same for the 4 cardinal directions of l_d (E, W, S, and N), we will only describe the case of light coming from the west (Figure 2 a). And for more simplicity, we will assume unit distance between each texel. The SHM can be calculated as

$$S_1(0,z) = H(0,z),$$
$$S_1(x,z) = \max\{H(x,z), S_1(x-1,z) - sl_a\}, \quad (1)$$

where $x \neq 0$ and s is a scaling constant. The height of the shadow is the larger value between height of current terrain and propagated height of shadow. Because Equation (1) is recur-

rence form with respect to x, we can calculate entire map from x=0.

A type 2 shadow is cast by a light coming in any direction. The direction of the light is the three-dimensional vector $\mathbf{l} = (l_x, l_y, l_z)$, where l_x and l_z are the texture coordinates and l_y is the direction of height. When $0 < l_x < l_z$ (Figure 2 b), the SHM is

$$S_2(0,0) = H(0,0),$$
$$S_2(0,z) = H(0,z),$$
$$S_2(x,0) = H(x,0),$$
$$S_2(x,z) = \max\{H(x,z), S_2(x - x_{off}, z-1) - sk\},$$

where $x \neq 0, z \neq 0, x_{off} = l_x/l_z$ and $k = l_y/l_z$. Although the formula looks complicated, the principle is the same with type 1 shadow.

Because $0 < x_{off} \leq 1$, the location of a texel is not quantized, it must be linearly interpolated. The equations above only cover 1/8 of the possible directions. The equations for the other directions are similar.

3.2. Determining shadow from the SHM

Testing light visibility is simple. A given position $\mathbf{p} = (p_x, p_y, p_z) \in \mathbb{R}^3$ is shadowed when $p_y < S(p_x, p_z)$. Because SHM has information at discretized coordinate, bilinear interpolation is used. This filtering is done by GPU.

However, depth fighting can occur due to round-off errors and filtering. We tried to introduce a shadow depth offset, which is common way to accommodate round-off error. But the approximation error due to texture filtering is more severe. So we devised another method, which not only accommodates round-off error, but also simulates soft shadow. We introduce a penumbra depth d_{pen} . When $S(p_x, p_z) - d_{pen} < p_y < S(p_x, p_z)$, we linearly interpolate the shaded color between the brightest color $c_{lightened}$ and the ambient color $c_{ambient}$:

$$\alpha = \frac{S(p_x, p_z) - p_y}{d_{pen}},$$
$$c = \alpha c_{ambient} + (1-\alpha) c_{liqntened},$$

where c is the result color which is applied in the shadow.

4. Implementation

Because our algorithm uses two dimensional arrays, correct utilization of the CPU cache is important. Random access to the memory leads to many cache miss and results in severe performance degradation. Reading and writing to this memory should be as sequential as possible. We parameterized a two-dimensional square map A to memory addresses M as follows; $A(x,z) = M(x + zw)$ where w is the width of the map. The form of the calculation differs depending on the direction of the incoming light. Figure 3 and the pseudocode below describe the calculation for the first two quadrants.

```
Q := quadrant of direction of incoming light
If(Q=1) {  // light from quadrant 1
  For(each z from 0 to w) {
    For(each x from 0 to w) {
      Calculate S(x, z)
    }
  }
}
If(Q=2) {  // light from quadrant 2
  For(each z from 0 to w) {
    For(each x from w to 0) {
      Calculate S(x, z)
    }
  }
}
// Code for other quadrants is similar.
```

We observed that this iteration order utilizes cache more than traversing z-axis as an inner loop.

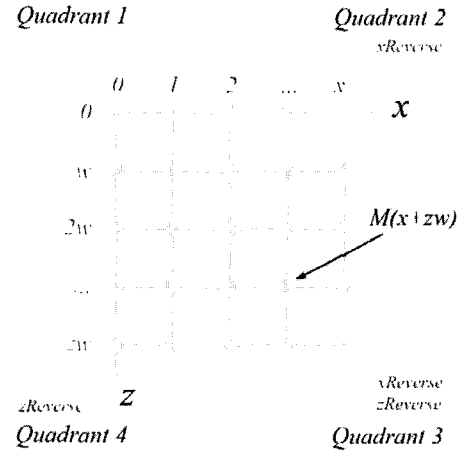


Figure 3: Orders of array parameterization and shadow calculation: Q2 and Q3 iterate along the *x-axis* in a negative direction, Q3 and Q4 iterate along the *z-axis*, also in a negative direction.

We implemented the algorithm with the C++ and the Cg language, and used the Ogre3D graphics library. The SHM is stored as a 32-bit floating-point type texture. However, only the latest graphics processors support bilinear texture filtering for 32-bit floating-point texture. Commodity hardware supports 16-bit floating-point texture filtering, but an additional overhead is required for the conversion.

The shadow receiver program determines whether a position is shadowed or not. It can be implemented as either a vertex or a pixel shader program. Because our algorithm is simple and current GPUs provide enough power for a pixel shader, we implemented it in that form. The source code is provided in the appendix.

5. Results

All the images presented in this paper are rendered with a GeForce 8800GTS which supports 32-bit floating-point texture filtering. The host system was an Intel Core 2 Duo 2.4GHz CPU with 2GB of system memory. Our program is single threaded.

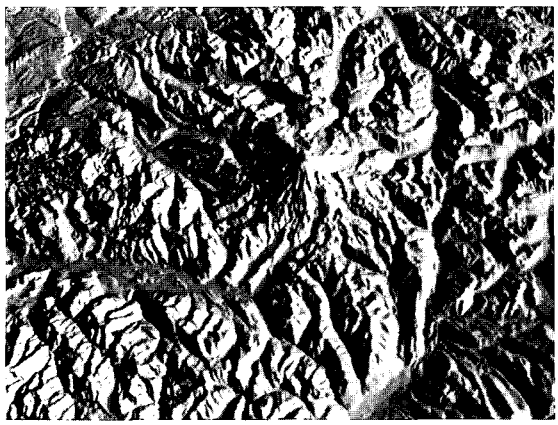
We rendered terrain data sampled at various resolutions (Table 1). For comparison, we rendered the same data without

using our algorithm, and also tried transferring the texture to the GPU at every frame only without computing the shadows to measure the bandwidth overhead.

The most expensive operation of building the SHM is memory access in a loop. Type 2 shadows read three numbers from memory and write one number to it for each texel. As seen in Table 1, severe performance degradation which exceeds algorithm complexity is observed at the highest resolution, because of poor CPU cache hit. This is less noticeable in a resolution of 512^2 or less, which utilizes the cache enough.

SHM dimension	256^2	512^2	1024^2
Triangle count	132K	528K	1.6M
No shadow (fps)	765	382	122
Texture transfer (fps)	719	330	83
Type 1 shadow (fps)	711	325	76
Type 2 shadow (fps)	708	308	49
Shadow volume [2] (fps)	212	58	17

Table 1: Frame rate of algorithms for data at various resolutions. Our method is significantly faster than the shadow volume [2]. A screen resolution is 1600 x 1200.



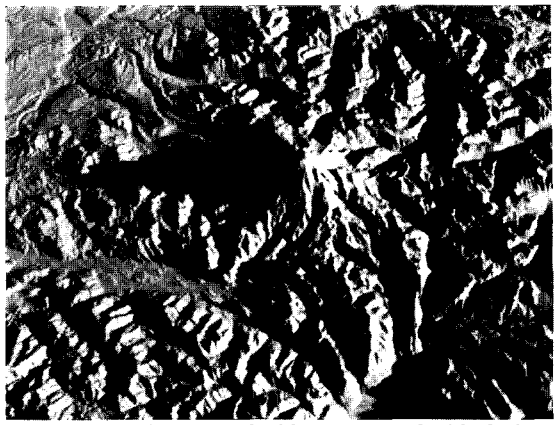


Figure 4: Terrain rendered without (top) and with shadow (bottom).

Figure 4 compares the performance of rendering using an N dot L lighting and model with our shadowing method. The more extensive and more realistic shadowing is clearly apparent.

Figure 5 and the accompanying video show an example of shadowing on a changing height surface. Using the SHM, this involves no extra cost.

Figure 6 compares results of soft shadowing with various penumbra depth values. When calculating a type 2 SHM, a linear interpolation is involved to propagate shadow height. It is not exact because the interpolation diffuses the height map. But this can be a good chance to simulate soft shadow. By simply specifying enough penumbra depth, soft shadow can be shown with very little cost.

Figure 7 shows an example of shadows being cast on to other objects. The same shadow receiver program is used for both the terrain and separate objects. Notice that the penumbra effect is also present in the shadow casted on the house in this figure.

Figure 5: Results of shadowing a dynamically changing height map (see the video).

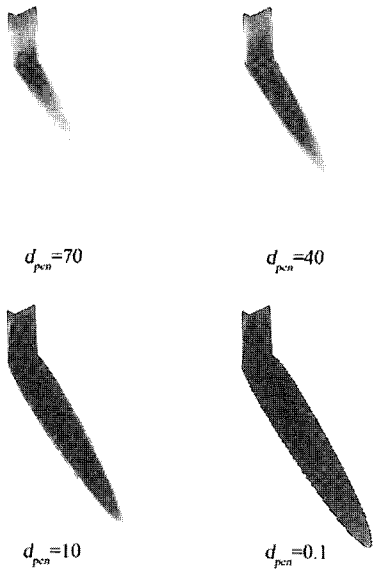


Figure 6: A penumbra depth allows soft shadow. This involves only one linear interpolation.

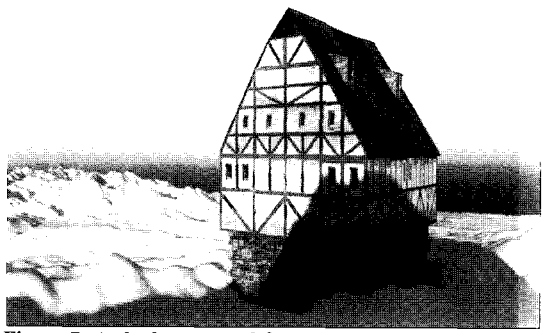


Figure 7: A shadow created from a height map is casted on to a separate object.

Some objects placed on the ground can be approximated to the height field. We construct the height field according to height of the object, and 'cap' the object on the height field, as shown in inset image of Figure 8. Then the height field cast a shadow which roughly mimics the shadow of the object. This method is especially useful for games and interactive city environment modeler, because objects can be dynamically placed and removed.

6. Summary and future work

We have introduced a novel method of modeling soft shadow from a height map and casting it on to an arbitrary object. This method is simple to implement, fast, and it does not require any precomputation.

A noticeable drop in performance occurs because of the bandwidth bottleneck when the SHM texture is transferred from the system memory to GPU memory for every frame. Another performance hot spot is accessing main memory in CPU. Because the GPU is more suited to texture access, calculating the SHM on a GPU would greatly improve the performance. But implementing it is a challenging problem because of sequential nature of the algorithm.

For visualizing high resolution terrain data, several level-of-detail methods are developed such as [3]. Adapting SHM for this type of data structures would be an interesting extension.

Policarpo et al. [7] has shown that a relief mapped surface can correctly intersect with another object. Although we did not implement the SHM for relief mapping, we expect the relief geometry could successfully cast shadow on to an intersected object. This may be demonstrated in future work.

References

[1] Blinn J. F., Simulation of Wrinkled Surfaces. ACM SIGGRAPH, 1978, 286-292.

[2] Crow F., Shadow Algorithms for Computer Graphics, ACM SIGGRAPH, 1977, pp. 242-248.

[3] Losasso, F., and Hoppe, H., Geometry clipmaps: terrain rendering using nested regular grids. ACM SIGGRAPH, 2004, 769-776.

[4] Max N. L., Horizon mapping: shadows for bump-mapped surfaces. The Visual Computer 4, 2, 1988, 109–117.

[5] Oat C., "Irradiance Volumes for Games," Proc. Game Developers Conference, 2005.

[6] Oat C. and Sander P. V., Ambient Aperture Lighting. Proceedings of 2007 simposium on interactive 3D graphics and games, 2007, 61-64.

[7] Policarpo F., Oliveira M. M., and Comba J., Real-Time Relief Mapping on Arbitrary Polygonal Surfaces. ACM SIGGRAPH, 2005, 935.

[8] Ren, Z., Wang, R., Snyder, J., Zhou, K., Liu, X., Sun, B., Sloan, P.P., Bao, H., Peng, Q., and Guo, B., Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. ACM SIGGRAPH, 2006, 955-966.

[9] Sloan P.-P. J. and Cohen M. F., Interactive Horizon Mapping. Proc. Eurographics Workshop on Rendering, 2000, 281-286.

[10] Sloan P.-P. J., Kautz J., and Snyder J., Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. ACM SIGGRAPH, 2002, 527–536.

[11] Sun, W., and Mukherjee, A., Generalized wavelet product integral for rendering dynamic glossy objects. ACM SIGGRAPH, 2006. 955–966.

[12] Tatarchuk N., Dynamic parallax occlusion mapping with approximate soft shadows. In Symposium on Interactive 3D graphics and games, 2006, 63-69.

[13] Williams, L., Casting Curved Shadows on Curved Surfaces, ACM SIGGRAPH, 1978, 270-274.

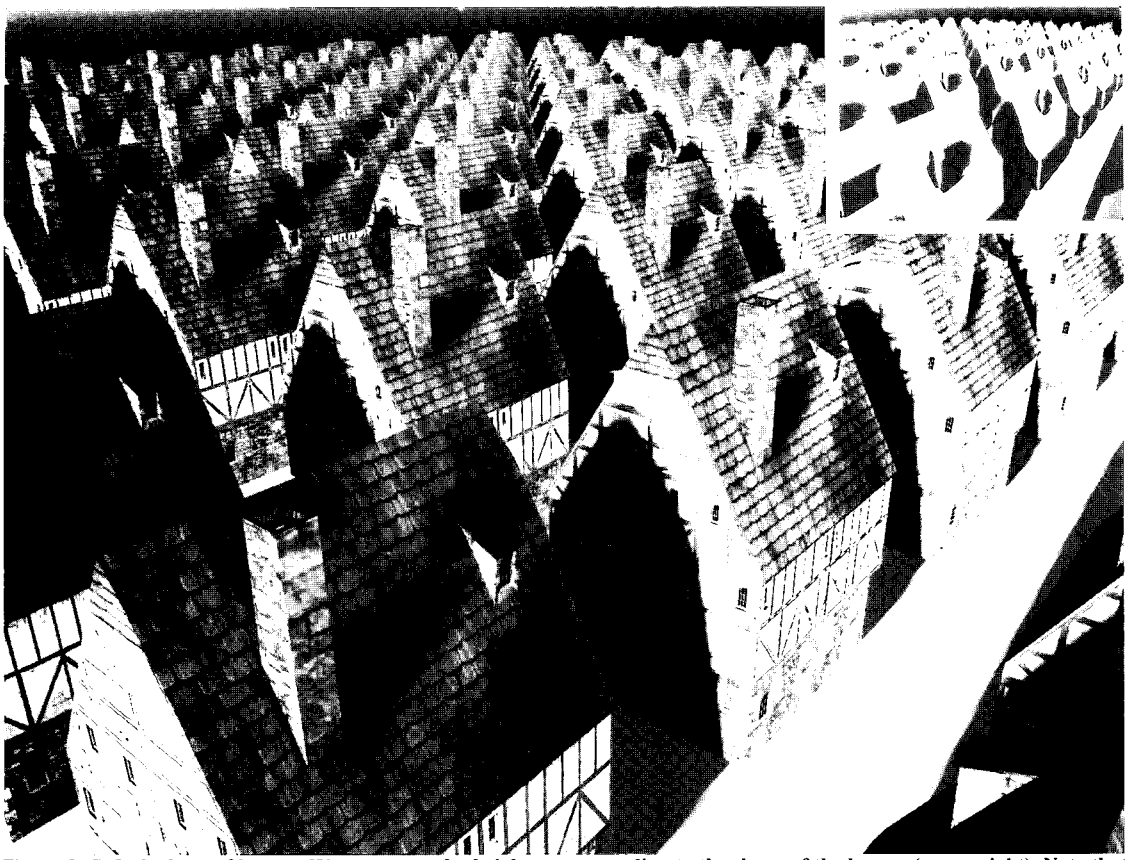


Figure 8: Soft shadows of houses. We constructed a height map according to the shape of the houses (upper right). Note that the soft shadow of the chimney is casted naturally on the roof. This scene is rendered at 220fps with a resolution of 1600 x 1200. A size of the SHM is 512^2.

Appendix: Shadow receiver Cg pixel shader program

```
void main_fp(float3 position : TEXCOORD0,
             float2 texCoord : TEXCOORD1,
             float shade : TEXCOORD2,
             out float4 color : COLOR,
             uniform float ambient,
             uniform float penumbraDepth,
             uniform sampler2D heightTexture,
             uniform sampler2D diffuseTexture)
{
// position is pre-scaled to heightTexture coordinate
float shadowHeight = tex2D(heightTexture, position.xz).r;
float penumbra = clamp((shadowHeight-position.y)/penumbraDepth, 0, 1);
float lightness = ambient+ lerp( shade, 0.0, penumbra);
color = lightness * tex2D(diffuseTexture, texCoord);
}
```