The High Resolution Hemicube Algorithm

Karel Nechvíle
Masaryk University
Botanick 68a, 602 00 Brno, Czech Republic
e–mail: kodl@fi.muni.cz

Abstract

In a process of the radiosity computation, the most demanding part is to evaluate a form factor (FF). Therefore, a lot of investigation has been done in order to speed up the part while preserving accuracy of the result. One of the first methods used for the form factor evaluation was the hemicube (HC) approach. Nevertheless, the original HC method still consumes too much time and memory, so that high resolution could be afforded only on really powerful hardware. In this paper we show, how can this method be used for fast and accurate form factor computation. We have implemented and compared several variants of the original scheme and propose a new algorithm which drastically reduce memory requirements of FF evaluation.

1 Introduction

The well-known hemicube approach [1] represents one of the first applicable methods for the form factor computation. However, this solution has several limitations (e.g. appearance of alias). Increasing the HC resolution can solve the problem, but it raises HW requirements as well. Therefore, the original method was modified [2] and more practical ways were found [3]. Efficiency of the methods depends also on a space sorting process. BSP trees [4] are frequently applied for this purpose in recently published algorithms [5, 6, 7], We show how BSP trees and hemicube algorithm can be merged to the fast and accurate computation.

2 Hemicube and BSP

In the classical HC method patches in environment are projected onto the faces of a hemicube. Patches are clipped against the hemicube edges to obtain the set of pixels the patch projects onto. The occluding or intervening problem is solved by maintaining and comparing the depth coordinate like in the Z-buffer algorithm. In addition to patch ID the HC pixel contains also the depth information (fig. 1a).

The lookup table for pixel values is maintained, too. Pixel values called delta form factors determine the form factor value for a particular pixel of projection. Hence, the memory requirements for higher HC resolutions limit applicability of this method. Insufficient memory can cause too much swapping, which prolongs or even stops the computation.

A way how to decrease the memory requirements is to involve visibility technique different from the Z-buffer algorithm. In our approach we use the BSP tree, which makes possible to determine the visibility from arbitrary viewpoint [4]. Thus, the memory requirements for a HC pixel are reduced to a patch ID (fig. 1b) and the time requirements for projecting a patch onto a HC face are also lowered. We will not discuss the memory requirements for BSP tree here. They depend on input scene polygon complexity and are small relatively to the HC memory requirements for medium complexity scenes. They were only about 1 kB in our test scenes.

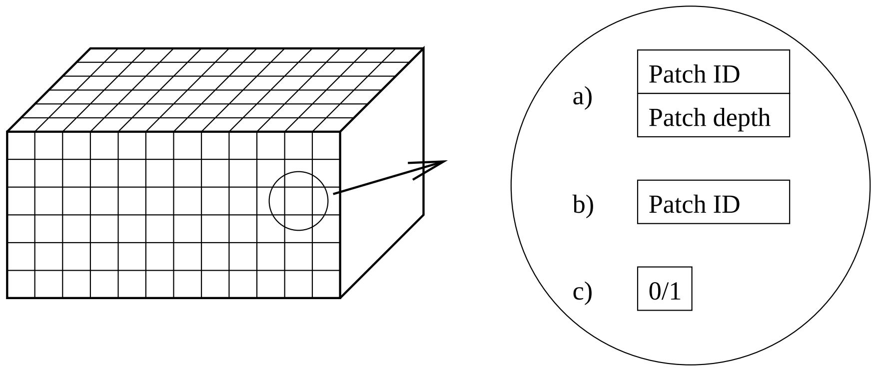

Figure 1: Memory requirements for the HC pixel

In the Z-buffer algorithm, depth coordinate needs to be interpolated first along the edges, then along HC raster line. Within the BSP based approach, the scan-line interpolation is not necessary. Unfortunately, the speed-up reached cannot be fully estimated, as it depends on HC resolution and environmental geometry. In our tests, depth interpolation steps make about 85 % of all interpolation steps. This number does not represent exactly the Z-buffer overhead, because some initialization routines needed are not taken into account.

In altered process, the faces are projected onto the hemicube using BSP tree. Back-to-front approach solves the visibility problem. When all patches have been projected, delta form factors are summed forming the patch form factors. New form factors are used for radiosity values update in shooting step.

3 New Front-to-Back Approach

The back-to-front approach is intuitively more efficient than the front-to-back. It is not necessary to test each pixel when projecting a patch. But if we decide to involve the front-to-back approach, we can take advantage of following properties.

We know, from the projection fashion, a patch will be projected only to free pixels. After processing the patch, no more patches can be projected on occupied pixels. For a patch, we need to determine its form factor. A form factor is computed from projected pixels.

Using front-to-back approach we can compute the form factors simultaneously while projecting patches onto the hemicube. When we find a free pixel, we sum the delta form factor for a patch and change the pixel FREE/OCCUPIED value. In this manner, maintaining the label part of the pixel is no more necessary. All we need to know is, whether a pixel is free or occupied. Space requirements for a HC pixel are, therefore, reduced to one bit (fig. 1c).

For example, if we count 32 bits for a patch ID and for a depth value, 64 bits for a delta form factor, 2048×2048 resolution then the HC memory requirements are 108MB, 60MB, and 24MB for the Z-buffered, BSP, and new bit buffered approach respectively.

4 Implementation and Results

We have implemented the FF evaluation using hemicube and BSP tree. As an alternative we also implemented the classical HC algorithm, which use the depth coordinate to solve the visibility problem. This method is signed as `Z-buffer` in graph 4. In the case of applying the BSP tree, following approaches were tested: back-to-front, front-to-back and bit-buffered front-to-back. The implementations of front-to-back and back-to-front methods are straightforward. For bit-buffered front-to-back approach more sophisticated way is needed.

The FREE/OCCUPIED value is stored as a bit in an integer. When projecting a patch the corresponding memory fragment for bit operations is determined (fig. 2a). The bit mask for a polygon is built up (fig. 2b) and free pixels are found out by applying the bit operations (fig. 2c). The delta form factors are summed to a patch

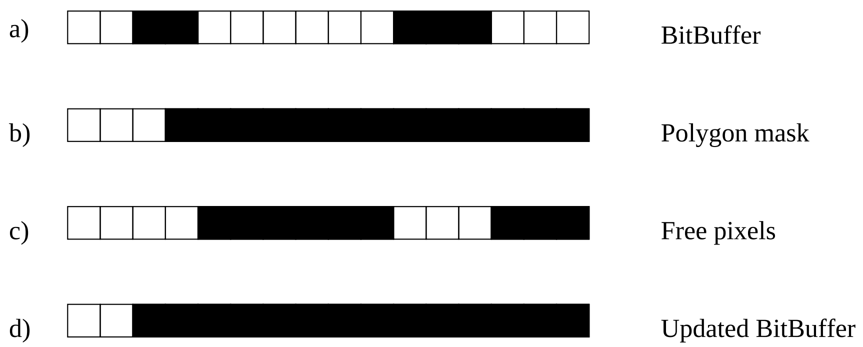

Figure 2: Bit operations

form factor for free pixels. Pixels covered by the polygon are signed as OCCUPIED and written back to the bit buffer (fig. 2d). This process is sequently applied for all integers overlapped by the projecting polygon.

Usually the whole integer overlapped by a polygon is processed. An optimalization can be achieved if we use a special code when manipulating only small part of the memory. This situation occurs when we project some polygon part that is close to

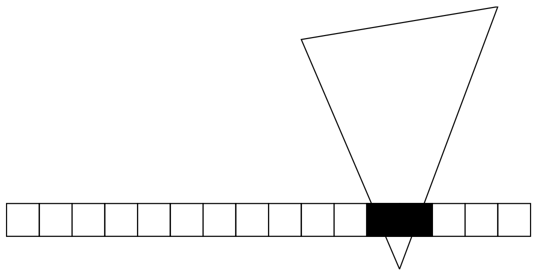

Figure 3: A polygon overlapping small part of the bit buffer

the polygon vertex (fig. 3). In this case the whole processed scan line often lies within one integer. We can use bit shift operations for faster treatment.

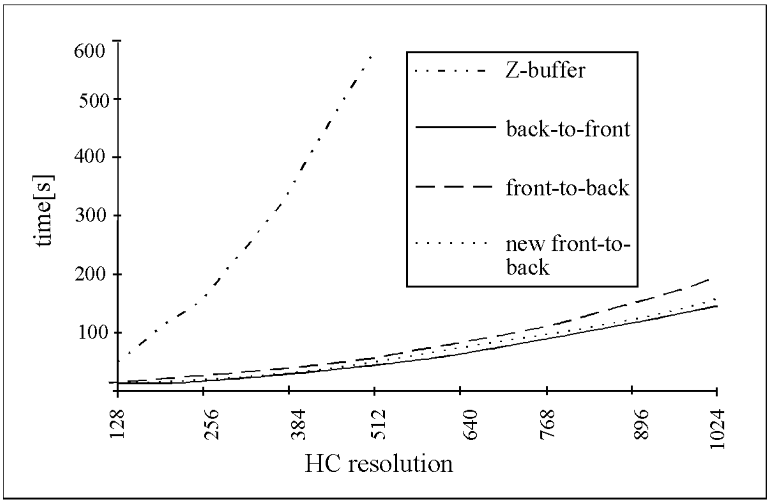

Figure 4: Time requirements of tested methods

Our test scene consisted of about 5000 elements. Methods described above were implemented and compared in combination with various HC resolutions. Graph 4 summarizes time requirements of these methods. Tests have run on SGI Indy (R4600 100MHz,32MB RAM).

5 Conclusion

As we expected, the slowest method is the (SW) Z-buffer approach. It is caused by more interpolation steps then in the BSP approach. More memory is also maintained and accessed. The fastest method (in lower resolutions) is the front-to-back BSP method. However, tests present that other modifications of the BSP based approach are also reasonable. We prefer our bit-buffered approach, because its lower memory requirements allow for high HC resolutions. In our tests the resolution reached up to 2048x2048 pixels (6 sec. pro iteration). We consider the new method presented as valuable alternative for fast and accurate form factor computation.

References

[1] Cohen, M., Greenberg, D.:"The Hemi-Cube: A Radiosity Solution for Complex Environments.", Proc. SIGGRAPH'85. In *Computer Graphics*, 19:3, July 1985, 31-40.

[2] Baum, D., Rushmeier, H., Winget, M.:"Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors.", Proc. SIGGRAPH'89. In *Computer Graphics*, 23:3, July 1989, 325-334.

[3] Wallace, J., Elmquist, K. and Haines, E.:"A Ray Tracing Solution for Diffuse Interreflection.", Proc. SIGGRAPH'89. In *Computer Graphics*, 23:3, July 1989, 315-324.

[4] Fuchs, H., Kedem, Z. and Naylor, B.:"On Visible Surface Generation by A Priori Tree Structures.", Proc. SIGGRAPH'80. In *Computer Graphics*, 14:3, July 1980,124-133.

[5] Chin, N., Feiner, S.:"Near Real-Time Shadow Generation Using BSP Trees.", in *Computer Graphics*, 23:3, July 1989, 99-106.

[6] Campbell III, A. T., Fussel, S.:"Adaptive Mesh Generation for Global Diffuse Illumination.", in *Computer Graphics*, 24:4, August 1990, 155-164.

[7] Lischinski, D., Tampieri, F. and Greenberg, D.: "Discontinuity Meshing for Accurate Radiosity.", *IEEE Computer Graphics and Applications*, 12:9, Nov 1992, 25-39.