Improved Billboard Using Bump Mapping

Zong-Ying Kuo
zkuo@usc.edu

Kübra Keskin
keskin@usc.edu

Peter Looi
looi@usc.edu

Adam Molani
molani@usc.edu

ABSTRACT

In the realm of computer graphics, optimizing rendering time is crucial, especially for tasks like PC gaming. Our paper introduces a novel approach that combines two established techniques, bump mapping and billboarding, to enhance 3D graphics rendering. Billboarding, often used in rendering expansive landscapes, involves strategically placing 2D images to represent 3D models. By integrating bump mapping, we include additional details into billboards, creating a compelling illusion of intricate 3D objects. Our method streamlines the rendering process, particularly for large models with high polygon counts, with significant implications for improving computer graphics performance.

Keywords

Billboards; Bump Mapping; Parallax Mapping; Normal Map.

1. INTRODUCTION

Rendering in computer graphics is a very compute-intensive process. Performance issues will often arise when there is a need to render many frames in short periods of time, such as with PC gaming. In this paper, we propose an approach to optimizing 3D graphics rendering time that combines two well-established techniques, bump mapping [1-4] and billboarding [5].

Billboarding is a technique that involves rendering a flat image of a 3D model in place of rendering the 3D model [5]. A common application of billboards is in 3D first person games that require the rendering of large forests. Trees that are farther from the camera will not have the entire tree model rendered, and instead it will be replaced with a 2D image of the tree. As billboarding relies on showing a 2D image of a 3D object, billboards are typically not rendered close to the camera, or else the risk of the viewer noticing that it is a 2D image will be high.

Bump mapping is a group of algorithms that utilize normal maps and height maps to add additional lighting and shape to a rendering [1-4]. Bump mapping is used in situations where additional detail on objects is needed, but it is impractical to add immense numbers of polygons for every little detail on the object.

The approach we have taken is to use bump mapping to add additional details to billboards, to create a convincing illusion of a 3D object being rendered. We explore the process of extracting information from a 3D model that can then be used to render billboards of that model with bump mapping. We also explored multiple bump mapping techniques and the tradeoffs with each. Ultimately, our results for this project have massive implications for the ability to simplify models that are very large or have a high polygon count, and to significantly improve computer graphics performance.

2. RELATED WORK

We referenced the concepts of bump mapping and perturbation vector maps from the paper 'Simulation of Wrinkled Surfaces' by J. F. Blinn, published in ACM SIGGRAPH Computer Graphics in 1978 [1]. This paper discusses how to define normal maps using tangent space and how to generate wrinkled surfaces using a height map.

To gain insights into parallax mapping and steep parallax mapping, we refer to the papers 'Detailed Shape Representation with Parallax Mapping' by T. Kaneko, T. Takahei, M. Inami, et al. [2], and 'Steep Parallax Mapping' by M. McGuire and M. McGuire [4]. These papers provide ideas on shifting uv coordinates by an offset and enhancing the offset approximation by dividing depth into layers.

The paper by Décoret et al. takes the concept of billboards one step further [5]. This paper introduces the concept of billboard clouds, which are a collection of billboards at different angles that, when rendered together, produce a result that resembles the original model. The paper also introduces the concept of a “viewcell”. A viewcell is a set of billboards that will be rendered if and only if the camera is located within a specific space. In the paper, they propose a solution that has different viewcells for different view angles towards the model, and different viewcells for different view distances. In our project, we use the idea of viewcells to create different anchor points between which the parallax mapping can interpolate between.

3. IMPLEMENTATION

3.1 Rendering Pipeline

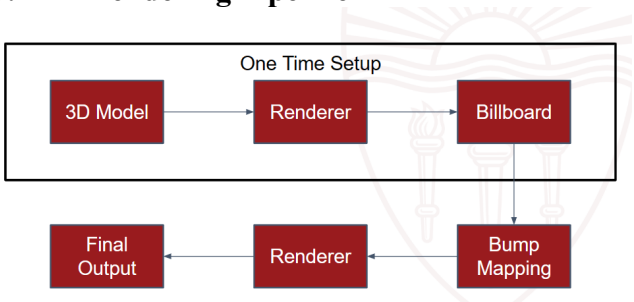

Figure 1. Flowchart of rendering

Our approach is to first create billboards for our 3D models. The billboards will be created by rendering the model multiple times from multiple different angles. Per-billboard height maps and normal maps will also be created at this stage. The billboards, height maps, and normal maps will be saved for future use. This is a one time conversion from model to billboard.

When the billboards are rendered into the world, this is when bump mapping is applied to give the flat billboard the illusion of being three-dimensional.

3.2 Bump Mapping

Bump mapping is a technique that gives the illusion of a wrinkled or bumped surface without increasing the geometric complexity. Instead of changing the geometry of the model, normal vectors of the surfaces are altered based on the perturbation function as demonstrated in Figure 2 [1]. This approach improves the appearance of textures and gives depth to surfaces, resulting in a more realistic rendering.

In our project, we implemented several types of bump mapping: 1) calculation of the perturbed normals based on a height map input, 2) directly using normal map input, 3) parallax mapping by using height map and normal map inputs, and 4) steep parallax mapping which is an improved version of parallax mapping. Any of these types can be used in our code by specifying which method to use in a parameter, and the details are explained below.

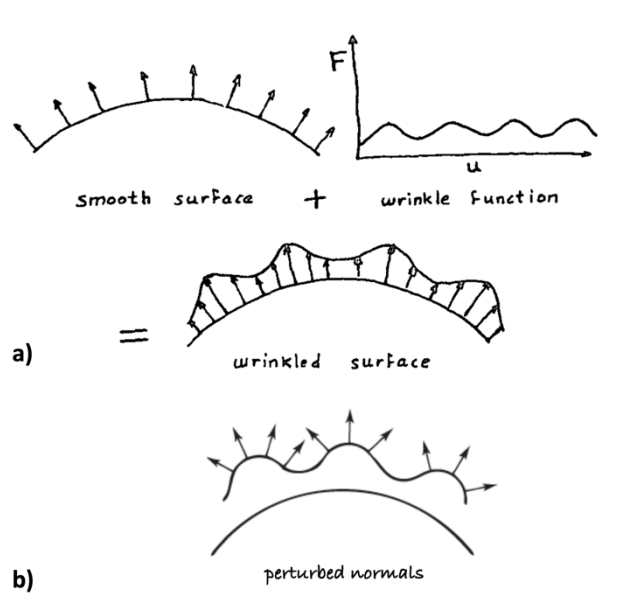

Figure 2. Demonstration of a bump mapping principle. (a) A wrinkled surface can be represented as a combination of a smooth surface and a wrinkle function [1]. (b) Bump mapping does not change the smooth surface model, but it perturbs the normals so that the surface looks wrinkled during the shading calculations.

3.2.1 Perturbed Normal Vector Generation

Perturbed normal vectors can be generated by looking at the intensity change between neighboring pixels of a height map. A bumpy surface based on u and v locations (P'(u,v)) can be expressed as

$$P'(u,v) = P(u,v) + F(u,v)\frac{N}{|N|}$$

where P(u,v) is the original surface, F(u,v) is the bump function which is stored in the height map, and N is the normal vector of the original surface [1]. Then the normal of the bumpy surface can be found by taking the cross product of partial derivatives along tangent and bitangent directions:

$$N' = P'_u \times P'_v$$

This perturbed normal can be written as the summation of the original normal vector and the perturbation vector as derived in Blinn's paper [1]:

$$N' = N + F_u\frac{N \times P_v}{|N|} + F_v\frac{P_v \times N}{|N|}$$

where F_u and F_v correspond to the partial derivatives of the bump function along tangent and bitangent.

In our project, we calculated finite differences between height intensities of close texture locations to obtain partial derivatives:

$$F_u(u,v) = (F(u+e,v) - F(u-e,v))/(2e)$$
$$F_v(u,v) = (F(u,v+e) - F(u,v-e))/(2e)$$

We also used the Sobel operator with a kernel size of 3 to calculate the partial derivatives along tangent and bitangent directions [6]. Different from calculating the finite difference along one direction by using only the neighbors along that direction, the Sobel operator uses information from all neighbors in the kernel. Based on the direction of the calculated derivative, we applied the Sobel operator transposed or not transposed. New perturbed normals were saved in RGB images.

We also defined a scale parameter to control the amount of perturbation we apply to the normals. Figure 3 demonstrates normal maps generated from the height map with different scale parameters and their corresponding renderings.

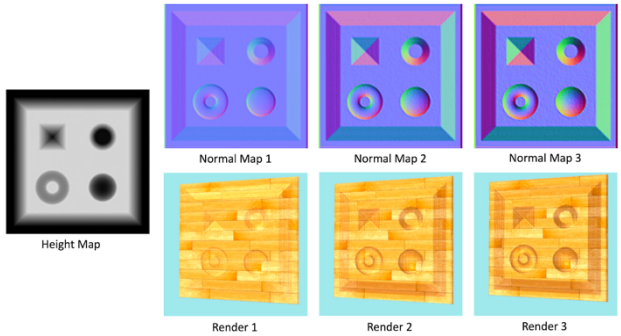

Figure 3. Normal maps with different amounts of perturbations are generated by Sobel filter normal perturbation calculation based on the height map.

3.2.2 Bump Mapping Algorithm

After we have the normal map, we can start to apply the new normals to our texture. Normal maps are stored in tangent space (Figure 4a), which is a space that is local to each triangle mesh. In order to perform shading calculations using the normal maps, we need to know the direction of the normals in shading space. Therefore, we need to find a transformation matrix that transforms tangent space to shading space and apply the matrix to the normals in tangent space:

$$Normal_{shading} = M(st) * Normal_{tangent}$$

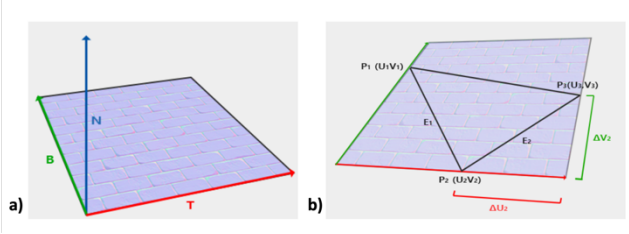

Figure 4. a) Tangent space. b) Relation between the tangent, bitangent vectors, and the triangle edges [7].

The matrix responsible for the transformation is referred to as the TBN matrix. To derive the TBN matrix, it is essential to compute the vectors that define the tangent space: tangent (T), bitangent (B), and normal vectors(N). From Figure 4b, we can observe that the tangent vector aligns with the U axis, the bitangent vector aligns with the V axis, and a relation exists between the tangent, bitangent vectors, and the edges of the triangle:

$$E_1 = \Delta U_1 * T + \Delta V_1 * B$$
$$E_2 = \Delta U_2 * T + \Delta V_2 * B$$

If we consider the edges of the triangles as vectors, we can observe that the E_1 vectors can be expressed as the sum of one vector with a length of ΔU_1 along the tangent vector axis and another vector with a length of ΔV_1 along the bitangent vector axis. The same approach can be applied to the E_2 vector. If we write the equation in matrix form, it would look like this:

$$\begin{bmatrix} E_1{}^T \\ E_2{}^T \end{bmatrix} = \begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix} \begin{bmatrix} T^T \\ B^T \end{bmatrix}$$

Moving the matrix to the other side of the equation, we get:

$$\begin{bmatrix} T^T \\ B^T \end{bmatrix} = \begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix}^{-1} \begin{bmatrix} E_1{}^T \\ E_2{}^T \end{bmatrix}$$

$E_1, E_2, \Delta U, \Delta V$ can all be calculated since we already have the xy and uv coordinates defined at each vertex, we plug them into the equation, and then we can get the tangent vector and the bitangent vector.

After calculating the TBN matrix, we apply it to transform the normals to shading space and perform light calculations. Figure 5 shows the rendering results with and without bump mapping. We can observe that the pixel color does change according to the new normals.

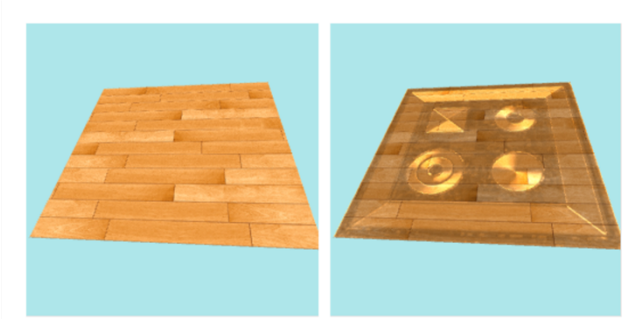

Figure 5. No bump mapping vs with bump mapping.

3.2.3 Parallax Bump Mapping and Steep Parallax Bump Mapping

In Figure 5, although we observe that the color of the pixel has indeed changed, the bump's effect is not pronounced, and it still appears rather flat. The reason is that we have not taken the viewer's angle into account.

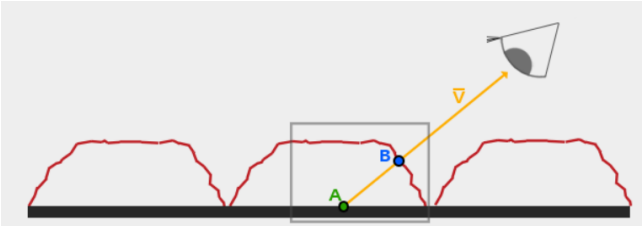

Figure 6. The impact caused by varying viewer perspectives [8].

Figure 6 illustrates the importance of considering the viewer's angle. When attempting to render the pixel at point A, we examine the normal map and retrieve the normal based on point A's uv coordinates. However, due to the viewer's angle, point A is blocked by a bump, causing the viewer to perceive point B instead. Consequently, we should use the normal at point B instead of point A. To address this issue, an algorithm known as parallax bump mapping is employed [2].

The concept behind parallax mapping involves introducing an offset to shift the uv coordinates from point A to point B and retrieve the normal vector at point B. The offset is calculated through approximation. We want to determine vector P (Figure 7), aligned with the view ray's direction, and the offset would be the projection length of P onto the UV plane. The magnitude of vector P is determined by examining the displacement map and retrieving the height (H(A)) at point A.

$$Vector\ P = View\ Ray * H(A) * scale$$
$$Offset = \|P(x/z, y/z)\|$$

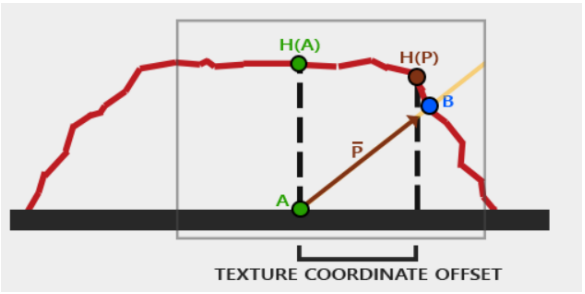

Figure 7. Calculating the P vector [8].

Furthermore, we have an improved parallax mapping algorithm called steep parallax mapping, which performs better offset approximation than parallax mapping.

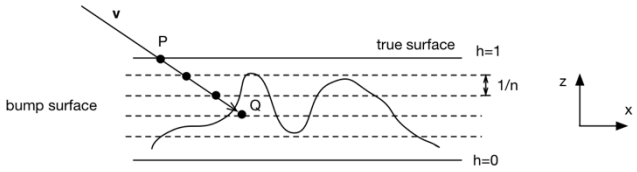

Figure 8. Steep Parallax Mapping [9].

Figure 8 illustrates the idea of steep parallax mapping [4]. We divide the height into multiple layers. We move along the view ray, assuming point P is the intersection between the view ray and the layers. We check whether the height of that layer is higher than the bump's height, continuing to check until the layer's height is lower than the bump (Point Q). We then retrieve the normal at point Q. The amount to shift the texture coordinates ΔTexCoords per layer would be:

```
ΔTexCoords = ViewRay.xy / #Layers
During each iteration, we do:
while currentLayerDepth > currentDepthMapValue:
          uv_coords += ΔTexCoords
          HeightMapLookup(uv_coords, Bump_height)
          currentDepthMapValue = Bump_height
          currentLayerDepth -= layerDepth
final_shifted_uv_coords = uv_coords
```

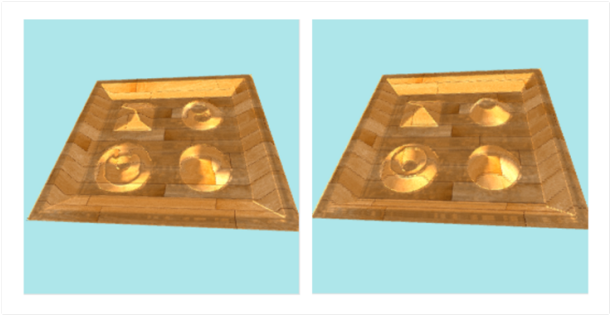

Figure 9. Left: parallax bump mapping, Right: steep parallax bump mapping

Figure 9 illustrates the outcome of bump mapping using the aforementioned algorithm. Both methods yield more prominent bumps, and we can discern the slopes at the plane's edges. In steep parallax mapping, the bumps appear more pronounced compared to regular parallax mapping, and they do not exhibit distortion. In parallax mapping, some distortion is noticeable on the bumps.

3.3 Billboard

Billboard creation involves a two-step process. First, the model is rendered using a standard 3D renderer from several different perspectives. Then, for each perspective, the render result is saved into a buffer for future use. This step saves not only the RGB pixel values of the render result, but it also saves the normals and height/depth at each pixel of the output image.

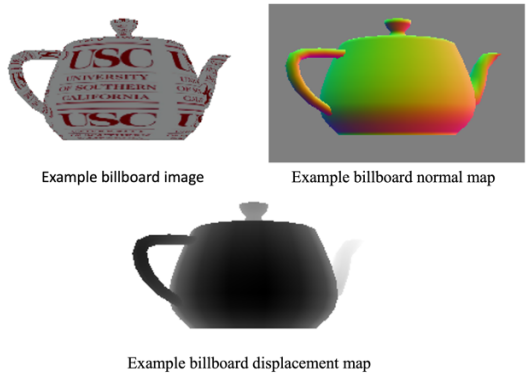

Figure 10. Example billboard image, normal map, and displacement map.

When creating the billboard renderings, the algorithm decides on the optimal camera distance in order to maximize the size of the model within the rendered image, while making sure that no part of the model gets cropped by the borders of the rendered image. The formula we used is

$$z = (r - 1)/tan\,(f/2)$$

where

z = distance from the origin the camera should be placed at

r = distance between the origin and the vertex in the model that is furthest from the origin

f = field of view in radians

This equation was simply derived from the image space to perspective space transformation

$x' = x/(z/d + 1)$ where $1/d = \tan(f/2)$.

And setting x' to 1, so that after the perspective transformation the radius of the model is guaranteed to be half the side length of the image.

4. RESULTS

4.1 Billboard

Using billboards without bump mapping, the resulting simplification of the model is of low quality. If the billboard is looked at from an angle, it is easy to tell that the object being rendered is a billboard (Figure 11b).

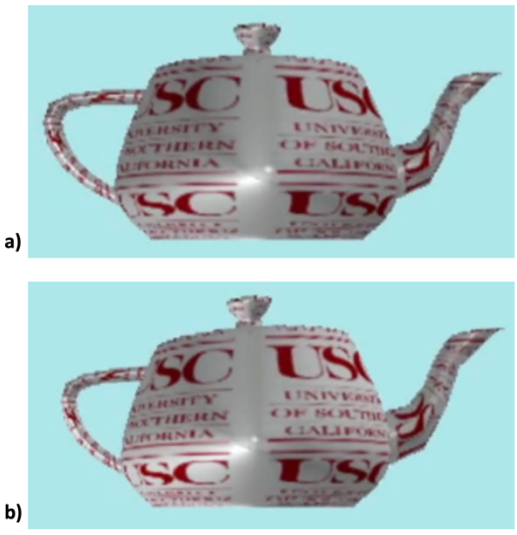

Figure 11. a) View of one of the billboards head-on. b) View of the billboard from an off angle, from which it is not hard to tell that the object being rendered is a billboard.

4.2 Billboard with Parallax Mapping

Adding bump mapping to the equation allows for a smooth transition between the viewcells that gives a convincing illusion of depth.

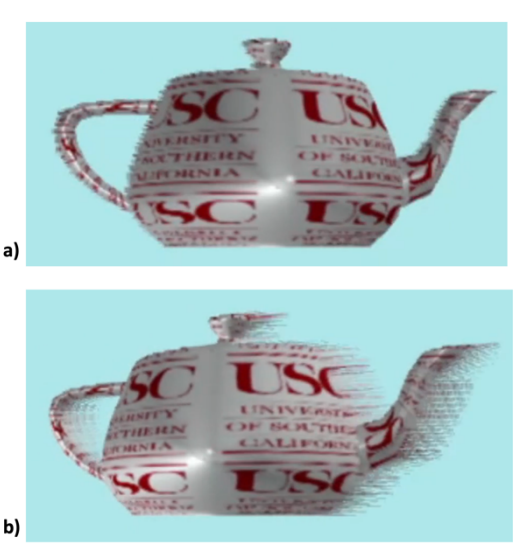

Figure 12. a) View of one of the billboards head-on. b) View of the billboard from an off angle utilizing parallax bump mapping to give the illusion of the 3D object rotating.

5. CONCLUSION

The ability to simplify a model into a single billboard has interesting implications on computer graphics performance. While rendering a model using conventional methods requires traversal through every triangle, making it at least a linear time complexity with respect to the number of triangles in the model $\Omega(n)$. However, using our approach, the time to render the simplified model does not depend on the size of the model. Rendering billboards with bump mapping is essentially a constant time complexity for all models, except for the one time setup.

The main drawback of this approach is that the quality of the final result is compromised. When looking at the billboards from an angle, some of the edges of the billboard get blurred. This blurring is simply one of the limitations of parallax bump mapping. In addition, there is still a noticeable jump while animating transitions between billboards. This is something that likely can be improved with more research. Another weakness of this approach is that it requires a significant amount of memory to store the billboards for each perspective along with height maps. Furthermore, it is not possible to deform the model once it has been converted to billboards.

This algorithm has the potential to save significant amounts of time when used on complex 3D models with high polygon counts. In addition, bump mapping billboards have the potential to massively speed up rendering static scene objects that do not move by combining many static objects into a single billboard. More research is needed to determine the upper limit to the effectiveness of this technique.

6. REFERENCES

[1] J. F. Blinn, “Simulation of wrinkled surfaces”, ACM SIGGRAPH Computer Graphics, vol. 12, no. 3, pp. 286–292, 1978.

[2] T. Kaneko, T. Takahei, M. Inami, et al., “Detailed Shape Representation with Parallax Mapping”, ICAT 2001, January 2001.

[3] W. Donnelly, "Per-pixel displacement mapping with distance functions", GPU gems 2.22, 2005.

[4] M. McGuire, M McGuire, “Steep Parallax Mapping”, I3D 2005 Posters Session, 2005.

[5] Décoret, F. Durand, F.X. Sillion, and J. Dorsey, “Billboard Clouds for Extreme Model Simplification”, ACM SIGGRAPH 2003, July 2003.

[6] I. Sobel, and G. Feldman. "A 3x3 isotropic gradient operator for image processing." a talk at the Stanford Artificial Project in (1968): 271-272.

[7] Anon. Normal mapping. Retrieved November 30, 2023 from https://learnopengl.com/Advanced-Lighting/Normal-Mapping

[8] Anon. Parallax mapping. Retrieved November 30, 2023 from https://learnopengl.com/Advanced-Lighting/Parallax-Mapping

[9] Anon. Retrieved November 30, 2023 from https://subscription.packtpub.com/book/game-development/9781789342253/5/ch05lvl1sec50/steep-parallax-mapping-with-self-shadowing