

Accurate Reflections Through a Z-Buffered Environment Map

Gustavo A. Patow

LIFIA, Depto. de Informática, Universidad Nacional de La Plata, C.C. 11.
(1900) La Plata, Buenos Aires. ARGENTINA.

e-mail: dagush@info.unlp.edu.ar

Abstract:

A new algorithm for computing accurate reflections with an environment map is presented. This new algorithm keeps the z-buffers originally calculated to obtain the map and uses them to find the best approximator to the real reflected point. It performs a linear search in the plane defined by the reflected vector and the position of the reflective surface. For each vector in this plane, the algorithm calculates how close to the real reflection point it is, and keeps only the one that approximates the reflected position best. The resulting images are identical to those computed with other methods (i.e. recursive Ray Tracing). The model offers the possibility of computing all the reflections with just one environment map per scene. The running time of the current implementation of the model is ten to fifty times slower than the implementation of the old reflection map models, but the expected final running time is only five times slower. The limitations of the new model are analysed and a solution is provided.

1 Introduction

Reflections play a very significant role in the Computer Graphics search for photorealism. Although exact solutions could be computed through a ray tracing algorithm, it is a very expensive method because of its computing time cost and because it needs to store the whole scene throughout the rendering. They can also be calculated with the help of an environment map (also known as a reflection map), that represents a relatively inexpensive way to render highly specular objects. This second method was developed by Blinn and Newell [1] in 1976: the environment to be reflected is mapped onto the surface of a sphere surrounding the objects to be rendered, with the centre of projection chosen conveniently. Thus, the mapped environment can be treated as a two dimensional texture map for the reflective object.

The unit reflection vector, \mathbf{B} , is obtained by reflecting \mathbf{I} (the normalised incident vector) about \mathbf{N} (the unit normal vector at the surface), and its coordinates could be used to index the reflection map. Blinn and Newell suggested converting \mathbf{B} to polar coordinates and reading from the map the light intensity for that direction. Instead of a sphere, the six projections onto the sides of a surrounding cube, aligned with the world coordinates (see [2], page 758, or [7], page 190), may be used. This is the most commonly used method for environment mapping [5], and can be implemented in hardware for real-time shading ([6]). Figure 1 shows a simple scene with a reflective sphere rendered with this model.

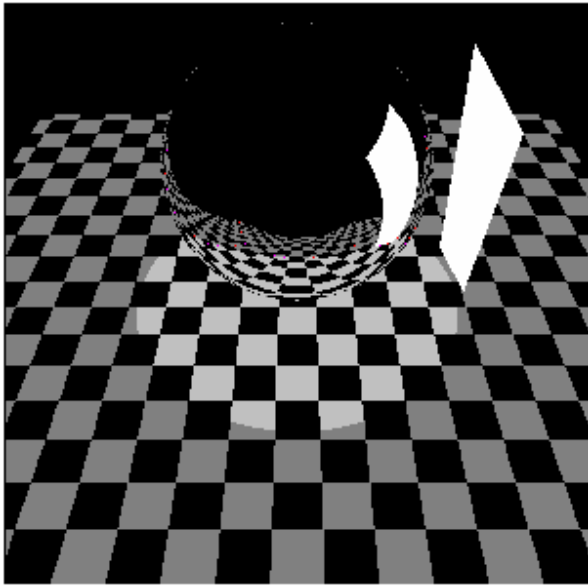


Figure 1: Example scene using the Blinn-Newell original reflection model

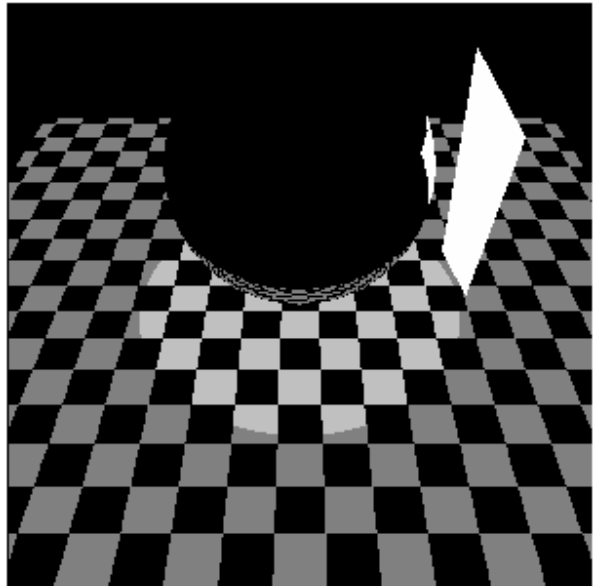


Figure 2: Sample scene with the Reflection Map indexed as a common texture.

This method only represents an approximation to the correct reflection information: by taking into account just the surface's reflection direction and not its position, it models an infinitely large environment sphere, ignoring parallax ([6]). This problem can be partially solved by using the surface's position to help determine the point in the reflection map to index, modelling a sphere of finite size ([2] page 759). For example, if we compute the reflection vector indexing the map only with the position on the surface, pointed by \mathbf{W} , we will represent a sphere in touch with the object, reflecting accurately only those objects that are very close to it. This last way of indexing the environment map is equivalent to use it as a common texture map. Figure 2 represents the same test scene with the "reflections" obtained with the \mathbf{W} vector.

The model presented in this paper represents a significative improvement on the ones mentioned before, since it allows to find the best aproximator to the reflection vector, within the environment map resolution.

Section 2 presents the new model and Section 3 explains the basic algorithm to implement it. In Section 4, there is the explanation of its limitations and the corresponding solutions, and Section 5 describes the current implementation detail. Finally, Section 6 presents the results and a comparison with previous models.

2 Proposed reflection model

It's easy to see that the unit reflection vector (\mathbf{B}), the position on the surface of the mirrored object (\mathbf{W}) and the reflected point (\mathbf{S}) belong to the same plane, which we call the reflection plane (see Figure 3). Note that all the origins are located at the centre of projection of the environment map.

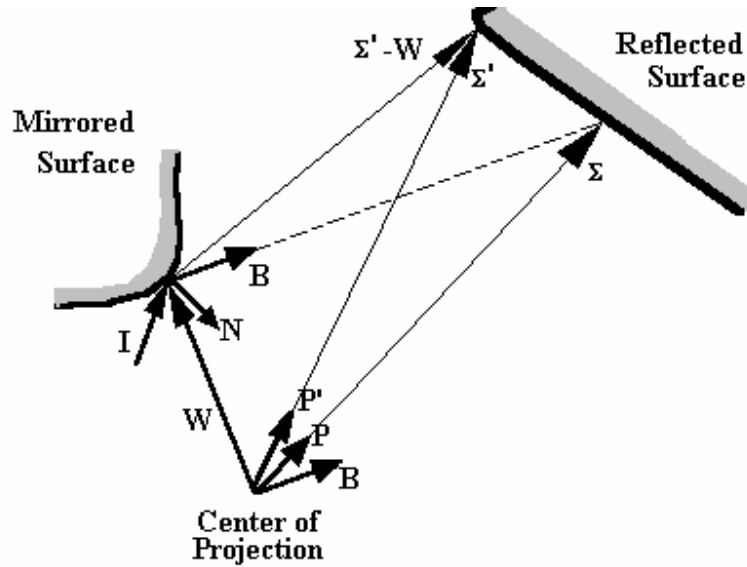


Figure 3: The reflection plane geometry

This way, we can formulate the following theorem, which can be easily demonstrated in geometrical terms:

Theorem:

The true unit reflected direction from the center of projection ($\mathbf{P} = \frac{\mathbf{S}}{\|\mathbf{S}\|}$, where $\frac{\mathbf{A}}{\|\mathbf{A}\|}$ means normalization to unity), could be written as a linear combination of the vectors \mathbf{B} and \mathbf{W} :

$$\mathbf{P} = [(1-t)\mathbf{B} + t\mathbf{W}] \quad (1)$$

with t in the range $[0,1]$ (this implies that, when $t = 0$, $\mathbf{P} = \mathbf{B}$; and when $t = 1$, $\mathbf{P} = \frac{\mathbf{W}}{\|\mathbf{W}\|}$).

We can demonstrate this theorem using the fact that \mathbf{S} is, by its definition, $\mathbf{W} + \lambda \mathbf{B}$ (with λ a real positive valued appropriate constant). Then, $\mathbf{S} = \mathbf{W} + \lambda \mathbf{B} = \eta ((1-t)\mathbf{B} + t\mathbf{W})$, where η is another real value. From that, we easily find that $t = (\lambda + 1)^{-1}$ and $\eta = (\lambda + 1)$; what presents the expected asymptotic behaviours for $\lambda \rightarrow 0$ and $\lambda \rightarrow \infty$.

We could say that the vector \mathbf{P} lies *between* the vectors \mathbf{B} and \mathbf{W} , on the same plane (See Figure 3). In other words, what the theorem means is that the reflected point from the centre of projection of the environment map (\mathbf{S}) is between the reflective surface (\mathbf{W}) and infinity (\mathbf{B}), what is an intuitive result. The angle between the last two is in the range $[0,\pi]$, but for the two extreme cases, the first one (\mathbf{B}) must be taken to compute the reflection. So, the effective range becomes $[0,\pi)$.

Of course, when computing the reflections, we do not know what λ , η or t are. So we must find an alternative way to compute them.

Here is where the z-buffer info becomes useful: for each pixel in the reflection map, we store the value of the distance from the center of projection to the surface point being reflected (α_i , with i being

each pixel in the map). In this way, $\alpha\mathbf{P}$ (α is the environment map z-value in the direction indexed by \mathbf{P}) represents the position from the center of projection of the surface point (i.e.: $\mathbf{S} = \alpha\mathbf{P}$). Of course, we are assuming an idealised reflection map here, with infinite resolution, and neglecting the difference between the vector \mathbf{P} and the original vector used to compute the reflection. This approximation will be analysed later, in the next section.

As was already mentioned, \mathbf{S} could be written as $\mathbf{W} + \lambda \mathbf{B}$, where λ is a real number. So, the problem that remains is to minimize the expression (see Figure 3):

$$\text{AngleBetween}(\mathbf{B}, (\mathbf{S}' - \mathbf{W})) \quad (2)$$

for all \mathbf{S}' with the constraint that the points $\mathbf{S}' = \alpha'\mathbf{P}'$ rely on the reflection plane, between \mathbf{B} and \mathbf{W} . The values for α' are stored in the z-buffer, indexed with \mathbf{P}' . When $\mathbf{P}' = \mathbf{P}$ the equation is zero, and it takes its theoretical minimum value.

With normal \mathbf{N} and incident vector \mathbf{I} , find reflected vector, \mathbf{B} .

if $(\mathbf{B} \cdot [\mathbf{W}]) = \pm 1$

then

 Compute reflection in the direction of \mathbf{B} .

else

For each pixel in $(\text{Reflection Plane} \cap \text{Reflection Map})$ **do**

 Find \mathbf{P}' from pixel position.

 Get α' from the reflection map (and then calculate \mathbf{S}').

if pixel currently indexed has a smaller angle than previous ones (Expression 2)

then

 keep new pixel.

if angle of best approximator $>$ tolerable_limit

then

 Reflect infinity (background, \mathbf{B}).

Figure 4: Pseudocode to compute the new reflection model. \cdot means dot product, and \cap means intersection between planes.

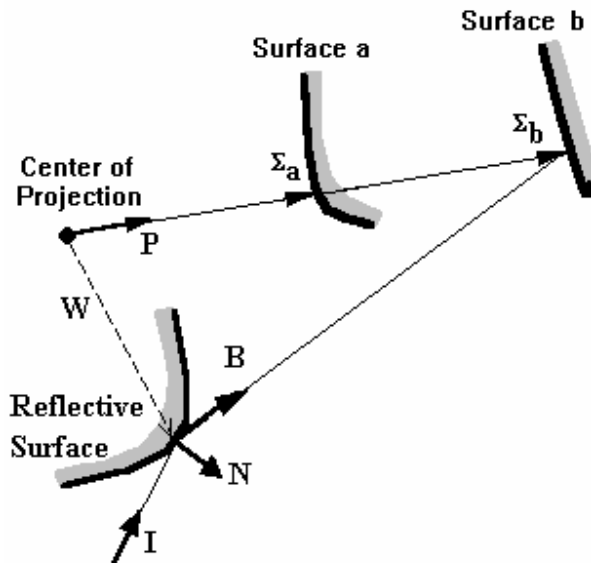


Figure 5: *Geometry of the hiding problem.*

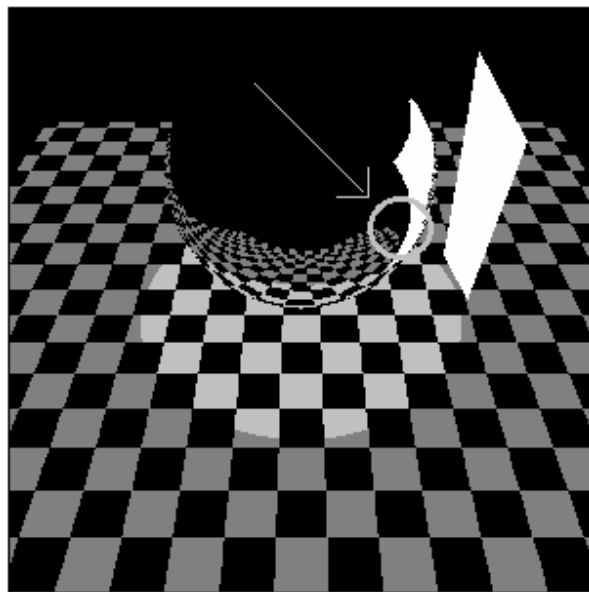


Figure 6: *Sample scene showing the hiding problem: there is a small black hole at the left of the reflection of the white polygon.*

3 Outline of the algorithm

The problem now could be presented in the following way: Find the vector \mathbf{P}' that best approximates the reflected point (Equation 2). Such vector lies on the reflection plane, between \mathbf{B} and \mathbf{W} (as stated in the theorem above). This means that only a linear, monodimensional search, is needed.

So, we need to inspect each possible vector \mathbf{P}' , but iterating with Equation 1 (“search in the \mathbf{P}' space”) using a small value for the step in t is a waste of time, since many pixels of the reflection plane could be examined more than once, or even not examined at all, depending on the step value taken. That is why the best solution is to iterate in the environment map image precision space, recovering \mathbf{P}' for each pixel in the reflection plane.

Which are those pixels? Those which lie on the intersection between the reflection plane (computed by the two reflection vectors) and the environment map used. If a sphere is used, then the pixels are those that rely on the smallest arc on its surface, with extremes in the points indexed by \mathbf{B} and \mathbf{W} . If a cubical environment map is chosen, the intersections of the plane with the faces of the cube must be calculated, and a line drawing algorithm should be used to index the pixels. The most straightforward line drawing algorithm is the simple DDA (Digital Differential Analyser) developed by Bresenham [2], but a modified DDA that identifies all pixels traversed by the line would be a better option (see Musgrave’s work [4], for a detailed description).

There is still an important issue to be considered: the discrete nature of the environment map used in the algorithm. As stated before, the environment map has only a finite resolution, so it is not true that the vector used originally to compute this map lies exactly on the reflection plane, and it’s no longer true that one of the reflection pixels in the map was originated from the reflection vector we are looking for. To partially solve this last problem, when iterating in image space, the true position of the pixel in the map is calculated, and the original reflection vector is then recovered. Expression 2 will not give an exact relation any more (the minimum could be non zero). Instead, we should try to find its minimum for

the environment map pixels that lie on the reflection plane, between the two vectors, and keep the final pixel only if that relation is below a given threshold (to prevent wrong reflections).

The pseudocode is presented in figure 4. As it can be easily seen, the algorithm complexity only depends on the number of reflective pixels rendered, and does not depend on the number or nature of the reflective objects being processed (opposed to what occurs with a recursive ray tracing implementation).

4 Model limitations and solutions

As the algorithm performs a linear search for the best approximator, situations like those presented in Figure 5 could arise: There are some points like S_b that are not visible from the center point (S_b is not present in the depth map because it is hidden from the environment map centre of reflection by the point S_a), but visible from another point of the scene. So the algorithm fails in its search, returning the background color (the small black hole at the left of the reflection of the white polygon in Figure 6). In fact, all of the points of Surface A hide all those behind it, as viewed from the center of projection. We could call this the “hiding problem”.

The obvious solution is to store more than one z-value for each pixel, keeping them in a depth-sorted list of surfaces. In that way, the first item in each list would be the traditional z-buffer info for that pixel. This solution would make the algorithm depend slightly on the geometry of the scene, but indirect visibility criteria could be used to cut out the lists, as those showed by Hall (see [2], page 782) for the combination of ray tracing with reflection maps.

The algorithm would proceed as described in the last section, making the linear search only for the first value in the list of each examined pixel. When this search fails (no best predictor found), then the second depth value in each environment map pixel traversed should be evaluated, and the procedure should be repeated until all the lists for all the pixels are finished, or until a best predictor is found. Of course, all the searches should be performed at once, keeping the best predictor for each “layer” on another list, and leaving the verification of which predictor is the best until the end. Different depth “layers” must not be mixed, because the algorithm could take a wrong reflection, giving the appearance of semi-transparent objects when reflecting them.

```
Actual ← B      { that is, t ← 0 }  
While faces of the reflection map indexed by Actual and W are not the same do  
  Get pixel indexed by Actual.  
  For each entry in the current pixel depth list do  
    Inspect depth and keep value if better than the respective previous one.  
    Increment t and use Equation (1) to get new Actual.  
  {both vectors index the same face, so }  
Do the DDA traversal in that plane, between the pixels indexed by Actual and W.
```

Figure 7: Pseudocode for the reflection plane traversal (the “For each pixel...” part in Figure 4)

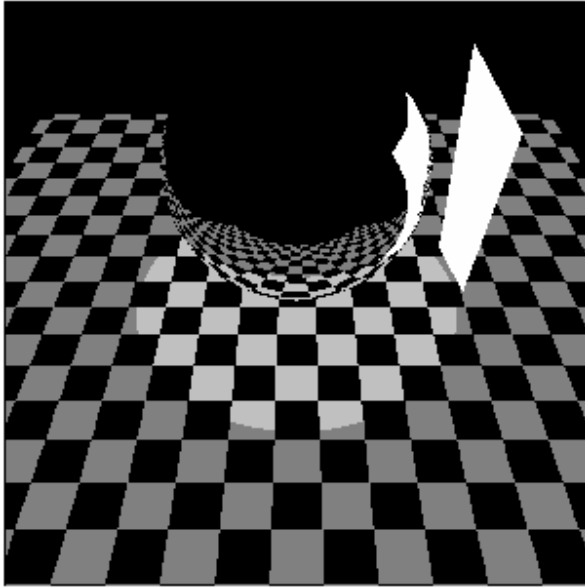


Figure 8: *Sample scene computed with the new Model.*

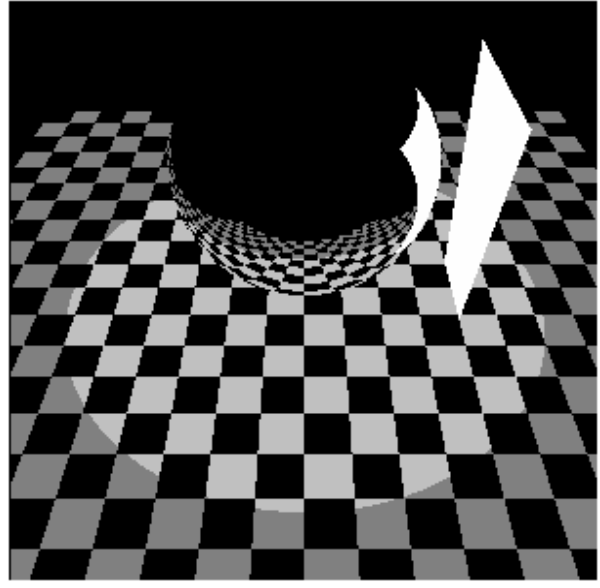


Figure 9: *Ray Traced Reference Scene.*

5 Implementation

The current implementation of the algorithm is a hybrid solution between the search in \mathbf{P}' vector space (Equation 1 with a small value for the step in t) and a full search in the reflection map image precision space (where a cubical environment map is used): the search (the “For each pixel...” part in the pseudocode in Figure 4) follows the pseudocode presented in Figure 7.

The “hiding problem” is addressed in the loop within each pixel, where separate comparisons for each depth “layer” are performed together, and the results stored in another list.

The step for t , in the first part of the search, was $1/200$; the threshold value used to discard the best predictor of the reflected point was set to approximately one degree. The resolution of the pictures and all the buffers used is $300 * 300$ pixels. Because of the simplicity of our scenes, only a maximum of two entries were needed for the depth lists at each pixel.

The reflection model is currently implemented in Pascal for The Microsoft Windows Environment, running on a 486DX2 PC, and the new reflection model implementation took about 500 lines.

6 Results

The same simple scene, but computed with the new model, can be seen in figure 8, and Figure 9 is a reference, ray traced scene (computed with the POV Ray Tracer program). As expected, the reflections are identical to those computed with the ray tracing program. All differences between both images are only due to the polygonal approximation made for the sphere in our implementation, and not due to the algorithm used. The obvious improvement could be easily seen if one compares with the results of the old reflection models, shown in Figures 1 and 2.

As mentioned before, the time consumed depends on the number of reflective pixels rendered. The measured time ratio per reflective pixel from Blinn's model to the new model is about 10 - 15 (that is, the last one is ten to fifty times slower than the first one), but it must be considered that only a hybrid solution was implemented, and a full image space search would increase considerably the speed of the algorithm. In fact, the measures show that the hybrid solution is three times faster than the search completely done in vector space, so it's reasonable to suppose that the real ratio of the search done completely in image space, to the model of Blinn and Newell, should be about 5.

7 Summary and conclusion

A new reflection model has been presented. It computes the best reflection vector among those belonging to the plane defined by \mathbf{B} and \mathbf{W} . For that reason, the algorithm requires to keep the z-buffers generated for the environment map calculations, and uses them for the computations. As seen from the images, the results are the same as those calculated with more precise, but slower methods (i.e. recursive Ray Tracing). The computational complexity of the algorithm depends only on the number of reflective pixels rendered on the final image, and not on the nature or number of the reflective objects in the scene.

Taking into account its limitations and the presented solution, the new model could be pushed further: only one z-buffered environment map is necessary for each scene, since it stores all the information needed to accurately compute all the reflections in it.

Acknowledgements

I wish to thank José Luis Castiglioni, Fernando Lyardet, Fernando Das Neves and Elisabet Carbonella for reading the different versions of the manuscript. I am indebted to Werner Purgathofer and, specially, to Christophe Schlick for their great help and infinite patience with me. This work was supported by a subsidy provided by the Fundación Antorchas.

References

- [1] Blinn, J.F. and M.E. Newell, "Texture and Reflection in Computer Generated Images", CACM, 19(10), October 1976, 542-547.
- [2] Foley, J., A. van Dam, S. Feiner and J. Hughes, "Computer Graphics Principles and Practice, 2nd. Edition", Addison-Wesley, 1990.
- [3] Glassner, A., "An Introduction to Ray Tracing", Academic Press, 1991.
- [4] Musgrave, F. K., "Methods for Realistic Landscape Rendering", Doctoral Dissertation, Yale University, 1993.
- [5] Upstill, S., "The RenderMan Companion". Addison-Wesley, 1992.
- [6] Voorhies, D. And J. Foran, "Reflection Vector Shading Hardware", Proceedings of SIGGRAPH'94, (July 1994), 163-166.
- [7] Watt, A. and M. Watt, "Advanced Animation and Rendering Techniques. Theory and Practice", Addison-Wesley, 1992.