Reflection Morphing

Andrew Martin, Voicu Popescu,
Computer Science, Purdue University

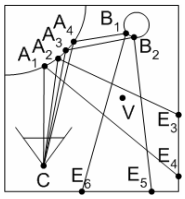
1. Introduction

Most scenes of interest to computer graphics applications contain reflective surfaces. Rendering such surfaces accurately and effectively is challenging. Most interactive graphics applications render reflections using environment mapping. Environment mapping approximates reflections drastically. In some situations, a reflected point is drawn hundreds of pixels away from its correct location. Environment mapping does not provide motion parallax, in other words, distant and near reflected objects do not move with respect to each other as the desired view translates. Accurate reflections can be obtained by ray tracing but that comes at the price of sacrificing interactivity.

We describe reflection morphing, a novel algorithm that renders accurate reflections on general reflectors at interactive rates. The difficulty in rendering reflections comes from the fact that a reflected vertex cannot be projected onto the desired view. Reflection morphing builds a set of ray octrees as a preprocess, which are then used at run time to morph the projection of reflected vertices. Once the projection problem is overcome, reflection morphing takes advantage of existing powerful feed-forward graphics hardware. The method provides correct motion parallax and supports moving objects, multiple reflectors, and higher order reflections.

2. Reflection morphing

Consider a desired view camera that is used to render a scene containing perfect reflectors. The reflectors do not have their own appearance but rather borrow it from the diffuse objects in the scene. Reflectors only affect the direction of a ray, not its color, and should thus be considered as being part of the camera not as part of the scene. Consider the scene in the figure. The generalized camera obtained by combining the camera C with the two reflectors can be described as a list of ray segments: (CA_1, CA_2, CA_3, CA_4, A_4B_1, A_3B_2, B_1E_6, B_2E_5, A_1E_4, A_2E_3). The points E are obtained by clipping the rays with a bounding box of the scene. If only the generalized camera could project an arbitrary 3D point efficiently onto its image plane, the problem of projecting reflected vertices would be solved and hardware could take care of rasterization to finish rendering the reflections.



In general, an arbitrary point does not lie exactly on one of the ray segments, so the projection is approximate. Moreover a point can project to multiple locations in the image plane. In the figure the point V projects at A_2 because it is closer to A_2E_3 than to A_1A_4 and at A_4 because it is closer to B_1E_6 than to B_2E_5.

One way of implementing projection for the generalized camera is to traverse the list of rays sequentially, compute the distance from every ray to the given point and return the rays that are closest to the point. Camera C alone has hundreds of thousands of rays, so an acceleration scheme is needed. We use *ray octrees* that subdivide the scene recursively and store at the leaves the subset of rays relevant to that particular region of space. Given a point, the leaf that contains it is located in logarithmic time, and the much shorter ray list stored at the leaf can be searched efficiently to find the closest ray.

Since the desired view changes for every frame, it is impossible to build the ray octree for the desired view. Instead we build ray octrees at the nodes of a regular 3D grid. We call the nodes *sampling locations* and the grid cells *sampling cells*. At run time, the current sampling cell is established using the position of the desired view. Each vertex is projected onto the 8 sampling locations using the ray octrees, and then the resulting 8 projection points are trilinearly interpolated to find the final projection point. Once the projection of the reflected vertex is found, the triangles are rasterized with the help of hardware.

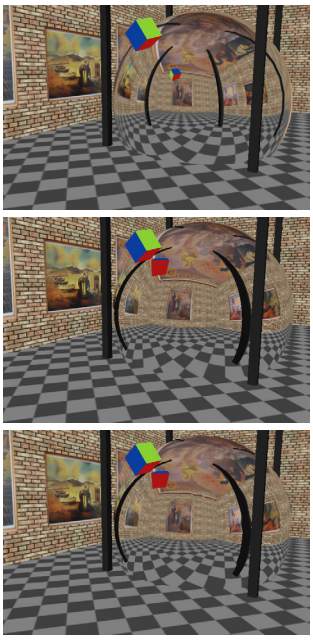


Figure 1 Environment mapping (*top*), Ray tracing (*middle*), Reflection Morphing (*bottom*).

The algorithm computes correct reflections at the 8 sampling locations and then morphs them into the final reflection to take into account the offset within the current sampling cell. The desired view and the diffuse objects can move freely. If the reflectors move, the ray octrees need be recomputed. Reflection morphing renders interactively very high-quality reflections (Figure 1, and accompanying video).

3. Results and discussion

If all vertices in a scene are dynamic, then each vertex must be morphed for each frame. However, in many cases, much of the scene is static. These vertices must only be morphed once for each sampling cell. This greatly improves performance. We measured a morphing performance of 20,000 dynamic vertices per second and 200,000 static vertices per second. Performance was measured on a Pentium 4 2GHZ, 2GB PC. Reflections can be rendered with high quality with a small amount of preprocessed data. With a ray map resolution of 256 x 256, an octree depth of 5, and one reflective bounce allowed, the preprocessed data required 4.1 – 5.4 MB of space for a sampling location. A location's average number of rays at a leaf was in the 10.4 – 13.5 range, and its maximum number of rays was in the 227 – 353 range. Preprocessing takes about 3 minutes per sampling location.

In the future, reflection morphing could be extended to render higher order reflections. Much of the necessary framework for this is already in place. It might also be extended to allow moving reflectors. The technique could also be modified to produce other effects such as refraction.