Sample-based cameras for fast accurate reflections

Voicu Popescu, Elisha Sacks, and Chunhui Mei

***Abstract*— This paper presents sample-based cameras for rendering accurate reflections on curved reflectors at interactive rates. The method supports change of view, moving objects and reflectors, higher order reflections, view-dependent lighting of reflected objects, and reflector surface properties. In order to render reflections with the feed forward graphics pipeline, one has to compute the image points where a reflected scene point projects. A sample-based camera is a collection of BSP trees of pinhole cameras that jointly approximate the projection function. It is constructed from the reflected rays defined by the desired view and the scene reflectors. A scene point is projected by invoking the cameras that contain it in their frustums. Reflections are rendered by projecting the scene geometry then rasterizing in hardware.**

***Index Terms*— reflections, interactive rendering, image-based rendering, sample-based graphics.**

I. Introduction

Fig. 1. Sample-based camera Fresnel reflections on automobile.

We present a novel algorithm for rendering reflections quickly and accurately. Reflections are important for interactive computer graphics applications because they provide visual cues about surface properties, shape, and relative position. The main techniques for rendering reflections are ray tracing and environment mapping. Ray tracing searches for the scene point seen at each pixel.

Purdue University, West Lafayette, IN 47907, USA.
E-mail: popescu@cs.purdue.edu

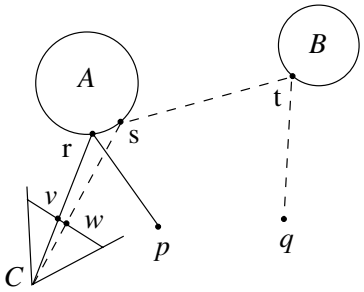


Fig. 2. Projection: computed reflected points v and w given scene points p and q and reflectors A and B.

Although accurate and general, ray tracing is too slow for interactive graphics because of the inefficient search for the inverse mapping. Environment mapping approximates the reflection with a pre-rendered image that is indexed using the direction of the reflected ray. Although fast, the technique is inaccurate and is limited to first order reflections. There are several variants of these methods, but none is accurate, general, and fast.

The feed forward pipeline, which first projects then rasterizes scene geometry, is the preferred approach in interactive graphics because of its efficiency. The challenge in rendering reflections with this approach is to project reflected points (Fig. 2). An accurate, efficient approximation is required because the projection cannot be expressed in closed form for curved reflectors.

We have developed a projection method that renders reflections with 1–5 pixels accuracy at interactive rates (Figs. 1 and 3). The method supports change of view, moving objects and reflectors, and reflections of any order. The projection function is represented with a sample-based camera (SBC): a collection of binary space partitioning (BSP) trees that store planar pinhole cameras at their leaves. A scene point is projected by invoking the cameras that contain it in their frustums. Reflections are rendered by projecting the scene geometry then rasterizing in hardware. The rasterization routine supports view dependent lighting of reflected objects and reflector surface properties, such as Fresnel and attenuation-

Fig. 3. Environment mapping (left), sample-based camera at 60 fps (middle), and ray tracing (right).

with-distance effects (Fig. 1).

SBCs are constructed from the reflected rays defined by the desired view and the scene reflectors. Fig. 4 shows an example with view C, reflectors A and B, and reflected rays $\{a_0e_0, a_1e_1, a_2b_2, a_3b_3, b_4e_4, b_5e_5\}$. Points e_i are obtained by clipping the reflected rays with a scene bounding box. Sets of neighboring rays that hit the same sequence of reflectors define pinhole cameras. The rays between a_0e_0 and a_1e_1 define camera c^a_{01}, the rays between a_2b_2 and a_3b_3 define c^a_{23}, and the rays between b_4e_4 and b_5e_5 define c^b_{45}. The first two cameras represent first order reflections, while the third represents second order reflections. The ray sets are chosen so that the cameras meet a user-specified projection accuracy.

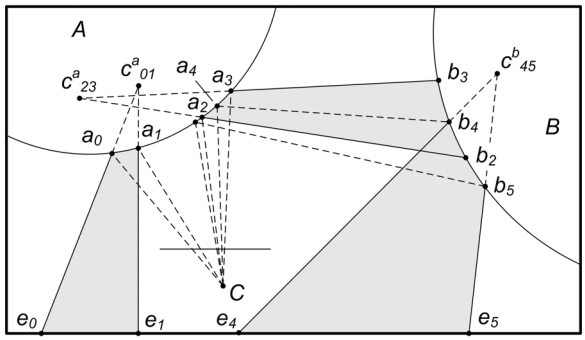


Fig. 4. Sample-based camera concept.

For first order reflections and a moderate output resolution (720x480), we build one SBC per frame on the fly (*runtime mode*). For higher order reflections or for high resolutions, we precompute SBCs at the nodes of a tetrahedral grid. At each frame, we retrieve the tetrahedron that contains the current view, project with its four cameras, and blend the projections. This *reflection morphing mode* achieves interactive performance on scenes with higher order reflections at 1440x960 resolution. The only case that it does not support is higher order reflections on independently moving reflectors.

The paper is organized as follows. Section 2 surveys prior work on rendering reflections. Sections 3–5 describe runtime mode and Section 6 describes the extension to reflection morphing mode. Section 7 discusses results and future work.

II. Prior work

The importance of rendering reflective surfaces has been recognized early on in computer graphics. Phong lighting and shading [1] is equivalent to reflecting light sources in shiny surfaces by searching for the appropriate eye, normal, and light vector combination. Reflections on arbitrary reflectors could be computed using hypothetical hardware that supports a very large number of lights. Planar reflectors are rendered by mirroring the camera across the reflector plane and using stenciling or texturing to confine the reflected world to the reflector [2]–[4].

A. Environment mapping

Interactive rendering systems approximate reflections on curved reflectors using environment mapping [5]–[8]. The environment map is a panorama of the scene rendered from the centroid of the reflector. The reflector is rendered by looking up reflected rays in the environment map using only their orientation. It is assumed that all reflected rays originate from the same point. The approximation works well for objects far from the reflector; for nearby objects, the errors are substantial. For example in Fig. 3, the front columns and the cube are close to the surface of the reflector. Ray tracing and SBCs correctly draw the reflections close to the real objects, whereas environment mapping fails to convey the proximity of the objects to the reflector. Other disadvantages of environment mapping are lack of motion parallax and no support for higher order reflections.

B. Projection methods

Better results can be obtained by solving the problem of projecting reflected points. Hanrahan and Mitchell describe a search procedure for the projection of reflected points if the reflector surface is given by an implicit equation [9]. Ofek and Rappoport [10] render reflections on tessellated reflectors by projection followed by feed forward rasterization. For each reflector, they compute a reflection subdivision consisting of one cell per reflector face. A scene point p is projected by finding its cell, interpolating the cell vertices to obtain an approximate reflection point q and surface normal n, and mirroring the ray $p-q$ around n. The reflection subdivision is searched using an approximate representation, called an explosion map.

Our algorithm has several advantages over Ofek and Rappaport, which is the closest prior work to ours. SBCs project with high, guaranteed accuracy specified by the user. The construction algorithm adaptively constructs a compact space partitioning that achieves this accuracy for each frame based on the view point and on the scene complexity. Explosion maps project approximately, at a fixed resolution without an error bound. We project higher order reflections in the same way as first order reflections, hence with the same cost and accuracy. Explosion maps render higher order reflections recursively, so the cost is proportional to the reflection order and the errors accumulate.

C. Image-based methods

The problem of reflections has also been studied by researchers in image-based modeling and rendering (IBMR). Light fields [11], [12] support view-dependent effects including reflections. A large number of rays need to be stored even for small scenes. To alleviate this problem, IBMR techniques were developed that use some explicit form of geometry. Surface light fields store all rays originating at each point of a surface [13], [14]. In view dependent texture mapping, a surface is sampled from several directions [15], [16]. Both techniques work well for surfaces of limited specularity. Highly reflective surfaces require a dense sampling of the possible view directions, which translates into an impractical number of samples.

Lischinski [17] proposes a scene representation based on layered depth images (LDIs) [18]. The scene geometry is captured with a set of 3 orthogonal LDIs. The view-dependent scene information is stored in a light field of low resolution LDIs, which provides glossy reflections. Mirror-like reflections are rendered by ray tracing the geometry LDIs, which alleviates the database size problem but reduces the performance below interactivity. Hakura [19] describes parameterized environment maps which are a set of precomputed reference reflection images. The images are parameterized such that they best match in least-mean-squares sense the true reflection when used as environment maps. Good reflections are obtained nearby the reference viewpoints and rendering takes advantage of the hardware supported environment mapping. The method has the disadvantage of lengthy preprocessing (more than 20 minutes per viewpoint) which restricts its use to 1D parameterizations of the viewpoint space and to static scenes.

Like IBMR methods, the SBC approach uses discrete representations of functions that are difficult to compute, which are then interpolated. Whereas IBMR methods rely on sampling the plenoptic function, SBCs sample the complex projection function of vertices in scenes with reflectors. In this analogy, the IBMR reference images correspond to the SBCs. Reference images become obsolete when a diffuse object moves and recomputing them requires handling the scene at its full complexity. SBCs are better suited for rendering highly specular reflections because they are independent of the diffuse part of the scene.

D. Hybrid methods

Between projection and IBMR, hybrid methods separate the geometry from the illumination of the reflector. Heidrich [20] captures the geometry of the reflector in a light field that maps rays to rays, rather than rays to colors. For a given view, the geometry light field provides the reflected rays that are then colored using an environment map, a regular light field, or ray tracing. The approach trades performance for accuracy. Cabral [21] combines BRDFs with lighting environments in precomputed radiance environment maps. Although hybrid methods allow one to modify the reflector and the illumination independently, they suffer from the other IBMR method disadvantages discussed above because the illumination is captured with images.

E. Ray tracing

Reflections can be computed accurately using ray tracing [22], [23], a general technique that produces high quality images. The ray tracing pipeline is less efficient than the feed forward pipeline because considerable computational effort has to be spent to decide which primitive affects a given output image pixel. Numerous research efforts are targeted at accelerating ray tracing. Wald [24], [25] has demonstrated real-time ray tracing on small scenes on a single general-purpose CPU with vector floating point extensions. Hardware has been developed to accelerate off-line ray tracing [26]. Complex scenes were ray traced at interactive rates on shared memory parallel computers [27] and on clusters [25]. The fixed function pipeline implemented in commodity graphics accelerators has been replaced with a pipeline that offers programmability at the vertex and fragment levels. The programs that could originally be executed to process vertices and fragments were too simple to implement ray tracing [28]. The programmability of GPUs has advanced sufficiently to allow limited ray tracing. But for the foreseeable future, GPUs will remain primarily feed forward rendering engines.

F. Non-pinhole camera models

Our solution for rendering reflections is based on a general, non-pinhole camera. Non-pinhole camera models have been studied in computer vision for 3D imaging applications. Examples include the push-broom camera [29] and the two-slit camera [30], which are special cases of a linear camera [31] that collects linear combinations of three rays. A linear camera cannot model the entire set of reflected rays defined by a pinhole and a set of curved reflectors. Computer graphics researchers have also explored non-pinhole cameras. Other than the light field discussed above, examples include multiple-center-of-projection cameras [32], multiperspective panoramas for cel animation [33], and image-based lenses for modeling real cameras [20]. None of these are suitable for rendering reflections.

The image-based lens technique is related to SBCs. The rays exiting a real lens are approximated with a set of pinhole cameras. However, the scene is rendered with *each* pinhole camera and the images are blended together. This is appropriate for simulating real lens effects, such as non-zero aperture, where the projection of a vertex is ambiguous. But projecting every vertex in every pinhole camera is prohibitive for interactive reflection rendering. The contribution of SBCs is to decompose a set of reflected rays into a set of non-overlapping pinhole cameras. This allows us to project reflected vertices quickly and thus to render reflections interactively.

III. Algorithm Overview

The input to our reflection rendering algorithm is a scene description, a desired view, a reflection order cutoff, a down-sampling factor, and a projection accuracy in desired image pixels. The scene consists of diffuse and reflective objects modeled with triangle meshes. Fig. 5 shows the main steps of the algorithm.

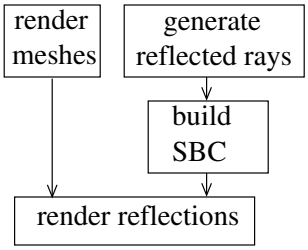


Fig. 5. Algorithm overview.

A. Render meshes

The reflective and diffuse meshes are rendered in hardware with z-buffering. Diffuse meshes are rendered as usual to generate their final image. Reflective meshes are rendered in the stencil buffer. The stencil is set to the id of the reflector to confine the reflection to the visible part of the reflector. Fig. 6 (left) illustrates this step on a scene with two spherical reflectors.

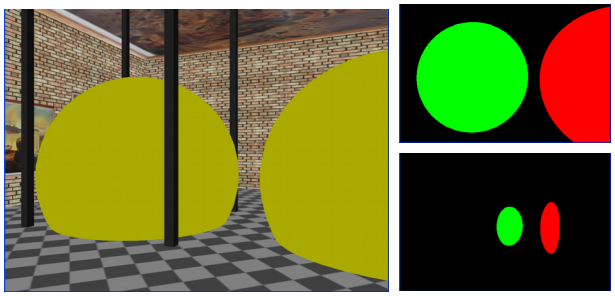


Fig. 6. Frame buffer after mesh rendering with non-zero stencil values visualized in yellow (left). First order (top) and second order (bottom) reflected rays.

B. Generate reflected rays

Reflected rays are generated from the desired view. The reflected rays are stored in a 2D map, which is typically down-sampled to half or quarter desired image resolution. Fig. 6 (right) shows a half resolution ray map with first and second order rays. A ray is represented by its tail, head, and first reflector point; these are $[a_1, e_1, a_1]$, $[a_3, b_3, a_3]$, and $[b_4, e_4, a_4]$ in Fig. 4. First order rays are computed on the GPU. Higher order rays are computed by ray tracing the reflectors alone, without the diffuse objects. Thus, the cost is proportional to the reflector complexity, which is typically a small fraction of the scene complexity.

C. Build cameras and render reflections

Section 4 describes how SBCs are built from the ray map and Section 5 describes how they are used to render reflections. Fig. 7 shows the output.

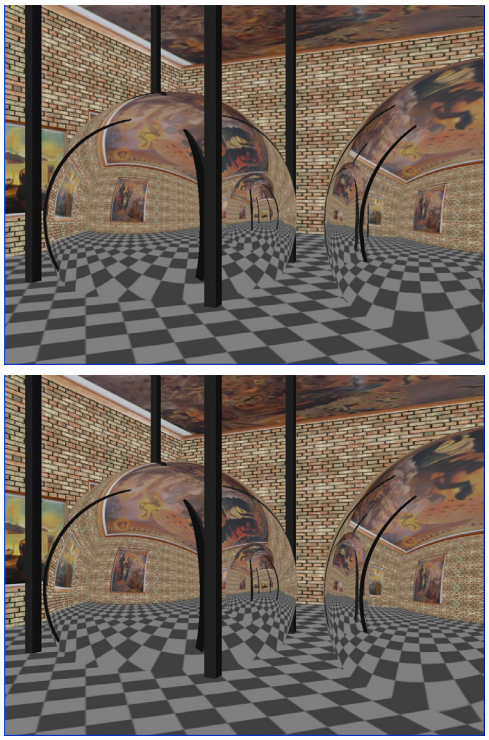


Fig. 7. Output image (top) and ray traced image (bottom).

IV. Sample-based camera construction

An SBC defines a mapping from a scene point to its reflections in the desired view. The SBC is built from the ray map. The rays are first partitioned into reflection groups. A reflection group $R_1 R_2 \cdots R_n$ comprises all the rays with the same reflection history. The reflection history of a reflected ray r is the list of reflectors encountered by the desired view ray that generates r. A projection function is computed for each reflection group. For example in Fig. 8, p is projected to first reflector point a_1 in reflection group A and to a_2 in AB. The SBC mapping is the union of the projection functions of its reflection groups.

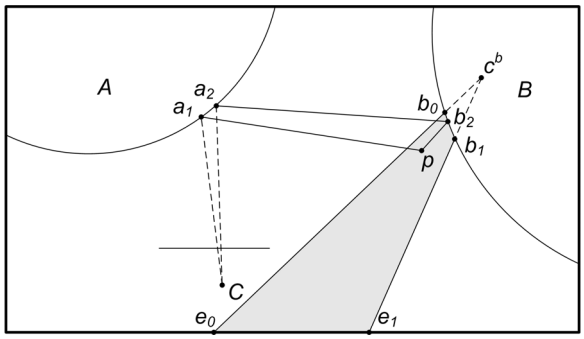


Fig. 8. Sample-based camera projection.

A reflection group projection function is defined by a set of pinhole cameras whose frustums encompass the rays of the group. Our example shows a camera for the AB reflection group with center of projection c^b, image plane $b_0 b_1$, and frustum $e_0 b_0 b_1 e_1$. A scene point is handled by the camera whose frustum contains it: the point is projected onto the camera image plane then is mapped to the first reflector. The example camera projects p to b_2 on image plane $b_0 b_1$ then maps b_2 to a_2 on A. A scene point that is not contained in any camera frustum does not project.

A camera is constructed for a set of rays as follows. The center of projection, o, is the least-squares fit of a point that lies on the rays. The equation for a ray with tail t and direction vector d is $o \times d = t \times d$, which yields three scalar equations in o_x, o_y, o_z. The image plane is fitted to the ray tails. The view frustum is chosen to contain the tails and the heads. The mapping from the image plane to the first reflector is a quadratic, $f(u, v)$, that is constructed by least-squares fitting the rays to their first reflector points. Each ray generates the equations $f(u_t, v_t) = q$ and $f(u_h, v_h) = q$ in which q is the first reflector point and (u_t, v_t) and (u_h, v_h) are the tail and head image plane projections.

The cameras are stored in the leaves of a BSP tree [34]. Fig. 9 shows the BSP tree for the A reflection

group of Fig. 8. It has 6 internal nodes with splitting planes 0–5 and stores cameras c_0–c_6 at its 7 leaves.

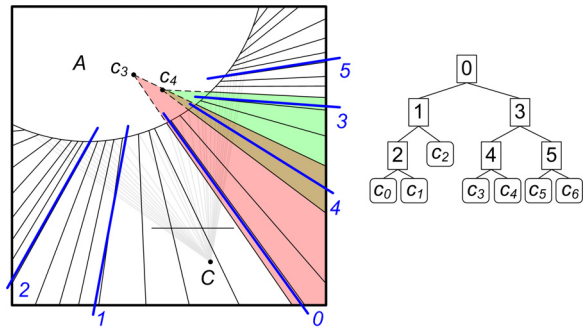


Fig. 9. Binary space partitioning and its tree.

```
BuildBSP(rays, Accuracy)
    phc = FitPinholeCamera(rays)
    if Error(phc, rays) < Accuracy
        return Leaf(phc)
    (plane, posSet, negSet) = Split(rays)
    if split fails
        return Leaf(rays)
    leftChild = BuildBSP(posSet, Accuracy)
    rightChild = BuildBSP(negSet, Accuracy)
    return Node(plane, leftChild, rightChild)
```

Fig. 10. BSP construction algorithm.

A reflection group is decomposed into a BSP tree of cameras by the algorithm in Fig. 10. The first step is to fit a camera to the set of rays in the group. If the fit is accurate, the camera is stored in a leaf node. The approximation error is estimated as the maximum error at the heads and tails of the rays. This error is the distance in pixels between the projections onto the desired view of the true and approximate first reflection points. If the fit fails, a plane is chosen heuristically to split the input roughly in half. It passes through the centroid of the ray tails, is parallel to the mean reflected ray direction, and is perpendicular to the diameter of the tails. The rays that intersect the positive/negative half spaces are assigned to the left/right subsets. A ray is assigned to both subsets if its unit frustum intersects the plane. The unit frustum of ray (u, v) in ray map coordinates is the set of four rays $(u \pm 1, v \pm 1)$. For example, the overlap between cameras c_3 and c_4 in Fig. 9 is due to the unit frustum that intersects plane 4. Splitting fails when either subset equals the input set; the rays are stored in a list at the leaf. Otherwise, the recursive case occurs.

The first reflector point of scene point p is found by first traversing the BSP tree to find the leaf that contains p. If the leaf contains a camera, the first reflector point is computed by the two-step procedure above. If the leaf contains a ray list, the unit frustum that contains p is found and the first reflector points of its rays are interpolated.

V. Rendering Reflections

A reflection is generated for every diffuse object in every reflection group. The diffuse meshes are projected into the desired view then are rasterized in hardware.

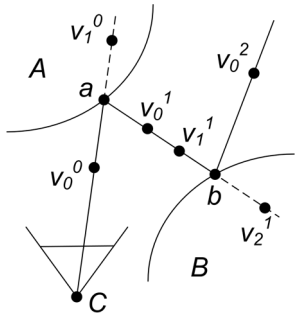


Fig. 11. Visibility cases.

A. Projection

A diffuse mesh vertex is projected into a reflection group by computing the first reflection point then offsetting along the desired view ray for correct visibility. Fig. 11 shows the visibility ordering along a reflection path from C to A to B and beyond. There are three visibility cases.

1) The closest scene point or reflector surface is visible (v_0^0 occludes a, which occludes v_1^0).
2) Within a reflection group, the point closest to the last reflector surface is visible (v_0^1 occludes v_1^1).
3) Between reflection groups, lower order points are visible (v_1^1 occludes v_0^2).

Case 1 is handled by z-buffering in Section III-A. Cases 2 and 3 are handled by offsetting the first reflector point (a) along its desired view ray (ca). For a k-order reflection of v, the offset is $(k - 1 + z/D)g$ where z is the distance from v to the reflection camera image plane, D is the scene diameter, and g is the depth range per reflection order. We choose g equal to $D/(k_{max} + 1)$ and set *yon* to $2D$.

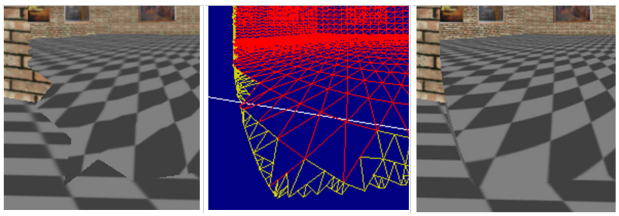


Fig. 12. Jagged reflection edges (left) alleviated by subdivision (middle, right).

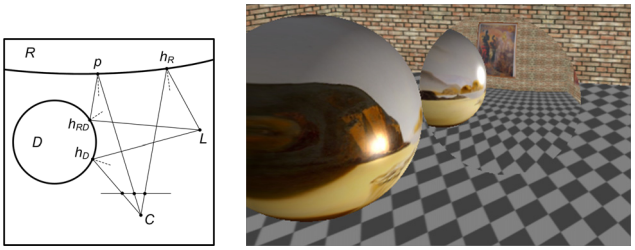


Fig. 13. Reflection with specular lighting.

B. Rasterization

A triangle is rendered when all three vertices project and is discarded when none project. A triangle is called mixed when one or two vertices project. Discarding mixed triangles creates jagged reflection edges. We solve this problem by subdividing mixed triangles whose area exceeds a threshold (Fig. 12).

When a reflected triangle is rasterized, its true, curved edges are approximated with straight edges. This approximation is acceptable only for small triangles. As a preprocess, we subdivide the scene triangles to limit the edge lengths.

C. Lighting and shading

SBCs render reflections by computing a reflected mesh for each diffuse mesh. The reflected mesh is placed and distorted to form the correct reflection when seen from the desired view. Lighting and shading that does not depend on the vertex scene position (lighting baked into vertex colors, diffuse directional lighting, and texture mapping) is unaffected by reflection and is carried out with the data (vertex colors, texture coordinates, normals) used for the unreflected mesh.

We support point light sources and specular lighting with GPU shaders that take into account the original position of the reflected vertices and the true eye vector. Fig. 13 shows a scene with a point light source L that is rendered with our method. The sphere D has a specular highlight at h_D, whereas its reflection on R has a highlight at h_{RD}. The mirror has a specular highlight at h_R. The eye vector at h_{RD} is given by $p - h_{RD}$ and not by $C - h_{RD}$. The highlights on the sphere correctly occur at different locations.

D. Reflector surface properties

SBCs provide a projection function for vertices that reflect from curved surfaces. This allows us to compute reflections by feed-forward rendering the reflected objects, which was previously possible only for planar reflectors. Because of the feed-forward approach, SBCs handle perfect, mirror-like reflectors directly (Fig. 14a). Such reflectors are challenging because reflection artifacts stand out and because they require a high sampling density in the case of image-based reflection rendering methods (light fields, view-dependent texture mapping).

Reflectors with a diffuse component are readily handled in the SBC framework. The diffuse component and the reflection are blended with a simple shader. The images c_i in Fig. 14 show Fresnel reflections with blending weights proportional to the square of the dot product of the eye vector and the surface normal. The weights are scaled linearly to the specularity interval $[s^f_{min}, s^f_{max}]$. In images d_i, the reflection weight is attenuated with the square of the distance between the reflected object and the reflector surface. This distance is already computed during SBC projection for visibility purposes. As the distance increases from 0 to d_M, the specularity decreases quadratically between s^a_{max} and s^a_{min}. Fig. 14b combines the two effects.

E. Antialiasing

Curved reflectors considerably minify distant parts of the scene, which makes the problem of antialiasing challenging in ray tracing, particularly for higher order reflections when the angle between neighboring rays grows large. Rendering reflections with the feed forward approach simplifies antialiasing. Each triangle is processed, which eliminates the danger of sub-sampling the geometry, and textures are correctly minified by mip-mapping.

VI. Reflection Morphing

We switch to reflection morphing mode when the ray map is too large for us to build the SBC

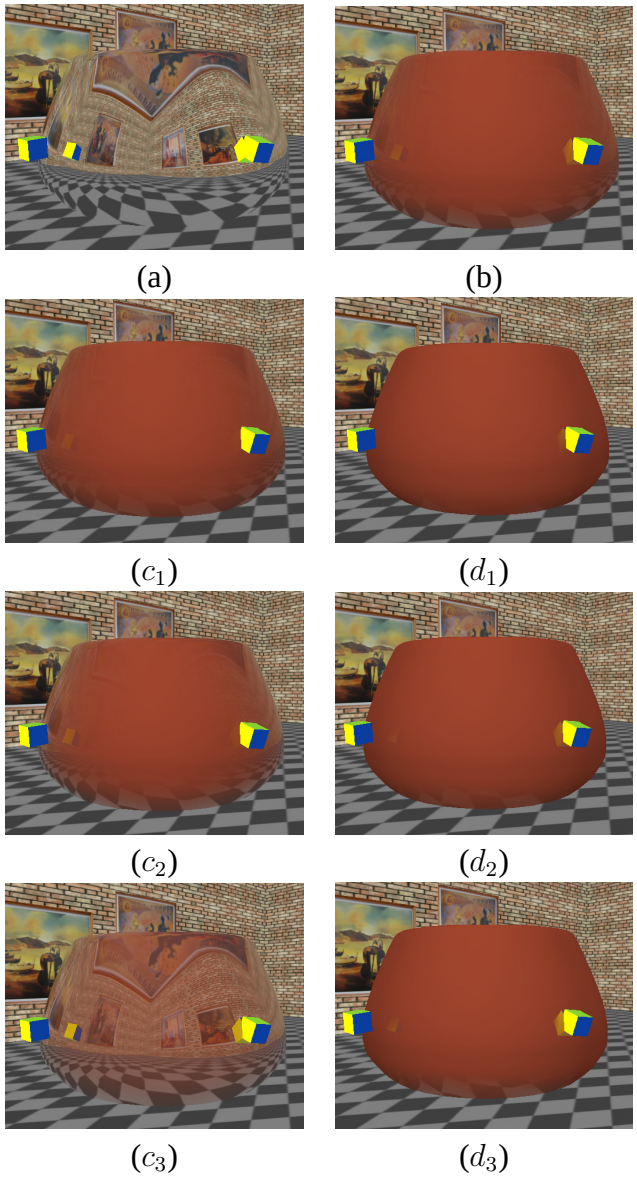


Fig. 14. Reflector surface properties: (a) perfect reflector; (b) reflector with diffuse component; (c_1–c_3) Fresnel reflections with s^f_{min}/s^f_{max} of 0.0/0.5 (c_1), 0.0/1.0 (c_2), and 0.5/1.0 (c_3); (d_1–d_3) Reflection attenuation with $d_M/s^a_{min}/s^a_{min}$ of 0.15/0.0/0.25 (d_1), 0.3/0.0/0.25 (d_2), and 0.3/0.0/0.5 (d_3).

on the fly or when higher order reflections are desired, which entails ray tracing the reflected rays. A regular 3D grid is attached to each reflector. Each grid cell is divided into 6 disjoint tetrahedrons. At every grid node, a panoramic SBC is built from a cube ray map to cover all view directions. The construction algorithm is as before. The grid of SBCs for reflector R handles all reflection groups that begin with R.

Reflections are rendered on each reflector using the four SBCs at the vertices of the tetrahedron that contains the current viewpoint. A vertex is projected with each SBC and the four projections are blended barycentrically. Static vertices that reflect on static reflectors are optimized: their four projections are reused with varying weights while the viewpoint stays within a tetrahedron. For moving reflectors, scene vertices are first transformed into the reflector coordinate system and are then projected. Higher order reflections are supported only if their reflectors have no relative motion, since such motion renders the reflected rays obsolete.

A vertex is ambiguous if it projects in some SBCs of its tetrahedron, but not in others. Such vertices cannot be morphed. Discarding triangles with ambiguous vertices produces visible artifacts (Fig. 15). The intersection of the four reference reflections is insufficient to render the desired view reflection. Ambiguous vertices are problematic at curvature discontinuities because there is a significant disparity between the four reference reflections. They also occur at reflector silhouettes, but the impact is negligible because the disparity between the four reference reflections is small.

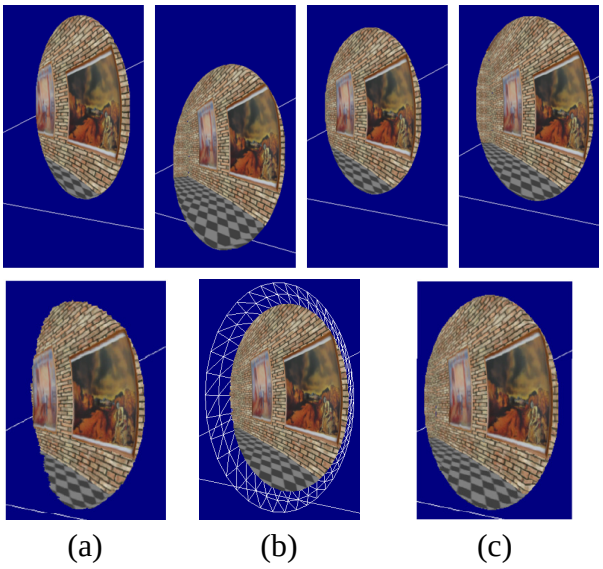


Fig. 15. Ambiguous vertices problem: four reference reflections (top), incorrect reflection (a), extended reflector for ray map construction in wireframe (b), and resulting correct reflection (c).

We handle ambiguous vertices in two ways. The first method uses extended reflectors during camera construction (Fig. 15). The reflectors are extended in small increments until no tail or head in any ray map is ambiguous. This approach takes into account the actual shape of the reflector and the scene bounding box, which is used to clip the reflected rays. The second method extends the field of view of the BSP

leaf cameras, which provides projections for points outside the ray maps.

VII. Discussion

We conclude the paper with results on the quality and speed of sample-based cameras followed by plans for future work.

A. Quality

Rendering reflections with an SBC entails approximations in the projection of reflected vertices and in the rasterization of reflected triangles.

The projection accuracy depends on whether the vertex maps to a BSP tree leaf with a pinhole camera or a leaf with a ray list. Pinhole cameras are guaranteed to project with the input accuracy (1 pixel for the images in this paper; see Fig. 17). Ray list projection interpolates between the four closest rays in the list, so the accuracy is one ray map pixel or better, which corresponds to d desired image pixels for a down-sampling factor of d.

Fig. 16 shows the effect of the ray map resolution on the reflection quality. (The same reflection is rendered in Fig. 14b with a 360x240/2 ray map.) For low resolutions, the pinhole camera fitting fails and most BSP tree leaves contain ray lists. Reflectors with strong diffuse components can be rendered with the 90x60/8 ray map, and good results are obtained on mirror-like reflectors with down-sampling factors of 4 or less.

In reflection morphing mode, the barycentric blending introduces an error that grows with the disparity between the four reference reflections. The error is largest when the reference SBC viewpoints are close to the reflector, but even then it remains small (5 pixels for our scenes).

During rasterization, the curved edges of a reflected triangle are approximated with straight edges, and the reflection inside the triangle is approximated by model space interpolation. We have found it easy to control this error with a fixed, uniform subdivision of the diffuse meshes.

Fig. 17 compares SBC and environment mapped reflections to ray tracing. The SBC provides a virtually perfect reflection. Small differences are visible at edges and on the near tiles of the floor whose triangles are large, hence less accurately rasterized. The environment mapped image is completely wrong: the reflections of the particle, of

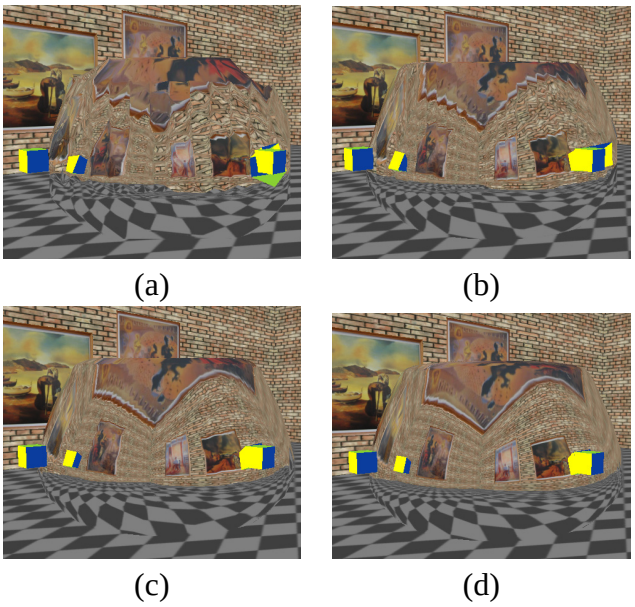


Fig. 16. SBC reflections with ray map resolutions/down-sampling factors of (a) 22x15/32, (b) 45x30/16, (c) 90x60/8, (d) 180x120/4.

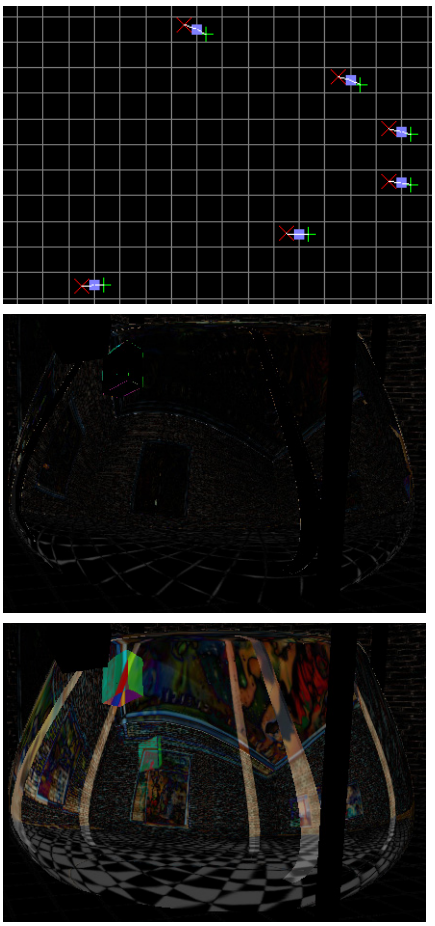


Fig. 17. Projection accuracy: (top) the head (diagonal red cross) and tail (straight green cross) of a few reflected rays are projected onto the desired image pixel grid (white) using their leaf camera; the approximate projections are within one pixel of the shared correct projection (blue square); (left) difference between SBC and ray traced images in Fig. 3; (right) difference between environment mapped and ray traced images.

the columns, and of the part of the floor near the reflector are tens to hundreds of pixels from their correct locations.

B. Speed

The reflection rendering time goes to ray map generation, SBC construction, and vertex projection.

Ray map generation: Maps of first-order reflected rays are generated in hardware by rendering the surfaces of the reflectors with a GPU program that encodes the direction (two fractional numbers) in the four color bytes. Z, color, and stencil are read back to obtain the tails, heads, and reflector ids of the reflected rays. 360x240/180x120 ray maps are generated in 27ms/9ms when every ray hits a reflector (the entire desired view is covered by reflectors). Timing data was obtained on a 3.4GHz 3GB Pentium 4 Xeon PC with a 256MB Quadro FX 3400 NVIDIA graphics card. The time needed to trace higher order reflected rays depends on the complexity of the reflector.

SBC construction: The splitting plane heuristic generates balanced trees for our test scenes. Constructing a balanced BSP tree from n input reflected rays takes $n \log n$ time and the traversal to find the first reflector point takes $\log n$ time. Fig. 19 gives SBC construction statistics for the reflectors and the views shown in Figs. 18, 14, 7, and 1. Each table entry is split in two: the left/right figures are for a quarter/half resolution ray map. The SBCs have a maximum size of a few MB; SBC construction performance is 10–15 Hz when using quarter resolution ray maps.

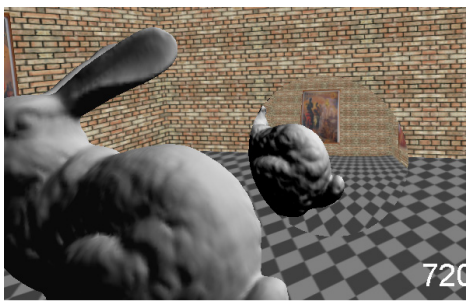


Fig. 18. Sample reflection.

In reflection morphing mode, we construct the SBCs as a preprocess. A 17x17 layer of the 17x17x8 grid used for our test scene is computed in an hour. Two layers fit in memory and are loaded at startup.

Scene	Reflected Rays *(x10³)*		Cov. *(%)*	Size *(KB)*		Construction time *(ms)*	
Spherical cap	4	16	19	153	597	21	33
Teapot body	7	28	33	191	703	93	295
Two spheres	9	35	40	549	2,331	65	328
Automobile	5	18	20	315	1,256	62	463

Fig. 19. SBC construction performance for four test scenes.

Vertex projection: In run time mode, vertex projection is performed in hardware. After the BSP is computed on the CPU, the BSP trees are packed and loaded into texture memory using the vertex texture technique. The vertex program first executes a loop that finds the leaf to which the vertex belongs. If no leaf is found, the vertex is discarded. Once the leaf is known, its data is used to move the vertex to the offset first reflector point. The average vertex projection performance is 20 million vertices per second (Mv/s) for pinhole camera leaves and 2.5 Mv/s for ray list leaves. Compared to projecting on the CPU, the GPU brings a 5 fold speedup. The longer ray list time is due to the sequential processing of the rays.

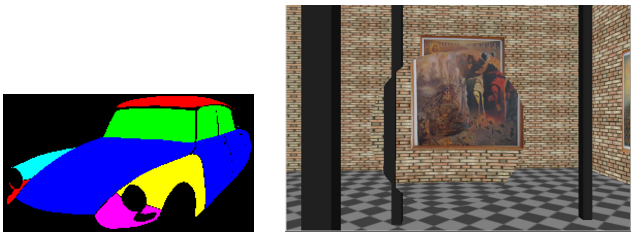
In reflection morphing mode, the projection is performed on the CPU and the projections of static vertices are reused. The average static/dynamic vertex projection performance is 12.5/1 Mv/s.

Overall performance: In run-time mode, performance is dictated by SBC construction. The automobile and teapot body scenes are rendered with average frame rates of 6 and 8 Hz, and the bunny is rendered at 15 fps. In reflection morphing mode, performance is dictated by the number of vertices. The teapot body and two spheres scenes have 10,000 vertices (20,000 triangles) and are rendered at 60/30 fps for 720x480/1440x960 output resolutions.

C. Compound reflectors

SBCs assume that each scene point has at most one reflection in each reflection group. This condition is satisfied if the rays of a reflection group do not intersect inside the scene volume. Convex reflectors satisfy this condition. We can handle concave reflectors when we can split them into pieces whose rays do not intersect. Fig. 1 was rendered by subdividing the automobile into 7 parts (Fig. 20a)

that generate 7 reflection groups. We use the same approach to store refraction rays, which allows us to render refractions with SBCs (Fig. 20b).



(a) (b)

Fig. 20. (a) automobile reflector subdivision; (b) refraction through bi-convex glass lens.

Some reflectors are rendered adequately by environment mapping, hence do not warrant the added cost of sample-based cameras. The two methods can coexist. Fig. 21 shows images where some reflectors are rendered with environment mapping (teapot lid, handle, and spout, and automobile bumper and wheel caps) while the others are rendered with an SBC.

Fig. 21. Hybrid reflections.

D. Conclusion

Sample-based cameras produce high quality reflections on curved reflectors at interactive rates by projecting reflected vertices efficiently then shading and rasterizing with graphics hardware. They produce images of ray tracing quality, and are more efficient because they avoid searching for the scene point visible at a pixel.

SBCs are more compact than image-based methods because they sample the projection function instead of sampling the reflections. A BSP tree of pinhole cameras is a powerful, versatile way of encoding a projection function. The number of cameras decreases with the curvature of the reflector surface and equals one for planar reflectors. Image-based methods are more efficient for complex *static* scenes with low specularity reflectors, since they do not require one to render the scene for every reflection. SBCs are better for dynamic scenes and for scenes with highly specular reflectors.

E. Future work

SBCs provide a new framework for accurately rendering view dependent effects at interactive rates. We will extend this work in several directions.

The SBC provides the requisite per-pixel data (reflected scene points, eye vectors, and reflector normals) for realistic simulation of many types of surfaces. We will extend our set of shaders to handle noisy, bumpy, and glossy reflective surfaces by integrating normal maps into the SBC framework.

We will attempt to remove the restriction of one projection per reflection group. One approach is to partition the reflected rays into disjoint sets, in part by splitting individual rays. Another approach is to find all the reflected unit frustums that contain a given scene point. We would need to group the reflections of the three vertices of a triangle to form reflected triangles. Alternately, we could dispense with connectivity and render reflections by splatting.

We aim to improve the performance of the algorithm. SBC construction and projection have good asymptotic running times. One approach to improve performance is to reduce the number of reflected rays by selecting a subset that adequately captures the reflections in the current view. Another approach is to implement SBC construction in hardware.

ACKNOWLEDGMENT

We are grateful to Piti Irawan, Chris McDonald, Mihai Mudure, Andrew Martin, and Jordan Dauble for contributing to this work, and to Chris Hoffmann for fruitful discussions. This research was supported by NSF grants CCR-9617600 and SCI-0417458.

REFERENCES

[1] B. Phong, "Illumination for computer-generated images," Ph.D. dissertation, University of Utah, 1973.

[2] P. J. Diefenbach, "Pipeline rendering: Interaction and realism through hardware-based multi-pass rendering," Ph.D. dissertation, University of Pennsylvania, 1996.

[3] T. Mcreynolds and D. Blythe, "Programming with opengl: Advanced rendering," 1997, sIGGRAPH 97 course.

[4] R. Bastos, K. Hoff, W. Wynn, and A. Lastra, "Increased photorealism for interactive architectural walkthroughs," in *Interactive 3D Graphics*, 1999, pp. 183–190.

[5] J. Blinn and M. Newell, "Texture and reflection in computer generated images," *Communications of the ACM*, vol. 19, no. 10, pp. 542–547, 1976.

[6] N. Greene, "Environment mapping and other applications of world projections," *IEEE Computer Graphics and Applications*, vol. 6, no. 11, pp. 21–29, 1986.

[7] P. Haeberli and M. Segal, "Texture mapping as a fundamental drawing primitive," in *Proceedings of the Fourth Eurographics Workshop on Rendering*, S. Cohen, Puech, Ed., 1993, pp. 259–266.

[8] D. Voorhies and J. Foran, "Reflection vector shading hardware," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1994, pp. 163–166.

[9] H. P. and D. Mitchell, "Illumination from curved reflectors," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1992, pp. 283–291.

[10] E. Ofek and A. Rappoport, "Interactive reflections on curved objects," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1998, pp. 333–342.

[11] M. Levoy and P. Hanrahan, "Light field rendering," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1996, pp. 31–42.

[12] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen, "The lumigraph," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1996, pp. 43–54.

[13] G. S. P. Miller, S. M. Rubin, and D. Ponceleon, "Lazy decompression of surface light fields for precomputed global illumination," in *Eurographics Workshop on Rendering*, 1998.

[14] D. N. Wood, D. I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D. H. Salesin, and W. Stuetzle, "Surface light fields for 3d photography," in *Proceedings of ACM SIGGRAPH*. ACM Press, 2000, pp. 287–296.

[15] P. Debevec, Y. Yu, and G. Borshukov, "Efficient view-dependent image-based rendering with projective texture-mapping," in *Proceedings of the 9th Eurographics Workshop on Rendering*, 1998, pp. 105–116.

[16] K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle, "View-based rendering: Visualizing real objects from scanned range and color data," in *Eurographics Rendering Workshop*, 1997, pp. 23–34.

[17] D. Lischinski and A. Rappoport, "Image-based rendering for non-diffuse synthetic scenes," in *Eurographics Rendering Workshop*, 1998, pp. 301–314.

[18] J. Shade, S. Gortler, L. He, and R. Szeliski, "Layered depth images," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1998, pp. 231–242.

[19] Z. Hakura, J. Snyder, and J. Lengyel, "Parameterized environment maps," in *Proceedings of the ACM Symposium on Interactive 3D Graphics*, 2001, pp. 203–208.

[20] W. Heidrich, H. Lensch, M. Cohen, and H. Seidel, "Light field techniques for reflections and refractions," in *Eurographics Rendering Workshop*, 1999, pp. 195–375.

[21] B. Cabral, M. Olano, and P. Nemec, "Reflection space image based rendering," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1999, pp. 165–170.

[22] T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, 1980.

[23] A. Glassner, *An introduction to ray tracing*. Academic Press, 1989.

[24] I. Wald, P. Slussalek, and C. Benthin, "Interactive distributed ray tracing of highly complex models," in *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, 2001, pp. 277–288.

[25] I. Wald, P. Slusallek, C. Benthin, and M. Wagner, "Interactive rendering with coherent ray tracing," *Computer Graphics Forum*, vol. 20, no. 3, pp. 153–164, 2001.

[26] D. Hall, "The ar350: Today's ray trace rendering processor," in *SIGGRAPH/Eurographics Workshop On Graphics Hardware*. ACM Press, 2001.

[27] S. Parker, W. Martin, P. Sloan, P. Shirley, B. Smits, and C. Hansen, "Interactive ray tracing," in *ACM Symposium on Interactive 3D Graphics*, 1999, pp. 119–126.

[28] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," in *Proceedings of ACM SIGGRAPH*, vol. 21, no. 3. ACM Press, 2002, pp. 703–712.

[29] R. Gupta and R. I. Hartley, "Linear pushbroom cameras," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 9, pp. 963–975, 1997.

[30] T. Pajdla, "Geometry of two-slit camera," Czech Technical University, Tech. Rep. 2002-02, 2002.

[31] Y. J. and M. L., "General linear cameras," in *8th European Conference on Computer Vision*, vol. 2, 2004, pp. 14–27.

[32] P. Rademacher and G. Bishop, "Multiple-center-of-projection images," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1998, pp. 199–206.

[33] D. N. Wood, A. Finkelstein, J. F. Hughes, C. E. Thayer, and D. H. Salesin, "Multiperspective panoramas for cel animation," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1997, pp. 243–250.

[34] H. Fuchs, Z. Kedem, and B. Naylor, "On visible surface generation by a priori tree structures," in *Proceedings of ACM SIGGRAPH*. ACM Press, 1980, pp. 124–133.

Voicu Popescu is an assistant professor of computer science at Purdue. He received his Ph.D. in 2001 from the University of North Carolina at Chapel Hill. His research interests lie in the areas of computer graphics, computer vision, and visualization.

Elisha Sacks is a professor of computer science at Purdue. He received his Ph.D. in 1988 from MIT under Gerald Sussman and Ramesh Patil. His research interests are geometric computing, scientific and engineering problem solving, mechanical design automation, and robotics.

Chunhui Mei is a postdoc in the Purdue computer science department.