Image-based Animation

Biswarup Choudhury
Indian Institute of Technology
Bombay & ETRI, South Korea
biswarup@cse.iitb.ac.in

Ambareesha Raghothaman
Indian Institute of Technology
Bombay
ambareesha04@gmail.com

Sharat Chandran
Indian Institute of Technology
Bombay
sharat@cse.iitb.ac.in

ABSTRACT

Animation has evolved over the years – from the early days of 2D animation to the present technology of using GPU-based shader programs for providing complex, photorealistic lighting. One thing has, however, remained constant – a geometrical model has been considered essential in computer animation. In this paper, we propose an alternative.
An image-based framework is presented for creating arbitrary motions of an object using only captured images of the object; no geometry of the object or the environment is provided by the user. *Photorealism is an immediate side effect as a consequence.* Specifically we preprocess a set of images of a static object under a set of carefully chosen lighting configurations. Now given an arbitrary environment in the form of images again, and any arbitrary three-dimensional path that the object is desired to move, our algorithm creates a motion sequence of the object — realistically composed in the new environment.

Keywords: Animation, image-based rendering, visual hull, image-based relighting, virtual and augmented reality.

1 INTRODUCTION

Traditionally computer-generated animation requires knowledge of object geometry, reflectance functions, lighting configurations, and the desired motion sequence. All of these are used in conjunction with computationally intensive global illumination techniques to render each frame of the desired motion. In this paper, we show there is an alternative. We create the same animation with no knowledge of the geometry of the objects from the user, and no surface properties. Even the environment that the objects are expected to 'live' in is provided in the form of images.

1.1 Motivation

Behind the scenes of any computer animated movie, the "layout crew" choreographs the characters in the set and uses a virtual camera to create shots. This painstaking process of changing the lighting, viewpoint, and character motion is repeated several times to capture the emotion of each scene. To get an idea of how long it takes to do such rendering, we quote Pixar who use a Renderfarm where each frame takes about six hours to render. Pixar claims that some frames have taken as many as ninety hours [19]. These numbers may represent the time taken to render a full resolution frame. Nonetheless, the time taken for a lower resolution frame is high, and more significantly, the whole process is laborious.

WSCG 2011 conference proceedings,
WSCG'2011, January 31 – February 3, 2011
Plzen, Czech Republic.

- **Question**: Can we reduce the production time and resources substantially?
  Answer: Yes, discounting the preprocessing time (which is highly dependent on the number and types of objects), our unoptimized implementation takes roughly a minute per frame.

- **Question**: If we had prior footage of an environment, can we realistically embed in it a shot, where a Ferrari is yanked, and hurled into the air ?
  Answer: Creating such an animation would require the use of a Ferrari in a set of a studio (or its complicated geometric model), but is infeasible due to commercial production reasons. Our method achieves similar effects (please see the smaller Spaceship video in the supplementary material in which three Spaceships fly in the Uffizi Gallery [7]).

1.2 Contributions

Clearly it is desirable to be able to quickly create an animation of an object performing an "impossible" motion in director-mood dependent environments, edit (camera position, lighting, and object path), and again quickly re-render it. An image-based solution is a promising alternative, which would enable the director to create any desired animation using only a set of captured images of the object. This paper describes a method that combines existing techniques to achieve this framework. Specifically, the novelty in our work is summarized as:

1. **Motion:** Unlike prior image-based methods, our static object captured in the studio can move freely in space at the whims and fancy of the user and live in any virtual or camera captured real world. The framework allows any complex motion, and does not need any object geometry from the user.

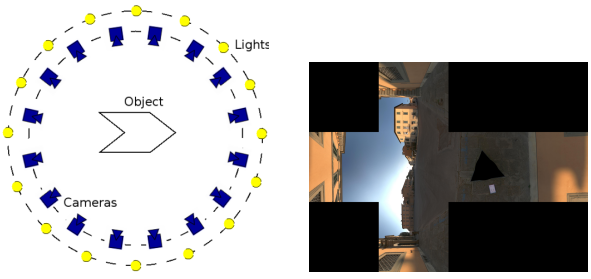
2. **Lighting:** Since the object is rendered in a novel environment, we cannot simply copy the studio acquired images and stick it to a frame of the animation by applying matting techniques. Worse, since the 'static' object is moving in every frame of the animation, at every frame, the lighting directions (of the novel environment) with respect to the object changes.

3. **Visual Hull:** Any image-based framework inevitably has to balance storage costs with accuracy. We use a variation of the popular visual hull technique to create a dense but "lightweight" intermediate representation.

2 RELATED WORK

Considerable research has been done to create animation sequences from captured videos. These can be categorized into video-based rendering (VBR), and video-based animation (VBA) techniques. The aim of VBR techniques ([15, 24, 29, 27]) is to render novel views of dynamic scenes using multiple-view video capture. These techniques reconstruct (often across the time domain) the surface of the moving object, and then use the video and the reconstructed geometry to create novel view sequences. Some VBR techniques ([4, 20]) reanimate dynamic scenes using model-based reconstruction techniques, which fit a generic model to observations from multiple views, but suffer characteristic limited visual quality. On the other hand, VBA techniques ([21, 22, 2, 1]) provide a representation of dynamic scenes captured from videos allowing synthesis of novel image sequences. These novel sequences are synthesized by concatenating captured video segments based on a transition graph. These techniques use similarity metrics to identify frames in the video which are candidate transition points. Some techniques have been proposed which use the concepts of VBR and VBA techniques to render novel views of reanimated animation sequences. For example, techniques such as [9, 25] focus was on capturing the appearance of a person performing a specified cyclic action (walking or jogging), and then reproducing the character performing the same motion under variable viewpoint and illumination.

While the techniques mentioned above use videos as input, there are techniques which use images to create novel animation sequences. In [3], the authors create an illusion of realistic animation of an object from images captured at different instant of time, by inserting motion blur between the consecutive images. Most image-based techniques segment the source image into layers, and create animations by applying dynamic brush strokes [11] or displacement maps [11] to each layer. These layers are then recomposited to form the animated sequence. A recent work [28] uses a source image of a group of moving animals (or birds) to identify



(a) (b)

Figure 1: The first schematic is the cross-section view of our capture space composed of two concentric spheres, one with cameras (blue) and the other with point light sources (yellow). The novel environment (b) specified as a cube map, is used to relight each frame of the desired animation.

the *ordered cyclic sequence of poses* of any animal in the group while in motion. This sequence is then used to create a motion cycle, which can be used to create novel animation sequences of the animal(s). This technique assumes that all the "key-poses" of the animal in motion are present in the source image, or they need multiple images to identify all the key-poses of the animal.

In all the techniques mentioned above, reanimating the object is mostly limited, primarily to the repetitive cyclic motions (human walking/jogging, animals walking, birds flying, water flowing) as is present in the input videos/images. In contrast, we propose a framework to synthesize arbitrary, complex animation sequences of the object, without recording any motion of the object. We require only images of the static object.

Thus, image-based animation as we conceive with all its variety has not been sufficiently explored.

3 OVERVIEW

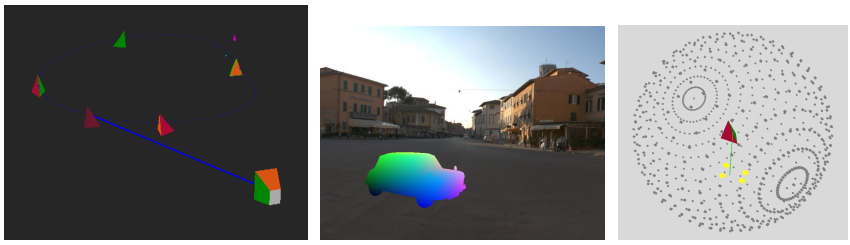
The general idea of image-based animation we propose in this paper can be implemented in various frameworks. Here we provide the following framework.

Inputs for Preprocessing: The primary input to our method are images of a single object captured from calibrated cameras which are distributed on a sphere (Figure 1(a)). For each acquisition camera, these images are captured under a set of lighting conditions sampled (Halton sampling as in [6]) on a concentric sphere.

The position of the (capture) cameras are recorded in a spherical coordinate system termed Acquisition Coordinate System (ACS), with the origin at the object centroid. The cameras may be assumed to be calibrated at the moment.

Inputs at Run-time: There are three inputs.

- The animator is presented with a GUI to specify in another coordinate system, the World Coordinate System (WCS), an arbitrary three-dimensional path along which he desires to animate the object. The



(a) One intermediate pose (b) Shape computation (c) Finding donor cameras

(d) Shape as seen by the four donor cameras

(e) Images relit in each of the donor cameras (f) Compositing

Figure 2: The method. In (a) we see a typical frame to be generated with the animation camera looking at a candidate position and pose. Using the specified view, we project a pre-computed visual hull to generate (b) the correct shape of the object of interest, in this case the MiniCooper car. The background is a novel environment given as input. In (c) donor cameras (shown in yellow) have been computed. In (d), the visual hull is conceptually projected on four donor cameras. The points P of the visual hull visible in (b) are also visible in the union (d) of these donor cameras, but not vice versa. In (e) we see how the MiniCooper would have appeared were pictures taken, not in the studio, but in the specified novel environment. Using some of these pixels, we see in (f) how the MiniCooper would look from the animation camera.

path is specified in the form of "key poses," each of which is created using mouse-clicks in the viewports of the GUI. Each 3D point on the path is considered as a pose of the object, comprising of positional coordinates and orientations.

- The desired vantage point and orientation of observing the desired animation is also specified in the WCS. We term it the Animation Camera.
- The novel environment, specified as a cube map in which the object is bathed (Figure 1(b)).

Once the key poses have been specified by the animator, we generate the intermediate poses to create the final (virtually continuous) animation sequence path using standard B-spline based techniques [10]. Suitable editing capabilities are also provided to change the path once specified, the animation camera parameters, and the number of intermediate poses generated.

Consider, in the WCS, an arbitrary animation camera position X looking at the object centroid position Y along the specified path. We construct the reverse view vector from Y to X and map this view vector from the WCS to the ACS, resulting in a virtual viewpoint in the ACS. Using an efficient searching algorithm, the closest candidate viewpoints (capture donor cameras) in the ACS are determined. The captured images from these donor cameras are used to synthesize the "view" as seen from the virtual viewpoint in the ACS (animation camera in WCS). This process is sketched in Figure 2.

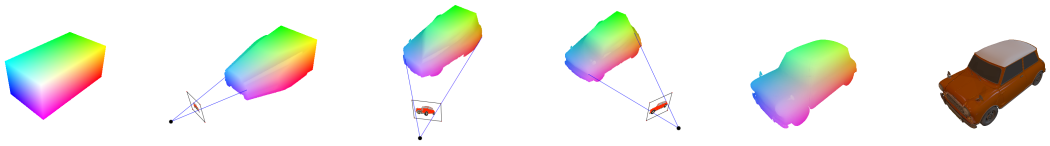
Our method works because of the phenomenon of *relative motion.* An object moving in front of a stationary camera (in our case, the animation camera in the WCS) produces the same image as that of the camera moving in the "opposite direction" with respect to the stationary object (captured as input in the ACS, in the preprocessing phase.) One can therefore compute the correct camera parameters. Then *persistence of vision* realistically creates the illusion of motion.

4 THE ALGORITHM

The shape of the object as seen from the animation camera is computed using careful view interpolation techniques (Section 4.1). The color of the object in the novel environment is computed using the "basis" lighting conditions obtained as input (Section 4.2). This is repeated for all poses (Y) along the path of the object in the WCS, each generating a frame of the animation.

4.1 Shape

An immediate way of finding the view from the animation camera would be to interpolate among the im-



(a) Initial point cloud (b) Iteration 1 (c) Iteration 5 (d) Iteration 9 (e) The final hull (f) Original Mini-Cooper

Figure 3: Visual Hull creation in progress: We start (Algorithm 1) with an initial point cloud. For each camera, using an image of the object silhouette, we determine the "relevant" points in the point cloud that needs to be stored. The remaining points in the point cloud are deleted. We keep iterating this procedure over all the cameras to produce our visual hull.

ages [5, 18], as viewed from the candidate closest viewpoints. However, a variety of rendering problems, including aliasing and ghosting artifacts manifest themselves. Thus the resulting image would most likely be unacceptable to the computer graphics community. To achieve higher visual fidelity, we could turn to techniques such as computing the point correspondences between the images of neighboring cameras [23, 13], or computing the optical flow between adjacent cameras [12]. Both of the above techniques are either highly time-consuming or are still error-prone. For highly complex models, visibility relationship from the novel viewpoint, and the ones taken earlier by the capture cameras should match.

Since we have the liberty to preprocess, we use the concept of image-based visual hulls [16, 17] as an intermediate step, for computing the view as seen from the animation camera. Note that in the input acquisition phase, and for each capture camera, we have the liberty of acquiring the image by placing a light source at the back of the object. This helps us to identify a proper silhouette of the object.

For each pose, the visual hull of the object is positioned and oriented in the WCS according to the specified pose parameters. The visual hull is then z-buffer projected on the animation camera (Figure 2(b)). At this point the shape of the object as seen from the animation camera is available. In the next section, we describe our novel visual hull computation algorithm, which computes a dense yet "lightweight" hull.

Visual Hull Revisited Various approaches [26, 14] to computing the visual hull of an object, given its silhouette data, have been proposed in the past. The idea behind all these algorithms is that if the camera parameters are known, the information from many 2D images can be used to obtain a good approximation of the shape of the object. This approximation is an upper bound estimate of the shape of the object. Since we get to arrange the acquisition setup, we use calibrated cameras; and for each camera, we capture an additional image of the object, in which the object can be well segmented from the background.

For computing the visual hull, we start with an initial cuboid point cloud (Figure 3(a)) of points. For each

Algorithm 1 CREATE-VISUAL-HULL (*CaptureCamList*)

```
1:  PointCloud = Create a point cloud
2:  for all camera in CaptureCamList do
3:    {SegmentedImage: Image with object segmented from background}
4:    for all P in PointCloud do
5:      (x,y) = Project P into camera
6:      if (x,y) = Background-pixel(SegmentedImage) then
7:        Delete P from PointCloud
8:      end if
9:    end for
10: end for
11: return SCOOP-OUT(PointCloud,CaptureCamList)
```

Algorithm 2 SCOOP-OUT(*PointCloud*, *CaptureCamList*)

```
1:  Delete points from PointCloud that have neighbors
2:  Mark remaining points in PointCloud as unseen
3:  for all camera in CaptureCamList do
4:    capzbuf ← ZBUFFER(CaptureCam, VisualHull)
5:    for all P in capzbuf do
6:      Mark P as seen
7:    end for
8:  end for
9:  Delete all unseen points from PointCloud
10: return PointCloud
```

acquisition camera, we project each point in the point cloud into its image plane and check, using the well-segmented image, whether the corresponding pixel belongs to the object or the background. If the pixel belongs to the object, we keep it; otherwise we delete it from the point cloud. As the iteration proceeds (Figure 3) a conservative estimate of the shape of the object is obtained.

The novelty in our visual hull algorithm begins here. The hull, thus created, includes points that are in the silhouette but also points invisible to all capture cameras. We term such a hull as *not* lightweight. The number of points 'inside' a non-lightweight hull are orders of mag-

nitude larger than those strictly on the bounding surface (shell), As we shall see later, our algorithm requires frequent projection of points in the visual hull; therefore a lightweight hull is desirable. On the other hand, the hull should be dense to capture fine details and avoid holes.

Our solution is to scoop out the hull interior. The first step is to exploit the fact that the initial point cloud was populated as a matrix. We therefore delete a point with six neighbors on the hull. We then z-buffer project the remaining points in the hull to track 'true positives,' i.e., points that are visible from at least one camera. Points which are not visible can be safely deleted. (Bitmaps are used for efficient implementation.)

In summary, first we start with a sparse point cloud and use Algorithm 1 to create a hollow, but sparse, visual hull. This hull is lightweight, but may not be dense enough. Therefore, for each point in the sparse hull, we populate a cube of space around it with closely spaced points. This becomes our new point cloud, on which we iterate Algorithm 1. In Section 5, we provide qualitative and quantitative results of our visual hull algorithm.

4.2 Color

The color of the visible points of the visual hull as seen by the animation camera is obtained from the images of its k closest capture cameras (donor cameras) in the ACS, relit with the lighting configuration specified as a novel environment in the WCS. Note that as the object moves along the desired path, the relative orientation of the light sources (corresponding to the novel environment) with respect to the object (pose) constantly changes. Thus, to determine the correct color information of the object as it moves along its desired path, one must incorporate this change of light sources' orientation for each pose of the object. For each pose of the object, we map the novel light sources' directions (in the WCS) to the ACS, and then use this mapped set of light source directions (in the ACS) as the novel illumination configuration to relight each of the k donor cameras. The relit images thus computed are used for determining the color of the pixels in the synthesized view for the virtual viewpoint (animation camera). However, the following questions need to be answered.

- For a candidate pixel p as seen in the animation camera, which pixel q in the donor camera d is relevant? Should we consider more than one donor for the same candidate pixel p?
- Overall, how many donor cameras are needed?

We answer these questions in an incremental, algorithmic fashion. At the end of shape computation we have the three dimensional coordinates of each visible point i in the visual hull. Starting with the closest donor camera, we determine the points in the visual hull visible to this donor camera. This procedure is incrementally, performed until all the visible points are tagged with at least one donor camera. This is again done using a z-buffer (Figure 2(b) and 2(d)).

More specifically, consider donor camera d, and a 3D point P (corresponding to a candidate pixel p) that is visible in the animation camera. The image corresponding to the donor camera d contains pixels corresponding to a set of points Q_d of points in the visual hull that are visible from d. Three cases may arise in increasing order of complexity:

1. $P \in Q_d$ & $P \notin Q_i$, $\forall i \neq d$: We have a clear (pixel) match for P in d, whose color value (after relighting) is assigned to pixel p.
2. $P \in [Q_d, Q_{d'}, \ldots]$, for some $d, d' \in [CaptureCamList]$: We perform a blend of the relit pixel values (as observed in all relevant donor cameras) to compute the color of pixel p.
3. $P \notin Q_d$, $\forall d \in [CaptureCamList]$: We assert a match for P in d, provided the depth value of a point in Q_d is close to the depth value of P.

Relighting At this stage we know which of the acquired images are relevant for the animation camera, i.e., the k donor cameras. We also know which pixels in these donor cameras map to the animation camera. Given our third input, the novel lighting environment specified as a cube map in the WCS, we obtain the lighting directions (in the WCS) and intensities of the 10 most significant light sources in the environment map (using HDRshop's lightgen plugin [7].) Using these 10 light sources mapped into the ACS, we perform relighting on the images corresponding to the k donor cameras using an algorithm based on the method in [6]. Note that, for every frame of the animation, since the relative configuration of these 10 light sources changes constantly with respect to the pose of the object, these 10 light sources' directions need to be mapped into the ACS for every frame (pose of the object).

5 IMPLEMENTATION AND RESULTS

We now describe some of the features of our implementation. All our experiments were performed on a Intel Centrino 1.73GHz Core Duo processor and 1GB RAM.

Preprocessing and Run-time: For proof of our concept, we simulated the capture setup by generating the input images using POV-Ray. In production, the system needs only photographs captured of the object on an acquisition setup as in [8] – which we do not have access to. Neither is the geometry of the object needed, and nor is there a need of a 3D renderer. Thus, our current experiments (using only the images of the objects as input) is indicative of the quality and resource (memory and computational) requirements of this animation framework for real world productions.

In our implementation, we use 762 cameras distributed on the bounding sphere. The number of basis

(a) Axe (b) Spaceship

Figure 4: Visual hull creation results: Each set is composed of two images, the one on the left is the visual hull computed using our algorithm, while on the right is a snapshot image of the same object. Observe the similarity of features displayed by our computed hull.

Figure 5: The Axe, in a pose, rendered under four different illumination conditions. Note the distinct change of color on the blade of the Axe.

lighting conditions (sampled on a concentric sphere using Halton sampling [6]) are fixed at 100. As a preprocessing step, we also compute a dense, but memory efficient visual hull using multiple iterations of Algorithm 1. We observed from our experiments that in most cases, two iterations are enough to extract a good estimate. For a path specified by the animator, we generated approximately 50-100 intermediate poses (final output frames). The computation of the intermediate poses takes a few seconds.

Shape: Once the visual hull is computed, to compute an accurate shape description of the object, devoid of aliasing effects along its edges and boundaries, we create, in software, a high resolution z-buffer (upto 4 times the resolution of the frame buffer) for the animation camera, i.e., for every pixel in the output frame, there exists four samples in the z-buffer, each corresponding to a 3D point in the visual hull. Thus we get a supersampled image as an output, which is used to give antialiased results. Figure 4 shows the qualitative results of the visual hulls of various objects created using our visual hull algorithm. Notice the accuracy of the reconstructed geometries with respect to the the original object shape. Table 1 provides the quantitative results of our algorithm, while Table 2 depicts results of comparison with a conventional visual hull algorithm.

Color: The location of the capture cameras are stored in a lightweight k-d tree (k=2) to enable efficient nearest neighbor search for donor cameras, once a virtual viewpoint is determined (Section 4.2). For efficiently determining the color of each pixel in an output frame, we could compute and store on disk (in the Preprocessing stage) the depth map corresponding to all the capture cameras. This speeds up the match computation. However in our current implementation, we chose to do a real-time computation of the depth map, rather than an explicit store (with a consequence of an increase in the final rendering time). For each pixel in the output frame, we chose to blend the color corresponding to the 4 samples in the z-buffer (of the animation camera) corresponding to it. This produces very realistic lighting and color blending effects. Figure 6 demonstrates screenshots of different animation sequences created with our algorithm on four objects, *Axe*, *Spaceship*, *MiniCooper*, and *Oak Leaf*. Please see supplementary material for generated animations and details.

Relighting: For our novel environments, we use the cube maps provided in [7], namely *The Uffizi Gallery, Florence*, *Glacier, Banff National Forest, Canada*, *Pisa courtyard, Italy*, and also generated a new environment, *woods*. As mentioned, using HDRShop lightgen plugin [7], we summarize the novel environment in terms of light sources. Figure 5 demonstrates the beautiful relighting effects faithfully reproduced on our *Axe* in the a fixed pose, but in different environments.

The entire run-time pipeline comprising of the continuous path (intermediate pose) generation, computing the view vector, shape determination, and color computation after relighting takes only around 1-2 minutes for a single output frame. Thus we are able to create the entire animation sequence within a few minutes.

6 FINAL REMARKS

In this paper, we have shown how to efficiently create realistic animations using only images as input, instead of traditional geometry-based input. We have employed four techniques to provide the alternative of animating an object on a specified arbitrary path in a novel environment. These techniques are, *B-Spline based interpolation* for the motion path specification, *classical coordinate transformations* for determining the correct pose of object, *visual hull* for view interpolation, and *lighting basis* for relighting.

Hull	Resolution	Initial Cloud	Before scoop-out	After scoop-out	Time(mins)
Axe	16	1.23×10^6	45463	18832	4
	32	9.6×10^6	344991	75321	10
	64	7.63×10^7	2794872	309013	31
Leaf	16	2.3×10^6	5045	5045	3
	32	1.8×10^7	39284	38037	4
	64	1.4×10^8	314928	222663	16
Mini	16	6×10^6	2143574	105098	37
	24	2×10^7	7220067	313382	120
Spaceship	16	7.4×10^6	901580	66329	40
	32	5.9×10^7	7213929	271692	120

Table 1: Quantitative results (size and preprocessing time) of our visual hull creation algorithm. The second column represents the sampling resolution of the initial point cloud, i.e., the number of points in a unit length; the third column depicts the total number of points in the initial cuboidal point cloud (Figure 3(a)); the fourth column indicates the number of points present (in the "solid" visual hull) in Algo.1 at step 10; the fifth column indicates the number of remaining points in the final visual hull after scooping the solid visual hull; the last column provides the total time taken in the studio to generate the visual hull, starting from the initial point cloud.

Hull	Resolution	Naive	Ours
Axe	64	1 hour	10 + 21 minutes
Mini	96	3 days	9 + 6 hours

Table 2: Preprocessing timing comparisons. **Naive** represents the time taken to create a visual hull beginning with a dense cuboidal point-cloud and using a classical algorithm. **Ours** represents by initially starting with a sparse point-cloud (Algorithm 1), and scooping out the hull interior. We report both times in the last column.

This animation framework could also be used as feedback for a final traditional CG animation. The director could quickly prepare a naive version of the final CG sequence by specifying a path, a novel lighting environment, a model; and later edit any of them to suit the mood of the story. Note that, not all three inputs need to be modified simultaneously at run time. For example, we might change the novel environment (see Figure 5), not the other two inputs. This would save man-hours spent by skilled personnel in the pre-production phase.

In future, research is required to incorporate non-rigid object motion into our animation framework. We also plan to incorporate special effects (for example, motion-blur, shadows) into the framework. Since IBR generates photorealistic outputs, the aesthetics of the models and the scene would be preserved.

REFERENCES

[1] N. Ahmed, C. Theobalt, C. Rossl, S. Thrun, and H.P. Seidel. Dense correspondence finding for parametrization-free animation reconstruction from video. In *CVPR '08*, pages 1–8, 2008.

[2] Christoph Bregler, Michele Covell, and Malcolm Slaney. Video rewrite: driving visual speech with audio. In *SIGGRAPH '97*, pages 353–360, 1997.

[3] Gabriel J. Brostow and Irfan Essa. Image-based motion blur for stop motion animation. In *SIGGRAPH '01*, pages 561–566. ACM, 2001.

[4] Joel Carranza, Christian Theobalt, Marcus A. Magnor, and Hans-Peter Seidel. Free-viewpoint video of human actors. *ACM Transactions on Graphics*, 22(3):569–577, 2003.

[5] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH '93*, pages 279–288, 1993.

[6] Biswarup Choudhury and Sharat Chandran. Data-intensive image based relighting. In *GRAPHITE '07*, pages 155–162. ACM, 2007.

[7] Paul Debevec. Light Probe Image Gallery. `http://www.debevec.org/Probes/`, 2008. Last visited: May, 2008.

[8] Paul Debevec, Andreas Wenger, Chris Tchou, Andrew Gardner, Jamie Waese, and Tim Hawkins. A lighting reproduction approach to live-action compositing. *ACM Transaction on Graphics*, 21(3):547–556, 2002.

[9] Per Einarsson, Charles F. Chabert, Andrew Jones, Wan C. Ma, Bruce Lamond, Tim Hawkins, Mark Bolas, Sebastian Sylwan, and Paul Debevec. Relighting human locomotion with flowed reflectance fields. In *EGSR '06*, pages 183–194, 2006.

[10] Martin Lillholm Erik Dam, Martin Koch. Quaternions, interpolation and animation. `www.itu.dk/people/erikdam/DOWNLOAD/98-5.pdf`, 2008. Last visited: May 2008.

[11] James Hays and Irfan Essa. Image and video based painterly animation. In *NPAR '04*, pages 113–120, 2004.

[12] Price Bibliography Keith. Keith Price Bibliography: Surface Reconstruction from Optical Flow. `http://www.visionbib.com/bibliography/optic-f753.html#KK4747`, 2008. Last visited: August, 2008.

[13] Price Bibliography Keith. Keith Price Bibliography: Virtual view generation, View synthesis, Image based rendering, IBR, Morphing. `http://www.visionbib.com/bibliography/describe487.html`, 2008. Last visited: August, 2008.

[14] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

[15] Marcus Magnor, Marc Pollefeys, German Cheung, Wojciech Matusik, and Christian Theobalt. Video-based rendering. In *SIGGRAPH '05 Courses*, page 1, 2005.

[16] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *SIGGRAPH '00*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., 2000.

[17] Wojciech Matusik, Hanspeter Pfister, Addy Ngan, Paul Beardsley, Remo Ziegler, and Leonard McMillan. Image-based 3d photography using opacity hulls. *ACM Transactions on Graphics*, 21(3):427–437, 2002.

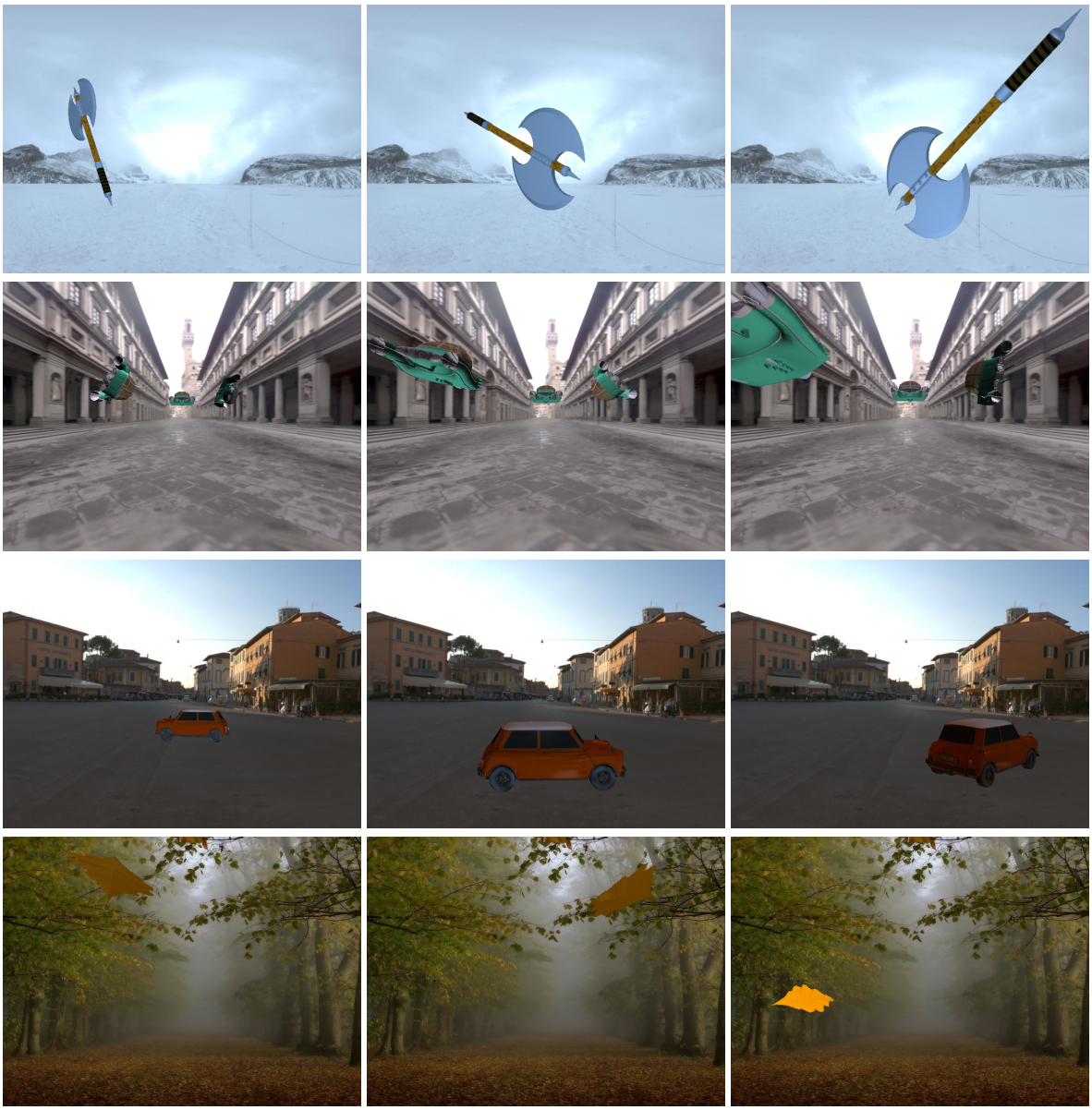


Figure 6: Images of Axe (row 1), Spaceship (row 2), MiniCooper (row 3) and Oak Leaf (row 4) rendered in different poses, i.e., while in motion, each under novel lighting conditions: Glacier, Uffizi, Pisa and Woods respectively.

[18] Leonard McMillan and Gary Bishop. Plenoptic modeling: an image-based rendering system. In *SIGGRAPH '95*, pages 39–46, 1995.

[19] Pixar. The Pixar Process. `http://www.pixar.com/howwedoit/index.html`, 2008. Last visited: May 2008.

[20] R. Plankers and P. Fua. Articulated soft objects for video-based body modeling. *ICCV '01*, 1:394–401 vol.1, 2001.

[21] Arno Schödl and Irfan A. Essa. Controlled animation of video sprites. In *SCA '02*, pages 121–127, 2002.

[22] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In *SIGGRAPH '00*, pages 489–498, 2000.

[23] Steven M. Seitz and Charles R. Dyer. View morphing. In *SIGGRAPH '96*, pages 21–30. ACM, 1996.

[24] J. Starck and A. Hilton. Virtual view synthesis of people from multiple view video sequences. *Graphical Models*, 67(6):600–620, 2005.

[25] J. Starck, G. Miller, and A. Hilton. Video-based character animation. In *SCA '05*, pages 49–58, 2005.

[26] Marco Tarini, Marco Callieri, Claudio Montani, Claudio Rocchini, Karin Olsson, and Therese Persson. "marching intersections: An efficient approach to shape-from-silhouette". In *Proceedings of the Conference on Vision, Modeling, and Visualization (VMV 2002)*, pages 255–262, 2002.

[27] Sundar Vedula, Simon Baker, and Takeo Kanade. Image-based spatio-temporal modeling and view interpolation of dynamic events. *ACM Transaction on Graphics*, 24(2):240–261, 2005.

[28] Xuemiao Xu, Liang Wan, Xiaopei Liu, Tien-Tsin Wong, Liansheng Wang, and Chi-Sing Leung. Animating animal motion from still. *ACM Transactions on Graphics*, 27(5):1–8, 2008.

[29] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM Transaction on Graphics*, 23(3):600–608, 2004.