Computational Light Painting and Kinetic Photography

Yaozhun Huang
City University of Hong Kong
Hong Kong, China
yaozhuang5-c@my.cityu.edu.hk

Sze-Chun Tsang
City University of Hong Kong
Hong Kong, China
szectsang2@cityu.edu.hk

Hei-Ting Tamar Wong
City University of Hong Kong
Hong Kong, China
httwong2-c@ad.cityu.edu.hk

Miu-Ling Lam*
City University of Hong Kong
Hong Kong, China
miu.lam@cityu.edu.hk

Figure 1: A gallery of unedited long exposure photographs taken with our computation light painting (Triceratops and Cow models) and kinetic photography (Mermaid and Humpback Whale models) systems. The pink curves illustrate the display or camera motion.

ABSTRACT

We present a computational framework for creating swept volume light painting and kinetic photography. Unlike conventional light painting technique using hand-held point light source or LED arrays, we move a flat-panel display with robot in a curved path. The display shows real-time rendered contours of a 3D object being sliced by the display plane along the path. All light contours are captured in a long exposure and constitute the virtual 3D object augmented in the real space. To ensure geometric accuracy, we use hand-eye calibration method to precisely obtain the transformation between the display and the robot. A path generation algorithm is developed to automatically yield the robot path that can best accommodate the 3D shape of the target model. To further avoid shape distortion due to asynchronization between the display's pose and the image content, we propose a real-time slicing method for arbitrary slicing direction. By organizing the triangular mesh into Octree data structure, the approach can significantly reduce the computational time and improve the performance of real-time rendering. We study the optimal tree level for different ranges of triangle numbers so as to attain competitive computational time. Texture mapping is also implemented to produce colored light painting. We extend our methodologies to computational kinetic photography, which is dual to light painting. Instead of keeping the camera stationary, we move the camera with robot and capture long exposures of a stationary display showing light contours. We transform the display path for light painting to the camera path for kinetic photography. A variety of 3D models are used to verify that the proposed techniques can produce stunning long exposures with high-fidelity volumetric imagery. The techniques have great potential for innovative applications including animation, visible light communication, invisible information visualization and creative art.

*Corresponding author

Expressive '18, August 17–19, 2018, Victoria, BC, Canada

ACM ISBN 978-1-4503-5892-7/18/08.
https://doi.org/10.1145/3229147.3229167

CCS CONCEPTS

• **Computing methodologies** → **Computational photography**; Motion path planning; Vision for robotics; • **Human-centered computing** → *Visualization systems and tools*;

KEYWORDS

computational light painting, real-time rendering, model slicing

ACM Reference Format:
Yaozhun Huang, Sze-Chun Tsang, Hei-Ting Tamar Wong, and Miu-Ling Lam. 2018. Computational Light Painting and Kinetic Photography. In *Expressive '18: The Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering, August 17–19, 2018, Victoria, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3229147.3229167

1 INTRODUCTION

Light painting is created by moving a light source in the space while being captured by long exposure. The technique has been used for over a century for artistic and scientific purposes. In 1889, physiologists and chronophotographers Étienne-Jules Marey and Georges Demeny created the first light painting when studying the movements of humans. They attached light bulbs at the joints of a human and took long exposure photographs when the person was walking. Light painting technique has been used by many famous artists and photographers including Man Ray, Pablo Picasso, Henri Matisse, and Gjon Mili. This aesthetic art form has engaged many professional photographers and hobbyists in the past and is still very popular now. Scientists and engineers have also used light painting technique to visualize various forms of signals, such as invisible light and radio frequency, by varying the brightness or color of the light at different locations.

A similar long exposure technique called kinetic photography can also create artistic light patterns. Rather than moving the light source, a camera is moved or thrown into the air (camera tossing) while its shutter opens. Both light painting and kinetic photography techniques involve the relative movement between the camera and the light source. The configuration of the two techniques are dual to each other. Sometimes kinetic photography is also called light painting.

In conventional light painting, the light source is usually point light or other simple handheld light made by attaching an array of LEDs to a stick or a ring. Whereas in conventional kinetic photography, the camera motion is uncontrolled. The images created by these approaches are limited to freehand strokes, abstract shapes and blurred light patterns.

The goal of this work is to provide a framework for creating long exposure photographs with high fidelity and representational 3D imagery while keeping the whole process highly automatic and accurate.

We present computational light painting and computational kinetic photography in which the intensity, color or shape of the light source is precisely controlled based on the instantaneous relative position between the camera and the light source during a long exposure. We use a flat-panel display as the light source in order to deploy high-resolution pixels in the space efficiently in a controlled manner. Similar to volumetric 3D display, when a flat display moves, it sweeps out a volume populated with pixels that can be captured by a camera with slow shutter speed. The volume can be used to display various kinds of data such as 3D models. One of the key challenges is how to ensure the shape fidelity of the light-painted 3D patterns.

We use a 6-axis robot arm to move the display. Apart from efficiency, the advantage of using robot is that the real-time pose of the display can be precisely controlled and acquired. No pose estimation or motion tracking is needed.

We develop a set of methodologies to tackle the challenges in the robotic light painting process. We further apply and extend the methodologies to accomplish computational kinetic photography.

The primary contributions of this work include:

- We calibrate the relative pose of the display and camera based on robotic hand-eye calibration.
- We develop an automatic path generation algorithm to yield the camera or display paths that can best accommodate the 3D shape of the target model.
- We propose a real-time, Octree-based model slicing algorithm that is applicable for arbitrary directions and can significantly improve rendering time.
- We apply texture mapping to create colored and textured light painting.
- We design a pipeline to convert our calibration, path generation, and contour rendering methods to solve the computational kinetic photography problem.
- We conduct experiments with a variety of 3D models to verify our methodologies.

To the best of our knowledge, this work is the first to propose swept volume kinetic photography. Kinetic photography has received relatively less attention in research domain, but it has immense potentials in emerging media technologies and commercial applications. For example, a moving car with a camera can capture the lights from a large screen installed on a highway or at the entrance of a tunnel. 3D information and imagery can be automatically and efficiently presented to the driver without the hassle of establishing wireless communication. Other application scenarios exist which suggest that computational kinetic photography is useful for visible light communication.

Light painting and kinetic photography have great potential for innovative applications such as visible light communication, invisible information visualization, animation, and creative art. However, this area remains largely unexplored. We are inspired to use computational methods to leverage the application range of these interesting photography techniques and to light on this untapped territory of research.

2 RELATED WORK

We provide an overview of existing works using light painting technique, as well as works in other related areas including hand-eye calibration and model slicing.

2.1 Light painting

Light painting has been widely used in sciences, commercial photography, entertainment, and creative arts. Conventionally, the

light source is moved by hand along some specific paths to create an immaterial structure, which is recorded in 2D images by camera long exposure [Acar and Kavuran 2016; Arnall et al. 2013; Wada et al. 2016]. Reed and Clemens [Reed and Clemens 2010] used a light source to write some characters in mid-air, in which the strokes are in pure hand-writing style. Some mobile applications are developed to realize swept volume light painting [Anzalichi [n. d.]]. However, since the light patterns do not synchronize with the device's pose, user's jiggly motions will result in distortion in the photos. Sturgeon and Ray [Sturgeon and Ray 2015] used light painting technique to visualize electromagnetic fields surrounding physical objects.

Computational light painting refers to the technique using long-exposure to record the light source which is precisely synchronized and computed at every time instant. Mann et al. [Mann et al. 2014] proposed an interactive wearable computational photography device which allows the wearer to see light painted virtual objects augmented in the real space. While this work used random moving path, the light source just generated simple shapes that contain limited information. Salamon, Lancelle, and Eisemann [Salamon et al. 2017] used virtual exposure to create computational light painting from video frames. However, the resultant light pattens are limited to single color strokes.

Robots have a wide variety of applications including photography. Kruysman and Proto [Kruysman and Proto 2013] used a robot arm to mount a display and draw volumetric light, but the robot path is limited to linear. Robotic immaterial fabrication is proposed by Keating and Oxman [Keating and Oxman 2013] using robotic arm to control light source for generating light objects. However, the display contents in some works are not real-time rendered and synchronized with robot motion [Keating and Oxman 2013; Kruysman and Proto 2013]. Besides, no path planning is taken into consideration. Crossman [Crossman 2014] mounted a RGB LED at the end effector of an industrial robot to draw volumetric, colored model. While using a point light source, the process is time consuming. Recently, we have introduced a computational light painting system [Huang et al. 2016] that swepts a flat-panel display showing successive contours of 3D model using a robot arm moving along some curved paths, resulting in three-dimensional high-fidelity light painting photographs. Despite that the results show convincing volumetric imagery, the frames displayed were not generated in real-time, thus causing shape distortion. Moreover, the motion paths of robot arm were manually created. The goal of this paper is to address the above challenges.

2.2 Calibration

For long exposure photography in the context of creating non-linear relative motion between the light source and the camera, the position and orientation of the light source must be under supervised control to keep the resultant light painting undistorted. Thus, a calibration process is necessary to obtain the geometric relationship between the display and the robot arm, when the former is rigidly mounted to the latter. Hand-eye calibration, as commonly used in robotic applications like visual servoing, is the technique for determining the relative transformation between a camera and the end-effector of robot where the sensor is attached to. Solving the transformation can allow the robot arm to act in the camera coordinate system in order to grip a target that the camera is looking at. In addition, placing the camera to the exact position and orientation with respect to the reference frame is essential for application likes 3D scene reconstruction. Such transformation is difficult to be measured by hand because the origin of the involved coordinate frames are most likely to be located inside the physical chassis of the devices. The hand-eye calibration problem is commonly described as a homogeneous transform equation of the form $\boldsymbol{AX} = \boldsymbol{XB}$, which can be solved by using the method of a two-stage solution [Cheung and Ahmad 1989; Tsai and Lenz 1989] for doing tool-to-end effector calibration based on the geometric interpretations of the eigenvalues and eigenvectors of a rotation matrix. Some other later works also follow the two-stage approach [Wang 1992; Zhuang et al. 1994]. Since the rotation component of the equation is decoupled from the translational one, such approach suffers from error propagation problem in which the errors in solving the rotation component will propagate to the translation component. These methods require at least two motions (three poses) with distinct rotation axes to yield a unique solution. Horaud and Dornaika [Horaud and Dornaika 1995] applied a simultaneous non-linear minimization for solving rotation and translation of the unknown transformation $\boldsymbol{X}$. The method overcomes the propagation problem but it requires a proper initial guess to achieve global minimum.

2.3 Model slicing

Another procedure necessary in our light painting process is to compute the display content in real-time and synchronize it with the robot arm motion. In order to create light patterns representing the surface of a 3D model in the long exposure, what is to be shown at every time instant should be the contour of the object obtained by slicing the model using the display surface as the slicing plane.

A 3D model is composed of unordered triangles. In order to obtain a 2D contour of a model sliced by an arbitrary plane, one could exhaustively check every single triangle to see if the slicing plane intersects with it and collect all the intersection segments [Chalasani et al. 1991]. In order to improve the computational efficiency, Tata et al. and Vatani et al. [Tata et al. 1998; Vatani et al. 2009] proposed some pre-processes to group all the triangles along z-direction. The approach is not applicable for arbitrary slicing directions as re-grouping is needed every time when the slicing direction changes, which makes the computational efficiency even lower than the traversal approach. Other researchers proposed alternative approach utilizing the topology information [Huang et al. 2002; Rock and Wozny 1991]. However, these methods still only relied on fixed slicing direction and it is time-consuming to find the first intersection point between the model and the slicing plane. Despite the fact that slicing algorithm has been widely explored, previous works always focused on slicing in one direction and the process is offline.

3 METHODOLOGIES

3.1 Calibration

Computational light painting requires a precise mapping between the display content and the robot arm position, which is based on the prerequisite that the rotation and translation between display

and robot arm is known. As our light painting deals with curved motion path, inaccurate pose of the display will cause distortion in the resultant long exposure photograph.

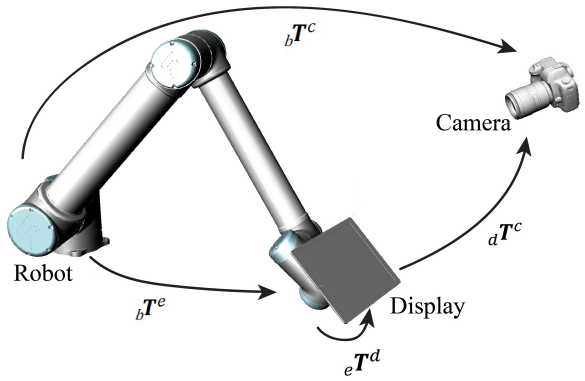


Figure 2: System calibration.

3.1.1 Preliminaries. We set the world coordinate frame at the base of robot arm. ${}_b\boldsymbol{T}^e$ denotes the homogeneous transformation of the end effector frame with respect to the world coordinate frame (i.e. robot pose), ${}_e\boldsymbol{T}^d$ denotes the homogeneous transformation of the display with respect to the end effector, and ${}_d\boldsymbol{T}^c$ denotes the homogeneous transformation of the camera with respect to the display. The geometric relationship of the computational light painting system is shown in Figure 2.

3.1.2 Hand-eye Calibration. The goal of calibration step is to solve the unknown transformation between the robot and the display mounted onto it (i.e. ${}_e\boldsymbol{T}^d$). First, a chessboard pattern is shown on the display to conduct camera calibration [Zhang 2000]. During the process, the display is moved to multiple arbitrary positions while the chessboard pattern is captured by a camera placed at a fixed position. The transformation from the camera to the display (chessboard) ${}_d\boldsymbol{T}^c$ is then obtained from the extrinsic parameters of the camera matrix. On the other hand, the robot pose ${}_b\boldsymbol{T}^e$ is also known by the robot controller. Let n be the number of robot poses during the camera calibration process. Each robot arm's position is recorded and paired with ${}_d\boldsymbol{T}^c_i$ for $i = 1$ to n. Thus, the only missing link in the transformation from robot base to camera is ${}_e\boldsymbol{T}^d$, which is constant throughout the whole calibration process since the display is rigidly attached to the robot arm. Moreover, since the camera pose ${}_b\boldsymbol{T}^c$ is fixed throughout the process, we have

$$ {}_b\boldsymbol{T}^c = {}_b\boldsymbol{T}^e_i \, {}_e\boldsymbol{T}^d \, {}_d\boldsymbol{T}^c_i = {}_b\boldsymbol{T}^e_j \, {}_e\boldsymbol{T}^d \, {}_d\boldsymbol{T}^c_j \quad \forall i, j \tag{1} $$

Our method to solve ${}_e\boldsymbol{T}^d$ is inspired by [Zuang and Shiu 1993]. We estimate the matrix using a nonlinear minimization technique, which can provide the quality and the confidence of the solution by analysing the depth and the width of the global minimum respectively. There are 9 unknown parameters associated with the rotational component and 3 with the translational component of ${}_e\boldsymbol{T}^d$. In fact, the rotation matrix can be expressed as 3 unknown rotation angles along x, y and z axis, representing 3 degree of freedom, of end effector coordinate frame. The direct manipulation of these 3 variables guarantees the orthogonality of the resultant rotation matrix. A set of $\sum_{i=1}^{n-1} i$ equations forms an overdetermined system. The result acquired from the closed-form, non-iterative solution [Cheung and Ahmad 1989] is used as the initial guess for Levenberg-Marquardt least square minimization.

$$ \min_x \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \| {}_b\boldsymbol{T}^e_i \, {}_e\boldsymbol{T}^d(x) \, {}_d\boldsymbol{T}^c_i - {}_b\boldsymbol{T}^e_j \, {}_e\boldsymbol{T}^d(x) \, {}_d\boldsymbol{T}^c_j \|^2 : x \in \mathbb{R}^6 \right\} \tag{2} $$

For computational kinetic photography where the camera instead of the display is mounted to the robot, we used the same calibration method to yield the transformation between the camera and the robot end effector.

3.2 Automatic Path Planning

In order to efficiently utilize the volume swept by the moving display, we develop an algorithm to automatically yield the robot path based on the geometry of the target 3D model. The idea is to find the longest path in the skeleton of the model that matches its shape tendency, and then smooth the path by curve fitting method.

3.2.1 Preliminaries. Various skeletonization methods can be used. Without loss of generality, a mesh-based curve skeleton extraction algorithm is used to obtain the skeleton of given 3D model [Tagliasacchi et al. 2012]. The process formulates the skeletonization problem via mean curvature flow. Figures 3(a) and 4(a) show the raw skeletons of a dragon model and a dolphin model obtained by the mean curvature flow method. Despite the raw skeleton $\boldsymbol{S}$ implies the structural information of the 3D model, however, it cannot be directly used as the robot path which is required to be smooth and robust. Our next target is to prune the unwanted branches from $\boldsymbol{S}$ while keeping the major backbone, which likely covers most parts of the model for light painting. The skeleton $\boldsymbol{S}$ is originally formed as a point set $\boldsymbol{P}$, in which p_e denotes an extremity point with only one adjacent point; p_l denotes a link point with two adjacent points; and p_j denotes a joint point with more than two adjacent points. The skeleton point set $\boldsymbol{P}$ is further grouped as a branch set $\boldsymbol{B}$, in which each branch is composed of a point sequence that includes p_j or p_e at both ends of the sequence and all p_l in between. An extremity branch is defined as a branch with at least one p_e at its both ends. The weight of a branch is recorded as the total length of it.

3.2.2 Longest Path Extraction. We turn the backbone extraction problem to the problem of finding the longest path in the skeleton and by treating the branch set $\boldsymbol{B}$ as a graph, Breadth-first Search (BFS) algorithm is used to realize it. The search starts from any vertex in the graph. The search conquers the unvisited neighbors of the current vertex at the same depth level before visiting any other vertices at the next level. The path with heaviest weight is recorded at each iteration until all vertices in graph are visited. Then, a second round of BFS is executed again starting from the extremity point in path with the heaviest weight in the first round. The result of this step is an unbranched, longest path in the object model skeleton $\boldsymbol{S}$. Figures 3(b) and 4(b) show the longest path of given models.

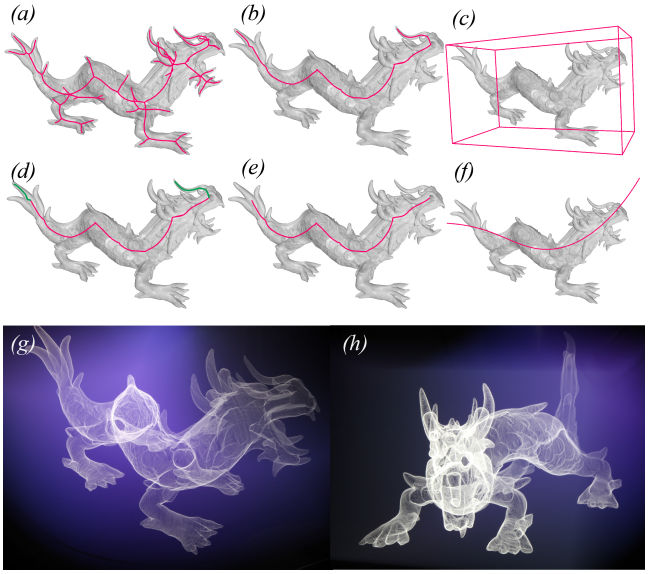


Figure 3: (a) The skeleton of a dragon model. (b) The longest path in the skeleton. (c) The OBB of the model. (d) The green branches are removed as they violate the major axis orientation of OBB. (e) The backbone obtained. (f) The final path automatically generated by polynomial curve fitting. (g) The light painting result with path generated by proposed path planning algorithm.

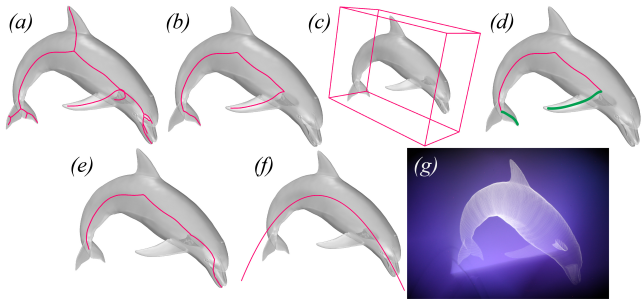


Figure 4: (a) The skeleton of a dolphin model. (b) The longest path in the skeleton. (c) The OBB of the model. (d) The green branches are removed as they violate the major axis orientation of the OBB. (e) The revised backbone by adding additional branches in the skeleton aligning with the OBB. (f) The final path automatically generated by polynomial curve fitting. (g) The light painting result with path generated by the proposed path planning algorithm.

3.2.3 OBB Refining. Despite in some cases the longest path obtained can already represent a reasonable reference curve, further pruning and planting are necessary for some models. The ultimate goal is to generate the most representative path of a 3D structure, which should follow the tendency of 3D object model. Oriented Bounding Box (OBB) is ultilized to define the tendency of object model. The covariance matrix of the model, which is a generalization of variance to higher dimension, is used to fit the OBB. Based on the algorithm by Gottschalk, Lin, and Manocha [Gottschalk et al. 1996], the eigenvectors of the covariance matrix determine the rotation of the fitting box with respect to the world coordinate frame. Given the 3D coordinates of the vertices of the target object, the mean position in 3D, denoted as $\hat{x}$, $\hat{y}$, $\hat{z}$, and covariance matrix, denoted as $\boldsymbol{C}$, are computed as follow:

$$\boldsymbol{C} = \begin{bmatrix} E[xx] - \hat{x}\hat{x} & E[xy] - \hat{x}\hat{y} & E[xz] - \hat{x}\hat{z} \\ E[yx] - \hat{y}\hat{x} & E[yy] - \hat{y}\hat{y} & E[yz] - \hat{y}\hat{z} \\ E[zx] - \hat{z}\hat{x} & E[zy] - \hat{z}\hat{y} & E[zz] - \hat{z}\hat{z} \end{bmatrix}$$
$$\hat{x} = \frac{1}{N}\sum_{i=1}^{N} x_i, \hat{y} = \frac{1}{N}\sum_{i=1}^{N} y_i, \hat{z} = \frac{1}{N}\sum_{i=1}^{N} z_i \quad (3)$$

where N is the number of vertices, $E[a]$ represents the expected value of argument a. Then, the eigenvectors are yielded from the covariance matrix. The transformation matrix of the OBB is determined by the position and the eigenvectors, and finally scaling the bounding box of the input model. Figures 3(c) and 4(c) show the OBB of the given models.

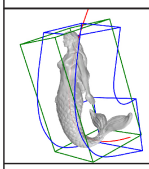
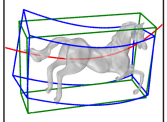
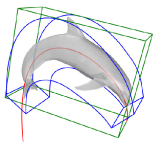
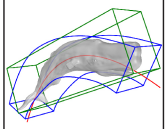
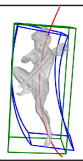
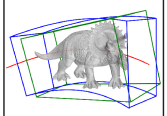
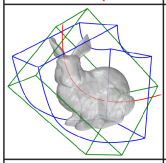
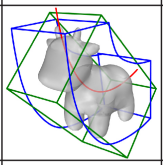
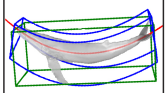
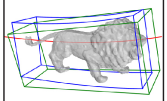
The same OBB computation process is also applied to every branch along the longest path, in which the 3D coordinates of the point set of the skeleton are used to build the covariance matrix. The branch with more similar direction with the OBB will be given higher priority. By comparing the OBB of each branch and the OBB of the entire model, the branch with very different directions with the OBB of the model will be pruned (green branches in Figures 3(d) and 4(d)). Then, we will get a shortened path of the model.

However, since the target path is required to cover the entire 3D model, in some cases, it is hard to be realized by a shortened path obtained from the former process as shown in Figure 4(d). Each unused branch connecting to either end nodes of the current path is sorted out, and added to the path if its OBB aligns well with the model OBB, thus extending the path as shown in Figure 4(e). The resultant path spans the entire object model and is defined as the final backbone of the model. Figures 3(e) and 4(e) show the backbone of given models. In some cases, no new branch is added to the backbone, such as the dragon model as shown in Figure 3(d) and (e).

3.2.4 Path Smoothen and Extension. The backbone is a subset of $\boldsymbol{B}$ of the given model, but sudden inflection point along the backbone makes the path unsafe for robot motion. On the other hand, the extracted backbone is completely inside the model body, further extension is needed to ensure the swept path covers the entire model. We use polynomial curve fitting to smoothen and extend the backbone. We treat all skeleton points on the backbone as the input control points, a smooth curve can be easily obtained. The resultant curve is a desirable path to be applied for robot motion. Figures 3(f) and 4(f) show the final light-painting paths automatically generated for the given models.

3.2.5 Evaluation. In addition to simplifying the path creation process for light painting, the automatically generated curved path shows higher space efficiency than a linear motion path. It means that with the same display area, light painting with a curved path can get a larger swept volume compared to linear path. In other

Table 1: Comparison of efficiency between curve path and OBB

Model	Area Ratio γ	Model	Area Ratio γ
	0.536		0.683
	0.591		0.742
	0.607		0.808
	0.609		0.847
	0.643		0.863

words, to obtain a fixed workspace, the display size required by a curved path is always smaller than that required by a linear path. To quantitatively evaluate the efficiency of the curved paths automatically generated by our method, we compute the ratio of the minimum display size required to cover various 3D models using the curved paths obtained and linear paths. The direction of linear path for each model is defined as the major axis direction of its OBB. Given a model, the ratio γ is defined as the minimum display area needed to cover the entire model when using curved path (A_c) versus the minimum display area needed when using linear path (A_{OBB}), which is $\gamma = \frac{A_c}{A_{OBB}}$. Table 1 shows the ratios γ for different models.

3.3 Real-time rendering and model slicing

During light painting, in order to guarantee the model is fully covered, the designed robot path is always longer than the backbone. Pre-rendered video or image stacks may work for linear robotic paths with constant speeds as proposed by most previous works. However, for curved paths which yield better efficiency, synchronization between the robot arm pose and the video content is critical for avoiding distortion of the resultant light painted object, which is composed of many closely stacked layers of contours in different orientations. Thus, an efficient slicing algorithm for obtaining the contour of a 3D mesh at arbitrary orientation is necessary. In this subsection, we describe an Octree-based model slicing algorithm used for generating the light contours in real-time.

3.3.1 Real-time Slicing of Un-textured Model. We adopt the slicing algorithm that we previously proposed [Wong et al. 2017]. The Axis Aligned Bounding Box (AABB) based Octree data structure is built to represent the hierarchical mesh information of the model. We determine the optimal tree level based on empirical approach in order to achieve the best computational efficiency. The Octree does not need to be rebuilt when the model is sliced. Furthermore, slicing of Octree-structured model can be implemented by parallel computing to further improve efficiency.

During real-time slicing, the position and orientation of the slicing plane are computed based on the robot pose streamed from the robot controller and the calibrated end effector-display transformation ${}_e\boldsymbol{T}^d$. The slicing plane will access the Octree from the root to leaf node recursively. All triangles intersecting with the slicing plane are extracted and line segments constructing the contour(s) are collected. The query will stop at the node that does not intersect with the slicing plane, which can efficiently reduce the number of triangles to be processed so as to speed up the rendering.

Table 2 compares the computational time using Octree-based and traversal (naive) slicing methods. The Octree level for each model was selected based on its triangle numbers [Wong et al. 2017]. For each model. 100 slicing planes were randomly sampled. The time involved to extract the intersection lines between the model and each slicing plane using both methods are recorded. We have performed the test on over 30 models and Octree method always yield a shorter computational time than the traversal method. Selected examples are listed in Table 2.

Table 2: Comparison of efficiency between Octree method and traversal method

Number of triangles	Octree level	Average slicing time using Octree (ms)	Average slicing time using traversal method (ms)
2424	0	1.14	1.27
16762	2	2.35	3.62
47730	3	3.28	9.91
99994	3	5.94	18.04
213504	3	11.21	37.27
220174	3	10.74	35.19
527916	4	21.48	84.90
1710982	4	76.96	277.25
2565012	4	128.09	425.50

3.3.2 Texture Mapping. We further extend our rendering technique to produce colored contours based on the model texture. The goal in this process is to color every position along the intersection line segments obtained in the slicing process with the corresponding color of the given model using the triangle vertices and the corresponding texture uv coordinates. As shown in Figure 5(a), given a model and a slicing plane, the corresponding line segment is shown in red and the intersection points between the triangles on of the mesh and the slicing plane are shown as yellow dots. The

sub-image in Figure 5(a) is the texture map of the model. Figure 5(a) and (b) show the geometric relationship of vertices of a green triangle in 3D space and the 2D texture map respectively. V_1', V_2', V_3' denote the 3D coordinates of the triangle vertices; M_a', M_b' denote the 3D coordinates of the intersection points; V_1, V_2, V_3 denote the uv coordinates of the triangle vertices; and M_a, M_b denote the uv coordinates of the intersection points. M_a and M_b can be easily obtained by linear interpolation:

$$M_a = V_2 + (V_1 - V_2)\left|\frac{M_a' - V_2'}{V_1' - V_2'}\right| \qquad M_b = V_2 + (V_3 - V_2)\left|\frac{M_b' - V_2'}{V_3' - V_2'}\right| \tag{4}$$

After finding the uv coordinates of M_a and M_b, the color at every position along the intersection line segment can be directly retrieved from texture pixels along M_a and M_b. When the process is applied to every line segment, a colored contour in 3D space will be produced as shown in Figure 6.

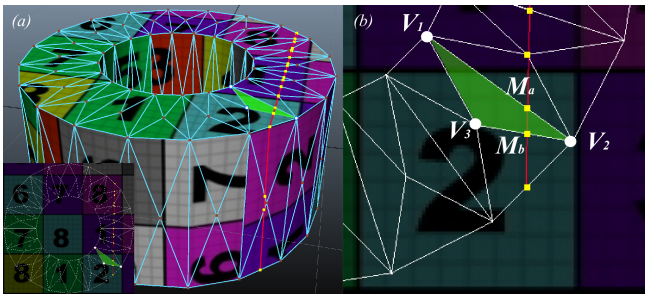


Figure 5: Texture mapping of intersection line segment.

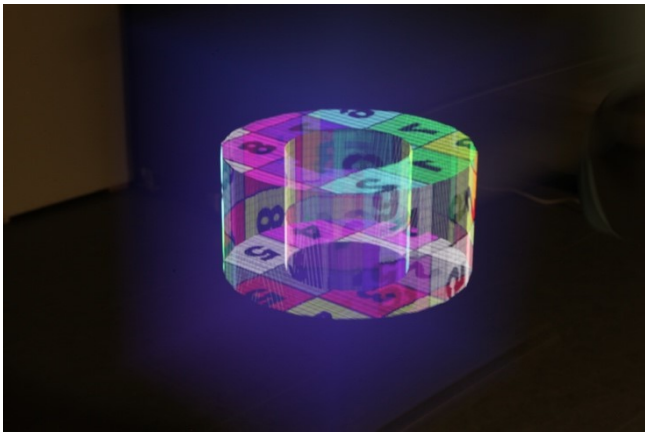


Figure 6: Computational light painting of textured model.

3.4 Computational Kinetic Photography

We extend our computational light painting framework to computational kinetic photography in which the camera is mounted to and moved by the robot while the display remains stationary. One advantage is that a large display or projection can be used because the camera, not the display, is set in motion. In conventional kinetic photography, the light patterns in the resultant long exposure is abstract while the background is blurry. The goal of our computational kinetic photography framework, as shown in the long exposures of Mermaid and Humpback Whale in Figure 1, is to produce clear and representational 3D light patterns by tightly synchronizing the display content with the camera motion at every time instant when the camera shutter opens.

All our methodologies developed for computational light painting can be applied to computational kinetic photography. However, additional steps are needed. The major difference between the two modes is that, in computational light painting, the camera position is arbitrary and can be changed any time as long as the display is within the field of view of the camera; while in computational kinetic photography, the relative position and orientation between the camera and the display must be pre-determined in order to define the camera motion. We swap the positions of camera and display, and use the same calibration method described in Section 3.1 to obtain the transformation between the camera and end effector . After calibration, the display pose with respect to the robot arm is also determined. Thus, we can use the information to produce the camera path.

4 EXPERIMENTS

We implement the proposed methods and conduct a set of experiments to verify our approach. The model-based robotic path generation algorithm is implemented in C++. We use a 6-axis industrial robot arm UR10 by Universal Robots to mount a flat-panel display (iPad Pro) for light painting or a camera for kinetic photography. A Canon Mark II 5D camera is used for taking long exposures. A PC with NVIDIA GeForce GTX 780 GPU and 2.6GHz CPU is used as for all computation including real-time rendering. The robot script (UR script) for controlling the robot is generated by Rhino plugin by HAL [Schwartz 2011]. The data transmission between the computer and the robot controller is established via local area network (LAN). When the robot is moving, the controller streams the robot pose to the PC through LAN at 60 frames per second. Another C++ program running on PC is developed for real-time control, which receives robot pose and implement the real-time slicing algorithm to render the light contours.

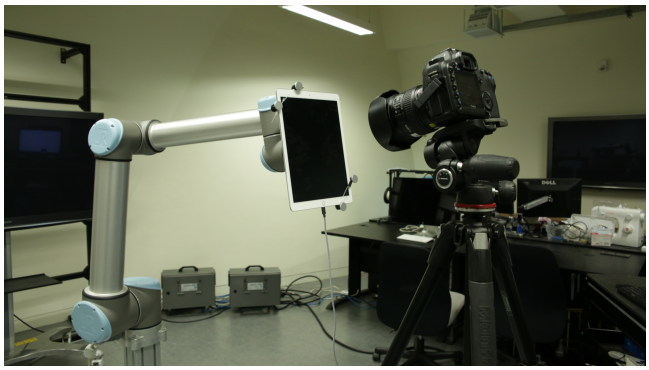


Figure 7: Computational light painting system.

4.1 Computational Light Painting Results

Figure 7 shows the set-up of our computational light painting system. A 12.9-inch iPad pro of resolution 2732-by-2048 is used as the display. The rendering is computed by the PC. The display is mounted on the robot and their relative transformation is obtained by calibration. As the display is moved by the robot, it creates a stack of contours in the space representing the 3D shape of object model. Figure 8 shows the resultant photograph by our computational light painting technique of a Horse model. The physical dimension of the light painted Horse was $95 \times 158 \times 207mm$ and the shutter time was 43 seconds. Computational light painting allows different camera angles as shown in Figure 9. The physical size of the light painted Tyrannosaurus was $502 \times 99 \times 177mm$ and the exposure time of both photos was 52 seconds.

Figure 8: Rendered image vs computational light painting photograph of a Horse model.

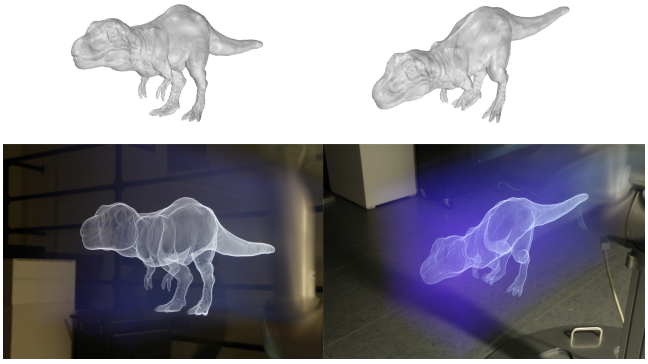


Figure 9: Rendered images vs computational light painting photographs of a Tyrannosaurus model captured from different viewing angles.

4.2 Computational Kinetic Photography Results

Figure 10 shows the set-up of our computational light painting system. We mount the camera on the robot and fixed a 50-inch TV of resolution 1920-by-1080. The pose of the TV as well as the transformation from the robot end effector to the camera are obtained by calibration. By setting the virtual camera in the dual light painting scene at different positions during the path conversion process, it is also possible to shoot the model from different viewing angles. Figure 11 shows the kinetic photography results of the same Tyrannosaurus model of $1254 \times 247 \times 444mm$ physical dimension from various perspectives. The camera exposure time was 27 seconds and 30 seconds respectively. The light sculptures of Mermaid and Humpback Whale in Figure 1 were also created by our computational kinetic photography system.

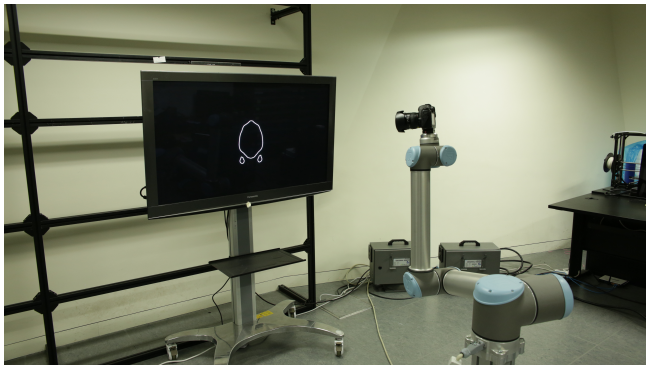


Figure 10: Computational kinetic photography system.

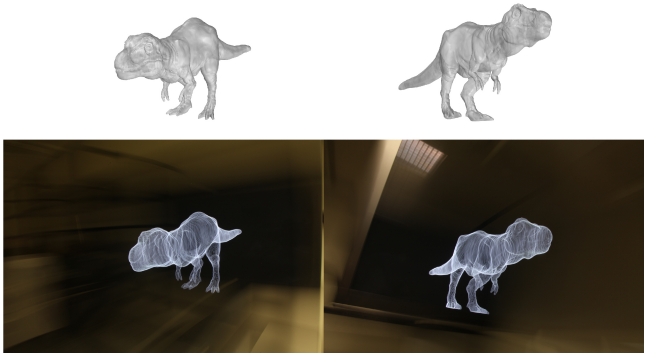


Figure 11: Rendered images vs computational kinetic photographs of the Tyrannosaurus model at different viewing angles.

5 CONCLUSIONS AND DISCUSSION

This work presented a novel approach to realize computational light painting and kinetic photography. To ensure shape fidelity, we used robot to precisely control and acquire the real-time pose of the display. In addition, we extended hand-eye calibration to acquire the geometric relationship among the system components, including display, robot and camera. The proposed method further automatically generated a curved motion path for the robot to accommodate the model shape and optimize the efficiency. We developed a Octree-based slicing algorithm to generate the display content in real-time under the challenging condition of arbitrary slicing direction. Most importantly, we provided a framework to realize computational, swept volume kinetic photography based on

the developed methodologies. To the best of our knowledge, this is the first work using kinetic photography to capture representational 3D imagery.

Our future work includes improving the automatic path generation method by taking more factors, such as the camera's field of view, models with complex topology and the robot's kinematic constraints, into consideration. The rendering method will also be enhanced to produce more realistic 3D imageries by removing the occluded parts of the models. We will also use the proposed methods for other applications, including animation, visible light communication, and augmented reality.

ACKNOWLEDGMENTS

This work was partially supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (CityU 11206216), National Natural Science Foundation of China (61502406), Croucher Foundation (9500016), CityU (7005016 & 7200452) and ACIM-SCM.

REFERENCES

Haci Mehmet Acar and Tamer Kavuran. 2016. Photo art creativity in the education: Light drawing. In *SHS Web of Conferences*, Vol. 26. EDP Sciences. https://doi.org/10.1051/shsconf/20162601083

Navid Anzalichi. [n. d.]. Dr. Light Painting. ([n. d.]). https://play.google.com/store/apps/details?id=anzalichi.light

Timo Arnall, Jørn Knutsen, and Einar Sneve Martinussen. 2013. Immaterials: light painting WiFi. *Significance* 10, 4 (2013), 38–39. https://doi.org/10.1111/j.1740-9713.2013.00683.x

K.L. Chalasani, B.N. Grogan, A. Bagchi, C.C. Jara-Almonte, A.A. Ogale, and R.L. Dooley. 1991. An algorithm to slice 3D shapes for reconstruction in prototyping systems. In *Proceedings of the 1991 ASME Computers in Engineering Conference*. 209–216.

Shiu Yiu Cheung and Shaheen Ahmad. 1989. Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form AX= XB. *IEEE Transactions on robotics and automation* 5, 1 (1989), 16–29. https://doi.org/10.1109/70.88014

Jeff Crossman. 2014. Industrial Light Painting. (2014). https://www.jeffcrossman.com/industriallightpainting

Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. 1996. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 171–180. https://doi.org/10.1145/237170.237244

Radu Horaud and Fadi Dornaika. 1995. Hand-eye calibration. *The international journal of robotics research* 14, 3 (1995), 195–210.

Syu-Hong Huang, Li-Chao Zhang, and M Han. 2002. An effective error-tolerance slicing algorithm for STL files. *The International Journal of Advanced Manufacturing Technology* 20, 5 (2002), 363–367. https://doi.org/10.1007/s001700200164

Yaozhun Huang, Sze-Chun Tsang, and Miu-Ling Lam. 2016. Computational swept volume light painting via robotic non-linear motion. In *ACM SIGGRAPH 2016 Posters*. ACM, 27. https://doi.org/10.1145/2945078.2945105

Steven Keating and Neri Oxman. 2013. Robotic immaterial fabrication. In *Rob| Arch 2012*. Springer, 256–266. https://doi.org/10.1007/978-3-7091-1465-0_30

Brandon Kruysman and Jonathan Proto. 2013. Tomography. (2013). http://www.kruysmanproto.com/

Steve Mann, Ryan Janzen, Tao Ai, Seyed Nima Yasrebi, Jad Kawwa, and Mir Adnan Ali. 2014. Toposculpting: Computational lightpainting and wearable computational photography for abakographic user interfaces. In *Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on*. IEEE, 1–10. https://doi.org/10.1109/CCECE.2014.6901118

Kram Reed and Weisshaar Clemens. 2010. Outrace. (2010). http://www.outrace.org/

Stephen J Rock and Michael J Wozny. 1991. Utilizing topological information to increase scan vector generation efficiency. In *Proceedings of Solid Freeform Fabrication Symposium, Austin, TX, Aug*. 3–5.

Nestor Z. Salamon, Marcel Lancelle, and Elmar Eisemann. 2017. Computational Light Painting Using a Virtual Exposure. *Computer Graphics Forum* (2017). https://doi.org/10.1111/cgf.13101

Thibault Schwartz. 2011. HAL. (2011). http://www.food4rhino.com/app/hal-robot-programming-control?ufh=

Luke Sturgeon and Shamik Ray. 2015. Visualising Electromagnetic Fields: An Approach to Visual Data Representation and the Discussion of Invisible Phenomena. In *In Proceedings of the 3th. Conference on Computation, Communication, Aesthetics and X*. ACM, 100–110.

Andrea Tagliasacchi, Ibraheem Alhashim, Matt Olson, and Hao Zhang. 2012. Mean curvature skeletons. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1735–1744. https://doi.org/10.1111/j.1467-8659.2012.03178.x

Kamesh Tata, Georges Fadel, Amit Bagchi, and Nadim Aziz. 1998. Efficient slicing for layered manufacturing. *Rapid Prototyping Journal* 4, 4 (1998), 151–167. https://doi.org/10.1108/13552549810239003

Roger Y Tsai and Reimar K Lenz. 1989. A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. *IEEE Transactions on robotics and automation* 5, 3 (1989), 345–358. https://doi.org/10.1109/70.34770

Morteza Vatani, AbdolReza Rahimi, Farshad Brazandeh, and Amir Sanati Nezhad. 2009. An enhanced slicing algorithm using nearest distance analysis for layer manufacturing. In *Proceedings of World Academy of Science, Engineering and Technology*, Vol. 37. 721–726.

Marina Wada, Akito Nakano, Yoshifumi Mizuno, and Hisakazu Hada. 2016. The Light Painting by a Fluorescence String Figure. In *Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology*. ACM, 48. https://doi.org/10.1145/3001773.3001823

Ching-Cheng Wang. 1992. Extrinsic calibration of a vision sensor mounted on a robot. *IEEE Transactions on Robotics and Automation* 8, 2 (1992), 161–175. https://doi.org/10.1109/70.134271

Hei-Ting Tamar Wong, Yaozhun Huang, Sze-Chun Tsang, and Miu-Ling Lam. 2017. Real-time model slicing in arbitrary direction using octree. In *ACM SIGGRAPH 2017 Posters*. ACM, 9.

Zhengyou Zhang. 2000. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence* 22, 11 (2000), 1330–1334. https://doi.org/10.1109/34.888718

Hanqi Zhuang, Zvi S Roth, and Raghavan Sudhakar. 1994. Simultaneous robot/world and tool/flange calibration by solving homogeneous transformation equations of the form AX= YB. *IEEE Transactions on Robotics and Automation* 10, 4 (1994), 549–554. https://doi.org/10.1109/70.313105

Hanqi Zuang and Yiu Cheung Shiu. 1993. A noise-tolerant algorithm for robotic hand-eye calibration with or without sensor orientation measurement. *IEEE transactions on systems, man, and cybernetics* 23, 4 (1993), 1168–1175. https://doi.org/10.1109/21.247898