# COPYRIGHT NOTICE

# Animated Depth Images for Interactive Remote Visualization of Time-Varying Datasets

Jian Cui[1], Zhiqiang Ma[2] and Voicu Popescu[1]

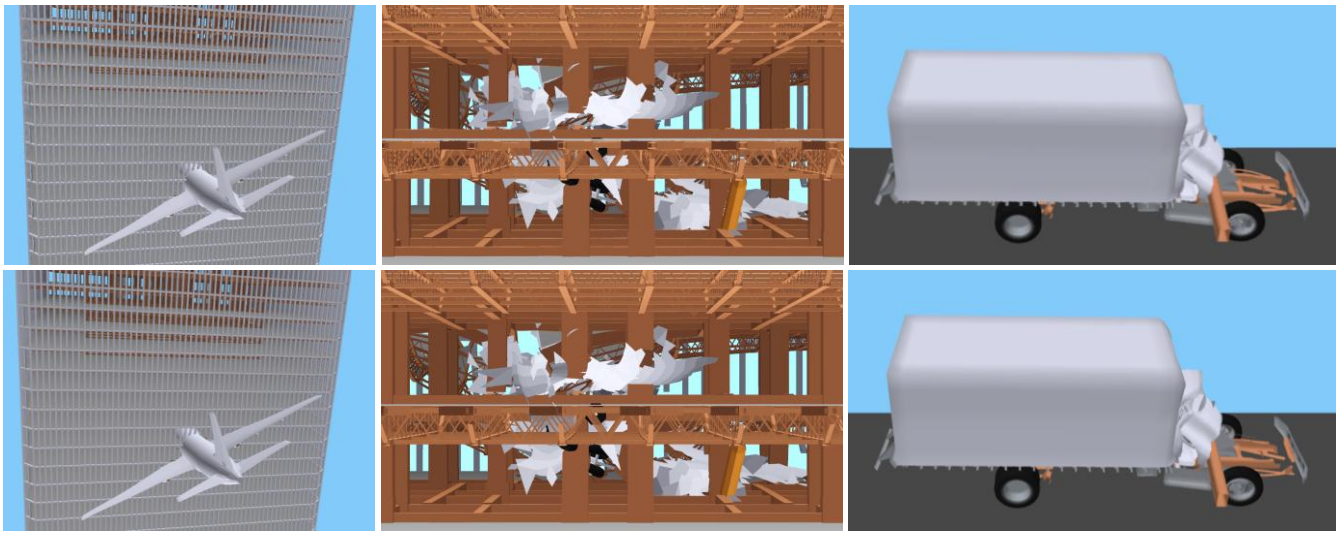

Fig. 1. Frames rendered from animated depth images (top) and from the original FEA dataset (bottom), for comparison. The animated depth images approximate sample trajectories with a user-specified maximum error of 10mm, which represents 0.01% of the spatial extent of the aircraft dataset (left and middle), and 0.1% for the truck dataset (right). An adaptive sampling strategy reduces disocclusion errors below 1% for viewpoint translations of up to 10m around the reference viewpoint.

**Abstract**—Remote visualization has become both a necessity, as dataset sizes have grown faster than computer network performance, and an opportunity, as laptop, tablet, and smartphone mobile computing platforms have become ubiquitous. However, the conventional remote visualization approach of sending a new image from the server to the client for every view parameter change suffers from reduced interactivity. One problem is high latency, as the network has to be traversed twice, once to communicate the view parameters to the server and once to transmit the new image to the client. A second problem is reduced image quality due to aggressive compression or low resolution.

We address these problems by constructing and transmitting enhanced images that are sufficient for quality output frame reconstruction at the client for a *range* of view parameter values. The client reconstructs thousands of frames locally, without any additional data from the server, which avoids latency and aggressive compression. We introduce *animated depth images*, which not only store a color and depth sample at every pixel, but also store the trajectory of the samples for a given time interval. Sample trajectories are stored compactly by partitioning the image into semi-rigid sample clusters and by storing one sequence of rigid body transformations per cluster. Animated depth images leverage sample trajectory coherence to achieve a good compression of animation data, with a small and user-controllable approximation error. We demonstrate animated depth images in the context of finite element analysis and SPH datasets.

**Index Terms**—Remote visualization, time-varying datasets, animation data compression, rigid-body decomposition, bounded error.

## 1 Introduction

The importance of remote visualization has grown and will continue to grow for the foreseeable future. One reason is that the amount of data obtained through observations and simulations increases much faster than our ability to transfer data from one geographic location to another. Another reason is that storing, processing, and displaying large datasets requires advanced capabilities which cannot and should not be replicated at all sites interested in a given dataset. Finally, the number of locations from where access to a given dataset is desired has increased with the proliferation of mobile computing platforms such as laptops, tablets, or even smartphones.

One approach in interactive remote visualization is to send the visualization parameters from the client interested in the visualization to the server who stores the dataset of interest, to compute the desired visualization frame on the server, and to send the frame to the client where it is displayed. The approach requires no storage, computing, or visualization capabilities at the client and therefore it is suitable for any type of client that can display an image. However, for networks such as the internet, the approach can suffer from long latency—the network has to be traversed twice for each frame, once to send the visualization parameters and once to receive the image. Moreover, even though a single frame is much

1. *Jian Cui and Voicu Popescu is with Computer Science Department in Purdue University, E-mail:{cui9, popescu@purdue.edu}*
2. *Zhiqiang Ma is with State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science and Engineering, Beihang University, Beijing, China, E-mail: mazhiqiang@cse.buaa.edu.cn*

more compact than the entire dataset, sending a frame several times a second still requires aggressive compression or reduced resolution. Another challenge is achieving scalability with the number of clients, as each client sends frequent frame requests to the server.

Based on the observation that a conventional image becomes obsolete with the slightest change in visualization parameters, we take the approach of sending enhanced images, or *superimages*, from the server to the client. A superimage contains sufficient data to enable the quality reconstruction of thousands of frames at the client, without any additional data from the server. Reconstruction is fast and local, which greatly alleviates latency and achieves interactive rates without aggressive compression or reduced resolution. Moreover, client requests to the server are much less frequent, which improves scalability with the number of clients.

The challenge of the proposed approach is to construct a superimage that is robust to visualization parameter changes. In the case of time-varying spatial datasets, visualization parameters typically include three translations ($x, y, z$), three rotations ($r_x, r_y, r_z$), and the focal length ($f$) for the view, as well as the time parameter ($t$). Whereas a frame is valid for a single point ($x_0, y_0, z_0, r_{x0}, r_{y0}, r_{z0}, f_0, t_0$) in this multidimensional space of visualization parameters, it is our goal to construct a superimage that is valid for an entire *volume* ($x_0+\Delta x, y_0+\Delta y, z_0+\Delta z, r_{x0}+\Delta r_x, r_{y0}+\Delta r_y, r_{z0}+\Delta r_z, f_0+\Delta f, t_0+\Delta t$). The superimage should allow reconstructing quality output frames for any visualization parameter values in the volume it covers.

Anticipating rotations and focal length changes is straight forward—the superimage should simply have a larger field of view and a higher resolution than the frame. This way the view can rotate and zoom in without running out of samples and without blurriness due to undersampling. The output image is computed through ray re-sampling and the result is correct, including for view dependent effects such volume rendering and reflections. Anticipating translations is much more challenging. The first step is to enhance the image with per pixel depth, which allows reconstructing frames from novel viewpoints. However, this is not sufficient as the result suffers from disturbing disocclusion errors even for small viewpoint translations. As the viewpoint translates, the output frame requires samples that are not visible form the viewpoint of the input depth image, and therefore are not stored in the input depth image. Such samples are missing from the reconstruction creating disocclusion errors. The main approaches for combatting disocclusion errors are to use multiple images and to combine them on the fly, to pre-combine multiple images offline into a non-redundant set of samples, or to render the depth image with a non-pinhole camera model. Such a camera not only captures samples visible from a reference viewpoint, but also captures samples hidden from the reference viewpoint but visible from nearby viewpoints.

For time-varying datasets the superimage should anticipate changes in the dataset time parameter. The problem has received little attention so far—depth image modeling/rendering efforts have only considered static datasets or datasets where motion was limited to a few objects. Truly time-varying datasets are challenging: motion essentially replicates data, and it creates complex occlusions.

In this paper we address the problem of creating an image that covers a *time interval* as opposed to a *time step*. We introduce animated depth images, which not only store depth and color samples, but also store an approximation of the trajectory of the samples over time. Like conventional depth images, animated depth images allow adapting the level of detail, provide occlusion culling, and bound the amount of data that has to be transferred. Unlike conventional depth images that can only capture a single snapshot of a dataset, animated depth images provide a quality approximation of a time-varying dataset for an entire time interval. Compared to a static/pre-computed video sequence, animated depth images have the advantage of interactivity, as the user is free to translate anywhere in the vicinity of a reference point, and the advantage of allowing to slow down the animation to any desired rate.

Animated depth images imply the following challenges. First, sample trajectories have to be stored compactly. Storing the trajectory of every sample is prohibitively expensive. Instead, we leverage sample coherence to assign trajectories to groups of samples based on a semi-rigid body decomposition of the image. Second, the reconstruction of output frames has to be done by taking into account sample connectivity that changes as samples move. The third problem is that of disocclusion errors, due not only to viewpoint translations, but now also due to motion within the dataset. We take the approach of sampling the dataset adaptively from multiple viewpoints and multiple time steps to derive on the server a good approximation of the set of necessary and sufficient samples, which are then transferred to the client. The approach is robust in the context of extremely challenging occlusion patterns. When the viewpoint desired at the client moves outside the neighborhood covered by the set of samples, or when the client desires to visualize the dataset at a time step outside of the interval covered, the adaptive sampling process is repeated, and a new set of animated samples are transferred from the server to the client.

We apply animated depth images in the context of interactive remote visualization of finite element analysis (FEA) datasets. FEA datasets are particularly challenging time-varying datasets since there are millions of degrees of freedom, with nodes moving independently, which creates complex occlusion patterns and massive data replication. Figure 1 and the accompanying video [47] (http://www.cs.purdue.edu/cgvlab/popescu/remotevis/) show that animated depth images enable the reconstruction of quality visualization frames that are comparable to frames rendered directly from the original FEA dataset. We also extend our approach to SPH datasets (video [47] and Figure 11).

## 2 Prior Work

We review prior work in remote visualization and prior work aimed at overcoming the problem of disocclusion errors.

### 2.1 Remote Visualization

We classify remote visualization approaches based on the computational load distribution between client and server. At one end of the spectrum is the approach of doing all the work on the server and of sending visualization frames to the client, which acts like a simple terminal that displays images [1, 2, 3, 4, 44]. The approach is appealing because it doesn't require any storage or computation capability at the client, which is particularly beneficial when the client runs on limited hardware such a smartphone [40]. Moreover the approach is general—any visualization algorithm and any type of dataset are supported as long as the server can produce the visualization frames which are displayed at the client. The approach suffers from the disadvantage of limited interactivity due to network bandwidth limitations and latency, which can be addressed by reducing resolution or by aggressive compression. Moreover, the approach implies frequent requests from the client to the server, i.e. once for every change in the visualization path requested by the user, which can lead to server overload, and to poor visualization service quality when connection to the server is lost. The problem has been addressed in the context of virtual environments by anticipating user interactions [43, 45], which however comes at a loss of generality and which is difficult to extend to the context of remote visualization where the visualization target might not be known a priori.

At the other end of the spectrum is the approach of reducing the dataset to a manageable size, to transfer the reduced dataset to the client, and to run the visualization algorithm at the client. There is a large variety of techniques for reducing dataset size, including multi-resolution and level of detail [5, 6], feature extraction [7, 8], progressive refinement [9, 10], occlusion culling [11, 12], and data compression [13, 14] techniques. One technique [15] targets dynamic datasets specifically and reduces the dataset by compressing the trajectories of the simulation nodes (i.e. vertices of finite element geometry) through rigid body decomposition. The strength of the general approach of reducing the dataset at the server is that once the reduced dataset is transferred to the client, the visualization doesn't

depend on the network anymore. One weakness of the approach is the need for data reduction algorithms for specific data types and visualizations. Another weakness is the challenge of reducing large datasets aggressively while preserving features of interest that are typically not known a priori.

In the middle of the spectrum are hybrid approaches: most of the work is done at the server while the client also shoulders part of the burden with the reward of improved interactivity. One example are approaches that use sophisticated compression schemes on the server that require decompression at the client in parallel [16, 17], or with the help of GPUs [18]. Another example are systems that send enhanced images, or superimages, from the server to the client, such as depth images [19, 20], or non-uniformly sampled images [21].

View-dependent effects such as volume rendering or reflections are challenging for such approaches since it requires either computing the expensive effect at the client, or increasing the size of the representation considerably to include view-dependent color. The visualization by proxy framework [37] succeeds at decomposing and translating a static volume dataset into a compact set of proxy depth and attenuation images that serve as an intermediate representation in the context of volume rendering. Occlusions are addressed using a single-pole occlusion camera [29], which creates multi-perspective proxy images that avoid simple occlusion patterns between a small number of features. Visualization by proxy provides a general framework where volume rendering operations can be quickly approximated, without accessing the original dataset. The coherent visualization of a *time-varying* volume dataset without access to the entire dataset, as needed for example in the case of remote visualization, has been proposed using ray attenuation functions [38]. The method has the limitations of not allowing viewpoint changes and of restriction to exploratory use due to approximation errors. Our method focuses on viewpoint changes in opaque surface rendering for dynamic FEA datasets, it handles arbitrarily complex occlusion patterns, and it enforces a user selected error bound on sample trajectory approximation.

The animated depth images introduced in this paper falls in the category of hybrid approaches. Hybrid approaches are general as far as the client is concerned—like conventional images, superimages insulate the client from the complexity and variety of visualization algorithms and dataset types. Hybrid approaches also improve interactivity—like reduced datasets, superimages are sufficient to reconstruct frames at the client without any additional data from the server. The challenge of the hybrid approach is to devise superimages that can cover a large volume of the multidimensional space of visualization parameters. Previous work was concerned with view rotations and focal length variations [21] and with viewpoint translations [19]. Animated depth images target the changes in the time parameter for time-varying datasets.

Like the previous method for dynamic dataset reduction through rigid body decomposition discussed above [15], our method compresses animation data by leveraging motion coherence of animated depth image samples. However, the previous method works at dataset level and does not scale with dataset size. Animated depth image are a hybrid remote visualization approach, with cost independent of dataset extent or resolution, and only dependent on output image resolution. In order to achieve this, the animated depth image approach contributes solutions to the problems of fast, hierarchical rigid body decomposition of the animated depth image samples, of adaptive sampling to avoid disocclusion errors due to viewpoint translation and sample motion, and of visualization output frame reconstruction from the animated depth image samples.

### 2.2 Alleviating Disocclusion Errors

The idea of using a depth image as a rendering primitive dates back to early image-based rendering work [22]. However, a single depth image is not sufficient—the slightest viewpoint translation creates disturbing disocclusion errors. Disocclusion errors have been addressed by combining multiple depth images at run time [22, 23, 24] or off-line [25, 26, 27]. Another approach is to render the depth

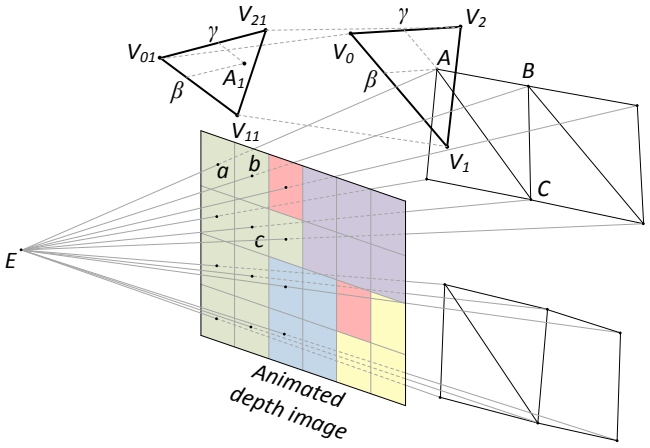


Fig. 2. Animated depth image illustration. Pixel *a* is unprojected to 3-D point *A*, which belongs to dataset triangle $V_0V_1V_2$. As the triangle moves to $V_{01}V_{11}V_{21}$, sample *A* moves to $A_1$. Sample motion is approximated by partitioning the image into rigid bodies (see color highlights). The implicit sample connectivity is used to define a 3-D mesh used for high-quality reconstruction of output frames.

image with a non-pinhole camera model, such as a multiple-center-of-projection camera [28], an occlusion camera [29], a general linear camera [30], or a graph camera [31]. The advantage of the non-pinhole camera approach is that the samples needed for the view region are arranged in a single-layer depth image with good pixel to pixel coherence, which is compact and compresses well. This is of great importance in our context where we aim to reduce the amount of data that has to be transferred from the server to the client. However, constructing non-pinhole camera models that capture all samples needed in the context of the complex occlusion patterns that arise in FEA datasets is challenging. We opt instead for the approach of pre-combining multiple depth images. This suits the remote visualization scenario well—the work of rendering multiple depth images and of combining them into a non-redundant set of samples is done at the server, before transmission.

## 3 Animated Depth Image Definition

An animated depth image stores:

(a) *color and depth samples* to approximate the color and geometry of the dataset,

(b) *sample trajectories* to approximate the motion in the time-varying dataset, and

(c) *sample connectivity* to enable a quality triangle-mesh-based reconstruction of visualization frames.

(a) Like a conventional depth image, an animated depth image is an image that stores color and depth per pixel, obtained by rendering the dataset for a given view $PPC_0$ and at a given time step $t_0$. The pixel data can be unprojected to a 3-D point with color using $PPC_0$. In Figure 2, the eye of $PPC_0$ is *E*, and pixels *a*, *b*, and *c* are unprojected to 3-D points *A*, *B,* and *C* using the $PPC_0$ rays *Ea, Eb,* and *Ec* and the depths $z_a$, $z_b$, and $z_c$ stored at the three pixels.

(b) Unlike a conventional depth image, an animated depth image also encodes the trajectories of its samples. Consider a sample *A* that belongs to a dataset triangle $V_0V_1V_2$ (Figure 2). The sample is defined by its barycentric coordinates $\alpha$, $\beta$, and $\gamma$:

$$A = \alpha V_0 + \beta V_1 + \gamma V_2 \quad (1)$$

As the triangle vertices move to $V_{01}$, $V_{11}$, and $V_{21}$, respectively, the sample moves to $A_1$ which is found using the sample's barycentric coordinates:

$$A_1 = \alpha V_{01} + \beta V_{11} + \gamma V_{21} \quad (2)$$

In complex time varying datasets, such as for example FEA datasets, triangles have complex trajectories modeled with hundreds

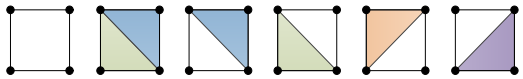

Fig. 3. Illustration of the six possible connectivity scenarios for a neighborhood of 2x2 samples.

of time steps. Storing the trajectory of each of individual sample of an animated depth image results in a large data size that precludes applications such as remote visualization. We leverage the local coherence of motion in time-varying datasets and group nearby animated depth image samples into *rigid bodies*. A rigid body is a cluster of samples whose motion is approximated well with a single sequence of rigid body transformations $X_1, X_2, \ldots, X_{n-1}$, where $X_i$ is a conventional 4×4 transformation matrix that encodes a rotation and a translation but no scaling, and $n$ is the number of time steps. Instead of storing the trajectory $A_0, A_1, \ldots, A_{n-1}$ of each sample in the rigid body, the animated depth image only stores the initial position $A_0$ of each sample and the sequence of transformations $X_1, X_2, \ldots, X_{n-1}$. The subsequent positions $A_i$ of the sample are approximated with:

$$A_i^* = X_i A_0 \tag{3}$$

Consider for example a piece of an axle of the truck in the FEA simulation shown in Figure 1 (right). When the truck impacts the barrier the piece breaks off and flies away spinning. The samples of the piece can be grouped into a rigid body because their motion throughout the simulation can be approximated with the same sequence of rigid body transformations. The piece can also bend slightly as it breaks off, as long as a user imposed maximum trajectory approximation error is not exceeded. If the piece bends significantly and the error threshold would be exceeded when using a single rigid body, the piece is approximated with two or more smaller rigid bodies. In Figure 2 there are four rigid bodies highlighted with green, blue, purple and yellow. Not all samples are assigned to rigid bodies (red in Figure 2). Such unassigned samples have their trajectory encoded explicitly.

(c) The animated depth image samples are used to reconstruct output visualization frames from novel views. One approach is to resort to a point-based rendering technique that does not require explicit sample connectivity. A high-quality reconstruction approach is to connect samples in a triangle mesh leveraging the connectivity defined implicit by the regular grid of pixels. However, not all four adjacent samples should be connected by two triangles. Like for a conventional depth image, samples should be disconnected if they are on opposite sides of a depth discontinuity: the silhouette samples of a foreground object should not be connected to their neighboring samples that belong to the background object. Animated depth images also require that samples be disconnected when they belong to surfaces that move apart over the course of the simulation. The animated depth image stores at sample $A$ the connectivity in the 2×2 sample neighborhood that has $A$ as its top left sample (Figure 2). The issue of connectivity for the purpose of reconstruction is of course orthogonal to the issue of rigid body decomposition for sample trajectory approximation (Figure 3).

## 4 Animated Depth Image Construction

Consider a time-varying dataset $D$ modelled with triangles whose vertices move on piecewise linear trajectories over $n$ time steps from $t_0$ to $t_{n-1}$. Given a reference view $PPC_0$ and a sample trajectory approximation error threshold $\varepsilon$, an animated depth image of $D$ is constructed in three major steps, with each step computing one of the main components of the animated depth image (Section 3):

(a) Compute samples by rendering $D$ at $t_0$ from $PPC_0$.
(b) Compress sample trajectory through rigid body clustering.
(c) Compute sample connectivity.

The first step (a) computes a conventional depth image of the dataset by rendering the triangles in $D$ at their $t_0$ position. The triangle color could originate for example from materials or from false color schemes, and could be encoded for example with a color per vertex or with textures. No matter what the origin of the color or encoding mechanism, the color is transferred to the depth image which stores a color sample per pixel. In addition to color and depth, each pixel stores the ID of the dataset triangle it samples. The triangle ID is used to compute the barycentric coordinates of the pixel sample using Equation 1.

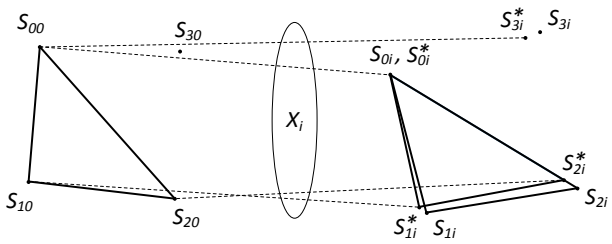


Fig. 4. Construction of rigid body transformation $X_i$. $X_i$ is maps $S_{00}$ to $S_{0i}$, triangle plane $S_{00}S_{10}S_{20}$ to triangle plane $S_{0i}S_{1i}S_{2i}$, and triangle edge $S_{00}S_{20}$ to triangle edge $S_{0i}S_{2i}$.

### 4.1 Rigid body clustering

The second step (b) of animated depth image construction computes a compact representation of sample trajectories by clustering samples into rigid bodies. This clustering is based on the reasonable assumption that samples that move together like a rigid body are close together in model space, and hence in image space. The sample trajectory approximation error is bound by the threshold $\varepsilon$. The clustering proceeds in bottom-up fashion with the following steps:

(b.1) Seed rigid bodies in 2×2 sample neighborhoods.
(b.2) Merge rigid bodies recursively.
(b.3) Finalize rigid bodies.

**Step b.1** takes a pass over the depth image computed at Step (a) and forms initial rigid bodies of 2×2 neighboring samples, whenever possible. Let $S_0, S_1, S_2$ and $S_3$ be the four samples of the 2x2 neighborhood. The samples form a rigid body if a sequence of rigid body transformations $X_1, X_2, \ldots, X_{n-1}$ places each of the four samples for each time step within epsilon of its true dataset position.

$$\begin{aligned} S_{ji}^* &= X_i S_{j0} \\ \left|S_{ji}^* - S_{ji}\right| &< \epsilon \\ 0 \le j \le 3, &\quad 1 \le i \le n-1 \end{aligned} \tag{4}$$

In Equation 4, $S_{ji}^*$ is the position of sample $S_j$ at time step $i$ as approximated using the rigid body transformations (Equation 3), $S_{ji}$ is the true position of sample $S_j$ at time step $i$ as given by the dataset using the barycentric coordinates computed at Step (a) (Equation 2), and the Euclidian distance between $S_{ji}^*$ and $S_{ij}$ has to be smaller than $\varepsilon$ for each sample $j$ and for each time step $i$.

We construct the rigid body transformations $X_i$ one at the time, starting with $X_1$ and ending with $X_{n-1}$. $X_i$ is constructed using three samples by adapting a previously developed method [15] as shown in Figure 4. $X_i$ is constructed by combining a translation that takes the initial position $S_{00}$ of sample $S_0$ to its position $S_{0i}$ at time step $i$, with a rotation that aligns the planes of triangles $S_{00}S_{10}S_{20}$ and $S_{0i}S_{1i}S_{2i}$, and with a rotation about the normal of the common triangle plane that aligns edges $S_{00}S_{20}$ and $S_{0i}S_{2i}$. Once $X_i$ is constructed, the approximate sample positions at time step $i$ are computed by applying the transformation $X_i$ to the initial sample positions.

The approximation error for $S_{0i}$ is 0 since transformation $X_i$ is constructed such that $S_{0i}^*$ and $S_{0i}$ coincide. If the distance between the true and approximated position of any of the other three samples, including $S_3$, exceeds $\varepsilon$, the four samples cannot form a rigid body and the iterative construction of the sequence of transformations $X_i$ stops. If all errors are within $\varepsilon$, the algorithm proceeds with constructing transformation $X_{i+1}$. Once $X_{n-1}$ is constructed, the four samples define a rigid body. Figure 5, top, illustrates the 2×2 sample

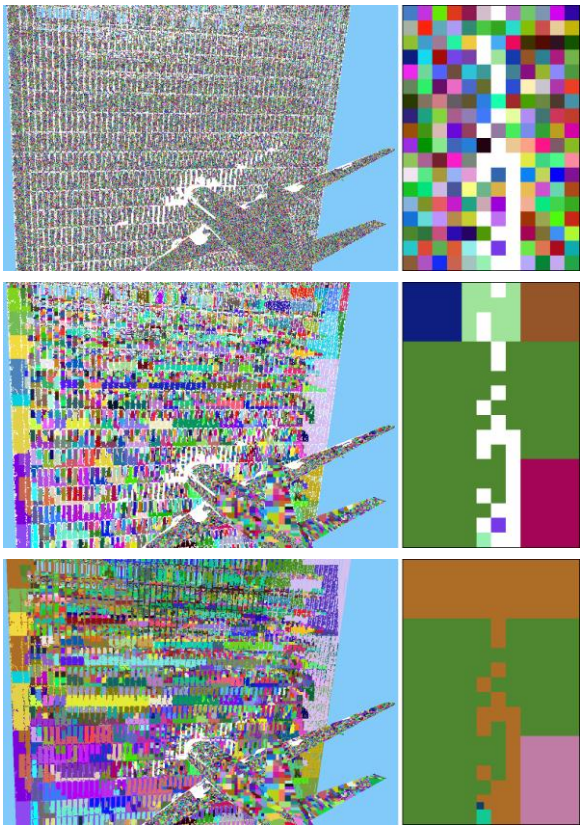

Fig. 5. Visualization of rigid bodies and magnified fragment after Step b.1 (top), Step b.2 (middle) and Step b.3 (bottom) of the animated depth image construction algorithm. The number of rigid bodies and the percentage of unassigned samples for each of the three images are 125,752 and 20.72%, 12,945 and 20.72%, and 6,322 and 1.45%.

rigid bodies constructed by Step b.1. For the 2×2 neighborhoods where rigid body construction fails, the four samples remain unassigned (white in Figure 5, top).

**Step b.2** reduces the number of rigid bodies through merging. Merging proceeds in bottom-up quadtree fashion. The rigid bodies of four neighboring nodes at the current level of the quadtree are merged to form the rigid bodies of the parent node at next level up. The inner horizontal and vertical boundaries are traversed one pair of samples at the time. If the two samples of a pair belong to different rigid bodies, the algorithm attempts to merge the two rigid bodies.

Given two rigid bodies $A$ and $B$, rigid body $B$ could be merged into $A$ if the sequence of rigid body transformations of $A$ approximates the trajectories of all the samples in $B$ within $\varepsilon$. When $B$ is merged into $A$, the rigid body $B$ is abandoned while the transformations in $A$ will represent all samples in both $A$ and $B$. Figure 5, middle, shows the rigid bodies obtained after Step b.2. The rigid bodies are larger and in smaller number compared to the starting seed rigid bodies (top).

Transformations $X_i$ are rigid body transformations that exclude scaling. Scaling would increase the trajectory modeling capability of the transformation, which would lead to finding more rigid body seeds at Step b.1. However, allowing for scaling can seed rigid bodies from four samples that belong to different dataset entities, and such anomalous rigid bodies cannot be merged, resulting in a large number of small rigid bodies which is inefficient.

**Step b.3** improves the rigid body partitioning of the animated depth image by overcoming limitations of Steps b.1 and b.2. First, Step b.1 only attempts to form a rigid body between the four samples of a 2×2 neighborhood. If the attempt fails, the four samples remain unassigned, whereas, for example, it could be that sample $S_0$ can be assigned to the rigid body to the left of the 2×2 neighborhood, or even to a distant rigid body. To overcome this limitation, Step b.3

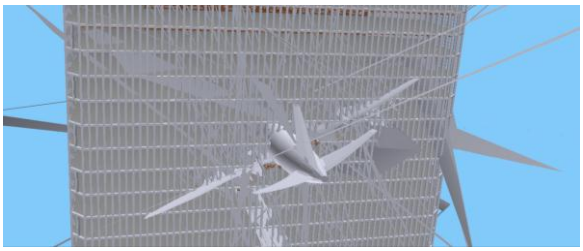

Fig. 6. Incorrect reconstruction that does not take into account the temporal changes in sample connectivity.

tests each unassigned sample for possible inclusion into each of the existing rigid bodies. Second, Step b.2 only attempts to merge rigid bodies that are adjadjacent. Step b.3 attempts to merge all pairs of rigid bodies. Figure 5, bottom, shows the final rigid bodies. Some large rigid bodies are formed by merging non-adjacent rigid bodies. As expected, the few remaining unassigned samples are concentrated around the impact region where samples move chaotically. The trajectories of the unassigned samples are approximated using the greedy polyline simplification algorithm of Ramer [32] and Douglas-Peucker [33], conforming to the same error threshold ε.

### 4.2 Sample connectivity computation

The third and final step (c) of the construction of the animated depth image computes sample connectivity to enable triangle-mesh-based reconstruction of output visualization frames.

For conventional depth images, connectivity is computed using the second order derivative of the depth map. Values larger than a threshold indicate depth discontinuities, and reconstruction triangles spanning across depth discontinuities are removed [24]. However, naively using such connectivity data computed at $t_0$ for all time steps of an animated depth image results in severe artifacts (Figure 6) due to sample motion and erosion. When a finite element, e.g. a piece of a structural steel beam, undergoes excessive stress, the element "erodes", i.e. it is eliminated from the FEA simulation for the subsequent time steps. When an element erodes, all the dataset triangles used to represent the element erode as well, as do all samples contributed by the eroding dataset triangles. A reconstruction triangle connecting three samples should clearly not outlive its first eroding sample, but this is not always sufficient.

Consider a structural steel beam in the aircraft impact simulation shown in Figure 1. Let's assume that the beam is modeled with six dataset triangles with vertices $V_0$ to $V_7$ (Figure 7), and let's assume that dataset triangles $V_1V_2V_6$ and $V_2V_5V_6$ erode at time step $i$. If the beam is far from the eye of the perspective camera $PPC_0$ used to construct the animated depth image, or if the beam is seen by $PPC_0$ at an angle, it can happen that neither $V_1V_2V_6$ nor $V_2V_5V_6$ has a sample in the animated depth image. In Figure 7, samples $S_0$, $S_1$, $S_2$ and $S_3$ skip $V_1V_2V_6$ and $V_2V_5V_6$. Simply checking for erosion at the vertices of the reconstruction triangles will lead to the erroneous conclusion that the reconstruction triangles do not erode.

The correct eroding time step of each reconstruction triangle is set as follows. Consider reconstruction triangle $S_0S_1S_2$. For each of its edges, e.g. $S_1S_2$, compute the shortest path $P_{12}$ between the dataset

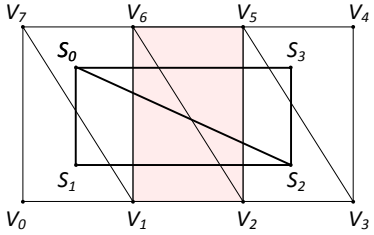


Fig. 7. Reconstruction triangles $S_0S_1S_2$ and $S_2S_3S_4$ should erode when dataset triangles $V_1V_2V_6$ and $V_2V_5V_6$ (red) erode, even though samples $S_i$ belong to dataset triangles that do not erode.

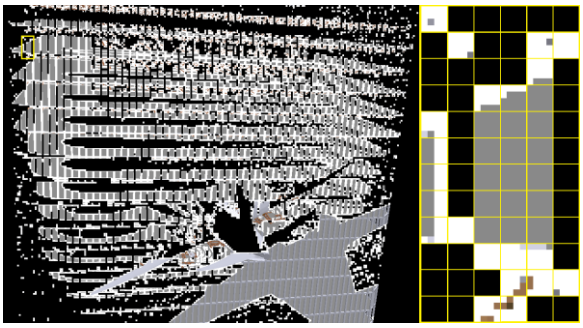

Fig. 8. Non-redundant samples gathered by our adaptive algorithm.

triangles of the two samples, i.e. $V_0V_1V_7$ and $V_2V_3V_5$, respectively. The shortest path is computed in the dataset triangle adjacency graph at $t_0$. The dataset triangle adjacency graph is an undirected graph with nodes corresponding to dataset triangles and with edges corresponding to dataset triangles sharing a vertex. The eroding time step of edge $S_1S_2$ is set as the first time step where $P_{12}$ is interrupted. A path is interrupted when one of the dataset triangles it enumerates erodes. Finally, the eroding time step of the reconstruction triangle is set to be the earliest of the eroding time steps of its three edges. For the example in Figure 7, each reconstruction triangle has two edges that erode at time step $i$ and one edge that does not erode, and thus the eroding time step for the two reconstruction triangles is $i$.

## 5 Adaptive Sampling in Space and Time

Like a conventional depth image, an animated depth image suffers from disocclusion errors when the viewpoint translates. Moreover, disocclusion errors also occur when sample motion uncovers new samples. We alleviate disocclusion errors by sampling the dataset adaptively from multiple viewpoints and at multiple time steps (Section 5.1), and by eliminating redundant samples (Section 5.2).

### 5.1 Adaptive Sampling

Given an animated depth image $ADI_0$ with view $PPC_0$ covering $n_t$ time steps starting at $t_0$, the goal is to enhance $ADI_0$ with sufficient samples to ensure a disocclusion error free reconstruction from anywhere in a neighborhood of $PPC_0$, and at any time in $[t_0, t_0 + n_t]$. We define the neighborhood of $PPC_0$ with an equilateral triangle of radius $q$ (i.e. the radius of its circumscribed circle), perpendicular to the view direction of $PPC_0$, and centered at its eye. The length $q$ is an input parameter. The larger the triangle, the bigger the viewpoint translation range at the client, but also the bigger the data size.

Our adaptive sampling scheme renders conventional depth images from various locations inside the viewpoint triangle and at various time steps. The sampling process stops when the percentage of non-redundant samples contributed by a new depth image drops below a threshold $g$. In order to enable the elimination of redundant samples, depth images and animated depth images are split into square tiles of size $t_w \times t_w$ (we use $t_w = 4$). Figure 8 shows the tiles containing the non-redundant samples contributed by a depth image. Tiles, like complete images, allow storing connectivity information compactly. Given a tile size $t_w$ and a threshold $g$ for the percentage of new samples contributed by a new image, our adaptive sampling algorithm proceeds as follows.

1. Initialize the set $S$ of animated depth image tiles to empty.
2. Subdivide viewpoint triangle and time interval recursively. For each new viewpoint $e$ and new time interval $t$:
   a. Render a depth image $DI_{et}$
   b. Compute non-redundant samples $DI_{et} - S$
   c. If $|DI_{et} - S| / |DI_{et}| > g$
      i. $S_{et}^* = \text{Tile}(DI_{et} - S, t_w)$
      ii. $S_{et} = \text{ConstructADI}(S_{et}^*)$
      iii. $S = S \cup S_{et}$
      iv. Continue recursive subdivision at $e$ and $t$

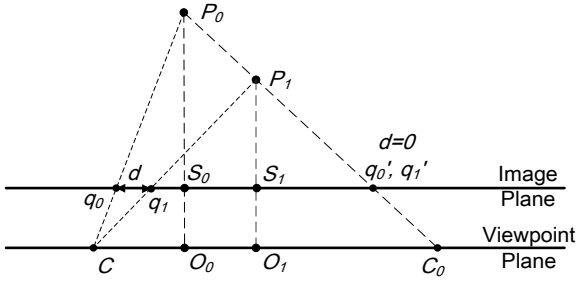


Fig. 9. Projections of samples $P_0$ and $P_1$ onto the image plane as the viewpoint translates on the viewpoint triangle plane. The distance $d$ between the projections is 0 at $C_0$ where $q_0'$ and $q_1'$ coincide, and it increases away from $C_0$.

The algorithm samples the space of possible viewpoints and time steps recursively. For each point in this space, a new depth image $DI_{et}$ is rendered at Step 2.a. Step 2.b checks for each sample in $DI_{et}$ whether it is redundant with the samples already collected in $S$. If the percentage of non-redundant samples in $DI_{et}$ is below the value of the input parameter $g$, the recursive subdivision stops. If $DI_{et}$ contributes a sufficient number of non-redundant samples, the non-redundant parts of $DI_{et}$ are partitioned into tiles (Step 2.c.i), an animated depth image tile is computed for each depth image tile (Step 2.c.ii), the new set of tiles is added to $S$ (Step 2.c.iii), and the subdivision continues recursively (Step 2.c.iv). The first depth image computed by the adaptive sampling algorithm corresponds to the center of the viewpoint triangle (also the eye of $PPC_0$) and to $t_0$. Since $S$ is initially empty, $S$ will contain the entire animated depth image constructed for $PPC_0$, which guarantees highest-quality reconstructions for the view $PPC_0$.

### 5.2 Sample Redundancy

The first task is to define sample redundancy. Depth images contain point samples and two samples will in general not correspond to the same 3-D point. We define two samples as redundant if and only if they project within one pixel in all views $PPC_i$, where $PPC_i$ is identical with $PPC_0$ except that the viewpoint can be anywhere inside the equilateral viewpoint triangle with radius $q$. If two samples belonging to different surfaces happen to project at nearby locations from a viewpoint, motion parallax separates the two samples when seen from a different viewpoint, and the samples are correctly labeled as non-redundant. Our definition of redundancy ignores view rotations which only introduce a negligible variation of the distance between the projections of two samples. Given two samples defined at different time steps $t_a$ and $t_b$, the samples are redundant if they are redundant when the second sample is brought to time step $t_a$.

The second task is to find a method for quickly checking for sample redundancy, given our definition. Figure 9 illustrates the projection of a pair of two samples $P_0$ and $P_1$ as the viewpoint translates on the plane of the viewpoint triangle. The image plane is constant as there are no view direction rotations. Let $C_0$ be the intersection between $P_0P_1$ and the viewpoint plane. For a viewpoint $C$, the square of the distance $d$ between the image plane projections $q_0$ and $q_1$ of $P_0$ and $P_1$ is given by:

$$d^2 = |q_0 - q_1|^2 = (w_0 - w_1)^2 \cdot x^2 \tag{5}$$

where $w_i$ ($i$ = 0, 1) and $x$ are defined as:

$$w_i = \frac{|q_i - q_i'|}{|C - C_0|} = \frac{|P_i - S_i|}{|P_i - O_i|}$$
$$x = |C - C_0| \tag{6}$$

Since $w_0$ and $w_i$ do not depend on the current viewpoint $C$, $d^2$ is a quadratic function in $x$, with the minimum value of 0 reached when $C = C_0$. Distance $d$ increases away from $C_0$. Over the entire viewpoint triangle, the maximum distance occurs at one of the three

vertices of the viewpoint triangle. Consequently, given two samples, the largest distance between the projections of the samples over all viewpoints inside the viewpoint triangle can be easily computed as the maximum over the three distances obtained at the viewpoint triangle vertices. If the maximum is less than one pixel, the new sample is discarded as redundant.

## 6 Visualization Frame Reconstruction

Given a set of animated depth image tiles $S$, an output image view $PPC_i$, and time parameter value $t_j$, the client reconstructs the corresponding visualization frame by rendering each tile in $S$.

A first step computes the positions at time $t_i$ of each 3-D sample stored in the tiles in $S$. If a sample belongs to a rigid body, the position is reconstructed by applying to the initial position of the sample the appropriate transformation from the rigid body trajectory. If the sample is unassigned, that is it does not belong to a rigid body, the current position is inferred from the trajectory of the sample which is stored explicitly in the animated depth image representation. The time value $t_i$ need not be one of the simulation time steps: the simulation can be visualized at arbitrarily slower speeds by interpolating simulation computed sample positions.

Once the current position of the 3-D samples is established, the frame reconstruction problem is reduced to the well-studied problem of reconstructing output images with novel viewpoints from input depth images. Any prior work method developed for rendering from depth images can be used, including splats [34], surfels [35], forward rasterization [36], and triangle-mesh reconstruction [22].

We use two reconstruction modes: an efficient point-based rendering approach where each sample is rendered with a 2×2 output image pixel splat, and a high-quality triangle-mesh reconstruction. A $t_w \times t_w$ tile defines a mesh of up to $(t_w\text{-}1) \times (t_w\text{-}1) \times 2$ triangles by connecting any 4 neighboring samples with 2 triangles. The connectivity information stored by the tiles is used to avoid defining triangles across depth discontinuities. Samples not connected in any triangle are drawn as points. Depending on the rendering capability at the client, more shading flexibility can be supported by incorporating into the animated depth image additional per-sample shading parameters. We have extended animated depth images to also store per-sample normal, which allows for dynamic relighting at the client (see Figure 10 and video [47]).

## 7 Extension to SPH Datasets

The animated depth image is a general compact representation of time-varying color and depth datasets. So far we have demonstrated animated depth images in the context of triangle meshes resulting from FEA simulations. Animated should be extendable to other types of time-varying datasets, with certain modifications. Consider, for example, a smoothed-particle hydrodynamics (SPH) dataset where the 3-D position of each particle center is recorded over a sequence of time steps. One option for visualizing such a dataset is to render each moving particle as an opaque, shaded sphere. Tessellating each sphere would result in a dataset of moving triangles that can be handled as described. However, tessellating each particle results in a one hundred-fold explosion in the number of primitives. Instead we modify our method to handle particles directly.

Like before, the animated depth image stores color and depth samples. A pixel sample is a reference to its particle and not a 3-D point—the triangle ID is replaced with the particle ID and no barycentric coordinates are needed. Rigid bodies are computed using

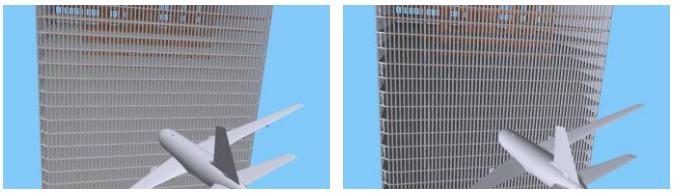

Fig. 10. Frames with lighting computed at the client.

Fig. 11. Frame reconstructed from an animated depth image (top) and frame obtained by rendering the original SPH dataset (bottom). The residual disocclusion error is 0.35%.

the center of the particle to which each sample belongs, and not the sample's 3-D point. The animated depth image encodes the trajectories of the centers of the particles and not the trajectories of individual samples. Sample connectivity is not needed, as output visualization frames are reconstructed by rendering each particle as an (independent) sphere. The adaptive sampling in space and time remains the same. Sample redundancy detection is now replaced with a simple test for particle ID uniqueness. For the example in Figure 11 the SPH dataset was captured using the animated depth image approach. The trajectory approximation error threshold is 1% of the radius of the sphere modelling the particle, the average disocclusion error rate is 0.17%, and the compression factor is 40.

## 8 Results and Discussion

We have applied animated depth images to multiple reference views in two FEA datasets—the truck dataset (Figure 1, right) and the aircraft dataset (Figure 1 left and middle), as well as to an SPH dam break simulation dataset (Figure 11). The truck dataset has 81 time steps and covers a region of 15m × 5m × 3.3 m. The truck dataset contains 0.63M triangles and 0.28M vertices, for a total of 23M vertex positions. The aircraft dataset has 170 time steps, it is segmented into 3 segments of 58, 58, and 56 time steps, and it covers a region of 110m × 90m × 60m. The aircraft dataset contains 2.08M triangles and 2.01M vertices, for a total of 342M vertex positions. The SPH dataset has 82 time steps, it covers a region of 100 × 18 × 20 and it contains 2.17M particles for a total of 178M particle center positions. The particles are rendered as spheres with radius 0.1.

### 8.1 Quality

We investigate quality along three directions: sample trajectory approximation, residual disocclusion, and reconstruction errors.

*Sample trajectory approximation error*

Animated depth images approximate sample trajectories with a user-controlled maximum approximation error. Larger error bounds $\varepsilon$ lead of course to fewer rigid bodies, fewer unassigned samples, and a more compact representation. Table 1 shows how the data size decreases as the approximation error $\varepsilon$ increases for the aircraft dataset. $\varepsilon$ is given in absolute values (e.g. 10mm), in approximate relative values (e.g. 10mm/100m = 0.0001 = 0.01%), and in maximum image plane error values (e.g. 0.01pix). The error in the image plane is estimated by projecting a segment of length $\varepsilon$ on the view $PPC_0$ of the animated depth image. The segment is parallel to the image plane and it is located at the depth of the closest sample in the animated depth image, which provides a conservative upper bound of the image plane error. The image resolution is 1,280 × 720.

Table 1. Data size variation with trajectory approximation error for the aircraft dataset.

| $\varepsilon$ | [mm] | 5 | 10 | 25 | 50 | 100 | 500 | 1000 |
|---|---|---|---|---|---|---|---|---|
| | [%] | 0.005 | 0.01 | 0.025 | 0.05 | 0.1 | 0.5 | 1.0 |
| | [pix] | 0.04 | 0.09 | 0.21 | 0.43 | 0.85 | 4.25 | 8.50 |
| Size [MB] | | 39 | 29 | 22 | 19 | 18 | 17 | 16 |

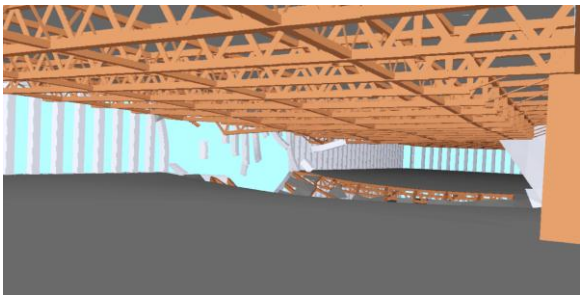

Fig. 12. Side view of the aircraft dataset.

*Residual Disocclusion Error*

Given an output frame $F$, we measure the residual disocclusion error rate in $F$ as the percentage of samples in the corresponding truth frame $F_0$ that are not present in $F$. Table 2 shows the number of depth images the adaptive sampling algorithm uses, the size of the resulting set of animated depth image tiles, and the maximum and average residual disocclusion error rates for various values of the convergence threshold $g$. The maximum and average residual disocclusion error rates are computed over a visualization sequence of 50,000 frames, which were reconstructed from viewpoints and at time steps that sample the viewpoint triangle and time step interval densely and comprehensively. As expected, a smaller $g$ value yields fewer residual disocclusion errors at the cost of sampling the dataset more. Disocclusion errors are not linear, as seen in the jump of the maximum error rate when the $g$ value changes from 0.3% to 0.5%. The size of the resulting representation decreases slowly as the residual disocclusion error is small (i.e. up to $g$ = 0.3%) and then it decreases rapidly indicating that samples are missed. For all the images shown in the paper and in the video, the value for $g$ is 0.1%.

Table 2. Adaptive sampling performance for various convergence threshold $g$ values for the aircraft dataset.

| $g$ [%] | Number of Sampling Depth Images | Data size [MB] | Residual Disocclusion Error Rate [%] | |
|---|---|---|---|---|
| | | | Max | Avg. |
| 0.05 | 208 | 24 | 0.31 | 0.084 |
| 0.1 | 184 | 24 | 0.31 | 0.11 |
| 0.2 | 136 | 23 | 0.37 | 0.14 |
| 0.3 | 124 | 23 | 0.46 | 0.16 |
| 0.5 | 76 | 20 | 40.0 | 0.65 |

Table 3 reports typical residual disocclusion error rates for the truck and aircraft datasets. The 3 regions of the aircraft dataset that were investigated are shown in Figure 1 left (*outside*), Figure 12 (*side*), and Figure 1 middle (*reverse*). The residual disocclusion errors is small in all cases.

The graph in Figure 13 shows the variation of the average and maximum residual disocclusion error rates over all viewpoints as a function of time step for the outside, side, and reverse regions of the aircraft dataset. The maximum graph line for the reverse region (solid green) varies considerably due to the fast and chaotic motion in that dataset region and the proximity of the reference viewpoint.

Table 3 Maximum and average residual disocclusion error rates.

| Dataset | | Residual Disocclusion Error Rate [%] | |
|---|---|---|---|
| | | Max | Avg. |
| Truck | | 0.23 | 0.05 |
| Aircraft | outside | 0.64 | 0.17 |
| | side | 0.46 | 0.16 |
| | reverse | 0.69 | 0.21 |
| SPH | | 1.1 | 0.17 |

*Reconstruction Error*

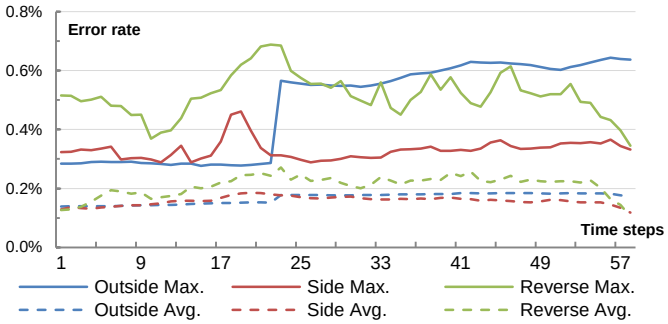


Fig. 13. Variation of residual disocclusion error rates over time steps.

Even if all samples needed are captured, the reconstructed frame will differ slightly from a frame rendered directly from the original dataset. Figure 14 shows that such differences are small. The largest errors are seen at residual disocclusion errors and at edges. The reasons for the differences include:

- the additional resampling introduced by the intermediate animated depth image representation; the output frame has the same resolution as the input animated depth image, whereas, for conservative reconstruction, the input should have twice the resolution of the output;
- the undersampling caused when the screen footprint of samples increases from the reference view due to view changes or sample animation;
- the conservative early elimination of a triangle between samples that erode at different time steps, as opposed to splitting the triangle into fragments each eroding at a different time.

## 8.2 Performance

### 8.2.1 Data size

Table 4 shows the data size variation for the animated depth image representation as a function of the length $q$ of the viewpoint triangle side. The data was measured for the outside region of the aircraft dataset, the output resolution is 1,280 × 720, and the number of simulation time steps is 58. As can be seen in the relative size row, the ratio of the data size to the viewpoint triangle edge length decreases as the viewpoint triangle gets bigger, which indicates that the animated depth image representation is more efficient as the viewpoint triangle grows. We chose a triangular viewpoint region for simplicity—more complex regions can be built from multiple triangles. Moreover, the adaptive sampling algorithm can be easily extended to more complex 2-D or 3-D regions (e.g. a cuboid sampled in octree fashion).

The viewpoint triangle is the set of viewpoints used by the adaptive sampling algorithm to capture all samples needed. However, the viewpoint triangle is a conservative approximation of the set of viewpoints from where the animated depth image representation has sufficient samples. Other viewpoints close to the

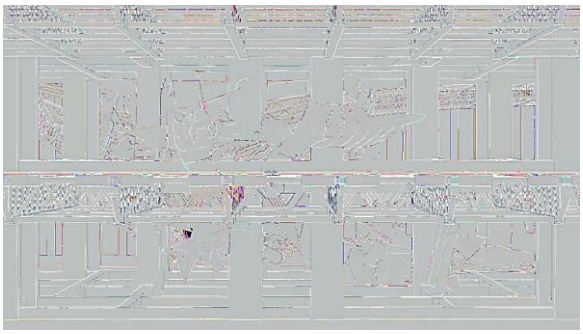

Fig. 14. Visualization of differences between a frame reconstructed from our method and the corresponding frame rendered from original dataset. The frames are shown in Figure 1, middle.

Table 4. Data size for various viewpoint triangle sizes.

| q [m] | 1 | 2 | 3 | 4 | 5 | 8 | 10 |
|---|---|---|---|---|---|---|---|
| Size [MB] | 16 | 19 | 21 | 25 | 26 | 36 | 42 |
| Rel. Size | 16 | 9 | 7 | 6 | 5 | 4.5 | 4 |
| C. F. | 17 | 15 | 15 | 14 | 14 | 13 | 13 |

triangle are likely to have sufficient samples, such as points off the triangle plane behind and in front of the center of the triangle, or points on the triangle plane just beyond the triangle. The image in Figure 15 shows the viewpoint triangle (solid orange) enlarged (orange triangular contour) and extruded (blue and yellow).

Reconstructions from 66% of the viewpoints inside the prism defined by the blue and yellow triangles have a smaller residual disocclusion error than reconstructions from the viewpoint triangle. Consequently the user can navigate the viewpoint away from the triangle viewpoint, and good reconstructions are obtained even at a considerable distance from the viewpoint triangle (compare the size of the prism to that of the viewpoint triangle in Figure 15). When the viewpoint leaves the viewpoint triangle, the user (or the system) can request a new animated depth image representation. Visualization continues using the current representation, *with good results*, until the new representation arrives from the server.

The last row of Table 4 gives the compression factor achieved by the rigid body decomposition and the compression of the trajectories of unassigned samples. The compression factor is computed by comparison to storing the trajectory of every sample uncompressed, with one position per simulation time step. Tables 1 and 2 report the variation of the size of the animated depth image representation with the trajectory approximation error threshold $\varepsilon$ and with the convergence factor $g$. For the SPH dataset, the animated depth image representation requires 49.4MB of storage space, a 41.67 compression factor over the original dataset (2.01GB) that stores each particle position for each time step.

### 8.2.2 Frame rate

Table 5 gives the average rate at which frames are reconstructed at the client from the animated depth image representation. The measurements were performed on an Intel i7 workstation with an nVidia GTX660 graphics card. Four output frame rendering modes are investigated. For *static* the simulation time step is fixed. For *dynamic* the simulation time advances from frame to frame. *PB* corresponds to a straight forward point-based reconstruction with 2x2 splats (Figure 16). *TM* corresponds to triangle mesh reconstruction. The primitives (points and triangles) are sorted in descending order based on their erosion times; this way the primitives needed at a time step are simply determined by choosing the appropriate prefix of the connectivity array without having to enable and disable individual triangles. As expected, higher frame rates are obtained for lower output resolutions, since that implies fewer samples during reconstruction, for the point-based reconstruction mode which is less expensive than the triangle mesh reconstruction, and for the static visualization mode since it does not imply updating the geometry for every frame. For all aircraft dataset experiments at the 1,280 x 270 resolution, the minimum, average and maximum frame rates are (98, 145, 212), (13, 18, 23), (16, 28, 40), and (9, 14, 20) for *Static PB, Static TM, Dynamic PB,* and *Dynamic TM* , respectively.

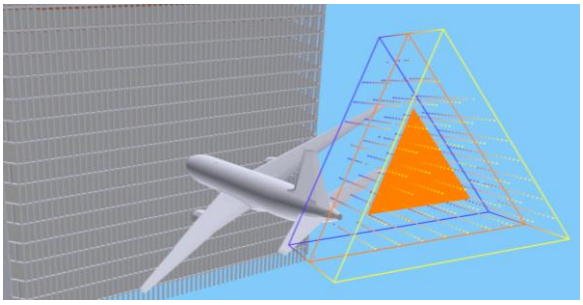

Fig. 15. Visualization of viewpoints outside of viewpoint triangle (solid orange) with conforming residual disocclusion error.

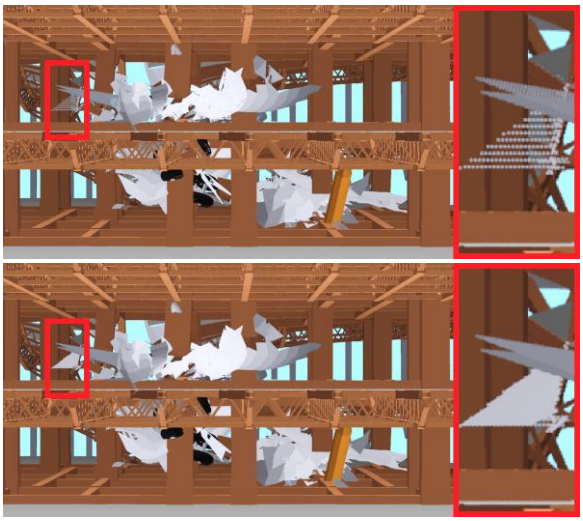

Fig. 16. Comparison between frames reconstructed using 2x2 pixel splats (*top*), and using a triangle mesh (*bottom*).

Before the client can reconstruct output visualization frames, the animated depth image representation has to be decompressed. Decompression time ranges between 0.4 and 2.5s (Table 5), which is comparable to the transmission time in the case of high bandwidth networks, and negligible in the case of low bandwidth networks, as discussed in Section 8.2.5.

Table 5. Frame rate for various visualization modes.

| Data-set | Sequence | Re-gion | Frame Resolution | Decom-pression [s] | Frame rate [fps] | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Static | | Dynamic | |
| | | | | | PB | TM | PB | TM |
| Truck | 0-80 | N/A | 1280 × 720 | 0.60 | 566 | 106 | 100 | 68 |
| Aircraft | 0-57 | out. | 1280 × 720 | 1.4 | 201 | 23 | 38 | 18 |
| | | | 960 × 640 | 0.66 | 266 | 31 | 50 | 27 |
| | | | 640 × 480 | 0.42 | 480 | 56 | 111 | 45 |
| | | side | 1280 × 720 | 1.2 | 111 | 14 | 23 | 12 |
| | | | 960 × 640 | 0.88 | 160 | 22 | 31 | 17 |
| | | | 640 × 480 | 0.48 | 311 | 46 | 60 | 33 |
| | | rev. | 1280 × 720 | 1.2 | 135 | 22 | 25 | 15 |
| | | | 960 × 640 | 0.85 | 194 | 28 | 56 | 21 |
| | | | 640 × 480 | 0.45 | 394 | 74 | 113 | 45 |
| | 58-115 | out. | 1280 × 720 | 1.40 | 212 | 21 | 38 | 20 |
| | | | 960 × 640 | 0.65 | 275 | 37 | 49 | 26 |
| | | | 640 × 480 | 0.40 | 505 | 57 | 75 | 47 |
| | | side | 1280 × 720 | 1.2 | 109 | 13 | 23 | 12 |
| | | | 960 × 640 | 0.89 | 157 | 21 | 30 | 15 |
| | | | 640 × 480 | 0.51 | 312 | 45 | 59 | 34 |
| | | rev. | 1280 × 720 | 1.8 | 119 | 16 | 22 | 12 |
| | | | 960 × 640 | 1.3 | 165 | 24 | 30 | 17 |
| | | | 640 × 480 | 0.72 | 319 | 40 | 53 | 36 |
| | 116-171 | out. | 1280 × 720 | 1.3 | 211 | 22 | 40 | 17 |
| | | | 960 × 640 | 0.55 | 298 | 36 | 54 | 28 |
| | | | 640 × 480 | 0.32 | 550 | 68 | 114 | 52 |
| | | side | 1280 × 720 | 1.1 | 112 | 16 | 24 | 12 |
| | | | 960 × 640 | 0.91 | 167 | 22 | 34 | 18 |
| | | | 640 × 480 | 0.51 | 315 | 42 | 58 | 31 |
| | | rev. | 1280 × 720 | 2.5 | 98 | 14 | 16 | 9 |
| | | | 960 × 640 | 1.9 | 137 | 18 | 22 | 12 |
| | | | 640 × 480 | 1.0 | 270 | 40 | 48 | 26 |
| SPH | 0-81 | N/A | 1280 x 720 | 0.33 | 3280 | 20 | 352 | 20 |

### 8.2.3 Scalability

Animated depth images inherit from conventional depth images the desirable property of cost independence from dataset size. Dataset size depends on two factors: extent and resolution. The animated depth image performs occlusion culling and geometry resampling to achieve cost independence from both factors. Consider an FEA simulation of an earthquake in a city with thousands of buildings. In output views that focus on one or a few buildings, the buildings not visible are culled away. In output views that show the entire city, the resolution of the animated geometry is reduced. The number of samples remains capped by the animated depth image resolution. Although the cost of a single animated depth image is capped, the adaptive sampling algorithm uses multiple animated depth images to prevent disocclusion errors. The total number of samples depends on the complexity of the occlusion patterns and on the size of the viewpoint triangle. Although, to the limit, in a dataset with an infinite number of infinitely small particles every two viewpoints gather disjoint sets of samples and thus the number of samples is infinite, for FEA datasets there is substantial sample redundancy between neighboring viewpoints and the number of samples needed for a given viewpoint triangle is bounded.

The cost of the animated depth image does depend on output image resolution. As the output image resolution grows, a quality reconstruction requires that the resolution of the animated depth images increases as well. In all our experiments the resolution of images increases as well. In all our experiments the resolution of the animated depth images equals the resolution of the output frame.

Table 6 reports the relative change in number of samples, in data size, and in reconstruction frame rate as the resolution increases from 640x480 to 1,280x720, which corresponds to an increase in number of pixels by a factor of 3. Data is provided for each sequence of the aircraft dataset and for each region (i.e. *O* for outside, *S* for side, and *R* for reverse). The number of samples never increases by a factor greater than 3. The storage size increases with the number of pixels sub-linearly, which is a strongpoint of the compression based on rigid body decomposition employed by our approach. The higher the resolution, the more coherent neighboring samples are, the more samples per rigid body, and the more effective the compression. The frame rate at higher resolution is typically higher than a third of the frame at lower resolution, which indicates good scalability.

Table 6. Relative cost increase as output image resolution changes from 640x480 to 1,280x720.

| | **Sequence** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **0-57** | | | **58-115** | | | **116-171** | | |
| | **O** | **S** | **R** | **O** | **S** | **R** | **O** | **S** | **R** |
| **Samples** | 2.35 | 2.92 | 2.97 | 2.40 | 2.90 | 2.92 | 2.52 | 2.97 | 2.82 |
| **Data size** | 2.13 | 1.96 | 2.13 | 2.17 | 2.16 | 2.24 | 2.56 | 2.19 | 2.13 |
| **Frame rate** | 0.38 | 0.43 | 0.38 | 0.29 | 0.43 | 0.49 | 0.32 | 0.37 | 0.43 |

### 8.2.4 Thin client performance estimate

The proliferation of mobile devices with quality displays and with limited graphics hardware acceleration capability presents the opportunity and demands that remote visualization solutions target such thin client scenarios.

Table 7. Visualization frame rate estimate for thin clients.

| **Device** | **Rendering Performance [Mtris/s]** | **Display Resolution [Mpix]** | **Frame rate [fps]** | |
|---|---|---|---|---|
| | | | **k = 1** | **k = 3** |
| **iPhone 5** | 133 | 0.73 | 91 | 30 |
| **Galaxy S4** | 79 | 2.1 | 19 | 6.3 |
| **Lumia 920** | 34 | 0.98 | 17 | 5.8 |
| **iPad 4** | 154 | 3.1 | 25 | 8.3 |

Table 7 gives frame rate estimates for three smartphones running the iOS, Android, and Windows operating systems, as well as for an iOS tablet. The frame rate $f$ is estimated with the formula:

$$f = T/(R * 2 * k) \quad (7)$$

where $T$ is the triangle rendering performance of the device [41], $R$ is the display resolution, 2 is the number of triangles per sample, and $k$ is the number of complete animated depth images with resolution $R$ from which the frame is reconstructed.

For the experiments reported we estimate $k$ by dividing the number of samples gathered by our adaptive sampling algorithm (and thus used to reconstruct output frames) to the number of samples in a complete animated depth image. For the experiments in Table 5, $k$ was 0.47 for the truck dataset (due to empty background pixels), and between 1.26 and 2.76 for the aircraft dataset. As shown in Table 7, even when the frame is reconstructed from the equivalent of three animated depth images (i.e. k = 3 column), an interactive frame rate can potentially be sustained by all devices. The iPhone has the best performance because of its advantageous triangle rendering performance to display resolution ratio. The iPad brings a substantial increase in display resolution that isn't backed up by a commensurate increase in triangle rendering performance.

The figures in Table 7 correspond to the *static TM* column in Table 5. Changing the triangles that are rendered for every frame as needed for the dynamic case translates to a high memory bandwidth requirement, and the frame rate will decrease as it did for the graphics card used for Table 5.

### 8.2.5 Comparison to other remote visualization approaches

The animated depth image approach enables remote visualization of dynamic datasets and produces high-quality frames that are very close to frames rendered directly from the dataset. Compared to transferring a conventional depth image, our representation requires a larger initial transfer, but then supports changing the view and advancing the simulation time at the client.

We now compare our approach to the conventional remote visualization approach of computing each frame on the server. The performance of a remote visualization system is characterized by three quantities: the startup time $t_0$, defined as the time it takes for the first frame $F_0$ to be displayed, the frame to frame latency $l$, defined as the average time elapsed from when a frame $F_i$ ($i > 0$) is requested by the user to when $F_i$ is displayed, and the total amount $D$ of data transferred for a remote visualization sequence. To estimate these quantities we have to further define the visualization context.

First, we need to define how the frame is compressed for the conventional approach. We investigate two scenarios: each frame is compressed individually using jpeg, and each frame is compressed by taking into account that previous frames have already been sent to the client. We approximate conservatively the second scenario by compiling off-line a video file for all the frames from a visualization sequence with the state of the art H.264 codec. This provides an upper bound on the compression performance that live streaming can achieve. For the aircraft dataset, the average per frame data size is 422kB for 1,280 × 720 resolution and individually compressed frames, 58kB for 1,280 × 720 resolution and streaming. For 640 × 480 resolution the same numbers are 45kB and 4kB, respectively.

Second, we need to estimate the time it takes the server to service the request from the client. We assume that the server has substantial computational resources so we consider this time as negligible.

Third, we need to estimate the ping time between the server and the client, defined as the time it takes a short message to be transferred from the client to the server and back, and the network download bandwidth, defined as the amount of data that can be transferred from the server to the client per second. Upload speed is not a concern since the request for a new frame implies small data amounts. Average ping times from our Purdue University laboratory to servers at Purdue University, at University of Illinois at Urbana-Champaign, at Columbia University, at University of North Carolina at Chapel Hill, at the University of Utah, at Stanford University, at

the Technical University of Berlin, Germany, at the Lomonosov Moscow State University, Russia, at the India Institute of Technology Delhi, India, at Tsinghua University, China, at the University of Tokyo, Japan, and at the University of Queensland, Australia are 81, 71, 57, 69, 105, 109, 162, 184, 285, 277, 222, 276ms, respectively. In our comparison we use the values of 50ms and 300ms for a short and a long ping time. The download bandwidths we measured in West Lafayette IN for 4G, 4G LTE, residential broadband, and wide area network (WAN) are 2.5, 10, 20, and 100 Mbps, respectively. In our comparison we use 1Mbps and 100Mbps for high and low bandwidth values.

Finally, we need to define a visualization sequence as the series of consecutive visualization frames requested by the user for a particular region of the dataset and for a particular interval of simulation time steps. In the case of the animated depth image approach, a visualization sequence is reconstructed from the same set of animation depth image tiles. The number of frames in such a visualization sequence depends on the dataset, on the region of the dataset, and on the time interval. We have observed the civil engineers in our project examine each of the *outside*, *side*, and *reverse* regions of the aircraft dataset for over 10 minutes, which at 30Hz implies sequences of 18,000 frames. In our comparison we assume visualization sequences of 10,240 frames.

Table 8. Comparison between conventional and animated depth image remote visualization for various network scenarios.

| Resolution | Conventional | | Animated Depth Image | | |
|---|---|---|---|---|---|
| | $l$ [ms] | $D$ [MB] | $t_0$ [s] | $l$ [ms] | $D$ [MB] |
| *Scenario A: ping 300ms, bandwidth 1Mbps* | | | | | |
| 1,280 × 720 | 603-3,447 | 580 – 4,220 | 192 | 2 – 111 | 24 |
| 600 × 480 | 300-502 | 40 – 450 | 88 | 2 – 38 | 11 |
| *Scenario B: ping 300ms, bandwidth 100Mbps* | | | | | |
| 1,280 × 720 | 300 | 580 – 4,220 | 1.92 | 2 – 111 | 24 |
| 600 × 480 | 300 | 40 – 450 | 0.88 | 2 – 38 | 11 |
| *Scenario C: ping 50ms, bandwidth 1Mbps* | | | | | |
| 1,280 × 720 | 478-3,322 | 580 – 4,220 | 192 | 2 – 111 | 24 |
| 600 × 480 | 56-377 | 40 – 450 | 88 | 2 – 38 | 11 |
| *Scenario D: ping 50ms, bandwidth 100Mbps* | | | | | |
| 1,280 × 720 | 50-58 | 580 – 4,220 | 1.92 | 2 – 111 | 24 |
| 600 × 480 | 50 | 40 – 450 | 0.88 | 2 – 38 | 11 |

Table 8 gives the performance of the animated depth image remote visualization approach for the aircraft dataset and compares it to that of conventional remote visualization. Four scenarios are investigated: long ping time and low bandwidth (A), long ping time and high bandwidth (B), short ping time and low bandwidth (C), and finally short ping time and high bandwidth (D). For each scenario two output resolutions are investigated. For the animated depth image approach, the data size is estimated by averaging the representation size over the three regions *outside*, *side*, and *reverse*, for each resolution (resulting in the values of 24MB and 11MB). The startup time $t_0$ is computed by dividing the data size to the bandwidth to obtain the 192s and 88s values. The frame to frame latency $l$ is computed by inverting the reconstruction frame rate, and it is given as a range, using the fastest and slowest frame rates given in Table 5 for the same resolution, and over all reconstruction modes (i.e. 566fps and 9fps for 1,280 x 720, and 550fps and 26fps for 640 x 480). $l$ does not depend on the network parameters (i.e. ping and bandwidth). In fact the visualization can continue at the client even if the connection to the server is lost after the initial transfer.

For the conventional approach, the total amount of data transferred $D$ is obtained by multiplying the average frame size by the number of frames in the sequence (i.e. 10,240). A frame size range is used, from H.264 sequence compression to compression of individual frames, as discussed above. The conventional approach transfers substantially more data. The breakeven points are 424 and 58 frames for 1,280 × 720, and 2,816 and 250 frames for 640 × 480.

For the conventional approach, the time for the first frame is the same as for any other frame, thus $t_0 = l$. We estimate $l$ as follows:

$$l = \max(t_{ping}, \frac{t_{ping}}{2} + \frac{f}{b}) \quad (8)$$

where $t_{ping}$ is the ping time, $f$ is the size of the frame, and $b$ is the bandwidth. If $b$ is sufficiently large for the network to transport a frame in half the ping time, $l$ is given by $t_{ping}$. In scenario A, the advantage of the animated depth image (ADI) approach over the conventional remote visualization (CRV) approach is substantial, for both resolutions, and even when the highest quality reconstruction is used for ADI and the most aggressive frame compression is used for CRV. In scenario B, the high bandwidth reduces $l$ for CRV to ping time, which still exceeds even the highest quality reconstruction time for ADI. In scenario C, ADI has substantial advantage for the 1,280 × 720 resolution. In scenario D, which corresponds to a very highly performing network, ADI has an advantage only for the faster reconstruction modes (i.e. *PB* static and dynamic, see Table 5). For all scenarios, the fastest reconstruction (i.e. 2ms) gives at least a 25 fold advantage for ADI over CRV.

We conclude that, compared to the conventional remote visualization approach of sending each frame from the server to the client, the animated depth image approach improves frame to frame latency in all but in the case of a very highly performing network, and the advantage increases with output frame resolution. Moreover, the frame to frame latency does not depend on the network condition. These advantages come at the cost of a longer startup time. The conventional approach will always be limited by the ping time, a network characteristic which whose improvement is challenging and costly.

### 8.3 Limitations

One of the limitations of animated depth images as of all hybrid approaches to remote visualization is the large startup time. Progressive refinement schemes such as transferring a lower resolution animated depth image for the reference view could help alleviate this problem. Another approach is to further reduce the size of the animated depth image, for example by compressing the color and depth maps. The trajectories of the unassigned samples currently take up to 50% of the overall storage requirement, so further improving the compression of those trajectories will translate in sizeable storage gains. Although the amount of residual occlusion errors is small even for the complex occlusion patterns in the dataset regions explored, the adaptive sampling algorithm proceeds nonetheless in greedy fashion. A global optimization approach could be developed to bound residual disocclusion errors.

A second limitation of animated depth images is that they do not support volume rendering. This limitation is inherited from conventional depth images which can model opaque surfaces by capturing the first surface sample seen along a ray, but cannot model transparency. This does not mean that animated depth images cannot be used to visualize opacity data. Visualizations of opacity data often take the first step of computing a surface of interest (e.g. isosurface) which can then be remotely visualized with our method. Volume rendering is just one example of the more general challenge brought to sample-based rendering by view dependent effects. Another example is rendering reflections. One option is to render reflections at the client, rendering capability permitting. We will also investigate the extension of animated depth images to store view-dependent color in a compressed form, leveraging the fact that color variability is limited by the targeted range of reconstruction viewpoints.

Finally, the rigid body decomposition of the set of samples stored by an animated depth image is done non-optimally in the interest of performance. The heuristic used is based on the reasonable assumption that samples whose motion is well approximated by a rigid body transformation are also samples that are close to each other in model space and thus in image space.

## 9 Conclusions and Future Work

We have described animated depth images, a novel type of image that not only stores color and depth samples but also stores sample trajectories. An animated depth image covers a time interval as opposed to the single time point covered by conventional depth images. The difference between an animated depth image and a set of conventional depth images comes from the fact that the trajectories of the samples are stored in a compact way that leverages sample trajectory coherence. The approximation is efficient: tight user-selected error bounds are met while achieving considerable storage savings. The approach does not rely on sample trajectory simplicity, but rather on similarity of trajectories of nearby samples. The approach uses one rigid body transformation per time step which allows modeling complex trajectories with little or no time step to time step coherence, as those arising in the impact and dam break simulations considered in this paper. As we have shown, despite the complexity of the motion in these simulations, trajectories do exhibit sample to sample coherence. As the spatial resolution of simulations continues to increase, so will the sample to sample coherence and thus the efficacy of our approach.

Compared to a video segment, an animated depth image affords interactivity. We have demonstrated the benefits of animated depth images in the context of remote visualization of FEA datasets, which exhibit complex occlusion patterns. We have shown that the approach can be extended to SPH simulation datasets. Since animated depth images are a general approximation of animated geometry, we anticipate that the approach can be extended to other representations. Like for any type of image, animated depth image size is independent of dataset size and their relative benefit increases with dataset size.

In computer graphics applications such as, for example, urban simulation or games, most of the dynamic scenes used are static where most of the moving triangles are part of explicitly defined rigid bodies (e.g. cars through the city) whose trajectories are encoded collectively with a single sequence or tree of transformations, and the motion of most non-rigidly moving triangles is governed by the motion of underlying skeletons (e.g. computer animation characters). This allows animating the triangles economically by skinning the skeleton for each new configuration. Animated depth images are not suitable for such scenes. Animated depth images are specifically designed for datasets where millions of simulation nodes move independently resulting in millions of triangles with small screen footprint whose vertices move independently from frame to frame. Such datasets are common in scientific visualization and they cannot be handled with the conventional scene graph, transformation hierarchy, and skeleton conceptualizations of motion used in computer graphics.

One direction for future work is incorporating animated depth images into an actual remote visualization system, with one or a few powerful servers, and with many thin clients. Output frame reconstruction is equivalent to straight forward triangle rendering, and even smartphones can render meshes with a number of triangles commensurate to their screen resolution, so we do not foresee problems with rendering performance at the client. One of the potential advantages of our hybrid method is to decrease the frequency of requests from the client to the server, compared to conventional remote visualization where each client makes a request for each frame displayed. However, the request for a new set of animation depth image tiles is more complex than simply rendering a single frame. In order to minimize the response time for such requests, we will investigate parallelizing the computation of an animated depth image and the adaptive space/time sampling over the set of processors available at the server. Prior work has accelerated the computation on the server side using a light field [39]—the approach could be adapted to pre-compute all animated depth images possibly needed by clients by tiling the viewing space.

Another direction for future work is to address other types of visualization, such as volume rendering, computation that is harder to cache in sample sets [42]. Finally, animated depth images introduce sampling robustness with respect to variations of the time parameter of time-varying datasets. In the future we will investigate other image generalizations that bring robustness with variations of other visualization parameters such as, for example, an isovalue used in isosurface extraction.

## 10 Acknowledgements

We would like to thank Paul Rosen for his help with the implementation and the anonymous reviewers for their help with improving this manuscript. We would also like to thank National High Technology Research and Development Program of China through 863 Program NO.2013AA01A604, National Natural Science Foundation of China through Projects 61190121 and 61272349 which supported the study of Zhiqiang.

## References

[1] G. Klimeck, M. McLennan, S. P. Brophy, G. B. Adams III, M. S. Lundstrom. nanoHUB.org: Advancing Education and Research in Nanotech. Computing in Sci. and Engineering, 10(5), pp. 17-23, 2008.

[2] S. Stegmaier, M. Magallón, and T. Ertl. A Generic Solution for Hardware-Accelerated Remote Visualization. In EG/IEEE TCVG Symp. on Data Visualization'02, pp. 87-94, 2002.

[3] F. Lamberti and A. Sanna. A Streaming-Based Solution for Remote Vis. of 3-D Graphics on Mobile Devices. IEEE TVCG, pp. 247-260, 2007

[4] Silicon Graphics, Inc. OpenGL Vizserver 3.0—ApplicationTransparent Remote Interactive Vis. and Collaboration, 2003.

[5] S. P. Callahan, J. L. D. Comba, P. Shirley, and C. T. Silva. Interactive Rendering of Large Unstructured Grids using Dynamic Level-of-Detail. IEEE Vis.'05, pp. 199-206, 2005.

[6] D. Luebke, M. Reddy, J. Cohen, A. Varshney, et al. Level of Detail for 3-D Graphics. Morgan-Kaufmann Publishers, 2002.

[7] Y. Livnat, S.G. Parker, C.R. Johnson. Fast Isosurface Extraction Methods for Large Image Data Sets. In Handbook of Medical Imaging, Academic Press, pp. 731--745, 2000.

[8] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann. Topology-Based Simplification for Feature Extraction from 3-D Scalar Fields. IEEE Vis. '05, pp. 272-280, 2005.

[9] S. P. Callahan, L. Bavoil, V. Pascucci, and C. T. Silva. Progressive Volume Rendering of Large Unstructured Grids. IEEE TVCG, pp. 1307-1314, 2006.

[10] H. Hoppe. Progressive Meshes. ACM SIGG., pp. 99-108, 1996

[11] S. Pesco, P. Lindstrom, V. Pascucci, and C. Silva. Implicit Occluders. In IEEE/SIGGRAPH Symp. on Volume Visualization, pp. 47-54, 2004.

[12] J. Gao and H.-W. Shen. Parallel View-Dependent Isosurface Extraction Using Multi-Pass Occlusion Culling. In IEEE Symp. on Parallel and Large Data Vis. and Graphics, pp. 67-74, 2001.

[13] L. Lippert, M. H. Gross, and C. Kurmann. Compression Domain Volume Rendering for Distributed Environments. Computer Graphics Forum, 16(3):C95-C107, 1997.

[14] M. Isenburg, P. Lindstrom, and J. Snoeyink. Streaming Compression of Triangle Meshes. Symp. on Geometry Processing, pp. 111-118, 2005.

[15] Paul Rosen and Voicu Popescu, Simplification of Node Position Data for Interactive Visualization of Dynamic Datasets, IEEE Transactions on Visualization and Computer Graphics, 2011

[16] K.-L. Ma and D. M. Camp. High Performance Visualization of Time-Varying Volume Data over a Wide-Area Network. In SC'00.

[17] Anna Tikhonova, Hongfeng Yu, Carlos D. Correa, Jacqueline H. Chen, Kwan-Liu Ma. A Preview and Exploratory Technique for Large-Scale Scientific Simulations. Eurographics Workshop on Parallel Graphics and Visualization (EGPGV) , 2011:111-120

[18] S. Stegmaier, J. Diepstraten, M. Weiler, and T. Ertl. Widening the Remote Visualization Bottleneck. IEEE ISPA, pp. 1-6, 2003.

[19] E. J. Luke, C. D. Hansen. Semotus Visum: a Flexible Remote Visualization Network. In Proceedings of the IEEE Conference on Visualization '02, pp. 61-68, 2002.

[20] W. Bethel, B. Tierney, J. Lee, D. Gunter, and S. Lau. Using High-Speed WANs and Network Data Caches to enable Remote and Distributed Visualization. In SC '00, Article No. 28, 2000.
[21] V. Popescu, P. Rosen, L. Arns, X. Tricoche, C. Wyman, C. Hoffmann, "The General Pinhole Camera: Effective and Efficient Non-Uniform Sampling for Visualization", IEEE Transactions on Visualization and Computer Graphics, 2010:777-790.
[22] L. Mcmillan, G. Bishop, Plenoptic modeling: An image based rendering system. In proc. SIGGRAPH '95, 39-46.
[23] W. Mark., L. MCMILLAN, G BISHOP. Post-Rendering 3D Warping. In proc. of 1997 Symposium on Interactive 3D Graphics (Providence, Rhode Island, April 27-30, 1997).
[24] V. Popescu, et al. The WarpEngine: An Architecture for the PostPolygonal Age. In proc. of SIGGRAPH 2000.
[25] N Max, K Ohsaki. Rendering trees from recomputed zbuffer views. In Rendering Techniques '95: Proc. of the Eurographics Rendering Workshop, 45–54, June 1995.
[26] J. Shade et al. Layered Depth Images, In proc. of SIGGRAPH 98, 231-242.
[27] C. F. Chang, G. Bishop., A Lastra. LDI Tree: A Hierarchical Representation for Image-Based Rendering. In proc. of SIGGRAPH'99.
[28] P Rademacher, G. Bishop. Multiple-center-of-Projection Images. In proc of SIGGRAPH '98, 199–206.
[29] C. Mei, V. Popescu, E Sacks. The Occlusion Camera. In proc. of Eurographics 2005, Computer Graphics Forum, vol. 24, issue 3, sept 2005.
[30] J. Yu, L. McMillan. General Linear Cameras. *In Proc of the European Conference on Computer Vision (ECCV)*, Vol. 2, pp. 14-27, 2004.
[31] V. Popescu, P. Rosen, N. Adamo-Villani. The Graph Camera. International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH Asia, 2009.
[32] U. Ramer, An iterative procedure for the polygonal approximation of plane curves, *Computer Graphics and Image Processing*, vol. 1 no. 3, pp. 244 – 256, 1972.
[33] D. Douglas and T. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.
[34] S. Rusinkiewicz and M. Levoy, QSplat: a multiresolution point rendering system for large meshes, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, p.343-352, July 2000
[35] H. Pfister , M. Zwicker , J. van Baar , M. Gross, Surfels: surface elements as rendering primitives, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, p.335-342, July .
[36] V. Popescu and P. Rosen, Forward Rasterization. ACM Transactions on Computer Graphics, 25(2), April 2006.
[37] A. Tikhonova, C. Correa, and K. Ma, Visualization by Proxy: A Novel Framework for Deferred Interaction with Volume Data, IEEE TVCG, VOL. 16, NO. 6, 2010
[38] A. Tikhonova, C. D. Correa and K.-L. Ma, An Exploratory Technique for Coherent Visualization of Time-varying Volume Data, Eurographics/ IEEE-VGTC Symposium on Visualization 2010, Volume 29 (2010), Number 3.
[39] Al-Saidi, A., Walker, D. W., & Rana, O. F. (2012). On-demand transmission model for remote visualization using image-based rendering. Concurrency Computation Practice and Experience.
[40] Paravati G; Celozzi C; Sanna A; Lamberti F. (2010) A Feedback-Based Control Technique for Interactive Live Streaming Systems to Mobile Devices. In: IEEE Transactions on Consumer Electronics, vol. 56:1, pp. 190-197. - ISSN 0098-3063.
[41] GFXBench. A unified 3D graphics performance benchmark suite. http://gfxbench.com.
[42] Lalgudi, H. G., Marcellin, M. W., Bilgin, A., Oh, H., & Nadar, M. S. (2009). View compensated compression of volume rendered images for remote visualization. IEEE Transactions on Image Processing, 18(7), 1501-1511.
[43] Pazzi, R. W. N., Boukerche, A., & Huang, T. (2008). Implementation, measurement, and analysis of an image-based virtual environment streaming protocol for wireless mobile devices. IEEE Transactions on Instrumentation and Measurement, 57(9), 1894-1907.
[44] Simoens, P., De Turck, F., Dhoedt, B., & Demeester, P. (2011). Remote display solutions for mobile cloud computing. Computer, 44(8), 46-53.
[45] Hutanu, A., Allen, G., & Kosar, T. (2010). High-performance remote data access for remote visualization. Paper presented at the Proceedings - IEEE/ACM International Workshop on Grid Computing, 121-128.
[46] Video accompanying this paper submission. http://www.cs.purdue.edu/cgvlab/popescu/remotevis/

**Jian Cui** received his BS in computer science from the Harbin Institute of Technology, China in 2009. He is a PhD candidate in computer science at Purdue University. His research interests span computer graphics and computer vision. His current work focuses on image generalization through camera model design to overcome the single viewpoint and uniform sampling rate limitations of conventional images.

**Zhiqiang Ma** is a Ph. D. candidate at the State Key Laboratory of Virtual Reality Technology and Systems of Beihang University. His research interests include remote visualization, global illumination and real time rendering

**Voicu Popescu** received the BS degree in computer science from the Technical University of Cluj-Napoca, Romania in 1995, and the PhD degree in computer science from the University of North Carolina at Chapel Hill in 2001. He is an associate professor with the Computer Science Department of Purdue University. His research interests lie in the areas of computer graphics, computer vision, and visualization. His current projects include camera model design, remote visualization, aggressive and exact visibility computation, and applications of computer graphics in education.