Luggage Visualization and Virtual Unpacking

Wei Li*[1], Gianluca Paladini†[1], Leo Grady‡[2], Timo Kohlberger§[1], Vivek Kumar Singh¶[1], and Claus Bahlmann‖[1]

[1]Siemens Corporation, Corporate Technology
[2]HeartFlow, Inc.

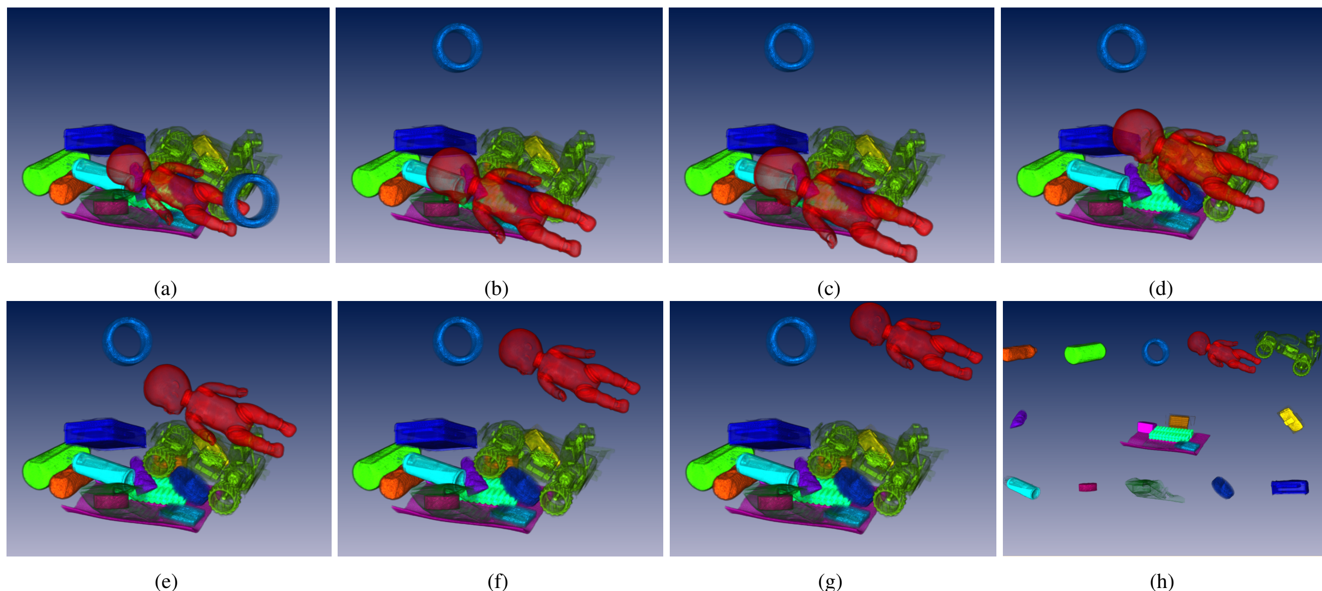


Figure 1: *Virtual unpacking: (a) all objects in packed positions; (b)-(g) images extracted from the animation of unpacking the red doll, which first moves towards the camera, then it flies to the top-right corner of the image; and (e) most objects are in unpacked positions.*

Abstract

We present a system for luggage visualization where any object is clearly distinguishable from its neighbors. It supports virtual unpacking by visually moving any object away from its original pose. To achieve these, we first apply a volume segmentation guided by a confidence measure that recursively splits connected regions until semantically meaningful objects are obtained, and a label volume whose voxels specifying the object IDs is generated. The original luggage dataset and the label volume are visualized by volume rendering. Through an automatic coloring algorithm, any pair of objects whose projections are adjacent in an image are assigned distinct hues that are modulated onto a transfer function to both reduce rendering cost as well as to improve the smoothness across object boundaries. We have designed a layered framework to efficiently render a scene mixed with packed luggage, animated unpacking objects, and already unpacked objects put aside for further inspection. The system uses GPU to quickly select unpackable objects that are not blocked by others to make the unpacking plausible.

1 Introduction

Luggage screening is unavoidable for either checked-in luggage or carried-on baggages in order to detect threaten and forbidden items, which is especially true for aviation security. The most trustable way is still physically opening the luggages and unpacking their contents. This unfortunately is very inefficient and intrusive. With modern scanning technology, such as computed tomography (CT), it is possible to obtain detailed information of luggages that are sufficient to reconstruct accurate geometric properties of the contained objects. Naturally, a luggage is packed, which means that many objects are in close contact with each other and appear connected in a scanned image. It usually involves human interaction to isolate individual objects for better inspection.

In this paper, we describe a visualization work flow to assist luggage screening. First, a volumetric representation of a luggage is segmented into meaningful real-world objects. Then the segmented objects are assigned distinguishable colors depending on the viewing parameters. The original luggage volume, the segmentation results, and the automatically assigned colors are fed into a volume rendering system. For any user defined unpacking direction, unpackable objects are determined and unpacking is visualized by showing that the transition of the objects flying from its original position to a unpacking destination. Unpacked objects can be manipulated individually, such as rotating, changing transfer function, etc, for inspection. The system keeps packed luggage, already unpacked objects, and the objects in transition in the same scene and allows repacking. Figure 1 illustrates the main capabilities of the system, where Figure 1(a) is a volume rendition of a segmented CT scan of luggage. By comparing Figure 1 (a)-(h), we can observe how the red doll first moves towards the eye, then flies to the top-right corner of the image. Figure 1(h) presents a scene where many objects are put at their unpacked positions. Please see the accompany videos that better visualizes the unpacking transition.

*li.wei@siemens.com
†gianluca.paladini@siemens.com
‡leograd@yahoo.com
§timo.kohlberger@siemens.com
¶vivek-singh@siemens.com
‖claus.bahlmann@siemens.com

For luggage segmentation, our system utilizes a state-of-the-art graph partitioning approach[Grady and Schwartz 2006] enhanced by a confidence measure of the quality of a segmentation. The confidence measure gives high scores to semantic meaningful real-world objects that are homogeneous and have well-defined boundaries against the surrounding segments, and it guides the segmentation algorithm to recursively split connected objects in an input volume until the segmentation quality can't be improved any more.

In order to assign distinct colors to objects appearing adjacent in an image, our system first generates a layered image of object IDs, from which whether any pair of objects are interfering each other is determined. A modified graph coloring algorithm is then performed to ensure any interfering pair are assigned distinguishable hues while the hue space is fully exploited. In order to reduce volume rendering cost as well as to keep smooth transitions across touching objects, we adopt a tinting approach, in which the assigned hues of an object is modulated onto a master transfer function.

To virtually unpack a luggage, instead of physically partitioning the original volume, we perform volume rendering in multiple passes and each pass is associated with the proper transfer function and geometric transformation. A brutal-force design requires the number of rendering passes equals to that of the objects in the worst case. This obviously is too expensive considering a typical luggage contains several dozens of objects. To efficiently handle a scene mixed with a volume of packed objects, unpacked objects, and the animation of objects being unpacked or restored, we have designed a layered approach. Each layer is responsible for rendering objects in a particular state as well as caches and displays the previous rendered images to avoid unnecessary rendering. The layer of unpacked objects is further decomposed into a set of 3D sprites to minimize memory usage.

Here are the main contributions of the paper:

- We propose the concept of virtual unpacking, which according to our knowledge has not been presented in the literatures.
- We design the tinting approach which is very efficient for rendering segmented volumes containing numerous labels, as well as avoiding unsmoothness across object boundaries.
- We customize the graph coloring algorithm to automatically assign colors to conflicting objects while fully utilizing the color space.
- We present two GPU accelerated methods for determining unpackable objects that are not blocked by others.
- We construct a layered rendering framework to efficiently visualize the unpacking.

In the remainder of this paper, we first review related work in section 2; then we describe luggage segmentation in section 3. Next, we present luggage visualization and virtual unpacking in sections 4 and 5 respectively, followed by implementation details and results in section 6. Finally, we conclude the paper with discussion and future work (7).

2 Related work

2.1 Transfer function design

Our system automatically assigns colors to different objects, which can be considered as a simplified version of automatic transfer function generation, which has been a popular topic and has attracted numerous publications, e.g. [Zhou and Takatsuka 2009][Chan et al. 2009]. Most of these works focus on adjusting the opacity of the transfer function so that most, if not all, the layered structures in a volume are visible. This is more or less a balancing of the transparencies of the structures. Adjusting transparency is also a common way of shifting highlights among objects. In luggage visualization, we rely on unpacking to reveal occluded parts and would like each object to have consistent appearance. Besides, without additional information, each object is considered equally important. We don't want to artificially distract an observer's attention by highlighting any object. Therefore, our system is based on a user customizable master transfer function, and just adjusts the hues of different objects. Some previous work of transfer function design also have an emphasis on aesthetics, such as assigning harmonic colors. In contrast, we focus on the distinguishability of the shape of an object from its background, and prefer to assign distinct hues to neighboring objects.

2.2 Exploded views

Virtual unpacking can be considered as a variant to the generation of exploded views of volumetric[Bruckner and Gröller 2006] or geometric[Li et al. 2008] datasets, in which an object is partitioned and displaced to reveal hidden details. The way that an object is splitted is based on a user's input or follows an assembly sequence. Each exploded part is either a portion of an organic object or designed to be assembled into a model of multiple parts. In exploded views, the context of each partition is extremely important and should be carefully preserved, and its explosion path is highly restricted. In contrast, for a packed luggage, there is usually no meaningful correlation between objects, and we have much more freedom in relocating them. Therefore our system prioritizes removing occlusion over keeping context, although it reserves the capability of restoring any unpacked object to its original position to deal with possible inaccuracy in a segmentation.

2.3 Visibility sorting and collision detection

Our unpackable object determination has certain similarity to GPU accelerated visibility sorting [Govindaraju et al. 2005], [Callahan et al. 2005] and collision detection [Govindaraju et al. 2003] in that we both utilize GPU to resolve the spatial relationship among primitives. In principle, an object is unpackable in a certain direction if it is fully visible in an orthogonal projection when the viewing direction is exactly the opposite of that of the unpacking. Naively, such a visibility order can be used as the unpacking order for a give direction. For our case, the unpacking direction can be frequently changed according to an inspector's interest. Therefore our algorithm is customized to only search for the objects taking the first place in the order. We also use occlusion ratio to deal with the situation that non object is fully unblocked. Similar to the mentioned references, our multi-pass method uses occlusion queries to retrieve the results of depth comparison of the rasterized primitives. But we also propose a single pass algorithm that does not require occlusion query.

3 Luggage Segmentation

The foundation of our approach for partitioning a luggage volume into meaningful real-world objects is isoperimetric graph partitioning[Grady and Schwartz 2006]. On top of that, we utilize an extension[Grady et al. 2012] that computes the confidence of the quality of a volume segmentation. Our focus in this paper is luggage visualization. Therefore, we only outline the major steps of the luggage segmentation approach we adopted. Please see [Grady et al. 2012] for more details.

To obtain a confidence measure, the algorithm annotates a large

number of good segments, that are homogeneous and have well-defined boundaries with the surrounding segments. The annotated segments are then fed into a model learning pipeline to train a confidence measure. After annotation, various features are computed, that include geometric properties (such as surface smoothness, curvature volume) and appearance properties based on density distribution (such as average L1 gradient, average L2 gradient, median and mean intensities). The approach also uses boundary based features such as the total cost of the cuts for isolating this segment from neighboring segments. All the features are invariant to rotation and translation. Each feature is further normalized to make it invariant to scale and object types. A statistical normalization is also performed by first computing the mean and the standard deviation for all the features over all segments followed by normalizing the feature scores by subtracting the mean and dividing by the standard deviation. Next, the method uses Principal Component Analysis (PCA) to reduce the dimensionality of the data. In the final stage, a Mixture of Gaussian model is fit over the PCA coefficients of all the segments in the training set to approximate the distribution of the good segments in the feature space. Once having a confidence measure, for any given luggage segmentation, the approach computes the PCA coefficients vector from the segmentation's normalized feature vector. Then a confidence score is obtained by calculating the likelihood of the coefficient vector using the Mixture of Gaussian stored in the segment oracle.

To perform luggage segmentation, first, a generic segmentation algorithm [Grady and Alvino] provides an initial segmentation that roughly separates all the target objects inside the luggage from the background voxels. A subsequent connected component analysis then assigns not-connected foreground segments different labels. Since most of them are still strong under-segmented, i.e. cover groups of target objects, a recursive splitting algorithm is run for each of those individual foreground segments. For every under-segmented region, the isoperimetric algorithm generates several different plausible binary separations. The confidence measure then is used in conjunction with the isoperimetric ratio to decide if any of the proposed splits generates sub-groups of objects that are sufficiently close to individual objects.

4 Luggage Visualization

4.1 Volume rendering with tinted VOIs

Volume rendering combined with masks specifying volume-of-interests (VOI) have been extensively deployed in volume visualization. These VOIs are assigned optical properties different than the rest of the volume, so that they are clearly distinguishable. Typically, each VOI is associated with a full color lookup table[Hadwiger et al. 2003], which provides the most flexibility in adjusting the appearance of the VOI.

It is a challenging task to figure out the true color of each segmented object from a gray volume. Note that an artificial object may be built with arbitrary optical properties, which is different than medical visualization. In our case, the most important requirement is that the shape of each individual object is clearly recognizable, whereas it is acceptable if the object is rendered with colors and opacities different from its true appearance. Sometimes, it is even undesirable to assign an object its true color. For example, if two objects of similar colors overlap in an image, we would like them to be rendered with contrast colors.

The segmentation masks generated in section 3 are non-overlapping binary volumes. That is, each voxel of the original gray volume can only belong to a single VOI, and all sub-voxel boundaries are rounded to the nearest voxel borders. All the binary volumes are combined to form a label volume with each of its voxel storing the ID (≥ 1) of the corresponding binary volume or a zero for background. Obviously, it is meaningless to interpolate between different IDs. Therefore the label volume should be sampled with the nearest neighbor method.

We rely on interpolated samples from the original density volume and proper opacity assignment in the transfer function to hide the unsmoothness of the boundaries of the binary masks. This requires the opacity of the transfer function to be C0 continuous. Obviously, there is no guarantee of such a property if each VOI is associated with an independent color lookup table as in[Hadwiger et al. 2003]. Consequently, we choose the following tinting approach for the luggage visualization.

- A master transfer function with C0 continuous opacities is assigned to the whole volume. The colors of the transfer function are preferably but not necessarily gray.
- Each visible VOI is assigned a tinting color that is multiplied onto the color obtained from the master transfer function for the samples falling into the VOI.
- If a VOI is set to be hidden, the corresponding tinting color and opacity are set to $(0, 0, 0, 0)$.

Note that hiding a VOI can introduce opacity discontinuity in the transfer function. Hence the blockiness of the binary masks can be noticeable if two objects have touched boundary of significant area and one of them is hidden. Fortunately, this situation is rare even in a tightly packed luggage. The opacities and colors of the transfer function can be changed dynamically highlight interested objects while keeps the tinting unchanged.

4.2 Color optimization

A typical luggage contains dozens of objects of various sizes. There are many overlaps of the objects in a projected 2D image. We would like to utilize color assignment to make overlapped objects visually separable. A natural choice is to use as many colors as the number of objects, and assign the colors either sequentially or randomly. However, it is difficult to distinguish two different but similar colors. The situation is made worse by rendering effects, such as alpha-blending, shading, and transparency.

If two objects are desired to have distinguishable colors, we declare them as mutually interfering. We adopt graph coloring in which each object is represented as a vertex, and any two interfering objects are connected by an edge. If the dataset can be viewed from an arbitrary angle, any two objects can potentially interfere. Therefore, it is rational to perform view-dependent color assignment.

We modify standard volume rendering to generate object ID images in which each volume sample belonging to a visible object outputs the object ID as its color, and all the samples are mapped with full opacity. Figure 2 shows as examples two object ID maps that are used for the scenes shown in Figure 1 and Figure 6 respectively. Note that the true object ID maps contain the indices of objects. Here we visualize the object IDs with their tinting colors. Each pixel of the resulting image contains the ID of the nearest object covering the pixel or the pixel is zero if no object projects onto it. Similar to depth peeling[Everitt 2001][Nagy and Klein 2003], we can generate multiple layers of these object IDs, by assigning zero opacity to all the objects presented in the previous layers and repeating the rendering until enough layers are generated.

We declare two objects, A and B, are interfering, if the following

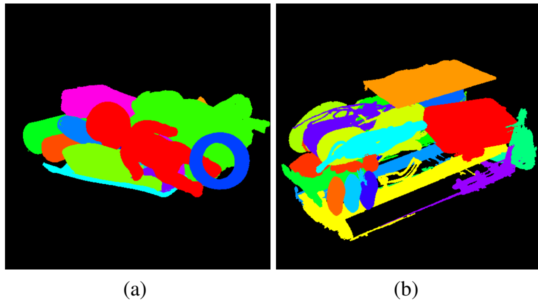


Figure 2: *The first layer of object ID maps for (a) the luggage scene in Figure1 and (b) the scene in Figure 6(b) respectively. Note that the true object ID maps contain the indices of objects. Here we visualize the object IDs with their tinting colors.*

.

formula is true:

$$\min_{layer(pA)\leq LT \wedge layer(pB)\leq LT}(dist(pA, pB)) \leq DT \quad (1)$$

where pA and pB are pixels in an image projected from the front faces of objects A and B respectively. $dist(pA, pB)$ computes the distance between the two pixels in image space. DT is a predetermined distance threshold. $layer(p)$ returns the layer index of pixel p. LT restricts the computation to the first LT layers.

Obviously, if all objects are opaque, we only need the first layer. For luggage visualization, objects are typically semi-transparent while having sufficient opacities. In practice, only samples from the first two layers present useful information, and a single-layer object ID image works pretty well for most of the use cases.

We generate an interference map from the object ID images through a 2D filtering where formula 1 is evaluated. The output is a 2D table of which each row maps to an object ID. So is each column. Initially, every cell of the table is cleared with the value $false$. If two objects interfere, the corresponding cell is set to $true$. We can read back the object ID images from the frame buffer into system memory and perform the interference detection in CPU. Alternatively, if scattering is supported, it can also be done on the GPU and avoids the read back. In our implementation, we utilize NV_shader_buffer_load OpenGL extension to allow the writing to the object interference table from the fragment shader. Other APIs such as Cuda and OpenCL can do the same. One thing to notice is that the table only have boolean values and any cell changes at most once from $false$ to $true$ during the detection. It does not matter if an entry is set to true multiple times. As a result, there is no need to use any lock to synchronize reading or writing to the table.

Once the edges of interfering objects are determined, any graph coloring algorithm can be utilized to determine the minimal number of colors, C. We choose the greedy algorithm that processes vertices in decreasing order of degrees. One important difference between our color optimization and standard graph coloring is that we intend to fully utilize the color space, instead of minimizing the number of colors. Therefore, we divide all the objects into C clusters according to the output of the graph coloring. Then each cluster is assigned one of the evenly partitioned regions in the color space. To make the tinting approach more effective, the color optimization is performed in HSV space, and only a hue (H) is assigned with S and V stay at their maximal values. This reduces the color space to an 1D circle. Next, the objects in a cluster is sorted according to the number of interfering neighbors in the next cluster. The objects are then evenly distributed in the color region assigned to the cluster, with the object having the most interferences furtherest away from the next cluster. Figure 3 illustrates such an assignment where all the objects are grouped into six clusters with different number of elements. Each object is assigned different hue with interfering objects separated to different clusters.

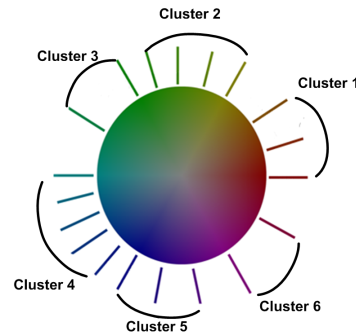


Figure 3: *Illustration of color assignment.*

It is still possible that two interfering objects assigned to different clusters are adjacent in the hue space. One option is to add additional hue margin between clusters. However, with the dataset we tested, this does not present noticeable problem.

5 Virtual unpacking

The goal of virtual unpacking is to visually separate individual objects without actually modifying the original volume. A typical work flow is to unpack objects one-by-one or group-by-group. The occlusion of the remaining objects reduces as the number of unpacked objects increases. Unpacked objects stay visible in the scene reminding users of the history, and can be individually inspected with standard volume rendering operations.

It is desirable to visualize the transition from packed to unpacked positions, so that the context of any object can be examined, and the impact of potential false segmentation is minimized. For a similar reason, our system also provides animated restoring if a user wants to repeat inspecting an unpacked object in its original environment.

5.1 Unpackable object determination

Only unblocked object can be unpacked. Otherwise, an object being unpacked may pass through others. To determine whether an object is unpackable along a certain direction $\vec{D}$, we first generate an object ID image I_{oid} use the opposite direction of $\vec{D}$ as the viewing direction. Then we perform another pass of modified volume rendering using the same viewing direction. Both of the two rendering passes use orthogonal projection. The goal of the second pass is to compute the occlusion ratio of each object A according formula 2:

$$\frac{\sum_{sA\in A \wedge A\neq objectId(proj(sA))}(1)}{\sum_{sA\in A}(1)} \quad (2)$$

where sA is a volume sample enclosed by A, and $objectId(proj(sA))$ is the object ID in I_{oid} where sA is projected onto the image plane. Basically, formula 2 computes the ratio of occluded samples versus the total samples of object A. Obviously, a zero ratio indicates that A is unoccluded. Because each object can have arbitrary shape, it is possible that every object

is blocked by others for a given direction. In the extreme, one object may be blocked in all the directions. When there is no object having zero occlusion ratio, the system either asks users to choose a different unpacking direction or select the object with the least occlusion ratio as unpackable.

One unpacking option is to use the up direction when a suitcase lies on its largest surface as the $\vec{D}$, which emulates the scenario of taking object out from an opened luggage. Another choice is to use the opposite of the viewing direction as the unpacking direction, which is roughly equivalent to unpacking the front-most object in the current view. Although it is different than an actual unpacking, this choice could be more useful for virtual unpacking. For example, a user may rotate a suitcase so that an interesting object is rotated to the front or at least with occlusion reduced. It is natural that the interested object be unpacked within the next few steps. As discussed in section 5.3, unpacking from the viewing direction also simplifies rendering. In this paper, we restrict our implementation to only perform unpacking in this direction.

The occlusion ratio table of all the objects that are still packed is updated whenever the unpacking direction is changed or when all the unpackable objects determined in the previous update are unpacked. We utilize GPU to accelerate the process. Depending on hardware capability, we have designed two approaches. The first approach exploits occlusion query that counts the number of fragments passed to the frame buffer. The following pseudo-code is executed in the fragment shader:

```
firstId = Ioid[fragmentCoord];
void find1stOccluded()
{
  if(idOf(sample) != queriedId) discard;
  if(queriedId == firstId) discard;
  if(! firstSampleOf(queriedId)) discard;
};
```

The modified volume rendering is performed N times, where N is the number of visible objects that are still unpacked. In each pass, $queriedId$ equals to the ID of the object being queried. Only the front-most samples belonging to the object pass if $queriedId$ is different than the corresponding value, $firstId$, in I_{oid}. Occlusion query returns the sum of occluded first-hit samples of $object_{queriedId}$. The total samples of the object can be evaluated similarly using only the first if statement of the above code. In practice, we just use the front face area of the bounding box of the object for approximation.

Each pass is optimized to only render the sub-volume defined by the bounding box of $object_{queriedId}$. But the accumulated time of tens of rendering passes may still introduce noticeable delay. Therefore, we also come up with a singe pass method taking advantage of the scattering and atomic counter capabilities of modern GPUs. The following pseudo-code is executed for each ray in a ray-casting setup.

```
void countOccludedSamples()
{
  currentId = -1;
  samples = 0;
  firstId = Ioid[fragmentCoord];
  while(moreSamples) {
    id = idOf(sample);
    if(id == firstId) continue;
    if(id == currentId) samples ++;
    else {
      updateCounter(currentId, samples);
      currentId = id;
      samples = 1;
    }
  }
  updateCounter(currentId, samples);
};

void updateCounter(currentId, samples)
{
  if(currentId > 0)
    atomicAdd(&counter[currentId], samples);
}
```

The pseudo-code counts contiguous samples belonging to an occluded object, and adds the partial sum to the counter table shared across all fragment shaders if the end of the ray is reached or the current object Id changes.

5.2 Volume rendering

The rendering of one or more objects moving from their original packed positions in the context of others that are still packed faces the same challenge as exploded views of volume data [Bruckner and Gröller 2006], multi-object or multi-modality volume rendering[Bruckner and Gröller 2005][Beyer et al. 2007], and multi-layer volume rendering [Kainz et al. 2009][Li 2010]. Because of the displacement of some portion of a volume, we can't infer visibility order of samples from their volume coordinates. The boundaries of the moving portions and the bounding box of the original volume partition the space into multiple regions. For a volume ray casting, rays are divided into segments each has different parameters from its predecessor and successor on the same ray. In the case of one object or volume portion overlaps with other non-empty part of the volume, multi-volume or fused volume rendering [Grimm et al. 2004][Bruckner and Gröller 2006] is required.

The fragmentation of rays introduces a lot of branches into the rendering code that severely degrades the performance. Besides, allowing one object to overlap and to pass through others is also not visually plausible for unpacking. In our implementation, we restrict the unpacking path to always start in the opposite of the viewing direction. Note that we use orthogonal projection when determining the unpackability. For a fully unblocked object, the displacement in the opposite of the viewing direction guarantees that the object being unpacked is alway in front of the objects that are still packed, either the camera is in perspective or orthogonal projection. Until the object being unpacked is completely outside the bounding box of the volume, it moves towards its unpacking location. As shown in Figure 1, the red doll first moves towards the camera as it appears larger in Figure 1(b) than (a) due to the perspective projection of the camera, than it moves in the up-right direction of the image space.

5.3 Unpacking animation

To efficiently visualize unpacking, we divide the whole scene into three layers. As shown in Figure 4, the first layer (4(a)) contains all the objects still unpacked and any background image plus decorations. The second layer (4(b)) is responsible for showing the animation of unpacking of individual objects, as well as allowing user to interactively inspect any unpacked object. The third layer (4(c)) shows all the unpacked objects as 3D sprites. Adding the three layers together produces the visualization in Figure 4(d).

To animate the unpacking of an object O, such as the brown box in Figure 4, we set its tinting color to be zero and perform the tinted volume rendering described in section 4.1, which essentially hides O in this layer. Next we continuously apply certain geometric transformation, such as translation and rotation on the whole volume,

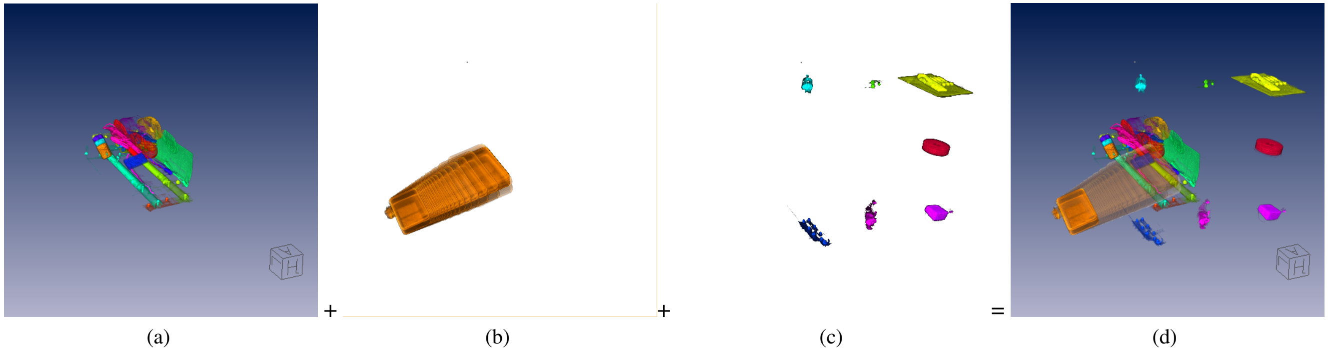


Figure 4: *Virtual unpacking. (a) Layer of packed objects. (b) Transition layer of objects being unpacked. (c) 3D sprites of unpacked objects. (d) Compositing of the three layers in (a) - (c) to efficiently visualize the unpacking.*

and perform the tinted volume rendering multiple times onto the animation layer, but this time setting the tinting color of all the objects except O to zero. If there are multiple objects being unpacked simultaneously, the rendering to the animation layer is applied multiple times at every animation step, one for each unpacking object. The sprite layer simply displays the 3D sprites of unpacked objects. The final image in the animation layer of an unpacking object is used to create a sprite of the object and added to the sprite layers.

The location of any volume sample v is transformed by cascaded matrices to generate a window coordinate w:

$$w = V \times T \times M \times v \tag{3}$$

where M is the model matrix, and T specifies additional transformation. Their combination transforms from volume coordinates to world coordinates. T defines additional transformations, while view matrix V transforms from world coordinates to camera coordinates. All the three layers share the same camera. The desired unpacked locations are given in eye coordinates as they should be independent on view directions, whereas the system changes the T to $T'T$ to displace objects. For an object O, its unpacked displacement in volume coordinates T' is given by 4

$$T' = (V \times M)^{-1}(u - V \times M \times o) \tag{4}$$

where u is the desired unpacked location in camera coordinates, and o is the offset vector of O pointing from the volume center to the center of the bounding box of O. $(\bullet)^{-1}$ computes the inverse of a matrix. Considering the offset vector o ensures the center of O is aligned with the desired unpacked location, regardless of its original position in the volume. A sprite is visualized by putting a camera-space rectangle mapped with the sprite image centered at its desired location u. The sprites always face the camera and align their vertical edges with the up vector of the camera. In addition, the sprites also scale accordingly as the camera zooms.

Note that the image of the unpacked volume layer is cached and reused during the animation. Actually, it is not updated until an object is unpacked or restored, or an user changes the position or orientation of the volume. The transformation for the rendering of the unpacking animation layer follows a predefined path that contains a few key frames. At each key frame, a transformation composed of translation and rotation is defined. The transformation applied to an unpacking object is simply interpolated from the adjacent two key frames. For the sprite layer, our system stores a set of sprites, instead of a big image composing all the them. The reasons for making such a decision are: 1) each object showing in a sprite usually only occupies a small portion of the view port, and distributed sparsely, as showing Figure 4 (c). Storing the sprites separately requires less memory than storing a big image covering the whole view port; 2) The system needs frequently adding new sprites, hiding or removing existing sprites. Storing them separately facilitates such management.

User may select any sprite for inspection. In that case, the corresponding sprite is hidden, and a volume rendering of the object is created in place of the sprite, again by setting the tinting colors to zero for all the objects except O. Users can perform standard manipulation, such as rotation, zoom, and changing transfer function, on O. When user exits the inspection mode, the final image is used to update the sprite.

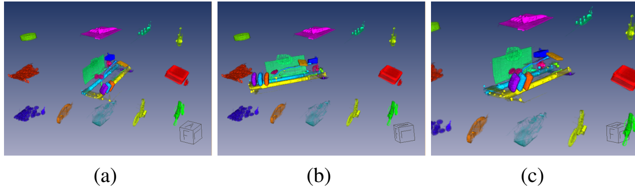
5.4 Interaction behavior

By analyzing the typical requirements of unpacking interaction, as well as exploiting the capabilities of our system, we design our system to respond to user interactions in the following way:

- Requirement for rotation and translation are considered operations to the current active object, which is determined by the position of mouse cursor when the left button is clicked. If a ray pick originated from the position hits any sprite, then the corresponding object becomes active. Otherwise, the active object is the whole volume composed of the objects that still packed.
- A double click of the left mouse button attempts to pick a sprite as in the previous case. If such a sprite is found, the system gets into the inspection mode, for which the volume rendering camera will automatically pan and zoom to make the active object appear in the center of the scene with comfortable zoom, in addition to forwarding all rotation and translation requests to the object.
- Whenever the viewing direction of the packed volume changes, a computation to determine unpackable objects as described in section 5.1 is performed, and their IDs are put into an unpackable object queue.
- If an unpacking requirement is received, the first one or more unpackable objects in the queue are unpacked and the process is animated as described in 5.3.
- If an restoring requirement is received, the latest unpacked objects are added back to the head of the queue, and their transitions from unpacked location to the packed volume are animated.

- A user zoom request is considered as an operation of the camera. Therefore all the three layers are zoomed in a synchronized fashion.

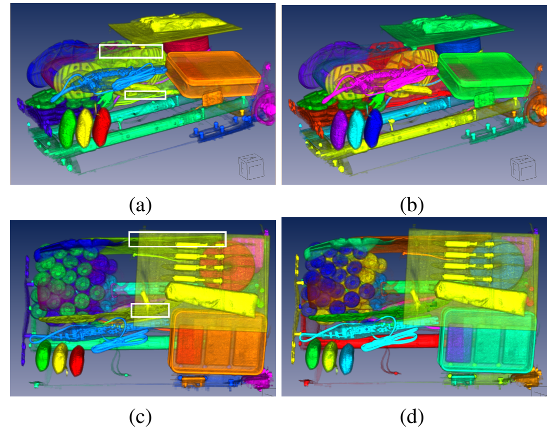
In our system, the sprites are not affected by any rotation or translation performed on the packed volume, which is evident by comparing Figure 5(a) and (b). Similarly, non active sprites are not affected by the manipulation on an active unpacked object either. All these transformations are applied in object space to the active object or the packed volume. In contrast, all the objects in a scene are zoomed together, as shown in Figure 5(c).



(a) (b) (c)

Figure 5: *Unpacked objects are shown as 3D sprites. (a) A screen shot in the middle of unpacking. (b) Sprites are invariant to the rotation and translation in the volume rendering of the packed objects; (c) The response of sprites to zoom operation.*

.



(a) (b)

(c) (d)

Figure 6: *Luggage visualization without ((a) and (c)) and with (((b) and (d) color optimization. Note that in the areas highlighted by white rectangles in Figures (a) and (c) different objects have indistinguishable color hues. In contrast, Figures (b) and (d) do not have this issue.*

.

6 Results

We have implemented the described virtual unpacking system on a Windows-based PC. The visualization computation is mainly performed on an Nvidia Geforce GTX 480 card. The volume renderer is developed using C++ and glsl. Luggage datasets are output from simulations to resemble the appearance of typical luggages going through CT scanners. Although they are not "real" luggage scans, we argue they pose similar challenges to the segmentation and the visualization. A typical luggage volume is of the size of $512 \times 512 \times 1024$ with each voxel occupies 2 bytes. The dataset has significant object variations from soaps to bottled water, wax candles to boots, different clothing etc.

Figure 6 compares the results before and after color optimization. The white rectangle in Figure 6(a) highlights areas where different objects present visually identical colors, although they are different, whereas there is no such an issue in 6(b).

The scene in Figure 6 is composed of 27 labeled objects. Rendering with such a transparency setting to a 1024^2 view port takes about 70ms. The animation of unpacking a single object in a view similar to Figure 4 takes about 50ms for a single frame. It is faster than rendering the packed volume, because only the unpacking object is rendered and the bounding box of the object is usually significantly smaller than the full volume. Restoring animation has similar performance to unpacking. Manipulating the packed volume in a scene similar to Figure 5 reaches about 20 fps. This is also slightly faster than the cases in Figure 6, because users usually would zoom out to make the unpacked objects visible in the scene, and this reduces the time for rendering the packed volume.

Figure 7 shows the data of unpackable object determination during a serial process of unpacking. We utilized the multi-pass occlusion query approach. Only one object is unpacked at each step. The X axis is the index of the unpacking step. The two series show the occlusion ratio in percentage of the objects being unpacked and the time (in 100 milliseconds) for determining unpackable objects at each unpacking step. There are zeros for the time series as the previous step finds more than one unpackable object, and there is no need to run the query for the steps following immediately. Excluding those zeros, the time generally decreases monotonically, because the number of packed objects and the query passes decrease as the unpacking proceeds. Even at the beginning of the unpacking, where the system performs the query 26 times, the overall time is just slightly more than 50 ms, which is about the time for a full quality volume rendering with the same viewing parameters. This because the query pass for any object is restricted to the bounding box of the object, which is usually much smaller than that of the whole volume. Moreover, a querying ray terminates after reaching the first sample of the object. Whereas for a scene with a lot of transparency as in Figure 6, a ray travels a much longer distance. The single pass approach for determining unpackable object requires the sampling of non-empty volume, which is similar to that of rendering a mostly transparent volume. Therefore, the single pass method can be slower than the multi-pass approach. Note that in Figure 7 there are non-zero values at some steps, which means that no object is fully unblocked, although the ratio is only 0.3% at most.

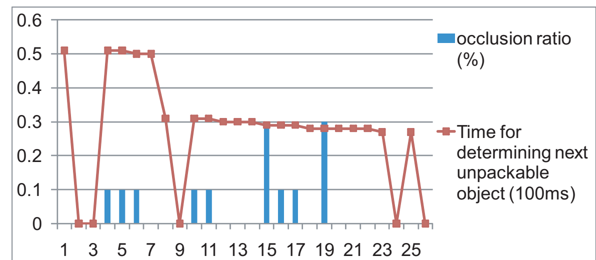


Figure 7: *Performance of determining unpackable objects*

Figure 8 presents the time for computing an single layer interference map of objects that is used for color optimization through an unpacking process. For timing purpose, color optimization is applied after the unpacking of every object, which is obviously unnecessary in practice. The average time is about 110 ms, that includes the rendering of an image of object IDs I_{oid}, the read back of I_{oid}

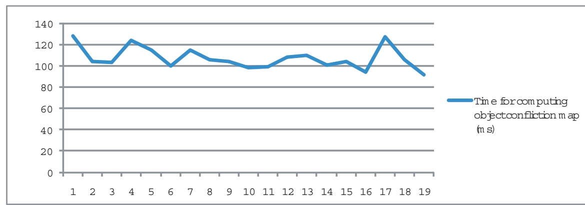


Figure 8: *Time for computing an single layer interference map of objects in an unpacking sequence*

from GPU to system memory, and the evaluation of formula 1.

Please check the accompany videos that have better presentations of the performance and dynamics of the unpacking and restoring processes.

7 Conclusion and Future Work

In this paper, we present the design of a fully workable system for luggage visualization and virtual unpacking. The key components of the system includes, a recursive segmentation method guided by a confidence measure, an efficient volume rendering approach that can visualize many segmentation masks, a method that automatically assigns distinctive colors to interfering objects as well as exploiting the full color space, GPU accelerated approaches for quickly determining unpackable object from any angle, a layered framework that minimizes the cost of combining unpacked volume, packed objects, and animations of objects being unpacked. Although some of the individual components may have appeared with similar form in the literature, our customization and integration of these approaches have not been reported are proved to work well.

One limitation of the our unpacking is that no collision detection is performed. In theory, an object may pass through others during the transition. We choose a work-around to avoid this overhead as well as to simplify rendering efforts. Obviously, it would be more realistic to consider collision and even allow interaction and deformation of objects when unpacking and packing objects. We would expect this to be accelerated on GPU as well.

Our current color optimization does not consider any the previous color assignment. It is unavoidable to have abrupt color changes when rotating to a different angle. It would be desirable that the new color assignment keeps the change to the minimum by assigning identical or similar hues to objects whenever possible. We expect such an optimal algorithm to be NP-hard, but wish to find an approximate but fast method in the future.

Acknowledgment

This material is based upon work supported by the U.S. Department of Homeland Security under Task Order Number HSHQDC-10-J-00396. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security. Datasets owned and provided by ALERT DHS Center of Excellence, Northeastern University, Boston MA.

References

BEYER, J., HADWIGER, M., WOLFSBERGER, S., AND BHLER, K., 2007. High-quality multimodal volume rendering for preoperative planning of neurosurgical interventions.

BRUCKNER, S., AND GRÖLLER, M. E. 2005. Volumeshop: An interactive system for direct volume illustration. In *Visualization*, H. R. C. T. Silva, E. Gröller, Ed., 671–678.

BRUCKNER, S., AND GRÖLLER, M. E. 2006. Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (9), 1077–1084.

CALLAHAN, S., COMBA, J., SHIRLEY, P., AND SILVA, C. 2005. Interactive rendering of large unstructured grids using dynamic level-of-detail. In *IEEE Visualization*, 199–206.

CHAN, M.-Y., WU, Y., MAK, W.-H., CHEN, W., AND QU, H. 2009. Perception-based transparency optimization for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (November), 1283 – 1290.

EVERITT, C. 2001. Interactive order-independent transparency. *NVIDIA white paper*.

GOVINDARAJU, N. K., REDON, S., LIN, M. C., AND MANOCHA, D. 2003. Cullide: interactive collision detection between complex models in large environments using graphics hardware. In *Graphics hardware*, 25–32.

GOVINDARAJU, N. K., HENSON, M., LIN, M. C., AND MANOCHA, D. 2005. Interactive visibility ordering and transparency computations among geometric primitives in complex environments. In *Interactive 3D Graphics and Games*, 49–56.

GRADY, L., AND ALVINO, C. The piecewise smooth mumford-shah function on an arbitrary graph. *IEEE Transactions on Image Processing 18*, 11, 2547–2561.

GRADY, L., AND SCHWARTZ, E. L. 2006. Isoperimetric graph partitioning for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 3 (March), 469–475.

GRADY, L., SINGH, V., KOHLBERGER, T., ALVINO, C., AND BAHLMANN, C. 2012. Automatic segmentation of unknown objects, with application to baggage security. In *European Conference on Computer Vision (ECCV)*.

GRIMM, S., BRUCKNER, S., KANITSAR, A., AND GRÖLLER, M. E., 2004. Flexible direct multi-volume rendering in interactive scenes, Oct.

HADWIGER, M., BERGER, C., AND HAUSER, H. 2003. High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. *Visualization*, 301–308.

KAINZ, B., GRABNER, M., BORNIK, A., HAUSWIESNER, S., MUEHL, J., AND SCHMALSTIEG, D. 2009. Ray casting of multiple volumetric datasets with polyhedral boundaries on many-core gpus. In *ACM SIGGRAPH Asia*, 152:1–152:9.

LI, W., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2008. Automated generation of interactive 3d exploded view diagrams. In *ACM SIGGRAPH*, 101:1–101:7.

LI, W. 2010. Multi-layer volume ray casting on gpu. In *Volume Graphics*, 5–12.

NAGY, Z., AND KLEIN, R. 2003. Depth-peeling for texture-based volume rendering. In *Pacific Graphics*, 429.

ZHOU, J., AND TAKATSUKA, M. 2009. Automatic transfer function generation using contour tree controlled residue flow model and color harmonics. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (November), 1481 – 1488.