SIGGRAPH2015 Xroads of Discovery

Fully automatic ID mattes with support for motion blur and transparency

Introduction and motivation

We present a fully automatic method of creating and encoding ID mattes. 3D scenes consist of an organized hierarchy of objects and relationships, and when 2D images are rendered that information is not preserved. ID mattes attempt to preserve organizational information by establishing a correspondence between items in the 3D scene and particular pixels in the image plane. Artists use this correspondence as masks to modify the 3D scene's items in 2D images.

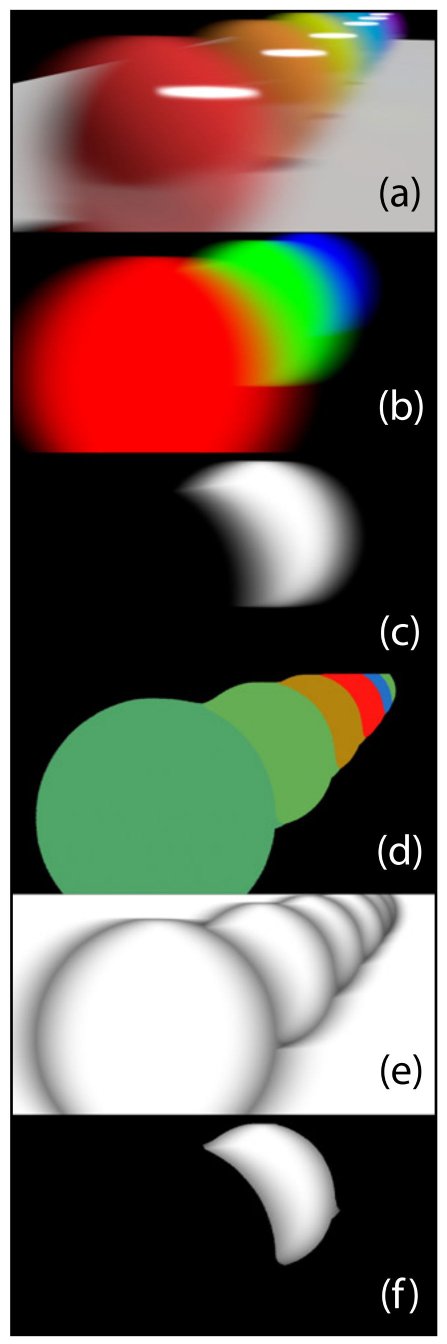

Figure 1: Given a beauty render (a) an RGBA matte (b) is used to generate an ID matte (c) by isolating one channel. Object IDs (d) and coverage (e) are used to generate an ID matte for any object, with artifacts (f).

Much effort is put into creating ID mattes. Our method produces a comprehensive set of ID mattes automatically, without the need for artist input. The resulting mattes provide a better compromise between compactness and accuracy than existing techniques. Our method also provides a robust bi-directional mapping between the ID mattes and the corresponding items in the 3D scene, conferring a number of benefits.

Existing techniques

In RGBA Mattes, an ID matte exclusively occupies a single channel of an RGBA image (Figure 1b, 1c). The number of images needed to provide compositors adequate coverage quickly accumulates. It can also be time-consuming for 3D artists to set up ID mattes, especially if they have to guess which mattes will be required.

Another approach is using a single ID-coverage pair (Figure 1d, 1e). These encode multiple ID mattes together using two channels, ID and coverage. The ID channel encodes the ID of one object per pixel. The coverage channel encodes how much of the pixel's value is contributed by this object. This method is unable to handle cases where multiple objects per pixel are important (Figure 1f). Multiple objects sharing pixels is very common; it occurs due to anti-aliasing, or when objects are motion-blurred, out of focus, or transparent. It is possible to guess the ID of the second object and use inverted coverage, but this covers two objects at most. ID generation is also an issue, as the renderer's internal object ID may not stay the same from shot to shot or even from frame to frame.

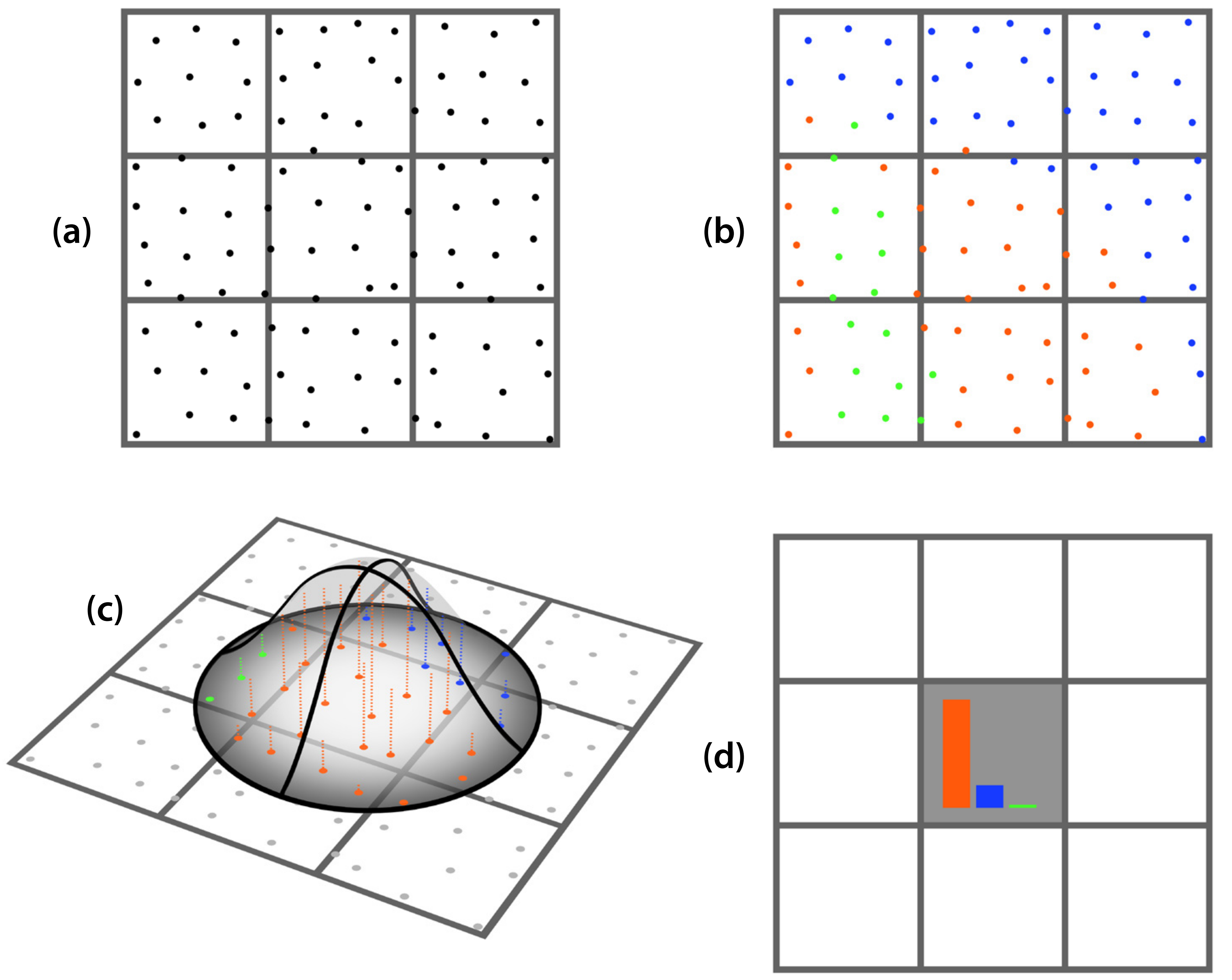

Figure 2: Samples on a pixel grid (a) have IDs computed and stored as an AOV (b). The samples are weighted using a filter kernel (c). The sample weights for each ID are ranked (d). In this example, orange is ranked highest. The weights will be the coverage values.

Our approach

In our approach we expand upon the ID-coverage pair technique by using *n* ranked ID-coverage pairs to add support for multiple objects per pixel. We also propose using a hashing system for converting organizational monikers from the 3D scene, such as object, namespace, and material names.

ID generation

In ID-coverage pairs, the identity of objects, or groupings of objects must be represented by a single data value. This value is known as the ID code, or ID. The canonical way of identifying objects in a 3D scene is to use human-readable string labels. Using a hash function, we simply convert these already-existing labels into ID codes.

String labels are by definition artist-friendly, because they are the same labels artists themselves use to identify objects in a scene. They can also be used as indices for accessing 3D scene components programmatically. In most productions, the string labels do not change from frame to frame or from shot to shot. Being stable, readable and indexable, these labels are ideal candidates for generating useful ID codes.

Names

In our implementation, we use three types of labels to generate three separate IDs: namespaces, object names, and material names. Objects are individual polygon meshes, curves, surfaces, etc. Namespaces differentiate hierarchical groups of objects, such as instances of an asset that has been duplicated several times. Material names are taken from shaders assigned to objects.

While object ID mattes and namespace ID mattes provide access to the scene hierarchy with different levels of granularity, material ID mattes provide access to objects with similar physical properties, such as all of the buttons on a shirt.

At render time, object and material names are typically provided with the namespace as a prefix, using a separator character. We process these full object names (Figure 3) into namespaces and short object names. The material names are processed similarly.

Full names in scene:
- 🟧🟥 bunny1:body
- 🟧🟦 bunny1:nose
- 🟧🟩 bunny1:whiskers
- 🟦🟥 bunny2:body
- 🟦🟦 bunny2:nose
- 🟦🟩 bunny2:whiskers
- 🟩🟧 flower1:leaf1
- 🟩🟦 flower1:leaf2
- 🟩🟨 flower1:stem
- 🟩🟪 flower1:petals

Namespace manifest:
- 🟧 bunny1
- 🟦 bunny2
- 🟩 flower1

Object manifest:
- 🟥 body
- 🟦 nose
- 🟩 whiskers
- 🟧 leaf1
- 🟦 leaf2
- 🟨 stem
- 🟪 petals

Figure 3: Names as found in a scene, processed to IDs (represented by colors). Two copies of a "bunny" asset are present.

Hashing

As the renderer samples the image, the three processed names are hashed to produce ID codes (Figure 2b). The IDs are stored as three arbitrary output variables (AOVs) at the sample level.

IDs can either be computed while sampling, or precomputed and stored with the objects they are associated with. In our implementation, we have opted to compute IDs while sampling. This generally simplifies our system. To allow for this, we have chosen the extremely small and fast hashing algorithm, DJB2.

In addition to storing ID codes per-pixel, we also store the processed names in a manifest, which allows us to recover the full string names from the ID codes using a reverse lookup. Using the manifest, we can provide descriptive names for ID mattes and are able to index specific objects within the 3D scene.

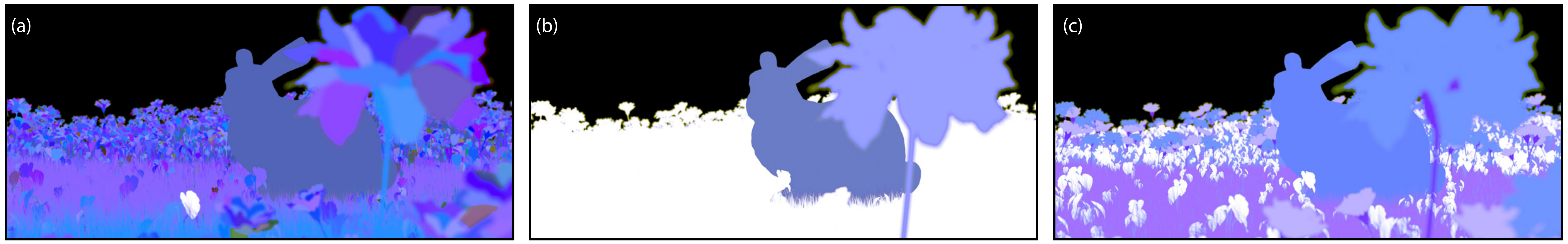

Figure 4: Three instances of the "keyable" image presented to artists, with matte selections in white, from the object IDs (a), namespace IDs (b), and material IDs (c), extracted by "keying" a leaf.

Filtering and encoding

The data contained in the three ID AOVs (Figure 4) is processed by a ranking sample filter. This algorithm produces a mapping from ID values to weights in the pixel. It uses an interchangeable filter kernel for weighting. Any filter kernel used to filter the "beauty" image can be reused by our ranking filter, allowing our extracted mattes to match the "beauty" results.

To process a pixel, we compute a table of weights per ID, and then rank the IDs by weight. We create the table by iterating through all samples which contribute to the pixel. The weight of each sample is computed using an arbitrary filter kernel, such as Gaussian or Blackman-Harris (Figure 2c). The weight is then added to the row of the table that contains a matching ID value. If no match is found, a new ID is added to the table.

In the case that a sample represents a partially transparent object, the sample will contain a number of sub-samples, each with an ID and opacity value. The weight of each subsample is the product of its own opacity, the combined transparency of the sub-samples in front of it, and the result of the filter kernel. Combined transparency can be computed as the product of one minus opacity, over the set of sub-samples.

We treat the background of the scene as a null ID, signifying empty samples. It is used when a sample does not collide with an object or only collides with transparent objects.

Once all samples have been computed, the IDs are ranked by the absolute value of their accumulated weights. This ensures that only the IDs with the lowest weights are discarded if the number of ID-coverage pairs is smaller than the number of IDs in the table.

Encoding

We use multi-channel OpenEXR files to encode the data. Each of our three AOVs is stored in a separate file, which includes an arbitrary number of ID-coverage pairs and a manifest. Each ID-coverage pair represents one rank, and one RGBA image can contain two ranks in its four channels. The manifest is simply a list of the processed names used to create the IDs and is stored as metadata.

The depth value, or the number of ranks stored, can be set to whatever is required. Each successive ID-coverage depth layer tends to contain fewer objects than the previous. As a result, each layer tends to be more compressible than the previous, and increasing depth to accommodate localized complexity is not very costly.

In our implementation, depth can be set by artists in 3D. The default value is six, which is almost always sufficient in practice. Although it is not required, we store a conventionally-filtered RGB version to help artists visualize the ID data.

Matte extraction

Matte extraction is straightforward (Figure 5) and computationally inexpensive. Given an ID value, iterate through the ID-coverage pairs. If the ID for a pixel matches the given value, the associated coverage value is the matte value for that pixel, and further pairs do not need to be checked.

To extract combined mattes for multiple specified IDs, iterate through the ID coverage pairs. If the ID for a given pixel matches any of the specified IDs, add the associated coverage value to the matte value for that pixel.

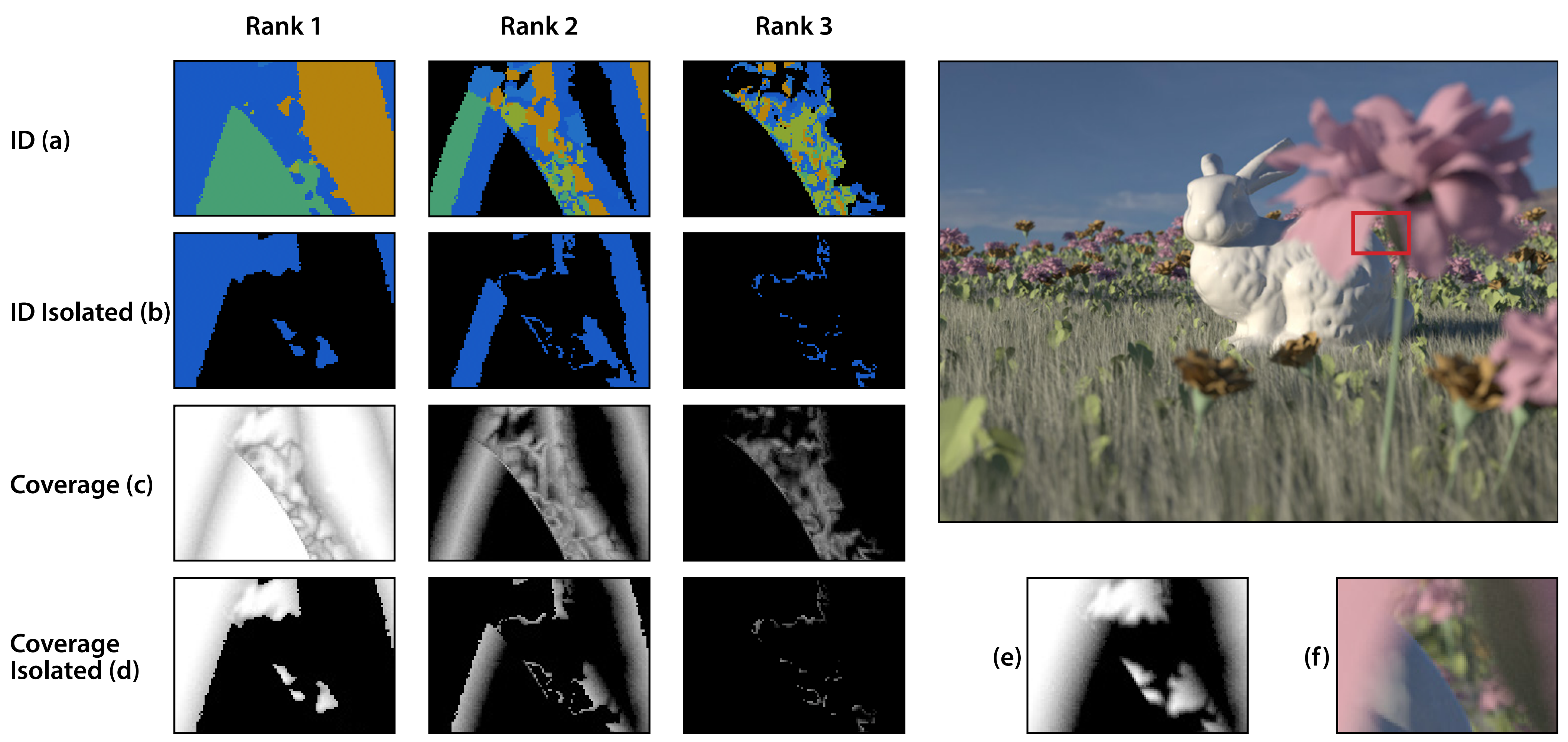

Figure 5: Three ranks of IDs (a) are tested for pixels matching an exact value (b). The associated coverage channels (c) are isolated to the region defined by the matching ID pixels (d). The sum of the isolated coverage values from each rank is the extracted matte (e). The same area of the "beauty" render (f) for comparison.

Implementation details

A "keyable" RGB image is constructed in the compositing software (Figure 4). This image's red and green channels are the conventionally-filtered versions of the ID data. The blue channel is the ID channel from the first (highest ranked) ID-coverage pair. When sampling a color for this "keyable" image using an eye-dropper, the resulting blue channel is used as the ID for extraction.

Since our system includes a bidirectional mapping between ID values and object names, the tools can easily provide a list of selected object names to artists and can also accept lists of names as input. Artists can generate lists of names interactively, using the eye-dropper tool to select and deselect individual objects.

The lists of names generated in 2D not only allow for clear communication with the 3D artist, but also can be used as an index to apply changes directly to the 3D scene. This workflow can be used to visualize changes to a 3D scene more quickly than the usual approach of tweaking parameters and rendering previews. Changes to a 3D scene can be tested in motion or under variable lighting, without having to re-render a sequence of frames for every round of edits. When finished, the 2D adjustments can be applied to the shaders.

Since deploying the toolset, we have received positive feedback from 3D artists and 2D artists alike. We have found that artists are using ID mattes where they previously would have resorted to other techniques, such as keying, rotoscopy, or requesting changes in 3D. We have also seen larger workflow benefits. Previously we would see a period at the end of jobs where 2D artists would have to ask 3D artists for more and more ID mattes. We no longer see this pattern. All required mattes can be trivially extracted from our images. As such, this is a production-proven replacement for practically all ID mattes.

Acknowledgements

Tony Barbieri
Wellington Fan
Brendan Fitzgerald
Nisa Foster
Andy Gilbert
Vladimir Jankijevic
Tram Le-Jones
Jean-Francois Panisset
Solid Angle
Psyop