CPSC 4040/6040 Computer Graphics Images

Joshua Levine
levinej@clemson.edu

Lecture 08
Deep Images

Sept. 15, 2015

Agenda

- Quiz02 will come out on Thurs.

- Programming Assignment 03 posted

Last Time: Green Screening

Smith-Blinn Formalization

Blue Screen Matting

Alvy Ray Smith and James F. Blinn[1]
Microsoft Corporation

ABSTRACT

A classical problem of imaging—the *matting problem*—is separation of a non-rectangular foreground image from a (usually) rectangular background image—for example, in a film frame, extraction of an actor from a background scene to allow substitution of a different background. Of the several attacks on this difficult and persistent problem, we discuss here only the special case of separating a desired foreground image from a background of a constant, or almost constant, *backing* color. This backing color has often been blue, so the problem, and its solution, have been called *blue screen matting*. However, other backing colors, such as yellow or (increasingly) green, have also been used, so we often generalize to *constant color matting*. The mathematics of constant color matting is presented and proven to be unsolvable as generally practiced. This, of course, flies in the face of the fact that the technique is commonly used in film and video, so we demonstrate constraints on the general problem that lead to solutions, or at least significantly prune the search space of solutions. We shall also demonstrate that an algorithmic solution is possible

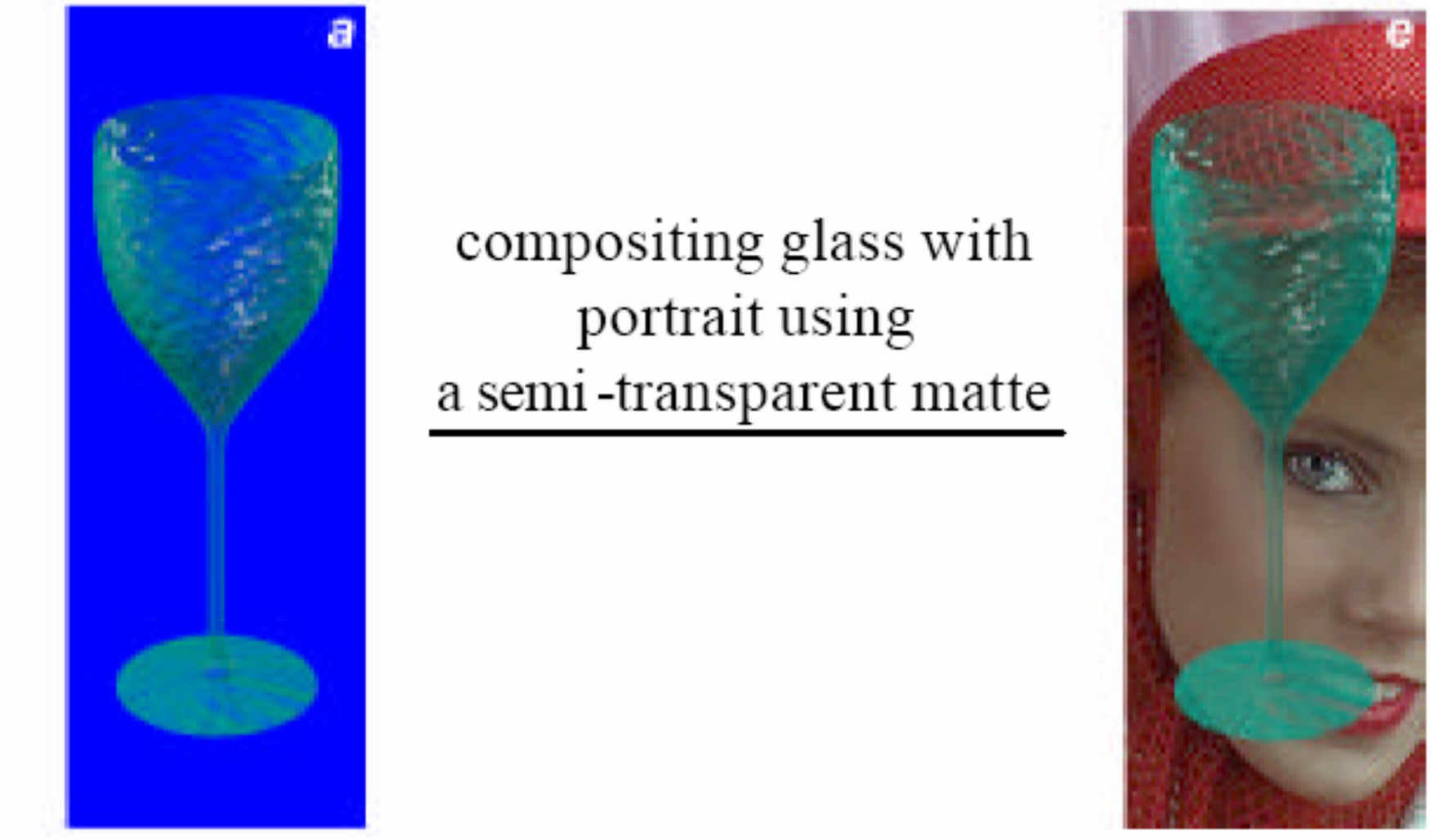
the color film image that is being matted is partially illuminated.

The use of an *alpha channel* to form arbitrary compositions of images is well-known in computer graphics [9]. An alpha channel gives shape and transparency to a color image. It is the digital equivalent of a holdout matte—a grayscale channel that has full value pixels (for opaque) at corresponding pixels in the color image that are to be seen, and zero valued pixels (for transparent) at corresponding color pixels not to be seen. We shall use 1 and 0 to represent these two alpha values, respectively, although a typical 8-bit implementation of an alpha channel would use 255 and 0. Fractional alphas represent pixels in the color image with partial transparency.

We shall use “alpha channel” and “matte” interchangeably, it being understood that it is really the holdout matte that is the analog of the alpha channel.

The video industry often uses the terms “key” and “keying”—as in “chromakeying”—rather than the “matte” and “matting” of the film industry. We shall consistently use the film terminology, after first pointing out that “chromakey” has now taken on a more sophisticated meaning (e.g., [8]) than it originally had (e.g., [19]).

Semi-Transparent Mattes



- What we really want is to obtain a true α matte, which involves semi-transparency
  - α between 0 and 1

Big Picture Idea: Matting is Ill-posed

- An infinite number of solutions exist to the matting equations:

$$R = (\boldsymbol{\alpha}_F R_F) + (1-\boldsymbol{\alpha}_F) R_B$$

$$G = (\boldsymbol{\alpha}_F G_F) + (1-\boldsymbol{\alpha}_F) G_B$$

$$B = (\boldsymbol{\alpha}_F B_F) + (1-\boldsymbol{\alpha}_F) B_B$$

- Four unknowns (R_F, G_F, B_F, $\boldsymbol{\alpha}_F$), but 3 equations

- Need to the problem somehow:
  - e.g. Petro Vlahos approach: G = kB, but could we constrain the system otherwise?

Simple Constraint: Assume Foreground has no Green

- In this case, $G_F = 0$, so the equations change to:

$$R = (\boldsymbol{\alpha}_F R_F) + (1-\boldsymbol{\alpha}_F) R_B$$

$$G = G_B - \boldsymbol{\alpha}_F G_B$$

$$B = (\boldsymbol{\alpha}_F B_F) + (1-\boldsymbol{\alpha}_F) B_B$$

- In total, 3 unknowns (R_F, B_F, $\boldsymbol{\alpha}_F$), and 3 equations

- What's wrong with this solution?

Simple Constraint: Assume Foreground has no Green

- In this case, $G_F = 0$, so the equations change to:

$$R = (\boldsymbol{\alpha}_F R_F) + (1-\boldsymbol{\alpha}_F) R_B$$

$$G = G_B - \boldsymbol{\alpha}_F G_B$$

$$B = (\boldsymbol{\alpha}_F B_F) + (1-\boldsymbol{\alpha}_F) B_B$$

First, solve for $\boldsymbol{\alpha}_F$

- In total, 3 unknowns (R_F, B_F, $\boldsymbol{\alpha}_F$), and 3 equations

- What's wrong with this solution?

Simple Constraint: Assume Foreground has no Green

- In this case, $G_F = 0$, so the equations change to:

$$R = (\boldsymbol{\alpha}_F R_F) + (1-\boldsymbol{\alpha}_F) R_B$$

$$G = G_B - \boldsymbol{\alpha}_F G_B$$

First, solve for $\boldsymbol{\alpha}_F$

$$B = (\boldsymbol{\alpha}_F B_F) + (1-\boldsymbol{\alpha}_F) B_B$$

- In total, 3 unknowns (R_F, B_F, $\boldsymbol{\alpha}_F$), and 3 equations

- What's wrong with this solution?

  - Excludes all greys except black, and a huge collection of other colors because white contains green.

Another Constraint: Grey/Flesh

- If we know that the foreground contains grey, we can set $R_F = G_F = B_F$

- This leaves us with three equations, but only two unknowns

$$R = (\boldsymbol{\alpha}_F R_F) + (1\text{-}\boldsymbol{\alpha}_F) R_B$$

$$G = (\boldsymbol{\alpha}_F R_F) + (1\text{-}\boldsymbol{\alpha}_F) G_B$$

$$B = (\boldsymbol{\alpha}_F R_F) + (1\text{-}\boldsymbol{\alpha}_F) B_B$$

Another Constraint: Grey/Flesh

- If we know that the foreground contains grey, we can set $R_F = G_F = B_F$

- This leaves us with three equations, but only two unknowns

$$R = (\boldsymbol{\alpha}_F R_F) + (1\text{-}\boldsymbol{\alpha}_F) R_B$$

$$G = (\boldsymbol{\alpha}_F R_F) + (1\text{-}\boldsymbol{\alpha}_F) G_B$$

$$B = (\boldsymbol{\alpha}_F R_F) + (1\text{-}\boldsymbol{\alpha}_F) B_B$$

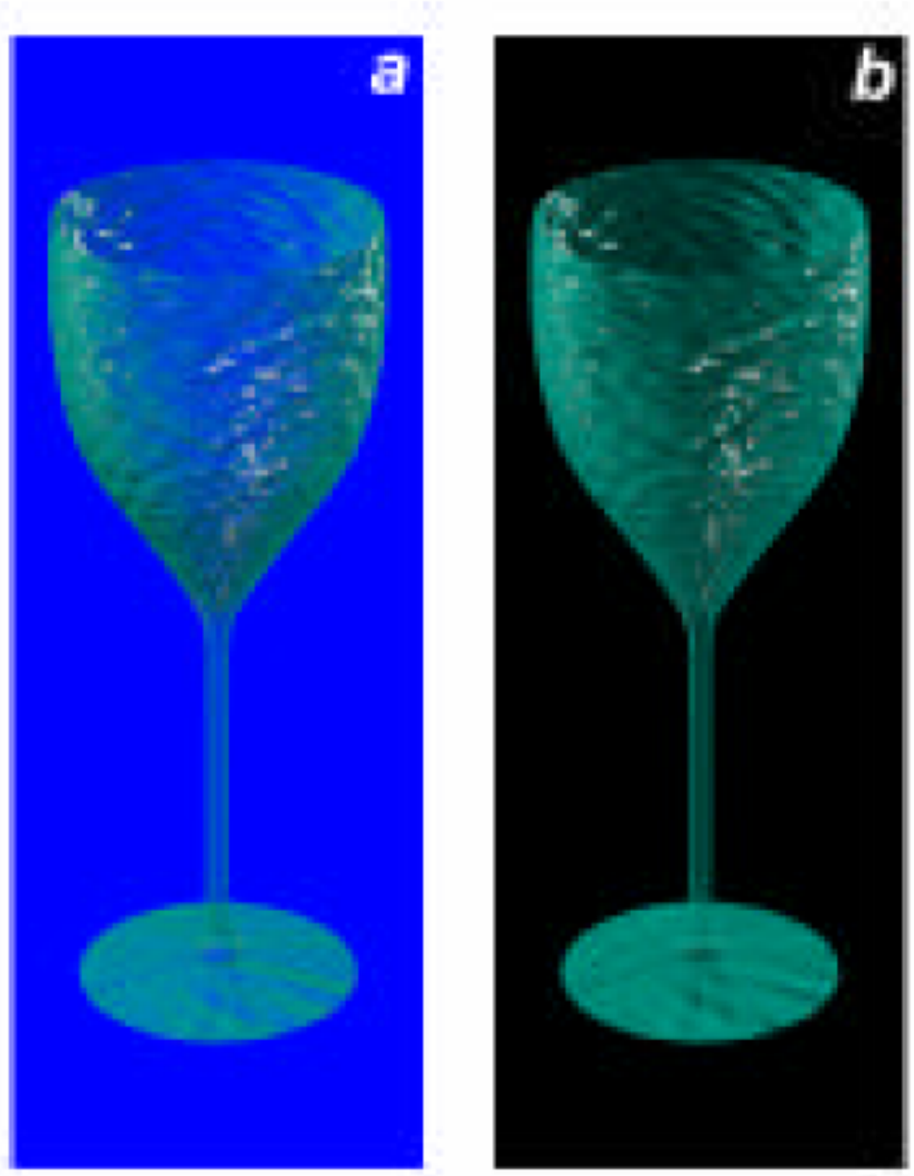
- Note: Could modify this using the fact that flesh tones typically are (R,G,B) = (d,½d,½d), where d determines the darkening of the skin due to race.

Triangulation Matting

- Instead of reducing the number of unknowns, we could instead increase the number of equations.

- One way to do this is photograph the object in front of two different backgrounds

- This is six equations (3 for each image) but only four unknowns.



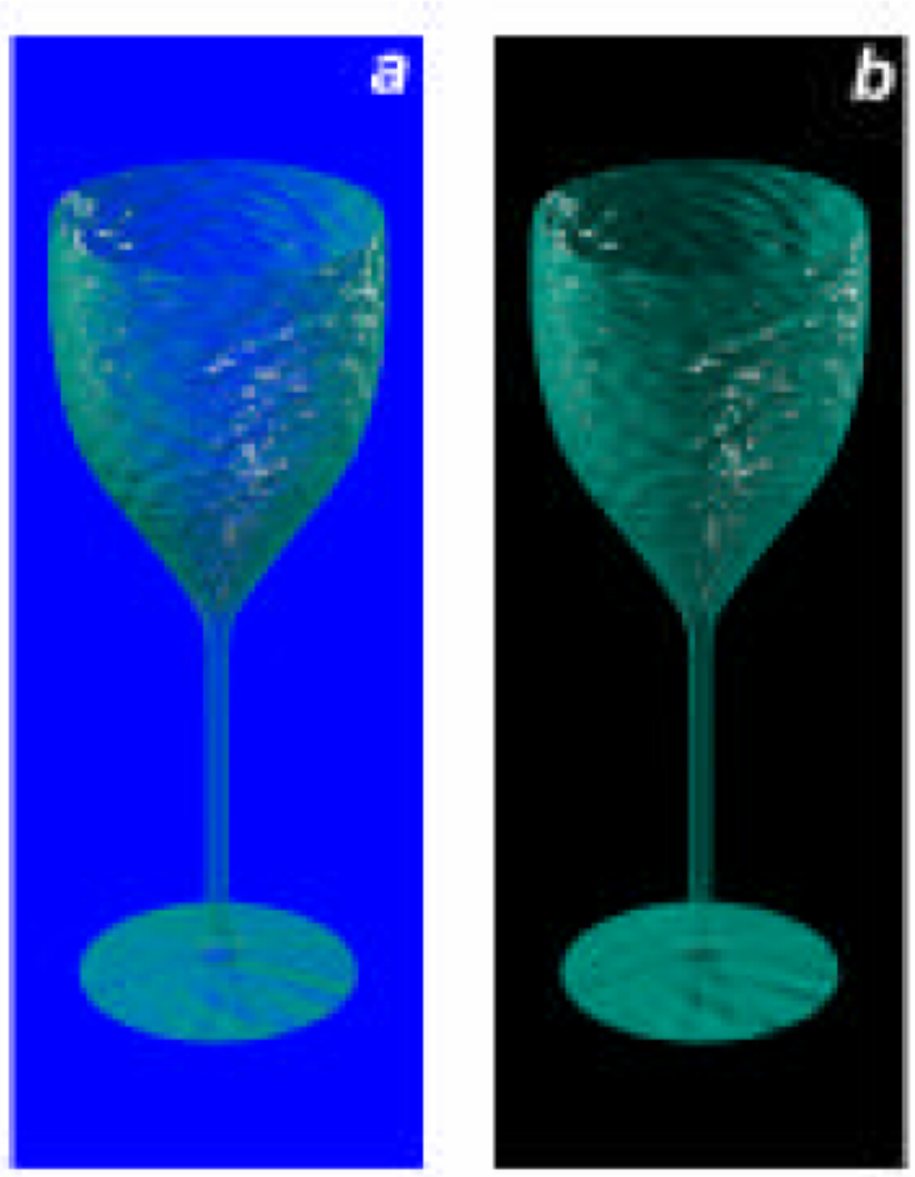
Triangulation Matting

- Instead of reducing the number of unknowns, we could instead increase the number of equations.

- One way to do this is photograph the object in front of two different backgrounds

- This is six equations (3 for each image) but only four unknowns.

- Note: the backgrounds need not be constant, you process this independently per pixel.



Triangulation Matting Algorithm

- For every pixel p in the composite image, given:
  - Backing color $C_{B1} = (R_{B1}, G_{B1}, B_{B1})$ at p
  - Backing color $C_{B2} = (R_{B2}, G_{B2}, B_{B2})$ at p
  - Composited pixel color $C_1 = (R_1, G_1, B_1)$ at p, and
  - Composited pixel color $C_2 = (R_2, G_2, B_2)$ at p, and
- Solve the system of 6 equations:

$$R_1 = \boldsymbol{\alpha}_F R_F + (1-\boldsymbol{\alpha}_F) R_{B1} \qquad R_2 = \boldsymbol{\alpha}_F R_F + (1-\boldsymbol{\alpha}_F) R_{B2}$$

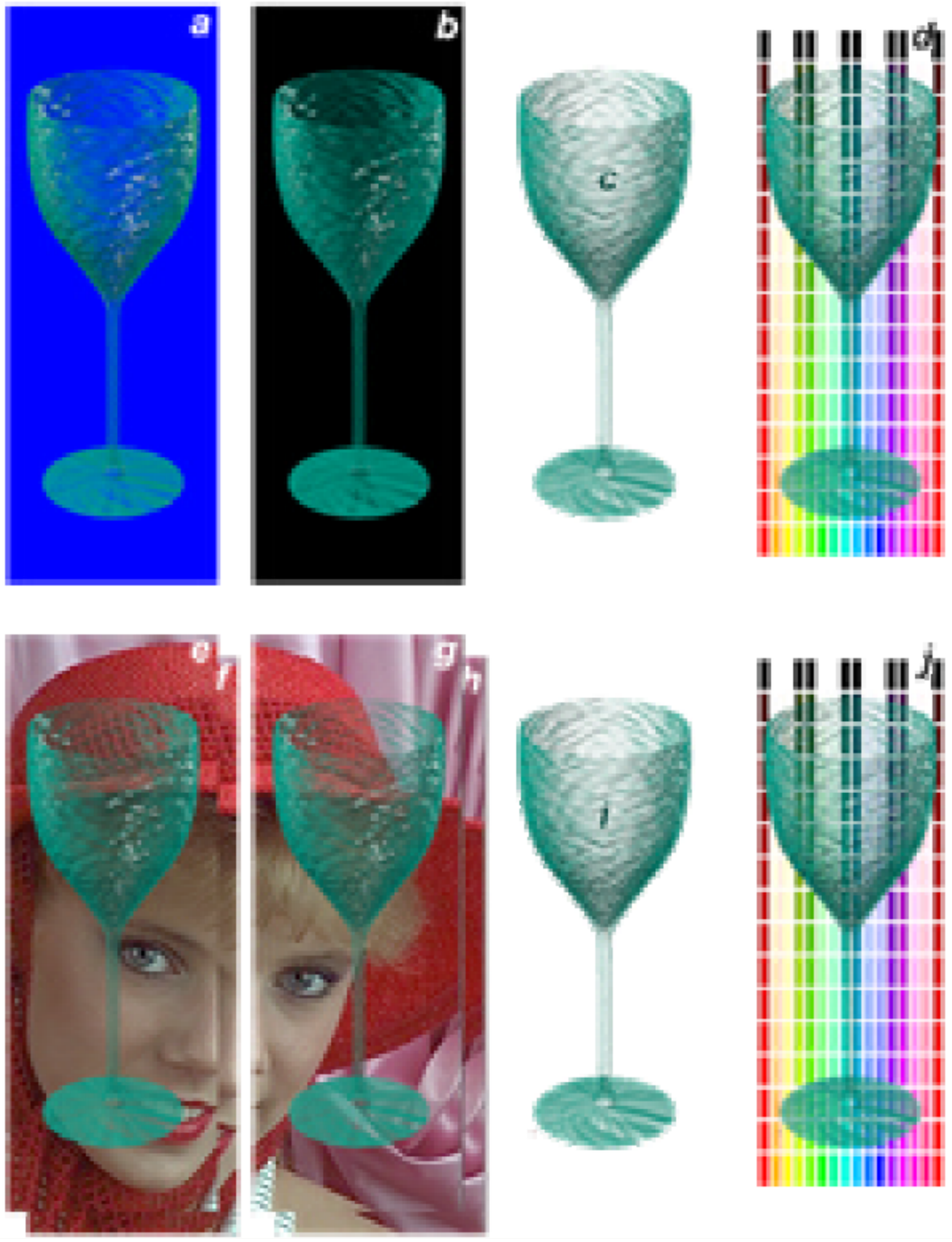
$$G_1 = \boldsymbol{\alpha}_F G_F + (1-\boldsymbol{\alpha}_F) G_{B1} \qquad G_2 = \boldsymbol{\alpha}_F G_F + (1-\boldsymbol{\alpha}_F) G_{B2}$$

$$B_1 = \boldsymbol{\alpha}_F B_F + (1-\boldsymbol{\alpha}_F) B_{B1} \qquad B_2 = \boldsymbol{\alpha}_F B_F + (1-\boldsymbol{\alpha}_F) B_{B2}$$

- For unknowns $(R_F, G_F, B_F, \boldsymbol{\alpha}_F)$

Triangulation Matting Examples

From Smith & Blinn's
SIGGRAPH'96 paper



More Examples

More Examples

Pitfalls and Problems

- Images do not look realistic
- Lack of Refracted Light
- Lack of Reflected Light
- Solution: Modify the Matting Equation

Can We Make This Look Better?

Deep Images

What are deep images?

- A **flat image** has at most one stored value or sample per pixel per channel.
    - For example, RGBA images, which contains three channels, and every pixel has exactly one R, one G, one B, and one A sample.
- A **deep image** can store an unlimited number of samples per pixel
    - Each of those samples is associated with a depth, Z, or distance from the viewer.

Terminology (Review?)

- **Pixel** - an image location (col,row)
- **Channel** - the components of data stored at a pixel
- **Sample** - an individual value of a channel at a pixel

- Deep images store a variable number of samples per pixel, include NO samples at some pixels. Each sample always has the same number of channels.

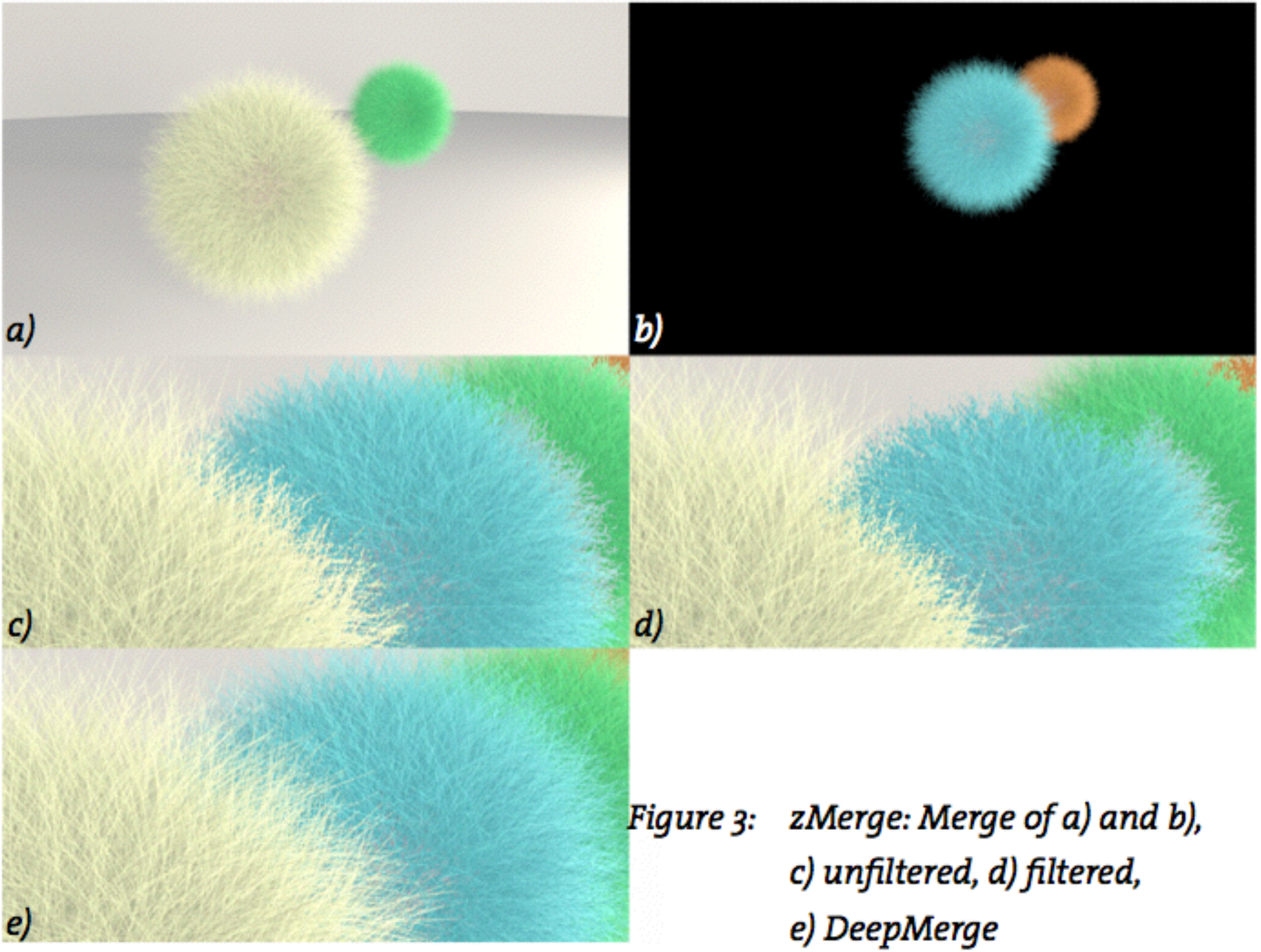


Figure 3: zMerge: Merge of a) and b), c) unfiltered, d) filtered, e) DeepMerge

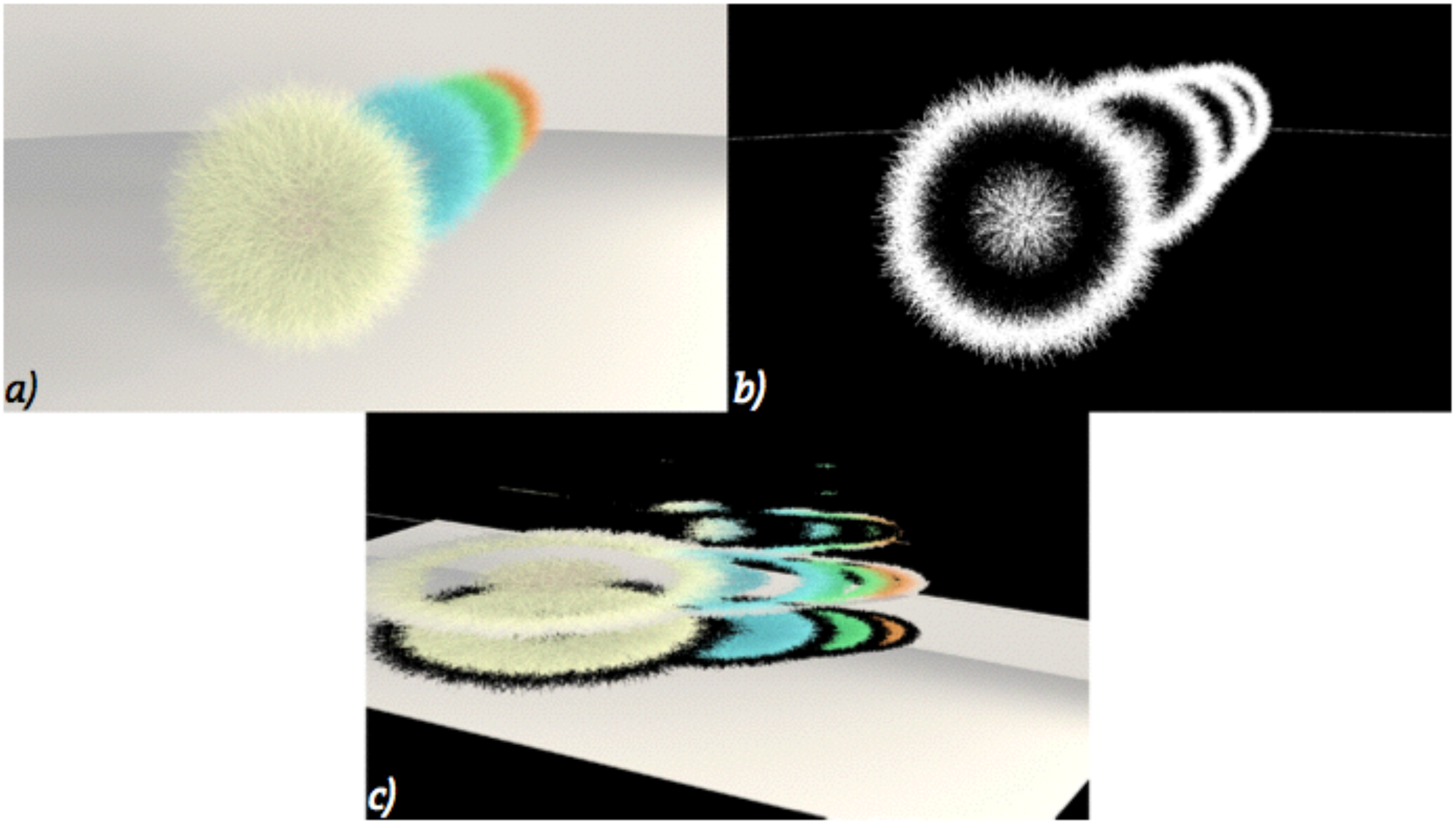


Figure 21: a) Fur rendering, b) pixels with more than one sample, c) sample count represented as the height of layers

Previous Example in 3-dimensions

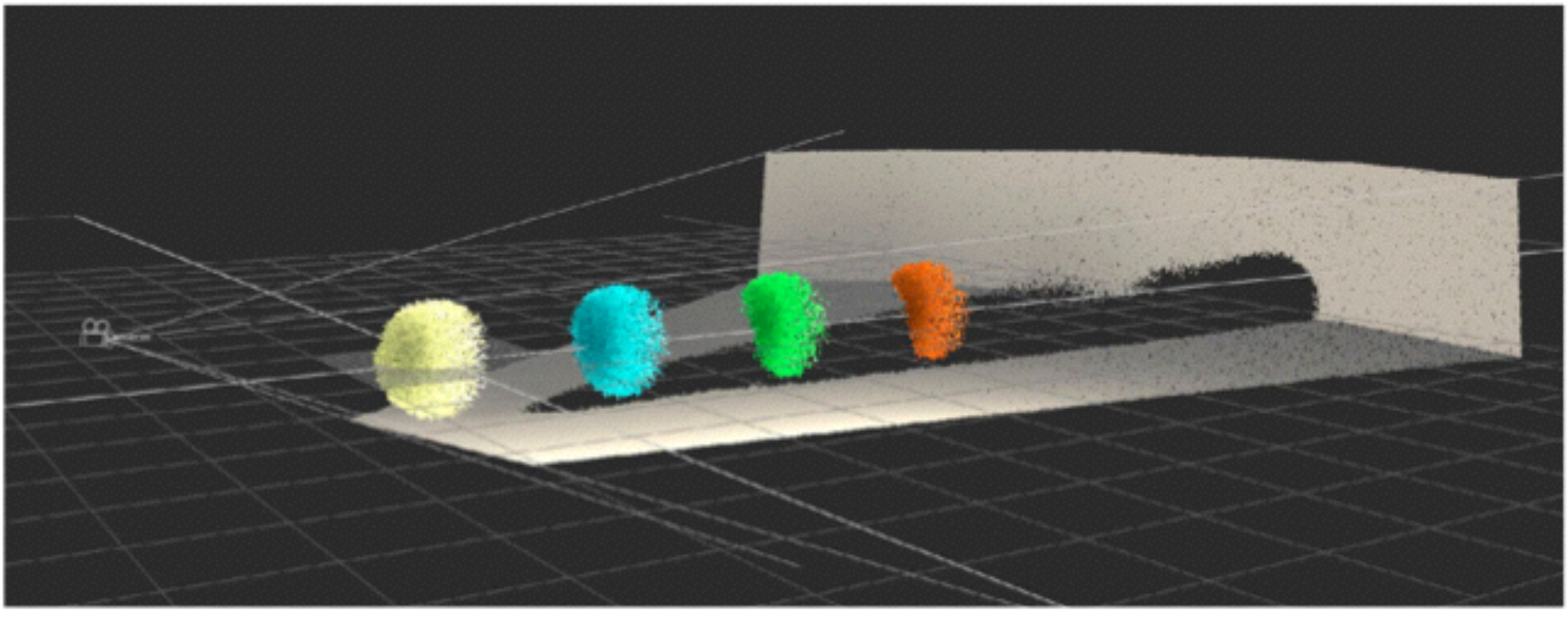


Figure 14: Output of the DeepToPoints node

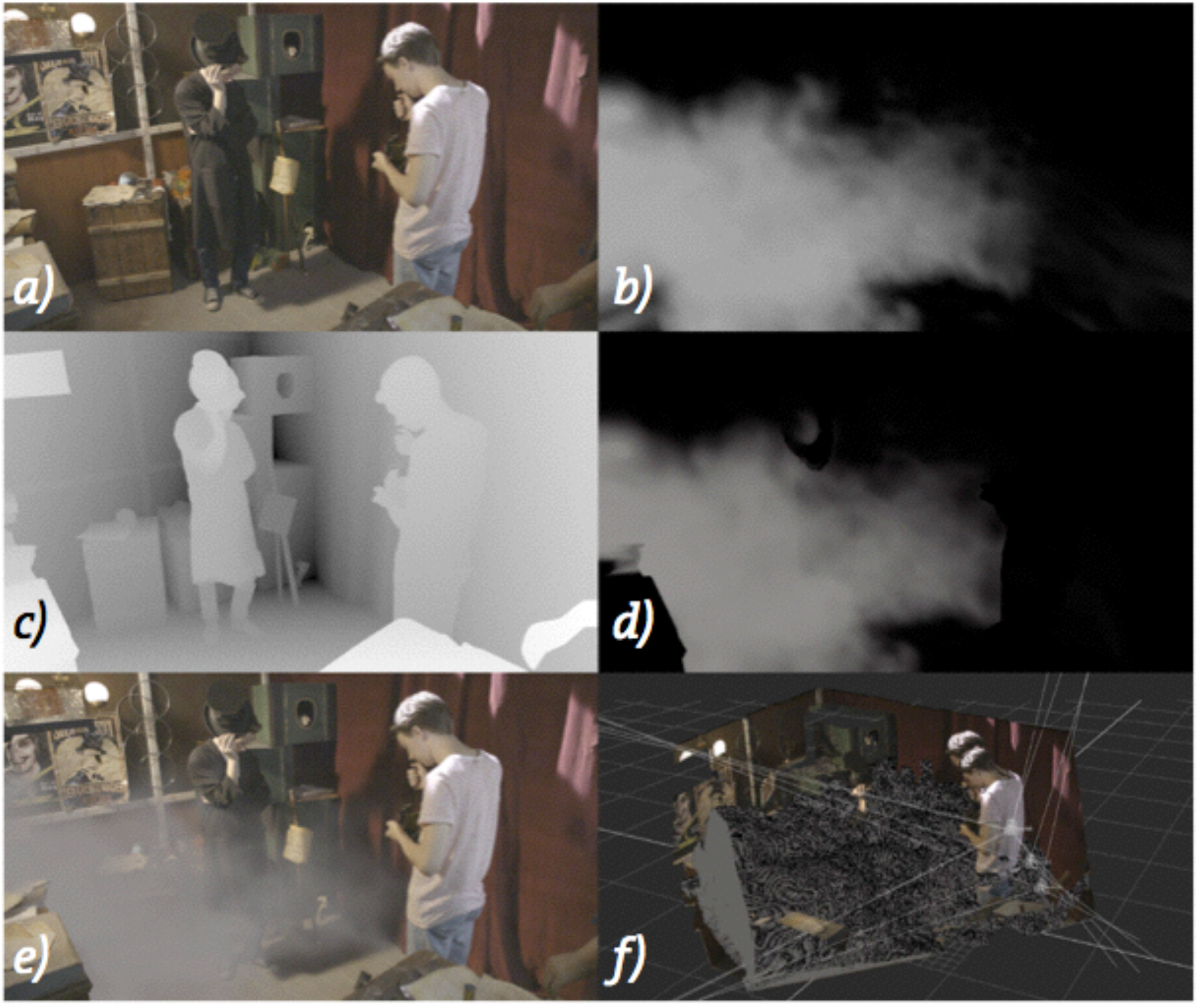


Figure 38: Process of DeepMerge with live action:

Why Use Deep Images?

- Numerous situations where flat image masking is insufficient, can lead to aliased features, blur artifacts, etc.
    - Cause: tools that falsify depth information as a post-process
- Solution: keep the entities separated and pre-composited, but still in a form that a compositing pipeline can manipulate
    - Significant savings over working with the 3D objects, significantly more flexibility over only images
    - Allows for other effects in 3D that would have to be “faked”, e.g. zBlur, depth of field, etc.

Deep Images Allow Combining Volumetric Effects and Shots

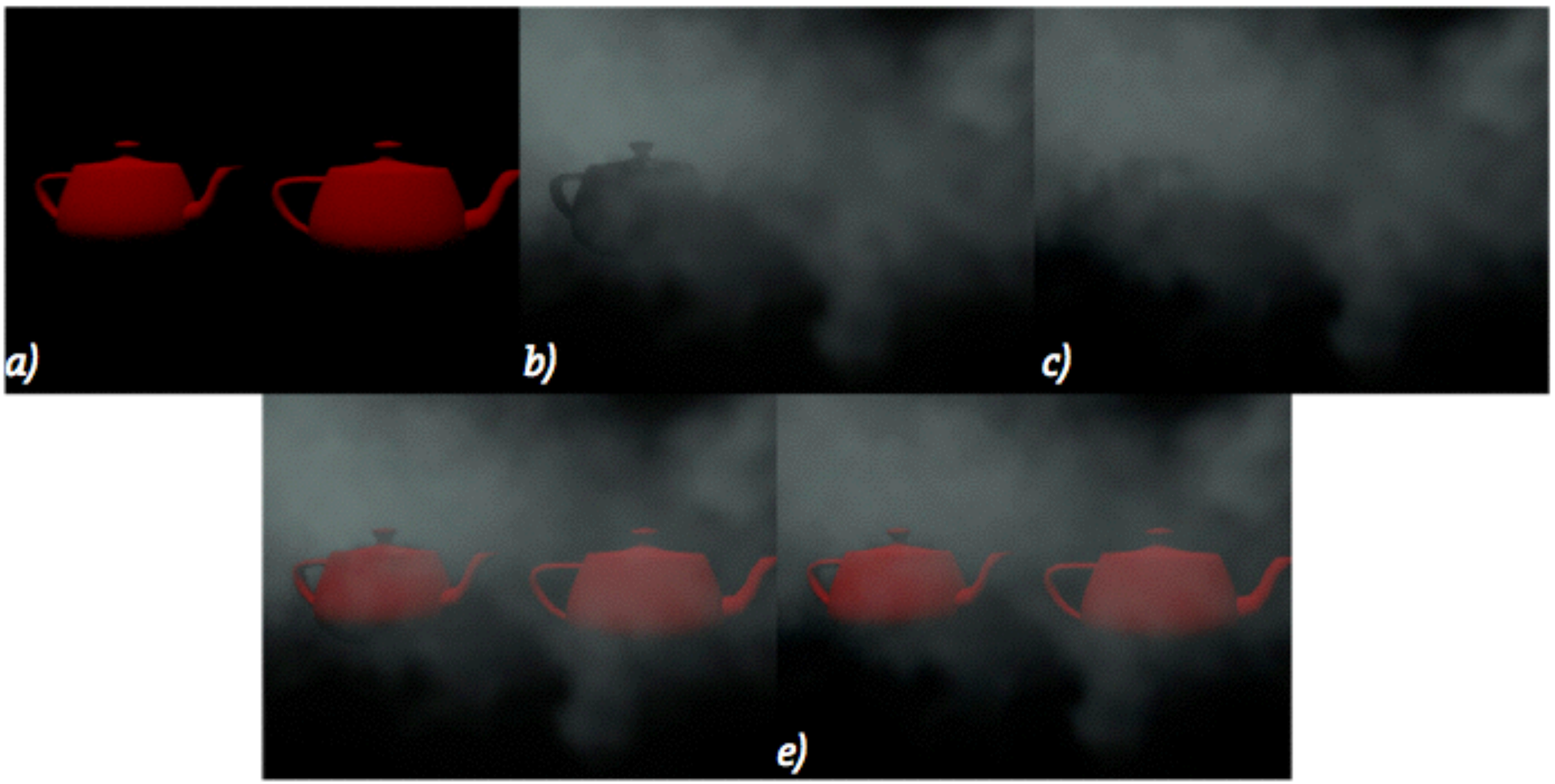


Figure 7: Teapots(a) get merged with fog (c) traditional holdout(b), d) and e) show the possibility to rearrange in compositing

Where Did Deep Images Come From?

Deep Shadow Maps

Tom Lokovic Eric Veach

Pixar Animation Studios*

Abstract

We introduce *deep shadow maps*, a technique that produces fast, high-quality shadows for primitives such as hair, fur, and smoke. Unlike traditional shadow maps, which store a single depth at each pixel, deep shadow maps store a representation of the fractional visibility through a pixel at all possible depths. Deep shadow maps have several advantages. First, they are prefiltered, which allows faster shadow lookups and much smaller memory footprints than regular shadow maps of similar quality. Second, they support shadows from partially transparent surfaces and volumetric objects such as fog. Third, they handle important cases of motion blur at no extra cost. The algorithm is simple to implement and can be added easily to existing renderers as an alternative to ordinary shadow maps.

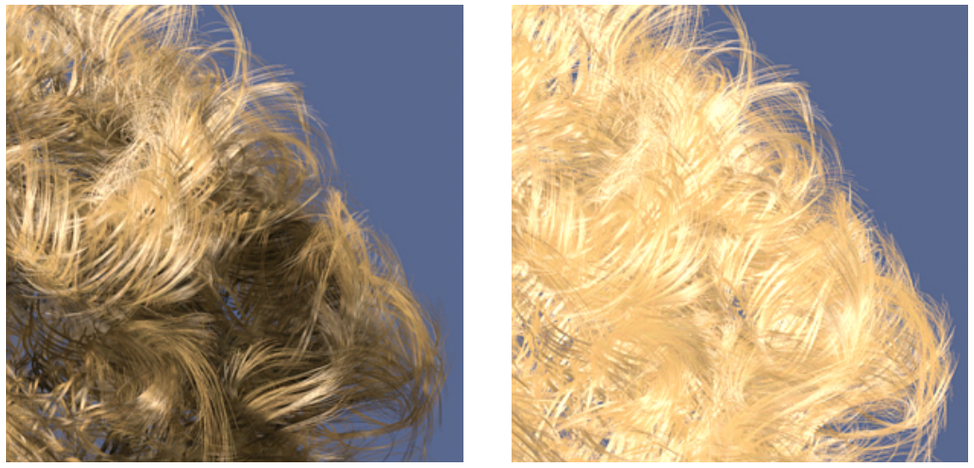


Figure 1: *Hair rendered with and without self-shadowing.*

1 Introduction

Rendering hair, fur, and smoke is difficult because accurate self-shadowing is so important to their appearance [2]. To demonstrate this, Figure 1 shows a small patch of curly hair rendered both with and without shadows. Notice that the shadows cast by portions of the hair onto itself have a great influence on the overall illumination and apparent realism of the rendering.

Traditional shadow maps [11] need very high resolutions to capture this type of self-shadowing accurately. Many more depth samples must also be accessed during shadow lookups to compensate for the higher frequencies in the shadow map. This is especially obvious in animations, where inadequate sampling results in *sparkling* through dense hair and fog.

Compared to ordinary shadow maps, deep shadows have the following advantages:

- They support semitransparent surfaces and volumetric primitives such as smoke.
- For high-quality shadows, they are smaller than equivalent shadow maps by an order of magnitude and are significantly faster to access.
- Unlike ordinary shadow maps, they support mip-mapping. This can dramatically reduce lookup costs when objects are viewed over a wide range of scales.

Where Did Deep Images Come From?

- Deep Shadow Maps use a light transmittance function per pixel over the depth.

- The transmittance function is calculated as a sum of surface and volume transmittance functions per pixel

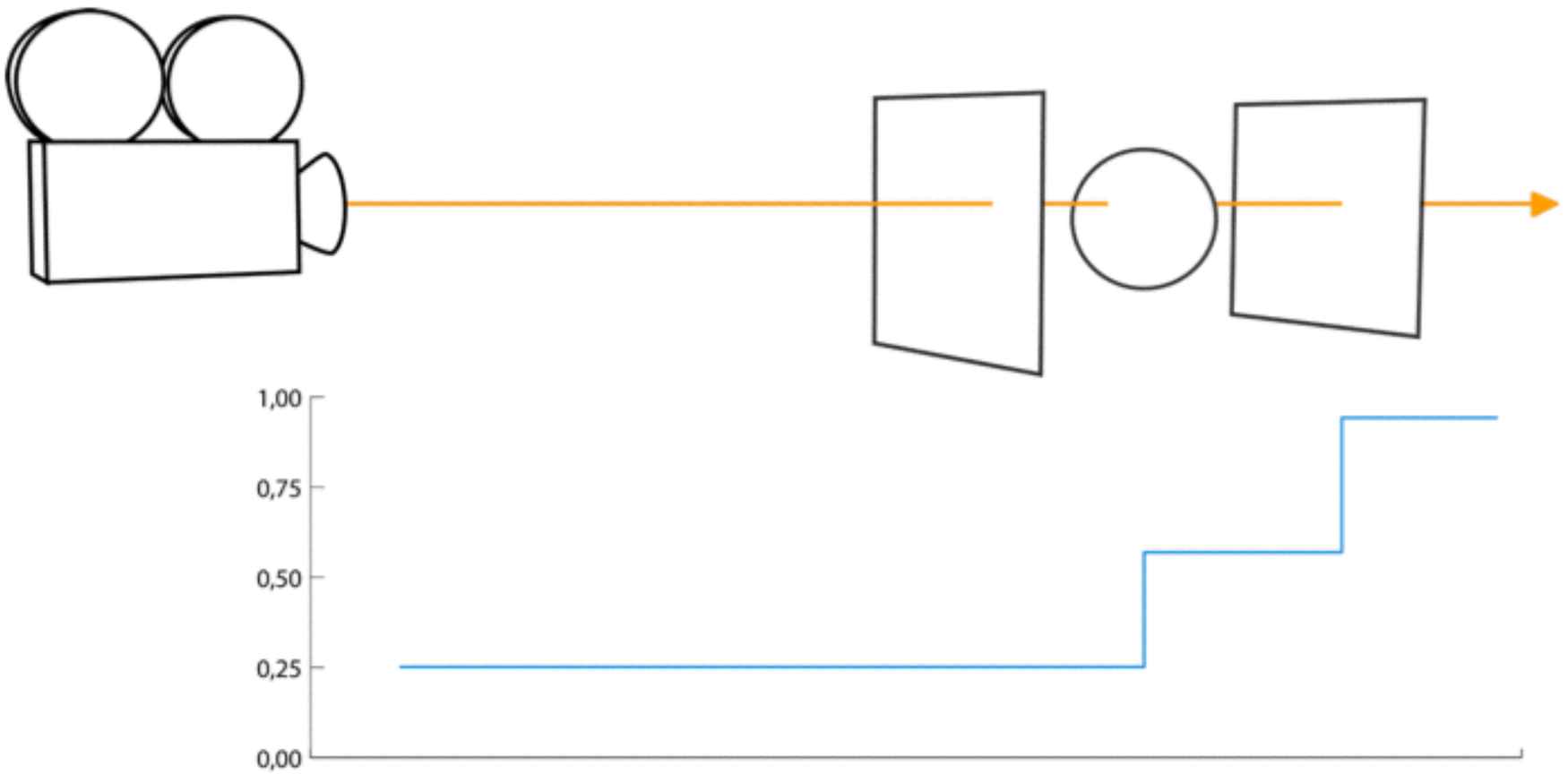


Figure 1: An illustration of the sample gathering proess

Deep Image Data in OpenEXR + OIIO

How Much Space Per Pixel?

- (4*16 = 64 bits) OpenEXR flat images store 4 color channels in half (16-bit) format
- (1*32 = 32 bits) Need to store the number of samples as an unsigned int
- (2*32 = 64 bits) Plus two floats for the front and back of the pixel
- So a single deep pixel is 160 bits. Each additional sample adds another 128 bits (and increments the number of samples.)
    - Note: OpenEXR also supports more than 4 channels per pixel

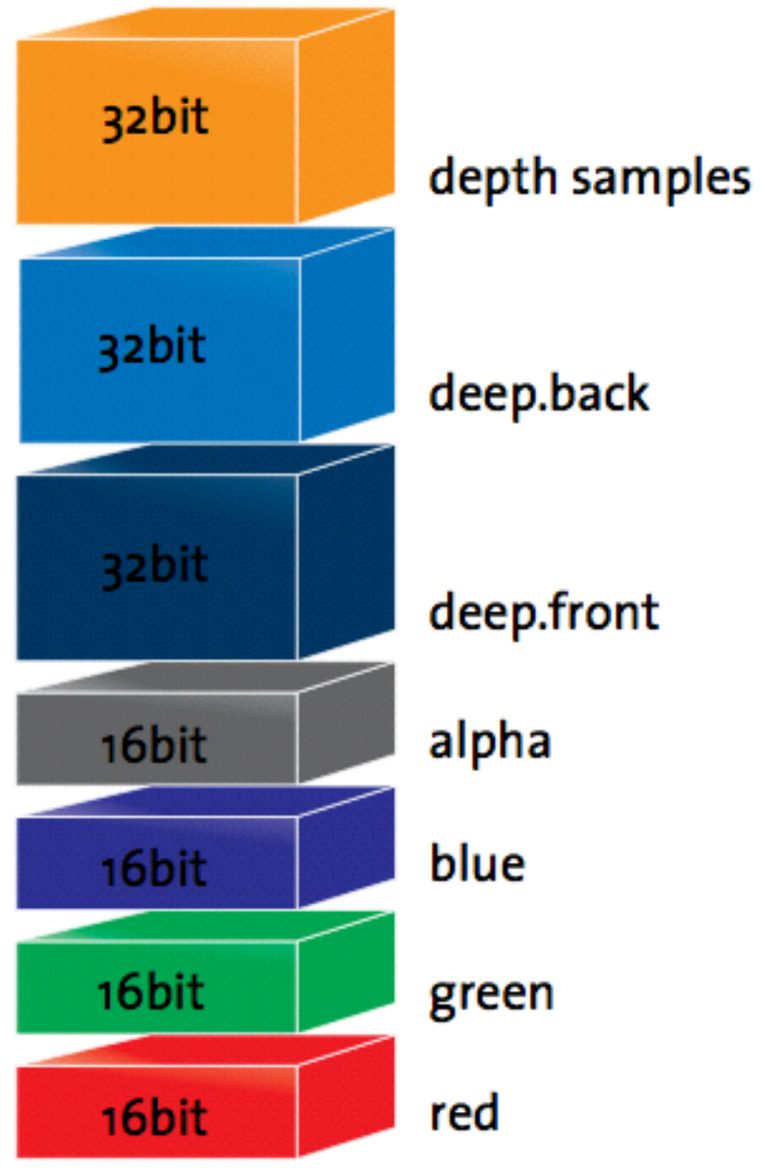


Figure 20: Bit depth of the elements that make up an OpenEXR 2.0 deep image

Reading Deep Images with OIIO

- Cannot use ImageInput::read_image() !
  - Why not? Recall how it is called (will be shown on the next slide)

Example OIIO Read

```
#include <OpenImageIO/imageio.h>
OIIO_NAMESPACE_USING

...

ImageInput *in = ImageInput::open(filename);
if (!in) {
   std::cerr << “Could not create: “ << geterror();
   exit(-1);
}

//after opening the image we can access
//information about it
const ImageSpec &spec = in->spec();
int xres = spec.width;
int yres = spec.height;
int channels = spec.nchannels;

//declare memory, open and read it
unsigned char* pixels = new unsigned char[xres*yres*channels];

//TypeDesc::UINT8 maps the data into a desired type (unsigned char),
//even if it wasn’t originally of that type
in->read_image(TypeDesc::UINT8, &pixels[0]);
in->close();
delete in;
```

Example OIIO Read

```
#include <OpenImageIO/imageio.h>
OIIO_NAMESPACE_USING

...

ImageInput *in = ImageInput::open(filename);
if (!in) {
   std::cerr << “Could not create: “ << geterro
   exit(-1);
}

//after opening the image we can access
//information about it
const ImageSpec &spec = in->spec();
int xres = spec.width;
int yres = spec.height;
int channels = spec.nchannels;

//declare memory, open and read it
unsigned char* pixels = new unsigned char[xres*yres*channels];

//TypeDesc::UINT8 maps the data into a desired type (unsigned char),
//even if it wasn’t originally of that type
in->read_image(TypeDesc::UINT8, &pixels[0]);
in->close();
delete in;
```

With a deep image, how much memory should be allocated?

Reading Deep Images with OIIO

- Cannot use ImageInput::read_image() !
- Cannot statically allocate the right amount of memory because the number of samples can vary pixel-to-pixel.
- Moreover, the channels could be stored with different type formats!

Instead, Read Deep Images into DeepData

```
struct DeepData {
   int npixels, nchannels;

// for each channel [c]
  std::vector<TypeDesc> channeltypes;

// for each pixel [z][y][x]
  std::vector<unsigned int> nsamples;

// for each channel per pixel [z][y][x][c]
  std::vector<void *> pointers;

// for each sample [z][y][x][c][s]
  std::vector<char> data;

   //class functionality
};
```

Instead, Read Deep Images into DeepData

```
struct DeepData {
   //DeepData members

   DeepData ();

// Set the size and allocate the nsamples[] vector.
   void init (int npix, int nchan,
         const TypeDesc *chbegin, const TypeDesc *chend);

// After nsamples[] has been filled in, allocate enough scratch space
// for data and set up all the pointers.
   void alloc ();

// Clear the vectors and reset size to 0.
   void clear ();

// Deallocate all space in the vectors
   void free ();

// Retrieve the pointer to the first sample of the given pixel and
// channel. Return NULL if there are no samples for that pixel.
   void *channel_ptr (int pixel, int channel) const;

// Retrieve sample value within a pixel, cast to a float.
   float deep_value (int pixel, int channel, int sample) const;
};
```

Example OIIO Deep Read

```
#include <OpenImageIO/imageio.h>
OIIO_NAMESPACE_USING

...

ImageInput *in = ImageInput::open(filename);
if (!in) {
   std::cerr << “Could not create: “ << geterror();
   exit(-1);
}

//after opening the image we can access
//information about it
const ImageSpec &spec = in->spec();
int WIDTH = spec.width;
int HEIGHT = spec.height;
int CHANNELS = spec.nchannels;

if (spec.deep) {
   DeepData deepdata;
   in->read_native_deep_image(deepdata);

   int p = 0;  // absolute pixel number
   for (int row = 0; row < HEIGHT; ++row) {
      for (int col = 0; col < WIDTH; ++col) {
         for (int s = 0; s < deepdata.nsamples[p]; ++s) {
            for (int c = 0; c < CHANNELS; ++c) {
               TypeDesc type = deepdata.channeltypes[c];
               //converts the channel value EVEN IF IT WASN’T a float!!
               float value = deepdata.deep_value(p,c,s);
         }  }
         p++; //increment the pixel
   }  }
}

in->close();
delete in;
```

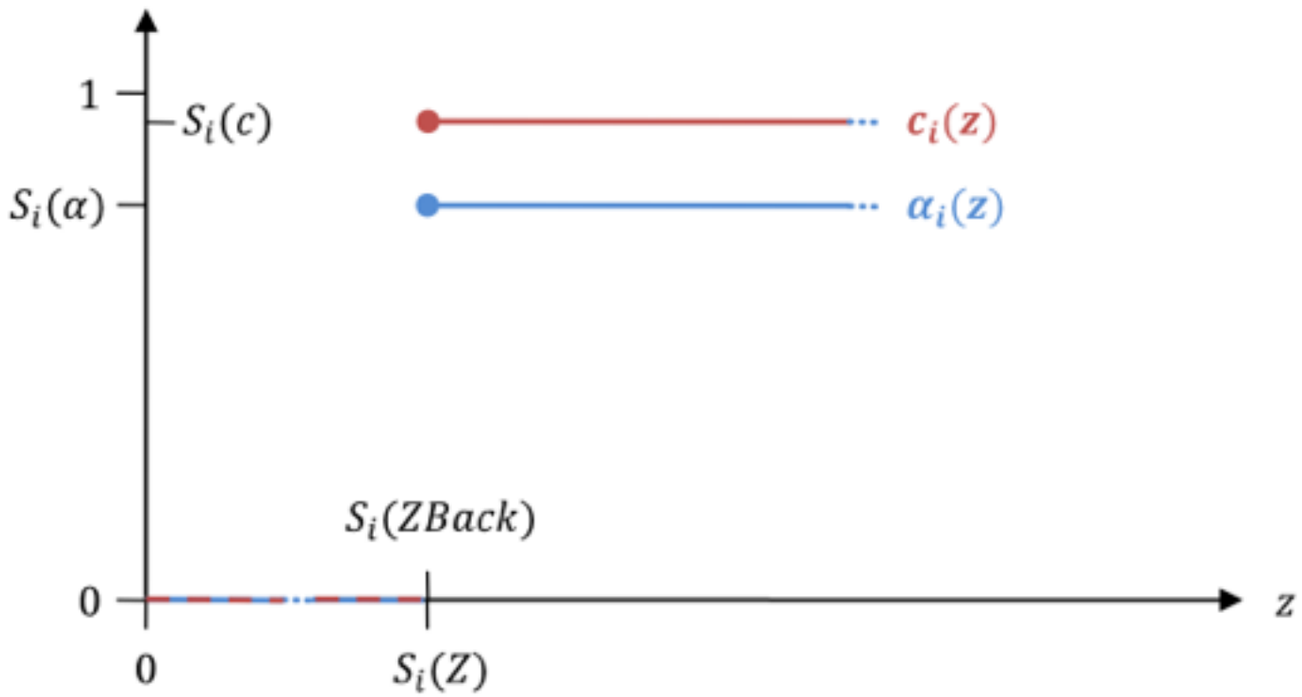
A Modified **over** Operator

One interpretation of a point sample (deep.front = deep.back)

For a point sample, $\alpha_i(z)$ and $c_i(z)$ are step functions:

$$\alpha_i(z) = \begin{cases} 0, & z < S_i(Z) \\ S_i(\alpha), & z \geq S_i(Z) \end{cases}$$

$$c_i(z) = \begin{cases} 0, & z < S_i(Z) \\ S_i(c), & z \geq S_i(Z) \end{cases}$$



- $S_i(c,x,y)$ is the i-th sample of some channel c at pixel (x,y)
- $S_i(c)$ abbreviates
- c could be any type of channel — alpha, Z, etc.

How to Composite a Single Deep Image?

- If it is sorted properly, we have $S_i(Z) < S_j(Z)$ when the sample index $i < j$

  - Interpretation: earlier samples are closer to the camera or in front of later samples.

- Simple way: ignore Z and apply the **over** operator to each channel c

  for(i = number of samples, i > 0; i++) {

  $S_{i-1}(c)$ **over** $S_i(c)$

  }

How to Composite a Single Deep Image?

- More complicated way: attenuate alpha and color based on depth.

Interpreting OpenEXR Deep Pixels

Florian Kainz, Industrial Light & Magic

Updated November 13, 2013

Overview

Starting with version 2.0, the OpenEXR image file format supports deep images. In a regular, or flat image, every pixel stores at most one value per channel. In contrast, each pixel in a deep image can store an arbitrary number of values or samples per channel. Each of those samples is associated with a depth, or distance from the viewer. Together with the two-dimensional pixel raster, the samples at different depths form a three-dimensional data set.

The open-source OpenEXR file I/O library defines the file format for deep images, and it provides convenient methods for reading and writing deep image files. However, the library does not define how deep images are meant to be interpreted. In order to encourage compatibility among application programs and image processing libraries, this document describes a standard way to represent point

How to Composite Multiple Deep Images?

- Need to, per pixel, identify the order in which pixels occur and place one over the other.