Focus Stacking Concept and Test

Author: Andreas Rudolph

Context: Xilinx Kria Challenge

Purpose: Develop and test a prototype of a focus stacking algorithm. As the Wolfram language includes a lot of high level functions that are needed for computational photography, it is better suited than Jupyter notebooks and python in order to study the mathematics behind, and to create a concept. However, in order to be run on a Xilinx Platform the concept needs to be ported into python.

Source of Information for this is the following article in the Wolfram knowledge base Source https://www.wolfram.com/language/12/image-computation-for-microscopy/3d-reconstruction-via-focus-stacking.html.en?footer=lang

The images used here are downloaded from Wikipedia: https://de.wikipedia.org/wiki/Focus_stacking

---

Focus Stacking

Several imaging application areas require images that show each detail in focus (i.e. microscopy, or macro photography). For technical-physical reasons it is sometimes not possible to take such an image in one shot. In such cases photographers often take a stack of images in which the images differ in the focus plane. These stacks are then combined into one image in the postprocessing step.

Typical focus stacking process

The typical process consist of the following steps:

- Take a focus stack that includes several images with variations in the focus plane.
- Align each image in the stack to one reference image within the stack.
- Calculate which parts of each image are in-focus and assign per-pixel weights that represent these parts.
- Calculate the image average of these areas and combine these parts info one image.

To identify in-focus pixels in an image, information about local sharpness is needed. Typically the Laplacian operator is used for this calculation, because the size of the operator directly indicates the sharpness of the pixels (the higher, the sharper). Gaussian blur is then added as the nearby pixels need to have similar weights.

The calculation of the weights depend on the method used for focus measurement. The depth from focus determines the sharpest pixel in a focal stack. Cameras that use the contrast method produce focus stacks that can be used to calculate depth maps, and with this all in-focus photos. For other methods each image and pixel the Laplacian is multiplied with the Gaussian blur. The result is added, and finally divided by the sum of weights.

Focus Stacking in the Wolfram language - 3D Reconstruction via Focus Stacking

For technical and optical reasons, microscopes, or photos taken with a macro lense have a shallow depth of field. Therefore the parts of the image located in front of the focal plane or behind it are out of focus and blurred. For some applications however it is important that the image is in focus from rear to front.

Focus stacking is an image processing method that supports this requirement. With this technique, a stack of images with different focal lengths is combined into one image. This is done by selectively collecting in-focus regions from each image in the stack.

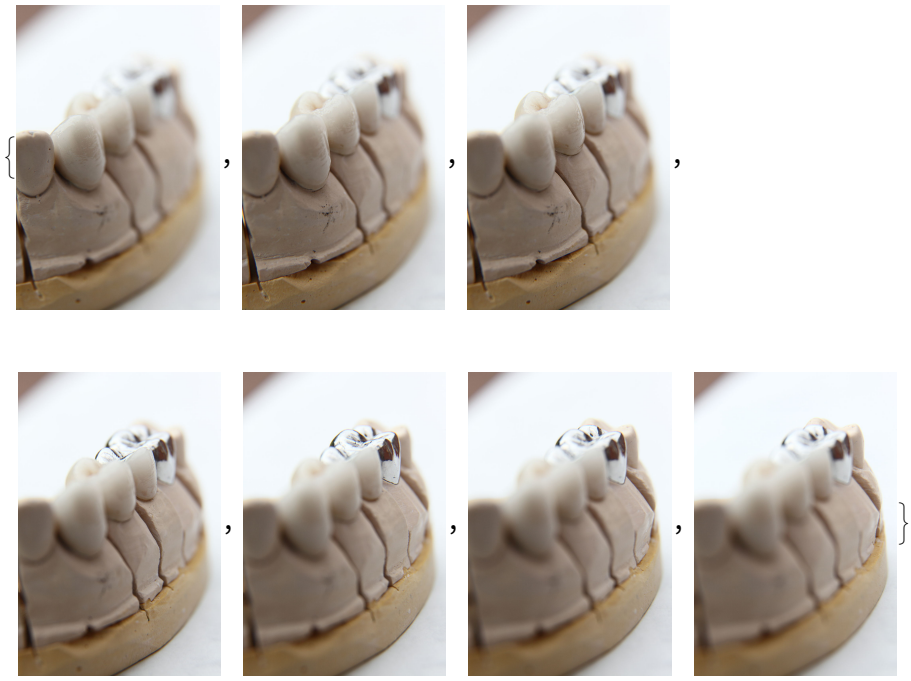
Preprocessing

In the first step, this notebook imports the sequence of images, align it to the first image, and show the sequence.

Import photos from the stack

In[12]:=
```
dir = FileNames[All,
    "/Users/andreasrudolph/Documents/Wolfram Mathematica/WOLFRAM_OWNWORK/
      FocusStack/Images/"];
stack = Map[Import, dir]
```

Out[13]=



The main picture from the stack serves as the basis for image alignment further below

In[25]:=
```
baseLine = stack[[1]];
```

In the images above, observe how in this stack the focus plane changes from image to image. The task is now to identify and combine the most sharpest areas of each image into a combined image. This can be done with the Wolfram standard command “ImageFocusCombine”. Observe that the complete image is in focus in all image planes.

In[ ]:=
```
DynamicImage[ImageFocusCombine[stack], ImageSize → 400]
```

Out[ ]=

Create a stacked image

However, “ImageFocusCombine”, which is build into the Wolfram Language, is not available in Python, and the command will be decomposed into their individual process steps.

First, “ImageFocusCombine[]” needs to identify in-focus pixels in each frame. To identify these pixels, information about the local sharpness is needed. For this task, the norm of the Laplacian operator is used, while the size of the operator indicates sharpness. The Laplacian filter picks up the high Fourier coefficients. If an image is out of focus, these coefficients are subdued. The Gaussian blur is added as the nearby pixels need to have similar weights.

As by the documentation of the Wolfram language, the “*LaplacianFilter is commonly used in image processing to highlight regions of rapid intensity change by approximating the second spatial derivatives of an image.*” The “*LaplacianGaussianFilter is a derivative filter that uses Gaussian smoothing to regularize the evaluation of discrete derivatives. It is commonly used to detect edges in images.*”

Calculate and display focusResponses

The following code maps each color channel of the image in the stack to the Laplace gaussian filter at a given radius and a defined standard deviation, and takes the Norm of the resulting value.

In[28]:=
```
focusResponses =
  Map[img ↦ Norm[ColorSeparate[LaplacianGaussianFilter[img, {3, 1}]]], stack];
```

In the notebook version of this notebook, it is possible to show an animation of the adjusted images. The following transaction adjusts the levels in each image, and rescales them to cover the range from 0 to 1.

In[33]:=
```
ListAnimate[ImageAdjust /@ focusResponses,
  ImageSize → 480, AnimationDirection → Forward];
```

Identify sharpest areas in an image

For each pixel, we now pick the layer that exhibits the largest Laplacian filter norm, as this value indicates the highest degree of sharpness. In the following code, “nonMaxChannelSuppression” represents a forward function that takes each image of the stack as an input. In the first part, the code calculates a 3D image from the focus stack. Afterwards this 3D image returns a 2D image of maximum projection onto the *x-y* plane as a variable “max”. The second part substracts the constant amount of the variable “max” from each channel value in *image*, and applies it to a unit step function.

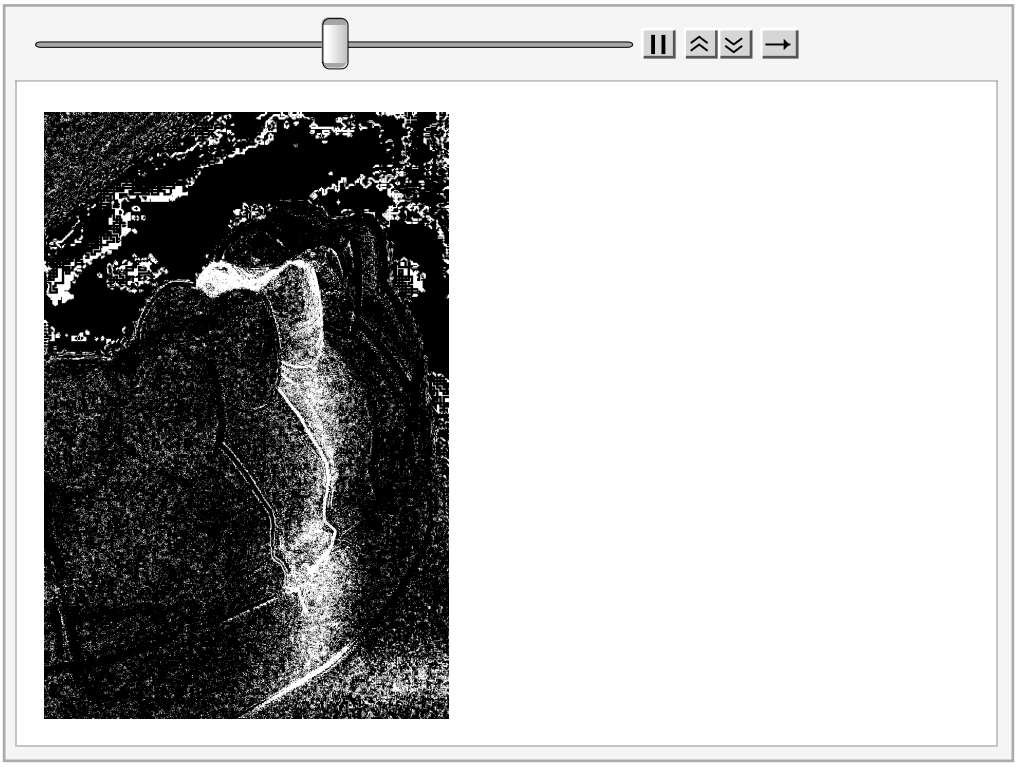
In[30]:=
```
nonMaxChannelSuppression[stack : {__Image}] :=
  With[{max = Image3DProjection[Image3D[stack], Top, "Max"]},
   Map[img ↦ UnitStep@ImageSubtract[img, max], stack]];
```

Using this forward function and the variable “focusResponses”, the “depthVol” volume contains the depth information of each pixel and is therefore able to define which are the sharpest areas of the images.

In[31]:=
```
depthVol = nonMaxChannelSuppression[focusResponses];
```

In[32]:=
```
ListAnimate[depthVol, ImageSize → 480, AnimationDirection → Forward]
```

Out[32]=



Calculate the stacked image

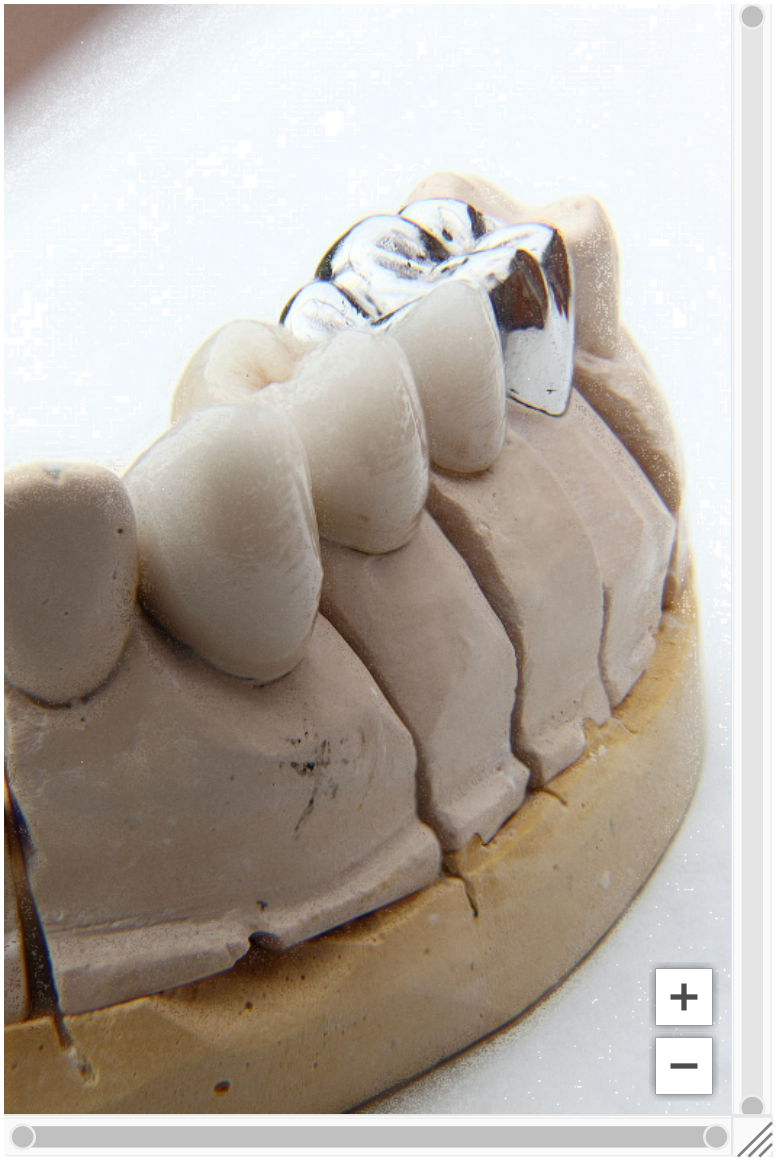
The resulting image stored in the variable “inFocus” is created by multiplying the resulting binary depth volume with the focal stack, and by adding up all layers. This operation only collects those pixel values that are in focus.

In[34]:=
```
inFocus = stack.depthVol;
```

This let’s you see the image interactively.

In[35]:=
```
DynamicImage[inFocus, ImageSize → 400]
```

Out[35]=



Observe that this image is sharp in all image planes, as the stacked image further above. If you compare this stacked image with the version that is created with ImageFocusCombine[], you can observe differences in the details as well as artifacts, which do not exist with “ImageFocusCombine”. See in particular the background or the last tooth.

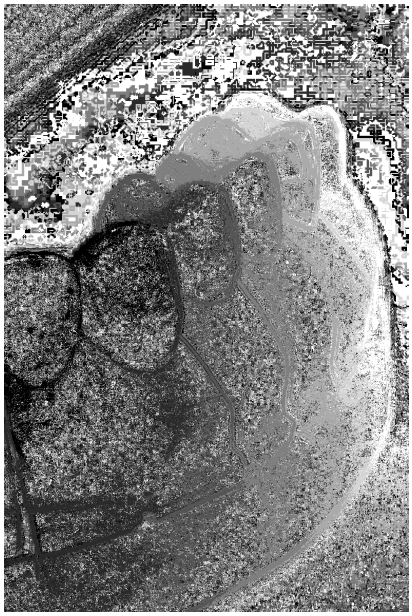
Calculation of depth maps

With the same technique and the variables calculated before, it is possible to calculate a 3D version of the image.

The variable/ volume “depthVol” calculated above, contains the depth information of each pixel of the images. The following code converts this volume into a two-dimensional depth map, named “depthMap”. According to the documentation for the Wolfram language, the function “ImageClip” thereby “clips all channel values in *image* to lie in the default range”. “Identity” is the Identity operation, and the “array” function generates a list of the length of the volume depthVol using values from 0 to 1.

In[•]:=
```
depthMap = ImageClip[depthVol.Array[Identity, {Length[depthVol]}, {0., 1.}]]
```

Out[•]=



As you can see, the depthMap is noisy and it is not not equally reliable for all pixel locations. If you compare the depthMap to the stacked image from above, you see that only the edges provide a clear indication if an image region is in focus or not.

The focusResponses variable calculated above stores the Norm of the LaplacianGaussianFilter applied to the color separated channels of the image. The variable “confidenceMap” totals these “focusResponses”

In[•]:=
```
confidenceMap = Total@focusResponses;
```

In the Wolfram language, the TimesBy operation x*=c multiplies x by c and returns the new value of x. Therefore the following code corrects the depthMap while it only takes depth measures into account that are larger than a given threshold.

In[•]:=
```
depthMap *= Binarize[confidenceMap, 0.05]
```

Out[•]=

The depth values can be regularized with the MedianFilter and you close gaps via the FillingTransform command.

In[•]:=
```
depthMap = FillingTransform@MedianFilter[depthMap, 3]
```

Out[•]=

Introduction of an image alignment step

Preprocessing

Above, we have imported the sequence of images, and worked with it right away.

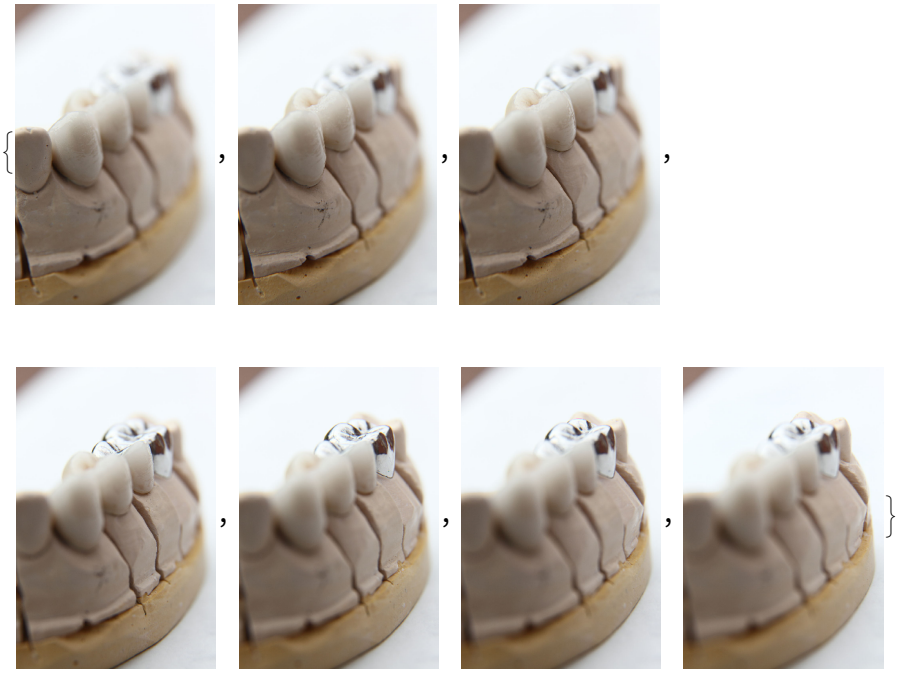
Here we first align the images , show them, and then work with these images:

In[36]:=
```
alignedstack = ImageAlign[baseLine, stack , TransformationClass → "Translation"];
```

In order to prevent that artifacts are introduced by the image alignment step, the command ImagePad is used, which pads the image with a number of background pixels.

In[37]:=
```
stack = ImagePad[#, -7] & /@ alignedstack
```

Out[37]=



In[38]:=
```
ListAnimate[stack, ImageSize → 480, AnimationDirection → Forward];
```

In the images above, observe how in this stack the focus plane changes from image to image. The task is now to combine the most sharpest areas of each image into a combined image. This can be done with the Wolfram standard command “ImageFocusCombine”. However, this command is not available in Python. Therefore the command will be decomposed into their individual process steps.

In[39]:=
```
DynamicImage[ImageFocusCombine[stack], ImageSize → 400]
```

Out[39]=

Create a stacked image

As above we create the stacked image from a series of images that differ in terms of focus. An identical process is used.

Calculate the focusResponses and Identify sharpest areas in an image

The Laplace gaussian filter works at a riven radius and standard deviation. However, both values determine how much pixel errors occur.

In[40]:=
```
focusResponses =
  Map[img ↦ Norm[ColorSeparate[LaplacianGaussianFilter[img, {6, 1}]]], stack];
nonMaxChannelSuppression[stack : {__Image}] :=
  With[{max = Image3DProjection[Image3D[stack], Top, "Max"]},
   Map[img ↦ UnitStep@ImageSubtract[img, max], stack]];
depthVol = nonMaxChannelSuppression[focusResponses];
ListAnimate[depthVol, ImageSize → 480, AnimationDirection → Forward];
```

Calculate the stacked image

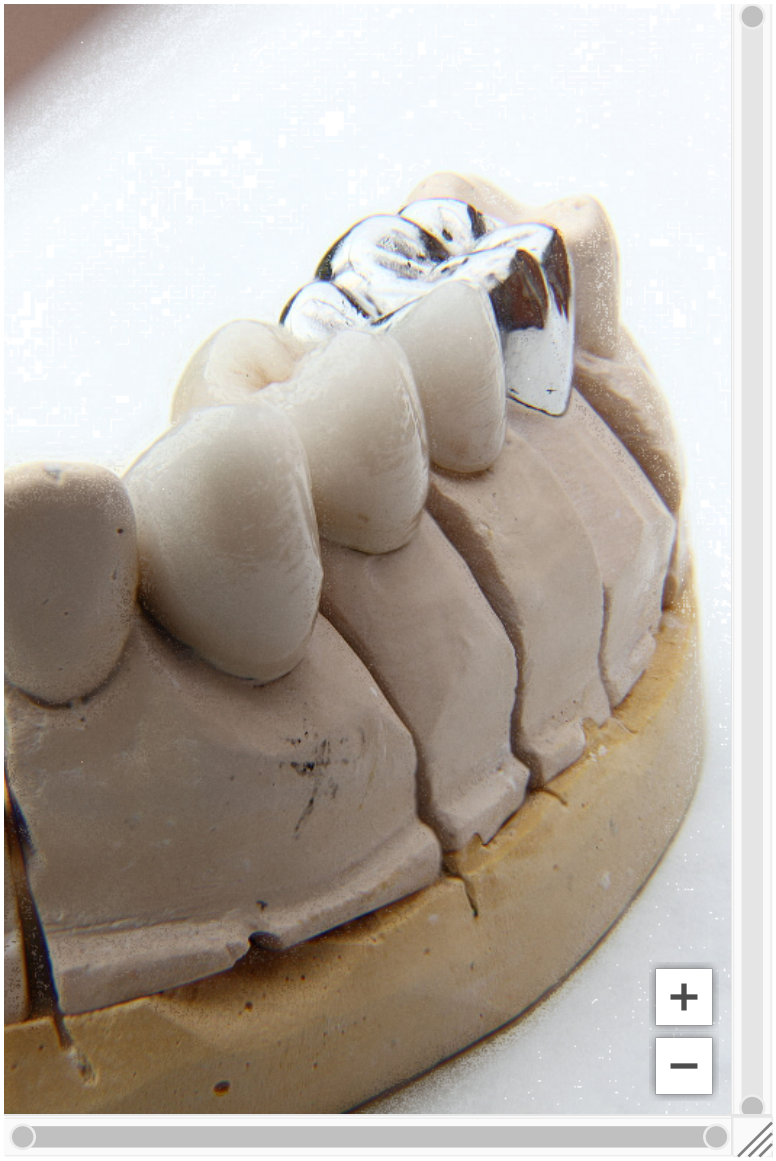
As before, the resulting image “inFocus” is created by multiplying the resulting binary depth volume with the focal stack, and by adding up all layers. This operation only collects those pixel values that are in focus.

In[•]:=
```
inFocus = stack.depthVol;
```

This let’s you see the image interactively. Observe that the image is sharp in all image planes, as the versions produced above. However, the alignment process has created artifacts and pixel errors.

In[•]:=
```
DynamicImage[inFocus, ImageSize → 400]
```

Out[•]=



Postprocessing

Inorder to evaluate the image artifacts, the following section we apply post processing steps, such as masking or brightness corrections. As you see this makes the pixel errors more visible, and it demonstrates cyclic errors in the background, that are probably already available in the original jpg images.

In[44]:=
```
mask = Erosion[ColorNegate@Binarize[ColorDistance[inFocus, Gray], 0.5], 1];
Lighter[ColorCombine@Map[inFocus ↦ BrightnessEqualize[inFocus, Masking → mask],
    ColorSeparate[inFocus]]]
```

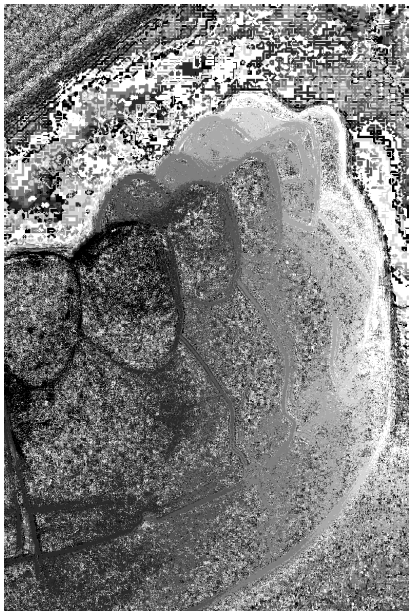
Out[45]=

Depth Maps

The following commands and variations further elaborate how the sharpness is distributed over the stacked version of the image. As you see not all edges are closed, and some areas of the filtered image include pixels, and other distorsions.

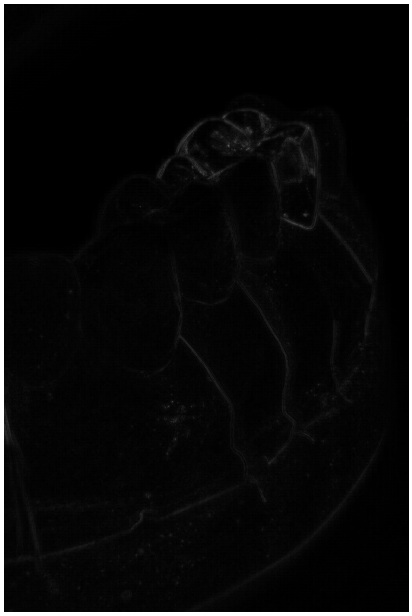
In[•]:=
```
depthMap = ImageClip[depthVol.Array[Identity, {Length[depthVol]}, {0., 1.}]]
depthMap = EntropyFilter[depthMap, 5]
confidenceMap = Total@focusResponses
depthMap *= Binarize[confidenceMap, 0.05]
depthMap = FillingTransform@MedianFilter[depthMap, 3]
```

Out[•]=



Out[•]=

Out[ ]=



Out[ ]=

Out[•]=

Summary - Focus Stacking

As you see above, the following process is able to produce a sharp stacked image out of a list of images that differ in focus plane.

- Take a focus stack that includes several images with variations in the focus plane.
- Align each image in the stack to one reference image within the stack.
- Calculate which parts of each image are in-focus and assign per-pixel weights that represent these parts.
- Calculate the image average of these areas and combine these parts info one image.

To identify in-focus pixels in an image, the local sharpness is calculated using the Laplacian operator. The size of the operator directly indicates the sharpness of the pixels (the higher, the sharper). Gaussian blur is then added to make sure that nearby pixels have similar weights. Then each image and pixel the Laplacian is multiplied with the Gaussian blur. The result is added, and finally divided by the sum of weights.

Depending on the inbound images, the resulting image may include image errors, and therefore the process requires postprocessing steps.