

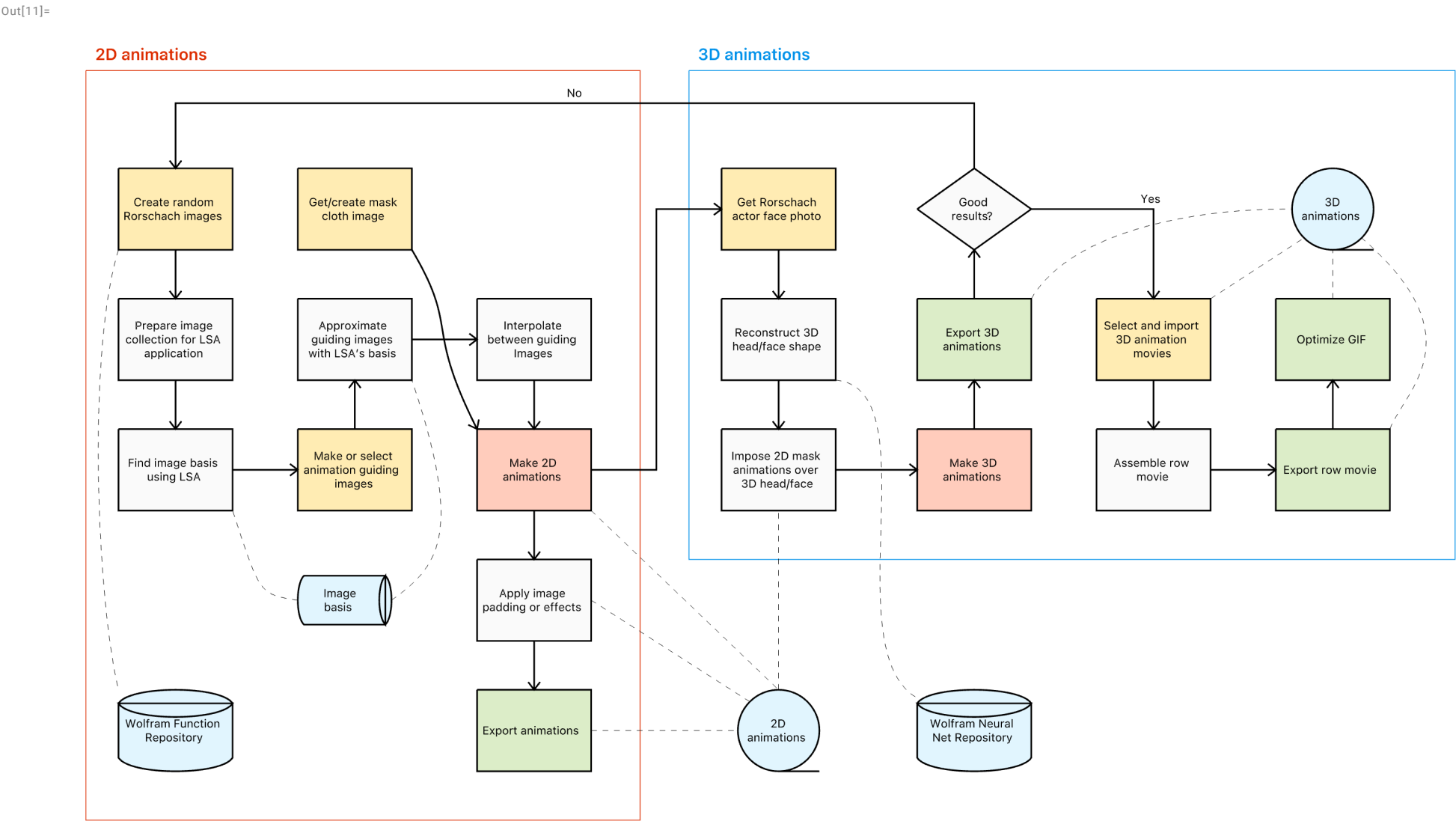
# Rorschach mask animations

Version 0.4

Anton Antonov  
MathematicaForPrediction at WordPress  
MathematicaForPrediction at GitHub  
SimplifiedMachineLearningWorkflows-book at GitHub  
April, October 2022

## Introduction

In this notebook we show a method to make animations of Rorschach images and project those animations over 3D surfaces. The work is inspired by the visual effects of Rorschach's mask in the 2009 movie "Watchmen".  
See the gallery "Attempts to recreate Rorschach's mask" for examples.  
The following flow chart summarizes the "creation loop":



The workflow has two main parts:

1. Making of 2D Rorschach animations
2. Imposing the 2D images over 3D (head) surfaces

**Remark:** More detailed explanations will be added in the subsequent versions of the notebook.

## "Why it worked?" summary

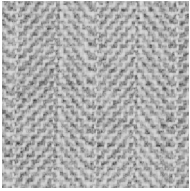
The reasons this "worked" are:

- WL functionalities
  - Comprehensive image processing and image effects functionalities
  - "Out of the box" parallel programming functionalities
    - Without parallel computations making the experiments would have been too slow for me to come up with something "good" within art competition's deadlines.
  - Fast Singular Value Decomposition
  - "Out of the box" neural network functionalities
- Computational elements
  - Previous work on random graphics generation (mandalas, scribbles)
    - Curation, documentation, and availability at Wolfram Function Repository (WFR)

- Having sound design of computation module is important if solutions have to be delivered reliably and quickly.
- Previous work on dimension reduction techniques
  - Latent Semantic Analysis (LSA) representation is uniquely fit for these kind of effects
- The 3D face reconstruction neural network and related documentation
  - Available through Wolfram's repositories / sites

## Mask cloth

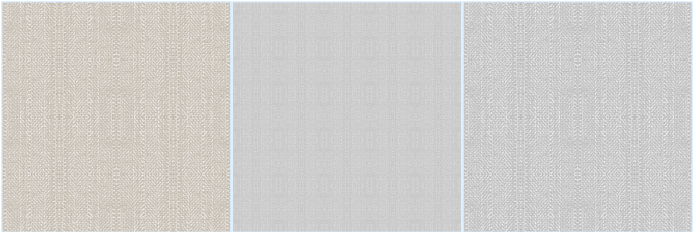
Here we get a cloth texture image and make a wallpaper with it. (Upon that wallpaper the animations are overlaid below.)

```
In[12]:= clothOriginal =  // ImageAdjust;
```

```
In[13]:= cloth = ColorReplace[clothOriginal, Black -> GrayLevel[0.8], 0.65];
clothSq = ImageAssemble[{cloth, ImageReflect[cloth, Left -> Right]}];
clothSq = ImageAssemble[{{clothSq}, {ImageReflect[clothSq, Top -> Bottom]}]];
clothFine = ImageAssemble@Table[clothSq, {5}, {5}];
cloth = ImageAssemble@Table[clothSq, {2}, {2}];
```

```
In[18]:= clothSepia = ImageEffect[cloth, "Sepia"];
```

```
In[19]:= ImageCollage[{cloth, clothFine, clothSepia}, Method -> "Rows", ImageSize -> Medium, ImagePadding -> 5, Background -> LightBlue]
```

```
Out[19]= 
```

## 2D Rorschach animations

In this section we do the following steps:

1. Generate training Rorschach images
  - 1.1. Using Wolfram Function Repository functions RandomScribble or RandomRorschach.
2. Apply Latent Semantic Analysis (LSA) workflows and extract image topics.
  - 2.1. The image topics make an image basis that can be used to approximated other images. See [AA2 and AA5 for details.]
3. Generate or obtain animation guiding images.
4. Interpolate between animation guiding images.
  - 4.1. The interpolation is done by interpolating between the image basis vectors.
5. Make animations
6. Apply image effects

### Get data

Here we generate the training images data. (We experiment with small image sizes, and we get good results we large image sizes.)

```
In[20]:= SeedRandom[902];
aRes = PrepareImageData["NumberOfImages" -> 700,
  ImageSize -> 200, "Dilation" -> False, "Generator" -> Hold[ResourceFunction["RandomScribble"] [
    "NumberOfStrokes" -> RandomChoice[{10, 2, 2} -> {RandomInteger[{5, 20]}, {12, 40}, {20, 16}}],
    "RotationAngle" -> RandomChoice[{Automatic, RandomReal[{0,  $\pi$ ], 3}], "OrderedStrokePoints" -> False,
    "ConnectingFunction" -> FilledCurve@*BezierCurve, ColorFunction -> (Black &), ImageSize -> Small]]];
{6.86957, Null}
Apply Rorschach effect:
{0.647298, Null}
Image resize:
{0.112655, Null}
Color convert to "GrayScale":
{0.271954, Null}
```

```
In[22]:= aDImages = aRes["Images"];
aDImageVecs = aRes["ImageVectors"];
```

**Remark:** The function PrepareImageData is from the package [AAp1].

## Apply LSA

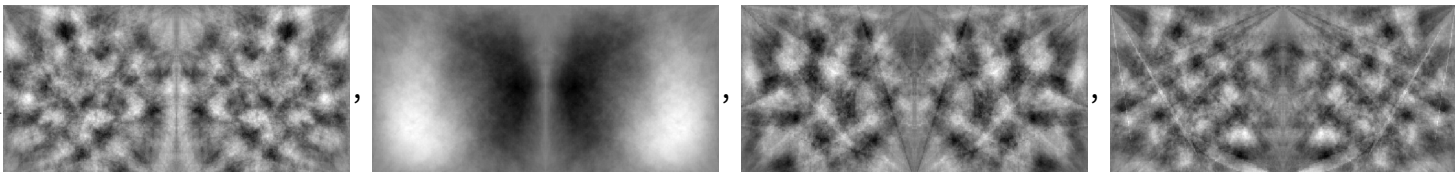
Here we apply LSA dimension reduction. (See [AA1-AA5].)

```
In[47]:= AbsoluteTiming[
  lsaObj = DimensionReductionWithLSA[aImageVecs];
]
```

```
Out[47]= {4.48991, Null}
```

Get the image basis and show a sample:

```
In[50]:= aLSABasis = GetImageBasis[lsaObj, aImages];
RandomSample[aLSABasis["ImageBasis"], 4]
```

```
Out[51]= {}
```

## Make animation through path









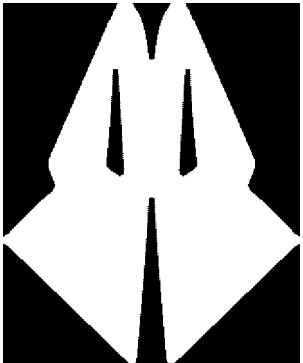
Here we sample the training images in order to get the animation guiding images:

```
In[52]:= (*SeedRandom[781];
  aPath=Sort[RandomSample[aImages,9]]*)
```

Alternatively, we create animation guiding images on the spot:

```
In[53]:= SeedRandom[78];
aPath = Sort[Association@Table[i → ColorNegate[ResourceFunction["RandomRorschach"][
  "NumberOfStrokes" → RandomChoice[{10, 2, 2} → {RandomInteger[{5, 20}], {12, 40}, {20, 16}}], "RotationAngle" →
  RandomChoice[{Automatic, RandomReal[{0,  $\pi$ }, 3]}], "OrderedStrokePoints" → RandomChoice[{False, True}],
  "ConnectingFunction" → FilledCurve@*BezierCurve, ColorFunction → (Black &)]], {i, 9}]]];
```

aPath

```
Out[55]= <| 2 →  , 5 →  , 7 →  ,
4 →  , 6 →  , 8 →  ,
3 →  , 1 →  , 9 →  >|
```

```
In[56]:= (*ImageCollage[Values[aPath],Method->"Grid",Background->LightBlue,ImagePadding->4]*)
```

```
In[57]:= AbsoluteTiming[
  aSampleCoeff = Map[RepresentImage[lsaObj, aImages[[1]], #] &, aPath];
]
Length[aSampleCoeff]
```

```
Out[57]= {7.33631, Null}
```

```
Out[58]= 9
```

```
In[59]:= AbsoluteTiming[
  lsCoeffList =
    Map[InterpolatedCoefficientMutation[#[[1]], #[[2]], 100] &, Partition[Append[Values@aSampleCoeff, aSampleCoeff[[1]], 2, 1]];
  lsCoeffList = Flatten[lsCoeffList, 1];
]
Dimensions[lsCoeffList]

Out[59]=
{0.001402, Null}

Out[60]=
{900, 80}

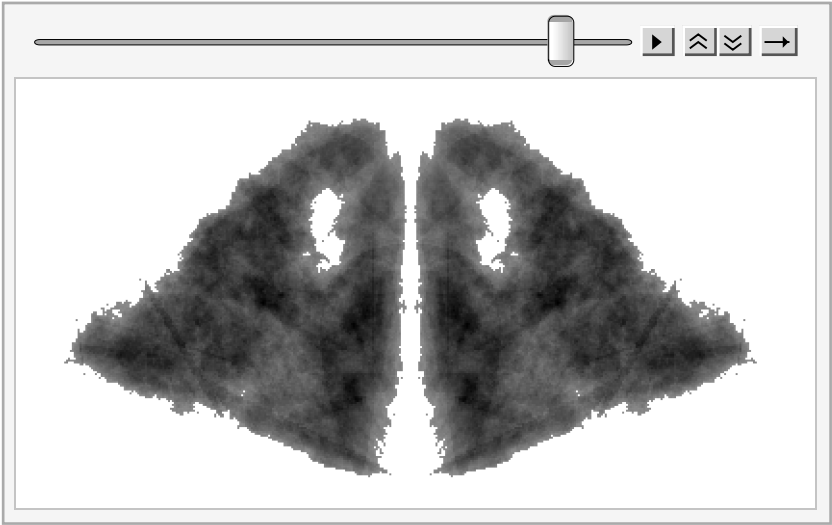
In[61]:= math = SparseArray[aLSABasis["H"]];
```

Rough-looking approximations

```
In[37]:= AbsoluteTiming[
  lsMotionRorschachs =
    ParallelMap[(
      vecRepresentation = #.math;
      ColorNegate@Dilation[#, 0.3] &@
      Image[Clip[#, {0.45, 1}, {0, 1}] &@Rescale[Partition[vecRepresentation, ImageDimensions[aDImages[[1]][[1]]]]]) &,
    lsCoeffList, Method -> Automatic];
]

Out[37]=
{4.69652, Null}

In[38]:= ListAnimate[lsMotionRorschachs]
Out[38]=
```



```
In[39]:= (*ListAnimate[ImageResize[#,600]&/@lsMotionRorschachs]*)
```

OilPaiting effect

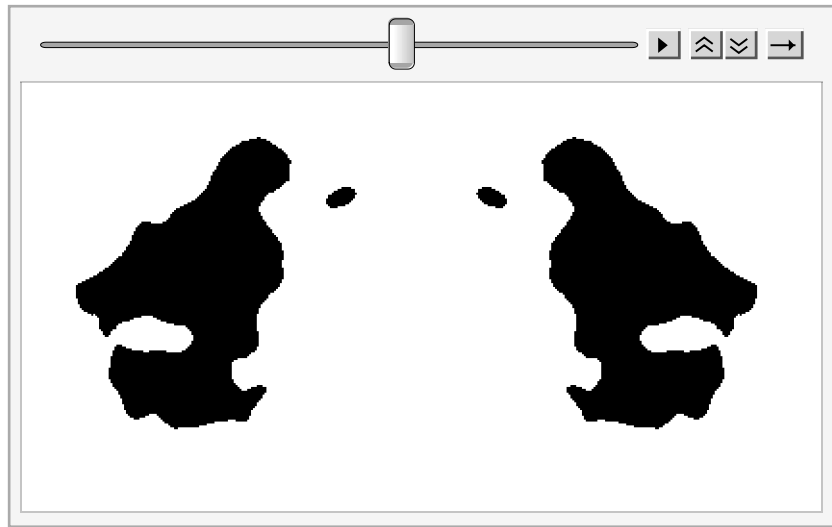
Here we redo the image manipulation with additional binarization in order to get "inkblot look" after applying "OilPainting" effect.

```
In[62]:= AbsoluteTiming[
  lsMotionRorschachsForOil =
    ParallelMap[(
      vecRepresentation = #.math;
      ColorNegate@Dilation[#, 0.3] &@Image[
        Unitize@Clip[#, {0.55, 1}, {0, 1}] &@Rescale[Partition[vecRepresentation, ImageDimensions[aDImages[[1]][[1]]]]]) &,
    lsCoeffList, Method -> Automatic];
]

Out[62]=
{4.28703, Null}

In[63]:= AbsoluteTiming[
  lsMotionRorschachsOil = ParallelMap[ImageEffect[#, "OilPainting"] &, lsMotionRorschachsForOil];
]
```

```
In[89]:= ListAnimate[lsMotionRorschachsOil]
Out[89]=
```



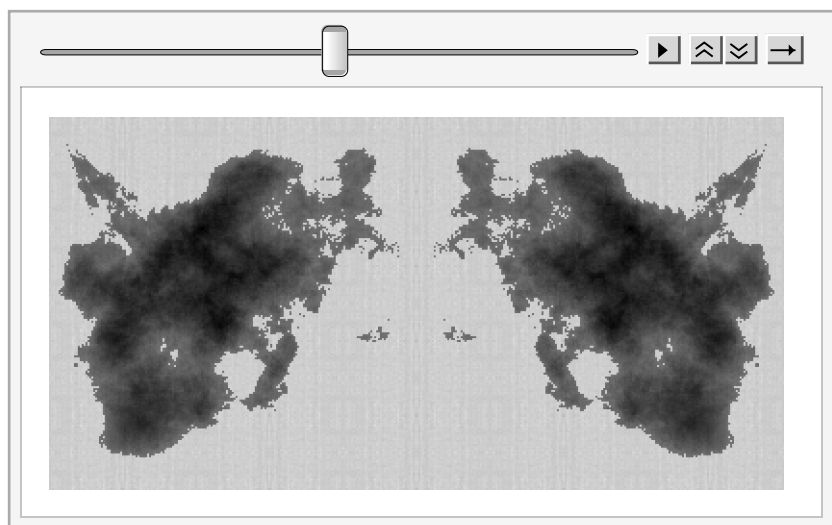
## Over (non-sepia) cloth

In this sub-section we place the animation images over the mask cloth image derived above.

```
In[64]:= clothSepia2 = ImageResize[clothFine, ImageDimensions[lsMotionRorschachs[[1]]]];
In[65]:= AbsoluteTiming[
  lsMotionRorschachs2 = ParallelMap[ImageMultiply[clothSepia2, #] &, lsMotionRorschachs];
]
Out[65]= {0.292969, Null}

In[66]:= AbsoluteTiming[
  lsMotionRorschachs3 = ParallelMap[ImageMultiply[clothSepia2, #] &, lsMotionRorschachsOil];
]
Out[66]= {0.321747, Null}

In[67]:= ListAnimate[lsMotionRorschachs2]
Out[67]=
```



## Head-mask padding

In order to produce more "centered" animations on the reconstructed face we pad the images of the animation:

```
In[68]:= AbsoluteTiming[
  padSize = Round[N@ImageDimensions[lsMotionRorschachs[[1]]][1] / (3)];
  lsPaddedMotionRorschachs = ParallelMap[ImagePad[#, padSize, White] &, lsMotionRorschachs];
]
Out[68]= {0.854722, Null}
```

## Export 2D animations

In this section we export the animations and a collage of the guiding images.

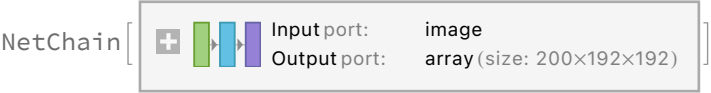
```
In[69]:= AbsoluteTiming[
  Export[FileNameJoin[{NotebookDirectory[], "AnimatedGIFs", "MotionRorschach-" <>
    StringReplace[DateString["ISODateTime"], ":" -> "-"] <> ".mp4"}], lsMotionRorschachs, "MP4", "DisplayDurations" -> 0.05];
]
Out[69]= {4.38745, Null}
```

```
In[70]:= AbsoluteTiming[
  Export[FileNameJoin[{NotebookDirectory[], "AnimatedGIFs",
    "MotionRorschach-guiding-images" <> StringReplace[DateString["ISODateTime"], ":" → "-"] <> ".jpg"}],
    ImageCollage[ColorNegate /@ Values[aPath], "Stretch", Method → "Grid", Background → LightBlue, ImagePadding → 3], "JPEG"];
]
Out[70]= {0.262647, Null}
```

## Facial 3D Reconstruction

In this section we follow the code an explanations in "Facial 3D Reconstruction".

```
In[71]:= net = NetModel["Unguided Volumetric Regression Net for 3D Face Reconstruction"]
Out[71]=
```



```
In[72]:= face = ;

face3D = Image3D[net[face]];

In[215]:= reconstruction = Blur@ImageReflect[ImageResize[face3D, Append[ImageDimensions[face], 105]], Bottom → Front]

coloredSlices = Map[SetAlphaChannel[face, #] &, Image3DSlices[reconstruction, All, 2]];

(*ImageReflect[Image3D[coloredSlices, ViewPoint → {1, -2, 0}, ViewAngle → 20 °, ImageSize → 300], Bottom → Front] *)

Remark: Of course, we can also use, say, photos of Jackie Earle Haley portraying Rorschach in the movie "Watchmen" (2009). For reproducibility purposes we use the photo in the linked article.
```

## Impose Rorschach animation over face

In this section we impose the 2D animation images over the 3D reconstructed face.

### Single image

In this sub-section we show the process of imposing a single animation image.

```
In[77]:= cloth2 = ImageResize[cloth, ImageDimensions[face]];

In[78]:= img2 = ImageResize[lsPaddedMotionRorschachs[[400]], ImageDimensions[face]];
img3 = ImageMultiply[cloth2, img2];
lsRoschachSlices = Map[SetAlphaChannel[img3, #] &, Image3DSlices[reconstruction, All, 2]];

In[81]:= Rasterize@ImageReflect[Image3D[lsRoschachSlices, ViewPoint → {-1.6, -2, 0},
  ViewAngle → 20 °, ImageSize → 300, ColorFunction → GrayLevel, Background → LightGreen], Bottom → Front]
Out[81]=
```



### Animation

In this sub-section we apply the imposing for all animation images.

```
In[82]:= cloth2 = ImageResize[cloth, ImageDimensions[face]];
Out[82]=
```

```

In[83]:= AbsoluteTiming[
  ls3DRorschachs =
    Table[ (
      img2 = ImageResize[img, ImageDimensions[face]];
      img3 = ImageMultiply[cloth2, img2];
      lsRoschachSlices = Map[SetAlphaChannel[img3, #] &, Image3DSlices[reconstruction, All, 2]];
      ImageReflect[Image3D[lsRoschachSlices, ViewPoint → {-1.6, -2, 0}, ViewAngle → 20 Degree,
        ImageSize → 300, ColorFunction → GrayLevel, Background → LightGreen], Bottom → Front]),
      {img, lsMotionRorschachs[[1 ;; -1 ;; 2]]}
    ];
]
Out[83]=
{22.7271, Null}

In[84]:= AbsoluteTiming[
  ls3DRorschachs2 = ParallelMap[ImageRotate[#, {-π / 3, Append[Most@ImageDimensions[#] / 2, 1]}] &, ls3DRorschachs[[1 ;; 12]]];
]
Out[84]=
{10.2781, Null}

In[85]:= lsAngles = Table[- (i - 1) * π / (3.2 * 360), {i, Length[ls3DRorschachs]}];
Length[lsAngles]
Out[86]=
450

In[87]:= AbsoluteTiming[
  ls3DRorschachs4 = MapThread[ImageRotate[#1, #2] &, {ls3DRorschachs, lsAngles}];
]
Out[87]=
{106.224, Null}

```

## Export 3D animations

Here we export the 3D animation:

```

In[88]:= AbsoluteTiming[
  Export[FileNameJoin[
    {NotebookDirectory[], "AnimatedGIFs", "HeadRorschach-" <> StringReplace[DateString["ISODateTime"], ":" → "-"] <> ".mp4"}],
    ls3DRorschachs4, "MP4", "DisplayDurations" → 0.05];
]
Out[88]=
{78.9623, Null}

```

## Multiple Rorschach heads in a row

In this section we get (a selection of) GIF animations and assemble them into one movie. The imported movie are expected to have the same number of frames. The corresponding frames are combined into one frame (in the new movie.)

Get the “custom rotation” batch of images:

```

In[*]:= lsFileNames = Sort@FileNames["HeadRorschach-2022-04-24T12*gif", FileNameJoin[{NotebookDirectory[], "AnimatedGIFs"}]];
Length[lsFileNames]
Out[*]=
3

```

Import the movie files and show the number of frames per file:

```

In[*]:= AbsoluteTiming[
  grs = Import[#] & /@ lsFileNames;
]
Length /@ grs
Out[*]=
{1.36481, Null}

Out[*]=
{450, 450, 450}

```

Assemble corresponding frames into one image:

```

In[*]:= Block[{c = 40},
  lsImgsMulti = MapThread[(d = ImageDimensions[#1];
    ImageAssemble[ImageTake[#, All, {c, -c}] & /@ {#1, ImageResize[#2, d], ImageResize[#3, d]}]) &, grs];
]
Length[lsImgsMulti]
Out[*]=
450

```



Export the assembled frames into a GIF:

```
In[*]:= AbsoluteTiming[
  resFileName = Export[FileNameJoin[{FileNameJoin[{NotebookDirectory[], "AnimatedGIFs"}],
    "Rorschach-heads-row-" <> StringReplace[DateString[Now, "ISODateTime"], ":" → "-"] <> ".gif"]],
    lsImgsMulti, "GIF", "AnimationRepetitions" → Infinity, "DisplayDurations" → 0.025];
]
Out[*]:= {7.89872, Null}
```

Check assembling result:

```
In[*]:= (*ListAnimate[lsImgsMulti]*)
```

---

## Going further

The process above can modified or extended in several ways:

- Different training images
- Different animation guiding images
- Additional image re-styling

### Using random mandalas

We can use random mandalas [AAf3] instead of random Rorschach images [AAf1]. See for example this animation: <https://imgur.com/ygLurTM> .

### Guiding through the original Rorschach images

I experimented with using the original Rorschach images as animation guiding images. (The results were not particularly better or more interesting looking.)

### Using image restyling

Also, we can use ImageRestyle to impose certain more "realistic" look into the derived animations. (Or any other style.)  
Here is an example with imposing the style of the original Rorschach images: <https://imgur.com/UWv4CUG> .

---

## Setup

```
In[*]:= Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/Misc/LSAMonForImageCollections.m"]
```

---

## References

### Articles

[AA1] Anton Antonov, "A monad for Latent Semantic Analysis workflows", (2019), Wolfram Community.  
[AA2] Anton Antonov, "LSA methods comparison over random mandalas deconstruction -- WL", (2022), Wolfram Community.  
[AA3] Anton Antonov, "Bethlehem stars: classifying randomly generated mandalas", (2020), Wolfram Community.  
[AA4] Anton Antonov, "[LIVE] Latent Semantic Analysis Workflows (WL Live-Stream Series)", (2019), Wolfram Community.  
[AA5] Anton Antonov, "Re-exploring the structure of Chinese character images", (2022), Wolfram Community.

### Functions

[AAf1] Anton Antonov, "RandomRorschach", (2022), Wolfram Function Repository.  
[AAf2] Anton Antonov, "RandomScribble", (2021), Wolfram Function Repository.  
[AAf3] Anton Antonov, "RandomMandala", (2019), Wolfram Function Repository.

### Packages

[AAp1] Anton Antonov, LSAMon for Image Collections Mathematica package, (2022), MathematicaForPrediction at GitHub.