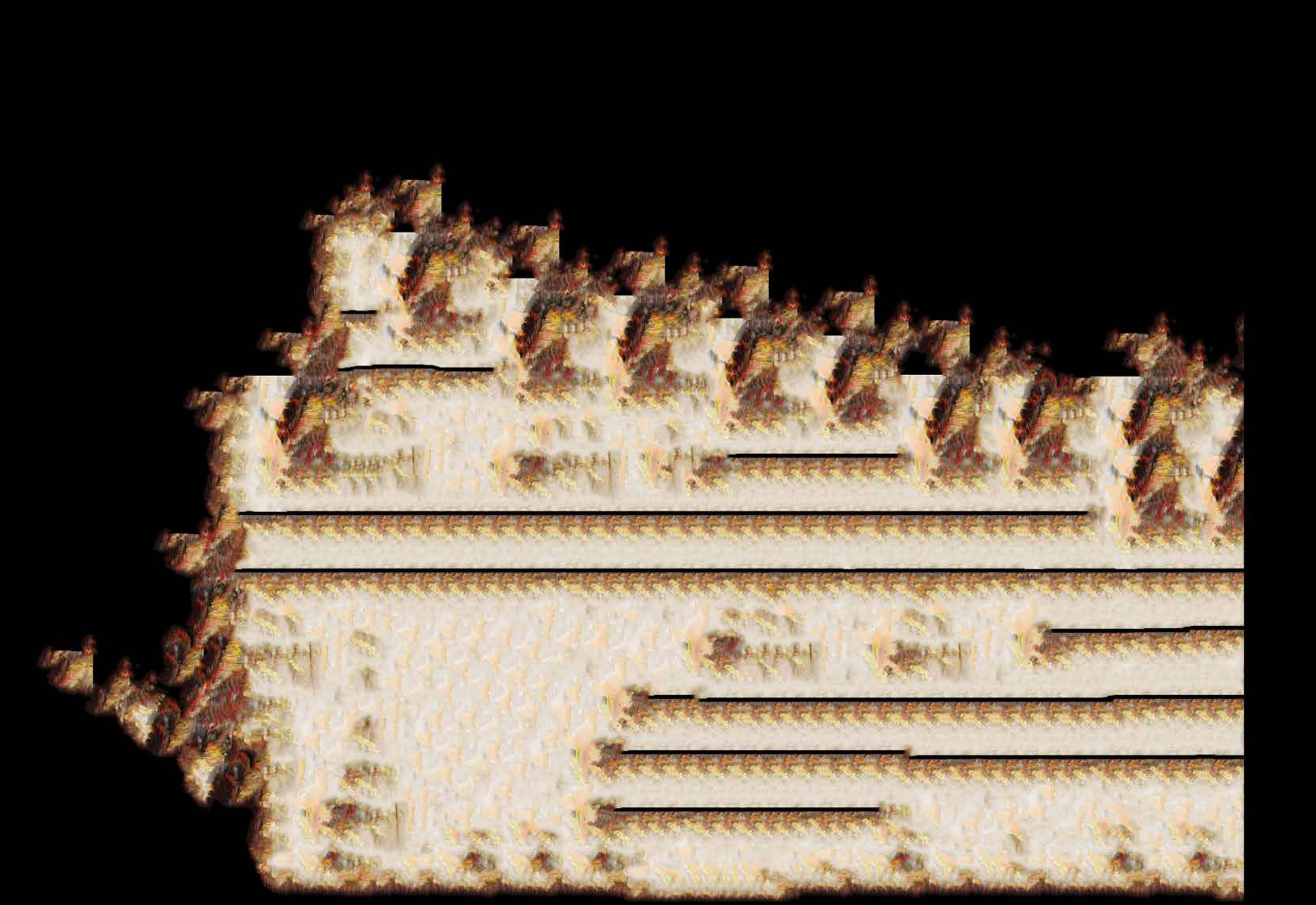
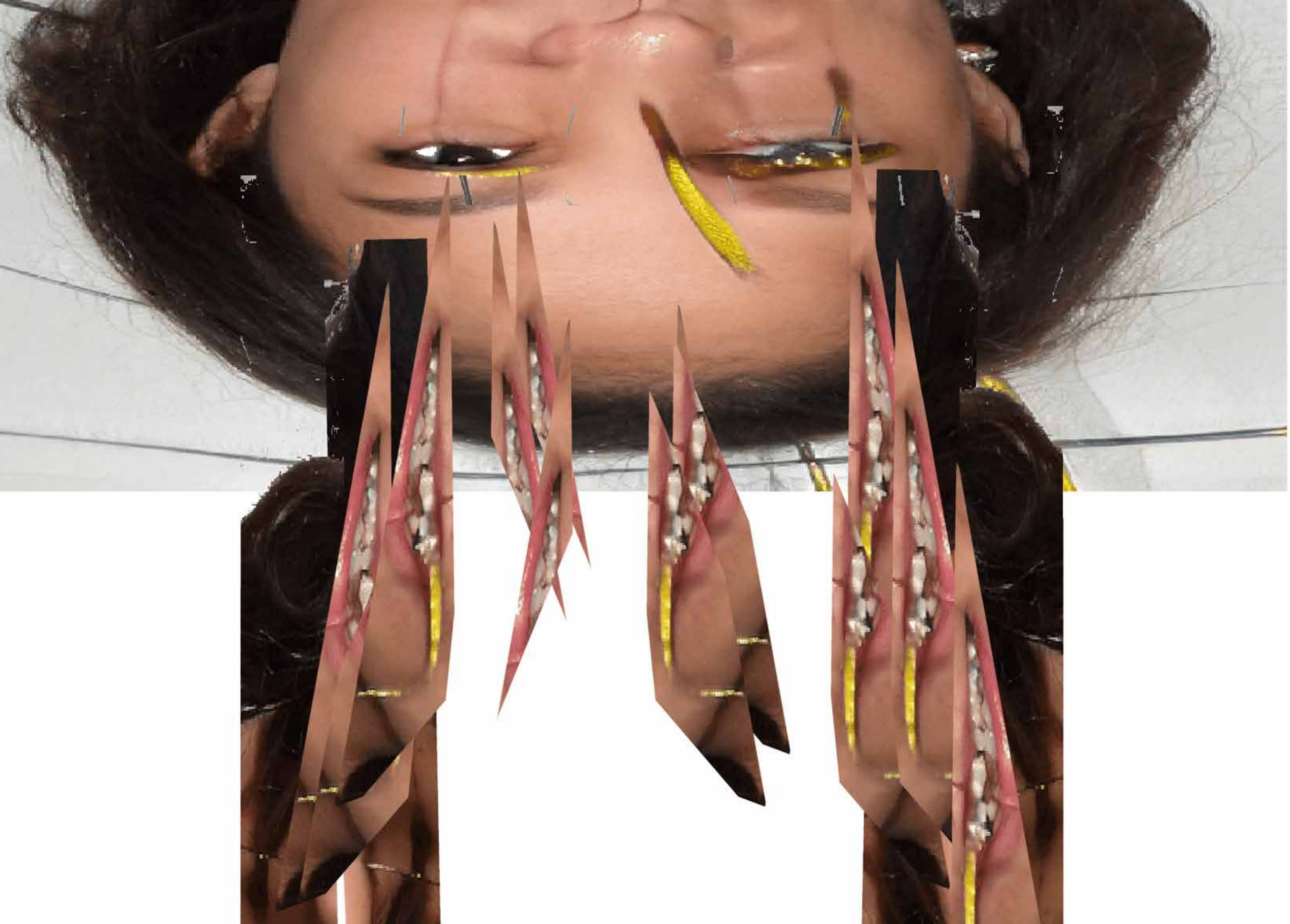
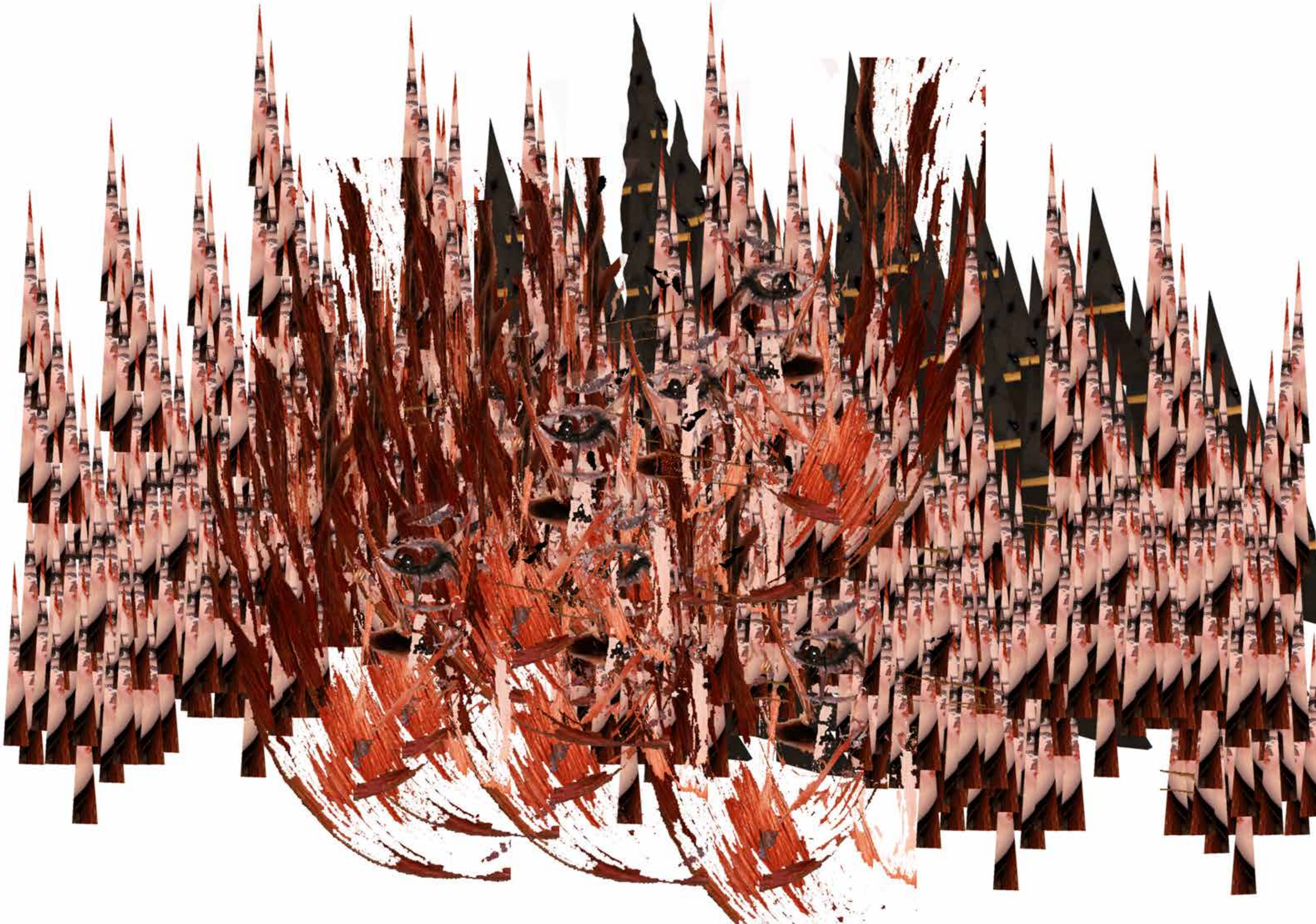
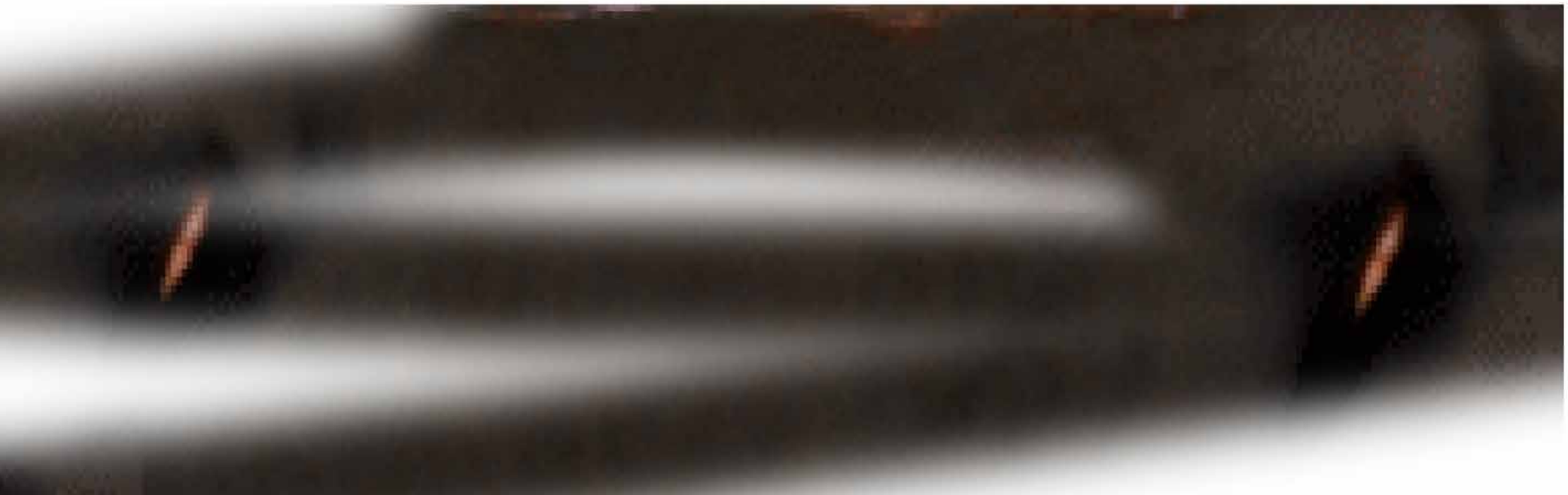


```
def generate_images(network_pkl, seeds, truncation_psi):
    print('Loading networks from "%s"...' % network_pkl)
    _G, _D, Gs = pretrained_networks.load_networks(network_pkl)
    noise_vars = [var for name, var in Gs.components.synthesis.vars.items() if name.startswith('noise')]

    Gs_kwargs = dnnlib.EasyDict()
    Gs_kwargs.output_transform = dict(func=tflib.convert_images_to_uint8, nchw_to_nhwc=True)
    Gs_kwargs.randomize_noise = False
    if truncation_psi is not None:
        Gs_kwargs.truncation_psi = truncation_psi

    for seed_idx, seed in enumerate(seeds):
        print('Generating image for seed %d (%d/%d) ...' % (seed, seed_idx, len(seeds)))
        rnd = np.random.RandomState(seed)
        z = rnd.randn(1, *Gs.input_shape[1:]) # [minibatch, component]
        tflib.set_vars({var: rnd.randn(*var.shape.as_list()) for var in noise_vars}) # [height, width]
        images = Gs.run(z, None, **Gs_kwargs) # [minibatch, height, width, channel]
        PIL.Image.fromarray(images[0], 'RGB').save(dnnlib.make_run_dir_path('seed%04d.png' % seed))

#----------------------------------------------------------------------------

def style_mixing_example(network_pkl, row_seeds, col_seeds, truncation_psi, col_styles, minibatch_size=4):
    print('Loading networks from "%s"...' % network_pkl)
    _G, _D, Gs = pretrained_networks.load_networks(network_pkl)
    w_avg = Gs.get_var('dlatent_avg') # [component]

    Gs_syn_kwargs = dnnlib.EasyDict()
    Gs_syn_kwargs.output_transform = dict(func=tflib.convert_images_to_uint8, nchw_to_nhwc=True)
    Gs_syn_kwargs.randomize_noise = False
    Gs_syn_kwargs.minibatch_size = minibatch_size

    print('Generating W vectors...')
    all_seeds = list(set(row_seeds + col_seeds))
    all_z = np.stack([np.random.RandomState(seed).randn(*Gs.input_shape[1:]) for seed in all_seeds]) # [minibatch, component]
    all_w = Gs.components.mapping.run(all_z, None) # [minibatch, layer, component]
    all_w = w_avg + (all_w - w_avg) * truncation_psi # [minibatch, layer, component]
    w_dict = {seed: w for seed, w in zip(all_seeds, list(all_w))} # [layer, component]

    print('Generating images...')
    all_images = Gs.components.synthesis.run(all_w, **Gs_syn_kwargs) # [minibatch, height, width, channel]
    image_dict = {(seed, seed): image for seed, image in zip(all_seeds, list(all_images))}

    print('Generating style-mixed images...')
    for row_seed in row_seeds:
        for col_seed in col_seeds:
            w = w_dict[row_seed].copy()
            w[col_styles] = w_dict[col_seed][col_styles]
            image = Gs.components.synthesis.run(w[np.newaxis], **Gs_syn_kwargs)[0]
            image_dict[(row_seed, col_seed)]
```







```
name = 'stylegan2_pytorch',
  packages = find_packages(),
  scripts=['bin/stylegan2_pytorch'],
  version = '0.7.2',
  license='GPLv3+',
  description = 'StyleGan2 in Pytorch',
  author = 'Phil Wang',
  author_email = 'lucidrains@gmail.com',
  url = 'https://github.com/lucidrains/stylegan2-p
  download_url = 'https://github.com/lucidrair
  keywords = ['generative adversarial networks', 'artificial inte
  install_requires=[
      'fire',
      'numpy',
      'retry',
      'tqdm',
      'torch',
      'torchvision',
      'pillow',
      'torch_optimizer',
      'contrastive_learner>=0.1.0'
  ],
  classifiers=[
      'Development Status :: 4 - Beta',
      'Intended Audience :: Developers',
      'Topic :: Scientific/Engineering :: Artificial Intelligence',
      'License :: OSI Approved :: MIT License',
      'Programming Language :: Python :: 3.6',
  ],
```