In [4]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Nov 17 17:41:47 2019

@author: jorgeagr
"""
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.losses import MSE
from itertools import combinations_with_replacement, product

height = 10
width = 10

mpl.rcParams['figure.figsize'] = (width, height)
mpl.rcParams['font.size'] = 20
mpl.rcParams['figure.titlesize'] = 'small'
mpl.rcParams['legend.fontsize'] = 'small'
mpl.rcParams['xtick.major.size'] = 12
mpl.rcParams['xtick.minor.size'] = 8
mpl.rcParams['xtick.labelsize'] = 18
mpl.rcParams['ytick.major.size'] = 12
mpl.rcParams['ytick.minor.size'] = 8
mpl.rcParams['ytick.labelsize'] = 18

def unpickle(file):
    import pickle
    with open(file, 'rb') as fo:
        cifar = pickle.load(fo, encoding='bytes')
        data = np.asarray(cifar[b'data'][:50], dtype=np.int)
    return data

def build_image(img_bits):
    r = img_bits[:1024].reshape(32,32)
    g = img_bits[1024:1024*2].reshape(32,32)
    b = img_bits[1024*2:].reshape(32,32)
    img = np.stack((r,g,b),axis=2)
    return img

epochs = 500
batch_size = 5

data = unpickle('../data/data_batch_1')
# Normalize
data = data / 255

# Create model and add layers
autoencoder = Sequential()
autoencoder.add(Dense(50, activation='sigmoid'))
autoencoder.add(Dense(len(data[0]), activation='sigmoid'))

# Glue it all together
autoencoder.compile(loss='mean_squared_error', optimizer=Adam())

# Train the model
train_hist = autoencoder.fit(data, data, epochs=epochs, batch_size=batch_size, verbose=0)

# Evaluate to get MSE across the 50 images
mse = autoencoder.evaluate(data, data)

np.random.seed(0)
indices = np.random.randint(0, len(data), 3)
```

```
50/50 [==============================] - 0s 1ms/step
```

In [5]:
```
fig = plt.figure()
ax = [plt.subplot2grid((3,3), loc, colspan=1, rowspan = 1, fig=fig) for loc in product([0,1,
2],[0,1,2])]
for a in ax:
    a.axis('off')
for i, ind in enumerate(indices):
    img_dat = data[ind]
    noisy_dat = img_dat + np.random.normal(0, 25, len(img_dat))/255
    noisy_mse = ((noisy_dat - img_dat)**2).mean()
    recons_dat= autoencoder.predict(noisy_dat.reshape(1,3072)).flatten()
    recons_mse = ((recons_dat - img_dat)**2).mean()
    ax[i].imshow(build_image(img_dat))
    ax[1].set_title('Original')
    ax[i+3].imshow(build_image(noisy_dat))
    ax[4].set_title('Noisy')
    ax[i+6].imshow(build_image(recons_dat))
    ax[7].set_title('Denoised')
    print('Image ID:', ind)
    print('Noisy-Original MSE:', noisy_mse)
    print('Denoised-Original MSE:', recons_mse, end='\n\n')
fig.tight_layout(pad=0.5)
#fig.savefig('../prob4.eps', dpi=100)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
255] for integers).

Image ID: 44
Noisy-Original MSE: 0.009054117128761339
Denoised-Original MSE: 0.0017303872390900263

Image ID: 47
Noisy-Original MSE: 0.009581174864566583
Denoised-Original MSE: 0.0018948382194266037

Image ID: 0
Noisy-Original MSE: 0.009481449691803118
Denoised-Original MSE: 0.003551061700551042
```