In [49]:
```python
from analysis.analysis import *
import cv2
import matplotlib
```

In [21]:
```python
# comparison values (from analysis.py)
print(f"Datasets: \t\t{datasets}")
print(f"Metrics: \t\t{list(metrics_dict.keys())}")
print(f"Methods: \t\t{methods}")
print(f"Lambdas (for DP): \t{list(lambdas)}")
print(f"Window Sizes: \t\t{w_sizes}")
print(f"Padding(cut gt image):\t{padding}")
```

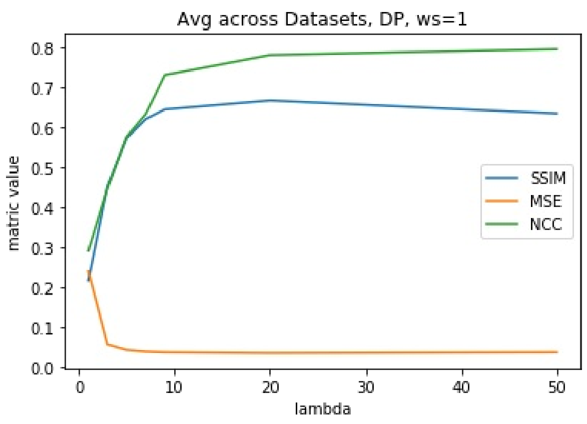
```
Datasets:               ['Art', 'Reindeer', 'Laundry', 'Dolls', 'Moebius', 'Books']
Metrics:                ['SSIM', 'MSE', 'NCC']
Methods:                ['DP', 'naive']
Lambdas (for DP):       [1, 3, 5, 7, 9, 20, 50]
Window Sizes:           [1, 3, 5, 7, 9]
Padding(cut gt image):  30
```

In [22]:
```python
avg_metrics_lambda = get_avg_metrics(get_metrics_lambda)
```

In the next cell we can see that, generally, 9.0 could be an optimal value for lambda as metrics stop improving.
If lambda is too big, the metrics would be worser.
Although, "optimal" value of lambda (~20) would vanish important small details, so for further comparison I would use almost optimal still adequate lambda = 9.0

In [23]:
```python
ax = avg_metrics_lambda.plot()
ax.set_xlabel("lambda")
ax.set_ylabel("matric value")
ax.set_title("Avg across Datasets, DP, ws=1")
```

Out[23]:
```
Text(0.5, 1.0, 'Avg across Datasets, DP, ws=1')
```

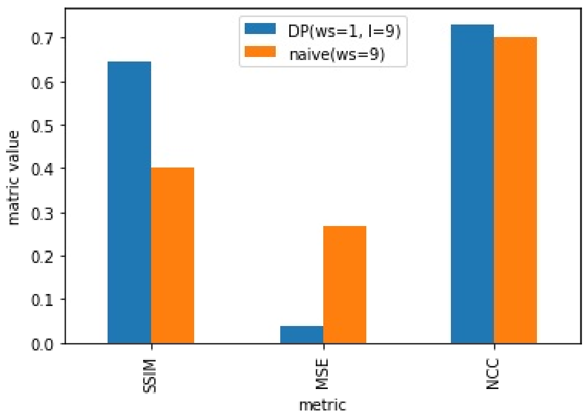


In [24]:
```python
avg_metrics_method = get_avg_metrics(get_metrics_method)
```

In the next cell we can see, that in general Metrics for minimum window size for Dynamic Programming approach are much better then for window size 9 naive approach
Accound that window size 9 would compute on my 16-cores laptop for ~10 minutes.
In the same time, DP would compute for ~9 seconds only which is ~90 times faster for better result

In [25]:
```python
ax = avg_metrics_method.T.plot.bar()
ax.set_xlabel("metric")
ax.set_ylabel("matric value")
```

Out[25]:
```
Text(0, 0.5, 'matric value')
```



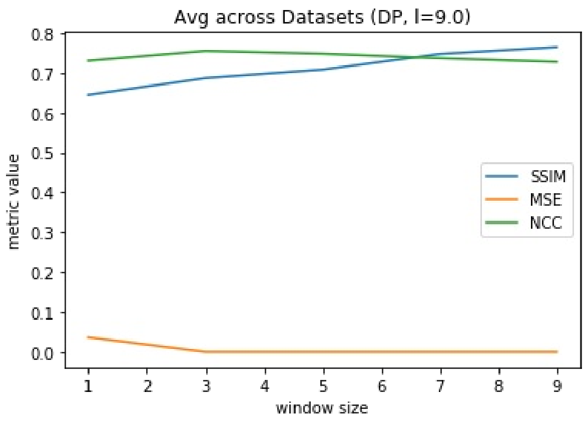
In [26]:
```python
metrics_w_size_naive_avg = get_avg_metrics(get_metrics_w_size_naive)
```

In [27]:
```
metrics_w_size_DP_avg = get_avg_metrics(get_metrics_w_size_DP)
```

In the next cell we can see that depending on the window size, performance of DP approach increases slightly, but it doesn't worth it for sure

In [28]:
```
ax = metrics_w_size_DP_avg.plot()
ax.set_xlabel("window size")
ax.set_ylabel("metric value")
ax.set_title("Avg across Datasets (DP, l=9.0)")
```

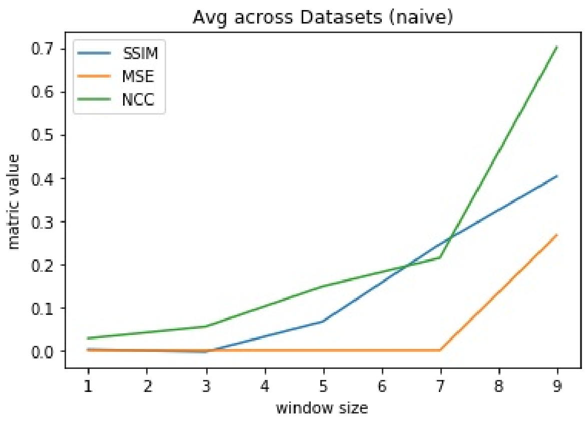
Out[28]:
```
Text(0.5, 1.0, 'Avg across Datasets (DP, l=9.0)')
```



In the next cell we can see that for naive approach, window size directly improves the performance

In [29]:
```
ax = metrics_w_size_naive_avg.plot()
ax.set_xlabel("window size")
ax.set_ylabel("matric value")
ax.set_title("Avg across Datasets (naive)")
```

Out[29]:
```
Text(0.5, 1.0, 'Avg across Datasets (naive)')
```

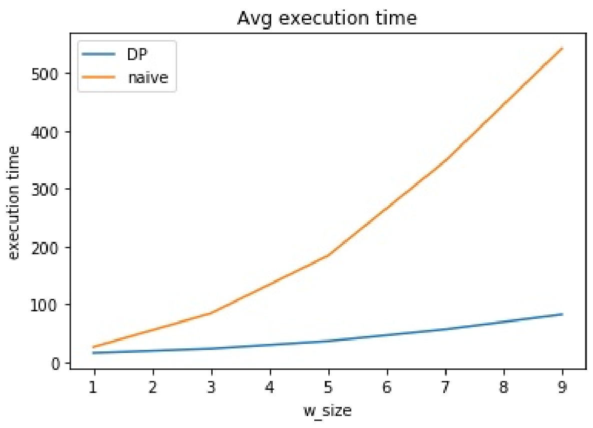


In [30]:
```
exec_times_ws = get_avg_metrics(get_time_method_ws)
```

In the next cell we can see that for both approaches, increasing the window size would increase the execution time more and more.
Although, dynamic programming approach is still much faster for every given window size

In [31]:
```
ax = exec_times_ws.plot()
ax.set_xlabel("w_size")
ax.set_ylabel("execution time")
ax.set_title("Avg execution time")
```

Out[31]:
```
Text(0.5, 1.0, 'Avg execution time')
```

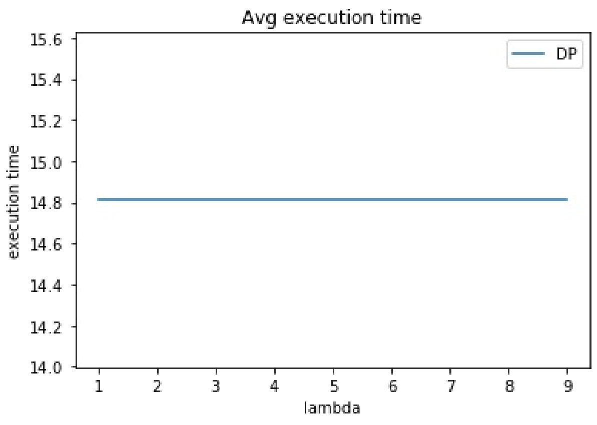


```
In [32]: exec_times_l = get_avg_metrics(get_time_DP_lambda)
```

In the following cell, we can see that execution time of the DP approach is not affected by lambda at all

```
In [33]: ax = exec_times_l.plot()
         ax.set_xlabel("lambda")
         ax.set_ylabel("execution time")
         ax.set_title("Avg execution time")
```

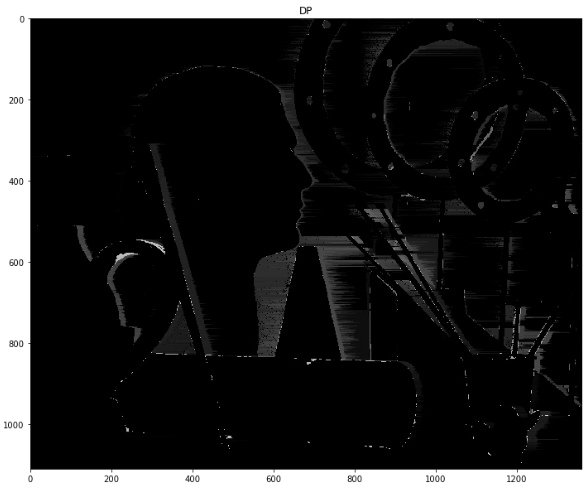
```
Out[33]: Text(0.5, 1.0, 'Avg execution time')
```

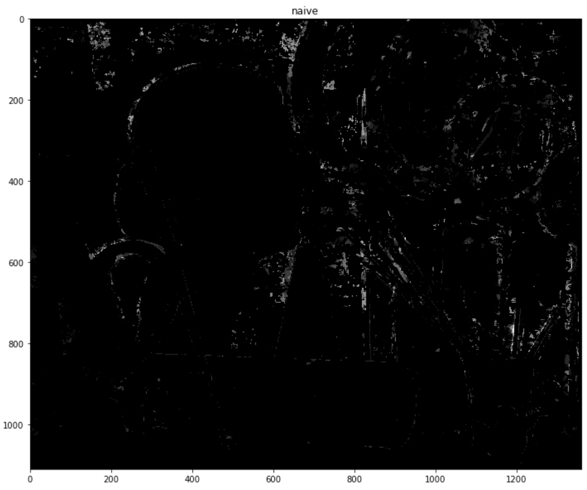


In the following cell you can see difference images for each dataset for DP (l=9.0, ws = 1) and naive (ws=9) approaches.
We can see that for naive approach the difference is sometimes very big.
Although, for DP approach the differences occur on the lines where it is more benefitial for optimization to keep the same disparity, which could be a problem.

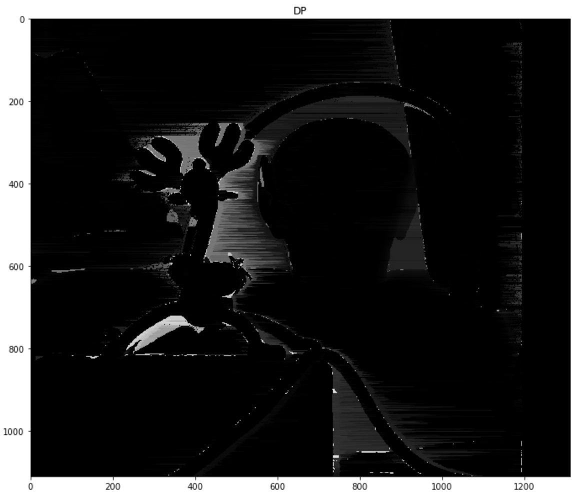
```
In [34]: # diff images
         for Dataset in datasets:
             display_image_diff(Dataset)
```

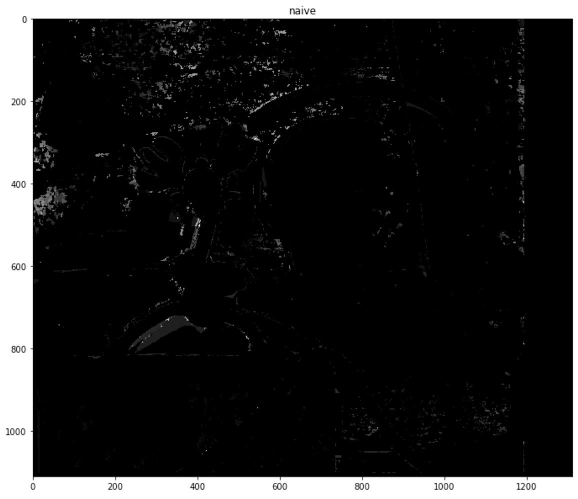
Dataset Art diff



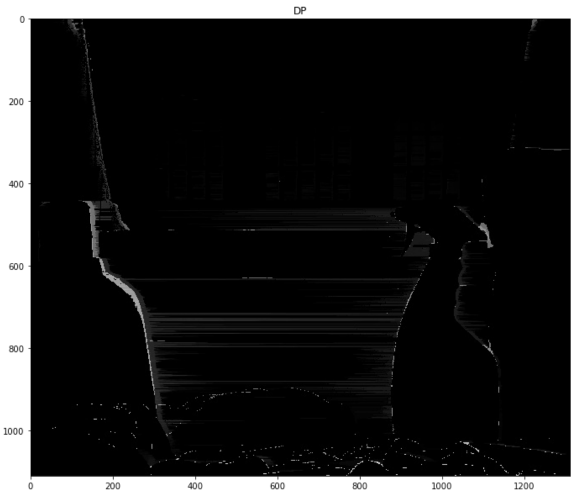


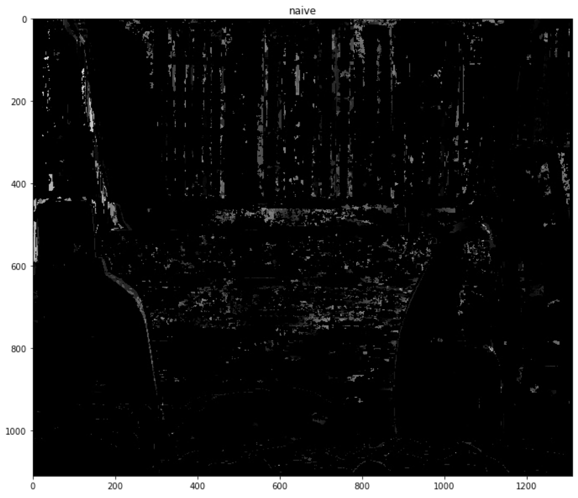
Dataset Reindeer diff



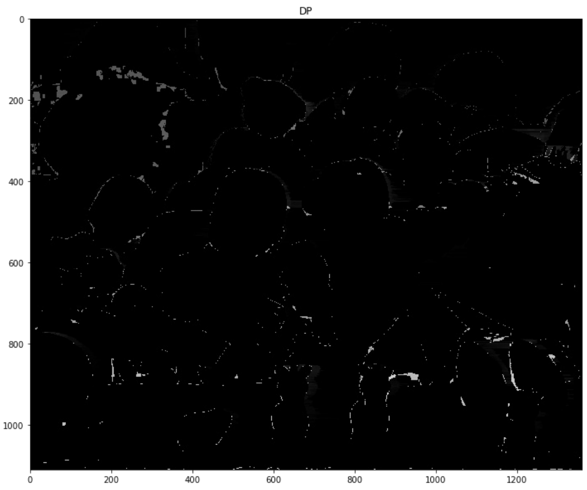


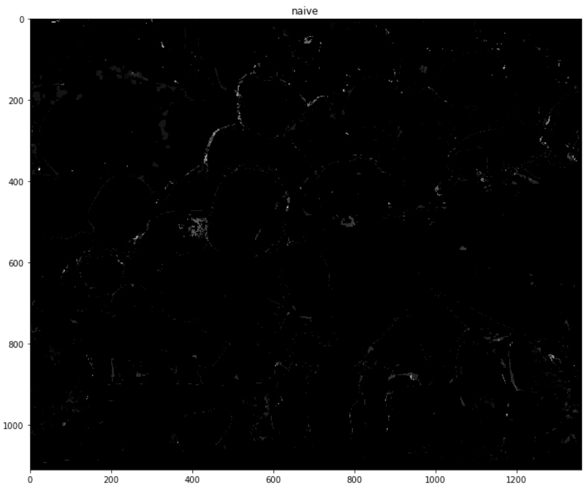
Dataset Laundry diff

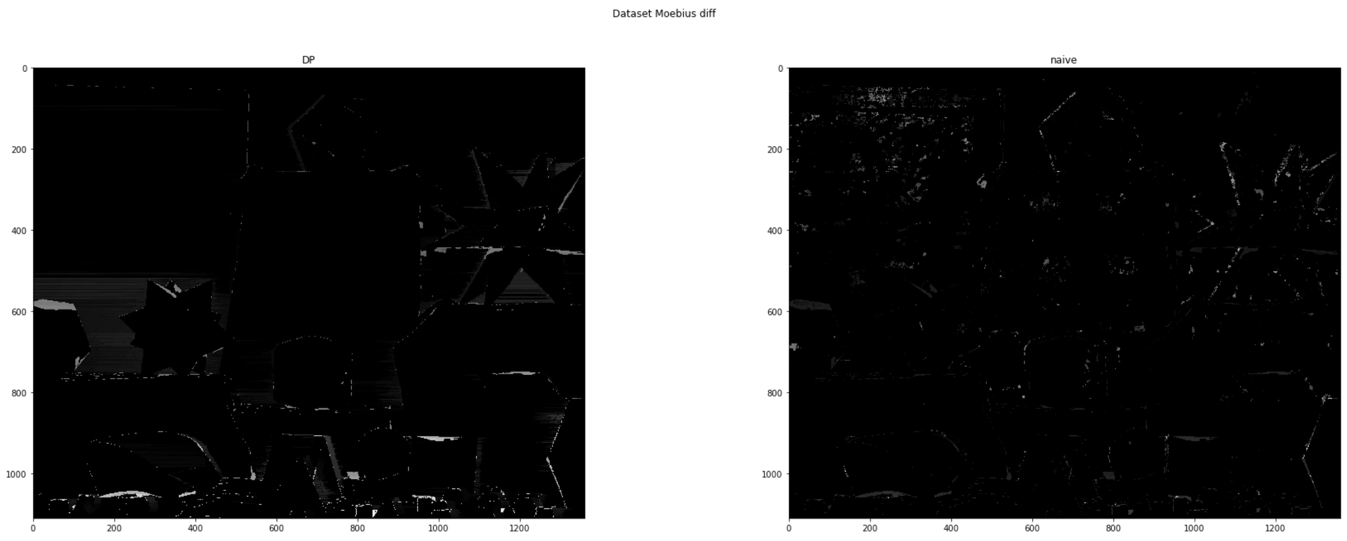


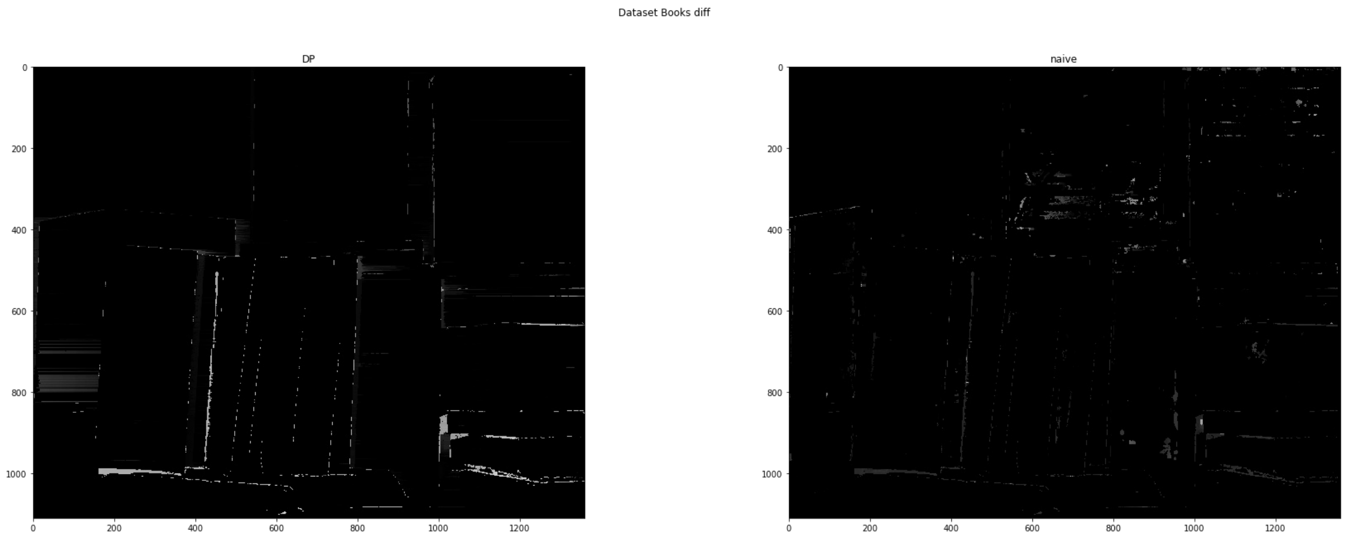


Dataset Dolls diff









As a conclusion:

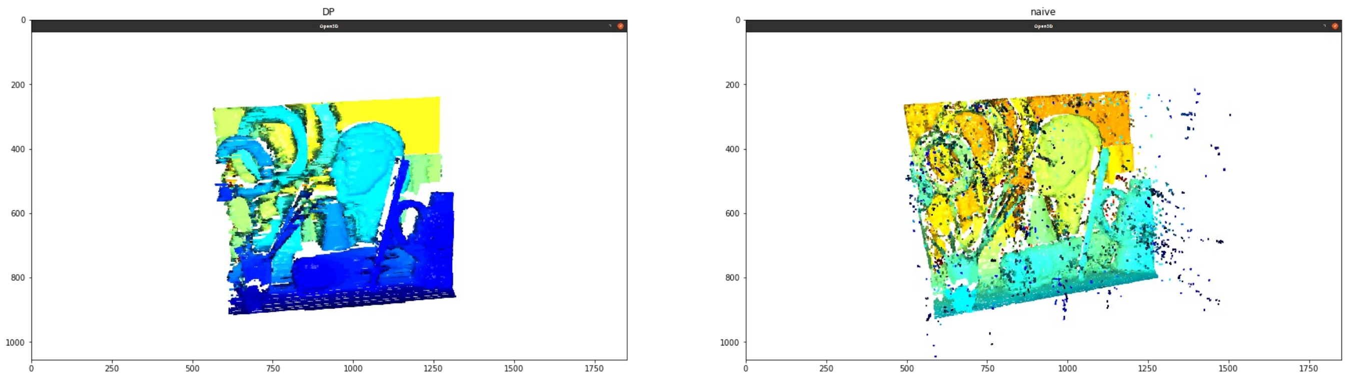
- DP runs much faster and outputs much better results comparing to naive approach;
- Picking up optimal lambda, we should remember that metrics do not account on a lost of small details;
- For naive approach, ws is a crutial parameter, which improves the performance and increases the processing time dramatically;
- DP approach almost does not depend on this value, but outputs better and faster result with ws=1
- DP approach suffers from vanishing small details as they require adding lambda two times for a small disparity match, although improving the overall metric

Account that the results are averaged across Datasets, all the testing functions can be found in analysis/analysis.py
It could seem like there is not a lot of code in the report, but it uses crazy amount of additional custom functions, caching the results.
This allows produce the plots without rerunning all the output.
(I left it running for a night)

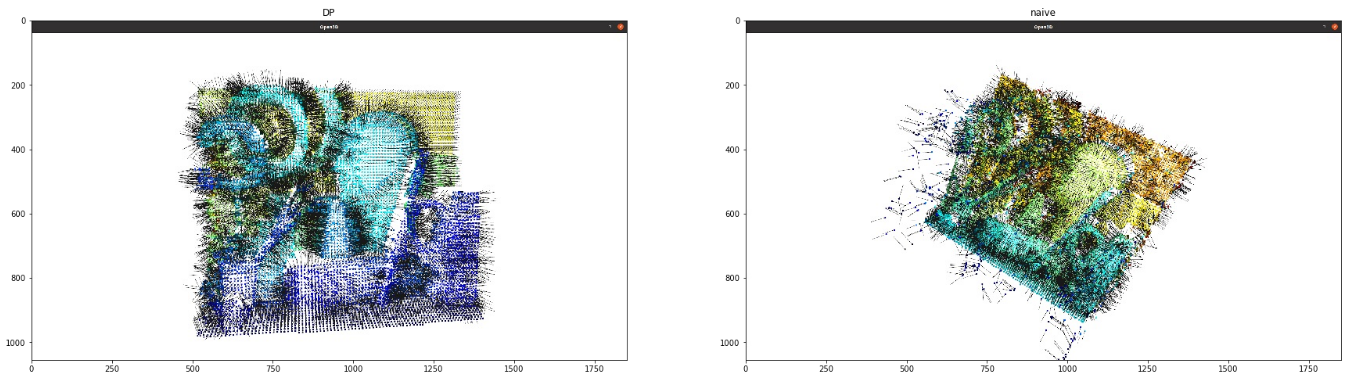
And some pointclouds, normals, mashes for optimal parameters for each method:

In [64]:
```python
vis_types = ["points", "normals", "mesh"]
cmap_reversed = matplotlib.cm.get_cmap('autumn')
for vis_type in vis_types:
    f, ax = plt.subplots(1, len(methods))
    f.set_figheight(10)
    f.set_figwidth(30)
    for i, method in enumerate(methods):
        img_path = os.path.join("output", "3d", f"{vis_type}_{method}.png")
        img = cv2.imread(img_path, cv2.IMREAD_ANYCOLOR)
        img[:, :, :] = img[:, :, ::-1] # fixing a wierd bug
        ax[i].imshow(img)
        ax[i].set_title(method)
    plt.suptitle(f"{vis_type}")
    plt.show()
```

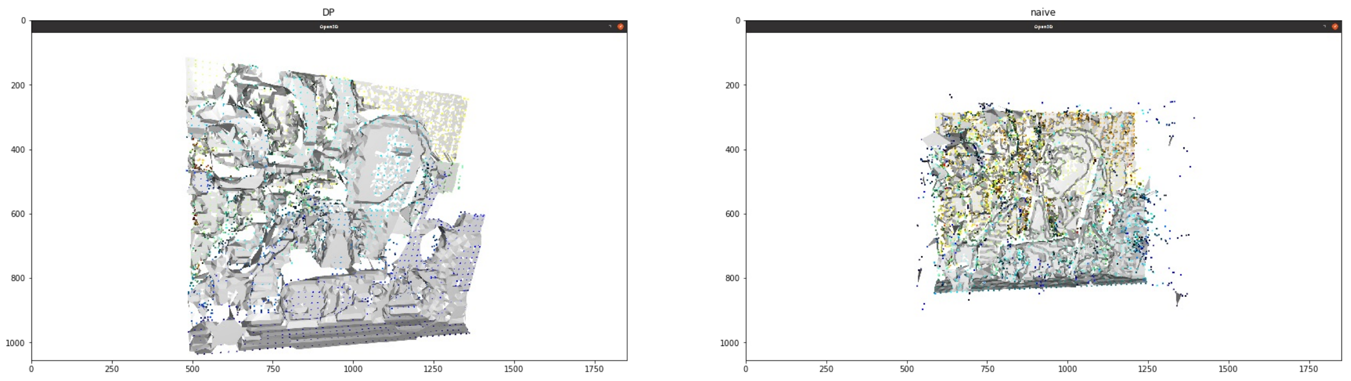
points



normals



mesh



Even though, naive approach is much more noisy, it seems that if we have enough computational resources, time and noise filtering, maybe sometimes it could be better to keep all the details of the picture, like small brush's hold.
But overall, dp is much cleaner "out of the box" without any additional filtering.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js