# Live Stitching

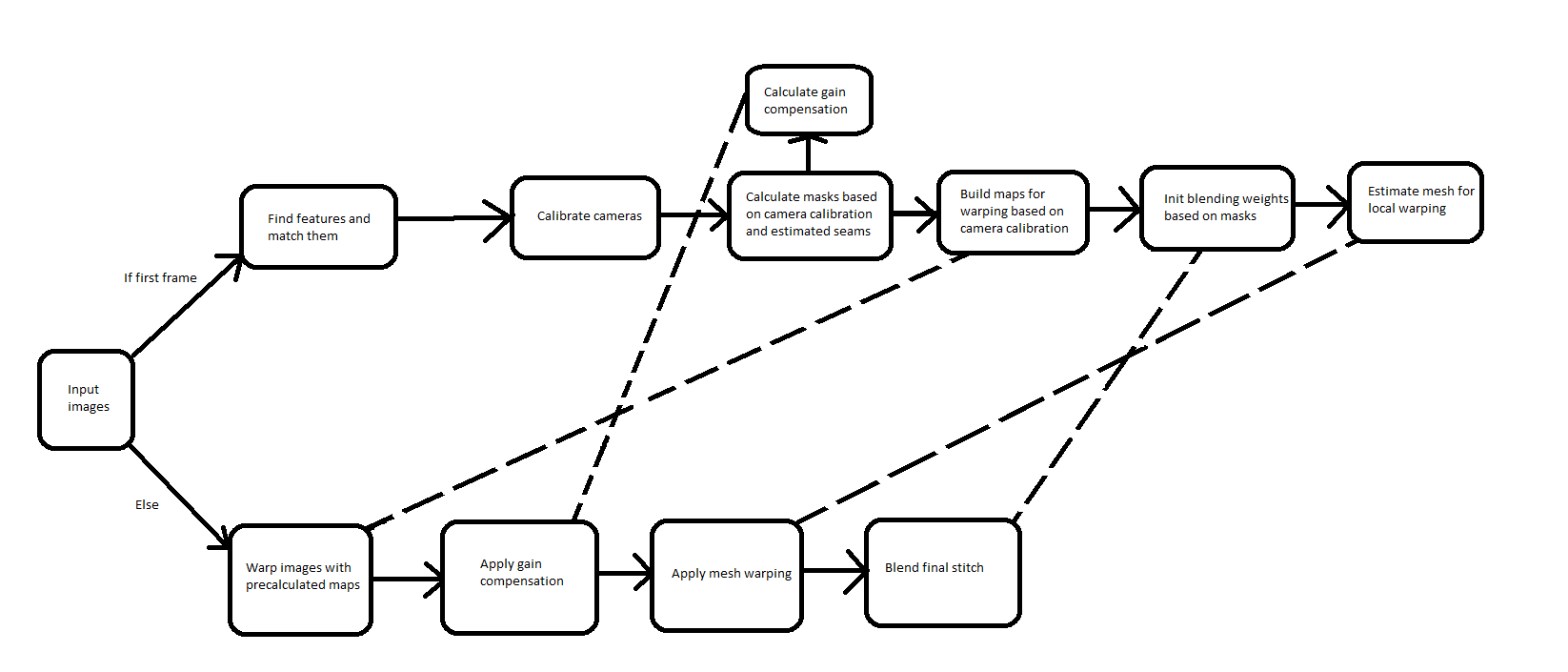


Figure 1: Stitching pipeline

## Input images

These come from either video files or TCP connections with several capture boards

## Find features

Find features using SURF or ORB and match them with an appropriate matching algorithm.

## Calibrate cameras

Assume the cameras are in same position with only rotation of between each camera. Assume the principal point is in middle of the picture and estimate focal length by minimizing the difference in position of matched features after warping.

## Calculate masks

Masks are calculated by first warping mask based on the camera parameters. After that a seam is estimated between each image to get the final mask.

## Gain compensation

Estimate required compensation in gain of the images so the exposure differences between cameras don’t create artefacts.

## Build warping maps

Build the warping maps based on the camera parameters.

## Initialize blending

Use the mask and estimated position of the images in final stitch to calculate blending weights so the weights can be applied fast at runtime.

## Estimate mesh for local warping

First remap the images based on the previously built warping maps. After that use methods described in Content Preserving Warping to estimate mesh which should minimize the parallax and ghosting artefacts in final stitch. Since the remap function in opencv uses backward maps and CPW uses forward maps, build the backward maps based on the result mesh from CPW.

## Warping with maps

Warp the input images based on the precalculated maps similarly to how it was done in mesh estimation.

## Apply gain compensation

Multiply the images with the calculated gain compensation values.

## Apply mesh warping

First remap the input images based on the CPW mesh. Second recalculate the blending weights by warping the masks with the CPW mesh and then calculating blending weights.

## Blend final stitch

Build laplacian pyramids of the images and add them to the final stitch weighted with the precalculated weight maps.

## Random notes and TODOs

* Do uploading images to GPU and stitching in different threads.
* Implement command line arguments
* Dynamically update the CPW mesh.
* Dynamically update the gain compensation
* Experiment with the parameters in CPW
* Perhaps add temporal warping to the mesh specified in <https://pdfs.semanticscholar.org/2e40/88cea7f133fea5804a60e9d80691e0c03bfa.pdf>.
* Unity project for creating synthetic data can be found in S:\81401\_Ultravideo\360Video\Software\cam360
* Test videos are in S:\81401\_Ultravideo\Work\360\_stitcher\videos
* Scale the final stitch size so the pixels per degree is same as in the target headset. E.g. HTC Vive has 1080 pixels for 100 degrees so the final stitch should be 3888 pixels wide.
* Recreation of GpuMat objects every frame is expensive. This was second main reason for why opencv code was slow. The main reason was not separating code which have to be run only once at the start.