Virtual Projected Windows

https://github.com/bpfel/windows

Cédric Uschatz
ETH Zürich
uschatzc@ethz.ch

Maria Vila Abad
ETH Zürich
vilmaria@ethz.ch

Linus Wigger
ETH Zürich
wiggerl@ethz.ch

GianAndrea Müller
ETH Zürich
muellegi@ethz.ch

Abstract

In the past few years video cameras, computer vision algorithms and the hardware needed to run them have been made readily available to the public. With this project we present the integration of three webcams, a beamer and the relevant software to produce the illusion of a virtual window for a single user. We endeavour to present the viewer with an immersive experience that catches the eye and invites to linger.
We have established a working beamer calibration, a calibrated stereo camera system enabling close to real time face detection and triangulation, a virtual three dimensional world with fast view rendering, a rack for mounting the hardware components and a single executable incorporating the whole online pipeline.

1. Problem Statement

The goal of this project is to project a viewer-position dependent image of a 3D scene onto a surface, creating the illusion of a virtual window, using a mobile projector and three cameras. The following work packages have been identified and assigned :

- Building a mounting system allowing to rigidly fix cameras and projector but giving the freedom to adjust the stereo baseline and camera angles. [GianAndrea]
- Calibrating the projector, triangulating the distance to the wall and computing the dimension of the virtual window. [Cédric]
- Establishing a working stereo camera, which includes mono and stereo camera calibration. [GianAndrea]
- Implementing face detection. [Maria]
- Triangulation to compute face position. [Maria]
- Creating a three dimensional world and handling the view projection. [Linus]

2. Related Work

2.1. Projector-Camera Calibration

For the calibration of the projector the implementation of Daniel Moreno and Gabriel Taubin is used, which they explain in their paper on Simple, Accurate, and Robust Projector-Camera Calibration [6]. They have to adapt the calibration procedure, in contrast to conventional stereo camera calibration systems, to the fact that projectors cannot directly measure the pixel coordinates of 3D points projected onto the projector image plane. To circumvent this issue they use high frequency structured light patterns on a checkerboard in order to identify which projector pixel causes the intensity value in the camera image plane. The decoded points in the neighbourhood of each corner in the camera image are used to find a local homography for every corner location. Once those corners are warped into the projector image plane they can use standard camera calibration functions in OpenCV to calibrate camera and projector assuming a pinhole model for camera and projector. The output of the software returns the intrinsics of the camera and the projector, the rotation and translation matrix of the projector with respect to the camera as well as radial and tangential distortion coefficients for camera and projector.

2.2. Face detection

Face detection has been extensively improved over the past years. For this reason, there exist many libraries that include face detection algorithms. Some of the most common ones are Matlab, OpenCV or dlib. Matlab has not been considered in this application as it does not run in C and thus it is harder to integrate in the pipeline. Moreover, due to the nonexistence of a dedicated GPU, only CPU based algorithms have been considered.
The OpenCV face detection algorithm that works almost in real time on CPU is the Haar Cascade Face Detector. It is based on Haar features proposed by Viola and Jones in [8]. Moreover, it does also include the improvements made by Lienhart et al. in [4]. This method is capable of detecting faces at different scales. However, its major drawbacks are

that it gives many false predictions, it does not work under occlusion and it does not work on non-frontal images.
Dlib also offers a face detection algorithm that works almost in real time on CPU. This method uses Histogram of oriented gradients (HOG) features and a Support-vector machine (SVM). It works for frontal and slightly non-frontal faces as well as faces under small occlusions. However, it does not detect small faces as it has been trained for minimum face size of 80 x 80 px.
It can be assumed that the user will not be occluded and that they will look at the window, resulting in slightly non-frontal images on the cameras. Therefore the dlib approach is better for this project as it is more accurate and works for slightly non-frontal faces.

3. Pipeline

3.1. Initialization

Projector-Camera Calibration As mentioned, the projector calibration software by Daniel Moreno and Gabriel Taubin [6] is used. Unfortunately even though the source code has been made available it does not compile due to errors in the graphical interface library Qt. Instead their GUI [5] is used, which makes it impossible to integrate it in the rest of the work. As a consequence the projector calibration needs to be done offline before rendering the 3D scene. However once calibrated and fixing the focus of the projector the intrinsic and extrinsic parameters are not changed anymore. A Logitech C920 webcam (1920 x 1080 px.) and a BenQ GP10 projector (1280 x 800 px.) are used. An example of a possible calibration result is given below. The translation vector is given in mm.

$$K_{\text{cam}} = \begin{bmatrix} 1428.6 & 0 & 933.0 \\ 0 & 1428.6 & 544.8 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$K_{\text{proj}} = \begin{bmatrix} 735.9 & 0 & 320.1 \\ 0 & 1474.8 & 800.7 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$R = \begin{bmatrix} 0.978 & -0.013 & -0.204 \\ 0.005 & 0.999 & -0.043 \\ 0.205 & 0.041 & 0.977 \end{bmatrix} \tag{3}$$

$$T = \begin{bmatrix} 145.2 & -158.3 & 25.5 \end{bmatrix} \tag{4}$$

$$e_{\text{cam}} = 0.33, e_{\text{proj}} = 0.31, e_{\text{stereo}} = 0.34 \tag{5}$$

Some important remarks on the calibration results:

- The principal point can be read out from the intrinsic matrix K_{proj}: The x component of the principal point of the projector is approximately at a quarter of the width resolution and not as expected at half the width resolution of the projector. Looking into the source code, they use the same number of vertical and horizontal light patterns and are therefore unable to resolve the higher resolution side. Since the projectors resolution is 1280 x 800 px their structured light scan can only resolve up to 800 px, so they halfed the resolution width of the projector to 640 px which explains the position of the principal point. To get the right resolution again, the first row of K_{proj} is multiplied by 2.

- For the camera the principal point lies in the middle of the image plane, whereas for the projector, after adapting the first row, the principal point lies in the middle with respect to pixel columns but at the bottom with respect to pixel rows. This makes sense, since normally projectors project their images only upwards from their center of projection! Comparing these values with the resolution of the camera (1920 x 1080 px.) and the projector(1280 x 800 px.) the results are reasonable.

- The rotation matrix R and translation vector T of the projector are with respect to the camera. The world coordinate system is set to the center of projection of the camera.

- The radial and tangential distortion are not listed, since they are reasonably small and are neglected for future calculations.

- The reprojection errors e_{cam}, e_{proj} and e_{stereo} are below a pixel and in the range of what the software promises.

In order to compute the distance to the wall four images with the resolution of the projector are projected onto the wall and captured by the camera, where each projected image has one white pixel in it and all other pixels are black. A fifth completely black image is projected and captured as well, then used to subtract the background from the other four images. In the resulting four binary images the highest intensity pixel is taken as correspondence point in the camera image plane. Using the OpenCV triangulate function with the corresponding point pairs and calibration matrices the distance to the wall with respect to the camera is computed. In order to get the distance with respect to the projector frame a coordinate transformation is done: $Z' = R \cdot Z + T$, where R and T are the rotation matrix and translation vector respectively and Z and Z' is the distance in the camera and projector coordinate system respectively. In order to compute the dimension of the virtual window perspective projection is used. The width of the window is computed by the formula $\Delta x = 2 \cdot Z' \cdot \frac{x_0}{f k_x}$ and the height by $\Delta y = Z' \cdot \frac{y_0}{f k_y}$ where x_0 and y_0 is the principal point of the projector, f is the focal length and k_x and k_y are the number of pixels per unit length. All the components can be directly read from the intrinsic matrix of the projector.

Symbol	Description
C_i	camera with index i
$\mathcal{A}_1$	frame of camera 1
$\mathcal{A}_2$	frame of camera 2
$\mathcal{B}$	beamer frame
$\mathcal{C}$	wall frame
$\mathcal{D}$	frame of camera 3
$\vec{r}_v$	position of the viewer
$\vec{r}_b$	position of the center of projection of the beamer
d_c	distance between C_1 and the origin of $\mathcal{B}$
d_w	distance between the wall and $\vec{r}_b$
d_b	distance between $\vec{r}_b$ and the origin of $\mathcal{B}$
$\vec{T}_s$	translation vector from C_1 and C_2
Results from stereo calibration	
$\boldsymbol{R}$	rotation matrix: $\mathbf{R}_{\mathcal{A}_1}\vec{r} = {}_{\mathcal{A}_2}\vec{r}$
${}_{\mathcal{A}_2}\vec{T}$	translation vector from $\mathcal{A}_2$ to $\mathcal{A}_1$
Notation	
${}_{\mathcal{X}}\vec{r}$	vector represented in frame $\mathcal{X}$
${}_{\mathcal{A}_1}\vec{e}_i^{\mathcal{B}}$	i^{th} coordinate direction of frame $\mathcal{B}$ represented in frame $\mathcal{A}_1$

Table 1. Notation

Stereo Camera Calibration Two cameras face the user and together form a stereo camera system which is used to detect a face and triangulate its position relative to the beamer fixed frame. The calibration of this system can be summarized in the following steps:

1. Calibrate each camera using a checkerboard and standard calibration functionality in OpenCV [1].
2. Calibrate the stereo system as above.
3. Define the beamer fixed frame as shown in figure 1. The different coordinate systems and the relevant names are defined in table 1.

In the following more detailed information on the registration of the coordinate systems and the positioning of the mounting system is given:

- The mounting system allows to vary the baseline of the stereo system as well as rotating the cameras about $\vec{T}_s$ and about the vertical direction. This is a challenge for the calibration which is tackled with a series of measures:

1. The distance between the cameras is found with the stereo camera calibration.
2. d_c is measured by hand and handed to the software as an argument in the top `CMakeLists`. d_b is also measured by hand and added to the distance between the projector and the wall.
3. The pitch angle of the cameras (rotation about $\vec{T}_s$) is assessed using face detection. The user places his face in front of the stereo camera and aligns vertically with the cameras. This position is recorded and its triangulation delivers the horizontal direction $\vec{e}_z$.

Using the results from the calibration above (see table 1), the orientation of the beamer fixed frame can be described as follows:

$$\vec{T}_s = -\boldsymbol{R}^{-1}\vec{T}$$
$${}_{\mathcal{A}_1}\vec{e}_1^{\mathcal{B}} = \frac{\vec{T}_s}{||\vec{T}_s||}$$
$${}_{\mathcal{A}_1}\vec{e}_2^{\mathcal{B}} = \vec{e}_z \times \vec{e}_1^{\mathcal{B}}$$
$${}_{\mathcal{A}_1}\vec{e}_3^{\mathcal{B}} = \vec{e}_1^{\mathcal{B}} \times \vec{e}_2^{\mathcal{B}}$$

Finally a transformation from $\mathcal{A}_1$ to $\mathcal{B}$ can be formulated using a homogeneous transformation

$${}_{\mathcal{B}}\vec{r} = \begin{bmatrix} \boldsymbol{R}_3 \begin{bmatrix} \left({}_{\mathcal{A}_1}\vec{e}_1^{\mathcal{B}}\right)^T \\ \left({}_{\mathcal{A}_1}\vec{e}_2^{\mathcal{B}}\right)^T \\ \left({}_{\mathcal{A}_1}\vec{e}_2^{\mathcal{B}}\right)^T \end{bmatrix} & \begin{bmatrix} d_c \\ 0 \\ 0 \end{bmatrix} \end{bmatrix} \begin{bmatrix} {}_{\mathcal{A}_1}\vec{r} \\ 1 \end{bmatrix},$$

where $\boldsymbol{R}_3$ denotes a rotation of 180 degree about the horizontal direction.
From this point the transformation to the wall frame $\mathcal{C}$ is a simple matter of shifting along the optical axis of the beamer.

3.2. Update Loop

Face Detection Face detection is performed over the frames captured on the two cameras facing the user. It is important to note that the face detection rate is slower than the camera image capturing rate. Therefore, all the frames recorded between each face detection are discarded. As mentioned before, face detection is performed using the HOG and SVM based algorithm of dlib. The algorithm returns the bounding box of the user's face. It's center point is later used to triangulate the user's position in camera coordinates.

Triangulation As the intrinsics and extrinsics of C_1 and C_2 are known, the user's face position in $\mathcal{A}_1$ coordinates can be obtained using triangulation [3]. This algorithm has been implemented for this project. With it, the user's face position, in $\mathcal{A}_1$ coordinates in an instant t, can be calculated given its position in a capture of C_1 and a capture of C_2 both taken at t.

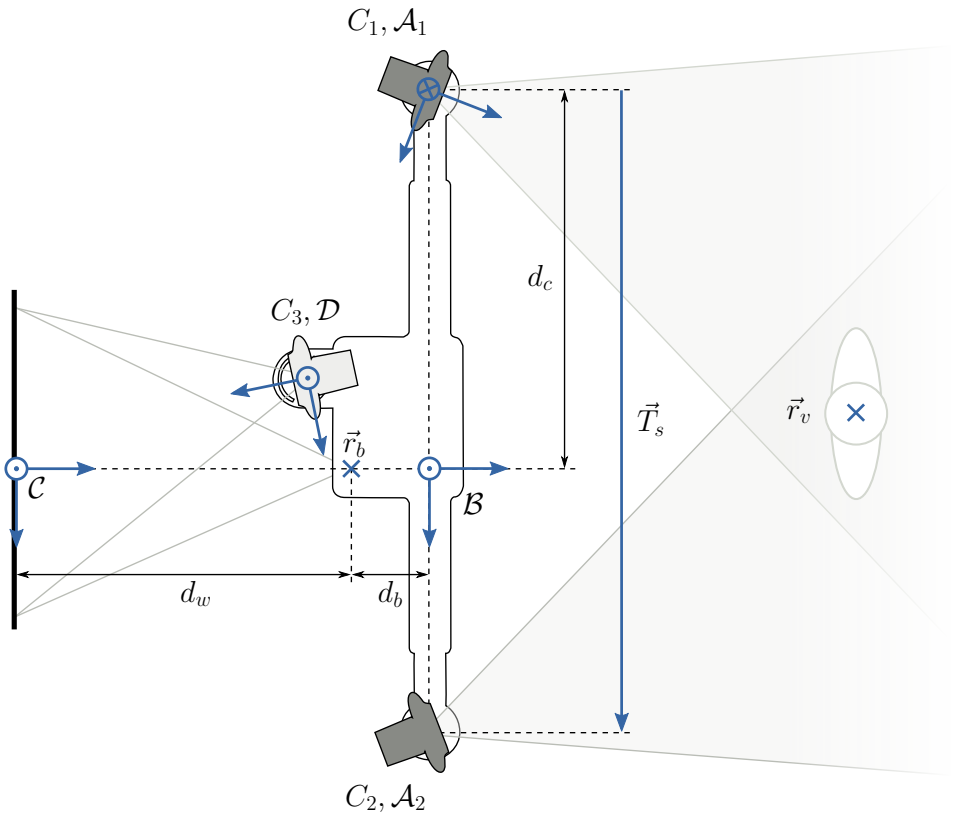

Figure 1. Depiction of the setup, showing the projector in the center, the projected window on the left and the viewer on the right. In dark grey are the cameras mounted bottom up, in light gray the camera mounted upright, in white is the mounting system, where below the center the projector is fixed.

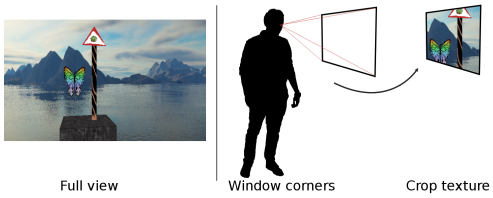

Figure 2. Rendering in two steps: First store the full view in a texture, then display part of it using the projector

Rendering the Scene The visual output depends on the eye position of the user and the dimensions of the projected area on the wall. For the latter, we assume that the projected area is a constant rectangle, which simplifies a few formulas and requires us to only calibrate the projector once. Given the viewer position as a result from the face detection part, we use OpenGL to render a simple 3D scene and display it on the projector.

The scene consists of a background skybox, a fixed cube with a sign on top, and a moving butterfly. There are two main steps to the rendering process, seen in 2. The first step consists on rendering the 3D scene from the eye position for a sufficiently big field of view. This rendering is then stored in a texture to be used in the second step. There, we compute the texture coordinates of the window corners. We then use these to display the part of the texture that is contained within the window, corresponding to the part of the scene that is actually seen through the window by the user. We also added an option to make the displayed image resemble an actual window by adding a 2D image of a window frame on top of the final rendering (this can be seen in 3).

The core of our OpenGL code is taken from [2], particularly ”6.1. Cubemaps”. Previous attempts to create everything from scratch failed to display any textures beyond simple color gradients.

4. Results

Our final application allows a single person to stand in front of the cameras and look through a ”window” that is projected onto the wall. At about 10 frames per second, the

Figure 3. Example of an output with window frame overlay

process may be interactive, but unless the user is moving very slowly, the difference to a real window is very obvious. This performance is mostly due to the face detection algorithm as testing with a fixed eye trajectory results in frame rates around 60 frames per second. There exists a trade-off between frame rate and maximum distance between the cameras and the user. This is specified with the resolution of the image as bigger ones allow for detection of smaller faces but require more computational power and thus have lower frame rate. The selected resolution, 480x360, allows to detect the user at a maximum of 2m and gives a frame rate of about 10fps. Parallelization of the face detection has been tried in order to increase the current frame rate. However, due to collisions with dlib, parallelization with OpenMP did not succeed. Moreover, parallelization using dlib's framework was also tested without any success.

If multiple people stand in view of the cameras, the perspective tends to jump randomly between them. The virtual eye position can also reach extreme positions when the viewer leaves the area in front of the setup.

A supplementary video for a qualitative evaluation is uploaded on Youtube [7].

5. Discussion

Currently, the 3D scene displayed outside the virtual window is hardcoded. Introducing a way to import meshes and textures directly without changing the code would allow for much more freedom in terms of what is seen in the virtual world.

The rendering pipeline is not optimized because the face detection turned out to be more expensive. The first steps in improving it would be to automatically compute a "sufficiently big field of view" that minimizes the amount of unused texture parts. Similarly, the resolution of the rendered texture is currently slightly higher than that of the display. Shrinking it to just what is needed to get a good output quality would further reduce unnecessary work.

Further improvements could be done by relaxing the assumption of a fixed orientation of the projector orthogonal to the wall as well as estimating the position of the viewer using a Kalman Filter. This would allow to run the algorithm smoother and partially circumvent the issue of low frame rates, making the impression of a virtual window more realistic.

References

[1] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek. A brief introduction to opencv. In *2012 Proceedings of the 35th International Convention MIPRO*, pages 1725–1730, May 2012. 4323

[2] J. de Vries. Learnopengl. https://github.com/JoeyDeVries/LearnOpenGL, Mar 2019. 4324

[3] R. I. Hartley and P. Sturm. Triangulation. *Comput. Vis. Image Underst.*, 68(2):146–157, Nov. 1997. 4323

[4] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *ICIP*, 2002. 4321

[5] D. Moreno and G. Taubin. Projector-camera calibration / 3d scanning software. 4322

[6] D. Moreno and G. Taubin. Simple, accurate, and robust projector-camera calibration. *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, 2012. 4321, 4322

[7] C. Uschatz, G. Mller, L. Wigger, and M. Vila Abad. Virtual projected window. https://youtu.be/zHXEV78XKo0, June 2019. 4325

[8] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137154, 2004. 4321