Kinect Projection Mapping

Thiago Motta, Manuel Loaiza, Alberto Raposo
Instituto Tecgraf de Desenvolvimento de Software Técnico-Científico
PUC-Rio – Pontifícia Universidade Católica do Rio de Janeiro
e-mail: {trmotta,manuel,abraposo}@tecgraf.puc-rio.br

Luciano Soares
Insper Instituto de Ensino e Pesquisa
e-mail: lpsoares@insper.edu.br

***Abstract*— Spatial augmented reality allows users to create a projected virtual environment on irregular surfaces. This demands an accurate Camera-Projector calibration process in order to produce precise 3D information to match the real object. This paper presents a framework to process data achieved from a calibration of a Kinect-Projector system in visualization applications, allowing the user to create an augmented reality environment without having extensive process of the Camera-Projector calibration, while maintaining a precise calibration to the projection on irregular surfaces. Additionally, different calibration techniques were evaluated in order to demonstrate the better approaches.**

Keywords—Calibration; Projection Mapping; Spatially Augmented Reality;

I. INTRODUCTION

Virtual systems that augment the real world characteristics have been showing to be relevant for several applications. The Augmented Reality technology [1] mixes virtual reality information with a real environment, traditionally adding graphics and sound to what we see and hear in order to create a unique experience in virtual interaction and simulation.

Tracking systems [2][3] have been developed allowing the geometric localization of a user on a visualization platform. With a set of cameras, Infrared (IR) light sensors and the use of markers to find specific points in the world, they modify the virtual environment based on a person movement. These systems, although reducing the need of non-intuitive interaction devices, still depend on displays to present data.

Traditional Projection Mapping environments, focused only on visualization [4][5], use a static mapping of the scene, measuring the dimensions of each object in this ambient, and then reproducing these objects at the virtual scenario. This kind of environment needs an extensive calibration in order to achieve an accurate mapping of the world, having a high cost of operation and not being able to be reused.

Taking these difficulties into account, the concept of dynamic mapping/projection mapping [6] was established. This concept defines the outcome attained when calibrating a scenario without measuring it, thus working for whichever objects placed at the scene, regardless whether they were added before or after the calibration.

In the context of this research, a framework was developed in order to take the calibration data into account and achieve a dynamic mapping using the Microsoft Kinect.

The developed framework aims at users wanting to study or use a tool capable of creating Augmented Reality environments in a dynamic and simple way. A Projection Mapping system can be used in a wide variety of cases, such as:

- Product presentation [7]
- Exhibitions [8].
- Advertising [9].
- Live performances [10].
- Entertainment [11][12].
- Medicine [13].

The proposed framework includes the following characteristics:

- Process data obtained from a Kinect-Projector system calibration in a more comprehensive way.
- Insertion of virtual 3D graphics in the captured scenes.
- Augmented scene reprojection.

The ultimate goal of this framework is to allow graphic applications to make use of the data derived from a Kinect-Projector calibration and apply them to a real environment, creating an Augmented Reality visualization of the captured scene in runtime.

The remainder of this paper is organized as follows: Section II presents related work to Projection Mapping. All the procedure required to achieve a Kinect-Projector system calibration is described in Section III. Section IV details the preliminary studies necessary in order to attain the proposed goal. How to take advantage of the Kinect system is presented in section V. System specification is described at section VI. Section VII presents the results, and Section VIII concludes the paper.

II. RELATED WORK

A Kinect-Projector system calibration has been used in projects such as: RGBDemo [14], ProCamCalib [15], OpenLight [16], and CameraProjectorCalibration [17]. These projects identify the aforementioned problems and aim to calculate the intrinsic and extrinsic parameters of the projector, in the case of Kinect, its IR and RGB cameras are both calibrated.

The initial part of this research consisted on studying the four calibration methods above in order to better understand their differences, pros and cons. After this, it was necessary to

create a solution to generate a dynamic mapping, thus validating the obtained calibration.

The idea of integrating visualizations alongside projections is not entirely new. However, previously, in order to receive a proper projection, the objects had to be known in advance or had to go through a reconstruction phase, which, besides compromising runtime visualization, also generated artifacts during the scanning process due to structured visible light technique.

One of the most traditional works on irregular surfaces image projection is the iLamps [18], where the authors present techniques for adaptive projection on non-planar surfaces, using a textured projection algorithm. With a different approach, the work developed at the Sunnybrook Health Sciences Center uses markerless tracking devices in order to see and navigate in tomography and magnetic resonance imaging data without the need of a keyboard and mouse [19], using hand gestures to browse the data layers to be visualized.

Image projection on a person's body with Kinect's aid is presented in an application of a futuristic massage [20]. The masseur uses a projector to project flow lines on a human body. However there is not an accurate matching of the image being projected with the body itself and the projected images cannot be used for further analysis.

A Kinect-Projector system has been also used in a system for surgical planning [21]. In this system, a spatially augmented reality surgical environment is constructed directly on the patient body. The registration method proposed in that paper uses fiducial markers attached onto the patient's skin in order to produce an accurate AR display on the physical body of the patient.

In spite of the variety of proposals in the area of visualization integrated to projection, we found only very few systems able to solve the problem of dynamic projection mapping with Kinect and with the flexibility to be used with most of the existing calibration methods.

III. Coordinate Systems Calibration

Before accomplishing a mapping, it is essential to calibrate a system, which commonly consists of a camera-projector pair.

Camera calibration, whose purpose is to extract the extrinsic and intrinsic parameters of a camera, is a major difficulty in Computer Vision. The obtained parameters define the 3D position, rotation, focal length, center of image, and distortion coefficient of a camera. These data are necessary to insert virtual models in a real environment captured by a camera.

A. Single camera calibration

The first calibration method is that of a single camera, being required to estimate, at the image obtained from the camera, a series of points p1, p2, ..., pn that matches known points P1, P2, ..., Pn of the tridimensional space. Once the points are found, it is possible to calculate the intrinsic and extrinsic parameters of the camera, such that the virtual points p'1, p'2, ..., p'n overlap in the best possible way the observed points.

A chessboard, with known dimensions, as seen in Figure 1, is a pattern frequently used to represent the known points. This pattern is moved in front of the camera's field of view while several snapshots are being made, keeping the different positions and rotation of the chessboard. The most used algorithms to perform this processing are the ones developed by Zhang [22] and Tsai [23].

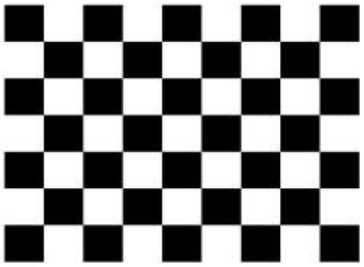

Figure 1 – Standard chessboard pattern 9x7

A calibration is accomplished using the coordinates system below, so that the camera transformation can be defined as the set of transformations performed by these systems:

- World Coordinate System (WCS)
- Camera Coordinate System (CCS)
- Image Coordinate System (ICS)
- Pixel Coordinate System (PCS)

$$[p] \simeq \underbrace{\begin{bmatrix} s_x & \tau & u_c \\ 0 & s_y & v_c \\ 0 & 0 & 1 \end{bmatrix}}_{(A)} \underbrace{\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{(B)} \underbrace{\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}}_{(C)} [P];$$

Equation 1 – Affine transformation

Equation 1 defines the following transformation:

(C) Multiplication by the RT matrix defines the transformation from WCS to CCS, called Change of 3D Referential.

(B) Multiplication defines the transformation from CCS to ICS, named Perspective Projection.

(A) Multiplication defines the transformation from ICS to PCS, called Affine Transformation.

Where in the equation:

f is the focal length;

s_x and s_y are the amount of pixels per length in both axis;

u_c and v_c are the orthogonal projection coordinates of the optical center on the projection plane;

τ is given by the tangent angle that the pixel matrix row makes with the perpendicular lines;

R and T correspond to the rotation matrix and the translation vector, respectively.

The final result is presented in Figure 2, where it is possible to note that the virtual points (the colored circles and lines) are overlapped to the inner pattern corners. Notice that even if the pattern is translated or rotated, the virtual points will remain overlapped to them.

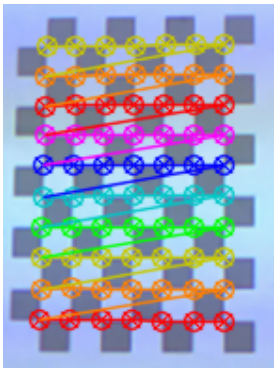

Figure 2 – Photo of calibration on a standard chessboard.

In spite of working for a simple configuration, this approach is not enough for the goal of the present research, but it works as a basis to achieve a higher-level complexity.

A dynamic-mapped system consists primarily of a camera, used to acquire the correspondence from the points at the WCS and the PCS, but it is still needed to present the user with the virtual data overlapped to the calibrated scene. To achieve such task, it is necessary to utilize a camera-projector pair, where the projector can be interpreted as the dual of a camera, thus requiring the calibration of two cameras.

B. *Calibration of Two Cameras*

Noting that each camera holds its own calibration parameters, we define as CCS1 and CCS2 the coordinate system of each camera, denoted by $[R_1 \quad T_1]$, and by $[R_2 \quad T_2]$, respectively, in relation to the same WCS used at the calibration.

The extrinsic parameters transformation of the first camera can be obtained by multiplying by $[I \quad 0]$, since the system is already at the desired position. Based on that, in the same way that a camera calibration is made by acquiring the points from the CCS in relation to the WCS, the calibration of two cameras is made by acquiring the points from CCS2 in relation to CCS1.

In order to obtain the extrinsic parameters of the second camera in relation to the first one, we need to find at CCS2 a known point to the CCS1, which comes down to attaining the rotation and translation matrix of a coordinate system in relation to each other. This can be done by creating a transition from CCS1 to WCS to then change from WCS to CCS2, where the first transition was made with the multiplication by $\begin{bmatrix} R_1^t & -R_1^t T_1 \\ 0 & 1 \end{bmatrix}$ and the second change made by multiplying by $\begin{bmatrix} R_2 & T_2 \\ 0 & 1 \end{bmatrix}$. In other words, the coordinates P1 and P2 of a point in space with relation to CCS1 and CCS2 satisfies the relations $P2 = RP1 + T$ and $P1 = R^t P2 - R^t T$.

C. *Calibration of a Camera-Projector*

The internal optics of a camera is close to that of a projector, thus both can be modeled in the same way.

The camera-projector calibration is achieved in a similar way to a calibration between two cameras, in which both cameras or the camera-projector pair demands to be calibrated regarding a common reference system in the world. The calibration allows correlating the points in the world to the captured points; therefore both cameras are positioned to capture the same scene.

The difference between calibrating two cameras and calibrating a camera-projector pair is that the projector does not capture points in a WCS, so the idea is to project a known image and determine the points found in the scene, as seen in Figure 3.

A convenient way to do it is by using a virtual checkered pattern (projected) over the same plane where the physical checkered pattern is located (printed). The inverse transformation of the projector is applied to the transformation of the camera to obtain the coordinates on the WCS from a point visualized by the projector.

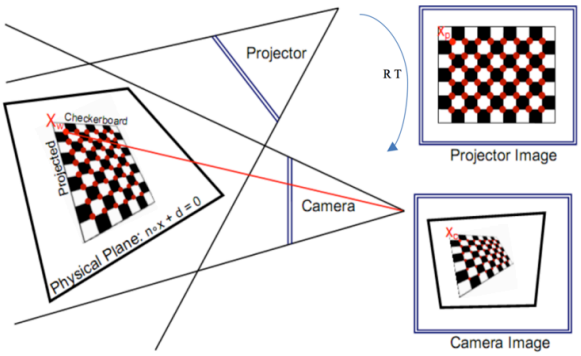

Figure 3 - Representation of a Projector-Camera system [16]

The calibration of the projector-camera system allows the identification of the 3D points on a scene and the reprojection of them over themselves. However, to truthfully reproduce a 2D image over a real 3D object, it is necessary to compute the texture extracted from the real world scene, to then apply the 2D image over the texture and finally project the result. To simplify the process, the Kinect was chosen as camera since it already has a system to capture the depth of an image.

D. *RGB-D – Kinect Camera Device*

The Kinect is a 3D camera sensor developed primarily as a natural interaction device. Taken its low cost in comparison to commercial RGB-D systems (D means depth) this system has been used for various goals, including mapping and 3D modeling [23].

The Kinect device has three key elements which allow capturing simultaneously an image with depth and color: a RGB Camera with a resolution of 640 x 480; an IR Camera of 640 x 480, and an IR Projector.

Having a camera-projector pair and a system named Light Coding [24], the Kinect is capable of detecting its distance from a point, turning it into a 3D runtime measuring device. The Light Coding, presented in Figure 4, encodes information on light patterns leaving the IR projected image over any surface, generating a deformity on the projected pattern. This offers the necessary information to calculate the distance for the 3D image.

Figure 4 - Kinect Light Coding

Operational features:

- Viewing angle: 43° vertical by 57° horizontal.
- Tilt's motor movement interval: ±27°.
- Audio: 16 kHz, 24-bit mono pulse code modulation.
- Audio: 4 microphones array with analogical-digital convertor (ADC) of 24 bits.
- Accelerometer: 2G/4G/8G, with 1° precision.
- Operational depth detection limits: 0.8m – 4.0m.
- Frames Rate: 30 frames per second.

E. Kinect-Projetor System Calibration

A Kinect device brings a new level of complexity: it includes another camera in the system. What was previously summarized as:

1) Capturing data from the scene using a RGB camera, resulting in 2D data;
2) Reprojecting the data using a projector.

Now, with the Kinect, becomes:

1) Capturing the data from the scene with an IR camera, resulting in 3D points;
2) Transforming those points to the space of the RGB camera;
3) Applying a new transformation to be sent to the projector space;
4) Reprojecting the data using a projector.

Kinect has a standard calibration of its sensors; nevertheless this calibration only provides an approximation of the points' correlation of different sensors and, for that reason, a calibration between them is necessary.

Regardless of following the same logic of calibrating two cameras, taking into consideration that one of the cameras only captures IR, an extra caution is added: It is necessary to separate the obtained images from the calibration in two different groups – the group which has the IR emitter blocked, therefore having no IR lighting, and the group which has IR lighting.

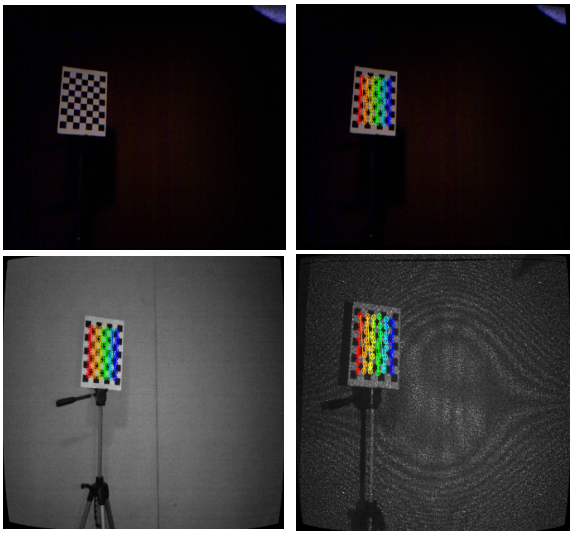

Figure 5 - IR-RGB Calibration

Figure 5 shows some images of the procedure of calibrating a Kinect system, in which the pattern to be perceived is presented in the image on the upper left. The upper right image represents the recognition of the pattern by the RGB camera. The lower left image presents the recognition of the pattern by the IR camera having the IR projector being obstructed and presenting external IR lighting. The lower right image represents the recognition of the pattern by the IR camera without obstruction of the IR projector. Notice that in the last case the image gets polluted, which can hinder the recognition of the pattern or generate wrong/imprecise recognition.

IV. Data Visualization

Knowing that Kinect is capable of recovering 3D data of a scene, it is natural to work with the data in a 3D environment in order to take advantage of the obtained information. Aiming to achieve this goal, traditional point cloud visualization was adopted, shown in Figure 6, consisting of a collection of 3D points that are not connected, providing a sparse visualization.

According to Nicolas Burrus [14], using a point cloud is a natural way to represent data from the Kinect since each pixel on the image of depth camera can be transformed into a 3D point.

A graphics rendering tool was necessary to create a virtual environment that uses the result of the Kinect-Projector system calibration with the point cloud and which also allows the addition of new elements to the scene.

OpenGL is a graphic library, which specifies a render API, and by its implementation not being structured by a platform, it is possible to use it in different architectures. In order to use OpenGL routines, the following spaces coordinates need to be defined.

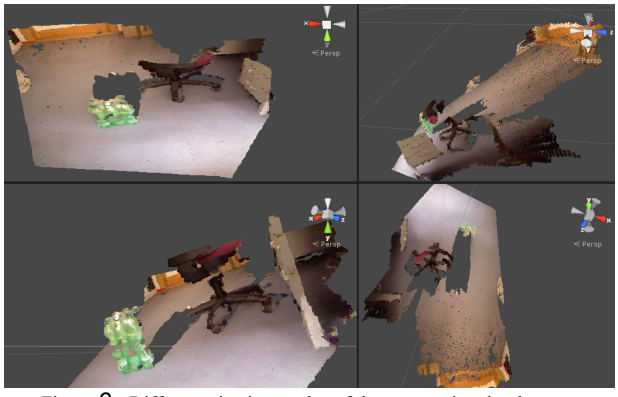

Figure 6 - Different viewing angles of the same point cloud.

A. Object Space Coordinates

This is the local coordinate system of an object, also known as Camera Coordinate System and it is the initial state of an object. This is the matrix found before any transformation is applied over this object.

The transformations provided by OpenGL are glRotatef() to rotate an object on a determined angle at a determined axis, glTranslatef() to translate an object and glScalef() to modify the size of an object at any axis.

B. Eye Space Coordinates

On OpenGL, objects are transformed from the Object's Coordinate System to the Eye's Coordinate System using the GL_MODELVIEW matrix. The GL_MODELVIEW matrix is a combination of matrices of Model and View ($M_{view} \cdot M_{model}$).

The Model transformation, Equation 2, is used to convert the Object's Coordinate System to the World's Coordinate System, while the View transformation is used to convert the World's Coordinate System into the Eye's Coordinate System.

$$\begin{pmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{pmatrix} = M_{modelView} \cdot \begin{pmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{pmatrix}$$

Equation 2 – Model-view transformation

C. Clipping Space coordinates

When the GL_PROJECTION matrix multiplies the Eye's Coordinate System, the result is the Clipping's Coordinate System. The GL_PROJECTION matrix defines the volume of visualization (frustum), which is how the vertices are projected on the screen (by perspective or orthogonal).

D. Normalized Coordinates Space

The division of the Clipping Coordinate System obtains the Normalized Coordinate System by the w component, known as perspective division, as observed in Equation 3.

$$\begin{pmatrix} x_{ndc} \\ y_{ndc} \\ z_{ndc} \end{pmatrix} = \begin{pmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \end{pmatrix}$$

Equation 3 – Perspective division

This new system works as the Screen Coordinate System, without the translation and the scale to the screen pixels. The interval of values gets normalized between -1 and 1 in the 3 axes.

E. ModelView Matrix

The OpenGL ModelView matrix is presented in Figure 7, the elements m12, m13, m14 are for the translation, glTranslatef(). The m15 element is a homogeneous coordinate used for projective transformation.

The three sets of elements, (m0, m1, m2), (m4, m5, m6) and (m8, m9, m10), are for the Euclidean transformations, performing the rotation's transformation glRotatef() and scale glScalef(), while this sets being combined with the elements (m12, m13, m14) are part of the affine transformation. Ahead, the three initially mentioned sets represent 3 orthogonal axes:

(m0, m1, m2): axis +X, vector left, (1, 0, 0) by standard
(m4, m5, m6): axis +Y, vector up, (0, 1, 0) by standard
(m8, m9, m10): axis +Z, vector forward, (0,0,1) by standard

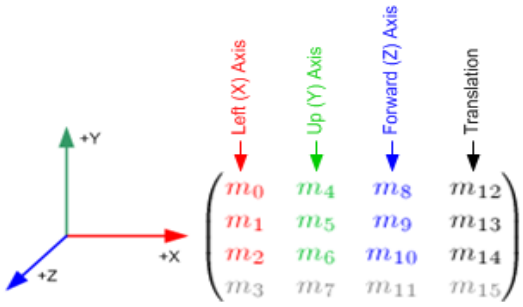

Figure 7 – 4 Columns GL_MODELVIEW Matrix

F. Projection Matrix

A computer display is a 2D surface, therefore, a scene rendered in 3D by OpenGL needs to be projected on the computer screen or projector as a 2D image and the GL_PROJECTION matrix is responsible for this transformation. It first transforms all the vertices of the Eye's Coordinate System to the Clipping's Coordinate System, so the Clipping's Coordinate System is transformed into the Normalized Coordinate System as each component of the Clipping's Coordinate System is divided by its corresponding w component.

The GL_PROJECTION matrix is used to define a frustum. This frustum determinates which objects will be cut from the scene. Also, it determines how the 3D scene is projected on the screen.

The OpenGL provides two ways to accomplish a GL_PROJECTION transformation:

glFrustum(): to produce a perspective projection.
glOrtho(): to produce an orthographic projection (parallel).

In the Perspective Projection, one 3D point on a pyramid of truncated frustum (in the Eye's Coordinate System) is mapped to the cube (in the Standardized Coordinate System), where the interval of its coordinates in the X axis changes from [left, right] to [-1,1], in the Y axis changes from [bottom, top] to [-1,1] and in Z axis changes from [near, far] to [-1,1] (Figure 8).

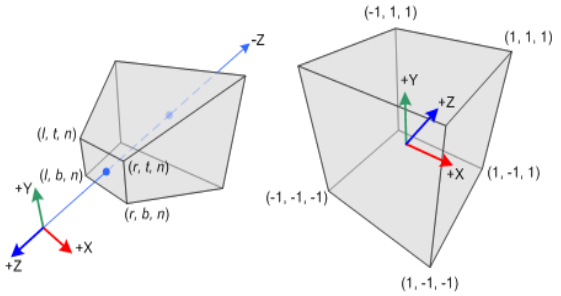

Figure 8 – Perspective Frustum and Normalized Coordinate Space

It is possible to perceive that the Eye Coordinate System is defined according to the Right Hand Rule, however the Normalized Coordinate System is defined by the Left Hand Rule, meaning that the camera, in the origin, is looking to the –Z axis on the eye system but looking to the +Z axis on the Normalized Coordinate System.

V. Kinect Access Preliminary Studies

The following libraries were studied to properly use the Kinect device: OpenNI [26], Microsoft Kinect SDK [27] e Open Kinect [28].

The Microsoft Kinect SDK is the most modular library to be used in the development of new applications, presenting well-defined modules, a range of examples available and full documentation. However, only a few examples which used this library were found regarding to the scope of this work.

The Open Kinect, despite of being one of the first libraries to provide support to Kinect, does not have a large community, with about 2000 active users, and its advances depends on the few users interested on improving it.

At last, there is the OpenNI library, created and developed by the creators behind the Kinect technology, PrimeSense [25]. It was the easiest to learn library, besides having a variety of demonstration programs and a manual with details and codes to understand how to use the main resources of Kinect. Taking into consideration the presented reasons, this was the library chosen for the development of this research.

Having the SDK for the Kinect been chosen, it was necessary to look for previously existing projects that solved the calibration of the Kinect-Projector system.

A. *RGBDemo*

RGBDemo [14] provides, among other functionalities, the Kinect camera calibration and the camera-projector system calibration. The driver used to calibrate the Kinect was the libfreenect, while the driver used to calibrate the Kinect-Projector system was the OpenNI. It was necessary to use both drivers once that OpenNI does not provide the option for simultaneous visualization between RGB camera and the data from IR camera. The resulting calibration format is YAML [29] and it consists of all the intrinsic and extrinsic parameters obtained. The estimated time to make a calibration was between 5 and 15 minutes for a set of 30 images, which not always had a coherent result, being then necessary to repeat the previous steps several times. By the end of the whole process, the consumed time could vary from 40 to 90 minutes, where the time taken by the algorithm to process all the images and generate the calibration matrices lasted about 10 minutes, while the rest of the process was used for the positioning of the chessboard.

B. *CameraProjectorCalibration*

This method [17] aims to be fully automatized, not requiring that the user provides commands through the keyboard to select an image, taking less time than the RGBDemo. The driver used on this method was the Microsoft Kinect SDK, once the program works together with OpenFrameworks [30], which does not support this SDK, and it was not possible to remove this dependency. Therefore it was necessary to emulate [31] the Kinect camera with a webcam in order to the CameraProjectorCalibration to interpret it as a video source and use the RGB camera information as an input to calibrate the Kinect-Projector system. Even though this method has a calibration between a pair of cameras, it was not possible to use it once the integration with the Kinect driver through the webcam did not provide the result of the IR camera and, therefore, the camera calibration used here was obtained with RGBDemo.

The total calibration time varies from 25 to 50 minutes, but this method achieved a coherent result on the first or second attempt, ensuring a depth allowed variation of almost two meters. Despite the total calibration time being shorter than the RGBDemo, the algorithm's processing time of this method is superior, being responsible by at least 15 minutes of the total time.

In spite of the differences between RBGDemo and the CameraProjectorCalibration, the following conclusions were observed over both solutions:

It is not recommended to perform the calibration with more than 30 images once the complexity of the equations used to find the intrinsic and the extrinsic parameters grow according to the numbers of images obtained, requiring more time and presenting higher chances of bad outcomes.

All of the calibrations were performed after a 60 minutes period of the Kinect-Projector being turned on, following the study presented in [32].

In order to perform a calibration that covered the biggest projection/vision area of the Kinect, it was necessary to adopt the following procedures to generate new images:

- Rotating the pattern for at least 5 angulations of a maximum of 20º each, as shown in Figure 9.1 (no rotation, rotation left, right, up and down and/or with some concatenations).
- Once having used the various possibilities, the pattern was moved to a new height, as shown in Figure 9.2 and the process repeated itself.
- Having explored at least 2 heights, the pattern is moved horizontally in order to explore the best possible way the system's field of vision, as shown in Figure 9.3.
- Finally, for the calibration of the Kinect-Projector system, it was necessary to also explore the depth of calibration, as shown in Figure 9.4. So, in addition to the procedures previously described, it was necessary to approach/distance the Kinect/Projector.

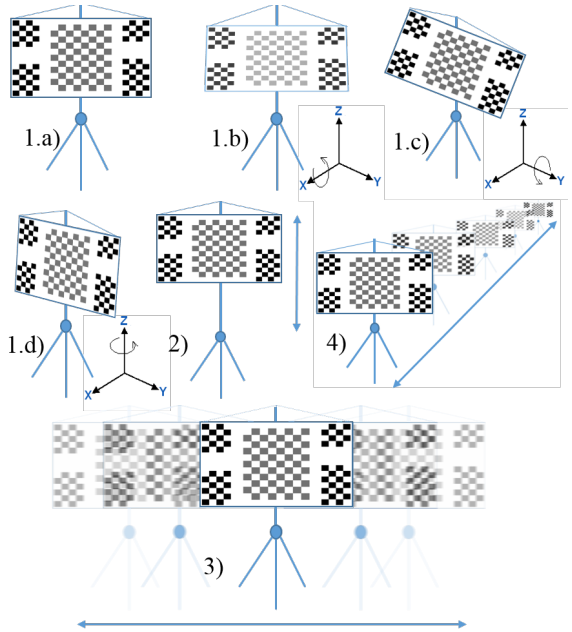

Figure 9 - Different positions and angulations used during the calibration.

To reassure that the result of the calibration was consistent, two different positions were tested with the Kinect relative to the Projector, as seen in Figure 10: having a rotation between them (Figure 10, left) and the case in which the Kinect was directly over the Projector and there was no apparent rotation between them (Figure 10, right). It was important to test these cases to prove the correctness of the calibration methods because the first case generates a system of equations more complex to be solved; therefore, it is more prone to failure. Once it was proven that the calibration still worked, even though it was not the ideal scenario, it was changed to the superposition of cameras so that a better result could be achieved.

Having finalized these studies, to get to the desired final result, it was necessary to study efficient methods which presented correct data from the Kinect, considering one method to visualize 2D data and another to visualize 3D data, as presented in the following.

Figure 10 – Different positions used for the Kinect-Projector System.

C. *Base Application: NIViewer*

In order to aid the development of the library that used the features offered by the OpenNI, the NIViewer was utilized as a base program. This program offers the main functions developed for the Kinect use, such as visualization of RGB Camera and IR, and also different depth maps, multiple visualizations, and possibility of audio and video recording, in addition to its management.

Figure 11 presents the main visualization methods offered by NIViewer, in which all are done from the Textures' Mapping in OpenGL, delivering only a 2D visualization of the resources available on Kinect. Despite the visualization not being in three dimensions it is possible to notice how the images under Depth Mapping present a better idea of depth than the one offered by the RGB Camera.

Figure 11 on the left shows a Depth Mapping made with a single color scale, in which the more the contrast the color presents, the closer the point is to the Kinect Device.

Figure 11 – Depth Mapping (A) and Colors (B).

D. *Point Cloud Library*

The Point Cloud Library (PCL) [33] offers wide visualization resources, management and treatment of point clouds, therefore adding several dependences to a project that would only use a point clouds' visualization. Aiming to simplify this research's development, a module capable of achieving similar results to PCL visualization was developed, as presented on Figure 12.

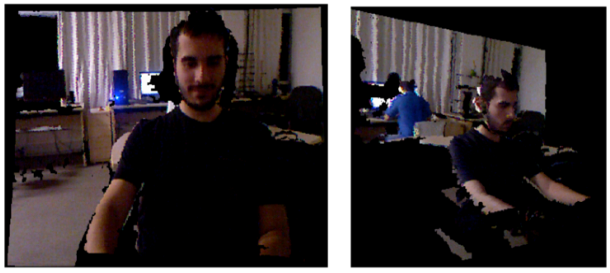

Figure 12 – Frontal view (left) and side view (right) of the point cloud.

VI. Project and System Specification

The developed framework details are described below:

A. System's Architecture:

1) Extraction of intrinsic and extrinsic parameters from the Kinect's RGB camera relative to the Kinect's IR camera:

a) Move the calibration pattern in front of the cameras.

b) Take diverse snapshots of the calibration pattern in different positions and angulations.

c) Process the snapshots using the RGBDemo calibration program.

2) Intrinsic and extrinsic parameters extraction from the projector relative to the RGB Kinect Camera:

a) Position the calibration pattern in front of the cameras.

b) Take diverse snapshots of the calibration pattern in different positions and angulations.

c) Process the snapshots using the RGBDemo calibration program or the CameraProjectorCalibration.

3) Correlation between the points of both calibrations:

a) Accomplish the transformation of the point in the IR Camera's Coordinate System to the Projector's Coordinate System.

4) Verification of quality in both calibrations:

a) Use a Point Cloud to validate the calibrations of the previous steps as verifying that the projected objects are over the real objects.

b) Case the previous step fails, restart from step 1.a)

5) Insertion of extra information to the real scene to create Augmented Reality:

a) Create a treatment on the Point Cloud generated on step 4 to insert visual information to the real scene.

Figure 13 presents a graphic showing the process.

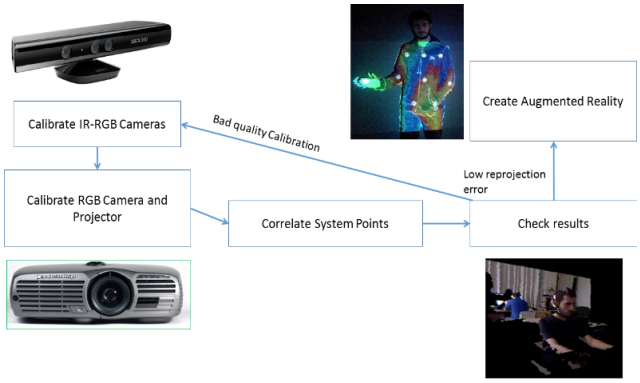

Figure 13 – System's architecture

B. System's Characteristics

The program and all of its dependencies were written in C++ and were developed in Visual Studio 2010. The driver used to capture the information from the Kinect was OpenNI 1.5.4 version, along with Prime Sense NITE driver 1.5.2 version and Sensor Kinect 5.1.2.1 version.

As dependencies, the Glut version 3.7.6 library was used and the OpenGL 3 version to the creation and management of the graphic elements. OpenCV [34] version 2.4.2 was utilized to ease the handling of algebraic data from the calibrations.

The projector used on the tests was the Projection Design evo2sx+, contrast of 2500:1, 2500 ANSI Lumens, 1600x1200 resolution and throw ratio 1.78 - 2.23 : 1.

The computer used on the tests was an Intel 2_{nd} Gen. Core i7-2670QM processor, 4 physical cores, 4 logical cores and clocked at 2.20 GHz, RAM memory of 8.00GB, GeForce GT540M dedicated graphic card, Intel(R) HD Graphics 3000 integrated graphic card, HD Serial ATA-300 of 750.0 GB, Microsoft Windows 7 64-bits and a screen resolution of 1600x900.

The calibration patterns used were printed on bond paper weighing 75 g/m2, A4 size for the Kinect calibration and A1 for the Kinect-Projector calibration. Both physical patterns were attached to featherweight paper and later attached to a camera tripod.

The projector was positioned upside down so the generated image would have as the highest point the height of the projector and its lowest point according to the projection distance. This solution was necessary so the Kinect could capture the whole projection area without generating occlusion and a person with an approximate maximum height of 1.8m could be projected on.

C. Libraries and Modules created

In order to recover the data obtained by the calibration between the cameras, a module called *Calibration* was created based on the Cámara Lúcida program [35]. This module is responsible for loading the calibration matrices on the program so that, in the future, they could alter the Modelview and Projection of OpenGL matrices and, in addition, to store the intrinsic and the extrinsic data of each camera in its corresponding device.

Modelview and projection matrices creation was done by the OpticalDevice module, which solves the required algebra as explained on Section III, in order to use as a set all of the matrices obtained by the calibration between the cameras.

The HelpMenu module was created to aid users to provide, on the screen, the necessary hotkeys to handle the program and its tooltips.

To make it possible to have different visualizations of the Point Cloud, it was needed to create a module capable of interpreting movements from the mouse and handling the Point Cloud according to their movements.

D. NIViewerLib

NIViewerLib is a library to access the functions offered by Kinect, and aiming reuse, management and maintenance of methods that utilize the resources offered by Kinect, a library that encompasses the main functions was developed, including:

1) Startup.
2) Acquisition and visualization of different streams offered by OpenNI.
3) Resizing the stream.
4) Test on the OpenGL context using simple matrices of Modelview and Projection.

A library for functionalities offered by Kinect was created. This library furnishes a modular use of the necessary methods, encapsulating modules of direct access to Kinect's raw data and enabling only the visualization of necessary functions.

The prototypes created of OpenNI library provide the same functionalities, all presented in 2D:

1) RGB/IR Camera's visualization.
2) Different depth maps over the IR Camera's visualization.
3) RGB Camera's visualization with depth information.
4) Side by side visualization from both cameras.
5) Management and visualization of the Skeleton in 2D and 3D.

E. Point Cloud Viewer

Modifications in the OpenGL software were conducted to ensure the use, management and maintenance of a point cloud. It is possible to create a depth map with a transparency factor – over the RGB Camera's Point Cloud –, providing easily understood information for the user about the distance in which each point is found. This mode will be referred here as Point Cloud with Depth Mapping. Figure 14 presents points that are closer to the Kinect under shades of blue; meanwhile the further ones are under shades of green.

The Point Cloud's software with Depth Mapping was necessary because, when trying to validate a calibration with the prototype, only the Point Cloud of the RGB Camera was being projected over the scene. We verified this wasn't an efficient validation method since it wasn't viable to visualize whether the projected points truly overlap the real points once both had the same color. With the depth mapping information and with the transparency factor added to the Point Cloud, it was easier to perform the validation.

Figure 14 - Front view (left) and lateral (right) Depth Point Cloud Mapping

VII. Results

The developed program allows the user to load the calibration files. Once the files are loaded, the program interprets the given matrices and generates the Modelview and the Projection's matrices necessary for the mapping projection. It is important to notice that the matrices given by the calibration program are on the OpenCV format and, therefore, are row oriented, requiring them to be transformed to the OpenGL format, which is column oriented.

The results obtained with the framework are presented on Figure 15, Figure 16, and Figure 17. All of the mapping formats presented by the framework are made with support from the point clouds and can be seen, on runtime, on any type of surface, whether it being well defined or not. Figure 15 presents a chessboard rotated about 30º relative to the Kinect-Projector system, as an example of a well-defined surface. It is possible to observe on this image that the depth difference, which exists from the lower left to the upper right border of the chessboard. The mapping used to present the difference is based on a color cycle from red to blue.

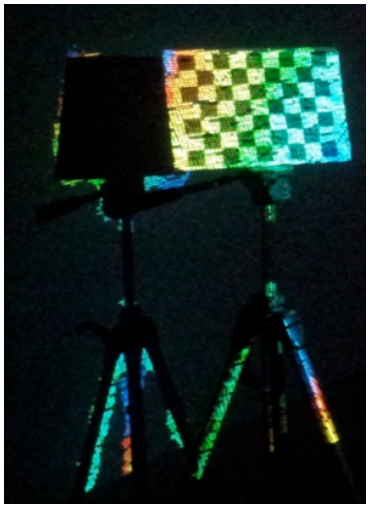

Figure 15 – Chessboard on a tripod under Depth Mapping with Cyclic Scale.

Figure 16 (left) allows the visualization of even more depth levels, accurately emphasizing the differences of distances between the shoulders, the torso and the hands.

On Figure 17 (right) the colors information obtained by the Kinect's RGB camera accurately overlaps the objects contained in the scene. On Figure 17 (left) is presented the Mapping Skeleton accurately overlapping the user. Both Figure 16 and Figure 17 are examples of non-smooth surfaces.

Figure 16 - Left: Depth Mapping with a Cyclic Scale. Right: Depth Mapping with a Two Color Scale. In both cases the skeleton mapping overlaps the corresponding mapping.

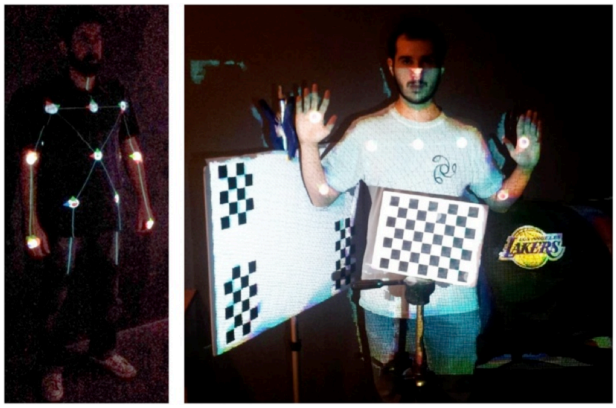

Figure 17 - Left: Mapping Skeleton. Right: Mapping Depth with the RGB camera information

The more shades a scene has and the less linear the borders are, the smaller are the chances of the whole surface to be fully and properly lit by the Kinect-Projector System, resulting in less information generated to the scene and worsening the behavior of the Mapped Projection to these objects.

A. *Difficulties encountered*

Almost all of the problems found by the end of the system development are part of the calibration process. They are presented below:

1) Lack of control of White Balance and Auto Exposure on Kinect: to be able to successfully use the Projector over the same object Kinect is observing, it was necessary to utilize the Microsoft SDK driver. However, the Kinect used was a XBOX 360 version, instead of the Kinect for Windows, and because of this, it was not possible to have any kind of control over the RGB camera. This drastically hindered the calibration since the light emitted by the projector, once filtered by the Auto Exposure, ended decreasing the natural lighting, worsening the physical pattern recognition. Four options were found to surpass this problem:

a) Add a new high quality camera to the system and execute the calibration of this camera with the Kinect RGB camera, adding a new complexity to the system.

b) Create a driver capable of recognizing other Kinect drivers as webcam. Drivers such as libfreenect and OpenNI 2.x version claim to have the necessary controls to surpass this problem, but building this driver would scape the envisioned theme.

c) Perform a refactoring in the whole calibration project in order to remove the dependency of the CameraProjectorCalibration input.

d) Acquire a Kinect for Windows. The tests were done using a standard version of the XBOX with an USB adaptor.

2) Kinect's RGB camera has a low quality image. It would be needed that both physical and logical patterns to be close to the Kinect so that the detection could be successful. Since the Kinect was positioned above the Projector and that the closer the pattern is to the Kinect, smaller the projection is going to be, this scenario hinders once again the projection of patterns. A balance between calibration quality and calibration distance had to be found. That is the reason why the physical pattern was printed on A1 paper: so that it could cover a larger calibration distance.

3) The ProjectorCameraCalibration program has performance bottlenecks, delaying the response of the system when capturing an image. Since the system is automated, these delays could often mean that the image was captured during the movement of the pattern – generating blurred images and resulting into low quality calibrations. The reason for this bottleneck was not found, therefore, the method found to minimize it was to increase the interval between the picture capturing, considerably hindering the total calibration time.

4) The amount of time to perform a calibration under any of the presented methods were superior to the calibration time of higher quality cameras.

5) Since the visualization program uses the OpenNI driver and the calibration program uses the Microsoft Kinect SDK driver, it is necessary to have both drivers installed in the computer and make the necessary exchanges between them, enlarging the dependency as a whole to obtain good results.

6) On Figure 16, it is possible to perceive that the projection goes beyond the body area. This happens because:

a) Since the visualization is made with a point cloud, each visualization pixel is a point designed by OpenGL, however the drawing of this point by the OpenGL has a minimum size, which is bigger that the one needed to properly fulfill the whole area.

b) When this image was made, the pixel size was defined as 3, but changing this point to size to 1 made the point cloud to look sparse, hindering the visualization.

c) Each point in a point cloud is given by the Kinect so, aiming to avoid the point cloud to look sparse with a smaller point on OpenGL, it would be necessary that the Kinect's IR projector could project more points.

d) Projection error: one of the parameters to verify if the calibration went properly is the projection error value, where the larger the number the less accurate the calibration was and the smaller the overlapping was going to be between the real and the virtual scene. Since in order to perform a projection mapping it is necessary to perform two calibrations, the final reprojection error will be increased in relation to the isolated error of each calibration.

VIII. Conclusion

This work presented a framework capable of reading a calibration performed by a Kinect-Projector system and validating it. This validation is done as different kinds of visualization of data originated in Kinect. Developers will be able to check their calibration methods through this framework, in addition to study the effects of the visualization methods presented here.

Due to the limitations of Kinect and of the calibration methods, future research can be conducted on the use of Kinect 2.0, which has a better resolution (1080p) and more resources than the Kinect 1.0, enabling it to solve various problems found here. Additionally, more efficient calibration methods can be studied, aiming to achieve better and faster visualizations in Mapping Projection.

Considering that Kinect 2.0 does not have a partnership with Primesense, it would be prudent to use a Microsoft driver instead of the OpenNI driver in order to reassure that the framework works with both Kinect versions, or that the required maintenance for both versions is as low as possible.

The calibration patterns currently used impose a restriction to this work, both for its hard maneuvering and for the difficulty of finding a light and rigid surface in order to avoid folds and distortions to the pattern. It is necessary to search smaller patterns and more efficient algorithms, capable of detecting them.

I. Acknowledgments

The authors would like to acknowledge the support of Tecgraf/PUC-Rio to this project. Luciano Soares was financed by CNPq, process 483195/2011-1.

II. References

[1] R. T. Azuma, "A Survey of Augmented Reality," Presence - Teleoperators and l Environments, vol. 6, pp. 355-385, 1997.
[2] Pinto, F., Buaes, A., Francio, D., Binotto, A. P. D., & Santos, P. (2008, August). BraTrack: a low-cost marker-based optical stereo tracking system. In SIGGRAPH Posters (p. 131).
[3] ART – AdvancedRealtimeTracking - http://www.ar-tracking.com/home/ - Last visited at 08/12/2014
[4] Nova3D – Projeção Mapeada - http://www.nova3d.com.br/#!projecao-mapeada.html - Last visited at 08/12/2014
[5] Bot&Dolly - http://thecreatorsproject.vice.com/show/projection-mapping-and-robots-combine-in-bot--dollys-new-film - Last visited at 08/12/2014
[6] Bimber, O., Raskar, R., &Inami, M. (2005). Spatial augmented reality. Wellesley: AK Peters.
[7] UAU Mídia Interativa - http://www.uaugrupo.com.br/site/ - Last visited at 08/12/2014
[8] LPMT - http://hv-a.com/lpmt/ - Last visited at 08/12/2014
[9] CarProjection - http://carprojection.com/ - Last visited at 08/12/2014
[10] A Dandy Punk - http://www.adandypunk.com/#!viddywell/c1t44 - Last visited at 08/12/2014
[11] ARPool - http://arpool.ca/ - Last visited at 08/12/2014
[12] Ziola, R., Grampurohit, S., Nate, L., Fogarty, J., Harrison, B. OASIS: Creating Smart Objects with Dynamic Digital Behavior,Workshop at IUI 2011.
[13] Kondo, D., Goto, T., Kouno, M., Kijima, R., & Takahashi, Y. (2004). A virtual anatomical torso for medical education using free form image projection. InProceedings of 10th International Conference on Virtual Systems and MultiMedia (VSMM2004) (pp. 678-685).
[14] RGBDemo - https://github.com/rgbdemo - Last visited at 08/12/2014
[15] Audet, S. (2012). Markerless Interactive Augmented Reality on Moving Planar Surfaces with Video Projection and a Color Camera (Phd. Tokyo Institute of Technology).
[16] Jones, B., Sodhi, R. (2010). OpenLight - Kinect-Projector Calibration.
[17] CameraProjectorCalibration - https://github.com/alvarohub/Example_CameraProjectorCalibration - Last visited at 08/12/2014
[18] Raskar, R., Baar, J., Beardsley, P., Willwacher, T., Rao, S. and Forlines, C. 2003. iLamps: geometrically aware and self-configuring projectors. ACM Trans. Graph. 22, 3 (July 2003), 809-818.
[19] Chung-Sayers, N. “TEAM USES *XBOX *KINECT TO SEE PATIENT IMAGES DURING SURGERY”, Sunnybrook report
[20] Interactive Healing Space - github.com/VideoAlchemy/TVToolkit/wiki Last visited at 10/12/2014
[21] Wen, R.; Nguyen, B. P.; Chng, C. B., and Chui, C. K. (2013). In situ spatial AR surgical planning using projector-Kinect system. In Proc. Fourth SoICT '13, pp. 164-171.
[22] Zhang, Z. (1998), A flexible new technique for camera calibration, Technical report, Microsoft Corporation.
[23] Tsai , R. Y. (1987), ‘A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses’, Ieee Journal Of Robotics And Automation, 323–344.
[24] Izadi, S.; Kim, D.; Hilliges, O.; Molyneaux, D.; Newcombe, R.; Kohli, P.; Shotton, J.; Hodges, S.; Freeman, D.; Davison, A.; KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera. In Proceedings of ACM Symposium on User Interface Software and Technology, Santa Barbara, CA, USA
[25] PrimeSense Technology - https://github.com/PrimeSense - Last visited at 08/12/2014
[26] OpenNI - http://structure.io/openni - Last visited at 08/12/2014
[27] Kinect for Windows SDK - http://www.microsoft.com/en-us/kinectforwindowsdev/Start.aspx - Last visited at 08/12/2014
[28] OpenKinect Project - http://openkinect.org/wiki/Main_Page - Last visited at 08/12/2014
[29] Oren Ben-Kiki, Clark Evans und Ingy döt Net. YAML Ain’t Markup Language (YAML). 2009. url: http://yaml.org/spec/1.2/spec.pdf.
[30] OpenFrameworks - http://www.openframeworks.cc/ - Last visited at 08/12/2014
[31] Kinect for PC and Skype means KinectCam - http://codingbytodesign.net/2013/03/17/kinect-for-pc-and-skype-means-kinectcam/ - Last visited at 08/12/2014
[32] Fiedler, D., & Müller, H. (2012, November). Impact of thermal and environmental conditions on the kinect sensor. In Proc. Int. Workshop on Depth Image Analysis.
[33] Aldoma, A., Marton, Z. C., Tombari, F., Wohlkinger, W., Potthast, C., Zeisl, B., & Vincze, M. (2012). Point Cloud Library. IEEE Robotics & Automation Magazine, 1070(9932/12).
[34] OpenCV, L. (& Adrian Kaebler-O’Reilly.
[35] Cámara Lúcida. R+D - http://www.camara-lucida.com.ar/ - Last visited at 08/12/2014