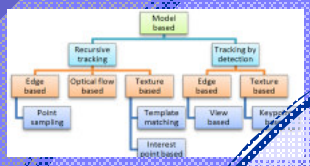
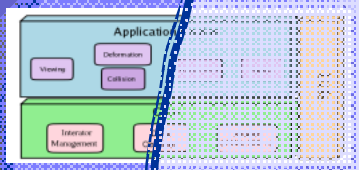


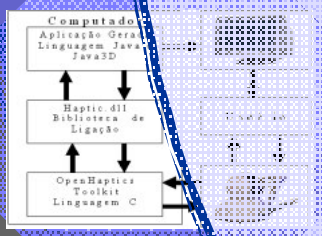


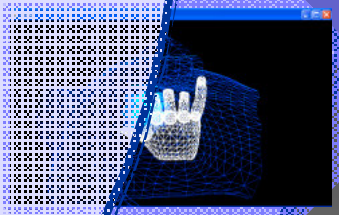
ABORDAGENS PRÁTICAS DE REALIDADE VIRTUAL E AUMENTADA

XI SYMPOSIUM ON VIRTUAL AND AUGMENTED REALITY

SVR2009

LIVRO DOS MINICURSOS





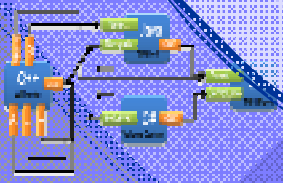
Organizadores

Fátima de Lourdes dos Santos Nunes

Liliane dos Santos Machado

Márcio Sarroglia Pinho

Claudio Kirner



XI SYMPOSIUM ON VIRTUAL AND AUGMENTED REALITY

25 a 28 de Maio de 2009

Porto Alegre - RS

ABORDAGENS PRÁTICAS DE REALIDADE VIRTUAL E AUMENTADA

LIVRO DOS MINICURSOS

Editora
Sociedade Brasileira de Computação – SBC

Organizadores
Fátima de Lourdes dos Santos Nunes
Liliane dos Santos Machado
Márcio Sarroglia Pinho
Claudio Kirner

Realização
Pontifícia Universidade Católica – PUCRS
Faculdade de Informática

Promoção
Sociedade Brasileira de Computação – SBC

Organizadores do SVR2009

Patrocinadores

Produção e Editoração
Fátima L. S. Nunes – EACH/USP
Liliane S. Machado - UFPB

Projeto Gráfico
Fátima L. S. Nunes – EACH/USP
Liliane S. Machado – UFPB

Dados Internacionais de Catalogação na Publicação (CIP)

S989a Symposium on Virtual and Augmented Reality
(11. 2009 : Porto Alegre, RS)
Abordagens práticas de realidade virtual e aumentada : livro dos minicursos [recurso eletrônico] / SVR 2009 ; org. Fátima de Lourdes dos Santos Nunes ... [et al.]. – Porto Alegre : SBC, 2009.

Sistema requerido: Adobe Abrobat Reader.

Obra bilíngüe: português, inglês

1. Informática. 2. Realidade de Virtual. I. Nunes, Fátima Lourdes dos Santos. II. Título.

Ficha Catalográfica elaborada pelo

Setor de Tratamento da Informação BC-PUCRS

Este livro foi especialmente editado, com tiragem limitada,
a partir de conteúdos desenvolvidos para os minicursos do
XI Symposium on Virtual and Augmented Reality,
realizado em Porto Alegre – Rio Grande do Sul,
de 25 a 28 de maio de 2009,
promovido pela Sociedade Brasileira de Computação.

Porto Alegre – RS
2009

Sumário

Apresentação

O XI Symposium on Virtual and Augmented Reality (SVR2009), tradicionalmente patrocinado pela Sociedade Brasileira de Computação (SBC), é um forum científico internacional com o objetivo de promover a troca de experiência e conhecimento entre pesquisadores, profissionais, estudantes e desenvolvedores relacionados com a pesquisa e desenvolvimento de ambientes virtuais e sistemas de realidade virtual, aumentada e misturada.

Os minicursos constituem atividades do SVR2009 com o objetivo de oferecer aos participantes conceitos e práticas relacionados aos tópicos abordados no evento. Nesta edição da conferência houve uma preocupação com o oferecimento de abordagens que privilegiam a prática, de forma a fornecer subsídios para o desenvolvimento de sistemas avançados em relação à interação. O conteúdo aqui apresentado reflete a experiência dos autores em pesquisa e desenvolvimento, com detalhes de implementação, visando a contribuir com aspectos dificilmente abordados na literatura científica da área.

Assim, esperamos que o conteúdo ofereça ao leitor essa abordagem prática almejada!

Aproveitem!

Os editores

Fátima de Lourdes dos Santos Nunes
Liliane dos Santos Machado
Márcio Sarroglia Pinho
Claudio Kirner

Presentation

The XI SYMPOSIUM ON VIRTUAL AND AUGMENTED REALITY, SVR 2009, traditionally sponsored by the Brazilian Computer Society (SBC), is an international scientific forum for the exchange of experience and knowledge among researchers, professionals, students and developers involved with research and development of virtual environments, virtual, augmented and mixed reality systems.

The minicourses are activities of the SVR2009 aiming at offering to participants concepts and practical sessions related to subjects of the the conference. In this edition there has been a special care for offering new approaches that point out the practice, providing subsidies for the development of systems with advanced interaction characteristics. The content presented here reflects the experience of the authors in research and development, with details of coding details, aiming at contributing with aspects that rarely are considered in the scientific literature of the area.

Thus, we expect that the content offers this practical approaching to the reader!

Enjoy it!

The editors

Fátima de Lourdes dos Santos Nunes
Liliane dos Santos Machado
Márcio Sarroglia Pinho
Claudio Kirner

Capítulo

1

Online Monocular Markerless 3D Tracking for Augmented Reality

João Paulo Lima, Francisco Simões, Lucas Figueiredo, Veronica Teichrieb and Judith Kelner

Abstract

This chapter surveys the area of markerless 3D tracking, specifically online and monocular, targeting Augmented Reality. Mathematical tools that are recurring in the development of markerless 3D tracking methods are presented. A taxonomy of markerless Augmented Reality techniques is described, explaining their main concepts. Outstanding techniques are detailed, with focus on model based methods, addressing operation, pros and cons. The model based markerless tracking categories are also evaluated taking into account different metrics, such as performance, processing load, accuracy and robustness.

1.1.Introduction

The subject of this chapter is developing Augmented Reality (AR) systems using online monocular markerless 3D tracking. Tracking is required in order to correctly positioning virtual information relative to the real environment.

In the following subsections, some definitions regarding AR and 3D tracking are presented, as well as the motivation for using online monocular markerless 3D tracking. Finally, the structure of this chapter is given.

1.1.1.Definition

AR systems support the coexistence of real elements (that are part of users' world) and synthetic ones (computer generated) in the same environment [Haller et al. 2007]. Nowadays, this kind of user interface has obtained more attention due to the fact that it allows users performing tasks in a more intuitive, efficient and effective way. Interactive AR interfaces may augment users' perception of the real world by adding

virtual information to it. In addition, users' interaction with the system may be augmented, making the computer as seamlessly as possible, by exploring the use of real objects for interaction with the application. Therefore, they may augment actions that users are capable to perform in the real world, both in quantity of tasks performed and their quality [Trevisan et al. 2002].

AR interfaces superimpose virtual information – 2D or 3D, textual or pictorial – onto real world scenes in real-time, registered in 3D, and allow users interaction with real and virtual elements simultaneously. In this kind of interface the real environment takes part of the application context. Figure 1.1 illustrates an AR system that verifies a mobile user actual coordinates in the real world, and permits him/her visualize information regarding visible places located in his/her neighborhood (for example, stores, restaurants, etc.) [Bonsor 2009]. This information is updated in real-time as the user moves in the real world. The system uses a Global Positioning System (GPS) to track user position, and he/she visualizes the real scene and the virtual information wearing a Head Mounted Display (HMD). An important aspect to be observed is that the synthetic elements are correctly registered in 3D with the real scene.

Figure 1.1. An example of an outdoor AR system [Bonsor 2009].

In AR the technical challenges lie in determining, in real-time, what should be shown where, and how. The latter problem is especially important when the visual appeal of the result is crucial. Then substantial effort must go into seamlessly fitting the information into the scene, according to the objectives of the system [Ferwerda 2003]. Ideally, AR proposes that the user must not be able to distinguish between real and virtual information, demanding that the virtual elements show both geometric (correct placement, correct size, occlusions identification) and photometric (shadowing, mutual reflections, chromatic adaptation to scene illumination) consistency. Even under simplified conditions these problems cannot be trivially solved.

The problem related to correctly positioning virtual information relative to the real environment, called registration, is solved by tracking the environment so that the synthetic elements can be adequately registered with the real scene. There are diverse

tracking technologies available, such as optical sensors, movement sensors, thermal imaging, ultrasound, magnetic sensors, GPSs, among others [Azuma 1997] [Azuma et al. 2001]. They capture features from the real world, and based on this information the AR system determines when, where and how the virtual scene should be exhibited.

Optical tracking is often used for this purpose due to cost, accuracy and robustness requirements. Two types of optical tracking can be cited: marker based and markerless. Marker based tracking is a more well established approach for registration. It makes use of known artificial patterns placed along the environment in order to perform camera pose estimation. On the other hand, markerless tracking differs from the former one by the method used to place virtual objects in the real scene. In markerless AR any part of the real environment may be used as a marker, since the system exploits natural features present in the real scene to perform tracking. Markerless AR has received more attention from researchers in the latest years, and presents important challenges to be overcome. This chapter describes in detail how online monocular markerless 3D tracking works in the context of AR systems.

1.1.2.Motivation

As mentioned above, markerless AR systems use natural features instead of fiducial markers in order to perform tracking. Therefore, there are no ambient intrusive markers that are not really part of the world. Furthermore, markerless AR counts on specialized and robust trackers. Another advantage is the possibility of extracting from the surroundings characteristic information that may later be used by the AR system for other purposes.

In this chapter, we address an online monocular markerless AR approach. Optical tracking presents some advantages when compared to its counterparts, such as higher precision and less sensibility to interference. Besides that, the use of a single camera allows lower cost and more compact systems. Calibration issues are also easier to be managed [Azuma et al. 2001].

Nonetheless, it is important to mention that tracking and registration techniques are more complex in markerless AR systems.

In several AR application scenarios, markerless tracking is mandatory or at least desirable [Lima et al. 2008]. An example of such application is an AR system for equipment maintenance [Platonov et al. 2006]. Using markers for tracking the equipment presents many disadvantages: tracking failures can occur due to occlusion of markers by the user's body and tools; the markers can hide important parts of the equipment; the equipment pose has to be calibrated with each marker present at the scene. Therefore, a markerless tracking approach is strongly advised in such scenario.

1.1.3.Outline

This document is structured for introducing the concepts related to online monocular markerless AR and describing the main model based 3D tracking techniques. In this section, the definition of online monocular markerless AR was presented and the motivation for using this technology was introduced. The remainder of this document is organized as follows.

Section 1.2, Mathematical Background, describes the main concepts of camera representation and robust pose estimation, required for performing markerless 3D tracking for AR, and more specifically model based tracking. Markerless AR systems major characteristics and the classification of techniques developed for online monocular markerless 3D tracking for AR will be presented by this document in Section 1.3, Markerless Augmented Reality. Diverse model based techniques, using recursive tracking and/or tracking by detection approaches, will be described in the section Model Based Techniques Description. In Section 1.5, Model Based Techniques Evaluation, some metrics are defined and used to analyze the model based techniques described in the previous section. Finally, the section Final Considerations draws some final considerations about the technology presented here, based on authors' experience in developing and using markerless AR systems.

1.2.Mathematical Background

Some basic concepts regarding camera representation and robust pose estimation for markerless 3D tracking are presented next. These topics provide the foundation for developing the techniques described in Sections 1.3 and 1.4.

1.2.1.Camera Representation

Camera tracking, which is a fundamental aspect in tracking and register phases, comes from recovering information that correctly describes a virtual camera used to position virtual objects in the real scene and to render these objects in the image. There are many models for projecting 3D objects onto 2D images, varying between simple pinhole (perspective) camera models to complex lenses models that simulate human eyes [Forsyth and Ponce 2002]. In this work, it was considered the pinhole camera model without distortion factors (lenses), which is a well known simple model that correctly approximates a virtual camera in terms of geometry.

In all camera models, virtual objects are defined in a general coordinate system, also called world coordinate system (w_x, w_y, w_z), in a way to have a generic description that does not depend on the camera system used (c_x, c_y, c_z). The camera system corresponds to the world coordinate system after applying a rotation and translation transform and, because of that, it is necessary to get object coordinates from the world coordinate system to the camera coordinate system before projecting it onto the image plane (see Figure 1.1). This affine transform is described by the composition of the rotation $R_{3\times3}$ and translation $t_{3\times1}$ matrices, resulting in a $[R|t]_{3\times4}$ matrix. When applied to the homogeneous coordinates of the 3D point, the composed matrix leads to the same 3D point in the camera coordinate system. This matrix is called extrinsic parameters matrix because of its relation with the virtual camera model movement.

It is also important to observe that, for other purposes like pose estimation, there are many ways to represent the rotation transform. One of them is the axes-angle representation, which corresponds to a vector representing a fixed rotation axis $(w_x, w_y, w_z)^T$, and its norm refering to a rotation angle θ. This representation has an one-to-one correspondence to the $R_{3\times3}$ form by using the Rodrigues and inverse Rodrigues formula [Brockett 1984].

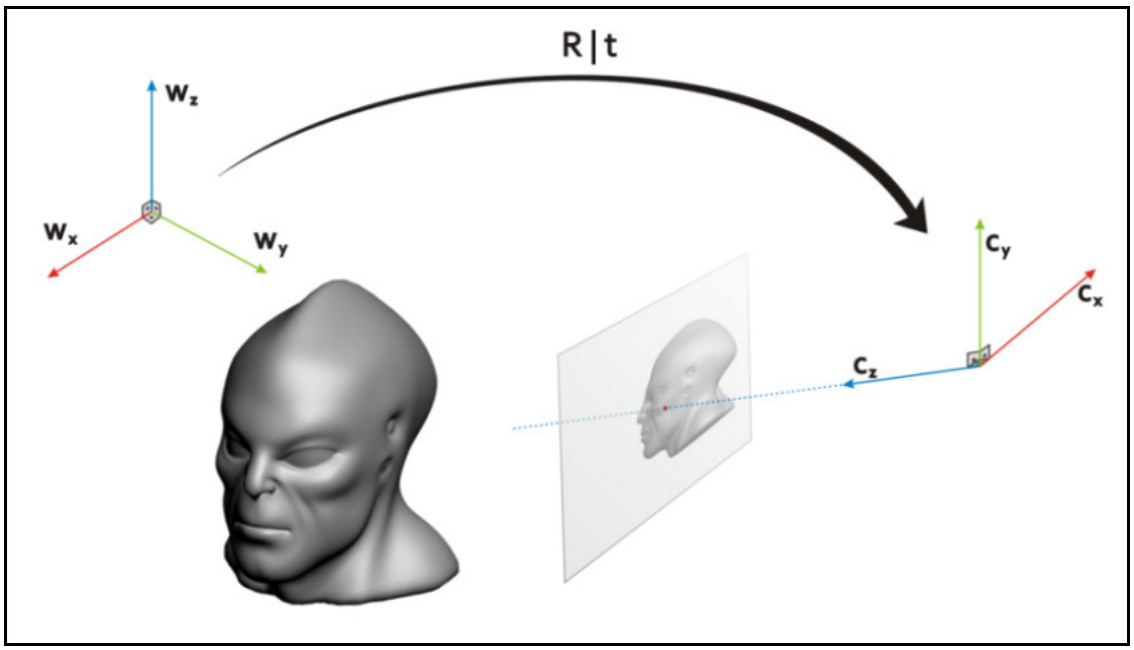


Figure 1.1. The 3D object, its projection onto the image plane and the relation between world (W_x, W_y, W_z) and camera (C_x, C_y, C_z) coordinate systems.

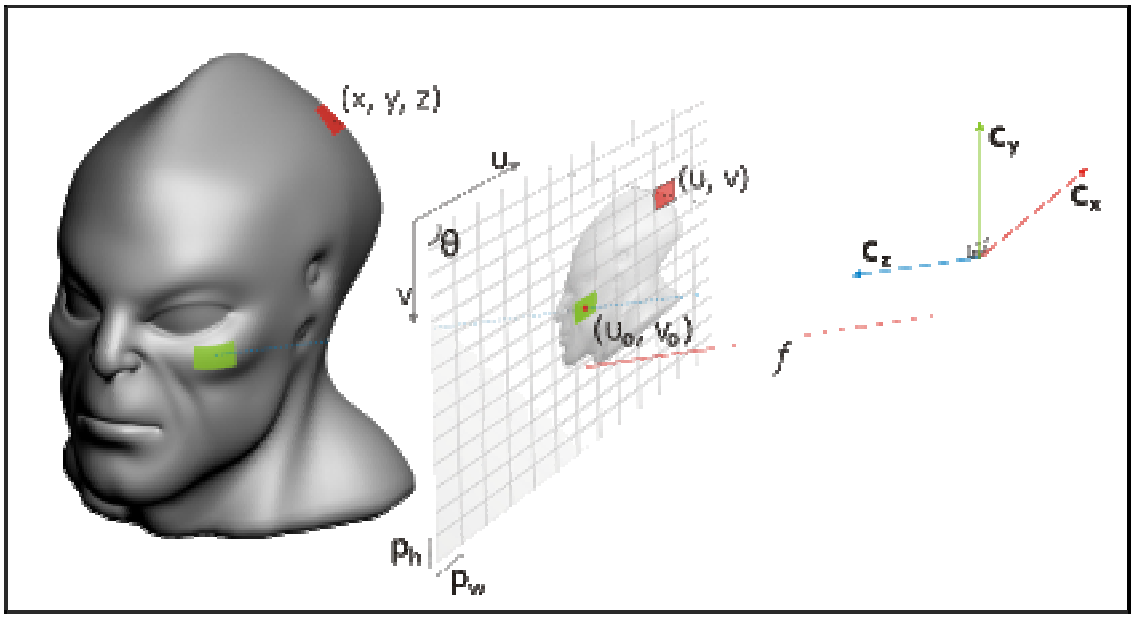


Figure 1.2. Perspective transform from 3D points to image points.

In the pinhole camera model, a point in image plane $m = [u, v, f]^t$ is obtained by projecting the 3D point $M = [X, Y, Z]^t$, written in camera coordinate system, onto the image plane by obeying to perspective projection conditions (see Figure 1.2). By similarity of triangles,

$$u = \frac{X}{Z} f + u_0 \text{ and } v = \frac{Y}{Z} f + v_0. \quad (1)$$

However, the image plane is divided in pixels units that correspond in the real world to well defined areas with dimensions written in millimeters. They are, by default, called pixel width (p_w) and pixel height (p_h)[1]. Considering that, the dimensions of u and v are not written in millimeters but in pixels, and by this the equation (1) must be rewritten as

$$u = \frac{X}{Z} \frac{f}{p_w} + u_0 \text{ and } v = \frac{Y}{Z} \frac{f}{p_h} + v_0. \quad (2)$$

By looking at the problem of projecting the 3D point again, a first version of the k transformation matrix comes up, which takes a 3D point in camera coordinates and returns its 2D image representation in homogeneous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f/p_w & 0 & u_0 \\ 0 & f/p_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix}. \quad (3)$$

If the pixels of the camera are not squared, it is added in the equation a new parameter, also called skew factor[2], that correlates the θ angle between u and v dimensions with its 3D point, turning the affine transformation (3) into:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f/p_w & -cot(\theta)/p_w & u_0 \\ 0 & f * cosec(\theta)/p_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X/Z \\ Y/Z \\ 1 \end{bmatrix}. \quad (4)$$

The final k matrix presented in equation (4) is called intrinsic parameters or calibration matrix because of its dependence on the real camera used to display the scene. By combining the intrinsic and extrinsic parameters matrices we have the camera projection matrix P that is responsible for getting 3D points from the world coordinate system and projecting them onto the camera image plane[3]:

[1] In most of real camera specifications, p_w and p_h parameters are not given. Instead, their relationship is provided, known as aspect ratio (a_r), where $a_r = p_w/p_h$.

[2] Since θ is generally near to 90°, the skew factor sk is generally only referenced as $-cot(\theta)$ and the influence of the $cosec(\theta)$ term is discarded.

[3] In order to finish the transformation from 3D to 2D points, it is also necessary to normalize the answer in terms of the scale factor s: $[su, sv, s]^T$ to $[u, v, 1]^T$.

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \underbrace{\underbrace{\begin{bmatrix} f/p_w & -\cot(\theta)/p_w & u_0 \\ 0 & f * \operatorname{cosec}(\theta)/p_h & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{K} * \underbrace{\begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{R|t}}_{P=K[R|t]} * \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \quad (5)$$

1.2.2.Pose Estimation

In order to estimate camera extrinsic parameters for a given frame, some correspondences between 2D points from the image and 3D points from the model are needed. In the following subsections, two classes of methods for pose estimation are described: Pespective-n-Point (PnP) and minimization of reprojection error.

1.2.2.1.Perspective-n-Point

PnP is basically the problem of estimating the camera pose $[R|t]$ given n 2D-3D correspondences. The first intuitive approach for solving this problem is to apply the equation $PM_i = sm_i$ to each correspondence i and then solve a linear system. This method is called Direct Linear Transformation (DLT) [Faugeras 1993] and can estimate all parameters of P (even if the intrinsic ones are not known). However, when using DLT to calculate P, in most cases, the number of correspondences must be higher than 15, which is more than the necessary when applying other methods and for some techniques is not an acceptable number. Furthermore, the DLT method minimizes an algebraic error, but for the pose estimation problem it is preferable to minimize a geometric error.

In many AR applications the intrinsic parameters do not change during the frame sequence since the same camera configuration is used the whole time. So it is preferable to obtain them separately, reducing in a considerable way the number of correspondences needed to estimate the current pose and probably also the estimation error. Encouraged by this context, the PnP problem explicitly uses the intrinsic parameters, which must be previously obtained, and estimates only the extrinsic parameters.

This way, when trying to solve the P3P problem, four solutions are reached. This means that it is not possible to find out a unique solution having only 3 correspondences. An approach to find the correct pose is adding a correspondence and solving the P3P problem for each subset of 3 correspondences; then, a common pose will emerge from the results. Solving P4P and P5P problems usually reaches a unique solution, unless the correspondences are aligned. For $n \geq 6$ the solution is almost always unique.

Several solutions have been proposed for the PnP problem in the Computer Vision and AR communities. In general they attempt to represent the n 3D points in camera coordinates trying to find their depths (which is the distance between the camera optical center C and the point M_i). In most cases this is done using the constraints given by the triangles formed from the 3D points and C. Then $[R|t]$ is retrieved by the Euclidean motion (that is an affine transformation whose linear part is an orthogonal transformation) that aligns the coordinates. [Lu et al. 2000] proposed an iterative, accurate and fast solution that minimizes an error based on collinearity in the object space. Later, EPnP [Moreno-Noguer et al. 2007] solution showed a $O(n)$ method for PnP if $n \geq 4$. It represents all points as a weighted sum of four virtual control points. Then the problem is reduced to estimate these control points in the camera coordinate system.

1.2.2.2.Minimization of Reprojection Error

In despite of being able to estimate the pose based solely on the 2D-3D correspondences, PnP methods are sensitive to noise in the measurements, resulting in loss of accuracy. In this scenario, a more adequate approach for calculating the pose is by minimization of the reprojection error. This consists in a non-linear least squares minimization defined by the following equation:

$$[R|t] = \underset{[R|t]}{arg\,min} \sum_{i=0}^{n} dist^2(Q(P, M_i), m_i), \quad (6)$$

where: M_i and m_i are correspondent 3D and 2D points in homogeneous coordinates, respectively; Q is the projection function, which takes as arguments the projection matrix P and the 3D point M_i and returns the 2D projected point; $dist$ is the Euclidean distance function between 2D points, which is called residual; and $[R|t]$ are the extrinsic parameters to be estimated.

Due to the fact that the Q function is non-linear, there is not a closed form solution to equation (6). In this case, an optimization method should be used, such as Gauss-Newton or Levenberg-Marquardt [Triggs et al. 2000]. These methods iteratively refine an estimate of the pose until an optimal result is obtained. The pose increment between consecutive iterations is calculated using the Jacobian matrix of Q. This matrix can be calculated analytically or using differentiation. A requirement for such kind of iterative method is a good initial estimate. Since the difference between consecutive poses is often small, the pose calculated for the previous frame can be used as an estimate for the current frame.

1.2.3.Robust Estimation

When calculating the pose, few spurious 2D-3D correspondences (named outliers) can ruin estimation even when there are many correct correspondences (named inliers).

There are two common methods to decrease the influence of these outliers: RANdom SAmple Concensus (RANSAC) [Fischler and Bolles 1981] and M-estimators [Lepetit and Fua 2005]. They are described next.

1.2.3.1.Random Sample Consensus

The RANSAC method is an iterative algorithm that tries to obtain the best pose using a sequence of random small samples of 2D-3D correspondences. The idea is that the probability of having an outlier in a small sample is much lower than when the entire correspondence set is considered.

The algorithm receives basically 4 inputs:

1.A set D of 2D-3D correspondences;

2.A sample size n, which is a small value (e.g. 6);

3.A threshold t, used to classify the correspondences as inliers or outliers. It consists in the maximum value allowed to the return of the $dist$ function from equation (6). A commonly used value for t is 2.0.

4.A probability P of finding a set that generates a good pose. This probability is utilized for calculating the iteration count of the algorithm. This value is usually set to 95% or 99%.

RANSAC works in the following way: initially, it is determined a number m of iterations to be executed by the algorithm, e.g. 500. The number of iterations can be decreased during algorithm execution, depending on how good is the pose by that time.

After this, algorithm execution begins. From the D set provided, n correspondences are randomly chosen. From this sample, a pose is calculated using any of the methods presented in Subsection 1.2.2. Next, the other correspondences that were not included in the sample are utilized to verify how good the found pose is. In order to do this, the $dist$ function from equation (6) is applied to the correspondence. If the distance is lower than the t threshold, the correspondence is an inlier. Otherwise, it is an outlier. After all the correspondences are tested, it is verified the percentage w of the correspondences in D that were tagged as inliers. If the current value of w is bigger than any previously obtained percentage, the calculated pose is stored, since it is the most refined by that time.

When a refined pose is found, the algorithm tries to decrease the number of iterations m needed. The idea behind this calculation is very straightforward. Since the n correspondences are sampled independently, the probability that all n correspondences are inliers is w^n. Then, the probability that there is any outlier

correspondence is $1 - w^n$. The probability that all the m samples contain an outlier is $(1 - w^n)^m$ and this should be equal to $1 - \Gamma$, resulting in:

$$1 - \Gamma = (1 - w^n)^m. \tag{7}$$

After taking the logarithm of both sides, the following equation can be obtained:

$$m = \frac{\log(1 - \Gamma)}{\log(1 - w^n)}. \tag{8}$$

1.2.3.2.M-Estimators

This method is often used together with minimization of reprojection error in order to decrease the influence of outliers. M-estimators apply a function to the residuals that has a Gaussian behavior for small values and a linear or flat behavior for higher values. This way, only the residuals that are lower than a c threshold have an impact on the minimization. A modified version of equation (6) is then used:

$$[R|t] = \underset{[R|t]}{arg\,min} \sum_{i=0}^{n} \rho(dist(Q(P, M_i), m_i)), \tag{9}$$

where ρ is the M-estimator function. Two of the most used M-estimators are Huber and Tukey [Lepetit and Fua 2005]. The Huber M-estimator is defined by:

$$\rho_{Hub}(x) = \begin{cases} \frac{x^2}{2}, & |x| \leq c \\ c\left(|x| - \frac{c}{2}\right), & |x| > c \end{cases}, \tag{10}$$

where c is a threshold that depends on the standard deviation of the estimation error.

The Tukey M-estimator can be computed using the following function:

$$\rho_{Tuk}(x) = \begin{cases} \frac{c^2}{6}\left[1 - \left(1 - \left(\frac{x}{c}\right)^2\right)^3\right], & |x| \leq c \\ \frac{c^2}{6}, & |x| > c \end{cases}. \tag{11}$$

The graphics of the Huber and Tukey M-estimator functions, which can be seen in Figure 1.3, highlight how the residuals are weighted according to their magnitude.

1.3.Markerless Augmented Reality

Markerless AR systems integrate virtual objects into a 3D real environment in real-time, enhancing user's perception of, and interaction with, the real world. Its basic difference from marker based AR systems is the method used to place virtual objects in the user's view. The markerless approach is not based on the use of traditional artificial markers, which are placed in the real world to support position and orientation tracking by the system. In markerless AR natural features present in the real scene, such as edges, optical flow and textures, may be used as a marker that can be tracked in order to place virtual objects.

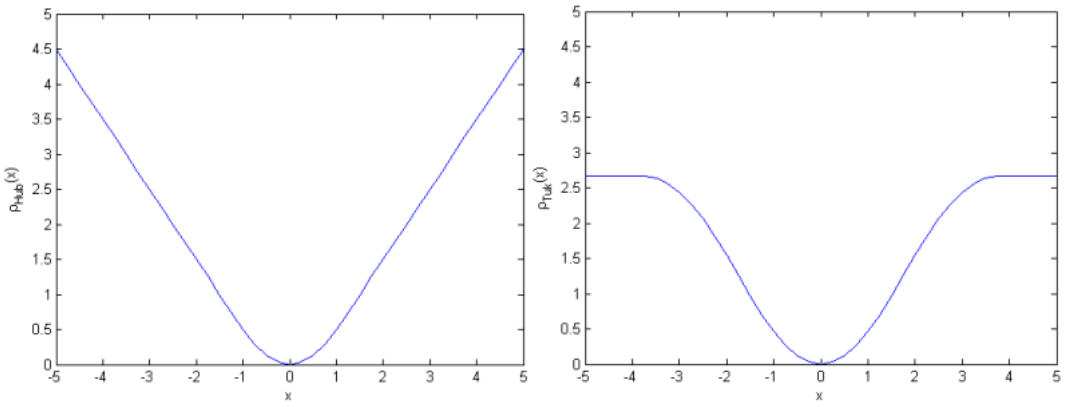


Figure 1.3. Huber M-estimator function with c=1 (left) and Tukey M-estimator with c=4 (right).

Techniques developed for online monocular markerless AR can be classified in two major types: model based and Structure from Motion (SfM) based, as described in [Teichrieb et al. 2007]. With model based techniques, knowledge about the real world is obtained before tracking occurs and is stored in a 3D model that is used for estimating camera pose. In SfM based approaches, camera movement throughout the frames is estimated without any previous knowledge about the scene, being acquired during tracking [Gomes Neto et al. 2008]. This chapter focuses model based techniques, since they are more commonly used in the markerless AR context.

Considering their tracking nature, model based techniques can be classified in two categories (Figure 1.4): recursive tracking, where the previous pose is utilized as an estimate to calculate the current pose [Wuest et al. 2005] [Vacchetti et al. 2004a] [Basu et al. 1996] [Jurie and Dhome 2001]; and tracking by detection, where it is possible to calculate the pose without any previous estimate, allowing automatic initialization and recovery from failures [Skrypnyk and Lowe 2004] [Wiedemann et al. 2008].

By taking into account the type of feature used for tracking, model based techniques can also be classified in three other categories: edge based, where camera pose is estimated by matching a wireframe 3D model of an object with the real world image edge information [Wuest et al. 2005] [Wiedemann et al. 2008]; optical flow based, which exploits temporal information extracted from the relative movement of the object projection onto the image in order to track it [Basu et al. 1996]; and texture based, which takes into account texture information presented in images for tracking [Vacchetti et al. 2004a] [Skrypnyk and Lowe 2004] [Jurie and Dhome 2001].

The edge based recursive tracking category comprises point sampling methods, which sample some control points along the edges of the wireframe 3D model and compare their projections with strong gradients present in the image [Wuest et al. 2005]. Texture based recursive techniques are classified in two subcategories: template matching, which applies a distortion model to a reference image to recover rigid object movement [Jurie and Dhome 2001]; and interest point based, which takes into account localized features in the camera pose estimation [Vacchetti et al. 2004a].

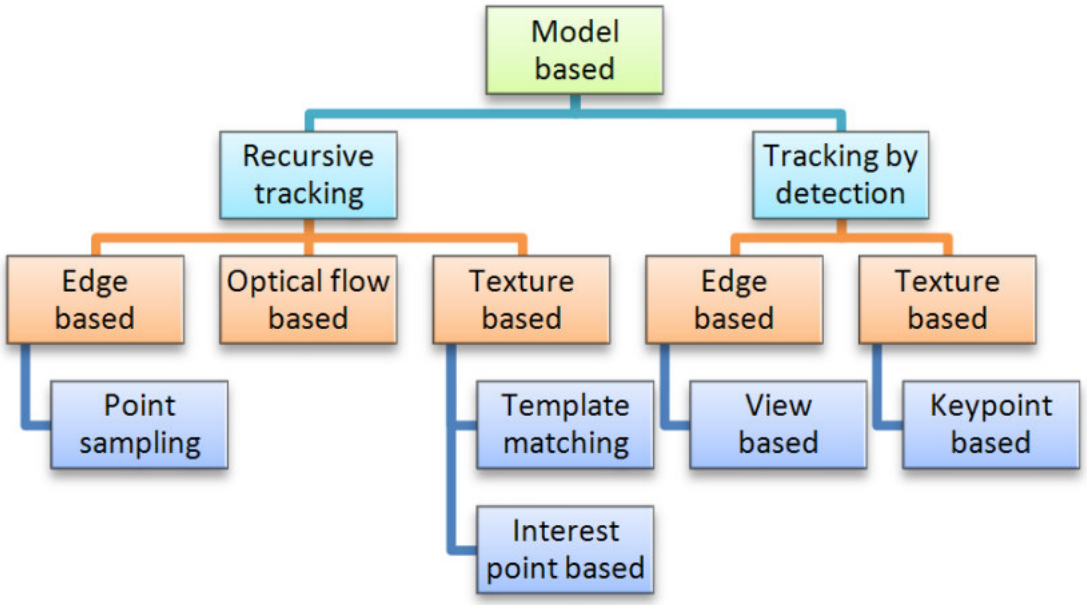


Figure 1.4. Model based online monocular markerless AR taxonomy.

Edge based tracking by detection techniques are called view based, since the current frame is matched with 2D views of the target object previously obtained from different positions and orientations [Wiedemann et al. 2008]. Texture based tracking by detection methods are named keypoint based [Skrypnyk and Lowe 2004]. Keypoints are features invariant to scale, viewpoint and illumination changes. They are extracted from the object image at every frame, providing 2D-3D correspondences needed for pose estimation.

1.4.Model Based Techniques Description

Some techniques currently used in markerless AR systems, with different tracking natures and that exploit different features of the real objects for tracking purposes are detailed next.

1.4.1.Recursive Tracking

The following subsections describe techniques that require an estimate of the current pose in order to perform tracking. They can be based on features such as edges, optical flow or textures.

1.4.1.1.Edge Based

In the markerless AR scenario, edge based techniques were the first to be developed, due to its high computational efficiency and relatively simplicity of implementation. The name comes from the type of features used to represent the model and to carry out the tracking, which occurs using edges. These techniques are most applied in scenarios containing polygonal objects. Although, they can also be used in scenes with other kinds of objects if they meet the restriction of strong gradient presence on its contours. Besides these applications, edge based techniques can deal, in general, with problems hardly manageable when using other techniques such as fast lighting changes because the edges are usually preserved, even when tracking specular objects.

1.4.1.1.1.Point Sampling

Point sampling techniques have, as a key feature, the tracking of control points, sampled along the edges of the tracked object. From these points, it is made a relationship between the 2D points extracted from each frame and their corresponding 3D edge in order to estimate camera pose. Due to its low computational complexity, these techniques were the first model based techniques to achieve concrete results in real-time.

The general pipeline for point sampling techniques is very simple, and from that there are many improvements to turn it more robust and faster [Lepetit and Fua 2005]. At first, in order to increase the computational efficiency, these techniques use a routine to discard model's edges that are not visible by using the previous pose to project them. This is done to reduce the number of points to be matched and avoid wrong correspondences, since the previous pose is a well known good approximation to the current pose. After discarding the invisible edges, which can be done using a Graphics Processing Unit (GPU) based approach to improve speed [Wuest et al. 2005], it is necessary to match every sampled point against the image. The match is generally done by a linear search in the line that is perpendicular to the projection of the 3D edge and passes through the projection of the point that is looking for the match. There are many approaches to decide that an image point is or is not a match. [Drummond and Cipolla 2002] use the match directly against contour points found in the search line. [Wuest et al. 2005] use an adaptive approach that takes as a match more than one high gradient point to be used in the pose estimation step (see Figure 1.5). In [Wuest et al. 2005] the search is also more robust because of the use of a Gaussian mask to do the match around the search line (see Figure 1.6).

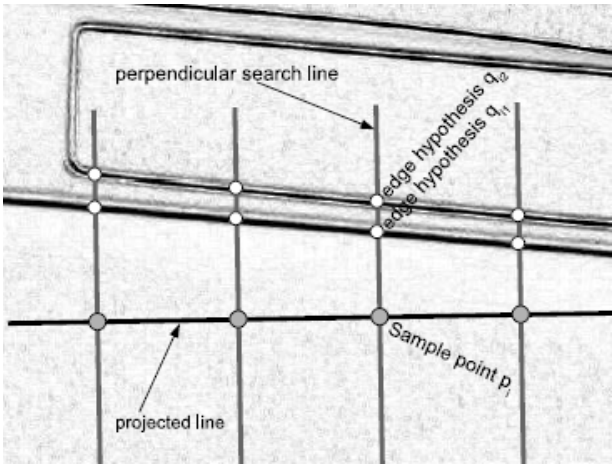


Figure 1.5. More than one hypothesis is stored as possible match [Wuest et al. 2005].

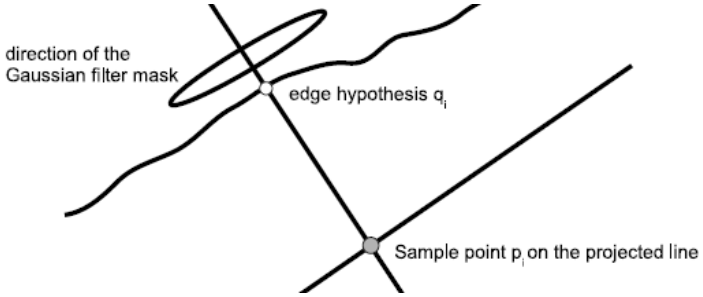


Figure 1.6. Gaussian mask parallel to projected edge to improve robustness of the matching process [Wuest et al. 2005].

After the matching phase, the system is able to estimate the camera pose based on an optimization process. In [Drummond and Cipolla 2002] it is used an iterative reweighted least squares (IRLS) to solve the problem in terms of rigid camera motion to be applied to the projection matrix estimated in the last frame. Basically, the system estimates the rigid motion of the camera between two consecutive frames by minimizing the sum of perpendicular distances between the projection of the 3D edges using the last frame pose and the matched contour points extracted from the image. By using enough control points, the system can use a least squares approach to find the six-dimension parameters (three for rotation and three for translation) that fit well the observed motion of the features in adjacent frames. In this formulation, the IRLS approach is then used instead of the naïve least squares approach because of the using of a weighting function that is applied to all measurements at each iteration, minimizing outliers influence.

In [Wuest et al. 2005] the non-linear estimation problem is solved by a minimization approach similar to the one described in equation (6), but taking into account multiple hypothesis also obtained in the matching phase to increase tracking robustness. The estimation algorithm chooses to use as match at each iteration step the hypothesis that is closer to the 3D edge projection with the undergoing estimated pose:

$$[R|t] = \underset{[R|t]}{arg\,min} \sum_{i=0}^{n} \Delta(p_i, m_{ij}), \text{ with} \tag{12}$$

$$\Delta(p_i, m_i) = |(m_i \quad p_i) \cdot (n_i)|, \tag{13}$$

where m_{ij} is the j-th hypothesis that minimizes Δ, which is the distance between the projected edge with normal n_i that contains the sampled point p_i and the matched point m_i.

As can be seen in Figure 1.7, point sampling techniques can achieve great results, tracking complex objects with a good accuracy. These techniques also achieve real-time performance even with a great occlusion area.

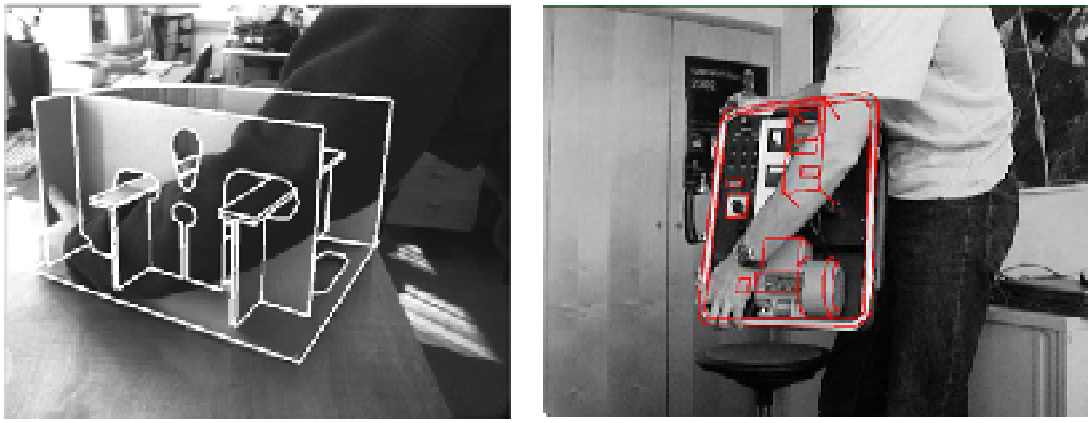


Figure 1.7. Real-time tracking of complex structures even in the presence of occlusions: [Drummond and Cipolla 2002] at left and [Wuest et al. 2005] at right.

1.4.1.2.Optical Flow Based

Differently from edge based methods, which rely on spatial information obtained by image-model matching, optical flow based tracking exploits temporal information. Such information is extracted from the relative movement of the object projection onto the image. After initialization, which is often manual, the optical flow between the frames captured at time t and t+1 is calculated. Then, the algorithm determines what points from the model projected onto the image at time t are still present in the image at time t+1. The displacement of these points over time is calculated using an algorithm such as the Kanade-Lucas (KL), described in [Lucas and Kanade 1981]. This information is used to provide the 2D-3D correspondences needed to estimate camera movement.

Due to its integration over time, 3D tracking based on optical flow presents smoother changes between consecutive poses. Another advantage is the moderate processing load needed. However, optical flow techniques tend to accumulate errors produced by sequential pose estimations, leading to a deviation from the correct camera calibration. Optical flow algorithms are also not robust against lighting changes and large camera displacements, originating errors in object tracking and requiring re-initialization. Figure 1.8 shows an optical flow based application that performs 3D face tracking [Basu et al. 1996].

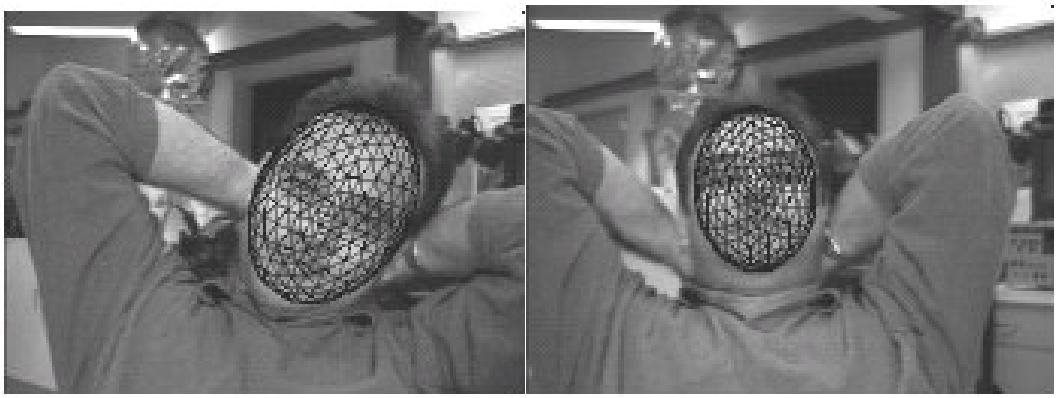


Figure 1.8. Optical flow based face tracking application [Basu et al. 1996].

In the past, edge and optical flow based methods for model based tracking were combined, since they are complementary with respect to the usage of spatial and temporal information [Haag and Nagel 1999]. More recently, [Pressigout et al. 2008] used both edges and optical flow without the need of a known motion model, which is the case of most AR applications. Texture based feature extraction and optical flow tracking were also joined together in a multithreaded manner in [Lee and Höllerer 2008]. AR results obtained using this method are illustrated in Figure 1.9.

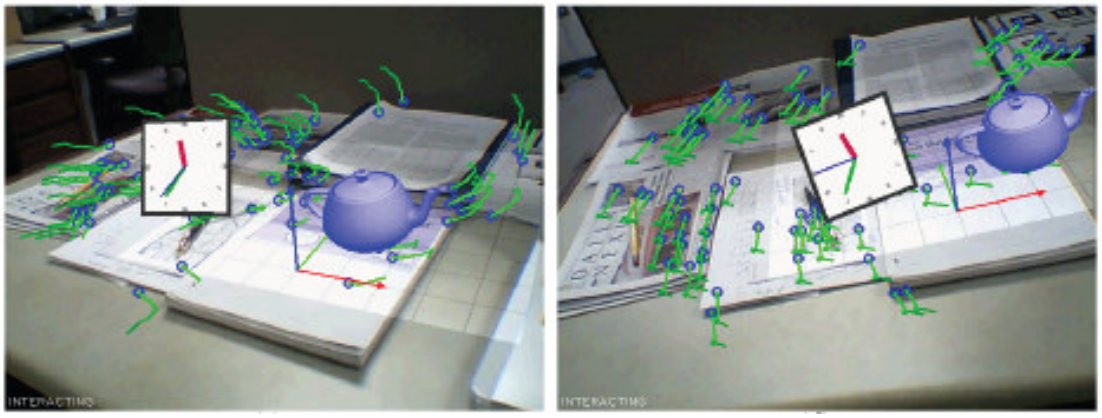


Figure 1.9. Optical flow combined with texture based tracking for AR [Lee and Höllerer 2008].

1.4.1.3.Texture Based

In the next subsections, the template matching and interest point based tracking subcategories are presented, which are both recursive methods that take texture information into account.

1.4.1.3.1.Template Matching

The template matching approach is based on global information, unlike feature based techniques. The strength of this subcategory lies in its ability to treat complex patterns that would be difficult to model by local features. They also present a low processing demand. However, there are some problems with variations in illumination, occlusions and fast camera movement. These techniques are also called sum-of-square-difference or SSD, as they consist in minimizing the difference between a region of the image and a reference template.

Basically, such techniques search for the parameters p of a function f that warps a template T into the target image I, so that tracking can be done. According to [Lucas and Kanade 1981], this is the general goal of the KL algorithm. Since it is a recursive technique, an initial estimate of the parameters p is provided. An update Δp is then calculated at each frame. In the original forward additive approach, Δp is found by:

$$\Delta p = \underset{\Delta p}{arg\,min} \sum_j [\Delta i_j]^2, \text{ with} \tag{14}$$

$$\Delta i_j = I\left(f(m_j, p + \Delta p)\right) - T(m_j), \tag{15}$$

where $m = (x, y, 0)^t$ describes the location of the template point. The parameter vector is updated by:

$$p \leftarrow p + \Delta p. \tag{16}$$

In the 3D tracking context, f is usually a homography and p contains the 9 elements of the $3x3$ homography matrix H that is multiplied by m. The 3D pose can then be retrieved in the following way:

$$R^1 R^2 t = K^{-1} H, \tag{17}$$

$$R^3 = R^1 \times R^2, \tag{18}$$

where R^i is the i-th column of the R matrix.

Some modifications have been proposed to allow the precomputation of constant values, improving the performance of the method. [Hager and Belhumeur 1998] decomposed the Jacobian matrix, decreasing the amount of online calculations. [Jurie and Dhome 2001] obtained better convergence rates by employing an offline training phase, which enables obtaining Δp as a linear function of Δi at runtime. [Baker and Matthews 2004] utilized the Inverse Compositional (IC) approach, where the roles of image and template are changed. Therefore, equation (15) is modified as follows:

$$\Delta i_j = T\left(f(m_j, \Delta p)\right) - I(f(m_j, p)). \tag{19}$$

The term IC comes from the fact that, instead of updating the parameter vector as in equation (16), the warping function is updated by compositing f with its inverse in the following way:

$$f(m_j, p) \leftarrow f(m_j, p) \circ f^{-1}(m_j, \Delta p). \tag{20}$$

This formulation is equivalent to the forward additive method described by equations (15) and (16), but allows the precomputation of several values, being more efficient at runtime.

The optimization depicted in equation (14) is commonly solved using the Gauss-Newton method (see Subsection 1.2.2.2), which is a first order minimization. [Benhimane and Malis 2004] created the Efficient Second-order Minimization (ESM), which is an adaptation of the second order Newton minimization. The Newton method theoretically has a higher convergence rate than first order ones, but demands more computation and is more constrained regarding the type of function that can be minimized. ESM tackles these problems by using an efficient approximation that allows

fast computation without compromising the convergence and allowing the minimization of a wider range of functions.

Another option to speed up the tracking is to use only a subset of the template pixels for pose calculation, which can be selected previously in an offline phase. [Dellaert and Collins 1999] proposed the Selective Pixel Integration, where the pixels to be used are randomly selected from the ones that contain more texture information. [Matas et al. 2006] selected the most adequate pixels to the linear approximation performed in [Jurie and Dhome 2001]. [Benhimane et al. 2007] do the same for the IC and ESM methods and the obtained AR results are shown in Figure 1.10.

Figure 1.10. Template matching for AR using subsets of the template image [Benhimane et al. 2007].

1.4.1.3.2.Interest Point Based

One of the most acclaimed model based approach, interest point based techniques have a local feature approach to deal with occlusion and illumination problems in an effective way. Since matching only a subset of image pixels reduces computational complexity and increases application's performance, interest point based techniques rely on finding local texture features in images and matching them to estimate camera pose. In an effort to improve accuracy, the keyframe concept can also be incorporated to these techniques in order to reduce or eliminate drift problems. Jitter problems are also handled by taking temporal information into account in the pose estimation problem [Vacchetti et al. 2004a]. This technique pipeline has gained many improvements in performance and robustness from the use of both online and offline information.

First of all, it is necessary to generate the offline information that will help the technique to avoid drift. This information appears as a keyframe, a structure responsible for keeping the 2D points extracted in a known camera pose, the camera pose, the 3D points and its normals corresponding to the extracted 2D points. Using this formulation, only a few keyframes are needed to start tracking the object. In order to generate a keyframe, it is necessary to capture a frame and discover the camera 3D pose for that frame. If the system is working with synthetic data, its pose is already known, but if it is handling with real data, the pose can be estimated with a commercial software like Autodesk ImageModeler [Autodesk 2009]. After having the correct pose, the system extracts the 2D points from the frame and backprojects them onto the 3D model to find the 3D points and its normals. Backprojection may be done using the Facet-ID approach that makes use of the graphics hardware to accelerate the process [Vacchetti et al. 2004a] (see Figure 1.11).

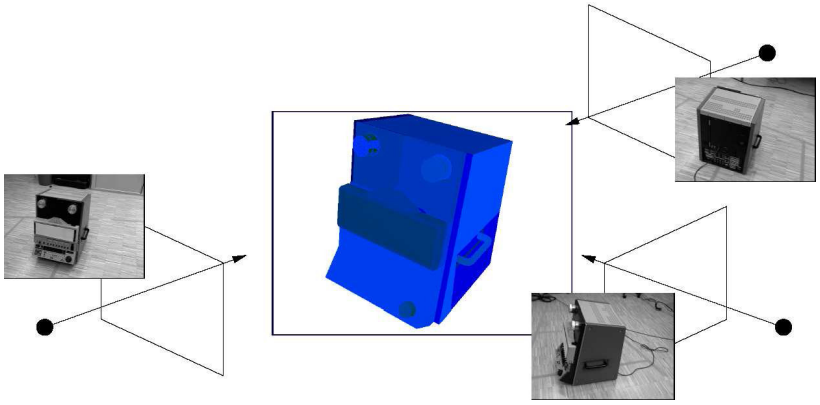


Figure 1.11. Keyframes generated from 3 different camera poses [Vacchetti et al. 2004a].

Once keyframes are generated, it is also necessary to know the first camera pose (initialization), which can be defined manually or through an automated process [Lepetit et al. 2003]. After that, tracking begins with the discovery of the nearest keyframe to the current frame. This is done by comparing the last pose with all keyframes poses using a Mahalanobis distance approach. However, the poses of the chosen keyframe and the current frame may be not close enough to allow the matching of their interest points. Due to this, an intermediate synthetic image with a pose near to the current frame is generated by applying a homography to the keyframe image (see Figure 1.12).

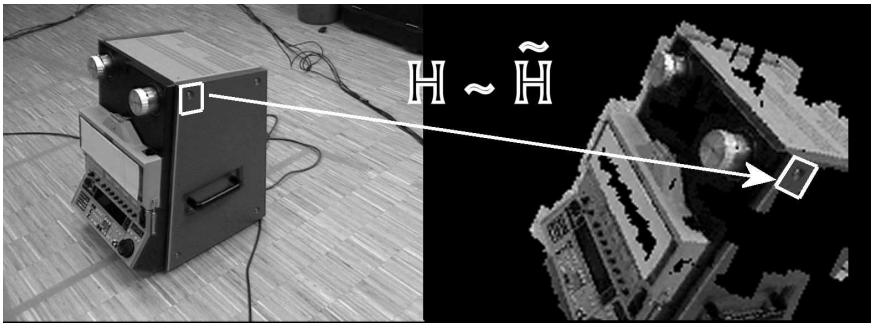


Figure 1.12. Intermediate image generation [Vacchetti et al. 2004a].

The extraction and matching of interest points can be done with the Harris corner detector [Harris and Stephens 1988] and the Zhang matcher [Zhang et al. 1995], respectively. The Harris detector is a local features detector that relies on an auto-correlation matrix to define the interest points, being used in both intermediate image and the current frame. Zhang feature matcher uses the normalized cross-correlation approach to decide which features from current frame image have correspondent features in the intermediate image. In this approach, the system finds the most similar feature in the intermediate image for each feature from the current image and vice-versa. After that, it just considers as a valid correspondence the pair of features that are matched from current to intermediate and from intermediate to current frame.

Based on matching results, camera pose can be estimated by the non-linear optimizer called Levenberg-Marquardt in association to a robust estimator like the Tukey M-estimator, as described in Section 1.2. In order to avoid an undesired jitter effect, it is also incorporated to the estimation process the use of previous pose

information, guarantying temporal continuity (see [Vacchetti et al 2004a] for more details).

The interest point based tracker can be used to track many kinds of objects, from a simple object like a tea box to a complex human face. The tracking is still stable even in the presence of occlusion or aspect changes (illumination, scale, background), as can be seen Figure 1.13. Although, hybrid techniques are also an option as in [Vachetti et al. 2004b] that uses an association between an edge based approach and a texture based interest point approach (see Figure 1.14).

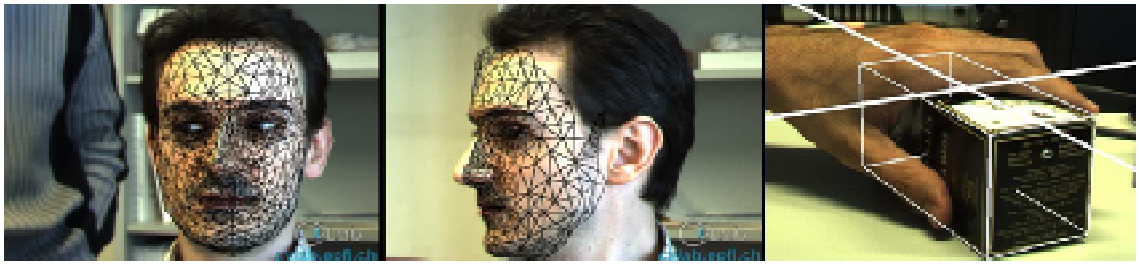


Figure 1.13. Tracking of a human face and a tea box using an interest point based technique [Vacchetti et al 2004a].

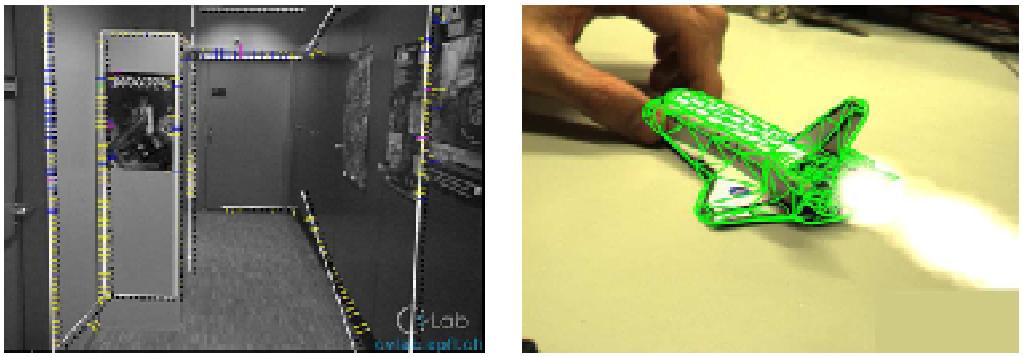


Figure 1.14. Hybrid tracking of a corridor and a spaceship [Vachetti et al. 2004b].

1.4.2.Tracking by Detection

The tracking by detection techniques are model based methods that detect the model even if there is no initial estimation. They are usually slower than the recursive methods and commonly need some offline process to arrange the model data for the tracking itself.

Tracking by detection techniques can be divided into edge based and texture based. Each one has its relevance to some application domain, as described below.

1.4.2.1.Edge Based

Edge based tracking by detection techniques use edge information from the object that will be tracked in order to detect it in a real image sequence. Usually, attempting to detect the edges, image processing algorithms are used to extract points with high gradient changes and then they are matched with the model edges. The pose that covers the greatest number of matched edges is kept as the current pose.

In general, edge based methods have the advantage of reflectance robustness when compared to texture based ones. This occurs due to the fact that, when the

specular component of the object is high, a lot of texture information is lost by light reflection. However, for edge detection this is not a problem. In fact, it can be an advantage, because in some cases, when the reflectance of a face is high, the edges become more highlighted.

1.4.2.1.1.View Based

The technique shown here is based on the work of [Wiedemann et al. 2008] and works well for non-planar objects. It is basically an edge based tracking by detection method that tries to find out matches of the current frame edges with model edges that were projected in various distinct views. These model edges are processed in an offline training phase and then stored for further fast matching.

In the offline phase, the 3D triangulated model of the object is colored based on each face normal vector, as shown in Figure 1.15. Each color component (R, G, B) of a face color refers respectively to a coordinate (X, Y, Z) of the normal vector of the face in question. By this way, the edge amplitude measured in this image is directly related with the 3D angle between the two neighboring faces.

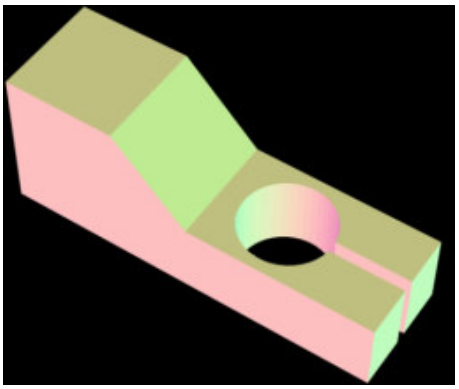


Figure 1.15. A clamp model colored based on the normal vectors of its faces [Wiedemann et al. 2008].

In a synthetic scene, the model is kept in the center of a sphere coordinate system and various images are taken of it, from distinct points of a spherical shell, as can be seen in Figure 1.16. The range of the covered views by the spherical shell is set by three parameters that are: λ (longitude), φ (latitude) and d (distance). First, a few number of images are taken from far positions attempting to cover a high pose range. In the next step (using a hierarchical structure), for each of the first images, children ones are taken and stored into the next level. These children images are taken with the intention of dividing the pose range of the previous level. As it seems necessary, levels are created in the same way. A link between the pose used to obtain the image and the image itself is maintained.

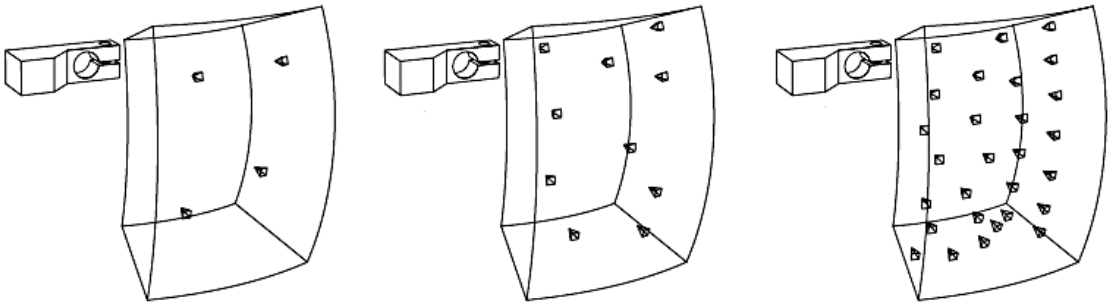


Figure 1.16. Model views (represented by small square pyramids) taken into a spherical shell. Few images from hierarchy level 1 (left), followed by subsequent levels 2 (middle) and 3 (right) [Wiedemann et al. 2008].

This hierarchical structure presented above is combined with the pyramid approach proposed in [Steger 2002]. Then, for each existent image new ones are generated by rotations and scales transformations. The higher is the image resolution more precise will be these transformations and consequently a greater number of new images will be generated. These transformed images are stored coupled with the untransformed one. Then a feature extraction algorithm, in this case a color gradient operator [Di Zenzo 1986], is applied to each image, which returns direction vectors for the image points.

After the offline structure is built and the direction vectors were extracted, the tracking itself can start. The runtime pipeline starts with camera frame grabbing and then the current frame is generated in various resolutions, to deal with the resolution multiplicity of the offline data. An exhaustive search is done in the top views images, with the current frame, trying to match it with the first level of the hierarchy structure. In more details, for each image in this level a similarity measure [Steger 2002] is done in all possible rotations, translations and scales (respecting the user ranges restrictions). A similarity measure between two images can be better understood observing the following equation:

$$s = \frac{1}{n}\sum_{i=1}^{n}\frac{(d_i, d'_i)}{\|d_i\|.\|d'_i\|}. \qquad (21)$$

In the equation above, n is the number of relevant points to be matched (in this case the model edge points projected onto the image), d_i is the direction vector of the i-th model point and d'_i is the correspondent point in the current image. At the end of the measure, if s surpasses a given value called s_{min}, then the two images are considered as a match case.

In sequence, the lower views hierarchy levels will be explored, however with no more need to dispend time with this exhaustive approach. Using the obtained information, the matching cases will be refined attempting to match only with their child views. The translations, rotations and scales that will be exploited are restricted too, and now the similarity measure with the children views is done only in a close neighborhood of the father pose.

If the object was detected in the previous frame, then there is no need to do the exhaustive search in the top hierarchy level. The matching is done in the neighborhood

of the previous obtained pose, accelerating significantly the performance of the technique. However, if after this neighboring restriction no matches are found, then the pose neighborhood is incrementally increased.

After the refined matches are found, a minimization can be done to find out a better pose. The fact is that not ever a precise pose will be found only through the matching process, since it would be an unrealistic restriction to consider that every pose to be tracked is present in the offline data. This refinement is done by recovering the edge points of the current image and using the obtained matching pose as an initial estimation for the procedure presented in Subsection 1.2.2.2, that will iterate to find out a more precise pose.

The view based method showed to be a precise approach for model based tracking as can be seen in some results shown in Figure 1.17. However, there must be some restrictions about the covered surface of the object because the technique is impracticable (in time consumption terms) if all possible views of the object are considered.

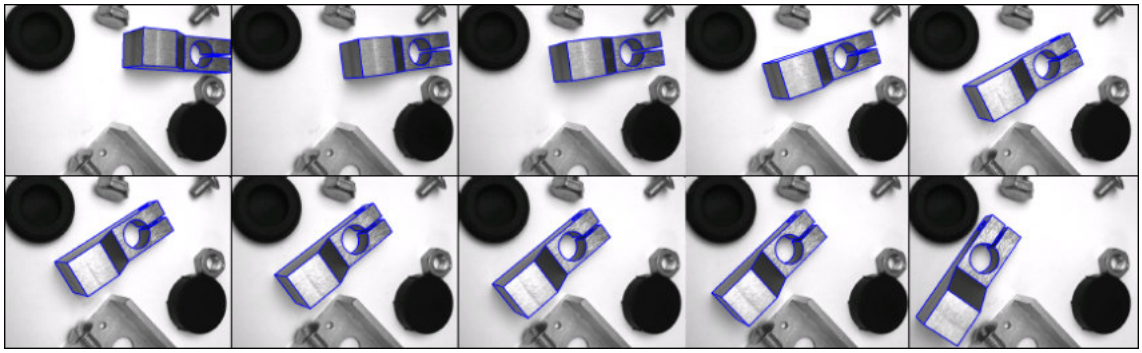


Figure 1.17. View based results with a clamp model [Wiedemann et al. 2008].

1.4.2.2.Texture Based

On the other hand, there are the texture based tracking by detection methods. They do not even care for edge information, but the object must have textured faces. Another restriction of these methods is that if the object material has a high reflection coefficient, then the texture tends to lose lots of its characteristics and the tracking can fail. The subsection below describes a texture based tracking by detection technique in more detail.

1.4.2.2.1.Keypoint Based

The keypoint based tracking by detection technique makes use of algorithms for extraction and matching of keypoints. Keypoints can be understood as relevant 2D points that can be recognized even with rotation, scale and illumination changes. This relevant information is stored into high dimensional descriptors that are associated to the keypoints.

Therefore, an offline phase is also needed, and then during the tracking some information is extracted from the current frame and attempted to be matched with the offline data. After that, the current pose is calculated attempting to be coherent with the greater number of matches. The technique is described in more detail below.

The offline phase is done by the following steps:

- Given the 3D model of the object that will be tracked, some keyframes are taken from a synthetic scene. These keyframes must cover all the faces of the object, and if some face is not in any one of the keyframes then this face will not be able to be recognized in the tracking process. For each keyframe is maintained a relation with the camera pose that originated it.
- Then, an algorithm of keypoint extraction is used in each keyframe (e.g. SIFT [Lowe 2004]). For each keypoint, its 3D model point is associated. All the keypoints are finally stored to be used in the matching process. In order to make the matching search faster, they are stored into a kd-tree structure that basically arranges the keypoints based on its descriptors.
- The image information at this point is no more needed. Now the keypoints are grouped by the image that they derived from and for each group the pose relation mentioned before is still maintained.

After the offline data is prepared (arranged into the kd-tree) the object itself can be tracked. The online tracking phase obeys the pipeline of extraction, matching and pose estimation, as is described below.

The current frame is grabbed by the camera and then a keypoint extraction is done in this image. Here the same method must be used as in the offline phase, since these new keypoints must be matched with the offline data.

The matching process, illustrated in Figure 1.18, takes each keypoint that was extracted from the current frame and does a Best-Bin-First search [Skrypnyk and Lowe 2004] in the offline data kd-tree. Then, the two nearest neighbors are returned. In order to verify if the nearest neighbor matches with the current keypoint, a threshold is applied to the ratio $dist0 \,/\, dist1$, where $dist0$ and $dist1$ are the Euclidean distances between each neighbor and the current keypoint. If the ratio is greater than the threshold, then the keypoint and the nearest neighbor are considered as a match case. If not, then the keypoint is discarded, because, given the cardinality of the descriptors, the probability that the distances are close is too low, and in most cases this indicates an invalid match case.

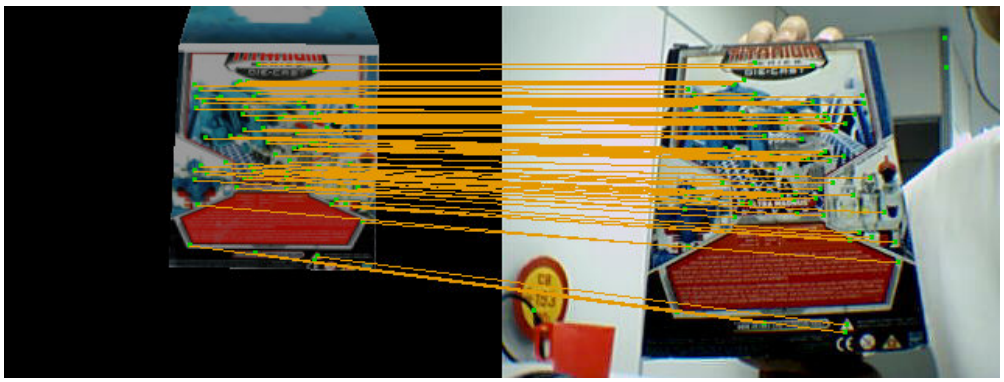


Figure 1.18. Keypoint matching.

If the set of matches reaches a minimum number (that can be specified by the user), then it is possible to calculate the current pose. If not, then the current frame is

considered as a tracking failure. As each match keeps a relation between the current 2D point and the 3D model point, it is possible to solve the pose problem using any of the methods described in Section 1.2. In [Skrypnyk and Lowe 2004], RANSAC is used together with a pose estimator similar to the one described in Subsection 1.2.2.2. This estimator needs an initial pose estimate to find out the current one. In these cases, if the object was successfully tracked on the last frame, the last pose is used as an initial estimate. If a tracking failure has occurred, or the current frame is the first one, the initial pose is the one relative to the keyframe that contributes to the greatest number of matches.

Figure 1.19 shows some results of the keypoint based technique. In general, problems encountered are low performance, jitter and a little of drift. Besides that, the technique is robust to partial occlusion, illumination changes and scene motions.

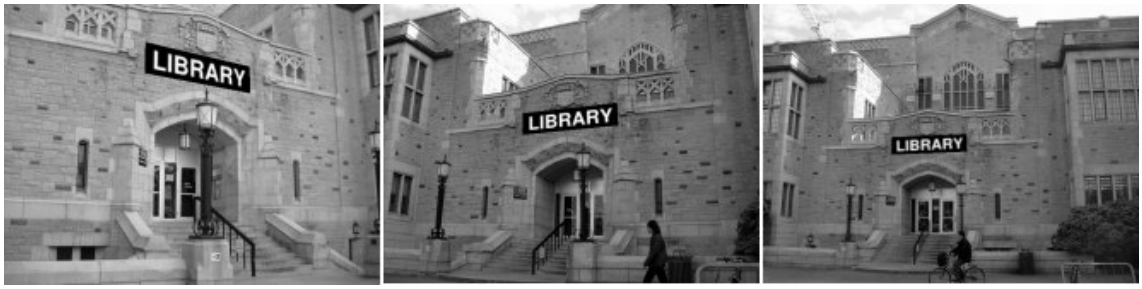


Figure 1.19. Keypoint based results using an entrance of a university library as the tracked model and augmenting it with a 2D sign [Skrypnyk and Lowe 2004].

Some hybrid techniques have exploited the keypoint based method focusing on the advantage of its robustness and tracking failures recovering. Figure 1.20 presents some results from the combination of the template matching and keypoint based approaches for tracking objects with planar surfaces [Ladikos et al. 2007]. In this implementation, template-based tracking runs until a large displacement occurs, and then the tracking switches to a keypoint based implementation. Other hybrid method was proposed by [Hinterstoisser et al. 2009], which also uses keypoint based tracking and template matching for planar objects detection, supporting flexible surfaces tracking due to the use of patches recognition.

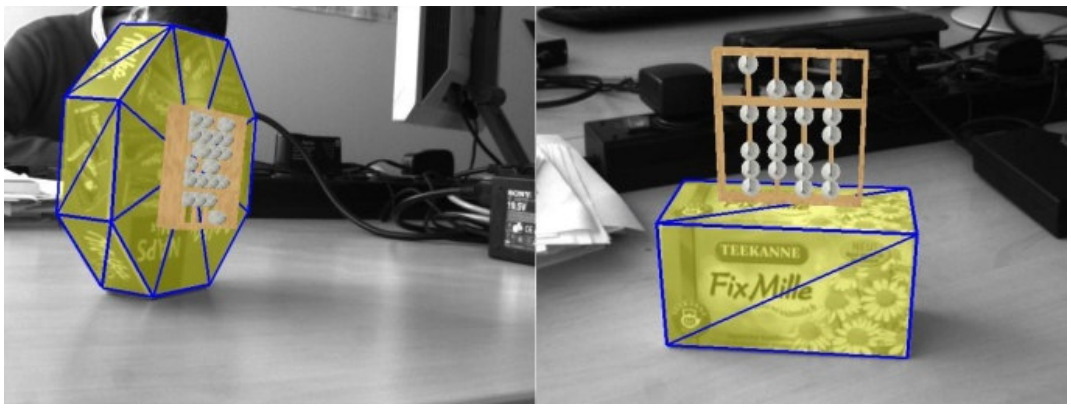


Figure 1.20. Augmentation results from template matching and keypoint based hybrid technique [Ladikos et al. 2007].

1.5.Model Based Techniques Evaluation

The presented approaches for model based markerless AR can be analyzed taking into account some relevant metrics. One of the most important metrics is the presence of automatic detection, where user interaction is not required to determine the initial camera pose. When evaluating an AR application, the processing load needed to perform tracking has to be quantified. If the time slice used to estimate camera pose is short, the remaining processing time can be dedicated to other tasks. Accuracy and robustness are the last two metrics considered in the methods evaluation. While accuracy is related to the correctness of pose estimation throughout the frames, robustness is about how resistant is the tracker to noise sources. Table 1.1 compares the model based markerless AR methods introduced in this chapter, according to the presented criteria. The comparison considers the features that are common to most of the techniques of a given category.

Table 1.1. Model based methods evaluation

Category	Method	Detection	Processing	Accuracy	Robustness
Recursive tracking	Edge based	No	Low	Jitter	Sensible to: • Fast camera movement • Cluttered background
	Optical flow based	No	Low	Cumulative errors	Sensible to: • Fast camera movement • Lighting changes
	Template matching	No	Low	Highly accurate	Sensible to: • Fast camera movement • Lighting changes • Occlusion
	Interest point based	No	High	Accurate	Sensible to: • Fast camera movement
Tracking by detection	View based	Yes	High	Accurate	Restricted range of poses
	Keypoint based	Yes	High	Jitter and drift	Robust

Model based markerless AR approaches can also be evaluated according to their applicability to the scenario. Edge based methods are more suitable when the tracked objects are polygonal, specular or when they have strong contours. If objects are textured, optical flow based techniques should be used (in case of constant lighting and not very large camera displacement). When optical flow is not an option, texture based methods may be the best solution, as they are more accurate. If the textured object is planar, template matching presents good results with low CPU load; otherwise, interest point based methods should be used. Keypoint based techniques suffer from jitter when they estimate each pose based only on current frame information. The temporal information should be taken into account in order to alleviate this problem. However, keypoint based tracking tends to be less accurate than texture based recursive tracking, due to lack of precision on feature matching. View based techniques are highly

accurate, but can cover only a restricted range of rotations and scales of the target object with low detection rates.

1.6. Final Considerations

This chapter has surveyed model based markerless 3D tracking techniques for AR that use only one camera and work in real-time. Nevertheless, there are other markerless AR approaches, such as that in [Cornelis 2004], which works offline and is mainly applied to video post production.

Beyond the advantages described of the previously presented tracking methods, SfM based techniques should also be highlighted, due to their ability to augment completely unknown scenes. Research regarding SfM applied to markerless AR is still in its infancy and therefore there are several open problems that need careful attention. For example, real-virtual interaction, which is exploited by model based markerless AR applications [Comport et al. 2006], has not yet been approached by SfM based ones. Indeed, SfM based techniques retrieve information about the environment that could be used to build complex applications taking advantage of such kind of interaction.

However, it is important to say that, according to the problem tackled, purely model based or hybrid approaches should be considered. Even though there are markerless techniques for AR, the use of markers is suitable for systems that do not mind the presence of artificial elements in the scene. In other words, the solution is defined by the kind of problem being faced.

References

Autodesk (2009), "Autodesk ImageModeler", http://www.autodesk.com/imagemodeler, April.

Azuma, R. (1997) "A Survey of Augmented Reality", Presence: Teleoperators and Virtual Environments, vol. 6, n. 4, p. 355-385.

Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S. and Macintyre, B. (2001) "Recent Advances in Augmented Reality", IEEE Computer Graphics and Applications, vol. 21, n. 6, p. 34-47.

Baker, S. and Matthews, I. (2004) "Lucas-Kanade 20 Years On: A Unifying Framework", International Journal of Computer Vision, vol. 56, n. 3, p. 221-255.

Basu, S., Essa, I. and Pentland, A. (1996) "Motion Regularization for Model-Based Head Tracking", Proc. International Conference on Pattern Recognition, vol. 3, p. 611-616.

Benhimane, S. and Malis, E. (2004) "Real-Time Image-Based Tracking of Planes Using Efficient Second-Order Minimization", Proc. International Conference on Intelligent Robots and Systems, p. 943-948.

Benhimane, S., Ladikos, A., Lepetit, V., Navab, N. (2007) "Linear and Quadratic Subsets for Template-Based Tracking", Proc. IEEE Conference on Computer Vision and Pattern Recognition, 6 p.

Bonsor, K. (2009) "How Augmented Reality Will Work", http://computer.howstuffworks.com/augmented-reality.htm, April.

Brockett, R. (1984) "Robotic Manipulators and the Product of Exponentials Formula", Proc. International Symposium on Mathematical Theory of Networks and Systems, pp. 120-127.

Comport, A. I., Marchand, E., Pressigout, M., Chaumette, F. (2006) "Real-Time Markerless Tracking for Augmented Reality: The Virtual Visual Servoing Framework", IEEE Transactions on Visualization and Computer Graphics, vol. 12, n. 4, p. 615-628.

Cornelis, K. (2004) "From Uncalibrated Video to Augmented Reality", PhD Thesis, Katholieke Universiteit Leuven.

Dellaert, F. and Collins, R. (1999) "Fast Image-Based Tracking by Selective Pixel Integration", Proc. ICCV Workshop of Frame-Rate Vision, 22 p.

Di Zenzo, S. (1986) "A Note on the Gradient of a Multi-Image", Computer Vision, Graphics, and Image Processing, vol. 33, no. 1, p. 116-125.

Drummond, T. and Cipolla, R. (2002) "Real-Time Visual Tracking of Complex Structures", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, n. 7, p. 932-946.

Faugeras, O. Three-Dimensional Computer Vision: A Geometric Viewpoint, MIT Press, 1993.

Ferwerda, J. (2003) "Three Varieties of Realism in Computer Graphics", Proc. SPIE Human Vision and Electronics Imaging, p. 290-297.

Fischler, M. and Bolles, R. (1981) "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography", Communications of the ACM, vol. 24, n. 6, p. 381-395.

Forsyth, D. and Ponce, J. Computer Vision - A Modern Approach, Prentice-Hall, 2002.

Gomes Neto, S., Bueno, M., Farias, T., Lima, J.P., Teichrieb, V., Kelner, J. and Santos, I. (2008) "Experiences on the Implementation of a 3D Reconstruction Pipeline", International Journal of Modeling and Simulation for the Petroleum Industry, vol. 2, n. 1, p. 7-15.

Haag, M. and Nagel, H.-M. (1999) "Combination of Edge Element and Optical Flow Estimates for 3-D Model-Based Vehicle Tracking in Traffic Image Sequences", International Journal of Computer Vision, vol. 35, n. 3, p. 295-319.

Hager, G. and P. Belhumeur (1998) "Efficient Region Tracking with Parametric Models of Geometry and Illumination", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, n. 10, p. 1025-1039.

Haller, M., Billinghurst, M., Bruce, T. Emerging Technologies of Augmented Reality: Interfaces and Design, Idea Group Publishing, 2007.

Harris, C. and Stephens, M. (1988) "A Combined Corner and Edge Detector", Proc. Alvey Vision Conference, p. 147-151.

Hinterstoisser, S., Kutter, O., Navab, N., Fua, P., Lepetit, V. (2009) "Real-Time Learning of Accurate Patch Rectification", Proc. IEEE Conference on Computer Vision and Pattern Recognition, 6 p.

Jurie, F. and Dhome, M. (2001) "A Simple and Efficient Template Matching Algorithm", Proc. IEEE International Conference on Computer Vision, p. 544-549.

Ladikos, A., Benhimane, S., Navab, N. (2007) "A Real-Time Tracking System Combining Template-Based and Feature-Based Approaches", Proc. International Conference on Computer Vision Theory and Applications, p. 325-332.

Lee, T. and Höllerer, T. (2008) "Hybrid Feature Tracking and User Interaction for Markerless Augmented Reality", Proc. IEEE Virtual Reality, p. 145-152.

Lepetit, V. and Fua, P. (2005) "Monocular Model-Based 3D Tracking of Rigid Objects", Foundation and Trends in Computer Graphics and Vision, vol. 1, n. 1, p.1-89.

Lepetit, V., Vacchetti, L., Thalmann, D., Fua, P., (2003) "Fully Automated and Stable Registration for Augmented Reality Applications", Proc. IEEE and ACM International Symposium on Mixed and Augmented Reality, p. 93-102.

Lima, J. P., Gomes Neto, S., Bueno, M., Teichrieb, V., Kelner, J. and Santos, I. (2008) "Applications in Engineering Using Augmented Reality Technology", Proc. CILAMCE.

Lowe, D. (2004) "Distinctive Image Features from Scale-Invariant Keypoints", International Journal of Computer Vision, vol. 60, no. 2, p. 91-110.

Lu, C., Hager, G., Mjolsness, E. (2000) "Fast and Globally Convergent Pose Estimation from Video Images", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, n. 6, p. 610-622.

Lucas, B. and Kanade, T. (1981) "An Iterative Image Registration Technique with an Application to Stereo Vision", Proc. Imaging Understanding Workshop, p. 121-130.

Matas, J., Zimmermann, K., Svoboda, T. and Hilton, A. (2006) "Learning Efficient Linear Predictors for Motion Estimation", Proc. Indian Conference on Computer Vision, Graphics and Image Processing, p. 445-456.

Moreno-Noguer, F., Lepetit, V., Fua, P. (2007) "Accurate Non-Iterative O(n) Solution to the PnP Problem", Proc. IEEE International Conference on Computer Vision, 8 p.

Platonov, J., Heibel, H., Meier, P. and Grollmann, B. (2006) "A Mobile Markerless AR System for Maintenance and Repair", Proc. IEEE International Symposium on Mixed and Augmented Reality, p. 105-108.

Pressigout, M., Marchand, E. and Mémin, E. (2008) "Hybrid Tracking Approach Using Optical Flow and Pose Estimation", Proc. IEEE International Conference on Image Processing, p. 2720-2723.

Skrypnyk, I. and Lowe, D. (2004) "Scene Modelling, Recognition and Tracking with Invariant Image Features", Proc. International Symposium on Mixed and Augmented Reality, p. 110-119.

Steger, C. (2002) "Occlusion, Clutter, and Illumination Invariant Object Recognition", Proc. Conference on Photogrammetric Computer Vision, vol. 34, part 3A, Commission III, p. 345-350.

Teichrieb, V., Lima, J.P., Apolinário, E., Farias, T., Bueno, M., Kelner, J. and Santos, I. (2007) "A Survey of Online Monocular Markerless Augmented Reality", International Journal of Modeling and Simulation for the Petroleum Industry, vol. 1, n. 1, p. 1-7.

Trevisan, D., Vanderdonckt, J. and Macq, B. (2002) "Analyzing Interaction in Augmented Reality Systems", Proc. ACM Multimedia - International Workshop on Immersive Telepresence, p. 56-59.

Triggs, B., McLauchlan, P., Hartley, R. and Fitzgibbon, A. (2000) "Bundle Adjustment – A Modern Synthesis", In: Vision Algorithms: Theory and Practice, Edited by Bill Triggs, Andrew Zisserman and Richard Szeliski, Springer Berlin, Germany.

Vacchetti, L., Lepetit, V. and Fua, P. (2004a) "Stable Real-Time 3D Tracking Using Online and Offline Information", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, n. 10, p. 1385-1391.

Vacchetti, L., Lepetit, V., and Fua, P. (2004b) "Combining Edge and Texture Information for Real-Time Accurate 3D Camera Tracking", Proc. IEEE/ACM international Symposium on Mixed and Augmented Reality. p. 48-57.

Wiedemann, C., Ulrich, M. and Steger, C. (2008) "Recognition and Tracking of 3D Objects", Proc. Pattern Recognition Symposium – Lecture Notes in Computer Science vol. 5096, p. 132-141.

Wuest, H., Vial, F. and Stricker, D. (2005) "Adaptive Line Tracking with Multiple Hypotheses for Augmented Reality", Proc. International Symposium on Mixed and Augmented Reality, p. 62-69.

Zhang, Z., Deriche, R., Faugeras, O., Luong, Q. (1995) "A Robust Technique for Matching Two Uncalibrated Images through the Recovery of the Unknown Epipolar Geometry", Artificial Intelligence, vol. 78, no. 1, p. 87-119.

Capítulo

2

Serious Games para Saúde e Treinamento Imersivo

Liliane S. Machado, Ronei M. Moraes e Fátima L. S. Nunes

Abstract

This chapter presents the recent developments related to games and how they have been used to improve training in medicine and education in health. Particularly, will be discussed the main components of these applications and their requirements for development (script, platform, devices, intelligence). Following, some applications will be presented considering these requirements.

Resumo

Este capítulo apresenta os recentes desenvolvimentos relacionados a jogos e como estes têm sido utilizados para auxiliar e incrementar o treinamento em medicina e áreas da saúde em geral. Particularmente, são abordados os principais componentes de aplicações desta natureza e seus requisitos de desenvolvimento (roteiro, plataforma, dispositivos e inteligência). Finalmente, serão discutidas aplicações sob os aspectos destes requisitos.

2.1. Introdução

Na computação os jogos podem ser caracterizados por aplicações baseadas em computação gráfica cujo objetivo é prover entretenimento, ou seja, experimentação em um ambiente interativo. Existem várias plataformas possíveis para um jogo eletrônico, tais como os computadores, os consoles (popularmente conhecidos como *video-games*), os mini-consoles (*handhelds*) e os dispositivos móveis (aparelhos celulares, Palms, etc). Cada uma dessas plataformas tem as suas próprias características de poder de processamento principal e de vídeo, capacidade das memórias e dispositivos de entrada/saída e até mesmo sistemas operacionais. Por exemplo, cada aparelho celular, mesmo os oriundos do mesmo fabricante, possui características diferenciadas como tamanho de tela, tipo de processador e até mesmo o sistema operacional embarcado.

Com o objetivo de lidar com as diferentes limitações individuais das plataformas e suas características próprias de jogabilidade, é necessária a adaptação dos jogos a cada uma delas. Dependendo do caso, um mesmo jogo poderá possuir versões ligeiramente diferentes para cada plataforma, sendo mantidas características muito similares. Porém, dependendo do tipo de jogo, as limitações levarão ao desenvolvimento de jogos completamente diferentes, tanto em termos gráficos, quanto de jogabilidade.

Existem várias concepções de gênero para os jogos eletrônicos. Pode-se citar os jogos de ação, aventura, corrida, educacionais, de entretenimento, esportivos, estratégia, infantis, combate e simuladores, dentre outros. Existem ainda jogos que são concebidos usando mais de uma desses conceitos. Neste trabalho, é enfocado um tipo especial de jogo eletrônico chamado *serious games*, que possui características presentes em mais de um dos gêneros citados acima. Particularmente, serão abordados os *serious games* com aplicação nas áreas de saúde e treinamento.

2.2. *Serious Games*

A sociedade tem experimentado uma categoria particular de jogos desenvolvida para abordar aspectos que não apenas o de entretenimento. Apesar de não haver uma definição precisa sobre o termo *serious games*, essa classe de jogos visa principalmente a simulação de situações práticas do dia-a-dia, com o objetivo de proporcionar o treinamento de profissionais, situações críticas em empresas, conscientização para crianças, jovens e adultos e mesmo para situações corriqueiras, como escolher os opcionais e a cor de um carro [Zyda, 2005]. Tais jogos, conhecidos como *serious games* utilizam a conhecida abordagem da indústria de jogos para tornar essas simulações mais atraentes e até mesmo lúdicas, ao mesmo tempo em que oferecem atividades que favorecem a absorção de conceitos e habilidades psicomotoras. Deste modo, o termo *serious games* passou a ser utilizado para identificar os jogos com um propósito específico, ou seja, que extrapolam a idéia de entretenimento e oferecem outros tipos de experiências, como aquelas voltadas ao aprendizado e ao treinamento [Blackman, 2005].

De forma a localizar temporalmente o surgimento dos *serious games*, pode-se dizer que isto ocorreu nos anos 80 com os simuladores desenvolvidos pelos Estados Unidos para a área militar. Assim, levando-se em conta que a realidade virtual nasceu com os simuladores de vôo na II Guerra Mundial, pode-se dizer que ela surgiu a partir de um conceito de *serious games*. Hoje, vários simuladores de vôo e de combate existem no mercado e alguns deles são considerados para o acúmulo de horas que um piloto de aviação civil deve possuir para estar apto a pilotar aviões de maior capacidade.O deslocamento em campos de batalha e a manipulação de veículos militares são também exemplos de aplicações. Atualmente, a conexão dos *serious games* à Realidade Virtual e Aumentada (RVA) encontra-se na proposta das aplicações e, principalmente, na forma de exploração dos recursos computacionais. A utilização da visualização estereoscópica e de dispositivos de interação intuitivos, a solução de problemas de processamento gráfico (*rendering*) e de modelagem, bem como o uso de métodos de simulação física para comportamento de materiais, são exemplos de características comuns aos *serious games* e à RVA. Entretanto, as aplicações de RVA tomaram um grande impulso na década de 90, mas a falta de validação e de estudos de aplicabilidade e demanda destes sistemas relegou vários deles ao ambiente de pesquisa

ou centros de excelência [Stone, 2009]. Assim, a visibilidade dos *serious games* resgata os avanços da RVA e aproxima-os da sociedade ao trazer aplicações que utilizam a tecnologia em um contexto aplicável de imediato. Adicionalmente, busca incluir a adoção de baixo custo nos projetos e o uso de dispositivos e plataformas convencionais sempre que possível.

Para fins de treinamento, os *serious games* são aplicados para simular situações críticas, que envolvam algum tipo de risco, tomada de decisões ou, ainda, desenvolver habilidades específicas. Em ensino, pode-se simular situações onde o uso de um conhecimento seja necessário para a evolução no jogo. Em alguns casos, ensino e treinamento podem ser combinados para simular situações onde se aprende algo para ser utilizado na própria simulação instantes depois. Os *serious games* também podem ser aplicados na conscientização humana sobre problemas sociais. Alguns exemplos de tais aplicações são apresentados a seguir.

a) Virtual University

O *Virtual University* é um jogo para auxiliar o seu usuário a entender o funcionamento de uma universidade, envolvendo a gestão de recursos humanos e várias formas de tomada de decisões administrativas e acadêmicas, visando a melhoria de vários indicadores universitários (Figura 2.1). O projeto é hospedado em www.virtual-u.org/.

(a) (b)

Figura 2.1. Projeto *Virtual University*: (a) Gestão de um Departamento de Inglês, via dados do próprio departamento; (b) Indicadores universitários de formação na graduação, mestrado e doutorado.

b) 3D Driving Academy

Este jogo teve como objetivo inicial recriar distritos de Paris, Londres e Berlim para a simulação de condução de veículos sob diferentes leis de trânsito européias. De fato, este sistema simula atualmente as leis de trânsito de seis países diferentes. Faz uso de uma *engine* de física para os veículos e de inteligência artificial para o controle de tráfego de centenas de carros, motocicletas, pessoas e semáforos presentes na simulação. Há ainda uma forma explícita de avaliação, na qual se verifica se o aprendiz

não violou alguma das leis de trânsito do país simulado (Figura 2.2). Mais informações sobre este jogo podem ser encontradas em www.3d-fahrschule.de.

(a) (b)

Figura 2.2. Jogo para simulação de leis de trânsito: (a) Simulação de direção sob tráfego leve; (b) Simulação de estacionamento.

c) Lemonade Tycoon 2

Este curioso jogo é direcionado à administração de negócios (Figura 2.3). Ele simula a gestão de um sistema de vendas ambulante de limonada. O usuário deve manipular a matéria-prima e o aperfeiçoamento das receitas, regulando os seus preços e as receitas. Deve se preocupar com as condições do tempo e movimentar os quiosques visando a melhora das vendas diariamente. Além disso, dispõe de uma Bolsa da Limonada, onde se pode comparar o desempenho dos seus negócios. Uma versão *freeware* deste jogo, limitada a apenas uma cidade, está disponível na Internet.

d) SimuTrans

O popular jogo SimCity, surgido na década de 1990, deu origem aos simuladores de cidades. Vários conceitos surgiram a partir dele como o alemão SimuTrans, que simula também aspectos econômicos e de transporte de uma cidade (www.simutrans.de). O jogo conta com várias traduções para diferentes línguas, incluindo o português. Na Figura 2.4, mostra-se a coordenação do acesso à marina usando-se trem e ônibus.

e) Personalização

Neste caso, a simulação é realizada via Web em páginas nas quais é possível escolher a cor e os opcionais de um veículo a ser adquirido, obtendo em tempo-real a configuração do carro com as opções selecionadas. Atualmente, a grande maioria das montadoras de veículos do mundo disponibiliza esse tipo de sítio para agilizar as suas vendas e fornecer informações detalhadas aos seus consumidores (Figura 2.5). Em vários sítios desta natureza é possível ao consumidor adquirir o veículo segundo as opções simuladas

diretamente da montadora e escolher a agência para retirar o veículo. Um exemplo brasileiro é o sítio: www.fiat.com.br/monte-seu-carro/.

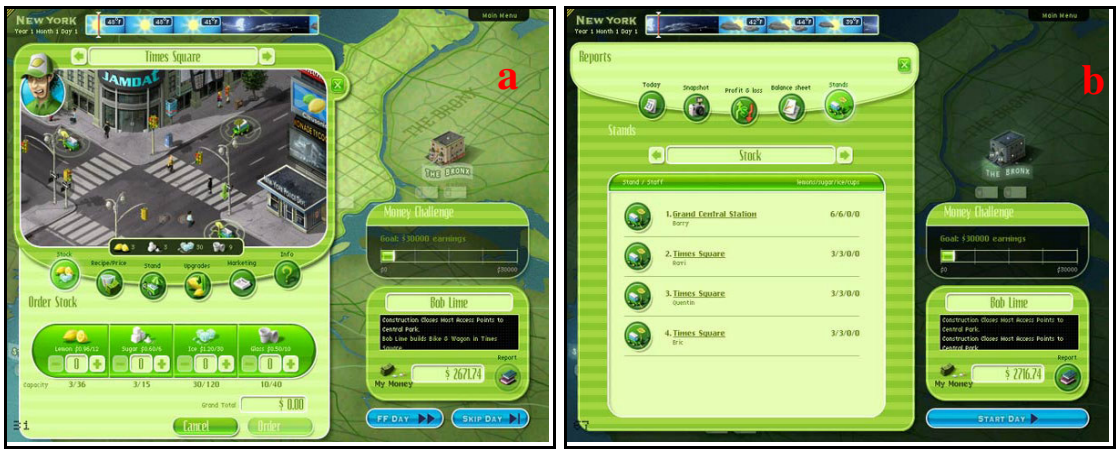
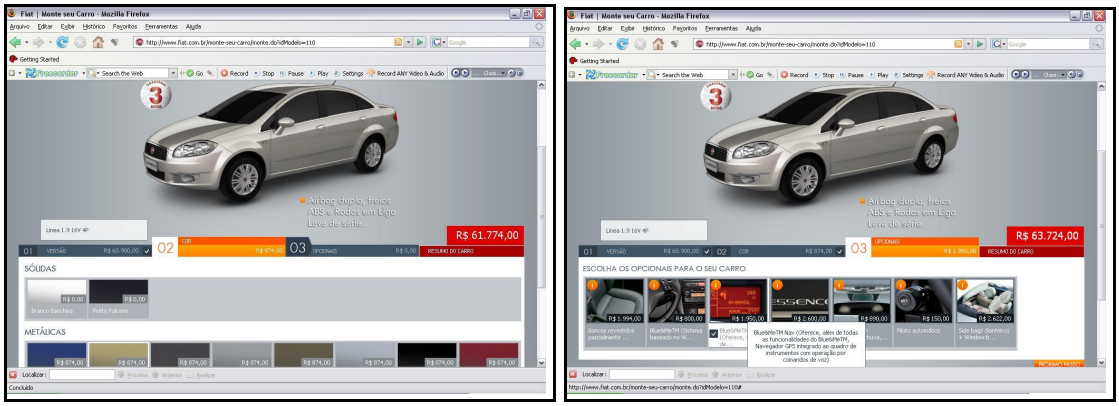


Figura 2.3. Jogo de negócios: (a) Alocação dos carrinhos de limonada em um setor da cidade; (b) Gerenciamento dos estoques de matéria prima.

Figura 2.4. O simulador de cidades SimuTrans.



(a) (b)

Figura 2.5. Customização de veículo: (a) Simulação das cores disponíveis para o carro e os seus valores em moeda corrente; (b) Disponibilidade dos opcionais e os seus respectivos valores.

f) Emergências

Vários simuladores para emergências e simulações de risco existem na literatura. Particularmente, o *Triple Zero Hero* é um jogo criado para o Departamento de Serviços Emergenciais de Queensland, na Austrália. Ele retrata diversas situações de emergência e o salvamento de pessoas em situações de risco. Sua forma de interação com o usuário pode ser realizada por meio de teclado e mouse ou por meio de *gamepads*. (Figura 2.6).

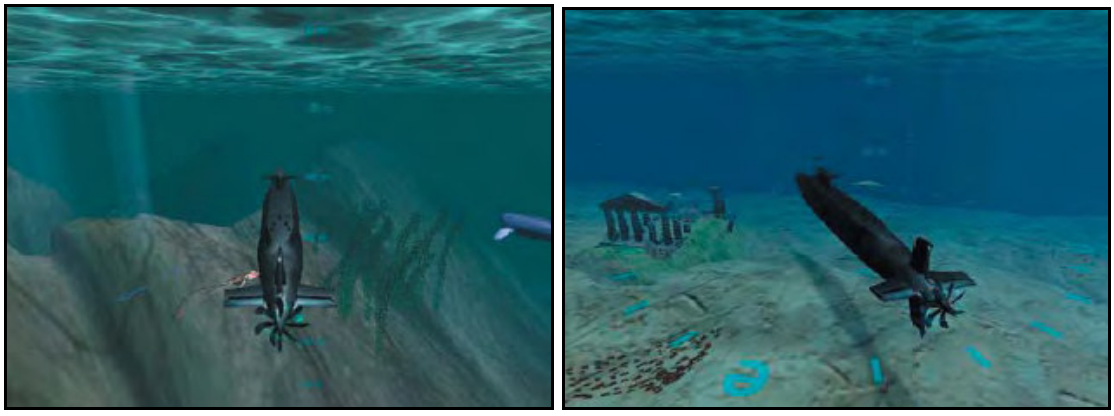
(a) (b)

Figura 2.6. (a) Tela de abertura do *Triple Zero Hero*, onde se pode ver a possibilidade de controle via *gamepad*; (b) Simulação de combate a um incêndio, com o indicador de tempo de vítimas na parte superior da tela.

g) Museus

Um interessante caso de museu interativo é o Museu *Cité de la Mer*, no qual há uma simulação interativa do primeiro submarino nuclear francês – *Le Redoutable*. Nela,

simula-se a navegação e manobra do submarino em diferentes temperaturas da água, diferentes níveis de salinidade (o que pode provocar problemas de instabilidade) e a presença de animais marinhos. O jogo é direcionado a visitantes do museu e a missão dos três usuários da simulação é conduzir o submarino em segurança a um determinado ponto do oceano (Figura 2.7). Informações adicionais podem ser encontradas em www.virtools.com.



(a) (b)

Figura 2.7. (a) Navegação na presença de animais marinhos e (b) alcançando um objetivo na simulação: um conjunto de ruínas imersas no oceano.

2.3. Aplicações em Saúde

Um dos setores que tem se beneficiado dos *serious games* visando o treinamento é o da saúde. As dificuldades encontradas na obtenção de materiais, validação de produtos e treinamento de pessoal, bem como a necessidade de novas abordagens para reabilitação e ensino de hábitos saudáveis, tornam os jogos um importante aliado do ensino, treinamento e simulação para a saúde, beneficiando profissionais e pacientes. A utilização destes jogos em ambientes imersivos e a inclusão de dispositivos não convencionais estabelecem uma relação direta com as aplicações de RVA, na qual o conceito de *serious games* pode contribuir para a motivação do aprendizado em ambientes virtuais. Johnsen *et al.* (2007) conduziu pesquisas que comprovaram o aprendizado efetivo e a transferência do aprendizado para ambientes reais quando tais aplicações são utilizadas para fins de educação e treinamento.

As finalidades das aplicações são as mais diversas e há vários grupos de pesquisa no mundo todo que têm a simulação de procedimentos médicos como objeto de investigação. Trabalhos recentes voltados a diferentes tarefas já oferecem realismo ao treinamento e têm potencial para se tornarem *serious games*. Outros projetos, já concebidos com objetivo definido para a saúde, têm sido difundidos e ampliados seus escopos de ação. Para facilitar a descrição destes projetos, estes podem ser separados em quatro categorias principais:

a) auxiliares de terapia

b) promoção da saúde e condicionamento físico

c) monitoramento da saúde

d) treinamento

Os jogos auxiliares de terapias são voltados a pessoas em processo de reabilitação e tratamentos físicos ou psicológicos. Provavelmente as aplicações mais conhecidas são aquelas voltadas ao tratamento de fobias, como medo de altura, de falar em público, de insetos (Figura 2.8) e de direção, dentre outros [Hodges *et al.*, 1995] [Juan *et al.*, 2005] [Paiva *et al.*, 2006][Slater *et al.*, 2006] [Juan *et al.*, 2007]. Tais aplicações buscam reproduzir situações de medo, introduzindo-as paulatinamente entremeadas a outras situações, de forma a habituar o usuário a elas por meio de experimentação. O uso da RVA é bastante comum nestes casos, nos quais dispositivos especiais costumam isolar o usuário da realidade, transportando-o para o mundo virtual. De maneira análoga, a própria realidade pode ser acrescida de itens relacionados ao tema, com a projeção de objetos virtuais em locais específicos do mundo real.

No contexto da RVA, já foi verificada a evolução de pacientes após o uso de sistemas com tais tecnologias considerando, por exemplo, a reabilitação de pacientes com seqüelas de atenção e percepção causadas por acidente vascular cerebral [Cardoso *et al.*, 2006] e a reabilitação da área do tornozelo como estudo de caso [Deutsch *et al.*, 2006]. Apesar destas aplicações não terem sido concebidas na forma de um jogo, os autores da última descobriram na avaliação realizada que o desempenho do paciente aumentou a partir do uso do sistema. O sucesso de tais aplicações também foi verificado na melhoria do controle postural e na minimização de quedas em pessoas idosas [Virk e McConville, 2006], o que demonstra efetividade e potencialidade destas aplicações como *serious games*. Comercialmente, estas aplicações poderiam ser adaptadas e exploradas para incentivar o usuário a se expor a situações temidas ou de risco, tornando a atividade desafiadora e prazerosa.

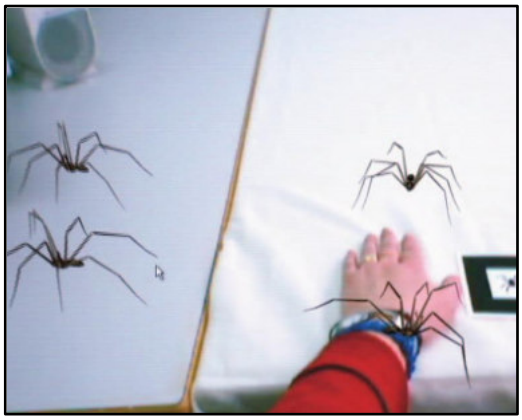


Figura 2.8. RVA para tratamento de fobia de insetos. Fonte: Juan *et al.* (2005)

Para promoção à saúde e condicionamento físico uma série de jogos tem sido lançada. Exemplos são os jogos *MindHabits*, cujo objetivo é permitir ao jogador realizar exercícios e atividades para manter atitudes positivas e diminuir o estresse no dia-a-dia (www.mindhabits.com), e *The Incredible Adventures of the Amazing Food Detective* (http://members.kaiserpermanente.org/redirects/landingpages/afd/), para ensinar e estimular bons hábitos alimentares. Outras iniciativas abordam ambientes tridimensionais, como o jogo *Ace's Adventures* (Figura 2.9) que visa condicionar a

realização de atitudes seguras no trânsito (www.rtassoc.com/gm_aces.html). Este é o propósito é similar ao do jogo S.A.F.E., que ensina segurança no trabalho e como proceder em situações emergenciais [ProcessIT, 2007] (Figura 2.10). Nestas aplicações, o objetivo é apenas um pretexto para fazer com que o usuário seja estimulado a realizar uma determinada ação, sendo ele o detentor de uma determinada necessidade a ser entendida.

Figura 2.9. *Serious game* para aquisição de boas atitudes como pedestre no trânsito.

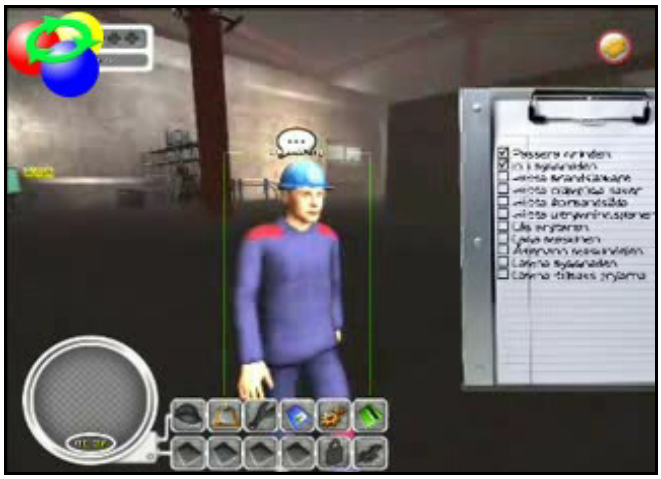


Figura 2.10. *Serious game* para segurança no trabalho.

Mais recentemente, a plataforma WiiFit tornou-se popularmente conhecida por permitir a realização de exercícios físicos de forma prazerosa e divertida usando um console de videogame [Hamilton, 2008]. Outro jogo visa estimular pessoas idosas a realizar exercícios através da dança e utiliza um tapete especial que mostra a posição dos passos a serem feitos e os reconhece para fins de pontuação (http://www.humanagames.com/) (Figura 2.11).

Figura 2.11. Exercícios físicos realizados em um *serious game* para dança que utiliza um tapete especial como plataforma.

Suhonen *et al.* (2008) apresentaram um estudo de caso com adolescentes usando *serious games* para desenvolver cuidados com a saúde. A finalidade do estudo era avaliar o componente social dos *sérios games* relacionados ao assunto. Para isso, foram selecionados quatro jogos com enredos dentro do contexto de cuidados com a saúde, sendo dois deles executados via Internet, um terceiro executado em dispositivos móveis e o último executado em formato de *exergame*, isto é, um jogo com o objetivo de fazer com que o usuário se exercite. Os adolescentes jogaram em pares e foram observados por dois pesquisadores, que anotaram seus comportamentos. A partir dos experimentos realizados concluíram que é importante destacar a os aspectos sociais, principalmente em relação ao desenvolvimento de jogos para múltiplos jogadores. Concluíram ainda que os jogos devem oferecer desafios de acordo com a competência dos jogadores, se possível devem prever a sua utilização em grupos e serem capazes de oferecer o desenvolvimento de habilidades.

O monitoramento da saúde vem ganhando destaque a partir da disponibilização de marcadores biológicos. Tais marcadores (sensores) têm sido utilizados para observar o aumento da atividade cerebral nas pessoas quando expostas aos jogos [Sawyer, 2008]. Esta linha tem sido mais voltada para observar o efeito dos jogos sobre as pessoas. Um estudo mostrou que crianças têm melhor recuperação e sentem menos dor quando são submetidas a uma terapia de jogos, ou seja, quando utilizam jogos durante o seu processo de recuperação [Das *et al.*, 2005].

O uso dos *serious games* para treinamento e simulação tem provavelmente a abordagem mais promissora no contexto da RVA. Devido às limitações encontradas no treinamento de procedimentos, o uso de aplicações desta natureza é capaz de prover meios efetivos de treinamento por meio da reprodução de situações reais. Um sistema de treinamento para acupuntura usa um humano virtual que considera a posição e a profundidade de cada um dos pontos de contato considerados no tratamento de acupuntura. Usando um dispositivo específico com sensores para simular a agulha real, o sistema fornece um julgamento dos procedimentos executados em tempo real [Kanehira e Shoda, 2007]. Afirmando que o objetivo da simulação é fornecer treinamento realista para aumentar a difusão de procedimentos inovadores e menos invasivos enquanto diminui a curva de aprendizado do cirurgião, Delinguette e Ayache (2005) simularam cirurgia hepática minimamente invasiva, disponibilizando módulos para planejamento cirúrgico e simulação dos movimentos necessários durante o procedimento. O sistema destaca-se pelo realismo oferecido para os procedimentos de deformação dos objetos flexíveis quando ocorre uma interação do usuário.

Um jogo para ensinar os passos de uma cirurgia cardíaca foi desenvolvido e está disponível a partir do sítio http://www.abc.net.au/science/lcs/heart.htm. Apesar de não se tratar de uma aplicação imersiva, o usuário pode aprender as ferramentas e passos envolvidos neste tipo de procedimento, além de precisar obedecer passos específicos para receber uma pontuação. A mesma abordagem é utilizada pelo jogo *Open Heart* comercializado pela empresa ISM Inc. O jogo é composto por um cenário real no qual o usuário precisa se deslocar, comunicar com funcionários e realizar procedimentos médicos em um hospital para conseguir operar um paciente (Figura 2.12). Apesar de limitar os movimentos do usuário, este jogo permite interação e apresenta um cenário real com pessoas reais, o que torna mais intuitiva a aplicação. Na linha de simuladores, outro trabalho foi desenvolvido para simulação de cirurgias cardíacas e, não concebido como um jogo, visa planejar uma cirurgia em função de dados de exames de um paciente [Sorensen e Mosegaard, 2006] (Figura 2.13).

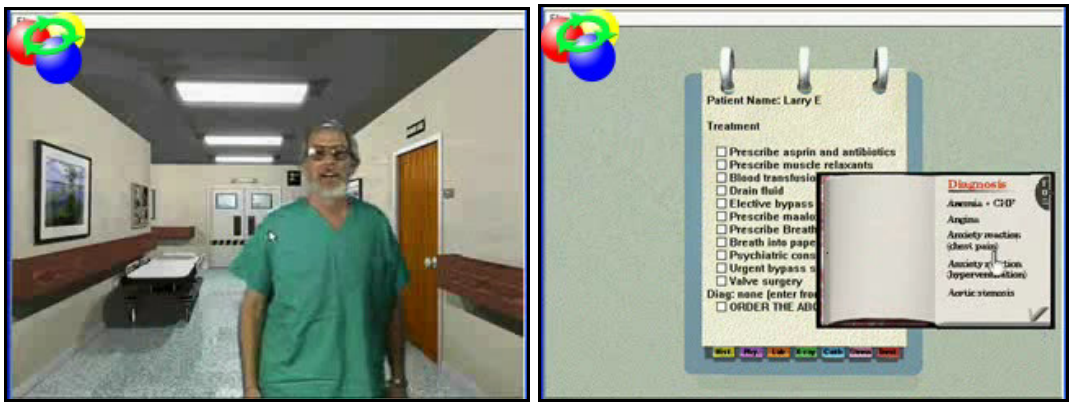


Figura 2.12. *Virtual Surgeon Open Heart*: *serious game* para prática de cirurgias cardíacas.

Visando a simulação de negócios, o jogo *Theme Hospital* surgiu no ano de 1997 como o segundo de uma série de jogos de simulação de negócios, que incluía o *Theme Park*, dentre outros. O *Theme Hospital* simulava o ambiente de um hospital, onde o usuário pode construir o seu próprio hospital, contratar médicos, enfermeiros, etc. e desenvolver o seu próprio conceito de hospital. Na medida em que o hospital vai se tornando rentável, o usuário deve investir em melhores equipamentos e pesquisar a cura de novas doenças. A Figura 2.14 mostra o ambiente de um hospital criado no Theme Hospital. Na parte inferior à esquerda da figura é possível observar o montante que o usuário dispõe para investimentos no hospital.

Na mesma linha de simuladores, um sistema para treinar cirurgia de escoliose foi apresentado por Cote *et al.* (2008), integrando um sistema biomecânico específico para cada paciente em um ambiente imersivo, colaborativo e com retorno háptico. Os autores afirmam que o sistema permite o treinamento colaborativo entre usuários remotos e a visualização das forças envolvidas no procedimento, assim como da correção resultante. Com o objetivo de simular treinamento de laparoscopia virtual, Soler *et al.* (2008) apresentaram um sistema que permite a visualização de objetos 3D reconstruídos a partir de imagens de CT ou RMN. O foco é permitir que o usuário adquira destreza manual, pois os movimentos realizados pelo médico ocorrem no sentido inverso no interior do paciente. Nos simuladores, as pinças utilizadas no

procedimento real são dispositivos hápticos que comunicam ao computador os movimentos do usuário. No ambiente de Realidade Virtual (RV), o usuário visualiza uma representação virtual das pinças, podendo manipular os tecidos e verificar os resultados de suas ações.

Figura 2.13. Sistema para treinamento e planejamento de cirurgias cardíacas baseada nos dados do paciente. Fonte: Sorensen e Mosegaard (2006)

Figura 2.14. Jogo Theme Hospital para simulação de negócios na área de saúde.

A simulação de procedimentos cirúrgicos também está sendo objeto de estudo em grupos de pesquisa brasileiros. Machado e Moraes (2009) utilizaram um sistema de coleta de medula óssea como embrião para o desenvolvimento de metodologias de avaliação do usuário, a partir de conceitos de *serious games* (Figura 2.15). Pesquisadores brasileiros dedicam-se ainda a ambientes para exames ginecológicos [Machado e Moraes, 2006], tratamento ortodôntico [Rodrigues *et al.*, 2006], e biópsias em geral (Figura 2.16) [Oliveira *et al.*, 2006].

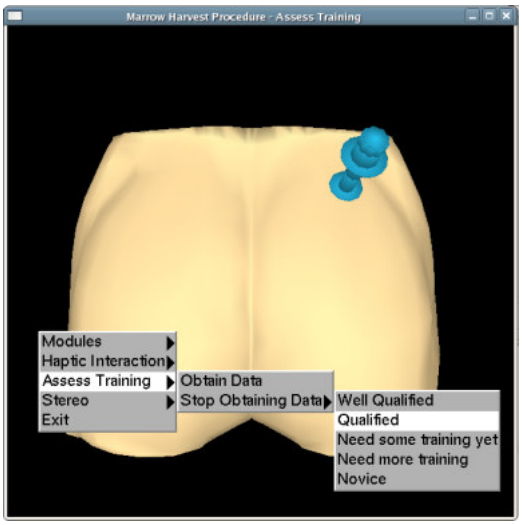


Figura 2.15. Simulador de coleta de medula óssea. Fonte: Machado e Moraes (2009)

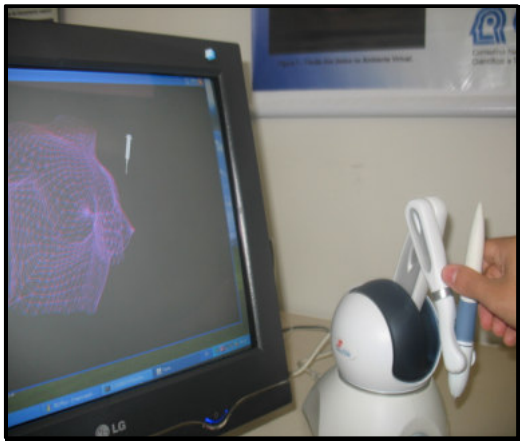


Figura 2.16. Sistema para realização de biópsia da mama.

Especificamente em relação a *serious games* aplicados à área de saúde, Marks et al. (2007) avaliaram a adequação de alguns *engines* de jogos para desenvolver aplicações para educação médica e simulação de procedimentos cirúrgicos, considerando os parâmetros de estabilidade, disponibilidade, possibilidade de customização do conteúdo e interação entre usuário via uma rede de computadores. Finalmente, Tran e Biddle (2008) discutiram a junção do envolvimento social com ferramentas técnicas para fornecer um ambiente que facilite a participação completa de profissionais com perspectivas disciplinares diversas, contribuindo com o desenvolvimento interativo dos *serious games*.

2.4. Como Criar um Jogo?

O estímulo das funções cognitivas, a motivação e a aquisição de conhecimento são elementos fundamentais em um *serious game*. Por se tratar de uma aplicação de propósito específico, o planejamento demanda o apoio de profissionais da área ao qual o conteúdo se relaciona. Estes irão auxiliar a equipe de desenvolvimento a delinear o

escopo do jogo, bem como as maneiras mais adequadas de abordar os conteúdos. Nesta etapa, a utilização de equipamentos especiais ou não-convencionais deverá ser analisada para adoção na aplicação final e exploração de seus benefícios ao longo do jogo. Deste modo, estereoscopia, sensações táteis, vibrações, projeções sobrepostas, monitoramento de movimentos e outras funcionalidades podem ser utilizados para garantir melhores resultados na simulação. Estes elementos também são fornecedores de subsídios para a elaboração do roteiro com foco no objetivo do jogo. Nos jogos eletrônicos tradicionais, este conjunto de especificações é detalhado em um documento chamado *design bible* que contém o roteiro do jogo, sua conceituação artística, detalhamentos da jogabilidade e definições da interface.

Nos *serious games* a *design bible* também é utilizada para guiar todo processo de desenvolvimento do jogo, contendo suas especificações e também o histórico de sua evolução conceitual. Do mesmo modo que nos jogos eletrônicos, nenhum desenvolvimento ou implementação deve ser iniciado sem que essa especificação esteja completamente pronta. A seguir são detalhados os elementos de uma *design bible*.

2.4.1. Roteiro

Como nos roteiros de filmes e comerciais, os roteiros dos jogos são fundamentais para o processo de criação. Além de documentar o diferencial do jogo criado em relação aos demais jogos existentes, eles devem citar claramente os elementos de entretenimento, desafios ao usuário, tipos e formas de interação (*mouse*, teclado, *joystick*, dispositivo háptico, etc.), forma de imersão a ser utilizada (visualização 2D ou 3D, monoscópica ou estereoscópica, tipo de projeção, ponto de vista: de cima, lateral ou em ângulo), pessoa do jogo (primeira pessoa ou terceira pessoa), classificação de gênero, etc. Entretanto, diferentemente dos roteiros de filmes, é importante haver espaço para a interferência do usuário no desencadeamento da história.

Particularmente, nos *serious games*, há a necessidade de preservar o aspecto lúdico, porém acrescentando-se algo mais. Como mencionado anteriormente, pode haver a necessidade de introduzir conceitos ao jogador, ou de treiná-lo para a tomada de decisões ou ainda desenvolver habilidades específicas. Em ensino, pode-se buscar simular situações onde o uso de um conhecimento seja necessário para a evolução no jogo. Em alguns casos, ensino e treinamento podem ser combinados para simular situações onde se aprende algo para ser utilizado na própria simulação momentos depois. Eventualmente, pode-se querer conscientizar o usuário sobre problemas sociais, como o tratamento a determinados grupos étnicos ou sobre a economia de água ou energia, etc. Esses conteúdos e a forma como eles serão abordados no jogo devem ser descritos em detalhes no roteiro, sempre com o foco no problema a ser abordado.

2.4.2. Conceituação Artística (*Game Design*)

A conceituação artística de um jogo é o seu projeto artístico e gráfico sobre o qual o roteiro se desenrolará. Várias possibilidades podem ser estudadas nessa fase, visto que existem várias formas possíveis de abordar o mesmo tema. Uma vez definida, essa conceituação é desenvolvida por um ou mais artistas, devido à complexidade das histórias e dos cenários. Nesta conceituação, as características dos cenários, os esboços dos personagens e a evolução da estória são desenhados em *story-boards*, deixando mais clara a forma como o usuário verá o jogo final.

Deve-se lembrar que, não menos importante, os sons que o jogo utilizará serão uma das formas de comunicação com o usuário. Formas diferentes de interação com personagens e/ou objetos facilitarão a sua identificação. Do mesmo modo, deve-se prever uma trilha sonora para o as diferentes fases do jogo e para a abertura.

A descrição das fases e os mapas gerais dos diversos cenários são fechados nesta etapa de conceituação artística. A partir disso, e com as descrições das texturas fundamentais, os artistas irão modelar computacionalmente o jogo.

2.4.3. Jogabilidade (*Game Play*)

Do mesmo modo que a conceituação artística, a jogabilidade possui várias possibilidades a serem estudadas para abordar um mesmo tema. Portanto, a jogabilidade é descrita a partir das regras do jogo e seu balanceamento (*game balancing*). Essas regras serão utilizadas pelos programadores em grande parte da implementação do jogo e também serão essenciais para a modelagem da inteligência do jogo.

Nos casos onde se faz necessário avaliar o usuário/jogador, essas regras serão levadas em conta para verificar se esse usuário conseguiu vencer os desafios do jogo de modo compatível com o esperado. Eventualmente, essa forma de avaliação pode também, em tempo real, modificar determinados desafios de modo que o usuário não possa finalizar o jogo sem que haja uma boa chance de que um determinado conceito tenha sido assimilado [Netto *et al.*, 2008].

2.4.4. Interface

A modelagem da interface divide-se em: *ingame* e *outgame*. A interface *ingame* é aquela disponibilizada durante o jogo e é responsável pela entrada de dados do jogador para a aplicação. A interface *outgame* é a forma de apresentar a introdução do jogo, sua configuração, instruções, etc.

Deve-se lembrar que a melhor interface é aquela que passa completamente despercebida para o jogador, permitindo que o mesmo possa focar-se no desenrolar da estória e das suas ações e reações. Interfaces muito elaboradas podem confundir o jogador ou chamar a atenção mais para si do que para o que é o foco principal do jogo: a interação com a estória. Do mesmo modo, uma interface complexa pode desmotivar o jogador e fazê-lo se desinteressar pelo jogo.

2.4.5. O Desenvolvimento do Jogo

Terminada a etapa de conceituação, o desenvolvimento de um jogo divide-se em dois caminhos distintos:

- Criação artística: elaboração dos elementos que serão usados para sua montagem, tais como modelos 3D, texturas, terrenos, sons e trilha sonora, dentre outros;
- Programação: implementação do motor do jogo (ou reutilização de um motor já existente) para a renderização gráfica e coordenação de tarefas, como rede para a comunicação com outros jogadores e/ou servidores e áudio para o gerenciamento de sons e trilha sonora do jogo, além da preparação para a integração dos elementos artísticos e implementação de protótipo(s).

Após a execução destas duas fases, passa-se à integração dos elementos e à montagem do primeiro protótipo completo do jogo. Essa integração pode não ser trivial e eventualmente se as fases anteriores não foram bem planejadas, vários problemas surgirão na integração. Uma vez concluída a integração, passa-se à depuração do jogo e vários testes são realizados até que todos os *bugs* sejam eliminados. Finalizada esta etapa, passa-se à documentação da versão final do jogo.

A fase seguinte é a distribuição do jogo. Em jogos eletrônicos de entretenimento, existem várias distribuidoras comerciais especializadas para a distribuição mundial de jogos. Entretanto, no caso dos *serious games*, muitas vezes essa fase não existe, pois a "encomenda" pode advir de uma empresa ou grupo delas ou mesmo uma encomenda governamental que se encarregará da distribuição do jogo entre suas repartições ou autarquias.

2.4.6. Outros Componentes

A pormenorização de alguns pontos importantes do desenvolvimento de jogos eletrônicos pouco explorados nas seções anteriores, como a inteligência e os controles sobre os personagens, merece ser abordada. Um deles é a inteligência artificial, que pode ser utilizada em dois momentos em um *serious game*: no controle e comportamento do jogo, bem como dos oponentes e aliados automatizados e sobre aspectos específicos do jogo como a avaliação do jogador. Uma conseqüência já mencionada é a possibilidade de que a partir dessa avaliação, o jogo possa se autorreconfigurar para garantir que um determinado conceito tenha sido assimilado.

A inteligência pode ficar centralizada em um subsistema do jogo, tomando decisões sobre ações específicas, ou descentralizada nos personagens, permitindo a esses ações isoladas. Eventualmente, pode-se unir as duas, diminuindo o custo computacional da inteligência centralizada e permitindo um número maior de opções ao longo do jogo.

Na grande maioria dos jogos, existem dois tipos de personagens: os personagens do jogador (*Player Character* ou simplesmente PCs) e os personagens controlados pelo computador (*NonPlayer Character* ou NPCs). No caso dos PCs, o controle em geral é do próprio jogador e o personagem não tem autonomia alguma, respondendo apenas às ações do usuário via teclado, *joystick*, *mouse* ou outro dispositivo de interação. Em alguns casos, isso pode ser configurado por meio da interface de controle do jogo, para que todos os personagens ou um grupo deles possa ter alguma autonomia e responder automaticamente a algumas situações do jogo. Em oposição, os NPCs são controlados totalmente pelo computador e possuem autonomia para responder automaticamente às situações do cenário. Como um "observador" de uma inteligência centralizada, estes podem transmitir informações a outros NPCs dentro de um raio de ação, o que aparenta uma maior inteligência do jogo, ou podem se comunicar apenas com a inteligência central que, por sua vez, analisará a situação e poderá emitir ações para outros NPCs. Pode-se, portanto, imaginar diferentes fases, em que haja diferentes tipos de resposta dos controladores sobre os NPCs.

Os controladores responsáveis por coordenar as ações tomadas pelos NPCs, podem ser específicos para diferentes propósitos (agentes). Por exemplo, pode-se ter um para locomoção do personagem no cenário e outro para conversação ou comunicação. Do mesmo modo, existem também diferentes arquiteturas para esses controladores. Eles

podem ser completamente independentes, onde cada ação é controlada de forma independente das demais ações; podem possuir uma dependência hierárquica, onde uma ação depende da existência de um contexto na hierarquia dos controladores; ou ainda possuir uma configuração em pilha, na qual as ações são tomadas apenas se uma seqüência de eventos for verificada. Esses controladores são divididos em controladores de alto-nível e de baixo nível. Os primeiros são responsáveis pelas ações mais gerais como as tomadas de decisão e coordenação de outros controladores. São responsáveis também pelo planejamento estratégico e por decisões gerais. Já os controladores de baixo-nível são responsáveis pelas ações de maior especificidade, como mecânicas, movimentação, ações de pegar e manipular, comunicação, etc.

Com respeito à avaliação do jogador, uma outra inteligência poderia monitorar as ações do jogador e verificar se as suas ações demonstram o domínio de conceitos ou técnicas necessárias para considerá-lo apto a desempenhar uma tarefa. No caso dos *serious games* para conscientização do usuário, pode-se avaliá-lo através de ações ou mesmo de interações específicas ao longo do próprio jogo (perguntas ou passatempos). Dependendo da forma como os escores dessa avaliação evoluem no decorrer do jogo, esse subsistema pode se comunicar com o sistema de inteligência central do jogo e disparar ações específicas de reconfiguração do próprio jogo, provocando uma nova execução de uma tarefa similar àquelas já realizadas ou interagir com o jogador, sempre visando garantir que um determinado conceito importante seja assimilado.

2.5. Inteligência Artificial para Jogos

A Inteligência Artificial (IA) pode estar presente em um jogo no controle de nível superior, tomando as decisões principais no desenrolar do enredo, e também no controle de nível inferior, tomando decisões particulares descentralizadas nos NPCs, na forma de agentes. Nesta subseção serão discutidas brevemente essas formas de inteligência.

2.5.1 Controle de nível superior e avaliação do jogador

Tanto no caso das inteligências centralizadas, tomadoras de decisões a partir de ações do jogo, quanto da avaliação do jogador, a decisão depende de um contexto do jogo e das ações e reações do jogador naquele contexto. Dessa forma, ambas são baseadas nos mesmos métodos que são descritos a seguir.

a) Sistemas Especialistas Baseados em Lógica Clássica

Nesse tipo de sistema especialista um sistema computacional procura realizar uma tarefa de modo similar a um ou mais especialistas da área. O conhecimento de um ou mais especialistas é armazenado em uma base de conhecimento usando uma representação baseada em lógica, como por exemplo, por regras.

As regras são formulações lógicas do tipo:

SE <condição> ENTÃO <conclusão>.

Uma regra pode ser composta por conjunções e disjunções e pode levar a mais de uma conclusão simultaneamente. Por exemplo:

SE (<condição1> OU <condição2>) E (<condição3>)

ENTÃO (<conclusão1> E <conclusão2>).

A regra acima com mais de uma conclusão é equivalente a um conjunto de regras com apenas uma conclusão:

SE (<condição1> OU <condição2>) E (<condição3>) ENTÃO <conclusão1>

SE (<condição1> OU <condição2>) E (<condição3>) ENTÃO <conclusão2>.

É necessário observar que é fundamental para o funcionamento deste tipo de sistema que haja consistência da Base de Conhecimento, ou seja, existe a necessidade de que não haja conflito entre as regras. Portanto, o sistema não pode concluir "sim" e "não" a partir das mesmas informações, o que levaria a uma inconsistência.

b) Sistemas Especialistas Baseados em Lógicas Não-Clássicas

Há casos, nos quais os atributos não podem ser medidos com a exatidão necessária, ou o próprio especialista verifica que o conhecimento não é válido em qualquer situação. Nestas situações, podem ser introduzidos coeficientes de certeza para codificar a incerteza na conclusão da regra. A informação incerta pode ser tratada através de probabilidades, crença e plausibilidade, possibilidades, entre outras.

c) Redes Neurais

Existem várias formas e tipos de Redes Neurais para diversas aplicações. Uma das mais populares é a Rede Neural *Multi-Layer Perceptron* (MLP) que é particularmente indicada para avaliação de contextos e inteligência artificial.

d) Redes Bayesianas

Como nas Redes Neurais, existem várias formas e tipos de Redes Bayesianas. A maioria delas é indicada para avaliação de contextos e inteligência artificial

2.5.2 Controle de nível inferior

No controle de nível inferior são tomadas decisões particulares e de modo descentralizado sobre os NPCs, na forma de agentes. Em geral, são utilizadas as conhecidas Máquinas de Estados Finitos e *Fuzzy* para modelar os estados sobre os quais os NPCs podem tomar decisões de forma descentralizada. Além delas, existem também as estratégias de busca para encontrar o menor caminho para controlar o direcionamento de personagens.

a) Máquinas de Estado

As Máquinas de Estados Finitos (MEF) não se enquadram propriamente na categoria de sistemas inteligentes, mas são técnicas muito utilizadas em jogos de computador. Em geral, em um grafo, aos nós associa-se um conjunto de ações a serem tomadas a cada estado da MEF, e os arcos denotam as possíveis transições de estados. As ações podem

ser implementadas diretamente na linguagem do motor do jogo, ou por *script*. O uso de *scripts* objetiva trazer maior comodidade ao processo de implementação da IA de um jogo. A desvantagem em se utilizar as MEFs é que, como os estados são finitos, o seu comportamento acaba sendo repetitivo e, portanto, previsível. Por outro lado, quanto maior o número de estados possíveis para um NPC, maior será a máquina de estados e, por conseguinte, maior será o grafo a ela associado, tornando a sua modelagem exponencialmente complexa. Portanto, neste caso, o seu processamento pode eventualmente não poder ser realizado em tempo-real, o que é fundamental para a continuidade do jogo. Em alguns casos, essas desvantagens podem ser parcialmente contornadas com o uso das Máquinas de Estados *Fuzzy* (MEFu). Nelas, os estados são modelados como conjuntos *fuzzy* com funções de pertinência, o que permite tomar decisões diferentes dependendo dos graus de pertinência dos estados envolvidos.

b) Estratégias de Busca

Formalmente, o espaço de busca é constituído por *n* nós conectados através de arcos. A cada arco pode ou não estar associado um valor, que corresponde ao custo *c* de transição de um nó a outro. A cada nó pode-se ter associado uma profundidade *p*, sendo que a mesma tem valor 0 (zero) no nó raiz e aumenta de uma unidade para um nó filho. Essas estratégias de busca podem ser aplicadas ao controle de ações dos personagens, como: ataque/defesa; simular ações com alguma inteligência ou encontrar uma trajetória entre localidades. Particularmente utilizados para o planejamento de trajetórias são os algoritmos de Busca Egoísta (*Best-First-Search*), de Dijkstra e o Algoritmo A*.

2.6. Ferramentas

O desenvolvimento de um *serious games* para a saúde exige um grupo multidisciplinar e a aplicação de conceitos particulares ao conteúdo abordado. A elaboração de interfaces e formas de comunicação usuário-sistema tem recebido particular atenção para oferecer meios efetivos de comunicação. Bibliotecas, *toolkits*, *engines* e *frameworks* podem auxiliar este desenvolvimento, sendo várias criadas com o propósito de facilitar a integração de tarefas, dispositivos e metodologias. Além das *engines* de jogos que têm sido utilizadas para o desenvolvimento dos *serious games*, bibliotecas, *toolkits* e *frameworks* de RVA também têm sido utilizados para facilitar esta tarefa. Entretanto, é importante observar que não há uma padronização nestes desenvolvimentos e os projetos atuais utilizam ferramentas bastante diferenciadas, sendo que alguns criam soluções próprias para o projeto em questão. Adicionalmente, várias das ferramentas aqui apresentadas podem ser utilizadas em conjunto para a criação de *serious games*. Para melhor organizar estas ferramentas, elas foram separadas em 4 (quatro) categorias:

- bibliotecas
- *toolkits*
- *frameworks*
- *engines*

2.6.1 Bibliotecas

As bibliotecas utilizadas nos *serious games* visam prover conjuntos de classes de suporte a tarefas de baixo-nível ou específicas de dispositivos. Mais comumente elas têm sido desenvolvidas para auxiliar o desenvolvimento de aplicações de RVA, não tendo como foco específico *serious games*. Entretanto, elas podem ser utilizadas nesta tarefa para facilitar ou abstrair o acesso a dispositivos ou métodos específicos.

Uma biblioteca para desenvolvimento na área médica é a MVL [Kuroda *et al.*, 2005] que oferece métodos para manipulação múltipla e deformação interativa, além de suporte a dispositivos hápticos. Outra biblioteca é a SSTML [Bacon *et al.*, 2006] que permite a integração de diferentes linguagens no processo de desenvolvimento e é voltada para a simulação de procedimentos cirúrgicos.

2.6.2 *Toolkits*

Dentre os *toolkits* utilizados em *serious games*, o mais conhecido é o *ARToolKit* [ARToolKit, 2009], que visa oferecer um conjunto de ferramentas para suporte a captura e sobreposição de imagens em cenários reais. Desse modo, ele oferece suporte ao rastreamento óptico e implementa técnicas de visão computacional para capturar e identificar marcadores, posicionando objetos virtuais sobre estes. Além disso, oferece alguns utilitários para configurar e testar o sistema de captura e reconhecimento dos marcadores. Em geral, as aplicações que utilizam realidade aumentada utilizam este *toolkit* ou estendem-no, criando soluções robustas. Um exemplo de aplicação desenvolvida com o ARToolKit pode ser observado na Figura 2.17.

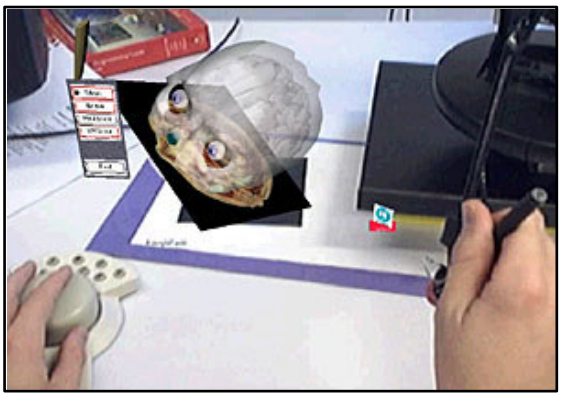


Figura 2.17. Aplicação médica usado o ARToolKit e sistemas hápticos.
Fonte: http://www.ict.csiro.au/images/NetworkTech/AugmentedReality1big.jpg

2.6.3 *Frameworks*

Os *frameworks* podem ser definidos como bibliotecas de classes que suportam funcionalidades, mas que baseiam-se em extensão e aplicam padrões de projetos. Nesta linha, dois grupos brasileiros tem trabalhado no desenvolvimento de *frameworks* para auxiliar o desenvolvimento de aplicações médicas. Devido às suas características, estes

frameworks podem ser utilizados e estendidos para o desenvolvimento de *serious games,* na área de saúde usando RVA, sendo, por este motivo, detalhados a seguir.

a) *ViMeT - Virtual Medical Training*

O *ViMeT* é um *framework* orientado a objetos que visa a fornecer um conjunto de classes que permite construir aplicações para exames de biópsia. As classes oferecem as funcionalidades de detecção de colisão com precisão, deformação, interação com equipamentos convencionais e não convencionais, estereoscopia, interface gráfica e modelagem de objetos tridimensionais [Oliveira *et al.*, 2006]. O ViMeT é implementado em linguagem Java, usando a API Java 3D, sendo que as funcionalidades citadas são implementadas como classes, conforme pode ser observado na Figura 2.18.

Há duas formas de instanciação do *framework*: utilização direta das classes apresentadas ou instanciação automática. Para esta última, uma ferramenta de apoio, denominada *ViMeTWizard*, auxilia a instanciação permitindo ao usuário a seleção de parâmetros referentes aos objetos e às funcionalidades disponíveis (Nunes *et al.*, 2007). Os parâmetros das aplicações geradas a partir da *ViMeTWizard* são armazenados em banco de dados, podendo ser posteriormente alterados para gerar novas aplicações. Na Figura 2.19 é apresentado o projeto de arquitetura do ViMeT, com as formas de instanciação citadas, as classes responsáveis pelas funcionalidades e a camada de persistência, que permite uma flexibilidade na manutenção do banco de dados.

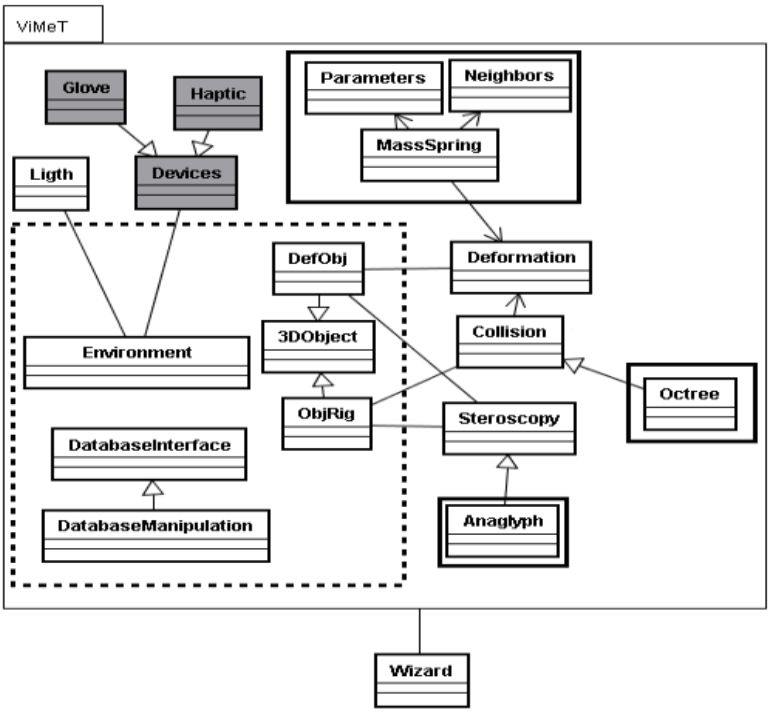


Figura 2.18. Diagrama com as classes do ViMeT. Fonte: Nunes *et al.* (2007)

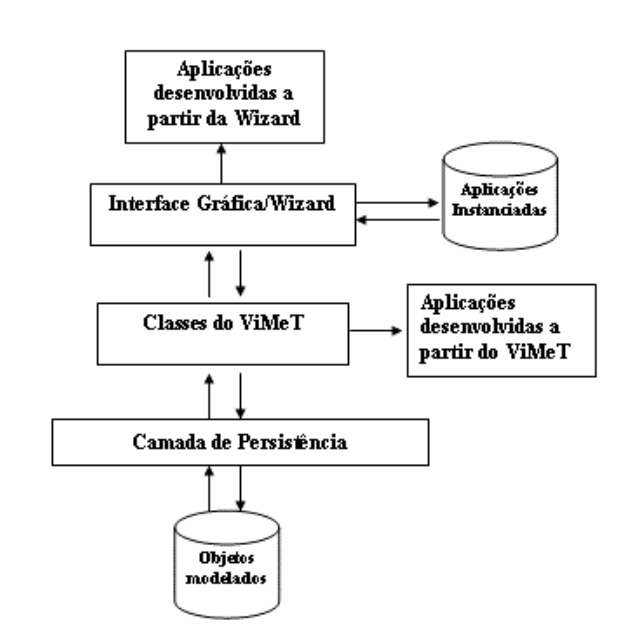
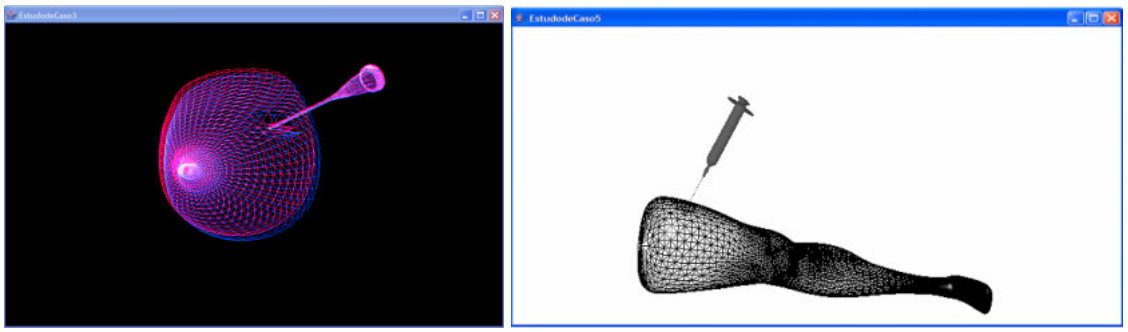


Figura 2.19. Projeto da arquitetura do ViMeT. Fonte: Oliveira *et al.* (2006)

Inicialmente, o *ViMeT* foi idealizado para facilitar a construção de aplicações que simulam exames de punção, havendo, neste caso, a necessidade de inserir dois objetos modelados: um que representa o órgão humano sobre o qual o exame será realizado, e outro que representa o instrumento que coleta material deste órgão. Assim, o desenvolvedor seleciona os objetos e determina características no que diz respeito à colisão, deformação, estereoscopia e interação. As Figuras 2.16 e 2.20 apresentam exemplos de aplicações geradas com o *ViMeT* com a importação de dois objetos: um deformável, que representa o órgão humano e outro que representa o instrumento médico.

Atualmente o *framework* está sendo expandido para contemplar maior realismo nas aplicações e considerar a geração de ferramentas de treinamento médico que ultrapassem os limites dos exames de biópsia. Uma das possibilidades é a utilização de uma quantidade maior de objetos para construir atlas virtuais de forma dinâmica e de acordo com o interesse do usuário. Um módulo de avaliação que contempla os *serious games* também está sendo planejado para oferecer auxílio mais significativo a docentes e aprendizes.



(a) **(b)**

Figura 2.20. Exemplos de aplicações geradas com o *framework* ViMeT.

b) *CyberMed*

O *CyberMed* é um *framework* livre desenvolvido para facilitar a criação de aplicações para a área de saúde [Machado *et al.*, 2009][Machado *et al.*, 2008a]. Seus componentes incluem um conjunto de classes responsáveis por tarefas específicas no contexto de uma aplicação baseada em RVA. Dentre elas destacam-se classes de geração de visualização estereoscópica, controle de deformação, detecção de colisão, interatividade, suporte a dispositivos de retorno háptico, avaliação online do usuário e sincronização de tarefas. Sua principal vantagem é a facilidade de integração de novos componentes e a possibilidade de uso individual de suas classes em outras ferramentas de desenvolvimento.

O *CyberMed* divide-se em um conjunto de três camadas: *utils*, *core* e *application engine*, que têm por finalidade prover uma série de serviços. Cada camada procura abstrair uma série de conceitos com o objetivo de facilitar a construção de aplicações baseadas em RV. A camada *Core* é responsável pelo controle dos estados internos do sistema. Algumas de suas funcionalidades estão ligadas a aquisição, cálculo, armazenamento e acesso aos dados do sistema. A aquisição de informações de modelos gráficos é feita através dos importadores de modelos (*Readers*), que podem ser estendidos com a finalidade de integrar padrões diversos de modelos ao *CyberMed*. Outra responsabilidade desta camada é o gerenciamento dos interadores que, porventura, sejam integrados ao *CyberMed*. A camada *Application Engine* tem por objetivo prover um conjunto de métodos para auxiliar o usuário na construção das aplicações. Esta camada oferece uma série de pacotes para a inclusão de visualização, colisão, rastreamento, deformação, avaliação, interação háptica e colaboração. Por fim, a camada *Utils* possui uma série de mecanismos que auxiliam o desenvolvedor em tarefas como cálculo de matrizes, operações de transformação lineares, sincronização de tarefas, etc. Entre outras funcionalidades, a camada *Utils* também possui um conjunto de métodos para a construção de menus (Figura 2.21).

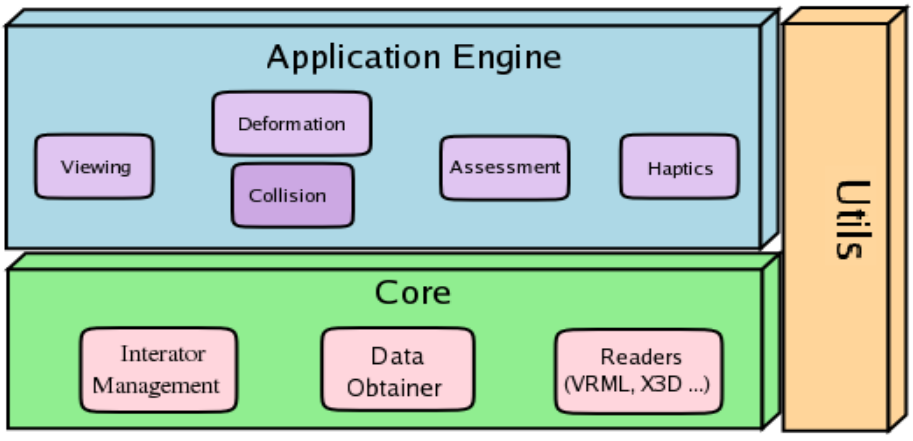


Figura 2.21. Arquitetura geral do CyberMed.

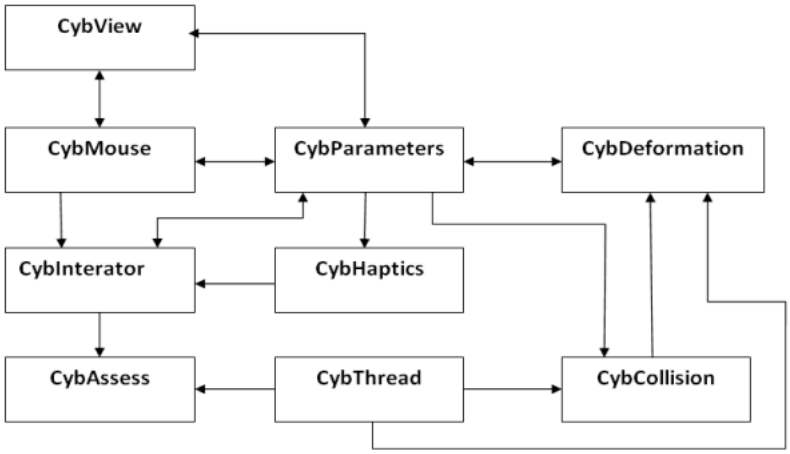


Figura 2.22. Diagrama com as principais classes do CyberMed. Fonte: Machado *et al.* (2008)

O *CyberMed* conta com uma classe fundamental para armazenamento dos dados da simulação que é utilizada para compartilhar informações entre os demais módulos. Como destaque, este *framework* apresenta uma classe de avaliação, que permite utilizar métodos de avaliação *online* do usuário, coletando e analisando seus dados de interação em tempo-real. Deste modo, o *framework* está apto a verificar as ações do usuário e emitir um escore de seu desempenho, o que favorece o desenvolvimento de *serious games*.

Na Figura 2.22 é possível observar o arranjo simplificado das principais classes do *CyberMed* para prover funcionalidades de (na figura, dispostas de cima para baixo, da esquerda para a direita): visualização, interação por dispositivos convencionais, armazenamento de modelos tridimensionais e suas propriedades, sincronização, deformação interativa, interação em tempo-real, suporte a sistemas hápticos, monitoramento e avaliação do usuário, sincronização de tarefas e detecção de colisão. As Figuras 2.15 e 2.23 apresentam aplicações desenvolvidas com o *CyberMed* para a área médica.

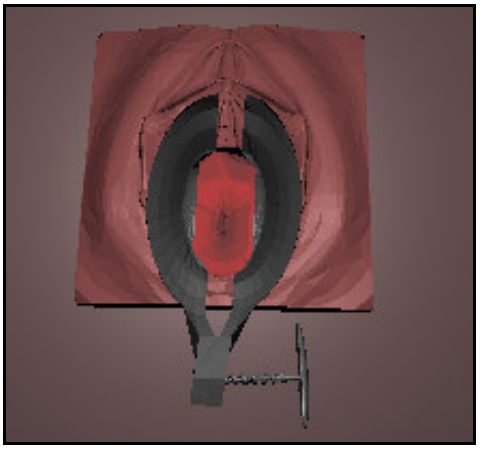


Figura 2.23. Visualização em sistema para aprendizado de exame ginecológico desenvolvido com o CyberMed. Fonte: Machado *et al.* (2008a)

c) *Outros*

Além dos *frameworks* apresentados, outros podem ser citados, tais como:

- ***Avango*** [Tramberend, 2001] – *framework* orientado a objetos que permite a criação de aplicações com classes específicas que herdam propriedades de distribuição, tendo a finalidade de auxiliar o desenvolvimento de aplicações distribuídas de ambientes virtuais (AVs) interativos. A distribuição de dados é realizada por replicação transparente de um grafo de cena compartilhado entre os processos participantes de uma aplicação distribuída, utilizando um sistema de comunicação para garantir o estado de consistência.
- ***basho*** [Hinkejann e Mannuss, 2004] – *framework* para criação de AV que apóia diferentes renderizadores e possui um núcleo pequeno com facilidade de controle e manutenção. Possui também um renderizador de grafo de cena, baseado em *OpenSceneGraph*.
- ***IVORY*** [Sprenger *et al.*, 1998)]– *framework* desenvolvido para visualização de informação baseada em física (força, massa, aceleração, potência, energia, entre outros), com classes implementadas em Java e objetos em VRML 2.
- ***SOFA*** [Allard *et al.*, 2007] – *framework* de código aberto com o objetivo principal de auxiliar na pesquisa de simulação de procedimentos médicos, permitindo criar simulações complexas e evoluir aplicações combinando novos algoritmos com os algoritmos disponíveis no *framework*.
- ***ViRAL*** (*Virtual Reality Abstraction Layer*) (Bastos *et al.*, 2004) – *framework* gráfico, baseado em componentes, desenvolvido com a linguagem de programação C++ e independente de plataforma. Consiste em uma camada interposta entre as aplicações e os sistemas de RV e que tem como objetivo principal facilitar o desenvolvimento de aplicações de RV que sejam operadas por interfaces WIMP (*Windows, Icons, Menus and Pointing Device*).
- ***VPat*** (*Virtual Patients*) (Freitas *et al.*, 2003) – *framework* que possui classes básicas para permitir o desenvolvimento de classes mais especializadas capazes de implementar algoritmos complexos de visualização e simulação de movimento e, ainda, suportar aplicações de RV.

2.6.4 *Engines*

As *engines* de jogos têm sido constantemente utilizadas no desenvolvimento de *serious games* por oferecerem gerenciamento do fluxo do código, simulação de física e/ou suporte a diferentes plataformas. Observa-se que o suporte a sistemas imersivos já está disponível em algumas delas, como a visualização estereoscópica e suporte a alguns dispositivos especiais. Entretanto, mecanismos de inteligência mais apurados, capazes de verificar aprendizado ou assimilação de conceitos, bem como gerar ações específicas relacionadas a isto, precisam ser implementadas de acordo com cada caso. O suporte a dispositivos hápticos também é pouco comum e, geralmente, aborda *joysticks* com *force feedback*.

No contexto das *engines* de jogos, a OGRE destaca-se por poder ser utilizada em diferentes plataformas com diversas configurações. Além disso, ela provê diversos *plugins* e ferramentas para gerar aplicações gráficas. Por ser de uso livre a amplamente divulgada, há uma série de jogos desenvolvidos com a OGRE. Entretanto ela não foi desenvolvida para ser uma *engine* de jogos, mas uma *rendering engine*, ou seja, ser genérica para prover gráficos a serem inseridos em aplicações diversas [OGRE, 2009]. Apesar disso, tem sido empregada no desenvolvimento de jogos.

Outra *engine* gratuita é a Panda3D que permite adicionar código escrito na linguagem de *script* Phyton. Criada pela Disney, ela oferece rotinas de renderização gráfica, suporte a estereoscopia com anaglifo e multiplexação, dentre outras funcionalidades, além de permitir ligar seu código a rotinas de C++ [Panda3D, 2009]. Esta *engine* foi utilizada no desenvolvimento de um *serious game* para ensino de geografia para o ensino fundamental. Neste jogo foi utilizada inteligência baseada em teoria das evidências, implementada em C++ com Phyton, para verificar os conhecimentos do jogador durante a busca por objetos em um campo (Figura 2.24) [Netto *et al.*, 2008]. Para o aprendizado de matemática, esta mesma *engine* permitiu criar um RPG (*roleplaying game*) em primeira pessoa em que os conhecimentos de geometria espacial adquiridos em sala-de-aula precisam ser empregados para garantir a vitória do jogador. A Figura 2.25 apresenta duas visualizações deste jogo, com o cenário de navegação e o ambiente com passatempos interativos [Morais *et al.*, 2008].

Figura 2.24. *Serious game* desenvolvido em Panda3D para apoio ao ensino de geografia. Fonte: Netto *et al.* (2008)

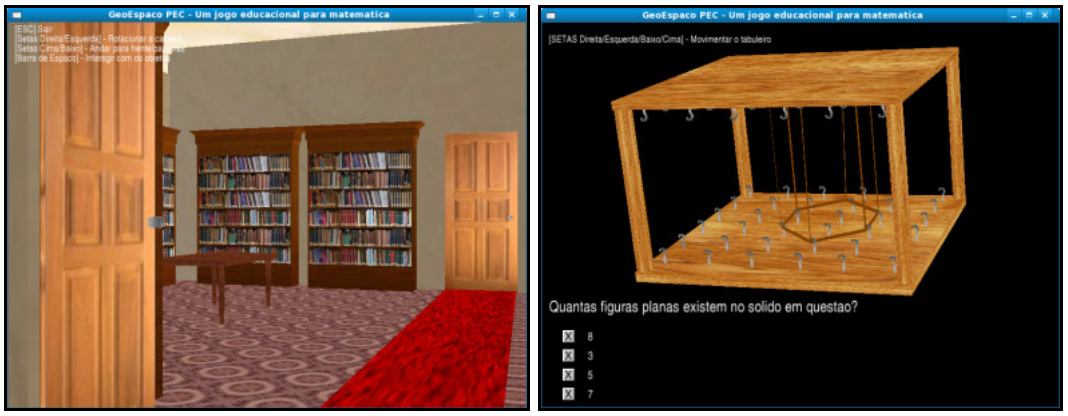


Figura 2.25. Serious game desenvolvido em Panda3D para apoio ao aprendizado de geometria espacial. Fonte: Morais et al. (2008)

2.7. Tendências

A utilização dos *serious games* tem ganhado destaque pela sua potencialidade de abrangência social. Observa-se, entretanto, que apesar das aplicações para saúde se multiplicarem e cada vez mais constituírem uma área bastante pesquisada pela comunidade de RVA, a abordagem destas na forma dos *serious games* ainda é pouco explorada, sendo mais comum em aplicações de treinamento em outros campos, como a área militar [Chatam, 2007].

O que se espera dos *serious games* no futuro próximo é sua maior inserção na sociedade. Para tanto, a criação de ferramentas que padronizem ou auxiliem seu desenvolvimento ainda precisa ser expandida. Além disso, é necessário aliar novos equipamentos a aplicações inovadoras com o objetivo de aproximar estas aplicações das situações reais [Sawyer, 2008]. Neste último caso, ressalta-se a necessidade de aplicar mais componentes biológicos nas aplicações voltadas à saúde em detrimento das simulações baseadas apenas em física.

Com a divulgação de novas aplicações e a validação destes sistemas será possível comprovar efetivamente que aprender pode ser divertido e que jogar pode ser uma atividade séria.

Referências

Allard, J.; Cotin, S.; Faure, F.; Bensoussan, P-J.; Poyer, F.; Duriez, C.; Delingette, H.; Grisoni, L. (2007) SOFA – an Open Source Framework for Medical Simulation. *Studies in Health Technologies and Informatics*, 125:13-18. IOS Press.

ARToolKit (2009) ARToolKit Home Page. Human Interface Technology Laboratory, University of Washington. Online: http://www.hitl.washington.edu/artoolkit/.

Bacon, J.; Tardella, N.; Pratt, J.; Hu, J.; English, J. (2006) The Surgical Simulation and Training Markup Language (SSTML): An XML-Based Language for Medical Simulation. *Studies in Health Technologies and Informatics*, 119: 37-42. IOS Press.

Bastos, T.A., Raposo, A.B., Gattas, M. (2005) Um Framework para o Desenvolvimento de Aplicações de Realidade Virtual Baseados em Componentes Gráficos. *Proc. XXV Congresso da Sociedade Brasileira de Computação*, pp. 213-223. SBC.

Blackman, S. (2005) Serious Games... and Less! *Computer Graphics*, 39(1):12-16. ACM.

Burdea, G. e Coiffet, P. (2003) *Virtual Reality Technology*. 2nd ed., Wiley Interscience.

Cardoso, L., Costa, R.M., Piovesan, A., Costa, M., Penna, L. (2006) Using Virtual Environments for Stroke Rehabilitation. *Proc. IEEE 5th International Workshop on Virtual Rehabilitation*, New York, p. 1-5.

Chatham, R.E. (2007) Games for Training. *ACM Communications*, 50(7): 36-43.

Cote, M.; Boulay, J.-A.; Ozell, B.; Labelle, H.; Aubin, C.-E. (2008) Virtual reality simulator for scoliosis surgery training: Transatlantic collaborative tests. *Proc. IEEE Int. Work. Haptic Audio Visual Environments and Games* (HAVE 2008), pp. 1-6 .

Das, D.A.; Grimmer, K.A.; Sparnon, A.L.; McRae, S.E.; Thomas, B.H. (2005) The efficacy of playing a virtual reality game in modulating pain for children with acute burn injuries: A randomized controlled trial. *BMC Pediatrics,* 5:1 (3 March 2005).

Delingette, H. and Ayache, N. (2005) Hepatic surgery simulation. *ACM Communications*, 48(2): 31-36.

Deutsch J.E.; Lewis J.A.; Burdea G. (2007) Technical and patient performance using a virtual reality-integrated telerehabilitation system: preliminary finding. *IEEE Trans Neural Syst Rehabil Eng*., 15(1):30-35.

Freitas, C. M. D. S., Manssour, I. H., Nedel, L. P., Gavião, J. K., Paim, M, T. C., Maciel, Â. (2003) Framework para Construção de Pacientes Virtuais: Uma aplicação em Laparoscopia Virtual. *Proc. Symposium on Virtual Reality*, pp. 283-294. Ribeirão Preto, SBC.

Gerson, N.; Sawyer, B.; Parker, J. (2006) Games and Technology: Developing Synergy. *Computer*, 39(12):129-130. IEEE.

Hamilton, A. (2008) Weighing Wii Fit: Serious Fun. *Time – Business & Tech*. Online: http://www.time.com/time/business/article/0,8599,1779642,00.html.

Hinkenjann, A., Mannuss, F. (2004) basho – A Virtual Environment Framework. *Proc. VII Symposium on Virtual Reality*, pp. 344 – 346. São Paulo.

Hodges, L.F.; Rothbaum, B.O.; Kooper, R.; Opdyke, D.; Williford, J.S.; North, M. (1995) Effectiveness of Computer-Generated (Virtual Reality) Graded Exposure in the Treatment of Acrophobia. *American Journal of Psychiatry*, 152(4): 626-628.

Johnsen, K.; Raij, A.; Stevens, A.; Lind, D. S.; Lok, B. (2007). The validity of a virtual human experience for interpersonal skills education. *Proc. SIGCHI Conf. on Human Factors in Computing Systems* (CHI '07), pp. 1049-1058. ACM.

Juan, M.C.; Alcañiz, M.; Monserrat, C.; Botella, C; Baños, R.M.; Guerrero, B.; (2005) Using Augmented Reality to Treat Phobias. *IEEE Computer Graphics and Applications*, 25(6):31-37.

Juan, M.C.; Alcañiz, M.; Calatrava, J.; Zaragoza, I.; Baños, R.; Botella, C.; (2007) An Optical See-Through Augmented Reality System for the Treatment of Phobia to Small Animals. *Lecture Notes in Computer Science*, 4563: 651-659. Springer.

Kanehira, R.; Shoda, A. (2008) Development of an Acupuncture Training System Using Virtual Reality Technology. *Proc. Fuzzy Systems and Knowledge Discovery Conference* (FSKD '08), 4:665 – 668.

Kuroda, Y.; Nakao, M.; Kuroda, T.; Oyama, H.; Yoshihara, H. (2005) MVL: Medical VR Simulation Library. *Studies in Health Technologies and Informatics*, 111: 273-279. IOS Press.

Machado, L.S.; Moraes, R.M. (2006) VR-Based Simulation for the Learning of Gynaecological Examination. *Lecture Notes in Computer Science*, 4282:97-104. Springer.

Machado, L.S.; Valdek, M.C.O.; Moraes, R.M. (2008a) On-Line Assessment System for a Training System with Multiple Stages Based on Virtual Reality. *Journal of Multiple-Valued Logic and Soft Computing*, 14(3-5):511-524.

Machado, L.S.; Souza, D.F.L; Souza, L.C.; Moraes, R.M. (2008) Desenvolvimento Rápido de Aplicações de Realidade Virtual e Aumentada Utilizando Software Livre. *Realidade Virtual e Aumentada na Prática*, pp. 5-33. Minicursos do SVR2008.

Machado, L.S.; Moraes, R.M. (2009) Qualitative and Quantitative Assessment for a VR-Based Simulator. *Studies in Health Technology and Informatics*, 142:168-173. IOSPress.

Machado, L.S.; Moraes, R.M.; Souza, D.F.L.; Souza, L.C.; Cunha, I.L.L. (2009) A Framework for Development of Virtual Reality-Based Training Simulators. *Studies in Health Technology and Informatics*, 142: 174-176. IOSPress.

Marks, S.; Windsor, J.;Wünsche, B. (2007) Evaluation of game engines for simulated surgical training. *Proc. 5th Int. Conf. on Computer Graphics and interactive Techniques in Australia and Southeast Asia* (GRAPHITE '07), pp. 273-280. ACM.

Morais, A.M.; Medeiros, D.P.S.; Machado, L.S.; Moraes, R.M.; Rego, R.G. (2008) RPG para Ensino de Geometria Espacial e o Jogo GeoEspaçoPEC. *Anais do Encontro Regional de Matemática Aplicada e Computacional.* CDROM. Natal. SBMAC.

Netto, J. C. M.; Machado, L. S.; Moraes, R. M. (2008). Teoria das Evidências Aplicada na Inteligência de um Jogo Educacional do Tipo RPG. *Anais do XVIII Simpósio Nacional de Probabilidade e Estatística (SINAPE).* Julho, São Pedro, Brasil. CDROM.

Nunes, F.L.S.; Oliveira, A.C.M.T.G.; Rossato, D.J.; Machado, M.I.C. (2007) ViMeTWizard: Uma ferramenta para instanciação de um framework de Realidade Virtual para treinamento médico. Proc. XXXIII Conf. Latinoamericana de Informática, 1:1-8. San José.

OGRE (2009) OGRE – Open Source 3D Graphics Engine. Online: http://www.ogre3d.org/.

Oliveira, A.C.M.T.G.; Pavarini, L.; Nunes, F.L.S.; Botega, L.C.; Justo, D.R.; Bezerra, A. (2006) Virtual Reality Framework for Medical Training: Implementation of a deformation class using Java. *Proc. ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. Hong Kong –China.

Paiva, J.G.S.; Cardoso, A.; Lamounier Jr., E. (2006) Interface for Virtual Automotive Route Creation in Driving Phobia Treatment. *Proc. VIII Symposium on Virtual Reality*, Belém, pp. 27-38.

Panda3D (2009) Panda3D – Free 3D Engine. Online: http://www.panda3d.org/.

ProcessIT (2007) Interactive Games Renew Safety Education. *ProcessIT Innovations Annual Report 2007*, pp.11. Online: http://www.processitinnovations.se/default.aspx?id=2067.

Rodrigues, M.A.F.; Silva, W.B.; Barbosa Neto, M.E.; Ribeiro, I.M.M.P. (2006) Um Sistema de Realidade Virtual para Tratamento Ortodôntico. *Proc. VIII Symposium on Virtual Reality*, pp. 431-444 . SBC.

Slater, M.; Pertaub, D.P.; Barker, C.; Clark, D. M. (2006) An experimental study on fear of public speaking using a virtual environment. *Cyberpsychology & Behavior*, 9(5): 627-633.

Soler, L.; Nicolau, S.; Fasquel, J.-B.; Agnus, V.; Charnoz, A.; Hostettler, A.; Moreau, J.; Forest, C.; Mutter, D.; Marescaux, J. (2008) Virtual reality and augmented reality applied to laparoscopic and notes procedures. Proc. 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI 2008). pp. 1399 – 1402.

Sorensen, T.S.; Mosegaard, J. (2006) Virtual Open Heart Surgery - Training Complex Surgical Procedures in Congenital Heart Disease. *Proc. Siggraph Emerging Technologies*. Artigo 35. ACM.

Sprenger, T.C.; Gross, M.; Bielser, D.; Strasser, T. (1998) IVORY - An Object-Oriented Framework for Physics-Based Information Visualization in Java. *Proc. IEEE Symposium on Information Visualization* (InfoViz'98), pp. 79-86.

Stone, R. (2009) Serious Games: Virtual Reality's Second Coming? *Virtual Reality*, 13(1):1-2. Springer.

Sawyer, B. (2008) From Cells to cell Processors: The Integration of Health and Video Games. *IEEE Computer Graphics and Applications*, 28(6):83-85.

Suhonen, K.; Väätäjä, H.; Virtanen, T.; Raisamo, R. (2008) Seriously fun: exploring how to combine promoting health awareness and engaging gameplay. *Proc. 12th Int. Conf. on Entertainment and Media in the Ubiquitous Era* (MindTrek '08). ACM, pp. 18-22.

Tran, M.Q.; Biddle, R. (2008) Collaboration in serious game development: a case study. *Proc. Conf. on Future Play: Research, Play, Share* (Future Play '08), pp. 49-56. ACM.

Tramberend, H. (2001) Avango: A Distributed Virtual Reality Framework. *Proc. Afrigraph '01*. ACM.

Virk, S.; McConville, K.M.V. (2006) Virtual Reality Applications in Improving Postural Control and Minimizing Falls. *Proc. 28th Annual Int. Conf. IEEE Engineering in Medicine and Biology Society* (EMBS '06). pp. 2694 - 2697.

Zyda, M. (2005) From visual simulation to virtual reality to games. *Computer* 38(9): 25-32. IEEE.

Capítulo

3

Interação com dispositivos convencionais e não convencionais utilizando integração entre linguagens de programação

Cléber G. Corrêa e Fátima L. S. Nunes

Abstract

This chapter aims at presenting a practical way of implementing interaction in Virtual Environments, considering conventional and non-conventional devices, by integrating programming languages. The integration of these devices can demand additional time in the development if some suitable concepts are not known. Publications of this research area, usually do not present details of implementing. This text intends to bridge this gap, by presenting the steps for the codification of programs using C and Java languages for joining keyboard, mouse, dataglove and haptic devices. The concepts presented can be extended for other languages and others devices.

Resumo

Este capítulo tem o objetivo de apresentar uma forma prática de implementar interação em Ambientes Virtuais, considerando dispositivos convencionais e não convencionais, por meio de integração entre linguagens de programação. A integração desses dispositivos pode demandar tempo adicional de desenvolvimento se conceitos adequados não forem conhecidos. A literatura da área geralmente não disponibiliza detalhes de implementação. Este texto pretende preencher esta lacuna, apresentando os passos para a codificação utilizando linguagens C e Java para acoplamento de teclado, mouse, luva de dados e dispositivo háptico. Os conceitos apresentados podem ser estendidos para outras linguagens e outros dispositivos.

3.1. Introdução

Um sistema é classificado como uma aplicação de Realidade Virtual (RV) se considera a coexistência de três aspectos: interação, imersão e envolvimento [Tori *et al.* 2006]. A construção de aplicações de RV deve, primeiramente, considerar os requisitos necessários para implementar adequadamente esses aspectos em função do objetivo a ser atingido. Em seguida, devem ser definidos parâmetros de investimento possível e realismo necessário a fim de que se possa obter uma relação viável entre benefícios e custos.

A interação trata da forma como o sistema deve responder a uma ação do usuário, a imersão diz respeito à sensação de presença do usuário no Ambiente Virtual (AV) e o envolvimento se preocupa com a motivação do usuário em relação ao uso do sistema. Embora essas três características estejam inter-relacionadas, cada uma delas deve observar princípios próprios.

Em aplicações de RV, a interação deve ser cuidadosamente planejada a fim de que o usuário atinja níveis de imersão e envolvimento desejados, devendo considerar tanto os objetivos a serem atingidos quanto as características e limitações do usuário. É o caso, por exemplo, de algumas aplicações de simulação para treinamento de procedimentos. Em um simulador de voo, por exemplo, a interação é um fator primordial, visto que as reações que o sistema oferece ao usuário determinam o grau de imersão e motivação que, por sua vez, determinam a continuidade da interação.

O foco do presente capítulo é a interação nas aplicações de RV. A seguir, serão adotados os termos "dispositivos convencionais" para os dispositivos comuns que estão presentes na maioria dos computadores pessoais (mouse, teclado, monitor de vídeo, por exemplo) e "dispositivos não convencionais" para aqueles equipamentos que não são comuns em tais máquinas, mas são bastante utilizados em aplicações de RV (como luvas de dados, equipamentos hápticos, óculos estereoscópicos, capacetes, entre outros).

Há várias bibliotecas e pacotes de software para a construção de aplicações de RV. Em geral, fornecem funções e procedimentos que facilitam a construção de sistemas e consideram os aspectos de interação, imersão e envolvimento. É comum tais bibliotecas disponibilizarem métodos de implementação de interação com os dispositivos chamados convencionais.

No entanto, quando se trata de interação com equipamentos não convencionais, verifica-se que a variedade de dispositivos disponíveis no mercado dificulta o estabelecimento de padrões e métodos de integração destes às aplicações. Assim, o desenvolvedor fica à mercê dos fabricantes, devendo integrar os *drivers* disponibilizados às suas aplicações ou, então, desenvolver novos *drivers*, o que constitui um esforço adicional, que demanda tempo extra no desenvolvimento de aplicações. E é aí que começam as dificuldades.

O objetivo deste capítulo é apresentar uma forma de integração de dispositivos convencionais e não convencionais em aplicações de RV, considerando os diferentes graus de realismo necessário e os recursos disponíveis, a partir da experiência obtida com o desenvolvimento de aplicações para treinamento médico [Corrêa *et al.* 2009].

Para atingir o objetivo proposto, na seção 3.2 são apresentados definições e conceitos sobre interação e dispositivos. Na seção 3.3 é apresentada a inclusão de

dispositivos em aplicações de RV, considerando a integração entre linguagens de programação Java e C, fornecendo detalhes de implementação. Na seção 3.4 são apresentados estudos de caso de tal integração, sendo oferecidos exemplos práticos e uma breve análise de desempenho. Finalmente a seção 3.5 fornece considerações finais sobre o conteúdo apresentado e, ao final do texto, são apresentadas as referências bibliográficas utilizadas.

3.2. Interação em aplicações de Realidade Virtual

Mine (1995) afirma que os Ambientes Virtuais (AVs) são considerados como a forma mais natural de interação entre homem e máquina, pois permitem que o ser humano use seus sentidos, como tato, audição e visão, de forma semelhante ao mundo real, para realizar operações, enviando e recebendo informações do computador.

A interação em AVs consiste no ato de o usuário executar uma ação com um objetivo específico, como selecionar ou manipular um objeto virtual. O AV deve retornar ao usuário uma reação em função da operação executada. Conforme destacam Pinho e Rebelo (2004), existem basicamente três componentes em uma interface homem-máquina: **dispositivo de entrada**, que captura alguma ação ou estímulo emitido pelo usuário; **função de transferência**, que tem o objetivo de mapear a ação capturada para um elemento ou elementos controlados pelo sistema e **dispositivo de saída**, que tem a finalidade de enviar uma resposta à ação executada.

Bowman *et al.* (2001a) salientam que as características dos dispositivos de entrada e saída, bem como suas limitações e custos, devem ser cuidadosamente estudados para que a interação seja adequada e o uso da aplicação seja viável considerando o contexto para o qual foi concebida. As técnicas de interação consistem em comandos e dados definidos na implementação do sistema de RV que permitem executar determinadas operações e, assim, contribuir para que o usuário se sinta motivado a utilizar o AV [Freitas *et al.* 2003].

Para que haja interação, a troca de informações entre usuário e sistema é essencial. Por isso, as características dos dispositivos de entrada e saída, bem como suas vantagens e limitações, devem ser levadas em consideração na construção de AVs.

3.2.1. Técnicas de Interação

De acordo com a literatura da área, as técnicas de interação são divididas em categorias e implementadas considerando o conceito de metáforas de interação. Pinho (2000) destaca que as técnicas de interação são responsáveis pela transferência ou mapeamento das entradas do usuário, como posições, movimentos de partes do seu corpo ou comandos, em ações dentro do ambiente tridimensional, gerando uma resposta do sistema, a qual é emitida pelos dispositivos de saída.

Bowman *et al.* (2001b) definem três categorias de operações de interação em AVs:

- **navegação**: trata do movimento do usuário dentro do AV durante uma simulação;
- **seleção/manipulação**: consiste na escolha de um objeto virtual do AV e a consequente modificação de suas características;

- **controle do sistema**: diz respeito ao estabelecimento de comandos específicos disponíveis para alterar o estado do sistema.

Especificamente em relação às formas de interação, Mine (1995) observa que diversas técnicas podem ser implementadas, classificadas em três grandes categorias:

- **interação direta do usuário:** quando uma ação do usuário (movimento de uma parte do seu corpo) resulta em uma ação no mundo virtual, usando, por exemplo, reconhecimento de gesto, apontamento, direção do olhar**;**
- **controles físicos:** envolve a utilização de botões, joysticks e outros dispositivos, com o intuito de que o usuário interaja com o AV. Nesta categoria, o tipo de dispositivo afeta a interação a ser executada;
- **controles virtuais**: um objeto do AV pode ser implementado como um controle a ser utilizado, proporcionando flexibilidade, mas podendo gerar certa dificuldade de interação com tal objeto, pois além de servir de controle, ele pode ser selecionado e manipulado.

3.2.1.1. Navegação

A navegação é a forma mais comum de interação e consiste no movimento do usuário dentro do ambiente sintético. Bowman *et al.* (2001b) dividem a navegação em três categorias:

- **exploração**: o usuário realiza uma investigação do ambiente;
- **busca**: movimentação do participante para um local específico;
- **manobras**: caracterizadas por movimentos de alta precisão.

Durante a navegação é importante estabelecer parâmetros de direção e velocidade. Mine (1995) define que controles físicos e virtuais, ou partes do corpo do usuário, podem determinar a direção para a qual o usuário deve se mover e com qual velocidade. Define, ainda, que a navegação também pode ser classificada considerando dois componentes denominados ***travel*** e ***wayfinding***.

O conceito de ***travel*** está relacionado ao movimento do ponto de vista do usuário de um lugar para outro. Existem cinco metáforas comuns para técnicas de interação do tipo ***travel***:

- **movimento físico**: o movimento do corpo do usuário é usado para movê-lo no ambiente;
- **manipulação manual do ponto de vista**: os movimentos das mãos do usuário são usados para definir o movimento deste dentro do AV;
- **apontamento**: consiste na especificação contínua da direção do movimento;
- **navegação baseada em objetivo**: o usuário especifica o destino, e sua representação no AV salta imediatamente para o novo local no ambiente;
- **planejamento de rota**: o usuário define o caminho que percorrerá no AV.

Satalich (2006) define que ***wayfinding*** é um processo dinâmico que se refere ao uso da habilidade espacial e percepção do ser humano presente em um determinado

ambiente, com o objetivo de encontrar um local neste, ou seja, o ser humano dispõe do conhecimento que já possui e adquire conhecimento durante a navegação para encontrar o caminho. Pinho *et al.* (2007) definem algumas técnicas para implementar o ***wayfinding***:

- **mapa**: indica os lugares e características do AV;
- **placas**: indicam ao usuário o objetivo, bem como outros locais dentro do AV;
- **bússola**: aponta para o objetivo, auxiliando o usuário na navegação;
- **rastro**: uma representação, como uma sequência de setas, é colocada nos lugares pelos quais o usuário passou.

3.2.1.2. Seleção/Manipulação

Esta categoria considera o ato de escolher um objeto no AV e alterar as suas propriedades. Mine (1995) lembra que é necessária a utilização de mecanismos ou técnicas para identificar um ou mais objetos que se deseja manipular, podendo destacar-se:

- **entrada de voz**: cada objeto possui um nome ou um identificador, que deve ser conhecido pelo usuário, para que ele possa identificá-lo em um grupo e selecioná-lo ao pronunciar seu nome;
- **seleção de lista**: uma lista com os nomes ou identificadores dos objetos é apresentada e o usuário pode selecioná-los utilizando técnicas diversas.

É necessário que se forneça um *feedback* ao usuário, com o intuito de confirmar o objeto selecionado, indicando que a operação de seleção realmente ocorreu. Isso pode ser feito por meio de recursos visuais, alterando uma ou mais de suas características.

Huff, Silva e Vasconcelos (2006) classificam a seleção em **local** e **a distância**. Na primeira categoria, o objeto a ser selecionado está ao alcance do usuário. Na seleção a distância o objeto não está ao alcance do participante da simulação, não havendo, portanto, um contato direto. Nesse caso, há necessidade do uso de técnicas, como apontamento, para indicar o objeto, na qual um feixe de luz é lançado da mão do usuário em direção ao objeto virtual.

Em Bowman *et al.* (2001b) são destacadas algumas técnicas de interação para seleção:

- **Go-Go**: mapeamento não linear da extensão da mão do usuário até que o objeto alvo seja alcançado;
- **ray-casting**: utilização de um raio, partindo da representação da mão do usuário (ou de outro ponto do AV), sendo utilizado para apontar um objeto no ambiente;
- **spotlight**: um cone de luz é lançado em uma determinada direção, sendo que os objetos dentro deste cone podem ser selecionados;
- **mundo em miniatura**: objetos são selecionados em uma representação em menor escala do AV, podendo ser manipulados na representação, o que é refletido no AV em escala normal.

Uma vez que o objeto está selecionado, a operação de manipulação pode ser realizada, a qual consiste na alteração de características do objeto. Segundo Hsu (2006), o ser humano pode usar suas mãos para explorar e manipular objetos no mundo real, sendo esta uma forma natural de interação com o AV. Por isso, equipamentos como luvas de dados são desejáveis, pois permitem que o movimento do usuário seja capturado e enviado ao computador por meio de sinais elétricos. Também é importante que o usuário possa visualizar uma representação de sua mão ou mãos dentro do AV.

Para manipular um objeto, modificar sua posição e rotação, controles físicos, como joysticks, e controles virtuais, como menus, também podem ser empregados. Mine (1995) lembra que um outro ponto a ser definido é o local do centro de rotação do objeto selecionado, que geralmente é aquele localizado na mão do usuário.

3.2.1.3. Controle do Sistema

O controle do sistema consiste em comandos para a comunicação com a aplicação, possibilitando alterações no seu estado ou no modo de interação [Bowman *et al.* 2001b, Flasar 2000].

As técnicas dessa categoria podem ser úteis para facilitar o processo de interação, transferindo, por exemplo, um objeto distante para próximo do usuário por meio de um comando de voz. Neste contexto, se um participante de uma simulação deseja manipular um objeto que está distante de sua área de alcance e precisa acionar uma determinada técnica de navegação para se aproximar deste objeto, o modo de interação necessita ser alterado, finalizando o processamento do evento atual (manipulação de objetos) para chamar uma função de navegação [Flasar 2000].

Considerando as propostas de Bowman *et al.* (2001b) e Flasar (2000), as técnicas de controle de sistema podem ser classificadas em quatro grupos:

- **sistema baseado em GUI (*Graphical User Interface*)**: comandos representados visualmente, como menus gráficos, que podem ser implementados no AV em uma, duas ou três dimensões;
- **comando de voz**: o sistema interpreta comandos de fala do usuário por meio de um reconhecedor de voz;
- **interação via gesto**: comandos são representados por movimentos do corpo ou partes do corpo do usuário, como o movimento dos dedos e das mãos;
- **ferramenta**: consistem em controles físicos e virtuais, como pedais e rodas de determinados dispositivos, e objetos dentro do próprio mundo virtual, que representem comandos para o sistema.

Flasar (2000) afirma que as técnicas dos quatro grupos citados podem ser combinadas, pois dependendo da situação durante o processo de interação, uma técnica pode ser mais útil do que outra.

Um resumo do conteúdo apresentado é disponibilizado nas tabelas a seguir. Na Tabela 3.1 pode-se observar as categorias que agrupam as técnicas de interação bem como suas diversas classificações. Na Tabela 3.2 são apresentadas as formas de interação que podem estar presentes em implementações de técnicas nas três categorias de interação.

Tabela 3.1. Classificação de interação em AVs.

Categoria	Classificação
Navegação	*Travel*
	Wayfinding
	Busca
	Exploração
	Manobra
Seleção	Local
	À distância
	Voz
	Lista
Manipulação	Escala (x, y, z)
	Translação (x, y, z)
	Rotação (x, y, z)
Controle do Sistema	Gesto
	Voz
	Baseado em GUI
	Ferramenta

Tabela 3.2. Classificação quanto às formas de interação.

Formas de Interação	Categorias
Direta	Navegação
	Seleção
	Manipulação
	Controle do Sistema
Controles Físicos	Navegação
	Seleção
	Manipulação
	Controle do Sistema
Controles Virtuais	Navegação
	Seleção
	Manipulação
	Controle do Sistema

A Tabela 3.3 apresenta uma classificação da navegação, que pode ser dividida em *travel* e *wayfinding*, e suas respectivas metáforas de interação e a Tabela 3.4 apresenta algumas técnicas de interação, sendo que a última delas é resultado da combinação de duas técnicas.

3.2.2. Dispositivos de interação

Os dispositivos de entrada e saída permitem ao usuário estabelecer uma comunicação com o sistema, enviando e recebendo dados do computador.

Em AVs, a interação pode ocorrer por meio de dispositivos comuns, como mouse, teclado, joystick, tela do computador, que são mais familiares aos usuários, mas também são mais limitados quando há a necessidade de executar determinadas tarefas. Como exemplo, salienta-se que a maioria dos mouses possui dois graus de liberdade, permitindo uma movimentação somente em duas direções.

Tabela 3.3. Metáforas de navegação.

Tipo	Metáforas
Travel	Movimento físico
	Baseada em objetivo
	Apontamento
	Manipulação manual do ponto de vista
	Planejamento de rota
Wayfinding	Mapa
	Placa
	Bússola
	Rastro

Tabela 3.4. Técnicas de seleção.

Categoria	Técnicas de Interação
Seleção	Go-Go
	Ray-casting
	Spotlight
	Mundo em miniatura

É possível a interação em ambientes tridimensionais não imersivos com tais dispositivos, com o usuário realizando operações de seleção, manipulação, navegação, visto que por meio de comandos do teclado ou do mouse, pode-se escolher um objeto, alterar sua posição, orientação e escala, alterar o ângulo de visão e a direção do observador.

Para uma interação mais próxima da realidade, aos sistemas de RV podem ser acoplados dispositivos não convencionais como luvas de dados, capacetes, óculos estereoscópicos, equipamentos hápticos, sensores corporais. A seguir são apresentados os mais utilizados.

O reconhecimento de gesto proporcionado por luvas de dados é uma forma de interação eficiente e altamente intuitiva para AVs [Eisenstein *et al.* 2003]. A Figura 3.1 mostra um par de luvas fabricado pela empresa *Fifth Dimension Technologies*.

Figura 3.1. 5DT Ultra Wireless Kit [5DT - *Fifth Dimension Technologies 2009*]

As luvas, geralmente constituídas de fibra sintética e sensores ópticos, procuram captar os movimentos das mãos, identificando o ângulo dos dedos e a posição e a orientação do pulso, por meio dos sensores [Thalmann 2007].

Em projetos de luvas, geralmente são utilizados sensores mecânicos ou de fibra óptica, com o intuito de captar os movimentos dos dedos. No caso da fibra óptica, um fio ou cabo óptico com junções é empregado no dispositivo para transferência de informações para o computador. Por isso, quando o usuário move seus dedos, consequentemente move as juntas, ocasionando a dobra do fio e a redução de passagem de luz, o que indica um movimento. Existem diversos tipos de luvas de dados no mercado, com variação de preço, número de sensores, desempenho e acurácia. A Tabela 3.5 apresenta algumas dessas luvas de dados, bem como seus preços e fabricantes.

Tabela 3.5. Preço de luvas de dados [EST - Engineering Systems Technologies 2009 e 5DT - *Fifth Dimension Technologies 2009*].

Nome do Produto	Fabricante	Preço
5DT *Glove* 5 *Ultra*	*Fifth Dimension Technologies*	US$ 995
5DT *Data Glove* 14 Ultra	*Fifth Dimension Technologies*	US$ 5,495
5DT *Data Glove* MRI *Series*	*Fifth Dimension Technologies*	US$ 3,495
5DT *Data Glove* 16 MRI	*Fifth Dimension Technologies*	US$ 6,995
5DT *Data Glove Ultra Wireless* Kit	*Fifth Dimension Technologies*	US$ 1,495
CyberGlove System (18 sensores)	*Immersion*	€ 13.632
CyberGlove2 System (18 sensores)	*Immersion*	€ 11.177
CyberGlove System (22 sensores)	*Immersion*	€ 19.995
CyberGlove2 System (22 sensores)	*Immersion*	€ 16.177
CyberTouch System	*Immersion*	€ 21.995
CyberGrasp System	*Immersion*	€ 59.995
CyberForce System	*Immersion*	€ 109.995

Na Figura 3.2 pode-se observar um joystick, dispositivo utilizado amplamente em jogos e em AVs, geralmente para a navegação, consistindo em um bastão a ser manipulado pelo usuário (indicando direção, ângulo) e botões para controle do jogo ou sistema.

Outros dispositivos ditos como não convencionais usados em ambientes sintéticos são os dispositivos hápticos. O termo háptico pode ser definido como a ciência do tato, envolvendo força e sensação propiciadas pelo toque [Burns *et al.* 2004].

Figura 3.2. Modelo de Joystick.

O toque consiste em uma sensação percebida quando a pele é submetida a estímulos mecânicos, elétricos, térmicos, químicos [Burdea 1996]. Esta sensação pode ser dividida em senso háptico e controle senso-motor. No que diz respeito ao senso háptico, leva-se em consideração o tato, que consiste de um conjunto de eventos que se iniciam com estímulos sobre a pele, tais como pressão ou vibração, os quais são captados por receptores (termoreceptores, receptores mecânicos, entre outros), onde descargas elétricas são geradas e transmitidas ao cérebro pelos nervos que, por sua vez, registra a sensação. Burdea (1996) afirma que o controle senso-motor está ligado ao fato de que os seres humanos combinam o senso de posição e cinestesia, ou seja, postura e movimentos, para exercer um controle motor durante uma atividade. O autor lembra que quando se discute a geração de força e a sensação tátil, alguns conceitos devem ser levados em conta, como:

- **feedback tátil:** ação aplicada à pele que indica alguma sensação;
- **feedback de força:** retorno de sensação de peso ou resistência de algo;
- **feedback cinestésico:** percepção de movimentos por órgãos existentes em músculos, tendões, juntas;
- **feedback proprioceptivo:** movimentos definidos por informações oferecidas de acordo com a postura (juntas do esqueleto).

Os dispositivos hápticos são desejáveis em aplicações de RV que determinam uma ação em função de uma força fornecida pelo usuário ou, ainda, devem retornar uma força ao usuário a partir de cálculos do sistema. A Figura 3.3 exibe o equipamento háptico PHANTOM Premium 1.5/6DOF, fabricado pela *SensAble Technologies*, que oferece 6 graus de liberdade.

Figura 3.3. PHANTOM Premium 1.5/6DOF [SensAble Technologies 2007].

Com um equipamento háptico, o usuário tem a possibilidade de utilizar o senso de toque para enviar e receber informações do computador, pois o dispositivo é movimentado pelo usuário e pode produzir um retorno de força na superfície da sua pele [Brewster *et al.* 2000]. Como exemplo de informação fornecida ao usuário durante a interação, tem-se o sentimento de textura e peso de objetos, podendo haver ou não uma integração com estímulos sonoros e visuais [Kopper *et al.* 2006].

Os equipamentos hápticos podem ser projetados como instrumentos cirúrgicos (endoscópios, agulhas, por exemplo), proporcionando aos usuários um grau mais elevado de realismo. Na Figura 3.4 é apresentada uma tesoura háptica com dois graus de liberdade, sendo um de rotação em um determinado eixo, com resolução angular de 0,056 graus e um retorno de força de no máximo 7,58 N; e o outro de translação em um determinado eixo, com resolução de 0,0152 mm e um retorno de força de 159 N [Jafry *et al.* 2003].

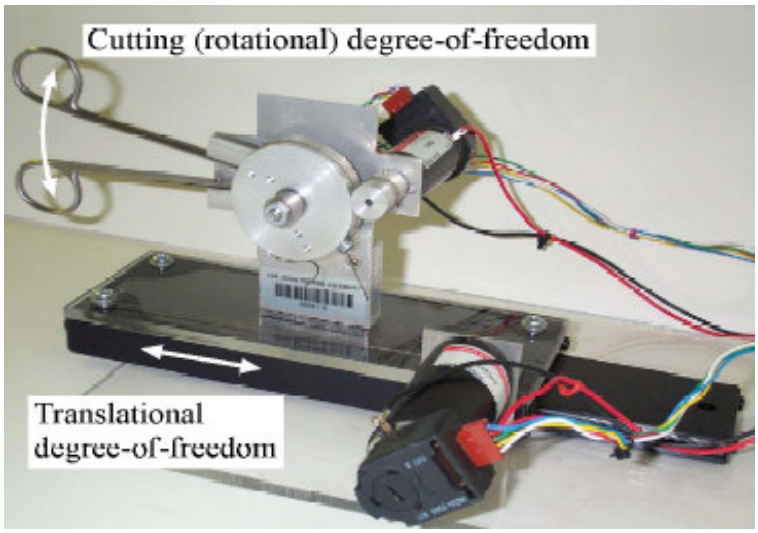


Figura 3.4. Tesoura háptica com dois graus de liberdade [Jafry *et al.* 2003].

A Tabela 3.6 apresenta preços, fabricantes e tipos de equipamentos hápticos disponíveis no mercado.

Tabela 3.6. Preço de dispositivos hápticos [EST - Engineering Systems Technologies 2009 e 5DT - Fifth Dimension Technologies 2009].

Nome do Produto	Fabricante	Preço
Haptic Master System	*Moog FCS*	€ 42.500
Delta Haptic Device 3 DOF	*Force Dimension*	€ 21.144
Delta Haptic Device 6 DOF	*Force Dimension*	€ 38.726
Omega 3 *Haptic Device*	*Force Dimension*	€ 13.856
PHANTOM *Omni*	*SensAble Technologies*	US$ 3,900
PHANTOM *Desktop*	*SensAble Technologies*	€ 10.000
PHANTOM *Premium* 1.0	*SensAble Technologies*	€ 16.773
PHANTOM *Premium* 1.5	*SensAble Technologies*	€ 22.727
PHANTOM *Premium* 1.5 HF	*SensAble Technologies*	€ 26.818
PHANTOM *Premium* 3.0	*SensAble Technologies*	€ 50.909
PHANTOM *Premium* 1.5/6DOF	*SensAble Technologies*	€ 45.455
PHANTOM *Premium* 1.5/6DOF HF	*SensAble Technologies*	€ 49.091
PHANTOM P*remium* 3.0/6DOF	*SensAble Technologies*	€ 66.636

A imersão é proporcionada por dispositivos que auxiliam na obtenção da sensação de presença, utilizando conceitos de estereoscopia. As exibições estereoscópicas permitem uma visualização de profundidade, tornando a visão mais próxima da realidade. Neste contexto, existe o conceito de que cada um dos olhos dos seres humanos vê imagens ligeiramente diferentes, as quais são interpretadas pelo cérebro, propiciando a percepção de profundidade.

Os óculos estereoscópicos, como os mostrados na Figura 3.5, geram imagens diferentes de uma mesma cena virtual para os olhos direito e esquerdo [Celes *et al.* 2004].

Os capacetes, denominados *Head Mounted Displays* (HMDs), como o mostrado na Figura 3.6, permitem uma imersão visual, combinando rastreadores de movimentos da cabeça e sistemas de geração de imagens. Oferecem uma visão estereoscópica (um display para cada olho), incluindo, ainda, um campo de visão de 360 graus, de acordo com a orientação da cabeça, e sistemas de áudio [Bernier *et al.* 2004].

Algumas limitações de capacetes incluem: peso, resolução de imagens, descompasso entre o movimento visualizado pelo olho e o percebido pelo sistema vestibular, que pode causar enjoos e náuseas; limitação na geração de imagens na área

periférica do olho; dificuldade de integração de sensores para rastrear posição de membros e corpo do usuário de maneira imersiva [Pinhanez 2004].

Figura 3.5. Óculos estereoscópicos [Absolute Technologies 2007].

Figura 3.6. 5DT *Head Mounted Display* [5DT - *Fifth Dimension Technologies* 2009]

Figura 3.7. Monitor ZScreen 2000i [5DT - Fifth Dimension Technologies 2009].

Existem diversos outros equipamentos que geram imagens, como VRD (*Virtual Retinal Display*), que exibem imagens diretamente na retina, displays autoestereoscópicos, que produzem imagens estéreos por meio de monitores LCD

(*Liquid Crystal Display*), e telas de plasma [Bernier *et al.* 2004], como o monitor mostrado na Figura 3.7, que apresenta imagens tridimensionais. Binóculos para geração de imagens estereoscópicas são mostrados na Figura 3.8.

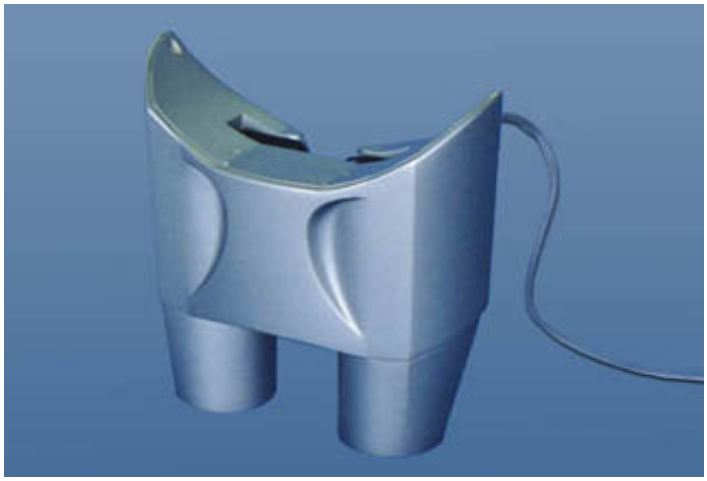


Figura 3.8. 5DT Binoculars series [5DT - Fifth Dimension Technologies 2009].

A Figura 3.9 apresenta a geração de imagens estéreos realizada por monitores LCD, fabricados pela empresa *Fifth Dimension Technologies*.

Figura 3.9. Monitores estereográficos [5DT - Fifth Dimension Technologies 2009].

A Tabela 3.7 apresenta dispositivos não convencionais disponíveis no mercado, para proporcionar visão estereoscópica, juntamente com seus preços, apresentados em moeda estrangeira, no caso, dólar americano e euro, e respectivos fabricantes.

Outro dispositivo a ser citado é o *Workbench*, que consiste em uma mesa com diversos equipamentos acoplados, como óculos, dispositivo háptico, luva de dados, rastreadores de posição, projetores, câmeras, entre outros. Nesta mesa, pode-se realizar a construção de objetos no computador por meio de objetos reais (*Workbench* perceptivo) [Leibe *et al.* 2000], ou pode-se projetar um AV que permita ao usuário interagir com o sistema. Um *Workbench* háptico visual, como o apresentado na Figura 3.10, pode ser utilizado em diversas aplicações, como: simulação cirúrgica, planejamento e treinamento, prototipagem virtual e visualização científica [Hansen *et al.* 2003].

Tabela 3.7. Preço de outros dispositivos não convencionais [EST - Engineering Systems Technologies 2009 e 5DT - Fifth Dimension Technologies 2009].

Nome do Produto	Fabricante	Preço
5DT HMD 800-26 2D	*Fifth Dimension Technologies*	US$ 2,995
5DT HMD 800-26 3D	*Fifth Dimension Technologies*	US$ 3,995
5DT HMD 800-40 3D	*Fifth Dimension Technologies*	US$ 9,995
Z800 3D Visor	*e-Magin*	€ 1.363
M920	*Icuiti*	€ 1.250
i-glasses PC/SVGA	*I-O Display System*	€ 817
i-glasses PC/SVGA PRO 3D	*I-O Display System*	€ 1.090
i-glasses VIDEO	*I-O Display System*	€ 899
Virtual Binocular SX stereo – 1280 x 1024 – 60 Hz – 40º FOV	NVIS	€ 18.091
nVisor SX - 1280 x 1024 – 60 Hz – 60º FOV	NVIS	€ 21.727
ProView XL 40STm	*Rockwell Optronics*	€ 50.000
Arvision HMD/*Goggles* – 800 x 600 – 40º FOV	*Trivisio*	€ 3.000

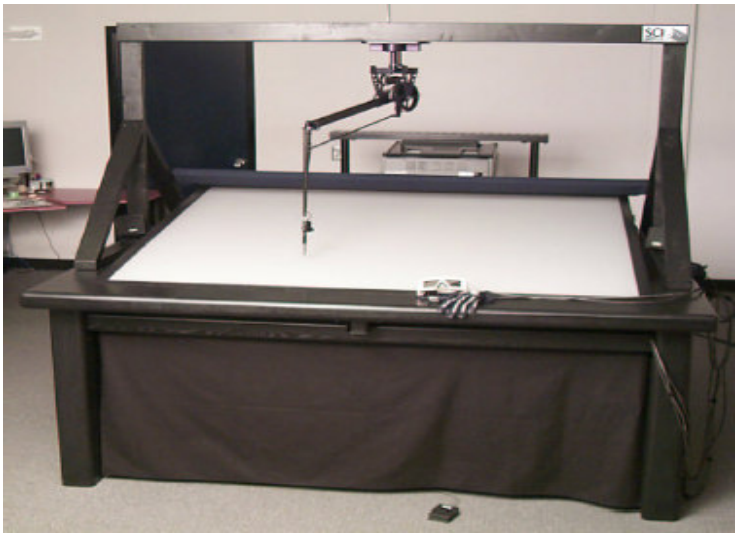


Figura 3.10. Exemplo de um Workbench Háptico Visual [Hansen *et al.* 2003].

Em alguns sistemas, devido à exaustão causada aos usuários pelo volume de dispositivos, sistemas de projeção foram desenvolvidos para construir AVs, como uma CAVE (*Cave Automatic Virtual Environment*), que consiste em um conjunto de telas

projetadas ao redor do usuário (de 2 a 6 telas), formando um cubo, onde este pode dispor de dispositivos, tais como óculos estereoscópicos [Ziegeler 2002]. No Laboratório de Sistemas Integráveis (LSI), da USP, há uma CAVE, que pode ser visualizada nas Figuras 3.11, 3.12 e 3.13, as quais apresentam a sala de projeção e a sala de controle, bem como o equipamento em funcionamento, gerando imagens nas paredes da sala.

Figura 3.11. CAVE existente na USP [LSI-USP 2009].

Figura 3.12. Sala de Controle da CAVE [LSI-USP 2009].

Figura 3.13. Projeção nas paredes [LSI-USP 2009].

Existem outras formas de projeção, como o Projetor Zscreen, mostrado na Figura 3.14, que propicia a geração de imagens tridimensionais.

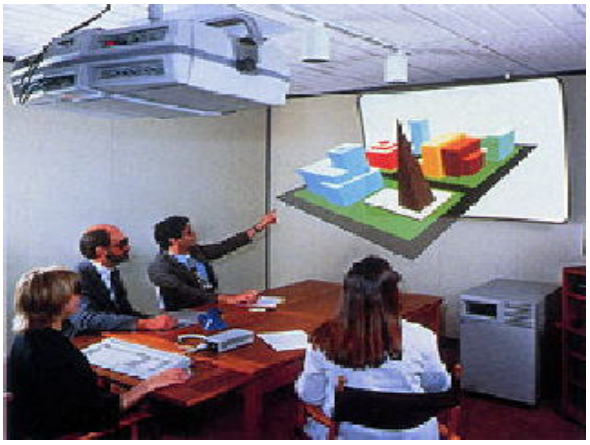


Figura 3.14. Projetor Zscreen [5DT - Fifth Dimension Technologies 2009].

3.2.3. Tecnologias de software

No mercado há várias tecnologias que fornecem recursos para a implementação de AVs, sendo que algumas delas oferecem rotinas de alto nível para facilitar a integração de dispositivos convencionais e não convencionais em aplicações de RV.

Direct3X [DirectX 2006] é uma coleção de APIs (*Application Programming Interface*) utilizada principalmente para desenvolver aplicativos 3D interativos e em tempo real. A *Open Graphics Library* (OpenGL) [Kilgard 1998] é uma API multiplataforma e multilinguagem para a implementação de aplicações capazes de produzir gráficos computacionais 2D e 3D. É um conjunto de funções que permitem aos desenvolvedores acessar os recursos gráficos sem atingir o nível de hardware. Dá suporte a iluminação, mapeamento de textura, transparência, animação, entre muitos outros efeitos especiais. A API fornece um pequeno conjunto de primitivas gráficas para construção de modelos, tais como pontos, linhas e polígonos.

WorldToolKit é uma biblioteca para o desenvolvimento de aplicações 3D de tempo real de alto desempenho. É portável e pode ser executada nas plataformas Windows e Unix [Sense8 2008]. Os objetos podem ter propriedades e comportamentos do mundo real, e estes mundos podem ser controlados por vários dispositivos de entrada. Possui funções de alto nível para configurar, interagir e controlar simulações em tempo real.

VRML (*Virtual Reality Modeling Language*) é uma linguagem independente de plataforma que permite a criação de AVs, com características de animação, movimentos de corpos e interação entre usuários [Ames *et al.* 1997]. A linguagem trabalha com geometria 3D e suporta transformações (rotação, translação, escala), texturas, luz e sombreamento. Outra característica importante da linguagem é o Nível de Detalhe (LOD, *level of detail*) que disponibiliza a quantidade certa de dados para um objeto baseado na sua importância na cena. No entanto, os recursos oferecidos para implementação de interação são limitados.

A API Java3D é uma biblioteca da Sun Microsystems Inc. [Java3D 2008], destinada à construção de aplicações e *applets* em Java. Fornece suporte a som 3D,

animação, interações, renderização paralela e otimizada. Consiste em uma hierarquia de classes que permite ao programador trabalhar com construções de alto nível para criar e manipular os objetos tridimensionais geométricos. Há classes e métodos para a implementação de *drivers* a fim de auxiliar no desenvolvimento de aplicações que utilizam dispositivos não convencionais para interação, visualização e navegação.

Os *frameworks* e bibliotecas específicas para RV oferecem recursos (geralmente um conjunto de classes em uma determinada linguagem de programação) para que funcionalidades comuns de RV, referentes principalmente à manipulação de AVs, interação e imersão possam ser reutilizadas e, assim, seja possível proporcionar um aumento de produtividade na implementação de ferramentas.

Diversos *frameworks* e bibliotecas são disponíveis na literatura, oferecendo abstração de dispositivos e sistemas de projeção, grafos de cena especializados, interação com o AV, suporte a sistemas distribuídos e renderização distribuída. Dentre os *frameworks* direcionados à construção de aplicações genéricas de RV, é interessante destacar o Avango [Tramberend 2001], o IVORY [Sprenger1998] e o ViRAL [Bastos 2005].

As bibliotecas e *frameworks,* em geral, oferecem abstração de dispositivos, com o objetivo de facilitar a integração desses em aplicações de RV. A seguir é apresentada com detalhes a integração de dispositivos em aplicações de RV usando as linguagens de programação Java e C++.

3.3. Integração de dispositivos em aplicações de Realidade Virtual

Como já enfatizado, em um sistema computacional os dispositivos de entrada e saída são responsáveis pela interface homem-máquina, provendo a comunicação entre o sistema e o usuário. Tais dispositivos permitem desta forma, a interação, um dos requisitos fundamentais em se tratando de sistemas de RV. Nos AVs podem ser adotados equipamentos convencionais e não convencionais, dependendo do grau de imersão e realismo que se deseja alcançar.

Um dos desafios encontrados na implementação desses sistemas, envolvendo a adoção de dispositivos não convencionais, é a incompatibilidade entre bibliotecas e *drivers* destes e aplicações que gerenciam a geração e a interação em AVs, visto que podem ter sido construídos com o emprego de linguagens de programação diferentes. Estas bibliotecas, com funções para facilitar a utilização dos dispositivos e os *drivers*, geralmente são oferecidas pelos fabricantes em linguagens de programação C e C++, propiciando uma programação em baixo nível. Já as aplicações, geralmente são criadas por meio de linguagens de programação e APIs diversas, que oferecem recursos de alto nível para facilitar a criação de ambientes tridimensionais.

3.3.1. Formas de integração

Diante do quadro que se apresenta, com a diversidade de linguagens de programação, duas abordagens surgem como alternativas: a construção de *drivers* próprios com a linguagem de programação usada na implementação da aplicação, bem como a criação de bibliotecas com recursos para operações com os dispositivos não convencionais; ou a integração entre as diferentes linguagens de programação, possibilitando a interface

entre a aplicação escrita em Java e as bibliotecas e *drivers* escritos em linguagem de programação C++, por exemplo.

Ambas apresentam vantagens e desvantagens. Na primeira, como desvantagens, pode-se citar: programação de baixo nível, conhecimento de diversos tipos de portas de entrada para conexão de equipamentos (USB - *Universal Serial Bus*, Serial, *FireWire*, entre outras), demanda de tempo para implementação e testes. Na segunda abordagem, as desvantagens identificadas são: diminuição da portabilidade da linguagem de programação de alto nível, conhecimento das linguagens de programação envolvidas no processo de integração, bem como das bibliotecas e funções disponibilizadas pelos fabricantes, além de recursos e técnicas disponíveis para realizar a integração.

Como vantagens para a primeira abordagem pode-se mencionar: não é necessário conhecer as duas linguagens de programação envolvidas, bem como recursos e técnicas que permitam a integração entre as linguagens adotadas, comunicação mais rápida entre aplicação e dispositivo, visto que não há uma camada adicional de software para prover a ligação entre os códigos escritos com linguagens de programação distintas e manutenção de portabilidade quando esta for uma característica da linguagem utilizada para implementar as aplicações.

Para a abordagem da integração de linguagens de programação, as vantagens observadas são: o tempo menor utilizado nos testes, não há a necessidade de conhecimento de programação de baixo nível, as bibliotecas e os *drivers* estão implementados, testados e aprovados pelos fabricantes, eliminando um relativo desperdício de tempo.

Para exemplificação da integração entre linguagens, foram adotadas as linguagens de programação Java e C++, as quais foram integradas por meio de recursos oferecidos pelo JDK (*Java Development Kit*). O processo de integração é descrito detalhadamente na próxima seção.

3.3.2. Integrando as linguagens de programação Java e C++

Para a integração das linguagens de programação se faz necessária a adoção de recursos, ferramentas e técnicas. Um dos recursos disponíveis é a interface de programação nativa, denominada JNI (*Java Native Interface*), que permite a interoperação entre aplicações escritas em Java e programas e bibliotecas escritos em outras linguagens, como *Assembly*, C e C++ [Cornell e Horstmann 2003]. O diagrama da Figura 3.15 apresenta as etapas de integração das linguagens de programação Java e C/C++ a serem descritas nesta seção.

Desta forma, pode-se escrever métodos denominados nativos usando a linguagem de programação Java, que acessam funções escritas em linguagem de programação C++, as quais constituem o chamado código nativo. Para a definição dos métodos nativos, no código escrito em Java, deve-se inserir a palavra-chave *native* em tais métodos, indicando à Máquina Virtual Java que eles serão executados por funções nativas. A Figura 3.16 apresenta a construção de uma classe simples em Java para acessar uma função escrita em linguagem de programação C++, na qual pode-se observar na linha 4 o método nativo *exibeFrase()*.

Pode-se notar também a presença do método *loadLibrary*, que indica ao Sistema Operacional a presença de uma biblioteca de ligação (*Biblioteca_teste1*), colocada como

parâmetro do método, a qual será gerada posteriormente e é responsável pela comunicação entre os códigos Java e C++ (linha 8). Por último, a linha 13 da Figura 3.16 apresenta a instanciação da classe, criando um objeto para execução do método nativo (*new ClasseNativateste1().exibeFrase()*). Continuando, o programa em Java deve ser compilado, gerando um arquivo *.class*, que será usado para a geração do arquivo de cabeçalho, cuja extensão é *.h*. No JDK há uma ferramenta denominada *javah*, que permite a criação do arquivo de cabeçalho por meio de uma linha de comando no *prompt*. O comando para o exemplo é descrito da seguinte forma: *javah –jni ClasseNativateste1*, especificando a ferramenta, a interface nativa e a classe escrita em Java. Um exemplo de arquivo gerado é apresentado na Figura 3.17 [Cornell e Horstmann 2003].

No caso da classe pertencer a um pacote, algumas alterações devem ser realizadas no comando, e a descrição passa ser a seguinte: *javah –jni –classpath . **Pacote**.ClasseNativateste1.java*. Tal comando deve ser executado no diretório pai do diretório onde está a classe com os métodos nativos. O nome do arquivo de cabeçalho é formado com a junção do nome do pacote com a classe, podendo ser renomeado, se necessário.

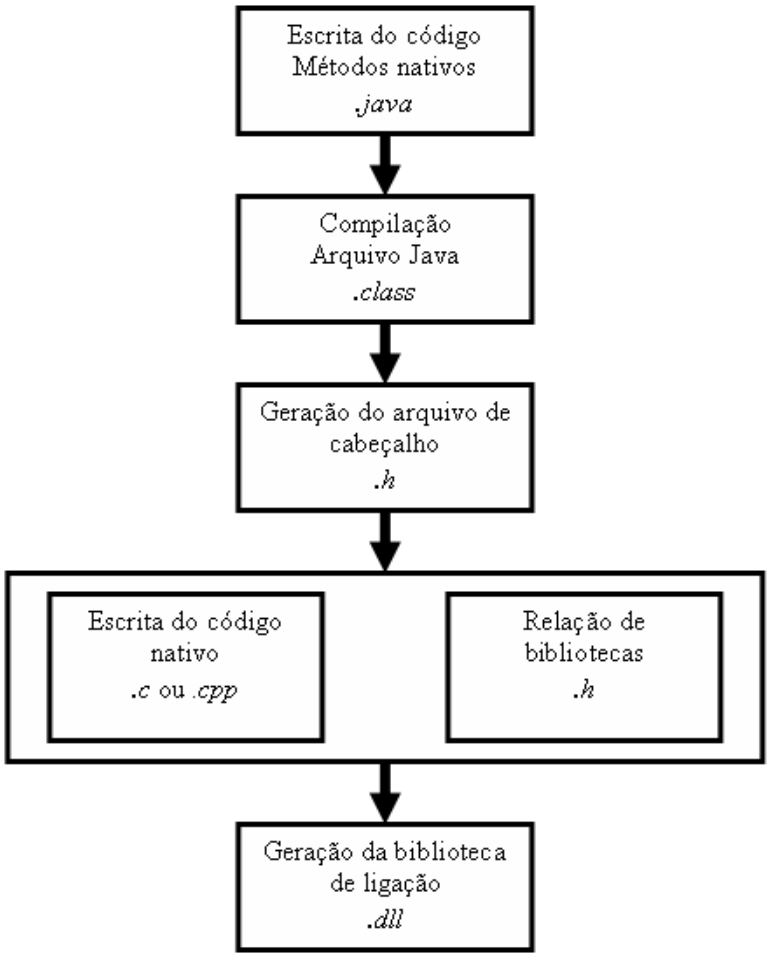


Figura 3.15. Etapas de integração das linguagens Java e C/C++.

```
0001
0002class ClasseNativateste1
0003{
0004    public native void exibeFrase();
0005
0006    static
0007    {
0008        System.loadLibrary("Biblioteca_teste1");
0009    }
0010
0011    public static void main (String[] args)
0012    {
0013        new ClasseNativateste1().exibeFrase();
0014    }
0015}
```

Figura 3.16. Classe nativa do exemplo 1.

```
0001/* DO NOT EDIT THIS FILE - it is machine generated */
0002#include <jni.h>
0003/* Header for class ClasseNativateste1 */
0004
0005#ifndef _Included_ClasseNativateste1
0006#define _Included_ClasseNativateste1
0007#ifdef __cplusplus
0008extern "C" {
0009#endif
0010/*
0011 * Class:     ClasseNativateste1
0012 * Method:    exibeFrase
0013 * Signature: ()V
0014 */
0015JNIEXPORT void JNICALL Java_ClasseNativateste1_exibeFrase
0016  (JNIEnv *, jobject);
0017
0018#ifdef __cplusplus
0019}
0020#endif
0021#endif
0022
```

Figura 3.17. Arquivo de cabeçalho para o exemplo 1.

No código da Figura 3.17, pode-se perceber na linha 2, a inclusão do arquivo *jni.h*, que juntamente com o arquivo *jni_md.h,* ambos disponibilizados no JDK, oferecem recursos para propiciar a ligação entre os métodos JNI e as funções nativas. Em determinadas implementações, é necessária a cópia dos referidos arquivos para o diretório de armazenamento do código nativo.

Voltando ao arquivo de cabeçalho, informações sobre a classe e os métodos são geradas, sendo que no exemplo tem-se um único método, podendo-se notar a assinatura do mesmo nas linhas 15 e 16, constituído pelo prefixo Java, o nome da classe e o nome do método, separados pelo caracter *underline* (*JNIEXPORT void JNICALL Java_ClasseNativateste1_exibeFrase(JNIEnv*, jobject)*). No caso da classe fazer parte de um pacote, o nome deste deve ser inserido entre o prefixo e o nome da classe.

As *strings JNIEXPORT* e *JNICALL* são definidas no arquivo *jni.h* e especificam as chamadas e as ligações entre os métodos nativos em Java e as funções nativas. Pode-se observar também a declaração *JNIEnv*, que consiste no ponteiro da interface JNI. Este aponta para uma matriz de ponteiros e, cada um destes últimos, aponta para uma

função nativa quando há múltiplas funções. A declaração do tipo *jobject* indica a utilização de um método não estático, obtendo uma referência para o objeto [Cornell e Horstmann 2003].

A próxima etapa consiste na elaboração do código na linguagem de programação nativa adotada (para exemplificação, a linguagem de programação C++). No cabeçalho do programa, composto pelas linhas de 1 a 3, deve-se invocar, usando a diretiva de compilação *#include*, o arquivo de cabeçalho gerado na etapa anterior e o arquivo *jni.h*, além de outras bibliotecas para execução de funções necessárias ao funcionamento do programa, como o arquivo *stdio.h*, que possui funções prontas para exibição de informações (comando *printf*), dentre outras. A Figura 3.18 mostra um exemplo simples seguindo a implementação.

```
0001#include <jni.h>
0002#include "ClasseNativateste1.h"
0003#include <stdio.h>
0004
0005JNIEXPORT void JNICALL Java_ClasseNativateste1_exibeFrase
0006(JNIEnv *env, jobject obj)
0007{
0008    printf("Teste numero 1\n");
0009}
```

Figura 3.18. Código nativo para o exemplo 1.

Pode-se notar a assinatura da função nativa com a representação do ponteiro *JNIEnv* e do tipo *jobject* (linha 6). O trecho a ser executado na linguagem de programação (no caso, C++) é inserido dentro da função nativa, conforme apresentado na linha 8.

A última etapa requer a utilização de um compilador (*Microsoft Visual C++ 6.0*, por exemplo) que permita com base no código nativo, a geração da biblioteca de ligação, as chamadas DLLs (*Dynamic-Link Libraries*) nos sistemas *Windows*. O nome desta biblioteca deve ser exatamente o mesmo inserido no método *System.loadLibrary*, escrito no programa em Java, no caso do exemplo, *Biblioteca_teste1*.

Para um melhor entendimento do processo de integração via JNI, será apresentado um exemplo com passagens de informações entre os códigos Java e C++ e vice-versa. Primeiramente, deve-se construir a classe nativa, a qual é mostrada na Figura 3.19.

A classe apresentada possui apenas um método nativo, mostrado na linha 4, definido como *private*, indicando que pode apenas ser executado pelo objeto instanciado a partir da classe à qual pertence, especificado como sendo do tipo inteiro (indicando que um valor inteiro será retornado). No exemplo, tem-se também a definição de um tipo inteiro (*int resultado*) para receber o resultado da soma calculada pelo método nativo (linha 8), a criação do objeto (linha 10), a execução do método passando o valor 2 como parâmetro (linha 12), a exibição do resultado (linha 14) e o carregamento da biblioteca de ligação a ser gerada (linha 19).

```
0001
0002class ClasseNativateste2
0003{
0004    private native int obterSoma(int numero);
0005
0006    public static void main (String args[])
0007    {
0008        int resultado;
0009
0010        ClasseNativateste2 teste2 = new ClasseNativateste2();
0011
0012        resultado = teste2.obterSoma(2);
0013
0014        System.out.println("Resultado: " +resultado);
0015    }
0016
0017    static
0018    {
0019        System.loadLibrary("Biblioteca_teste2");
0020    }
0021}
```

Figura 3.19. Classe nativa do exemplo 2.

O próximo passo é a compilação do arquivo *.java*, que define a classe *ClasseNativateste2* para criação do arquivo *.class*. Em seguida, utilizando a ferramenta *javah*, gera-se o arquivo de cabeçalho (*.h*), o qual é apresentado na Figura 3.20.

```
0001/* DO NOT EDIT THIS FILE - it is machine generated */
0002#include <jni.h>
0003/* Header for class ClasseNativateste2 */
0004
0005#ifndef _Included_ClasseNativateste2
0006#define _Included_ClasseNativateste2
0007#ifdef __cplusplus
0008extern "C" {
0009#endif
0010/*
0011 * Class:     ClasseNativateste2
0012 * Method:    obterSoma
0013 * Signature: (I)I
0014 */
0015JNIEXPORT jint JNICALL Java_ClasseNativateste2_obterSoma
0016  (JNIEnv *, jobject, jint);
0017
0018#ifdef __cplusplus
0019}
0020#endif
0021#endif
```

Figura 3.20. Arquivo de cabeçalho para exemplo 2.

No arquivo gerado, pode-se notar na linha 2 a invocação do arquivo *jni.h,* informações sobre a classe e o método, além da assinatura do método nativo (linhas 15 e 16), que é do tipo *jint* e passa como parâmetro também um tipo *jint* (*JNIEXPORT **jint** JNICALL Java_ClasseNativateste2_obterSoma(JNIEnv *, jobject, **jint**)*). Os tipos Java possuem tipos correspondentes em JNI, conforme apresentado na Tabela 3.8, que mostra a maioria dos tipos [Sun 2007].

Tabela 3.8. Tipos em Java e JNI.

Tipo em Java	Tipo JNI
void	void
int	jint
byte	jbyte
char	jchar
short	jshort
boolean	jboolean
double	jdouble
float	jfloat
class	jclass
int[]	jintArray
short[]	jshortArray
float[]	jfloatArray
double[]	jdoubleArray

Na próxima etapa, deve-se construir um programa em linguagem de programação C++, que realizará a soma de acordo com o valor passado como parâmetro pelo método nativo escrito em linguagem de programação Java. A Figura 3.21 mostra o programa nativo.

```
0001#include <jni.h>
0002#include "ClasseNativateste2.h"
0003
0004JNIEXPORT jint JNICALL Java_ClasseNativateste2_obterSoma
0005(JNIEnv *env, jobject obj, jint numero)
0006{
0007    return numero + 3;
0008}
```

Figura 3.21. Código nativo para exemplo 2.

O código nativo executa a soma e retorna o valor ao método nativo escrito em linguagem de programação Java. A próxima seção apresenta uma forma detalhada de geração de bibliotecas de ligação a partir do *Microsoft Visual C++ 6.0*.

3.3.2.1. Criação de Bibliotecas de Ligação com o *Microsoft Visual C++ 6.0*

Para a criação de uma biblioteca de ligação para o ambiente Windows (DLL – *Dynamic-Link Library*) usando o *Microsoft Visual C++ 6.0*, deve-se executar as seguintes etapas:

1- Na tela do compilador, na barra de menus, clique em *File* e depois em *New*. Uma caixa de diálogo semelhante à da Figura 3.22 é apresentada, onde deve-se fornecer um nome para a biblioteca de ligação, no caso do exemplo (*Biblioteca_teste1*), especificar um diretório para armazenamento dos arquivos do projeto e na guia *Projects* escolher a opção *Win32 Dynamic-Link Library*.

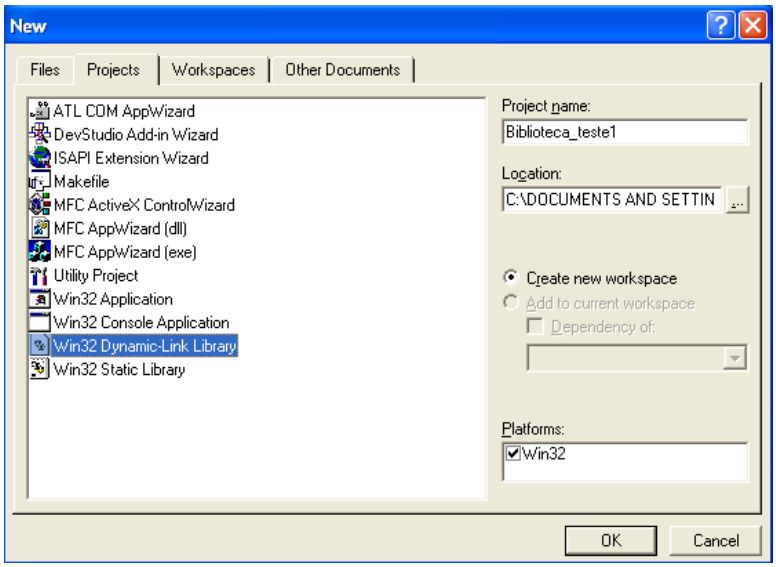


Figura 3.22. Definição de um projeto DLL.

2 – Clique no botão *OK*, em seguida, no botão *Finish* da tela que aparece na sequência com a opção *An empty DLL project* selecionada, e no botão *OK* da tela seguinte para finalizar o processo de criação. Uma pasta será criada no diretório indicado com o nome do projeto, contendo diversos arquivos e a subpasta *Debug*, na qual será armazenada a biblioteca de ligação após a geração.

3 – Defina os arquivos do projeto, código-fonte e arquivos de cabeçalhos necessários à execução do programa, o que é realizado, dentre outras formas, com os recursos oferecidos nos menus ou na parte esquerda da interface, na guia *File View*. O código-fonte pode ser escrito no próprio projeto ou utilizando um editor de texto qualquer, sendo que neste último caso, precisa ser acrescentado ao projeto clicando-se com o botão direito do mouse sobre a pasta *Source Files*, na guia *File View*, escolhendo a opção *Add Files to Folder*, para procurar e incluir o arquivo *.c* ou *.cpp*. No caso da primeira alternativa, deve-se clicar no menu *File*, em seguida *New*, escolher a guia *Files*, a opção *C++ Source Files*, fornecer um nome com a extensão e depois indicar o local de armazenamento do arquivo para poder digitar o programa na própria interface.

O arquivo de cabeçalho criado pela ferramenta *javah* pode ser adicionado ao projeto de duas formas. A primeira é clicando-se no menu *File*, em seguida *New*, escolhendo a guia *Files*, a opção *C/C++ Header Files*, fornecendo um nome e depois indicando o local de armazenamento do arquivo para poder digitar o arquivo na própria interface. A segunda é clicando-se com o botão direito do mouse sobre a pasta *Header Files*, na guia *File View*, escolhendo a opção *Add Files to Folder*, para procurar e incluir o arquivo *.h*. Por uma questão de organização, pode-se colocar na pasta do projeto, onde estão diversos arquivos criados pelo compilador, o arquivo de cabeçalho (*.h*), o arquivo escrito em C ou C++ e cópias dos arquivos *jni.h* e *jni_md.h*, encontrados no JDK, sendo que o diretório destes pode ser definido por meio de recursos do próprio compilador, eliminando a necessidade de cópia desses arquivos. A Figura 3.23 apresenta a tela com as definições desta etapa.

4 – No menu *Build* escolha a opção *Build All*, que permite a compilação do projeto, indicando possíveis erros. Se não houver erros, é gerada a DLL na subpasta *Debug* com as especificações fornecidas.

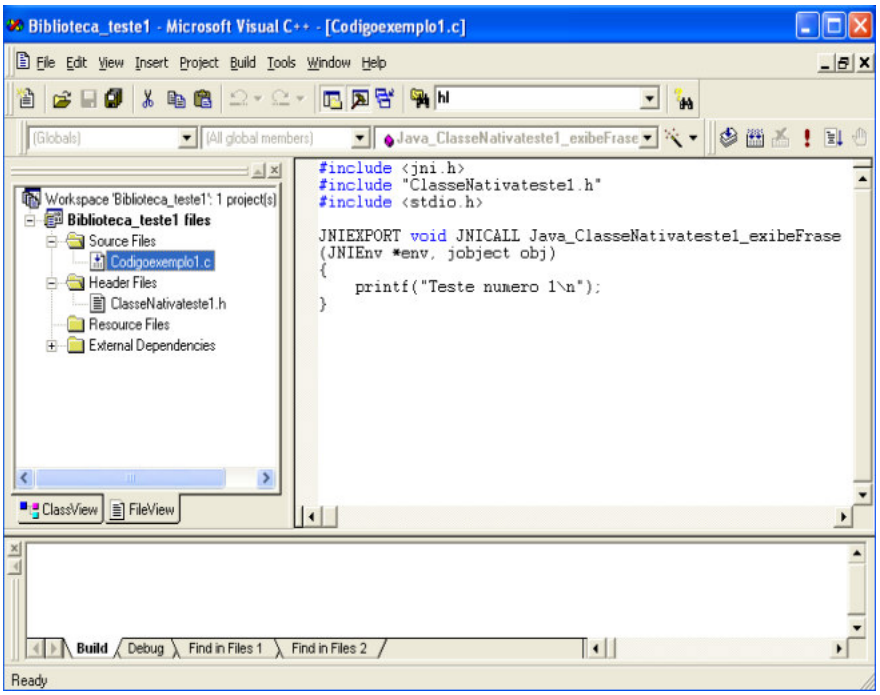


Figura 3.23. Especificações para um projeto DLL.

5 – Copie o arquivo *.dll* para o diretório onde estão os arquivos *.java* e *.class,* para que na execução do código em Java a biblioteca de ligação possa ser acionada, propiciando a comunicação entre os métodos em linguagem de programação Java e as funções em linguagens de programação C/C++.

A próxima seção trata do uso de JNI para inserir dispositivos não convencionais, que possuem *drivers* e bibliotecas construídos com linguagem de programação C++, em aplicações implementadas em Java e Java3D a partir de estudos de caso.

3.4. Estudos de Caso

Para apresentar exemplos práticos da integração descrita será usado um *framework* denominado ViMeT (*Virtual Medical Training*), que consiste um conjunto de classes com o objetivo de facilitar a geração de aplicações voltadas ao treinamento médico com baixo custo, tendo como domínio as simulações de exames de biópsia [Oliveira *et al.* 2006].

As características e funcionalidades são oferecidas por meio de classes e métodos de relevância para gerar AVs no domínio citado, implementados em linguagem de programação Java e a API Java3D, com o intuito de aproveitar determinados benefícios, tais como: portabilidade, gratuidade e facilidade para construção de um AV. As funcionalidades relevantes consideradas para simulações desta natureza são: criação de um AV dinâmico; importação e definição de características de objetos virtuais para representar um órgão humano, um instrumento médico e uma mão virtual (composta por um conjunto de objetos); deformação do objeto tridimensional que representa o órgão humano no ponto onde eventualmente ocorre a colisão entre este e o objeto que representa o instrumento médico; detecção de colisão entre os objetos; estereoscopia; suporte a dispositivos convencionais e não convencionais para propiciar a interação

humano-computador, tanto na manipulação dos objetos virtuais quanto no controle do sistema.

As Figuras 3.24 e 3.25 apresentam exemplos das interfaces de aplicações geradas, com os objetos virtuais carregados em *wireframe*, os quais são associados aos dispositivos, sendo que o equipamento háptico movimenta o instrumento médico (representado por uma seringa) usado para simular a extração de material celular e a luva de dados movimenta os dedos da mão virtual, indicando os atos de abrir e fechar a mão do usuário para soltar e segurar a parte do corpo onde será realizada a inserção da agulha.

Todos os objetos virtuais envolvidos na simulação de exames de biópsia podem ser carregados ao mesmo tempo, conforme mostrado na Figura 3.26. Da mesma forma, dois dispositivos, ou mais, podem ser utilizados simultaneamente. Assim, os AVs são gerados de acordo com as especificações do usuário, que pode escolher os dispositivos, os objetos e as funcionalidades conforme suas necessidades.

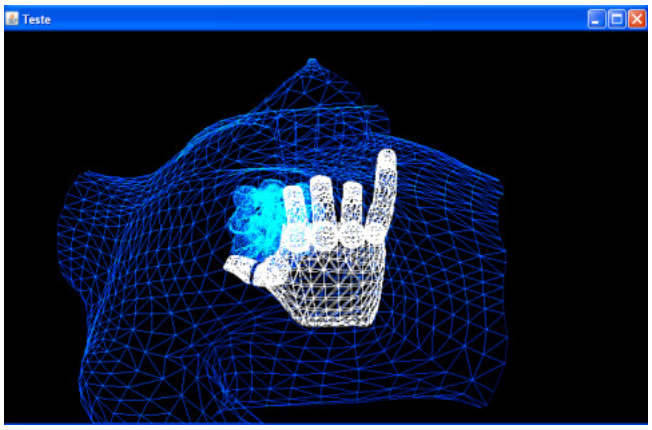


Figura 3.24. Treinamento com a mão virtual, sem a seringa.

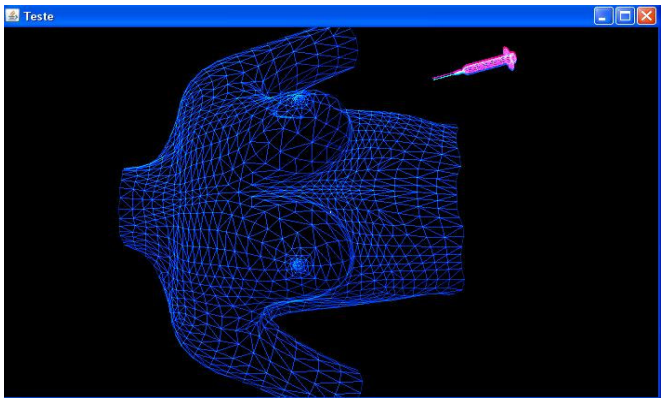


Figura 3.25. Treinamento com a seringa virtual, sem a mão virtual.

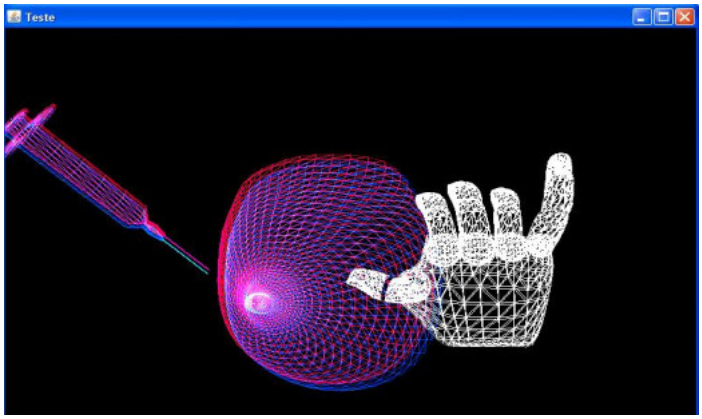


Figura 3.26. Treinamento com três objetos no AV.

3.4.1. Exemplo de integração de linguagens para aplicações de RV

Os dispositivos no ViMeT compõem o chamado módulo de interação. Os dispositivos são: teclado e mouse comuns, equipamento háptico e luva de dados. Os dois primeiros são classificados como convencionais. Os dispositivos não convencionais são um equipamento háptico PHANTOM *Omni*, apresentado na Figura 3.27 e uma luva de dados *5DT DataGlove 5 Ultra*, mostrada na Figura 3.28.

Figura 3.27. PHANTOM *Omni*
[SensAble Technologies 2007].

Figura 3.28. *DataGlove 5 Ultra*
[5DT- Fifth Dimension Technologies 2009].

Os dispositivos convencionais podem ser implementados com recursos da linguagem de programação Java e a API Java3D, entretanto, os não convencionais não oferecem até momento, *drivers* e bibliotecas escritos em linguagem de programação Java, somente em C e C++. Desta forma, o recurso JNI foi adotado para solucionar tal problema, servindo como estudo de caso para integração de dispositivos em aplicações de RV e Realidade Aumentada (RA).

No projeto, foram implementadas classes com métodos nativos para cada dispositivo não convencional, denominadas *NativeGlove* e *NativeHaptic*, além das classes *Glove* e *Haptic*, as quais são responsáveis pelo controle da interação com luva de dados e dispositivo háptico, respectivamente, conforme pode ser observado no diagrama de classes da Figura 3.29. O controle é realizado por classes estendidas da classe *Behavior*, disponibilizada pela API Java3D, implementadas nas classes *Glove* e *Haptic*, onde os métodos nativos são invocados e as informações recebidas são

transferidas para o AV, causando as reações nos objetos virtuais. Essas classes recebem o nome de *GBehavior* e *HBehavior*, respectivamente.

Para os dispositivos convencionais foram construídas as classes *Mouse* e *Keyboard*, sendo que o controle de execução do dispositivo mouse é realizado por meio de recursos da API Java3D e o controle do dispositivo teclado é feito por um classe estendida da classe *Behavior*, denominada *KBehavior*, semelhante às implementações das classes para a luva de dados e equipamento háptico, no entanto, contando também com recursos da linguagem de programação Java.

Desta forma, cada dispositivo é instanciado separadamente na aplicação, podendo ser usado simultaneamente com outro, permitindo uma combinação de dispositivos no caso da inclusão de novos equipamentos. As aplicações podem ser construídas com diversos dispositivos levando em consideração o grau de realismo desejado e a disponibilidade financeira.

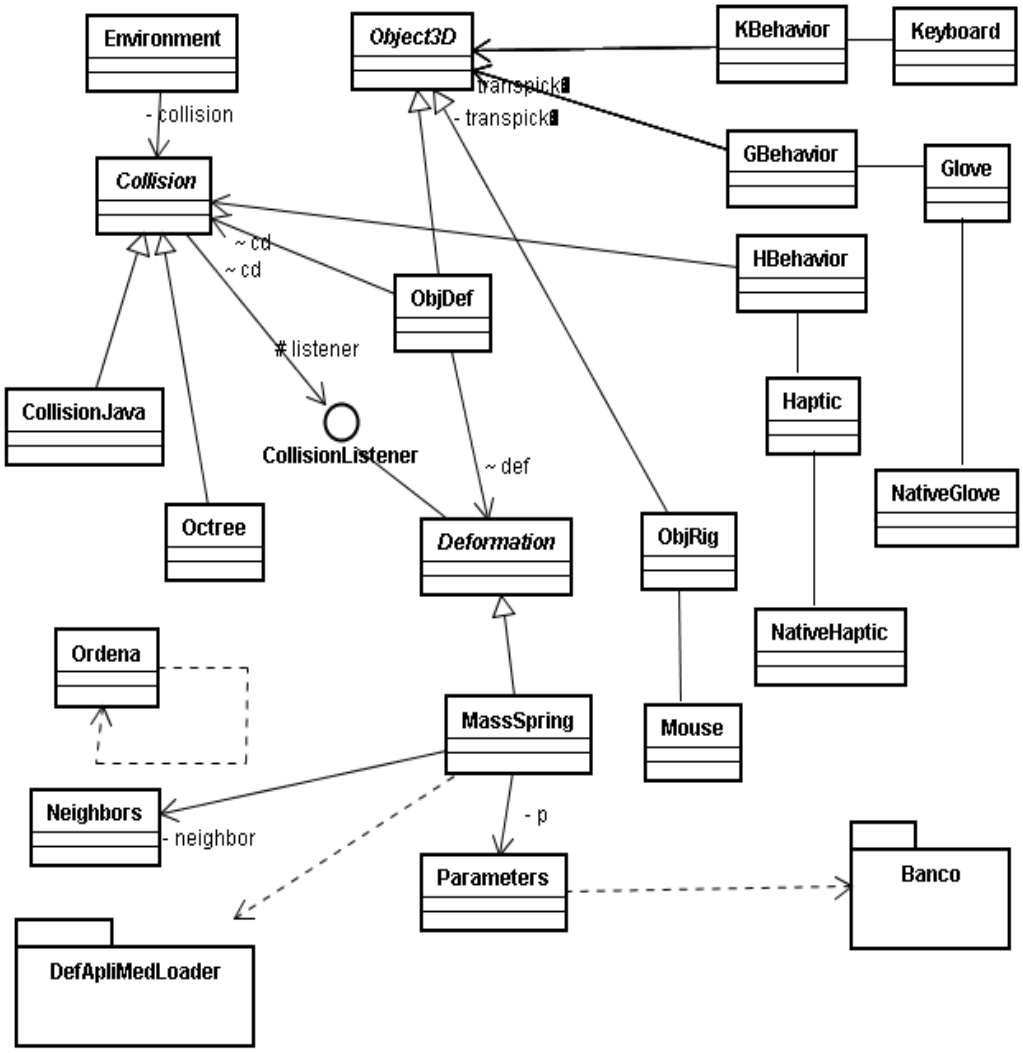


Figura 3.29. Diagrama de classes do ViMeT.

Os métodos *initialize* e *processStimulus*, provenientes da classe *Behavior*, em conjunto com a classe *WakeupOnElapsedFrames*, propiciam a geração de um laço que

verifica informações do dispositivo e realiza o mapeamento das mesmas para os objetos virtuais a cada quadro ou a cada grupo de quadros. O argumento fornecido a *WakeupOnElapsedFrames* especifica o número de quadros para execução de um bloco de código, sendo que se este valor for 0, o bloco será executado a cada quadro, conforme apresenta a Figura 3.30 (linha 15), se for 5, a cada 5 quadros.

A Figura 3.30 mostra trechos do código para a luva de dados (classe *Glove*), sendo que codificações semelhantes foram utilizadas no desenvolvimento das classes *Keyboard* e *Haptic*, para teclado e dispositivo háptico, respectivamente.

Desta forma, usando a luva de dados, quando o usuário flexiona os dedos, os objetos virtuais que representam os dedos são movimentados no AV; usando o teclado, ao pressionar ou liberar teclas específicas, os objetos virtuais que representam os dedos são movimentados, indicando o ato de segurar e soltar o órgão humano. No caso da utilização do mouse, o usuário movimenta o dispositivo e pressiona os botões deste para mover o objeto virtual que representa o instrumento médico, realizando a translação e rotação do objeto, semelhante à utilização do dispositivo háptico, empregado para mover o instrumento médico virtual (translação e rotação) e proporcionar também um retorno de força.

```
0001
0002public class Glove
0003{
0004    ...
0005
0006    private class GBehavior extends Behavior
0007    {
0008        ...
0009        WakeupOnElapsedFrames wAct;
0010        ...
0011
0012        public void initialize()
0013        {
0014            //Execução a cada frame - parâmetro 0
0015            wAct = new WakeupOnElapsedFrames(0);
0016            wakeupOn(wAct);
0017        }
0018
0019        public void processStimulus(Enumeration criteria)
0020        {
0021            //Métodos nativos, controle
0022            //Movimentação de objetos tridimensionais
0023
0024            ...
0025
0026            wakeupOn(wAct);
0027        }
0028    }
0029}
```

Figura 3.30. Partes do código da Classe *Glove*.

No caso do equipamento háptico, foram criados métodos nativos para inicializar, controlar, verificar falhas de comunicação e finalizar o uso do dispositivo. Os métodos nativos acionam funções escritas em linguagem de programação C++, oferecidas pela biblioteca do dispositivo, denominada *OpenHaptics Toolkit*. A Tabela 3.9 apresenta os principais métodos nativos da classe *NativeHaptic*.

Tabela 3.9. Principais métodos da classe *NativeHaptic*..

Métodos	Função
initializeDevice ( )	Inicializa o dispositivo
verifyCallback ( )	Verifica a execução da *callback*
finishDevice ( )	Finaliza o dispositivo
sendData (dataarray)	Envia dados usando um vetor, como o retorno de força nos eixos x, y e z
receiveData ( )	Recebe dados, como uma matriz de transformação 4x4
receiveTranslation ( )	Recebe a translação, valores nos eixos x, y e z
Button1State ( )	Verifica se o botão 1 está pressionado ou não
Button2State ( )	Verifica se o botão 2 está pressionado ou não

Também para a luva de dados foram criados métodos para inicializar, controlar, e finalizar o seu uso. A Tabela 3.10 apresenta os principais métodos nativos da classe *NativeGlove*.

Tabela 3.10. Principais métodos da classe *NativeGlove*.

Métodos	Função
openGlove( )	Abre conexão com a luva, verifica portas existentes e o número de sensores
closeGlove ( )	Fecha a comunicação com a luva de dados
getRawSensorData( )	Obtém os valores de todos os sensores
getScaledSensorData ( )	Obtém as escalas de todos os sensores
setCalibration(upper, lower)	Realiza a calibração em um intervalo de dois valores
getGesture ( )	Obtém o número do gesto
getSensorRaw (number)	Obtém o valor de um único sensor
getSensorScaled (number)	Obtém a escala de um único sensor

Para exemplificação, serão descritas nas próximas seções as implementações dos dispositivos citados adotados no *ViMeT*.

3.4.1.1. Visão geral da integração e interação com dispositivos

Como mencionado, os dispositivos convencionais foram controlados com recursos da linguagem de programação Java e a API Java3D. Desta forma, para a utilização do teclado no módulo de interação, foi utilizada uma interface disponível na linguagem de programação Java, denominada *KeyListener*, que define os eventos desse dispositivo e permite a utilização de diversos métodos, tais como: *keyPressed* (responsável pela indicação do pressionamento de uma determinada tecla); *keyReleased* (responsável pela indicação da liberação de uma determinada tecla); *keyTyped* (responsável pela indicação do pressionamento de uma tecla que não seja de ação) e *getKeyChar* (que permite o retorno de uma *string* com o nome da tecla pressionada) [Bezerra *et al.* 2008] .

Para integração do mouse foram utilizadas classes disponíveis na API Java3D, denominadas *MouseRotate* (que verifica movimentos de rotação causados pelo dispositivo), *MouseTranslate* (que verifica movimentos de translação no eixos x e y) e

MouseZoom (que verifica movimentos de translação no eixo z), sendo acionadas de acordo com o botão a ser pressionado pelo usuário.

Os dispositivos não convencionais foram implementados usando a integração de linguagens. As Figuras 3.31 e 3.32 apresentam os diagramas de integração de linguagens de programação e interação para os dispositivos não convencionais adotados até o momento. Ambos mostram os fluxos de informações entre as partes da aplicação desenvolvidas entre as diferentes linguagens de programação, e entre o usuário e os dispositivos luva de dados e háptico, representados pelas setas.

Na Figura 3.31 pode-se observar as informações da luva de dados sendo captadas por funções escritas em linguagem de programação C++, que compõem a biblioteca oferecida pelo fabricante juntamente com o *driver*. Tais informações são transferidas externamente para a parte da aplicação desenvolvida em Java e Java3D por meio de uma biblioteca de ligação, no caso, *5DTGlove.dll*. Esta parte da aplicação realiza alterações na rotação dos objetos que representam os dedos da mão virtual, apresentando os movimentos da mão do usuário no monitor de vídeo.

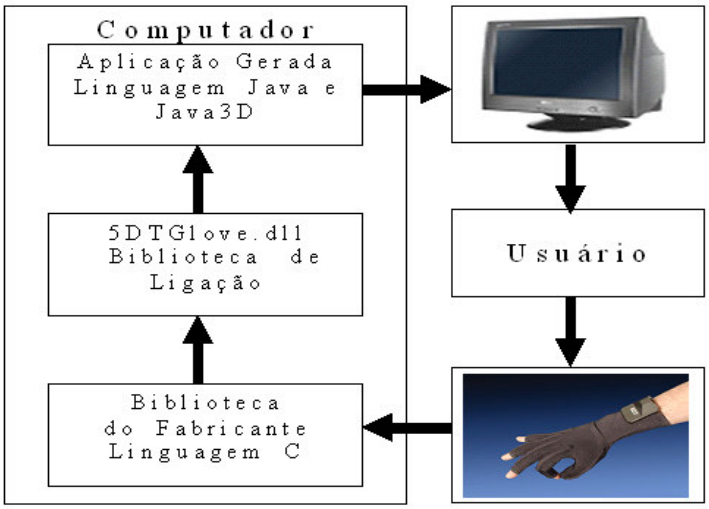


Figura 3.31. Diagrama de integração e interação para a luva de dados.

Na Figura 3.32 é apresentado um diagrama semelhante ao anterior, mas agora para descrever a integração de linguagens de programação e a interação entre usuário e computador com o dispositivo háptico. Neste caso, a biblioteca de ligação é denominada *Haptic.dll*, e permite a comunicação entre a aplicação desenvolvida em Java e Java3D, e as funções escritas em linguagem de programação C++, oferecidas pelo *OpenHaptics Toolkit*, o qual consiste em um conjunto de funções para facilitar a programação de operações com o PHANTOM *Omni*.

Pode-se observar a existência de setas em ambas as direções, diferente do diagrama anterior, indicando que a aplicação desenvolvida em Java e Java3D, recebe informações do dispositivo háptico, captadas pelas funções do *OpenHaptics Toolkit,* e também processa e envia informações ao dispositivo para promover o *feedback* tátil. Desta forma, o equipamento háptico se comporta como um dispositivo de entrada e saída.

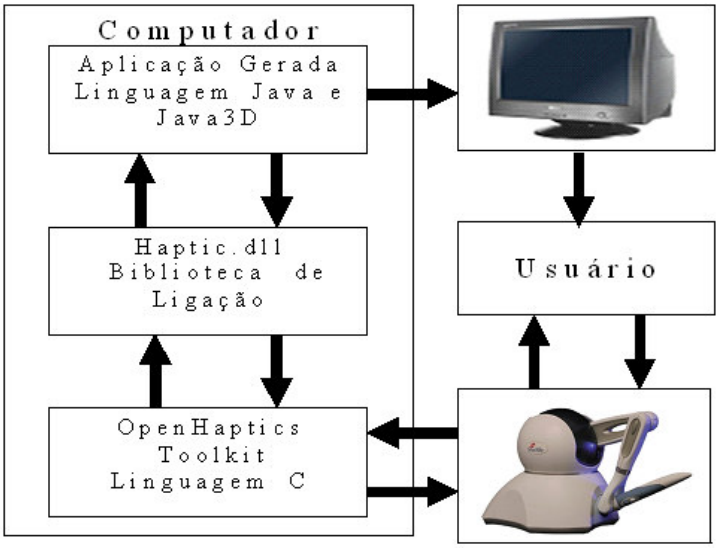


Figura 3.32. Diagrama de integração e interação para o dispositivo háptico.

A construção de duas bibliotecas de ligação deve-se ao fato da necessidade de diferentes configurações do compilador, especificadas pelos fabricantes de acordo com manuais de instruções dos equipamentos, e também para propiciar uma interação com combinações de dispositivos, visto que podem ser instanciados independentemente.

3.4.1.2. Integração da luva de dados

Conforme mencionado, os métodos nativos para a luva de dados (Tabela 3.10) foram agrupados em uma classe (*NativeGlove*) e são usados na instanciação da classe *Glove*, responsável pelo controle da interação de tal dispositivo. A Figura 3.33 apresenta a função nativa para estabelecer a comunicação com a luva de dados, podendo ser usada para outras luvas do mesmo fabricante.

A luva de dados é acionada pelo método nativo *openGlove()*, que permite o estabelecimento da comunicação com o equipamento, verificando as portas do tipo USB por meio da função *fdScanUSB*, localizando a luva de dados (linha 18), definindo o tipo de luva ou luvas de dados conectadas (DG14U_R, DG14U_L, DG5U_R e DG5U_L) e o lado ao qual a luva de dados se refere (mão direita ou mão esquerda), o que pode ser observado no trecho entre as linhas 20 e 41.

Este mesmo método aciona também a função *fdOpen*, que realiza efetivamente a comunicação com o dispositivo, armazenando-se o endereço que representa a luva de dados em um ponteiro denominado *pGlove*, definido como sendo do tipo *fdGlove* (linha 48).

Uma variável *verify* do tipo inteiro foi implementada para verificar a ocorrência de erro de inicialização para tratamento pela aplicação (linhas 48 a 56). Desta forma, se houver um erro na inicialização do equipamento, a variável *verify* recebe o valor –1; caso contrário, esta variável recebe o valor 1, indicando que operações podem ser realizadas. As operações com a luva são realizadas utilizando ponteiro *pGlove* como parâmetro nas funções. A função *fdGetNumSensors* também é empregada para a obtenção do número de sensores do dispositivo (linha 54), uma vez que o software pode ser utilizado por outras luvas de dados do mesmo fabricante.

```
0001//Função para iniciar comunicação com a luva de dados
0002JNIEXPORT jint JNICALL Java_ViMeT_NativeGlove_openGlove(JNIEnv *env, jobject obj)
0003{
0004   #ifdef WIN32
0005      szPort = "USB";
0006   #else
0007      szPort = "/dev/fglove";
0008   #endif
0009
0010   strcpy(szPortToOpen,szPort);
0011
0012   if (strcmp(szPort,"USB") == 0)
0013   {
0014      unsigned short aPID[5];
0015      int nNumFound = 5;
0016
0017      //Verifica nas portas USB Data Gloves disponíveis
0018      fdScanUSB(aPID,nNumFound);
0019
0020      for (int c = 0; c < nNumFound; c++)
0021      {
0022         printf("Luvas disponíveis nas portas USB:\n");
0023         printf("%i - ",c);
0024         switch (aPID[c])
0025         {
0026            case DG14U_R:
0027               printf("Data Glove 14 Ultra Right\n");
0028               break;
0029            case DG14U_L:
0030               printf("Data Glove 14 Ultra Left\n");
0031               break;
0032            case DG5U_R:
0033               printf("Data Glove 5 Ultra Right\n");
0034               break;
0035            case DG5U_L:
0036               printf("Data Glove 5 Ultra Left\n");
0037               break;
0038            default:
0039               printf("Desconhecida\n");
0040         }
0041      }
0042
0043      sprintf(szPortToOpen,"USB%i",0);
0044      fdOpen(szPortToOpen);
0045
0046   }
0047
0048   if (NULL == (pGlove = fdOpen( szPortToOpen)))
0049   {
0050      verify = -1;
0051   }
0052   else
0053   {
0054      NumSensors = fdGetNumSensors(pGlove);
0055      verify = 1;
0056   }
0057
0058   return verify;
0059
0060}
```

Figura 3.33. Código em C++ para inicialização da luva de dados.

No código em linguagem de programação C++ outras funções foram implementadas para serem acionadas por métodos nativos, como as funções fdGetSensorRawAll e fdGetGesture, adotadas para captar os valores de todos os sensores e o número de um gesto, respectivamente, acionadas pelos métodos nativos getRawSensorData() e getGesture().

O trecho de código para a captação dos valores dos sensores e dos gestos dentro das funções nativas é apresentado na Figura 3.34.

```
0001
0002//Obtém o valor de todos os sensores
0003JNIEXPORT jshortArray JNICALL
0004Java_ViMeT_NativeGlove_getRawSensorData(JNIEnv *env, jobject obj)
0005{
0006    jshortArray result = env->NewShortArray(NumSensors);
0007    fdGetSensorRawAll(pGlove,data);
0008    env->SetShortArrayRegion(result,0,NumSensors,(jshort*)data);
0009    return result;
0010}
0011
0012//Obtém o número do gesto pré-definido pelo fabricante
0013JNIEXPORT jint JNICALL
0014Java_ViMeT_NativeGlove_getGesture(JNIEnv *env, jobject obj)
0015{
0016    jint gesture = fdGetGesture(pGlove);
0017    return gesture;
0018}
```

Figura 3.34. Código nativo para receber informações da luva de dados.

Pode-se observar que a primeira função nativa é definida como um vetor do tipo *short* (linha 3), e o valor a ser retornado na variável *result*, é um vetor do mesmo tipo (declarado na linha 6) com o tamanho definido de acordo com o número de sensores, informação verificada na inicialização. Na linha 7 observa-se a função disponibilizada pelo fabricante, que usa o ponteiro *pGlove* para obtenção dos valores dos sensores simultaneamente, armazenando-os no ponteiro *data* (vetor do tipo *short*), e posteriormente transferidos para *result* (linha 8).

Já a segunda função nativa é definida como sendo do tipo *int* e possui a variável do tipo inteiro *gesture*, que recebe o valor da função fornecida pelo fabricante (linha 16) de acordo com o gesto captado, o qual é retornado quando o método nativo é executado no programa escrito em linguagem de programação Java e apresentado no *prompt* de comandos.

Os valores obtidos por meio das funções em linguagem de programação C++, são retornados para um vetor declarado na classe *Glove*. Cada valor do vetor é utilizado como parâmetro para definir o ângulo de inclinação a ser aplicado na esfera de rotação (base de cada dedo), permitindo a rotação do conjunto de objetos que representa cada dedo. Para isso, é usado um conjunto de métodos e classes da API Java3D, como o método *setRotation* da classe *Transform3D*. Esta classe é acionada pelo método *set* implementado na classe *TransformGroup,* associado a cada objeto virtual que representa cada esfera que, por sua vez, está associada a cada dedo da mão virtual.

A função *fdClose*, empregada para fechar a comunicação com a luva de dados, recebe como parâmetro o ponteiro *pGlove*, e é acionada pelo método nativo *closeGlove()*, presente no método *finalize*, cuja execução ocorre quando o objeto é destruído, antes que o *Garbage Collector* comece a ser executado.

O esquema apresentado na Figura 3.35 mostra a execução da aplicação, no que diz respeito à integração, relacionando a parte implementada em Java (retângulos) e a parte implementada em C++ (retângulos com as bordas arredondadas).

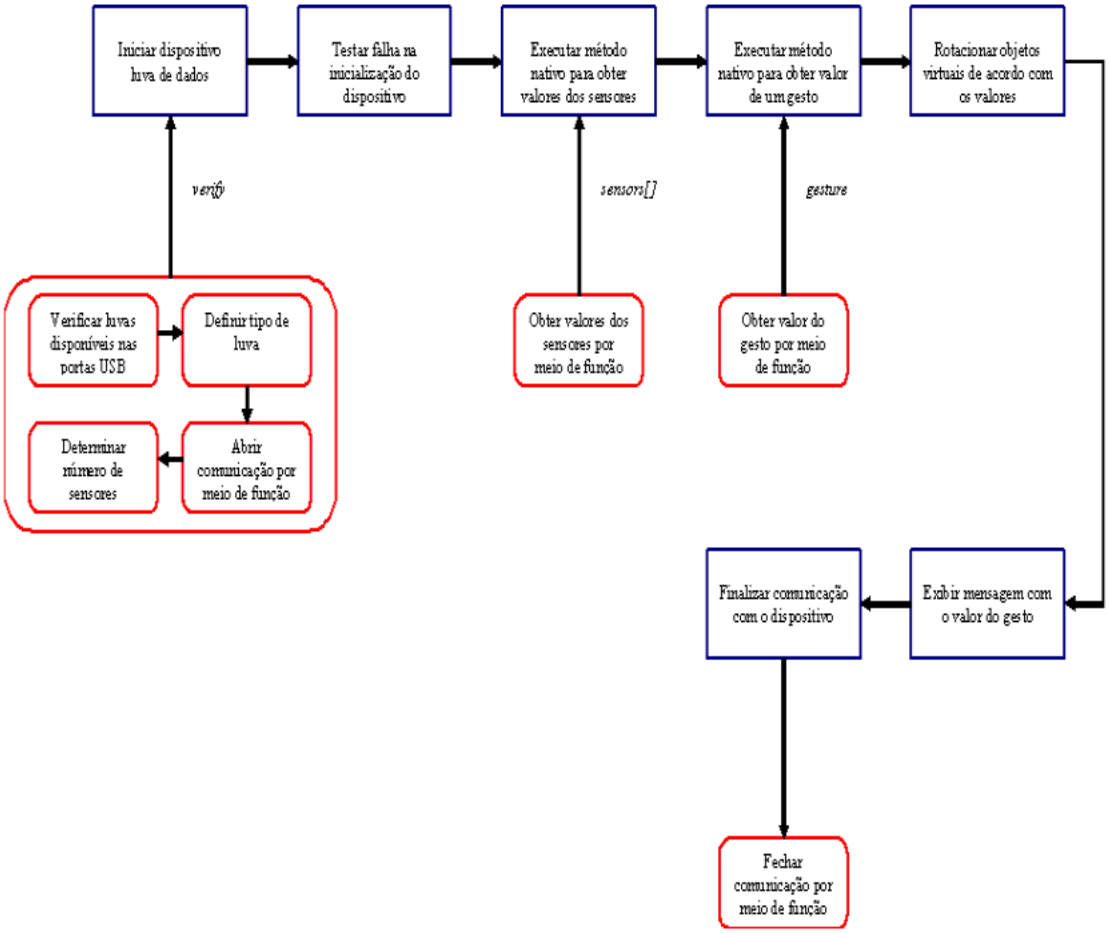


Figura 3.35. Esquema de integração da luva de dados.

3.4.1.3. Integração do dispositivo háptico

Os métodos nativos para o equipamento háptico (Tabela 3.9) foram agrupados em uma classe (*NativeHaptic*) e são usados na instanciação da classe *Haptic*, responsável pelo controle da interação com tal dispositivo. Os métodos nativos acionam funções provenientes do *OpenHaptics Toolkit*, que acompanha o dispositivo e oferece funções para programações em baixo e alto nível.

Para que se possa trabalhar com o dispositivo é necessário primeiramente inicializá-lo, o que é realizado por meio da função *hdInitDevice*. Em seguida, o retorno de força é habilitado por meio da função *hdEnable* (*HD_FORCE_OUTPUT*), e o *Scheduler* é definido como assíncrono (*hdSchedulerAsynchronous*). O *Scheduler* é responsável por acionar as *callbacks*, podendo ser síncronos ou assíncronos, com prioridades de execução definidas pelo usuário. A *callback* definida como principal no projeto é apresentada na Figura 3.36.

Uma função *callback* consiste em uma *thread*, uma forma de elevar o processamento para manter a frequência de comunicação com o dispositivo em 1000 Hz, adequada para interação. A execução da *callback* é iniciada pelo comando *hdStartScheduler*.

```
0001HDCallbackCode HDCALLBACK MainCallback(void *data)
0002{
0003    hdBeginFrame(hdGetCurrentDevice());
0004
0005        //Funções de estado do OpenHaptic Toolkit
0006
0007        //Função para obter posição
0008        hdGetDoublev(HD_CURRENT_POSITION, position);
0009
0010        //Função para obter matriz de transformação
0011        hdGetDoublev(HD_CURRENT_TRANSFORM, transform);
0012
0013        //Função para obter a velocidade
0014        hdGetDoublev(HD_CURRENT_VELOCITY, velocity);
0015
0016        //Função para obter o estado do botão 1
0017        hdGetIntegerv(HD_CURRENT_BUTTONS, &currentButton1);
0018        hdGetIntegerv(HD_LAST_BUTTONS, &lastButton1);
0019
0020        if ((currentButton1 & HD_DEVICE_BUTTON_1) != 0 &&
0021        (lastButton1 & HD_DEVICE_BUTTON_1) == 0)
0022        {
0023            button1 = 1;  //Botão 1 pressionado
0024        }
0025        else if ((currentButton1 & HD_DEVICE_BUTTON_1) == 0 &&
0026        (lastButton1 & HD_DEVICE_BUTTON_1) != 0)
0027        {
0028            button1 = 0; //Botão 1 está liberado
0029        }
0030
0031        //Função para obter o estado do Botão 2
0032        hdGetIntegerv(HD_CURRENT_BUTTONS, &currentButton2);
0033        hdGetIntegerv(HD_LAST_BUTTONS, &lastButton2);
0034
0035        if ((currentButton2 & HD_DEVICE_BUTTON_2) != 0 &&
0036        (lastButton2 & HD_DEVICE_BUTTON_2) == 0)
0037        {
0038            button2 = 1; //Botão 2 está pressionado
0039        }
0040        else if ((currentButton2 & HD_DEVICE_BUTTON_2) == 0 &&
0041        (lastButton2 & HD_DEVICE_BUTTON_2) != 0)
0042        {
0043            button2 = 0;  //Botão 2 está liberado
0044        }
0045
0046        //Realiza o retorno de força
0047        //Função para o retorno de força
0048        hdSetDoublev(HD_CURRENT_FORCE, force);
0049
0050
0051    hdEndFrame(hdGetCurrentDevice());
0052
0053    //Verifica erros durante a execução da CALLBACK
0054    if (HD_DEVICE_ERROR(error = hdGetError()))
0055    {
0056            fprintf(stderr,"CALLBACK encerrada");
0057            getch();
0058            return HD_CALLBACK_DONE;
0059    }
0060
0061    return HD_CALLBACK_CONTINUE;
0062}
```

Figura 3.36. Código da Callback principal.

Na inicialização, as funções mencionadas são acionadas na execução do método em Java *initializeHapticDevice*, que retorna os valores para a variável *verify*, indicando consistências de dados do dispositivo. O valor 1 indica que o dispositivo está pronto

para ser utilizado, -2 indica erro na inicialização do *Scheduler*, e –1 indica erro na inicialização do dispositivo.

No que diz respeito à *callback* implementada, esta contém as funções do *OpenHaptics Toolkit* para obter a posição do braço do dispositivo no espaço tridimensional real (linha 8), obter a matriz de transformação (linha 11), obter a velocidade de movimentação nos três eixos (linha 14), obter o estado dos botões 1 e 2 (trecho entre as linhas 17 a 29 e trecho entre as linhas 32 a 44, respectivamente), além de acionar o retorno de força (linha 48). O trecho de código entre as linhas 54 e 59 permite a verificação de erros durante a execução da *callback*, podendo encerrá-la em caso de problemas durante a execução.

Estas informações são enviadas e recebidas para a parte da aplicação em Java por meio de diversas funções nativas, como as funções mostradas na Figura 3.37. A primeira função (linhas 3 a 9) é responsável pelo recebimento de valores que indicam o retorno de força nos três eixos, usando para isso um vetor de três posições; a segunda (linhas 12 a 18) trata da obtenção da matriz de transformação 4 x 4, a qual é armazenada em um vetor de dezesseis posições, permitindo que a parte da aplicação em Java realize rotações nos três eixos no objeto virtual que representa o instrumento médico; a última função nativa (linhas 21 a 27) visa a obtenção de um vetor de três posições com os valores de translaçãodo dispositivo, que também serão utilizados pela parte da aplicação em Java para movimentar o instrumento médico virtual.

```
0001
0002//Obtém a força calculada pela aplicação Java
0003JNIEXPORT void JNICALL
0004Java_ViMeT_NativeHaptic_sendData
0005(JNIEnv *env, jobject obj, jdoubleArray dataarray)
0006{
0007    env->GetDoubleArrayRegion(dataarray, 0, 3, force);
0008    env->ReleaseDoubleArrayElements(dataarray,force,0);
0009}
0010
0011//Retorna a matriz de transformação
0012JNIEXPORT jdoubleArray JNICALL
0013Java_ViMeT_NativeHaptic_receiveData(JNIEnv *env, jobject obj)
0014{
0015        jdoubleArray array2 = env->NewDoubleArray(16);
0016        env->SetDoubleArrayRegion(array2,0,16,(jdouble*)transform);
0017        return array2;
0018}
0019
0020//Retorna a posição do dispositivo nos eixos x, y e z
0021JNIEXPORT jdoubleArray JNICALL
0022Java_ViMeT_NativeHaptic_receiveTranslation(JNIEnv *env, jobject obj)
0023{
0024    jdoubleArray array3 = env->NewDoubleArray(3);
0025    env->SetDoubleArrayRegion(array3,0,3,(jdouble*)position);
0026    return array3;
0027}
```

Figura 3.37. Funções nativas do equipamento háptico.

Uma variável denominada *verifyThread*, definida em um código nativo separado, e acionada pelo método *verifyCallback* (em Java), também verifica o estado de execução da *callback* principal, indicando à parte da aplicação em Java se houve ou não a ocorrência de erro, sendo que o valor –1 indica erro na execução e o valor 1 indica que ela está sendo executada corretamente. Ao término da execução da aplicação, sem a ocorrência de erros, um método nativo finaliza o *Scheduler* e desabilita o dispositivo, encerrando a comunicação.

O esquema disponibilizado na Figura 3.38 apresenta a execução da aplicação, no que diz respeito à integração, relacionando a parte implementada em Java (retângulos) e a parte implementada em C++ (retângulos com as bordas arredondadas).

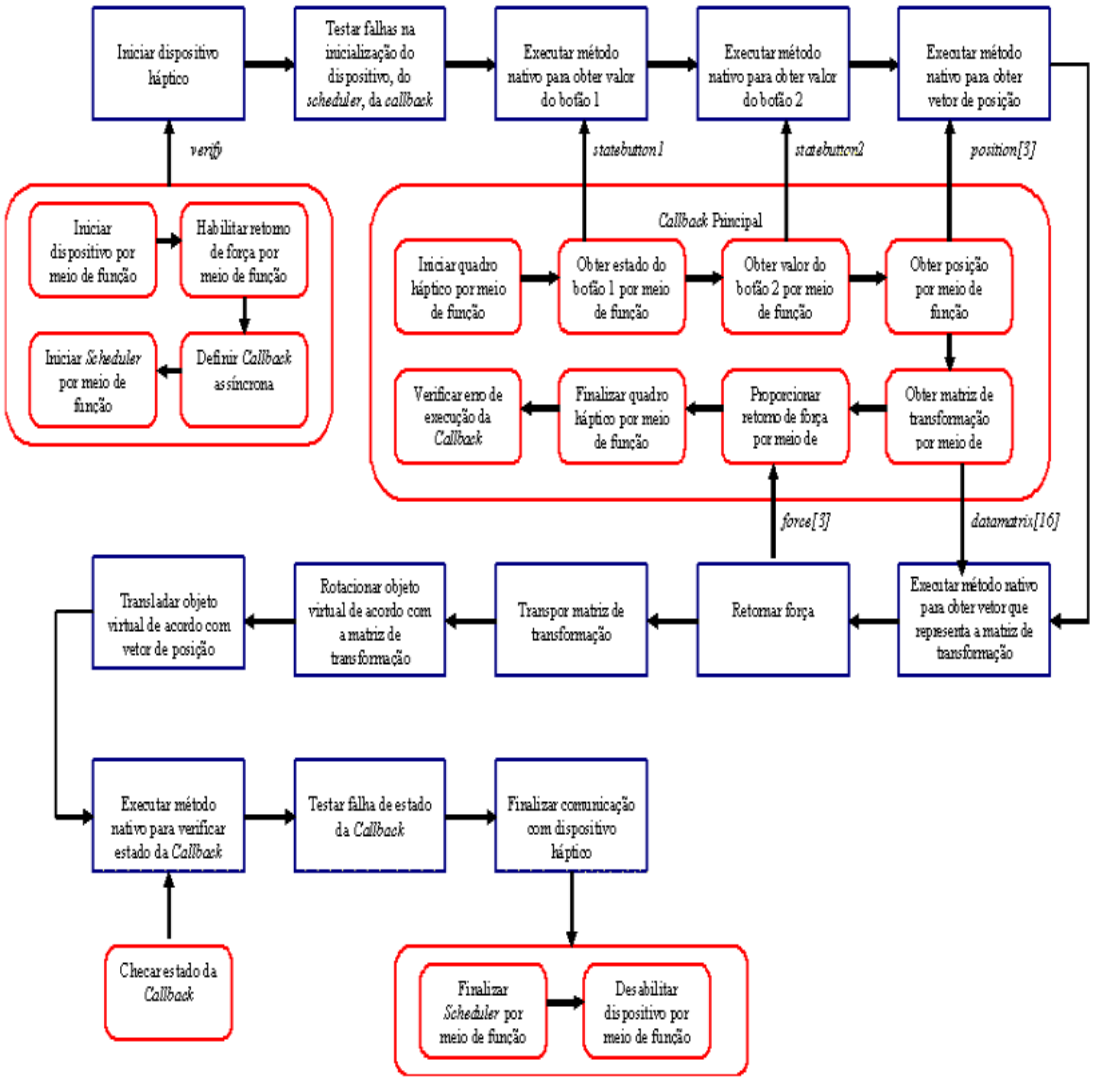


Figura 3.38. Esquema de integração do dispositivo háptico.

3.4.2 Análise de desempenho

A análise de desempenho computacional está relacionada ao hardware empregado (dispositivos e computador para processamento) e arquitetura de software (linguagens de programação, bibliotecas de dispositivos, APIs), sendo de grande importância em sistemas interativos. De acordo com Kirner (2008), as taxas de 20 de quadros por segundo ou superiores a este valor são consideradas ideais para interação, no entanto, taxas de 8 a 10 quadros são consideradas aceitáveis para oferecer a imersão em sistemas de RV. Outra questão é a precisão desejável em aplicações para a Medicina, que exige dispositivos e técnicas computacionais sofisticadas, influenciando também no desempenho da aplicação [Nunes *et al.* 2007].

Nos exemplos apresentados, deve-se considerar a camada de software adicional para integração das linguagens de programação, processamento mais elevado quando se

trata da comparação entre dispositivos convencionais e não convencionais, visto que os últimos possuem diversos graus de liberdade, sensores, atuadores, trabalhando com um número maior de informações. A luva de dados usada possui cinco sensores de fibra óptica, um para cada dedo, com o objetivo de capturar a flexão de cada um deles, já o dispositivo háptico PHANTOM *Omni* possui seis graus de liberdade de movimentação, podendo capturar movimentos de rotação e translação nos três eixos, três graus de liberdade (x, y e z) para retorno de força e botões para ativar comandos.

Uma das formas de averiguar o desempenho computacional é mensurar o número de quadros exibidos a cada segundo (fps - frames per second). As taxas de quadros por segundo para os dispositivos não convencionais foram as seguintes: de 57 a 61 fps usando a luva de dados e de 59 a 61 fps usando o equipamento háptico. Tais taxas foram consideradas satisfatórias, propiciando uma interação em tempo real, ou seja, sem atrasos na exibição dos quadros conforme as ações dos usuários. O computador utilizado nos testes possuía processador Pentium 4, de 3.0 GHz, 1GB de RAM, com placa de vídeo NVIDIA GeForce 5500.

3.5. Conclusões

Neste capítulo foram apresentadas considerações sobre a integração de dispositivos convencionais e não convencionais em aplicações de RV. Verifica-se que o uso de algumas aplicações torna-se inviável ou inadequado sem a presença de equipamentos não convencionais. No entanto, a integração desses não é trivial quando a tecnologia de software utilizada na aplicação difere da tecnologia utilizada na implementação dos *drivers* dos equipamentos. O estudo e a descoberta de uma forma adequada para integrar esses dispositivos podem constituir tarefas que demandam tempo de desenvolvimento, recurso que pode ser melhor empregado nas funcionalidades do sistema de RV propriamente dito.

A partir desse contexto, foi apresentada uma forma de integração das linguagens de programação Java e C/C++, utilizando interface nativa, como uma solução viável para o problema citado, tendo sido apresentados estudos de caso a partir de um módulo de interação implementado no *framework* ViMeT, que auxilia na integração de teclado, mouse, luva e dispositivo háptico. A solução apresentada pode ser adotada para outras linguagens e outros dispositivos.

Dessa forma, espera-se ter fornecido uma contribuição prática ao desenvolvimento de aplicações de RV, visto que os detalhes técnicos de implementação raramente são disponibilizados em publicações científicas da área.

Referências

5DT - Fifth Dimension Technologies (2009), http://www.5dt.com/hardware.html, Março.

Ames, A. L.; Nadeau, D. R.; Moreland, J. L. The VRML 2.0 Sourcebook, 2nd ed. New York: John Wiley & Sons, 1997, p. 654.

Bastos, T. A.; Raposo, A. B.; Gattas M. (2005) "Um Framework para o Desenvolvimento de Aplicações de Realidade Virtual Baseados em Componentes Gráficos", In: XXV Congresso da Sociedade Brasileira de Computação, São Leopoldo, p. 213-223.

Bernier, F.; Branzan-Albu A.; Drouin, R.; Laurendeau, D.; Limieux, F.; Mokhtari, M.; Quellet, D. (2004) "Virtual Environment and Sensori-Motor Activities: Visualization", In: WSCG – 12nd International Conference in Central Europe Computer Graphics, Visualization and Computer Vision, Plzen-Bory, República Tcheca, p.102-112.

Bezerra, A., Nunes, F. L. S., Corrêa, C. G. (2008) "Interação com Equipamentos Convencionais e Não Convencionais em Treinamento Médico", In: X Symposium on Virtual and Augmented Reality, João Pessoa, PB, Brasil, p. 275-278.

Bowman, D. A.; Gabbard, J.; Hix, D. (2001a) "Usability Evaluation in Virtual Environments: Classification and Comparison of Methods", In: Computer Science, eprints.cs.vt.edu/archive/00000541/01/VE_usability_presence.pdf, Dezembro, ACM Press.

Bowman, D. A.; Jr, J. J. L.; Kruijff, E.; Poupyrev, I. (2001b) "An Introduction to 3-D User Interface Design", In: Presence: Teleoperators and Virtual Environments, Volume 10, Número 1, MIT Press, p. 96-108.

Brewster, S.; Gray, P.; Mcgee, M.; Oakley, I. (2000) "Putting the Feel in Look and Feel", In: CHI – Conference on Human Factors in Computing Systems, Hague, Holanda, ACM Press, p. 415-422.

Burdea, G. C. Force and Touch Feedback for Virtual Reality. John Wiley & Sons, New York, USA, 1996.

Burns, J. M.; Chila, A. G.; Eland, D. C.; Howell, J. N.; Jr, R. R. C.; Srivastava, M.; Williams, R. L. (2004) "The Virtual Haptic Back for Palpatory Training", In: 6th International Conference on Multimodal Interfaces, State College, PA, USA, p. 191-197.

Celes, W.; Gattas, M.; Raposo, A. B.; Szenberg F. (2004) "Visão Estereoscópica, Realidade Virtual, Realidade Aumentada e Colaboração", http://www.tecgraf.puc-rio.br/publications/artigo_2004_visao_estereoscopica_realidade_virtual.pdf, Pontifícia Universidade Católica do Rio, Rio de Janeiro, Abril.

Cornell, G., Horstmann, C. S. (2003) "Métodos Nativos", Core Java 2, Volume 2, Recursos Avançados, In: Pearson Education do Brasil, São Paulo, SP, Brasil, p. 755-785.

Corrêa, C., Nunes, F. L. S. Bezerra, A., Carvalho Jr, P. M. (2009) "Evaluation of VR Medical Training Applications under the Focus of Professionals of the Health Area", In: 24th ACM Symposium on Applied Computing, Honolulu, Hawaii, EUA, ACM Press, p. 821-825.

DirectX, Site Oficial (2006), http://www.microsoft.com/windows/directx/default.mspx, Novembro.

Eisenstein, J.; Ghandeharizadeh, S.; Golubchik, L.; Sahabi, C.; Yan, D.; Zimmermann, R. (2003) "Device Independence and Extensibility in Gesture Recognition". In: IEEE Virtual Reality, IEEE Press, p. 207-214.

EST – *Engineering Systems Technologies* (2009) Disponível em: <http://www.est-kl.com/pricelist/pricelist_gb.html>. Acesso em: 13 Mar. 2009.

Flasar, J. (2000) "3D Interaction in Virtual Environment", In: 4th Central European Seminar on Computer Graphics, Budmerice, Eslováquia, p. 21-31.

Freitas, C. M. D. S.; Navarre, D.; Nedel, L. P.; Palanque, P.; Schyn, A. (2003) "Usando Modelagem Formal para Especificar Interação em Ambientes Virtuais: Por que?" In: SVR – Synposium on Virtual Reality, Ribeirão Preto, São Paulo, Brasil, p. 81-92.

Hansen, C. D.; Johnson, C. R.; Ikits. M. (2003) "A Comprehensive Calibration and Registration Procedure for the Visual Haptic Workbench", In: Workshop on Virtual Environments, Volume 39, Zurique, Suíça, ACM Press, p. 247-254.

Hsu, J. (2006) "Active Interaction Devices", http://www.hitl.washington.edu/scivw/EVE/I.D.1.a.ActiveInteraction.html, Dezembro.

Huff, R.; Silva, I. C. S. da; Vasconcelos, A. B. (2006) "Seleção de Objetos em Ambientes Virtuais com Mouse 3D", In: VIII SVR – Symposium on Virtual Reality, Belém, PA, Brasil.

Jafry, H; Johnson, K. W.; Okamura, A. M.; Webster III, R. J. (2003) "The Haptic Scissors: Cutting in Virtual Environments", In: ICRA – IEEE Conference on Robotics and Automation, Volume 1, Taipei, Taiwan, IEEE Press, p. 828-833.

Java3D - Java3D API (2008), v. 1.3.1, http://java.sun.com/products/java-media/3D/, Março.

Kilgard, M. J. OpenGL Programming for the X Window System. 4th ed. Massachusetts. Addison-Wesley. 1998.

Kopper, R. A. P.; Pinho, M. S.; Rieder, R.; Santos, M. C. C. dos; Silva, F. B. de A. e; Trombetta, A. B. (2006) "Uma Avaliação Sobre o Uso de Estímulos Táteis em um Ambiente Virtual", In: VIII SVR Symposium on Virtual Reality, Belém, PA, Brasil.

Kirner, C. (2008) "Sistemas de Realidade Virtual", http://www.dc.ufscar.br/~grv/tutrv/tutrv.htm, Universidade Federal de São Carlos, São Carlos, SP, Brasil, Dezembro.

Leibe, B.; Pair, J.; Singletary, B.; Starner, T. (2000) "MIND-WARPING: Towards Creating a Compelling Collaborative Augmented Reality Game", In: 5th International Conference on Intelligent User Interfaces, New Orleans, Louisiana, United States, ACM Press, p. 256-259.

LSI-USP - Laboratório de Sistemas Integráveis – Universidade de São Paulo (2009). Disponível em: <http://www.lsi.usp.br/interativos/nrv/caverna.html>. Acesso em: 13 Mar. 2009.

Mine, M. R. (1995) "Virtual Environment Interaction Techniques", http://citeseer.ist.psu.edu/cache/papers/cs/1796/ftp:zSzzSzftp.cs.unc.eduzSzpubzSztechnical-reportszSz95-018.pdf/mine95virtual.pdf, Dezembro.

Nunes, F. L. S. (2007) "Aplicações Médicas Usando Realidade Virtual e Realidade Aumentada", Kirner, C., Siscoutto, R., Realidade Virtual e Aumentada – Conceitos, Projeto e Aplicações, Livro do 9º Symposium on Virtual and Augmented Reality, Petrópolis, RJ, Brasil, p.234-235.

Oliveira, A. C. M. T. G., Pavarini, L., Nunes, F. L. S., Botega, L. C., Justo, D. R., Bezerra, A. (2006) "Virtual Reality Framework for Medical Training: Implementation of a deformation class using Java", ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry, Hong Kong, China.

Pinhanez, C. (2004) "Interfaces Não convencionais", In: Livro do VII Symposium on Virtual Reality: Realidade Virtual – Conceitos e Tendências, Editora Mania de Livro: São Paulo, p. 135

Pinho, M. S. (2006) "Interação em Ambientes Tridimensionais", Minicurso, In: WORKSHOP DE REALIDADE VIRTUAL, Gramado, RS, Brasil, http://www.inf.pucrs.br/~pinho/3DInteraction/, Outubro.

Pinho, M. S. and Rebelo, I. B. (2004) "Interação em Ambientes Virtuais Imersivos", In: Realidade Virtual – Conceitos e Tendências, Editora Mania de Livro, São Paulo, p. 109.

Pinho, M. S.; Silva, F. B. de A. (2007) "Avaliação de Técnicas de Auxílio a Wayfinding em Ambientes Virtuais", In: IX Symposium on Virtual and Augumented Reality, Petropólis, RJ, Brasil, p. 143-151.

Satalich, G. A. (2006) "Navigation and Wayfinding in Virtual Reality: Finding Proper Tools and Cues to Enhance Navigation Awareness", http://www.hitl.washington.edu/publications/satalich/ref.html, Dezembro.

SensAble Technologies (2007), http://www.sensable.com, Março.

Sense8 (2008), http://www.sense8.com, Março.

Sprenger, T. C., Gross, M., Bielser, D., Strasser, T. Ivory. (1998) "An Object-Oriented Framework for Physics-Based Information Visualization in Java", In: Proceedings of the IEEE Symposium on Information Visualization, IEEE CS Press, p. 79-86.

Sun. (2007) "Java Native Interface Specification", http://java.sun.com/j2se/1.4.2/docs/guide/jni/spec/jniTOC.html, Junho.

Thalmann, D. (2007) "Using Virtual Reality Techniques", http://citeseer.ist.psu.edu/cache/papers/cs/27653/http:zSzzSzvrlab.epfl.chzSzPublicationszSzpdfzSzThalmann_VRS_93.pdf/using-virtual-reality-techniques.pdf, Fevereiro.

Tori, R.; Kirner, C., Siscoutto, R. (editores) (2006) Fundamentos e Tecnologia de Realidade Virtual e Aumentada, Porto Alegre, Sociedade Brasileira de Computação, http://www.interlab.pcs.poli.usp.br, Janeiro, p. 396.

Tramberend, H. (2001) "Avango: A Distributed Virtual Reality Framework", In: Proceedings of Afrigraph, ACM Press.

Ziegeler, S. B. (2002) "Using Virtual Environments to Visualize Atmospheric Data: Can It Improve a Meteorologist's Potential to Analyze the Information?", http://citeseer.ist.psu.edu/cache/papers/cs/26818/http:zSzzSzwww.erc.msstate.eduzSzvailzSzpubszSz2002zSzsean01.pdf/ziegeler02using.pdf, Fevereiro.

Chapter
4

Interação Multimodal em Ambientes Virtuais

Daniela Gorski Trevisan, Luciana P. Nedel, Carla M. D. S. Freitas,
Jean Yves Lawson e Benoit Macq

Abstract

This tutorial aims at introducing and exploring the concepts and techniques for developing applications based on non-conventional interaction, i.e., non-WIMP (windows-icon-menu-pointer) interfaces. Multimodal interaction is defined by the use of different interaction modalities to support the same task. Developing such techniques is usually more difficult than implementing a single interaction method, and such difficulties need to be addressed from the beginning, since the conceptual design phase. In this tutorial we discuss the principles that have to be taken into account during the design phase of such techniques, and present a development methodology based on the multimodal framework OpenInterface (www.openinterface.org). This framework allows rapid prototyping and modeling of multimodal interfaces.

Resumo

Este tutorial tem por objetivo introduzir e explorar os conceitos e técnicas envolvidos na concepção de aplicações que necessitem de formas de interação alternativas à modalidade WIMP (windows-icon-menu-pointer). Interação multimodal pressupõe o uso de mais de uma modalidade de interação na solução de uma mesma tarefa e, desta forma, o desenvolvimento dessas técnicas apresenta dificuldades adicionais, que precisam ser abordadas desde o momento da concepção. O tutorial discutirá os princípios que devem ser considerados na fase de design, apresentará uma metodologia de desenvolvimento baseada na plataforma multimodal OpenInterface (www.openinterface.org). Esta plataforma tem por objetivo fornecer a rápida prototipagem e modelagem de interfaces multimodais.

4.1. Introdução

Uma modalidade pode ser interpretada como uma forma particular de representar um pensamento. Assim, uma interação de humano para humano pode ser considerada uma interação multimodal e um sistema que suporta modalidades humanas como gestos, fala, movimentos corporais, pode ser considerado um sistema multimodal [Coutaz 1991].

Desde os trabalhos de Bolt (1980) e Bolt e Herranz (1992), técnicas de interação multimodais vem sendo consideradas como uma maneira de aumentar a satisfação e conforto do usuário através de uma interação natural. Assim, a exploração de vários canais de comunicação entre um sistema e seus usuários pode ter um grande impacto na sua eficiência de utilização. Por exemplo, a interface do usuário tem influência na quantidade de comandos que o usuário pode executar em um determinado tempo assim como na taxa de erros cometidos pelo usuário [Reason 1990]. Vários estudos têm mostrado que o uso de dois dispositivos de apontamento em uma interface gráfica normal resulta em uma interação mais eficiente e compreensível do que a obtida usando mouse e teclado [Buxton e Meyers 1986] [Kabbash, Buxton e Seller 1994] [Zhai, Barton e Selker 1997]. Adicionalmente, modalidades complementares, como gestos e fala, podem ser usadas para reforçar e esclarecer a comunicação entre o usuário e o sistema [Oviatt 1999].

Não obstante, a interação com multimodalidade não é nenhuma panacéa. Os estudos de Dillon, Edey e Tombaugh (1990) e Kjeldskov e Stage (2004) revelaram que quando as relações entre multimodalidades não são projetadas corretamente, elas não são nem melhor compreendidas, nem mais eficientes. A fim de determinar a real contribuição de diferentes modalidades na interação com o usuário, muitos estudos empíricos foram realizados:

- Usabilidade e aceitação por parte do usuário de diferentes dispositivos e novas técnicas de interação [Bowman, Gabbard e Hix 2002] [Hinckley, Pausch, Proffitt e Kassel 1998] [Nedel, Freitas, Jacob e Pimenta 2003] [Poupyrev, Weghorst, Billinghurst e Ichikawa 1998];
- Usabilidade percebida, de acordo com o tipo de tarefas executadas [Dybkjær, Bernsen e Minker 2004] [Jöst, Haubler, Merdes e Malaka 2005] e com o contexto de uso (i.e. condições internas x externas, aplicações móveis) [Baille e Schatz 2005];
- Exatidão da interação com multimodalidade [Balbo, Coutaz e Salber 2003] [Kaster, Pfeiffer e Bauckhage 2003] [Suhm, Myers e Waibel 1999] [Holzapfel, Nickler e Stiefelhagen 2004].

Tais investigações mostram que o baixo nível dos dados capturados (i.e. eventos do usuário como cliques do mouse e fala) e o alto nível das intenções do usuário devem ser combinados com o objetivo de determinar a exatidão e a usabilidade percebida. É evidente que muitos usuários apreciam utilizar diferentes dispositivos não-convencionais apesar de seu baixo desempenho na execução de tarefas [Kaster, Pfeiffer e Bauckhage 2003]. Entretanto, devido à falta da suporte adequado para tratar a avaliação de usabilidade de interfaces multimodais, os resultados de investigações

empíricas parecem, às vezes, ou contraditórios ou não podem ser generalizados a outros domínios de aplicação.

Apesar disso, as interfaces multimodais estão se tornando mais comuns. Atualmente, há uma gama de dispositivos que integram técnicas de interação multimodais tais como: a caneta em PDAs, o reconhecimento da fala em telefones móveis e a combinação de diversos dispositivos de entrada e saída (por exemplo, luvas e captores do movimento) para jogos 3D.

O projeto descrito em [Tse 2005], [Tse 2006a] e [Tse 2006b] aborda uma pesquisa realizada com três jogos: *Warcraft III, The Sims* e *Virtual Surgery.* Esses jogos foram adaptados para suportar entradas multimodais e interação *multiplayer*. O objetivo do projeto é criar um ambiente onde se possa jogar entre vários jogadores interagindo de múltiplas formas, sendo essas interações via voz e gestos sobre uma mesa digitalizadora.

A interação com gestos, como demonstrado na Figura 4.1, possibilita o jogador selecionar um objeto, um personagem ou até mesmo selecionar uma área desejada, ou um conjunto de personagens para realizar uma determinada tarefa. Com a palma da mão os jogadores ainda podem mover os mapas e aplicar *zoom* sobre os objetos.

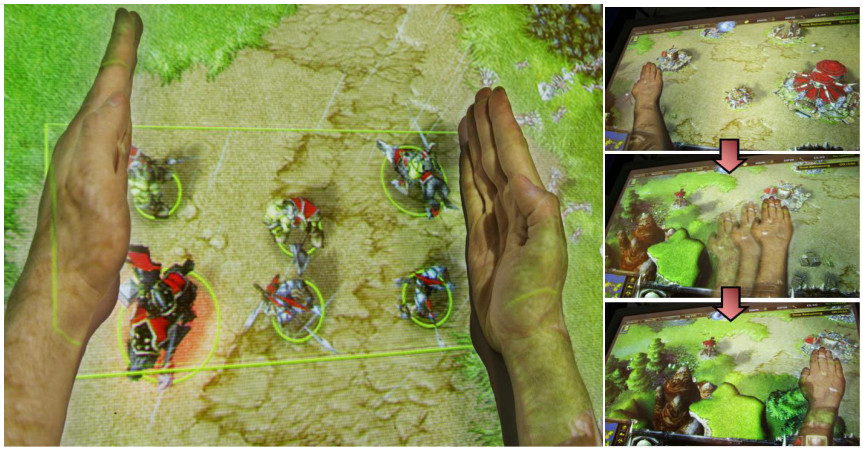


Figura 4.1: Warcraft III, seleção de região com duas mão (à esquerda) e movimentação da tela com uma mão (à direita). Fonte: [Tse 2006b].

Nesse contexto, a interação com voz fornece ao jogador a liberdade de passar comandos aos personagens sem a necessidade de um teclado ou mouse, dando mais agilidade e realismo ao jogo. A interação com voz também possibilita ao jogador selecionar unidades chamando pelo seu nome específico, por exemplo, é possível chamar, no *WarCraft III,* um construtor dizendo "*builder*". Através da fala seguida de gestos, o jogador pode selecionar uma unidade e a deslocar para uma região já com um objetivo em vista. Por exemplo, selecionando uma unidade do exército pode-se enviá-la a atacar uma unidade inimiga dizendo "*unit one attack*" e, em seguida, apontar com gestos a unidade a ser atacada. Simplificando, o jogador pode fazer uma sequência de comandos não permitidos no jogo original com apenas uma interação com gestos e a voz, como mostra a Figura 4.2.

Entretanto, existe uma carência de métodos e ferramentas apropriadas tanto para projetar quanto para examinar a usabilidade de sistemas com interfaces multimodais. A

exemplo de outros tipos de sistemas interativos, o desenvolvimento de sistemas multimodais é um processo iterativo composto de três etapas: projeto, desenvolvimento e avaliação.

Esse tutorial tem como foco a rápida prototipagem de interfaces multimodais para ambientes virtuais através do uso da plataforma OpenInterface.org. Nas próximas seções, serão discutidos os princípios que devem ser considerados na fase de projeto e será apresentada uma metodologia de desenvolvimento baseada na referida plataforma.

Figura 4.2: *Warcraft* III, gesto multimodal com voz e um dedo (à esquerda) e gesto multimodal com voz e dois dedos (à direita). Fonte: [Tse 2006b].

4.2. Concepção e Design de Interfaces Multimodais

Um sistema multimodal processa dois ou mais modos de entrada de dados do usuário, como: fala, gestos, olhar, movimento corporal e da cabeça, em coordenação com o sistema multimídia de saída [Oviatt 2002]. Nesse sentindo, segundo Coutaz (1991) os sistemas multimodais podem ser classificados de acordo com duas taxonomias: interfaces multimodais exclusivas e sinérgicas. Uma interface multimodal exclusiva possui múltiplas modalidades de interação, mas constrói uma entrada ou saída de dados com apenas uma modalidade de interação. Em contraste, uma interface multimodal sinérgica constrói uma entrada ou saída de dados com mais de uma modalidade de interação.

Além dessa taxonomia, Martin (1998) classifica as interações multimodais de uma forma mais específica, dividindo-as em cinco tipos básicos de cooperação entre modalidades, sendo elas: Transferência, Especificação, Equivalência, Redundância e Complementaridade. Já em Coutaz (1995) as interações com modalidades foram definidas em quatro grupos: Complementaridade, Atribuição, Redundância e Equivalência, originando as chamadas propriedades "CARE". A seguir, essas propriedades são detalhadas em exemplos práticos ilustrando o processo de design de sistemas interativos multimodais.

Várias modalidades são ditas equivalentes entre si se for possível para uma mesma tarefa utilizar mais de uma modalidade, contudo apenas uma de cada vez

[Martin 1998]. A Figura 4.3 mostra um exemplo de equivalência entre modalidades dispostas no sistema descrito em Coutaz (1995). O sistema permite obter informações sobre vôos especificando o destino desejado. Assim, especificando como destino *Pittsburgh*, o usuário, para obter informações sobre essa viagem, pode falar ou escrever a frase "vôos para *Pittsburgh*", ou ainda digitar "*Pittsburgh*" no campo "destino" do formulário de solicitação. Alternativamente ele pode interagir com a janela "ferramentas" selecionando a cidade de destino a partir do menu "conhecer cidades".

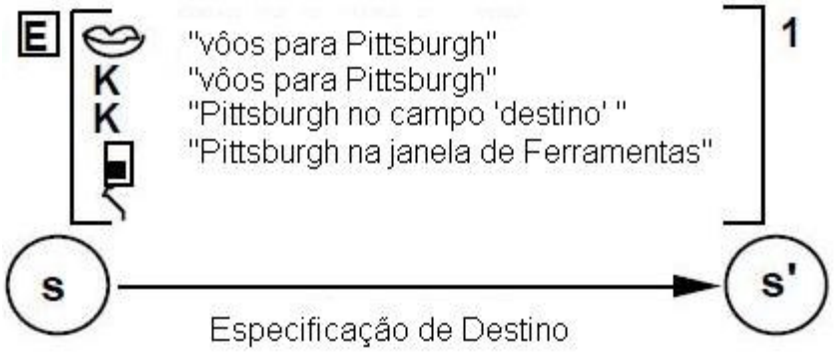


Figura 4.3: Exemplo de equivalência em MATIS. O símbolo "boca" representa a fala, o símbolo "K" representa o uso de um teclado e o símbolo "mouse" representa a seleção com o mouse, adaptado de Coutaz (1995).

Duas ou mais modalidades são redundantes quando a mesma informação for processada por essas modalidades com o mesmo poder computacional [Tzovaras 2008]. Se, por exemplo, para um ambiente multimodal qualquer, o usuário utilizar o *mouse* para selecionar o botão de sair de um ambiente e usar a fala para realizar o mesmo processo, isso deve ser reconhecido pelo sistema e tratado de forma a permitir uma interação com a mesma eficiência com as duas modalidades.

Atribuição diz respeito a uma tarefa que só pode ser executada com uma única modalidade [Coutaz 1995]. Para ilustrar o conceito, usaremos como exemplo um ambiente onde só é possível sair através da interação direta com o *mouse* no botão de saída. Assim, qualquer outro tipo de interação, como, por exemplo, interação com a voz dizendo "sair", não funcionará, pois a tarefa de sair do programa foi atribuída somente à modalidade de interação direta.

Para Martin (1998), o conceito de Atribuição é visto como Especialização. Assim, quando há uma relação de um para um entre um conjunto de informações e uma modalidade de gestão deste conjunto, falamos de especialização absoluta. Contudo, ele nos mostra que podemos definir especialização mais precisamente com as idéias de especialização de modalidade relativa e especialização de dados relativos. Especialização de modalidade relativa pode ser definida como a particularidade de muitos sistemas que possuem sons especificamente para tratar erros. Se eles forem utilizados unicamente para transmitir essa informação podem ser considerados como especialização de modalidade relativa. Em contrapartida, se esses erros não forem unicamente transmitidos por sons, mas também por gráficos ou textos, o sistema pode ser considerado de especialização de dados relativos.

A junção de duas modalidades para a realização de uma tarefa em um ambiente multimodal é conhecida como complementaridade [Coutaz 1995]. No ambiente descrito por Fonseca (1998), um editor gráfico pode utilizar complementaridade para aplicar algumas ações sobre uma figura geométrica, como demonstrado na Tabela 4.1, onde, para pintar uma figura deve-se dizer o comando *"Fill"* seguido de um gesto de apontamento da figura a ser colorida. A mesma idéia é utilizada para eliminar uma figura do editor: pronuncia-se o comando "*Delete"* seguido de um gesto apontando a figura na tela a ser apagada.

Tabela 4.1: Alguns comandos complementares do editor EDDY. Adaptado de Fonseca (1998).

Fill ▼[4]*This Red*	Preenche de vermelho o objeto apontado
Delete ▼ *This*	Apaga o objeto apontado

De acordo com essas definições podemos extrair uma comparação conforme disposto na Tabela 4.2, onde:

- Equivalência e Especificação dispensam a fusão;
- Redundância e Complementaridade requerem fusão;
- Equivalência e Redundância requerem uma transmissão de informações idênticas e
- Especialização e Complementariedade requerem uma transmissão de informações distintas.

Tabela 4.2: Comparação de equivalência, especificação, redundância e complementaridade, adaptado de Martin (1998).

	Mesmas Informações	Informações Diferentes
Sem Fusão	Equivalência	Especificação
Com Fusão	Redundância	Complementaridade

Para que seja possível a aplicação da propriedade de complementaridade ou mesmo a propriedade de redundância, é necessário fazer uso de algum mecanismo que trate as entradas oriundas de duas ou mais modalidades que realizam uma só tarefa. Sendo assim, o sistema deve utilizar algum mecanismo de fusão para unir essas duas ou mais modalidades para realizar uma dada tarefa.

4Este símbolo serve para indicar que a palavra que se segue deve ser acompanhada de um gesto simultâneo de apontar.

4.3. Mecanismos de Fusão e Fissão

Multimodalidade pode se apresentar de duas maneiras diferentes:

- Modalidades de entrada: envolvem o uso de vários dispositivos de entrada além de mouse, teclado, tais como, sistemas de reconhecimento de voz, *eye tracking*, captura de posições e gestos, etc.
- Modalidades de saída: envolvem o uso de vários dispositivos de saída tais como som espacial, visualizações tridimensionais, múltiplas telas, etc.

Uma das principais questões a respeito de sistemas multimodais é a fusão e a fissão das modalidades. Assim, um sistema multimodal pode permitir a interação de modalidades conjuntas, dando mais agilidade ao sistema, fazendo uso da propriedade de complementaridade ou redundância. Esse tipo de interação requer um tratamento específico, um mecanismo de fusão de modalidades, pois as entradas devem ser reconhecidas como uma única interação, sem ambiguidades.

Cada sistema adota o mecanismo mais apropriado ao seu ambiente, fazendo uso de tecnologias existentes, como é o caso do sistema MATIS descrito em Coutaz (1995), onde a fusão é representada por um modelo 2D, ilustrado na Figura 4.4, no contexto de uma arquitetura conhecida como PAC-Amodeus. Esse mecanismo de fusão é composto por partes estruturais denominadas "caldeirão". Cada "caldeirão" é resultado de uma interação do usuário com o sistema. Usando o exemplo de complementaridade da Figura 4.4, podemos entender melhor o funcionamento desse mecanismo. Para realizar uma tarefa T, o usuário utiliza as modalidade Y e Z complementarmente, sendo que, a modalidade Y, utilizada no tempo t_i forma o primeiro caldeirão, mostrado na Figura 4.4 na parte inferior esquerda do exemplo de complementaridade, e a modalidade Z, utilizada no tempo t_{i+1}, forma o segundo caldeirão, mostrado na parte inferior direita. Ambos, os caldeirões se fundem em um novo caldeirão mostrado na parte superior direita da Figura 4.4. Deve-se atentar para o tempo decorrente entre uma interação e outra. Cada sistema trata entradas duplas de forma distinta, mas parecida, ou seja, é estipulado um tempo limite entre as interações que compõem uma entrada de forma complementar, para que o sistema possa entender o tipo de interação adotada pelo usuário.

Já o sistema descrito por Elting (2003) utiliza uma forma diferente para realizar a fusão entre seus componentes de entrada. A fusão das modalidades é feita através do componente denominado "Modelo de Entrada Polinomial", que realiza uma integração semântica dos conceitos das modalidades. Esse sistema permite a interação de usuários com eletrodomésticos a partir de fala, gestos e interação com uma interface gráfica.

Para melhor ilustrar o conceito de fusão semântica, um usuário pode dizer "ligar" e apontar para um eletrodoméstico. Assim, o resultado da fusão das modalidades é a combinação do comando de voz com a especificação do objeto apontado. Semântica é utilizada para deduzir se o objeto selecionado tem relação com a interação feita, por exemplo, se o usuário falar "gravar programa" e apontar para uma lâmpada, a análise semântica irá acusar um erro na relação entre as modalidades, pois o usuário apontou para o objeto errado.

Assim como os mecanismos de fusão funcionam no intuito de associar modalidades de entrada, os mecanismos de fissão tem a função de associar modalidades

de saída de um sistema. Em Wahlster (2003), é descrito um sistema onde os mecanismos de fissão das modalidades de saída têm o objetivo de reconhecer os tipos de dados de entrada no sistema e, com sua interface de usuário antropomórfica e afetiva, comunicar-se com o usuário com as mesmas modalidades de entrada (fala, gestos e expressões faciais). Desta forma, combina as modalidades de saída para uma interação final com o usuário mais realística e agradável.

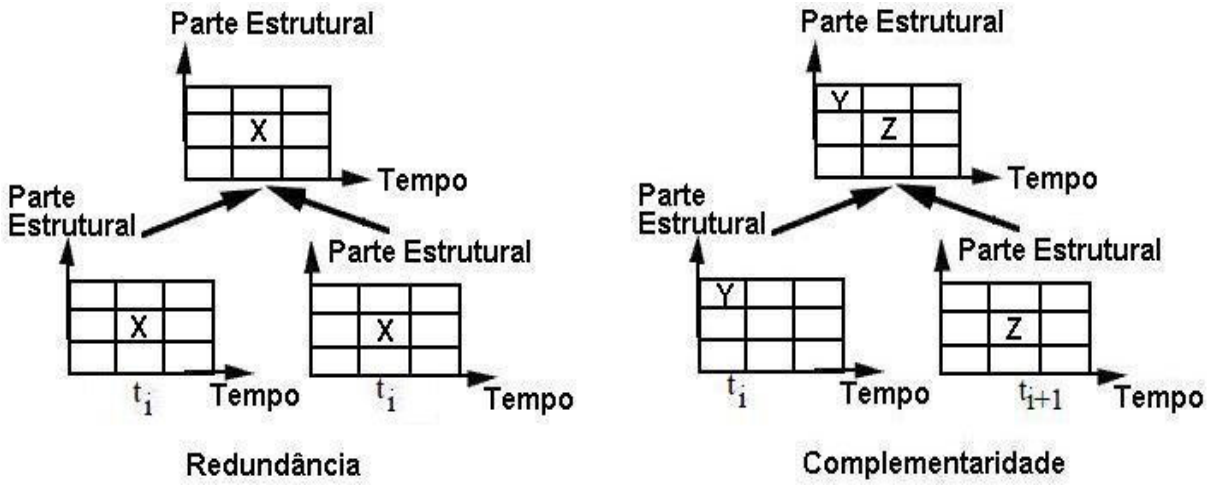


Figura 4.4: O efeito de redundância e complementaridade no mecanismo de fusão da arquitetura PAC-Amodeus. Adaptado de Coutaz (1995).

Como visto acima, todo mecanismo de fusão e fissão é disposto de forma a ser reutilizado por outra interação, ou no intuito de facilitar a manutenção desses sistemas. A próxima seção propõe o uso da plataforma OpenInterface.org para conceber e prototipar interfaces multimodais através de seus componentes heterogêneos e reutilizáveis.

4.4. OpenInterface - Ferramenta de Design para Interfaces Multimodais

Embora diversos sistemas com interfaces multimodais estejam sendo construídos, seu desenvolvimento constitui, ainda, uma difícil tarefa. As ferramentas dedicadas à interação multimodal são atualmente poucas e limitadas ou dirigem-se a um problema técnico específico como, por exemplo, o mecanismo de fusão [Nigay e Coutaz 1995] e a desambiguidade mútua [Oviatt 2000], ou são dedicados a modalidades específicas. Por exemplo, o Toolkit GT2k é projetado para suportar o reconhecimento de gestos [Westeyn *et al.* 2003].

Durante a fase de concepção de aplicações multimodais, o projetista pode especificar a modalidade ou modalidades de interação que podem ser aplicadas a uma determinada tarefa do sistema que está sendo desenvolvido. Visando implementar tal funcionalidade, OpenInterface (www.openinterface.org) integra os conceitos propostos pela plataforma multimodal CARE conforme mencionado anteriormente.[5]

[5] OpenInterface é uma plataforma de código aberto desenvolvida pelo Laboratório TELE da Université catholique de Louvain (Bélgica) em colaboração com o Laboratório de Interação Humano-Computador IMAG/UFJ (França) inicialmente no âmbito do projeto europeu SIMILAR (www.similar.cc) e atualmente suportado pelo projeto www.oi-project.org/.

4.4.1 Visão Geral

OpenInterface [Serrano *et al.* 2008] [Lawson *et al.* 2008] faz uso de uma arquitetura de software baseada em componentes que possibilita conectar modalidades implementadas em diferentes linguagens de programação. A Figura 4.5 ilustra uma visão geral do funcionamento da plataforma. Cada componente é integrado e registrado na plataforma através da descrição de sua interface seguindo a definição CIDL (*Component Interface Description Language*), a qual é baseada em XML. Após, o núcleo da plataforma automaticamente gera os *proxies* para realizar a integração propriamente dita do componente na plataforma. Uma vez que os componentes foram registrados e integrados na plataforma eles estão prontos para serem usados no projeto de uma aplicação multimodal através da definição de um *pipe* de execução. Esse passo poder ser feito manualmente através da descrição XML das conexões seguindo a definição PDCL - *Pipeline Description and Configuration Language* da plataforma ou através do seu editor gráfico OIDE - *OpenInterface Interaction Development Environment.*

A plataforma é composta por um conjunto de ferramentas de software agregadas nos seguintes grupos:

- **OpenInterface Kernel**[6]: é o núcleo computacional da plataforma. É o *middleware* que irá configurar os *pipelines* dos componentes, isto é, dada a descrição de um *pipeline* de execução, o kernel da plataforma irá inicializar e estabelecer os caminhos de conexão entre eles[7];
- **OpenInterface Collaborative Development Repository**[8]: fornece uma ferramenta para centralizar todos os esforços correspondentes ao desenvolvimento concorrente de software para a plataforma (componentes, *plugins*, editores, etc.);
- **OpenInterface Interaction Development Environment (OIDE)**[9]**:** fornece um editor gráfico desenvolvido em Java para o projeto de aplicações multimodais. Permite ao usuário manipular e agrupar graficamente componentes com o objetivo de especificar a interação multimodal dedicada a uma tarefa específica do sistema interativo em desenvolvimento;
- **OpenInterface Modality Repository**[10]: fornece um repositório extensível de modalidades para o projeto de aplicações multimodais.

A plataforma possibilita interação com três níveis de usuários:

- Programadores, que podem contribuir tanto com o desenvolvimento do núcleo da plataforma, por exemplo, estendendo-a para receber componentes de outras linguagens quanto inserindo novas modalidades de interação na plataforma;
- Projetistas de interação, que podem conceber uma aplicação multimodal através da composição de componentes já integrados na plataforma fazendo uso do Editor Gráfico, e

[6] https://forge.openinterface.org/projects/oikernel/
[7] Os exemplos abordados nesse tutorial foram realizados na versão 0.3.5.2.1 do Kernel da plataforma.
[8] https://forge.openinterface.org/
[9] https://forge.openinterface.org/projects/oide/
[10] http://dolak.dcs.gla.ac.uk:8080/OIRepository/

- Usuários finais que podem interagir diretamente com a aplicação multimodal projetada.

Neste trabalho faz-se uso de tal plataforma tanto para inserir novas modalidades de interação (componentes) quanto para desenvolver protótipos de aplicações multimodais e, assim, investigar as questões relacionadas, por exemplo, com a avaliação de usabilidade de tais sistemas.

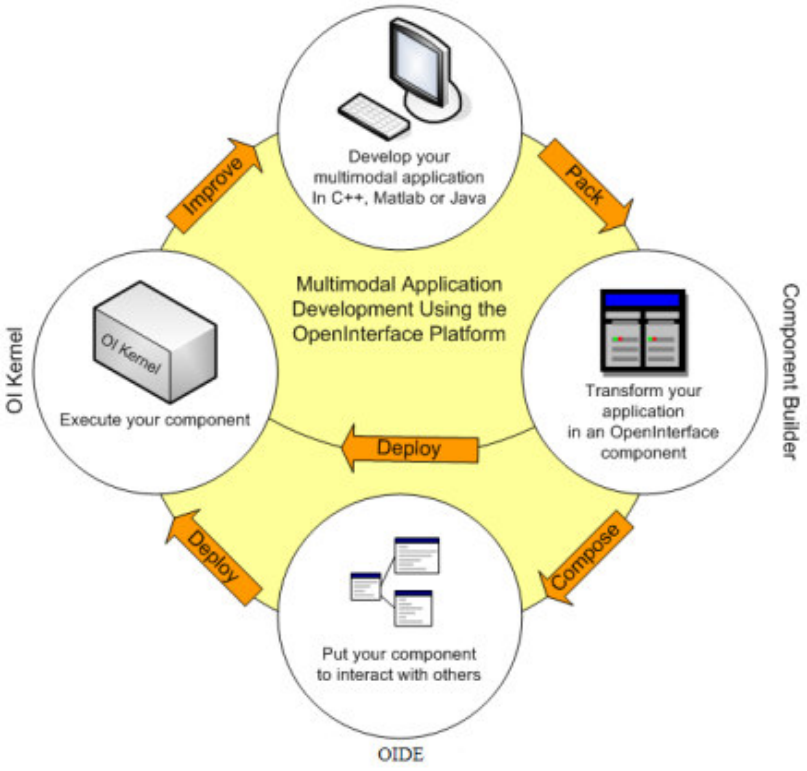


Figura 4.5. Desenvolvimento de uma aplicação multimodal usando a plataforma OpenInterface [Lawson et al. 2008].

4.4.2 Componentes

Componentes são os objetos-base manipulados pela plataforma OpenInterface. Cada componente representa um pedaço empacotado de software que fornece um conjunto de serviços e funcionalidades. Esse software pode ser desde *drivers* de dispositivos de entrada/saída até algoritmos de tratamento de sinais, módulos de rede, interfaces gráficas, etc.

Componentes são caracterizados por suas interfaces, ou seja, pontos de acesso que contém a descrição de dados que podem ser enviados ou recebidos por essas caixas pretas. Esses pontos de acesso da interface do componente são chamados *pins* e podem ser visualizados na Figura 4.6.

Existem três tipos de *pins*:

- *Sink*: *pin* através do qual o componente recebe dados;
- *Source*: *pin* através do qual o componente envia dados;

- *Callback*: *pin* através do qual o componente envia eventos assíncronos.

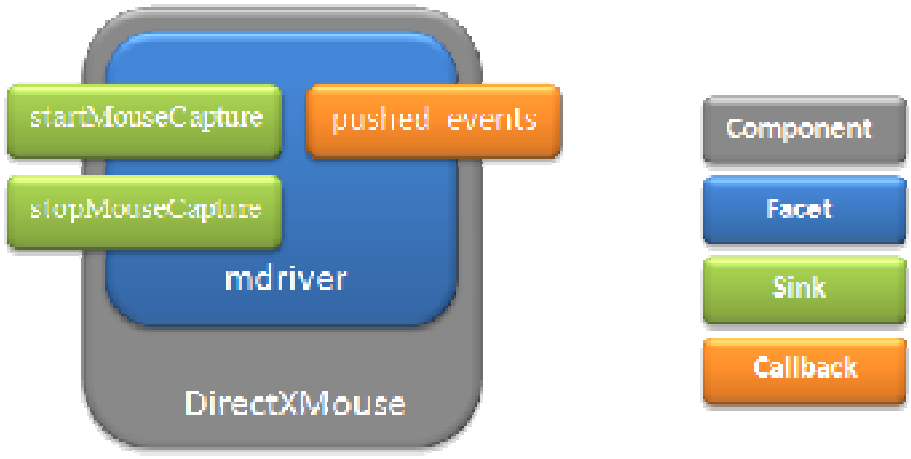


Figura 4.6. Exemplo de componentes OpenInterface.

Para ser capaz de manipular o componente, a plataforma OpenInterface precisa conhecer a descrição da interface do componente. Esta descrição é feita pela CIDL – *Component Interface Description Language*[11]. Uma vez que a descrição do componente foi fornecida, o componente pode ser facilmente reutilizado em qualquer aplicação OpenInterface.

Até o momento, dois tipos de componentes foram integrados: (1) componentes básicos, que permitem ao projetista especificar a modalidade "pura" de interação; e (2) componentes de composição ou de fusão, que permitem ao projetista especificar a utilização combinada de modalidades. A Figura 4.7 lista alguns componentes que fornecem modalidades de interação pura e que estão disponíveis no repositório de componentes da plataforma. Por outro lado, componentes que associam uma determinada modalidade dependendo do tipo de dado que está sendo manipulado, bem como a tradução automática entre modalidades, precisam ser desenvolvidos e integrados na plataforma.

Conectores

Conectores são componentes que encapsulam características genéricas tais como transformação de dados, filtros, protocolos de comunicação, etc. O nome conector não se restringe somente às funcionalidades da comunicação orientada (OSC, TCP, UDP, Multicast, etc), mas também inclui características genéricas como fusão de dados, fissão de dados, etc. Devido ao seu objetivo genérico, conectores não podem ter uma descrição CIDL fixa; sua descrição é construída sob demanda sempre que as entradas e saídas são conectadas. Este é o principal motivo pelo qual os conectores devem ser implementados dentro do núcleo, beneficiando-se assim da arquitetura genérica da plataforma.

[11] Disponível em http://www.openinterface.org/platform/documentation.

	Integrated Component
Gestural	•Sketch Recognizer •Gesture Recognizer •Head Tracker (OpenCV) •Wiimote (wiiuse) •Finger Tracking •Data Gloves (5DT, etc.) •Touch Pad (Synaptics) •Face Tracker •Infra-Red dot Tracking (OptiTrack) •Stylus •Hand Posture (HandVU) •Mouse, Keyboard •ARToolkit
Visual	•Generic Pie Menu •Camera(Firewire, USB Webcam) •VirtualMouse •Virtual Pointer
Audio	•Speech Recognizer (Sphinx)

Figura 4.7. Lista de componentes atualmente integrados e disponíveis no repositório de componentes de modalidades de interação da plataforma OpenInterface12 [Lawson et al. 2008].

Pipelines

Componentes OpenInterface podem ser agrupados através de conexões, gerando uma rede de componentes no intuito de gerenciar algumas tarefas avançadas. Esta inter-conexão de componentes é chamada de *pipeline*, conforme ilustra a Figura 4.8.

Componentes podem ser agrupados em *pipelines* conectando-os através de *sinks* e *sources* dos vários componentes. Desta maneira, o dado produzido pelo *pin source* de um dado componente será enviado para o *pin sink* de outro componente, permitindo assim a troca de dados e eventos entre os componentes.

Uma das grandes vantagens da plataforma OpenInterface é que componentes utilizados em uma mesma aplicação podem ser heterogêneos, ou seja, podem ser implementados em diferentes linguagens de programação. A própria plataforma é quem gerencia a tradução e a comunicação do dado entre as diferentes linguagens de programação através de ferramentas apropriadas. Até o presente momento a plataforma suporta componentes fornecidos em C/C++, Matlab e Java, mas suporte a outras linguagens de programação pode ser facilmente incorporado.

Para que um determinado *pipeline* possa ser manipulado pela plataforma, este deve ser descrito segundo as definições *PDCL - Pipeline Description and Configuration Language*.

[12] Repositório público de componentes OpenInterface disponível em http://forge.openinterface.org.

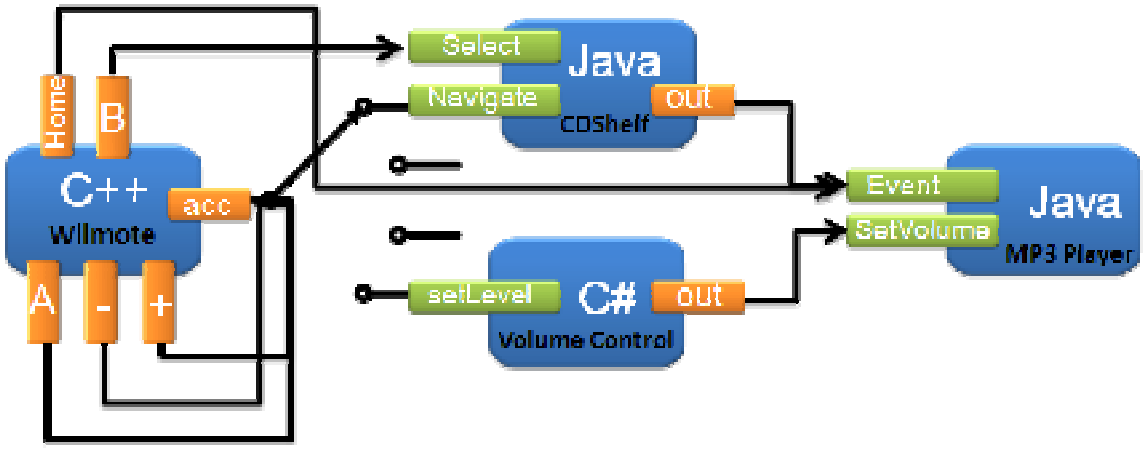


Figura 4.8. Exemplo de um pipeline OpenInterface para um player multimodal [Lawson et al. 2008].

Esta descrição define os componentes que serão usados no *pipeline* e como eles serão conectados. Assim, através da descrição PDCL, um *pipeline* poderá ser executado pela plataforma, que irá inicializar seus vários componentes e estabelecer a comunicação entre eles.

Até o momento a plataforma oferece 3 tipos de conectores básicos:

- *Ignite Plug*: este conector é usado para lançar ou inicializar um componente sem bloquear todo o *pipeline* de execução. Isto é possível inicializando o componente em uma *thread* dedicada (Figura 4.9);

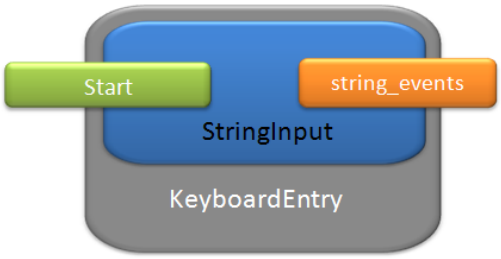


Figura 4.9. Exemplo de pipeline usando o conector básico Ignite Plug.

Inicialização em uma thread do *pin sink*.

```
<Ignite id="ignite_1" source="kb_start" threaded="yes" />
```

CIDL do *sink* Start.

```
<Sink id="kb_start">
<Interface type="function">
<Name value="Start"/>
</Interface>
</Sink>
```

Single Plug: este conector é usado para ligar um *callback* a um *Sink* ou *Source*. Quando estamos ligando uma interface *callback* a uma interface *Pin* a correspondência entre os valores de origem (*callback*) e os valores do destino deve ser especificada. Isto é feito através de um *Filter* com os elementos *In* e *Out* (Figura 4.10);

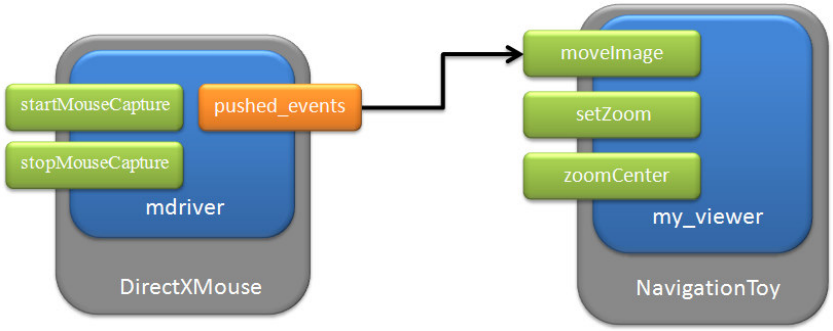


Figura 4.10. Exemplo de pipeline usando o conector básico Single Plug.

Conectando um *callback* a um *sink*.

```
<Plug id=" plug1 " source=" pushed_events" target ="moveImage ">
 <Filter>
 <In>
 <TargetValue target =" dx ">
          <SourceValue  source=" dx " />
 </TargetValue>
 <TargetValue target=" dy ">
 <SourceValue source=" dy " />
 </TargetValue>
 </ In>
</ Filter>
</ Plug>
```

- *Custom connectors*: é possível através de um único conector do tipo Multicast conectar 3 componentes em um *pipeline*, conforme ilustra a Figura 4.11. Este tipo de conector fornece recursos para enviar dados ao mesmo tempo para um número desconhecido de *pins*;

Pipelines podem igualmente ser criados e manipulados através do editor gráfico da plataforma. Uma vez que um determinado pipeline tenha sido criado, este pode ser facilmente modificado e ter seus componentes alterados por outros, permitindo assim a rápida avaliação da interação multimodal projetada.

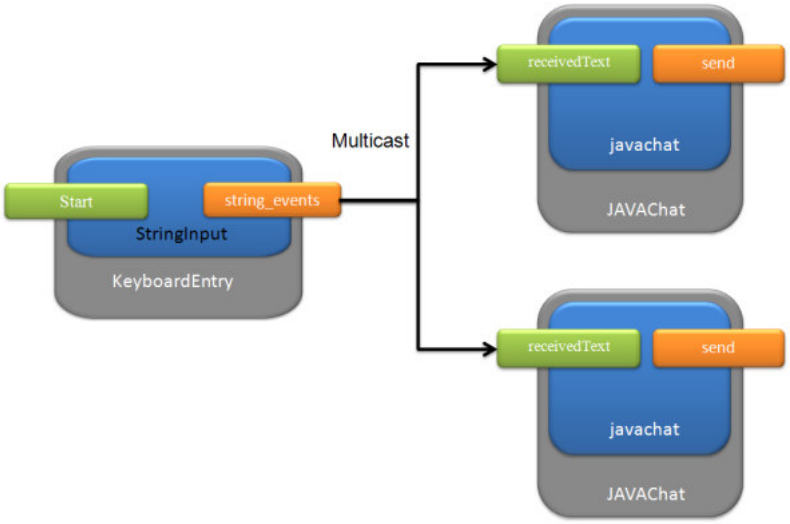


Figura 4.11. Exemplo de *pipeline* usando o conector customizado *Multicast*.

4.4.3 Gerando Componentes OpenInterface

Um componente deve ser fornecido como um pacote através de um arquivo compactado. A estrutura interna recomendada de um pacote de componente é predefinida e ilustrada na Tabela 4.3.

Tabela 4.3. Estrutura do pacote do componente.

Diretório	Função
etc/	Contém a descrição CIDL do componente
lib/	Contém as bibliotecas de dependências do componente (como .jar, .dll, .so, etc), se existir
bin/	Contém código binário do componente, atual implementação do componente + interface (.h + .dll, .class, .jar)
slib/	Contém dependências a serem instaladas no diretório de bilbiotecas do kernel
sbin/	Contém os binários a serem instalados no diretório de binaries do kernel
third_party_lib/	Contém dependências a serem instaladas no diretório de linguagem independente
share/	Outros arquivos necessários para o componente executar corretamente (como diretório de DATA, arquivo de configuração, etc.), se existir
src/	Código fonte do componente, se este for disponibilizado
doc/	Documentação para o componente
Oi.package.manifest	Arquivo de manifesto do pacote. Descreve a estrutura interna do pacote a eventuais dependências com outros componentes

Basicamente, existem dois tipos de componentes: componentes regulares, que são aqueles implementados na sua própria linguagem de programação, com suas bibliotecas específicas, e componentes do tipo conectores que são *plugins* para o núcleo da plataforma OpenInterface e, portanto, fazem uso da especificação da plataforma conforme descrito na sua API[13].

[13] https://forge.openinterface.org/frs/download.php/86/Plugins_spec.pdf

Neste texto, abordamos apenas os componentes regulares, fornecendo instruções de como transformar um componente qualquer num componente OpenInterface. Componentes regulares são então, qualquer software que poderá ser usado para testar, melhorar e projetar novas interações. Exemplos desses softwares são:

- Visualizadores;
- Dispositivos (câmeras, hápticos, reconhecedores de voz, acelerômetros, *joystick*, etc.);
- Mecanismos de fusão (semântica, temporal, etc.);
- Aplicações interativas completas, como games, etc.

Criar um novo componente ou refazer um componente já existente exige alguns cuidados se quisermos que esse componente seja reutilizável por outros usuários: a interface de entrada e saída do componente deve ser bem projetada para facilitar seu uso e também a sua possível conexão com outro componente que poderá ter sido escrito em uma outra linguagem de programação. Assim, projetar a interface de entrada e saída de dados do componente envolve basicamente duas partes:

- Decidir quais funcionalidades serão expostas na interface pública do componente;
- Especificar a interface das funcionalidades que serão expostas e os tipos de dados que serão compartilhados.

Decidir quais funções do seu componente serão compartilhadas ou expostas para a plataforma é um passo importante para a reutilização do código. Por exemplo, um componente do tipo *device* deve fornecer as seguintes funções na sua interface:

- *Start* e *stop* o dispositivo;
- Configurar o dispositivo;
- Receber e/ou enviar dados de forma síncrona ou assíncrona.

Já em um componente do tipo *viewer,* a definição da interface é guiada pelas tarefas que o usuário final irá realizar. Cada tarefa deve ser compartilhada como um *sink* ou *source*. Por exemplo, em um simples visualizador de imagens, o usuário deveria ser capaz de realizar as seguintes tarefas:

- Selecionar uma imagem em uma lista de imagens;
- *Zoom in/out;*
- *Pan left/right/up/downm*
- *Rotate left/right.*

Uma vez decidido quais funções expor na interface do componente, o próximo passo consiste em assegurar que o tipo de dado a ser compartilhado pode ser facilmente passado para o outro componente que possivelmente foi escrito em outra linguagem de programação e está sendo executado em outro processo na memória.

O estado atual de desenvolvimento do núcleo da plataforma OpenInterface permite apenas passar valores entre componentes escritos em linguagens diferentes. Isto

significa que o único tipo de dado suportado entre dois componentes que não são da mesma linguagem são os seguintes:

- Tipos de dados simples como *boolean*, *byte*, *char*, *short*, *int*, *long*, *int64*, *float*, *double*, *string* e suas representações *unsigned*;
- Vetores de tipos simples;
- Objetos como sendo agregados de tipos simples.

É possível passar tipos de dados com uma estrutura complexa, somente entre componentes escritos na mesma linguagem.

4.4.4 Criando um Componente C/C++ para a Plataforma OpenInterface

1. Defina a interface pública do componente e declare em um arquivo de cabeçalho.
2. Implemente e compile o componente como uma biblioteca compartilhada.
3. Escreva a descrição CIDL do arquivo de cabeçalho.
4. Empacote o componente.

Estes quatro passos serão ilustrados aqui através da implementação de um componente para a detecção de ruído. Esse componente foi implementado em C/C++ e é capaz de detectar cada vez que o usuário assopra no microfone.

Passo 1 - Defina a interface do seu componente e crie o cabeçalho (*header*). Tomando o exemplo do componente NoiseDetector, ele possui 2 funções, cada uma de um tipo: uma função para iniciar o componente, e a outra, para permitir que o NoiseDetector use um *callback* conforme ilustra a Figura 4.12.

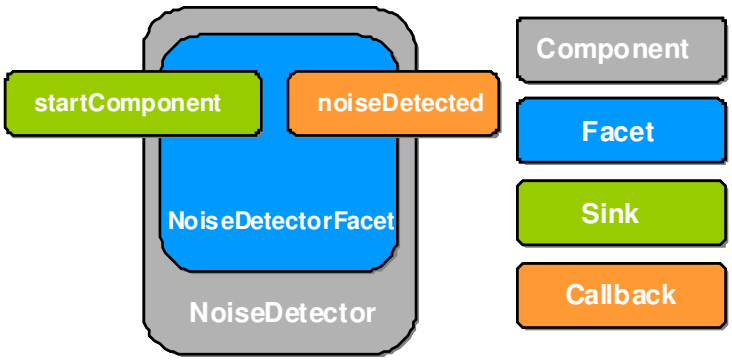


Figura 4.12. Descrição da interface do componente NoiseDetector.

O cabeçalho desse componente está descrito no arquivo **NoiseDetector.h**

```
/* O tradicional include-guard */

#ifndef NOISE_DETECTOR_H
#define NOISE_DETECTOR_H

/* Algumas macros pra permitir o uso do header tanto no programa cliente quanto na DLL */

#ifdef NOISEDETECTOR_EXPORTS
#define NOISEDETECTOR_API __declspec(dllexport)
#else
#define NOISEDETECTOR_API __declspec(dllimport)
#endif

/* Vamos usar uma interface C na DLL, mesmo que ela seja em C++ */
#ifdef __cplusplus
extern "C"
{
#endif

/* Função que será chamada pela OpenInterface (esta função iniciará o componente) */
NOISEDETECTOR_API void StartNoiseDetector();

/* Função que permite ao resto do sistema dizer qual callback deverá ser chamada pelo
NoiseDetector */
NOISEDETECTOR_API void SetNoiseCallback(void (*cback)(bool noiseDetected));

#ifdef __cplusplus
}
#endif
#endif
```

Quando um componente deseja que uma função sua seja "chamável" por outros componentes, ela deve ser declarada da primeira forma. Quando ocorre o contrário, o componente deseja chamar uma função externa (um *callback*), ele deve declarar apenas uma função que receba o ponteiro do *callback*. Posteriormente, o componente poderá usar esse ponteiro sempre que precisar informar um evento novo ao resto do sistema.

A função que inicia o componente poderá inicializar variáveis e/ou poderá entrar num *loop* principal. Conforme será visto posteriormente (na criação do *pipe*), é possível configurar a plataforma OpenInterface para abrir uma *thread* separada para o seu componente. Assim um programa já existente precisará de pouquíssimas alterações para funcionar dentro da OpenInterface, mesmo que ele dependa de um *loop* principal.

As funções são declaradas como tendo uma interface em C para facilitar a interoperabilidade entre diferentes tipos de compilador (no nosso caso, VC++ e GCC-MinGW).

Passo 2 - Implemente e compile o componente como uma biblioteca compartilhada. Crie um novo projeto no seu ambiente de desenvolvimento C++ (ou C) para gerar uma biblioteca dinâmica (DLL). Assim, as configurações de compilação e de ligação serão ajustadas automaticamente. Os exemplos usados neste tutorial foram testados com o MS Visual C++ 2003 (ainda que a OpenInterface use o MinGW para gerar seus *proxies*).

A implementação completa do componente NoiseDetector pode ser encontrada no arquivo NoiseDetector.cpp. A seguir serão ilustradas somente a implementação das funções expostas na interface do componente.

```
/*Chamada ao header correspondente para que a dll seja corretamente exportada*/
#include "NoiseDetector.h"
/* configura a chamada da função callback */
NOISEDETECTOR_API void SetNoiseCallback(void (*cback)(bool))
{
        NoiseInform = cback;
}

/*inicializa o componente*/
NOISEDETECTOR_API void StartNoiseDetector()
{
        PaError err = Pa_Initialize();
        CHECK_PA_ERROR_RETURN(err);
        err = Pa_OpenDefaultStream(
                &gStream,       /* passes back stream pointer */
                1,              /* input channels */
                0,              /* output channels */
                paFloat32,      /* 32 bit floating point output */
                44100,          /* sample rate */
                256,            /* frames per buffer */
                0,              /* number of buffers, if zero then use default minimum */
                RecordCallback, /* specify our custom callback */
                NULL);          /* pass our data through to callback */
```

```
        CHECK_PA_ERROR_RETURN(err);

        err = Pa_StartStream(gStream);
        CHECK_PA_ERROR_RETURN(err);
}
```

Exercício 1 - Consulte o código dos componentes fornecidos e identifique como os passos 1 e 2 foram implementados.[14]

1. Componente NoiseDetector: envia dados (NoiseDetector.cpp NoiseDetector.h)
2. Componente Cube: recebe dados (cube.cpp Cubedll.h)
3. Componente Gesture: envia dados (gestualComp.c gestualComp.h)

Passo 3 - Escreva a descrição do componente em XML. Vamos seguir, como exemplo, a descrição do componente NoiseDetector, contida no arquivo NoiseDetector_cfg0.xml. Comece o XML com as seguintes linhas, necessárias para qualquer arquivo XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Component PUBLIC "-//OpenInterface//DTD Component XML V0.1//EN"
"component.dtd">
```

Se a codificação do arquivo não for UTF-8 (consulte as configurações do seu editor de texto ou a caixa de diálogo *"Salvar como..."*, dependendo do editor), mude a declaração correspondente no XML (por exemplo, para iso-8859-1 ou Latin1).

A seguir, use as seguintes linhas como modelo para identificar e localizar o seu componente:

```
<Component id="NoiseDetector">
        <Name value="NoiseDetectorComponent" />
        <Language value="c++" />
        <Container>
                <Name value="NoiseDetector" />
                <Format value="dll" />
                <!- - Localização do componente dentro da pasta
                "component_repository" - ->
                <Location>Noise_component</Location>
        </Container>
```

[14] O código dos exemplos encontra-se em www.inf.ufrgs.br/cg/OI-Tutorial/

O conteúdo da tag *Container* indica como encontrar o seu componente. Por isso ela inclui os itens *nome*, *tipo (extensão)* e *localização (pasta)*. Os outros itens servem para associar um identificador único ao seu componente e indicar a linguagem em que ele foi escrito.

Abra as *tags* IO, *Facet* e especifique o nome do *header file* do seu componente na *tag* Bin, seguindo o modelo abaixo:

```
<IO>
 <Facet id="NoiseDetectorFacet">
        <Bin>
                <CustomType type="cppclass" name="" def="NoiseDetector.h" />
        </Bin>
```

A partir de agora, podemos declarar as funções que o componente exporta. Para uma função que estará disponível para outros componentes chamarem, declara-se um *Sink*. Para dizer que este componente deseja chamar funções de outros componentes, declara-se um *Source*.

Sink

Declarar uma função *Sink* é simples. Basta usar a marca *Sink* e escolher um identificador para ela. No exemplo abaixo, está a declaração do *sink* "startComponent", que servirá para iniciar o detector de ruído.

```
 <Sink id="startComponent">
        <Interface type="static_function">
                <Name value="StartNoiseDetector" />
        </Interface>
 </Sink>
```

Dentro da marca *Sink*, declara-se a interface da função. Como esta função não tem parâmetros, basta declarar seu nome (conforme está no código em C ou C++) de acordo com o exemplo acima.

Source

Vamos agora ver como criar uma função do tipo *Source*. Aqui também veremos como definir a interface de funções com parâmetros (a mesma maneira de declarar parâmetros pode ser usada em *Sinks*). A principal diferença é que agora usamos as marcas *Source* e *Callback*. Em seguida, define-se a interface da função que será chamada. Dentro da marca *Argument*, coloca-se a lista de argumentos para a função. Neste caso, há somente um argumento, do tipo booleano.

```
<Source id="noiseDetected">
 <Callback>
        <Interface type="static_function">
                <Name value="NoiseInform" />
                <Argument>
                        <Param name="noiseDetected">
                                <PrimitiveType name="bool" />
                        </Param>
                </Argument>
        </Interface>
```

Além da interface do *callback*, é necessário fornecer uma maneira pela qual outro componente possa registrar seu *callback*. No nosso exemplo, essa tarefa é feita pela função SetNoiseCallback declarada no *header* "NoiseDetector.h". Essa função também possui só um parâmetro, sendo ele do tipo *ponteiro para função*. Observe a declaração abaixo.

```
    <Setter>
     <Interface type="static_function">
     <Name value="SetNoiseCallback"/>
     <Argument>
      <Param name="cback">
            <Descr>Callback for noise/silence events</Descr>
            <CustomType type="cCallback" name="noiseCallback" def="NoiseDetector.h"/>
      </Param>
     </Argument>
    </Interface>
   </Setter>
  </Callback>
 </Source>
</Facet>
</IO>
</Component>
```

Compilando a descrição CIDL de um componente

Uma vez que você tenha a descrição CIDL e o código binário (dll) do componente, você pode registrá-los na plataforma, executando os seguintes comandos:

Gerar *proxies* para o componente:

- proxyBuilder <*your component CIDL*>.xml

Compilar os *proxies* gerados:

- make -f <*your component CIDL*>.MAKEFILE

Deve-se observar que estes dois comandos devem ser executados cada vez que um componente for modificado (código fonte do componente e/ou sua descrição XML).

Passo 4 – Empacote o componente. Para finalizar o componente, você deve compactar o componente seguindo a estrutura da plataforma conforme especificado na Tabela 4.4 a seguir, que toma como exemplo o componente NoiseDetector.

Tabela 4.4. Estrutura de diretórios do componente.

Diretório	Arquivo
bin/	NoiseDetector.h NoiseDetector.dll
etc/	NoiseDetector_cfg0.xml
src/	NoiseDetector.h NoiseDetector.cpp
package.manifest	<!DOCTYPE OIConfig PUBLIC "-//OpenInterface//DTD OIConfig XML V0.1//EN" "xconf.dtd"> <OIConfig id="inf.ufrgs.br.openinterface.component.c.noisedetector.cpp"> <PropertyGroup name="OI_PACKAGE_INFO"> <Property name="type" value="component"/> <Property name="cidl" value=" NoiseDetector_cfg0.xml"/> </PropertyGroup> </OIConfig>

Exercício 2 – Forneça a descrição CIDL para os componentes a seguir[15] e gere o pacote conforme a estrutura hierárquica da plataforma.

1. Componente Cube: recebe dados (cube.cpp Cubedll.h). Resposta: cube_noise_cfg0.xml
2. Componente Gesture: envia dados (gestualComp.c gestualComp.h) Resposta: Gesture_cfg0.xml

[15] Diponíveis em www.inf.ufrgs.br/cg/OI-Tutorial/

4.4.5 Conectando os Componentes em um Pipe - Abordagem Manual

Com os componentes prontos, vamos criar um *pipe* que os integre para serem executados. O *pipe* conectará as saídas de um componente com as entradas do outro.

O *pipe* começa com as declarações normais do XML e, então, declara uma lista de componentes (*ComponentList*). O atributo *descr* indica qual é o nome do arquivo XML que descreve o componente desejado, enquanto o atributo *id* atribui um identificador pelo qual o componente será referenciado posteriormente neste mesmo arquivo.

No exemplo abaixo, os componentes usados são o NoiseDetector já visto e o componente Cube, que servirá como programa principal e receberá comandos de NoiseDetector.

A seguir, eles serão identificados como *c3* e *c1*, respectivamente; o componente *c2* foi removido do exemplo.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Component PUBLIC "-//OpenInterface//DTD Component XML V0.1//EN"
"component.dtd">
<Pipeline>
 <ComponentList>
        <Component id="c1" descr="Cube_cfg0.xml"/>
        <Component id="c3" descr="NoiseDetector_cfg0.xml" />
 </ComponentList>
```

Então, declara-se uma lista de *facets*. Em cada um, indica-se o componente com sua *facet* e atribui-se um novo *id* à combinação.

```
 <FacetList>
        <Facet id="Cube_viewer" name="Cube_facet" component="c1" />
        <Facet id="NoiseDetectorFacet" name="NoiseDetectorFacet" component="c3" />
 </FacetList>
```

A próxima declaração é a *PinList*. Nela, atribuímos um novo identificador para cada par *Facet+Source* ou *Facet+Sink*.

```
<PinList>
        <Pin id="CubeStart" facet="Cube_viewer" name="startComponent" />
        <Pin id="shakeEvent" facet="Cube_viewer" name="shakeEvent" />
        <Pin id="StartNoiseDetector" facet="NoiseDetectorFacet" name="startComponent" />
        <Pin id="NoiseDetectorSource" facet="NoiseDetectorFacet" name="noiseDetected"/>
</PinList>
```

Finalmente, começamos a definição do *pipe* propriamente dito. O exemplo abaixo é composto das marcas *Plug* (para fazer a conexão entre os *pins sinks* e *sources*) e *Ignite* (para iniciar os componentes). Os nomes usados nos atributos *source* e *target* são aqueles declarados na seção *PinList*.

```
<Pipe>
 <Plug source="NoiseDetectorSource" target="shakeEvent">
    <Filter>
        <In>
            <TargetValue target="noiseDetected">
                <SourceValue source="noiseDetected"/>
            </TargetValue>
        </In>
    </Filter>
 </Plug>
 <Ignite source="CubeStart" threaded="yes" />
 <Ignite source="StartNoiseDetector" threaded="no" />
</Pipe>
</Pipeline>
```

A marca Plug indica os Pins de origem e destino que serão conectados entre si. Na marca Filter, fazemos um mapeamento dos parâmetros das funções. Este caso é o trivial, onde os parâmetros são exatamente iguais e têm o mesmo nome.

Nas marcas Ignite, indicamos quais Pins devem ser chamados para iniciar os componentes e se desejamos que eles sejam executados em threads separadas ou não (no caso, a biblioteca usada no NoiseDetector, PortAudio16, já cria sua própria thread para receber o áudio). A Figura 4.13 ilustra a conexão que foi realizada entre dois componentes utilizando o conector do tipo Plug source.

[16] http://portaudio.com/

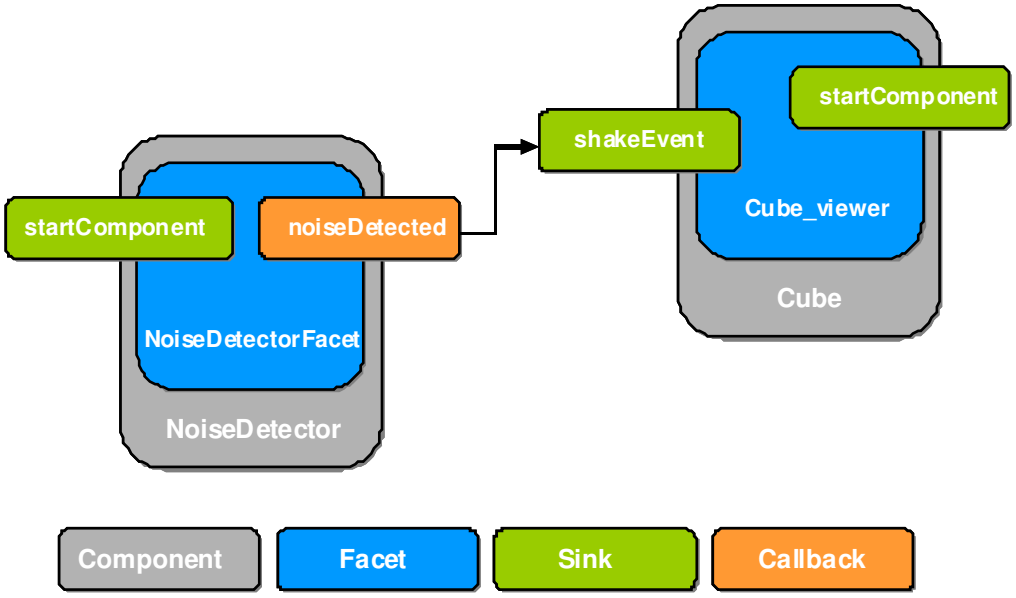


Figura 4.13. Pipeline de conexão entre os componentes NoiseDetector e Cube[17].

4.4.6 Executando pipes na Plataforma OpenInterface

Abra uma janela de comandos tipo terminal (por exemplo, DOS) e antes de executar o *pipe* propriamente dito, gere os *proxies* dos componentes e compile-os através dos comandos:

- proxyBuilder cube_noise_pipe.xml
- make –f cube_noise_pipe.MAKEFILE

Esses dois comandos devem ser executados somente uma vez, ou cada vez que a descrição de qualquer componente que compõe o *pipe* for alterada.

Para lançar o *pipe* de execução execute o comando:

- buildpipe cube_noise_pipe.xml

[17] A descrição desse *Pipe* encontra-se no arquivo *cube_noise_pipe.xml* diponível em www.inf.ufrgs.br/cg/OI-Tutorial/

Exercício 3 - Esse exercício usa o componente *gesture* fornecido pelo ARToolkit[18]. Para tanto, antes de mais nada é necessário verificar se esse componente está executando sem problemas. Conecte a sua WebCam e execute o programa **gestualTest.exe** no diretório **C:\OpenInterface\OIKernel\examples**

Se tudo correu bem, o próximo passo será fazer a integração desse componente na plataforma OpenInterface.

1. Usando NoiseDetector_cfg0.xml como exemplo forneça a descrição xml (CIDL) do componente gestualComp. Resposta: Gesture_cfg0.xml

2. Altere a descrição do componente cube (cube_noise_cfg0.xml) para esse receber também eventos do componente gestualComp. Resposta: cube_ar_noise_cfg0.xml

3. Altere a descrição do *pipe* (cube_noise_pipe.xml) para esse incluir também eventos do componente gestualComp conforme ilustra a Figura 4.14. Resposta: cube_ar_noise_pipe.xml

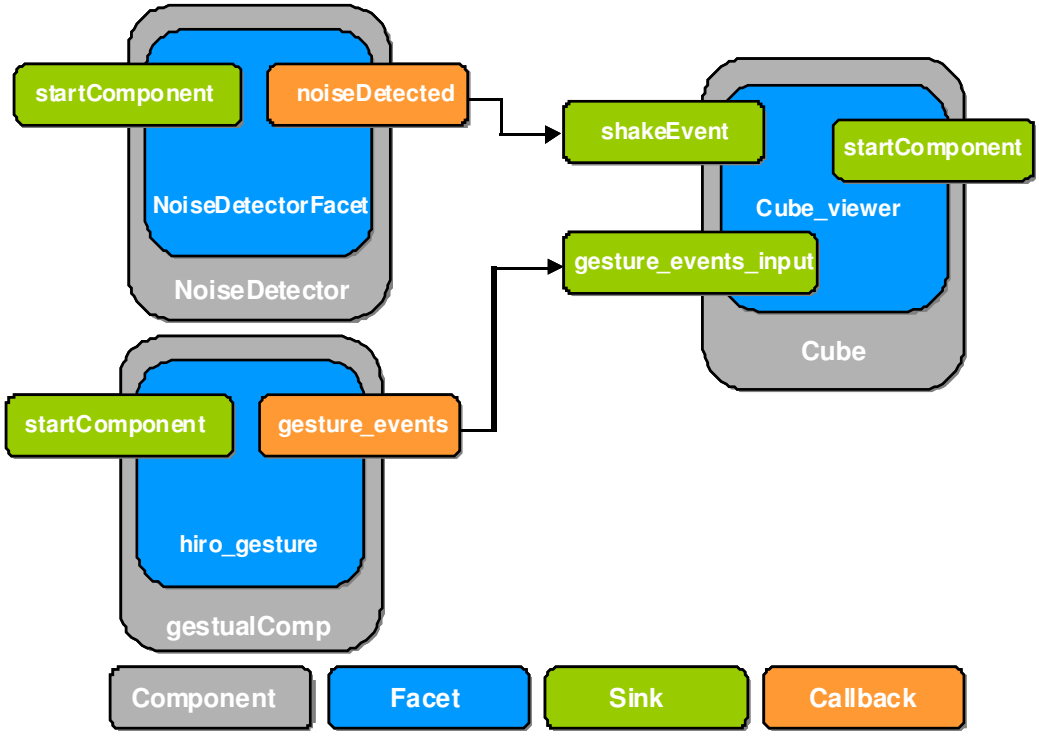


Figura 4.14. Pipeline de conexão entre os componentes que enviam dados NoiseDetector e gestualComp e o componente que recebe dados Cube[19].

[18] http://sourceforge.net/projects/artoolkit/

[19] A descrição desse *Pipe* encontra-se no arquivo *cube_ar_noise_pipe.xml* diponível em www.inf.ufrgs.br/cg/OI-Tutorial/.

DICA: lance o componente **gesture** em um processo separado utilizando a marca***RemoteSite***

```
<Component id="c2" descr="Gesture_cfg0.xml" >
  <RemoteSite name="localhost" lname="localhost" port="12345" />
</Component>
```

A seguir, execute em um terminal DOS a partir do diretório *example,* o comando **spawn 12345.** Finalmente, lance o novo pipe executando o comando **buidlpipe nome_pipe.xml** em outro terminal DOS.

Exercício 4 – Considerando os aspectos de design apresentados anteriormente, identifique o mecanismo de fusão/fissão sendo usado no exemplo do Exercício 3.

4.4.7 Editor Gráfico

Uma vez que os componentes foram integrados corretamente na plataforma eles serão automaticamente reconhecidos pelo editor gráfico OIDE. Esse editor tem por objetivo principal facilitar o processo de composição da aplicação multimodal através da geração automática do seu pipe de execução. A Figura 4.15 ilustra o componente NoiseDetector e o componente Cube, com suas respectivas interfaces, disponíveis para serem reutilizados no desenvolvimento de aplicações multimodais. Esse editor necessita da SDK Java instalada e encontra-se atualmente em desenvolvimento podendo apresentar problemas eventuais durante sua execução. Para maiores detalhes consulte o site do projeto https://forge.openinterface.org/projects/oide/

4.4.8 Aplicações Multimodais OpenInterface

Várias aplicações multimodais tem sido desenvolvidas através da plataforma OpenInterface. Por exemplo, Rosa *et al.* (2007) desenvolveram uma ferramenta de modelagem volumétrica para uso em construção de biomodelos. Nesse trabalho, uma interface com ferramentas 3D apropriadas para segmentar o volume visualizado foi adaptada para suportar interações multimodais através da captura de posições 3D no espaço (no intuito de posicionar a ferramenta no volume) e da detecção do sopro (para aplicar a ferramenta já posicionada).

No trabalho descrito por Benoit *et al.* (2006), vários componentes foram integrados através de uma abordagem híbrida e distribuída com o objetivo de detectar o nível de atenção do motorista em um simulador de direção. A abordagem é dita híbrida pelo fato dos componentes OpenInterface interagirem através de trocas de mensagens,

com componentes que não foram integrados na plataforma. Já a abordagem distribuída deve-se ao fato de que os componentes são executados em máquinas distintas. Assim, componentes desenvolvidos em Matlab, para captura e análise de sinais fisiológicos, como batimentos cardíacos e nível de transpiração da pele, foram fusionados com outros componentes destinados à análise de expressão facial através de visão computacional e desenvolvidos em C/C++. Para tanto, um componente de fusão de sinais utilizando redes bayesianas foi desenvolvido e integrado na plataforma OpenInterface para estimar o possível nível de sonolência do motorista. A saída desse componente de fusão consiste na entrada do componente de fissão, que é o responsável por escolher o tipo de mensagem/alarme e o nível de vibração da direção que serão emitidos para despertar o motorista. Essa abordagem envolvendo componentes com alto grau de processamento, serviu para verificar a capacidade da plataforma OpenInterface em gerenciar aplicações multimodais complexas e em tempo real.

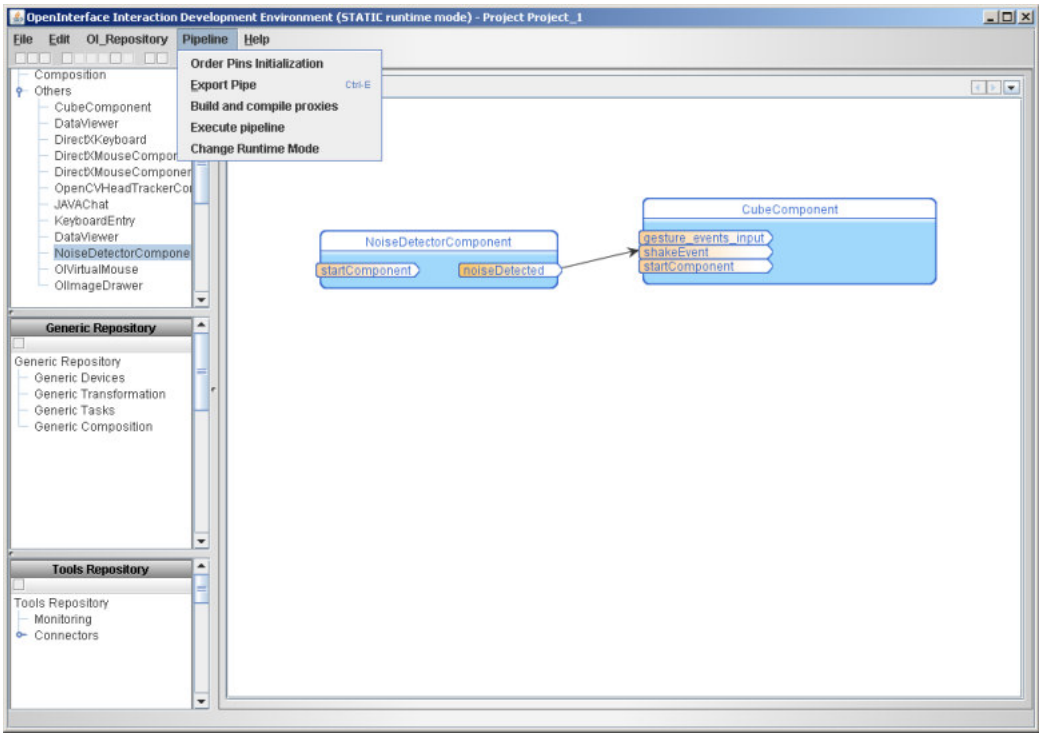


Figura 4.15. Editor gráfico OpenInterface.

4.5. Comentários Finais

Este capítulo introduziu os conceitos de multimodalidade, identificou os tipos de interação multimodal e propôs uma metodologia e uma ferramenta que permitem a concepção e prototipação de interfaces multimodais de forma fácil e intuitiva, garantindo uma boa estruturação do código graças ao uso de componentes.

Esperamos que este texto seja útil não apenas para pesquisadores e profissionais atuantes na área de realidade virtual, mas também para todos que trabalham com a concepção e desenvolvimento de interfaces, em geral. Com o crescente uso de aplicações interativas nas mais diversas plataformas (ATMs, telefones celulares,

notebooks, GPS, TV digital interativa, etc) e por usuários com perfis cada vez mais distintos, entendemos que o uso de interfaces multimodais representa atualmente o melhor caminho para garantir usabilidade no desenvolvimento de interfaces.

Agradecimentos

Este tutorial é parte dos resultados gerados pelo projeto de colaboração bilateral entre CNPq (Brasil) e FNRS (Bélgica) através do projeto MIDAS: *Multimodal Interaction Design for Advanced Systems*.

Referências

Baille, L., Schatz, R. Exploring Multimodality in the Laboratory and the Field. In: ICMI'2005, Trento, Italy, Ocotber 4-6, 2005.

Balbo, S., Coutaz, J. Salber, D. Towards Automatic Evaluation of Multimodal User Interfaces. Intelligent User Interfaces, Knowledge-Based Systems, 6(4). 2003. pp. 267-274.

Benoit, A., Bonnaud,, L., Capilier, A., Ngo, P., Trevisan, D. G., Lawson, L., Levacic, V., Mancas, C., Chanel, G. Multimodal Focus Attention and Stress Detection and feedback in an Augmented Driver Simulator In: Artificial Intelligence Applications and Innovations ed. : Springer Boston, 2006, v.204, p. 337-344.

Bolt, R. A. Put-that-there: Voice and gesture at the graphics interface. Computer Graphics, 14 (3), p262-270, 1980.

Bolt, R. E., Herranz, E. Two-handed gesture in multi-modal natural dialog. ACM UIST'92 Symposium on User Interface Software and Technology, ACM Press, 1992, 6-14.

Buxton, W., Myers, B. A. A study in two-handed input. Human Factors in Computing Systems, CHI'86 Conference Proceedings. ACM Press, 1986, 321-326.

Bouchet, Jullien; Mansoux, Benoit; Nigay, Laurence.A Component-Based Approach: ICARE. The Future of User Interface Design Tools. Trabalho apresentado no ACM CHI 2005 Workshop. Apr. 2005. 2p.

Bowman, D., Gabbard, J., and Hix, D. A Survey of Usability Evaluation in Virtual Environments: Classification and Comparison of Methods. Presence: Teleoperators and Virtual Environments, vol. 11, no. 4, 2002, pp. 404-424.

Coutaz, Joëlle; Caelen, Jean. A Taxonomy for Multimedia and Multimodal User Interfaces. In ERCIM (European Research Consortium for Informatics and Mathematics). p 143-148. Lisbonne, 1991.

Coutaz, Joëlle; Nigay, Laurence; Salber, Daniel. Multimodality from the User and System Perspectives. In Proc. ERCIM (European Research Consortium for Informatics and Mathematics), workhop on User Interface For All, Heraklion. 1995.

Dillon, R. F., Edey, J. D., Tombaugh, J. W. Measuring the true cost of command selection: techniques and results. Humanc Factos in Computing Systems, CHI'90 Conference Proceedings, ACM Press, 1990, 19-25.

Dybkjær, L., Bernsen, N. O., Minker, W. (2004) New Challenges in Usability Evaluation - Beyond Task-Oriented Spoken Dialogue Systems. In: Proceedings of ICSLP, (vol. III), pp 2261-2264.

Elting, Christian *et al.* Proceedings of the 5th international conference on Multimodal interfaces: Architecture and Implementation of Multimodal Plug and Play. Vancouver, British Columbia, Canada. Pg 93 – 100. ed. ACM. New York, NY. 2003.

Hinckley, K., Pausch, R., Proffitt, D., Kassel, N. F. Two-handed virtual manipulation. ACM Transactions on Computer-Human Interaction, 5 (3), 1998, 260-302.

Holzapfel, H., Nickler, K., Stiefelhagen, R. Implementatin and Evaluation of a Constraint-based Multimodal Fusion System for Speech and 3D Pointing gesture. In: ICMI'2004, State College, Pennsylvania, USA, October 13-15 2004. pp. 175-182.

Jöst, M., Häußler, J., Merdes, M., Malaka, R. Multimodal interaction for pedestrians: an evaluation study. IUI 2005: 59-66

Kabbash, P., Buxton, W., Sellen, A. Two-handed input in a compound task. Human Factors in Computing System CHI'94 Conference Proceedings, ACM Press, 1994, 417-423.

Kaster, T., Pfeiffer, M., Bauckhage, C. Combining Speech and Haptics for Intuitive and Efficient Navigation through Image Database. In Proc. ICMI'2003, Vancouver, Canada. November 5-7 2003. pp. 180-187.

Kjeldskov, J., Stage, J. (2004) New Techniques for Usability Evaluation of mobile systems. International Journal on Human-Computer Studies, 60 (5), pp 599-220.

Lawson, J-Y. Vanderdonckt, J. and Macq, B. Rapid Prototyping of Multimodal Interactive Applications Based on Off-The-Shelf Heterogeneous Components, Adjunct Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology, pp. 41-42, 2008.

Martin, J. C.. TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces. Intelligence and Multimodality in Multimedia interfaces. (ed.) John Lee, AAAI Press, 1998.

Nedel, L.; Freitas, C.M.D.S.; Jacob, L.; Pi, M.. Testing the Use of Egocentric Interactive Techniques in Immersive Virtual Environments. In: INTERACT 2003 - The IFIP Conference on Human-Computer Interaction, 2003, Zurich. Human-Computer Interaction: INTERACT´03. Amsterdam: IOS Press, 2003. p. 471-478.

Oviatt, S. Ten myths of multimodal interaction. Communications of the ACM, 1999, 42(11):74--81.

Oviatt, Sharon. Multimodal Interfaces. In: JACKO, J; SEARS A. In Handbook of Human-Computer Interaction. p. 286-301. Lawrence Erlbaum: New Jersey, 2002.

Poupyrev, S. Weghorst, M. Billinghurst, T. Ichikawa. Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. Proceedings of Eurographics, Computer Graphics Forum, 17(3), 1998. pp. 41-52

Reason, J. (1990). Human error. Cambridge, England: Cambridge University Press.

Rosa Jr, R. S.; Farias, M. A. C.; Trevisan, D. G. ; Nedel, L. P.; Freitas, C. M. D. S. . A Multimodal Medical Sculptor. In: INTERACT 2007, 2007, Rio de Janeiro. 11th IFIP TC 13 International Conference on Human-Computer Interaction, 2007. v. 4663. p. 637-640.)

Serrano, M., Nigay, L., Lawson, J. L., Ramsay, A., Murray-Smith, R., Denef, S. The OpenInterface framework: a tool for multimodal interaction.In CHI '08. ACM, New York, NY, 2008, 3501-3506.

Suhm, B., Myers, B., Waibel, A. Model-based and Empirical Evaluation of Multimodal Interactive Error Correction. In Proc. CHI99, Pittsburgh, USA, 15-20 May 1999. pp. 584-591.

Tavares, R. *et al.* Eletromagnetismo: Objetos de aprendizagem e a construção de significados baseados em um ambiente de múltiplas representações. Tecnologia e Educação para Todos. Trabalho apresentado no XIX SBIE - Simpósio Brasileiro de Informática na Educação, Fortaleza, 2008. 4p.

Tse, Edward; Shen, Chia; Greenberg, Saul; Forlines, Clifton. Enabling interaction with single user applications through speech and gestures on a multi-user tabletop. In Proceedings of the Working Conference on Advanced Visual interfaces. AVI '06. ACM, New York, NY, p.336-343. 2005.

Tse, Edward; Greenberg, Saul; Shen, Chia. Exploring Interaction with Multi User Speech and Whole Handed Gestures on a Digital Table. Demonstration Adjunct proceedings ACM UIST. Montreux: Suíça. Oct 2006a. 2p.

Tse, Edward; Greemberg, Saul; Shen, Chia; Forlines, Clifton. Multimodal Multiplayer Tabletop Gaming. International Workshop on Pervasive Gaming Applications (PerGames). University of Calgary, Computer Science, Science. May. 2006b. 12p.

Tzovaras, D. Multimodal User Interfaces From Signals to Interaction. DOI: 10.1007/978-3-540-78345-9. (ed.) Springer. 2008. 315p.

Zhai, S., Barton, A. S., Selker, T. Improving browsing performance: a study of four input devices for scrolling and pointing tasks. Proccedings of INTERACT'97: The IFIP Conference on Human-Computer Interaction, 1997, 286-292.

Wahlster, W.. Towards Symmetric Multimodality: Fusion and Fission of Speech, Gesture and Facial Expression. In KI 2003: Advances in Artificial Intelligence. Vol. 2821. ed. Springer. Heidelberg, Berlin. September, 2003

Westeyn, T. et al. Georgia tech gesture toolkit: supporting experiments in gesture recognition. Proc. of ICMI03 (2003), 85–92.

Biografias dos autores

(em ordem alfabética)

Carla M. Dal Sasso Freitas é doutora em Ciência da Computação pela UFRGS (1994), onde é professora desde 1980. Sua área principal de pesquisa é visualização interativa de dados, incluindo tanto os aspectos de rendering associados às técnicas de visualização de dados volumétricos e de informações como os aspectos de interação necessários para o efetivo emprego das técnicas pelos usuários. Também tem trabalhado intensamente em métodos de avaliação de usabilidade de técnicas de visualização. No contexto de interação humano-computador tem orientado alunos de mestrado e doutorado na área e publicado trabalhos nessas três linhas: desenvolvimento de interfaces para visualização de dados e de informações, desenvolvimento de técnicas de interação 3D e sua avaliação e avaliação de usabilidade de técnicas de visualização. Contato: carla@inf.ufrgs.br

Cléber Gimenez Corrêa é Mestre em Ciência da Computação pelo Centro Universitário Eurípides de Marília (UNIVEM), título obtido em 2008; Graduado em Tecnologia em Processamento de Dados pela Faculdade de Tecnologia de São Paulo, Extensão Ourinhos (FATEC - Ourinhos), no ano de 2002. O projeto de mestrado consistiu na implementação e avaliação auxiliada por profissionais da área, de interação em um sistema de RV para treinamento médico, envolvendo dispositivos convencionais e não convencionais, e técnicas de integração de linguagens de programação (Java e C/C++). Áreas de Interesse: Realidade Virtual, Treinamento Médico, Interação com Dispositivos Convencionais e Não Convencionais. Filiação: LApIS – Laboratório de Aplicações de Informática em Saúde (EACH/USP). Contato: correacleber@yahoo.com.br

Daniela Gorski Trevisan é doutora em Ciências Aplicadas pela Université catholique de Louvain, UCL, Bélgica (2006) tendo desenvolvido pesquisas na área de interfaces multimodais e sistemas de realidade mista.. Em 2006 fez parte do consórcio de desenvolvimento da plataforma multimodal OpenInterface e em 2007 atuou como pós-doutoranda júnior do CNPq junto ao grupo de Computação Gráfica da UFRGS e atualmente, é professora efetiva no curso de Ciência da Computação na Universidade Estadual de Santa Catarina. Trabalhos recentes incluem o desenvolvimento e avaliação de usabilidade de novas técnicas de interação para ambientes 3D aplicadas a área médica e de entretenimento. Contato: dtrevisan@joinville.udesc.br

Fátima de Lourdes dos Santos Nunes é professora da Escola de Artes Ciências e Humanidases (EACH), da Universidade de São Paulo (USP). É Doutora em Física Computacional (IFSC/USP), – área de processamento de imagens e fez Pós-Doutorado em Engenharia Elétrica – área de Visão Computacional (EESC/USP). Possui Mestrado em Engenharia Elétrica (EESC/USP) – área de processamento de imagens e Bacharelado em Ciência da Computação (Unesp/Bauru). Tem experiência nas áreas de processamento de imagens,

realidade virtual e banco de dados, atuando principalmente na construção de ferramentas para treinamento médico, em processamento de imagens médicas e recuperação de imagens baseada em conteúdo. Atualmente coordena o Laboratório de Aplicações de Informática em Saúde (LApIS – EACH/USP). Tem bolsa de produtividade nível 2 do CNPq. É membro da Comissão Especial de Realidade Virtual e vice-coordenadora da Comissão Especial de Computação Aplicada à Saúde, ambas da Sociedade Brasileira de Computação. Contato: fatima.nunes@usp.br

Francisco Simões é mestrando em Ciência da Computação no Centro de Informática-UFPE desde 2009 e pesquisador no Grupo de Pesquisa em Realidade Virtual e Multimídia (GRVM-UFPE) desde 2007. Atuando principalmente com Computação Gráfica, Visão Computacional e Realidade Aumentada, ministrou cursos e publicou diversos trabalhos em conferências e periódicos nacionais e internacionais. Como Cientista da Computação possui sólidos conhecimentos nas linguagens de programação C/C++ e Java, no ambiente de desenvolvimento Matlab® e nas bibliotecas VXL, OpenCV, OGRE, OpenGL e ARToolkit. Contato: fpms@cin.ufpe.br

João Paulo Lima está cursando o mestrado em Ciência da Computação no CIn-UFPE desde 2008. É um pesquisador do GRVM desde 2004, tendo participado de vários projetos nas áreas de Realidade Aumentada e Rastreamento 3D. João publicou 2 artigos em periódico, 1 capítulo de livro e 16 artigos em conferências, todos relacionados a Realidade Aumentada e Visão Computacional. Ele também ministrou 3 cursos. Possui vasta experiência no desenvolvimento de aplicações utilizando as bibliotecas OpenCV, VXL, ARToolKit, ARToolKitPlus, OpenGL e OGRE, assim como as linguagens de programação C e C++. Contato: jpsml@cin.ufpe.br.

Judith Kelner recebeu seu título de Mestre em Ciência da Computação do CIn-UFPE, em 1981. Concluiu o Doutorado em Ciência da Computação na University of Kent em Canterbury, em 1993. Atualmente, é professora adjunta IV no CIn-UFPE, coordenadora do GRVM e vice-coordenadora da CERV-SBC. Envolveu-se na administração do CIn-UFPE por quase dez anos, ocupando cargos de coordenadora do curso de extensão, diretora e vice-diretora. Possui bolsa de produtividade em pesquisa do CNPq, atuando em Realidade Virtual/Aumentada, Interfaces Avançadas, Redes de Ambiente, Qualidade de Serviços e Multimídia. Publicou vários artigos em periódicos e conferências nacionais/internacionais e capítulos de livro. Contato: jk@cin.ufpe.br.

Liliane dos Santos Machado é professora do Departamento de Informática da Universidade Federal da Paraíba. Doutora em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo (2003), Mestre em Computação Aplicada pelo Instituto Nacional d e Pesquisas Espaciais (1997) e Bacharel em Ciências da Computação (1994), coordena projetos na linha de realidade virtual, jogos e tecnologia educacional junto ao Laboratório de Tecnologias para o Ensino Virtual e Estatística (LabTEVE) da UFPB. Possui diversos trabalhos publicados e produtos registrados. Membro da IEEE, ACM e SBC, além de membro da Comissão Especial de Realidade Virtual, seu interesse de pesquisa compreende as áreas de realidade virtual, sistemas hápticos, sistemas colaborativos, avaliação online, jogos e tecnologias educacionais. Contato: liliane@di.ufpb.br.

Lucas Silva Figueiredo é graduando em Ciência da Computação no Centro de Informática - UFPE. Desde o início de 2008 é pesquisador júnior do GRVM-UFPE. Como principais áreas de atuação tem Realidade Aumentada, Rastreamento 3D e Visão Computacional. Contato: lsf@cin.ufpe.br.

Luciana Porcher Nedel é doutora em Ciência da Computação pelo Swiss Federal Institute of Technology (EPFL), Lausanne, Suiça (1998) e atua como professora adjunta junto à Universidade Federal do Rio Grande do Sul (UFRGS) desde 2002. Desde então vem desenvolvendo atividades de pesquisa junto ao grupo de computação gráfica do Instituto de Informática. Suas principais áreas de interesse são: visualização interativa, sistemas multi-displays, interação não-convencional e animação de personagens em ambientes virtuais. Em 2005, realizou estágio sabático de 2 meses na Université Paul Sabatier (IRIT) em Toulouse, França e atuou como professora convidada na Université catholique de Louvain (UCL), na Bélgica, na área de interação, também por 2 meses. Publicou 8 artigos em periódicos, mais de 50 em anais de eventos e 9 capítulos de livros. Contato: nedel@inf.ufrgs.br

Ronei Marcos de Moraes é professor associado da Universidade Federal da Paraíba. Possui pós-doutorado na Engenharia Elétrica da Escola Politécnica da Universidade de São Paulo (2001), doutorado em Computação Aplicada pelo Instituto Nacional de Pesquisas Espaciais (1998), mestrado em Engenharia Elétrica pela Universidade Federal da Paraíba (1992) e graduação em Estatística pela Universidade Estadual de Campinas (1988). Possui bolsa de Produtividade em Desenvolvimento Tecnológico e Extensão Inovadora do CNPq, sendo atuante na área interdisciplinar, nos seguintes temas: métodos estatísticos e geoestatísticos em saúde pública, conjuntos nebulosos, ensino virtual, educação a distância, jogos educacionais, realidade virtual e métodos de avaliação de treinamento baseados em realidade virtual. Possui diversos trabalhos publicados e produtos registrados. Participa do corpo editorial do Open Virtual Reality Journal. Contato: ronei@de.ufpb.br.

Veronica Teichrieb é professora adjunta no Centro de Informática da Universidade Federal de Pernambuco – CIn-UFPE, onde co-coordena o Grupo de Pesquisa em Realidade Virtual e Multimídia – GRVM. Seus interesses de pesquisa incluem Realidade Aumentada, Modelagem e Simulação Gráfica e Física e Interação 3D. Ela publicou diversos trabalhos em periódicos e eventos nacionais e internacionais, e ministrou vários tutoriais e cursos. Veronica recebeu seus títulos de Mestre e Doutora em Ciência da Computação do CIn-UFPE, em 1999 e 2004, respectivamente. Em 2001, ela foi pesquisadora visitante na Aero-Sensing Radarsysteme GmbH, Alemanha, durante um doutorado-sanduíche. Contato: vt@cin.ufpe.br.

ABORDAGENS PRÁTICAS DE REALIDADE VIRTUAL E AUMENTADA

LIVRO DOS MINICURSOS

XI Symposium on Virtual and Augmented Reality

SVR 2009

25 a 28 de Maio de 2009

Porto Alegre – RS

ISBN 857669239-2