Dynamic Adjustment of Stereo Parameters for Virtual Reality Tools

Felipe Carvalho, Daniel R. Trindade, Peter F. Dam, Alberto Raposo
Tecgraf / Department of Informatics
Pontifical Catholic University of Rio de Janeiro
Email: {kamel, danielrt, peter, abraposo}@tecgraf.puc-rio.br

Ismael H. F. dos Santos
CEMPES Petrobras, Rio de Janeiro, RJ - Brasil
Email: ismaelh@petrobras.com.br

***Abstract*—Due to emerging new technologies in the development of interactive 3D applications (eg games and virtual reality), stereoscopic visualization is becoming a common feature. However, this fact does not solve some problems (nausea and headaches - cybersickness) related with the generation of this type of visualization. Some parameters have to be carefully chosen to create a comfortable stereo view, for example, eye distance, zero parallax plane distance, and the treatment of partially clipped objects in negative parallax. This paper presents a technique based on a CubeMap structure to dynamically adjust stereo parameters during the usage of two virtual reality tools in multi-scale 3D scenarios.**

***Keywords*-virtual reality; interaction; 3D techniques; stereoscopy;**

I. INTRODUCTION

Currently new stereoscopic technologies are increasingly present in our lives. What was once just fun colored glasses, today are modern solutions that allow you to view 3D graphics without loss of visual characteristics such as brightness or color.

Besides cinema's entertainment, stereoscopic visualization reached new areas, among them are games and virtual reality. These two areas share the use of interactive 3D graphics, which enables users to change the virtual scenes at any time.

When someone looks at an object in the real world, the eyes focus on the object and also converge (turning an eye toward the other) on it. After the process of focusing and convergence of the eyes, the brain fuses the two images (from the left and right eyes) into one, allowing viewing stereoscopic or depth. In Figure 1 there are the basic elements to understand the stereoscopic configuration process inside a 3D interactive application. Often an application needs to set two parameters: eye distance and fusion distance. The first parameter is used as an offset to create two different virtual view points, one for the right eye and other for the left.

Since the parallax is the distance between the corresponding points from the images of the right and left eyes, three important elements are identified based on the value of this distance: zero parallax plane, positive parallax region and negative parallax region. These different parallaxes indicate different distances from the viewer and

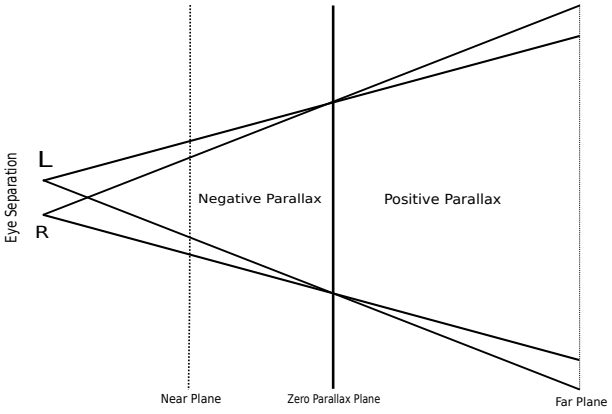

Figure 1. Stereoscopic elements.

the second parameter refers to it. The fusion distance is the position of the zero parallax plane. A virtual object near this plane appears to be in plane of the screen, objects inside the positive parallax appear to be behind the screen, and those which appear to be in front of the screens belong to the negative parallax region.

Objects with high parallax values (positive or negative) cause discomfort because the eyes have a hard time converging on them. Very often this is treated using the clipping planes, i.e., near plane is used to decrease the negative parallax region while the far plane is used to decrease the positive parallax region.

The benefits of stereo are well known [1], [2], [3], but the development of applications of this nature must take care to avoid problems that could cause discomfort to users [4], [5], [6], such as nausea, headaches, etc. One of the causes of these efects is the convergence / accommodation problem [7], [8]. When someone looks at a screen or monitor, the eyes are accommodated on the plane of the screen, but are converged using the parallax between the left and right images. For some people, this is perceived as discomfort. To minimize the negative effects of this problem, convergence plane must be positioned on the screen or monitor, and one should carefully choose the distance between the eyes to avoid a high disparity in the projected images.

Another important problem is the conflicts between the clipped objects and parallax depth [9]. If an object has a

negative parallax and is being blocked (partially clipped) by the borders of the screen, the sensation of stereoscopic depth is seriously impaired. This problem occurs because of the conflict between the resulting three-dimensional depth of the negative parallax interposed between the scene objects and the screen. One solution to this problem is to place the near plane on the zero parallax plane position, but this approach cancels any visualization in the negative parallax region.

In virtual reality applications, often these problems are found in tasks such as navigation or object manipulation. For instance, while navigating in a 3D scene, composed by objects of different scales (multi-scale environment [10]), if the eye distance is maintained constant throughout the scenario it will cause problems of convergence. This is explained by the fact that an eye distance X used in a virtual room, can become unreasonably large if the user navigates into places smaller than this room. In these others places, the unmodified eye distance becomes too separated and resulted parallax becomes larger.

The objective of this work is to propose a solution capable of dynamically adjusting the stereo parameters in virtual reality applications. The techniques presented here are based on a data structure called cubemap, initially proposed by [11]. Also some virtual reality navigation techniques are discussed, since these have a large influence in the stereo adjustment we propose.

This paper is structured in the following manner: section II presents related work. Section III briefly describes the cubemap structure. Section IV describes improvements made to some navigation techniques that are important for the stereo parameter's dynamic adjustment, which is presented in section V. Finally, section VI concludes the paper.

II. Related Work

The attempt to automatically adjust stereo parameters was an object of study for several authors. In their majority these works use the scene's depth information to establish the eye separation and the zero parallax plane.

Jones et al [12] created a method that takes into account the geometry of the real world to determinate the stereo parameters. Based on previous studies, they established the maximum depth interval for a given display in which the majority of people would be capable of merging two images. With this data in hands, along with the distance between the eyes of a real person and their distance to the screen, they are capable of calculating the virtual eye separation. Their idea is to map the virtual world scene to the real world, in a way that the relationship between the disparities of the objects closest and furthest to the camera stays as close as possible to the relationship existing in the real world.

In his work Holliman [1] uses the same concept as [12], but introduces the idea of an interest region. Instead of linearly mapping the scene to the noticeable depth interval, Holliman creates three regions: the first, closer to the near clipping plane; the second, containing the zero parallax plane; and the third, closer to the far clipping plane. The second region is the largest, and concentrates the objects of interest in the scene. With this, Holliman has the advantage of reducing the distortion caused by the method presented in [12], which uses linear mapping. However in the two other regions there still is distortion. Furthermore, this method depends on the display type and environment that is being used.

Ware et al [13] conducted experiments to identify how users adjust stereoscopy parameters and how those users would react to the dynamic adjustment of those values. First, tests were performed with users to know what the maximum rate of changes in stereo that a person could support. Next, the users were asked to manually adjust the stereo parameters until the visualization was comfortable. With this data, Ware et al created a two pass algorithm to appropriately adjust the virtual eye separation and zero parallax plane distance. This method takes into account the direct input from users who participated in the tests and, because of this, does not guarantee appropriate behavior for all people.

III. The Cubemap

The purpose of the cubemap, as proposed by McCrae et al. [11], is to provide information about the virtual environment at a given moment. Given a camera position, this structure is constructed from 6 rendering passes, each in a different direction in order to cover the whole environment. The FOV of the camera is 90^0, therefore the combination of the 6 resulting frustums yields a cube. At each pass, a shader is used to calculate the distance from the fragment generated to the camera. The computed distance values are normalized in relation to the *near* and *far* values, and stored in the *alpha* channel of the positions related to the fragments. Rendering is made in 32-bit float images. Such procedure is performed at each frame, or each time the camera position changes. The image resolution used for rendering does not have to be high, since only an estimation needs to be obtained.

In our implementation of the cubemap, we also use the *RGB* channels of the images to store a unit vector pointing from the position of the generated fragment to the camera. This simplifies the construction of the force map we use to avoid collisions (section IV-B).

IV. Navigation Techniques

In this section we will present three navigation techniques. Althouth they are not directly connected to stereoscopy, they are important to avoid some problems in this sense. These techniques were first proposed by Maccrae et al [11] and were evaluated by [14].

A. Fly with Automatic Speed Adjustment

Navigation speed is related to the scale of the environments to be explored. Larger environments require faster speeds, while the opposite is more convenient on smaller scales.For instance, the camera speed is expected to be higher when navigating from one planet to another, but much lower when navigating from a city to another.

In several applications, the scale of the virtual world does not change much and is well known, allowing a fixed navigation speed to be used. This is the case of many games, for instance. Multiscale environments, however, require a way to estimate the current scale in order to adjust the navigation speed accordingly.

McCrae et al. [11] use $minDist$, the minimum distance in the cubemap, as an estimation to determine the current scale the camera is at. Based on that, first we developed a *fly* tool which could be controlled by the user by pressing the arrow keys of the keyboard to move the camera while guiding the direction of the motion with the mouse movements. Navigation speed was adjusted automatically according to equation 1.

$$V_{nav} = k\ minDist \tag{1}$$

where V_{nav} is the adjusted navigation speed and k is a parameter of unit 1/s that causes an increase or reduction in the acceleration applied to the camera. We noticed that two situations caused discomfort and disorientation for some users. The first situation is when k is too high. In this situation, when users move away from the geometry, the camera accelerates too quickly, producing an effect similar to teleporting and making users lose their location. The second situation happens for low k values, which can considerably increase the time required to reach the intended destination, making the navigation tedious. Thus, the value of k should be selected in a way that it balances these extreme situations.

B. Collision Detection and Treatment

Not allowing the camera to cut through objects in a virtual environment can be crucial in some situations. In immersive environments, for example, colliding with an object can halt the immersion and leave the user disoriented. When stereoscopy also exist, the collisions could break the stereo effect.

McCrae et al. [11] used information from the distance cube to obtain a collision factor that causes the camera to smoothly dodge the closest objects. The idea is that each point in the cubemap located at a distance smaller than a given radius r produces a repulsion factor given by:

$$F(x,y,i) = w(dist(x,y,i))\ norm(pos(x,y,i) - camPos) \tag{2}$$

$$w(d) = e^{\frac{(r-d)^2}{\sigma^2}} \tag{3}$$

where $F(x,y,i)$ is the repulsion factor produced by point p referring to position (x,y) of image i of the cubemap. Value $dist(x,y,i)$ is the distance from p to the camera. The term $pos(x,y,i)$ is the position of p in world space, and $camPos$ is the camera position. Function $norm(v)$ indicates the normalized vector of v. In equation 3, σ is a parameter that indicates the smoothness of the collision factor. The higher σ is, the smoother will be the calculated factor. Considering a spherical region with radius r and centered on the camera position, equation 3 results in determining a collision penalty that grows exponentially from the moment when point p enters this region.

The repulsion factors referring to equation 2 are computed for each position in the cubemap where $d < r$ and then are combined into a single factor:

$$F_{collision} = \frac{1}{6\ cubeRes^2} \sum_{x,y,i} F(x,y,i) \tag{4}$$

where $cubeRes$ is the resolution of the distance cube.

When we applied the factor given by equation 4 to the camera, the *fly* tool behaved as described in the previous section: as the camera moves, $F_{collision}$ ensured that it smoothly deviate the objects that crossed its path. The user then can navigate through the environment without worrying about choosing a collision-free path as the system is in charge of this task.

C. Automatic adjustment of clipping planes

The non-configuration of the clipping planes can lead to problems ranging from the undue clipping of objects to appearance of artifacts on objects distant from the camera. When in stereo, these problems become even worse, becoming more evident to the users. Therefore the correct adjustment of the clipping planes is an important requirement to create a correct stereo effect.

Maccrae et al [11] developed a technique of automatic adjustment of the clipping planes. It consists in using the $minDist$ information available in the CubeMap to select optimal values for the *near* and *far* parameters. Their idea is to maintain the visible geometry always between *near* and *far*. At each frame the need to update the clipping plane is checked based on the following equations:

$$n = \begin{cases} \alpha n & \text{if } minDist < An \\ \beta n & \text{if } minDist > Bn \\ n & \text{otherwise} \end{cases} \tag{5}$$

$$f = Cn \tag{6}$$

In the 5 and 6 equations, n is the *near* value, f is the *far* value, $minDist$ is the minimum distance (not normalized) stored in the distance cube, α, β, A, B are constants that indicate when and how the clipping planes should be adjusted. In the implementation present in [11], as well as in our test applications, the values $\alpha = 0.75$, $\beta = 1.5$, $A = 2$, $B = 10$ yielded satisfactory results. Finally, C expresses the ratio between n and f. C should be chosen so that objects

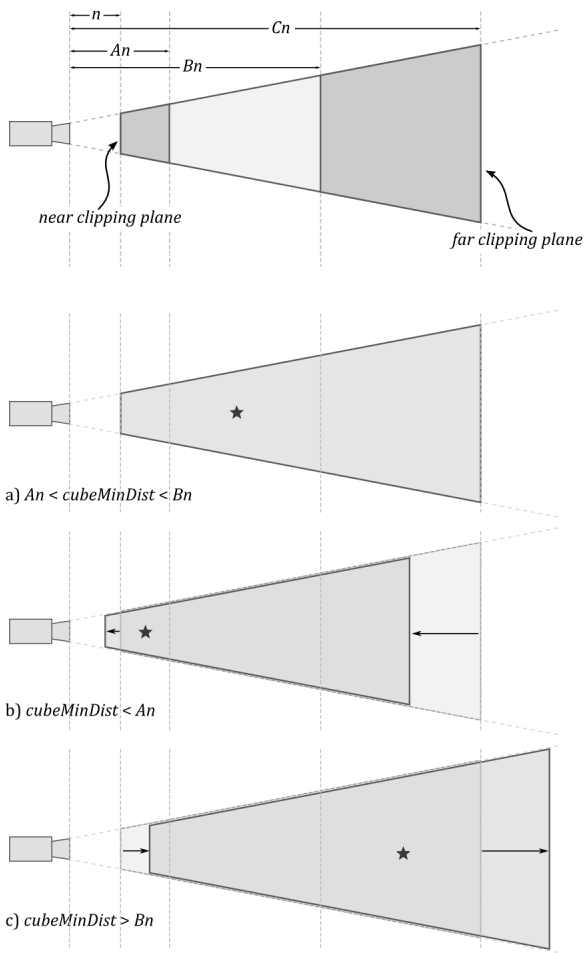

Figure 2. Automatic adjustment of clipping planes [11]

near *far* plane will not be clipped. At the same time C should not be too high or else that would cause a loss of precision in the *depth buffer*. In our applications, C was fixed at 10000.

The figure 2, taken from [11], illustrates how the equations 5 and 6 act on the values of the clipping planes. In case *(a)*, the closest point in the scene, represented by a star, is located in the $[An, Bn]$ interval, which doesn't require any modification to the clipping planes. This is the ideal case. In case *(b)*, $minDist$ has become smaller than An, in other words, it is closer to the *near* plane. The clipping planes are then adjusted to lower values to ensure, with some safety margin, that the scene's geometry stays inside the view frustum. Similarly in case *(c)* the clipping planes are adjusted to larger values, since $minDist$ is further away from the camera ($minDist > Bn$).

V. Dynamic adjustment of stereo parameters

In our applications, we have primarily two ways that the user can interact with the scene: one is using the *fly* navigation tool and the other is using the *examine* manipulator. As each on of these techniques has its own characteristics, we developed two types of dynamic adjustment of stereo parameters.

A. Stereo Adjustment for Fly

As seen in previous sections, a large part of the navigation parameters are adjusted based on the lowest value of the CubeMap at a certain moment. Our approach is to use this value to dynamically adjust the stereo parameters as well:

$$Dist_{pzero} = minDist \tag{7}$$

$$Eye_{sep} = k * minDist \tag{8}$$

Where $Dist_{pzero}$ is the distance from the camera to the parallax zero plane, $minDist$ is the lowest value in the CubeMap, Eye_{sep} is the virtual distance between the eyes and k is a constant. This adjustment is dynamically made when fly navigation tool is active.

The 7 equation creates a stereoscopy effect where the scene's objects are in positive parallax most of the time. This behavior is similar to the one observed in [13] and is based on providing a comfortable depth sensation to the user.

The approach used here is different from the one in [13] since the latter uses the nearest point based on a scan in the main camera's depth buffer. We use $minDist$, which represents the nearest point to the camera taking into consideration a 360^o view. This implies that $minDist$ could refer to a point that is not localized in the user's field of view. The use of $minDist$ may not seem to make sense, since the stereo parameters would be adjusted based on what the user can see. However, the way the fly tool works may create some problems that cannot be solved with only the front camera's depth information.

As described previously, the fly tool allows the user to rotate the camera by 360^o on it's own axis. Because of that an object that previously wasn't in the camera's field of view may abruptly appear in case the rotation is fast.

In his work, [13] performed tests that show that there is a maximum rate at which the stereo parameters may be adjusted without being noticeable. To respect this restriction they interpolated the parameters to smoothen the consecutive adjustments so the user would not notice it. This smoothening, however, conflicts with the previous note: when an object abruptly appeared in the field of view the user could see two separate images during the time that the parameters were being interpolated. For this reason, [13] abandoned the interpolation.

When $minDist$ is used, the previously described problems do not occur. Because $minDist$ represents a global minimum distance to a certain camera position, the camera's rotation does not cause an abrupt change to the stereo parameters, due to $minDist$ not being

changed. Furthermore, the appearing of objects in the camera's frustum does not break the stereoscopy because the parameters have already been adjusted taking into consideration the possibility of that object entering the frustum. In other words, using $minDist$ as a base for the stereoscopy's dynamic adjustment corresponds to anticipating actions the user may take. This feature is an advantage when we take into account that virtual reality applications are dynamic in the sense of providing the user with tools for navigation and manipulation. This is not the case, for example, of stereoscopy in other applications, such as cinema, where the parameters may be adjusted according to what the movie's director wants people to see.

Due to the stereo parameters maximum update rate restriction we decided to perform adjustments only when the clipping planes are adjusted. This way every time the camera reaches a point where the clipping planes need to be adjusted, the stereo is also adjusted and all objects in the scene are placed in positive parallax. From this moment some objects may enter in the negative parallax region if the camera moves closer to the position that yielded $minDist$. This may lead to objects in negative parallax being clipped by screen borders. We consider this problem irrelevant, however, since they occurs for a briefly. Also, our applications are built for immersive systems such as caves and large screens. Dueto the size of these environments, this effect can be ignored since these areas are almost always out of the user's field of view.

The eye separation is adjusted according to the 8 equation. The idea of this is to consider the current scale to set Eye_{sep}. For example, if the camera is in a room scale, it might be reasonable that Eye_{sep} have a value that is in accordance to the eye separation of a real person. But if the camera is viewing an entire planet, then Eye_{sep} should be reconfigured to larger value. With the adjustment provided with equation 8, Eye_{sep} is appropriately updated as the camera moves in the scene. The k constant is choosed to provide the user with a comfortable depth sensation. In our tests we discovered that $k = 0.01$ provides a satisfactory effect.

Finally, it is important to mention the relation between the automatic navigation velocity adjustment and the collision detection described in sections IV-A and IV-B with the dynamic stereo adjustment described here.

One of the requirements for this adjustment to work is that the variation of values in the cubemap throughout successive frames should not be high, so we can guarantee that values of $minDist$ will have a smooth variation. This is only achieved thanks to the continued adjusting of the navigation velocity, which avoids the camera stuttering. The figure 3 shows the curve generated by the successive values of $minDist$ for a given camera path when the velocity adjustment is activated. It is possible to observe that the curve is smooth and shows no peaks. This also contributes to the proper functioning of the clipping planes and collision detection.

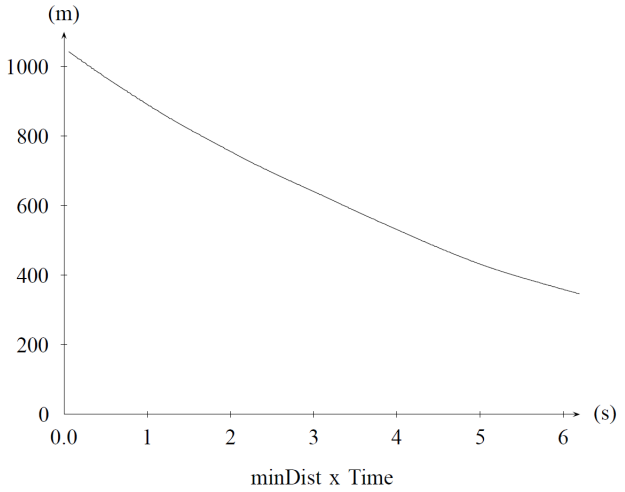

Figure 3. Graph representing the behavior of the $minDist$ curve.

The collision detection is also fundamental to the stereo's correctness. Without this the camera would be able to pass through objects. This would cause objects to be clipped by the near plane, causing a break in the stereo and, consequently, discomfort to the user. The collision detection described in IV-B avoids that this happens, causing the camera to smoothly swerve around objects in it's path.

An implementation of the fly tool with automatic adjustment was done in Unity3D [15]. In the figure 4 there is a screenshot of the developed application showing an object (castle tower) partially in negative parallax. Another screenshot was taken to view the parallax in relation to the global minimum, in this case, placing it before the castle (Figure 5).

B. Stereo Adjustment for Examine

The examine tool is based on rotating around a target object, an object in which one is interested in visually inspecting. The authors of this work believe that for this it is necessary to explicitly indicate an object, which makes it easier to adapt the stereoscopy parameters using object attributes such as position and bounding box.

In this approach we decided to position the zero parallax plane on the object's center. This creates an effect where half of it is coming out of the screen and half is going into it. Besides that, the interaxial eye distance is determined assuming a certain constant multiplied by the chosen object's bounding box. For our tests we used a value of 0,18 for the constant.

In figure 6 are some images taken from Environ software [16] where this approach has been implemented. A representation of the viewing cone has been created inside the 3D scenario, and this representation is scaled according to the chosen object. Along with this scale adjustment there is the repositioning of the parallax plane. The use of a

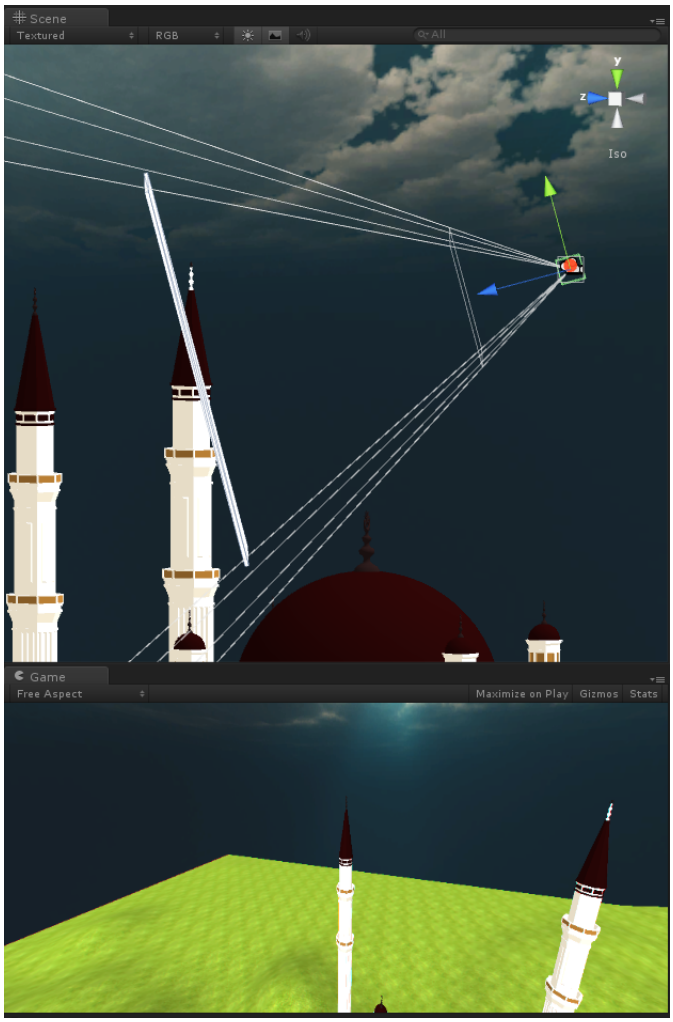

Figure 4. Fly tool on Unity3D.

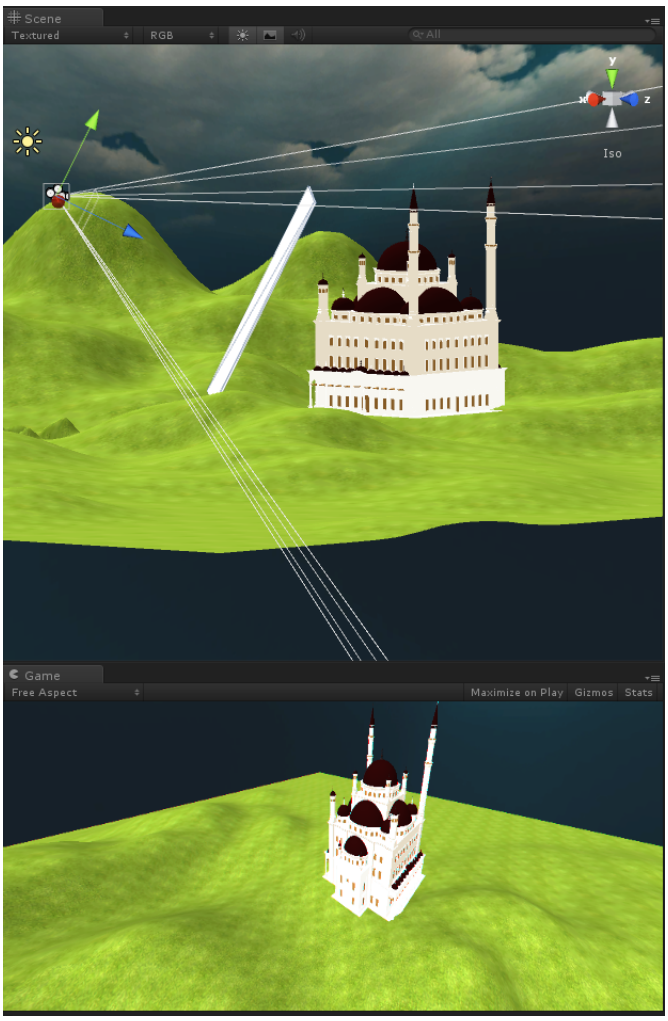

Figure 5. Fly tool on Unity3D.

linear interpolation softens the transition of the stereoscopy parameters from one chosen object to another. This was proven necessary due to the different size of objects in the same scene.

VI. Conclusion

Development of applications with stereoscopy support requires important care, or else physiological problems may be caused to the users such as nausea and headaches. Depending on the 3D content that will be shown, dynamic adjustment of stereoscopy parameters may be necessary so that the visualization remains comfortable in all points of the 3D scene, for example, in multi-scale environments.

This work presented some approaches of stereoscopy adjustment in two tools used in virtual reality applications: fly and examine. Both respectively contemplate the navigation and 3D manipulation tasks. For the fly tool a CubeMap structure was used to automatically adjust the stereoscopy parameters. Basically, the structure supplied a global minimum distance for the user's camera, and this distance is used as basis to position the zero parallax plane, and likewise the eye distance. This approach was implemented in an application developed using Unity3D.

For the examine tool we described how to obtain a comfortable scale to expand the eye distance and zero parallax distance. It was assumed, in this case, that the user would choose an object and then, based on that, the bounding box size was used as reference to determine the eye distance. Finally the zero parallax plane is placed in the same position as the chosen object.

We believe that our approach produces a comfortable stereo effect and helps users in the sense that they do not need to configure the parameters. In our primary tests we found that the graphical results were well received by users. As future work, we are planning to conduct more accurate usability tests to evaluate our solution.

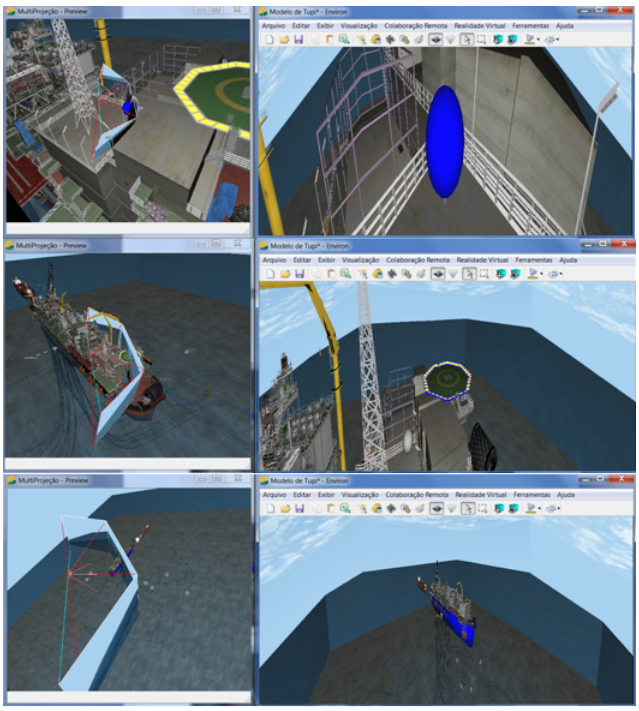

Figure 6. Examine tool on Environ.

ACKNOWLEDGEMENTS

Tecgraf is a laboratory mainly supported by Petrobras. Alberto Raposo receives an individual grant by FAPERJ.

REFERENCES

[1] N. Holliman, "3d display systems," *Handbook of Optoelectronics*, 2004.

[2] L. Lipton, "Stereographics developers handbook," *www.stereographics.com/support/downloads support/handbook.pdf*, 1997.

[3] D. McAllister, "Stereo computer graphics and other true 3d technologies," *Princeton University Press*, 1993.

[4] M. Wopking, "Viewing comfort with stereoscopic pictures: An experimental study on the subjective effects of disparity magnitude and depth of focus," *Journal of the Society for Information Display*, vol. 3, no. 3, pp. 101–103, 1995.

[5] Y.-Y. Yeh and L. D. Silverstein, "Limits of fusion and depth judgment in stereoscopic color displays," *Hum. Factors*, vol. 32, pp. 45–60, January 1990.

[6] A. Woods, T. Docherty, and R. Koch, "Image distortions in stereoscopic video systems," in *STEREOSCOPIC DISPLAYS AND APPLICATIONS*, 1993.

[7] C. Ware, C. Gobrecht, and M. Paton, "Dynamic adjustment of stereo display parameters," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, pp. 56–65, 1998.

[8] K. Noro, "Industrial application of virtual reality and possible health problems," *Jpn. J. Ergonomica*, vol. 29, pp. 126–129, 1993.

[9] N. A. Valyrus, *Stereoscopy*. London, U.K.: Focal Press, 1966.

[10] G. W. Furnas and B. B. Bederson, "Space-scale diagrams: understanding multiscale interfaces," in *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995, pp. 234–241.

[11] J. McCrae, I. Mordatch, M. Glueck, and A. Khan, "Multiscale 3d navigation," in *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*. New York, NY, USA: ACM, 2009, pp. 7–14.

[12] G. Jones, D. Lee, N. Holliman, and D. Ezra, "Controlling perceived depth in stereoscopic images," in *STEREOSCOPIC DISPLAYS AND VIRTUAL REALITY SYSTEMS VIII*, 2001, pp. 200–1.

[13] C. Ware, C. Gobrecht, and M. Paton, "Dynamic adjustment of stereo display parameters," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, pp. 56–65, 1998.

[14] D. R. Trindade and A. B. Raposo, "Improving 3d navigation in multiscale environments using cubemap-based techniques," in *ACM SAC '2011: Proceedings of the Symposium On Applied Computing*. ACM Press/Addison-Wesley Publishing Co., 2011.

[15] U. Technologies. (2011, Mar.) Unity3d. [Online]. Available: http://unity3d.com/

[16] A. Raposo, I. Santos, L. Soares, G. Wagner, E. Corseuil, and M. Gattass, "Environ: Integrating vr and cad in engineering projects," *IEEE Comput. Graph. Appl.*, vol. 29, pp. 91–95, November 2009.