Blender and Immersive Gaming in a Hemispherical Dome

Paul David Bourke
WASP/iVEC, University Of Western Australia
35 Stirling Hwy, Crawley
Perth, 6009. Australia.
Phone: 61 8 64888097
Email: paul.bourke@uwa.edu.au

Dalai Quintanilha Felinto
EAU / Universidade Federal Fluminense
Rua Passo da Pátria, 156 - São Domingos
Niterói / Rio de Janeiro, Brazil

Abstract

In the following we will discuss a cost effective immersive gaming environment and the implementation in Blender [1], an open source game engine. This extends traditional approaches to immersive gaming which tend to concentrate on multiple flat screens, sometimes surrounding the player, or cylindrical [2] displays. In the former there are unnatural gaps between each display due to screen framing, in both cases they rarely cover the 180 horizontal degree field of view and are even less likely to cover the vertical field of view required to fully engage the field of view of the human visual system. The solution introduced here concentrates on seamless hemispherical displays, planetariums in general and the iDome [3] as a specific case study. The methodology discussed is equally appropriate to other realtime 3D environments that are available in source code form or have a suitably powerful means of modifying the rendering pipeline.

Keywords

Immersion, Blender, iDome, Peripheral vision, Gaming, Virtual reality, Planetarium, Fisheye.

1. Introduction

It is appreciated by the gaming industry that filling the peripheral vision of the player greatly heightens engagement with games played within a 3D virtual environment. This is not surprising since this is how we experience our real 3D world [4]. It is also well accepted in the virtual reality [5] community that peripheral vision is a key contributor to the sense of immersion, of “being there” [6], also referred to as “presence” [7]. The most straightforward way of extending the field of view of play is to add more screens, preferably in a curved arrangement around the player. This approach when it is to be extended beyond 2 displays requires either additional graphics cards or display extenders such as the Matrox dual and triple head [8] to go units. Matrox themselves in the past have released cards with three output ports (Parhelia series) and more recently AMD have demonstrated their 6 port Eyefinity [9] graphics card. The requirement on the software is support for multiple user defined perspective frustums, usually asymmetric (offaxis) frustums. It should be noted that using for a single wide perspective frustum is only applicable to multiple displays aligned in a plane.

These options however fail to really fill the human players field of view which extends to almost 180 degrees horizontally and about 120 degrees vertically. They also generally involve borders around the physical displays, which can be corrected for as if one was looking through a window with frames, but is still an obstruction for most 3D worlds. Car and flight simulators may be the exception where the display borders can be made to coincide with car or cockpit window frames.

In the virtual reality domain these shortcomings are often solved with either a cylindrical display or a number of flat walls surrounding the player, to remove the display borders a number of data projectors are employed. This introduces the complexities of multiple projectors (typically 3 or 4) and the requirement for edge blending for a seamless result. Such systems generally impose significant demands on the software model and still do not in general fill the players vertical field of view.

An observer at the center of a hemisphere onto which digital images are projected will have their field of view totally covered. This is surely the ultimate immersive environment, one where no display border and no imagery from the real world impinges on the players view of the virtual world. Historically the institution where this occurs is a planetarium, any visitor to a modern digital planetarium is well aware of the immersive effect. It is the implementation of this in the iDome, a small scale “personal planetarium”, that is the discussion of the remainder of this paper. A seamless hemispherical display that engages our whole visual field of view, and the support for games created using the Blender Game Engine (BGE).

It should be pointed out that the desire for hemispherical immersion has lead some gamers to implement solutions that fill the players field of view without modifying the images generated by the game. While they provide extremely distorted experiences particularly in the peripheral area, they clearly have some attraction and provide some heightened sense of immersion. Examples of these are the so called “jDome” [10] and “TOOB” [11] configurations. They both rely on a wide field of view setting in the game and use a single data projector, as such they have the advantage of working with a wide range of existing games without modification. The downside is the distortion that increases towards the rim of the hemisphere due to the use of a standard perspective projection rather than the required fisheye projection. In contrast, the goal here is to provide a distortion free experience across the players entire field of view. It is expected that for many game genres this will provide a gaming advantage since the player will be able to perceive events and threats in their far visual field, an evolutionary explanation for peripheral vision in the real world.

2. Fisheye projection

The single perspective projection most games provide is generally inadequate for immersive systems that surround the gamer. In essence, a single perspective projection does not capture the field of view the surround display can present to the gamer. For a cylindrical display surface the most appropriate geometric projection is a portion of a cylindrical panorama. For a hemispherical display the most natural geometric projection is known as a fisheye projection. Only a fisheye projection captures the visual information required for subsequent display on a hemispherical surface, in the same way that a perspective projection captures the visual information within a rectangular frustum and suitable for display on a flat plane. More precisely, the fisheye projection traditionally used is called an equiangular fisheye, it is one of many possible ways of mapping a 3D vector onto a unit circle. An equiangular fisheye is one where the relationship between a vector into the scene and position on the fisheye image is as illustrated in figure 1. They key is the relationship between the radius r of the point on the 2D fisheye image and the angle ϕ of the 3D vector from the camera into the scene, namely:

$$r = \phi / (A/2)$$

Where A is the aperture of the fisheye image, in this discussion A = π radians for a hemispherical fisheye. In a practical sense this mapping gives the most unbiased distribution of 3D information across the fisheye projection.

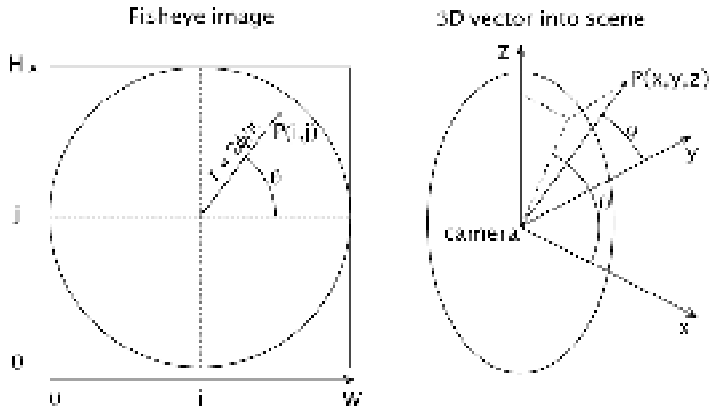

Figure 1. Relationship between a point on a fisheye image P(i,j) and the vector P(x,y,z) into the scene. The virtual camera is at the origin of this local coordinate system looking down the y axis and an up vector along the z axis.

3. iDome

The iDome, see figure 2, is a small 3m diameter personal hemispherical display device. The imagery on the dome surface is generated not with a projector and fisheye lens but by employing a spherical mirror [12] to scatter the projectors light across the wide angles required. The image distortion this would normally introduce is compensated for in realtime by predistorting the fisheye [13] content such that the result on the dome appears correct.

While the remainder of this discussion will focus on the iDome environment and it's unique image generation solution it should be pointed out that the techniques discussed are applicable to most hemispherical dome arrangements [14]. This includes the two most common single projector projection systems, namely a data projector with a fisheye lens or a spherical mirror.

4. Blender Implementation

"Blender is the free open source 3D content creation suite, available for all major operating systems under the GNU General Public License." [1] Blender is often associated with its powerful modeling, rendering and animation capabilities, less well known is the realtime and gaming engine capabilities. This includes a full physics engine, collision detection, dynamic constraints, realtime shadows, GLSL shaders and so on. The open source nature of Blender means that it is possible to add support for new functionality, such as the generation of fisheye views required for the hemispherical dome surround displays.

The generation of fisheye images involves the well understood technique of rendering to cubic maps. That is, rendering multiple 90 degree field of view (horizontally and vertically) perspective projections that cover the required field of view of the display surface, these images are then processed so as to produce the fisheye projection. The most efficient arrangement is to use 4 renderings with frustums passing though the vertices of a unit cube centred at the camera, the camera is pointed towards the

midpoint of the edge of the cube, see figure 3. The fisheye image is formed by applying the 4 renderings as textures to 4 suitably constructed meshes, the vertices and associated texture coordinates of which are designed so as to form the desired fisheye projection.

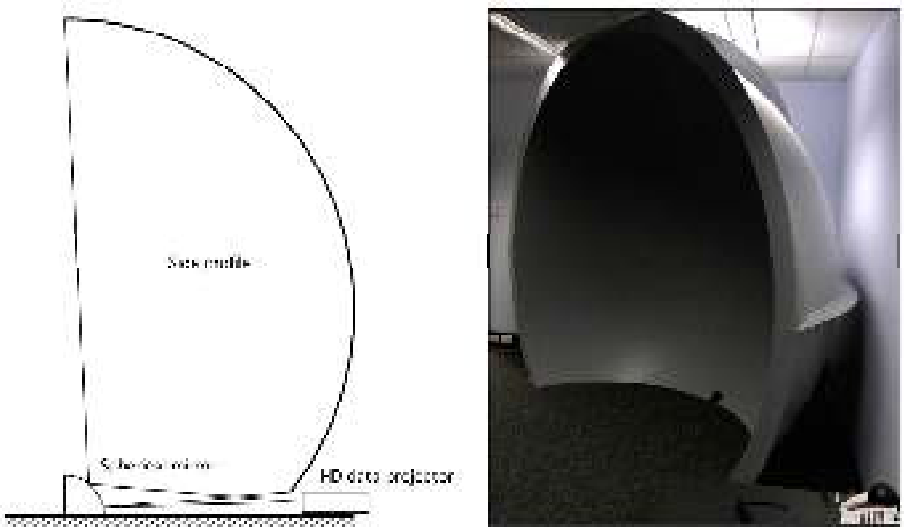

Figure 2. The iDome, a 3m diameter truncated hemispherical dome in profile and photographed on site. An important issue with a fisheye lens system is that it occupies the middle of the dome, this is region best occupied by the player. In contrast, he spherical mirror projection positioning frees up the entrance to the dome significantly.

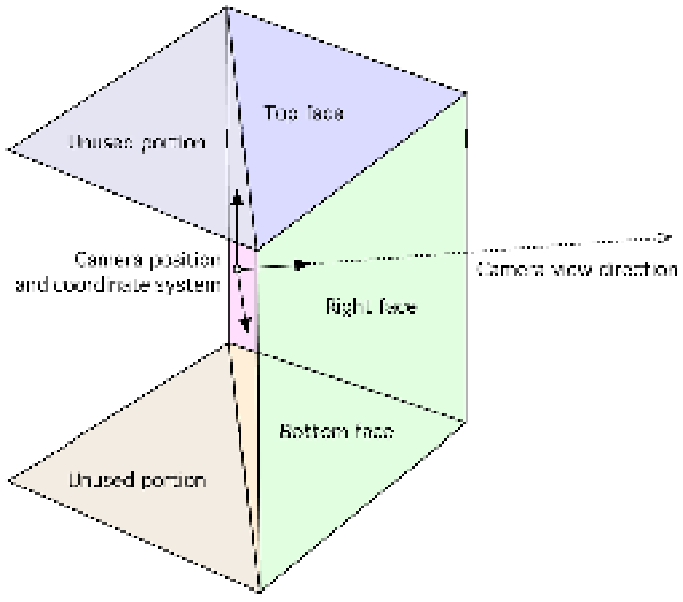

Figure 3. Camera coordinate system with respect to the 4 faces of the cube that define the 4 rendering frustums. This is the minimum number of cube faces required in order to capture the field of view required for a 180 degree fisheye.

5. Mesh generation

The formation of the meshes that will form the fisheye image from the 4 cubic face rendered images has been added to the Blender Game Engine and will appear as a built-in capability from version 2.49

onwards. There are three distinct stages to the creation of these meshes, these are now outlined and illustrated in figure 4a, 4b, 4c.

5.1 Stage 1: Tessellation

Tessellation of the 4 faces of the cube, see figure 4a. A triangular mesh tessellation is chosen where each edge of a triangle is bisected to form 4 smaller triangles. Initially the square left and right cube faces are split into two right angle triangles. Only the portions of the top and bottom faces that are actually used are tessellated. On every iteration of the tessellation the number of triangles increases by a factor of 3. At each stage of the tessellation the texture coordinates of each vertex is calculated as the average of the texture coordinates of the split edge. The texture coordinates of the right face are shown in the first two tessellation iterations in figure 4a. These texture coordinates are associated with the rendering of the frustum corresponding to the right face of the cube, as such they are preserved during any subsequent modification to the mesh. The degree of tessellation is a compromise between the quality of the warping (the more smaller triangles the better) and performance (the more triangles the higher performance penalty). The tessellation starts with 6 triangles, in general 4 iterations have been employed giving a total mesh triangle count of 486.

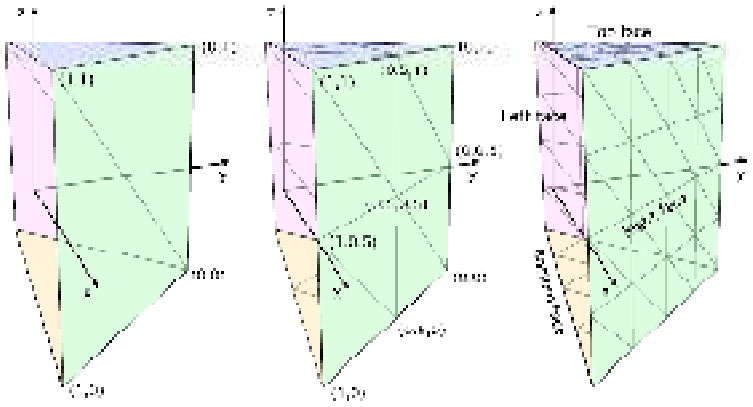

Figure 4a. Iterative tessellation of the faces of the cube that define the 4 perspective view frustums. Texture coordinates for the right cube face shown in the first two iterations.

5.2 Stage 2: Inflation

The mesh from step 1 is inflated to form a hemisphere, see figure 4b. This modifies the vertices but retains the texture coordinates. Each vertex in this inflated hemisphere can be considered a unit vector from the camera into the 3D world, the same vector as the corresponding vertex in the tessellated cube faces in figure 4a.

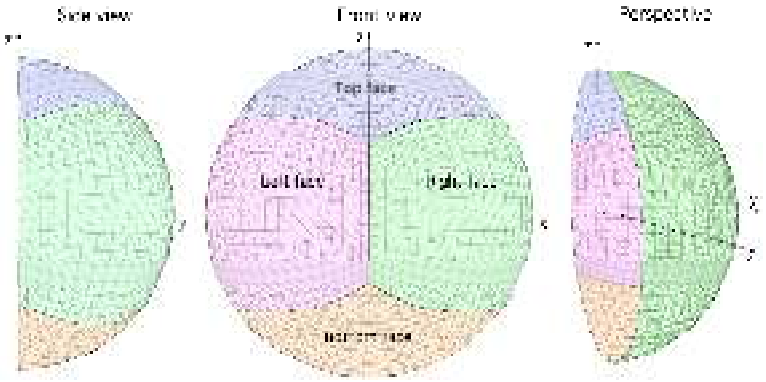

Figure 4b. Inflation of the cube vertices to form a hemisphere while retaining the texture coordinates.

5.3 Stage 3: Flatten

The final stage involves flattening the hemispherical mesh according to the equiangular fisheye projection equations. This modifies the vertices so they lie in the x-z plane (see figure 1) and the (u,v) coordinates provide the mapping from the cube face images. The normalised image coordinates on the interval (-1,1) are given by the following:

$$x = r \cos(\theta)$$

$$z = r \sin(\theta)$$

where, for a 180 degree fisheye

$$r = \phi / (\pi/2)$$

and

$$\phi = \text{atan2}(\sqrt{(P^2_x + P^2_z)}, P_y)$$

$$\theta = \text{atan2}(P_z, P_x)$$

where atan2() is the standard math library function for calculating angles on the range –π to π.

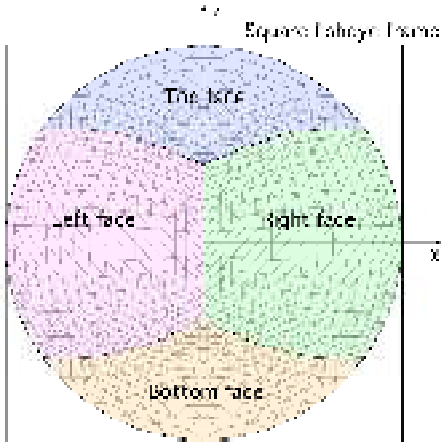

Figure 4c. Flattening the hemisphere according to the equiangular fisheye equations.

5.4 Fisheye warping

If a data projector with a fisheye lens is employed then there is nothing left to do besides rendering the above 4 textured meshes using an orthographic camera. For a projection system based upon the spherical mirror an additional pass applies the image from the orthographic camera as a texture onto

another mesh that performs the warping required. This second mesh is then rendered using another orthographic camera to produce the image finally sent to the data projector.

The vertices and texture coordinates of this warping mesh are created with external software and saved in a plain text file. The format of this file is quite simple, a 2 line header describing the type and dimensions of the mesh followed by the data for each vertex. This file format has been standardised by the author for describing warping for the spherical mirror projection and as such has been widely adopted. This mesh is in general different for each physical installation since it depends on the geometry of the hardware, that is, the projector-mirror-dome positions, the optics of the data projector, the size of the dome and mirror, and so on. The appearance of the mesh used for the HD (16x9) projector of the iDome is illustrated in figure 4d. While not shown here, the vertices of the mesh also contain a multiplicative intensity value that can be used to compensate for brightness variation across the dome arising from different light path lengths.

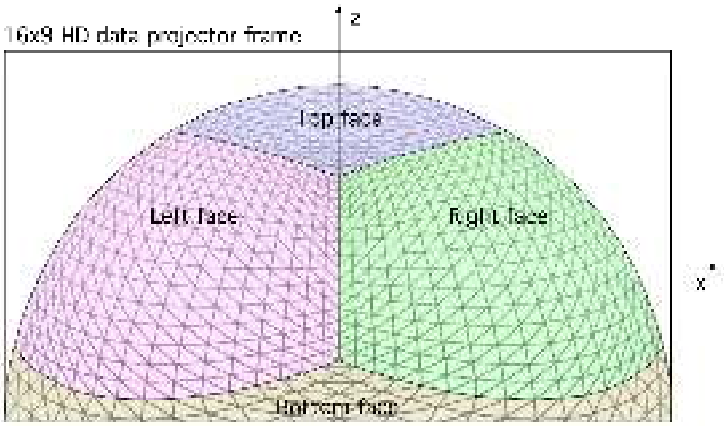

Figure 4d. The appearance of the mesh onto which the fisheye image is applied as a texture in order to create a warped fisheye for the iDome configuration (HD 1920x1080 data projector).

For a fisheye based data projection system a 5 pass rendering pipeline is used, that is, 4 cubic render-to-textures passes of the 3D world and a final orthographic rendering of the mesh that creates the fisheye image. For a spherical mirror there is an additional render-to-texture to capture the orthographic rendering of the fisheye textured mesh and the result applied to the warping mesh. The overall pipeline for a particular car racing game ("Club Silo" from Luma Studio) is illustrated in figure 5. The game being played in the iDome is shown in figure 6.

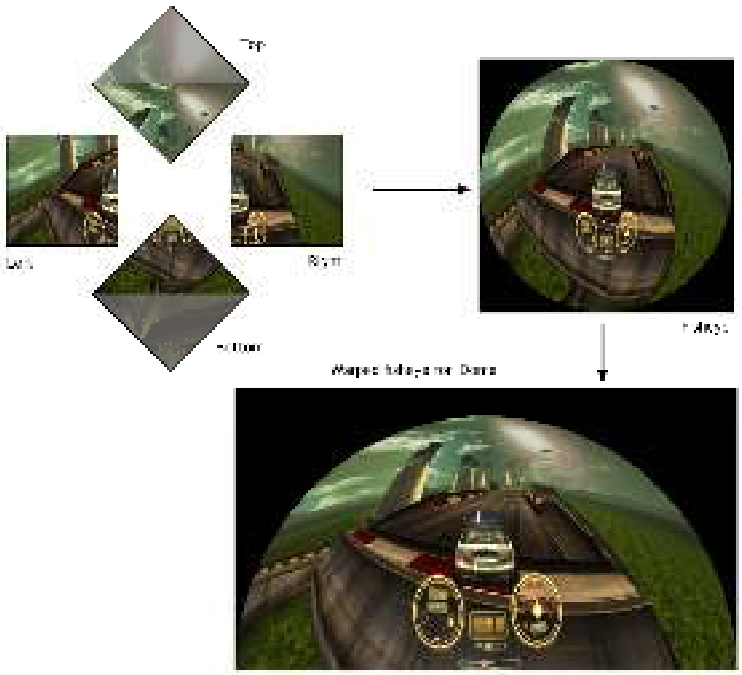

Figure 5. Pipeline showing the various render passes required to form a fisheye image or the warped equivalent from the 4 cube maps.

Figure 6: iDome and car racing game example. While the warped image sent to the data projector (figure 5) appears distorted, the imagery is perfectly correct on the dome surface.

Conclusion

We have demonstrated how fisheye and warped fisheye projections can be created using the Blender Game Engine. Such projections are required for fisheye lens or spherical mirror projection into hemispherical displays using a single data projector, the most affordable option for gaming. The benefit of an open source project such as Blender is that any game written in that software can be readily converted to support this unique immersive display, for example, the YoFrankie game shown in

figure 7. The multipass rendering pipeline described is the most straightforward approach for realtime digital fulldome and can readily be incorporated into other gaming engines.

Figure 7. YoFrankie, an open source 3D game project, being played in the iDome.

The authors would like to acknowledge support from the SAT (Society for Arts and Technology, Montreal, Canada) Metalab immersion research program as well as Benoit Bolsee for assistance with the Blender source tree and committing early versions of the dome support code.

References

[1] Blender. http://www.blender.org/ (Web reference).

[2] E.L. Smith. *Realtime graphics for visual simulation*. Siggraph 98 Course Notes #20. 1998.

[3] P.D. Bourke. *Low Cost projection Environment for Immersive Gaming*. JMM (Journal of Multimedia), ISSN: 1796-2048, Volume 3, Issue 1, pp 41-46, 2008.

[4] B. Ball, C. North. *The effects of peripheral vision and physical navigation on large scale visualization*. ACM International Conference Proceedings Series; Vol 322. Proceedings of graphics interface. ISSN:0713-5424. 2008.

[5] C. Cruz-Neira. *Surround-screen projection based virtual reality*. Proceedings of Siggraph 93.

[6] M. Bailey, M. Clothier, N. Gebbie. *Realtime Dome Imaging and Interaction: Towards Immersive Design Environments*. Proceedings of ASME 2006 International Design Engineering Technical Conference, DETC2006-99155, September 2006.

[7] H.G. Hoffman, T. Richards, B. Coda, A. Richards, S.R. Sharar. *The Illusion of Presence in Immersive Virtual Reality during an fMRI Brain Scan*. CyberPsychology and Behavior, Vol 6, Number 3, 2003.

[8] Matrox display expansion products. http://www.matrox.com/graphics/en/products/gxm/. (Web Reference).

[9] AMD Eyefinity, http://www.amd.com/us/products/technologies/eyefinity/. (Web Reference).

[10] jDome. http://www.jdome.com/ (Web Reference).

[11] TOOB. http://thinkoutofbox.com/ (Web reference)

[12] P.D. Bourke. *Using a spherical mirror for projection into immersive environments*. Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia. pp 281-284. 2005.

[13] P.D. Bourke. *iDome: Immersive Gaming With The Unity Game Engine*. Proceedings of the Computer Games & Allied Technology 09 (CGAT09), Research Publishing Services, ISBN: 978-981-08-3165-3, pp 265-272, 2009.

[14] W. Hashimoto, H. Iwata. *Ensphered Vision: Spherical immersive display using convex mirror*. Transactions of the Virtual Reality Society of Japan, 4 (3) 497-486, 2001.