Creating fisheye image sequences with Unity3D

Research · July 2015

DOI: 10.13140/RG.2.1.1853.2963

CITATIONS
0

READS
476

1 author:

Paul Bourke

121 PUBLICATIONS **1,072** CITATIONS

SEE PROFILE

Creating fisheye image sequences with Unity3D

Written by Paul Bourke, Contributions from Nick Oliver
May 2015

Sample: unityfisheyesample.zip, built in Unity V5

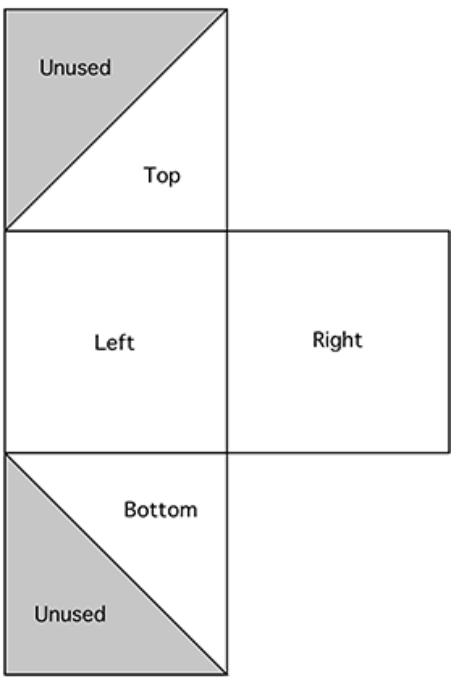
The following are notes related to the creation of fisheye frames from the Unity game engine. There are other documents here describing the real-time generation of dome content from Unity, the theme here is using the engine to create fisheye frames that might then be saved and used for movie playback, although the technique has wider application. There are two approaches: simply creating the required visual field and then using external tools to create the fisheye (or spherical image), or creating the fisheye image within Unity itself. Both cases require a means of saving camera views to a file (see script at the end of this document). The relative merits and limitations of each approach will be also be discussed.

Saving individual camera images

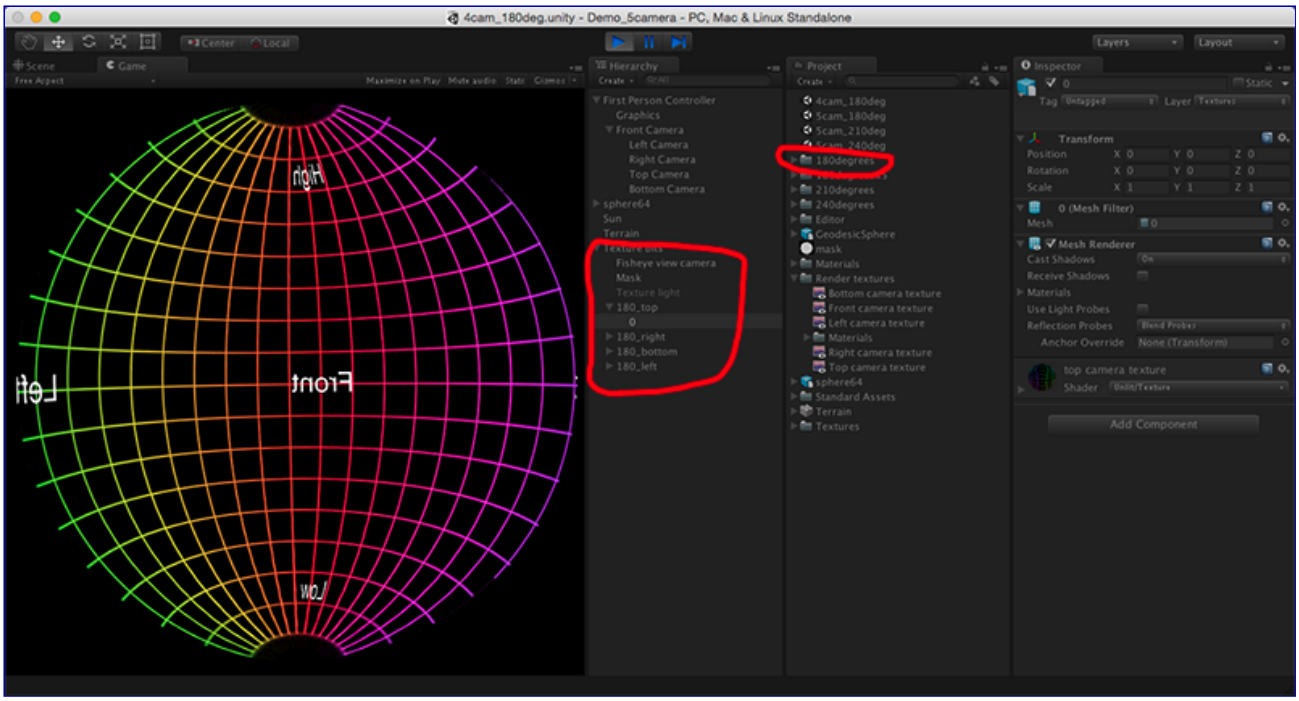
The basic approach to both techniques is to replace the single camera in Unity with a multicamera rig. The multiple cameras are arranged in the same fashion as one might choose for cubemaps, that is, each is orthogonal to it's neighbours and with a 90 x 90 degree FOV. If render-to-texture is available (Unity Pro version pre version 5) then each camera view can be rendered to a texture that is then saved. The resulting images are combined with post production tools such as cube2dome.

Creating fisheye directly

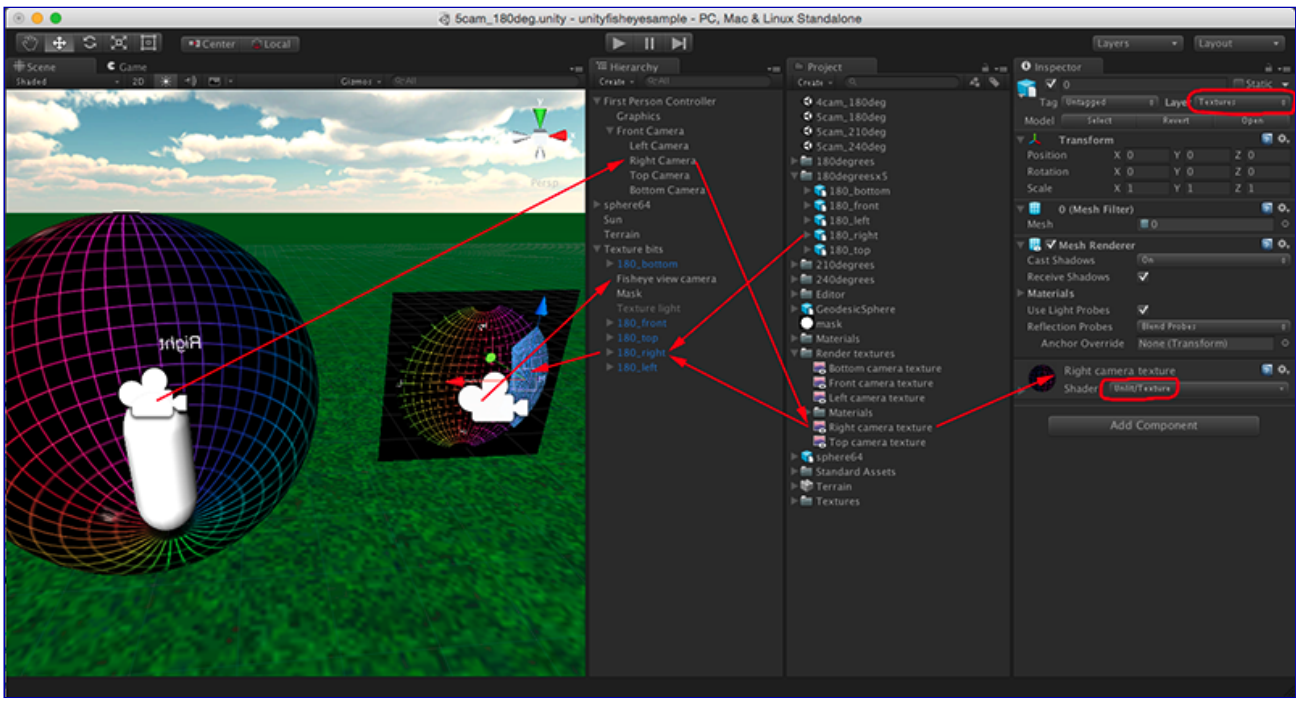
The fisheye can be created directly in Unity by taking each of the camera textures and applying them to a mesh (vertex coordinates and texture coordinates) precisely crafted to create a fisheye. For the minimum case of a 180 degree fisheye only 4 cameras are required, the obj file that will combined these to form a fisheye is given here (180x4.zip). Unlike the more common camera arrangement where the "front" camera is aligned with the center of one of the cube faces, here the front camera is pointing half way along the edge between the left and right camera. The textures for a 5 camera rig to create a 180 degree fisheye is given here (180x5.zip)



A 4 camera and 5 camera rig is included in the sample project.

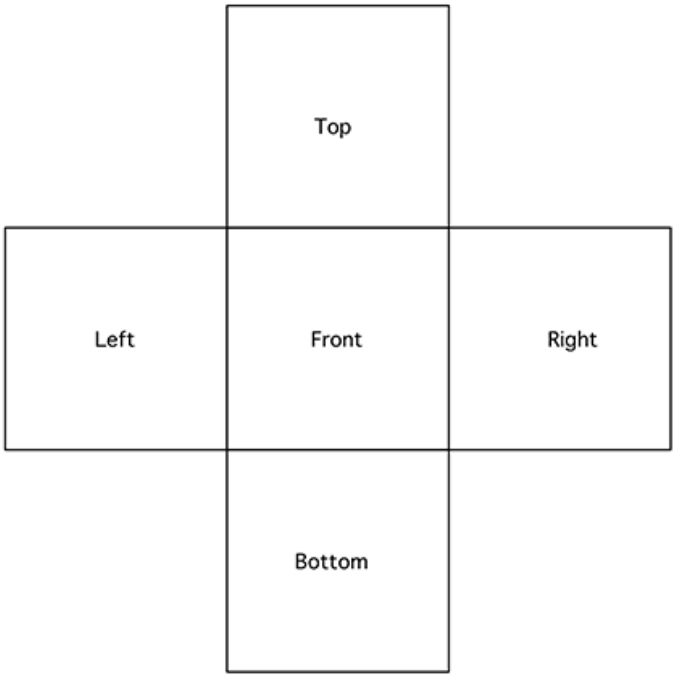


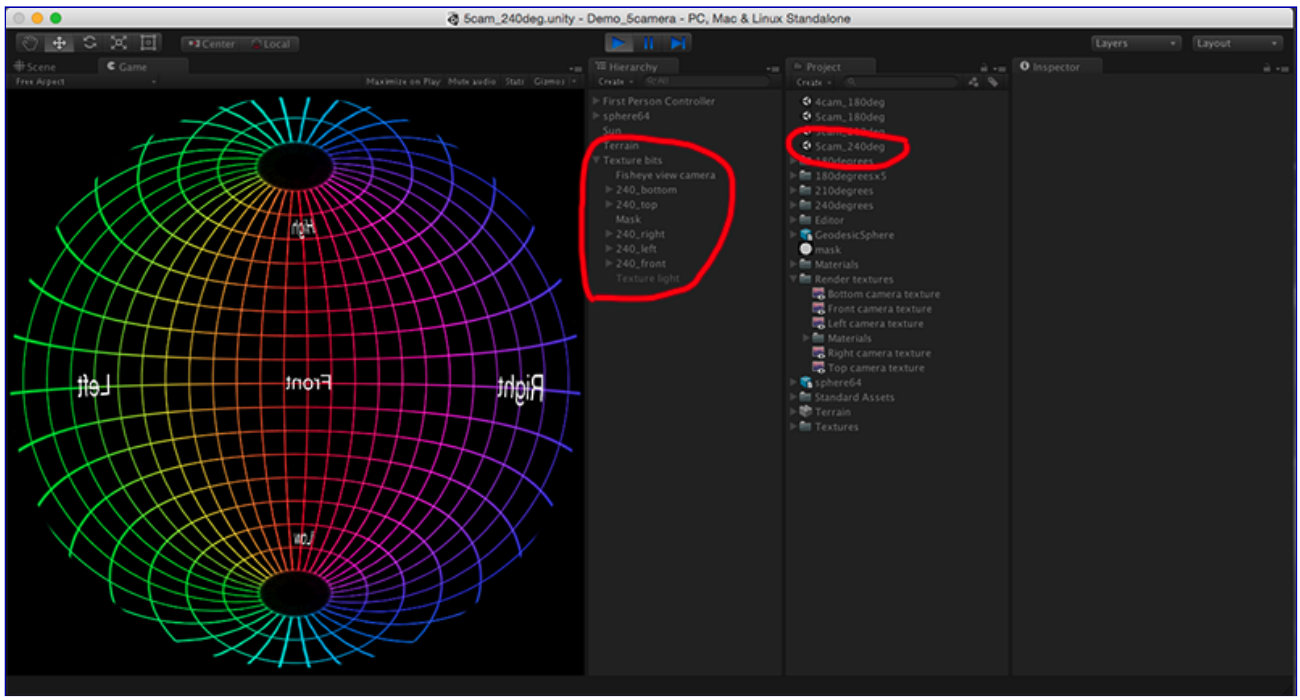
The key components / flows for the right camera are illustrated below, see the sample project. The right camera renders to a texture, that texture is applied to the mesh section that performs the geometry correction, a final orthographic camera captures the view comprising of just the 5 mesh sections. The geometry to achieve this is placed on a layer by itself, invisible to the scene camera rig. All textures use the unlit texture shader although a diffuse shader could be used with a directional light source on the texture layer to provide some brightness control.



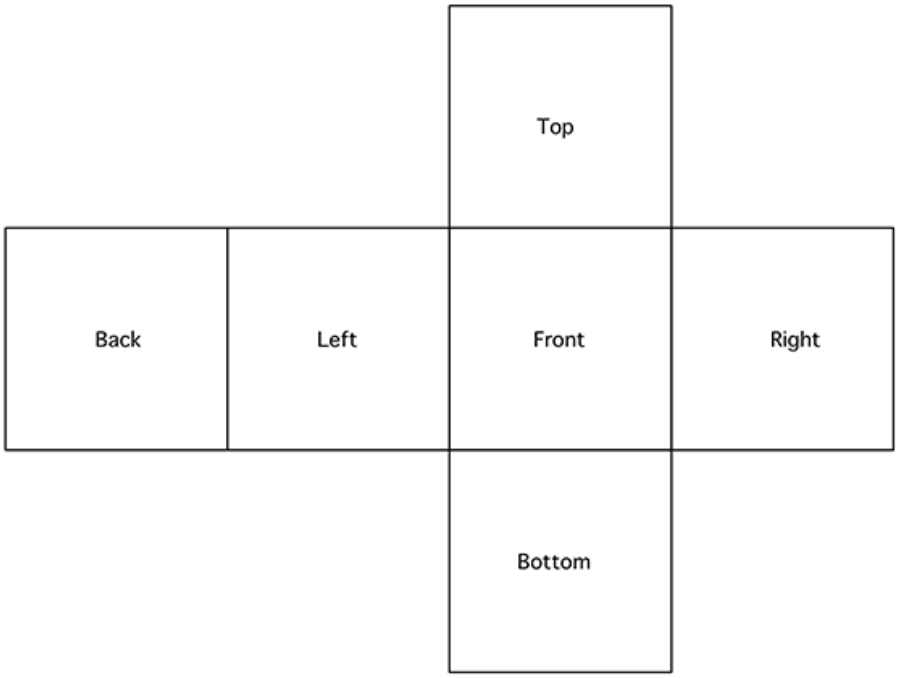
Notes

- Creating the fisheye individually is limited to 4K fisheye images (at the time of writing). Whereas saving the camera streams directly would allow up to 8K fisheyes.
- The discussion here is not limited to 180 degree fisheye projections, fisheyes are defined for larger (and smaller) angles. However the 4 camera rig described above is limited to 180 fisheyes, it is the minimum/optimal number of cameras. Wider fisheye projections require more cameras, the next most common arrangement is a front camera along with a left, right, top, bottom camera providing the visual field of view for up to 270 degrees. The obj files to combine the camera views to a 240 degree fisheye can be found here (240degx5.zip), the sample project also includes a 210 degree fisheye.





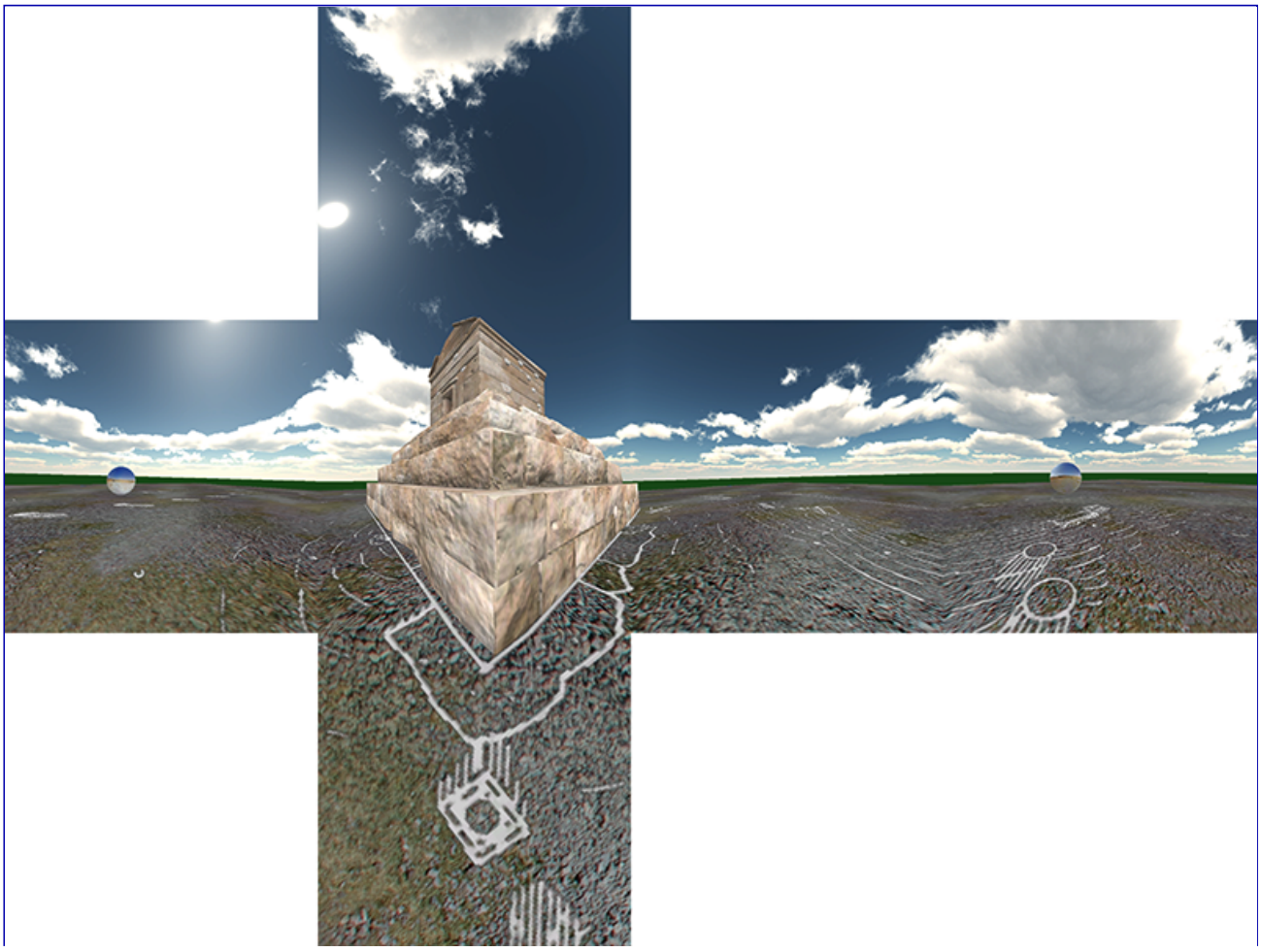
- Some thought needs to be given to the size of the render textures. Typically for a fisheye of resolution N, the camera views contributing to that fisheye should be N/2.
- Using the full 6 cameras (front, back, left, right, top, bottom) has two benefits, first it allows one the most flexibility in post production to orientate the fisheye for different dome orientations. Specifically, domes in the planetarium angle to front looking domes such as the iDome. Second, it allows for spherical projections to be formed and the corresponding possibility for navigable movies.



- The mask in the sample project is not stricktly necessary can can be disabled. It is useful for limiting the fisheye to a circle when fisheye meshes greater than 180 degrees are used.

Example

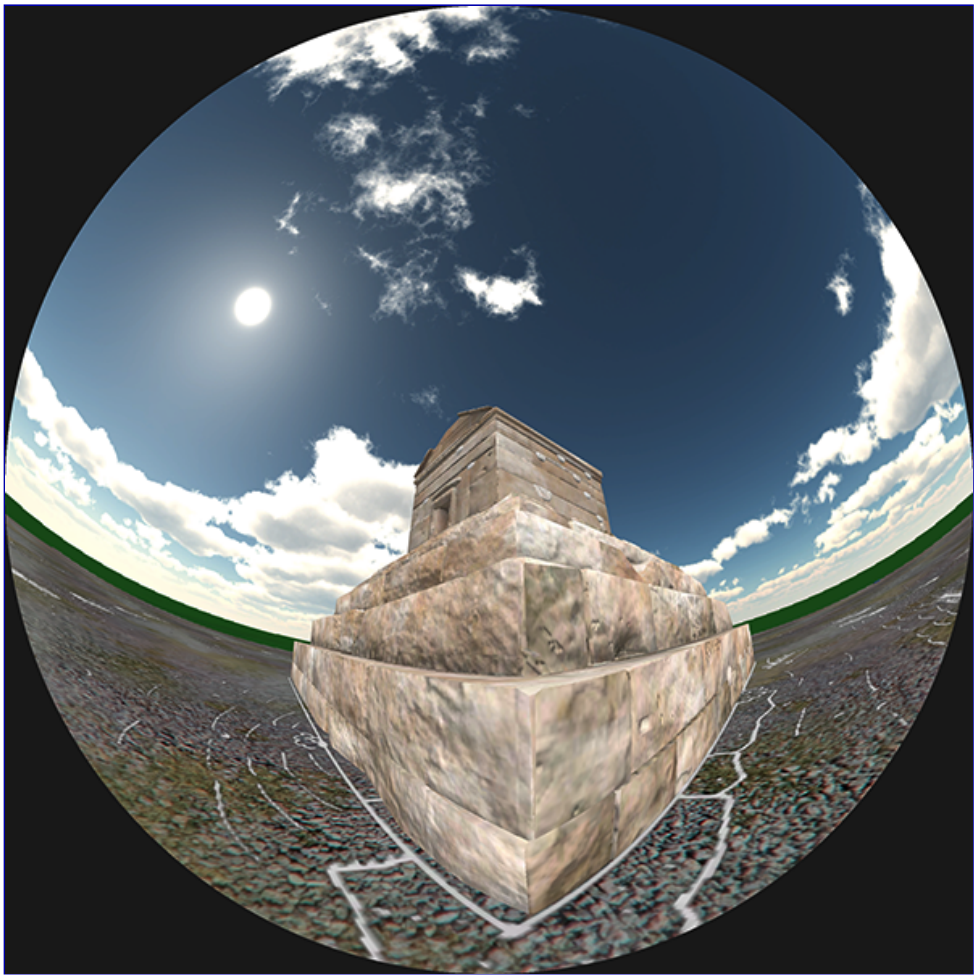
The following example is created from a project of the Cyrus temple in Iran, all 6 camera views are created, the following script saves the texture images: TextureDump.js.zip, it can be readily used to create a frame along each position of a scripted smooth path.



Using "cube2dome" and the following command line, the fisheye below is created. This reveals the strength of the 6 camera approach in that the angle of the camera rig in Unity is not important, any of the infinity of possible fisheye angles can be created

in post production.

```
cube2dome -a 2 -vt 60 %c0.tga
```



Limitations

- Care needs to be taken regarding antialiasing which can result in seams across the resulting camera views in fisheye space.
- The continuity between camera views can be broken for some post processing image effects in Unity. One example are lens flares, these are performed in image space rather then in 3D.
- The continuity between camera views can be broken for geometry that changes depending on the camera view direction. An example of this may be billboards, especially grass for the downward pointing camera.
- While 4 is better performing for realtime (one less render pass), 5 cameras are slightly better since it conforms to the cube maps orientation people normally create from rendering packages. So may be better for compositing and it can minimise any edge effects in the direct front view.
- For textured planes that extend into the distance larger than normal Ansio values for the bilinear filtering may be required in order to minimise filtering differences between the scene cameras.
- "Application.CaptureScreenshot()" requires that the player window is exactly square in this context. Capturing and saving the render textures can be achieved by using Texture2D.ReadPixels() and Texture2D.EncodeToPNG().

Future

- Much of this could be more elegant if it were possible to create a global vertex shader, this would create the fisheye in a single pass. The downside is that large close geometry needs to be tessellated, that is, the line between two vertices in fisheye space is not straight.