Off-the-shelf software limitations when stitching camera views for fulldome.
An over-rendering the camera-view technique to avoid seam edge issues.

By: Tom Casey - President/Creative Director - Home Run Pictures

Summary of paper:

All the available 3D computer generated animation software applications are designed primarily to create "framed" views (film and video). The application developers typically treat "what's beyond the view" as a throwaway and problems arise when stitching the multiple camera views when creating fulldome imagery. This paper will detail an approach that employs rendering an area beyond the needed camera view to "trick" the rendering software into creating camera-views that will stitch together without visible seam edge issues. An explanation as to what elements in a scene can potentially cause problems, why these elements cause problems and how the over-rendering technique can eliminate the problem will be discussed. Although the solution will be implemented with the scripting tools available in Alias Maya software, similar techniques can be employed in any of the various software tools being used to create fulldome imagery.

Introduction:

The most common method for creating fulldome imagery using 3D computer generated imagery software applications involves the rendering of multiple camera views that are then stitched together. This method is popular because it is not limited to any particular software tool and is easy to implement in all the applications... Maya, Lightwave, 3D Max, etc. The artist/animator creates a grouping of five cameras with their eye view at a common point in space. Each camera is then rotated to look in directions correlating to front, back, left, right and top (up) at exactly a 90 degree difference to each other... in traditional geometry you would say each camera is looking out from the origin towards the +x, -x, +z, -z, and +y axis assuming a y-up orientation (the -y is not rendered since it would be a view looking down). These five views are each rendered with a square 90 degree field of view so the entire 360 degree spherical scene is seen. When stitched together with the tools provided by the fulldome projection vendors, a master image is created that is essentially a polar projection of the surface of the dome.

In most instances, this method is an ample approach to generate the imagery desired. Objects in the scene made from various geometric descriptions (polygons, b-splines, nurbs, etc.) render correctly and the views stitch together with no artifacts at the seams of the five camera views. Unfortunately, all that is desired for the creation of imagery, can not be accomplished using straight geometric models. Today's software tools are more robust than ever and provide ways to accomplish the various looks desired using a variety of programming add-ons. Techniques, such as glows and particle generated atmospheres are accomplished using programming implementations that are at times computationally intensive. In order to reduce rendering times shortcuts are taken... and this is where problems arise.

Two common problems:

This paper will address two problems that arise when attempting to create fulldome imagery using the multiple camera approach. The first problem involves the use of glows around geometric objects in a scene. Since the subject material for many fulldome shows is of an astronomical nature, there are many instances where it is desirable to have glows around an object... a star, a planet's atmosphere, etc. The standard implementation for creating glows in most software packages will not stitch together without the seams being visible.

The second problem occurs when trying to achieve any atmospheric effect using a particle approach... a horizon, a nebula, a cloud, etc. In many software packages, the particles will not render correctly as they move close to the edges of a camera's view, once again creating a mismatch with the adjacent camera's view and a visible seam in the final master image.

Why these problems occur:

Both these problems are related to something that is a key goal in computer generated imagery applications... speed. In both cases, the developers opted to take a shortcut to achieve faster rendering of the final camera image.

In the case of glows, the software writers for just about every application use a 2D (flat 2-dimensional) technique to achieve the desired effect. Glows are added as a post process (after the initial 3D rendering) where the objects in a scene that a glow is desired around have the glow added after rendering... in other words, the glow is a process implemented after the image is rendered and uses the finished image's pixels as its source. Since the final image is now only a 2-dimensional picture, the frame is treated as "this is all there is." Glowing objects at the edges of the frame ignore the reality that they would continue past the frame and a glow which is not truly accurate is created. This works fine for "framed" applications, but will not work when stitching frames together for fulldome uses. The seam areas of the camera frames will not match.

Computer generated particle effects are generally render-intensive with, at times, tens of thousands of particles used to simulate what is desired. Dramatic effects can be achieved when the camera view moves through a "cloud" of particles. Since most particle rendering implementations attempt to minimize the number of particles necessary to render a look, many just clip away (delete) particles as they near the edge of the camera view. Once again this works fine for "framed" applications, but not for fulldome use. The particles will appear to pop on and off as they pass through a seam.

The over-rendering technique:

One method to overcome these limitations is to use an over-rendering technique to "trick" the rendering software to ignore the frame edge... or as in our implementation, move the frame edge out a little to create a correct "virtual" inner-edge for stitching purposes. For example, if your final camera images need to be 100x100 pixels in size, you render views that are 120x120 pixels and trim off 10 pixels around all the edges. Thus the new "edges" we end up stitching where rendered as an "inner" part of the image and will not display the unwanted edge seam artifacts. This technique will work for most instances involving object glowing and particle "popping" issues (depending on your software tool of choice). It will even vary greatly depending on what software version you have... application programers tend to change their implementations from version to version. Testing the technique for a given situation is the only way to determine if it will solve the problem in that instance.

There are two questions that need answering in the implementation of this technique. First it is necessary to determine how much "over-render" is necessary and second, what new camera angle to use to render the "over-render" and still maintain the necessary 90 degree field of view for each camera when we trim off the extra pixels.

The first question requires looking at each scene element and how objects pass out of the camera's field of view. In the case of a glow around an object, it is necessary to determine how wide the glow is when the object is near the edge. In a simplified treatment we will look at a sphere with a glow. Illustration A depicts a possible scenario.

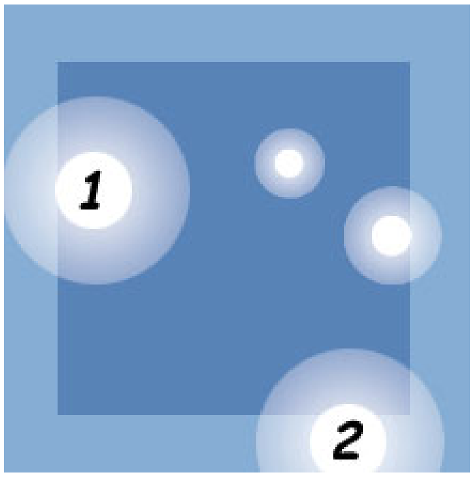

Illustration A

The dark blue square represents the area that will be the final desired view for this particular camera with the lighter blue representing our "over render." Sphere number 1 is just at the edge of the final view, but the glow around it is partially outside the final view. In order to produce a glow that is not compromised by the frame edge, it will be necessary to render additional pixels out to the full extent of the glow. This way when the frame is trimmed and stitched, the glow should match up with the glow created by the adjacent camera view.

The way the glow render process is implemented in you software package will determine how you figure this. Typically the glow values are not measured in pixel width, but a value that represents amount of glow verses total resolution of the image. If your software package allows you to render out the glow portion of the image separately, it may be possible to look at that image and count the pixels in the glow to determine the necessary over-render. The only way to truly determine the correct over-render is by testing. Eventually, you might be able to calculate this with some inside knowledge about your particular software application, but generally it is best to just test the frames suspected of having seam issues.

Notice that sphere number 2 has almost entirely moved outside the area we intend to keep for stitching. The glow around it will still be created and will then match across the seam during stitching. Once you have determined the pixel width necessary to fully cover for glows, this additional number of pixels will be added to the desired resolution to end up with your over-render resolution. In illustration B, two trimmed frames are shown side by side in their final stitched orientation with glows that now are correctly rendered.

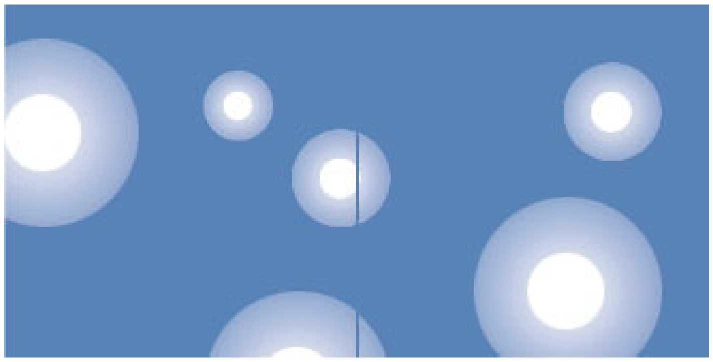
Illustration B

The same process can be used to determine how much over-render is necessary when using particles to create a desired effect. Illustration C depicts a possible scenario.

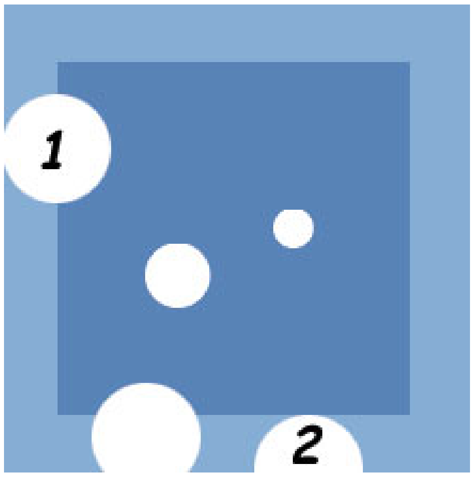

Illustration C

Some software packages will determine if a particle is beyond the frame edge and clip (delete) it from the render. This works fine and may be unnoticeable if the particle is not very large. But in the case where the camera view is traveling through the particles and some particles are very close to the viewer, large particles will pop off. One workaround would be to increase the number of particles and make them smaller, but this could create much longer render times. The additional time required to render the over-render must be compared to the increased render time with more, smaller particles to see which is a better solution. Still, the problem of the camera view passing so close to the particles that they appear large in the view and are clipped may only be avoided by using the over-render technique.

Most applications consider the middle of the particle as a determining factor when leaving the edge of the frame, so adding half the size of the particle's closest size at the edge of the frame is usually sufficient for the over-render technique to work. In some instances you may need to over-render the complete size of the particles. Once again, your particular software application may be different and testing is the only sure way to determine what will work for you.

Determining over-render camera field of view:

Once you have arrived at the necessary over-render to achieve a clean final stitch of your camera views, you will need to determine what expanded camera field of view is necessary while maintaining the final pixel resolution after trimming off the extra pixels. We know that the final frame must represent a 90 degree field of view to stitch correctly, so some simple trigonometry is necessary to determine what resolution as well as field of view to render the over-rendered pre-trimmed frame for this to work.

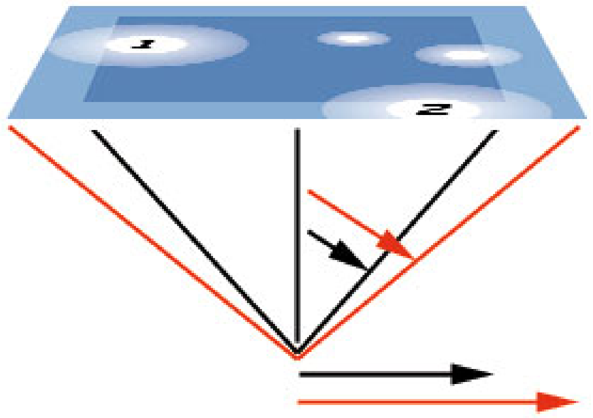

Illustration D

In order for the base of the triangle to have the desired number of pixels in the final rendered image that we use to stitch, we need to increase the resolution for the over-render by a certain number of pixels and determine the larger field of view angle for our camera. It's easiest to think in terms of half the image resolution and then double it so we can use a right triangle and all the mathematical relationships that can be assumed there. In this illustration the black lines represent the 90 degree field of view we need to correctly stitch our fulldome master and the red lines indicate the area that will be over-rendered (and trimmed). So you can see that the field of view will expand to create the extra pixels we will eventually trim off. You will need to create an appropriate equation to implement this in your particular software. Your computer or a programmable calculator can be used to do this. The rendering camera's field of view (FOV) for our sample 120x120 pixel image being trimmed to 100x100 would be calculated...

FOV = atan(120*tan(90*PI/360)/100)*360/PI

Below is the script we use at Home Run Pictures, written using the Alias Maya MEL language that determines the field of view from an input of the desired final resolution in pixels and the desired over-render resolution in pixels.

```
global proc float expandedFov( float $oldFov, float $oldRes, float $newRes ) { float $PI =
3.1415926535897932; return( atan( $newRes * tan( $oldFov * $PI/360)/$oldRes) * 360/$PI ); }
```

The desired final resolution and field of view (90 degrees for fulldome) is entered along with the over-render resolution and the script returns the new camera field of view to render with. If your software allows you to render a section of your full image, you could even go further and have it render only the area you want, but still consider the over-render area in its calculations... thus avoiding the need to trim off the over-render. This will work for particles, but not work with glows since the rendered image's final pixels are what is used to calculate the glow.

Conclusion:

Once you have your over-rendered camera frames, you will just need to trim off the extra pixels and stitch as usual. There will be instances where the seams will not stitch correctly without very large over-renders being necessary... making the additional render costs undesirable. In this case, and if there are only a few frames with problems you can choose to over-render only those frames as a fix.

At Home Run Pictures, we have found this technique to work effectively for most glowing and particle scenarios where seam visibility problems occurred during the stitching process.

For further information, contact: Tom Casey - Home Run Pictures + www.hrpictures.com

100 First Avenue, Suite 450 + Pittsburgh, PA 15222 + tom@hrpictures.com + 412-391-8200