RenderWare Graphics

White Paper

Optimizing Static Geometry

Contact Us

Criterion Software Ltd.

For general information about RenderWare Graphics e-mail info@csl.com.

Developer Relations

For information regarding Support please email devrels@csl.com.

Sales

For sales information contact: rw-sales@csl.com

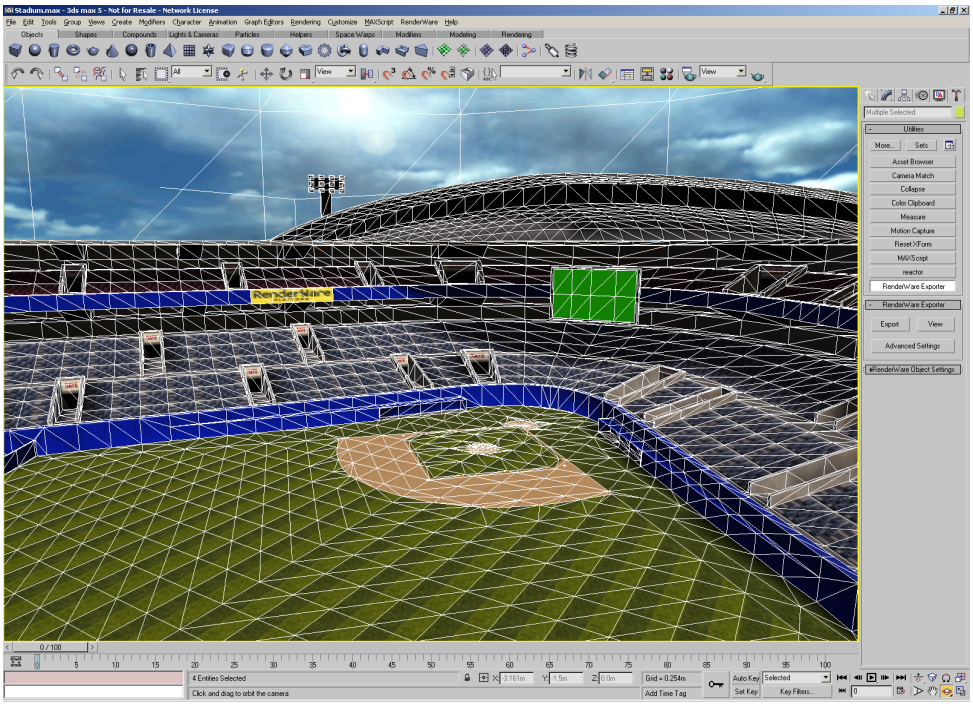
Acknowledgements

Contributors RenderWare Development and Documentation Teams

Optimizing Static Geometry For Use With RenderWare Graphics

Baseball Stadium Case Study

This example refers to a specific case where an artist was asked to produce a baseball stadium model for an in-house demo. The performance was found to be slow (on PlayStation 2) and we were asked to find out why and improve it so it would frame lock at the required 60hz at all times.



Here is the model as it arrived. On the face of it a nice clean piece of artwork, competently modeled and textured. So, what could possibly cause problems, and how do you go about tracking them down?

This can be divided into three areas, which we will look at in turn

- Texture Maps
- World Sector Generation
- Tri-Stripping

Texture Maps

Texture RAM is very limited on PlayStation 2 and texture uploads can be a major cause of poor performance. The first thing to check is the texture library for size and color depth.

In this case there were several 512 x 512 pixel textures, many 256 x 256 pixel textures, a couple of non-square 512 x 256 pixel textures, one 512 x 128 pixel texture, one 126 x 126 pixel texture and a couple of alpha masked textures. All files are 8 bit `.PNG` format.

With the exception of the 126 x 126 pixel map, all the textures are all power of two sizes, which is essential to avoid them being resampled automatically to power of two on loading by RenderWare Graphics. Even the 'over square' 512 x 256 and 512 x 128 proportioned textures are fine as they are all power of two ratios. All the textures are 8 bit 256 indexed color (as opposed to 16 bit RGB color), which is also the best option for PlayStation 2.

However, almost all the textures in this library are too large and will take up too much PlayStation 2 RAM resulting in performance crippling texture uploads. All the square textures (including the 126 pixel square map) were re-sized to 128 x 128. The 512 x 256 to 128 x 64 and 512 x 128 to 128 x 32.

Another important consideration to bear in mind when thinking about texture sizes is the impact of mipmapping. Because mipmapping intelligently switches to lower resolution versions of your original texture when rendering it can dramatically improve performance. This benefits not just the number of textures that will fit in cache but also reduces the memory bandwidth used when rendering and thus the maximum fill rate. On PlayStation 2 it is very important to tune K and L values to ensure that the smallest mipmap levels possible are used without blurring your textures unacceptably. Note that, while mipmapping is recommended, you should normally avoid using trilinear mipmapping on PlayStation 2 as the increased memory bandwidth it requires reduces performance.

As the `.PNG` format supports embedded alpha the separate alpha mask textures were merged with the color maps.

These changes helped the demo to frame-lock at 60Hz when the camera was close up to the world. However, when the camera was pulled back far enough to have all of the world geometry in its frustum the demo was still failing to frame-lock at 60Hz. So on to the next issue: World Sector Generation.

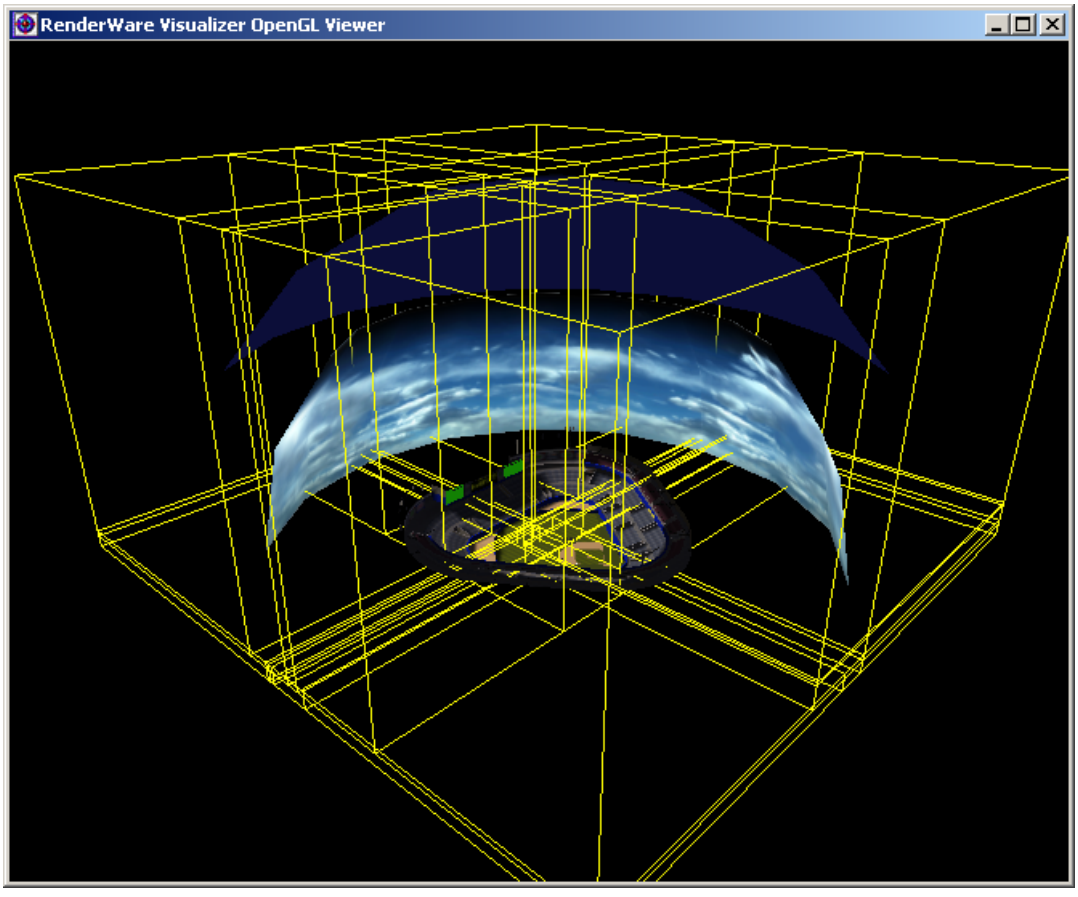
World Sector Generation

Static 'world' geometry is exported in BSP (Binary Space Partition) format in a RenderWare Graphics binary stream `rwID_WORLD` chunk. This geometry is divided by the exporter into a number of 'world sectors' for scene management purposes (geometry culling and so-forth).

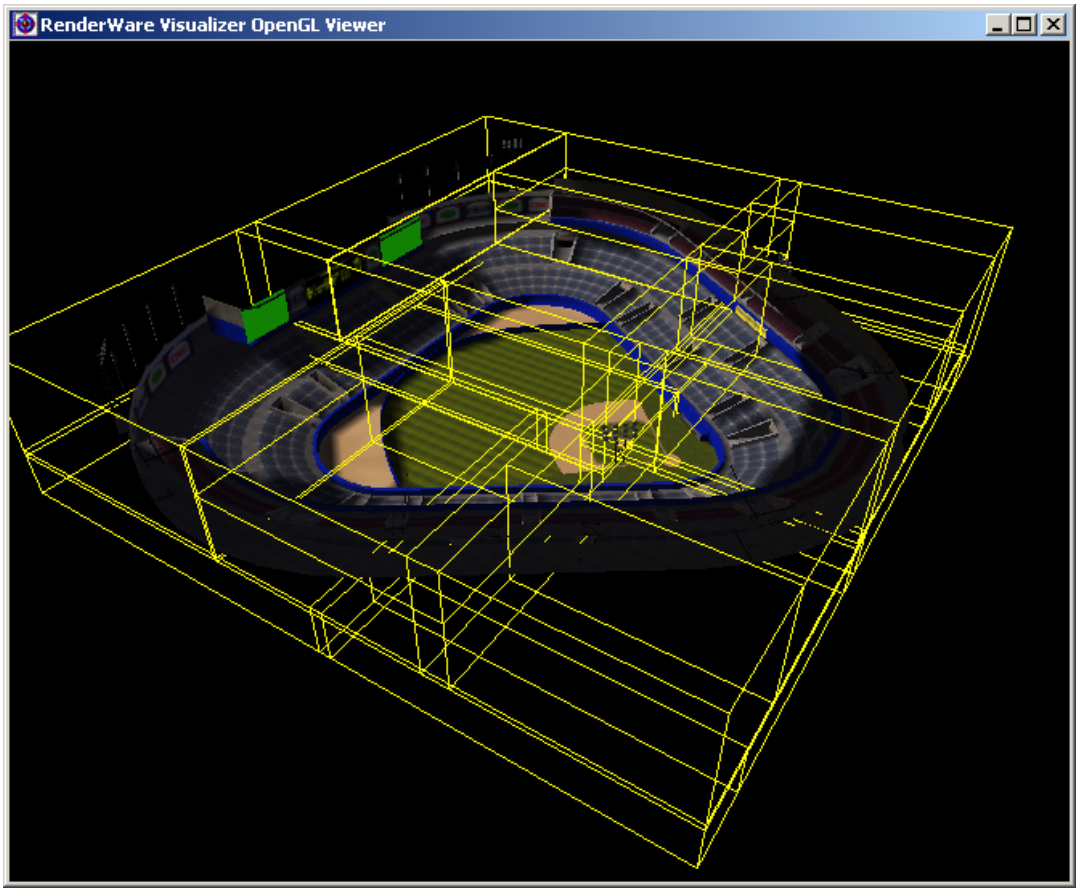
The automatic subdivision of a world can be controlled in the exporters. However, the way in which sectors are organized are still largely determined by the geometry itself. Note, while it is possible to build a world manually using RenderWare Graphics hints, this is only recommended with a high degree of understanding of the way RenderWare Graphics world sectors work. The automatic subdivision of a world is generally an efficient process except when geometry is encountered which is difficult for the Exporter's sectoring algorithms to resolve. This can lead to inefficient sector shapes and excessive triangle splitting and can also break many tri-trips (discussed in the next section).

A classic example of this is having a 'sky dome' exported as part of the world with huge scale differences between dome polygons and the world polygons. The Sectorization process will need to deal with these differences in size by chopping up the large polygons in an apparently arbitrary way creating very nasty areas of tessellation where these intersections occur. Furthermore, this can lead to extremely poor result for the generation of PVS data.

The Baseball Stadium has a sky dome that is being exported in this way, and a look at the resulting World Sectors in the WorldViewer clearly shows the symptoms:



Once this dome has been removed from the world model, the world sectoring can be resolved by RenderWare Graphics in a far more practical way:

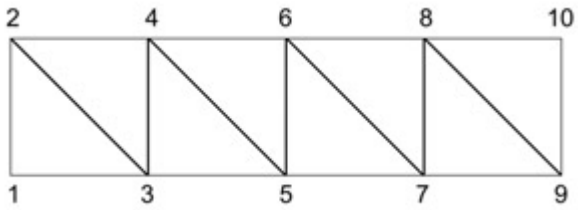


The sky dome can still be used in the Application as a separate clump.

Triangle Strips

Why are tri-strips important? Here's a quick explanation of what actually constitutes a tri-strip.

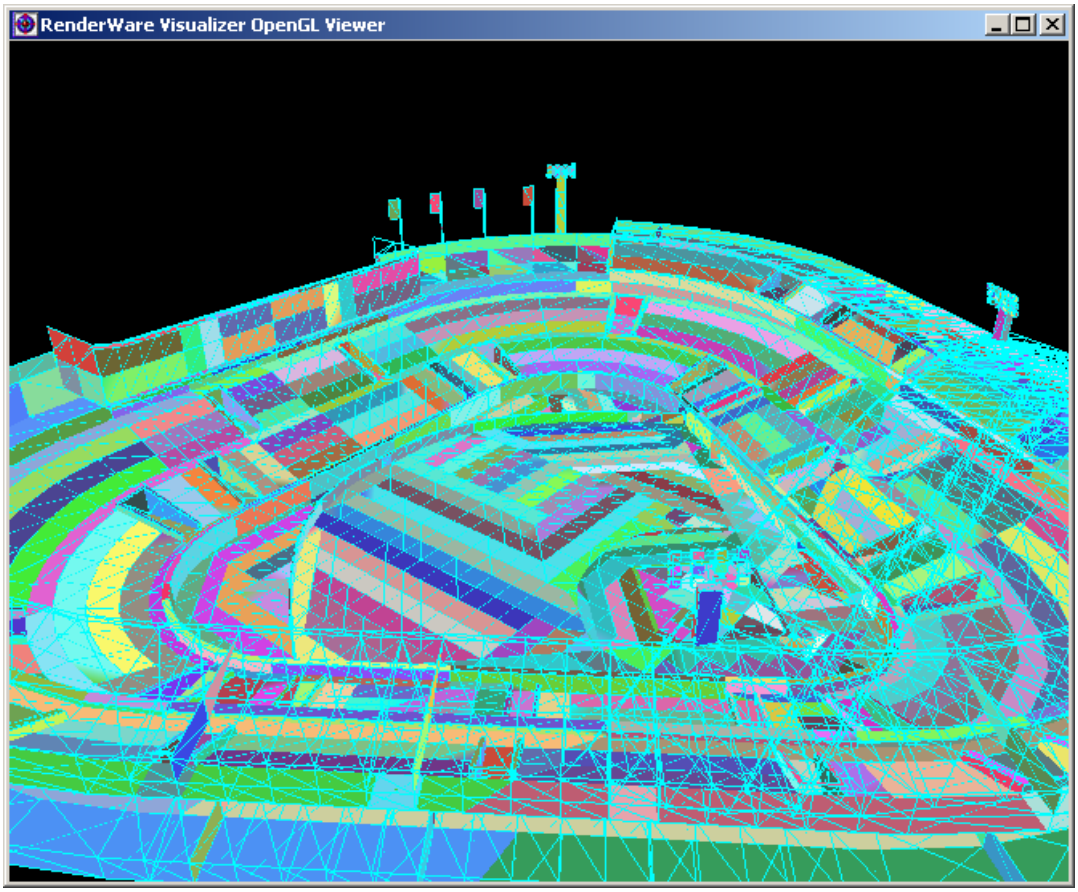
In the mesh shown below it takes three vertices (1, 2 and 3) to draw the first polygon but each subsequent polygon needs only one vertex to define it. Vertex 4 creates the second polygon by re-using 2 and 3, vertex 5 creates the third using 3 and 4, etc.



By sharing the vertices between polygons, a tri-strip is a very efficient way of storing geometry and the longer the strip the better the result. Note that anything that makes one of those polygons unique, such as a material or normal that's different from its neighbors, will break the tri-strip.

Creating geometry that will tri-strip efficiently is key to optimizing its performance on PlayStation 2. Attempting to tri-strip geometry which is not tri-strip friendly will actually increase poly count and perform worse than it's non-stripped equivalent.

So how does the stadium model measure up? The export of the original model showed the following profile. Each colored area represents a different tri-strip:



As can clearly be seen the longest tri-strips are sometimes no more than around six triangles while some are just a single triangle in length. This is not good tri-stripping performance. Each tri-strip is joined to the next by a degenerate triangle, invisible to the eye. When there are this many short strips a considerable amount of additional geometry may be created by these degenerates. Sometimes in these circumstances it can be a better option to export the model as tri-lists instead.

There are six conditions which will cause tri-strips to break.

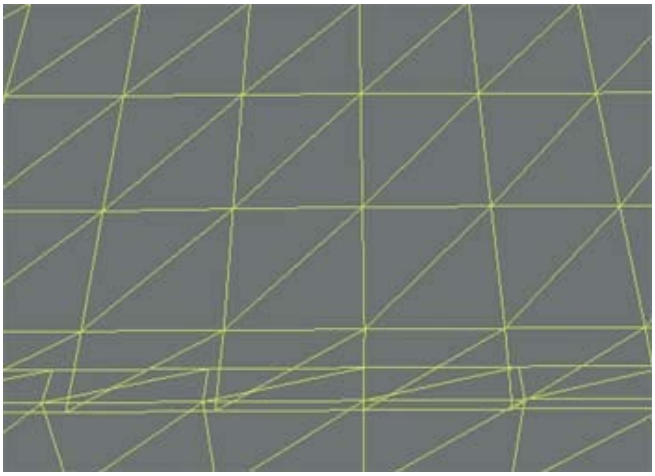
- Geometry Structure
- Surface Smoothing
- UV Mapping
- Material Assignments
- Pre-lighting

- World Sectoring

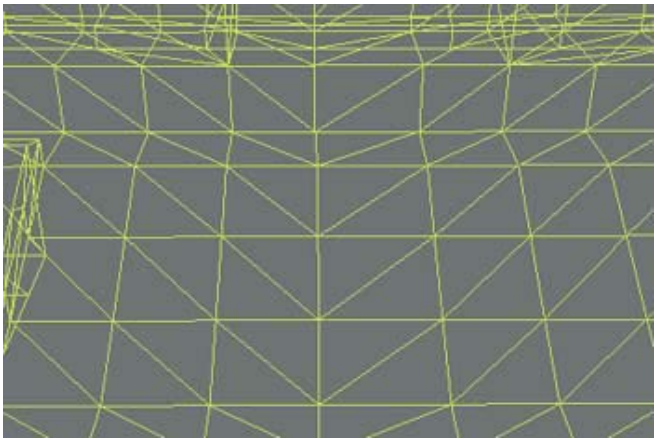
Let's detail each in turn:

Geometry Structure

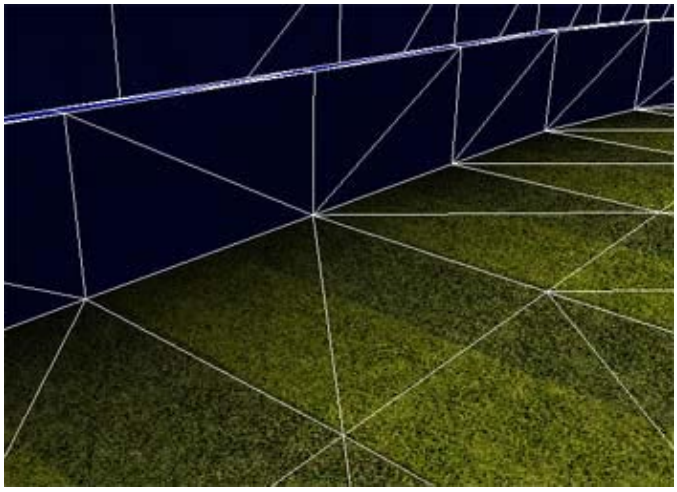
In order to tri-strip, the geometry needs to be as continuous as possible. The stadium is good in most areas in this respect.



There are one or two places where this is not the case though. It appears that a large part of the stadium has been 'mirrored'. Mirroring is a common and convenient technique for the artist but not good for tri-strips as it creates (in this case) a whole series of tri-fans, which really don't need to be there. A tri-fan is a series of triangles radiating from a central vertex and which cannot be tri-stripped. These should be eliminated wherever possible.



What is required is that all the triangle edges on one or other side of the mirror need to be turned to make the geometry continuous, possibly costing the artist as much time as he saved when he mirrored the object in the first place.



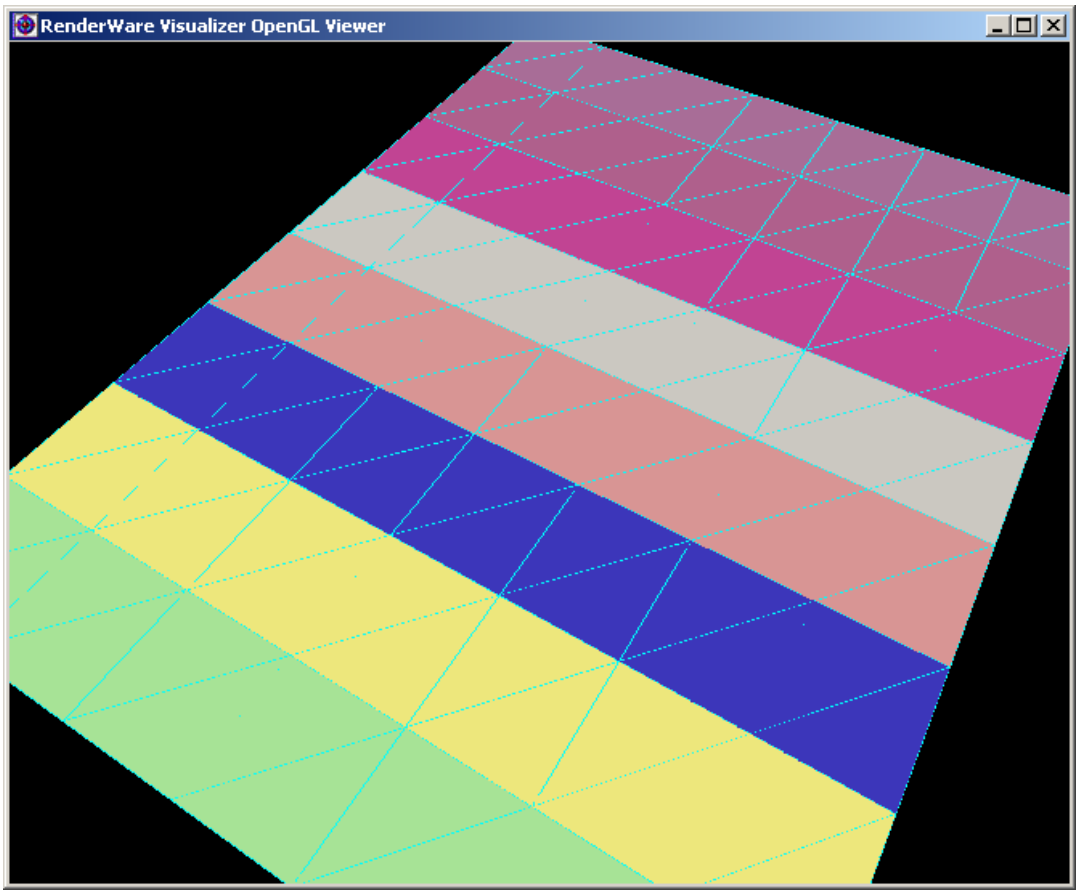
This mirror seam runs right through the whole model, so there is a fair amount of work to be done there.

Surface Smoothing

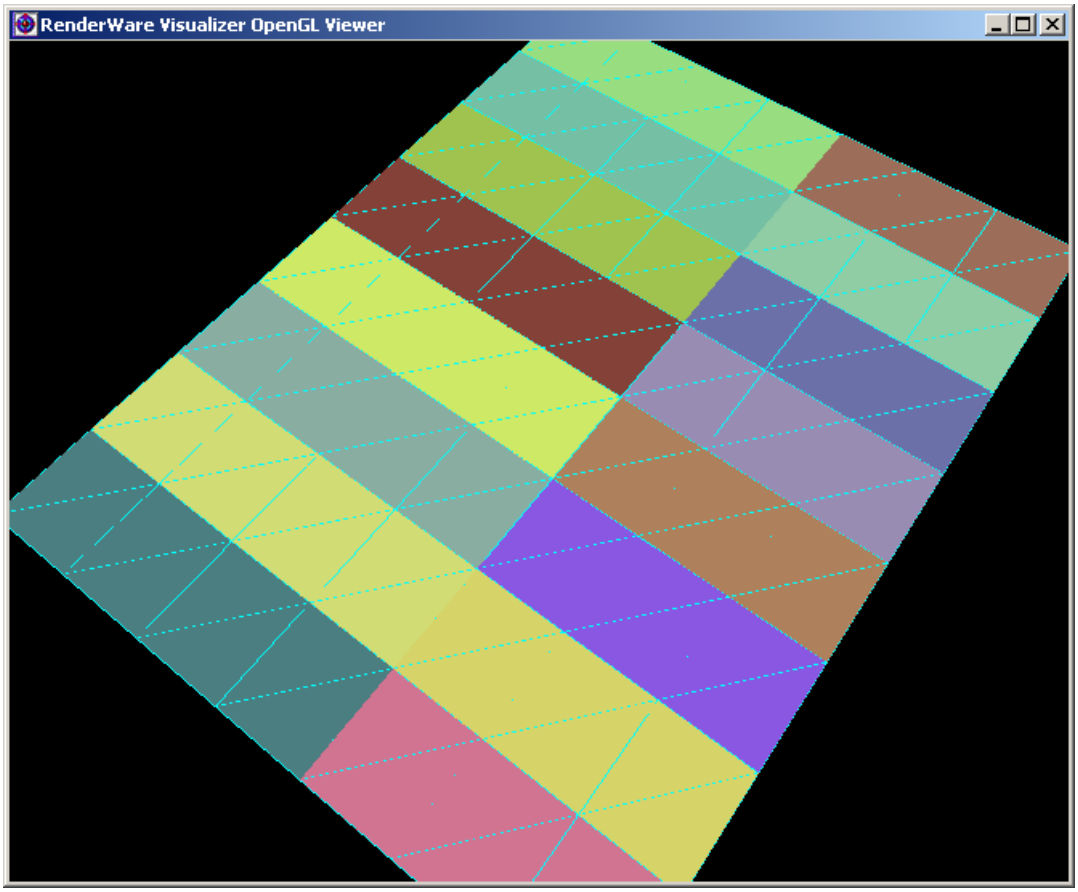
The second factor required for adjacent triangles to strip is that their vertices vectors are normalized, or smoothed. In 3ds max all polygons should be assigned to a single Smoothing Group for optimum performance. The equivalent in Maya is that all edges should be Fully Smooth. We recommend that all geometry is smoothed in this way unless a hard edge is specifically required by the model.

UV Mapping

When materials are tiled across a surface they are made to repeat a given number of times. If the tiling values are sequential (0,1,2,3,4,5,etc...) they will not affect tri-stripping performance.



If the values are 'clamped' at a given value (0,1,2,0,1,2,etc...) then the tri-strips will be terminated where the repeats occur.

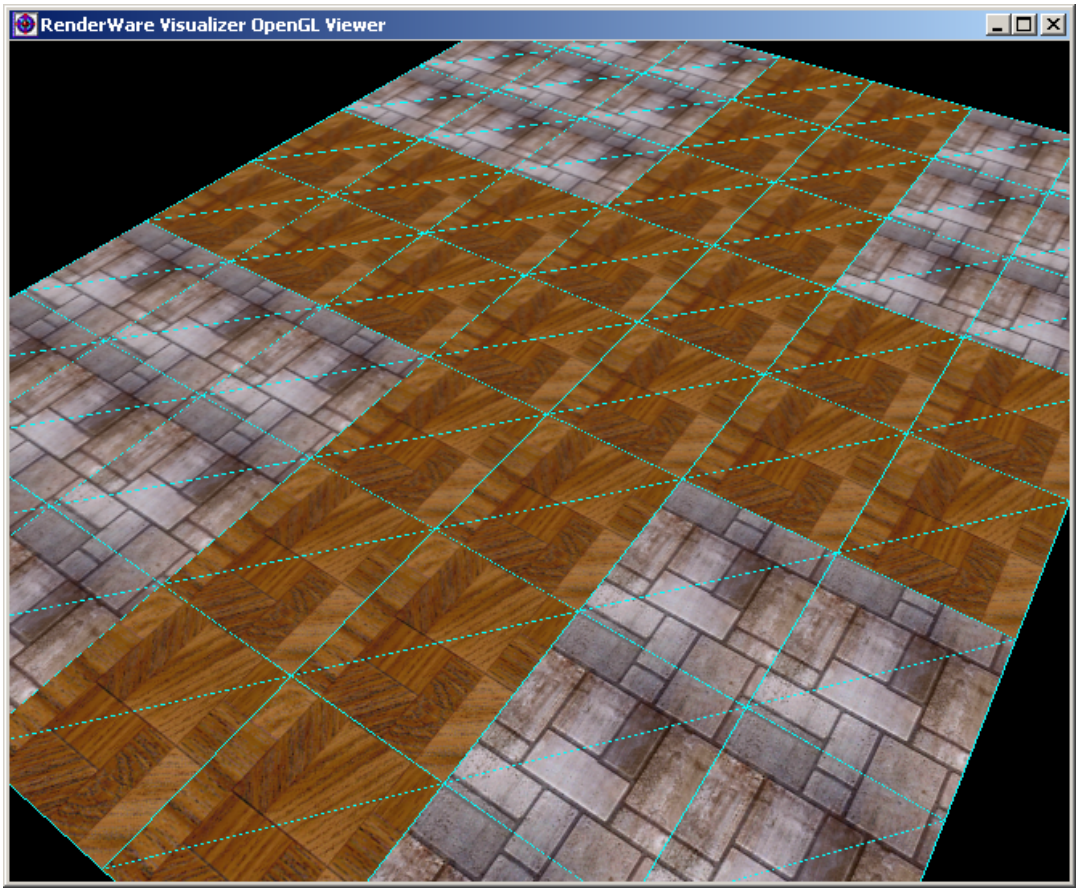


On some platforms, such as PlayStation 2, clamping the values is desirable so a degree of experimentation would be required to obtain the best compromise.

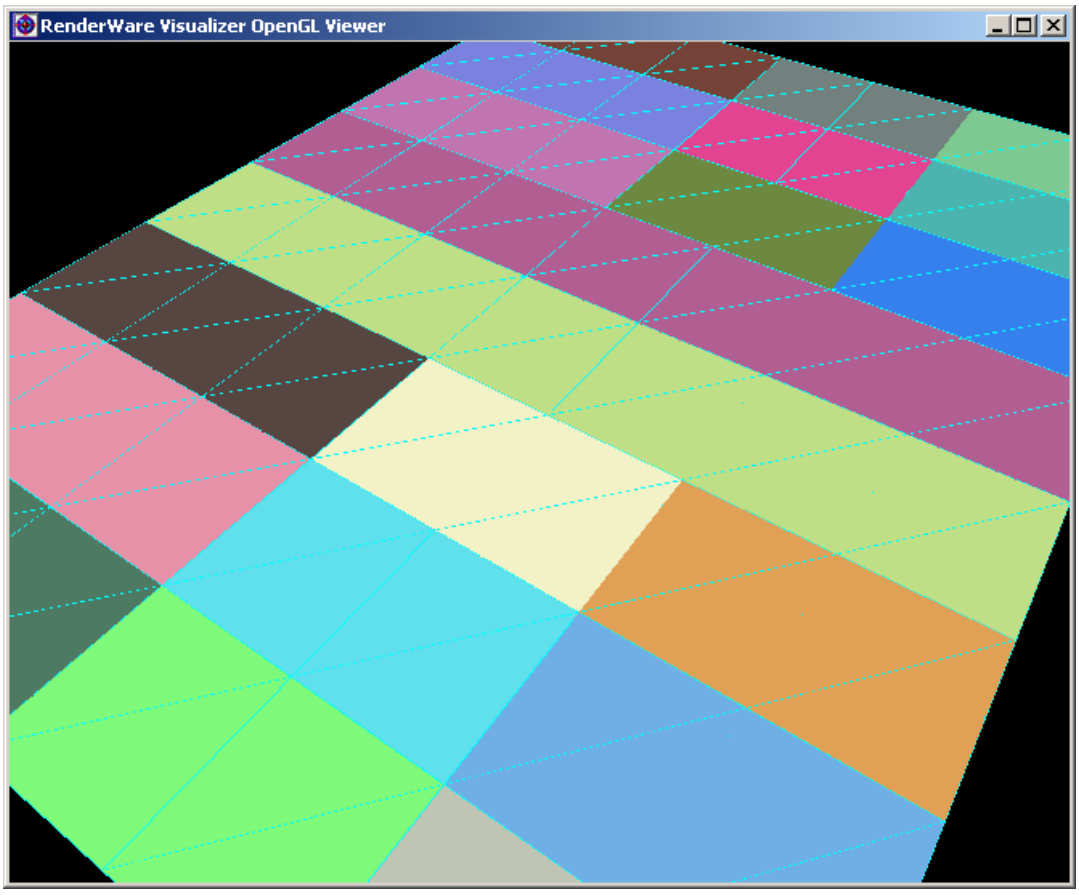
Material Assignments

Again where a single texture map is repeated there will be no problems with tri-strips.

When different maps are on adjacent triangles that share common vertices the strips will terminate. As in the following example:



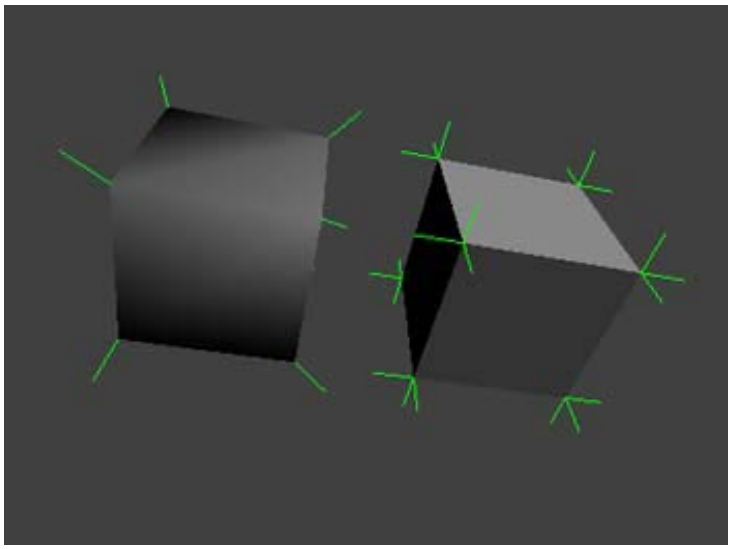
With unclamped UV mapping this will give the following tri-stripping result:



Pre-lighting

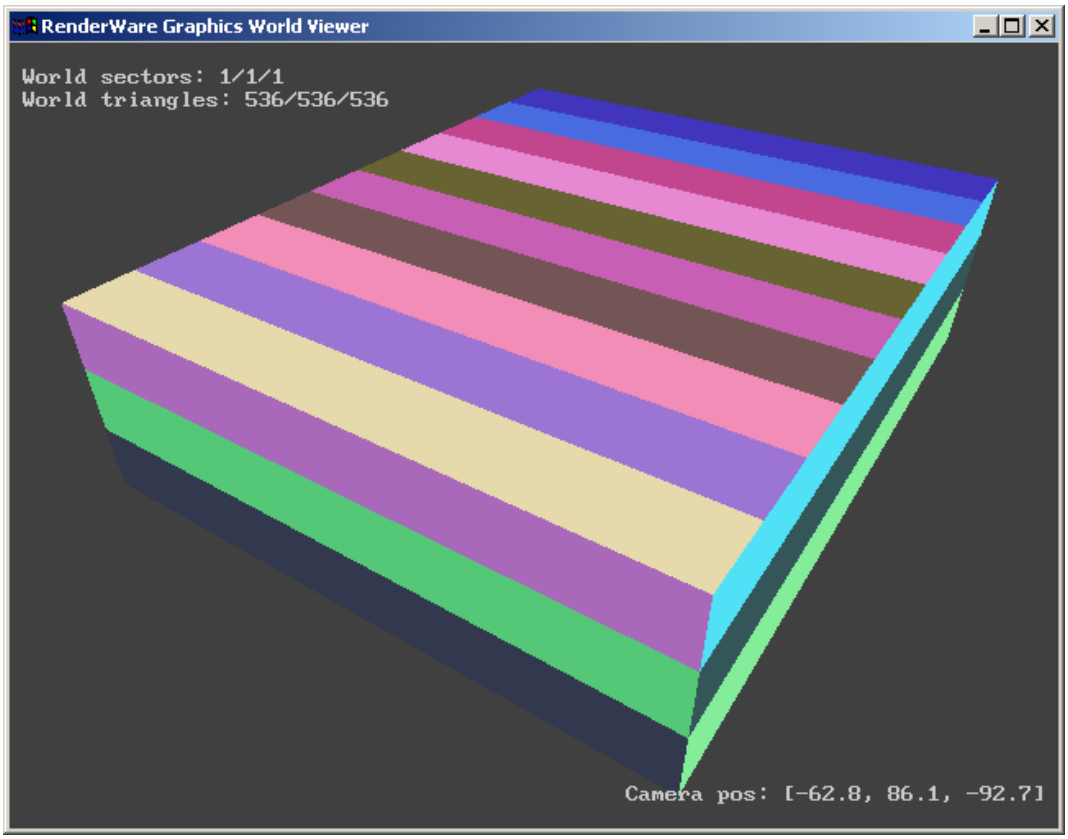
When vertex lighting is employed, there are certain restraints that need to be applied to avoid breaking tri-strips. The easiest way to describe this is to imagine prelighting two cubes as below.

The left hand cube has normalized vectors and therefore the pre-lighting values are common to all attached triangles. Where pronounced changes in vertex light values occur between adjacent triangles as in the hard edges of the cube on the right vertices may be split two, three or even more times to reflect this, thus terminating tri-strips.



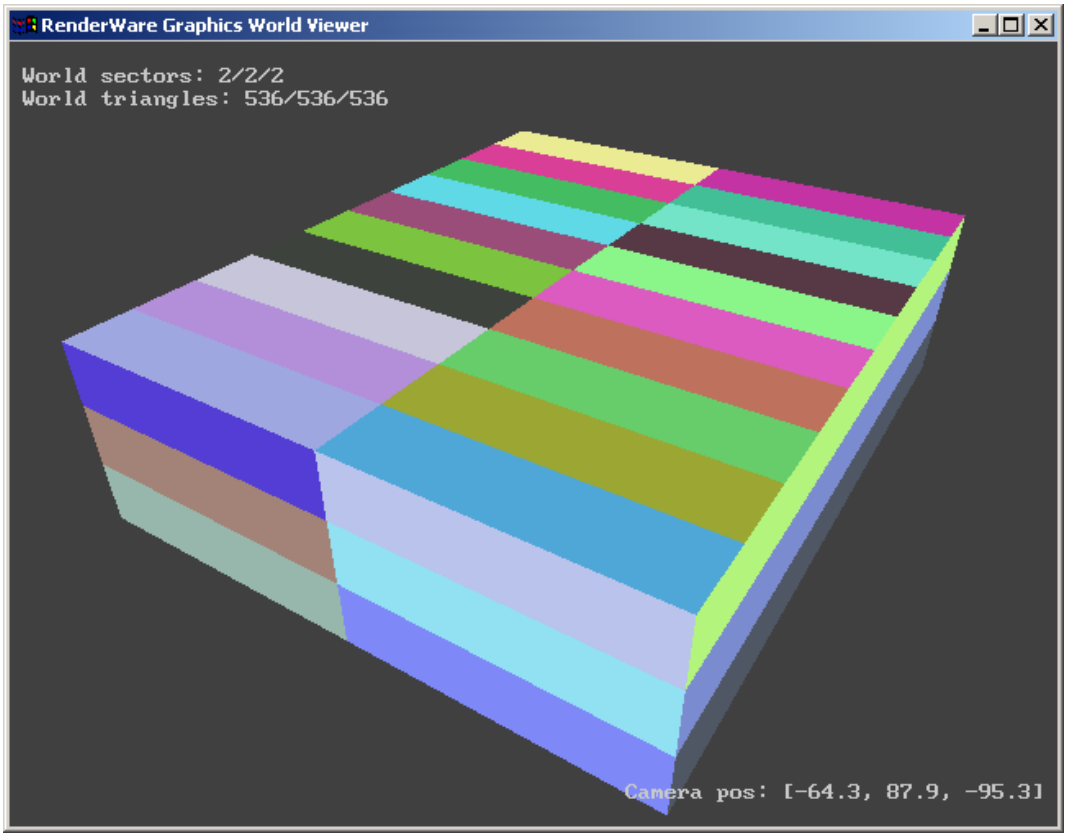
World Sectoring

World Sectors will also break tri-strips where they intersect the geometry. In the example below all the geometry is contained within a single sector so the strips are unbroken:



In the next example the World Sector maximum poly count has been reduced dividing the geometry into two world sectors more or less along the center line the of the geometry.

As can clearly be seen the tri-strips at the end of the object have been split into two parts:



Backface Culling on PlayStation 2

The performance of RenderWare Graphics' PlayStation 2 backface culling has been greatly improved from the 3.2 release onward, and we strongly recommend developers to have backface culling switched on where the triangles are large on the screen.. This generally applies when the player is inside a world. Developers' details are given in the platform-specific API Reference, under `RpSkySelectTrueTSClipper` and `RpSkySelectTrueTLClipper`.

Conclusion

All of the above tri-stripping issues were present in the Stadium's geometry, and this is not surprising. World Sectors, UV clamping, Prelighting and all the other things are all desirable (read necessary) elements in a real word piece of artwork. The trick is to find a workable 'trade off' between all these things to get optimum performance whilst still looking good.

With the Tunnel Tri-Stripper, you can set the quality of the result on a scale against time taken to calculate. If you are designing for Xbox or GeForce3 and above graphics card based games try CacheAware Tri-Stripping. This will create tri-strips that are optimized for NVIDIA hardware. For more detail on using the Tri-Strippers please refer to the Artists' Documentation.

We suggest that you export and preview your work often, giving particular attention to tri-stripping performance. Working this way will help you to identify problem areas relatively early on in the design cycle allowing you to amend the artwork in 'bite sized' chunks. Do not model an entire world before exporting it only to discover the bad news all in one go.