

Rendering optimisation in XSI

written by Christoph Schinko

Content:

1. Geometry approximation
 - 1.1. Static surface approximation
 - 1.2. Adaptive surface approximation
2. Rendering
 - 2.1. What is rendering?
 - 2.2. Rendering methods
 - 2.3. Acceleration methods

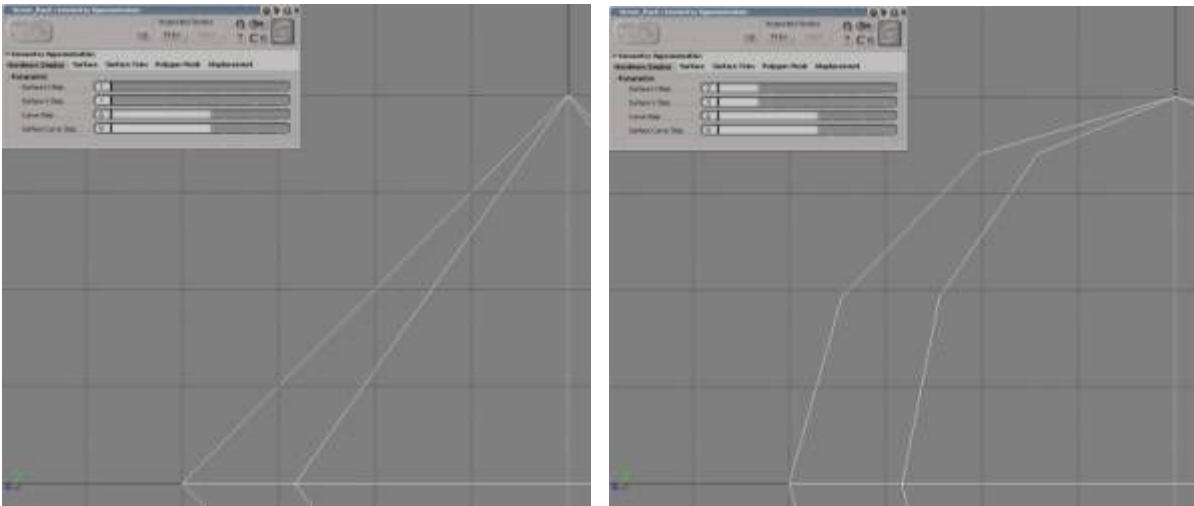
1. Geometry approximation

is a very important issue in the 3D world. While proprietary software of some companies like Pixar is using subdivision surfaces, which automatically makes the object look great in a close up and saves rendering time when it's in the background, it'll take some time until those tools are available to all of us. For now we'll stick with the given tools which are pretty good as well.

In Softimage XSI even the U and V subdivisions of a model can be animated. This can be useful setting up rough LOD (level of detail). Animated carefully, the jump between the individual steps cannot be seen once rendered. Note that imported objects do not have construction history, so the subdivisions cannot be animated.

1.1. Static surface approximation

Parametric steps are the most common values to define the detail between knots or lines on an object.

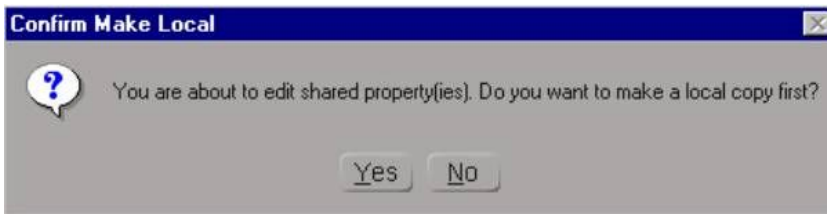


Picture 1: parametric steps of a sphere

Beside the static surface approximation, the parametric steps, XSI has all necessary options to handle adaptive surface approximation. This helps a lot to realize a form of level of detail (LOD), which can save rendering time enormously.

1.2. Adaptive surface approximation

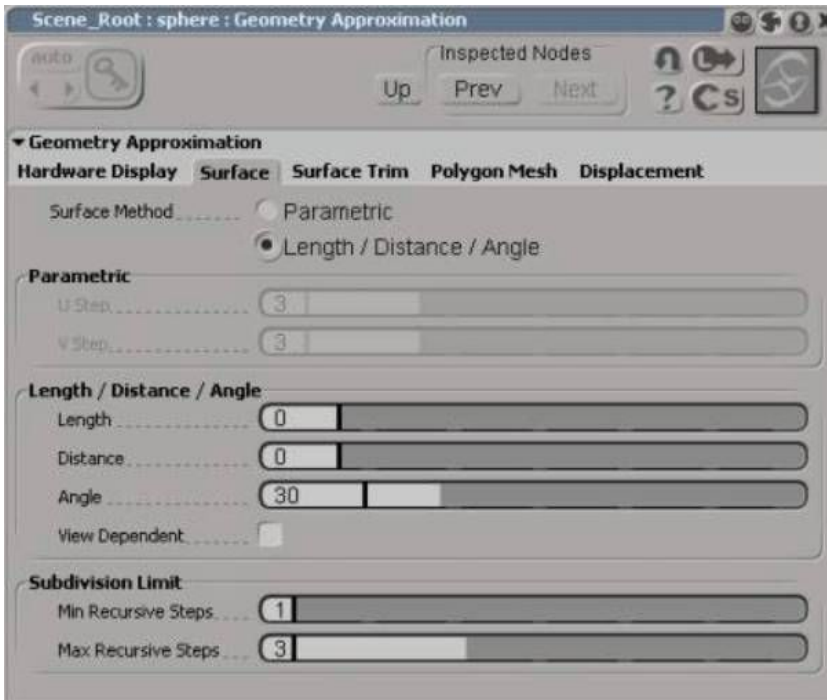
With the object selected you'll find the point geometry approximation in the property editor. When clicking on the icon to open the window, the following question appears:



Picture 2: Confirm make local copy

What that means is that the object didn't get its own properties yet, it is using the scene defaults. If a local copy is created, the object gets its individual geometry approximation node that can be changed as desired.

When no local copy is created the scene defaults are edited and all objects will be affected by changes.



Picture 3: Surface approximation window - LDA

The following options appear in the window:

- Hardware Display
- Surface
- Polygon Mesh
- Surface Trim
- Displacement

We can already guess that we are able to set different step values for the display in the view port, the actual rendered surface, trims on that object and displacement maps. This opens up new possibilities to optimise rendering.

Hardware Display sets how detailed the object will be displayed in the XSI view ports, meaning that the object itself is not affected by those values it just speeds up interaction in the user interface (UI). Besides the parametric steps of 3D objects, we can also define the steps between the knots of NURBS curves and the steps for surface curves (isolines, boundaries).

Polygon Mesh lets us adjust the handling of discontinuity and subdivision.

Surface, Surface Trim and Displacement allows us to choose between two methods of geometry approximation. Parametric and Length/Distance/Angle (LDA).

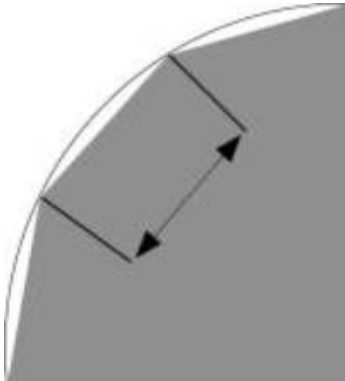
The LDA method is very similar to the solutions used in Softimage.

The LOD of an object depends on how close it is to the camera. So if it's far away, the subdivisions

don't have to be that high, but as that object or character gets closer to the camera more and more corners are seen and it just doesn't appear smooth anymore. Now we can define certain values and XSI subdivides the object more when these values aren't met anymore. This happens only during rendering, because that's when NURBS objects get tessellated or converted into triangles (still). So that's how we can control the degree of tessellation.

The dry facts:

Length defines at which length a tessellated edge of a surface will be subdivided more.



Picture 4: diagram length

Distance measures the distance between the tessellated edge and the actual surface. Is the distance greater than the given value, the object will be subdivided further.



Picture 5: diagram distance

Angle considers the angle between the normals of two tessellated adjacent edges. Does the angle exceed the value, subdivision will occur.



Picture 6: diagram angle

All those three values are connected to the other options in the menu, *view dependency* and *Subdivision limits*.

View dependency switches from measuring the defined values in camera space to using raster space.

If values are set too low XSI would subdivide objects forever, so there we can define how many times the geometry can be subdivided.

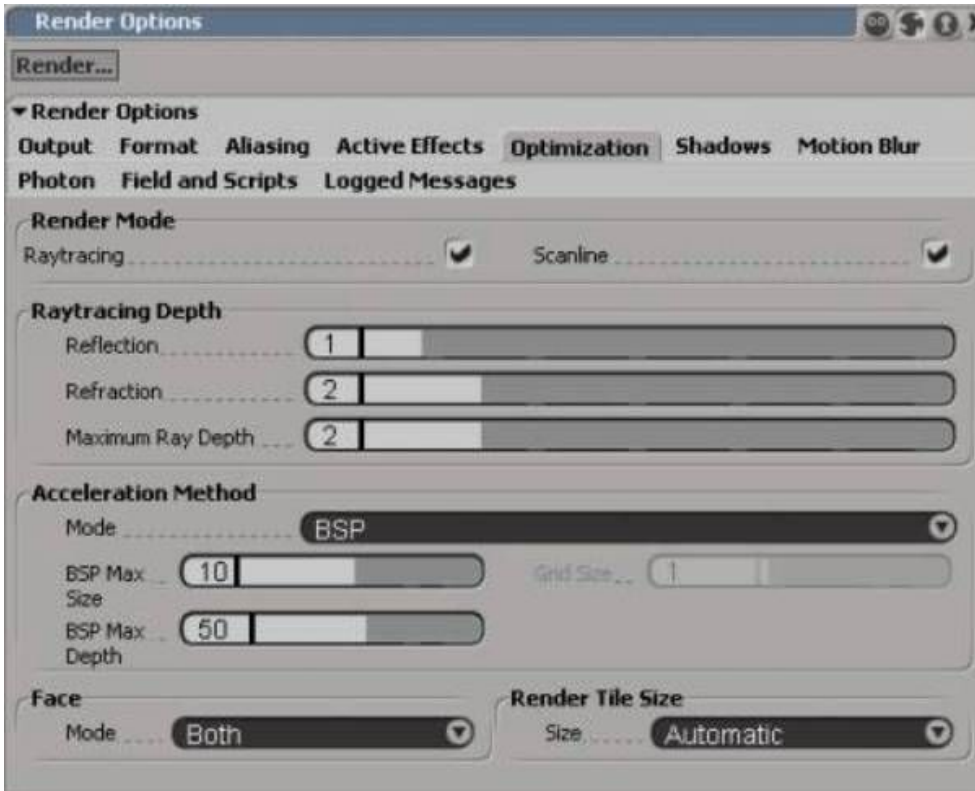
Subdivision limits can be set for every object individually and tell XSI the minimum and maximum

subdivision samples, called *min* and *max recursive steps*.

2. Rendering

Another very powerful way of speeding up the rendering of a scene is to optimise the various settings that tell XSI how to prepare a scene for rendering and which method it shall use.

In order to do that successfully, one has to understand what happens when the software is "rendering".



Picture 7: Rendering optimisation

2.1. What is rendering?

Rendering is the calculation of your scene in XSI's internal three dimensional space to create an image or image sequence. For this procedure the render engine sends out one ray at a time, from where the camera is positioned. This ray travels in a straight manner from the camera centre into the XSI space until it hits a triangle (objects get divided into triangles -> tessellation). If there is no object in that direction, the ray would just keep on going forever and XSI would end up in infinite recursion. To prevent that, a process called subdividing is started in which the renderer sets borders around your scene, in the form of a box. To make this box easier to handle and calculate for the render engine it is cut in half again, first in x, then in y and z. Those eight boxes get cut in half again, following the same order. This process continues until all boxes are easy to handle for the renderer, and certain settings in the render options determine how far XSI will go with subdividing the scene.

When a ray hits a triangle, it is calculated which color the material has and if the pixel is hit by light. If the light has ray traced shadows on, rays will be emitted from the light source as well, to see where shadows are cast.

If an object is transparent the ray passes through, according to how transparent a surface is. However, the ray continues its straight path and adds the remaining color of the first object to the second or third one it hits.

If a refraction index is set, the rays path will be altered and it keeps going in a slightly different direction. Like when light passes through another medium as it does from air to water.

Same with reflections, the ray will be shot back (reflected) from the object and will show whatever comes after that.

Reflection maps are fake reflections, handled as textures.

As we can see, there's basically two ways of travelling rays. Straight and not-straight.

Also, at certain occasions the ray can be split into two or more rays that shoot in different directions. That happens for example when a partly transparent object is reflective at the same time or has refraction as well.

It can be set how many rays XSI will handle at the same time and when they stop bouncing around, from reflection to reflection.

Those values are found in the render **options->optimisation->ray tracing depth**. Here we can set the maximums for reflections, refractions and the global maximum ray depth.

But in this menu there is a lot more to find...

2.2. Rendering methods

XSI uses two algorithms of rendering a scene:

- Scan line
- Ray tracing

Scan line ("straight"): Is the more simplistic approach, the image being calculated scan line by scan line instead of object by object. While features like transparency and reflection maps can be displayed using scan line, everything that would re-direct the ray (like reflection, refraction, area lights,...) cannot be computed using scan line. The beauty (and speed) in scan line lies in that only the quadrant the renderer is working on has to be loaded into memory. As no rays are traced it is not necessary to keep the whole scene in the RAM. Although scan line is much faster than ray tracing, the missing features make the result look less realistic than a ray traced image.

Ray tracing ("not-straight"): is the "photo realistic" way of rendering an image. It can make the ray change direction and handle all the tricky stuff like reflection, refraction, ray traced shadows,... Results look great, but rendering time can be very time consuming. That is why mental ray supports two ways of acceleration methods to speed up rendering. The switches for those settings are found under render **options->optimisation->render mode**.

2.3. Acceleration methods

In mental ray there are two rendering acceleration algorithms to choose from:

- BSP tree
- Grid

BSP tree

The BSP tree (binary space partitioning) is the more common way to render a scene it gives good results for almost every case, although it is sometimes very slow for scenes bigger than 150.000 triangles. To find out how big your scene is, go to **Edit->Info Scene** or hit **Ctrl+Enter**.

Functionality:

The BSP tree is defined by the scene and the settings you specify. When the scene is subdivided into those little boxes mentioned before, a BSP tree is build which contains information on how much stuff is in each cluster or box. Depending on how complex a model is, more boxes will be created where the object is positioned. That means that those clusters will be smaller than the big ones, where there is nothing complex happening. The clusters have different size, which greatly improves rendering speed. A soccer player in the middle of a stadium would be a great example. Also scenes with a lot of reflection, refraction, motion blur,... are handled very well by the BSP tree method. Unfortunately it is very hard to estimate the "stuff" happening in all the boxes, you never know how deep the tree will grow. So there is no control over how big the tree will be and how much RAM it will need, which leads us to another problem.

The other major drawback is in distributed rendering, the BSP tree has to be present on all machines

rendering, so the tree will be sent to every involved machine over the network, which can take forever.

Settings:

Basically, there is two values to adjust when using the BSP tree method. Max Depth and Max Size.
Max Depth: It is the number of subdivisions mental ray is allowed to make when building the tree. The default value is 50 boxes per branch of the tree.

Max Size: The limit of triangles in a box. The default value is 10.

Those two values are related to each other.

If the tree has many subdivisions more memory will be needed, but pre-processing and rendering time will be shorter and vice versa. It's always a trade off.

Optimisation:

Experimenting with those settings can reduce rendering time greatly.

When the verbose option is activated while rendering, the size of the largest leaf node and the number of candidate triangles per ray.

If these numbers are much bigger than 10, try building a deeper tree by increasing the maximum tree depth (30+). That's what the manual says.

Personally, I prefer to sit down with a stop watch, try out some extreme combinations and interpolate between them until I have the best rendering speed.

It is not unusual to get a scene rendered twice as fast after careful optimisation.

Grid

The grid acceleration method gives much faster pre-processing time and is better able to handle scenes bigger than 150.000 triangles. Again, it depends on the scene how fast it will be eventually. Controlling of the memory allocation and a straight forward structure of the tree gives great advantage in network rendering.

However, even a heavy scene with unevenly distributed complexity of the models will prove the grid method to be the wrong one compared to the BSP tree method.

Functionality:

Grid divides the scene into evenly sized boxes, which means a very simple tree. Therefore network traffic won't be so heavy, although local RAM could be used extensively.

Settings:

To adjust the box size, use *Grid Size*.

All settings are found under render **options->optimisation->acceleration method** or **property->rendering options**.

I hope this tutorial could help you understand a little more about the process rendering and the optimisation of it. Have fun!

Please e-mail feedback to stoph@christoph-schinko.com
V1.7 written by Christoph Schinko 09/00