Autodesk® Beast™ 2011

Beast XML and Lua Reference Manual

Autodesk and Beast are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries.

Introduction

Beast uses an XML configuration system for a majority of its settings.
Please Refer to the Beast API documentation on how to specify to Beast which XML configuration to use. This manual also contains a Lua reference for advanced baking together with some sample code.

XML Settings

Here you will find short descriptions of the most important XML-tags in Beast. This manual is intended to be a reference, not an introduction to Global Illumination.

FrameSettings

```
<FrameSettings>
    <autoThreads>true</autoThreads>
    <autoThreadsSubtract>0</autoThreadsSubtract>
    <renderThreads>2</renderThreads>
    <dither>true</dither>
    <inputGamma>1.0</inputGamma>
    <outputCorrection>
        <colorCorrection>None</colorCorrection>
        <gamma>1</gamma>
    </outputCorrection>
    <tileScheme>Hilbert</tileScheme>
    <premultiply>true</premultiply>
    <premultiplyThreshold>0.0</premultiplyThreshold>
    <outputVerbosity>
        <errorPrint>true</errorPrint>
        <warningPrint>true</warningPrint>
        <benchmarkPrint>false</benchmarkPrint>
        <progressPrint>true</progressPrint>
        <infoPrint>false</infoPrint>
        <verbosePrint>false</verbosePrint>
        <debugPrint>false</debugPrint>
    </outputVerbosity>
</FrameSettings>
```

autoThreads If enabled, Beast will try to auto detect the CPU configuration and use one thread per core.
autoThreadsSubtract If autoThreads is enabled, this can be used to decrease the number of utilized cores, e.g. to leave one or two cores free to do other work.
renderThreads If autoThreads is disabled, this will set the number of threads beast uses. One per core is a good start.
dither If the output is LDR, and dither is true, the resulting image will be dithered. Default is true.
inputGamma A float value specifying what gamma the input data has. *Always* set this to 1.0 / 2.2 = 0.454545, which is the gamma the Beast API assumes is used.
colorCorrection Set the mode of output color correction to None, Gamma or SRGB. The Beast API assumes this is set to Gamma.
gamma A float value specifying what gamma the output data should have. The Beast API assumes this is set to 2.2.
tileScheme Different ways for Beast to distribute tiles over the image plane.

- **LeftToRight**: Render from left to right.
- **Hilbert**: A way for Beast to achieve maximum coherence, e.g., the fastest rendering time possible.
- **Random**: A good way to get an early feel for the whole picture without rendering everything.
- **Concentric**: Starts in the middle and renders outward in a spiral.

tileSize A smaller tile gives better ray tracing coherence. There is no “best setting” for all scenes. Default value is 32, giving 32x32 pixel tiles. The largest allowed tile size is 128.
premultiply If this box is checked the alpha channel value is pre multiplied into the color channel of the pixel. Note that disabling premultiply alpha gives poor result if used with environment maps and other non constant camera backgrounds. **Disabling** premultiply alpha can be convenient when composing images in post.
premultiplyThreshold This is the alpha threshold for pixels to be considered opaque enough to be “un multiplied” when using premultiply alpha.
outputVerbosity Different levels of textual output that Beast can produce.

- **debugFile** Save all log messages to a file named debug.out.
- **errorPrint**
- **warningPrint**
- **benchmarkPrint**
- **progressPrint**
- **infoPrint**
- **verbosePrint**
- **debugPrint** Used for development purposes.

RenderSettings

```
<RenderSettings>
    <bias>0.005</bias>
    <giTransparencyDepth>2</giTransparencyDepth>
    <ignoreLightLinks>false</ignoreLightLinks>
    <maxRayDepth>6</maxRayDepth>
    <maxShadowRays>4294967295</maxShadowRays>
    <minShadowRays>0</minShadowRays>
    <reflectionDepth>2</reflectionDepth>
    <reflectionThreshold>0.001</reflectionThreshold>
    <shadowDepth>2</shadowDepth>
    <shadowsIgnoreLightLinks>false</shadowsIgnoreLightLinks>
    <transparencyDepth>50</transparencyDepth>
    <tsIntersectionNormalization>true</tsIntersectionNormalization>
    <tsIntersectionOrthogonalization>true</tsIntersectionOrthogonalization>
    <tsOddUVFlipping>true</tsOddUVFlipping>
    <tsVertexNormalization>true</tsVertexNormalization>
    <tsVertexOrthogonalization>true</tsVertexOrthogonalization>
    <vertexMergeThreshold>0.001</vertexMergeThreshold>
</RenderSettings>
```

maxRayDepth The maximum amount of ”bounces” a ray can have before being considered done. A bounce can be a reflection or refraction.
reflectionDepth The maximum amount of reflections a ray can have before being considered done.
reflectionThreshold If the intensity of the reflected contribution is less than the threshold, the ray will be terminated.
giTransparencyDepth Controls the maximum transparency depth for Global Illumination rays. Used to speed up renderings with a lot of transparency (for example trees).
shadowDepth Controls which rays that spawn shadow rays. If set to 1, only primary rays spawn shadow rays. If set to 2, the first secondary ray spawns a shadow ray as well.
maxShadowRays The maximum number of shadow rays per point that will be used to generate a soft shadow for any light source. Use this to shorten render times at the price of soft shadow quality. This will lower the maximum number of rays sent for any light sources that have a shadow samples setting higher than this value, but will not raise the number if shadow samples is set to a lower value.
minShadowRays The minimum number of shadow rays that will be sent to determine if a point is lit by a specific light source. Use this value to ensure that you get enough quality in soft shadows at the price of render times. This will raise the minimum number of rays sent for any light sources that have a minShadowSamples setting lower than this value, but will not lower the number if minShadowSamples is set to a higher value. Setting this to a value higher than maxShadowRays will not send more rays than maxShadowRays.
ignoreLightLinks If true, light-links will be ignored and all available light sources will be used.
shadowsIgnoreLightLinks Make objects cast shadows from all light sources, not only the light-linked light sources.
bias An error threshold to avoid double intersections. For example, a shadow ray should not intersect the same triangle as the primary ray did, but because of limited numerical precision this can happen. The bias value moves the intersection point to eliminate this problem. If set to zero this value is computed automatically depending on the scene size.
tsOddUVFlipping Using this setting will force Beast to mirror tangent and binormal when UV has odd winding direction.
tsVertexOrthogonalization Orthogonalize tangent space basis vectors (tangent, binormal and normal) at every vertex.
tsVertexNormalization Normalize tangent space basis vectors (tangent, binormal and normal) at every vertex.
tsIntersectionOrthogonalization Orthogonalize tangent space basis vectors (tangent, binormal and normal) at every intersection point.
tsIntersectionNormalization Normalize tangent space basis vectors (tangent, binormal and normal) at every intersection point.
vertexMergeThreshold Triangle vertices that are closer together than this threshold will be merged into one (if possible depending on other vertex data).

AASettings

```
<AASettings>
    <maxSampleRate>0</maxSampleRate>
    <minSampleRate>-2</minSampleRate>
    <contrast>0.1</contrast>
    <diagnose>false</diagnose>
    <clamp>false</clamp>
    <minValue>0</minValue>
    <maxValue>1</maxValue>
    <filter>box</filter>
    <filterSize>
```

```
        <x>1</x>
        <y>1</y>
    </filterSize>
</AASettings>
```

Adaptive Sampling - Number of Samples per pixel

Max Sampling / Min Sampling	-4	-3	-2	-1	0	1	2	3	4
-4	1/256	1/256 - 1/64	1/256 - 1/16	1/256 - 1/4	1/256 - 1	1/256 - 4	1/256 - 16	1/256 - 64	1/256 - 256
-3	1/64	1/64	1/64 - 1/16	1/64 - 1/4	1/64 - 1	1/64 - 4	1/64 - 16	1/64 - 64	1/64 - 256
-2	1/16	1/16	1/16	1/16 - 1/4	1/16 - 1	1/16 - 4	1/16 - 16	1/16 - 64	1/16 - 256
-1	1/4	1/4	1/4	1/4	1/4 - 1	1/4 - 4	1/4 - 16	1/4 - 64	1/4 - 256
0	1	1	1	1	1	1 - 4	1 - 16	1 - 64	1 - 256
1	4	4	4	4	4	4	4 - 16	4 - 64	4 - 256
2	16	16	16	16	16	16	16	16 - 64	16 - 256
3	64	64	64	64	64	64	64	64	64 - 256
4	256	256	256	256	256	256	256	256	256

The AASettings contains the anti aliasing settings for Beast.

maxSampleRate Controls the maximum number of samples per pixel. Default value is 0.
minSampleRate Controls the minimum number of samples per pixel. Values less than 0 allows using less than one sample per pixel (if AdaptiveSampling is used). Default value is 0.
contrast If the contrast differs less than this threshold Beast will consider the sampling good enough. Default value is 0.1.
diagnose Enable this to diagnose the sampling. The brighter a pixel is, the more samples were taken at that position.
clamp To work efficiently on LDR images, the sampling algorithm can clamp the intensity of the image samples to the [minValue..maxValue] range. When rendering in HDR this is not desired. Clamp should then be disabled.

- **minValue** the sample intensity will be higher than this value.
- **maxValue** the sample intensity will be lower than this value.

filter The sub-pixel filter to use. The following filters are available (default value is Box):

- **Box** Smoothing
- **CatmullRom** Smoothing
- **Cubic** Smoothing
- **Gauss** Smoothing
- **Lanczos** Sharpen
- **Mitchell** Sharpen
- **Triangle** Smoothing

filterSize The width and height of the filter kernel in pixels, given by setting the sub elements x and y (float). Default value is 1.0 for both x and y.

GISettings

Beast collects all global illumination algorithms under the main category Global Illumination.

```
<GISettings>
    <clampMaterials>none</clampMaterials>
    <diffuseBoost>1</diffuseBoost>
    <emissiveScale>1</emissiveScale>
    <enableCaustics>false</enableCaustics>
    <enableGI>false</enableGI>
    <ignoreNonDiffuse>true</ignoreNonDiffuse>
    <prepassMaxSampleRate>0</prepassMaxSampleRate>
    <prepassMinSampleRate>-4</prepassMinSampleRate>
    <primaryIntegrator>finalgather</primaryIntegrator>
    <primaryIntensity>1</primaryIntensity>
    <primarySaturation>1</primarySaturation>
    <secondaryIntegrator>pathtracer</secondaryIntegrator>
    <secondaryIntensity>1</secondaryIntensity>
    <secondarySaturation>1</secondarySaturation>
    <specularScale>1</specularScale>
</GISettings>
```

GI integrators

primaryIntegrator and **secondaryIntegrator** The Global Illumination system allows you to use two separate algorithms to calculate indirect lighting. You can for instance calculate multiple levels of light bounces with a fast algorithm like the Path Tracer, and still calculate the final bounce with Final Gather to get a fast high-quality global illumination render. Both subsystems have individual control of Intensity and Saturation to boost the effects if necessary. Available algorithms include:

- **FinalGather**
- **PathTracer**
- **MonteCarlo**

There is also the option to use None both as primary and secondary GI. For secondary GI, use None when you don't need deeper indirect light or if you have a primary GI that cannot use secondary GI. Using None for primary GI should be done when baking advanced passes that perform their own gather, e.g. RNM(when not using radiance cache) and the LUA pass. When these perform their own gather, only the secondary GI will be sampled, making any primary GI useless.
primaryIntensity and **secondaryIntensity** Tweak the amount of illumination from the primary and secondary GI integrators. This lets you boost or reduce the amount of indirect light easily.
primarySaturation and **secondarySaturation** Lets you tweak the amount of color in the primary and secondary GI integrators. This lets you boost or reduce the perceived saturation of the bounced light.
diffuseBoost This setting can be used to exaggerate light bouncing in dark scenes. Setting it to a value larger than 1 will push the diffuse color of materials towards 1 for GI computations. The typical use case is scenes authored with dark materials, this happens easily when doing only direct lighting since it's easy to compensate dark materials with strong light sources. Indirect light will be very subtle in these scenes since the bounced light will fade out quickly. Setting a diffuse boost will compensate for this. Note that values between 0 and 1 will decrease the diffuse setting in a similar way making light bounce less than the materials says, values below 0 is invalid. The actual computation taking place is a per component pow(colorComponent, (1.0 / diffuseBoost)).
specularScale This setting can be used to exaggerate or decrease specular light effects. All materials specular color is multiplied by this factor when they are used by the GI.

emissiveScale This setting globally scales all materials emissive components by the specified value when they are used by the GI.
clampMaterials This setting controls if the materials should be clamped in any way for GI purposes. Typically you can use this to avoid problems with non physical materials making your scene extremely bright. This affects both the specular and diffuse components of materials. Valid values are:

- **None**, No clamping at all
- **Component**, clamps each color component (R, G, B) individually.
- **Intensity**, clamps the intensity of the color to 1. This can be useful to make sure the color of a surface is preserved when clamping. If using component clamp on a color like (3, 1, 1) will give the color (1, 1, 1) which means that all color bleeding is lost.

prepassMin/**MaxSampleRate** Sets the sample rate used during the GI precalculation pass. The prepass is progressive, rendering from a low resolution up to a high resolution in multiple passes. The prepassMinSampleRate sets the initial resolution, where a negative value means a lower resolution than the original image (in powers of two), e.g. -4 gives the original resolution divided by 16. The prepassMaxSampleRate sets the resolution of the final prepass, e.g 0 giving the same resolution as the original resolution. The default values are -4/0.

Final Gather

```
<GISettings>
    <fgAOContrast>1</fgAOContrast>
    <fgAOInfluence>0</fgAOInfluence>
    <fgAOMaxDistance>0</fgAOMaxDistance>
    <fgAOScale>1</fgAOScale>
    <fgAOVisualize>false</fgAOVisualize>
    <fgAccuracy>1</fgAccuracy>
    <fgAttenuationStart>0</fgAttenuationStart>
    <fgAttenuationStop>0</fgAttenuationStop>
    <fgCacheDirectLight>false</fgCacheDirectLight>
    <fgCheckVisibility>false</fgCheckVisibility>
    <fgCheckVisibilityDepth>1</fgCheckVisibilityDepth>
    <fgClampRadiance>false</fgClampRadiance>
    <fgContrastThreshold>0.1</fgContrastThreshold>
    <fgDepth>1</fgDepth>
    <fgDisableMinRadius>false</fgDisableMinRadius>
    <fgEstimatePoints>15</fgEstimatePoints>
    <fgExploitFrameCoherence>false</fgExploitFrameCoherence>
    <fgFalloffExponent>0</fgFalloffExponent>
     <fgGradientThreshold>0.5</fgGradientThreshold>
    <fgInterpolationPoints>15</fgInterpolationPoints>
    <fgLightLeakRadius>0</fgLightLeakRadius>
    <fgLightLeakReduction>false</fgLightLeakReduction>
    <fgMaxRayLength>0</fgMaxRayLength>
    <fgNormalThreshold>0.2</fgNormalThreshold>
    <fgPreview>true</fgPreview>
    <fgRays>300</fgRays>
    <fgSmooth>1</fgSmooth>
    <fgUseCache>irradiance</fgUseCache>
    <fgUseLegacy>false</fgUseLegacy>
</GISettings>
```

Final Gather is a high quality GI method that should be used as the primaryIntegrator in most cases.
fgUseCache Selects what caching method to use for final gathering.

- **Off** (Brute Force): disables caching and performs a final gathering for every shading point (same as Monte Carlo, see below).
- **Irradiance**: caches the irradiance at selected points in the scene and uses interpolation in between the points. This is the default method.
- **RadianceSH**: caches radiance SH functions at selected points in the scene and uses interpolation in between the points. The radiance cache is very useful in some advanced baking passes (e.g. Radiosity Normal Maps), where directional indirect lighting is needed.

fgRays Sets the maximum number of rays to use for each Final Gather sample point. A higher number gives higher quality, but longer rendering time.
fgDepth Sets the number of indirect light bounces calculated by final gather. A value higher than 1 will produce more global illumination effects, but note that it can be quite slow since the number of rays will increase exponentially with the depth. It's often better to use a fast method for secondary GI. If a secondary GI is used the number of set final gather bounces will be calculated first, before the secondary GI is called. So in most cases the depth should be set to 1 if a secondary GI is used.
fgInterpolationPoints Sets the number of final gather points to interpolate between. A higher value will give a smoother result, but can also smooth out details. If light leakage is introduced through walls when this value is increased, checking the sample visibility solves that problem, see fgCheckVisibility below.
fgPreview Turn this on to visualize the final gather prepass. Using the Preview Calculation Pass enables a quick preview of the final image lighting, reducing lighting setup time.
fgUseLegacy Enables the older final gather method. With this method points are placed according to the geometry in the scene. More samples are placed near walls and corners of the geometry. The contrast difference of the samples are not considered. When the legacy method is enabled the controls for the new method (fgInterpolationPoints, fgEstimatePoints, fgContrastThreshold, fgNormalThreshold and fgGradientThreshold) is not used.
fgAccuracy (legacy) Controls the sample point density. Higher accuracy generates more points and a better result at the expense of longer rendering time.
fgSmooth (legacy) Applies a filter that reduces noise in the final gather solution. This is much faster than increasing accuracy or sending more rays. 1.0 is the default value. Values below 1.0 gives a sharper and noisier look, values above 1.0 gives a smoother look with less details preserved.
fgDisableMinRadius (legacy) When enabled more points will be created in corners of the geometry, which can reduce artifacts. However don't enable this unless you need to, since it will increase the render time.
fgEstimatePoints Sets the minimum number of points that should be used when estimating final gather in the pre calculation pass. The impact is that a higher value will create more points all over the scene. The default value 15 rarely needs to be adjusted.
fgContrastThreshold Controls how sensitive the final gather should be for contrast differences between the points during pre calculation. If the contrast difference is above this threshold for neighbouring points, more points will be created in that area. This tells the algorithm to place points where they are really needed, e.g. at shadow boundaries or in areas where the indirect light changes quickly. Hence this threshold controls the number of points created in the scene adaptively. Note that if a low number of final gather rays are used, the points will have high variance and hence a high contrast difference, so in that case you might need to increase the contrast threshold to prevent points from clumping together.
fgNormalThreshold Controls how sensitive the final gather should be for differences in the points normals. A lower value will give more points in areas of high curvature.
fgGradientThreshold Controls how the irradiance gradient is used in the interpolation. Each point stores it's irradiance gradient which can be used to improve the interpolation. However in some situations using the gradient can result in white "halos" and other artifacts. This threshold can be used to reduce those artifacts.
fgMaxRayLength The max distance a ray can be traced before it's considered to be a "miss". This can improve performance in very large scenes. If the value is set to 0.0 the entire scene will be used.
fgClampRadiance Turn this on to clamp the sampled values to [0, 1]. This will reduce high frequency noise when Final Gather is used together with other Global Illumination algorithms.
fgCacheDirectLight When this is enabled final gather will also cache lighting from light sources. This increases performance since fewer direct light calculations are needed. It gives
an approximate result, and hence can affect the quality of the lighting. For instance indirect light bounces from specular highlights might be lost. However this caching is only done for

depths higher than 1, so the quality of direct light and shadows in the light map will not be reduced.
fgCheckVisibility Turn this on to reduce light leakage through walls. When points are collected to interpolate between, some of them can be located on the other side of geometry.
As a result light will bleed through the geometry. So to prevent this Beast can reject points that are not visible.
fgLightLeakReduction This setting can be used to reduce light leakage through walls when using final gather as primary GI and path tracing as secondary GI. Leakage, which can happen when e.g. the path tracer filters in values on the other side of a wall, is reduced by using final gather as a secondary GI fallback when sampling close to walls or corners. When this is enabled a final gather depth of 3 will be used automatically, but the higher depths will only be used close to walls or corners. Note that this is only used when path tracing is set as secondary GI.
fgLightLeakRadius Controls how far away from walls the final gather will be called again, instead of the secondary GI. If 0.0 is used a value will be calculated by Beast depending on the secondary GI used. The calculated value is printed in the output window. If you still get leakage you can adjust this by manually typing in a higher value.
fgAttenuationStart The distance where attenuation is started. There is no attenuation before this distance. Note that fgAttenuationStop must be set higher than 0.0 to enable attenuation.
fgAttenuationStop Sets the distance where attenuation is stopped (fades to zero). There is zero intensity beyond this distance. To enable attenuation set this value higher than 0.0. The default value is 0.0.
fgFalloffExponent This can be used to adjust the rate by which lighting falls off by distance. A higher exponent gives a faster falloff. Note that fgAttenuationStop must be set higher than 0.0 to enable attenuation.
fgAOInfluence Controls a scaling of Final Gather with Ambient Occlusion which can be used to boost shadowing and get more contrast in you lighting. The value controls how much Ambient Occlusion to blend into the Final Gather solution.
fgAOMaxDistance Max distance for the occlusion. Beyond this distance a ray is considered to be visible. Can be used to avoid full occlusion for closed scenes.
fgAOContrast Can be used to adjust the contrast for ambient occlusion.
fgAOScale A scaling of the occlusion values. Can be used to increase or decrease the shadowing effect.
fgAOVisualize Visualize just the ambient occlusion values. Useful when tweaking the occlusion sampling options.

Path Tracer

```
<GISettings>
    <ptAccuracy>1</ptAccuracy>
    <ptCacheDirectLight>true</ptCacheDirectLight>
    <ptCheckVisibility>false</ptCheckVisibility>
    <ptConservativeEnergyLimit>0.95</ptConservativeEnergyLimit>
    <ptDefaultColor>
        <r>0</r>
        <g>0</g>
        <b>0</b>
    </ptDefaultColor>
    <ptDepth>5</ptDepth>
    <ptDiffuseIllum>true</ptDiffuseIllum>
    <ptFile></ptFile>
    <ptFilterSize>3</ptFilterSize>
    <ptFilterType>box</ptFilterType>
    <ptNormalThreshold>0.707</ptNormalThreshold>
    <ptPointSize>0</ptPointSize>
    <ptPrecalcIrradiance>true</ptPrecalcIrradiance>
    <ptPreview>true</ptPreview>
    <ptSpecularIllum>true</ptSpecularIllum>
    <ptTransmissiveIllum>true</ptTransmissiveIllum>
 </GISettings>
```

The Path Tracer algorithm calculates indirect lighting by tracing a (high) number of rays from the camera/light map into the scene. These rays are bounced and traced further into the scene, depending on the properties of the surfaces they hit. Surfaces with a high diffuse reflectance will have a higher probability to continue the ray paths, hence giving more color bleeding to other surfaces. The path is only terminated when the probability for absorption is high enough. This is called an unbiased solution, since all possible light paths are considered, and the paths are not truncated to a maximum depth. Illumination from the ray bounces are stored at selected points in the scene, resulting in a set of cache points. A parameter, ptPointSize, is used to set the distance between the cached points. A smaller distance will generate more cache points, and enable better capturing of details, but also higher memory usage and longer rendering time. Once the pre-render pass completes, the path tracer cache holds an approximation of the indirect light. The quality of the solution depends on the number of paths used and the density of the cache. If the path tracer solution is used as Secondary GI, with Final Gather as Primary GI, the settings can be set quite low and still produce good quality.
ptAccuracy Sets the number of paths that are traced for each sample element (pixel, texel or vertex). For preview renderings, you can use a low value like 0.5 or 0.1, which means
that half of the pixels or 1/10 of the pixels will generate a path. For production renderings you can use values above 1.0, if needed to get good quality.
ptPointSize Sets the maximum distance between the points in the path tracer cache. If set to 0 a value will be calculated automatically based on the size of the scene. The automatic
value will be printed out during rendering, which is a good starting value if the point
spacing needs to be adjusted.
ptPrecalcIrradiance If enabled the cache points will be pre-filtered before the final pass starts. This increases the performance using the final render pass.
ptNormalThreshold Sets the amount of normal deviation that is allowed during cache point filtering.
ptFilterType Selects the filter to use when querying the cache during rendering. None will return the closest cache point (unfiltered). The filter type can be set to None, Box, Gauss or Triangle.
ptFilterSize Sets the size of the filter as a multiplier of the Cache Point Spacing value. For example; a value of 3.0 will use a filter that is three times larges then the cache point spacing. If this value is below 1.0 there is no guarantee that any cache point is found. If no cache point is found the Default Color will be returned instead for that query.
ptPreview If enabled the pre-render pass will be visible in the render view.
ptCacheDirectLight When this is enabled the path tracer will also cache lighting from light sources. This increases performance since fewer direct light calculations are needed. It gives an approximate result, and hence can affect the quality of the lighting. For instance indirect light bounces from specular highlights might be lost.
ptCheckVisibility Turn this on to reduce light leakage through walls. When points are collected to interpolate between, some of them can be located on the other side of geometry. As a result light will bleed through the geometry. So to prevent this Beast can reject points that are not visible.

Monte Carlo

```
<GISettings>
    <mcDepth>2</mcDepth>
    <mcMaxRayLength>0</mcMaxRayLength>
    <mcRays>16</mcRays>
</GISettings>
```

This is a brute force GI method that calculates GI by sending rays in a hemisphere around the shading points. No caching is used so a new calculation is done for each shading point. An importance sampling method is used for secondary bounces, so a high number of bounces can be used without exploding the render time. This Monte Carlo method is unbiased and can be used for reference images when used as primary GI. It can also be used as secondary GI together with final gather to produce a multi bounce high quality result without light leakage.

mcRays Sets the number of rays to use for each calculation. A higher number gives higher quality, but longer rendering time.
mcDepth Sets the number of indirect light bounces calculated by monte carlo.
mcMaxRayLength The max distance a ray can be traced before it's considered to be a "miss". This can improve performance in very large scenes. If the value is set to 0.0 the entire scene will be used.

EnvironmentSettings

Image Based Lighting (IBL)

```
<EnvironmentSettings>
    <iblBandingVsNoise>1</iblBandingVsNoise>
    <iblEmitDiffuse>true</iblEmitDiffuse>
    <iblEmitLight>false</iblEmitLight>
    <iblEmitSpecular>false</iblEmitSpecular>
    <iblGIEnvBlur>0.05</iblGIEnvBlur>
    <iblImageFile>-</iblImageFile>
    <iblIntensity>1</iblIntensity>
    <iblSamples>300</iblSamples>
    <iblShadows>true</iblShadows>
    <iblSpecularBoost>1</iblSpecularBoost>
    <iblSwapYZ>false</iblSwapYZ>
    <iblTurnDome>0</iblTurnDome>
</EnvironmentSettings>
```

Image Based Lighting is both an algorithm to light a scene with a LDR/HDR image, and a simple environment for images. The IBL is by default set to act as a simple environment.
iblImageFile The image file to be used. You can use most image types, but you should of course use an HDR image if you can, like EXR or HDR. The file should be in Lat-Long format. If you need to convert images, HDRShop is a good alternative. If a LDR image is used, gamma correction will be applied.
iblTurnDome The sphere that the image is projected on can be rotated around the up axis. The amount of rotation is given in degrees. Default value is 0.0.
iblSwapYZ Swap the Up Axis. Default value is false, meaning that Y is up.
iblGIEnvBlur Pre-blur the environment image for Global Illumination calculations. Can help to reduce noise and flicker in images rendered with Final Gather. May increase render time as it is blurred at render time. It is always cheaper to pre-blur the image itself in an external application before loading it into Beast.
iblEmitLight Turns on the expensive IBL implementation. This will generate a number of (iblSamples) directional lights from the image.
iblEmitDiffuse To remove diffuse lighting from IBL, set this to false. To get the diffuse lighting Final Gather could be used instead.
iblEmitSpecular To remove specular highlights from IBL, set this to false.
iblSamples The number of samples to be taken from the image. This will affect how soft the shadows will be, as well as the general lighting. The higher number of samples, the better the shadows and lighting.
iblShadows Controls whether shadows should be created from IBL when this is used.
iblIntensity Sets the intensity of the lighting.
iblSpecularBoost Further tweak the intensity by boosting the specular component.
iblBandingVsNoise Controls the appearance of the shadows, banded shadows look more aliased, but noisy shadows flicker more in animations.

ADSSettings

```
<ADSSettings>
    <gridDensity>1</gridDensity>
    <gridMaxDepth>4</gridMaxDepth>
    <gridThreshold>25</gridThreshold>
    <instancingThreshold>500</instancingThreshold>
    <useInstancing>autodetect</useInstancing>
    <useSSE>true</useSSE>
</ADSSettings>
```

gridMaxDepth Decides how deep the Acceleration Data Structure can subdivide.
gridThreshold Decides how many triangles that can reside in a leaf before it is split up. The Recursion Depth has precedence over the threshold. A leaf at max depth will never be split. The Recursion Depth and Recursion Threshold are advanced settings that shouldn't be altered unless Acceleration Data Structures are second nature to you.
useSSE Beast uses specialized vector instructions to speed up the rendering. The cost is higher memory usage. Turn off SSE if Beast starts swapping.
useInstancing This tag lets the user control if instances should be used or not during rendering. Instancing allows the user to place a single shape in several different places, each with an individual transform. This preserves disk space and will lower memory consumption when rendering, but might increase render times quite a bit if there are many large instances.
The value of useInstancing can be set to

- **Never** Never use instancing while rendering.
- **AutoDetect** Use instancing whenever a shape is used more than once (default).

instancingThreshold Specifies the minimum number of triangles that an instance is allowed to have in order for geometry instancing to be used.

Lua

Beast uses Lua to define advanced bakings.

vec3

A vector of three float values. It can be constructed with the function vec3(x,y,z).

- **+**: This is aliased to add3.
- **-** (binary): This is aliased to sub3.
- **-** (unary): This is aliased to neg3.
- *****: This is aliased to mul3.
- **/**: This is aliased to div3.
- **[i]**: Returns the i:th component of a vec3.

Functions for retrieving implicit data

vec3 getPoint(): Returns the world-space intersection point of the fragment.
vec3 getNormal(): Returns the normal of the fragment.
vec3 getTangentU(): Returns the tangent of the fragment.
vec3 getTangentV(): Returns the bitangent of the fragment.
vec3 getViewDir(): Returns the vector from the fragment to the eye in world space.
vec3 getUV(): Returns the texture coordinates of the fragment. Uses the first two components of vec3.

vec3 getdTdX(): Returns the projection of the x axis of the screen space pixel in texture space. Uses the first two components of vec3.
vec3 getdTdY(): Returns the projection of the y axis of the screen space pixel in texture space. Uses the first two components of vec3.
float getT(): Returns the distance from ray origin to current fragment.

Functions for lighting

The following functions gives you information about all light sources that are light-linked to the point that is being baked/rendered. The function getLights() returns the number of light sources, and the other functions let you query the light sources for information by using an index.

int getLights(): Returns the number of lights.
bool getLightName(i): Returns the name of the light.
vec3 getLightDir(i): Returns the direction to the light from the point in world space.
float getLightDist(i): Returns the distance from the light to the point in world space.
vec3 getLightCol(i): Returns the color of the light. Includes attenuation and shadowing.
vec3 getLightColUnshadowed(i): Returns the color of the i:th light, not taking shadows into account.
bool getLightAmb(i): Returns true if the light is ambient.
bool getLightInd(i): Returns true if the light is indirect.
bool getLightCastShadows(i): Returns true if the light cast shadows.

Functions for ray tracing

vec3 shootRay(vec3 o, vec3 d): Shoots a ray with origin=o and direction=d with shading. Returns the color of the intersection.
float shootOccRay(vec3 o, vec3 d, [float tmax]): Shoots a ray with origin=o and direction=d without shading. Returns the distance to the closest intersection or -1.0f if no intersection is found. The maximum distance to intersection can be specified in tmax if desired. The occlusion ray ignores the transparency of the occluder.

Utility functions

float length(vec3): Returns the length of the vector.
float smoothstep(min, max, x): Returns the smoothstep function.
float step(min, max, x): Returns the step function.
float dot3(vec3 x, vec3 y): Returns the scalar product x.y.
vec3 cross3(vec3 x, vec3 y): Returns the cross product of x and y.
float add3(vec3 x, y): Returns x+y. y could be float or vec3.
float sub3(vec3 x, y): Returns x-y. y could be float or vec3.
float neg3(vec3 x): Returns -x. x is vec3.
float mul3(vec3 x, y): Returns x*y. y could be float or vec3. If y is vec3 the multiplication is element-wise.
float div3(vec3 x, float y): Returns x/y.
vec3 reflect(vec3 v, vec3 normal): Returns v reflected around the normal.
vec3 refract(vec3 v, vec3 normal, float inIor, float outIor): Returns v refracted around the normal. inIor is index of refraction for the material the ray is leaving, outIor the index of refraction for the material the ray is entering.
vec3 normalize(vec3 v): Returns v normalized.
int getSampleCount(): Return value: Returns the number of sample cells used in the distribution. Note that the number of sample cells not necessarily equals the minimum number of samples set in the setup function.
vec3 getSampleValue(int): Argument 1: index of sample cell to query for value. Return value: value (color) as vec3.

vec3 getSampleDirection(int, boolean): Argument 1: index of sample cell to query for direction. Argument 2: transform direction vector, false: gather space, true: world space. Return value: direction as vec3.
vec3 getSampleMean(): Return value: mean value of all sample cells as vec3 (not weighed by sample cell area)
vec3 getSampleProjectedSum(vec3, boolean): Argument 1: projection vector. Argument 2: include values with negative weight. Return value: sum of all sample cell values as vec3, weighed using the scalar product between sample cell direction and given vector.
float getSampleOcclusion(): Return value: Returns a value [0.0, 1.0] representing the occlusion over all sample cells. 0.0 represents no occlusion and 1.0 represents full occlusion.
vec3 getGatherSpaceVector(int): Argument 1: [1,3] index of gather space vector. Return value: a gather space vector in world space
array getBasisCoefficients(int): Argument 1: [0,3] generate basis coefficients for chosen basis from following data:

- 0: color intensity
- 1-3: corresponding color channel

Return value: array (table) of coefficients
array evalSHBasis(vec3, int): Argument 1: direction to evaluate SH function for. Argument 2: [1,10] number of SH bands to use. Return value: array (table) of SH basis values, size of array is (number of bands)^2
vec3 gatherToWorld(vec3): Argument 1: vector to transform. Return value: transformed vector.
vec3 worldToGather(vec3): Argument 1: vector to transform. Return value: transformed vector.

Programmable Baking

Beast has a LUA pass, giving you the possibility to script your own baking. This separate pass exists for both texture bake and surface transfer, as well as for the point cloud bake functionality.

The LUA bake script

The LUA bake script should have the following structure:

```
function setup()
-- settings for baking
end
function basis(x, y, z)
-- define function basis to fit data to
-- optional
end
function bake()
-- compute the data which is output to texture or point cloud
-- optional
end
```

Setup function

This function is executed once before baking and is used to set the different baking settings.

Available functions:
bset(string setting, (-) value) Give a setting the specified value.

gather.useRadianceCache(bool)

Uses the radiance cache instead of brute force sampling. All gather functionality is controlled via the final gather settings. **basis.type** should be set to ”sh”.

gather.sampletype(string)

- “none”, “n”
- “indirect illumination”, ”ii”
- “dynamic illumination”, “dyni”
- “direct dynamic illumination”, “ddyni”
- “indirect dynamic illumination”, “idyni”

Default value is “direct dynamic illumination”. This settings determines the type of light to gather. If set to none, the gather before each execution of the basis function is disabled. This is useful when e.g. only baking direct static light, and no gather is needed.
“**none**”, Does no gather at all.
“**indirect illumination**”, Gathers indirect illumination only. If you want direct illumination, you’ll need to manually add it in the bake function.
“**dynamic illumination**”, Gathers outgoing lighting depending on incoming light direction, also known as precomputed radiance transfer.
“**direct dynamic illumination**”, As dynamic illumination, but not considering indirect light.
“**indirect dynamic illumination**”, As dynamic illumination, but only considering indirect light.

gather.space(string)

– “world space”, “ws”
– “object space”, “os”
– “tangent space”, “ts”
Default value is “tangent space”. Used to specify the space in which to perform the gather.

gather.distribution(string)

- “cosine weighted”, “cw”
- “equiareal”, “ea”

Specify if the gather distribution should be cosine-weighted around the surface normal, or if the gather space should be uniformly sampled, "equiareal". Default value is “equiareal”.

gather.minsamples(int) [>1]

Default value is 128. Specifies minimum number of samples used for gather. This number is used to determine the number of sample cells to use in the gather distribution. Note that the actual number of sample cells used not necessarily equals this
number since the sample distribution needs to distribute the samples over the (hemi)sphere in an efficient way.

gather.maxsamples(int i) [>1]

Default value is 256. Specifies maximum number of samples used for gather, is used to set a upper limit for super sampling in the gather distribution. Cannot be lower than the gather.minsamples value.

gather.coneangle(int) [0, 360]

Default value is 180. Cone angle to use for gather distribution. Should be set to 180 or 360 in most cases.

gather.clamp(boolean)

Default value is false. Specify whether or not to clamp the values sampled during gather.

gather.cosweighdynamic(boolean)

Default value is false. Specify whether gathered dynamic light should be cosine weighed according to the sample normal.

gather.aomaxdistance(float)

Sets the maximum distance a ray can take before being considered unoccluded. This will affect the occlusion appearance. If not set the whole scene size will be used. For indoor scenes this must be set, otherwise all surfaces will be fully occluded.

gather.aoscale(float)

A scaling of the occlusion value for occlusion sampling. Can be used to boost the shadowing effect.

gather.aocontrast(float)

Controls the contrast in the occlusion. A higher value will increase the contrast, making dark areas more dark and bright areas more bright.

gather.aoinfluence(float)

A value [0,0, 1.0] that controls if the output values from LUA should be scaled by ambient occlusion automatically. The value is a blend between 0.0 => no occlusion scaling and 1.0 => full occlusion scaling.

output.size(int) [>1]

Default value is 4. Specify the size of the array returned by the bake function.

basis.type(string)

- “none”, “n”
- “ptm”
- “sh”
- “custom”, “c”

Default value is “none”. Determines the type of function basis to fit gathered data to. If set to other than none, the bake function can be omitted from the script, and the coefficients for the specified function basis will be generated automatically. If set to custom, the basis function must be defined.

basis.rgb(boolean)

Default value is false. This setting should be taken into consideration when baking function coefficients. If the bake function returns one set of coefficients for each color channel, this setting should be set to true, and the bake function should return the coefficients for all three channels as one concatenated array. If generating coefficients for the intensity of gathered data, this settings should be set to false.

basis.sh.bands(int) [1, 10]

Default value is 2. Specify the number of bands to use if generating SH coefficients.

Basis function

This function defines the values of the function basis you want to fit the gathered data to. It only needs to be defined if output has been set to custom basis (basis.type is set to custom). An array with the values must be returned. The function is called once for every direction used in the gather before baking. The following example shows how a PTM would be defined:

```
function basis(x, y, z)
  return {x*x, y*y, x*y, x, y, 1}
end
```

Bake function

The bake function is executed once for every pixel (or super sample if super sampling is enabled). It must return an array with the data that should be output to the texture or point cloud. It is also important that the size of the returned array is the same as specified in the setup function (output.size).

Using the Radiance Cache with Lua

Baking with Lua can become quite slow since the gather is executed once per sample. You can work around this to some extent by using the radiance cache. To enable this you need to set gather.useRadianceCache to true and basis.type to ”sh” in the setup function. GI with Final Gather with Radicance SH needs to be enabled as well.

- There are some drawbacks with using the cache:
- The cache only contains 3 band SH. If you are trying to bake higher orders, the cache will act as a low pass filter.
- getSampleCount will always return 1 when using the cache.
- getSampleValue cannot be used with the cache.
- getSampleDirection cannot be used with the cache.
- getSampleMean cannot be used with the cache.

Examples

Occlusion

This shows a very simple example baking red for surfaces in occlusion relative to the first light source and green otherwise.

```
function bake()
    -- Check for occluders between here and light source number one.
    occ = shootOccRay(getPoint(), getLightDir(1))
    if (occ > 0) then
        return {1,0,0,1}  -- Red
    else
        return {0,1,0,1}  -- Green
    end
end
```

RNM

Typical setup to bake indirect illumination to Radiosity Normal Maps. Please note that there are a couple of different standard definitions around for the three RNM basis vectors.

```
function setup()
    -- Sampling hemispherically in tangent space.
    bset("gather.sampletype", "indirect illumination")
    bset("gather.space", "tangent space")
    bset("gather.distribution", "equiareal")
    bset("gather.minsamples", 300)
    bset("gather.maxsamples", 600)
    bset("gather.coneangle", 180)

    bset("output.size", 9)
end

function bake()
    -- Basis definition.
    basis1 = vec3(0.8165, 0.0, 0.577)
    basis2 = vec3(-0.408, 0.707, 0.577)
    basis3 = vec3(-0.408, -0.707, 0.577)

    rgb1 = getSampleProjectedSum(basis1, false)
    rgb2 = getSampleProjectedSum(basis2, false)
    rgb3 = getSampleProjectedSum(basis3, false)

    n = getSampleCount()
    rgb1 = (rgb1 * 2.0) / n
    rgb2 = (rgb2 * 2.0) / n
    rgb3 = (rgb3 * 2.0) / n

    result = {rgb1[1], rgb1[2], rgb1[3],
              rgb2[1], rgb2[2], rgb2[3],
              rgb3[1], rgb3[2], rgb3[3]}

    return result
end
```

SH

Typical setup to bake indirect illumination to Spherical Harmonics. We use three SH, one for each color channel. Each SH has two bands. This makes a total of 12 components. The bake function here is actually implicit. Unless you want to rearrange the components or perform other operations then removing the bake function completely will render the same result.

```
shbands = 2;
shbasiscount = shbands*shbands;

function setup()
    -- Sampling spherically in world space.
    bset("gather.sampletype", "indirect illumination")
    bset("gather.space", "world space")
    bset("gather.distribution", "equiareal")
    bset("gather.minsamples", 700)
    bset("gather.maxsamples", 1000)
    bset("gather.coneangle", 360)

    bset("basis.type", "sh")
    bset("basis.rgb", true)
    bset("basis.sh.bands", shbands)

    bset("output.size", 3 * shbasiscount)
end

function bake()
    red = getBasisCoefficients(1)
    grn = getBasisCoefficients(2)
    blu = getBasisCoefficients(3)

    result = {}
    for i = 1, shbasiscount do
        result[i] = red[i]
    end
    for i = 1, shbasiscount do
        result[i + shbasiscount] = grn[i]
    end
    for i = 1, shbasiscount do
        result[i + 2*shbasiscount] = blu[i]
    end

    return result
end
```

Dominant light direction

Here we demonstrate a Dominant Light Direction pass. Unlike the fixed RNM basis vectors we project lighting onto the normal direction and direction of the most incoming lighting. The dominant light direction is derived from the second SH band. Please note that we're working in tangent space only taking the upper hemisphere into account.

```
function setup()
    -- Sampling hemispherically in tangent space.
    bset("gather.sampletype", "indirect illumination")
    bset("gather.space", "tangent space")
    bset("gather.distribution", "equiareal")
    bset("gather.minsamples", 300)
    bset("gather.maxsamples", 600)
    bset("gather.coneangle", 180)

    bset("basis.type", "sh")
    bset("basis.rgb", true)
    bset("basis.sh.bands", 4)
    bset("output.size", 8)
end

function bake()
    -- Tangent space directions to get illumination from.
    normal = vec3(0.0, 0.0, 1.0)
    lightDir = dominantLight()

    -- Project indirect light onto those directions.
    if lightDir == vec3(0.0, 0.0, 0.0) then
        rgb1 = {0.0, 0.0, 0.0} -- No dominant light direction, no light.
    else
        rgb1 = getSampleProjectedSum(lightDir, false)
    end

    rgb2 = getSampleProjectedSum(normal, false)

    n = getSampleCount()
    rgb1 = (rgb1 * 2.0) / n
    rgb2 = (rgb2 * 2.0) / n

    -- Output indirect lighting projected on normal and dominant light
direction.
    -- Dominant light direction stored in LDR alpha channel.
    result = {rgb1[1], rgb1[2], rgb1[3], lightDir[1]/2+0.5,
              rgb2[1], rgb2[2], rgb2[3], lightDir[2]/2+0.5}

    return result
end

function dominantLight()
    coefficients = getBasisCoefficients(0)

    -- Get dominant light direction from second band sh coefficients.
    dir = vec3(coefficients[4], coefficients[2], math.max(0.0,
coefficients[3]))

    if length(dir) > 0.000001 then
        return normalize(dir)
    else
        -- Can't find a proper direction, at least not in this hemisphere.
        return vec3(0.0, 0.0, 0.0)
    end
end
```

Direct lighting

Sometimes you want to include direct lighting from certain light sources. We can accomplish this by adding a light loop in the bake function projecting the direct light contribution onto our basis. This example shows how it can be done with RNM but the same principle applies to other basis types like SH for example.

```
function setup()
    -- Sampling hemispherically in tangent space.
    bset("gather.sampletype", "indirect illumination")
    bset("gather.space", "tangent space")
    bset("gather.distribution", "equiareal")
    bset("gather.minsamples", 300)
    bset("gather.maxsamples", 600)
    bset("gather.coneangle", 180)

    bset("output.size", 9)
end

function bake()
    -- Basis definition.
    basis1 = vec3(0.8165, 0.0, 0.577)
    basis2 = vec3(-0.408, 0.707, 0.577)
    basis3 = vec3(-0.408, -0.707, 0.577)

    rgb1 = getSampleProjectedSum(basis1, false)
    rgb2 = getSampleProjectedSum(basis2, false)
    rgb3 = getSampleProjectedSum(basis3, false)

    n = getSampleCount()
    rgb1 = (rgb1 * 2.0) / n
    rgb2 = (rgb2 * 2.0) / n
    rgb3 = (rgb3 * 2.0) / n

    -- Direct light loop.
    nlights = getLights()
    for i = 1, nlights do
        if (not getLightAmb(i)) then
            -- Transform light direction to tangent space.
            -- Dot with basis vectors.
            col = getLightCol(i)
            localLightDir = normalize(worldToGather(getLightDir(i)))
            weight = dot3(basis1, localLightDir)
            if (weight > 0) then
                rgb1 = rgb1 + col * weight
            end
            weight = dot3(basis2, localLightDir)
            if (weight > 0) then
                rgb2 = rgb2 + col * weight
            end
            weight = dot3(basis3, localLightDir)
            if (weight > 0) then
                rgb3 = rgb3 + col * weight
            end
        end
    end

    result = {rgb1[1], rgb1[2], rgb1[3],
              rgb2[1], rgb2[2], rgb2[3],
              rgb3[1], rgb3[2], rgb3[3]}

    return result
end
```

External Links

LUA:
http://www.lua.org/

RNM:
http://www.valvesoftware.com/publications/2004/GDC2004_Half-Life2_Shading.pdf

SH:
http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.html

SH tricks, mentions to formula for extracing dominant direction from SH:
http://www.ppsloan.org/publications/GDC08SH_web.pptx