A SSAO Sample for the Ogre3D Sample Framework

Simon Wallner
`me@simonwallner.at`

November 30, 2010

In this document, I present the results of my *ssao sample project.* The sample has been implemented with the Ogre3D graphics engine and uses *NVIDIA's cg* as the shading language. Six techniques have been implemented, and the resulting images are compared to groud truth renderings created with *mental ray* and *Maya2011.*

Developer documentation can be found in the appendix.

1. Introduction

This document accompanies a sample for the *Ogre3D sample framework*[1], implementing a few *screen space ambient occlusion* techniques. The sample has been created with the mentioned framework, and uses *NVIDIA's cg language* for the shader code.

2. Download and Installation

The implementation is a fork of the main Ogre3D repository (version 1.7), publicly available via *bitbucket*[2]. Bitbucket uses *mercurial*[3] as a source control management tool, but downloads of complete source packages are provided as well.

It can be built with the instructions found at [4] and uses *CMake* as a cross platform build tool. Due to the large size of the project, building can take a while.

After building the project, the sample browser can be run from the output folder, and should list the *SSAO Sample* to start right away. Due to time and hardware limitations, the sample has only been tested on the development machine[5] using OpenGL.

[1]`http://www.ogre3d.org/tikiwiki/SoC2009+Samples`
[2]`http://bitbucket.org/simonwallner/ogre-ssao-sample`
[3]`http://mercurial.selenic.com/`
[4]`http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Building+Ogre`
[5]Windows7 32bit, ATI Radeon 4830

2.1. Source Structure

The source code of the sample can be found in the `/Samples/SSAO` folder. Media (scripts, meshes and textures) can be found at `/Samples/Media/SSAOMedia/`. In a second repository[6], a matlab script to generate textures, compare reference and result renderings and this document can be found.

3. Implementation Details

A few techniques have been selected for implementation. The focus in the implementation was a working, easy to read implementation. Speed and optimization was not a concern. There should be a great potential in the present implementations to improve speed and tuning the parameters for optimal quality.

A *G-buffer* is used to hold the required information (scene depth, world space position, fragment normals) for the shading processes. All meshes are rendered as solid white, but it should be possible to integrate the code in fully featured environments with relative ease.

Computation of the occlusion and filtering has been split, so that any method can be used with any filter.

Most techniques are described in my accompanying bachelor's thesis. These will not be described here in detail, and only implementational details will be given.

3.1. Unsharp Masking the Depth Buffer

Unsharp masking the depth buffer [Luft et al., 2006] uses unsharp masking on the depth buffer to derive the spatial importance function. A multi pass approach was chosen with two passes for the separated gauss filter (width = 19) and a final pass to compute the visual importance and *depth darkening*. Only depth information is needed.

3.2. Crease Shading

Crease shading [Fox and Compton, 2007] uses surface normals to determine the degree to which two surfaces face each other. Many parameter are available to tune the result.

A regular, stippled, diamond shaped sampling kernel is used, and samples are picked in screen space.

3.3. Crytek's SSAO

Crytek's SSAO technique [Kajalin, 2009] uses only the depth buffer and distributes samples in a sphere around the fragment's position. The volume of this sphere is approximated and used as the occlusion value. A regular sampling kernel is used that is randomly rotated for each fragment. Kernel size can either be set in screen or in world space.

[6]`http://bitbucket.org/simonwallner/ssao_extras`

3.4. Hemispher MC

Based on ideas found in Crytek's implementation and [Ritschel et al., 2009], *HemisphereMC* uses importance sampling to approximate the volume of the normal aligned hemisphere. Cosine distributed samples over the hemisphere are used and the sample distance is a linear function of the sample number:

$$d(s_i) = \frac{(i+1) * r}{n} \quad \text{for } i = 0..n-1 \tag{1}$$

where $d(s_i)$ is the distance of the i-th sample, r is the radius of influence in world space, and n is the number of samples. Due to the limited time, sample distribution for this approach is not ideal. The samples have been generated over the whole sphere, which has then been compressed to the upper hemisphere, resulting in uneven sampling near the equator. Elevation angles have been inverted to achieve the cosine distribution. See the matlab script for additional information and visualization.

Interleaved sampling is used to reduce variance and computation happens in a single pass.

3.5. Horizon Based AO

Horizon based AO [Bavoil and Sainz, 2009] uses ray marching to determine the horizon angle of a fragment. The implementation follows the paper closely.

3.6. Volumetric AO

In a recent paper [Szirmay-Kalos et al., 2010] present a new approach to SSAO. Using importance sampling and a specific distance attenuation function, the integral can be reformulated to a volumetric integral over a tangent sphere with radius $= r/2$, where r is the radius of influence. The volume of this sphere is then approximated by computing the length of path segments through this sphere. Samples are distributed on a disk centered at the center of the tangent sphere. The paper uses *poisson-disk* sampling which I could not reproduce, again, due to time constraints. I use a combination of uniform sampling over the unit disc of many samples with a subsequent reduction by k-means clustering to n clusters. This, simple to implement approach, gives nice results that look similar to poisson-disk sampling. Figure 1 shows the result for 10240 uniform samples reduced to 256 samples.

4. Filter

Three post processing blur filters are provided to improve the image quality. The simplest is the naive 4×4 box filter. A little more advanced is the depth aware *smart box filter*. It is also a 4×4 box filter, but weights are chosen according to the depth values in order to prevent blurring over discontinuities and to preserve sharp edges. The third filter is a

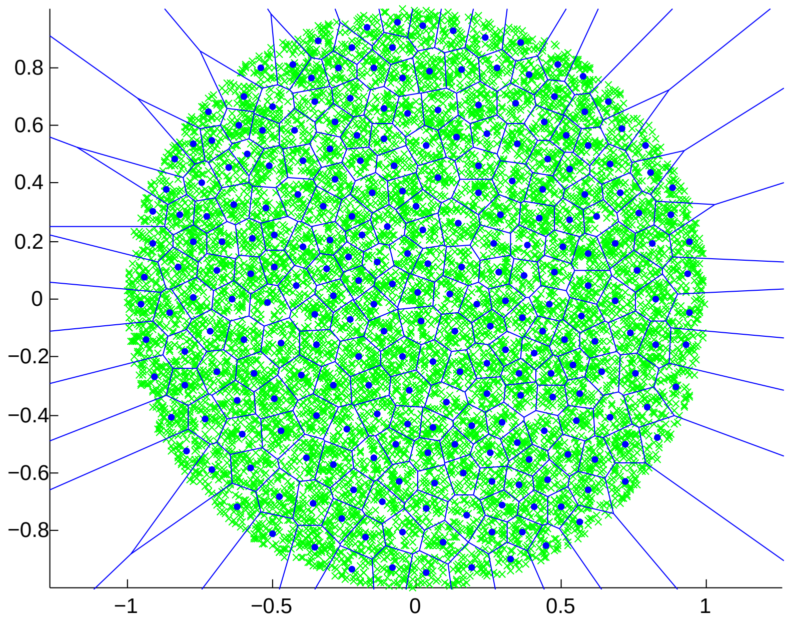


Figure 1: An easy to implement way to achieve a poisson-disk like sample distribution. Uniformly distributed samples are generated at first and then reduced by k-means clustering to the desired number of samples. Uniform samples are shown in green, final samples in black with the induced Voronoi regions.

cross bilateral filter [Tomasi and Manduchi, 1998] with kernel width = 13. The following photometric weighting function is used:

$$w_p = \frac{1}{(1+\Delta d)^{\alpha}} \tag{2}$$

where Δd is the depth difference and α is a user defined parameter.

5. Results and Discussion

As mentioned above, performance was not a concern in the implementation and therefore no individual performance figures are given. Overall the performance was around 100 fps at a resolution of 800×600 on the development machine.

Difference images are also not given, due to the lacking comparison metric, and uncertainties regarding color management, which could drastically bias the result. Gamma correction has been neglected in the implementation and color management was also not enforced in the ground truth rendering. With all these issues, the best way to assess the quality is to directly compare the results side by side. High-res (1600×1200) images along a rudimentary comparison script for matlab can be found in the extras repository for further comparison.

Figures 3 and 4 show the resulting renderings along a ground truth rendering, rendered with *Maya2011* and *mental ray*. Table 1 shows an overview of the techniques with infos about the sampling strategies.

5.1. Unsharp Masking

Unsharp masking with *depth darkening* simply adds a kind of drop shadow to the scene. Although this does not resemble ambient occlusion in most cases, it might do in certain cases like foliage, etc. False occlusion by distant occluders however is not desirable in most situations. Another problem is that the effect highly depends on the scene's depth complexity and must be hand tuned for each scene.

5.2. Crease Shading

Crease shading gives surprisingly good results. Occlusion is soft and smooth. Due to the regular sampling pattern, no noise is introduced and hence no post processing is required. The sampling kernel can be scaled to a certain degree without noticeable banding artefacts. It is rather simple to implement and provides great adjustability by the many parameters.

5.3. Crytek's ssao

Crytek's ssao is very smart in reducing the computational cost. (although it is mentioned that an even more optimized version is used in production). By using a randomly rotated, regular sampling kernel only a single random texture fetch is required per fragment. Sample distances are scaled exponentially, which emphasises nearby samples and sampling density becomes rather thin towards r. Iterative multiplication of the sample length might also be prone to numerical issues, if the starting value is very small and the number of iterations is high. A different, or at least not iterative scaling function is suggested here to avoid these problems. It must be noted that numeric problems are not proven here, merely seemed to pop up during development.

Crytek integrates over the full sphere to approximate the occlusion. Asides from *wasting* samples in the lower hemisphere this approach is inherently flawed if parts of the sphere are occluded by distant occluders. In this case, a default occlusion value is used for these samples that represents the neutral case, i.e. 0.5. This can lead to too much or too little occlusion. Figure 2 illustrates the problem.

5.4. Hemisphere MC

Using a surface aligned hemisphere solves crytek's problems. Additionally pseudo cosine distributed samples and a linear sample distance function are used. This approach is not physically based but gives more pleasant results and samples are a bit more distributed towards the perimeter of the sphere than in crytek's base method.

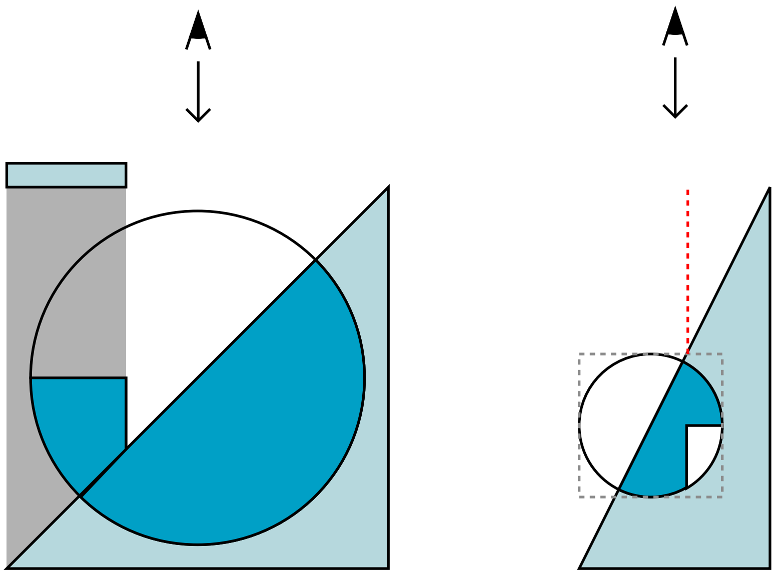


Figure 2: This illustration shows the two problems: *left:* Falling back to a default occlusion value of 0.5 in areas behind distant occluders leads to extraneous occlusion. The dark blue area signifies the resulting approximated volume. *right:* Under steep angles, parts of *neutral* surfaces, i.e. with a volume of 0.5, can be to open due to self occlusion of the integration sphere. Samples, that fall to the right of the red, stippled line are treated as distant occluders and again the default value of 0.5 is assumed. In this case The approximated volume is less than the actual volume.

5.5. Horizon Based

This approach is physically based and also gives nice results. Banding occurs in the cornell scene, due to the relatively low directional resolution of 4 in this picture. Sample count could not be risen higher, due to a hard constraint of the cg compiler, that limits the output to 1024 instructions.

5.6. Volumetric AO

Also physically based, volumetric AO gives sharper and more pronounced shadows than the horizon based approach.

6. Conclusion

Subjectively compared to the ground truth renderings, *HemisphereMC* and *Volumetric AO* give the *best* results, but all five tested ambient occlusion techniques have their strengths and weaknesses, which makes them viable AO solution in different situations.

Temporal incoherence, like flickering, was never an issue in any technique. Using interleaved sampling drastically improves the image quality and filtering with a matching box filter is very cheap. If independent random vectors are used for every fragment, heavy noise is introduced, that can be only mitigated by heavy post process filtering.

Performance-wise, no useful assumptions can be made. All techniques performed about the same, and the articles and the results hint that performance is bandwidth limited, as texture fetches for random values and samples seem to dominate the runtime. I am certain that there is a great potential for optimization in the current implementation.

Another important aspect would be to introduce gamma correction to the algorithms. Perceived realism might improve with proper color management.

All techniques use different sample distributions and sampling strategies. It would be worth to mathematically investigating the sample distributions and formally compare them over the techniques, complete with the impact different distributions have on different techniques. Having a unified sample distribution would also benefit a possible formal comparison with the ground truth image.

7. License

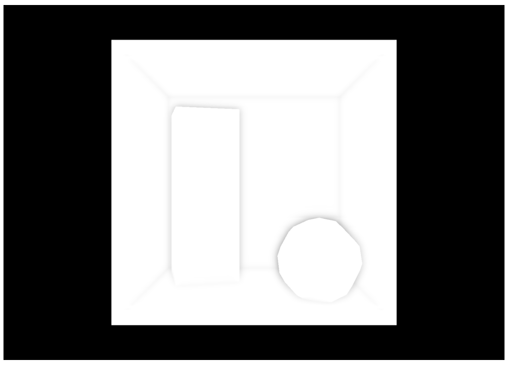
The Šibenik cathedral model was created by Marko Dabrovic[7]. The original Ogre3D project is available under the MIT license[8].

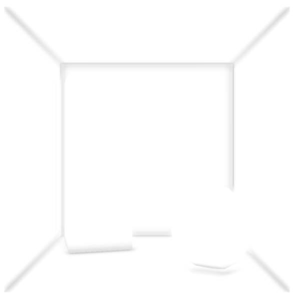
All source code, content and media that was created in the course of this project by myself (in the fork and the extras repository) is also available under the MIT license unless no other licensing claims apply. (by the original authors of the techniques, etc...).

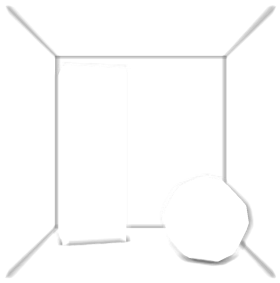
If the results of this project are of any use to you, it would be nice if you dropped me a few lines about it.

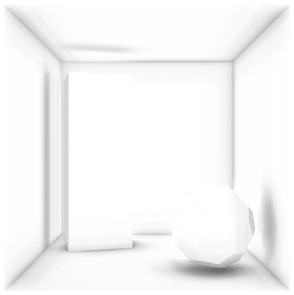
[7] `http://hdri.cgtechniques.com/~sibenik2/`

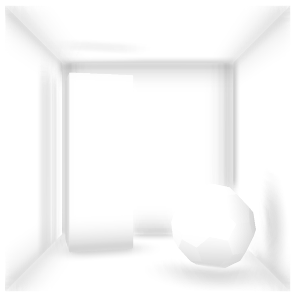
[8] `http://ogre.svn.sourceforge.net/viewvc/ogre/trunk/COPYING?revision=9087`

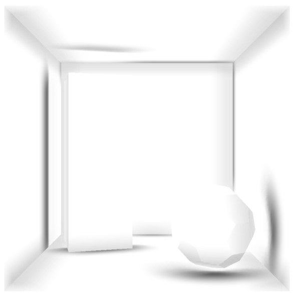

(a) unsharp masking


(b) crease shading


(c) crytek


(d) hemisphereMC


(e) horizon


(f) volumetric AO

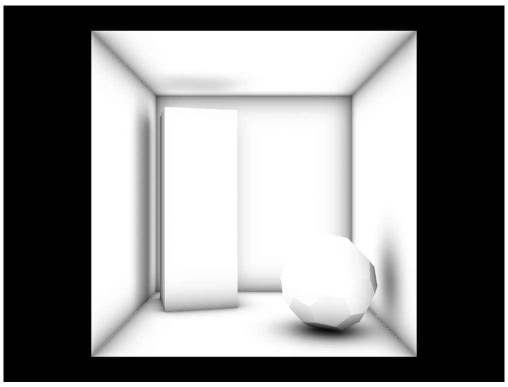
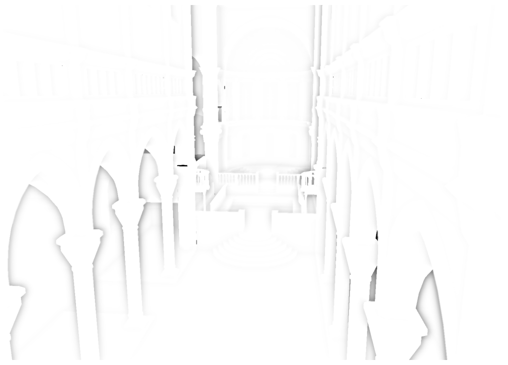
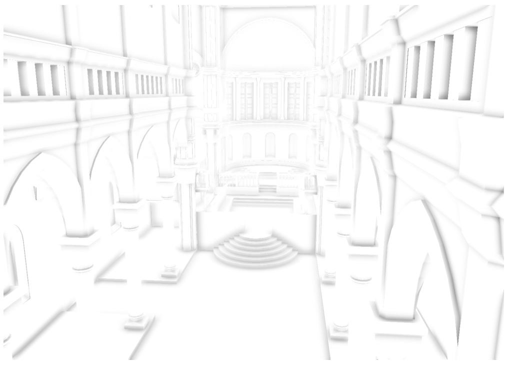
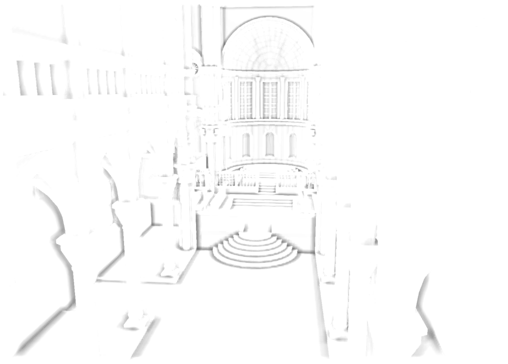
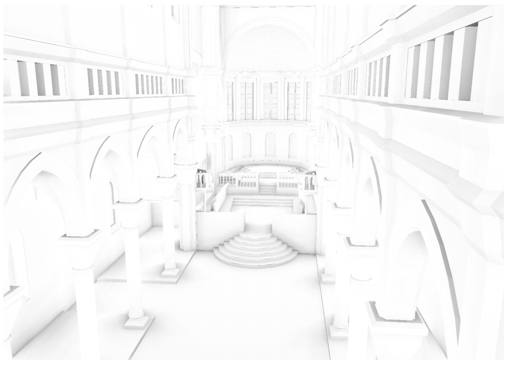

(g) ground truth

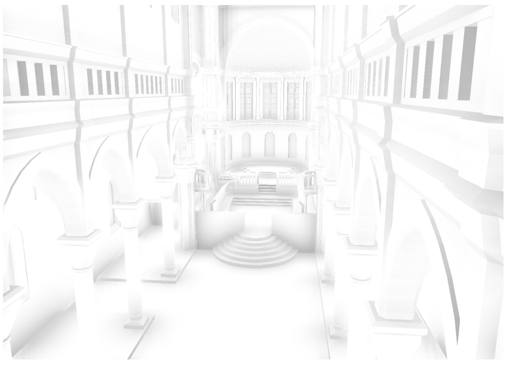
Figure 3: Renderings of the cornell box for all 6 techniques along the ground truth solution.

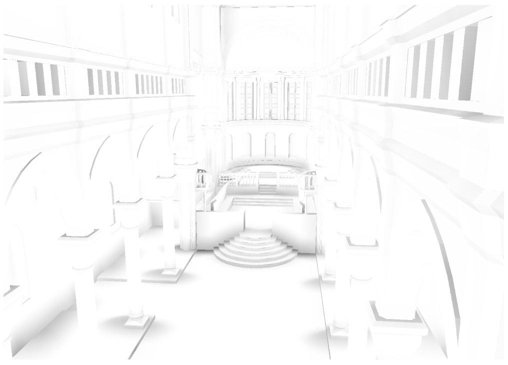

(a) unsharp masking


(b) crease shading


(c) crytek


(d) hemisphereMC


(e) horizon


(f) volumetric AO

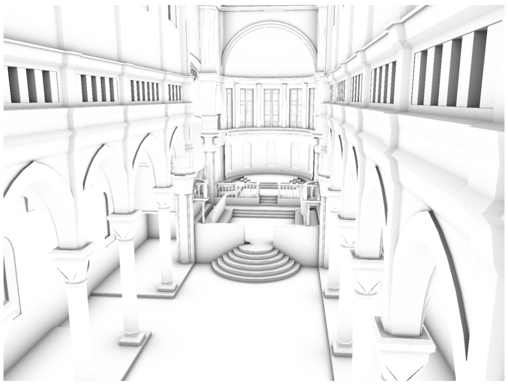

(g) ground truth

Figure 4: Renderings of the Šibenik cathedral for all 6 techniques along the ground truth solution.

technique	**random texture fetches**	**strategy**	**screen/world**	**interleaved**	**number of samples**
unsharp masking	–	separated gaussian blur			19×19
crease shading	–	regular diamond shaped pattern	+/-		24
crytek	1 per fragment	randomly rotated regular pattern	+/+	+	32
hemishpere MC	1 per sample	Pseudo importance sampling in the normal aligned hemisphere.	+/+	+	32
horizon based	1 per fragment	randomly rotated regular directions with distance jittered samples	+/+	+	28
volumetric AO	1 per sample	reformulated integral, sampling over unit disc of tangent sphere	+/+	+	32

Table 1: This table gives a brief overview over the implemented techniques. The last column gives the number of samples used for the renderings.

References

[Bavoil and Sainz, 2009] Bavoil, L. and Sainz, M. (2009). Image-space horizon-based ambient occlusion. *in ShaderX7, W. Engel, Ed. Charles River Media, March.*

[Fox and Compton, 2007] Fox, M. and Compton (2007). Ambient occlusive crease shading. *http://www.shalinor.com/research.html.*

[Kajalin, 2009] Kajalin, V. (2009). Screen space ambient occlusion. *in ShaderX7, W. Engel, Ed. Charles River Media, March.*

[Luft et al., 2006] Luft, T., Colditz, C., and Deussen, O. (2006). Image enhancement by unsharp masking the depth buffer. *SIGGRAPH '06: SIGGRAPH 2006 Papers.*

[Ritschel et al., 2009] Ritschel, T., Grosch, T., and Seidel, H.-P. (2009). Approximating dynamic global illumination in image space. *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games.*

[Szirmay-Kalos et al., 2010] Szirmay-Kalos, L., Umenhoffer, T., Toth, B., Szecsi, L., and Sbert, M. (2010). Volumetric ambient occlusion.

[Tomasi and Manduchi, 1998] Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. pages 839 –846.

A. Developer Documentation

This section gives a brief overview over the code.

A.1. Sample Plugin

This project has been created as a sample for the new Ogre sample browser. This browser can load and start samples at runtime, and also provides a nice set of gui functionalities. The ssao sample is compiled into a dll (on windows) which is then loaded by the browser.

A.2. File/Folder Structure and Media

The source code of the samples (`Sample_SSAO.h` and `Sample_SSAO.cpp`) can be found in the `./Samples/SSAO/` folder. These two files contain everything that is needed for the sample. Further documentation is included in the source.

Media files are located in `./Samples/Media/SSAOMedia/`. Scripts meshes and textures can be found in their respective folders.

Other important files are `resources.cfg` and `samples.cfg` and their debug versions (denoted by the `_d` postfix) found in the `./CMake/Templates` folder. At the configuration phase in the build cycle, these files are configured and copied into the right location of the build folder hierarchy.

The `resources` file contains all the resources that the resource manager should search/load at runtime. For this sample it contains three lines defining the locations of the scripts, meshes and textures.

The `samples` file defines which samples should be loaded when the sample browser is started.

A.3. Compositors and Scripts

Ogre uses its own formats to define materials and post processing *compositors*. For this sample, all SSAO compositors and all post filter compositors are each kept in a single file.

Materials and cg shaders mostly have their own files and are named like the techniques.

A.4. Getting started with the code

To get started with the code, It would be best to start at the `Sample_SSAO::setupContent()` function and work forward from it. This method is kind of the main entry point and is called at the startup of the sample. An instance of the sample itself is created at sturtup of the the sample browser.

A few *magic strings* and const strings are used for the compositor names and other gui related names and strings. Those can be found at the top of the `Sample_SSAP.cpp` file.