Sycles

Cycles integration into Softimage

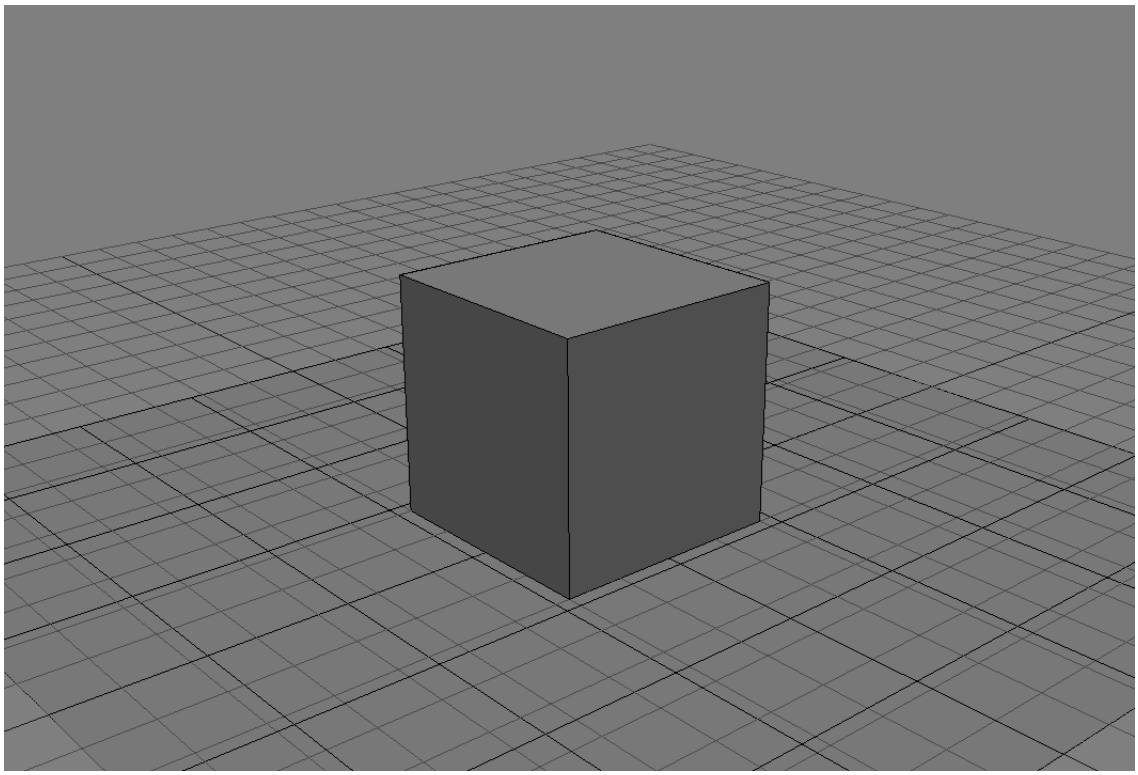
How to...

March 22, 2020

Contents

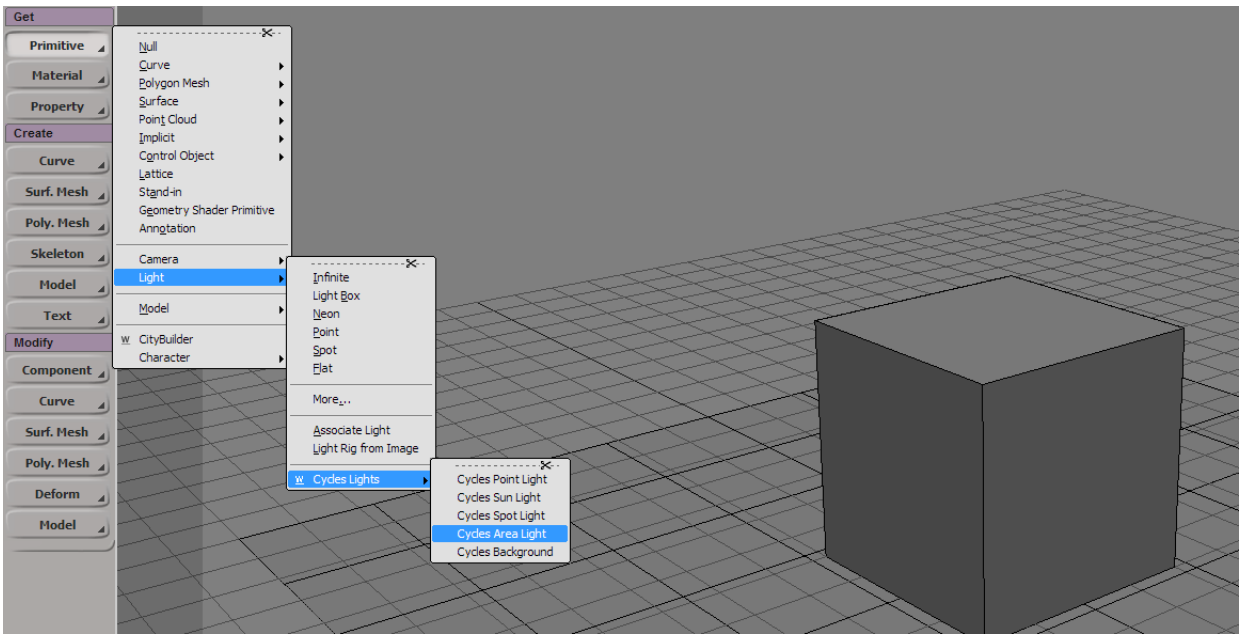
1 How to start rendering

Suppose we have a scene: a cube and a plane.

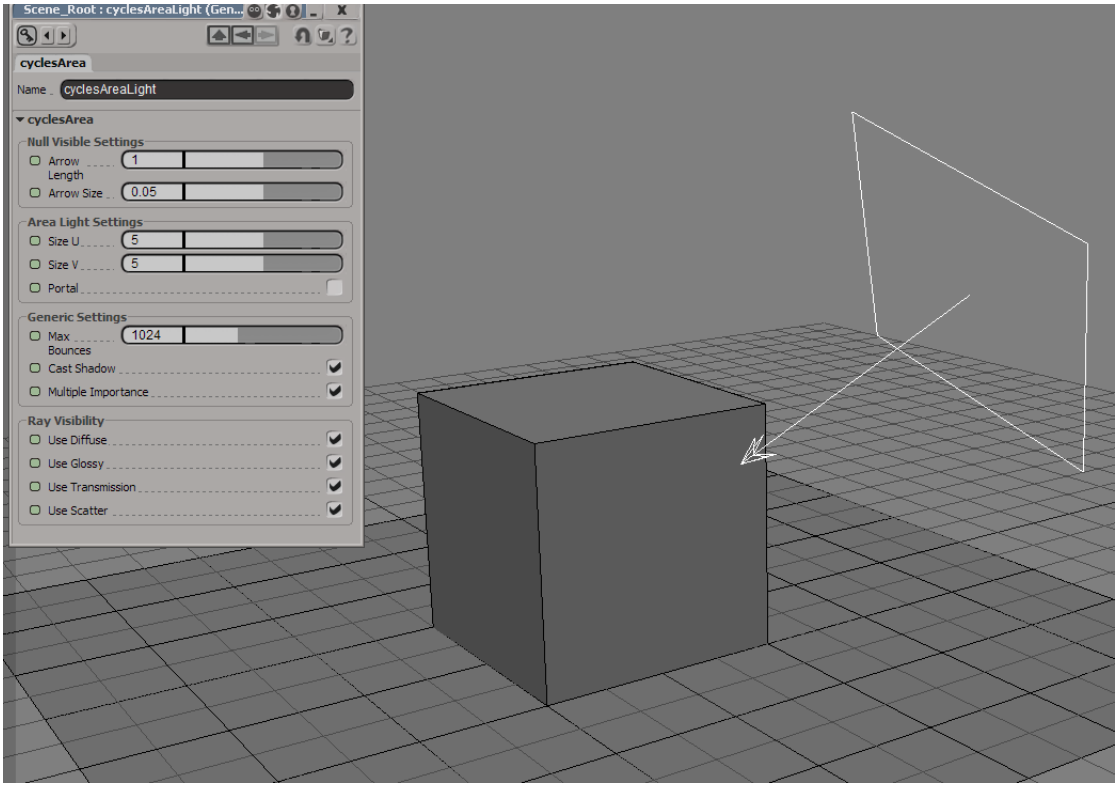


Add a rectangular light source, by choosing

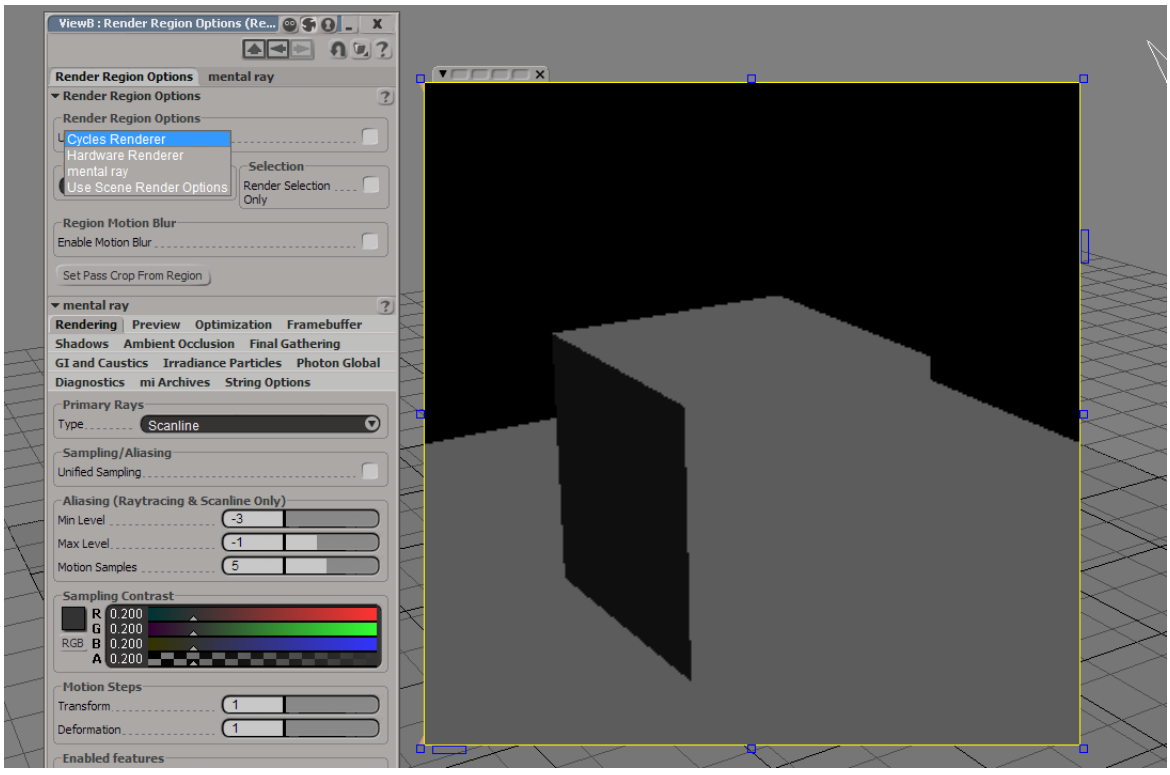
`Primitive` – `Light` – `Cycles Lights` – `Cycles Area Light`



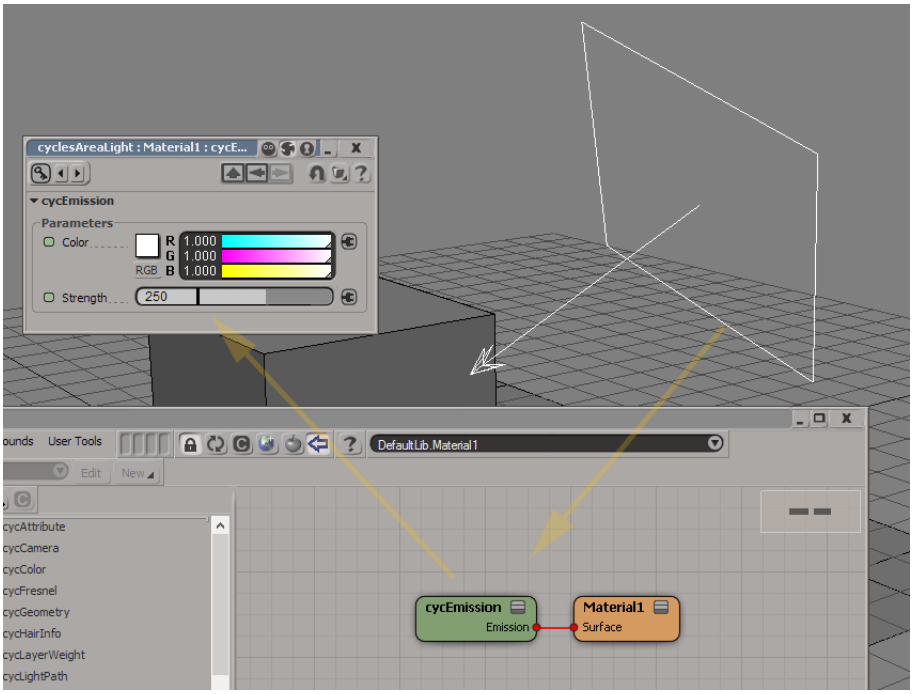
Set its size `5x5`.



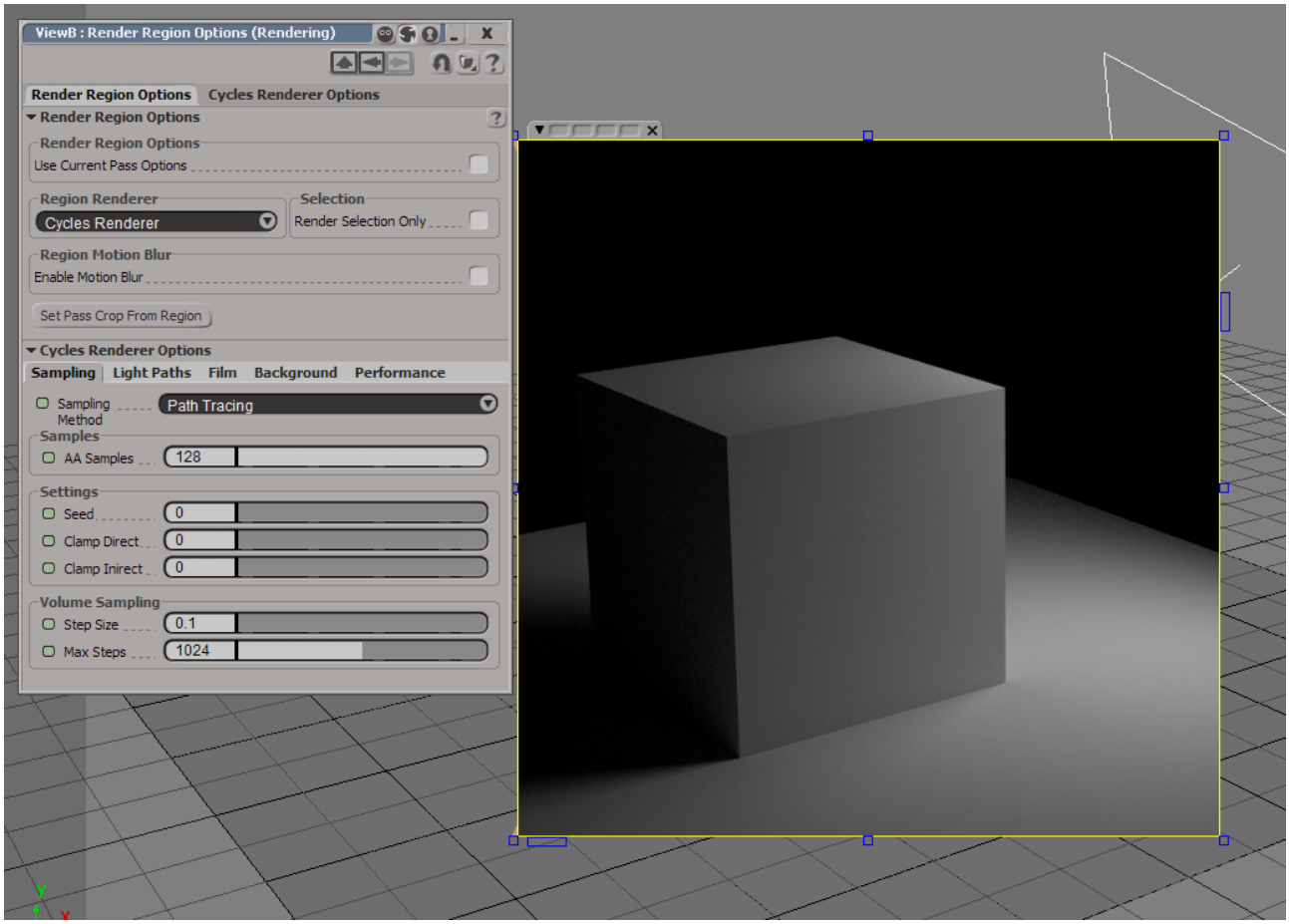
Then select the render area and in the settings we specify `Cycles Renderer`.



Nothing to see. This is because the power of the light source is too small. Go to the `Render Tree` of the light source and set the value `Strength` of the node `Emission` equal to 250.

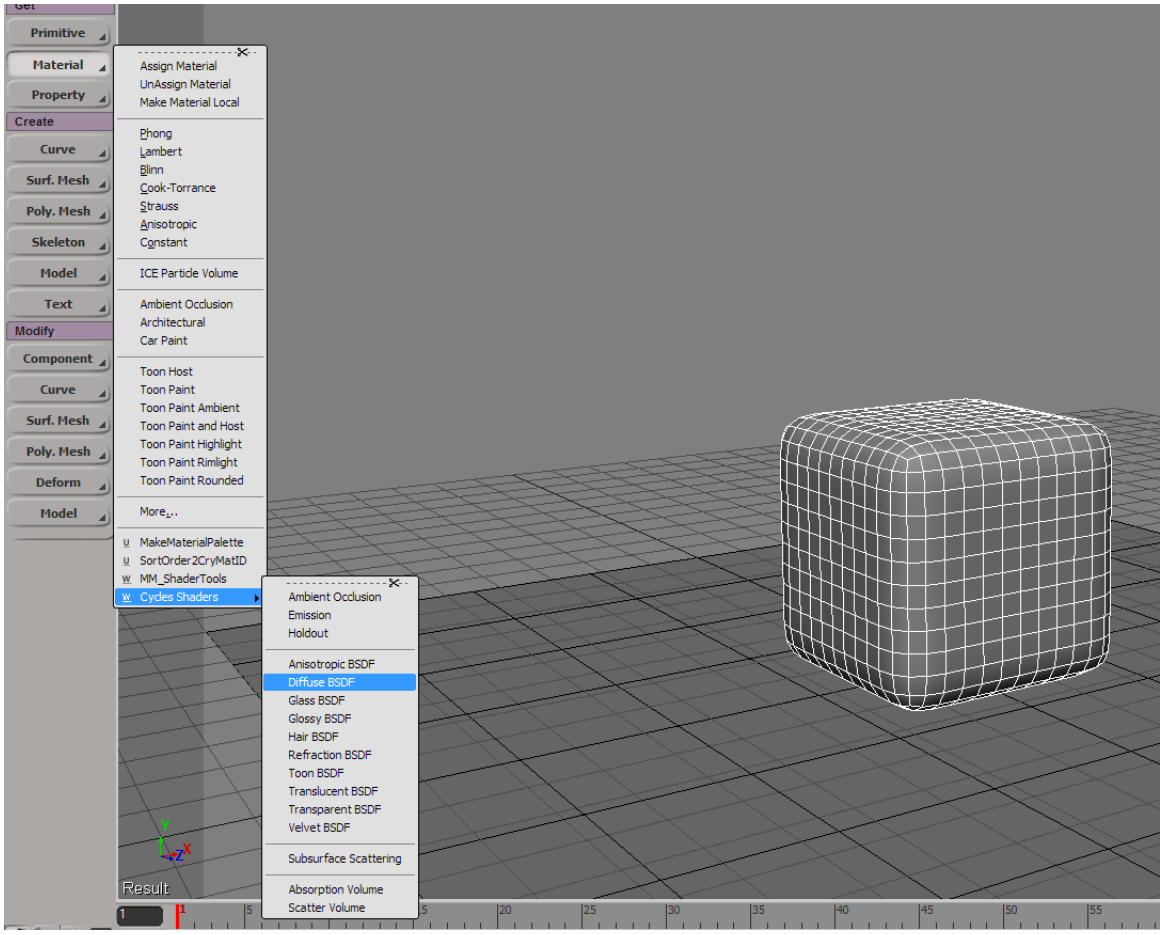


We get the result.

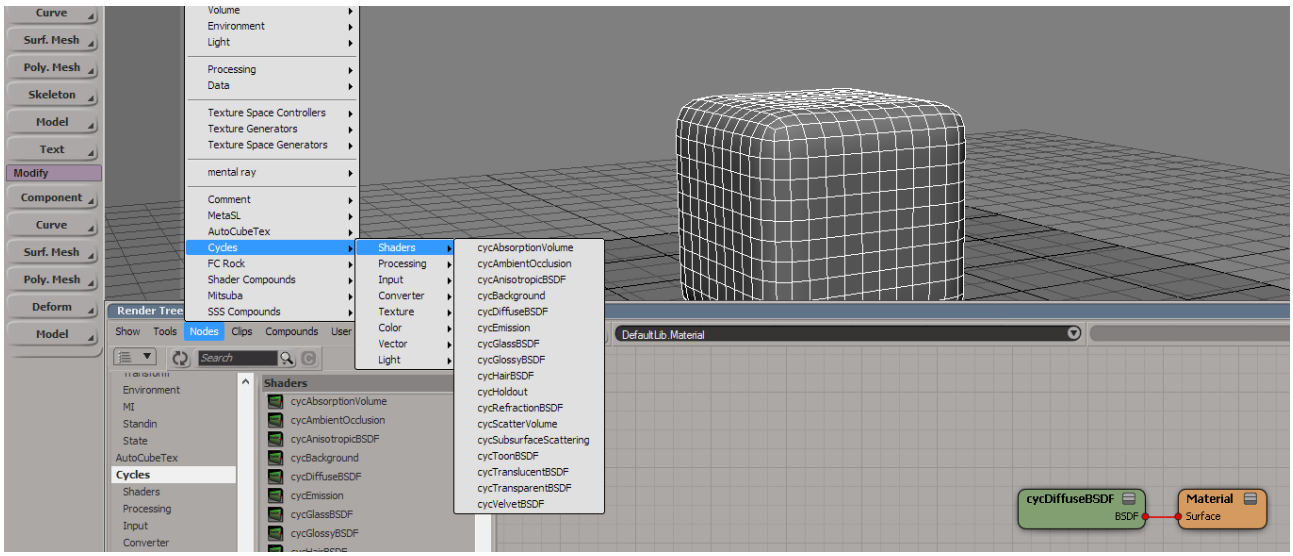


2 How to assign shaders

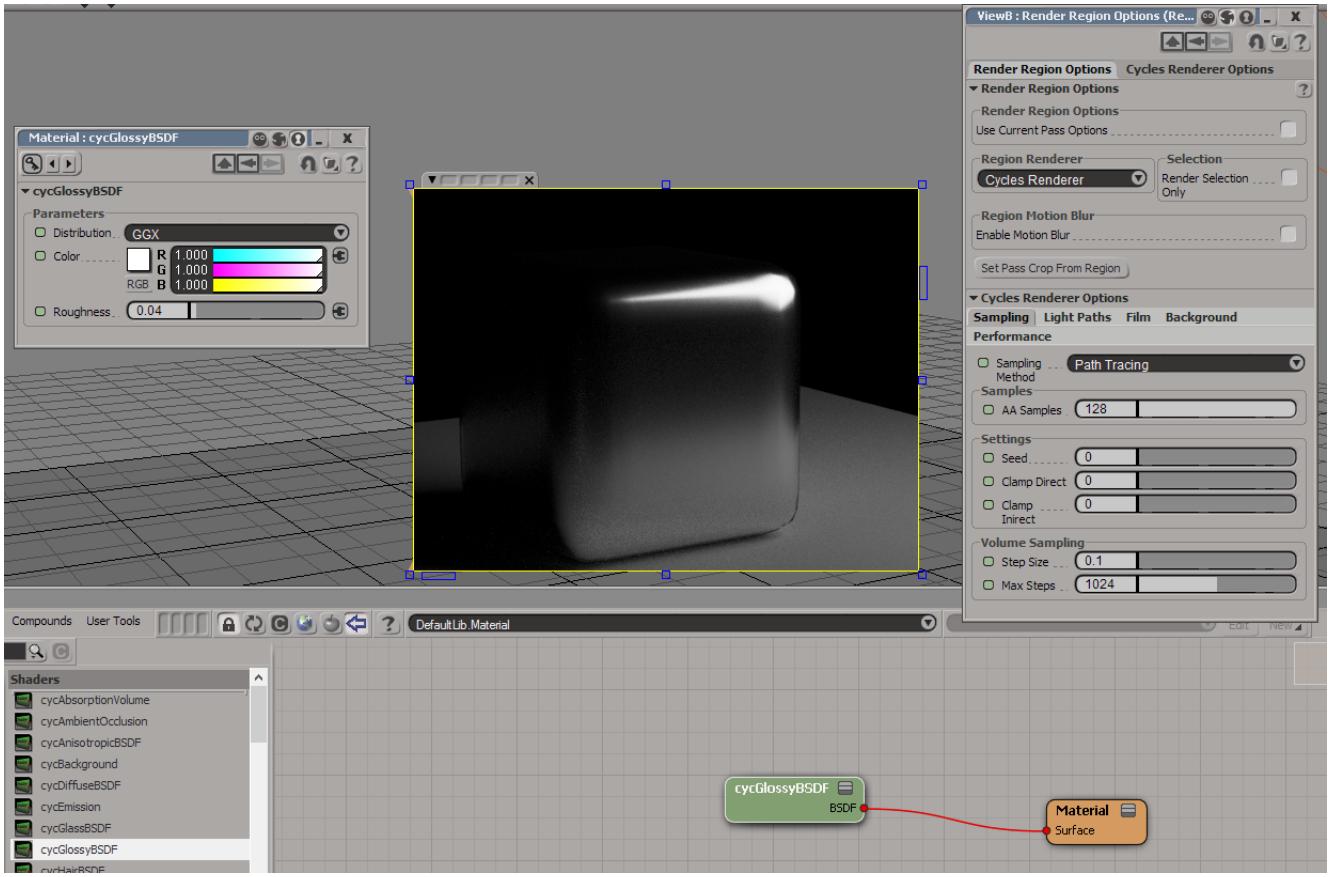
Two ways. You can use the command `Material` – `Cycles Shaders` and select which shader to assign to the selected object.



Or you can directly connect nodes in `Render Tree`. Everything that Cycles understands is in a separate section.

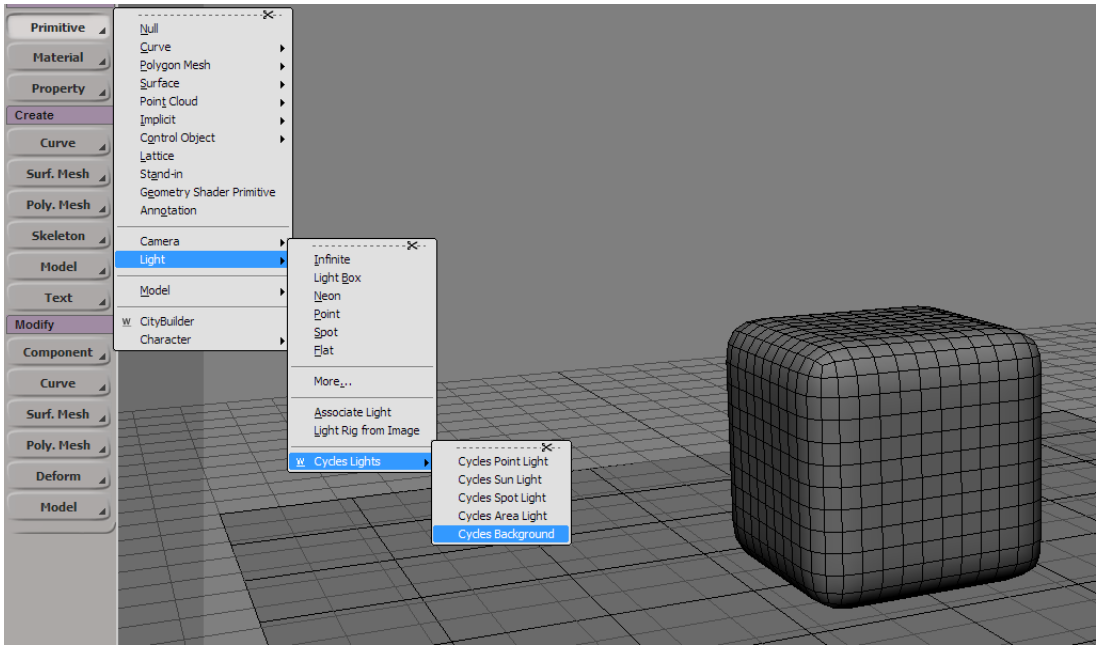


As an example, let's connect the node `GlossyBSDF` to the port `Surface` of the material's root node. We well obtain the result.

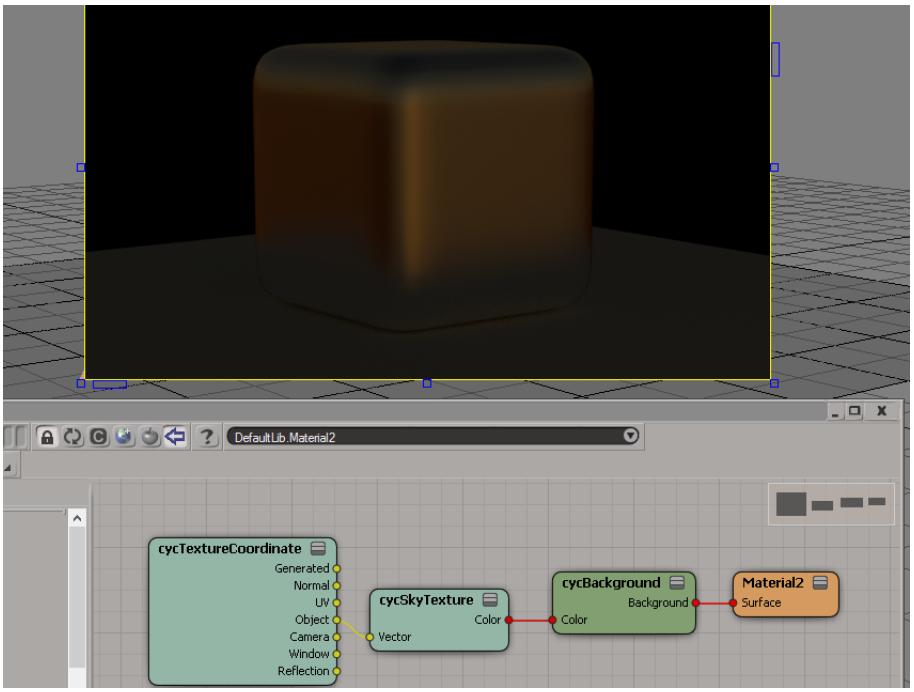

Material : cycGlossyBSDF
cycGlossyBSDF
Parameters
Distribution
GGX
Color
R 1.000
G 1.000
RGB B 1.000
Roughness
0.04
ViewB : Render Region Options (Re...
Render Region Options
Cycles Renderer Options
Render Region Options
Render Region Options
Use Current Pass Options
Region Renderer
Cycles Renderer
Selection
Render Selection Only
Region Motion Blur
Enable Motion Blur
Set Pass Crop From Region
Cycles Renderer Options
Sampling
Light Paths
Film
Background
Performance
Sampling Method
Path Tracing
Samples
AA Samples
128
Settings
Seed
0
Clamp Direct
0
Clamp Inirect
0
Volume Sampling
Step Size
0.1
Max Steps
1024
Compounds
User Tools
DefaultLib.Material
Shaders
cycAbsorptionVolume
cycAmbientOcclusion
cycAnisotropicBSDF
cycBackground
cycDiffuseBSDF
cycEmission
cycGlassBSDF
cycGlossyBSDF
cycGlossyBSDF
BSDF
Material
Surface

3 How to use sky and sun for environment

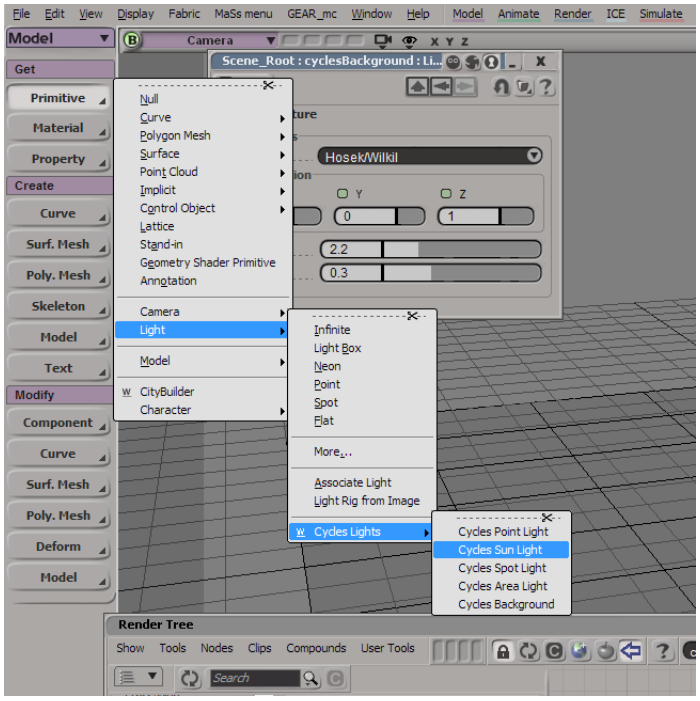
Suppose we have a scene: a cube and a plane. Adding a light source of the type `Background`.



To the shader of this light source add the node `SkyTexture`. Connect it to the port `Color` of the node `Background`. To the port `Vector` of the node `SkyTexture` connect the output port `Object` of the node `TextureCoordinate`. This is necessary in order to explicitly tell the renderer which texture coordinates for the environment to use. Render.

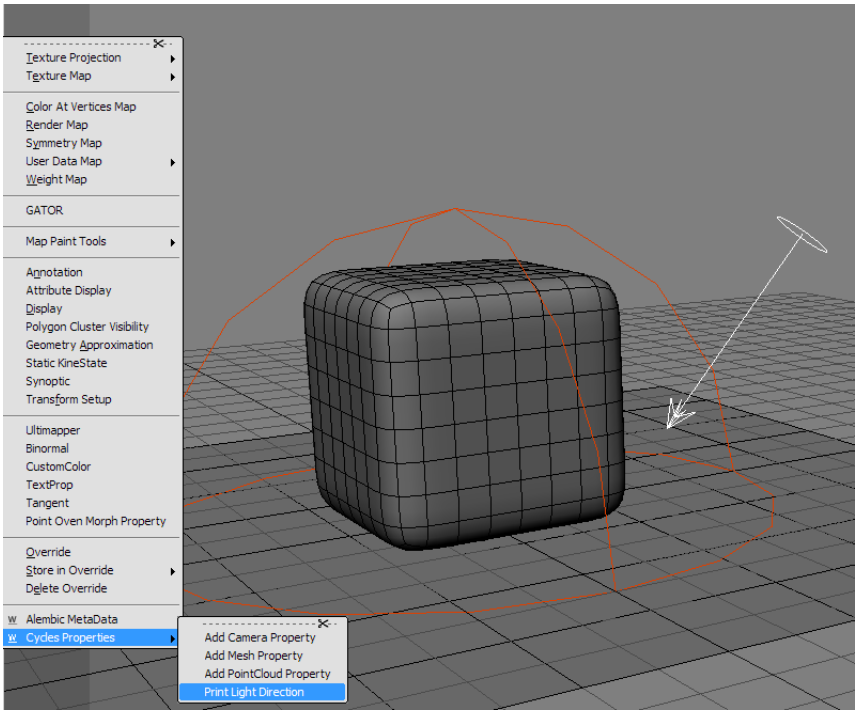


Something wrong. This is because the vector for `Sun Direction` has the value `(0, 0, 1)`. This is means that the sun directed horizontally. We should correct it. Add the light source `Cycles Sun Light`.

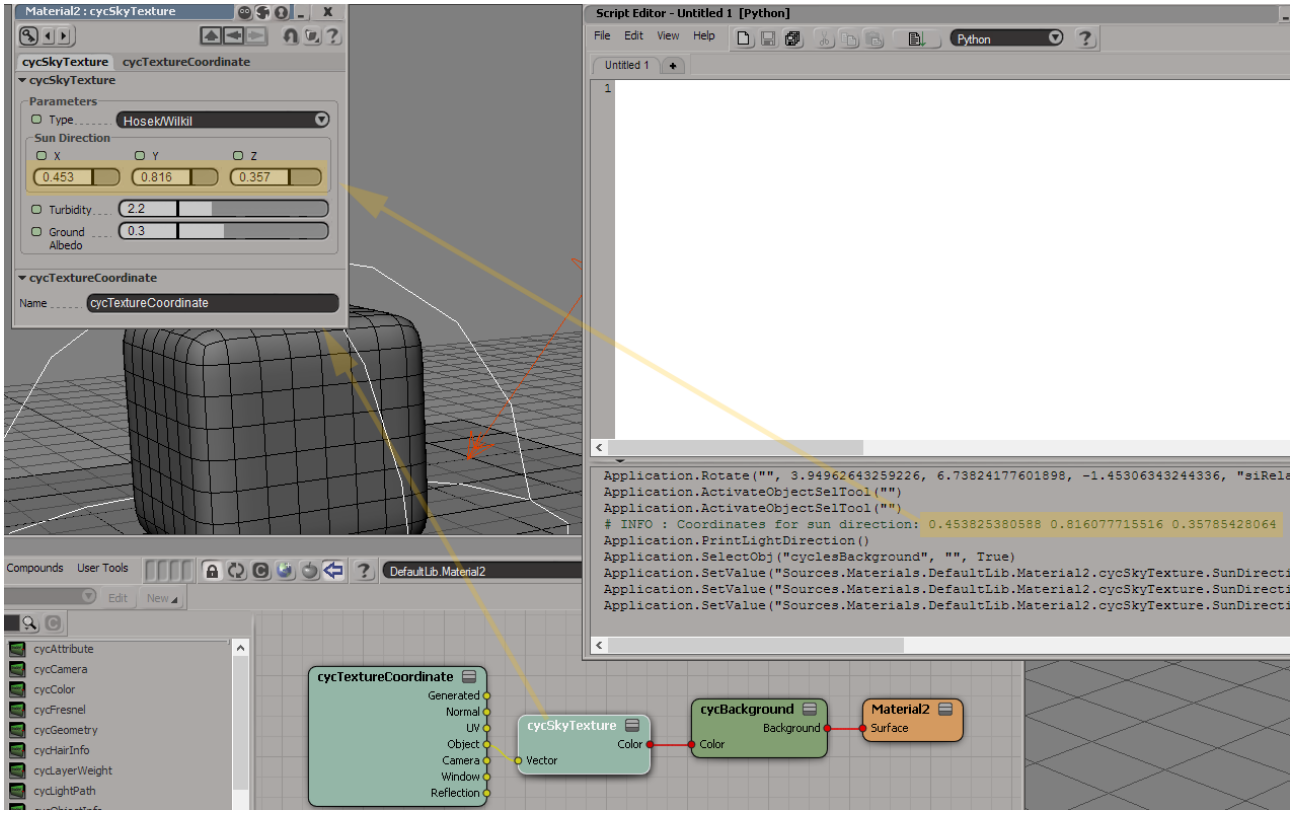


Next we need to know its direction. To do this, select the sun and choose

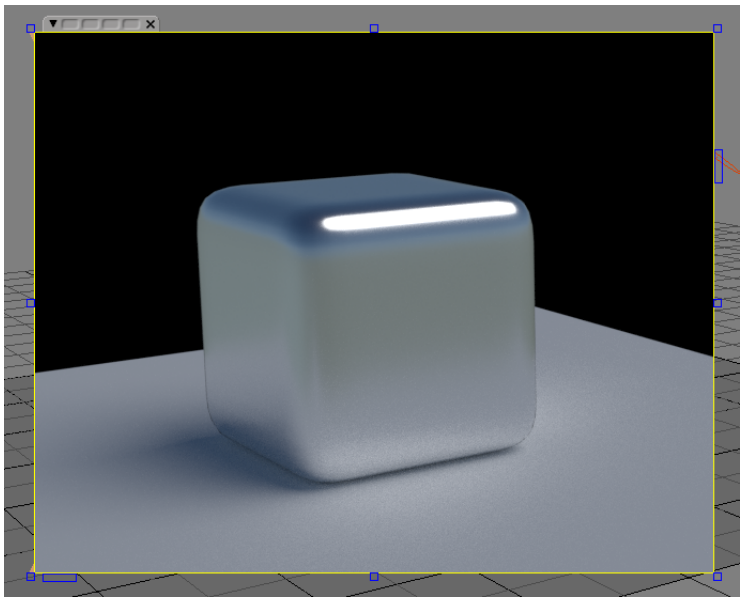
`Property` – `CyclesProperties` – `Print Light Direction`.



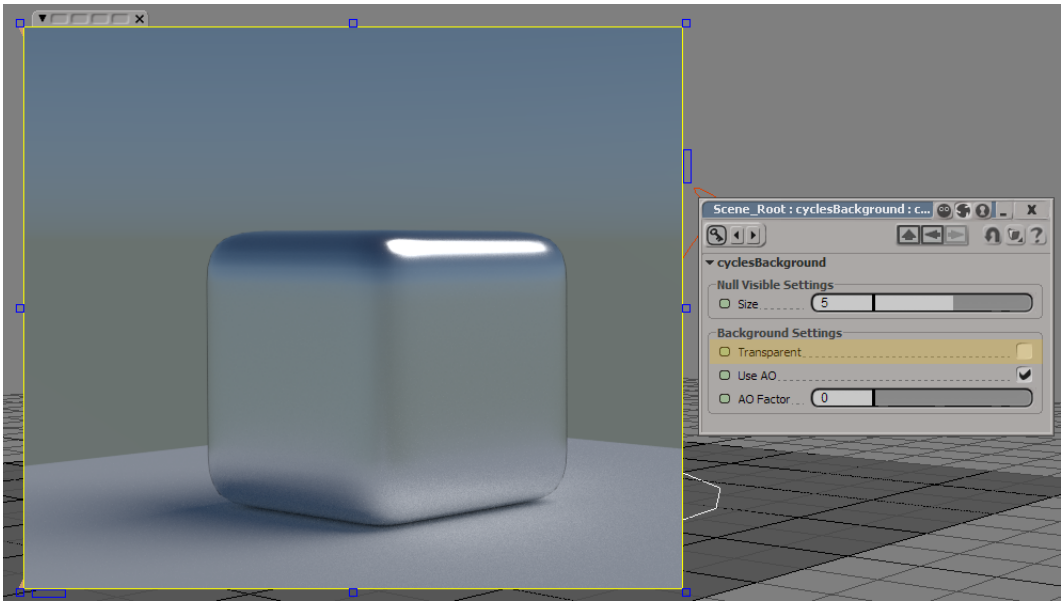
Three numbers will appear in the log, which we should insert into the fields of the direction of the sun in the node `SkyTexture`.



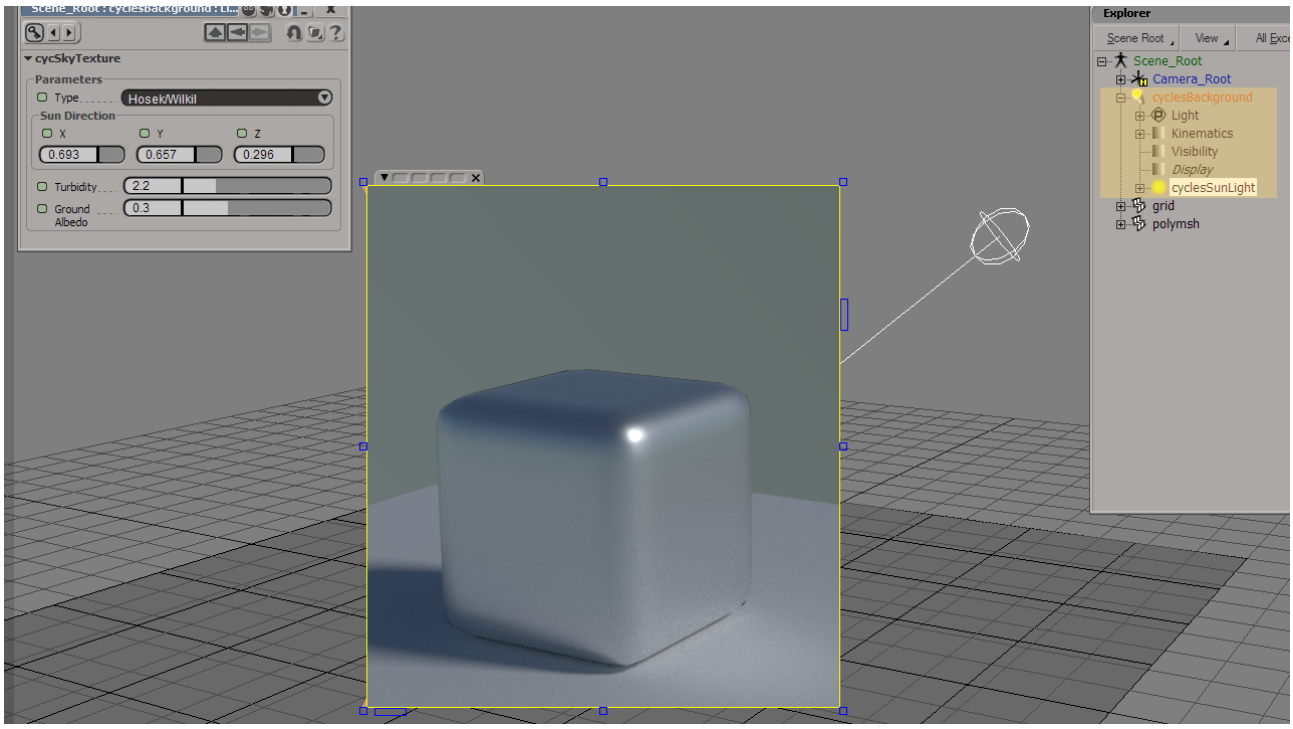
Render. Now the color of the sky and the direction of the sun are coordinated.



Yes, to see the actual color of the sky, it is necessary to check off the parameter `Transparent` in the properties of a light source of the type `Background`.

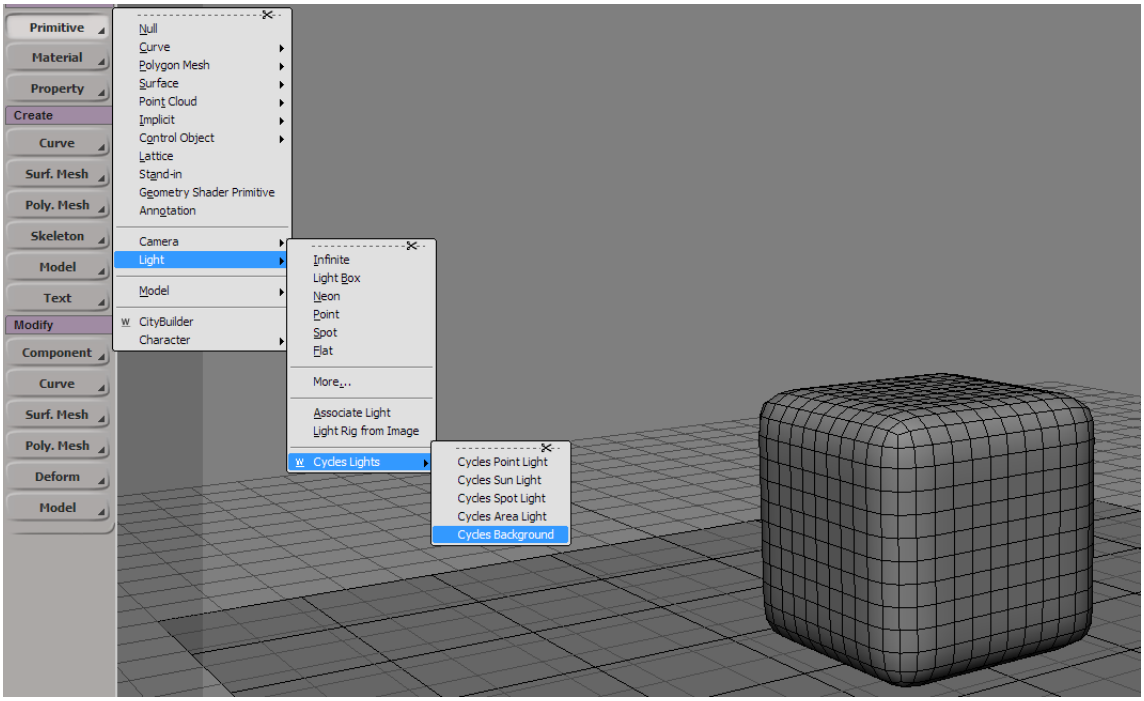


Maybe someone will say: “What the f..! Every time after the sun’s shift, manually rewrite the values of the direction?!” Our answer is “No”. If the sun is a child of the background object, then the renderer will ignore the values in the fields `Sun Direction`, and using the spatial values of the first child object that has the type of the sun.

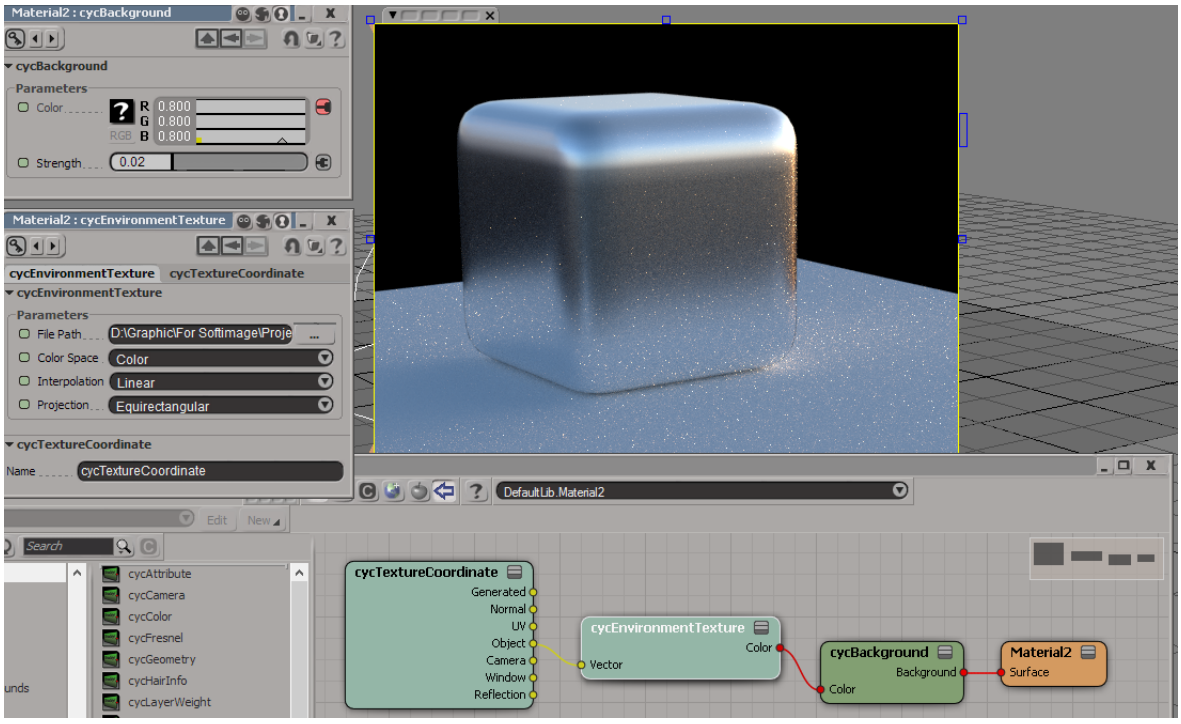


4 How to use HDR for the environment

Suppose we have a scene: a cube and a plane. Let's add a light source of the type `Cycles Background`.

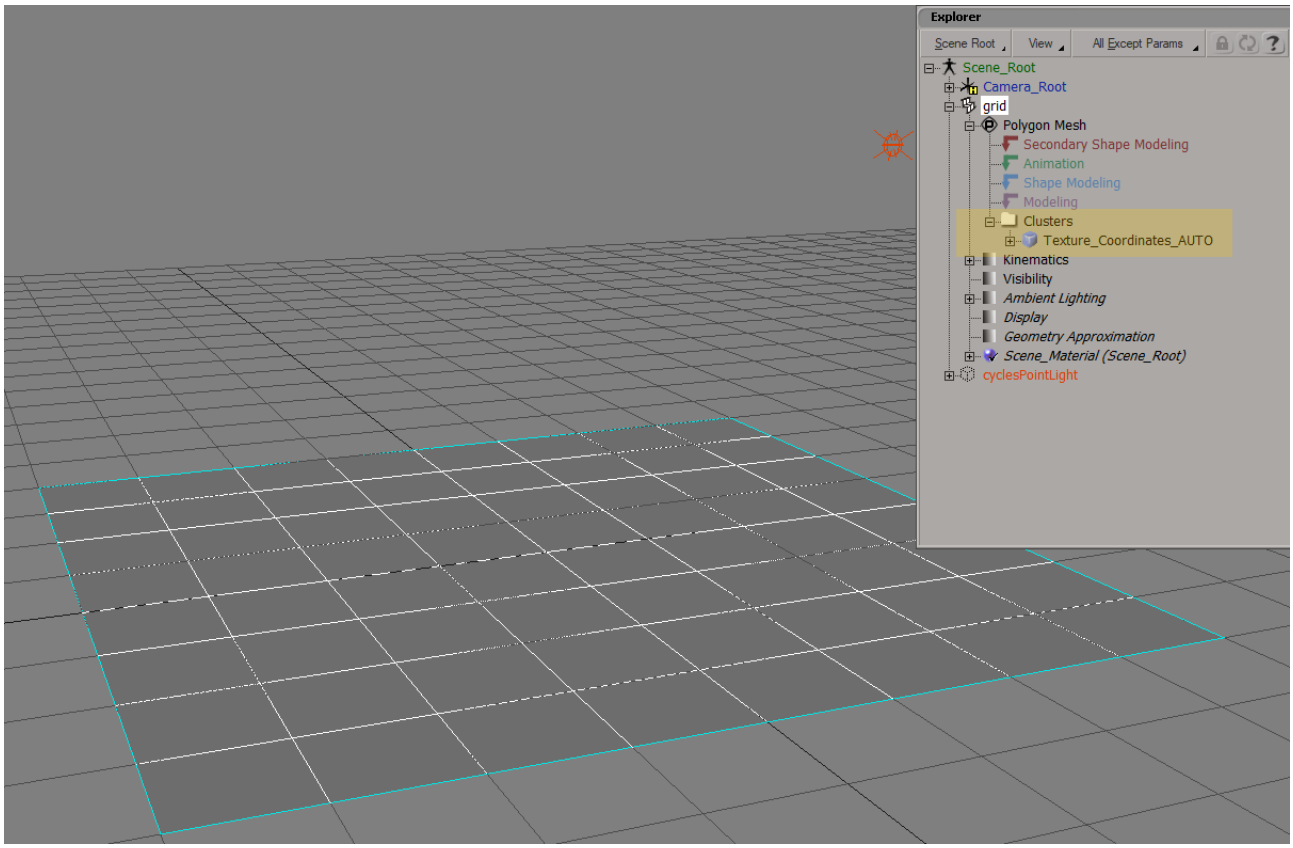


Add into the shader the node `EnvironmentTexture` and connect it to the port `Color` of the node `Background`. Connect the output of the node `TextureCoordinates` to the port `Vector`. Next, select in the node `EnvironmentTexture` any picture, set the strength of the light source and render it.

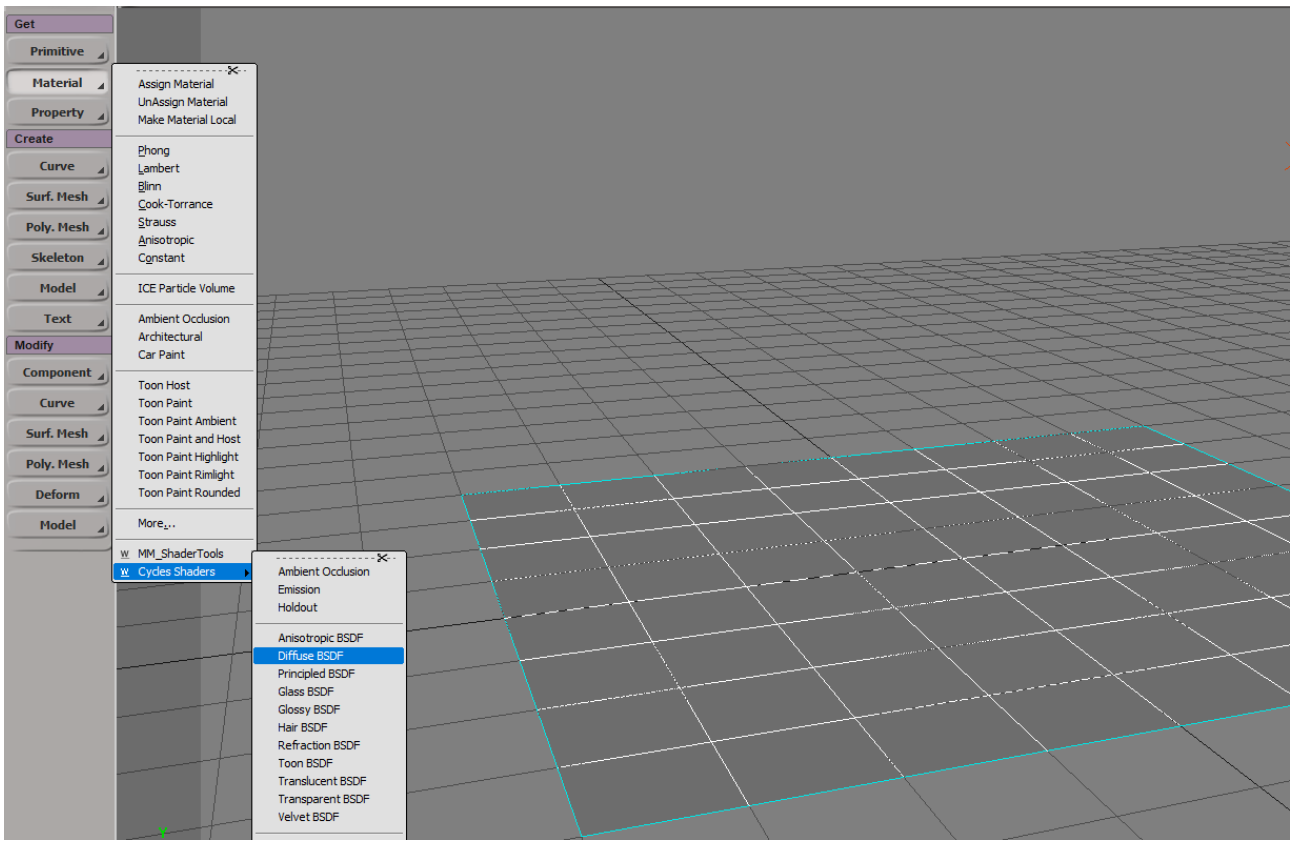


5 How to use Bump

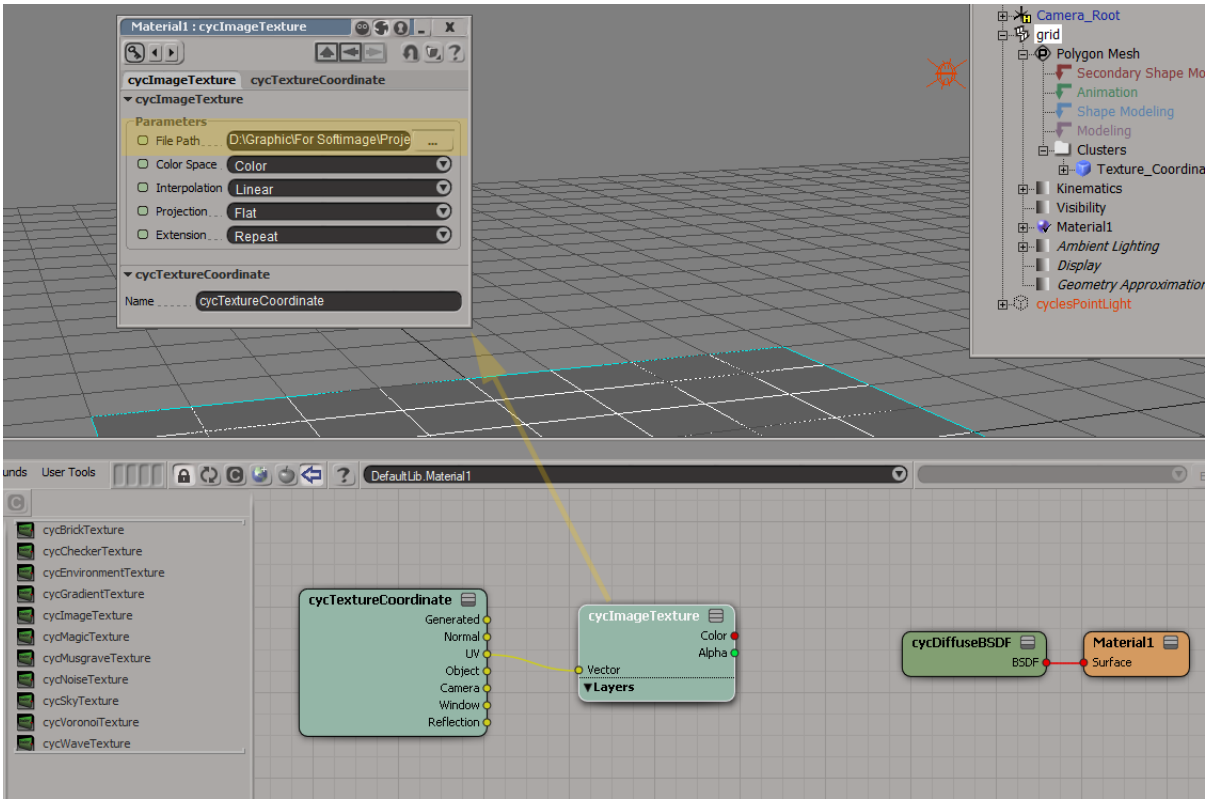
Suppose we have a scene: the plane and the light source. The plane contains uv coordinates.



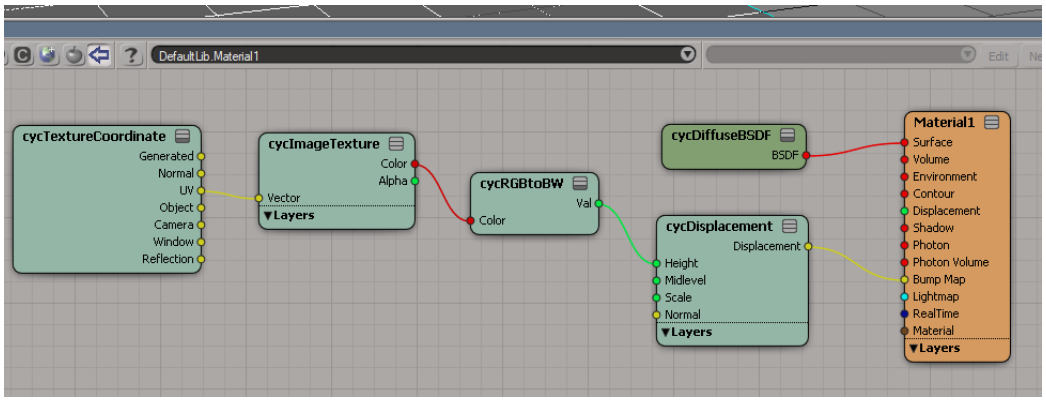
Assign to the plane the shaser `DiffuseBSDF`.



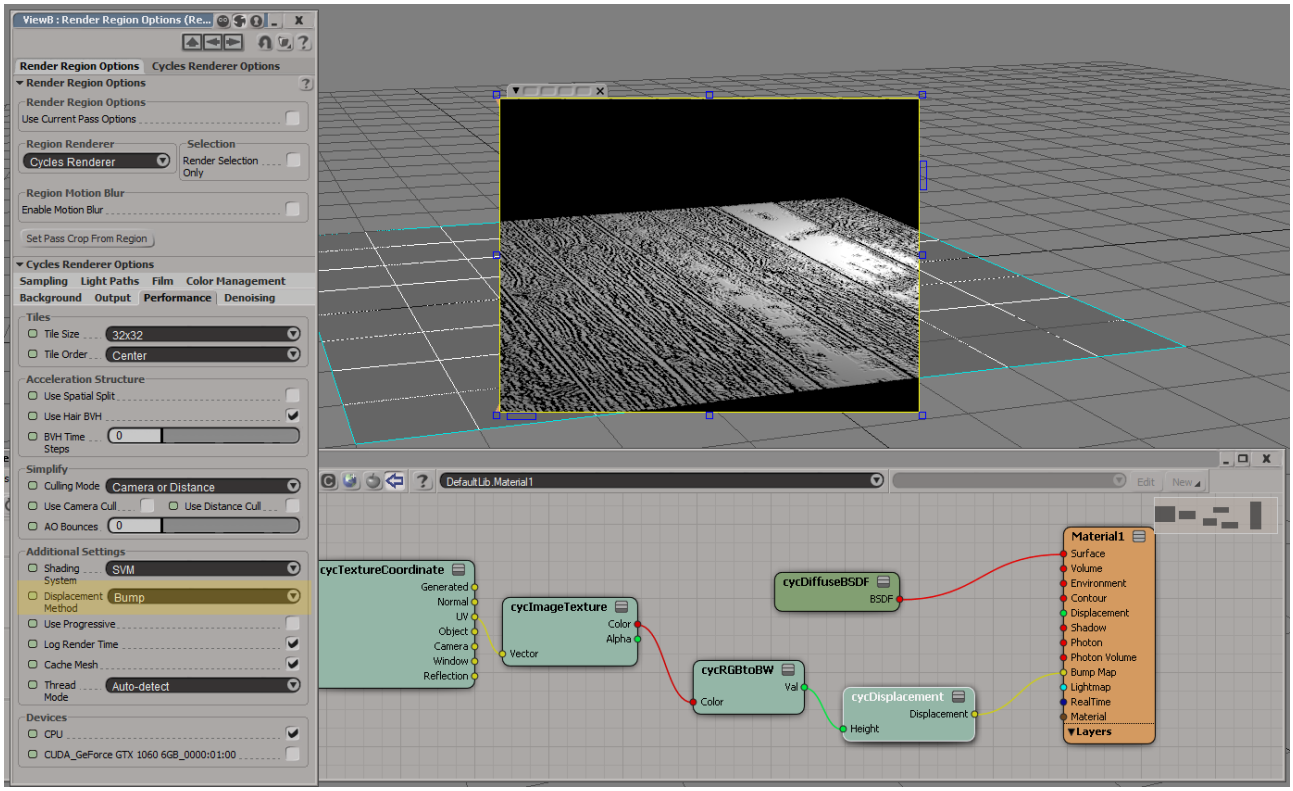
Next add to this shader two nodes: `TextureCoordinate` and `ImageTexture`. Connect the output `UV` of the first node to the input `Vector` of the second one. Select the image for the bump map.



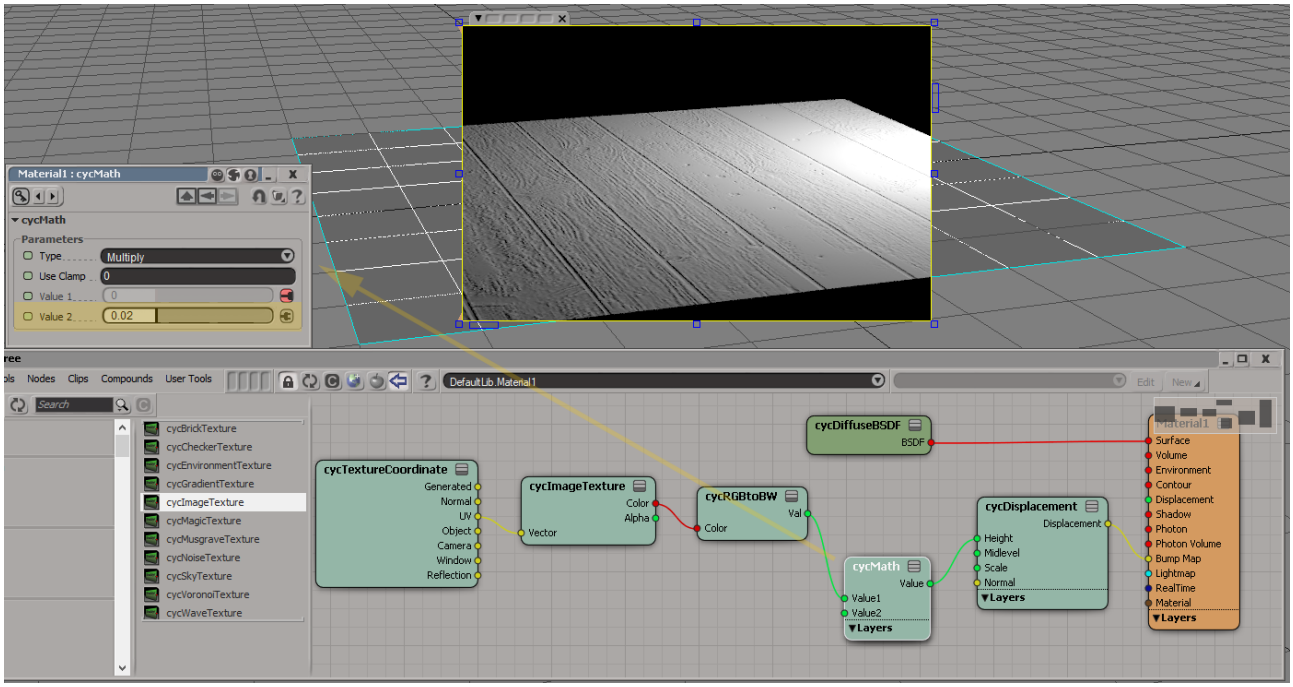
Convert the output `Color` into black-and-white image with the help of the node `RGBtoBW`, and connect the result to the input `Height` of the node `Displacement`. The unique output of this node has vector type. That's why we connect it to the input port `Bump Map` of the material's root node. This is the unique vector type port of it.



In the render properties set the mode `Displacement Method` to the `Bump`. Render.

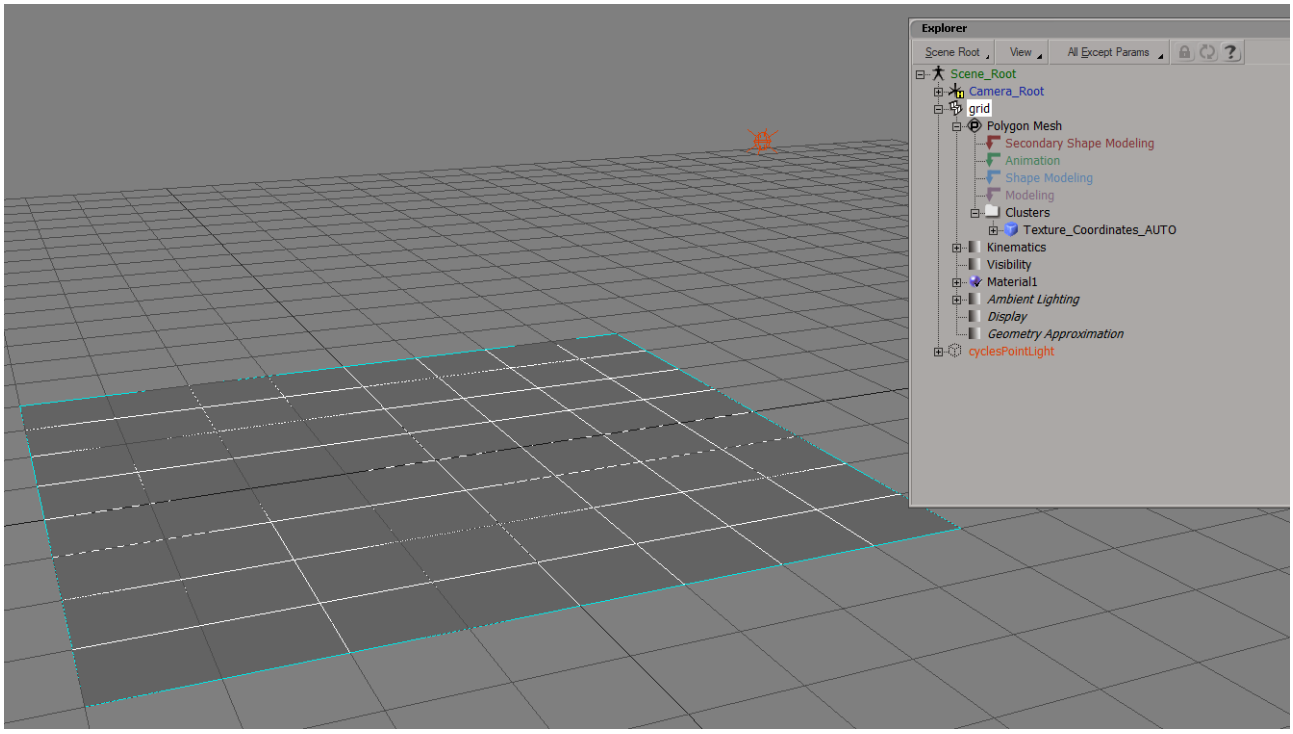


To adjust the bump strength we can use the node `Math`. Insert it before the port `Height` of the node `Displacement`, switch the node to `Multiply` and the value `Value 2` be responsible to the power of the bump.

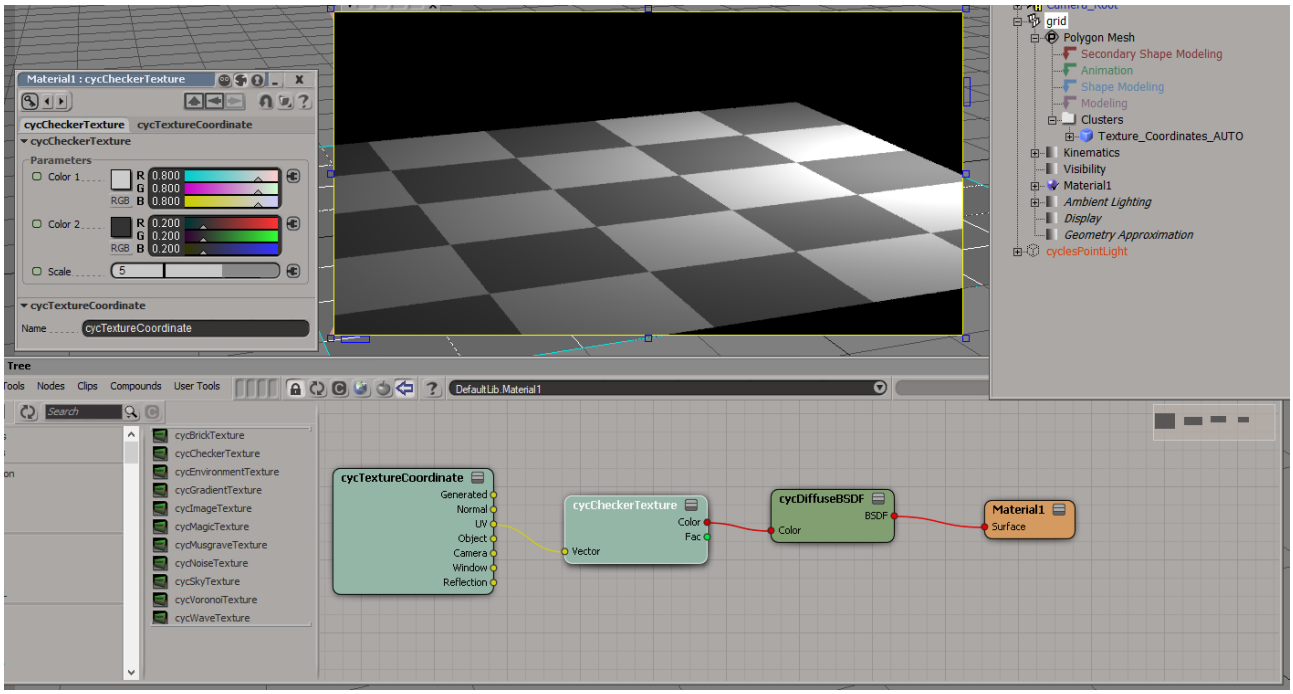


6 How to make true displacement

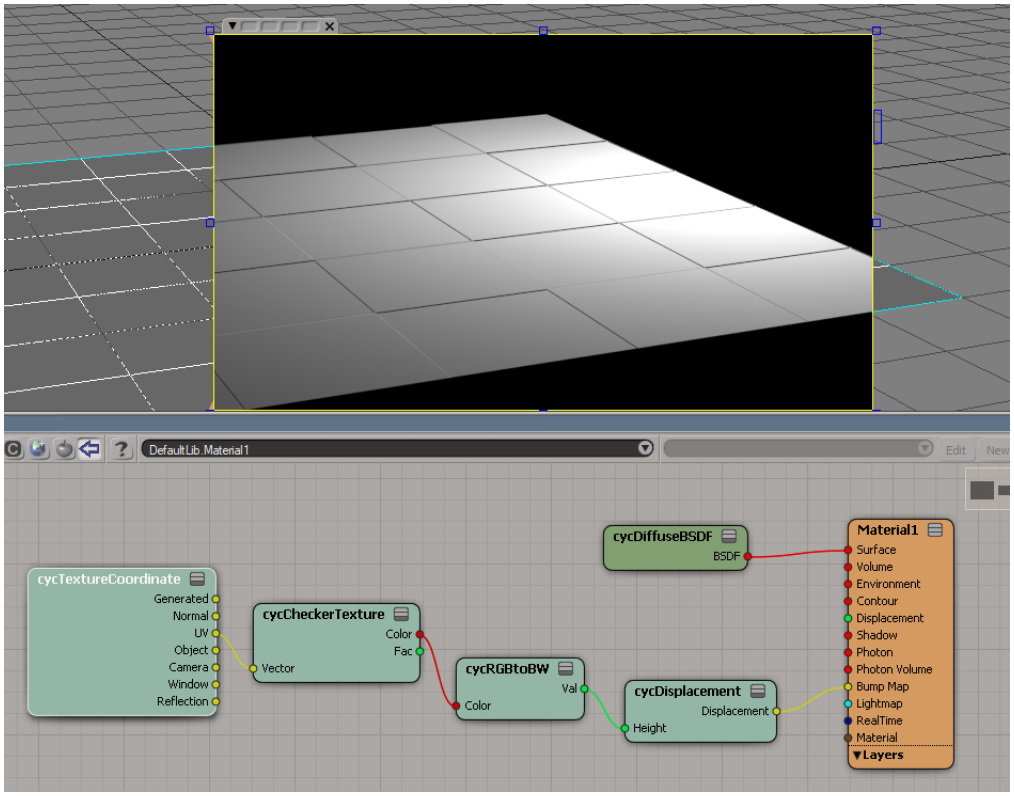
Suppose we have a scene: the plane and the light source. The plane contains uv coordinated.



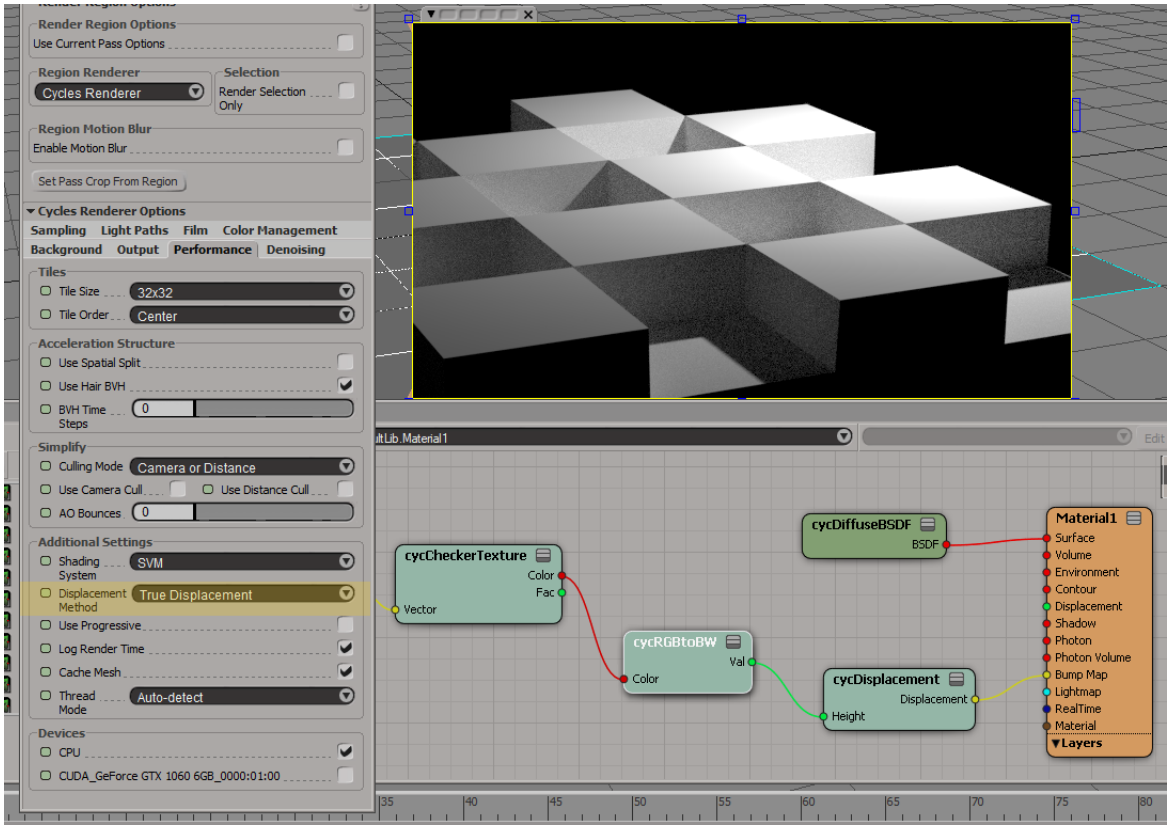
Assign the shader `DiffuseBSDF` o the plane and add to it the node `CheckerTexture`. Connect it output to the port `Color` of the node `DiffuseBSDF`. To make the texture overlap along texture coordinates, we should use the node `TextureCoordinate`.



Connect the output `Color` to the port `Height` of the node `Displacement` by passing throw the converter to white-and-black format. The output of the node `Displacement` connect to the port `Bump Map` of the material's root node. Render and see the simple bump.



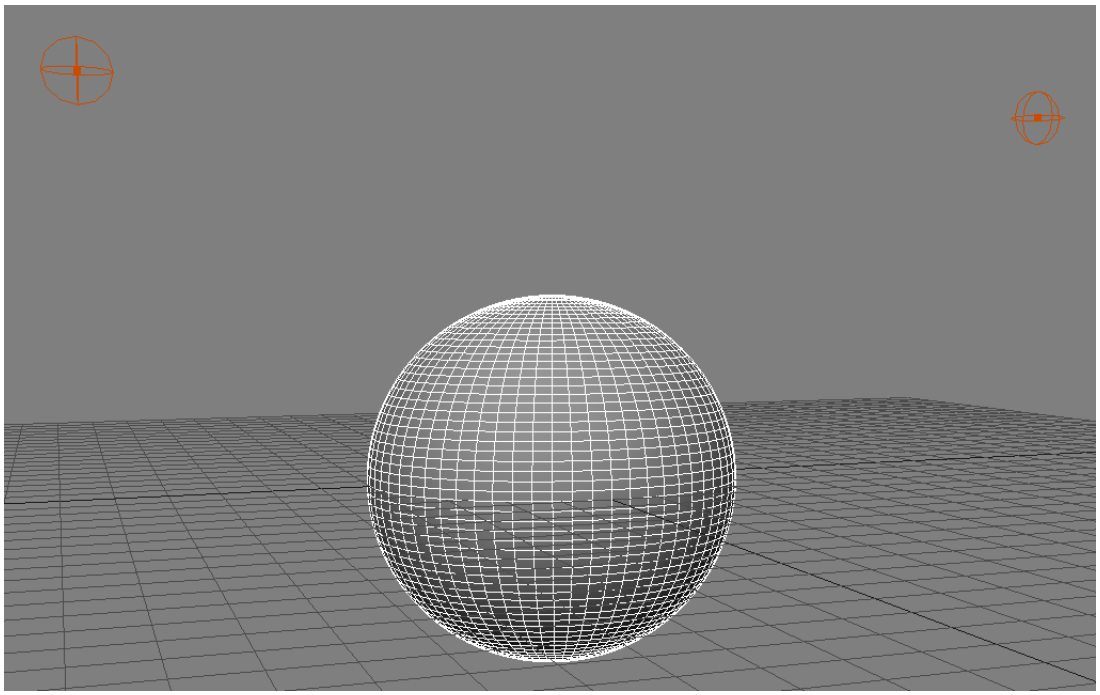
But how we can create displacement? To do this we should fo to the render properties and in the section `Performance` choose `True displacement` for the parameter `Displacement Method`. For correct result increase the number of polygons on the plane.



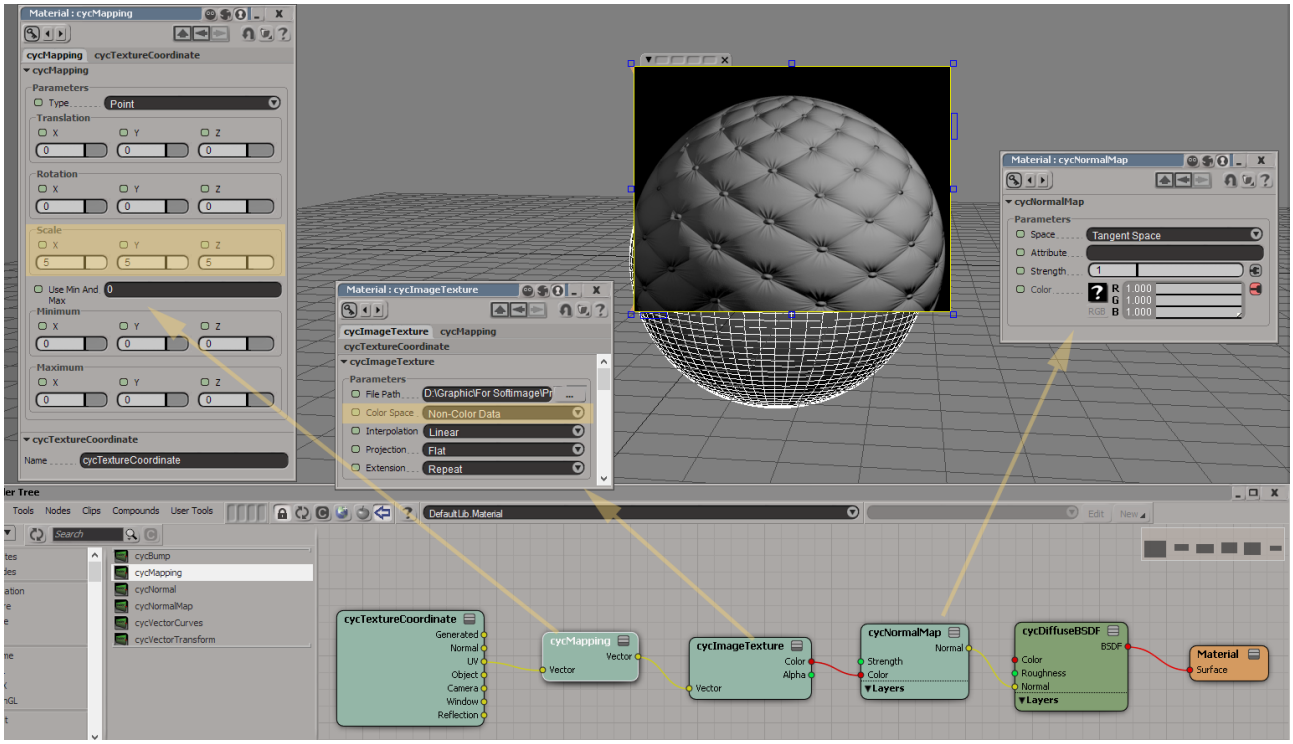
In the most cases we should use the method `Both Displacement`. In this mode the geometry deformed by texture, and if the number of polygons is not enough, then the deformation simulated by bump.

7 How to use Normal Map

Suppose we have a scene: a sphere and a pair of light sources. On the sphere, of course, there are texture coordinates.

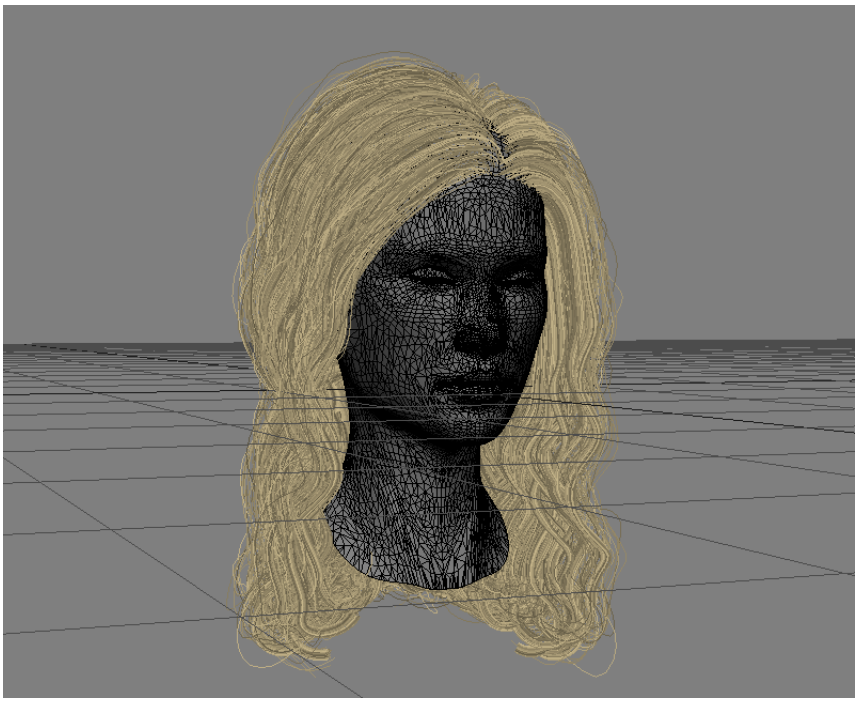


Add to the shader of the sphere the node `NormalMap`. To the port `Color` connect the node `ImageTexture`. In the node for the texture chose the actual file with the normal map and select the value `Non-Color Data` for the value of the parameter `Color Space` (so that gamma correction does not occur). Finally, specify which texture coordinates should be used for the normal map. The connection is made through the node `Mapping` so that you can specify a five times tiling of the map.

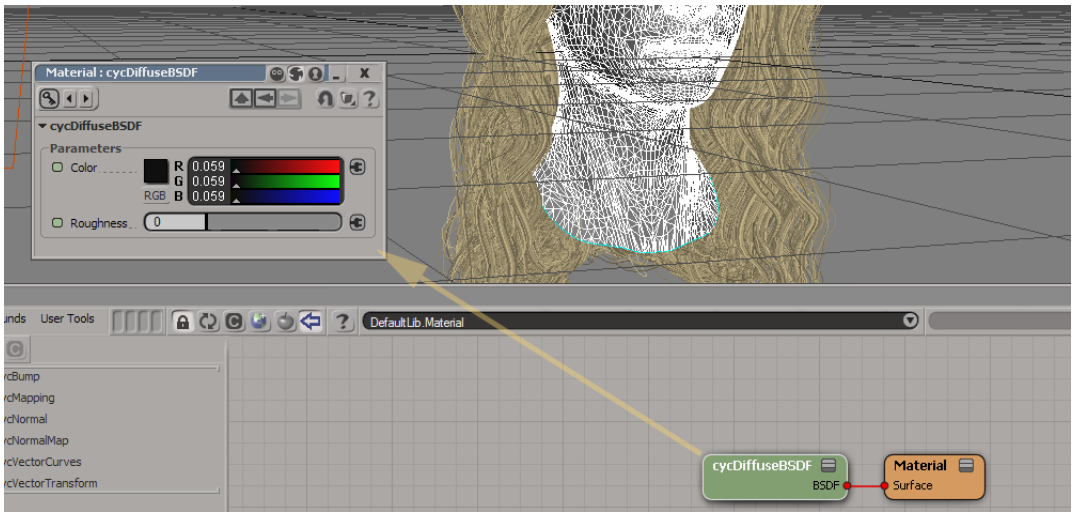


8 How to render hairs

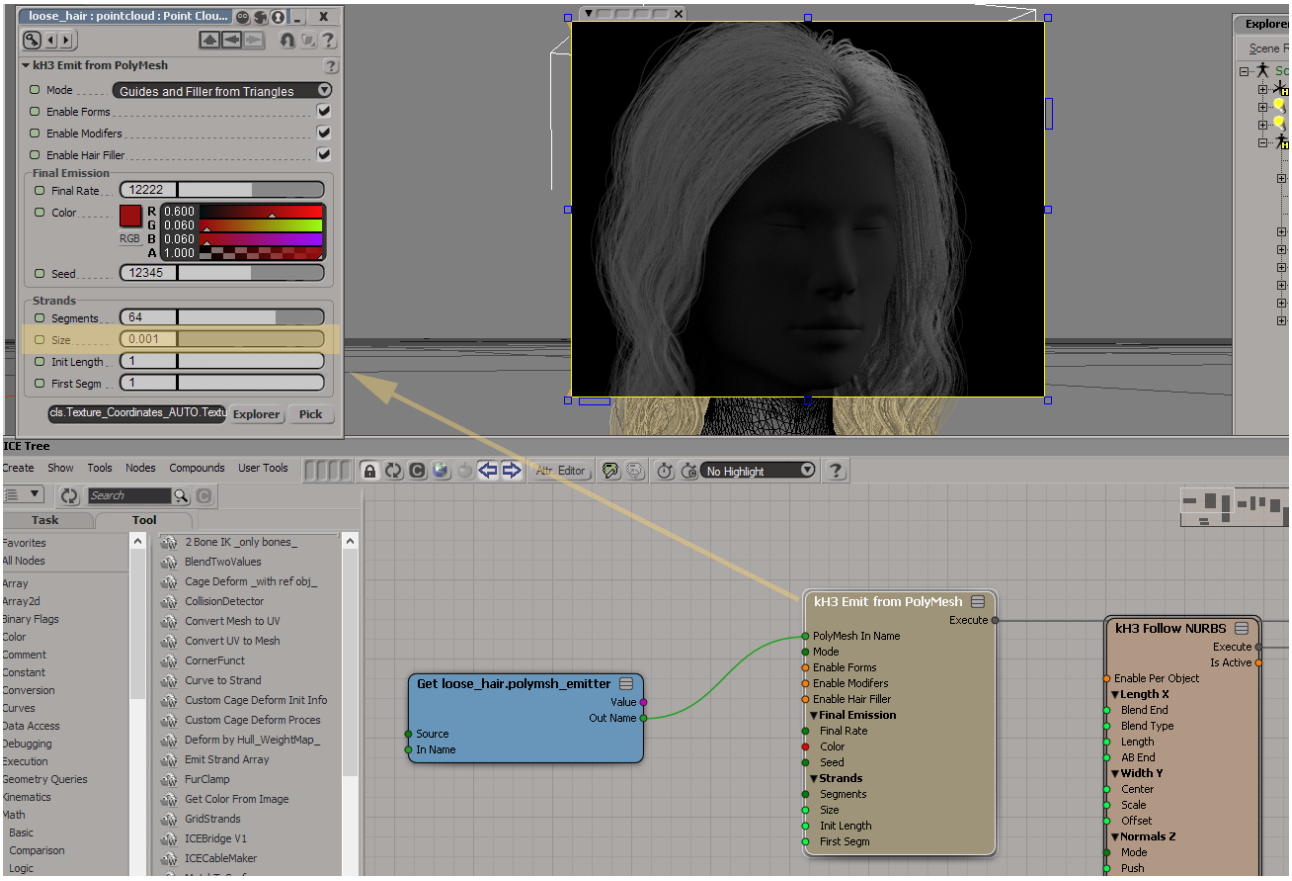
Suppose we have a head with hair made with strands.



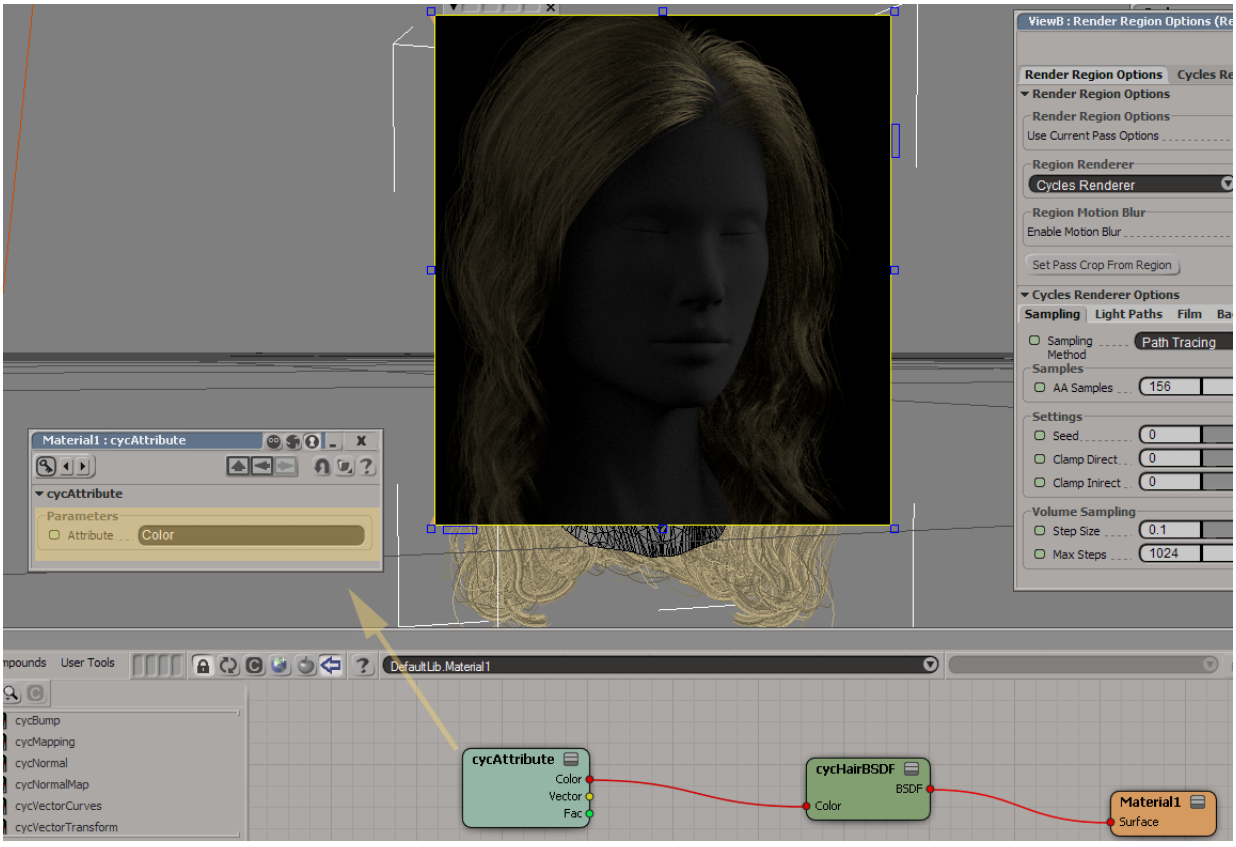
So that the head does not get in the way, we assign it a dark material.



We go into the ICE-tree of the hair and set their thickness. In our case, set the value of the parameter `Size = 0.001`. We do not touch anything else. Render.

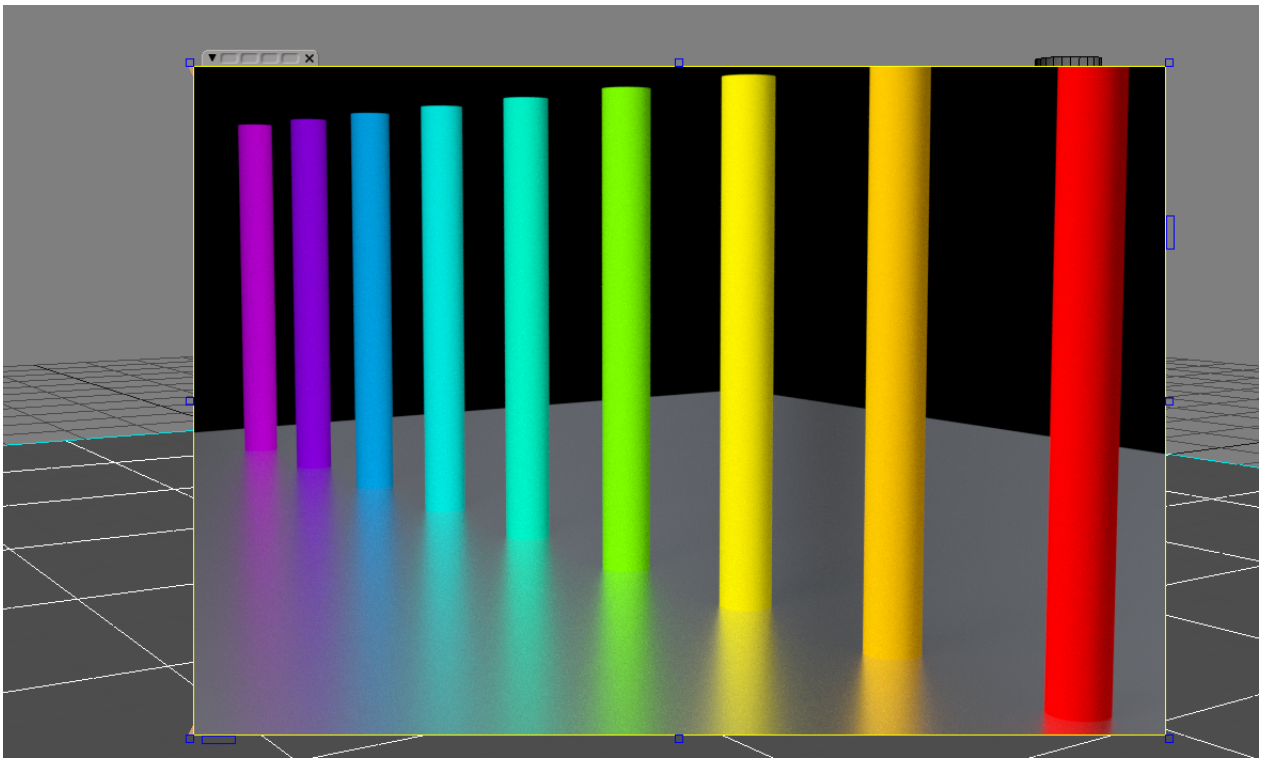


Now the hair has standard `DiffuseBSDF` shader, and he can not understand what color of the hair should be. Assign a shader `HairBSDF`, add the node `Attribute`. Set the name of the attribute `Color` and connect the output `Color` port to the port `Color` of the node `HairBSDF`. Now that's another matter.



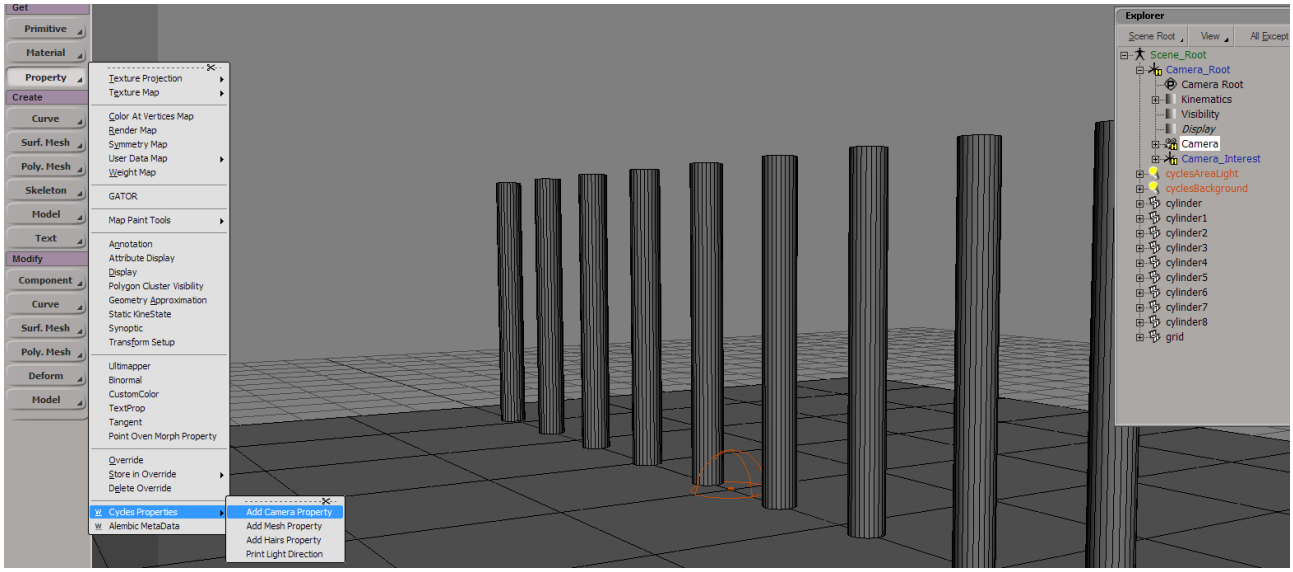
9 How to render DOF

Suppose we have a scene: the plane and the colored bars. We want the green bar to be in focus, and the rest are blurred.

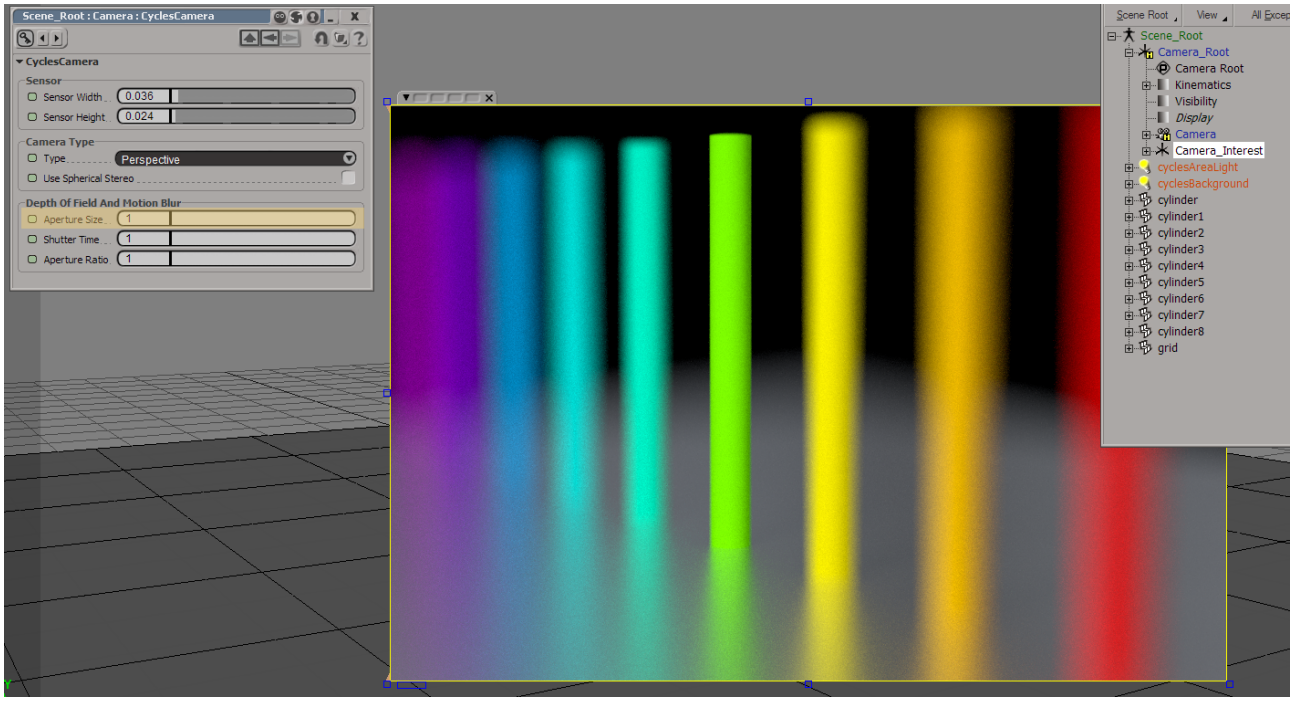


Select the render camera and add the property by the command

`Property` – `Cycles Properties` – `Add Camera Property`.

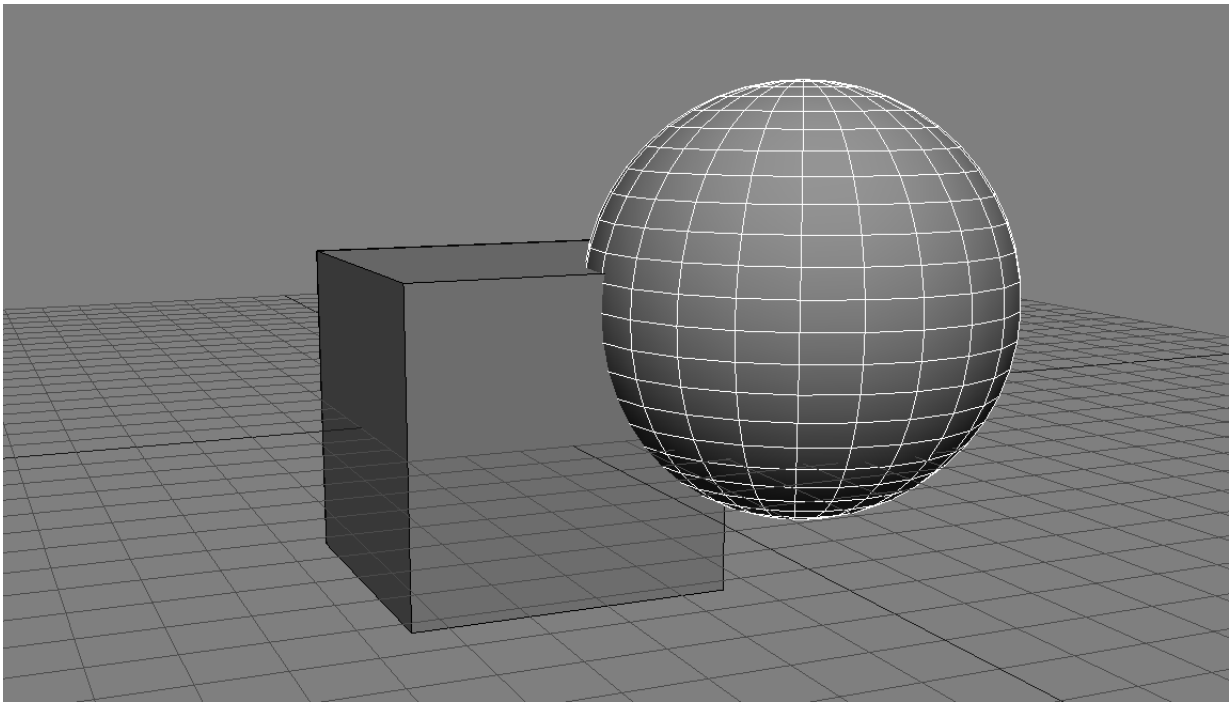


Set `Apertue Size = 1`. The focus of the camera will be in the same place as `Camera_Interest`.

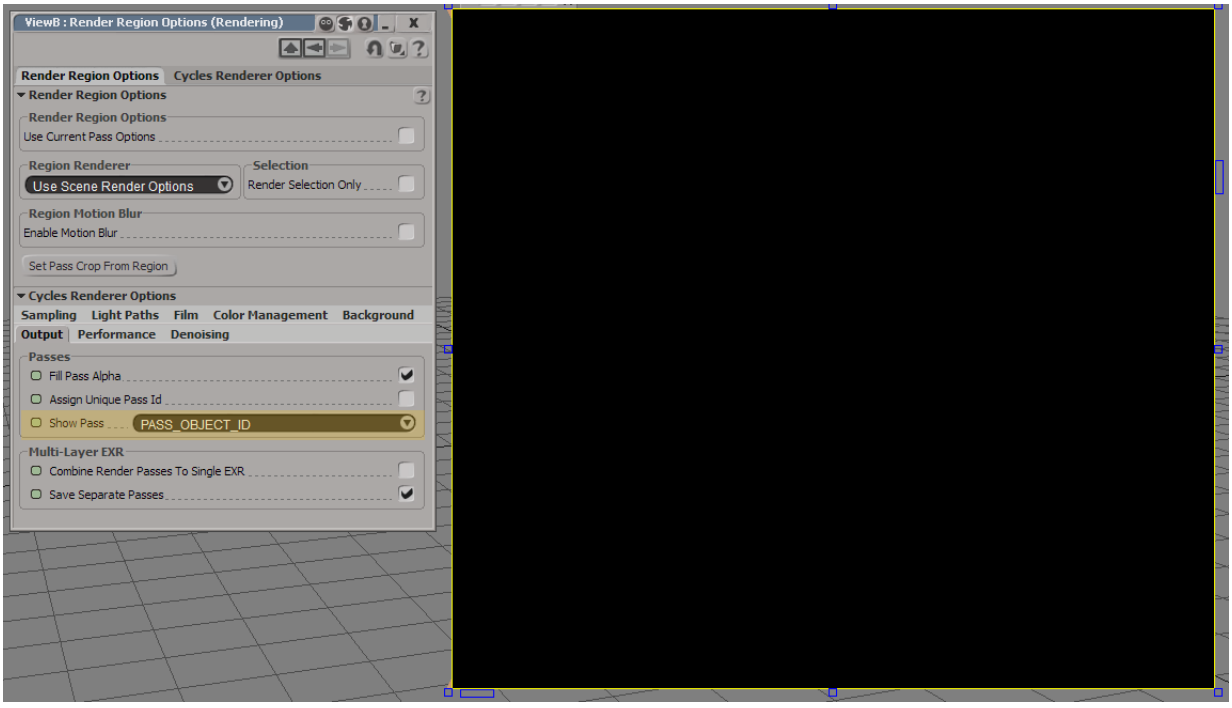

Scene_Root : Camera : CyclesCamera
CyclesCamera
Sensor
Sensor Width 0.036
Sensor Height 0.024
Camera Type
Type Perspective
Use Spherical Stereo
Depth Of Field And Motion Blur
Aperture Size 1
Shutter Time 1
Aperture Ratio 1
Scene Root
View
Scene_Root
Camera_Root
Camera Root
Kinematics
Visibility
Display
Camera
Camera_Interest
cyclesAreaLight
cyclesBackground
cylinder
cylinder1
cylinder2
cylinder3
cylinder4
cylinder5
cylinder6
cylinder7
cylinder8
grid

10 How to render Object ID pass

Suppose we have a scene: a cube and a sphere. And they are intersected.

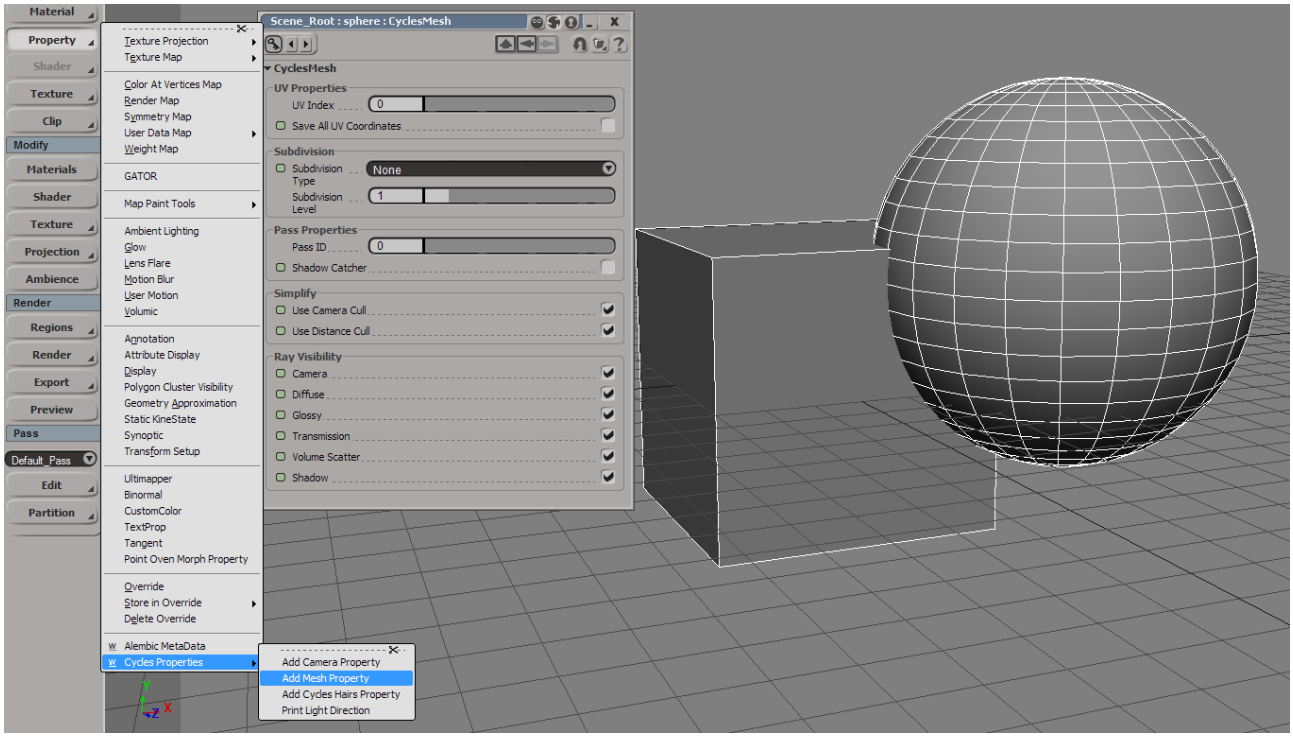


We would like to render `Object ID pass`, and then use it for compositing. Now this channel is empty. Everything is black.

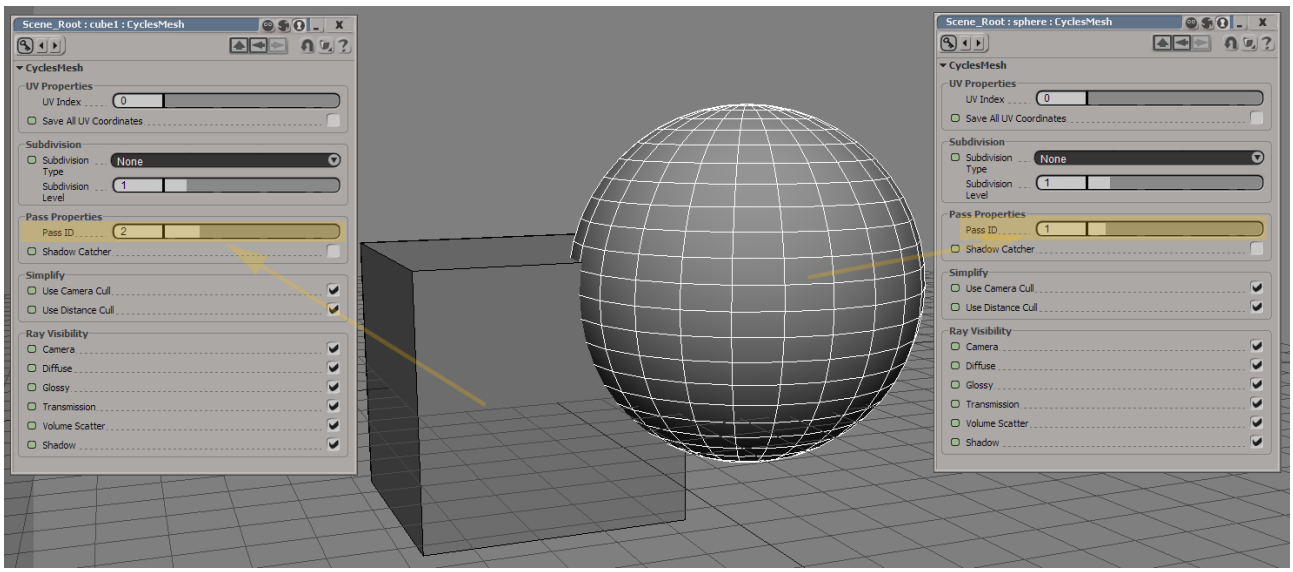


Select both objects and assign them the `CyclesMesh` property. To do this, choose the command

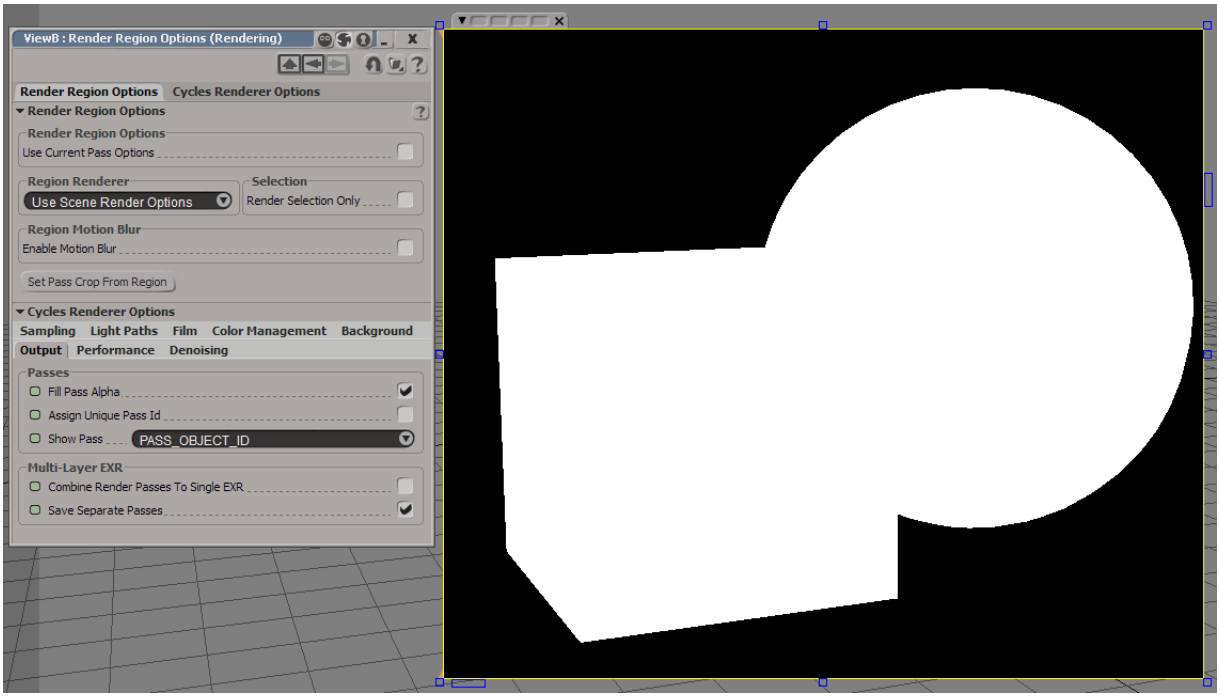
`Property` – `Cycles Properties` – `Add Mesh Property`.



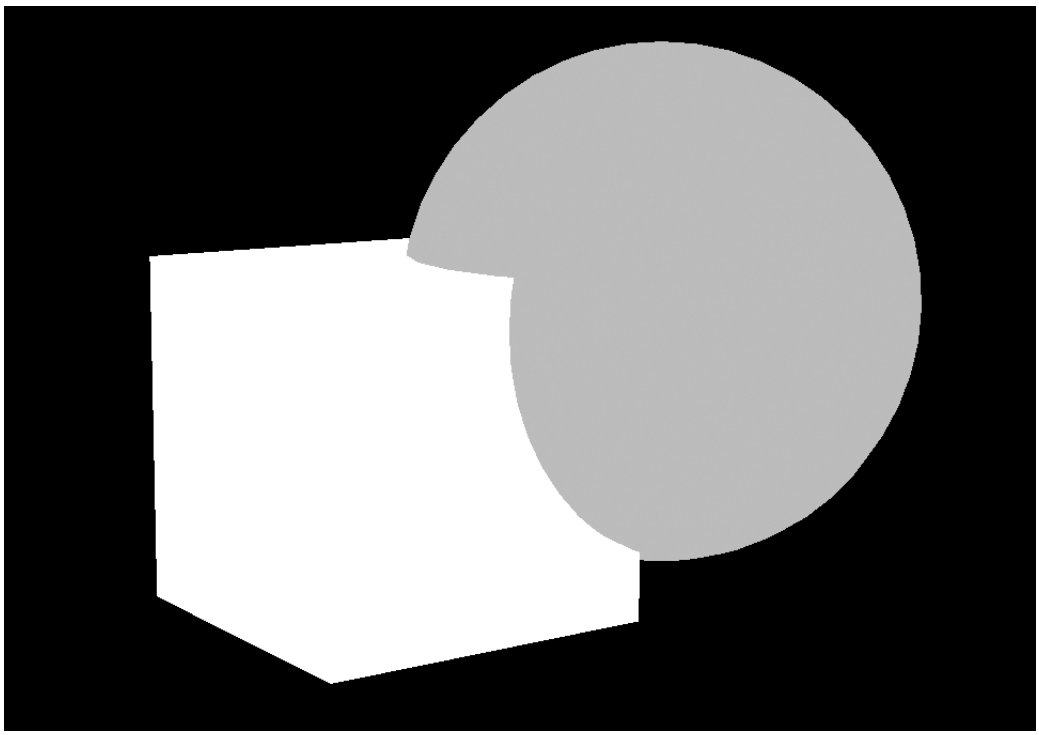
For the sphere set `Pass ID = 1`, and for cube `Pass ID = 2`.



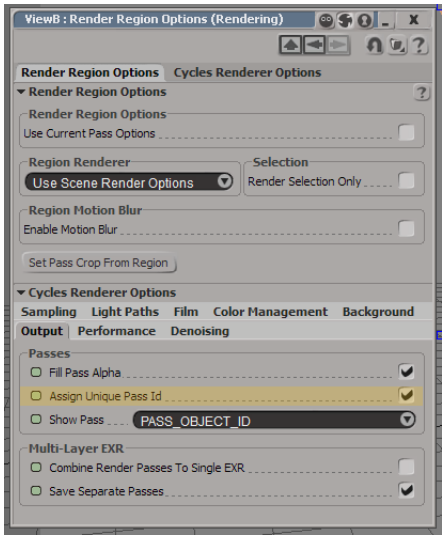
Render, and see that now the channel contains the data.



It seems that both objects are filled with the same color, but it is not. These are different shades of gray. If you render a 32-bit channel and shift levels, you can easily verify this. In general, the meaning of the channel `Object ID` is that objects with the same value `Pass ID` are displayed in one color. Those who have `Pass ID = 0` are not displayed at all.



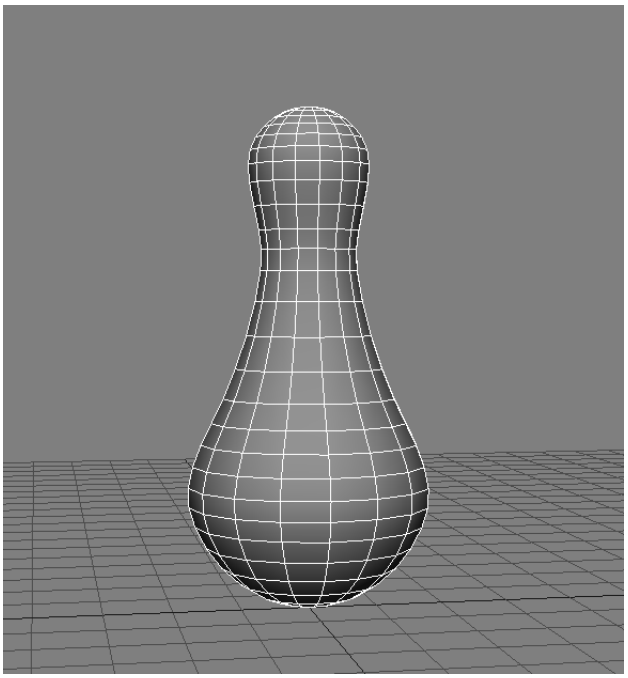
In the render properties you can turn on the option `Output` – `Passes` – `Assign Unique Pass Id`. After that, all the objects in the scene will automatically be assigned different values of the `Pass ID`.



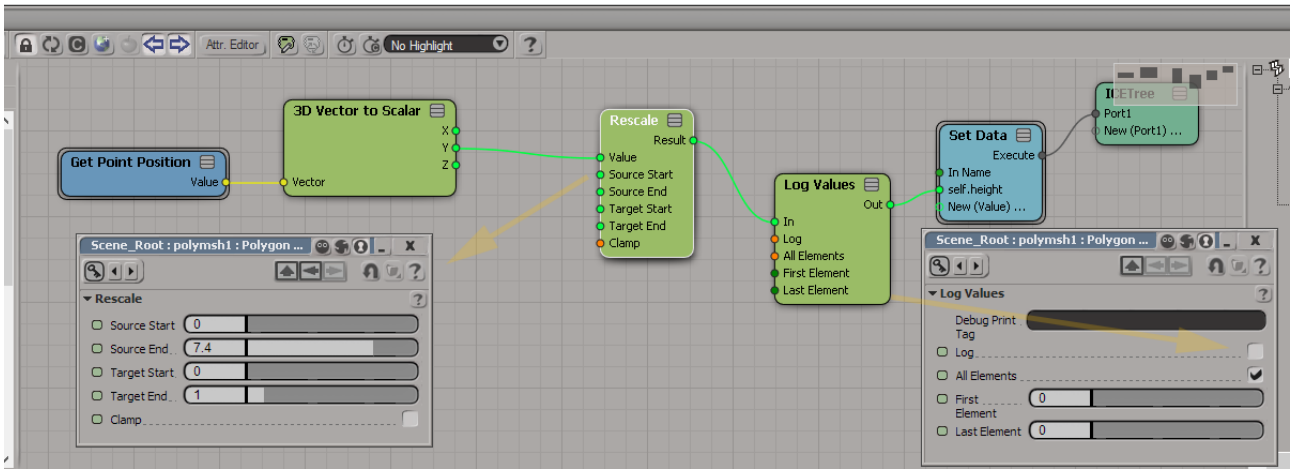
The output of the channel `Material ID` works in a similar way. It always have different values for different materials.

11 How to render ICE-attributes

Suppose we have a scene with a simple object.

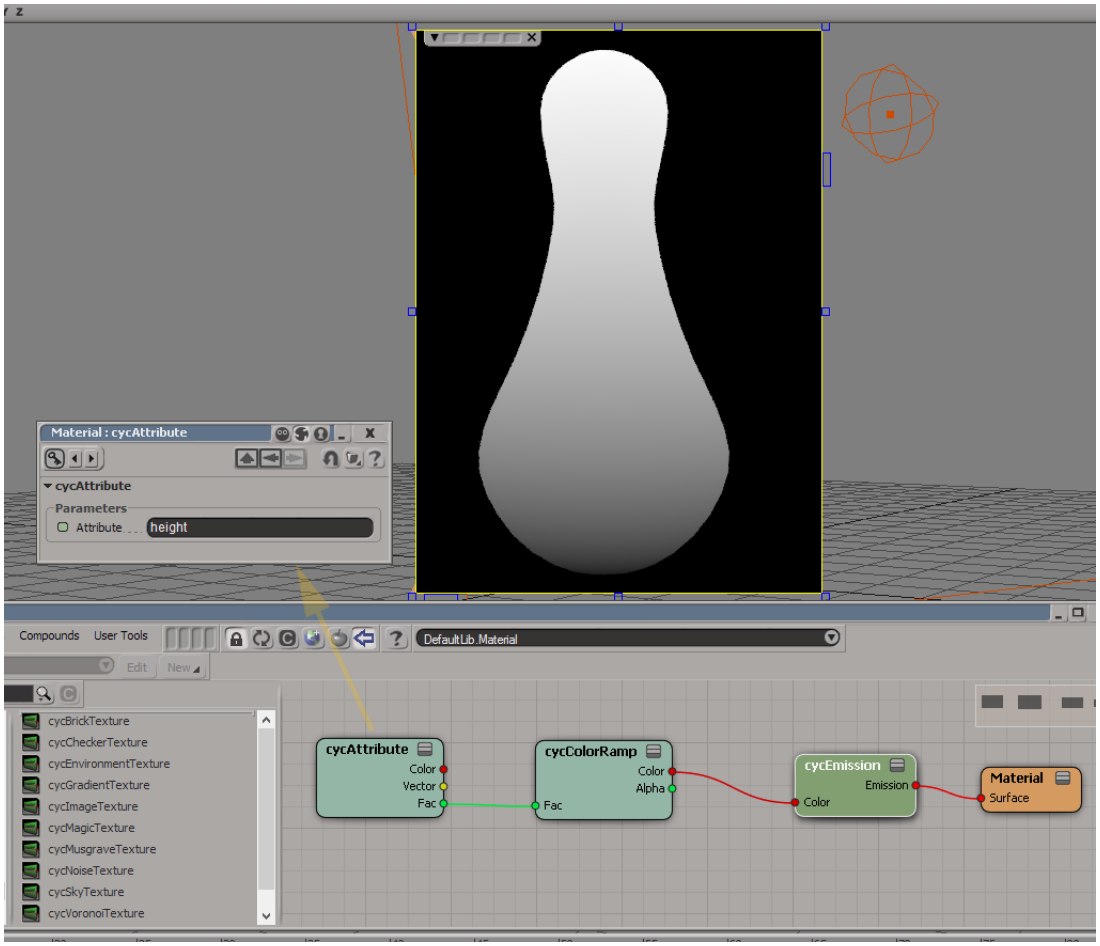


This object contains the following ICE-tree:



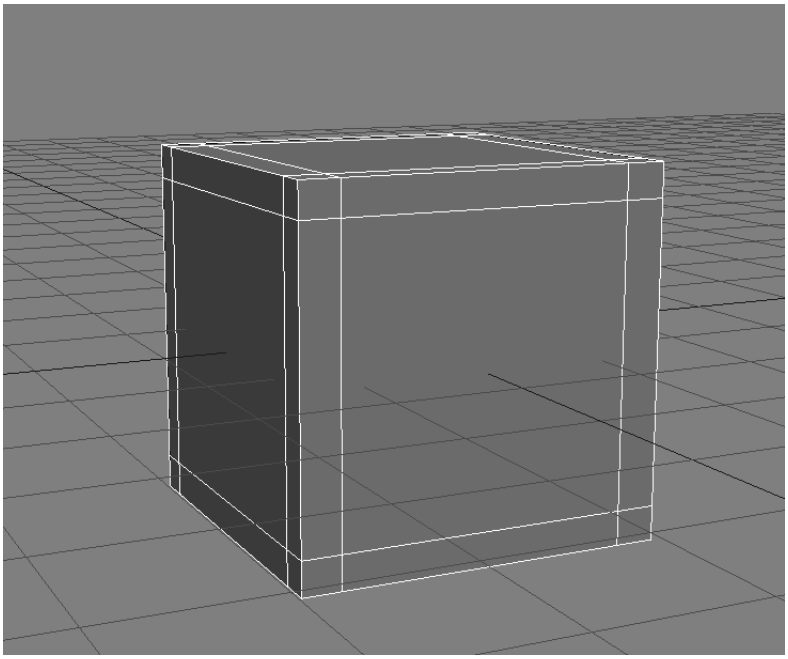
This ICE-tree writes to each vertex a value from 0 to 1, depending on how high the vertex is above the ground. In the node `Rescale` the value `7.4` is the average height of the object. All values are stored to the `height` attribute. The node `Log Values` is used to force the ICE to calculate values. The ICE is very clever, and if it think that the attribute is not used anywhere, it often does not calculate it. By the node `Log Values` we force to do all calculations, even with the `Log` option disabled.

Assign to our object `Emission`-shader. Add the node `Attribute` and set the name of the ICE attribute (`height` in our case). Finally path the `float`-output of this node through `Color Ramp` and connect everything to the port `Color`. On the render we see, how the object's color changes from black to white from the bottom to top.

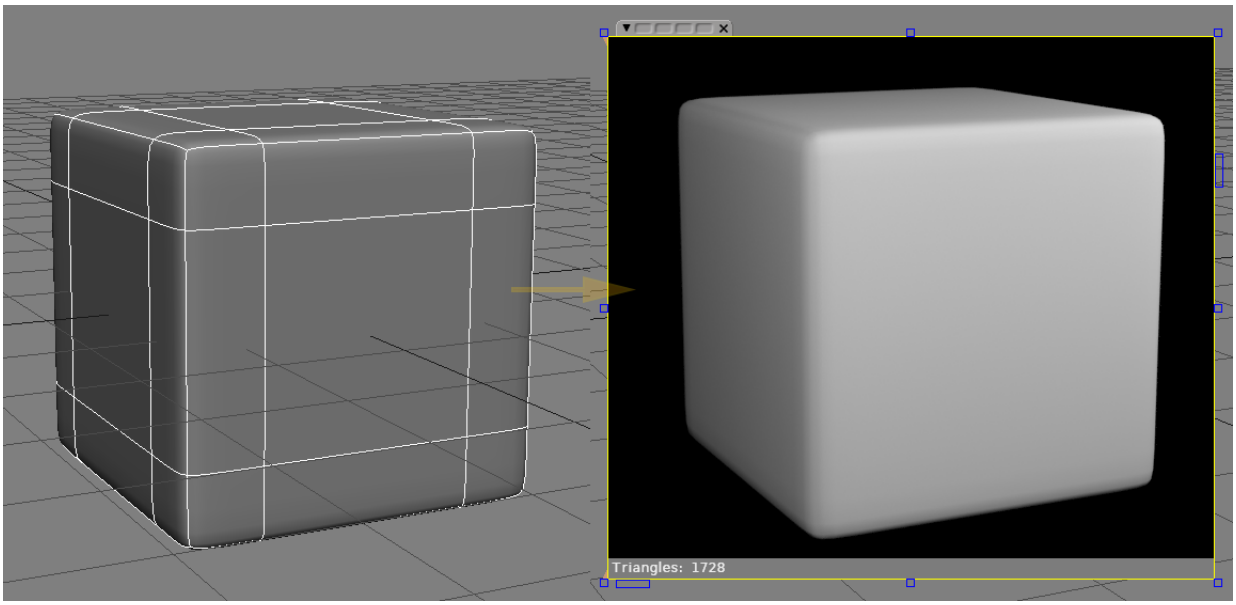

Material : cycAttribute
cycAttribute
Parameters
Attribute
height
Compounds
User Tools
DefaultLib.Material
Edit
New
cycBrickTexture
cycCheckerTexture
cycEnvironmentTexture
cycGradientTexture
cycImageTexture
cycMagicTexture
cycMusgraveTexture
cycNoiseTexture
cycSkyTexture
cycVoronoiTexture
cycAttribute
Color
Vector
Fac
cycColorRamp
Color
Alpha
Fac
cycEmission
Emission
Color
Material
Surface

12 How to render subdivide surfaces

Suppose we have a cube.

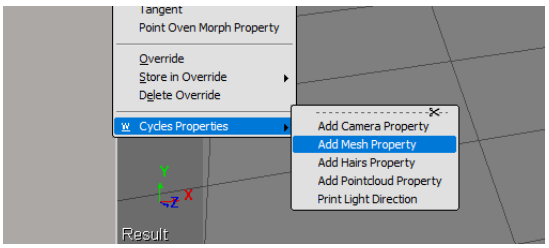


If you press plus on the keyboard 3 times, the cube will become smoothed. And on the render it is exactly the same.

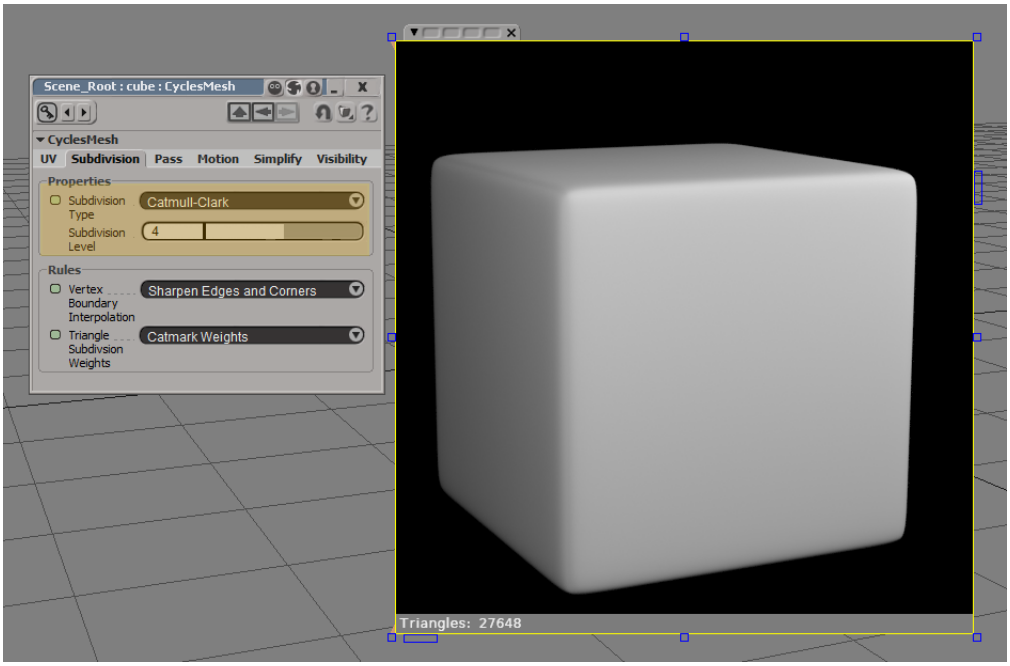


If you would like to do a subdivision of the object only during the render, then assign the property `CyclesMesh` to the object by choosing the command

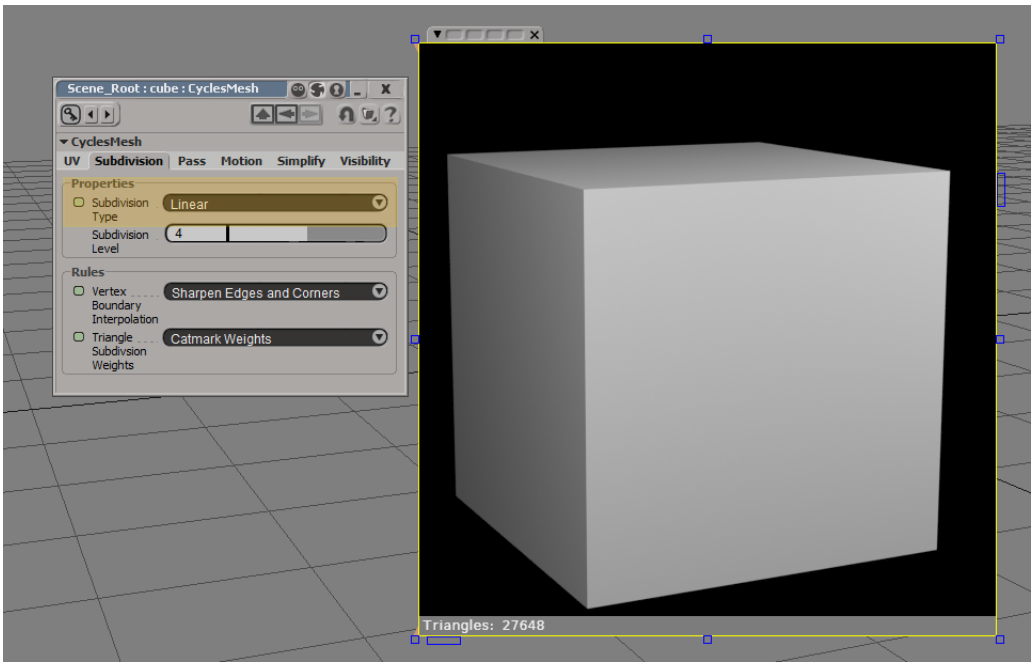
`Property` – `Cycles Properties` – `Add Mesh Property`.



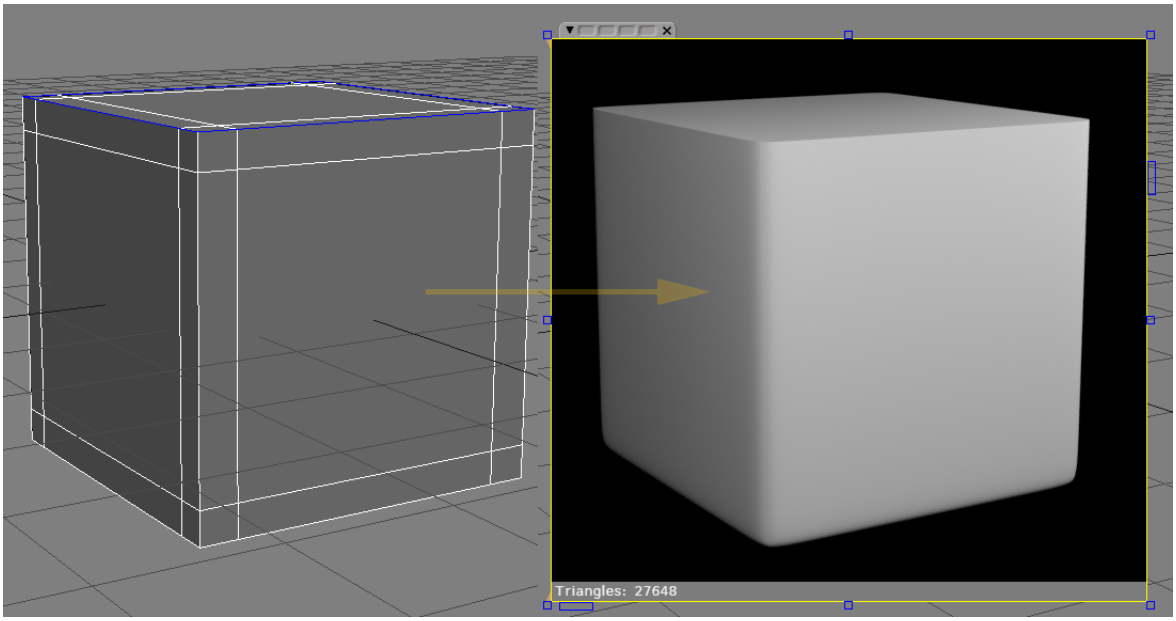
On the tab `Subdivision` choose the type `Catmull-Clark` and set the number of subdivision steps equal to `4`. Render it.



If you select the type `Linear`, then the polygons will be subdivided, but the form of the object remains the same. Useful when using displacement.



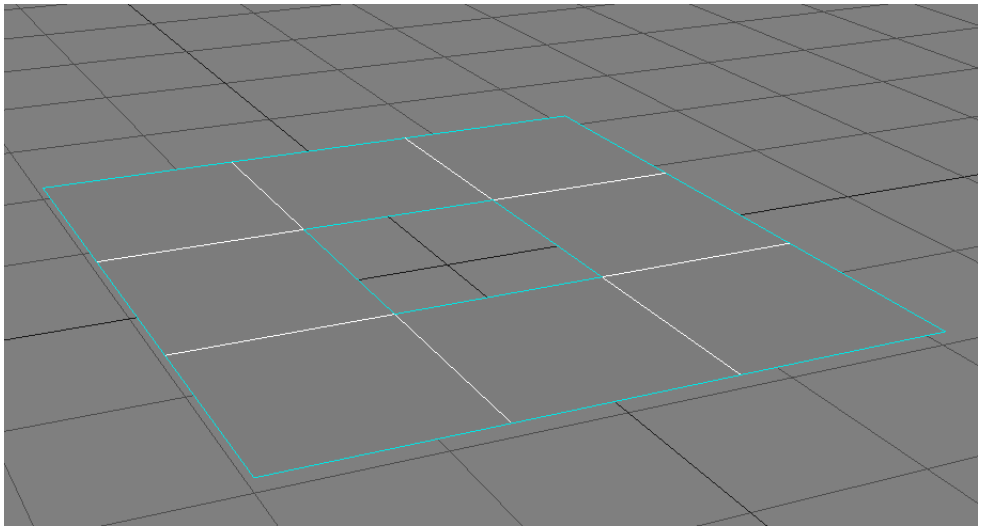
An object can have hard edges, but even in this case the result on the render is correct.



Properties have two additional options:

`Vertex Boundary Interpolation` and `Triangle Subdivision Weights`.

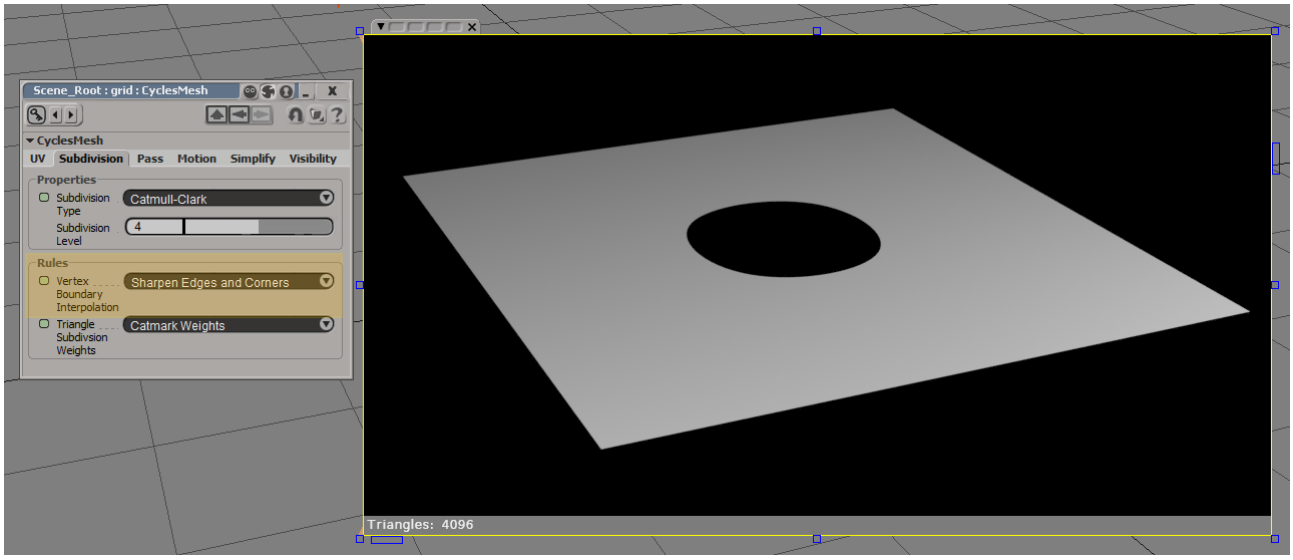
The first option `Vertex Boundary Interpolation` define how to smooth the edges and vertices on the boundary. Consider an example. Take a plane with a hole in the middle.



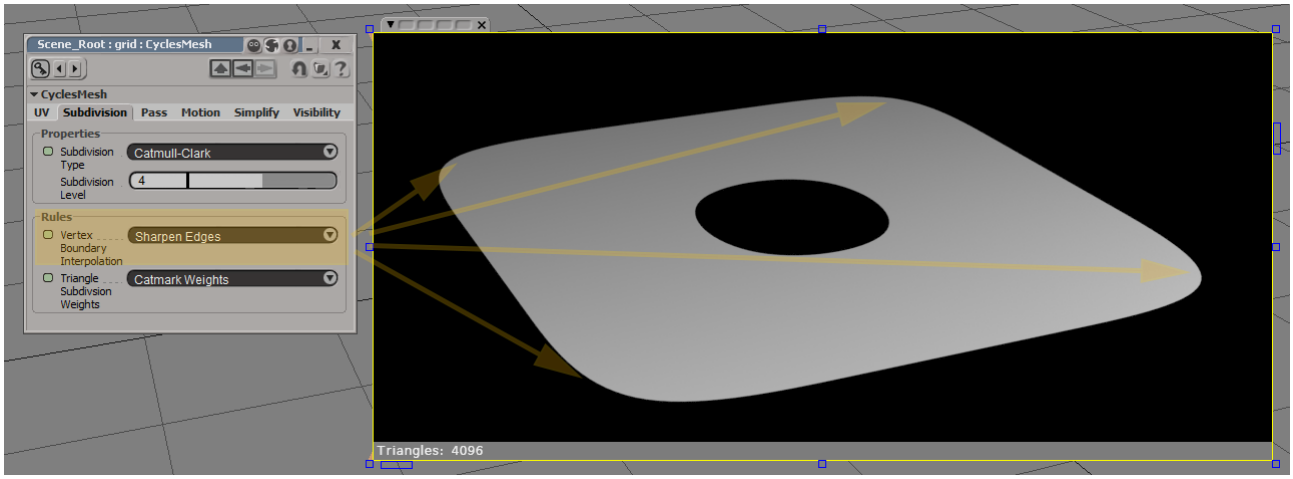
If the value of the parameter `Vertex Boundary Interpolation` is equal to

`Sharpen Edges and Corners`,

then both edges and vertices will be snapping to the boundary of the original mesh.



If the value of this parameter is equal to `Sharpen Edges`, then corners will be smoothed.



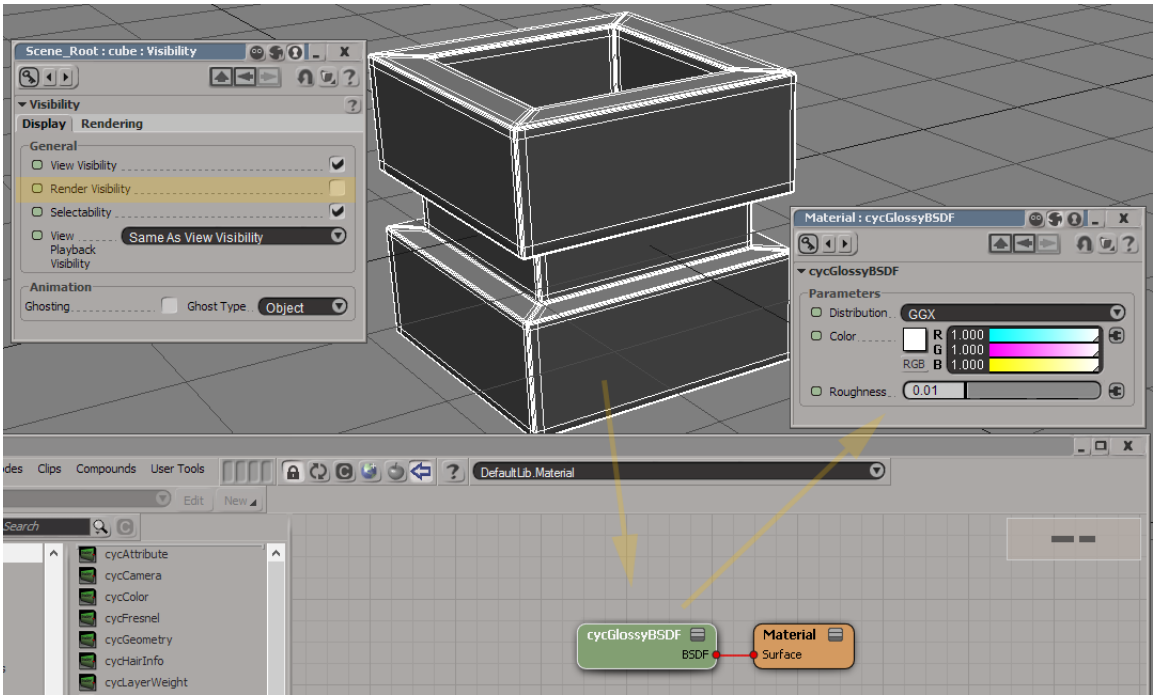
Parameter `Triangle Subdivision Weights` influences the way of subdividing the triangular polygons of the model. The difference between modes

`Catmark Weights` and `Smooth Triangle Weights`

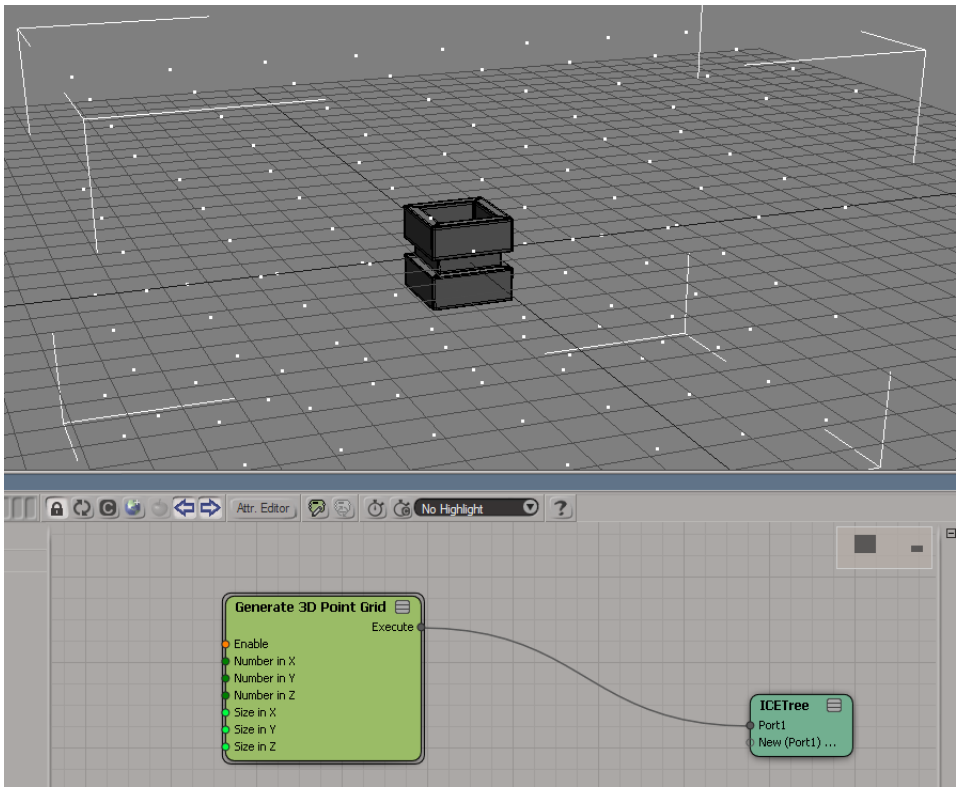
in most cases quite small.

13 How to render ICE-instances

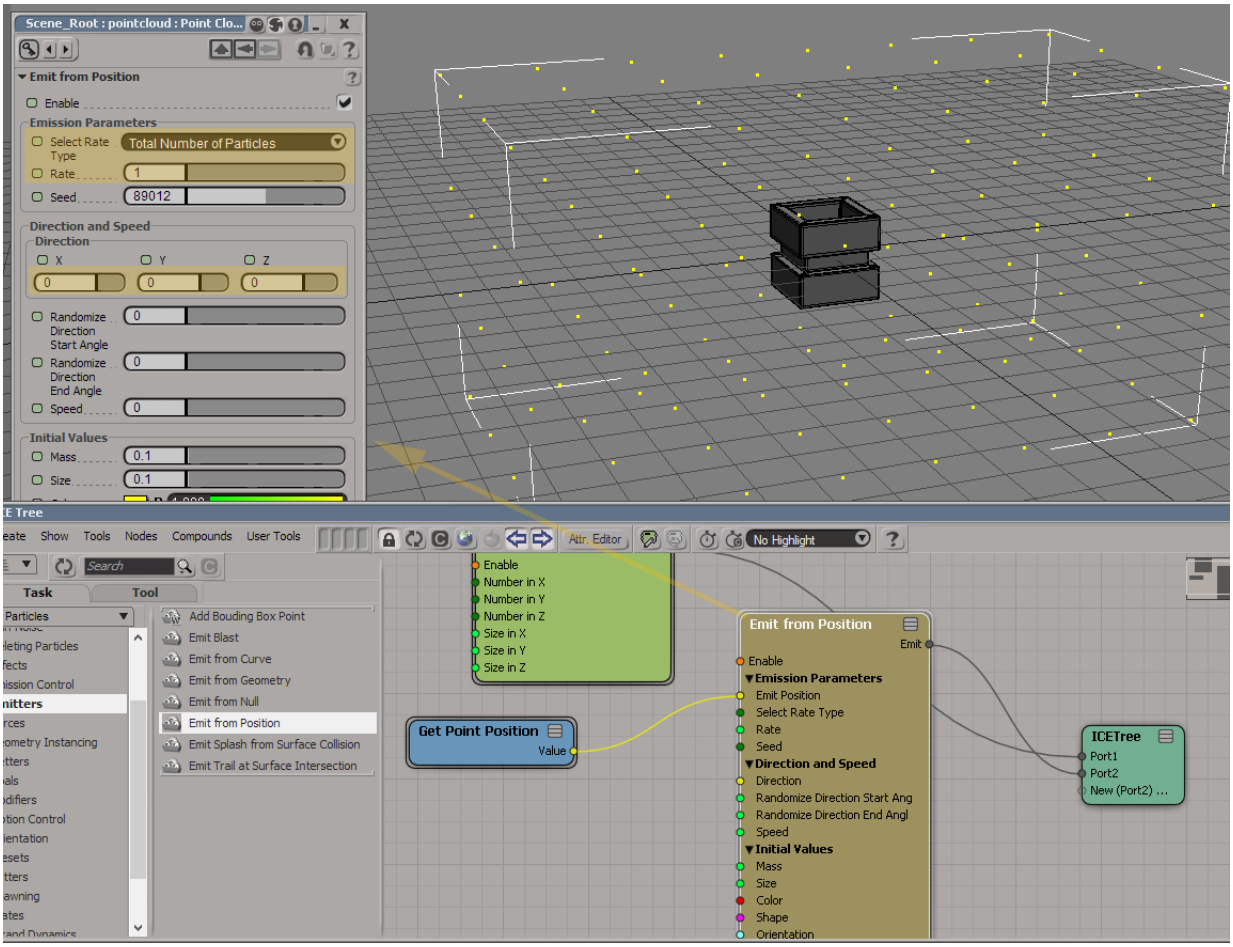
There are no features here. Everything works just like for Mental Ray. Suppose we have an object that we want to instantiate. It has some shader and exclude from visibility on render.



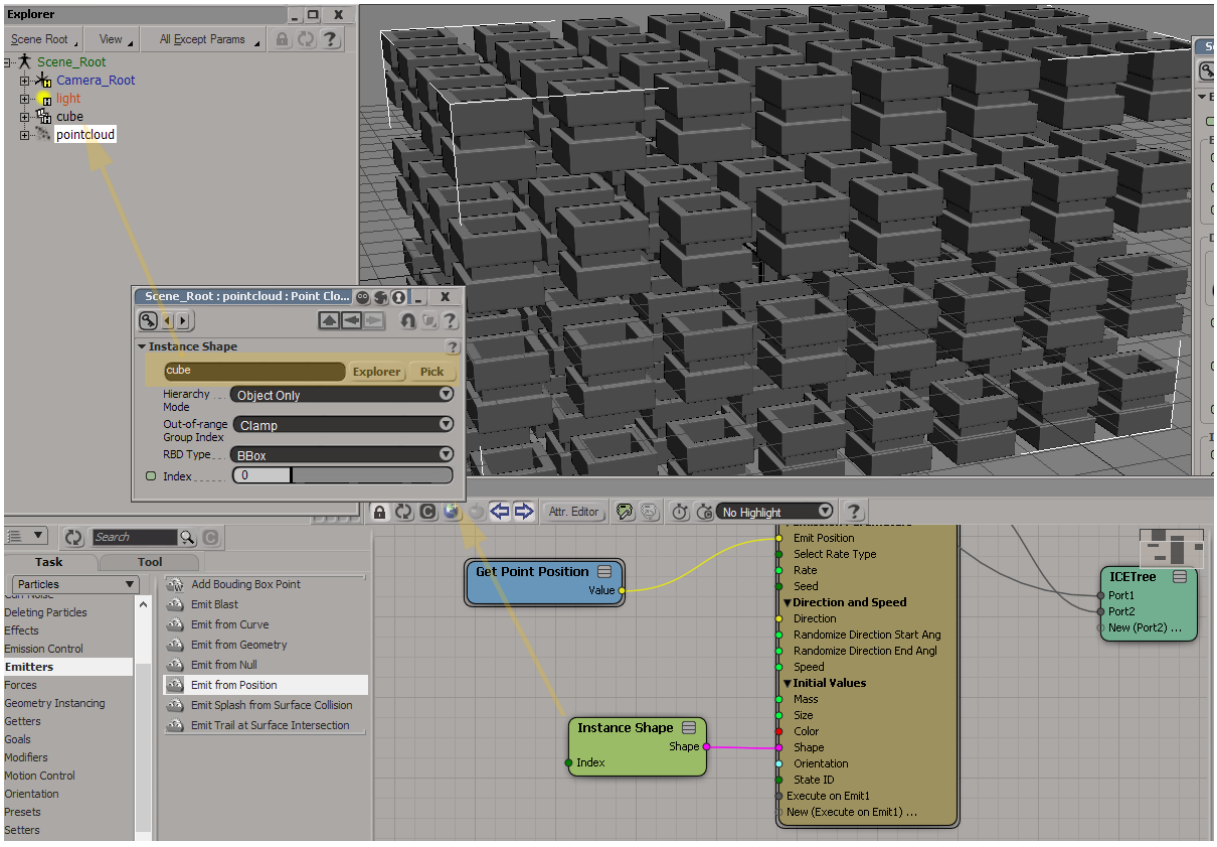
Create an empty `Point Cloud`, an ICE-tree and generate a cloud of points.



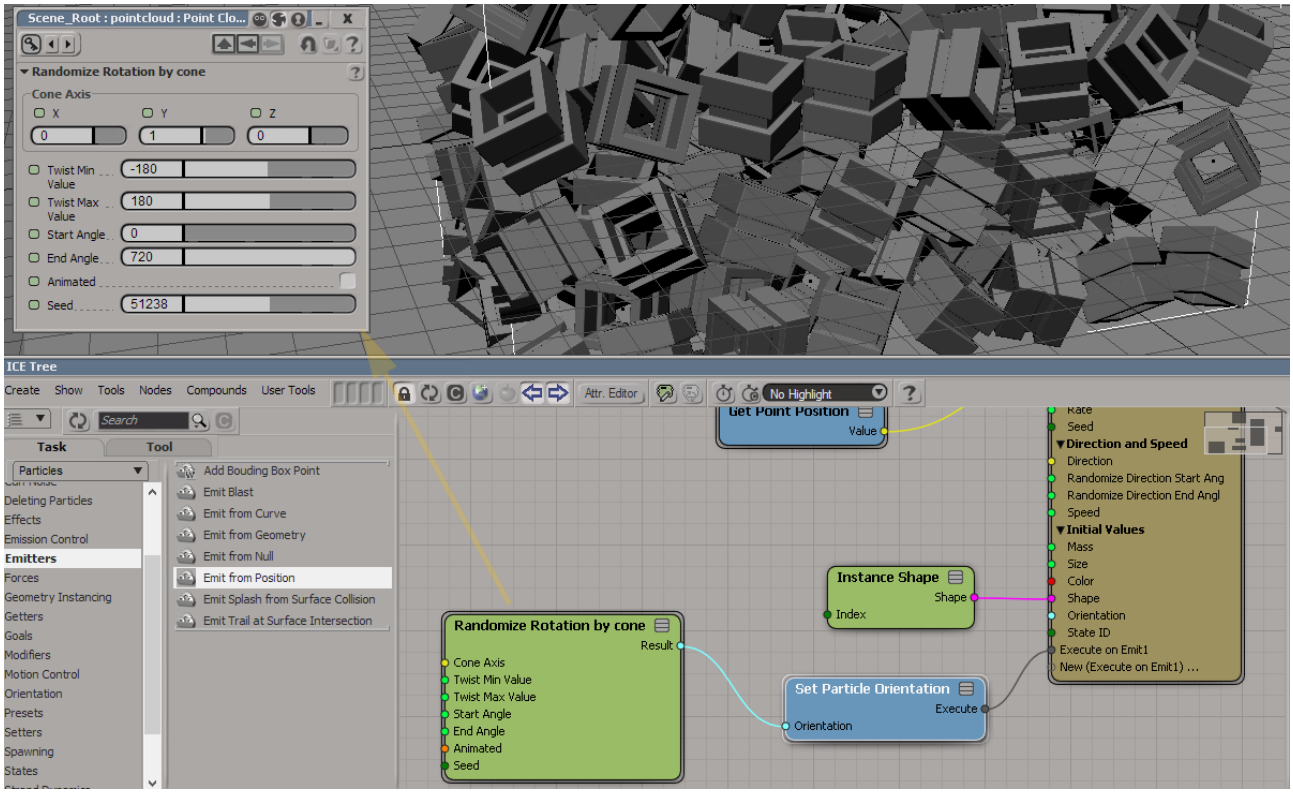
Next add the node `Emit from Position`. Set parameters as in the picture and connect the node `Get Point Position` to the port `Emit Position`. This will generate particles just in the places where the grid points were.



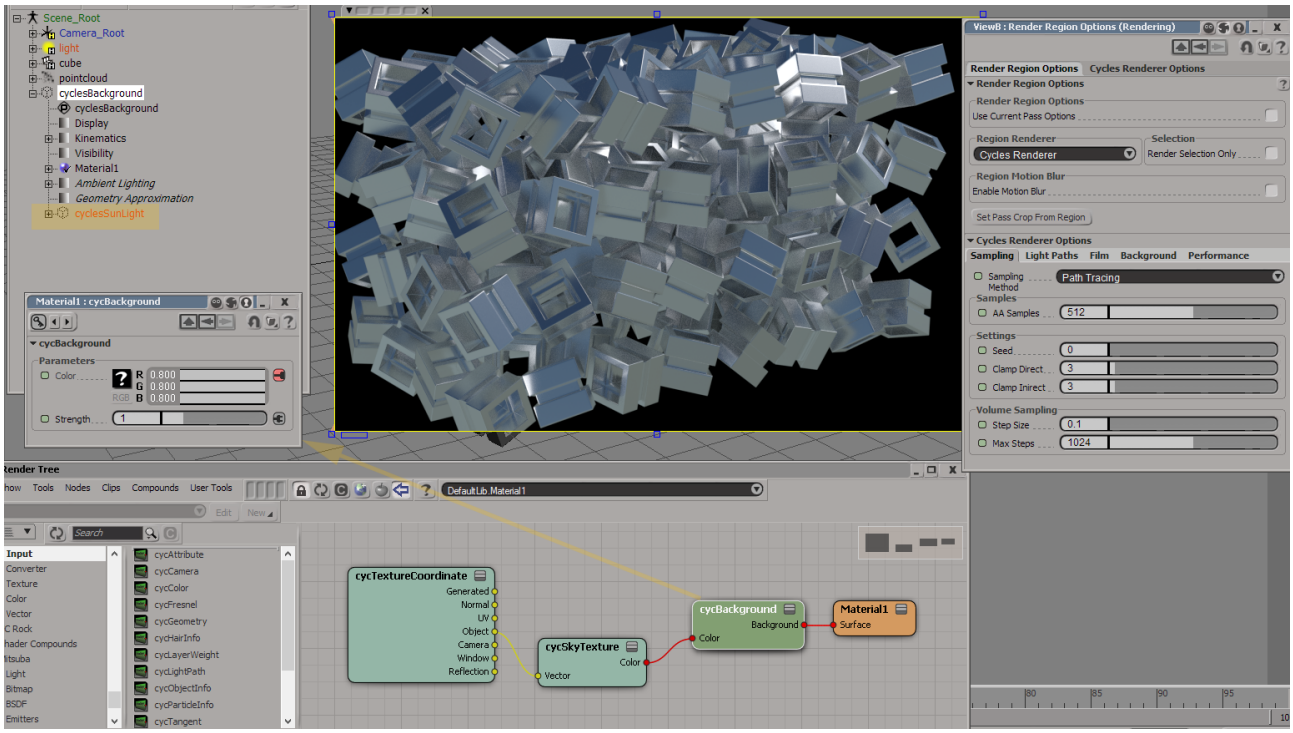
Instead of particles, you can use primitives like `Disc`, `Rectangle`, `Sphere`, `Box`, `Cylinder`, `Cone` and `Capsule` (looks like a `Sphere`). You can also use any polygon object. Using the node `Instance Shape` set our master object.



Add a bit of randomness to the orientation of the instances.

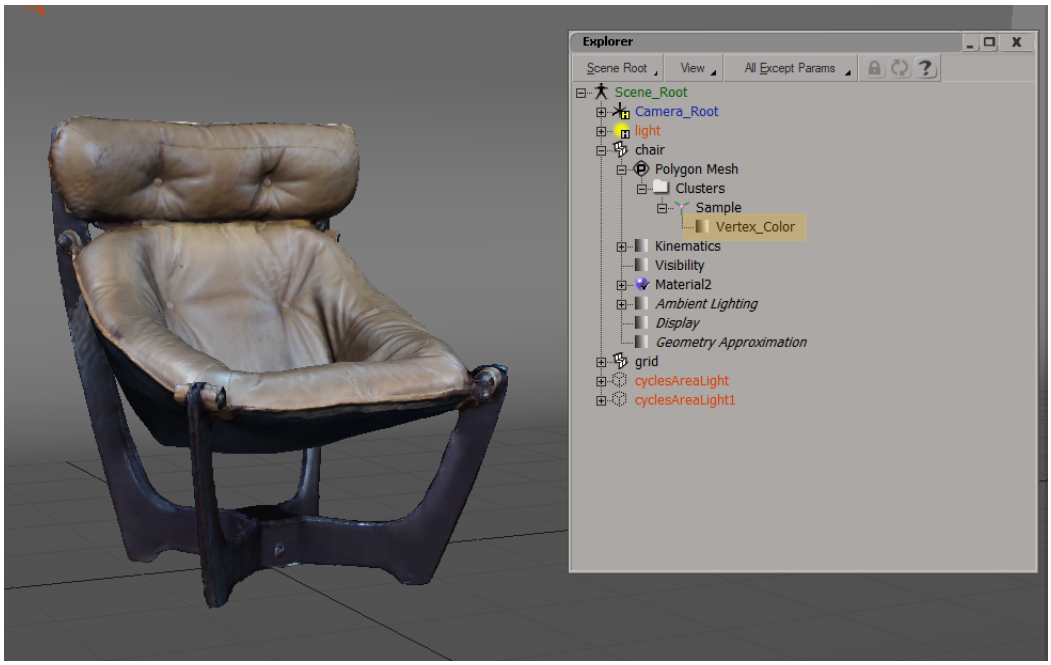


Finally, create a light source of the type `Background`, add a light source of the type `Sun` as a child object. To the background shader add the node `SkyTexture`. Render.

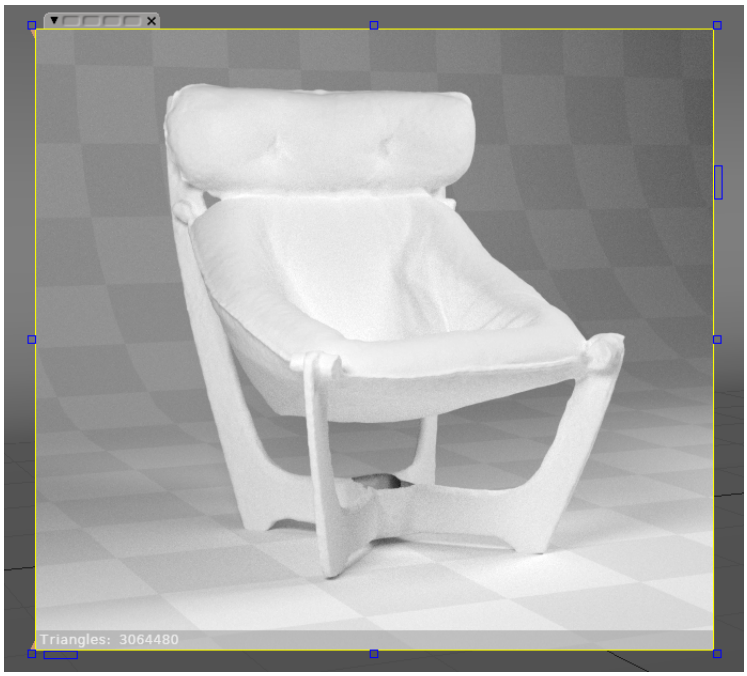


14 How to render Vertex Color

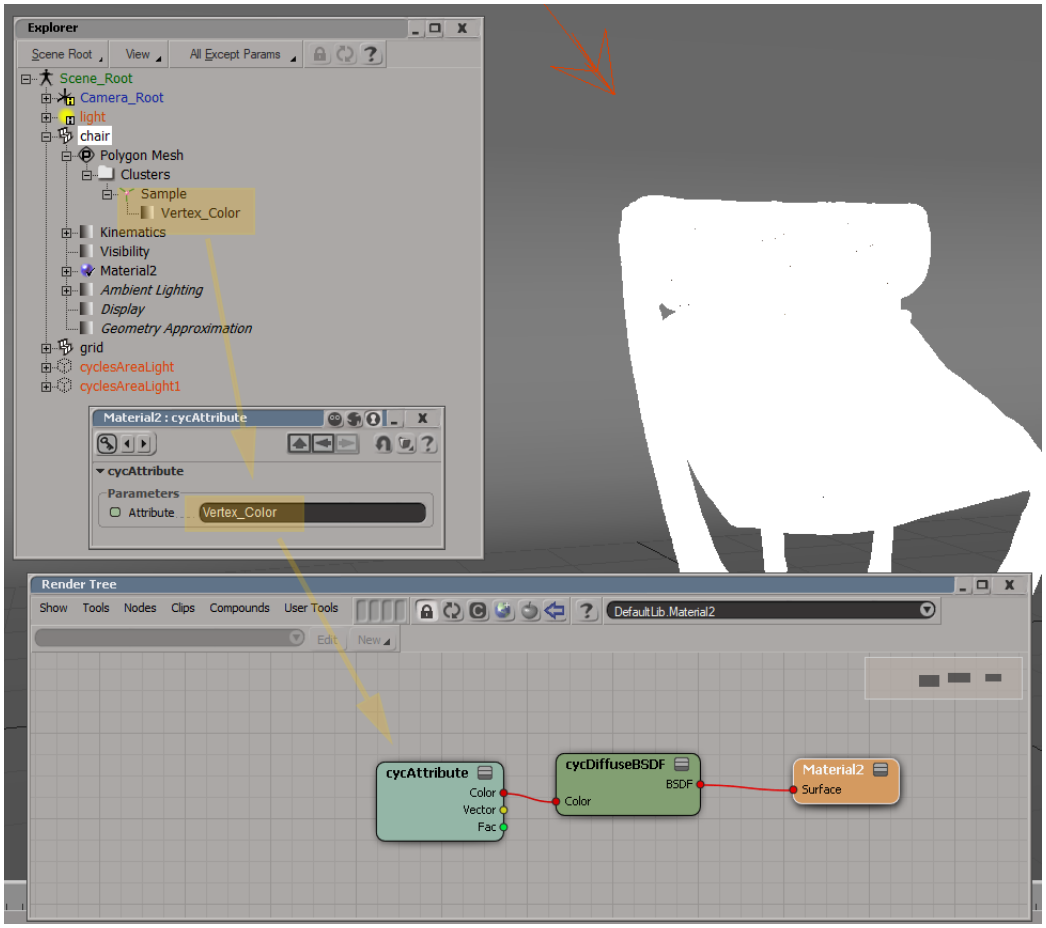
Suppose we have a scene with an object that contains Vertex color.



On the render this object has a neutral color, since `Cycles` does not know how it should use vertices' colors.



Add to the material the node `Attribute` and set the name of the cluster, which containing the vertex colors data (`Vertex_Color` in our case).

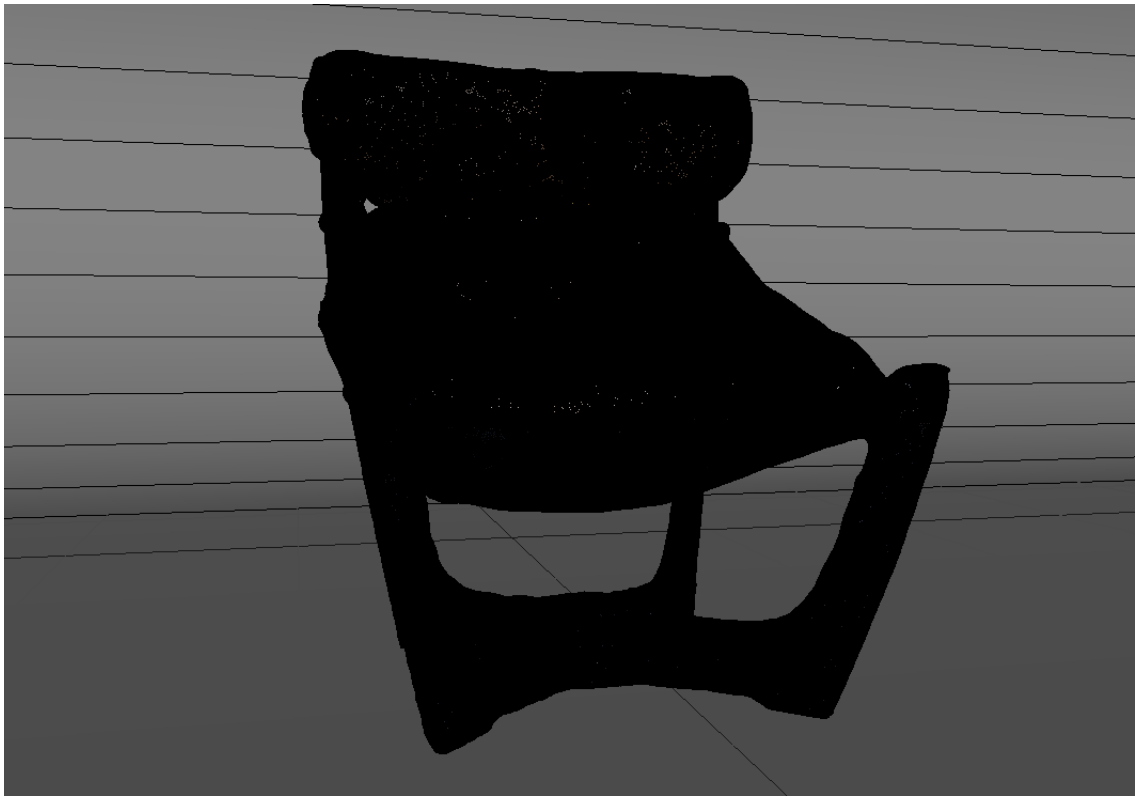

Explorer
Scene Root
View
All Except Params
Scene_Root
Camera_Root
light
chair
Polygon Mesh
Clusters
Sample
Vertex_Color
Kinematics
Visibility
Material2
Ambient Lighting
Display
Geometry Approximation
grid
cyclesAreaLight
cyclesAreaLight1
Material2 : cycAttribute
cycAttribute
Parameters
Attribute
Vertex_Color
Render Tree
Show
Tools
Nodes
Clips
Compounds
User Tools
DefaultLib.Material2
Edit
New
cycAttribute
Color
Vector
Fac
cycDiffuseBSDF
BSDF
Color
Material2
Surface

Render.

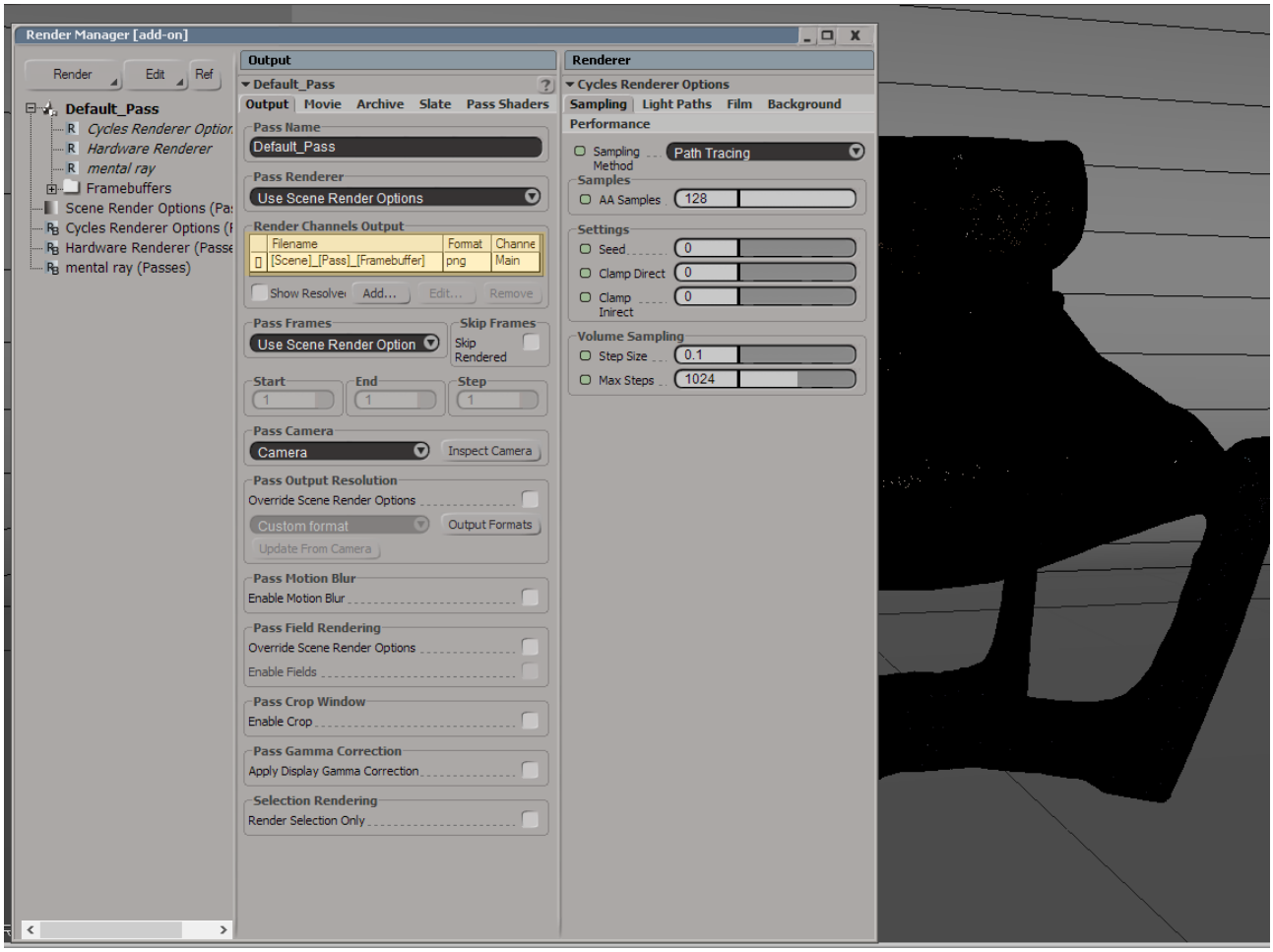

Triangles: 3064480

15 How to render to Multi-Layer EXR

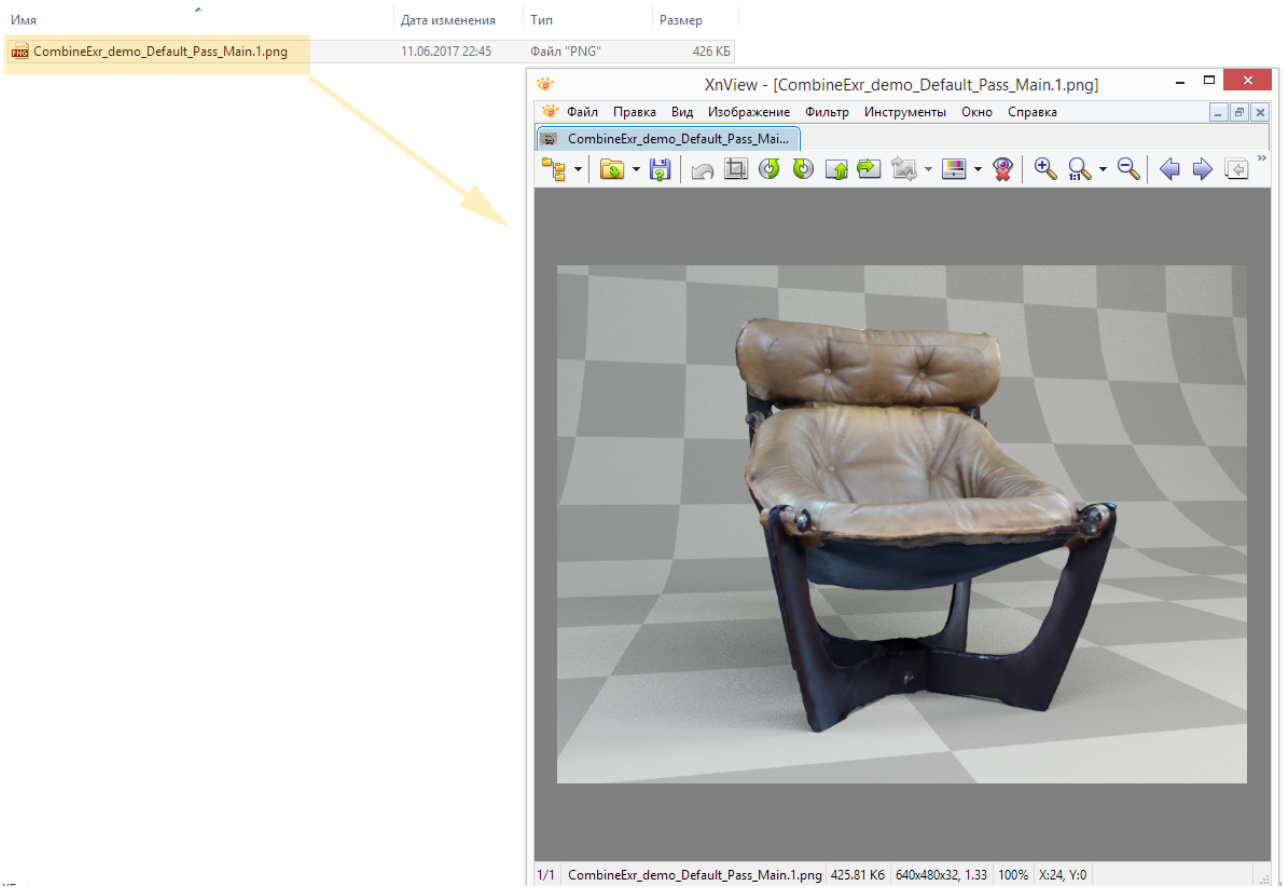
Suppose we have a scene: an armchair in a simple studio.



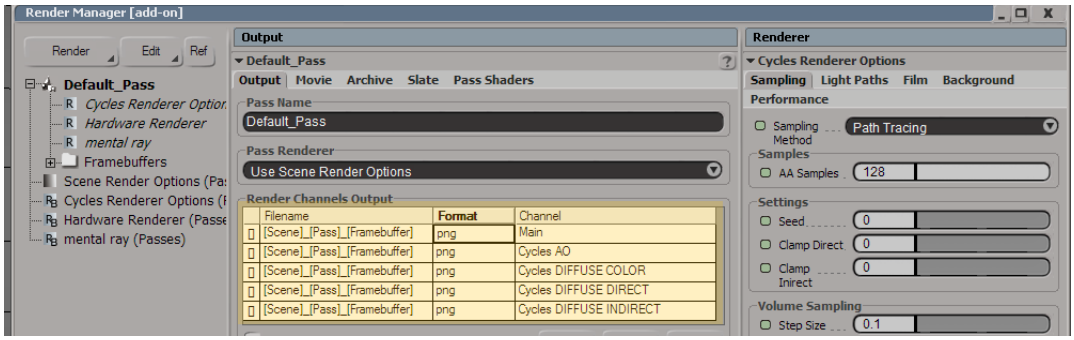
Select in the `Render Manager` one output channel and save it in `*.png` format.



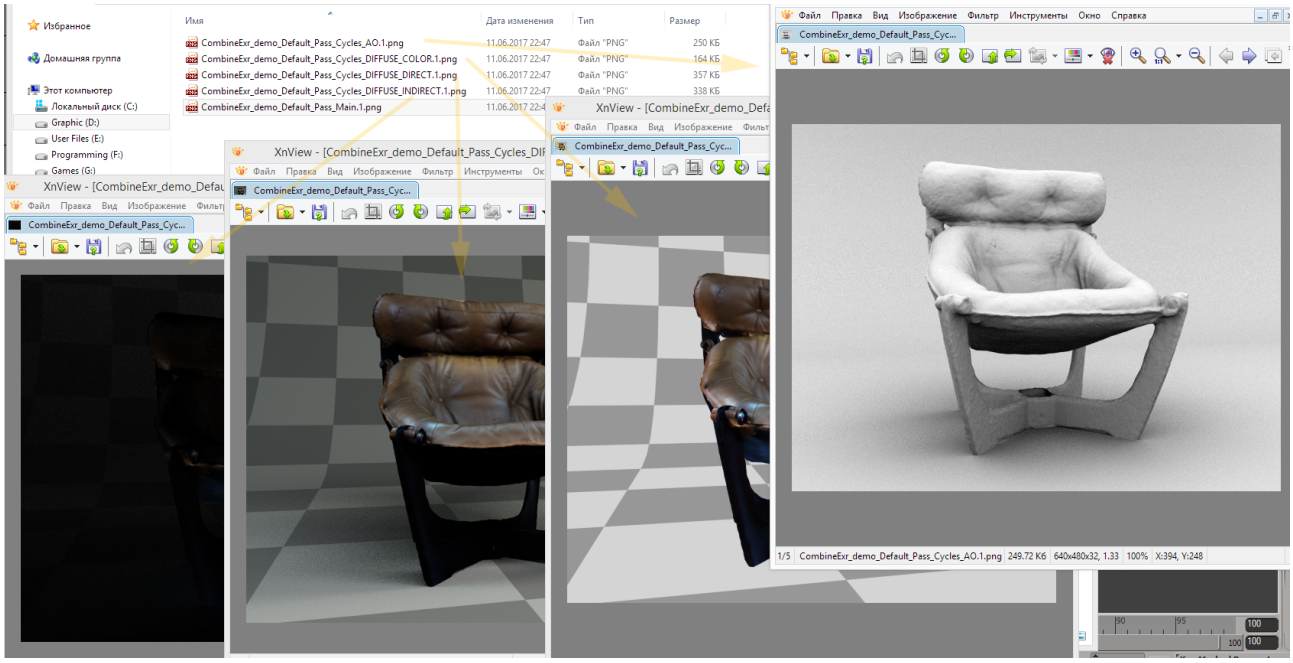
As a result we obtain one png-file.



Next add additional channels to the render. We will also save them in `*.png` format.

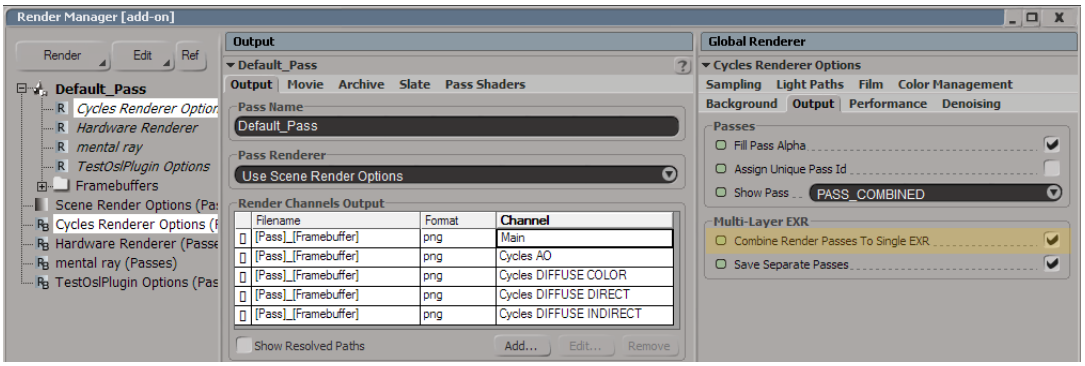


As a result we get a set of files, each with its own data.

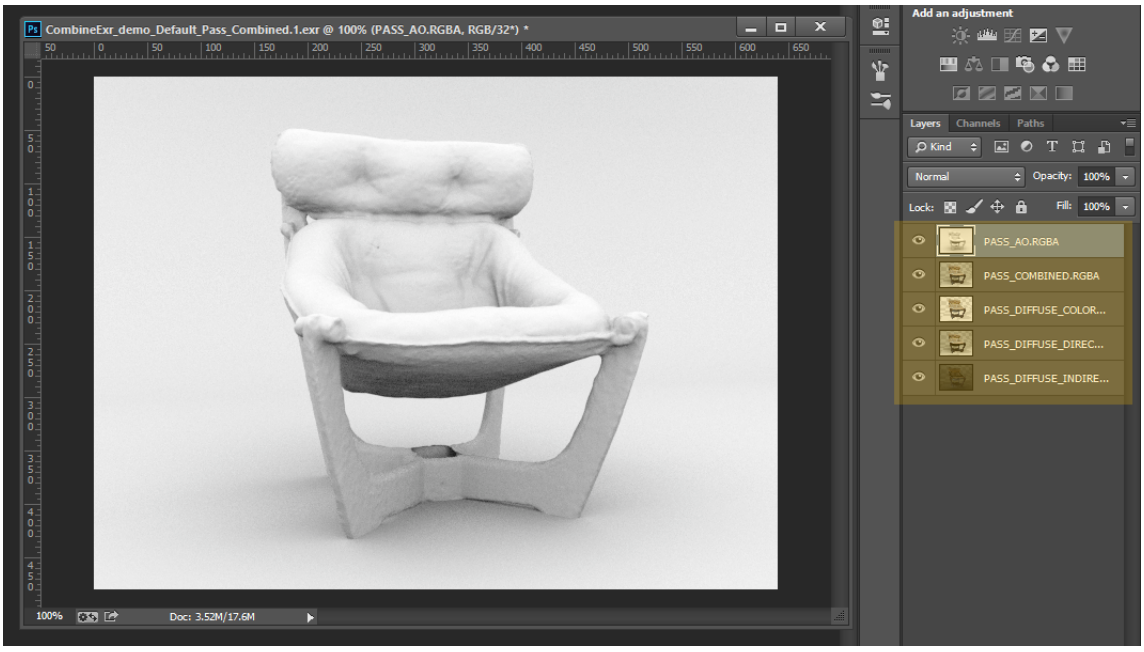


All selected for the render channels can be saved to a single layered 32-bit exr-file. For this, in the section `Output` – `Multi-Layer EXR` turn on the parameter

`Combine Render Passes To Single EXR`.

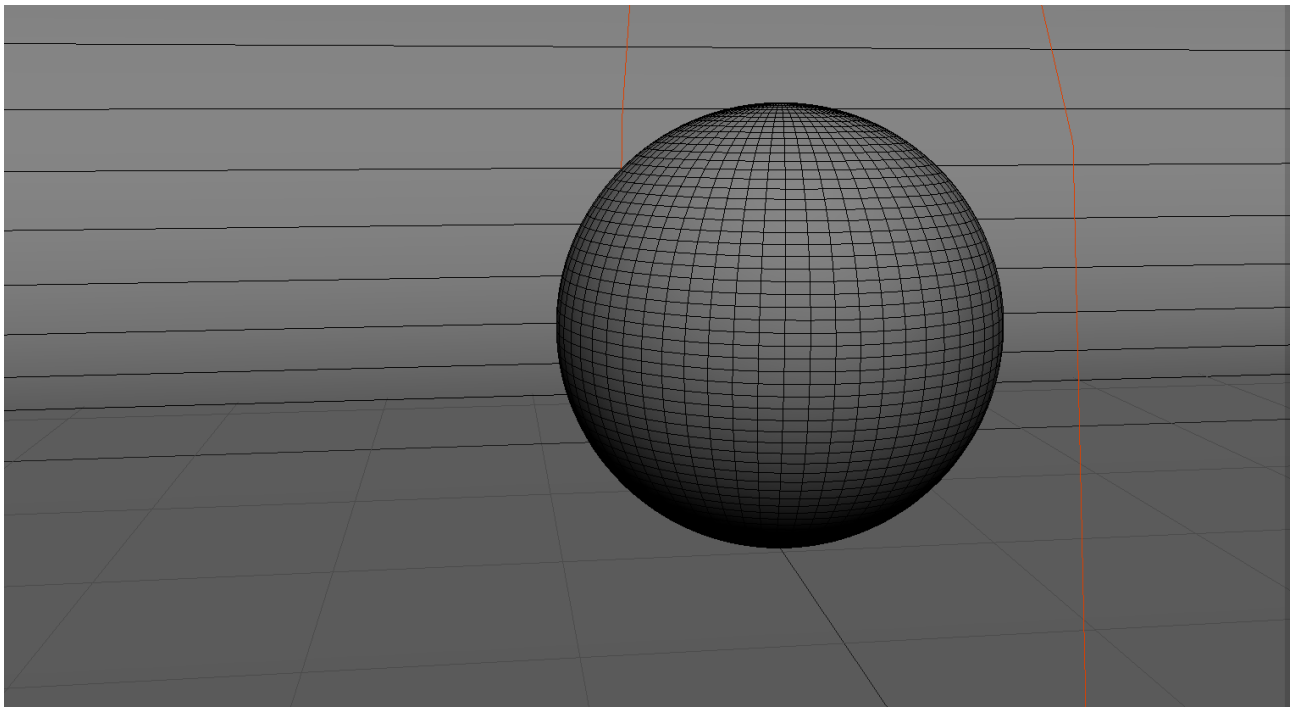


As a result an exr-file with all selected channels will be created additionally. If you turn off the `Save Separate Passes` option, then no other files other than this layered file will be saved.



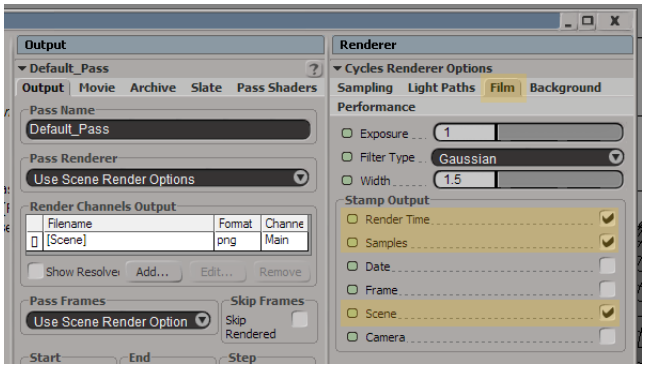
16 How to use Stamp Output

Suppose we have a scene: a glass sphere in a simple studio.

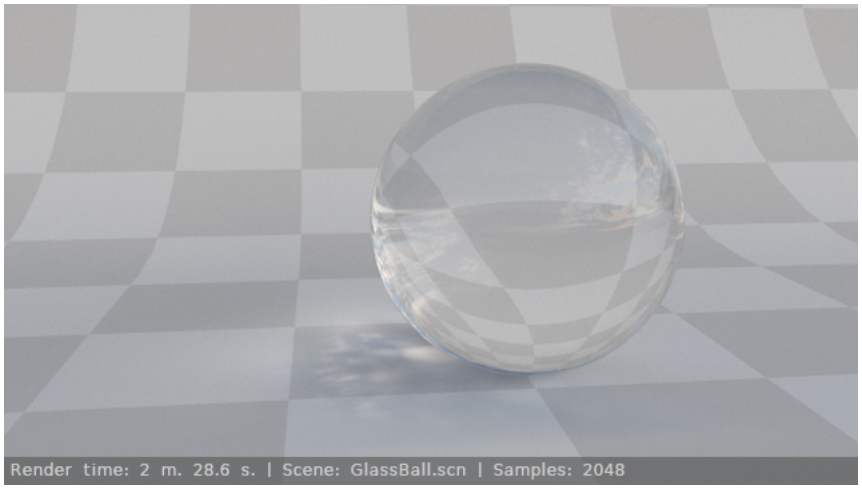


As a result, the rendered image is the following:

In the section `Film` – `Stamp Output` of the render properties turn on the options `Render Time`, `Samples` and `Scene`.



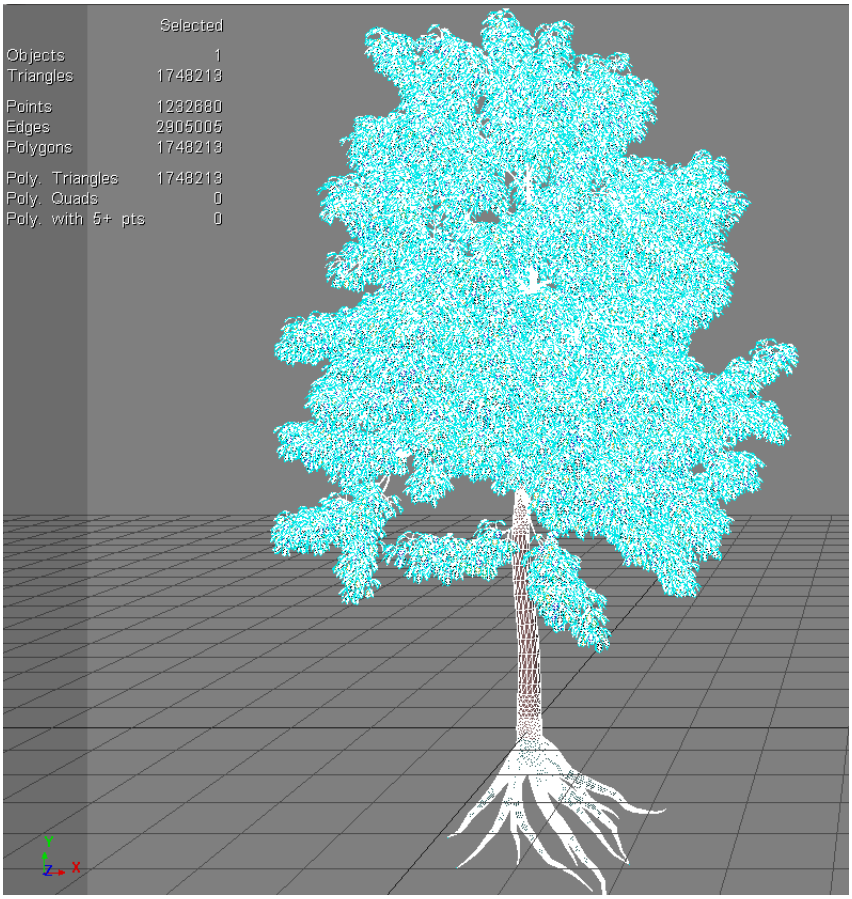
As a result, a strip with information about the selected parameters will be added to the rendered image.



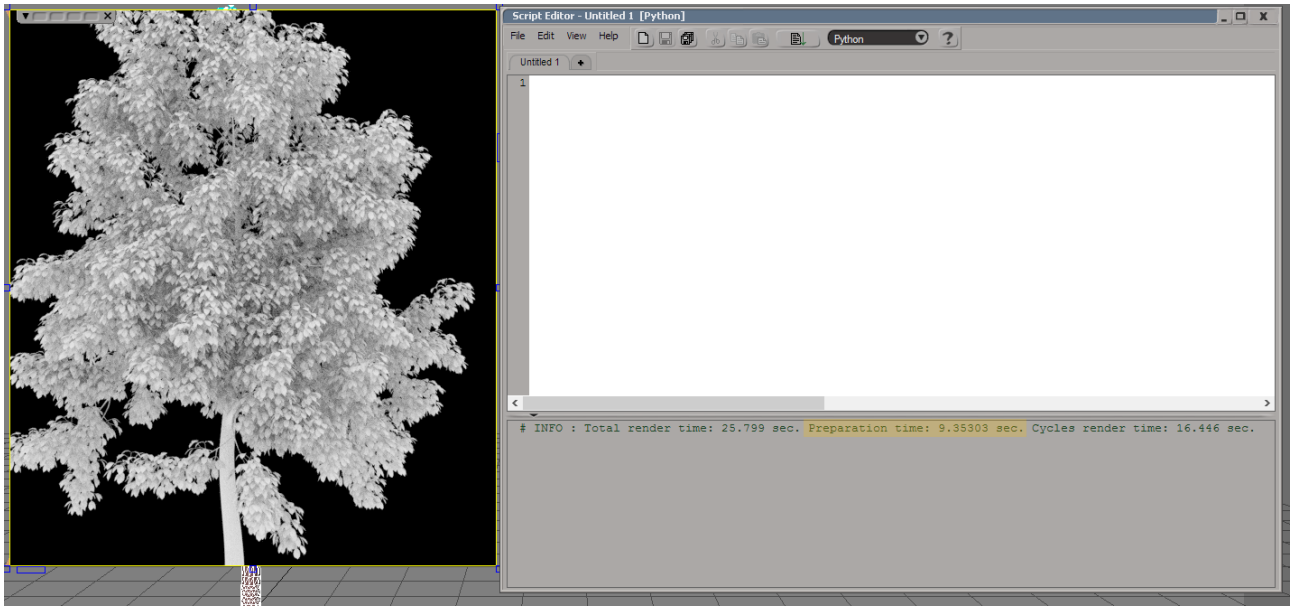
The meaning of the parameters is as follows:
`Render Time` – the render time of the image;
`Samples` – the number of AA samples in the render settings;
`Date` – date of rendering (day, month, year and time);
`Frame` – the number of the rendered frame;
`Scene` – the name of the scene;
`Camera` – the name of the camera from which the render takes place;
`Triangles` – the total number of triangles in all polygonal object of the scene;
`Curves` – the total number of curves in all hair objects of the scene;
`Objects` – the number of objects on the scene;
`Lights` – the number of light sources on the scene.

17 How to use Cache

Suppose we have a complex object. For example, a tree.

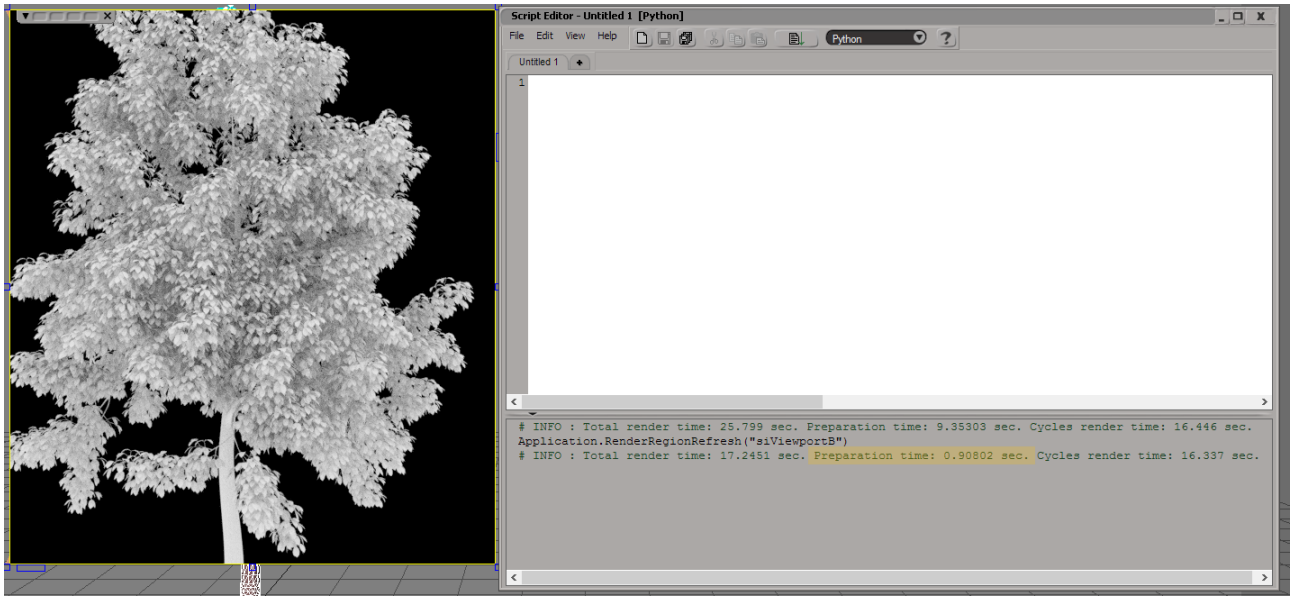


Render it and look at the log.



It says that the total render time is 25 seconds, of which 9 seconds is the preparation of the render. Preparation includes the collection data about the scene, what objects are on it, what shaders are assigned to them, and the longest time is export of all geometry of the scene to a format that the rendering engine understands. As a matter of fact all 9 seconds was spend to export geometry of a tree.

Do not touch anything and just click to update the render.



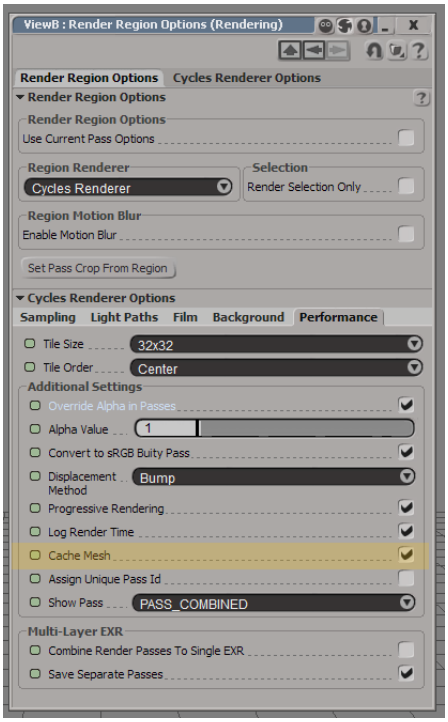
Now the preparation time for the scene took less than a second. For this, the cache is intended. After each render, all the data about the objects exported to the engine is saved, and if the object is to be rendered at the next time, the saved data is used instead of creating new ones. The cache is not used in the following cases:

1. The object changes the number of vertices or polygons;
2. The number of subdivs or subdivision type is changed (from linear to Catmull-Clark or vice versa);
3. The index of used uv-coordinates map is changed;
4. The object shader requires an attribute that was not stored in the cache;
5. A full pass rendering.

In connection with these, it is necessary to remember all the time about the following:

1. If the object has a polygon cluster with new material, then the renderer will notice this only when the cache is updated;
2. If the object is deformed (by bones, for example), then the renderer will not update the location of the vertices;
3. If the shader requires an attribute that can not be generated (for example, texture coordinates are needed, but there is no ones on the object), the geometry export will happen every time again.

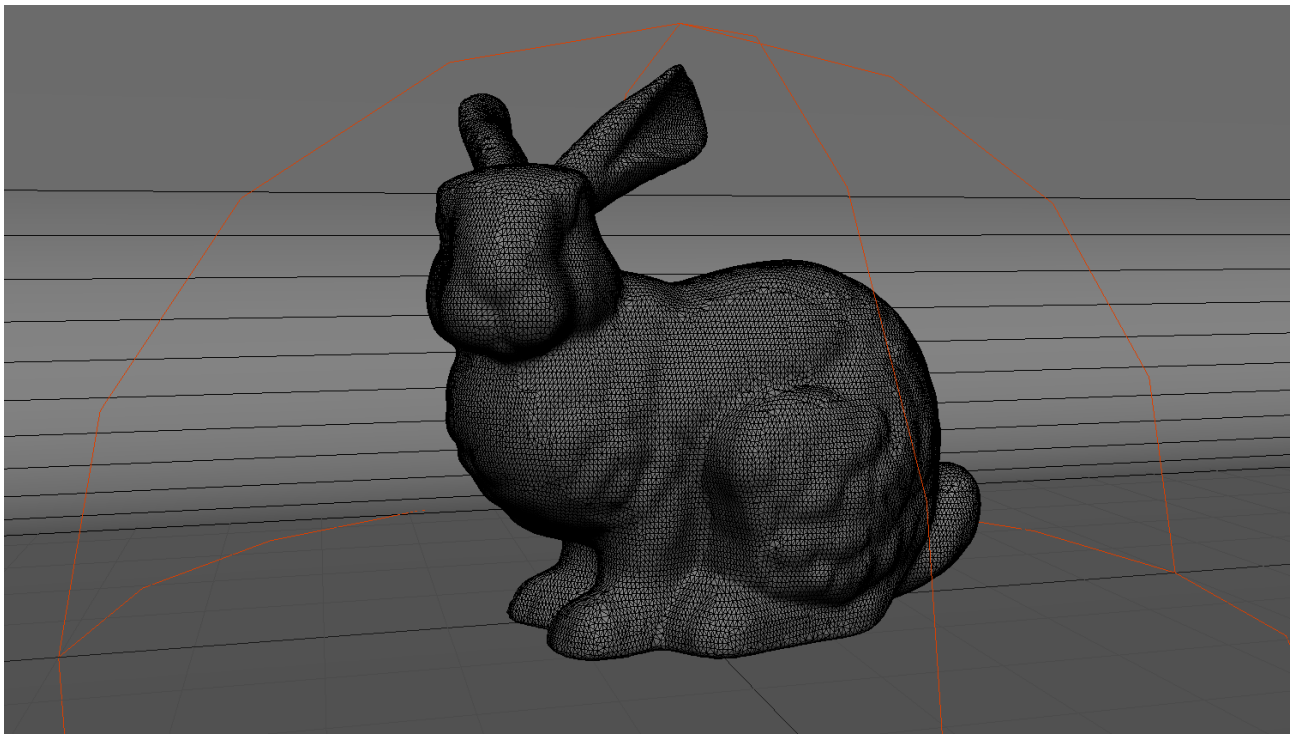
To force to export all geometry each time, it's enough to turn off the `Cache Mesh` parameter in the tab `Performance` of the render settings.



Since the re-creation of the cache occurs every time at the end of the rendering, the easiest way to update the cache of one particular object is to exclude this object from one render session, and then turn it back on.

18 How to use color profiles

Suppose we have a scene: a glass bunny in a simple studio.

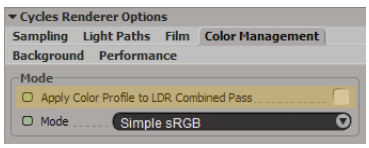


Render it.

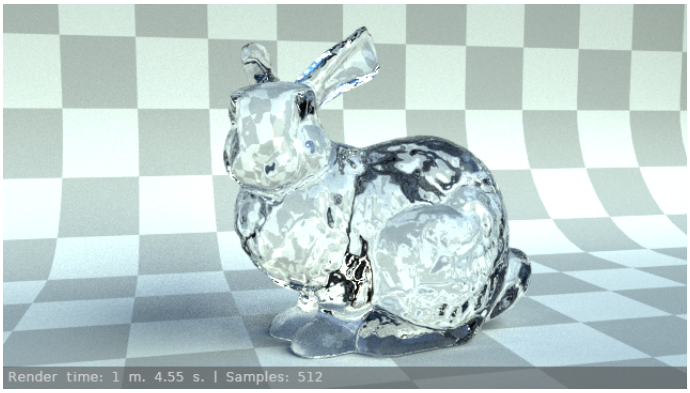
By default, if the content of the `Combined` pass is displayed in `Region preview` or stored in `* .png` format, then it converts the color space to sRGB. This option can be turned off by turning off the option

`Apply Color Profile to LDR Combined Pass`

in the tab `Color Management` of the render settings.



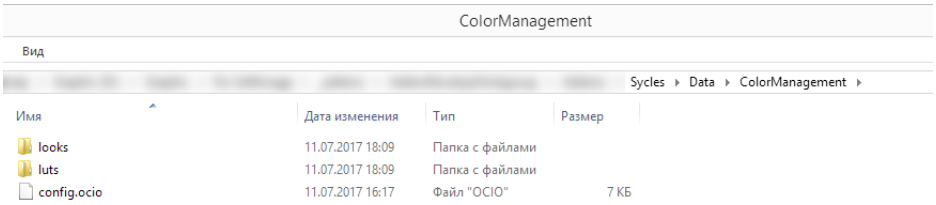
After the shutdown, the render result is stored in the linear space.



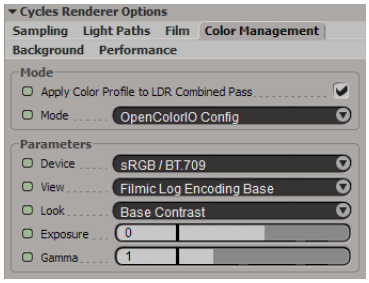
There is the possibility to use the color profiles `OpenColorIO`. To do this, to the folder

`...\Sycles\Data\ColorManagement\`

you should placed the file `config.ocio` and all connected with it files.

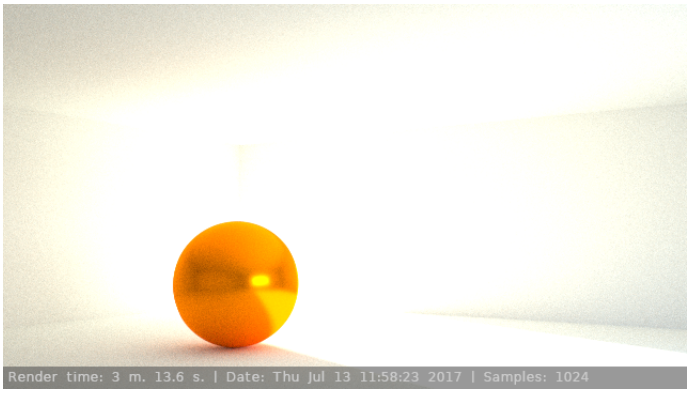


These profiles can be downloaded from the site `http://opencolorio.org/`. The add-on already includes profiles from the "Filmic Blender" add-on for Blender. To use them, select the value of the parameter `Mode` – `OpenColorIO Config`.



The rendering result is now obtained as follows:

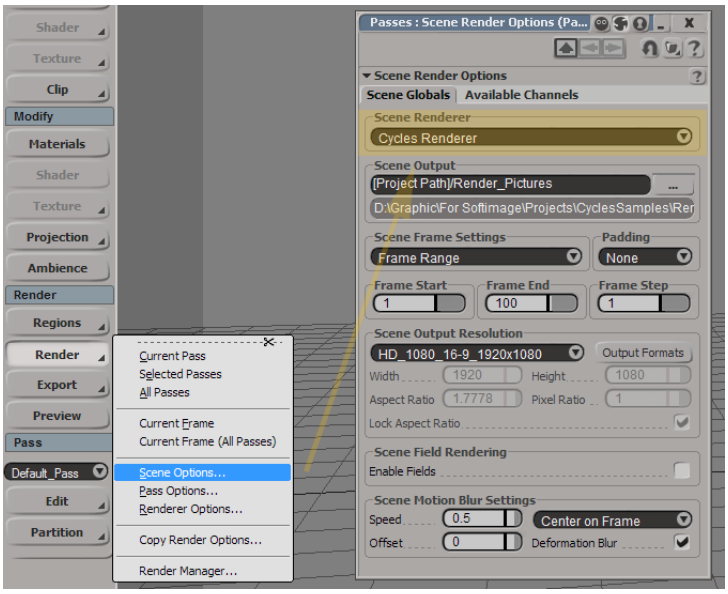
Another example. The same scenes with the same lighting parameters. In the first case, the color space sRBG is used:



In the second case, from the OpenColorIO config:

19 How to use shaderballs

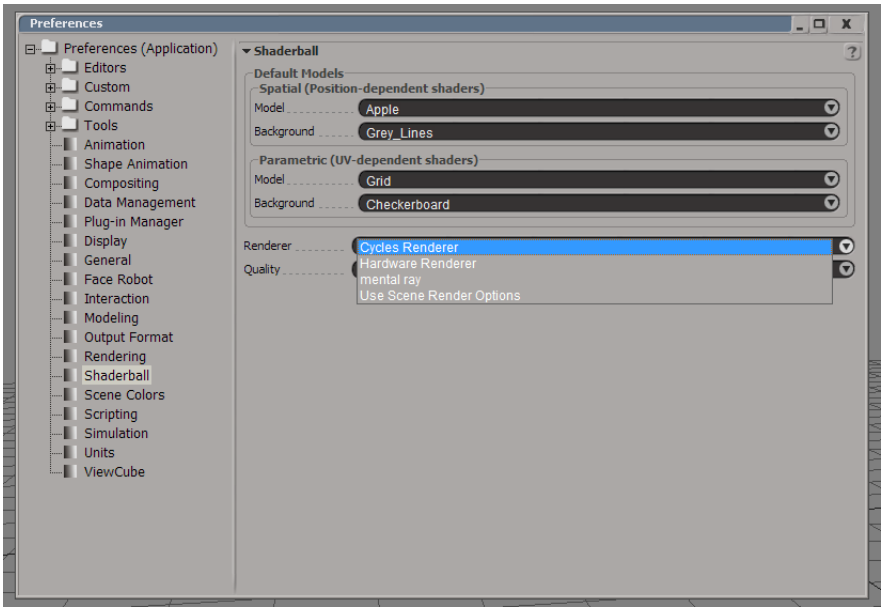
To make shaderballs work, you need to make sure that Softimage understands which renderer to use when rendering it. You can do this by setting the value `Cycles Renderer` of the parameter `Scene Renderer` in the window `Render` – `Scene Options...`.



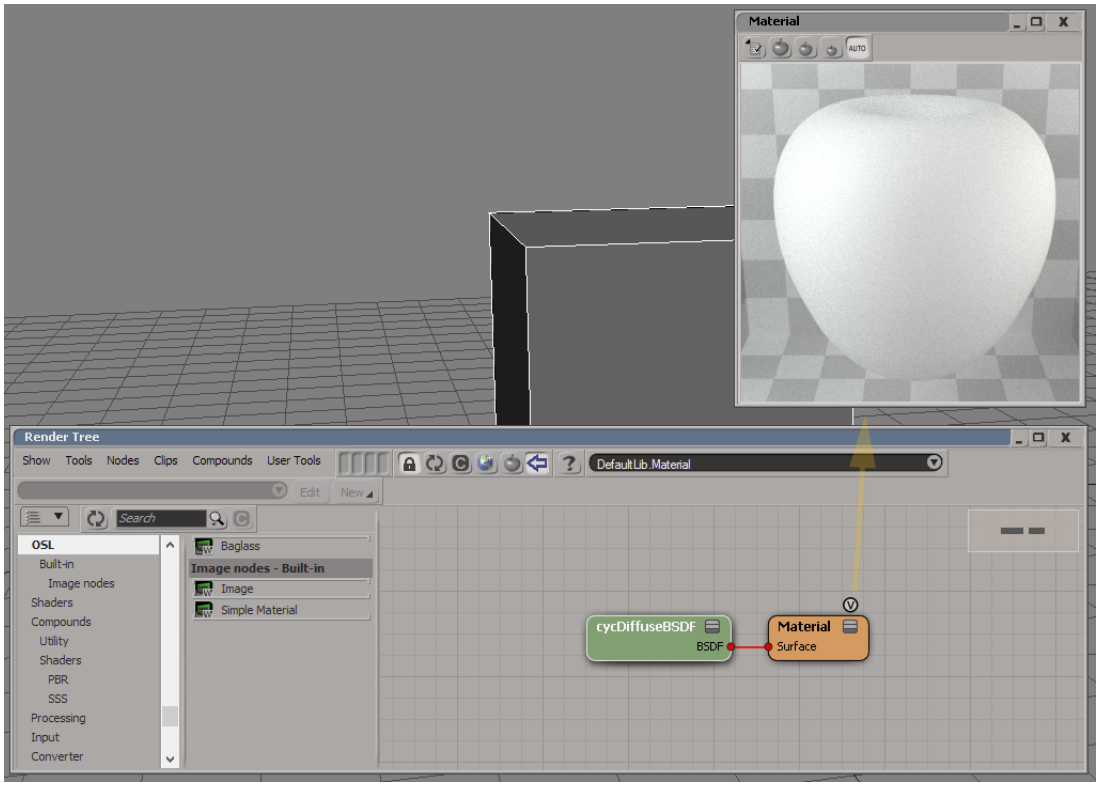
Or in the settings

`File` – `Preferences` – `Shaderball`

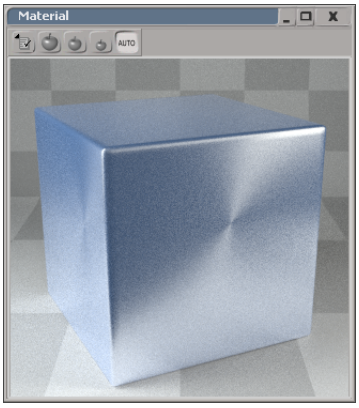
sets the value `Cycles Renderer` for the parameter `Renderer`.



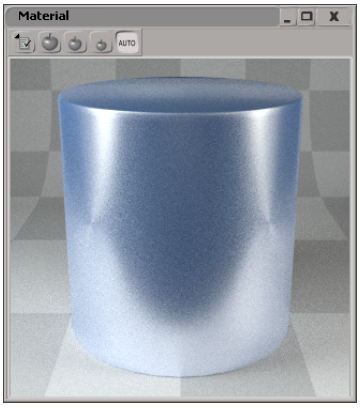
By clicking on preview icon the window with shaderball appears



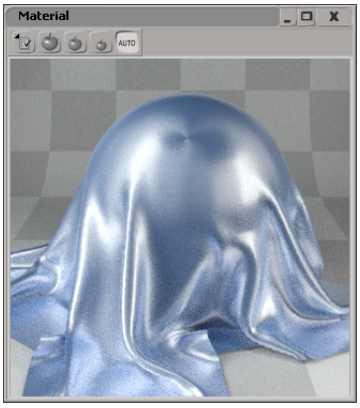
Instead of the standard apple, you can use several other objects.



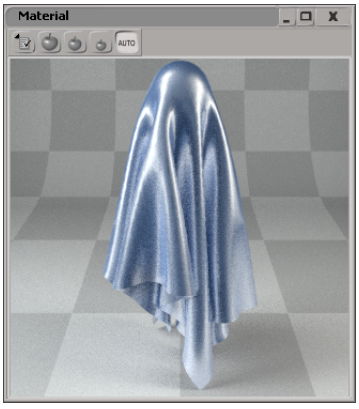
Sycles_Cube



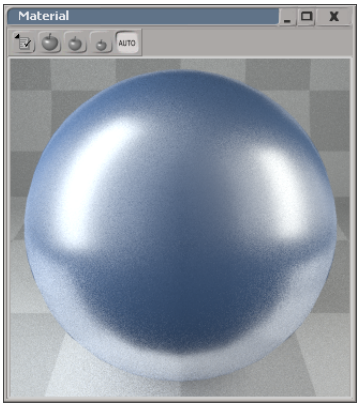
Sycles_Cylinder



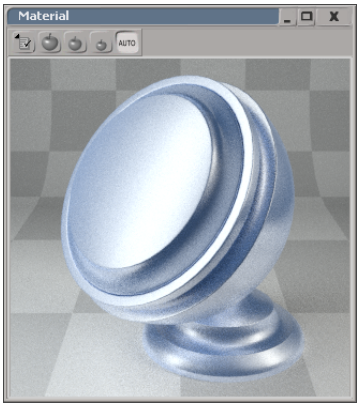
Sycles_Fabric01



Sycles_Fabric02

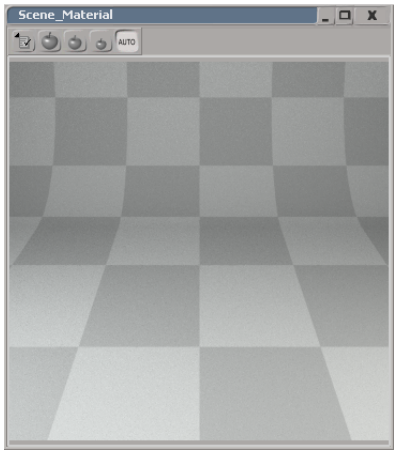


Sycles_Sphere



Sycles_Substance-like

Also there is a new object for the background:



Sycles_Back

To add new models of shaderballs, you should copy the content of the folder

...\Sycles\Application\Shaderballs

to the folder

Path-to-Softimage\Application\Shaderballs

Shaderball render settings, camera position, light sources and the background object shader are hardcoded. There are three light sources: left, right and behind the camera. A checker texture is used on the background object. Some parameters can be changed by editing the file

...\Sycles\Application\Plugins\config.ini

Parameters that can be edited:

checker_light_color – the color in the interval $(0, 1)$ of the light cell of the checker texture;

checker_dark_color – the color in the interval $(0, 1)$ of the dark cell of the checker texture;

checker_scale – number of cells in the checker texture;

render_mode – value 1 switches the render mode to progressive;

render_samples – the number of samples;

film_exposure – an analog of the parameter Film – Exposure of the render settings;

max_bounces – an analog of the parameter Light Paths – Bounces – Max of the render settings;

diffuse_bounces – an analog of the parameter Light Paths – Bounces – Diffuse of the render settings;

glossy_bounces – an analog of the parameter Light Paths – Bounces – Glossy of the render settings;

transmission_bounces – an analog of the parameter Light Paths – Bounces – Transmission of the render settings;

clamp_direct – an analog of the parameter Sampling – Settings – Clamp Direct of the render settings;

`clamp_indirect` – an analog of the parameter `Sampling – Settings – Clamp Indirect` of the render settings;

`shading_system` – value 1 activates osl-shaders support;

`left_light_color_r, g, b` – color components in the interval $(0, 1)$ of the left light source;

`left_light_strength` – intensity of the left light source;

`left_light_ignore_glossy` – the value 1 makes the left light source invisible in the reflections;

`right_light_color_r, g, b` – color components in the interval $(0, 1)$ of the right light source;

`right_light_strength` – intensity of the right light source;

`right_light_ignore_glossy` – the value 1 makes the right light source invisible in reflections;

`center_light_color_r, g, b` – color components in the interval $(0, 1)$ of a central light source;

`center_light_strength` – intensity of the central light source;

`center_light_ignore_glossy` – the value 1 makes the central light source invisible in reflections;

20 How to use OSL-shaders

Let's create the simplest OSL-shader. For this, in any text editor write the following code:

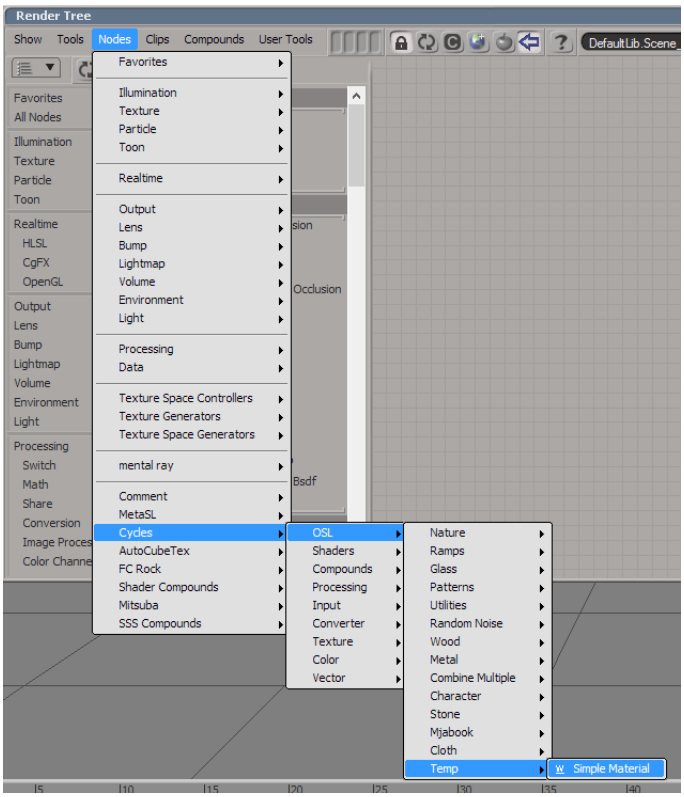
```
//XSICategory: Temp
shader simple_material(
    color Diffuse_Color = color(0.6, 0.8, 0.6),
    float Noise_Factor = 0.5,
    output closure color BSDF = diffuse(N))
{
    color material_color = Diffuse_Color*mix(1.0, noise(P*10.0), Noise_Factor);
    BSDF = material_color * diffuse(N);
}
```

Save it as a file with any name and extension `*.osl`. Put this file to the folder

`\Application\osl\`

of any workgroup. Next run Softimage and see that our shader `Simple Material` appeared in category

`Cycles` – `OSL` – `Temp`.



To what category the shader is placed depends on what is written in the first commented line of the shader after the words `XSICategory`. If nothing is written or there is no such line at all, then the shader will be placed to the category

`Cycles` – `OSL`.

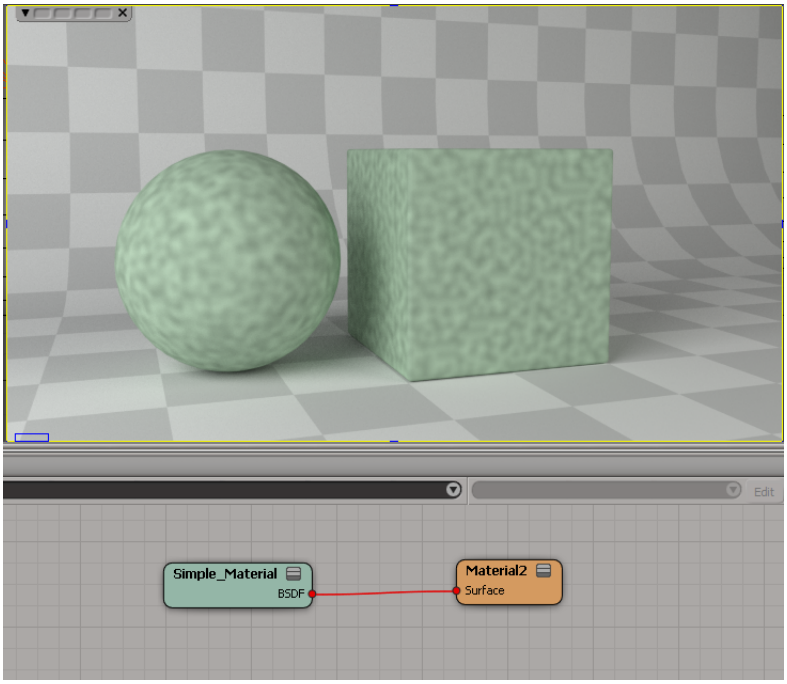
If it is written something like

`Category/Subcategory/Subsubcategory`

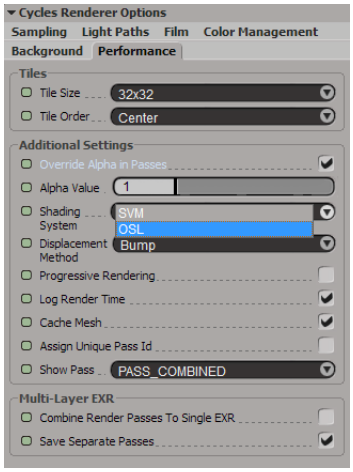
then the shader will be placed to the category

`Cycles` – `OSL` – `Category` – `Subcategory` – `Subsubcategory`.

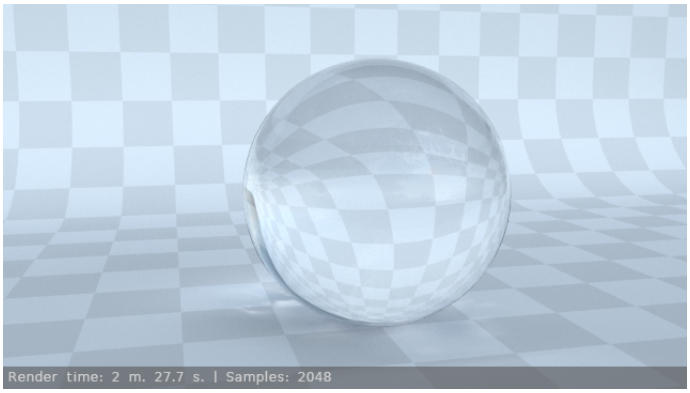
Connect the output port of our shader `Simple Material` to the port `Surface` of the material's root node and see the result.



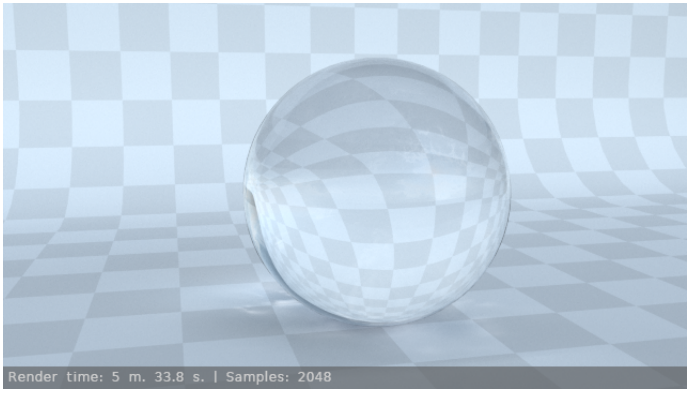
If you don't see result, then you should turn on OSL-shaders support in render settings. Go to the tab `Performance` of the render settings and set value of the parameter `Shading System` equal to `OSL`.



Use the rendering mode with OSL-shaders support very carefully and only if it absolutely necessary, since it greatly increases the rendering time. Here is an example: a simple scene with a glass ball. The glass shader is the usual `GlassBSDF`. Without support of OSL-shaders:



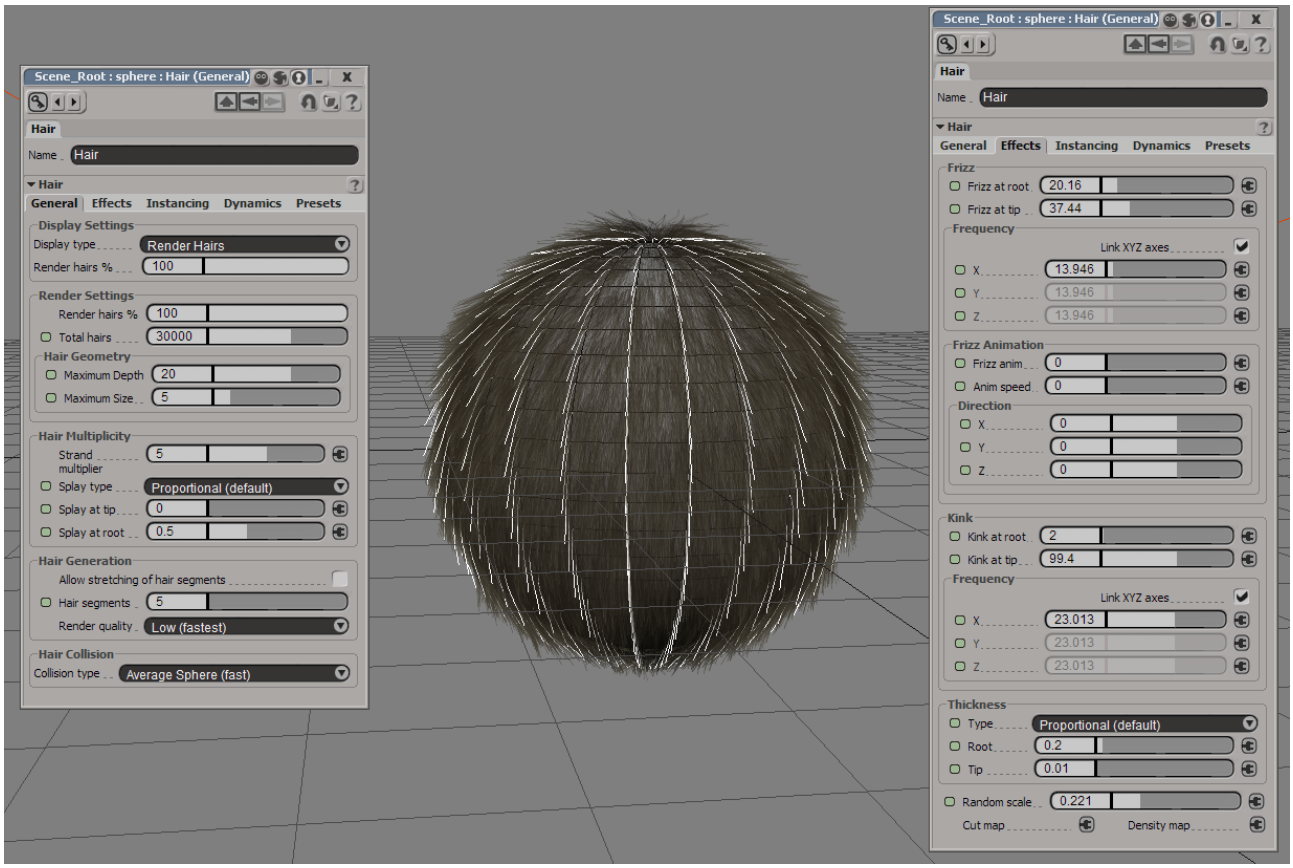
With OSL-shaders support:



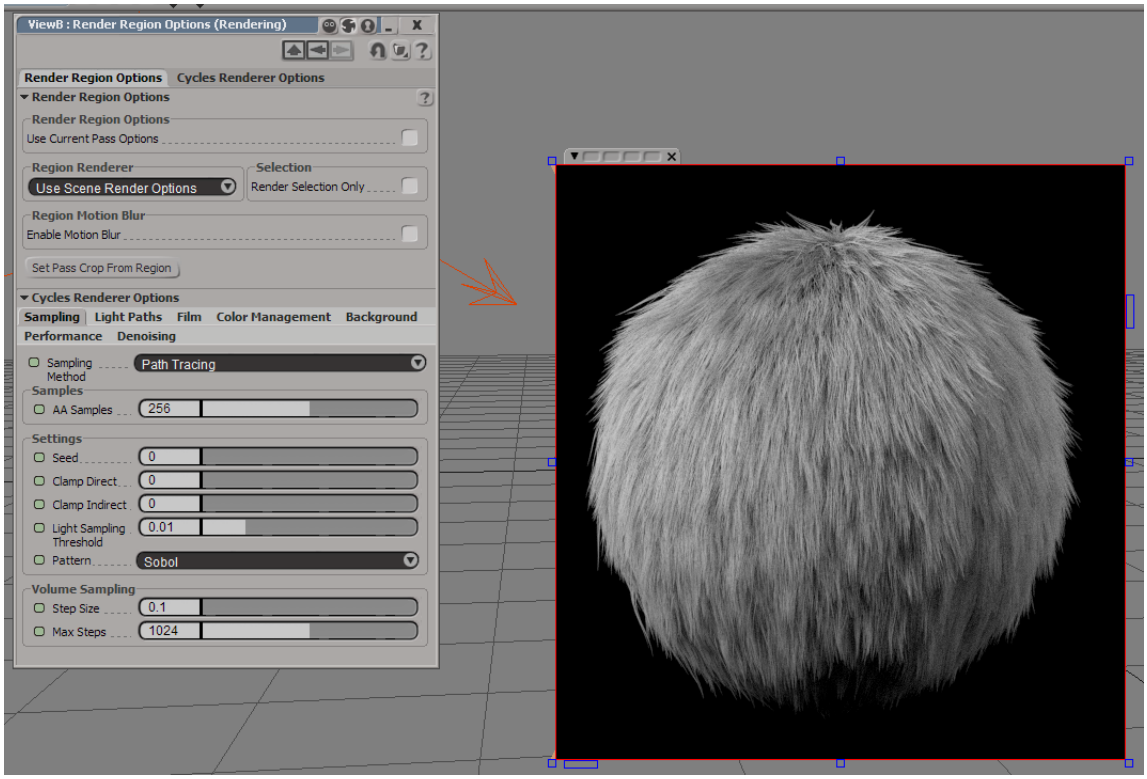
The results are identical, but the rendering time differs by almost two times.

21 How to render standard XSI hairs

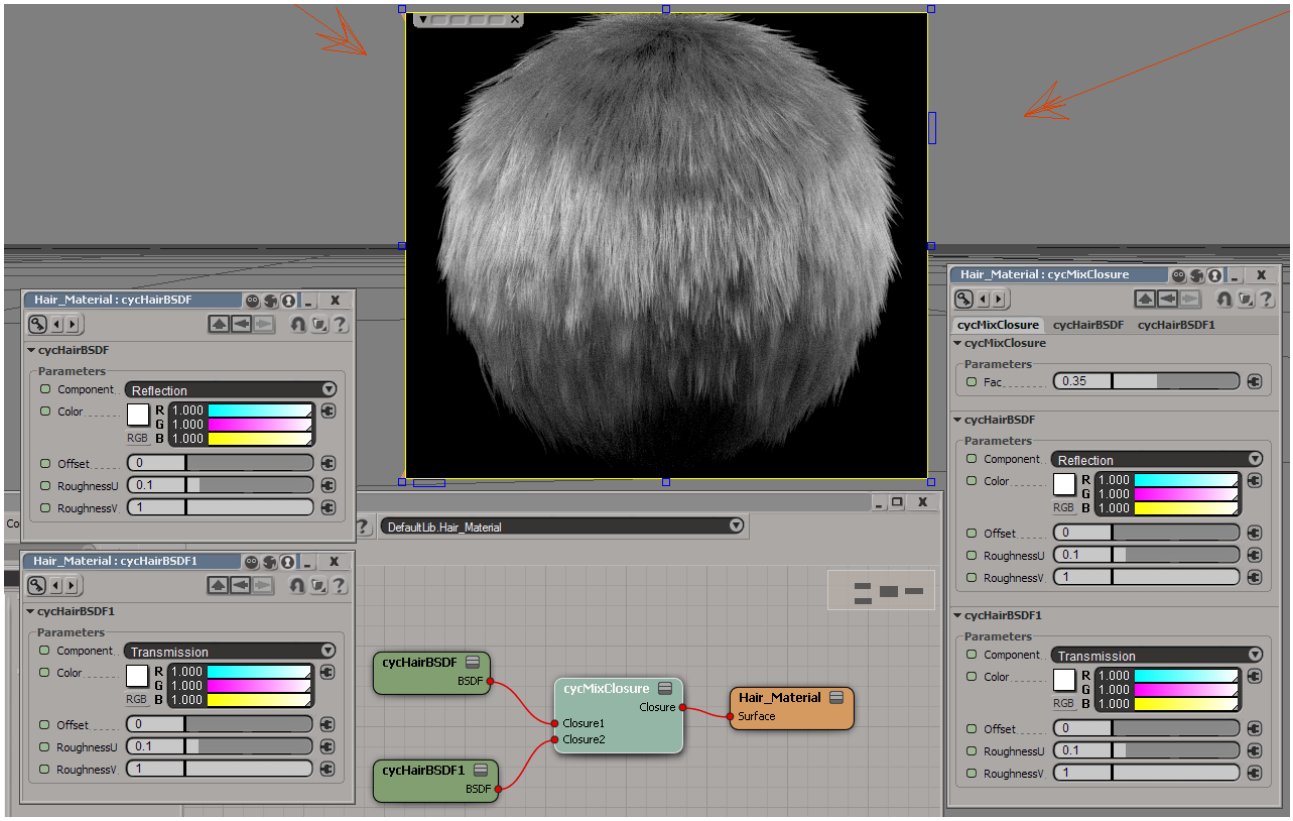
Suppose we have a scene: a ball with standard Softimage hairs.



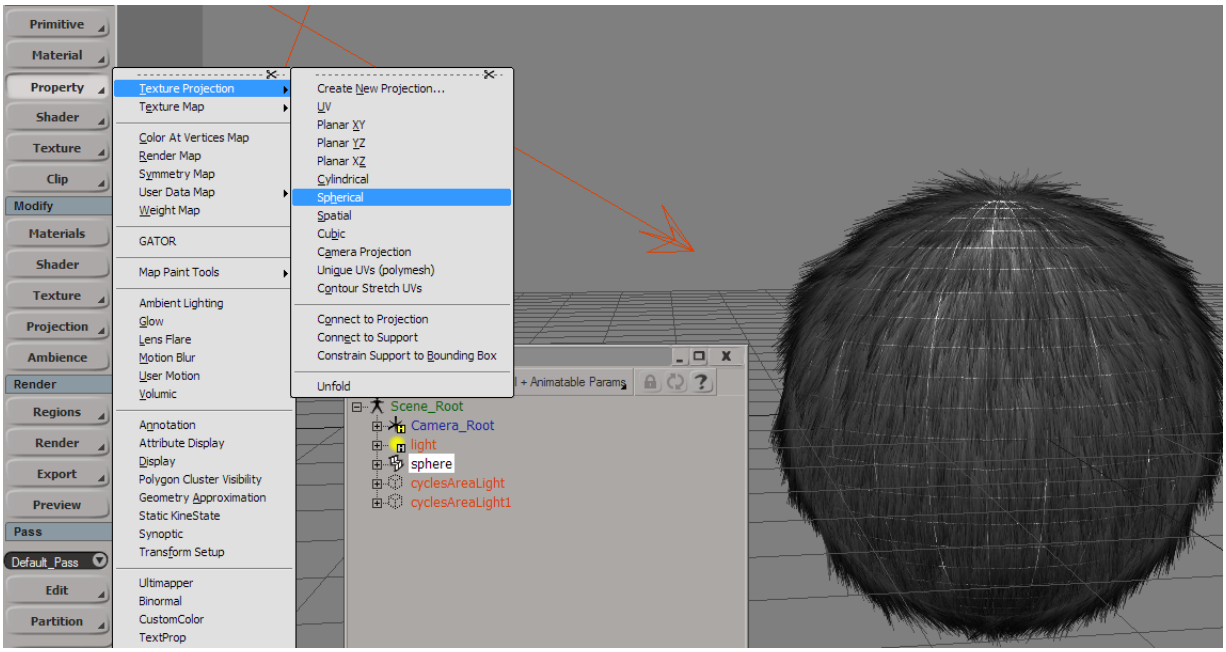
Render it. There are two parameters for the thickness of hairs: the thickness of the roots is taken from `Thickness` – `Root` and for tips from `Thickness` – `Tip`.



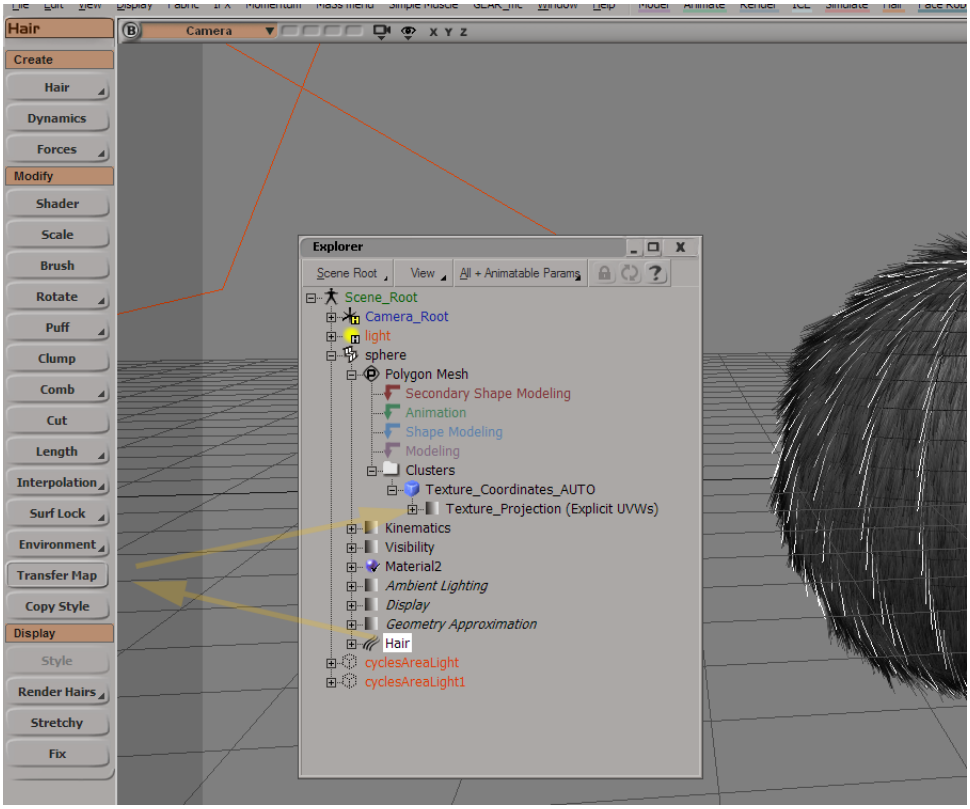
Select the hair and assign a new material to it. Mix two instances of the node `HairBSDF` with proportion `0.35`. The first node has the mode `Reflection`, the second – `Transmission`. Render.



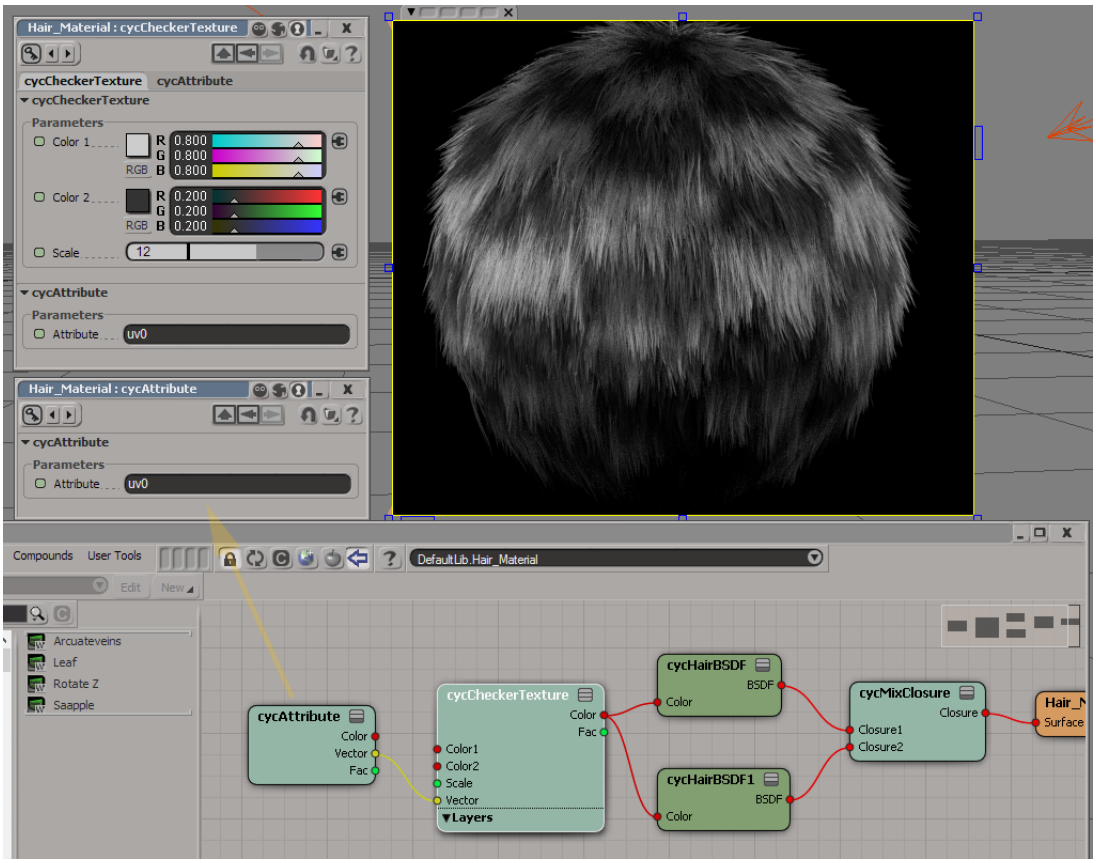
The only way to specify the color of each individual hair is to use texture coordinates. Add spherical texture coordinates to the sphere.



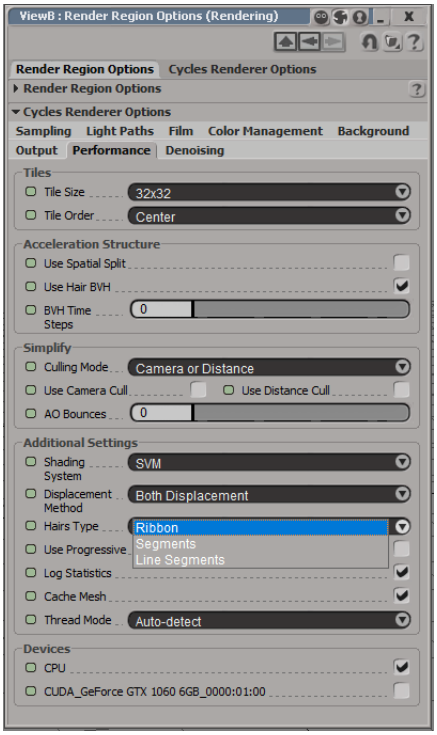
Transfer it to the hairs.



During hairs rendering all texture coordinates on the object are written to vector attributes with the names `uv0`, `uv1` etc. In the hair material add the node `Attribute with value \verbuv0`—— and connect the output `Vector` to the input `Vector` of the node `CheckerTexture`. Also connect the output `Color` to ports `Color` of nodes `HairBSDF`.



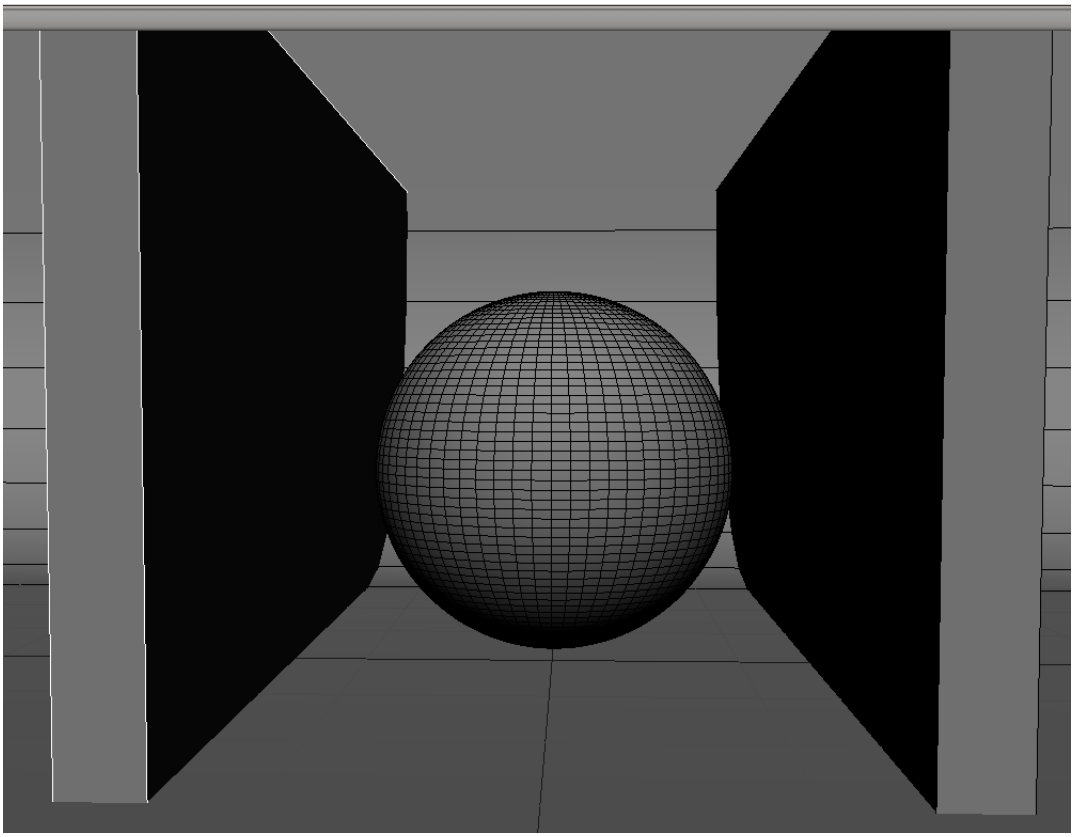
Now we can see that the hairs have a checker coloring. We can set the quality of hairs geometry in the render properties `Performance` – `Hair Type`.



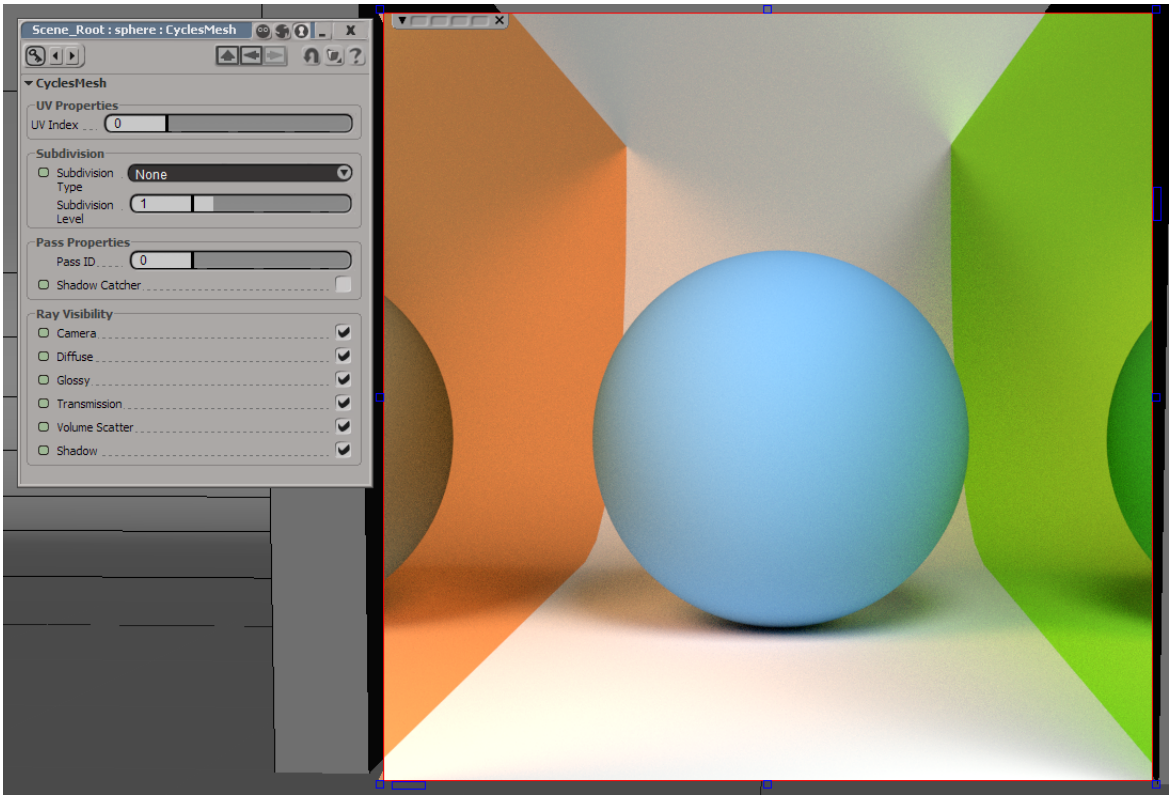
If the mode `Ribbon` is selected, then the hairs will be rendered as flat planes oriented towards to the camera. In the mode `Segments` hairs rendered as cylindrical shapes. In the mode `Line Segments` the hairs rendered as isolated cylindrical segments with gaps between individual segments.

22 How to use ray visibility

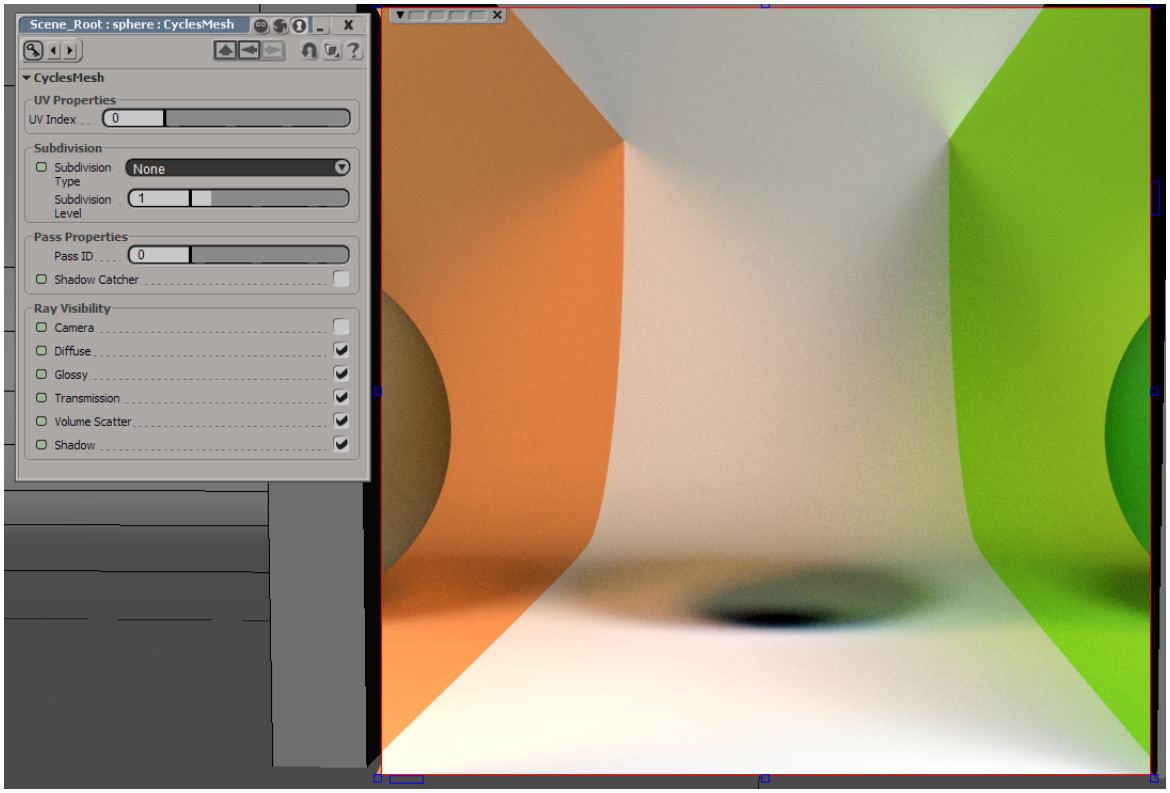
Suppose we have a scene with sphere ad boxes.



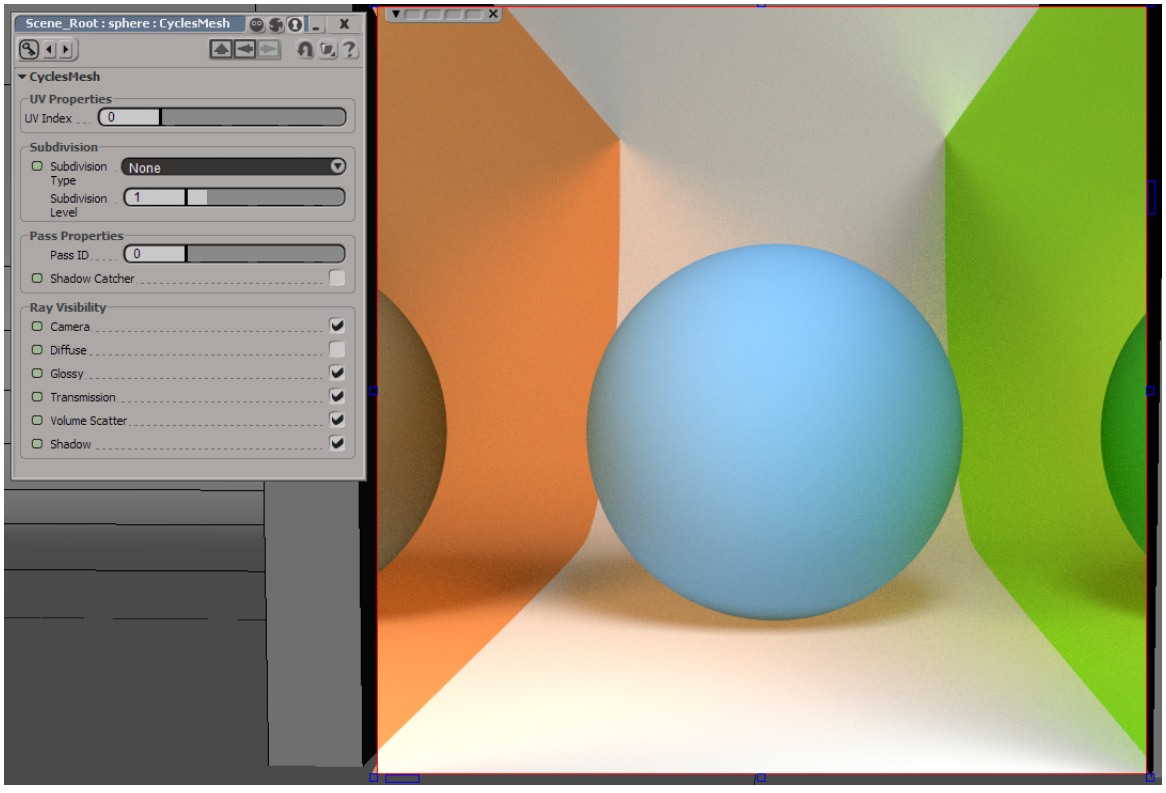
Add to the sphere the property `CyclesMesh`. Render.



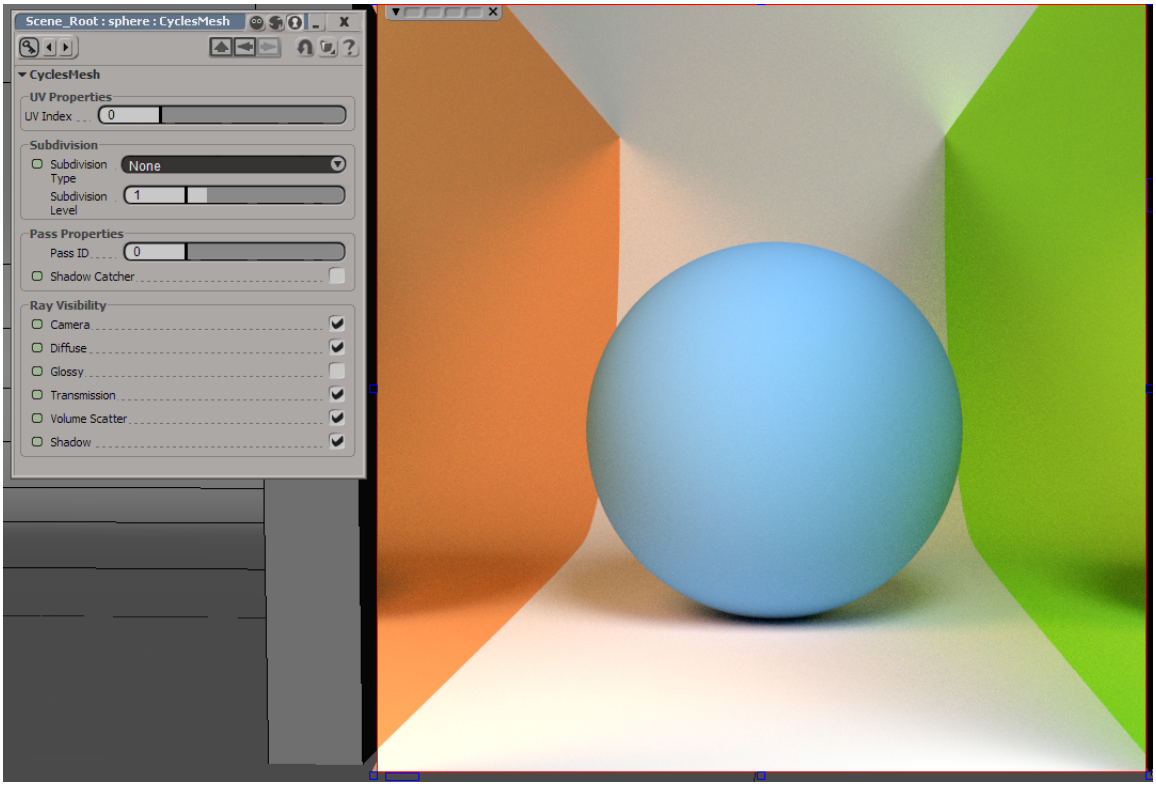
In the case of parameter `Ray Visibility` – `Camera` is tuned off the object is not rendered directly, but it visible in reflections and it participates in the GI calculation.



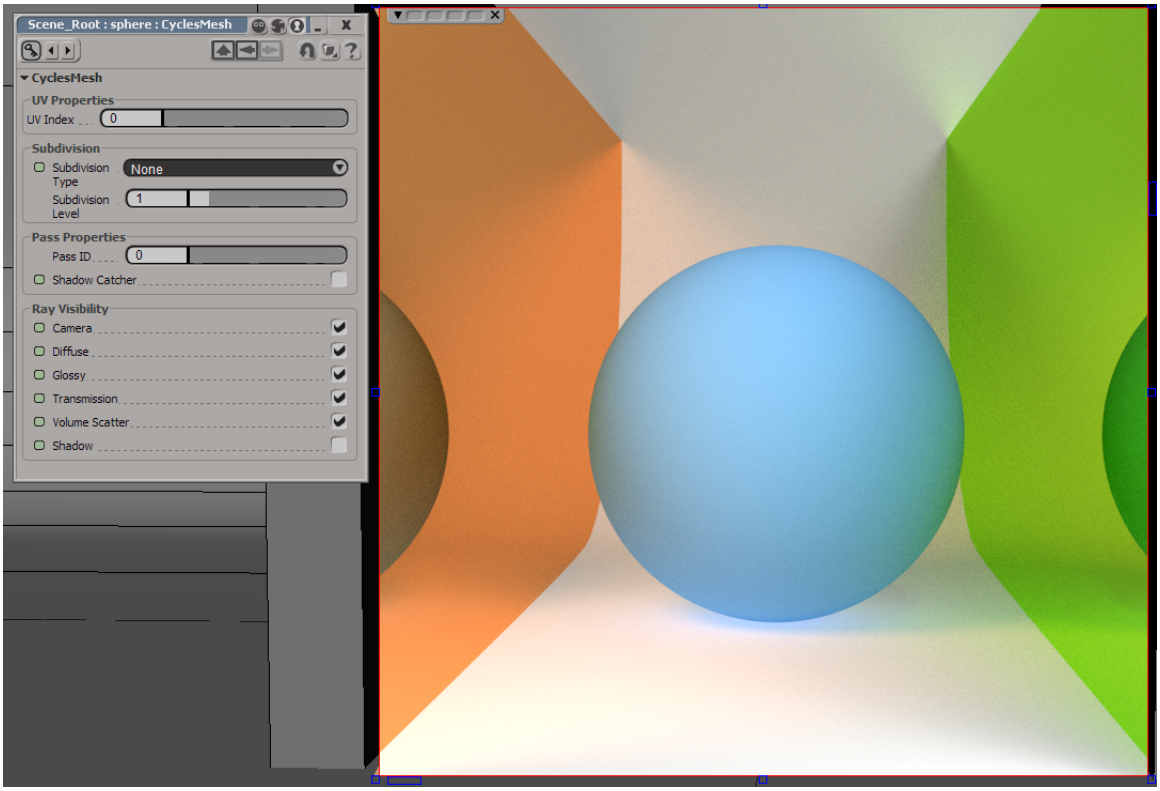
In the case of parameter `Ray Visibility` – `Diffuse` is tuned off the object does not participate in the GI calculation.



In the case of parameter `Ray Visibility` – `Glossy` is tuned off the object becomes invisible in reflections.

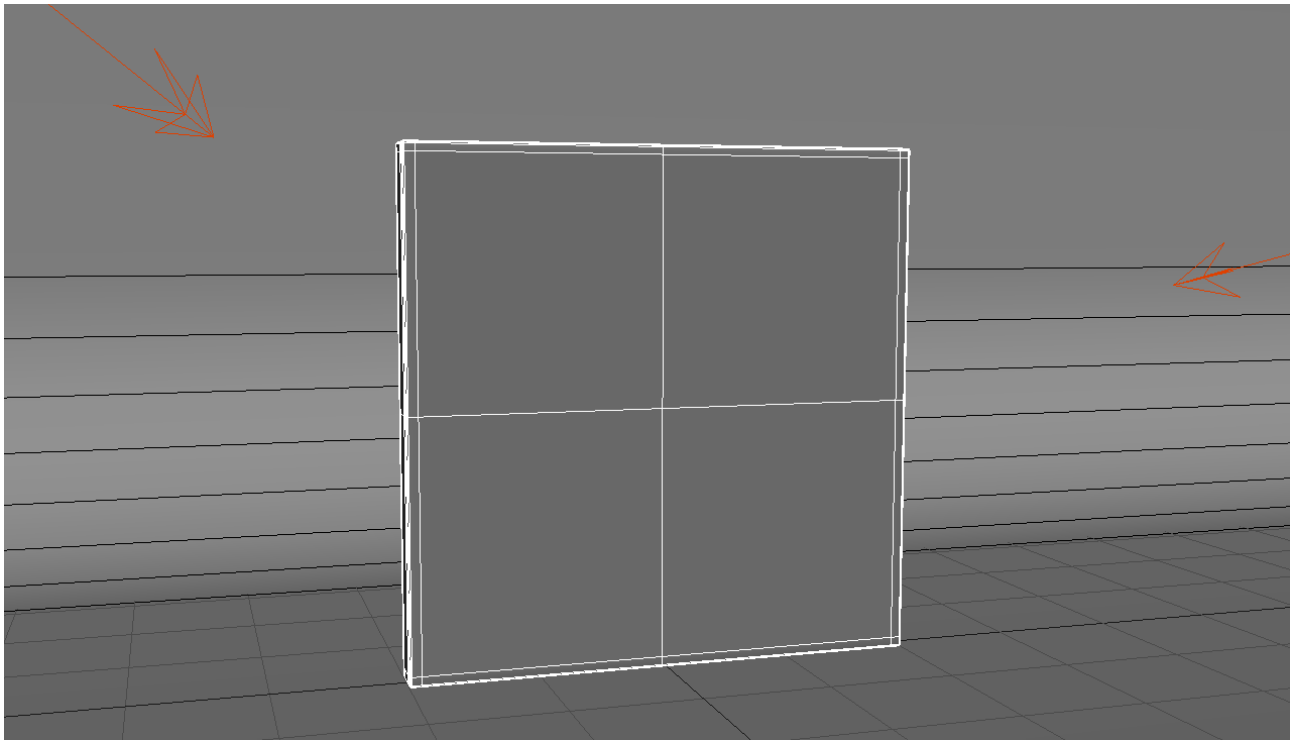


In the case of parameter `Ray Visibility` – `Shadow` is tuned off shadows for the object are not calculated.



23 How to use multiple uv coordinates

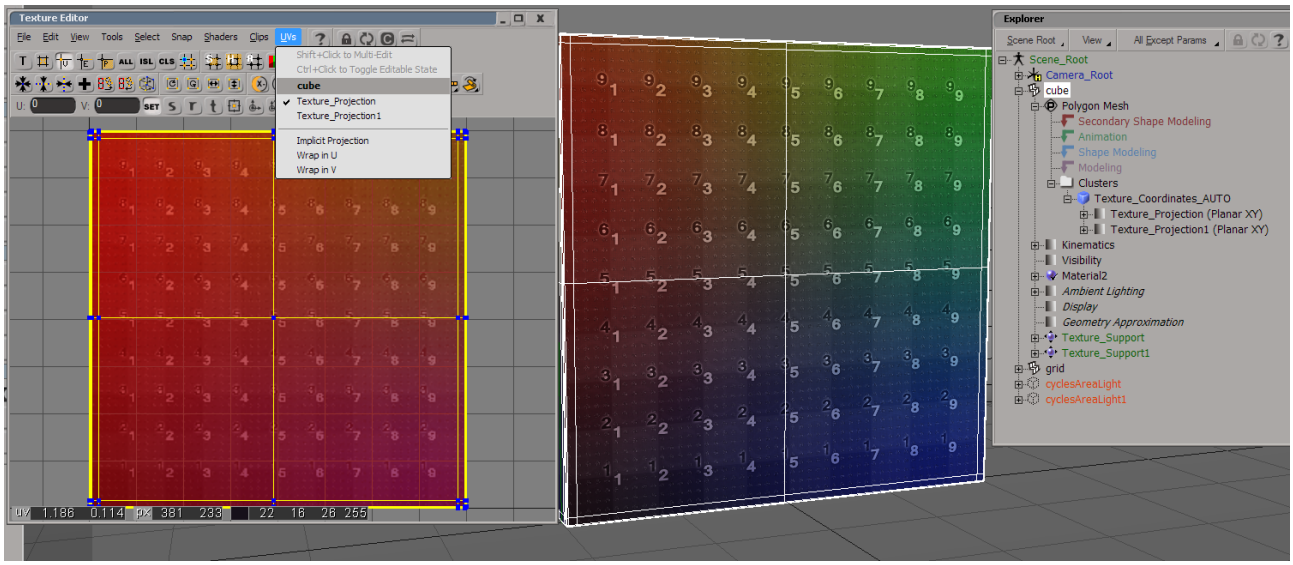
Suppose we have a simple scene with quad and some light sources.



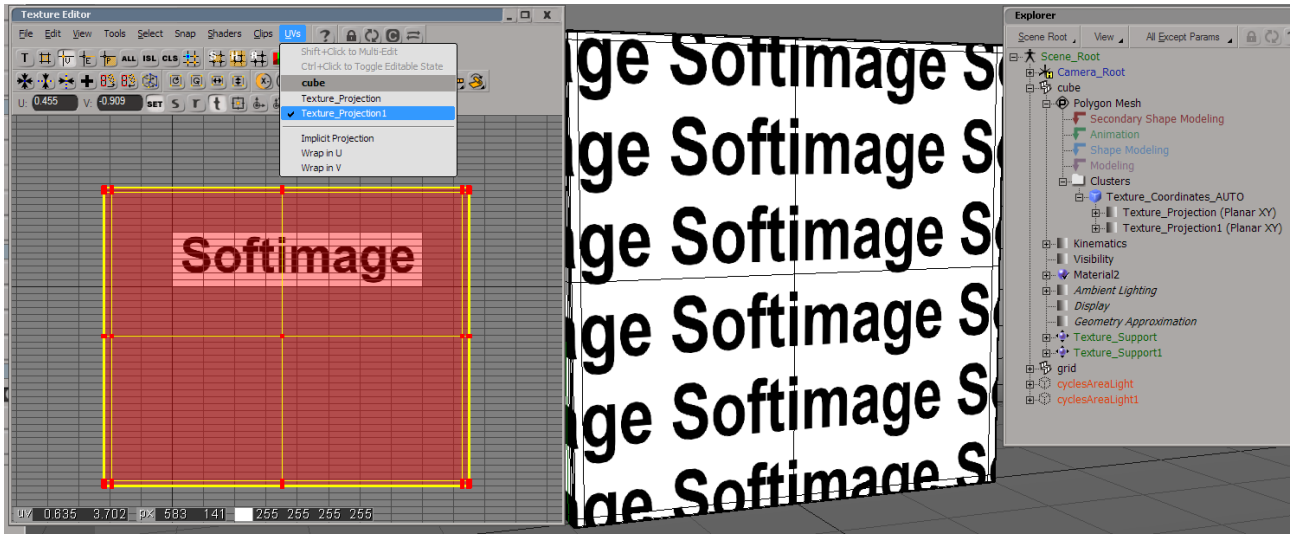
We would like to make it looks like a brick wall with some label on it. For example this label:

Softimage

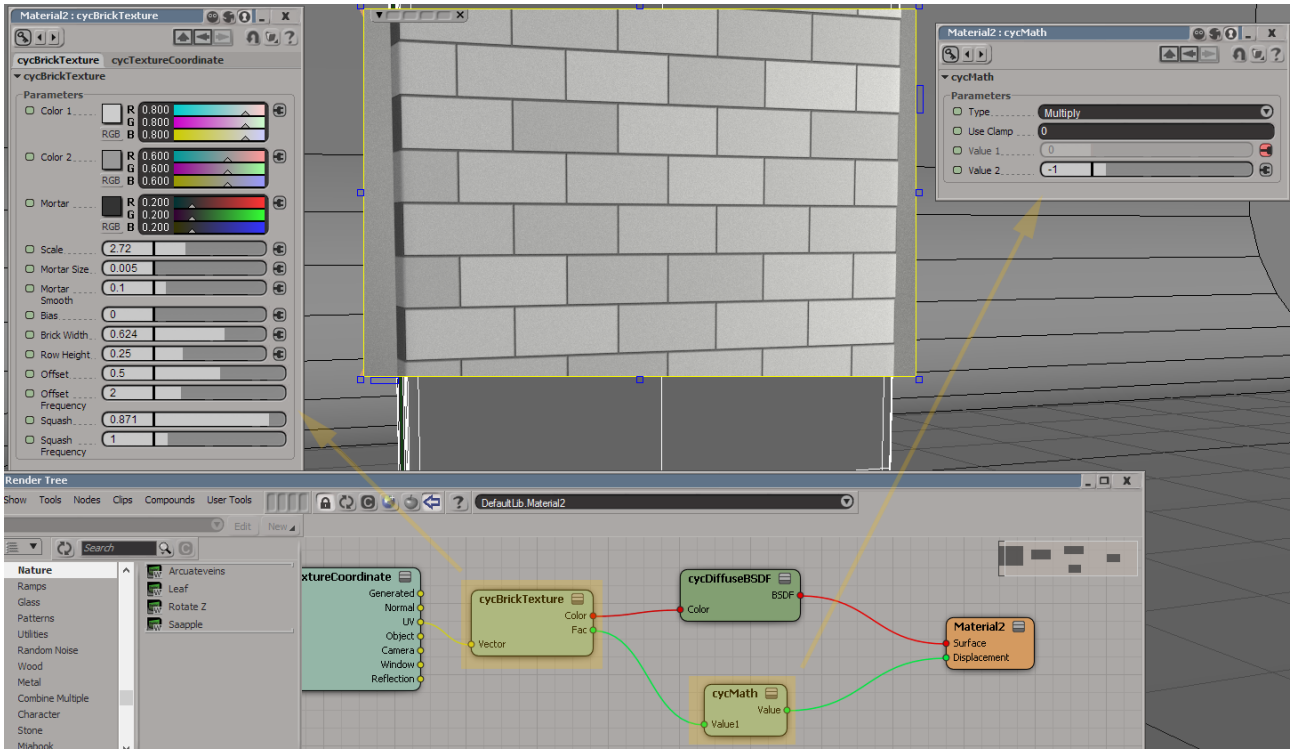
Add to out quad two texture projections to the XY-plane. The first projection leave unchanged.



Scale the second projection so that the label placed in the desired location.

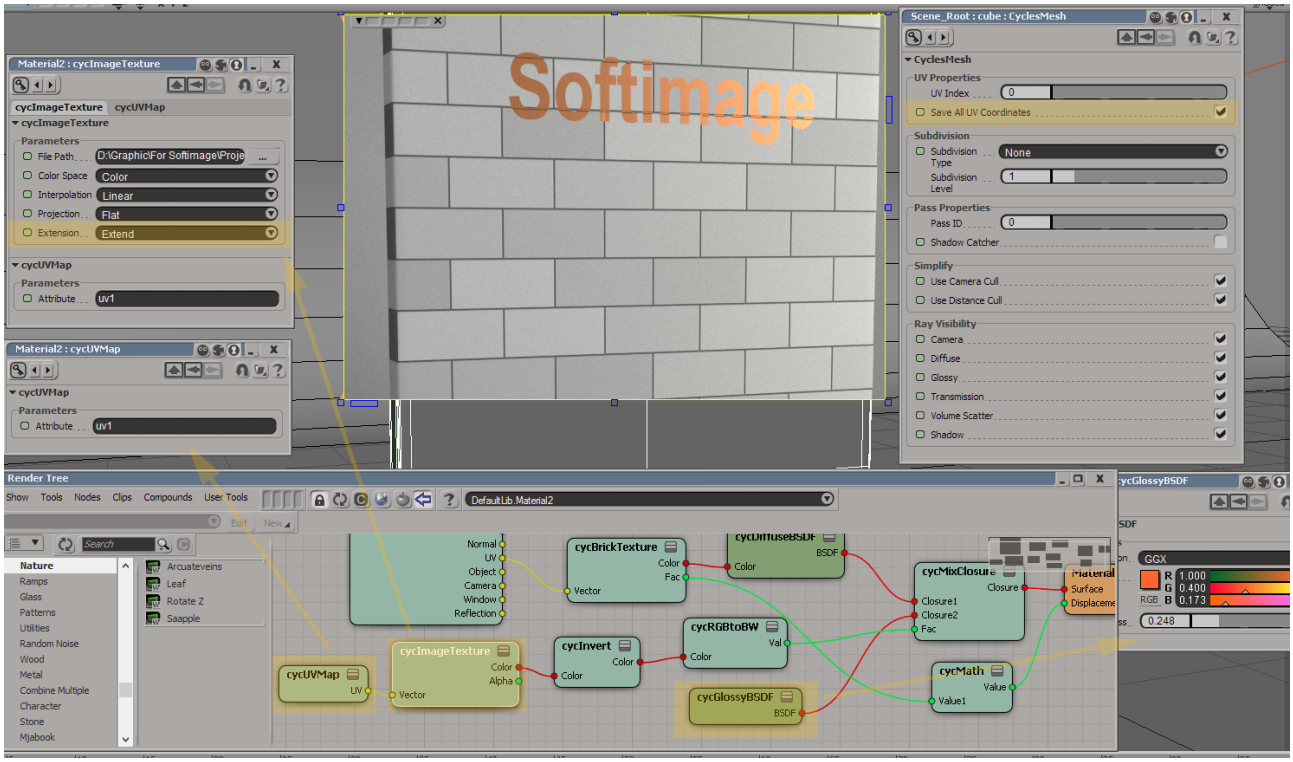


Apply to the quad the material like a brick wall. Use the node `TextureCoordinate` for setting texture coordinates. These coordinates contains data about the first (unchanged) texture projection.



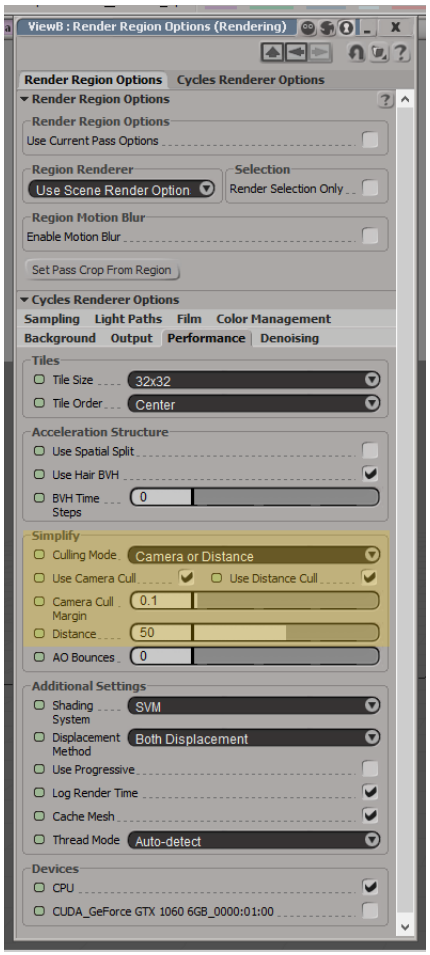
Now we should say to render engine how to use the second texture projection. Add the property `Cycles Mesh` to the quad and turn on the parameter `Save All UV Coordinates`. Now with the help of the node `UVMap` we can select data from any texture coordinates on the object. All these coordinates have names `uv0`, `uv1`, ...

Add to the material the image with the label. In the node `ImageTexture` set the value of the parameter `Extension` is equal to `Extend`. This will set the texture untiled. Set as texture coordinates the result of the node `UVMap` with attribute value is equal to `uv1`. Finally, use this texture as a mask.


Softimage
Material2 : cycImageTexture
cycImageTexture
cycUVMap
Parameters
File Path
Color Space
Color
Interpolation
Linear
Projection
Flat
Extension
Extend
Material2 : cycUVMap
Attribute
uv1
Scene_Root : cube : CyclesMesh
CyclesMesh
UV Properties
UV Index
Save All UV Coordinates
Subdivision
None
Pass Properties
Pass ID
Shadow Catcher
Simplify
Use Camera Cull
Use Distance Cull
Ray Visibility
Camera
Diffuse
Glossy
Transmission
Volume Scatter
Shadow
Render Tree
Show
Tools
Nodes
Clips
Compounds
User Tools
DefaultLib.Material2
Nature
Ramps
Glass
Patterns
Utilities
Random Noise
Wood
Metal
Combine Multiple
Character
Stone
Mjabook
Arcuateveins
Leaf
Rotate Z
Saapple
Normal
UV
Object
Camera
Window
Reflection
cycBrickTexture
Color
Fac
Vector
BSDF
cycMixClosure
Closure1
Closure2
Fac
Closure
Surface
Displacement
cycRGBtoBW
Val
cycInvert
cycUVMap
cycImageTexture
Alpha
cycGlossyBSDF
cycMath
Value
Value1
GGX
R 1.000
G 0.400
B 0.173
0.248

24 What is Camera Cull and Distance Cull

In the section `Performance` — `Symplify` of the render properties we can turn on parameters `Use Camera Cull` and `Use Distance Cull`.

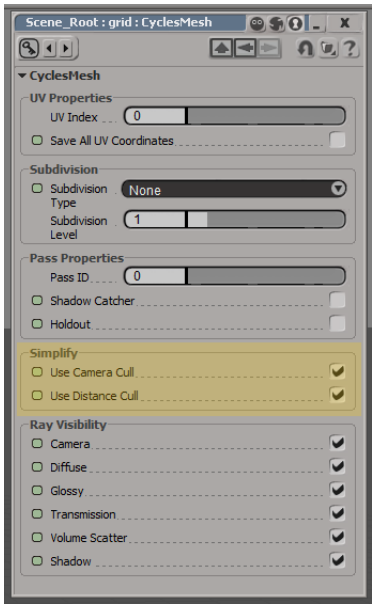


When the parameter `Use Camera Cull` is turned on, the each object on the scene is checked before being added to the rendering engine, whether it is visible from the camera or not. If the object is outside the camera's visibility cone, the value of `Camera Cull` for it becomes `True` and the object does not appear on the render. The parameter `Camera Cull Margin` define how far the object should be from the boundary of visibility cone in order to still used in the render. The larger this parameter, the farther the object can be from the visibility cone and at the same time take the value of the parameter `Camera Cull` = `False`.

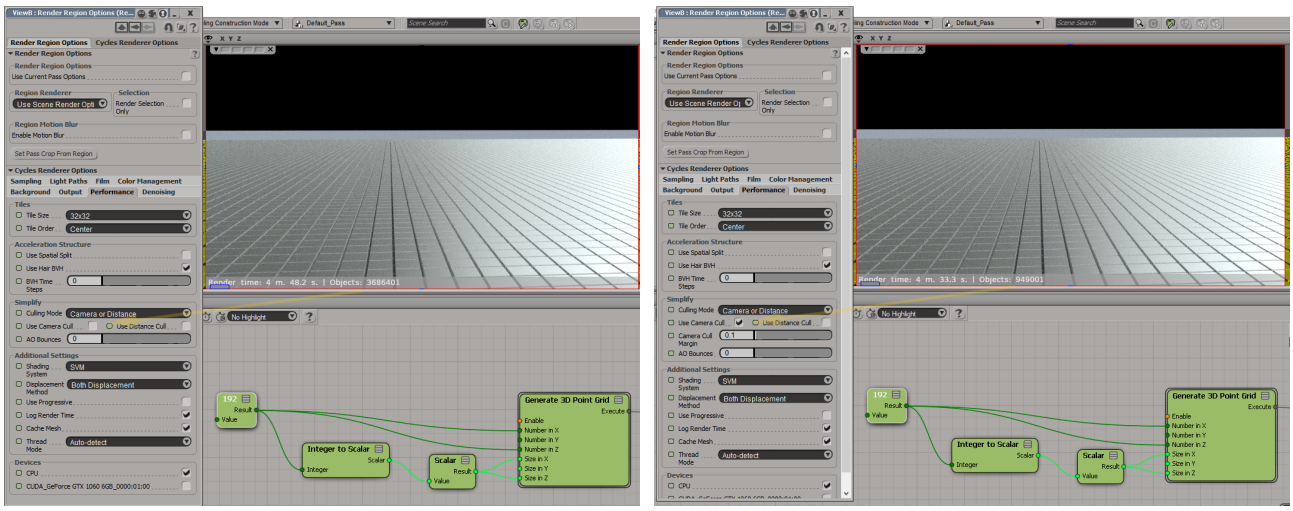
If the parameter `Use Distance Cull` is turned on, then the render check the distance between the object and the camera. If the distance between the object and the camera is greater than the value of the parameter `Distance`, then the value of the parameter `Distance Cull` for this object becomes `True` and this object excluded from the render.

The value of the parameter `Culling Mode` can take two values: `Camera or Distance` and `Camera and Distance`. The value of this parameter does not matter if only one method of excluding objects from the render is used. But if both methods are enabled, then for the mode `Camera or Distance` the object is excluded if it should be excluded by at least one of the methods, and for the mode `Camera and Distance` it is excluded in the case when it should be excluded by both methods.

An each object can be forcibly excluded from the culling. For this, in the properties `CyclesMesh` and `CyclesHairs` you should turn off the options `Use Camera Cull` and `Use Distance Cull`. For objects that do not have these properties, the parameters are considered to be enabled.



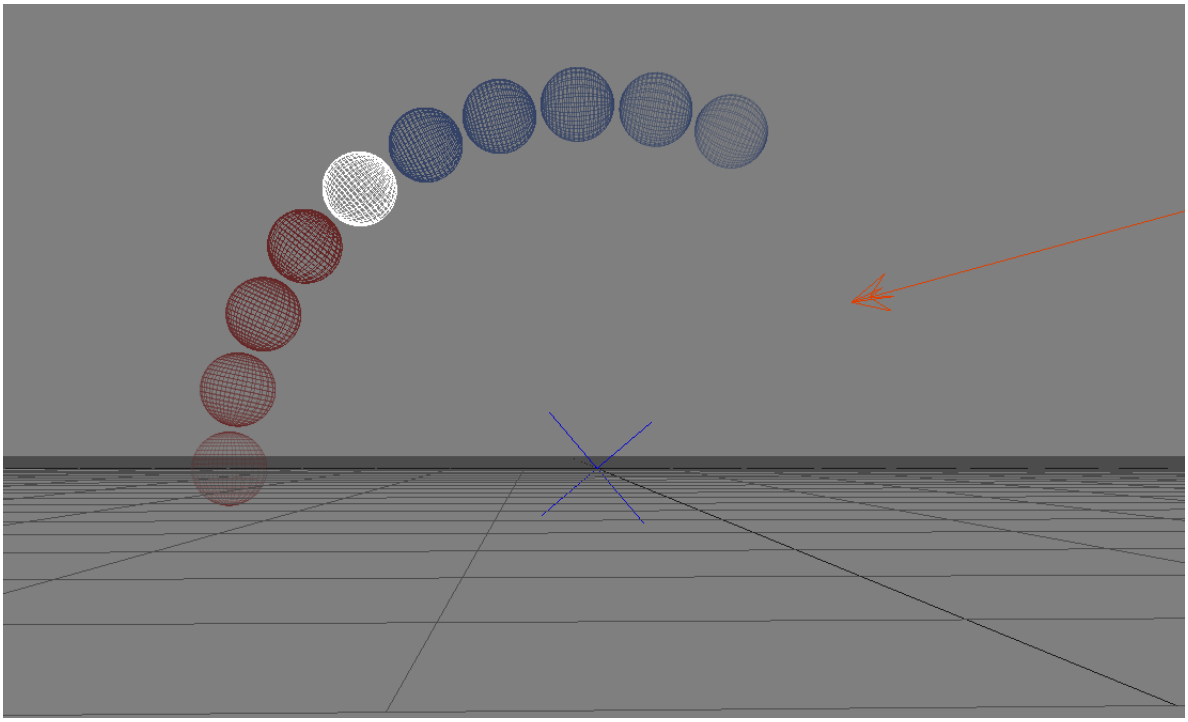
Let's consider an experiment. Generate a lot of cubes and render it with two modes: first we will render all objects, and then render with culling invisible object. Results:



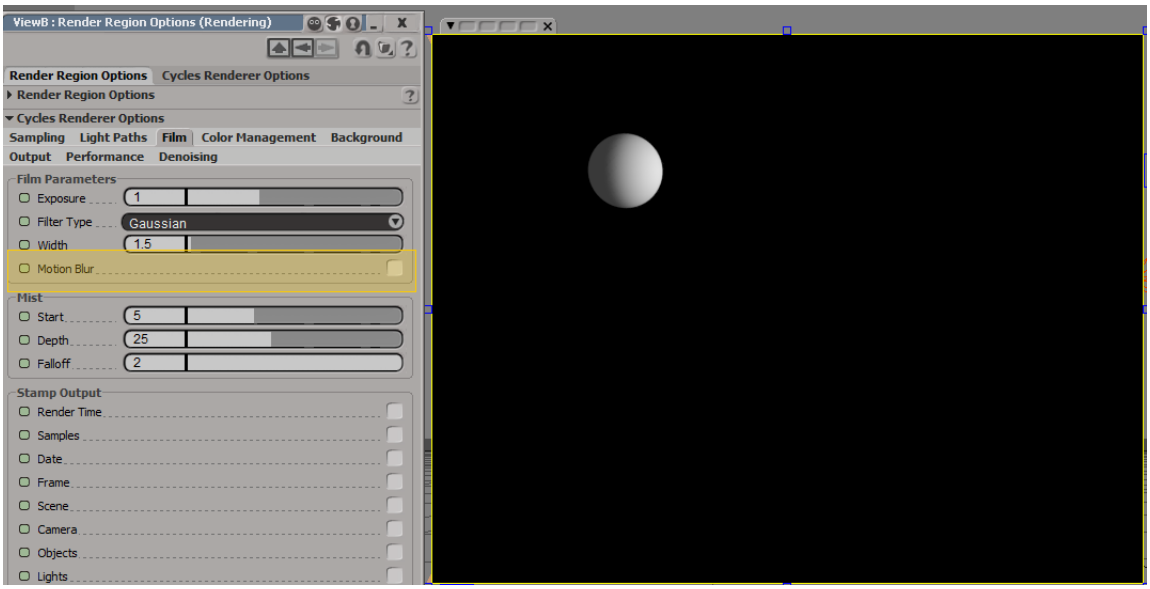
In the first case the number of objects 3 686 401, render time 4 min. 48 sec. In the second case the number of objects 949 001, and render time 4 min. 33 sec. Slightly faster, but not too much. It is because with the culling we should check conditions and this increase computations before the render.

25 How to render motion blur

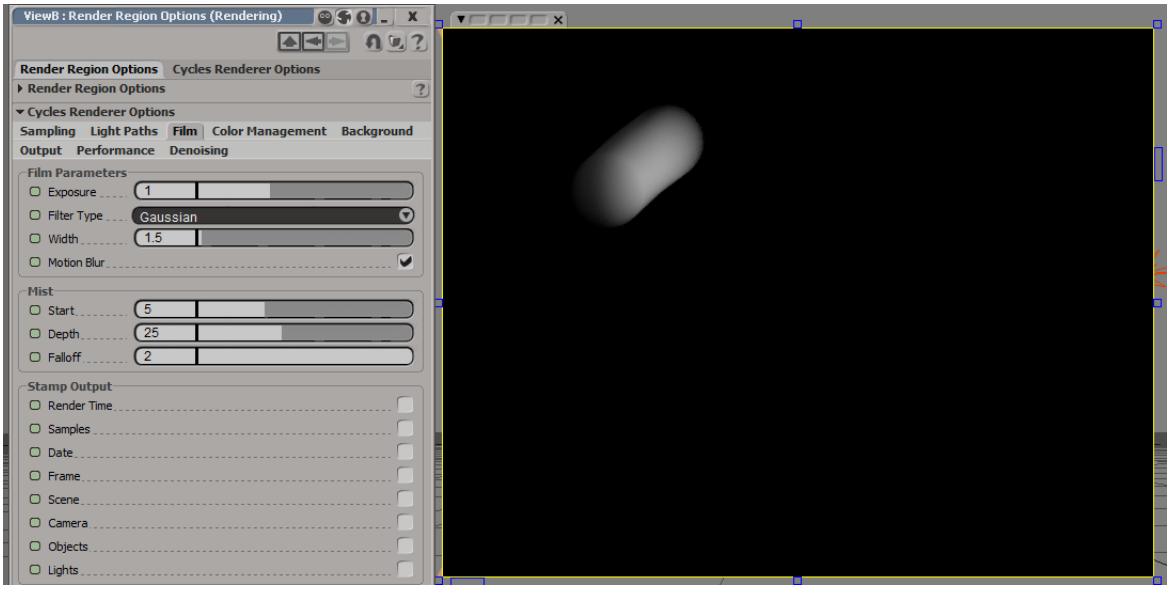
Suppose we have a scene in which the ball is spinning in a circle. Actually, this ball is a child of the null, which located in the center of the scene and this null is spinning around the z axis.



To enable effect of the motion blur we should activate the parameter `Film – Motion Blur` in the render properties.

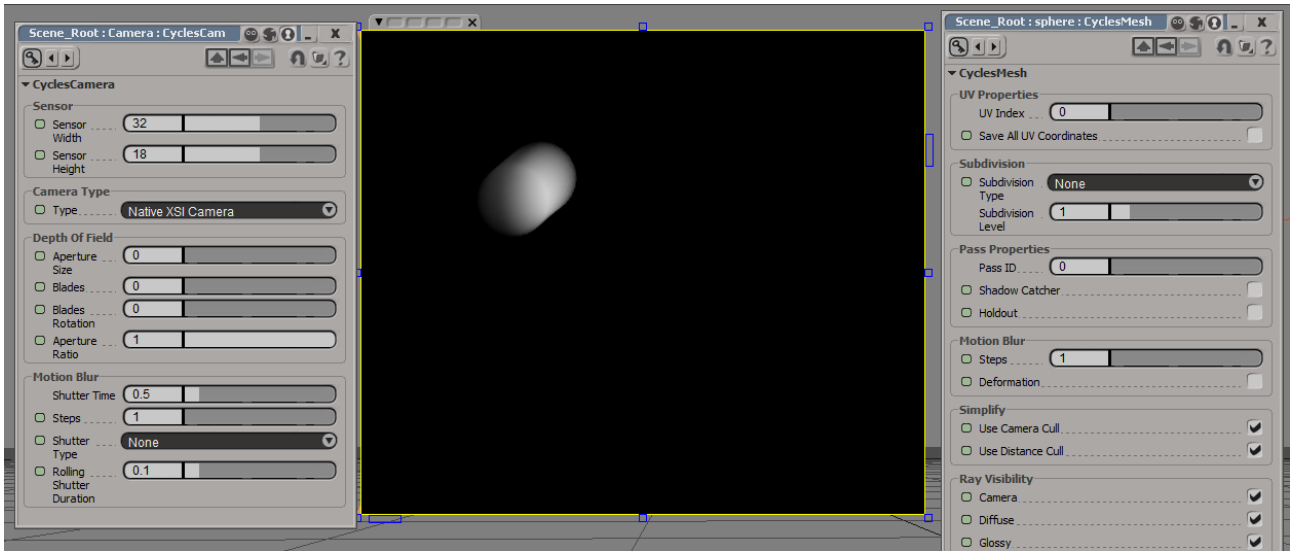


The result.

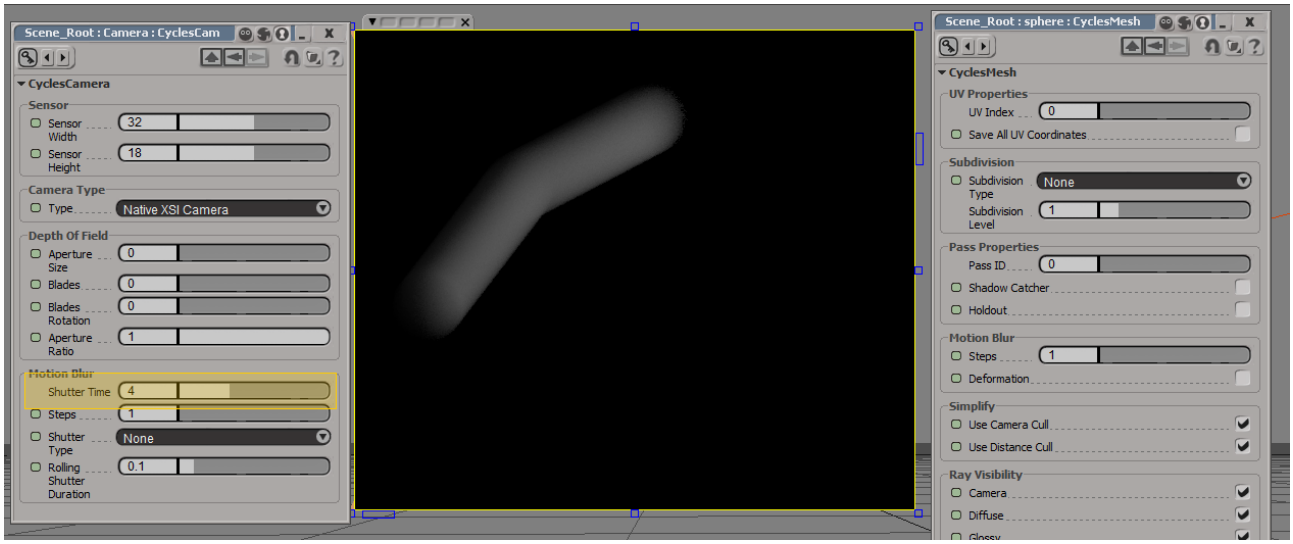


For a more accurate adjustment of the effect, it is necessary to add to the camera and to the ball properties by commands

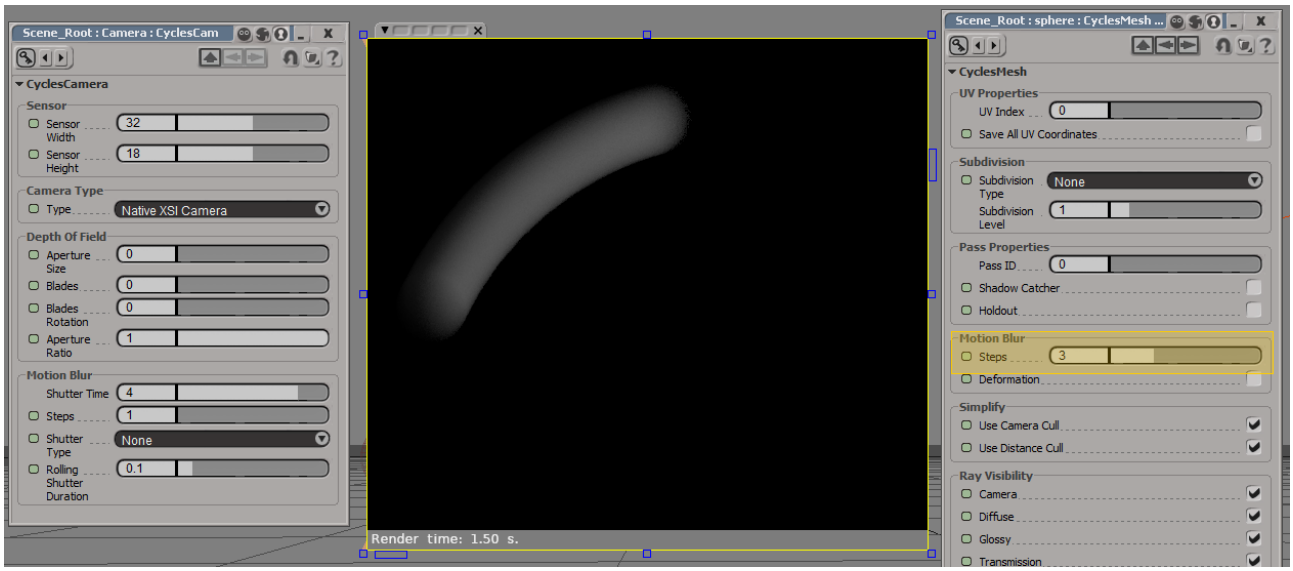
`Property` – `Cycles Properties` – `Add Camera Property`
and
`Property` – `Cycles Properties` – `Add Mesh Property`.



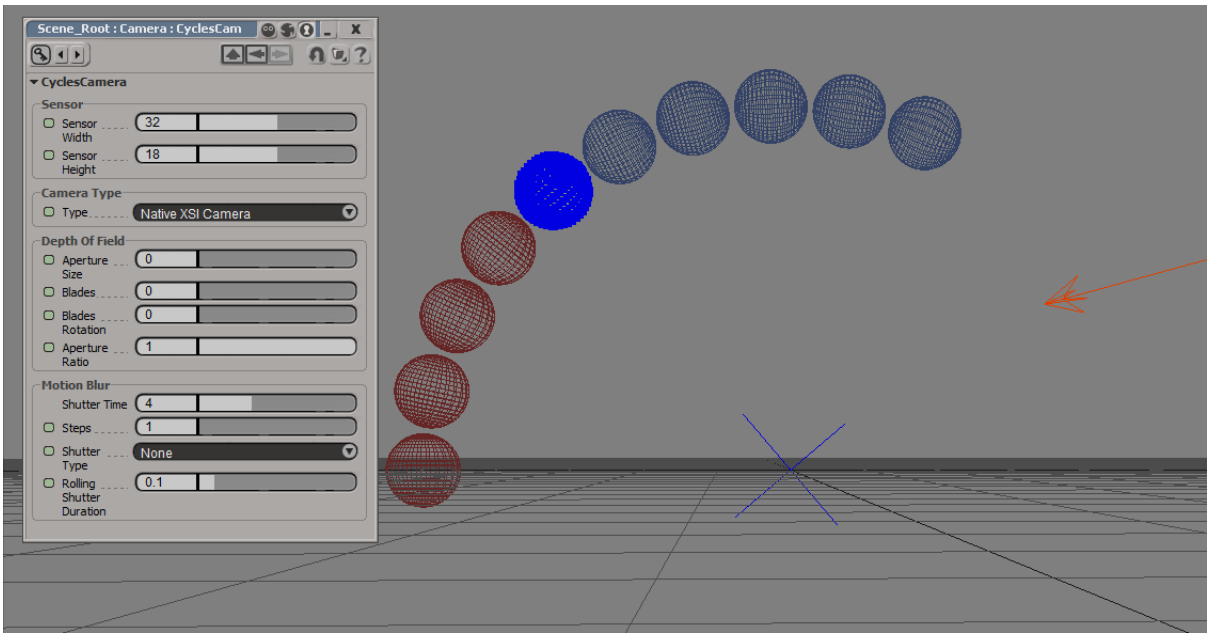
Change value `Shutter Time` on the camera to something greater (for example 4). But we see that the trajectory of the blur is linear.



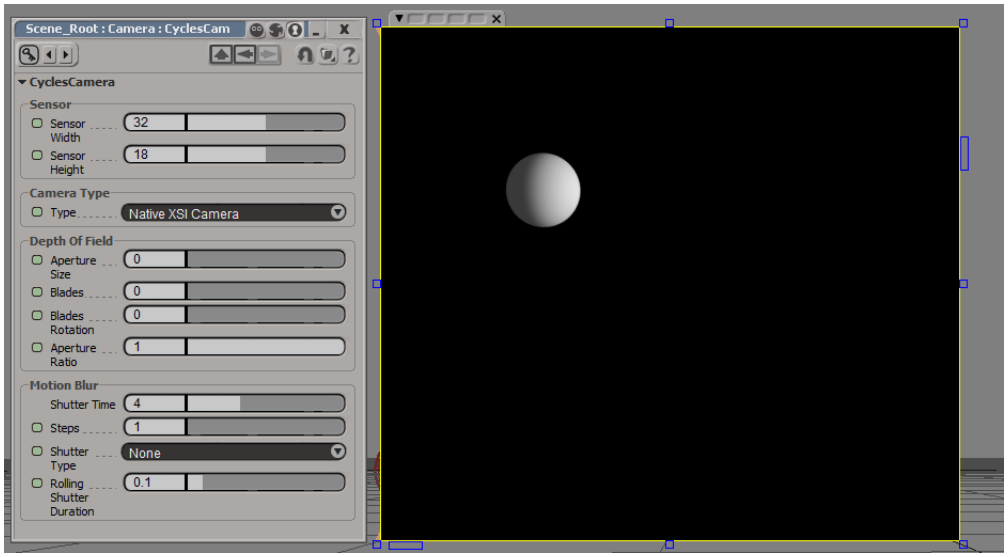
To fix it we should change the parameter `Steps` on the ball to some greater number (for example 3). Now all looks correct.



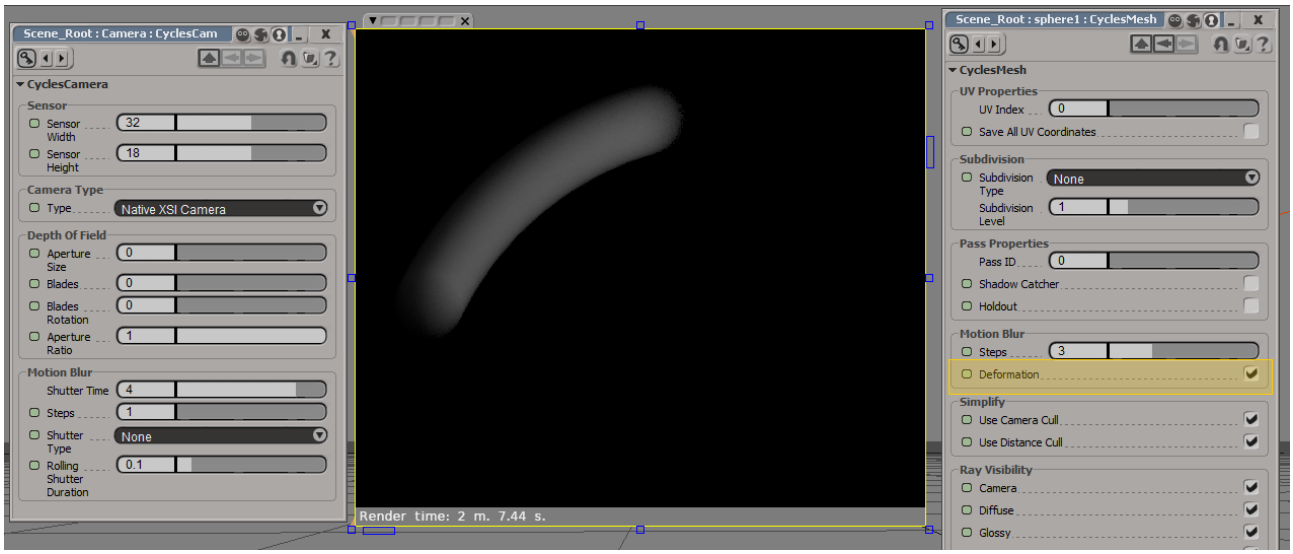
Next consider the same scene, but only difference that the ball is skinned to rotated null.



Render and see that there are no motion blur effect.



It's because the object is not moved in the space, but it's vertices are moved by skinning deformation. To fix it add to the ball the property `Cycles Mesh` and turn on the parameter `Motion Blur` – `Deformation`.

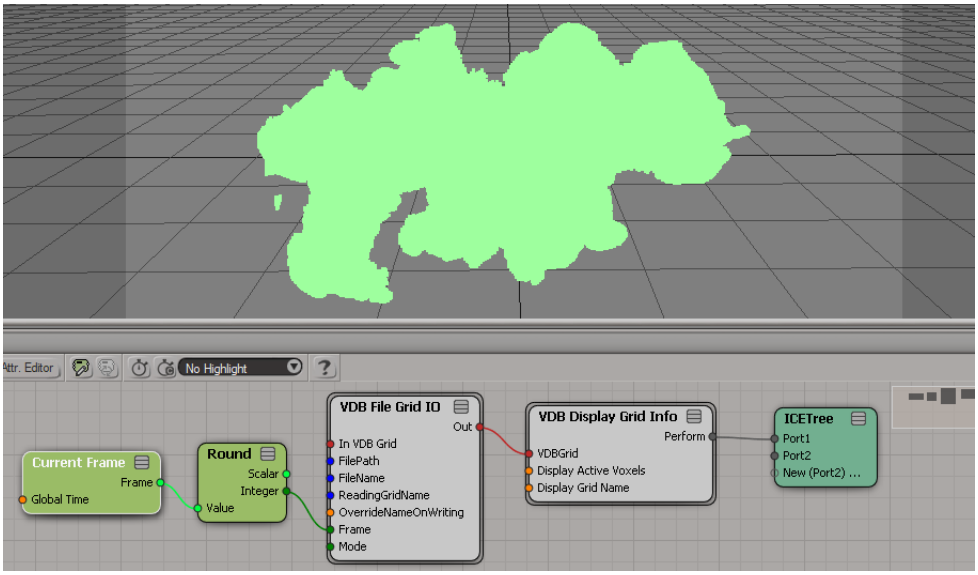


By default this parameter is turned off, because is some cases it increase the render time. Like in our case: rendering without deformation blur is spend one and a half seconds, but with deformation blur is nearly two minutes.

26 How to render OpenVDB

Suppose w have a scene with *.vdb file loaded by

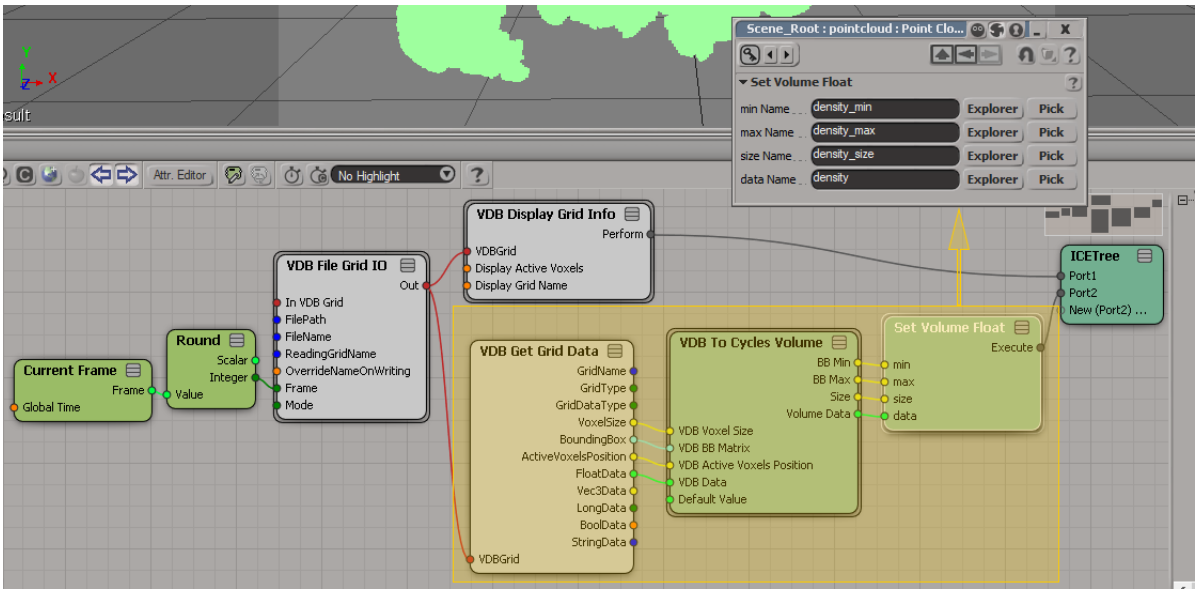
`OpenVDB for Softimage`.



In our case this file contains only one grid with the name `density`. To transfer the data about this grid to the render, we should create four attributes:

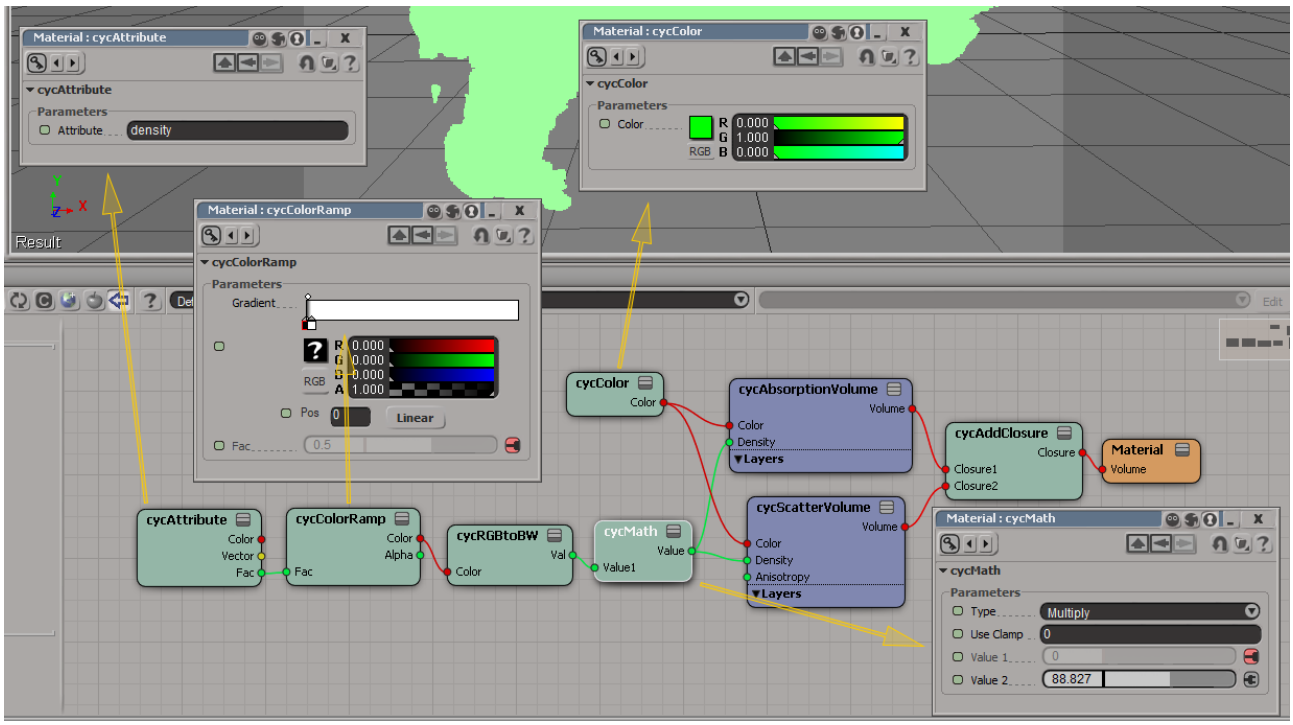
1. `density` with array of float-values, which define density of voxels;
2. `density_size` with one vector, which contains three integers. These integers define the number of voxels in each dimension. ;
3. `density_min` with coordinates of the minimum vertex of the voxel cube. All three coordinates of this vertex are minimal among all cube's vertices;
4. `density_max` with coordinates of the maximal vertex of the voxel cube.

The name of the attribute with data not necessary coincide with the name of the grid, but other three attributes should start with the same name and ended by `_size`, `_min` and `_max`.

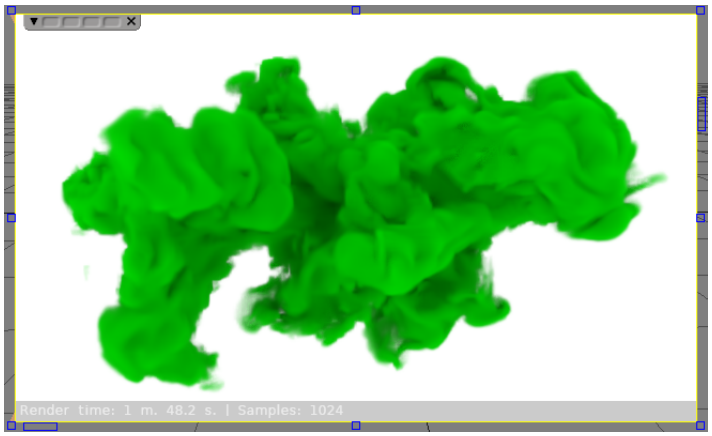


Get data from the grid by th node `VDB Get Grid Data`. Output port `FloatData` o this node contains all needed data, but it arranged in wrong order and there are some gaps in it for voxels with zero density. To restore the complete data we should use the node `VDB To Cycles Volume`. Next we create all desired attributes by the node `Set Volume Float`.

Assign to the object material like in the picture. The node `Attribute` is the most important because it allows the render to use stored volume data.

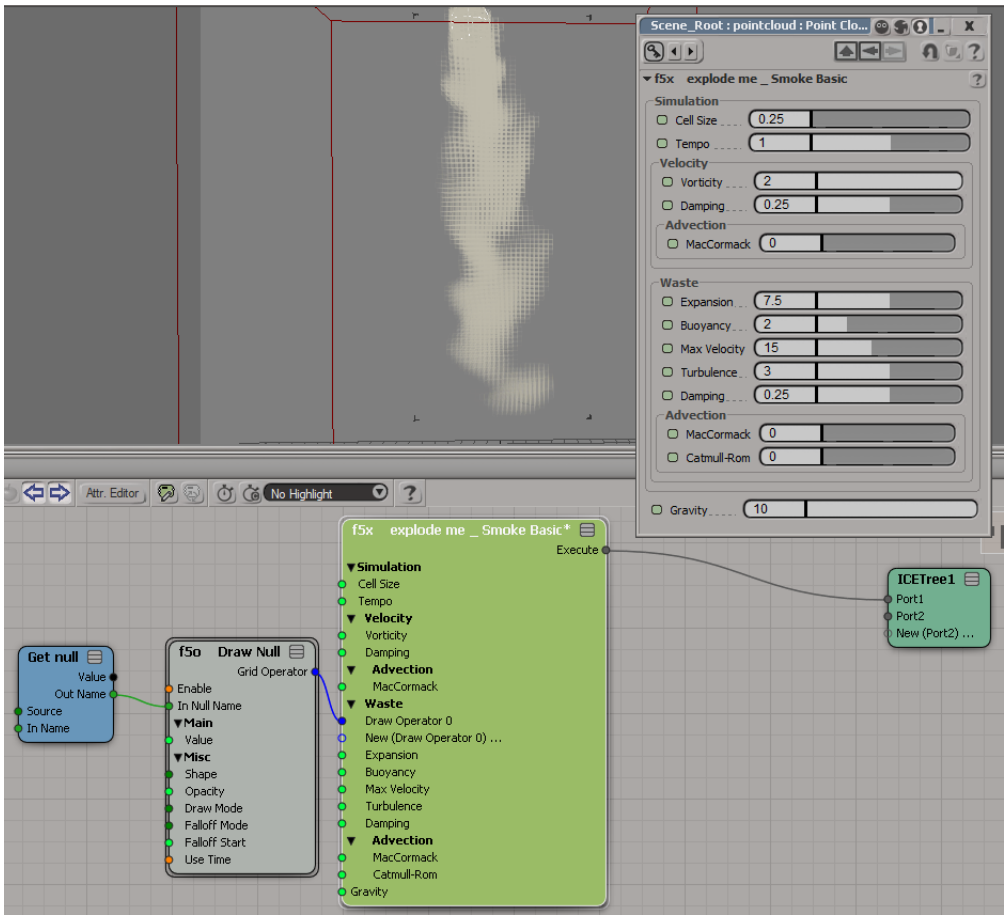


Render.

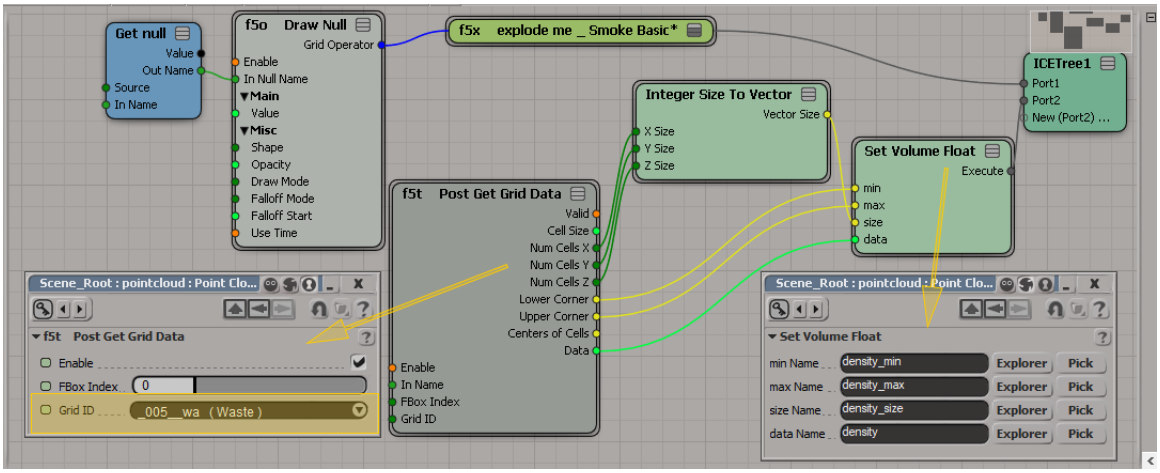


27 How to render emFluid

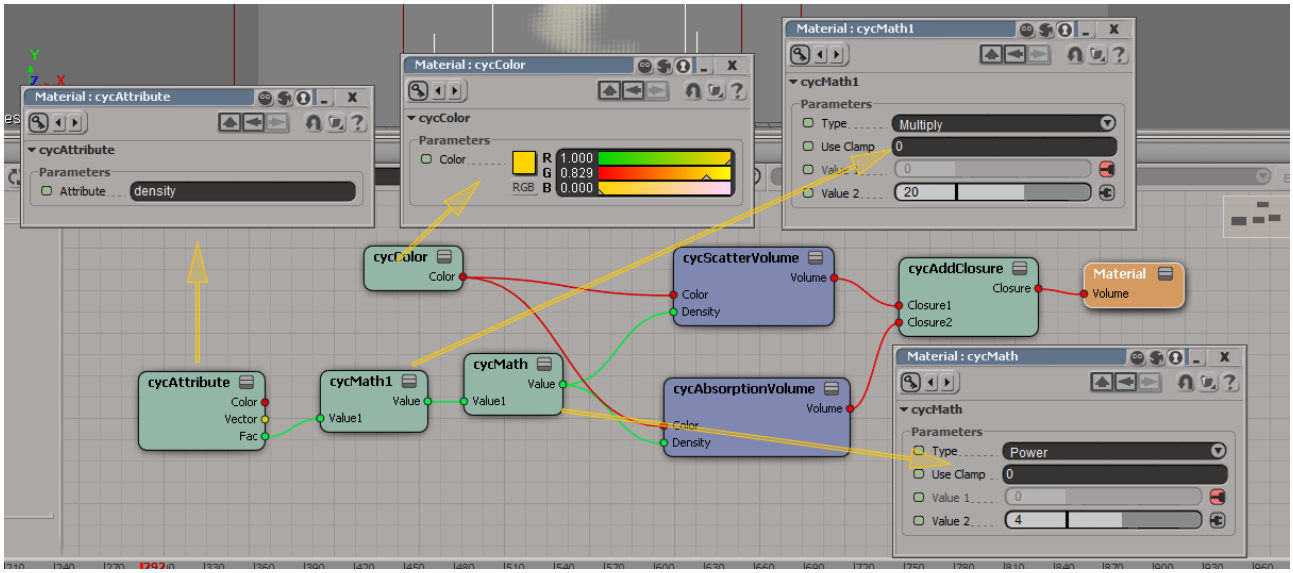
Suppose we have a scene with a smoke generated by `emFluid`.



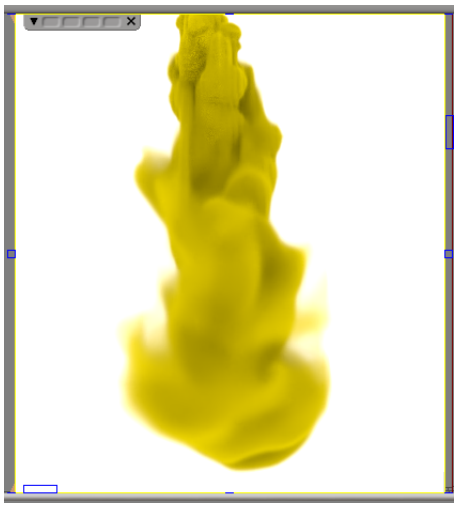
We can get the data by using the node `Post Get Grid Data`. As identifier of the grid we should use `Waste`, because in our case all simulated data stored in this grid. Record four attributes like in the case of rendering of the `OpenVDB`.



Assign the material to the smoke.

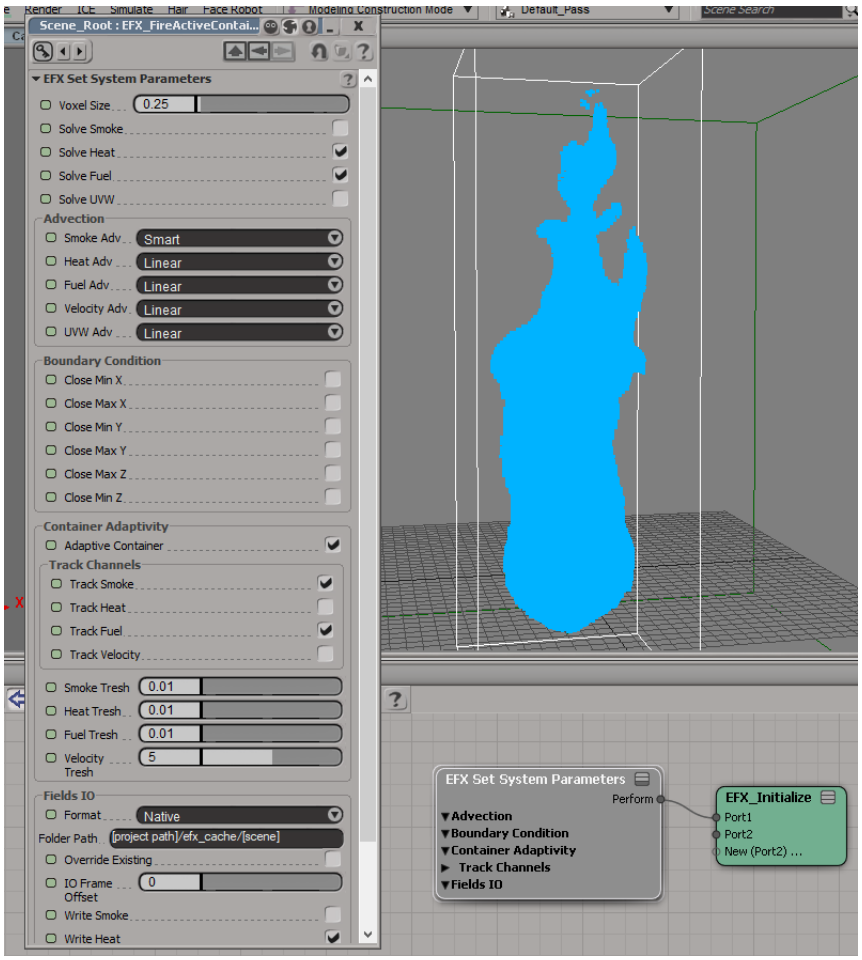

Material : cycAttribute
cycAttribute
Parameters
Attribute
density
Material : cycColor
cycColor
Parameters
Color
R 1.000
G 0.829
RGB B 0.000
Material : cycMath1
cycMath1
Parameters
Type
Multiply
Use Clamp
0
Value 2
20
cycColor
Color
cycScatterVolume
Volume
Color
Density
cycAddClosure
Closure
Closure1
Closure2
Material
Volume
cycAttribute
Color
Vector
Fac
cycMath1
Value
Value1
cycMath
Value
Value1
cycAbsorptionVolume
Volume
Color
Density
Material : cycMath
cycMath
Parameters
Type
Power
Use Clamp
0
Value 1
0
Value 2
4

Render.

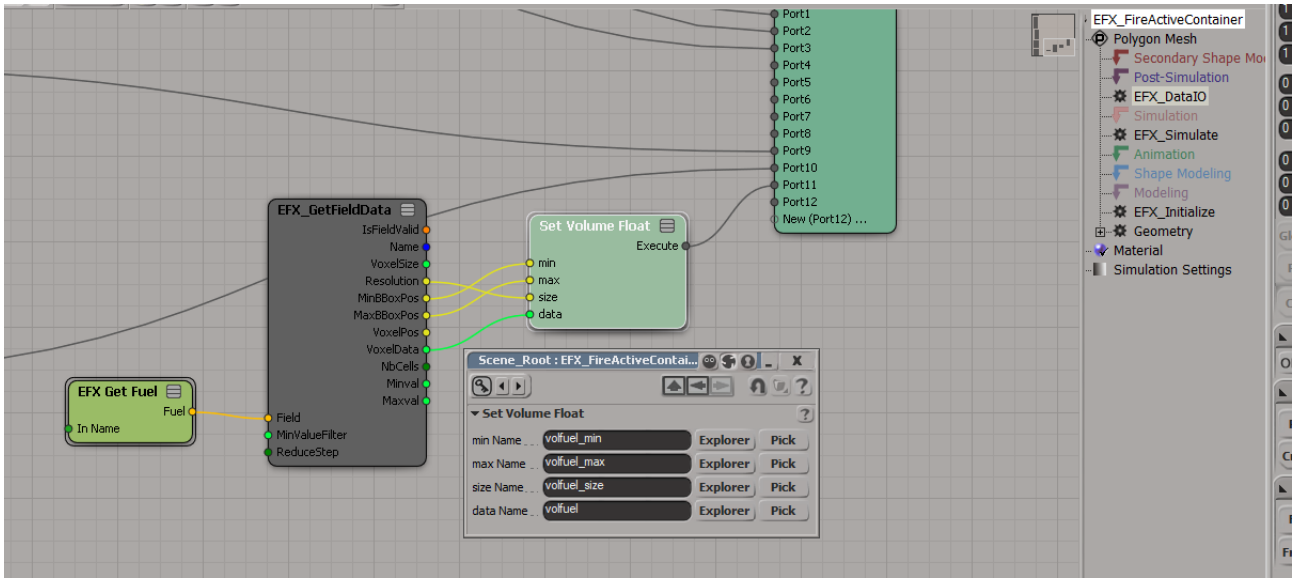


28 How to render Explosia FX

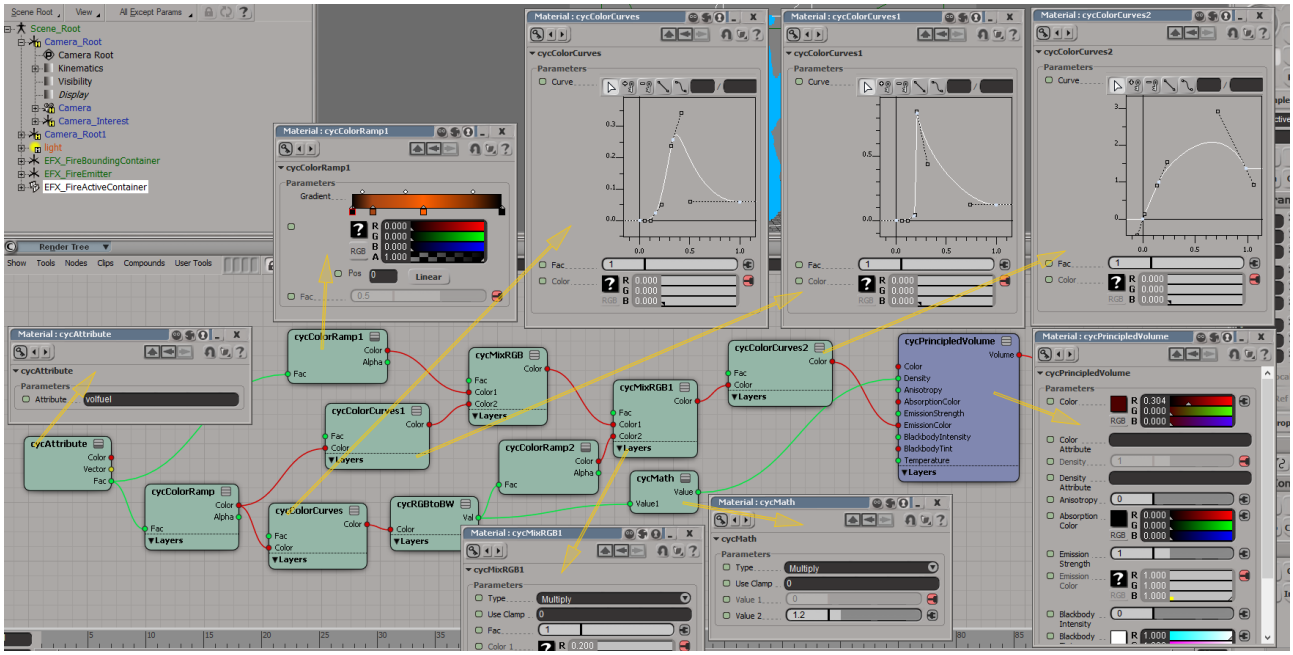
We will use the standard scene with fire.



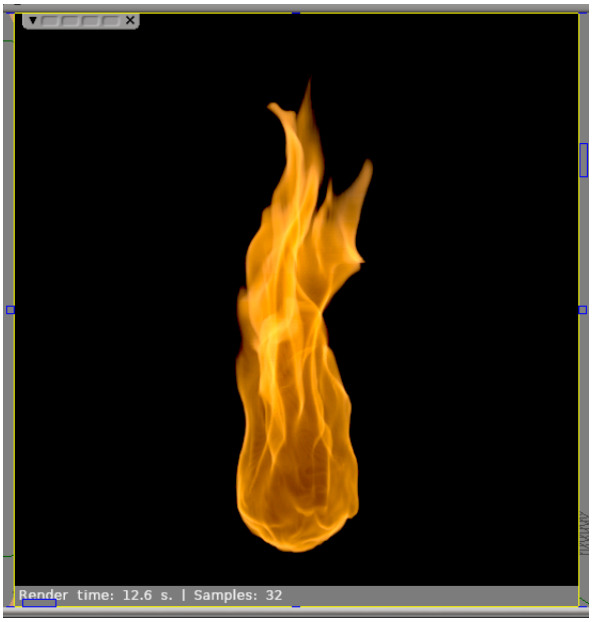
For the fire all desired data contains in the `fuel` attribute. We should get data from this attribute in `Post-Simulation` section of the ICE-stack. Use the node `EFX_GetFieldData` and connect it to the node `Set Volume Float` with the attribute `volfuel` (because the attribute `fuel` is busy by Explosia).



Use material like in the picture bellow. All nodes except illustrated has default values of parameters.

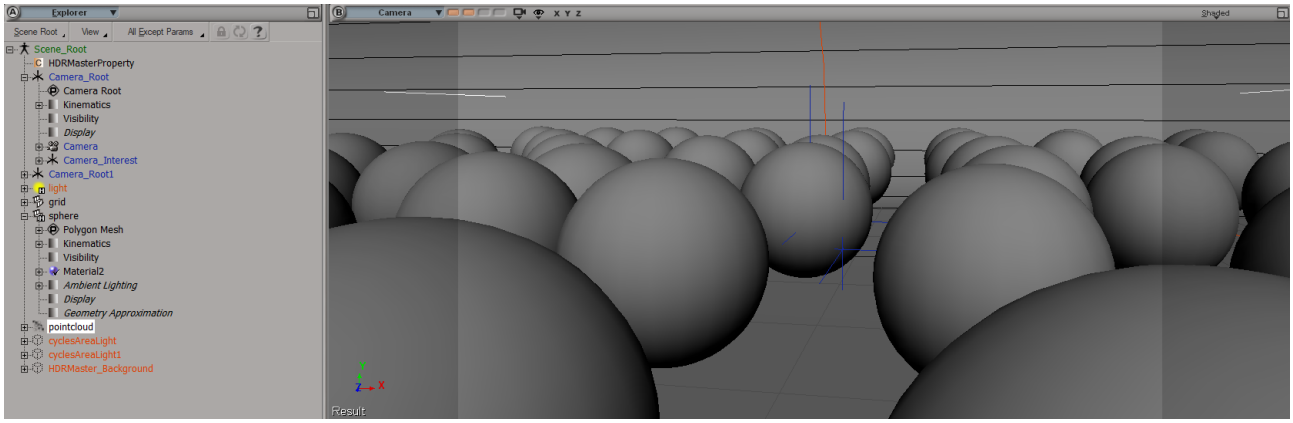


Render.

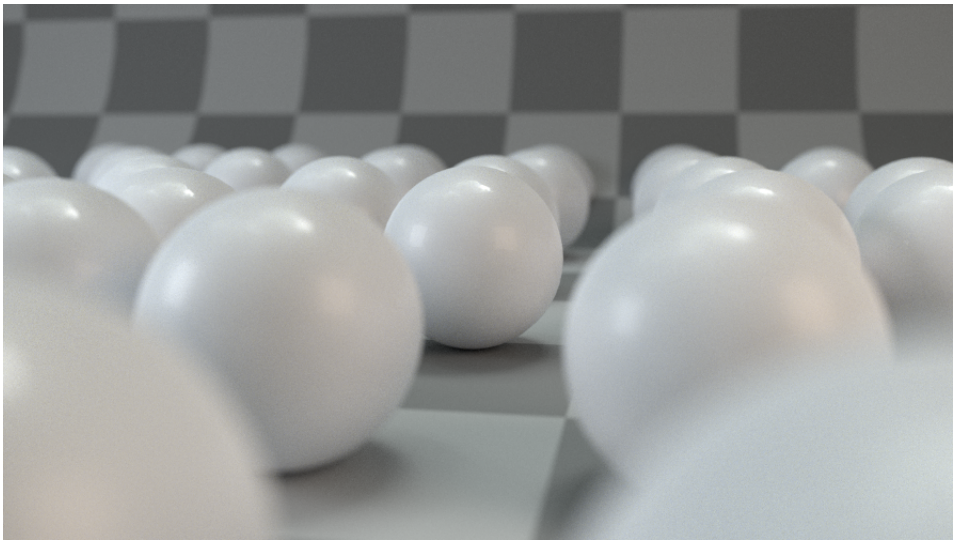


29 How to render and use Cryptomatte passes

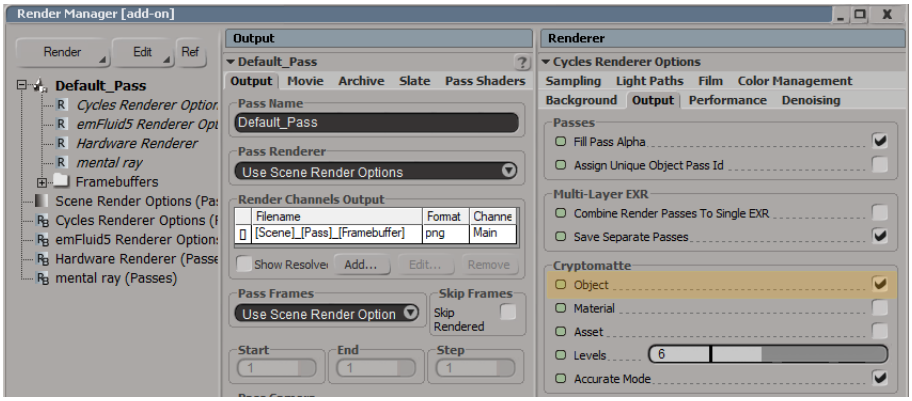
Suppose we have a simple scene with spheres.



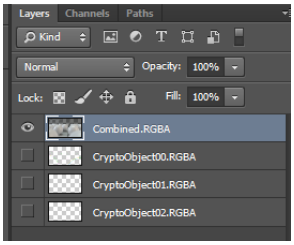
Render result:



To saving Cryptomatte passes, go to the tab `Output` of the render settings and turn on the parameter `Cryptomatte` – `Object`. This means that during rendering, information about where each object is located will be saved.



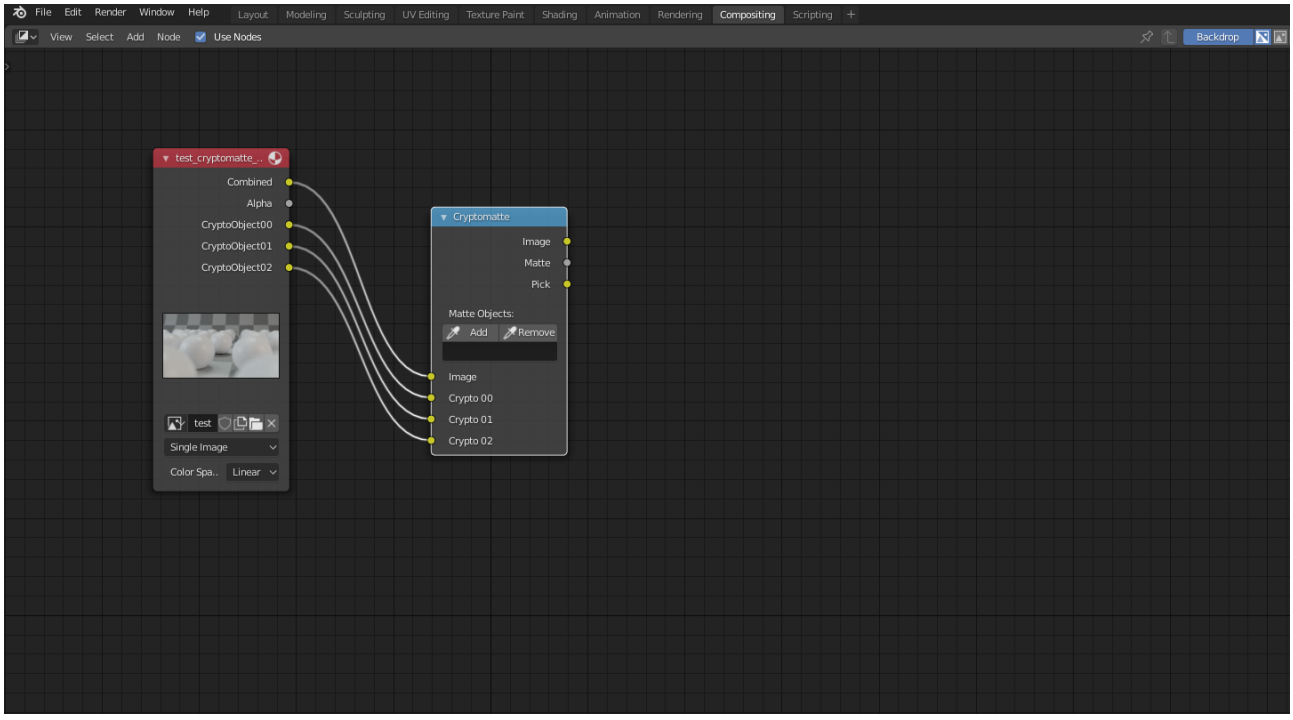
After rendering, an additional file with the extension `* .exr` will be created. The name of this file ends with the word `Cryptomatte` and it contains 4 layers: an image with the final render and three layers with information about the scene objects.



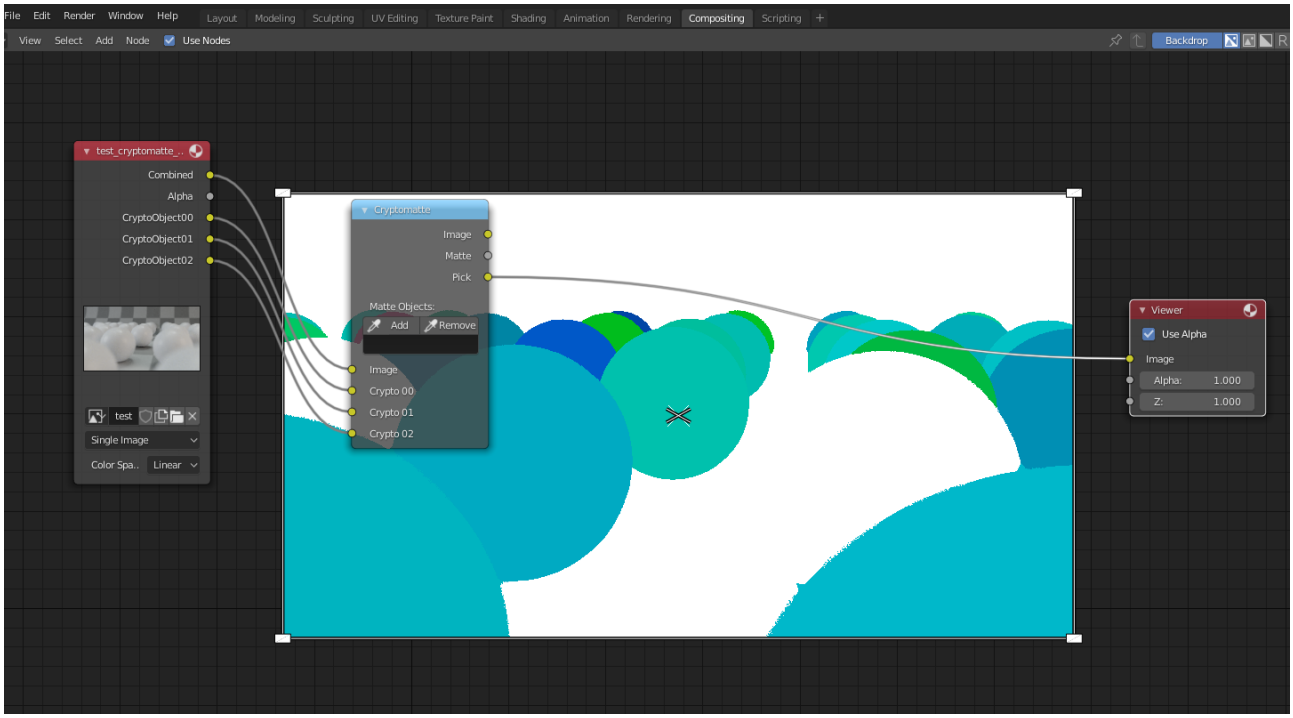
Now, as an example, let's try to use the rendered Crytomatte passes to change the color of one sphere. We will use Blender, although you can use any other composer.

Add the `Input` – `Image` node to the Blender working canvas and choose our `exr`-file:

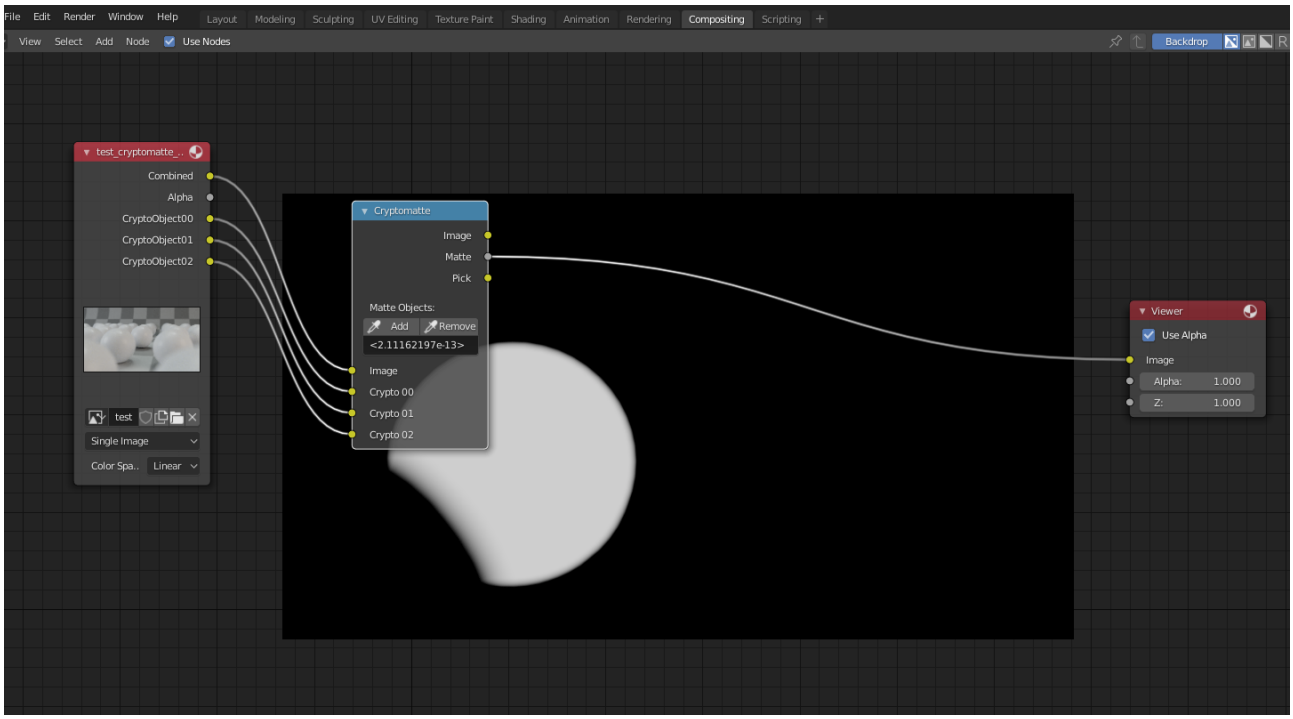
Add the node `Matte` – `Cryptomatte` and connect ports:



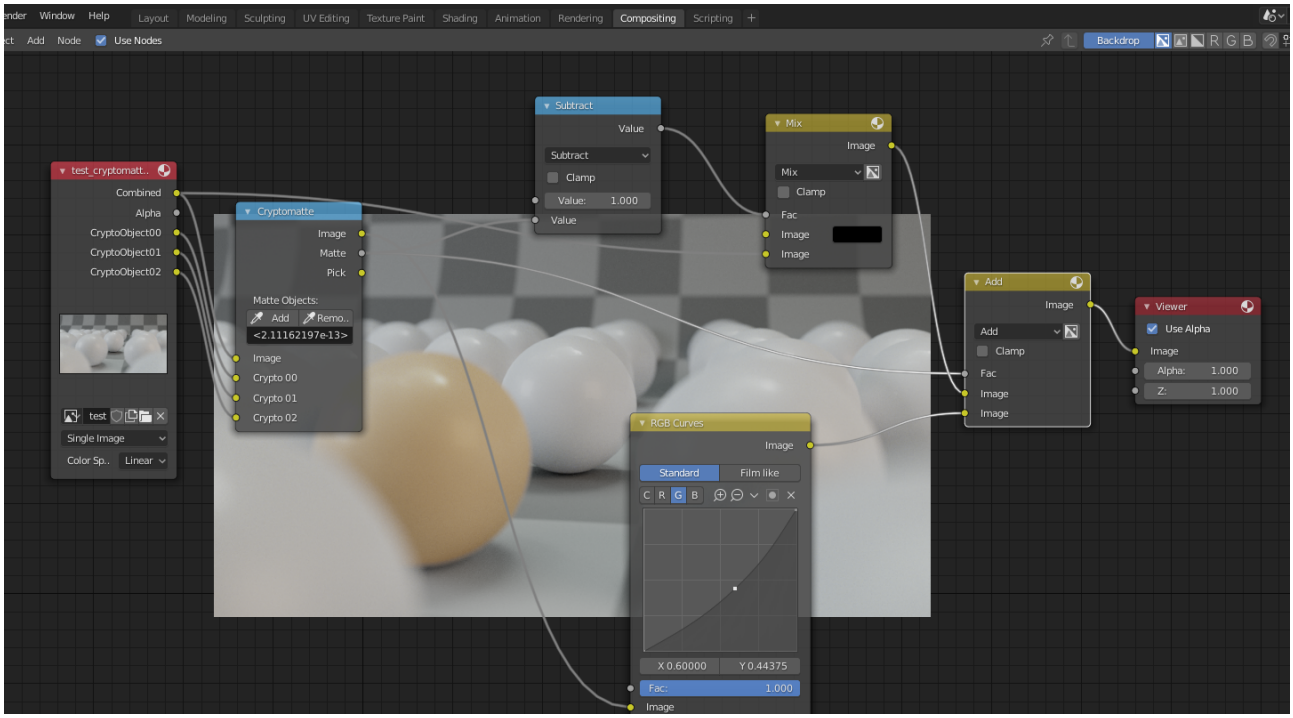
To see the information inside the file, add the node `Output` – `Viewer` and connect the `Pick` port to it:



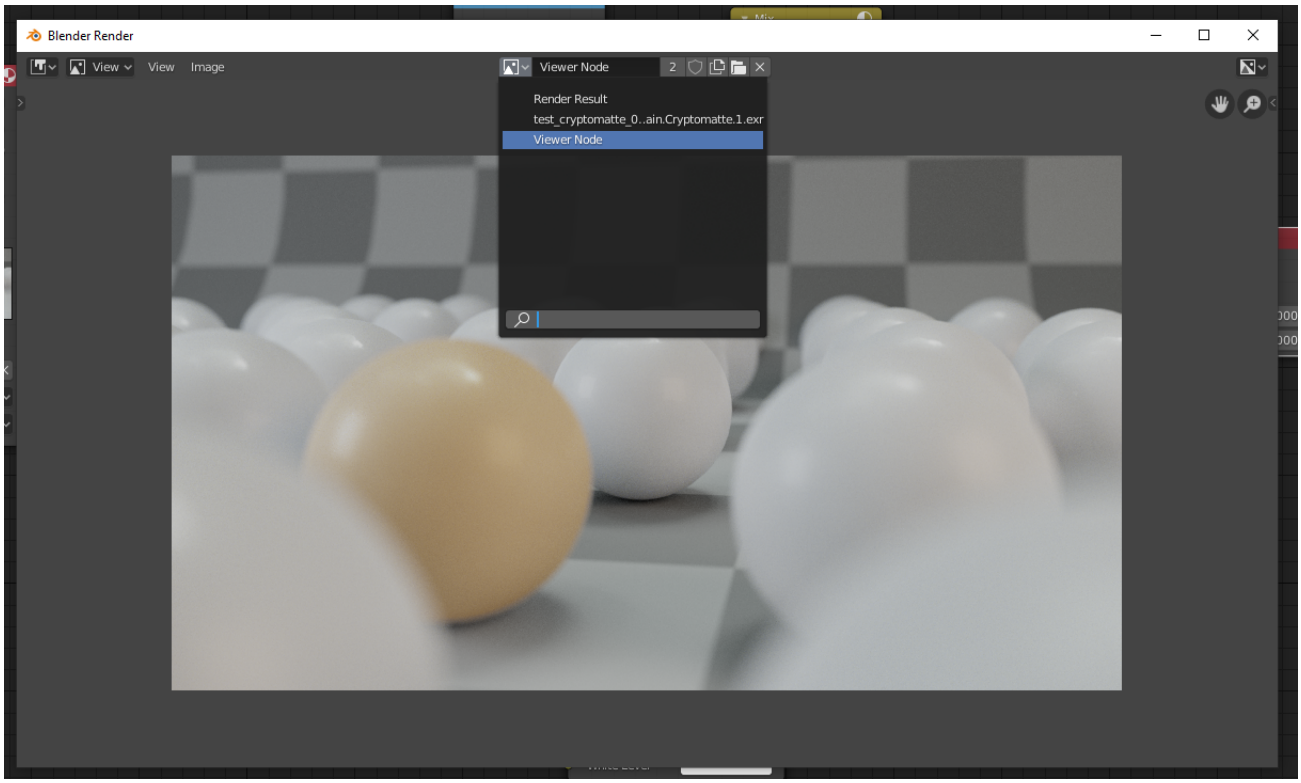
Now pick any sphere and look at the generated mask:



Add some compositing magic:



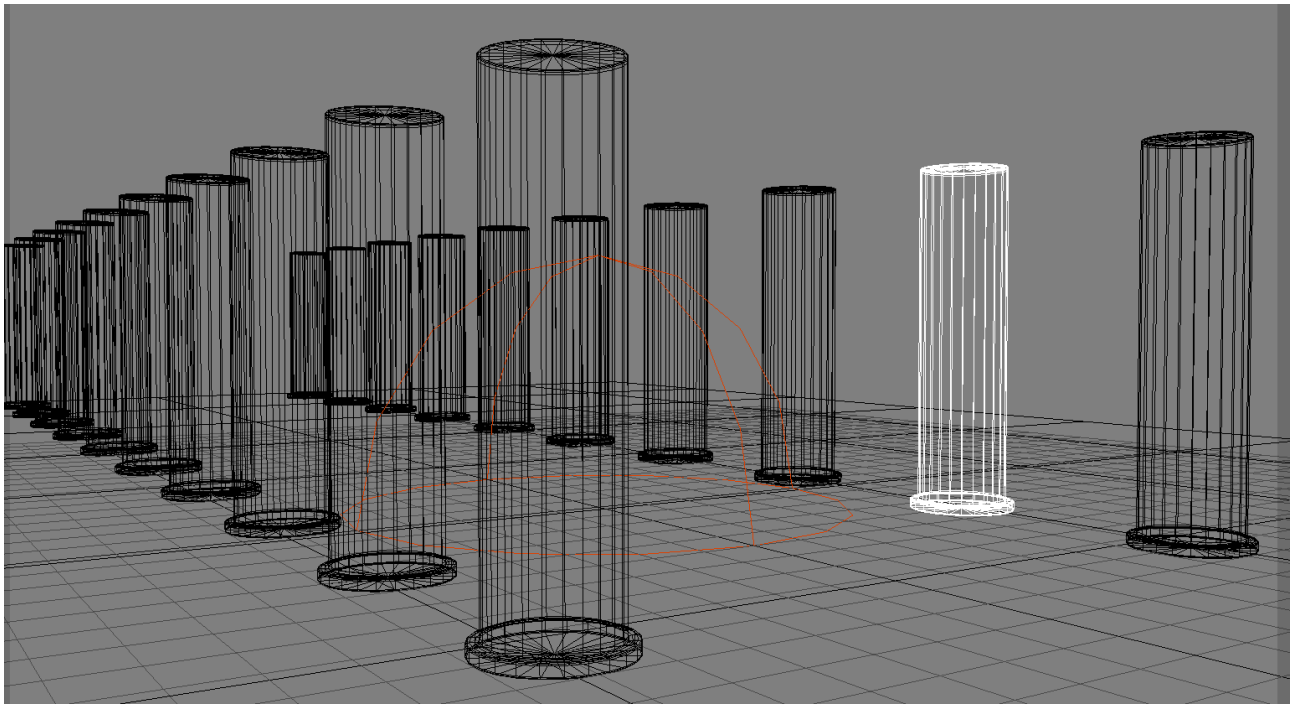
To save the composition result, open the window for the rendered results by pressing `F11` and selecting in this window `ViewerNode`:



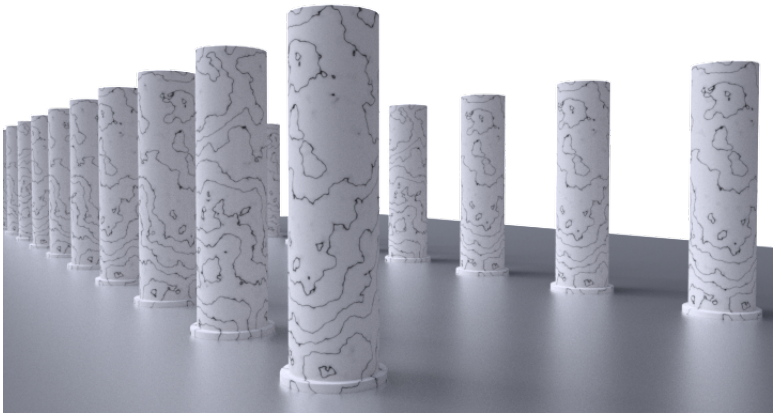
Press `Image` – `Save As...` and save the result.

30 How to render AOVs

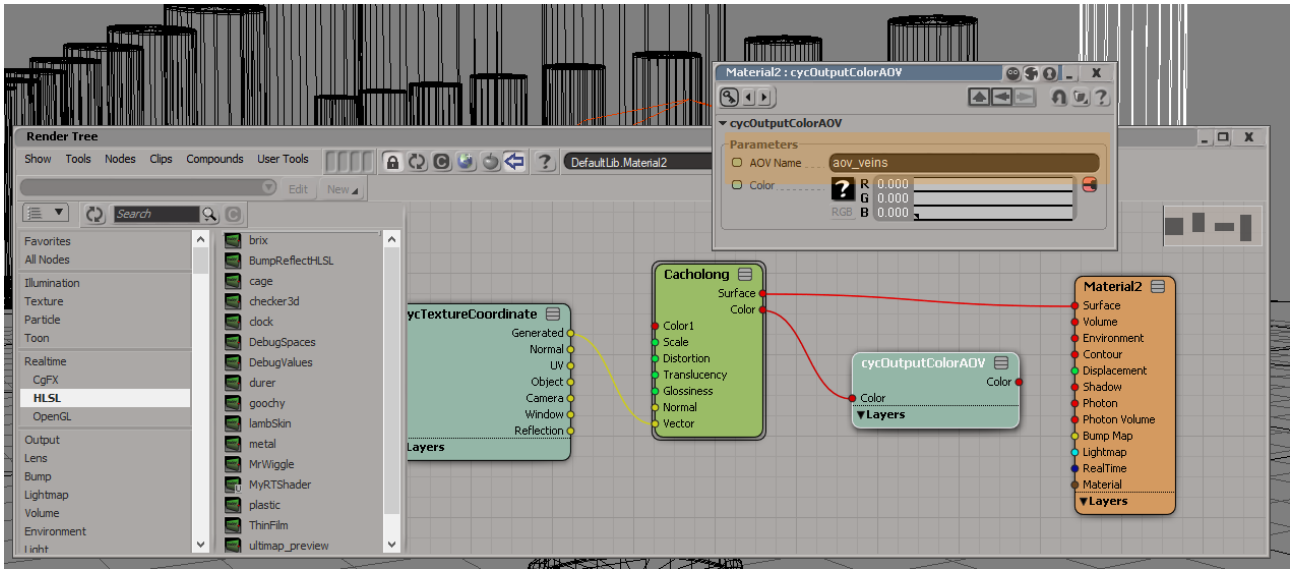
AOV (Arbitrary Output Variables) is a user pass, which we can use for store any data from shaders during rendering. Consider an example, where we have a scene – columns on a plane.



Material of the columns looks like a stone

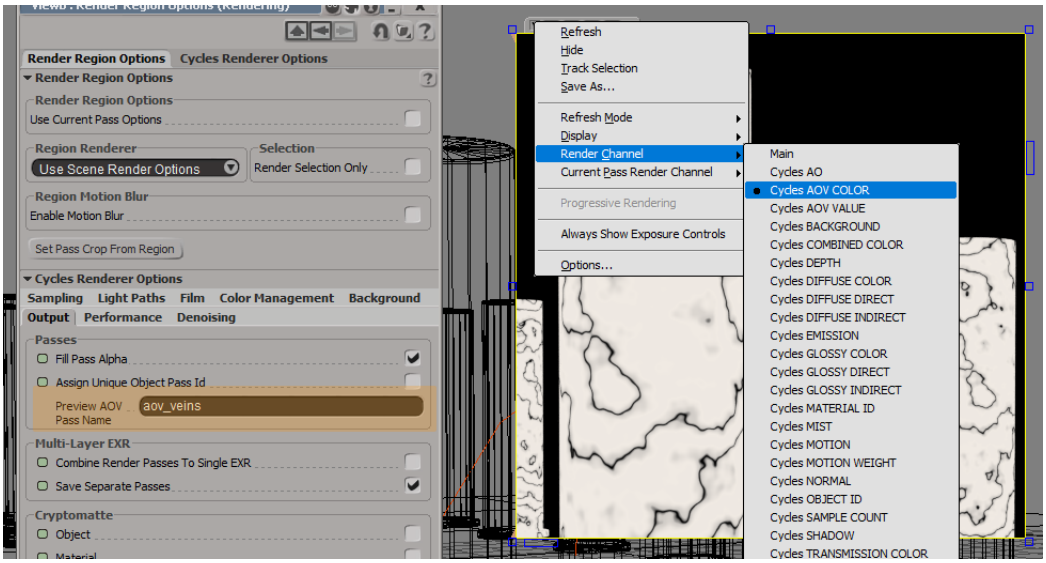


We would like to save the black veins from the texture into a separate pass. To do this, add the node `cycOutputColorAOV` to the material of columns and connect to its input port `Color` port of the stones shader node, which contains a mask of these veins. Of course, what connect to the input of the node `cycOutputColorAOV` we should prepare before. Call the pass by `aov_veins`.

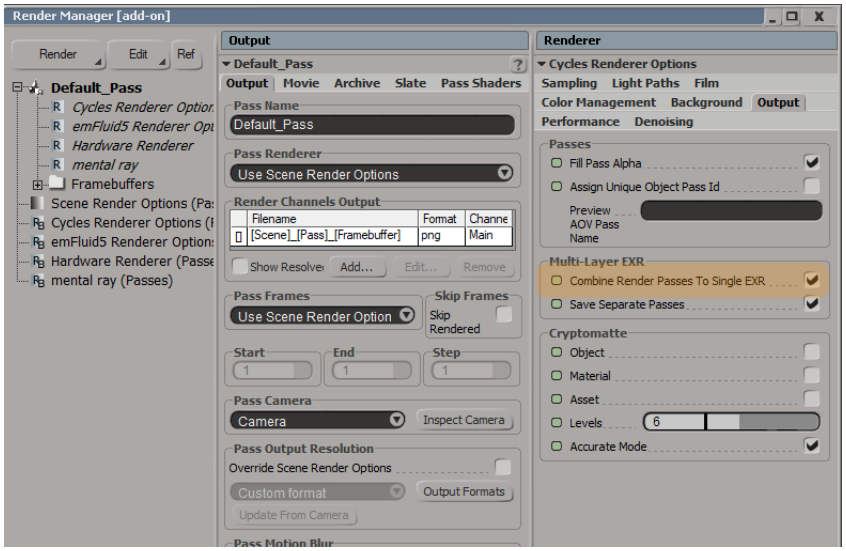


Connect the output of the node `cycOutputColorAOV` somewhere longer is not necessary. But you can also connect to where the output data port should be connected. In this case, the node will be used as a simple passthrough node.

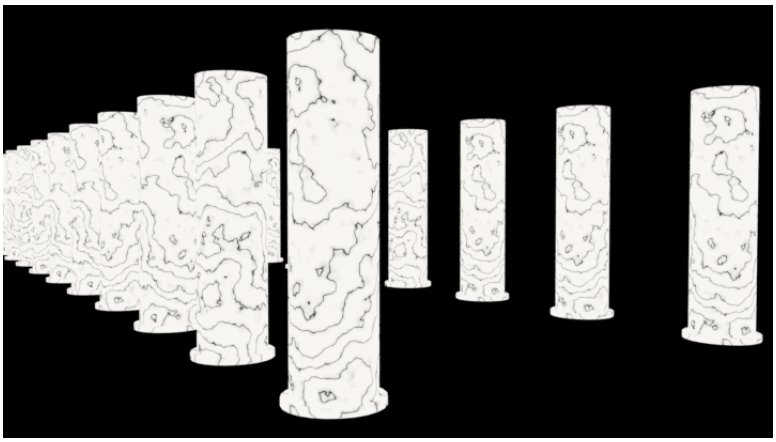
We can view the content of AOV passes in the quick render window. To do this, select `Cycles AOV COLOR` in the `Render Channel` of the preview window. In general, we can store information in many different passes, so the render does not know in which one to show. It is necessary to set the name of the AOV pass to the parameter `Preview AOV Pass Name` in the `Output` tab of the render preferences.



The only way to save all AOV passes as a picture is to use the function for save all output passes as single multilayer exr file. To do this, turn on the parameter `Combine Render Passes To Single` in the `Output` tab of render settings.



The result:



All the same works for the case when we want to save in the pass not a color, but a numerical value. The only difference is that we need to use the `cycOutputValueAOV` node.