

## Installation

hook5 files must be installed (unpacked) in the Binaries folder of the game. Starting the game should be done from the Binaries folder, using either the Launcher or the executable. The installation of Physx binaries is required (**physx3\_x86.dll**, **physx3common\_x86.dll**, **physx3cooking\_x86.dll** – for further information see the links provided in the hook5 readme), without physics, hook5 will not work even if physics objects in-game are not used, the reason being is the implementation of the Physx connection (static linking). You must also install the **msvc 2010 (x86) redistributable** and **msvc 2012 (x86) redistributable** packages.

Optionally, you can install HBAO and HairWorks binaries – they are necessary for the Nvidia HBAO+ effect and the hairstyles using Nvidia HairWorks, but they are not required for hook5 to function and if they are not installed HBAO and HairWorks will not be available.

If you want to use the PhysX and HairWorks hairstyles, then you need the basic addons (modsgarden, addons) of these hairstyles and hair replacers (modsgarden, hook5 extra files) for the PhysX and HairWorks hairstyles to work properly. I also recommend downloading and installing the starter content for hook5, which contains examples of rooms, clothes, as well as examples of body textures.

## Difference between Basic and Extended versions of hook5:

Functions	Basic	Extended
PBR Shading	Yes	Yes
IBL Shading	Yes	Yes
HDR	Yes	Yes
Pass file support	Yes	Yes
Support for level_definition and lighting	Yes	Yes
Auto reload of settings and in-game rendered GUI	Yes	Yes
Outline	Yes	Yes
Subsurface	Yes	Yes
DOF	Yes	Yes
Glow	Yes	Yes
Bloom	Yes	Yes
SMAA	Yes	Yes
Sharpen	Yes	Yes
POM	No	Yes
CustomizableSkin	No	Yes
hook5 objects: statics, fire, water, mirrors with in-game editing	No	Yes
Auto reload of textures and pass files	No	Yes
Auto reload of level_definition and in-game editing	No	Yes
Tessellation	No	Yes
Displacement	No	Yes
PhysX support	No	Yes
HairWorks support	No	Yes

## Main keys (can be remapped)

F2 – Cycles through and displays bound statics, bound lights, Physics bounds.

F3 – Enable / Disable stereo / BP display.

F4 – Enable / Disable hook5 main menu.

F9 – Update the environment cubemap.

F11 – Shader Recompilation.

PrintScreen – Takes a screenshot of current scene.

Ctrl + PrintScreen – Takes a high resolution screenshot.

Double click the left mouse button – Select / deselect hook5 objects.

Middle mouse button – Manipulation of hook5 objects.

## FAQ

### 1. Why does the model have green eyes? (The absolute bestseller!)

Turn off Eyes Specular under the Body Tab of the Customizer menu.

### 2. Why does the game crash at startup?

- A) Check that all required components are installed (Take a look at the hook5\_readme.txt provided by the hook5 update.).
- B) Run the game only from the binaries folder (Either Launcher or executable).
- C) Check the contents of the file mainfx\_errors.txt for error messages. If the file contains the message “KEY check failed”, then this means that changes have been made to the PC components and the key is no longer valid: you will have to obtain a new key by contacting hook5patreon. Error messages containing LUT textures can be ignored as long as you do not use the settings of the Cartoon section to modify the diffuse and specular light components.

### 3. I downloaded a PhysX hair (PhysX / HairWorks) and they don't appear in-game or they appear but float in the air in the center of the room, what is going wrong?

All PhysX / HairWorks hairstyles require the installation of basic hair addons (modsgarden / addons, addonsscripts) and the installation of special hair replacers (modsgarden / hook5 extra files). All replacers were made with game version 7.5 but also function in version VX.

### 4. Does hook5 work with Version VX of the game?

I tested hook5 with version VX of the game and at the moment of testing hook5 didn't show any errors. The main functions of hook5 work.

## 1. General information.

Hook5 is a graphical plugin that intercepts 3DSVila2's DirectX8 API functions calls and converts them to DirectX11 API functions calls, adding advanced rendering features to the game. Advanced rendering can be divided into three groups: advanced rendering for game objects, rendering Hook5 objects (light sources, statics, fire, water, PhysX and HairWorks objects) and post-effects rendering. Hook5 renders the scene using deferred lighting technology which is based on Physical Based Rendering (PBR) theory thus certain requirements have to be met for the data of objects being rendered: information about Normals and TBN (Tangent, Binormal, Normal – The basis of tangent-binormal space for an object's polygons) and information about the object's surface material (metal characteristics, glossiness, glow, surface details). Since we can't bind additional data to the game's resources by using the game functions, we use dynamic data binding via pass files: when the game calls the DirectX8 functions to create the texture stored in the ActiveMod or ActiveMod\subfolder folder, hook5 intercepts these function calls and performs an additional check for the pass file of the same name. If such a file exists, hook5 associates the texture with the pass file data and subsequently, this data is used every time the game renders this object. Herein also lies the reason for the main limitation of hook5: we cannot affect the appearance of textured objects which are not located in ActiveMod, since it is not possible to bind additional data to such objects for rendering.

### 1.1 How does the game engine render the image?

Each game engine is arranged in another way, so it would be not be very clever to lay out a certain frame processing sequence, hence I will roughly describe three main steps:

- **First step:** Processing scripts, physics, user actions, etc.
- **Second step:** Drawing objects of the scene.
- **Third step:** Apply postprocessing effects.

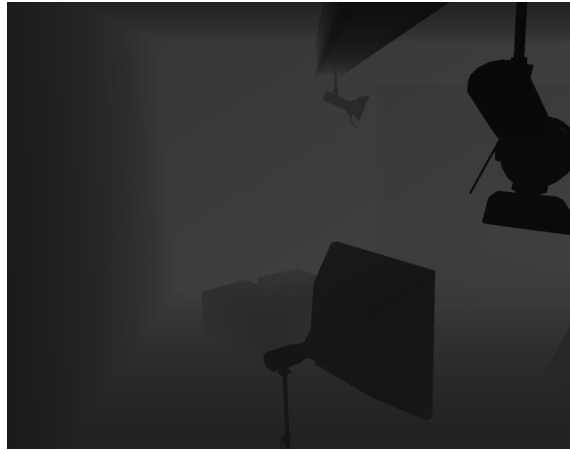
I will not talk about the first one, although this stage is also present in hook5. I don't think that this information will be relevant to you. The construction of shadows and the rendering / construction of the scene are performed at the second stage, so each object that constitutes the scene and is captured in the current frame is sequentially drawn. Depending on the type of rendering (Forward Rendering, Deferred Rendering), the rendering is done in different ways – either directly with the calculation of light (Forward Rendering), or the lighting is rendered after the objects are drawn (Deferred Rendering) followed by the rendering of translucent objects utilizing Forward Rendering even if Deferred Rendering was used initially. In both cases, the processing of object textures takes place – diffuse, normal or relief textures and the textures responsible for the ability of the object's surface to reflect light (usually called a specular texture).

Postprocessing effects are applied to the scene at the third stage. Almost all the post effects can be compared to picture enhancing effects of Photoshop: the picture rendered by the second stage is taken and a shader effect is applied. This altered picture is the basis for the next shader effect and, after being applied, is again the basis for the next shader effect. This cycle repeats until all shader effects are applied. Finally the picture is drawn on screen.

### 1.2 What is a depth buffer?

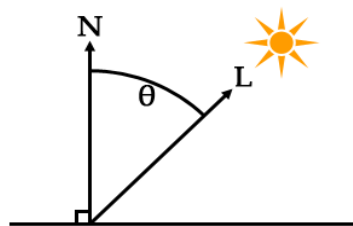
A depth buffer is a special texture in which the distance of each pixel from the camera of the current frame is recorded. In old game engines, such a buffer was built automatically, but access to it was either absent or was very slow. Through the evolution of graphics, it became possible for shaders to use the depth buffer as a regular texture, reading information from it using standard shader functions, without any diminishing effect on performance. Due to this, various advanced shading techniques and post effects have become possible: Deferred Shading, all Ambient Occlusion techniques, Outline, Screen Space Local Reflections, Depth Of Field and many

others. Currently, the depth buffer in game engines plays an important role in the construction of the final image.



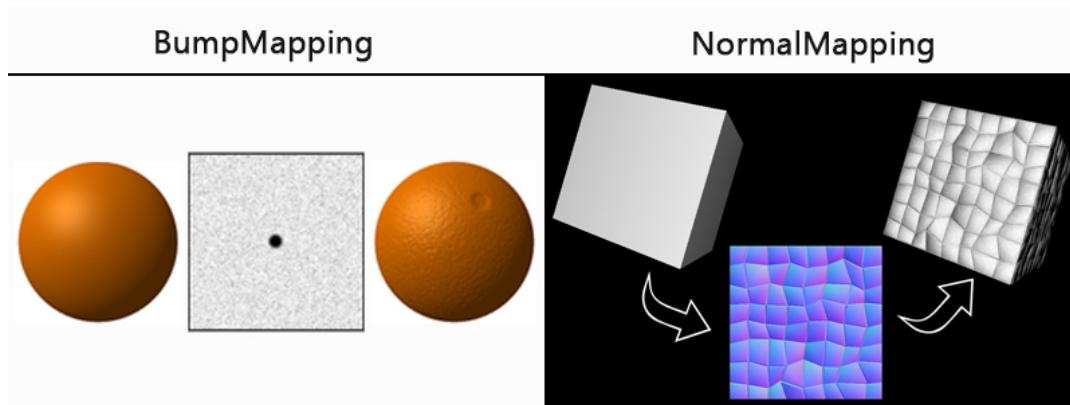
### 1.3 What is Normal Mapping ?

Depending on a certain background, the illumination of the surface depends on the angle between the normal of the surface and the inverse vector of the incidence of light. Despite the fact that modern mathematical theories behind the calculation of the illumination' intensity have become much more complex, the Phong function is still used in the majority of those light calculations.



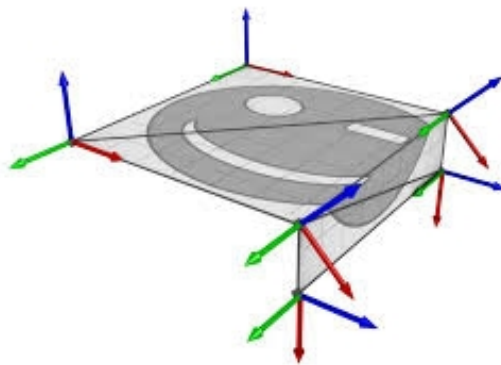
In very old games, the lighting was pre-calculated (lightmaps) and after the first shaders appeared, vertex lighting was used. This created a rather uniform distribution of illumination on the polygon's surface and there was no relief as such. By the time it became possible to implement pixel lighting, an idea arose to use textures to define the surface curvature of a flat polygon.

Bump Mapping was the first technique that existed. By using a texture which represented the height of the relief, the pixel shader, by calculating the difference in height of the neighboring pixels, determined the deviation of the polygon normal from the baseline. The modified normal was then used to performed the illumination calculation. The successor to the Bump Mapping technique was Normal Mapping, which is still used today. The texture (RGB images) utilized in Normal Mapping doesn't contain the height of the relief, but a modified normal. Every normal stored in the texture is a single  $(X, Y, Z)$  vector in 3D space of which the  $x$ ,  $y$  and  $z$  component can range from  $-1$  to  $1$ . It was also possible to keep only positive values and the normal was stored in the texture by converting the normals to RGB values, which range from  $0$  to  $1$ , with the help of the function  $RGB\ Normal = Normal * 0.5 + 0.5$ .



In addition, Normal Mapping should contain and transform to changes made to the basis of the model in the TBN polygon space – transformation matrix from the textural space (tangent space) to the polygon space (model space) – since the normal in the texture is always directed at the viewer, and the polygon can be oriented arbitrarily on the model. Imagine a cube which sides are covered with the same texture – the normal in the texture, for example  $xyz = 0,1,0$  (up) without TBN transformation will be the same for each side: it will be directed upwards on the left, right, top, bottom, front and back side of the cube. A transformational TBN matrix allows us to transfer the normal to the corresponding space of the polygon. For instance the normal  $0,1,0$  (up) changes to  $-1,0,0$  for the left side of the cube,  $0, -1,0$  for the bottom side of the cube and so on .

TBN bases calculated for the model vertices:



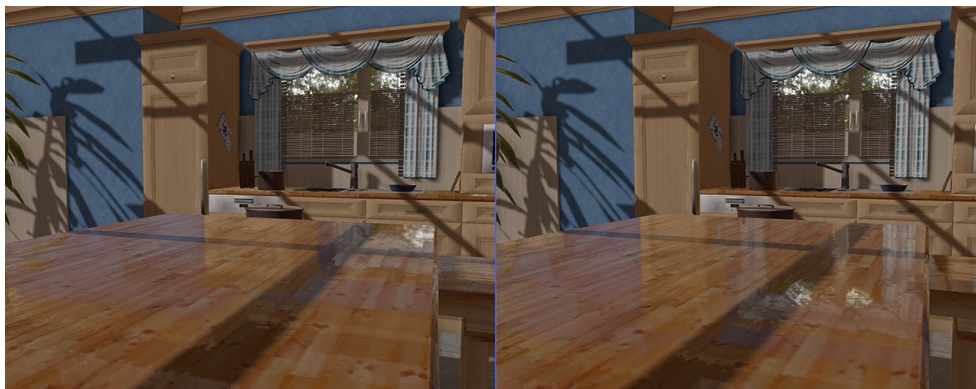
The problem is that the 3DsexVilla developers either did not know about Normalmapping, or did not intend to use it. Therefore the game models do not contain information about TBN and the game basically does not work with Normal Mapping. In order to make it possible to use normal textures, I had to use a fake TBN generation method for the game's models. The method is based on partial derivatives of positions and texture coordinates and although it gives acceptable results in most cases, in some it leads to artifact due to incorrect TBN computations and, as a result, incorrectly calculated lighting: the diffuse can be overly dark or the specular overly bright. As a rule those artifacts occur mostly when the models' polygons are heavily distorted during animations. Unfortunately, I have no solution for this problem, at least not for now. Hook objects, unlike game objects, use an "honest" pre-calculated TBN, so even PhysX hairs do not have such artifacts.

Depending on which application generated the normal texture, you may need to invert the Green channel of the normal texture – in hook5 the 3dmax rule is used, where the positive Y direction of the normal components in the texture space is up, whereas, for example, in Maya the positive Y direction is down. Simply put, if you get an inverted relief on an object, you just need to invert the green channel of your normal texture.

#### 1.4 What is a Cubemap and what is the purpose of ParallaxCorrection?

Cubemap is a texture containing 6 images in the directions to the right, left, up, down, forward and backward (in our case “back” would be more appropriate). Textures of this type are used to create information about the environment at any point / area of the scene. Previously, cubemaps were mainly used to display long-range level planes like the sky, mountains or the horizon. Later they began to use only diffuse illumination from the environment to imitate environment lighting. In this case, cubemaps could be small, they couldn’t be very bright and didn’t have much detail. Still later, the IBL (Image Based Lighting) technique appeared which uses two cubemaps: one to simulate specular lighting and another to simulate diffuse lighting. The texture to simulate specular lighting contains several levels, each of which is previously blurred in accordance with the PBR lighting theory. Hook5 utilizes cubemaps in three cases: the first and foremost case is the dynamic generation of ambient lighting maps for IBL using the current scene / room, the second one is drawing the map as SkyBox / SkyDome for the scene and the third case is the use of a previously prepared cubemap to simulate ambient lighting. Cubemaps are different from the content of the scene. One side of the Cubemap is just an image in one direction, it does not contain information about distance and depth. Furthermore, the IBL technique doesn’t take into account distances to objects and the illumination depends only on the normals and roughness of the objects’ surfaces: the reflections in the room won’t be affected by different objects which surfaces are oriented in one direction, but are located in different parts of the scene. The Parallax Correction technique is integrated into Hook5 (it can be turned on / off in `_level_definition.txt`) to correct reflection errors by modifying the reflection vector to get more correct reflections. This technique requires Denmark bounding volumes (EnvironmentBound) which is an approximation of the volume of the stage.

The difference between reflections with and without using ParallaxCorrection:



However, there are two drawbacks concerning ParallaxCorrection:

1. ParallaxCorrection can’t improve reflections in all cases and should be applied according to the situation.
2. When ParallaxCorrection is on, objects that are outside the EnvironmentBound will receive distorted reflections, as the technique corrects reflections for objects located inside the bounding box volume.

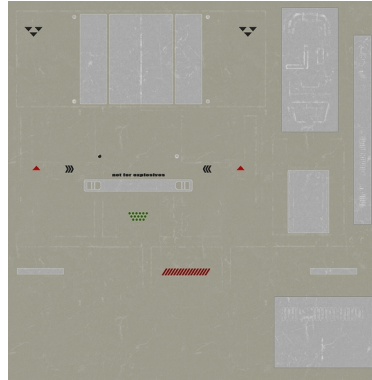
#### 1.5 What is Physically Based Rendering (PBR)?

Since the surface of an object / model is complex, i.e. it can consist of several materials (e.g. a machine with a wooden and a rubber end, a leather boot with a rubber sole, metal lace loops and laces made out of cloth), textures are used to define PBR data (an image file). PBR data is divided into 3 (4) textures:

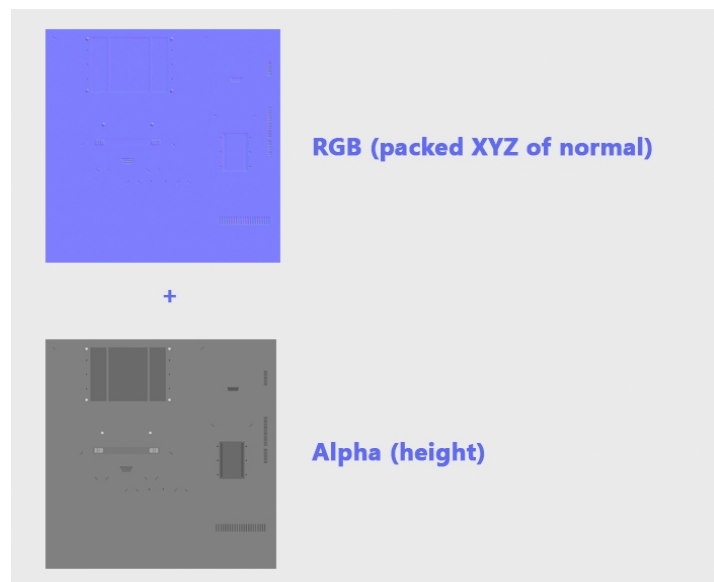
### 1.5.1. The diffuse texture – defining the main color of the object.

Diffuse texture should not contain any information about lighting, i.e. no shadows and no glare. In contrast to old game engines, where the diffuse texture also contained information about an object's lighting, only the "pure" color of the object's surface should be contained in the texture.

Example of a diffuse texture:



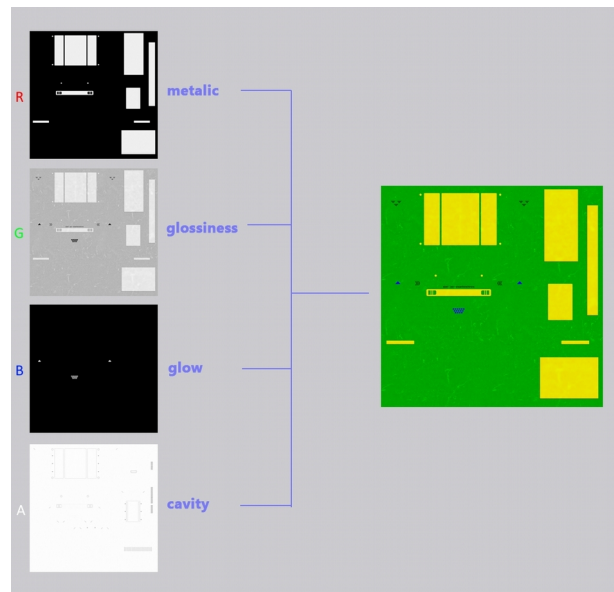
### 1.5.2. Normal texture and height map – defining the quality of the object's surface.



The normal textures' RGB channels contain X (red), Y (green) and Z (blue) components of the vectors of an object's surface, the alpha channel contains information about the height of the surface relief which is used in POM (Parallax Occlusion Mapping and Bump Offset).



### 1.5.3. PBR / Specular texture – giving the object a realistic appearance.



A specular or PBR texture contains the object's basic surface parameters necessary for the calculation of the object's illumination – metallicity, glossiness, luminosity and cavity (ambient shading of the relief). Many times I read sentences by users like "I made a green specular texture and got what I wanted" or "I painted the specular texture in orange and got a metallic sheen" on modsgarden's forum. This is an absolutely wrong approach to understanding PBR textures: each channel contains its own data type independent from other channels, but in the final render of the image they all work together – you should not be worried about the final color of a specular texture, you should be worried about what the channels contain. To alleviate this misconception let's take the red channel (metallicity) as an example: the red channel of the specular texture should be either white or black at a given point / surface area of an object if the surface at this point or area is metallic ( white) or non-metallic (black). In cases of rusty or painted metal, gray should be used instead of white, but remember to always use black for non metallic materials.

Red Channel (metallicity) – As noted before, starting from the colour black the more the colour of the channel reaches white the greater the metallicity.

Green Channel (glossiness) – Regardless of whether a part of an object's surface is metal or not, this channel determines the glossiness of an object, i.e. its ability to reflect light. Whereas white defines maximum reflection / specular gloss, black represents a minimum reflection / specular gloss of the surface and gray gradations represent intermediate variants.

Blue Channel (light intensity) – Defines the Glow effect's intensity for an object' surface. The final colour is determined by the colour of the diffuse texture multiplied by the intensity in the Blue channel and the values of the glow\_intensity parameter of the object (see below).

Alpha Channel (cavity) – Ambient shading of low surface relief, which cannot be taken into account by AO techniques (SSAO, HBAO). This channel is almost always white, with some shading here and there.

The final result, taking into account all three textures (example taken from the Spaceship room, the texture of the box was changed for the sake of clarity):





PBR shading allows you to simulate about 90% of all materials, with the exception of materials with a high anisotropy and subsurface scattering.

### 1.6 What does h5m stand for?

h5m is a model file format that hook5 can load and draw in the scene. The model file in h5m format can be obtained by using the H5MConverter which converts the model from the well-known obj- to the h5m format. This format was made in order to increase the model's loading speed. The data in this file is saved and organized in such a way that it can be efficiently used by the Graphics API to render the scene without the need to resort the data after loading which happens to be the case with other formats. In addition to that, when exporting an obj file, the converter calculates the correct TBN basis for each vertex of the mesh, which is necessary for the correct normalmapping of hook5 objects, and saves it to h5m.

Your model should be optimized and triangulated before exporting (or during export) to obj file format and you must enable the generation of a material library (mtl file), normal and texture coordinates. The texture information from mtl will be saved in the h5m file. A diffuse texture is mandatory (the color of the material from the obj file is not taken into account).

You can also manually create a material library in which the following texture parameters should be present:

- map\_Kd any\_diffuse\_texture\_name.dds – diffuse texture definition
- map\_Ks any\_specular\_texture\_name.dds – specular texture definition
- map\_bump any\_normals\_texture\_name.dds – normal texture definition

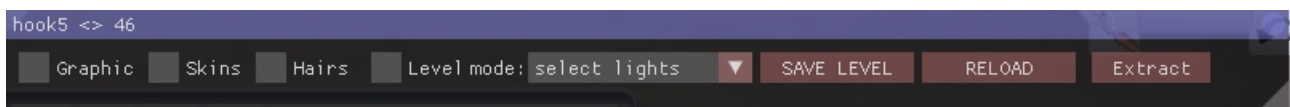
The normal texture and specular texture may not be specified in the material file, in this case, the information about them will be missing in the h5m file. If such a model is included in the level / room definition, hook5 will try to load files using the diffuse texture name with corresponding suffixes, for example: if the diffuse texture is "RedBrickWall.dds", hook5 will try to load the normal texture "RedBrickWall\_norm.dds" and the specular texture "RedBrickWall\_spec.dds".

## 1.7 How do you create an animation with textures using Hook5?

Hook5 allows you to load a set of textures instead of a single texture and use this set as frames for an animation that plays at a freely customizable speed. Hook5 will load the texture set defined under *diffuse\_override* in the text-file of the same name as the texture to be replaced. If the assigned texture contains the “\_num001” suffix, which means that this is the first frame in the frame set (textures). Hook5 will then successively load all textures with a suffix “\_num00x”, x denoting an increasing number in the suffix, within that folder until the last frame is rendered.

For example: *diffuse\_override* = TVScreen\_num001.dds: hook5 loads and repeats the texture sequence TVScreen\_num001.dds, TVScreen\_num002.dds, TVScreen\_num003.dds ... TVScreen\_num020.dds and so on. The speed of the animation for game objects is set by the parameters *diffuse\_swap\_speed*, *normal\_swap\_speed* and *specular\_swap\_speed* in frames per second. For hook5 objects, the parameters of the swapspeed parameter group override are used (see below for hook5 objects).

## 1.8 Hook5 GUI Ingame main menu.



**Graphic** – The graphics customization window.

**Skins** – CustomizableSkin parameters window (only available if there is a character with a designated CustomizableSkin in the scene).

**Hairs** – HairWorks model parameters window (only available if there is a character in the scene with a designated HairWorks hairstyle).

**Level** – Level parameters window (\_level\_definition.txt).

**Selection mode** – Object selection mode (static objects, lights, physical pools).

**SAVE LEVEL** – Saving changes made to \_level\_definition parameters.

**RELOAD** – Load the latest saved version of the current \_level\_definition.

**Extract** – Export the current scene to a file with the obj file format (the scene can then be loaded in a 3D editing software).

The windows and parameters will be discussed further below.

## 2. Basic graphics settings: main11.fx (Postprocessing Effects.)

The settings in the main11.fx file are organized into meaningful and convenient sections. Most of the parameters in the main11.fx file refer to postprocessing effects (effects which are applied to the whole image), with the exception of several global parameters and parameters for the eye shader. Changes made to the parameters of the [global\_constant] section requires the recompilation of shaders which can be done by pressing the F11 (default) key on your keyboard, changes to the remaining parameters will be applied automatically.

### [global\_constants]

- **iShadowMapSize** – Determines the size of the shadow maps. The size, i.e. the width and the height, should be to the power of two and the larger the size the higher the quality of shadow casting, while at the same time diminishing performance. The default value is 2048.
- **iLightPackMode** – Determines the mode how the lighting calculation results are packed. Currently only one mode is implemented, so the value 0 should be set.
- **lphysXSolverIterations** – The number of physics miscalculations. Higher values improve the quality of physics calculations, but will affect performance.
- **funcRecompile,**
- **funcBoundview,**
- **funcSeparation,**
- **funcCubemap,**
- **funcHookGui,**
- **funcScreeshot,**
- **funcHiResScreeshot,**

**funcXXX:** A key binding in the format: funcXXX = keycode, shift\_state, ctrl\_state. For example:  
funcHiResScreeshot = 44, 0, 1 (key: PrintScreen, without pressing shift, with pressing control).

### [global\_settings]

- **fFovMultiplier** – Changes the FOV (Field of View) ingame. FOV will be changed only for rendering, so some operations, such as selecting a transform gizmo or cutting off invisible objects, may not work correctly. It should be set to the default value when using PoseEdit. The default value is 1.
- **iProfileEnable** – Enable information about the render stages: 0 – disabled, 1 – not used anymore, 2 – GPU profile window.
- **ITextureTrackingEnable** – Enable / Disable texture tracking, possible values 1 / 0. All trace files, \_pass files, current \_level\_definition.txt and game textures located in ActiveMod are tracked. Textures that have been changed will be reloaded. If you are not creating textures for the game, disable texture tracking.
- **IEnableBackgroundRendering** – Turn on / off rendering while the game window is inactive / minimized / in the background, possible values 1 / 0. Useful for those who work simultaneously with other applications – the inactive window of the game does not render frames and does not consume computer performance. The default value is 0.
- **iUseCubemapPrepass** – Turning on / off an additional pass for rendering of the environment texture, possible values 1 / 0. A double pass rendering results in a better illumination of the scene's final render. The default value is 1.

- **iMultithreadRenderEnable, iMaxThreads** – Turning on multithreaded rendering and setting the number of used threads. At first this feature was implemented, but in practice it didn't give any performance boost. Due to an incompatibility with HairWorks is bound to be disabled or will be disabled in the future.
- **IDoVSyncInWindowed** – Turns vertical synchronization On / Off in windowed mode, the default value is 0.
- **cScreenshotFolder** – The path to where Hook5 saves the screenshot when pressing either screenshot key set under [\[global\\_constants\]](#), if no path is specified, the screenshots will be saved in the binaries folder.
- **IScreenshotFormat** – Sets the format in which the screenshots are saved: 0 – bmp, 1 – jpg, 2 – png.

#### [\[anaglyph\]](#) – Stereo and VR settings:

- **iSeparationMode** – Separation / separation mode / BP. Activating either of the following modes in the main.fx file does not automatically display the picture on screen. In order to display the desired image you must press the specified key under [\[global\\_constants\]](#). (F3 by default):  
 0 / Anaglyph Mode: Color separation (blue and red), requires anaglyph stereo glasses.  
 1 / SideBySide1 Mode: Split Screen – Both left and right frames on the screen, camera position is the same as in anaglyph mode.  
 2 / Oculus VR Mode: OculusHome support, requires restarting the game.  
 3 / SideBySide2 Mode: Split Screen – Both left and right frames are displayed on screen, camera position is the same as in Oculus VR.  
 4 / OpenVR Mode: OpenVR support, requires restarting the game.
- **FAnaglyphSeparation** – Force a specific separation value for the left and right images in anaglyph and side by side modes. Higher values translate into higher distances.
- **FAnaglyphConvergent** – Set a specific convergence for the left and right images in anaglyph and side by side modes. Changing this value translates into a lower or higher Pop-out effect.
- **IOVREnableReflection** – Enables / Disables the display of reflections in OpenVR Mode.
- **fUIScaleH, fUIScaleV** – Scale of the interface in game.
- **FSBSBarrelStrength** – Sets the strength for the Barrel Distortion effect. This is necessary to reduce the distortion of VR device lenses.
- **FSBSFovScaleH** – Horizontal FOV (Field of View) scaling for SBS modes.
- **FSBSFovScaleV** – Vertical FOV (Field of View) scaling for SBS modes.
- **FSBSScale** – Sets the aspect ratio scale of the picture for SBS modes.
- **FSBSOffsetX** – Horizontal frame offset from the center of the image for SBS modes.

#### [\[render\\_settings\]](#)

- **iUseDebug** – Enable rendering of intermediate results of the render, 0... 14.
- **iUseDiffuseLUT** – Enable / Disable the use of LUT textures to modify diffuse lighting. A DiffuseLUT.bmp texture is required.
- **iUseSpecularLUT** – Enable / Disable diffuse lighting for specularity. A DiffuseLUT.bmp texture is required.
- **iUseSkinWetness** – If set to 0, the sweat intensity value of the game will be used, if set to 1, the global parameter fSkinWetness will be used. The default value is 1.

- **fSkinWetness** – A parameter which determines the glossiness of the skin from completely dry to wet. The green channel of the specular texture is used to simulate the wetness effect, which can range from completely dry (0) to an oil like appearance (1). The default value is 0.
  - **iSubsurfaceEnable** – Enables / Disables the Screen Space Subsurface Scattering effect. Subsurface Scattering handles light that penetrates and moves within the area under a surface. The default value is 1.
  - **fSubsurfaceTakeColor** – Determines the amount of color of the diffuse texture which will be used to determine the light passing through the skin, 0... 1
  - **iSubsurfaceExcludeIBL** – Switches the inclusion of IBL (Image Based Lighting) in the skin's subsurface scattering rendering pass on / off. Possible values 1 / 0.
  - **fSubsurfaceSpecularIBL** – The power of specular reflection from ambient lighting. The default value is 1.
- The last two parameters should not exist, since they contradict the physics of how light interacts with skin. The first was made purely for experimental purposes – it uses ambient lighting after rendering the subsurface scattering effect. The second was introduced due to the fact that, unlike raytracing, we cannot calculate the overlap of ambient lighting and specular flare often occurs where it should not be.
- **iEpidermalSteps, fEpidermalRadius, iSubdermalSteps, iSubdermalRadius** – The number of steps and the blur radius of the subsurface scattering illumination passes.

Illumination is divided into two passes. The Epidermal pass renders the light that returns after a shallow penetration of the skin. The Subdermal pass renders the light that interacts with tissues under the skin like muscles, vessels, etc. and, after multiple reflections, leaves the surface of the skin. It is assumed that the second component has a more washed out appearance and is very saturated with red. The implementation of the SSSS effect in Hook5 is such that the Subdermal component will be combined with the Subsurface skin texture (stage4) in Hook5.

- **fDiffuseAmount** – The amount of the diffuse component in the final render.
- **fEpidermalAmount** – The amount of the epidermal component in the final result.
- **fSubdermalAmount** – The amount of the subdermal component in the final result.
- **iUseEyeShadowing** – Turn on / off the effect that eyelids will cast shadows on the eye.
- **fEyeIrisScale** – Scales the iris of the eye.
- **fEyeShiftUp** – Changes the position of the eyes vertically, without affecting the mesh of the head.
- **fEyeSpecular** – Changes the specularity of the eyes.
- **fEyeReflectionAmount** – Changes the amount of the eyes' reflection.
- **ISSAOEnable** – Ambient occlusion shading mode:
  - 0 – Disabled
  - 1 – SSAO (Screen Space Ambient Occlusion).
  - 2 – MXAO (Marty McFly's Ambient Obscurance).
  - 3 – Nvidia HBAO+ (only available in the extended version of the hook5)

The technique of ambient occlusion shading allows to simulate the effect that overlapping nearby objects have a shading effect on each other while light passes the scene. The technique uses the depth buffer and a rendering technique that resembles a kind of ray tracing. For a pixel, a random scatter calculation of rays is performed taking into account the surface normal. Each ray is then checked for an intersection with the depth buffer (flat volume representation in camera space) and if an intersection is

present, the degree of shading from this point is determined. Furthermore, the average value of all the rays is calculated, which results in a kind of ambient shading.

- **FSSAORadius** – Defines the radius of the space in which nearby objects overlap.
- **FSSAOSTrength** – Adjusts the strength / contrast of the AO effect.
- **FSSAOBias** – Adjusts the brightness of the effect.
- **iSSAOBlurEnable** – Turns on / off the blur pass for AO effect.
- **iSSAONormalsEnable** – Switches the use of AO's surface normals on / off.
- **fSSAONormalsFade** – Straightens the surface normals (Changes the expressiveness / influence of the AO surface normals.)
- **iMXAOSTeps** – The number of miscalculation steps (MXAO only).
- **iHBAODetailEnable** – Switches an additional pass in the HBAO technique (uses a smaller radius) on / off.
- **ISSLREnable** – Turns the ScreenSpaceLocalReflections on / off. The SSLR effect (reflections in screen space) is a method that give the observer the impression that real time reflections of light are calculated. The basic method is to calculate the intersection of light beams with the help of the depth buffer (a flat representation of the volume in the camera space), to calculate the reflected vector for the pixel's projective space and to carry out the passage along the ray with a given step. Each step the depth buffer is checked for intersection with the ray and if an intersection is detected, the intersection point is refined. The result is that dynamic reflections can be obtained without a dependency on the scene's complexity with the drawback of having a bunch of artifacts and erroneous reflections.
- **iSSLRStepSize** – The rendering step size in pixels.
- **iSSLRStepsCount** – The number of rendering steps for one pixel of the image.
- **fSSLRSkipThreshold** – The size / distance at which the intersection with objects will be ignored. Helps to reduce the number of incorrect reflections.
- **iOutlineEnable** – Turning on / off the outline effect (a cartoon-like line around objects).
- **fOutlineAmount** – The strength of the outline effect.
- **fOutlineThreshold** – The maximum depth difference of overlapping objects that will be ignored by the effect.
- **iDOFEnable** – Turns the Depth of Field effect on / off (DepthOfField, Depth of Field).

The effect of depth of field blurs objects that are out of focus. Often used in photography and constantly used in movies this effect was actually implemented in computer graphics to obtain a cinematic effect. A similar effect can also be found in human vision. This effect is dependent on the depth buffer to find the object's edges and then determines the degree of pixel blurring. Translucent objects that disable writing to the depth buffer won't be taken into account.

- **iDOFType** – DOF mode:
  - 0 – The focus is placed on an object located in the center of the screen.
  - 1 – The focus is set to the near distance, distant objects are always blurred.
- **iDOFSteps** – The number of blur steps for objects out of focus.
- **fDOFBokehBias** – Not used yet.
- **fDOFBokehBiasCurve** – Not used yet
- **fDOFBokehBrightnessMultiplier** – Increases the brightness of objects which are out of focus.

- **fDOFBokehBrightnessThreshold** – The maximum level of brightness from which there will be an increase in the brightness of the object that is out of focus.
- **fDOFRadiusScaleMultiplier** – The DOF blur radius in pixels.
- **iSMAAEnable** – Enable / Disable the SMAA anti-aliasing.
- **iSMAAQuality** – Not used, the default settings are for the effect.
- **iGlowEnable** – Turn on / off the object's glow effect. The glow effect does not affect the lighting and works like a Photoshop filter. It creates a copy of the image from which all pixels are removed that do not produce glow. This new image is blurred and again combined with the original image, creating a glow effect.
- **fGlowAmount** – The strength of the glow effect.
- **iHDRBloomEnable** – Enables a glow effect (an imaging artifact similar to the effect of a real world camera) for bright objects in the scene:
  - 0 – Off,
  - 1 – Switches to the default method,
  - 2 – Switches to an advanced method using Compute Shaders.

The algorithm is the same for both methods, however, the method using CS works faster whereas the default method is to be used if the advanced method isn't working correctly.

- **iHDRFuncType**
  - 0 – The old rendering method of HDR images without gamma texture correction,
  - 1 – A new version with gamma correction.

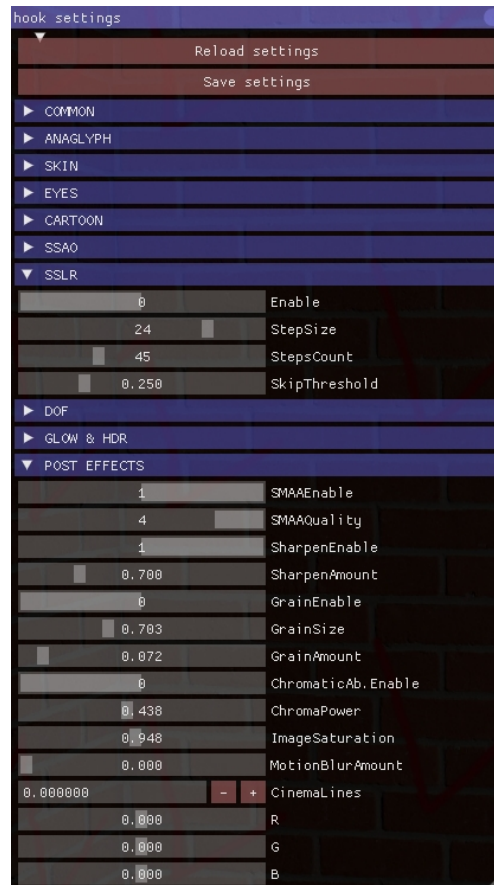
The default value is 1. Old games such as 3D Sex Villa did not produce any images within the HDR (HighDynamicRange) space. The images displayed on the monitor had to be in the brightness range between the values zero and one. Everything above the value one would have been discarded. The developers had therefore been tasked to keep the lighting of the scene inbetween those limits when developing the levels. If several light sources had been present in the scene the illumination was summed up and confined to the limits of the screen. In reality, brightness levels go far beyond monitor display capabilities. The evolution of computer graphics made it possible to perform lighting calculations in HDR, using real brightness values. However, the final result must still conform to the relative brightness range of the display between zero and one. This is what the HDRFunc function does – it calculates the optimal range of brightness of an image located in HDR space, and compresses this range to the required one. The result is an image with a high contrast that can be displayed on-screen.

- **IsSharpenEnable** – Turns sharpening of the image on / off.
- **fSharpenAmount** – Changes the strength of the sharpening effect.
- **iGrainEnable** – Turns the "Film Grain" - effect on / off.
- **fGrainSize** – Changes the size of the grain.
- **fGrainAmount** – Changes the strength of the "Film Grain" - effect.
- **iChromaticAberrationEnable** – Enable / Disable chromatic aberration.
- **fChromaPower** – The strength of the chromatic aberration effect.
- **fMotionBlurAmount** – Strength of the Motion Blur effect.
- **fCinemaLines** – The size of the black bars above and below the image.



- **RGBOffset** – The offset of each R, G, B color component of the final image, three values, one for each channel and separated by commas.
- **iShadowEnable** – Enable (1) / Disable (0) dynamic shadows.
- **iShadowBlurStep** – Defines the shadow edges' blur steps.
- **iShadowBlurStepAlpha** – The same as above but only for alpha-blended objects.
- **fImageSaturation** – Adjustment for the image's saturation.
- **iHiResScreenScale** – Screenshot size multiplier.

All dynamic parameters can be changed using hook5's graphical interface ingame:



In addition to the graphics settings the main11.fx file contains the basic level settings for the default game rooms (which are not in ActiveMod and cannot contain their own \_level\_definition.txt) and for the customizer: customizer\_environment, customizer\_sunlight, default\_room\_environment, default\_room\_sunlight.

The description of these sections' parameters can be found further below with the parameters' description of the \_level\_definition.txt.

### 3 Advanced ingame object rendering.

#### 3.1 Pass files and parameters.

The name of the pass file consist of the texture's name to which we want to bind the data for the extended rendering and an added suffix named "\_pass". The passfile must be saved in txt format. For example:

- Texture's name: RedBrickWall.png
- Passfile's name: RedBrickWall\_pass.txt

The pass file should start with a section called [pass] followed by the required pass file parameters. The defining parameter is the name of the shader group, that can be either specified or not. If not specified simple shaders will be used for this group of objects:

Name	The name of the shader to be used for shading the object. Can also be omitted.
Discard	Excludes the object for rendering.
CustomModel	Shader for dynamic game objects (clothes, hair, etc.).
CustomStatic	Shader for static game objects (room objects,landscape objects etc.)
CustomSkin	Shader only used for the body.
CustomWater	Shader used for water rendering.
CustomCum	Shader for bodily fluids.
BackgroundPass	A masterpiece for drawing SkyBox objects (backs, backgrounds)
EyesPass	Shader for the eyes.
PhysXModel	Shader only used for skinned PhysX objects.
NvHair	Shader for Nvidia HairWorks hairs.

#### Parameters that can be used by CustomModel, CustomStatic, CustomCum shaders and EyesPass:

- **stage2** – The path to the object's normals texture relative to the ActiveMod folder's path.
- **stage3** – The path to the object's PBR texture relative to the ActiveMod folder's path.
- **diffuse\_override** – The path to the object's diffuse texture relative to the ActiveMod folder's path.  
Allows you to override the object's main diffuse texture in game.
- **relief\_mode**:  
0 – Bumpmapping,  
1 – Normalmapping,  
The default value is 1. Despite the fact that fake TBN is used normal mapping gives better results.
- **cull\_mode (default 1)**:  
0 – Draw all polygons.  
1 – Draw only polygons directed to the camera.  
2 – Draw only polygons turned away from the camera.  
3 – Draw polygons first directed to the camera, then turned away from the camera.  
4 – Draw polygons first turned away from the camera, then turned to the camera.  
Take note: Mode 3 is better to be used for two-sided turns without alphablending. Mode 4 should be used for two-sided objects with alphablending.

- **backface\_mult** – Color multiplier for backfacing polygons, x\*
- **backface\_curve** – Color modifier for backfacing polygons, x\*
- **bumping\_scale** – Allows you to set up the relief intensity for Bumpmapping, x\*
- **height\_offset\_scale** – Enhances the effect of Normalmapping by shifting the texels depending on the height map (Alpha channel of the normal texture), x\*.
- **height\_offset\_steps** – A simplified BumpOffset method is used, when set to the default value of 0. Parallax Occlusion Mapping (POM) is activated whenever values other than 0 are used. There is no point in enabling POM when the elevation is less than 1-1.5 cm. Be aware that POM can reduce system performance.
- **alpha\_threshold** – This value defines the threshold at which the pixel will not be drawn on screen i.e. they will be rendered transparent, the threshold can range from 0 to 255 (default 0).
- **force\_as\_static** – Forces the conversion of an object to a group of static game objects, possible values: true / false.
- **cast\_shadow** – Enables / Disables the object's shadow casting. Possible values: true / false. Disabling shadow casting for an object (e.g. floors) allows you to save some performance, especially when there are large numbers of light sources that cast shadows.
- **glow\_intensity** – The power of the object's glow-effect, x\*. Influences the final intensity of glow:  $\text{Glow} = (\text{diffuse\_map.color}) \cdot (\text{spec\_map.blue\_channel}) \cdot (\text{glow\_intensity})$ .
- **mirror\_source, mirror\_visibility** – Allows you to bind a hook5 mirror (see below) to the object and adjust the visibility.
- **diffuse\_swapspeed, normal\_swapspeed, specular\_swapspeed** – Sets the speed at which the textures change if a set of textures is used.

#### CustomSkin parameters:

(Only used for body parts)

- **stage4** – Path to the subsurface skin texture relative to the ActiveMod folder. Subsurface textures channel usage:  
RGB channel – Subsurface Color,  
Alpha – Strength of subsurface effect.
- **tessellation\_level** – Enables tessellation that increases the number of polygons and smooths the mesh's surface. Possible values 0 / 1.
- **displacement\_scale** – Strength of the surface's displacement. Uses the Alpha channel of skin's normal texture.

#### CustomWater parameters:

(The water shader doesn't use the diffuse texture)

- **stage2** – The path to the first wave's normal texture relative to the ActiveMod folder.
- **stage3** – The path to the second wave's normal texture relative to the ActiveMod folder.
- **deep\_color** – The water's color at greater depths, format: RGB \*.
- **shallow\_color** – The water's color at smaller depths, format: RGB \*.
- **w1\_scale\_uv** – Scaling of the texture coordinates for the first wave texture, xy \*.
- **w1\_scale\_normal** – Scaling of the first wave's normal texture, x \*.
- **w1\_speed** – Rate at which the first wave texture moves, xy \*.
- **w2\_scale\_uv** – Scaling of the texture coordinates for the second wave texture, xy \*.
- **w2\_scale\_normal** – Scaling of the second wave's normal texture, x \*.
- **w2\_speed** – Rate at which the second wave texture moves, xy \*.
- **fog\_distance** – The distance at which water completely loses its transparency, x \*.
- **reflection\_amount** – The reflection power of the water surface, x \*.

- **reflection\_colorize** – The water’s reflection color, x \*
- **refraction\_amount** – The water’s refraction strength, x \*.

#### PhysXModel parameters:

- **physx\_model** – Path to the hook5’s PhysX model relative to the ActiveMod folder.
- **physx\_hair** – Path to the hook5’s PhysX hair model relative to the ActiveMod folder.
- **physx\_rig** – Path to the hook5’s PhysX ragdoll data relative to the ActiveMod folder, repX.
- **hair\_normal\_scale** – Scale of the hair normals, x\*.
- **hair\_normal\_shift** – Shift of the hair normals, x\*.

#### NvHair Parameters:

##### Position and Scale

- **scale** – The scale of the hair model, values for: X, Y, Z.
- **offset** – The offset of the hair model relative to the head, values for: X, Y, Z.
- **hair\_rotation** – The rotation in degrees of the hair model relative to the head, values for: X, Y, Z.

#### Physical

##### General:

- **nvho\_mass\_scale** – Determines the weight of the hair by increasing or decreasing the mass of the hair.
- **nvho\_damping** – The strength of the movement’s influence by e.g. air drag and water, that slows down the fur strands.
- **nvho\_inertia\_scale** – Increases the hair’s inertia (how fast the hair reacts to movement).
- **nvho\_inertia\_limit** – Defines a velocity threshold at witch the inertia scale becomes active (“Nvidia! Oh, my!”).

##### Collision

- **nvho\_friction** – Friction strength with collision shapes, like the body.
- **nvho\_backstop\_radius** – Approximates the surface of the growth mesh to use as collision. A value of 0 does not use the solution while a value of 1.0 uses the average edge length of the entire growth mesh to determine an approximation. A side effect of this is that it can artificially fluff up the root of the hairs. This can be desired or undesired.

#### Style

##### Volume:

- **nvho\_density** – The amount of hair strands produced by interpolation between the guide curves. A value of 0 produces no hair. Currently, a density value of 1.0 produces 64 hairs per each triangle of growth mesh. A value above 1.0 will scale the number of hairs correspondingly (a value of 2 would yield 128 hairs or a value of 10 would have 640). If a texture control is used, then this control becomes a multiplier for the texture. A texture control by default is sampled per vertex in UV space.
- **nvho\_length\_scale** – Grows the hair length down from 100% (1) to 0% (0).
- **nvho\_length\_noise** – The strength of random value noise that is applied to the length of the interpolated hairs resulting in non uniformly interpolated hairs.

##### Strand Width:

- **nvho\_width** – Determines the base width of each hair. All other parameters in this group as well as their textures act as multipliers to this value. This value is always set in millimeters, regardless of what the scene scale is set to.
- **nvho\_width\_noise** – The strength of random value noise that is applied to the overall width of individual hairs.  $x *$
- **nvho\_width\_root\_scale** – Multiplier to the width scale that applies toward the root of each individual hair. If a texture map is used, then the sampled texture values per each hair will be multiplied to this constant. The combined value then acts as multiplier to the base scale value.  $x *$
- **nvho\_width\_tip\_scale** – Multiplier to the width scale that applies toward the tip of each individual hair. If a texture map is used, then the sampled texture values per each hair will be multiplied to this constant. The combined value then acts as multiplier to the base scale value.

### Strand Stiffness:

(Controls how closely the hair strand stays to its initial groomed position.)

- **nvho\_stiffness** – The limits on how close each individual hair stays to the skinned position (Resting Stiffness). If a texture map is used, then the texture will influence the stiffness calculation for each simulated (guide) hair.
- **nvho\_stiffness\_curve** –
- **nvho\_stiffness\_strength** – The strength of the spring (bounciness) of the hair to return from a simulated position to its resting stiffness. Full springiness is 0 and 1 is no spring.
- **nvho\_stiffness\_strength\_curve** –
- **nvho\_stiffness\_damping** – The speed of the spring (bounciness) of the hair to return from a simulated position to its resting stiffness. 1: full damping (slowest / no motion), 0: no damping (fastest).
- **nvho\_stiffness\_damping\_curve** –
- **nvho\_root\_stiffness** – The strength of stiffness that weakens toward the tip of the hair. If a texture map is used, then the texture will influence the stiffness calculation for each simulated (guide) hair.
- **nvho\_tip\_stiffness** – The strength of stiffness that weakens toward the root of the hair. If a texture map is used, then the texture will influence the stiffness calculation for each simulated (guide) hair.
- **nvho\_bend\_stiffness** – The strength of how hair tries to maintain initial curvy hair shapes.
- **nvho\_bend\_stiffness\_curve** –
- This information is directly copied from NVIDIA's HairWorks Viewer [Reference](#). According to my observations, I can say that stiffness determines the preservation of the hair's shape at rest, while stiffness\_strength is responsible for how fast the spring returns from a simulated position to its stiffness position (rest position). Concerning the relationship between the parameter and the parameter curve, I have no idea, since neither the formulas nor the explanations make any sense to me and nothing can be found in the documentation. At first I thought that the parameter is determined by a curve (four points along the length of the hair + Bézier smoothing) and the parameter is a common factor (scaling the amplitude of the curve), but I cannot say for sure, since the hair's response to changes IN GENERAL is not obvious. You can understand by the names what the other stiffness parameters are for but keep in mind that not all changes are clearly visible when looking at the hairstyle in game. In general, experiment, yes experiment.

### Waviness:

(Determines the number of sine waves applied to each individual strand, creating the look of waves or curls.)

- **nvho\_wave\_freq** – Determines the number of waves generated down the length of each individual strand.

- **nvho\_wave\_freq\_noise** – The strength of noise that is applied to the frequency of each individual hair.
- **nvho\_wave\_root\_streighten** – Determines how much of the root of each individual hair should remain without waves. A value of 0 has no effect. A value of 1.0 makes waves linearly scale up toward the tip, where the tip would get the full wave scale, while the root will have no waves.
- **nvho\_wave\_scale** – Determines the amplitude of the waves. This value is always set in centimeter unit, regardless of what the scene scale is set to.
- **nvho\_wave\_scale\_noise** – The strength of noise that is applied to the scale of each individual hair.
- **nvho\_wave\_scale\_clump** – The strength of the waviness of clumped strands. This value is multiplied against the waviness scale.
- **nvho\_wave\_scale\_strand** – The strength of the waviness of individual strands. This value is multiplied against the waviness scale (**wave\_scale**).

### Clumping:

Determines how much each hair follows a clumped shape. Each clump is generated for each growth mesh triangle, and this value controls how much the hair shape is bent toward the center hair. Clumps are defined per vertex.

- **nvho\_clump\_scale** – The strength of clump that is applied to each individual hair. If a texture map is used, then the sampled texture value per hair will act as a multiplier to this constant value.
- **nvho\_clump\_noise** – The strength of the percentage of hairs that are included in each clump.
- **nvho\_clump\_roundness** – Determines the roundness of each clump. A value of 0 is a concave clump, a value of 2.0 is a convex clump, and the default value of 1.0 is a neutral clump.

## Graphics

### Color:

- **nvho\_use\_tip\_color** – Enables / Disables the use of the tip color shader.
- **nvho\_tip\_color** – The color that is used for the tips of strands. This is determined by loading a texture input and is sampled per hair. (R, G, B)
- **nvho\_root\_tip\_falloff** – The blend between the root color and tip color. The Root / Tip Color Weight position is the center of the falloff.
- **nvho\_root\_tip\_weight** – Balances how much of the Root and Tip color is used along the length of the strands.

### Shadow:

- **nvho\_shadow\_density** – For shadow rendering, we can use less, but thicker hairs for performance. This value is used as multiplier on top of density value. Lower values make hair shadow rougher, but make rendering faster. A value of 0.5 is typically a reasonable choice.
- **nvho\_shadow\_sigma** – The distance at which the hair is considered to receive shading.

### Diffuse:

- **nvho\_diffuse\_blend** – The diffuse blend attempts to match the rendered shading from the hair with the mesh surface's shading. By setting this value to 1.0, users can ignore hair diffuse term and match mesh lighting more closely. A value of 0.0 will force hair lighting only.
- **nvho\_normal\_weight** – The strength of the normal derived from the hair normal bone to calculate shading weighted against the diffuse blend. Typically used for long hair.

### Specular:

Applies a shiny effect to the hair.

- **nvho\_spec\_power** – Adjusts the specular's intensity.
- **nvho\_spec\_scale** – The strength at which the primary specular is applied.

#### **Glint:**

Applies a sparkly effect to the hair.

- **nvho\_glint\_scale** – Determines how much noise is present in the glint.

In the extended version of hook5 hair parameters can be dynamically changed using the GUI's "hairs" window. The settings will be available only if the character using the HairWorks hair model is present in the scene:



#### **Possible Values:**

- **RGB** – Three integer color values separated by commas, in the range of 0...255.
- **x** – A single value (mostly fractional, in some cases integer).
- **xy** – Two values (mostly fractional, in some cases integer) separated by a comma.
- **xyz** – Three values (mostly fractional, in some cases integer) separated by a comma.



## 4 Hoo5 Object Rendering.

### 4.1. Level Definition

The `_level_definition.txt` defines the room's parameters. This file can be added to any room in the ActiveMod folder. With the help of this file, light sources, static hook objects, mirrors, water, fire, fog can be added to the room and global lighting parameters for this room can be defined. Global lighting parameters and parameters for inserted objects are each represented in their respective sections.

#### [environment]

- **ambient\_color** – Ambient room lighting color (RGB).
- **ambient\_intensity** – Ambient light intensity. At the moment ambient lighting is used when rendering the environment cubemap's illumination. When rendering a scene, ambient light data is used from the environment cube itself for the scene's illumination, while ambient illumination is not involved in rendering the scene.
- **lightmaps\_enable** – Turns lightmaps on / off (if available) for a room.
- **no\_lightmaps\_in\_cubemaps** – Disables the use of lightmaps when rendering cubemaps.
- **no\_level\_shadows** – Disables shadows for the room.
- **hdr\_exposure** – Exposure parameter for the room (functions like a gamma setting for room lighting).
- **hdr\_bloom\_clamp** – The minimum level of brightness below which a pixel will not be taken into account in the HDRBloom effect.
- **hdr\_bloom\_amount** – The total value of the HDRBloom effect for this room
- **glow\_scale** – Changes the scale of the glow created by the Glow effect.
- **glow\_cubemap\_scale** – Changes the scale of the glow created by the Glow effect when rendering cubemaps.
- **glow\_softness** – The softness of the glow's halo.
- **saturation\_scale** – Changes the saturation of the image for this room
- **cubemap** – Denotes the path relative to ActiveMod to the cubemap environment file. If it is specified, the given Cubemap texture will be used to calculate the illumination of the environment, if it is not specified, the environment texture will be generated from the current scene.
- **cubemap\_source\_scale** – Cubemap texture color scaling.
- **cubemap\_source\_curve** – Sets the gamma value of the cubemap texture.
- **cubemap\_source\_saturation** – Sets the saturation of the Cubemap texture.
- **cubemap\_source\_angle** – The angle of rotation of the cubemap texture relative to the vertical axis, 0...360 degrees.
- **cubemap\_center** – x, y, z position of the EnvironmentBound, used in conjunction with ParallaxCorrection.
- **cubemap\_size** – (size\_x, size\_y, size\_z) The size of the EnvironmentBound, used in conjunction with ParallaxCorrection.
- **cubemap\_color** – EnvironmentBound color (R,G,B,A).
- **parallax\_correction** – Enable / Disable ParallaxCorrection.
- **skybox\_enable** – Enables / Disables the use of the SkyBox.

- **skybox\_cubemap** – The path relative to ActiveMod to the SkyBox's cubemap texture.
- **sky\_intensity** – Sets the intensity of the SkyBox cubemap.
- **skybox\_curve** – Sets the gamma of the SkyBox cubemap.
- **skybox\_saturation** – Sets the saturation of SkyBox cubemap.
- **skybox\_angle** – The rotation of the SkyBox cubemap relative to the vertical axis, 0...360 degrees.
- **fog\_base\_level** – The height level from which the fog is generated. 0 by default, the floor level might be different in some rooms, so that it is necessary to take the difference to the actual floor level of the room into account.
- **fog\_density** – The relative fog density in a range from 0 to 1. At 0 fog rendering is disabled.
- **fog\_height\_density\_falloff** – The decrease in density with the distance in height in relation to the fog\_base\_level, possible range: 0...1.
- **fog\_sun\_scattering\_amount** – The amount the fog scatters the sunlight, possible range: 0...1.
- **fog\_sun\_halo\_amount** – The power of the sunlight's halo, possible range 0...1.
- **fog\_sun\_halo\_spread** – The size of the halo from sunlight, possible range 0...1.
- **fog\_affect\_sun\_strength** – The decrease in brightness of sunlight that passed through the fog, possible range 0...1.
- **fog\_color** – The fog's color, RGB components (0...255).

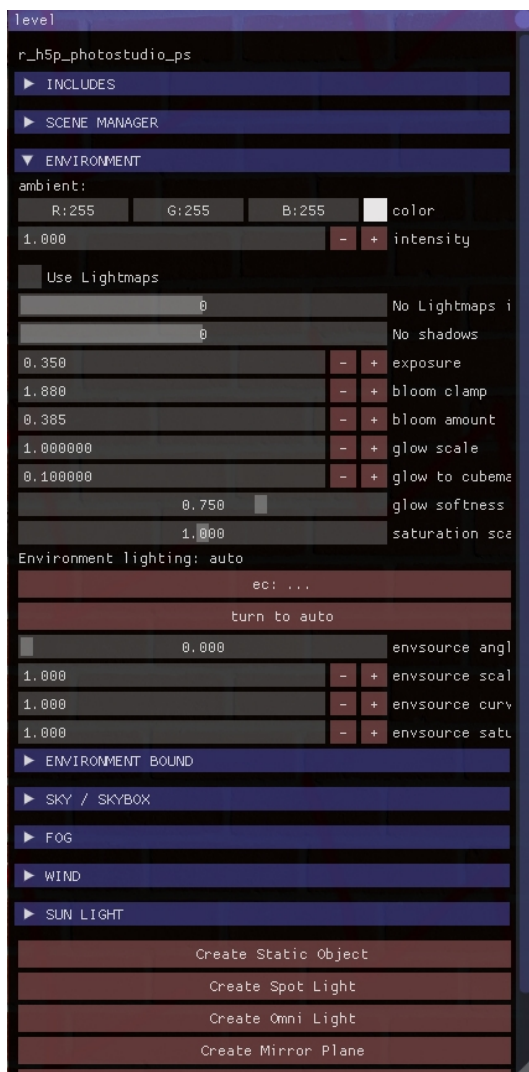
#### [sun\_light] – Parameters for the sunlight.

- **type** – Defines the hook5 object type, the value of sunlight.
- **enable** – Enables / Disables the sunlight for the level (room).
- **rotation** – Changes the direction of sunlight in degrees along the three global axes X, Y, Z.
- **color** – Changes the color of the sunlight in RGB values (0-255).
- **intensity** – Changes the intensity of the sunlight.
- **shadow\_param** – parameters for building shadows, xy. Where x is the distance from the camera that will be covered by the first shadow mask, y is the maximum distance from the camera. Shadows from sunlight are created using the CascadedShadowMaps method by using 4 shadow masks. The second and third shadow masks will cover a distance 2 and 4 times greater than the first one. The last shadow mask covers the distance from the end of the third to the maximum shadow distance. The maximum distance should be set to the minimum visible area, as this parameter significantly affects the performance and quality of shadows.
- **game\_control** – True / false, if set to true then the direction of sunlight will be determined by the game and can be changed with the keys L / Ctrl+L. If set to false then the rotation parameter of this section will be used to determine the direction of sunlight.
- **sunrays\_enable** – Switches the sunrays effect on / off.
- **sunrays\_check\_distance** – The range of the sunray effect, in meters.
- **sunrays\_step\_size** – The length of the miscalculation step. A small step will lead to an increase in the number of calculation steps and a decrease in performance, a large step will give smaller detail to the effect while increasing performance.
- **sunrays\_density** – The maximum density of ray visualization at the specified distance.

**[includes]** – Additional level definition files can be included here.

The parameters of this section are the file name of the definition file and a Boolean parameter, for example: `_additional_objects.txt = true`, where true means that definitions from this file must be applied to the given level / room.

In the extended version of hook5, the level parameters are represented by the "level" GUI window:



The SCENE MANAGER grid gives access to the list of all additional objects used in a given level / room, which allows you to select objects by simply selecting the item in the list. A window with parameters corresponding to this object's type will be displayed for the selected object. Below the SUN LIGHT menu entry there are buttons for the creation of all available hook5 objects.

Sections of hook5 objects.

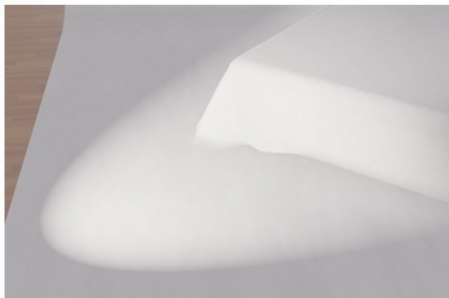
The name of the section is also the name of the object. The object's class / type is defined by the field type in the section parameters. Make sure that the section names do not match, otherwise the properties of the objects may be loaded incorrectly, which may cause glitches in the game.

### [some\_spot\_light] – Parameters for the Spotlight

- **type** – Type of the hook5 object, possible value: spotlight.
- **enable** – Enables / Disables the object.
- **position** – Position of the spotlight in world coordinates, xyz.
- **rotation** – Rotation of the light source relative to the global axes, xyz.
- **color** – The color of the spotlight, values in RGB.
- **intensity** – The intensity of the light source.
- **bound\_color** – The color for the limiting light source (debug rendering), RGBA.
- **fov** – The angle of the light cone, in radians.
- **near** – The minimum lighting start distance. It should not be zero and is connected with creation of a shadow map for a light source. The maximum distance is determined by the inverse square law or by the clamp\_range parameter.
- **source\_radius** – The parameter defines the size of the light source and is used to modify the function of diffuse and specular lighting.
- **source\_curve** – The degree / strength of the intensity drop, the parameter slightly contradicts the inverse square law, but was introduced to be able to adjust the intensity distribution as desired. A value of 2 corresponds to the inverse square law and smaller values give a more uniform distribution.
- **fov\_fade** – Switches the decrease in intensity at the light's cone on / off.
- **range\_fade** – Switches the decrease in intensity at the distance on / off.
- **cast\_shadow** – Turns the light's shadow casting on / off.
- **clamp\_range** – The distance at which lighting and shadow calculation is turned off to save performance. It is advised to be used when the source has a high intensity and the distance at which the light is still visible exceeds the visible space of the scene / room.
- **shadow\_bias** – The displacement of the depth / distance of the object's surface during shadow calculation, it is necessary to eliminate artifacts of shadow generation known as self\_shadowing errors. The default value is 0.002.

#### Spotlight example

```
[spot_front]
type          = spotlight
enable        = true
position      = 0.441943, 2.722074, 1.625985
rotation      = 0, 0, 0
color         = 255, 255, 255
intensity     = 6.000000
bound_color   = 255, 255, 39, 116
fov           = 1.5
near          = 0.010000
range         = 2.5
source_radius = 0.000000
source_curve  = 2.000000
fov_fade      = 1
range_fade    = 1
cast_shadow   = true
shadow_bias   = 0.000180
clamp_range   = 6.000000
```



#### Spotlight properties window

selected object properties

object:  
spot\_3

<input checked="" type="checkbox"/>	enable
-2.498557	pos:X
2.250169	pos:Y
2.671298	pos:Z
-30.525223	rot:X
-121.669510	rot:Y
24.274311	rot:Z
R:230 G:225 B:210	color
20.000000	intensity
0.600000	fov
0.500000	near
0.020000	source radius
<input checked="" type="checkbox"/> 1.000	source curve
<input checked="" type="checkbox"/> 1	range fade
<input checked="" type="checkbox"/> 1	fov fade
<input checked="" type="checkbox"/> 1	cast shadow
0.002000	shadow_bias
8.000000	clamp_range
R:155 G:110 B: 0 A: 69	bound color

### [some\_omni\_light] – Parameters for the OmniLight.


- **type** – Type of the hook5 object, possible value: omnilight.
- **enable** – Enables / Disables the object, possible values: true / false.
- **position** – Position of the omnilight in world coordinates, xyz.
- **color** – The color of the omnilight, values in RGB.
- **intensity** – The intensity of the light source.
- **bound\_color** – The color for the limiting light source (debug rendering), RGBA.
- **near** – The minimum lighting start distance. It should not be zero and is connected with creation of a shadow map for a light source. The maximum distance is determined by the inverse square law or by the clamp\_range parameter.
- **source\_radius** – The parameter defines the size of the light source and is used to modify the function of diffuse and specular lighting.
- **source\_curve** – The degree / strength of the intensity drop, the parameter slightly contradicts the inverse square law, but was introduced to be able to adjust the intensity distribution as desired. A value of 2 corresponds to the inverse square law and smaller values give a more uniform distribution.
- **range\_fade** – Switches the decrease in intensity at the distance on / off.
- **cast\_shadow** – Turns the light's shadow casting on / off.
- **clamp\_range** – The distance at which lighting and shadow calculation is turned off to save performance. It is advised to be used when the source has a high intensity and the distance at which the light is still visible exceeds the visible space of the scene / room.
- **shadow\_bias** – The displacement of the depth / distance of the object's surface during shadow calculation, it is necessary to eliminate artifacts of shadow generation known as self\_shadowing errors. The default value is 0.002.

Note that the parameters **rotation**, **fov**, **fov\_fade** are not used for this type of light source.

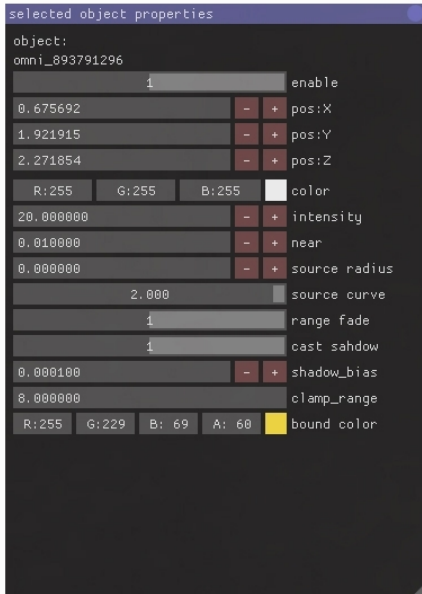
In the advanced version of hooks5, you can use the object properties window to edit the parameters of the light source. Use the light source selectors (double-click on the selector to select, double-click on an empty spot in the scene to deselect) or SCENE\_MANAGER to select the desired light source:

#### omnilight example

```
[omni_1108013808]
enable = false
type = omnilight
position = 5.192848, 3.228252, 0.000000
color = 255, 255, 255
intensity = 12.000000
near = 0.010000
source_radius = 0.000000
source_curve = 2.000000
range_fade = 1
cast_shadow = true
shadow_bias = 0.002000
clamp_range = 0.000000
bound_color = 255, 229, 51, 15
```



#### omnilight properties window



Parameter	Value	Unit / Range
enable	1	checkbox
pos:X	0.675692	- +
pos:Y	1.921915	- +
pos:Z	2.271854	- +
color	R:255 G:255 B:255	RGB
intensity	20.000000	- +
near	0.010000	- +
source radius	0.000000	- +
source curve	2.000	slider
range fade	1	checkbox
cast shadow	1	checkbox
shadow bias	0.000100	- +
clamp range	0.000000	- +
bound color	R:255 G:229 B:69 A:60	RGBA

**[some\_object]** – Parameters for static Hook5 objects.

- **type** – Type of the hook5 object, possible value: static.
- **enable** – Enables / Disables the object, possible values: true / false.
- **position** – Position of the static object in world coordinates, xyz.
- **rotation** – Rotation of the static object relative to the global axes, xyz.
- **scale** – The object's scale on its own coordinate axes, xyz.
- **file** – Denotes the h5m model's path relative to ActiveMod, which will represent a static object in the scene. For the creation of hook5 objects see below.
- **color\_multiplier** – A color in RGBA. The color of the diffuse texture will be multiplied by this color before shading.
- **metallness\_multiplier** – Increases / decreases the metallicity of the object (RED channel of the model's specular texture).
- **glosiness\_multiplier** – Increases / decreases the glossiness of the object (GREEN channel of the model's specular texture).
- **glow\_intensity** – Changes the intensity of the glow.
- **cast\_shadow** – Turns the object's shadow casting on / off.
- **alpha\_mask** – Enables / Disables the pixel exclusion mode for transparency.
- **alpha\_blend** – Enables / Disables alphablend mode.
- **cull\_mode** – Culling mode.

**The Override parameter group** allows you to reassign the textures of the model. The model in h5m format saves information about materials and names of the textures used and the model in h5m format can contain several materials, each of which is assigned to a certain part of geometry (subset). Override allows you to reassign any texture (diffuse, normals, specular) of any subset (the number of subset should start from 0 to the number of subset used in the model minus 1), for example:

- `override.1.diffuse.texture` = the path to the new texture, such a parameter overrides the diffuse texture of the subset #1 model.
- `override.0.specular.texture` = the path to the new texture, this parameter overrides the specular texture of the subset #0 model.

In addition, override supports animated textures to set the rate of texture change use the parameter group `override` – `swapspeed` (works only for override texture):

- `override.3.diffuse.texture` = `TVScreen_num001.dds`
- `override.3.diffuse.swapspeed` = 30

These parameters set the first frame of the texture set as a new diffuse texture for subset #3 and the texture change rate to 30 frames per second.

Also override supports texture shift by texture coordinates (`.uvspeed`), which allows you to specify the rate of displacement of texture uv-coordinates, values: xy:

- `override.7.diffuse.texture` = `SomeNewTexture.dds`
- `override.7.diffuse.uvspeed` = 0.4, 0.1



## Static object example

```
[static_109268096]
enable = false
type = static
style = object
file = _hook5data\objects\backpack.h5m
position = 0.000000, 0.085391, 1.519288
rotation = 0.000000, -72.731857, -0.000000
scale = 0.010000, 0.010000, 0.010000
color_multiplier = 255, 255, 255, 255
metallness_multiplier = 1.000000
glosiness_multiplier = 1.000000
glow_intensity = 0.000000
cast_shadow = true
alpha_mask = false
alpha_blend = false
cull_mode = 0
```



## Static object properties window

selected object properties

object:  
static\_109268096  
vc: 8872, ic: 42723, sc 1

☐ enable

model file:  
\_hook5data\objects\backpack.h5m

0.000000	-	+	pos:X
0.085391	-	+	pos:Y
1.519288	-	+	pos:Z
0.000000	-	+	rot:X
-72.731857	-	+	rot:Y
-0.000000	-	+	rot:Z
0.010000	-	+	scale:X
0.010000	-	+	scale:Y
0.010000	-	+	scale:Z

R:255 G:255 B:255 A:255 ☐ color\_multiplier

1.000 ☐ metallness\_mul

1.000 ☐ glosiness\_mul

0.000000 ☐ glow\_intensity

1 ☐ cast\_shadow

0 ☐ alpha\_mask

0 ☐ alpha\_blend

0 ☐ cull\_mode

CLONE

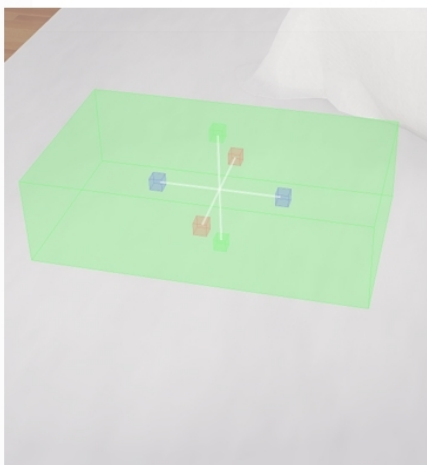
[some\_physbound] – Parameters for the physbound.

- **type** – Type of the hook5 object, possible value: physbound.
- **position** – Position of the physbound in world coordinates, xyz.
- **rotation** – Rotation of the physbound relative to global axes, xyz
- **size** – Size of the physbound relative to its own coordinate axes, xyz.
- **bound\_color** – The color of the physbound in the debug mode.

The physbound object is used in the PhysX simulation, in particular, PhysX hair models will not be able to penetrate physbound objects placed in the scene. In this case, the bound object is not shown.

## Physbound example

```
[bound_847351256]
type = physbound
position = -0.670143, 0.000000, 1.381777
rotation = 0.000000, 0.000000, 0.000000
size = 0.630000, 0.300000, 1.090000
bound_color = 0, 255, 0, 76
```



## Physbound properties window

selected object properties

object:  
bound\_847351256

-0.670143	-	+	pos:X
0.000000	-	+	pos:Y
1.381777	-	+	pos:Z
0.000000	-	+	rot:X
0.000000	-	+	rot:Y
0.000000	-	+	rot:Z
0.630000	-	+	size:X
0.300000	-	+	size:Y
1.090000	-	+	size:Z

R: 0 G:255 B: 0 A: 77 ☒ bound color



[some\_fire] – Parameters for the fire object.

- **type** – Type of the hook5 object, possible value: hook\_fire.
- **enable** – Enable / Disable the fire object (true / false).
- **position** – Position of the fire in world coordinates, xyz.
- **emitter\_size** – The length and width of the fire particles' emitter, xy.
- **samples\_texture** – The path to the fire particles' texture relative to ActiveMod.
- **samples\_dimension** – The number of vertical and horizontal fire particles on the texture, xy.
- **samples\_max\_size** – The maximum size of the fire particles, xy.
- **samples\_count** – The number of particles.
- **swap\_speed** – The animation playback speed in frames per second.
- **color\_scale** The particles' color scale.
- **glow\_amount** – The amount of glow produced by the particles.

The texture of the particles in this case should contain animation frames, that is, it is actually an animation of a single flame tongue. In order to have the illusion of different flames so that the flames do not all look the same, a start frame from the available **samples\_dimension** is randomly generated for each flame and then the animation frames are swapped at the speed of **swap\_speed**.

hook\_fire example

```
[fire_858201264]
enable = true
type = hook_fire
position = -0.518653, 0.000000, 1.487066
samples_texture = _hook5data\particles\fire_samples.c
samples_dimension = 8, 8
emitter_size = 0.400000, 0.400000
samples_min_size = 0.000000, 0.000000
samples_max_size = 1.000000, 1.000000
samples_count = 15
color_scale = 1.000000
glow_amount = 0.450000
```



hook\_fire properties window

selected object properties			
object:			
fire_858201264			
1		enable	
-0.518653	-	+	pos:X
0.000000	-	+	pos:Y
1.487066	-	+	pos:Z
0.40		0.40	emitter size
texture file:			
_hook5data\particles\fire_samples.dds			
8		8	samples dimension
0.00		0.00	samples min size
1.00		1.00	samples max size
		15	samples count
30.000000	-	+	swap speed
3.000000	-	+	color scale
1.000000	-	+	glow amount

[some\_water] – Parameters for water.

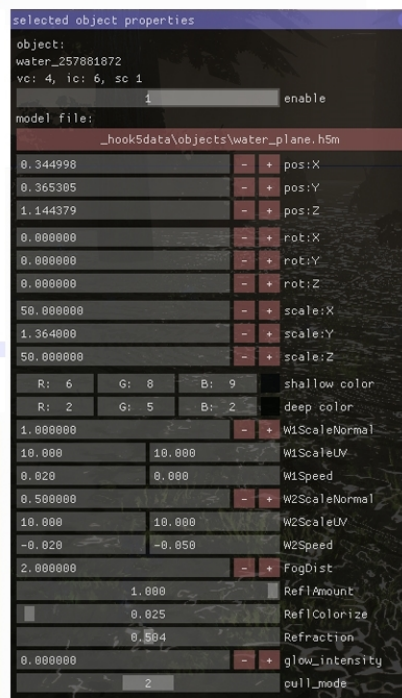
- **type** – Type of the hook5 object, possible value: hook\_water.
- **enable** – Enable / Disable the water object (true / false).
- **position** – Position of the water object in world coordinates, xyz.
- **rotation** – Rotation of the water object relative to global axes, xyz
- **scale** – Scale of the water object relative to its own coordinate axes, xyz.
- **file** – The path relative to ActiveMod to the h5m object that will represent the surface of the water. The object should be a plane, with its normal facing upward. The plane can be of any shape.
- **world\_mapping\_uv** – Not used.
- **deep\_color** – The water's color below the surface.
- **shallow\_color** – The water's color at surface level.
- **wave1\_scale\_normal** – Scales the first wave texture's normal.
- **wave1\_scale\_uv** – Scales the first waves texture coordinates.
- **wave1\_speed** – The displacement speed of the first wave texture.
- **wave2\_scale\_normal** – Scales the second wave texture's normal.
- **wave2\_scale\_uv** – Scales the second waves texture coordinates.
- **wave2\_speed** – The displacement speed of the second wave texture.
- **fog\_distance** – Sets the distance at which water loses transparency.
- **reflection\_amount** – The strength of the water's reflections.
- **reflection\_colorize** – The strength of the color of reflections.
- **refraction\_amount** – The strength of the water's refraction.
- **glow\_intensity** – The intensity of the glow.
- **cull\_mode** – Culling mode.

hook\_water example

```
[water_257881872]
enable = true
type = hook_water
style = object
file = _hook5data\objects\water_plane.h5m
position = 0.344998, 0.365305, 1.144379
rotation = 0.000000, 0.000000, 0.000000
scale = 50.000000, 1.364000, 50.000000
world_mapping_uv = false
shallow_color = 6, 8, 9, 255
deep_color = 2, 5, 2, 255
wave1_scale_normal = 1.000000
wave1_scale_uv = 10.000000, 10.000000
wave1_speed = 0.020000, 0.000000
wave2_scale_normal = 0.500000
wave2_scale_uv = 10.000000, 10.000000
wave2_speed = -0.020000, -0.050000
fog_distance = 2.000000
reflection_amount = 1.000000
reflection_colorize = 0.025000
refraction_amount = 0.504000
glow_intensity = 0.000000
cull_mode = 2
override.0.diffuse.texture = _hook5data\water\water_ocean.dds
override.0.normals.texture = _hook5data\water\water_ripple.dds
```



hook\_water properties window



[some\_mirror] – Parameters for the mirror object.

- **type** – Type of the hook5 object, possible value: mirror.
- **enable** – Enable / Disable the mirror object (true / false).
- **position** – Position of the mirror object in world coordinates, xyz.
- **rotation** – Rotation of the mirror object relative to global axes, xyz
- **scale** – The size of the mirror plane relative to its own coordinate axes, xyz
- **own\_render** – Has to be turned off if you linked the mirror object to a texture of a 3DSV2 object. Reflections will be built along the plane itself ignoring the link, if enabled.
- **darkness** – Darkens reflections. Completely black at a value of 1.
- **diffuse\_map** – The diffuse texture's path relative to ActiveMod. The diffuse color will be mixed with the reflection color using the alpha channel value.
- **normal\_map** – The normal texture's path relative to ActiveMod. The normal texture is used to simulate distortion on the mirror.
- **distortion\_scale** – The mirror distortion's power / intensity.
- **specular\_map** – The specular texture's path relative to ActiveMod. The important part is that the RED channel is not used for metallicity, since this parameter does not matter for the mirror, instead, this channel is used as a mask for the mirror, which is necessary when you want to alter the shape of the mirror from the default rectangle shape.
- **uv\_offset, uv\_scale** – Parameters which can be used to adjust texture coordinates.

### Mirror example

```
[mirror_plane_178917408]
type = mirror
enable = true
position = -1.250974, 0.555395, 2.077781
rotation = 0.000000, -120.891212, -0.000000 _samples.c
scale = 0.306000, 0.416000, 1.000000
own_render = true
darkness = -431602080.000000
diffuse_map =
normal_map =
distortion_scale = 0.000000
specular_map =
uv_offset = 0.000000, 0.000000
uv_scale = 1.000000, 1.000000
```



### Mirror properties window

selected object properties		
object: mirror_plane_178917408		
1	enable	
0.658000	- + width scale	
0.570000	- + height scale	
-1.855684	- + pos:X	
0.653898	- + pos:Y	
1.823580	- + pos:Z	
0.000000	- + rot:X	
-95.699577	- + rot:Y	
-0.000000	- + rot:Z	
1	own_render	
-431602080.000000	- + darkness	
diffuse map file:		
normals map file:		
0.000000	- + distortion scale	
specular map file:		
0.000000	- + uv offset:X	
0.000000	- + uv offset:Y	
1.000000	- + uv scale:X	
1.000000	- + uv scale:Y	

## 5 Selection and manipulation of hook5 objects.

In hook5, for convenience, three modes of working with objects are implemented:

1. Static object mode (static objects, fire, water, mirrors).
2. Light source mode (spotlight, omnilight).
3. A mode for working with bounds (physbound).

To adjust the object's parameters and its transformation it is necessary to double-click with the left mouse on the object body (for light sources on the selector / marker). For the selected object a parameter window for this type of object will be displayed. The window will be displayed as long as the object is selected and cannot be closed. To deselect an object, double click the left mouse button on a place of the scene without any object. The parameter window will be automatically closed when the object is deselected.

You can manipulate (move, rotate, scale-up) the object while it is selected. Manipulation is implemented identically for all objects – grab the manipulator with the middle mouse button and drag it in the desired direction. The game controls are configured so that the left mouse button turns the camera and the right mouse button shows the context menu and when used they would conflict with hook5's configuration. Therefore, by default, the object moving mode is enabled. While pressing the Ctrl key the object rotation mode and while pressing the Shift key the object scaling mode will be activated.

## 6 Creating PhysX models

The main idea is as follows: we create a system of bones in a 3D editor which will represent a strand of hair (a group of strands of hair, or maybe no hair at all – any style). Each group of this system of bones should be attached to the main bone with the name "root" (by this name hook5 identifies the base bone that will be responsible for the transformation of the hairstyle). For the current scene, we then take the transformation of the hairstyle (or another object which we are replacing with the physics model) and assign its bones to the "root" bone. We then perform the physics simulation and get the transformation for each bone in the system. Furthermore, these transformations can be used to render the skin model associated with this bone system. Binding the skin of the model to the object / texture of the game is carried out through the pass file, the description of the parameters is given above. Here is an example:

```
[pass]
name = PhysXModel
stage2 = _hook5data\hairmaps\hair_normal_map.dds
physx_rig = R9hair047.RepX
physx_hair = R9hair047.h5s
physx_model = some_model_part.h5s
```

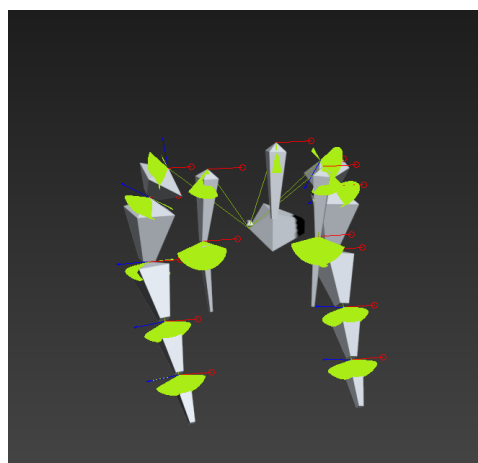
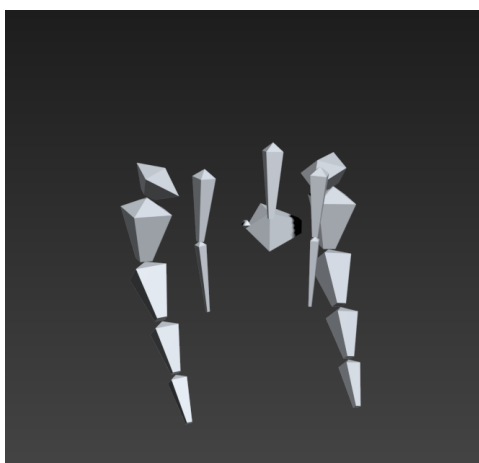
Here .RepX is a physics file (obtained with a mixture of physics plugin for 3d editor), which contains all the information about the bone system that will use physics in hook5 to simulate physics.

The .h5s file is a skin file for a bone model tied to the bone system (Skinning is done in the same manner as it is done for clothing.) The .h5s file is obtained with the help of a script for 3DS Max (I will release it for 3DS Max, unfortunately I don't use Blender).

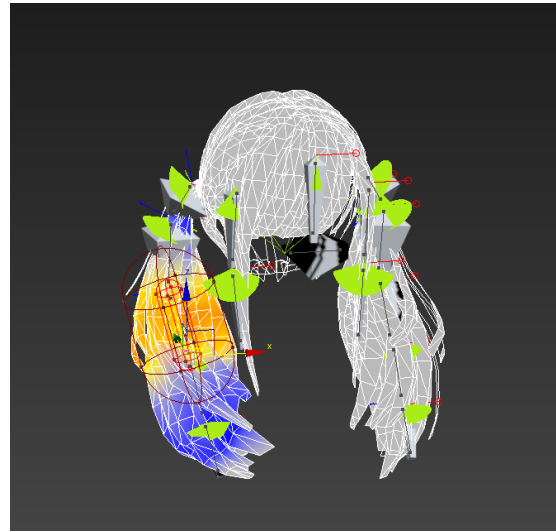
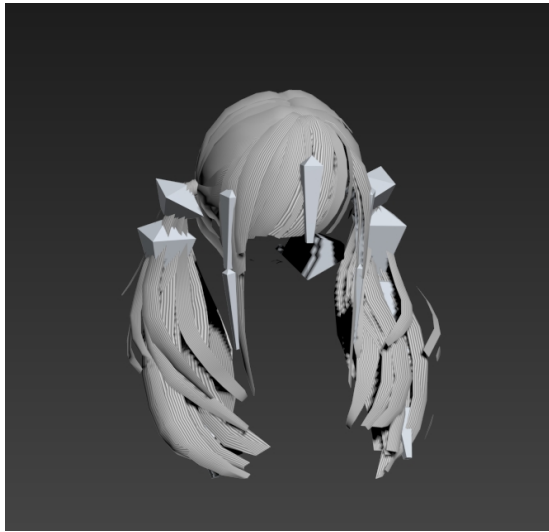
Follow these steps to create a physics model:

1. Create or import a 3D hair model (or a different object).
2. Create a bone system with a master bone named "root".
3. With the help of the **DynamicRig (DynamicRigidSystem)** plugin assign a physics modifier to the system of bones (or each bone separately). The type of the main bone should be Kinematic, the type of all other bones should Dynamic.
4. Adjust the physical values for the bones and joints (Joint).
5. Create a skin modifier and weight the mesh to the model's bones.
6. Export the physics system to a RepX file.
7. Export the model mesh to a .h5s file.
8. Create a pass file and attach both files.

The bone system and the bone system with an assigned physics modifier:



Hairstyle model and hairstyle model with assigned skin modifier:



Please take into account that a replacer for the hairstyle (or no hairstyle) is necessary – you cannot attach the physics model to the skinned object of the game. For objects with skin the object's transformation is unknown, since the game performs all skinning calculations on the CPU. This information is not available for hook5 and therefore, replacers are necessary to apply physics to a model. The hair replacers exchange the skin model hairstyle with a simple hairstyle model without skinning which is then used by the renderer. Hook5 is then able to use the transformation matrix of the replacer for these objects. In principle, you can use existing replacers to connect your physics models, just create RepX and .h5s files and connect them to the appropriate pass file. There, you can also override the textures.

**More information about the actual development state, info about last updates and discussions about problems and solutions can be found in the hook5 threads on [modsgarden.cc](https://modsgarden.cc).**