Marvelous Techniques

Kirnu Interactive - Arto Vaarala 2016

Thank you for purchasing this package. Really appreciated! I have put numerous of hours to create it and I hope you find some good techniques in it you can use in your own creations.

What is Marvelous Techniques

The package is build for you to learn important techniques about how to make beautiful and powerful scenes for your game. You can of course use all the assets in your own game if you like, but I hope you'll create something even cooler than what's inside in this package.

The package consists of many shaders, scripts, models and other assets which are explained in this document. The workflow explained in the following chapters will show you that when you have specific way of doing things it eases the process of making games as well makes it many times quicker.

Scenes

When you are studying the scenes please pay attention to how different colors are used there. When changing one color in the CustomLighting shader the scene seems to break up completely. This is because choosing the right colors is the most important thing in getting the scene to look good.
In my opinion it's very important to choose the main color theme for the scene first and to stick to it even it might feel hard to get everything to look good. The more you make your own scenes the better you become in it.
Concentrate on only one color source at one time. That means disable all post processing effects when you are building the scene. It only makes it harder to get all colors to be in balance with each other.
It's also good to remember that when you don't have much experience in how textures and lighting works together you should choose very neutral texture colors (or lighting colors) but not both at the same time. This way you'll learn quickly your own workflow how to get the mood to the scene you are building.
If you are writing your own shader keep them as simple as possible. This will help you in the long run and you'll build your own library of reusable shaders. As you can see from the shaders included in the package one can get very good looking results with very simple shaders.

The best scene to experiment and learn the custom lighting shader shaders would be 'The Cubes' scene. It's very minimalistic and with it it's easy to see how scene changes by changing the values.

And remember: If you don't like the scenes that's because your taste is different than mine :) I don't consider myself as an artist or anything like that. I know you can do better.

The scene files don't include the Game view screen size. All the scenes are best viewed either 1024x768 or 768x1024 in size

The Workflow

There are as many possible workflows as there are people. The following is just a suggestion and you can modify it as you like. This suggestion does not cover any game play specific tasks. Only building the scene. The workflow consists of these steps:

1. ### Modeling

   After you have decided what kind of scene you are building you start modeling all the scene objects. The meshes are combined together in the end so there can be as many different objects as you like.
   Design the objects so that they can easily share as few materials as possible.
   You can remove all the mesh faces you know will not be seen in the scene. This fastens the combining process but is not necessary because the combining process is fast enough already.

2. ### Building the scene

   The modeling and the scene building phases are quite often overlapping. This is because while building the scene you notice that there is need for new special mesh with different textures or something similar.
   It is recommended that when starting to build the scene you create one ROOT object and place it at 0,0,0. All other scene meshes go under this root object. This helps centering the object being built and rotating it becomes easier.
   Remember to create two tags for the project: **keep** and **nocompare.** These tags are described in detail in the Combine Meshes Tool-section.

   To make this even easier you should make a prefab of every object you use in the scene. This way you can update all the objects sharing the same prefab with one click.

3. ### Creating lighting

   This phase can be either the hardest or the easiest one. It depends how you want the scene to look and do you know how to get that look.
   As said earlier it's often easier to start with simple lighting schemes and after mastering them move to more advanced ones.
   The best shader to start building the lighting is the CustomLighting-shader. Make new material with this shader and apply a texture to it. Use the same material in as many objects as possible. When there are several materials giving the lighting it might get too complicated to handle.
   The lighting will be tweaked more when the lightmapping is done. This is because for example changing the lightmap tint color affects how the other colors are seen.

Shadows

If you want to add shadows to your lightmaps you have to add a directional light to the scene. The Baking section should be set to ‘Baked’.
In order to custom lighting shaders to cast shadows the shader property ‘Cast Shadows’ should be enabled.

4. Combining the meshes

The mesh combining task is fairly easy. You should just remember that if some object acts as a rotator object it must be combined separately.
Mesh combine script creates one big mesh of all meshes sharing the same material. Combining improves the performance (fewer vertices), improves lightmapping (no seams between objects, no leaking of light because all mesh triangles that are not visible are removed)

5. Generating layout texture UVs

This step is only mandatory if you are planning to use layout textures in your combined meshes.
A layout texture is a texture which will be placed over the selected mesh so the whole object will use the same texture. For example a building object may benefit for brick or concrete texture to make it look more ‘real’.
In order to generate UVs for layout texture you have to use UV layout generator script. Create an empty GameObject and attach the script to it. Place all objects to be layout textured under this GameObject.
Tweak the script scale values and click ‘Generate UVs’. Mesh UV3s will be set to correspond the new UV values.
In the custom lighting shader assign a texture to Layout Texture slot and tweak the ‘Layout Texture Power’ property to get the effect you are after.

It is recommended to use textures which do tile and contains no noticeable patterns. This helps the texture to blend over the objects better without any visible seams.

Note that this step must be done after combining the meshes because the script uses a shared mesh property of the mesh. This means all objects which share the same mesh will get the same UVs and therefore will not look correct.

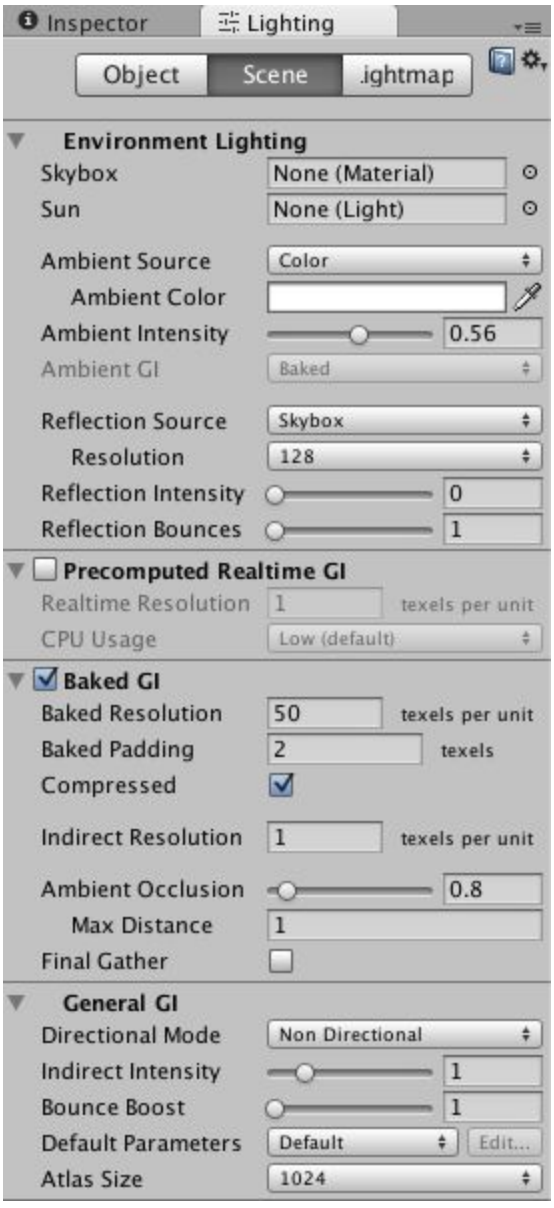
6. Lightmapping

The lightmapping gives the scene the final depth by applying realistic shadows to the meshes. Notice that in the example scenes only Baked GI is used with Ambient Occlusion.
To learn the best settings for your scenes try to tweak different values and see how it changes the overall lighting.

You can see the example settings for the lightmapping.

- Skybox and Sun should be set as None.
- Ambience Source should be color and the color should be white. Also intensity should be somewhere near 0.5 or there will be no AO in the lightmap.
  When baking shadows from lights AO value depends on the intensity of the light.

- Ambient Occlusion slider under Baked GI section sets the overall AO amount.



Scripts

CameraOverlay

This script needs to be attached to the Camera object. Camera object must have a Quad object as a child. The Quad object must have material with shader CameraOverlay.
See the Overlay prefab in the Prefab folder.

UV Layout Generator

This script generates UVs used when layout texture is used. See the 'Generate layout texture UVs' section for more information.

The scale values are used to stretch the layout texture. If both x and y values are 1 the texture will be 1,1 in size and tiled over the object.

Create Ocean Tool

This tool can be found from top toolbar -> Tools -> Marvelous -> Create Ocean
Create Ocean tool creates a specific size ocean object which can be used with the OceanWaver script and OceanDepth shader.
Every vertex has a random color. This color is used in OceanWaver script to determine the y-position of the vertex. By randomizing the y-position the Ocean waving looks much better.

To create a ocean object just select the tool from the menu. New ocean is created at position 0,0,0.

Combine Meshes Tool

This tool can be found from top toolbar -> Tools -> Marvelous -> Combine meshes
Combine Meshes tool combines all meshes sharing the same material under a selected GameObject into one big mesh.
Combining improves the performance (fewer vertices), improves lightmapping (no seams between objects, no leaking of light because all mesh triangles that are not visible are removed)
The combine meshes script procedure is described below:

1. At first the script creates a CustomMesh object of all meshes under the selected GameObject. The combine behavior can be modified with two predefined tags.

   keep: When the GameObject has this tag it will be compared to other GameObjects but nothing is removed from it.
   nocompare: When the GameObject has this tag it will not be compared to other meshes but kept as it is. This tag can be used with complex objects which are not touching with other objects.

2. Next the MeshGrid is created. The grid helps to speed up the comparing process when only meshes sharing the same GridItem will be compared with each other.
3. All the mesh triangles sharing the same grid index are compared and triangles overlapping are removed.
4. Quads are generated from remaining mesh triangles and they are compared like in the phase 3.
5. All meshes sharing the same material are combined together and placed to the scene.
6. The name of the combined mesh will be "_Combined_" + name + "_material_" + material.name. The original mesh will be unactivated so it won't be visible in the scene anymore.

Objects combined are given 'lightmap static' flag automatically. This means they are ready to be lightmapped.

To use the mesh combine tool first select a root object containing all objects to be combined. Then select the tool from the toolbar.

DirectionalLightManager

When Custom Lighting Directional Light property is enabled one must use this script to set the custom light direction.
Attach this script to some transform (Cube,Directional Light etc...). When transform is rotated the light direction changes.

CustomLightingManager

With this manager you can change Custom Lighting shader properties in all materials existing in the scene simultaneously.

EditModeGridSnap

When added to object the object is snapped to the grid. This is very helpful script when building the scene.

Params:
Snap Value: Specifies the grid size.

ObjectRotator

This script enables smooth horizontal scrolling of GameObject. The rotation is snapped to every 90 degrees.

Params:
Transform To Rotate: The transform which will be rotated

Inside Transform To Rotate: If set to true the Transform this script is attached to will not be rotated by script but automatically by the Transform To Rotate object.
Speed: The rotation speed.

OceanWaver

This script makes the Ocean GameObject to wave like real ocean. To get the best performance create the Ocean GameObject with the Create Ocean tool.

Every vertex has random color. This color is used to determine the height of the vertex when animated.

DistanceLight

With this script you can set values to all distance light shaders in the scene.
To use this script create a game object and attach the script to in. The game object position will be the distance light position.

Marvelous Bloom image effect

This script must be attached to the Camera object.
MarvelousBloom script uses the Bloom shader to apply colored bloom post effect over the screen.

Params:
Bloom Color: Bloom effect color
Threshold: The color threshold of bloom effect. The bigger the threshold is the more colors are taken in the bloom effect process
Intensity: The intensity of the bloom effect
Resolution: The resolution of the image where the bloom effect is rendered. Higher resolution produces more accurate bloom.
Blur iterations: To create the bloom effect the image must be blurred. Bigger iteration count results more blurred image and therefore the bloom effect is more prominent.

ScreenTextureBlend image effect

This script must be attached to the Camera object.
ScreenTextureBlend script uses the ScreenTextureBlend shader to apply post effect over the screen.
First it applies the gradient texture blend and then adds the vignetting effect.
The blend modes are similar to PhotoShop blend modes. Blend mode shader script is by Unity Forum user Aubergine.

Params:
Blend Mode: Blend mode applied to the gradient texture.
Blend Intensity: How much the gradient texture is blended to the original image.
Vignette Intensity: How much Vignetting effect is applied to the image.
Vignette Max Value: Vignette effect max value.
Gradient Texture: The gradient texture

Splasher

A special script to randomly instantiate splash particles to random place at random time,

Params:
Splash prefabs to be instantiated: These are selected randomly every time new splash is instantiated.
Min,Max time: Time slot used to randomise next splash instantiation.

EnableCameraDepthTexture

Place this script to the Camera object in order to enable the depth texture generation.

Shaders

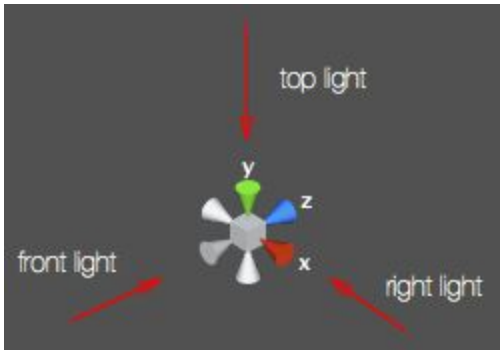
- All Custom Lighting shaders support vertex color. The vertex color is multiplied by the custom lighting.
- Shaders which have letters RS support real time shadows. This means shadows cast by dynamic object are visible when using these shaders.

DistanceLight

Distance light shader will render a point light style effect. The light source can be set to the shader manually or a Distance Light script can be used.
Distance Light script collects all distance light shaders in the scene and set all the shader properties to all shaders simultaneously. The light position is taken from the game object the script is attached to.
Properties:
Max Distance: Maximum distance where effect is applied. The ramp texture is stretched virtually from the light position to the maximum distance.
Light Position: Virtual distance light position.
Light Ramp texture: The light color is taken from this texture. Texture top values are used near the light and bottom values when object is far away from the light.

CustomLighting

This shader creates special lighting effect without any ‘real’ lights. You can specify three light source colors for front, top and right lights. Front light lights to direction +Z, top light lights to direction -Y and right lights to direction -X.



It's also possible to use custom light direction. Enable Directional Light property and use DirectionalLightManager to set light direction.

This shader supports only static shadows baked into lightmaps.

Properties:
Texture: Main texture
Layout Texture: Layout texture
Layout texture power: How much layout texture is applied to the final color
Front color: The color of the front light.
Top color: The color of the top light.
Right color: The color of the right light.
Rim color: The back light color. This light lights to directions: +Y, -Z, +X
Light tint: This color is **multiplied** to the main directional light (calculated from front, top, right colors) color.
Rim power: How much rim color is applied to the final color
Ambient Color: This color is **added** to the main directional light (calculated from front, top, right colors) color.
Ambient Power: Specifies how much ambient light is added to the main directional light.
Lightmap Tint: The lightmap tint color.
Lightmap Power: Specifies how much light map color is added to the main directional light.
Shadow Light: This parameter affects how much light map color is lighted. When this value is raised all pixels in lightmap which have low value will be ignored. This property can be used to only show the strongest/darkest lightmap values.
Lightmap Enabled: Is lightmap enabled.
Directional Light: is directional light enabled. Use DirectionalLightManager to set custom light direction.

CustomLightingSimple

Similar to the CustomLighting shader but this shader uses only one main color and id therefore easier to use. The main color density values are controlled with top, right and front light properties.

Properties:
Main Color: The main color of this shader
Top Light: If the value is over 1 then the light be brighter otherwise darker.
Right Light: If the value is over 1 then the light be brighter otherwise darker.
Front Light: If the value is over 1 then the light be brighter otherwise darker.
All other properties are the same as in the CustomLighting shader.

CustomLightingSimpleSoftFog

This shader behaves similarly to the CustomLightingSimple shader but has additional fog properties.
See properties from CustomLighting and CustomLightingFog shaders.

CustomLightingHardFog

This shader behaves similarly to the CustomLighting shader but has additional fog properties. This fog effect isn't linear but fog color is always the same. Can be used to simulate the water level when object is sunk in water or something similar.

For Color: The color of the fog.
Fog Y-start pos: The Y-position where the for starts.

CustomLightingSoftFog

This shader behaves similarly to the CustomLightingHardFog shader but the fog color is linear instead of hard. The for can also be animated in y-direction.

Properties:
Fog Height: The height of the fog. The fog color is blended linearly from fog y-start position to fog height amount to upwards.
Fog Animation Height: How much fog is moving in +-Y direction
Fog Animation Frequency: The frequency of the fog animation.
Use Fog Distance: Is distance based fog enabled. Distance is calculted between camera position and vertex position
Distance Start: Distance where the fog starts
Distance End: Distance where the fog ends
Distance Density: The density of the distance fog

CameraOverlay

This shader is transparent texture shader with alpha controls.

Properties:
Alpha Power: The texture alpha value is multiplied by this value.

Level: The texture color is multiplied by this value.

DefaultTexture

Renders a texture taking the alpha value into account. If the alpha value == 0 the color will be discarded.

Properties:
Texture: The main texture.

FlagWave

Special shader for waving flags.

Properties:
Texture: The main texture.
Color: The main color.
Wave Magnitude: The wave animation magnitude.
Wave Frequency: The wave animation speed.
Wave Length: The wave animation wavelength.

FloatingLeaf

Special shader for floating leaves. This shader can be used with the particle system. The two colors can be selected with vertex color. If vertex color r-value is >0.7 the Color 2 is used. Otherwise the Color 1 is used.

Properties:
Color 1: If vertex color.r <= 0.7 this color is used
Color 2 If vertex color.r > 0.7 this color is used

Flower

Special shader to color a flower. The color fader from Color 1 to Color 2.

Properties:
Color 1: The first color
Color 2: The second color.
X Power: Specifies how much the vertex x-position affects the color fade.
Y Power: Specifies how much the vertex y-position affects the color fade.
Z Power: Specifies how much the vertex z-position affects the color fade.

OceanDepth

This shader renders a foaming ocean.
The foam effect is generated by using the screen depth texture. To enable the depth texture generation place the EnableCameraDepthTexture script to the main Camera object.
The main pixel color is taken from the color map texture. The x-position of the pixel is taken from the dot product of the light direction and the vertex normal. The pixel is lighter when the vertex normal points to the light direction.
The y-position of the pixel is taken from the uv.y value of the vertex.

Properties:
Foam map: Texture used to render the foam.
Color Map: Texture to colorize the ocean.
Light dir: The direction of light which affect how the colors are calculated.
Foam speed: The speed of the foam effect.
Foam color: The tint color of the foam.
Foam Power: The visibility value of the foam.
Foam Level: The world y-position where the foam can bee seen.
Foam Threshold: How big the foam are is.

OceanOpaque

Simplified version from the Ocean depth shader. This shader is opaque and does not support foam.

OneColor

Simple one color shader. Used to specify one flat color for object.

Properties:
Color: The object color.

TextureAnimation

This shader animates texture in x- and y- direction. It has the custom lighting scheme built in.
It also has the waving animation properties which enable the object to be ‘waved’.

Properties:
Texture: The texture to be animated.
U Speed: x- direction animation speed.
V Speed: x- direction animation speed.
Wave Magnitude. The magnitude of the waving animation.
Wave Frequency. The frequency of the waving animation.
Wave Length. The wavelength of the waving animation.

The rest of the properties are same as the CustomLighting shader excluding the lightmapping section.

Transparent

Transparent shader with a texture. The vertex color.r value is used as an alpha. Value 0 is fully opaque.
Transparent2 shader is the same as this one except it has z-depth of 2.

Properties:
Color: The tint color.

WaterBox

A special shader to render a waving box/cube of water. Only the vertices with y-position > 0.9 will be animated.

Properties:
Opacity: The opacity of the water box.
Wave Magnitude. The magnitude of the waving animation.
Wave Frequency. The frequency of the waving animation.
Wave Length. The wavelength of the waving animation.
The rest of the properties are same as the CustomLighting shader excluding the lightmapping section.

WaterFall

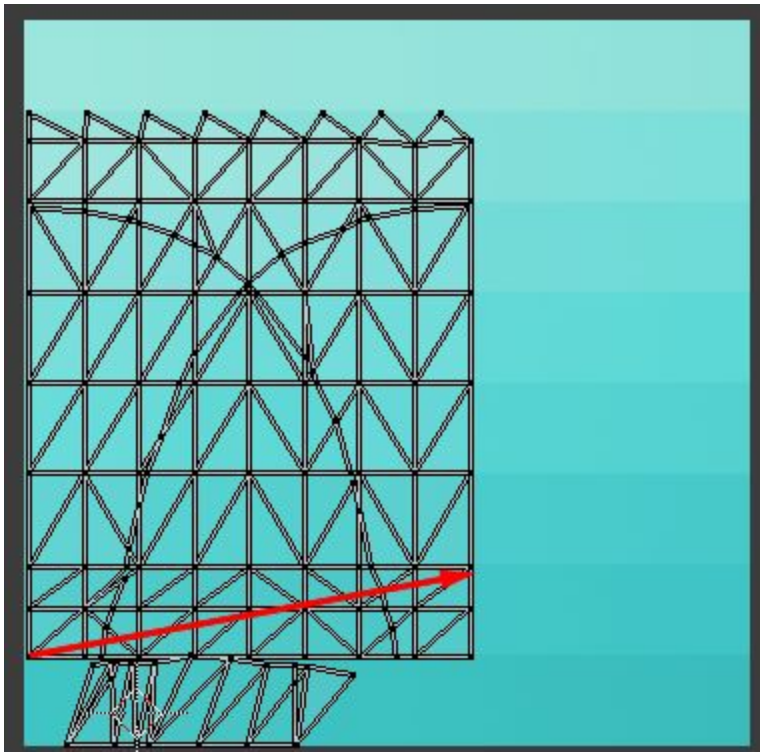
This shader is used to render a waterfall object with moving texture and highlights on the top and the bottom of the waterfall.

Properties:
U Speed: The x-axis speed of the texture
V Speed: The y-axis speed of the texture
Wave Magnitude. The magnitude of the mesh waving animation.
Wave Frequency. The frequency of the mesh waving animation.
Wave Length. The wavelength of the mesh waving animation.
Color: The main color of the waterfall.
Highlight Length: Specifies how big the highlight effect is.
Highlight Fade: Specifies how much the highlight effect is faded
Highlight Color: The color of the highlight effect.

WaterPool

This shader renders a moving texture over the plane object.

The picture below shows what's the principle of the shader. The UV coordinate of the vertex is moved to the right side of the next gradient line. Amazingly this creates the waving effect :)



Properties:
Wave Magnitude. The magnitude of the mesh waving animation.
Wave Frequency. The frequency of the mesh waving animation.
Wave Length. The wavelength of the mesh waving animation.
Speed: The speed of the texture animation.

LinearGradientBG

Renders a linear gradient effect. This effect is always rendered as a background.

Properties:
Top Color: The top color of the gradient.
Bottom Color: The bottom color of the gradient.
Ratio: Top and bottom color ratio. If value is 1 only top color is used. If it is 0 only bottom color is used. For any other value the color is interpolated between the colors.

TransparentAdditiveFG

Additive foreground shader. Nice shader to create glowing objects.

VignetteFG

A foreground shader which renders vignetting effect.

Properties:
Max value: Maximum value for the effect
Intensity: Intensity of the effect.